云数据库 GaussDB

开发指南（集中式_V2.0-3.x）

文档版本 01

发布日期 2025-01-24

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

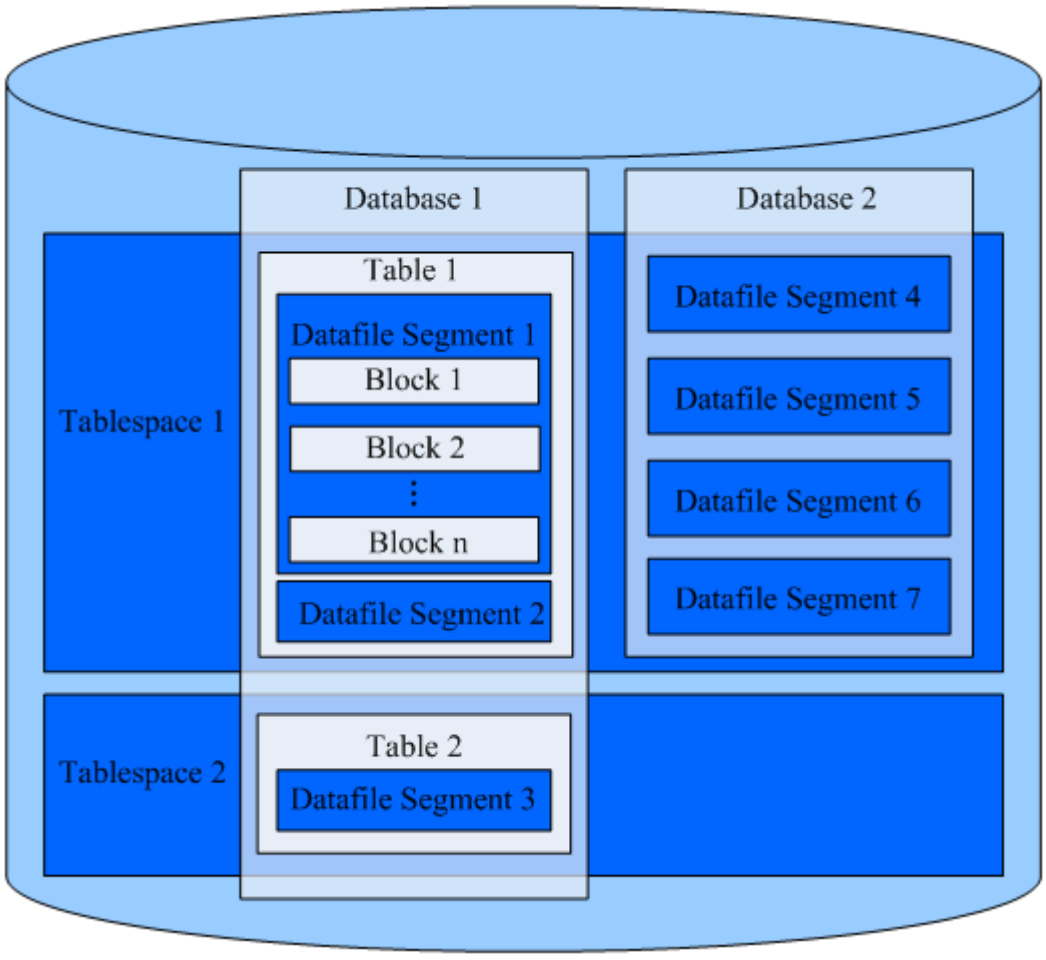
目 录

1 数据库系统概述

1.1 数据库逻辑结构图

GaussDB的数据库节点负责存储数据，其存储介质也是磁盘，本节主要从逻辑视角介绍数据库节点都有哪些对象，以及这些对象之间的关系。数据库逻辑结构如图1-1。

图 1-1 数据库逻辑结构图

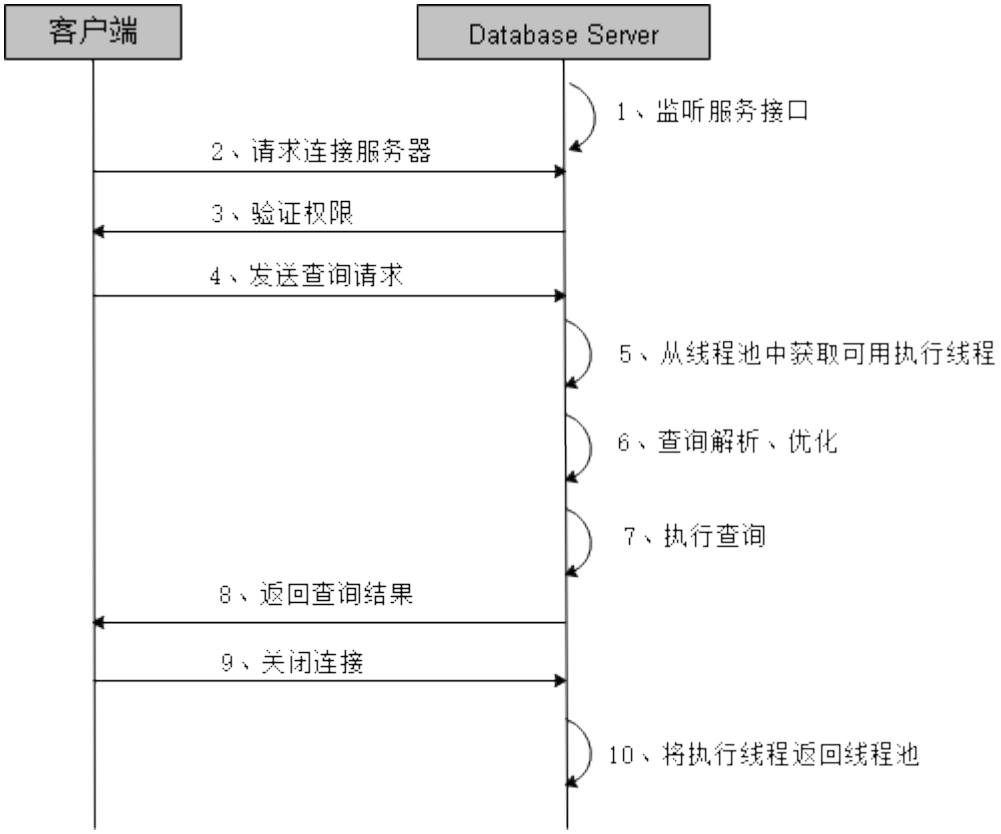


说明

- Tablespace，即表空间，表空间是一个目录，实例中可以存在多个表空间，其中存储的是它所包含的数据库的各种物理文件。每个表空间可以对应多个Database。
- Database，即数据库，用于管理各类数据对象，各数据库间相互隔离。数据库管理的对象可分布在多个Tablespace上。
- Datafile Segment，即数据文件，通常每张表只对应一个数据文件。如果某张表的数据大于1GB，则会分为多个数据文件存储。
- Table，即表，每张表只能属于一个数据库，也只能对应到一个Tablespace。每张表对应的数据文件必须在同一个Tablespace中。
- Block，即数据块，是数据库管理的基本单位，默认大小为8KB。

1.2 数据查询请求处理过程

图 1-2 GaussDB 服务响应流程



1.3 管理事务

事务是用户定义的一个数据库操作序列，这些操作要么全做要么全不做，是一个不可分割的工作单位。GaussDB数据库支持的事务控制命令有启动、设置、提交、回滚。GaussDB数据库支持的事务隔离级别有READ COMMITTED、REPEATABLE READ和SERIALIZABLE，不推荐使用READ UNCOMMITTED，SERIALIZABLE等价于REPEATABLE READ。

事务控制

以下是数据库支持的事务命令：

- 启动事务

  用户可以使用START TRANSACTION和BEGIN语法启动事务，详细操作请参见**START TRANSACTION**和**BEGIN**。

- 设置事务

  用户可以使用SET TRANSACTION或者SET LOCAL TRANSACTION语法设置事务特性，详细操作请参见**SET TRANSACTION**。

- 提交事务

  用户可以使用COMMIT或者END完成提交事务的功能，即提交事务的所有操作，详细操作请参见**COMMIT | END**。

- 回滚事务

  回滚是在事务运行的过程中发生了某种故障，事务不能继续执行，系统将事务中对数据库的所有已完成的操作全部撤销，详细操作请参见**ROLLBACK**。

事务隔离级别

事务隔离级别决定多个事务并发操作同一个对象时的处理方式。

说明

在事务中第一个数据操作语句（SELECT，INSERT，DELETE，UPDATE，FETCH，COPY）执行之后，事务隔离级别不能再次设置。

- **READ COMMITTED**：读已提交隔离级别，事务只能读到已提交的数据而不会读到未提交的数据，这是缺省值。

  实际上，SELECT查询会查看到在查询开始运行的瞬间该数据库的一个快照。不过，SELECT能查看到其自身所在事务中先前修改的执行结果，即使先前修改尚未提交。请注意，在同一个事务里两个相邻的SELECT命令可能会查看到不同的快照，因为其它事务会在第一个SELECT执行期间提交。

  因为在读已提交模式里，每个新的命令都是从一个新的快照开始的，而这个快照包含所有到该时刻为止已提交的事务，因此同一事务中后面的命令将看到任何已提交的其它事务的效果。这里关注的问题是在单个命令里是否看到数据库里绝对一致的视图。

  读已提交模式提供的部分事务隔离对于许多应用而言是足够的，并且这个模式速度快，使用简单。不过，对于做复杂查询和更新的应用，可能需要保证数据库有比读已提交模式更加严格的一致性视图。

- **REPEATABLE READ**：事务可重复读隔离级别，事务只能读到事务开始之前已提交的数据，不能读到未提交的数据以及事务执行期间其它并发事务提交的修改（但是，查询能查看到自身所在事务中先前修改的执行结果，即使先前修改尚未提交）。这个级别和读已提交是不一样的，因为可重复读事务中的查询看到的是事务开始时的快照，不是该事务内部当前查询开始时的快照，就是说，单个事务内部的select命令总是查看到同样的数据，查看不到自身事务开始之后其他并发事务修改后提交的数据。使用该级别的应用必须准备好重试事务，因为可能会发生串行化失败。

- **SERIALIZABLE**：GaussDB目前功能上不支持此隔离级别，设置该隔离级别时，等价于REPEATABLE READ。

说明

REPEATABLE READ基于多版本快照实现，可能出现写偏斜的场景，如果需要避免该场景出现，请先对事务涉及的行进行SELECT FOR UPDATE操作。写偏斜的场景示例如下：

场景一：表a拥有id、value两个字段，类型均为int，插入两条数据，假定a业务逻辑上需要满足两条数据value和小于等于10。并发开启两个事务，基于读取的值进行更新并修改相应值，修改后在事务内均满足value和小于等于10，提交后，最终value和等于12，违反a的业务逻辑假定。

```
gaussdb=# create table a(id int, value int);
CREATE TABLE
gaussdb=# insert into a values(1,4);
INSERT 0 1
gaussdb=# insert into a values(2,4);
INSERT 0 1
session1 :
gaussdb=# start transaction isolation level repeatable read;
START TRANSACTION
gaussdb=# select * from a;
id | value
----+-------
1 |     4
2 |     4
(2 rows)
gaussdb=# update a set value = 6 where id = 1;
UPDATE 1
gaussdb=# select * from a;
id | value
----+-------
1 |     6
2 |     4
(2 rows)
session2:
gaussdb=# start transaction isolation level repeatable read;
START TRANSACTION
gaussdb=# select * from a;
id | value
----+-------
1 |     4
2 |     4
(2 rows)
gaussdb=# update a set value = 6 where id = 2;
UPDATE 1
gaussdb=# select * from a;
id | value
----+-------
1 |     4
2 |     6
(2 rows)
session1：
gaussdb=# commit;
COMMIT
session2:
gaussdb=# commit;
COMMIT
gaussdb=# select * from a;
id | value
----+-------
1 |     6
2 |     6
(2 rows)
```

场景二：表a拥有两个字段id、value，类型均为int，其中id为主键，并发主键删除插入，事务内可能读到两条主键的值，违反主键约束。

```
gaussdb=# create table a(id int primary key, value int);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "a_pkey" for table "a"
CREATE TABLE
gaussdb=# insert into a values(1,10);
```



```
INSERT 0 1
session1:
gaussdb=# start transaction isolation level repeatable read;
START TRANSACTION
gaussdb=# delete a where id = 1;
DELETE 1
session2:
gaussdb=# start transaction isolation level repeatable read;
START TRANSACTION
gaussdb=# select * from a;
id | value
----+-------
1 |    10
(1 row)
session1:
gaussdb=# commit;
COMMIT
session2:
gaussdb=# insert into a values(1, 100);
INSERT 0 1
gaussdb=# select * from a;
id | value
----+-------
1 |    10
1 |   100
(2 rows)
```

1.4 相关概念

数据库

数据库用于管理各类数据对象，与其他数据库隔离。创建数据对象时可以指定对应的表空间，如果不指定相应的表空间，相关的对象会默认保存在PG_DEFAULT空间中。数据库管理的对象可分布在多个表空间上。

表空间

在GaussDB中，表空间是一个目录，实例中可以存在多个表空间，里面存储的是它所包含的数据库的各种物理文件。由于表空间是一个目录，仅起到了物理隔离的作用，其管理功能依赖于文件系统。

模式

GaussDB的模式是对数据库做一个逻辑分割。所有的数据库对象都建立在模式下面。GaussDB的模式和用户是弱绑定，弱绑定是指虽然创建用户的同时会自动创建一个同名模式，但用户也可以单独创建模式，并且为用户指定其他的模式。

用户和角色

GaussDB使用用户和角色控制对数据库的访问。根据角色自身的设置不同，一个角色可以看做是一个数据库用户，或者一组数据库用户，一个用户唯一对应一个角色。在GaussDB中角色和用户之间的区别只在于角色默认是没有LOGIN权限的。在GaussDB中可以使用角色叠加来更灵活地进行管理。

事务管理

在事务管理上，GaussDB采取了MVCC（多版本并发控制）结合两阶段锁的方式，其特点是读写之间不阻塞。GaussDB的astore存储引擎没有将历史版本数据统一存放，

而是和当前元组的版本放在了一起。GaussDB的astore存储引擎没有回滚段的概念，但是为了定期清除历史版本数据，GaussDB的astore存储引擎引入了VACUUM线程。一般情况下，除非用户要做性能调优，否则不用特别关注VACUUM线程。GaussDB的ustore存储引擎是将历史版本数据统一存放到undo回滚段里，由undo回收线程统一清理历史版本数据。此外，GaussDB对于单语句查询（没有使用BEGIN等语句显示启动事务块）是自动提交事务的。

2 数据库安全

2.1 用户及权限

2.1.1 默认权限机制

数据库对象创建后，进行对象创建的用户就是该对象的所有者。数据库安装后默认情况下，未开启**三权分立**，数据库系统管理员具有与对象所有者相同的权限。也就是说对象创建后，默认只有对象所有者或者系统管理员可以查询、修改和销毁对象，以及通过**GRANT**将对象的权限授予其他用户。

为使其他用户能够使用对象，必须向用户或包含该用户的角色授予必要的权限。

GaussDB支持以下的权限：SELECT、INSERT、UPDATE、DELETE、TRUNCATE、REFERENCES、CREATE、CONNECT、EXECUTE、USAGE、ALTER、DROP、COMMENT、INDEX和VACUUM。不同的权限与不同的对象类型关联。有关各权限的详细信息，请参见**GRANT**。

要撤销已经授予的权限，请参见**REVOKE**。对象所有者的权限（例如ALTER、DROP、COMMENT、INDEX、VACUUM、GRANT和REVOKE）是隐式拥有的，即只要拥有对象就可以执行对象所有者的这些隐式权限。对象所有者可以撤销自己的普通权限（SELECT、INSERT、UPDATE、DELETE），例如，使表对自己以及其他人只读，系统管理员用户除外。

系统表和系统视图要么只对系统管理员可见，要么对所有用户可见。标识了需要系统管理员权限的系统表和视图只有系统管理员可以查询。有关信息，请参考**系统表和系统视图**。

数据库提供对象隔离的特性，对象隔离特性开启时，用户只能查看有权限访问的对象(表、视图、字段、函数)，系统管理员不受影响。有关信息，请参考**ALTER DATABASE**。

不建议用户修改系统表和系统视图的权限。

2.1.2 管理员

初始用户

数据库安装过程中自动生成的账户称为初始用户。初始用户也是系统管理员、安全管理员、审计管理员、监控管理员、运维管理员和安全策略管理员，拥有系统的最高权

限，能够执行所有的操作。如果安装时不指定初始用户名称，则该账户与进行数据库安装的操作系统用户同名。如果在安装时不指定初始用户的密码，安装完成后密码为空，在执行其他操作前需要通过gsql客户端修改初始用户的密码。如果初始用户密码为空，则除修改密码外无法执行其他SQL操作以及升级、扩容、节点替换等操作。

初始用户会绕过所有权限检查。建议仅将此初始用户作为DBA管理用途，而非业务应用。

系统管理员

系统管理员是指具有SYSADMIN属性的账户，默认安装情况下具有与对象所有者相同的权限，但不包括dbe_perf模式的对象权限和使用Roach工具执行备份恢复的权限。

要创建新的系统管理员，请以初始用户或者系统管理员用户身份连接数据库，并使用带SYSADMIN选项的**CREATE USER**语句或**ALTER USER**语句进行设置。

```
gaussdb=# CREATE USER sysadmin WITH SYSADMIN password "********";
```

或者

```
gaussdb=# ALTER USER joe SYSADMIN;
```

ALTER USER时，要求用户已存在。

安全管理员

安全管理员是指具有CREATEROLE属性的账户，具有创建、修改、删除用户或角色的权限。

要创建新的安全管理员，三权分立关闭时，请以系统管理员或者安全管理员身份连接数据库。三权分立打开时，请以安全管理员身份连接数据库，并使用带CREATEROLE选项的**CREATE USER**语句或**ALTER USER**语句进行设置。

```
gaussdb=# CREATE USER createrole WITH CREATEROLE password "********";
```

或者

```
gaussdb=# ALTER USER joe CREATEROLE;
```

ALTER USER时，要求用户已存在。

审计管理员

审计管理员是指具有AUDITADMIN属性的账户，具有查看和删除审计日志的权限。

要创建新的审计管理员，三权分立关闭时，只能以系统管理员或者安全管理员身份连接数据库。三权分立打开时，只能以初始用户身份连接数据库，并使用带AUDITADMIN选项的**CREATE USER**语句或**ALTER USER**语句进行设置。

```
gaussdb=# CREATE USER auditadmin WITH AUDITADMIN password "********";
```

或者

```
gaussdb=# ALTER USER joe AUDITADMIN;
```

ALTER USER时，要求用户已存在。

监控管理员

监控管理员是指具有MONADMIN属性的账户，具有查看dbe_perf模式下视图和函数的权限，亦可以对dbe_perf模式的对象权限进行授予或收回。

要创建新的监控管理员，请以系统管理员身份连接数据库，并使用带MONADMIN选项的**CREATE USER**语句或**ALTER USER**语句进行设置。

```
gaussdb=# CREATE USER monadmin WITH MONADMIN password "********";
```

或者

```
gaussdb=# ALTER USER joe MONADMIN;
```

ALTER USER时，要求用户已存在。

运维管理员

运维管理员是指具有OPRADMIN属性的账户，具有使用Roach工具执行备份恢复的权限。

要创建新的运维管理员，请以初始用户身份连接数据库，并使用带OPRADMIN选项的**CREATE USER**语句或**ALTER USER**语句进行设置。

```
gaussdb=# CREATE USER opradmin WITH OPRADMIN password "********";
```

或者

```
gaussdb=# ALTER USER joe OPRADMIN;
```

ALTER USER时，要求用户已存在。

安全策略管理员

安全策略管理员是指具有POLADMIN属性的账户，具有创建资源标签，脱敏策略和统一审计策略的权限。

要创建新的安全策略管理员，请以系统管理员用户身份连接数据库，并使用带POLADMIN选项的**CREATE USER**语句或**ALTER USER**语句进行设置。

```
gaussdb=# CREATE USER poladmin WITH POLADMIN password "********";
```

或者

```
gaussdb=# ALTER USER joe POLADMIN;
```

ALTER USER时，要求用户已存在。

2.1.3 三权分立

默认权限机制和**管理员**两节的描述基于的是数据库创建之初的默认情况。默认情况下拥有SYSADMIN属性的系统管理员，具备系统最高权限。

在实际业务管理中，为了避免系统管理员拥有过度集中的权利带来高风险，可以设置三权分立，将系统管理员的创建用户和审计管理的权限分别授予安全管理员和审计管理员。

三权分立后，系统管理员将不再具有CREATEROLE属性（安全管理员）和AUDITADMIN属性（审计管理员），即不再拥有创建角色和用户的权限，也不再拥有查看和维护数据库审计日志的权限。关于CREATEROLE属性和AUDITADMIN属性的更多信息请参考**CREATE ROLE**。

初始用户的权限不受三权分立设置影响。因此建议仅将此初始用户作为DBA管理用途，而非业务应用。

三权分立的设置方法为：将GUC参数**enableSeparationOfDuty**设置为on。

警告

如需使用三权分立权限管理模型，应在数据库初始化阶段指定，不建议来回切换权限管理模型。需要注意的是，当需从非三权分立权限管理模型切换至三权分立权限管理模型时，应重新审视已有用户的权限集合。如用户具备系统管理员权限和审计管理员权限，则需要进行权限裁剪。

三权分立后，系统管理员对其他用户的非系统模式不再具有权限，因此在未被授予其他用户模式的权限前，也不能访问放在其他用户模式下的对象。三权分立前的权限详情及三权分立后的权限变化，请分别参见表2-1和表2-2。

表 2-1 默认的用户权限

对象名称	初始用户（id为10）	系统管理员	安全管理员	审计管理员	普通用户
表空间	具有所有的权限。	对表空间有创建、修改、删除、访问、分配操作的权限。	不具有对表空间进行创建、修改、删除、分配的权限，访问需要被授权。		
模式		对除dbe_perf以外的所有模式有所有的权限。	对自己的模式有所有的权限，对其他用户的非系统模式无权限。		
自定义函数		对所有用户自定义函数有所有的权限。	对自己的函数有所有的权限，对其他用户的函数仅有调用权限。		
自定义表或视图		对所有用户自定义表或视图有所有的权限。	对自己的表或视图有所有的权限，对其他用户的表或视图无权限。		

表 2-2 三权分立较非三权分立权限变化说明

对象名称	初始用户（id为10）	系统管理员	安全管理员	审计管理员	普通用户
表空间	无变化。 依然具有所有的权限。	无变化	无变化		
模式		权限缩小。 对自己的模式有所有的权限，对其他用户的非系统模式无权限。	无变化		
自定义函数		在未被授予其他用户的非系统模式的权限前，不能访问放在其他用户模式下的函数。	无变化		

对象名称	初始用户（id为10）	系统管理员	安全管理员	审计管理员	普通用户
自定义表或视图		在未被授予其他用户的非系统模式的权限前，不能访问放在其他用户模式下的表或视图。	无变化		

须知

PG_STATISTIC系统表和PG_STATISTIC_EXT系统表存储了统计对象的一些敏感信息，如高频值MCV。进行三权分立后系统管理员仍可以通过访问这两张系统表，获取统计信息里的敏感信息。

2.1.4 用户

使用CREATE USER和ALTER USER可以创建和管理数据库用户。数据库系统包含一个或多个数据库，用户和角色在整个数据库系统范围内是共享的，但是其数据并不共享。即用户可以连接任何数据库，但当连接成功后，任何用户都只能访问连接请求里声明的那个数据库。

非三权分立下，GaussDB用户账户只能由系统管理员或拥有CREATEROLE属性的安全管理员创建和删除。三权分立时，用户账户只能由初始用户和安全管理员创建。

在用户登录GaussDB时会对其进行身份验证。用户可以拥有数据库和数据库对象（例如表），并且可以向用户和角色授予对这些对象的权限以控制谁可以访问哪个对象。除系统管理员外，具有CREATEDB属性的用户可以创建数据库并授予对这些数据库的权限。

创建、修改和删除用户

- 创建用户，请使用SQL语句**CREATE USER**。

  例如：创建用户joe，并设置用户拥有CREATEDB属性。

  ```
  gaussdb=# CREATE USER joe WITH CREATEDB PASSWORD "********";
  CREATE ROLE
  ```

- 创建系统管理员，请使用带有SYSADMIN选项的**CREATE USER**语句。
- 删除现有用户，请参见**DROP USER**。
- 更改用户账户（例如，重命名用户或更改密码），请参见**ALTER USER**。
- 查看用户列表，请查询视图**PG_USER**。

  ```
  gaussdb=# SELECT * FROM pg_user;
  ```

- 查看用户属性，请查询系统表**PG_AUTHID**。

  ```
  gaussdb=# SELECT * FROM pg_authid;
  ```

永久用户

GaussDB提供永久用户方案：创建具有PERSISTENCE属性的永久用户，具有PERSISTENCE属性的用户能够使用service_reserved_connections通道连接数据库。

说明

service_reserved_connections为带有persistence属性预留的最小连接数，不建议设置过大。

```
gaussdb=# CREATE USER user_persistence WITH PERSISTENCE IDENTIFIED BY "*********";
```

只允许初始用户创建、修改和删除具有PERSISTENCE属性的永久用户。

2.1.5 角色

通过GRANT把角色授予用户后，用户即具有了角色的所有权限。推荐使用角色进行高效权限分配。例如，可以为设计、开发和维护人员创建不同的角色，将角色GRANT给用户后，再向每个角色中的用户授予其工作所需数据的差异权限。在角色级别授予或撤销权限时，这些更改将作用到角色下的所有成员。

GaussDB提供了一个隐式定义的拥有所有角色的组PUBLIC，所有创建的用户和角色默认拥有PUBLIC所拥有的权限。关于PUBLIC默认拥有的权限请参见**GRANT**。要撤销或重新授予用户和角色对PUBLIC的权限，可通过在GRANT和REVOKE指定关键字PUBLIC实现。

要查看所有角色，请查询系统表PG_ROLES：

```
SELECT * FROM PG_ROLES;
```

创建、修改和删除角色

非三权分立时，只有系统管理员和具有CREATEROLE属性的用户才能创建、修改或删除角色。**三权分立**下，只有初始用户和具有CREATEROLE属性的用户才能创建、修改或删除角色。

- 创建角色，请参见**CREATE ROLE**。
- 在现有角色中添加或删除用户，请参见**ALTER ROLE**。
- 删除角色，请参见**DROP ROLE**。DROP ROLE只会删除角色，并不会删除角色中的成员用户账户。

内置角色

GaussDB提供了一组默认角色，以gs_role_开头命名。它们提供对特定的、通常需要高权限的操作的访问，可以将这些角色GRANT给数据库内的其他用户或角色，让这些用户能够使用特定的功能。在授予这些角色时应当非常小心，以确保它们被用在需要的地方。**表2-3**描述了内置角色允许的权限范围：

表 2-3 内置角色权限描述

角色	权限描述
gs_role_signal_backend	具有调用函数pg_cancel_backend、pg_terminate_backend和pg_terminate_session来取消或终止其他会话的权限，或调用函数pg_terminate_active_session_socket来关闭活跃会话和客户端的socket连接，但不能操作属于初始用户和PERSISTENCE用户的会话。
gs_role_tablespace	具有创建表空间（tablespace）的权限。

角色	权限描述
gs_role_replication	具有调用逻辑复制相关函数的权限，例如kill_snapshot、pg_create_logical_replication_slot、pg_create_physical_replication_slot、pg_drop_replication_slot、pg_replication_slot_advance、pg_create_physical_replication_slot_extern、pg_logical_slot_get_changes、pg_logical_slot_peek_changes、pg_logical_slot_get_binary_changes、pg_logical_slot_peek_binary_changes。
gs_role_account_lock	具有加解锁用户的权限，但不能加解锁初始用户和PERSISTENCE用户。
gs_role_pldebugger	具有执行dbe_pldebugger下调试函数的权限。
gs_role_public_dblink_drop	具有执行删除public database link对象的权限。
gs_role_public_dblink_alter	具有执行修改public database link对象的权限。

关于内置角色的管理有如下约束：

- 以gs_role_开头的角色名作为数据库的内置角色保留名，禁止新建以“gs_role_”开头的用户/角色/模式，也禁止将已有的用户/角色/模式重命名为以“gs_role_”开头。
- 禁止对内置角色进行ALTER和DROP操作。
- 内置角色默认没有LOGIN权限，不设预置密码。
- gsql元命令\du和\dg不显示内置角色的相关信息，但若显示指定了pattern为特定内置角色则会显示。
- 三权分立关闭时，初始用户、具有SYSADMIN权限的用户和具有内置角色ADMIN OPTION权限的用户有权对内置角色执行GRANT/REVOKE管理。三权分立打开时，初始用户和具有内置角色ADMIN OPTION权限的用户有权对内置角色执行GRANT/REVOKE管理。例如：

```
GRANT gs_role_signal_backend TO user1;
REVOKE gs_role_signal_backend FROM user1;
```

2.1.6 Schema

Schema又称作模式。通过管理Schema，允许多个用户使用同一数据库而不相互干扰，可以将数据库对象组织成易于管理的逻辑组，同时便于将第三方应用添加到相应的Schema下而不引起冲突。

每个数据库包含一个或多个Schema。数据库中的每个Schema包含表和其他类型的对象。数据库创建初始，默认具有一个名为public的公共Schema，所有用户都拥有此Schema的usage权限。此外，每个数据库都包含一个pg_catalog Schema，它包含系统表和所有内置数据类型、函数和操作符。只有系统管理员和初始化用户可以在public和pg_catalog Schema下创建普通函数、聚合函数、存储过程和同义词对象，只有初始化用户可以在public和pg_catalog Schema下创建操作符，其他用户即使授予public和pg_catalog模式的create权限后也不可以创建上述五种对象。可以通过Schema分组

数据库对象。Schema类似于操作系统目录，但Schema不能嵌套。默认只有初始化用户可以在pg_catalog模式下创建对象。

相同的数据库对象名称可以应用在同一数据库的不同Schema中，而没有冲突。例如，a_schema和b_schema都可以包含名为mytable的表。具有所需权限的用户可以访问数据库的多个Schema中的对象。

通过CREATE USER创建用户的同时，系统会在执行该命令的数据库中，为该用户创建一个同名的Schema。

数据库对象创建在数据库搜索路径中的第一个Schema内。默认情况下的第一个Schema及如何变更Schema顺序等更多信息，请参见**搜索路径**。

创建、修改和删除 Schema

- 创建Schema，请参见**CREATE SCHEMA**。默认初始用户和系统管理员可以创建Schema，其他用户需要具备数据库的CREATE权限才可以在该数据库中创建Schema，赋权方式请参见**GRANT**中将数据库的访问权限赋予指定的用户或角色中的语法。
- 更改Schema名称或者所有者，请参见**ALTER SCHEMA**。Schema所有者可以更改Schema。
- 删除Schema及其对象，请参见**DROP SCHEMA**。Schema所有者可以删除Schema。
- 在Schema内创建表，请以schema_name.table_name格式创建表。不指定schema_name时，对象默认创建到**搜索路径**中的第一个Schema内。
- 查看Schema所有者，请对系统表PG_NAMESPACE和PG_USER执行如下关联查询。语句中的schema_name请替换为实际要查找的Schema名称。
  ```
  gaussdb=# SELECT s.nspname,u.usename AS nspowner FROM pg_namespace s, pg_user u WHERE nspname='schema_name' AND s.nspowner = u.usesysid;
  ```
- 查看所有Schema的列表，请查询PG_NAMESPACE系统表。
  ```
  gaussdb=# SELECT * FROM pg_namespace;
  ```
- 查看属于某Schema下的表列表，请查询系统视图PG_TABLES。例如，以下查询会返回Schema PG_CATALOG中的表列表。
  ```
  gaussdb=# SELECT distinct(tablename),schemaname from pg_tables where schemaname = 'pg_catalog';
  ```

搜索路径

搜索路径定义在**search_path**参数中，参数取值形式为采用逗号分隔的Schema名称列表。如果创建对象时未指定目标Schema，则该对象将会被添加到搜索路径中列出的第一个Schema中。当不同Schema中存在同名的对象时，查询对象未指定Schema的情况下，将从搜索路径中包含该对象的第一个Schema中返回对象。

- 查看当前搜索路径，请参见**SHOW**。
  ```
  gaussdb=# SHOW SEARCH_PATH;
   search_path
  ----------------
   "$user",public
  (1 row)
  ```
  search_path参数的默认值为："$user", public。$user表示与当前会话用户名同名的Schema名，如果这样的模式不存在，$user将被忽略。所以默认情况下，用户连接数据库后，如果数据库下存在同名Schema，则对象会添加到同名Schema下，否则对象被添加到Public Schema下。
- 更改当前会话的默认Schema，请使用SET命令。

执行如下命令将搜索路径设置为myschema, public，首先搜索myschema，然后搜索public。

```
gaussdb=# SET SEARCH_PATH TO myschema, public;
SET
```

2.1.7 用户权限设置

- 给用户直接授予某对象的权限，请参见**GRANT**。

  将Schema中的表或者视图对象授权给其他用户或角色时，需要将表或视图所属Schema的USAGE权限同时授予该用户或角色。否则用户或角色将只能看到这些对象的名称，并不能实际进行对象访问。

  例如，下面示例将Schema tpcds的权限授予用户joe后，将表tpcds.web_returns的select权限授予用户joe。

  ```
  gaussdb=# GRANT USAGE ON SCHEMA tpcds TO joe;
  gaussdb=# GRANT SELECT ON TABLE tpcds.web_returns to joe;
  ```

- 给用户指定角色，使用户继承角色所拥有的对象权限。

  a. 创建角色。

  新建一个角色lily，同时给角色指定系统权限CREATEDB：

  ```
  gaussdb=# CREATE ROLE lily WITH CREATEDB PASSWORD "********";
  ```

  b. 给角色赋予对象权限，请参见**GRANT**。

  例如，将模式tpcds的权限授予角色lily后，将表tpcds.web_returns的select权限授予角色lily。

  ```
  gaussdb=# GRANT USAGE ON SCHEMA tpcds TO lily;
  gaussdb=# GRANT SELECT ON TABLE tpcds.web_returns to lily;
  ```

  c. 将角色的权限授予用户。

  ```
  gaussdb=# GRANT lily to joe;
  ```

  说明

  当将角色的权限授予用户时，角色的属性并不会传递到用户。

- 回收用户权限，请参见**REVOKE**。

2.1.8 行级访问控制

行级访问控制特性将数据库访问控制精确到数据表行级别，使数据库达到行级访问控制的能力。不同用户执行相同的SQL查询操作，读取到的结果是不同的。

用户可以在数据表创建行访问控制(Row Level Security)策略，该策略是指针对特定数据库用户、特定SQL操作生效的表达式。当数据库用户对数据表访问时，若SQL满足数据表特定的Row Level Security策略，在查询优化阶段将满足条件的表达式，按照属性(PERMISSIVE | RESTRICTIVE)类型，通过AND或OR方式拼接，应用到执行计划上。

行级访问控制的目的是控制表中行级数据可见性，通过在数据表上预定义Filter，在查询优化阶段将满足条件的表达式应用到执行计划上，影响最终的执行结果。当前受影响的SQL语句包括SELECT、UPDATE和DELETE。

场景一：某表中汇总了不同用户的数据，但是不同用户只能查看自身相关的数据信息，不能查看其他用户的数据信息。

```
--创建用户alice, bob, peter
gaussdb=# CREATE USER alice PASSWORD '********';
gaussdb=# CREATE USER bob PASSWORD '********';
gaussdb=# CREATE USER peter PASSWORD '********';

--创建表all_data，包含不同用户数据信息
```



```
gaussdb=# CREATE TABLE all_data(id int, role varchar(100), data varchar(100));

--向数据表插入数据
gaussdb=# INSERT INTO all_data VALUES(1, 'alice', 'alice data');
gaussdb=# INSERT INTO all_data VALUES(2, 'bob', 'bob data');
gaussdb=# INSERT INTO all_data VALUES(3, 'peter', 'peter data');

--将表all_data的读取权限赋予alice，bob和peter用户
gaussdb=# GRANT SELECT ON all_data TO alice, bob, peter;

--打开行访问控制策略开关
gaussdb=# ALTER TABLE all_data ENABLE ROW LEVEL SECURITY;

--创建行访问控制策略，当前用户只能查看用户自身的数据
gaussdb=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role = CURRENT_USER);

--查看表详细信息
gaussdb=# \d+ all_data
                        Table "public.all_data"
 Column |          Type          | Modifiers | Storage  | Stats target | Description
--------+------------------------+-----------+----------+--------------+-------------
 id     | integer                |           | plain    |              |
 role   | character varying(100) |           | extended |              |
 data   | character varying(100) |           | extended |              |
Row Level Security Policies:
    POLICY "all_data_rls" FOR ALL
      TO public
      USING (((role)::name = "current_user"()))
Has OIDs: no
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no, enable_rowsecurity=true

--切换至用户alice，执行SQL"SELECT * FROM public.all_data"
gaussdb=# SELECT * FROM public.all_data;
 id | role  |    data
----+-------+------------
  1 | alice | alice data
(1 row)

gaussdb=# EXPLAIN(COSTS OFF) SELECT * FROM public.all_data;
                       QUERY PLAN
---------------------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Seq Scan on all_data
         Filter: ((role)::name = 'alice'::name)
 Notice: This query is influenced by row level security feature
(5 rows)

--切换至用户peter，执行SQL"SELECT * FROM public.all_data"
gaussdb=# SELECT * FROM public.all_data;
 id | role  |    data
----+-------+------------
  3 | peter | peter data
(1 row)

gaussdb=#  EXPLAIN(COSTS OFF) SELECT * FROM public.all_data;
                       QUERY PLAN
---------------------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Seq Scan on all_data
         Filter: ((role)::name = 'peter'::name)
 Notice: This query is influenced by row level security feature
(5 rows)
```


须知

PG_STATISTIC系统表和PG_STATISTIC_EXT系统表存储了统计对象的一些敏感信息，如高频值MCV。若创建行级访问控制后，将这两张系统表的查询权限授予普通用户，则普通用户仍然可以通过访问这两张系统表，获取统计对象里的敏感信息。

PG_STATS、PG_EXT_STATS、PG_GTT_STATS为统计信息查询视图，校验用户对列的SELECT权限。若普通用户拥有SELECT权限且表上存在行级访问控制，可能查询到原本无法查询但刚好出现在统计信息高频值里的部分数据（非整行数据，仅是某列的一个值，统计信息是按列收集计算的）。

2.2 数据库审计

背景信息

数据库审计功能对数据库系统的安全性至关重要。数据库审计管理员可以利用审计日志信息，重现导致数据库现状的一系列事件，找出非法操作的用户、时间和内容等。

关于审计功能，用户需要了解以下几点内容：

- 审计总开关**audit_enabled**支持动态加载。在数据库运行期间修改该配置项的值会立即生效，无需重启数据库。默认值为on，表示开启审计功能。
- 除了审计总开关，各个审计项也有对应的开关。只有开关开启，对应的审计功能才能生效。
- 各审计项的开关支持动态加载。在数据库运行期间修改审计开关的值，不需要重启数据库便可生效。

目前，GaussDB支持以下审计项如表2-4所示。如需要修改具体的审计配置项，请联系管理员进行处理。

表 2-4 配置审计项

配置项	描述
用户登录、注销审计	参数：**audit_login_logout** 默认值为7，表示开启用户登录、退出的审计功能。设置为0表示关闭用户登录、退出的审计功能。不推荐设置除0和7之外的值。
数据库启动、停止、恢复和切换审计	参数：**audit_database_process** 默认值为1，表示开启数据库启动、停止、恢复和切换的审计功能。
用户锁定和解锁审计	参数：**audit_user_locked** 默认值为1，表示开启审计用户锁定和解锁功能。
用户访问越权审计	参数：**audit_user_violation** 默认值为0，表示关闭用户越权操作审计功能。
授权和回收权限审计	参数：**audit_grant_revoke** 默认值为1，表示开启审计用户权限授予和回收功能。

配置项	描述
对用户操作进行全量审计	参数：**full_audit_users** 默认值为空字符串，表示采用默认配置，未配置全量审计用户。
不需要审计的客户端名称及IP地址	参数：**no_audit_client** 默认值为空字符串，表示采用默认配置，未将客户端及IP加入审计黑名单。
数据库对象的CREATE，ALTER，DROP操作审计	参数：**audit_system_object** 默认值为67121159，表示对DATABASE、SCHEMA、USER、SQL Patch这四类数据库对象的CREATE、ALTER、DROP操作进行审计。
具体表的INSERT、UPDATE和DELETE操作审计	参数：**audit_dml_state** 默认值为0，表示关闭具体表的DML操作（SELECT除外）审计功能。
SELECT操作审计	参数：**audit_dml_state_select** 默认值为0，表示关闭SELECT操作审计功能。
COPY审计	参数：**audit_copy_exec** 默认值为1，表示开启copy操作审计功能。
执行存储过程和自定义函数的审计	参数：**audit_function_exec** 默认值为0，表示不记录执行存储过程和自定义函数的审计日志。
执行白名单内的系统函数审计	参数：**audit_system_function_exec** 默认值为0，表示不记录执行系统函数的审计日志。
SET审计	参数：**audit_set_parameter** 默认值为0，表示不记录set操作审计日志
事务ID记录	参数：**audit_xid_info** 默认值为0，表示关闭审计日志记录事务ID功能。

3 数据库使用入门

3.1 连接数据库

3.1.1 使用 gsql 连接

gsql是GaussDB自带的客户端工具。使用gsql连接数据库，可以交互式地输入、编辑、执行SQL语句。具体连接方式请参考**通过gsql连接实例**。

须知

集中式场景下：客户端工具通过连接主DN访问数据库。因此连接前，需获取主DN所在服务器的IP地址及端口号信息。正常业务使用禁止直接连接其他DN访问数据库。

3.1.2 应用程序接口

用户可以使用标准的数据库应用程序接口（如ODBC和JDBC），开发基于GaussDB的应用程序。

支持的应用程序接口

每个应用程序是一个独立的GaussDB开发项目。应用程序通过API与数据库进行交互，在避免了应用程序直接操作数据库系统的同时，增强了应用程序的可移植性、扩展性和可维护性。**表3-1**为GaussDB所支持的应用程序接口及其下载地址。

表 3-1 数据库应用程序接口

API	下载地址
ODBC	• Linux下： 驱动程序：GaussDB-Kernel_数据库版本号_操作系统版本号_64bit_Odbc.tar.gz unixODBC源码包：https://gitee.com/src-openeuler/unixODBC/blob/openEuler-22.03-LTS-SP1/unixODBC-2.3.7.tar.gz • Windows下： 驱动程序：GaussDB-Kernel_数据库版本号_Windows_Odbc.tar.gz
JDBC	• 驱动程序：GaussDB-Kernel_数据库版本号_操作系统版本号_64bit_Jdbc.tar.gz • 驱动类：org.postgresql.Driver

使用JDBC和ODBC接口连接数据库属远程连接，因此需要GaussDB已经做了支持远程连接的配置。相关操作请联系管理员处理。

更多支持的应用程序接口详细信息请参见**应用程序开发教程**。

3.2 操作数据库

本节描述使用数据库的基本操作。通过此节您可以完成创建数据库用户、创建数据库、创建表及向表中插入数据和查询表中数据等操作。

3.2.1 创建数据库用户

默认只有数据库安装时创建的管理员用户可以访问初始数据库，您还可以创建其他数据库用户账号。

```
gaussdb=# CREATE USER joe WITH PASSWORD "********";
```

当结果显示为如下信息，则表示创建成功。

```
CREATE ROLE
```

如上创建了一个用户名为joe，密码为********的用户。

如下命令为设置joe用户为系统管理员。

```
gaussdb=# GRANT ALL PRIVILEGES TO joe;
```

使用GRANT命令进行相关权限设置，具体操作请参见**GRANT**。

说明

关于数据库用户的更多信息请参见**用户及权限**。

3.2.2 创建和管理数据库

前提条件

用户必须拥有数据库创建的权限或者是数据库的系统管理员权限才能创建数据库，授予创建数据库的权限请参见**用户及权限**。

背景信息

- 初始时，GaussDB包含两个模板数据库template0、template1以及一个默认的用户数据库postgres。postgres默认的兼容数据库类型为O（即DBCOMPATIBILITY = A ），该兼容类型下将空字符串作为NULL处理。
- CREATE DATABASE实际上通过拷贝模板数据库来创建新数据库。默认情况下，拷贝template0。请避免使用客户端或其他方式连接及操作两个模板数据库。

  说明

  - 模板数据库中没有用户表，可通过系统表PG_DATABASE查看模板数据库属性。
  - 模板template0不允许用户连接，模板template1只允许数据库初始用户和系统管理员连接，普通用户无法连接。
- 数据库系统中会有多个数据库，但是客户端程序一次只能连接一个数据库。也不能在不同的数据库之间相互查询。GaussDB中存在多个数据库时，需要通过-d参数指定相应的数据库实例进行连接。

注意事项

如果数据库的编码为SQL_ASCII（可以通过“show server_encoding;”命令查看当前数据库存储编码），则在创建数据库对象时，如果对象名中含有多字节字符（例如中文），超过数据库对象名长度限制（63字节）的时候，数据库将会将最后一个字节（而不是字符）截断，可能造成出现半个字符的情况。

针对这种情况，请遵循以下条件：

- 保证数据对象的名称不超过限定长度。
- 修改数据库的默认存储编码集（server_encoding）为utf-8编码集。
- 不要使用多字节字符作为对象名。
- 因为误操作导致在多字节字符的中间截断，从而导致无法删除数据库对象，如果出现这种现象，请使用截断前的数据库对象名进行删除操作，或将该对象从各个数据库节点的相应系统表中依次删除。

操作步骤

步骤1 使用如下命令创建一个新的数据库db_tpcc。

```
gaussdb=# CREATE DATABASE db_tpcc;
CREATE DATABASE
```


说明

- 数据库名称遵循SQL标识符的一般规则。当前角色自动成为此新数据库的所有者。
- 如果一个数据库系统用于承载相互独立的用户和项目，建议把它们放在不同的数据库里。
- 如果项目或者用户是相互关联的，并且可以相互使用对方的资源，则应该把它们放在同一个数据库里，但可以规划在不同的模式中。模式只是一个纯粹的逻辑结构，某个模式的访问权限由权限系统模块控制。
- 创建数据库时，若数据库名称长度超过63字节，server端会对数据库名称进行截断，保留前63个字节，因此建议数据库名称长度不要超过63个字节。
- 数据库默认创建在pg_default表空间下。若要指定表空间，可以使用如下语句：
  ```
  gaussdb=# CREATE DATABASE db_tpcc WITH TABLESPACE = hr_local;
  CREATE DATABASE
  ```
  其中hr_local为表空间名称，关于如何创建表空间，请参见**创建和管理表空间**。
- 创建完db_tpcc数据库后，可以选择继续在默认的postgres数据库进行其他操作，也可以按如下方法退出postgres数据库，使用新用户连接到此数据库执行创建表等操作。
  ```
  gaussdb=# \q
  gsql -d db_tpcc -p 8000 -U joe
  Password for user joe:
  gsql ((GaussDB Kernel 503.1.XXX build f521c606) compiled at 2021-09-16 14:55:22 commit 2935 last mr 6385 release)
  Non-SSL connection (SSL connection is recommended when requiring high-security)
  Type "help" for help.

  db_tpcc=>
  ```

步骤2 查看数据库。

- 使用\l元命令查看数据库系统的数据库列表。
  ```
  gaussdb=# \l
  ```
- 使用如下命令通过系统表pg_database查询数据库列表。
  ```
  gaussdb=# SELECT datname FROM pg_database;
  ```

步骤3 修改数据库。

用户可以使用如下命令修改数据库属性（比如：owner、名称和默认的配置属性）。

- 使用如下命令为数据库设置默认的模式搜索路径。
  ```
  gaussdb=# ALTER DATABASE db_tpcc SET search_path TO pa_catalog,public;
  ALTER DATABASE
  ```
- 使用如下命令为数据库重新命名。
  ```
  gaussdb=# ALTER DATABASE db_tpcc RENAME TO human_tpcds;
  ALTER DATABASE
  ```

步骤4 删除数据库。

用户可以使用**DROP DATABASE**命令删除数据库。该命令删除了数据库中的系统目录，并且删除了磁盘上带有数据的数据库目录。用户必须是数据库的owner或者系统管理员才能删除数据库。当有人连接数据库时，删除操作会失败。删除数据库时请先连接到其他的数据库。

使用如下命令删除数据库。
```
gaussdb=# DROP DATABASE human_tpcds;
DROP DATABASE
```

----结束

3.2.3 创建和管理表空间

背景信息

通过使用表空间，管理员可以控制一个数据库安装的磁盘布局。这样有以下优点：

- 如果初始化数据库所在的分区或者表空间已满，又不能逻辑上扩展更多空间，可以在不同的分区上创建和使用表空间，直到系统重新配置空间。
- 表空间允许管理员根据数据库对象的使用模式安排数据位置，从而提高性能。
  - 一个频繁使用的索引可以存储在性能稳定且运算速度较快的磁盘上，比如一种固态设备。
  - 一个存储归档的数据，很少使用的或者对性能要求不高的表可以存储在一个运算速度较慢的磁盘上。
- 管理员通过表空间可以设置占用的磁盘空间，用以在和其他数据共用分区的时候，防止表空间占用相同分区上的其他空间。
- 表空间可以控制数据库数据占用的磁盘空间。当表空间所在磁盘的使用率达到90%时，数据库将被设置为只读模式，当磁盘使用率降到90%以下时，数据库将恢复到读写模式。

  建议用户使用数据库时，通过后台监控程序或者Database Manager进行磁盘空间使用率监控，以免出现数据库只读情况。
- 表空间对应于一个文件系统目录，比如数据库节点数据目录/pg_location/mount1/path1是用户拥有读写权限的空目录，那么可以在这个目录下创建一个绝对路径表空间。

  使用表空间配额管理会使性能有30%左右的影响，MAXSIZE指定每个数据库节点的配额大小，误差范围在500MB以内。请根据实际的情况确认是否需要设置表空间的最大值。

GaussDB自带了两个表空间：pg_default和pg_global。

- 默认表空间pg_default：用来存储非共享系统表、用户表、用户表index、临时表、临时表index、内部临时表的默认表空间。对应存储目录为实例数据目录下的base目录。
- 共享表空间pg_global：用来存储共享系统表的表空间。对应存储目录为实例数据目录下的global目录。

注意事项：

在公有云场景下一般不建议用户使用自定义的表空间。用户自定义表空间通常配合主存（即默认表空间所在的存储设备，如磁盘）以外的其它存储介质使用，以隔离不同业务可以使用的IO资源，而在公有云场景下，存储设备都是采用标准化的配置，无其它可用的存储介质，自定义表空间使用不当不利于系统长稳运行以及影响整体性能，因此建议使用默认表空间即可。

操作步骤

- 创建表空间

  a. 执行如下命令创建用户jack。

  ```
  gaussdb=# CREATE USER jack IDENTIFIED BY '********';
  ```

  当结果显示为如下信息，则表示创建成功。

  ```
  CREATE ROLE
  ```


b. 执行如下命令创建表空间。
```
gaussdb=# CREATE TABLESPACE fastspace RELATIVE LOCATION 'tablespace/tablespace_1';
```
当结果显示为如下信息，则表示创建成功。
```
CREATE TABLESPACE
```
其中“fastspace”为新创建的表空间，“数据库节点数据目录/*pg_location/tablespace/tablespace_1*”是用户拥有读写权限的空目录。

c. 数据库系统管理员执行如下命令将“fastspace”表空间的访问权限授予数据用户jack。
```
gaussdb=# GRANT CREATE ON TABLESPACE fastspace TO jack;
```
当结果显示为如下信息，则表示授予成功。
```
GRANT
```

- 在表空间中创建对象

  如果用户拥有表空间的CREATE权限，就可以在表空间上创建数据库对象，比如：表和索引等。

  以创建表为例。

  - 方式1：执行如下命令在指定表空间创建表。
    ```
    gaussdb=# CREATE TABLE foo(i int) TABLESPACE fastspace;
    ```
    当结果显示为如下信息，则表示创建成功。
    ```
    CREATE TABLE
    ```
  - 方式2：先使用set default_tablespace设置默认表空间，再创建表。
    ```
    gaussdb=# SET default_tablespace = 'fastspace';
    SET
    gaussdb=# CREATE TABLE foo2(i int);
    CREATE TABLE
    ```
    假设设置“fastspace”为默认表空间，然后创建表foo2。

- 查询表空间
  - 方式1：检查pg_tablespace系统表。如下命令可查到系统和用户定义的全部表空间。
    ```
    gaussdb=# SELECT spcname FROM pg_tablespace;
    ```
  - 方式2：使用gsql程序的元命令查询表空间。
    ```
    gaussdb=# \db
    ```

- 查询表空间使用率

  a. 查询表空间的当前使用情况。
  ```
  gaussdb=# SELECT PG_TABLESPACE_SIZE('example');
  ```
  返回如下信息：
  ```
   pg_tablespace_size
  --------------------
             2146304
  (1 row)
  ```
  其中2146304表示表空间的大小，单位为字节。

  b. 计算表空间使用率。

  表空间使用率=PG_TABLESPACE_SIZE/表空间所在目录的磁盘大小。

- 修改表空间命名

  执行如下命令对表空间fastspace重命名为fspace。
  ```
  gaussdb=# ALTER TABLESPACE fastspace RENAME TO fspace;
  ALTER TABLESPACE
  ```

- 删除表空间
  - 执行如下命令删除用户jack。


```
gaussdb=# DROP USER jack CASCADE;
DROP ROLE
```

- 执行如下命令删除表foo和foo2。

```
gaussdb=# DROP TABLE foo;
gaussdb=# DROP TABLE foo2;
```

当结果显示为如下信息，则表示删除成功。

```
DROP TABLE
```

- 执行如下命令删除表空间fspace。

```
gaussdb=# DROP TABLESPACE fspace;
DROP TABLESPACE
```

说明

用户必须是表空间的owner或者系统管理员才能删除表空间。

3.2.4 创建和管理表

3.2.4.1 创建表

背景信息

表是建立在数据库中的，在不同的数据库中可以存放相同的表，甚至可以通过使用模式在同一个数据库中创建相同名称的表。

创建表

执行如下命令创建表。

```
gaussdb=# CREATE TABLE customer_t1
(
    c_customer_sk           integer,
    c_customer_id           char(5),
    c_first_name            char(6),
    c_last_name             char(8)
);
```

当结果显示为如下信息，则表示创建成功。

```
CREATE TABLE
```

其中c_customer_sk 、c_customer_id、c_first_name和c_last_name是表的字段名，integer、char(5)、char(6)和char(8)分别是这四个字段名称的类型。

说明

- 默认情况下，新的数据库对象是创建在“$user”模式下。关于模式的更多信息请参见**创建和管理schema**。
- 除了创建的表以外，数据库还包含很多系统表。这些系统表包含数据库安装信息以及GaussDB上运行的各种查询和进程的信息。可以通过查询系统表来收集有关数据库的信息。请参见**查看系统表**。
- 关于创建表的更多信息请参见**CREATE TABLE**。

3.2.4.2 向表中插入数据

在创建一个表后，表中并没有数据，在使用这个表之前，需要向表中插入数据。本小节介绍如何使用**INSERT**命令插入一行或多行数据，及从指定表插入数据。如果有大量数据需要批量导入表中请联系管理员处理。

背景信息

服务端与客户端使用不同的字符集时，两者字符集中单个字符的长度也会不同，客户端输入的字符串会以服务端字符集的格式进行处理，所以产生的最终结果可能会与预期不一致。

表 3-2 客户端和服务端设置字符集的输出结果对比

操作过程	服务端和客户端编码一致	服务端和客户端编码不一致
存入和取出过程中没有对字符串进行操作	输出预期结果	输出预期结果（输入与显示的客户端编码必须一致）。
存入取出过程对字符串有做一定的操作（如字符串函数操作）	输出预期结果	根据对字符串具体操作可能产生非预期结果。
存入过程中对超长字符串有截断处理	输出预期结果	字符集中字符编码长度是否一致，如果不一致可能会产生非预期的结果。

上述字符串函数操作和自动截断产生的效果会有叠加效果，例如：在客户端与服务端字符集不一致的场景下，如果既有字符串操作，又有字符串截断，在字符串被处理完以后的情况下继续截断，这样也会产生非预期的效果。详细的示例请参见表3-3。

说明

数据库DBCOMPATIBILITY设为兼容TD模式，且GUC参数**td_compatible_truncation**设置为on的情况下，才会对超长字符串进行截断。

执行如下命令建立示例中需要使用的表table1、table2。

```
gaussdb=# CREATE TABLE table1(id int, a char(6), b varchar(6),c varchar(6));
gaussdb=# CREATE TABLE table2(id int, a char(20), b varchar(20),c varchar(20));
```

表 3-3 示例

编号	服务端字符集	客户端字符集	是否启用自动截断	示例	结果	说明
1	SQL_ASCII	UTF8	是	gaussdb=# **INSERT INTO** *table1* **VALUES**(*1,reverse('123A A 78'),reverse('123A A 78'),reverse('123A A 78')*);	id |a|b|c ----+------ +------+------ 1 | 87| 87| 87	字符串在服务端翻转后，并进行截断，由于服务端和客户端的字符集不一致，字符A在客户端由多个字节表示，结果产生异常。

编号	服务端字符集	客户端字符集	是否启用自动截断	示例	结果	说明
2	SQL_ASCII	UTF8	是	gaussdb=# **INSERT INTO** *table1* **VALUES**(*2,reverse('123A 78'),reverse('123A 78'),reverse('123A 78')*);	id |a|b|c ----+------+------+------ 2 | 873| 873| 873	字符串翻转后，又进行了自动截断，所以产生了非预期的效果。
3	SQL_ASCII	UTF8	是	gaussdb=# **INSERT INTO** *table1* **VALUES**(*3,'87A 123','87A 123','87A 123'*);	id | a | b | c ----+-------+-------+------- 3 | 87A1 | 87A1 | 87A1	字符串类型的字段长度是客户端字符编码长度的整数倍，所以截断后产生结果正常。
4	SQL_ASCII	UTF8	否	gaussdb=# **INSERT INTO** *table2* **VALUES**(*1,reverse('123A A 78'),reverse('123A A 78'),reverse('123A A 78')*); gaussdb=# **INSERT INTO** *table2* **VALUES**(*2,reverse('123A 78'),reverse('123A 78'),reverse('123A 78')*);	id |a|b|c ---- +--------------------+-------- +-------- 1 | 87 321| 87 321 | 87 321 2 | 87321| 87321| 87321	与示例1类似，多字节字符翻转之后不再表示原来的字符。

操作步骤

向表中插入数据前，意味着表已创建成功。创建表的步骤请参见**创建和管理表**。

- 向表customer_t1中插入一行数据。

  数据值是按照这些字段在表中出现的顺序列出的，并且用逗号分隔。通常数据值是文本（常量），但也允许使用标量表达式。

  ```
  gaussdb=# INSERT INTO customer_t1(c_customer_sk, c_customer_id, c_first_name) VALUES (3769, 'hello', 'Grace');
  ```

  如果用户已经知道表中字段的顺序，也可无需列出表中的字段。例如以下命令与上面的命令效果相同。

  ```
  gaussdb=# INSERT INTO customer_t1 VALUES (3769, 'hello', 'Grace');
  ```

  如果用户不知道所有字段的数值，可以忽略其中的一些。没有数值的字段将被填充为字段的缺省值。例如：

  ```
  gaussdb=# INSERT INTO customer_t1 (c_customer_sk, c_first_name) VALUES (3769, 'Grace');

  gaussdb=# INSERT INTO customer_t1 VALUES (3769, 'hello');
  ```

  用户也可以对独立的字段或者整个行明确缺省值：

  ```
  gaussdb=# INSERT INTO customer_t1 (c_customer_sk, c_customer_id, c_first_name) VALUES (3769, 'hello', DEFAULT);

  gaussdb=# INSERT INTO customer_t1 DEFAULT VALUES;
  ```

- 如果需要在表中插入多行，请执行如下命令：

  ```
  gaussdb=# INSERT INTO customer_t1 (c_customer_sk, c_customer_id, c_first_name) VALUES
      (6885, 'maps', 'Joes'),
  ```

```
    (4321, 'tpcds', 'Lily'),
    (9527, 'world', 'James');
```

如果需要向表中插入多条数据，除此命令外，也可以多次执行插入一行数据命令实现。但是建议使用此命令可以提升效率。

- 如果从指定表插入数据到当前表，例如在数据库中创建了一个表customer_t1的备份表customer_t2，现在需要将表customer_t1中的数据插入到表customer_t2中，则可以执行如下命令：

```
gaussdb=# CREATE TABLE customer_t2
(
    c_customer_sk             integer,
    c_customer_id             char(5),
    c_first_name              char(6),
    c_last_name               char(8)
);

gaussdb=# INSERT INTO customer_t2 SELECT * FROM customer_t1;
```

  说明

  从指定表插入数据到当前表时，若指定表与当前表对应的字段数据类型之间不存在隐式转换，则这两种数据类型必须相同。

- 删除备份表。

```
gaussdb=# DROP TABLE customer_t2 CASCADE;
```

  说明

  在删除表的时候，若当前需删除的表与其他表有依赖关系，需先删除关联的表，然后再删除当前表。

3.2.4.3 更新表中数据

修改已经存储在数据库中数据的行为叫做更新。用户可以更新单独一行、所有行或者指定的部分行。还可以独立更新每个字段，而其他字段则不受影响。

使用UPDATE命令更新现有行，需要提供以下三种信息：

- 表的名称和要更新的字段名
- 字段的新值
- 要更新的行

SQL通常不会为数据行提供唯一标识，因此无法直接声明需要更新哪一行。但是可以通过声明一个被更新的行必须满足的条件。只有在表里存在主键的时候，才可以通过主键指定一个独立的行。

建立表和插入数据的步骤请参见**创建表**和**向表中插入数据**。

需要将表customer_t1中c_customer_sk为9527的地域重新定义为9876：

```
gaussdb=# UPDATE customer_t1 SET c_customer_sk = 9876 WHERE c_customer_sk = 9527;
```

这里的表名称也可以使用模式名修饰，否则会从默认的模式路径找到这个表。SET后面紧跟字段和新的字段值。新的字段值不仅可以是常量，也可以是变量表达式。

比如，把所有c_customer_sk的值增加100：

```
gaussdb=# UPDATE customer_t1 SET c_customer_sk = c_customer_sk + 100;
```

在这里省略了WHERE子句，表示表中的所有行都要被更新。如果出现了WHERE子句，那么只有匹配其条件的行才会被更新。

在SET子句中的等号是一个赋值，而在WHERE子句中的等号是比较。WHERE条件不一定是相等比较，许多其他的操作符也可以使用。

用户可以在一个UPDATE命令中更新更多的字段，方法是在SET子句中列出更多赋值，比如：

```
gaussdb=# UPDATE customer_t1 SET  c_customer_id = 'Admin', c_first_name = 'Local' WHERE c_customer_sk = 4421;
```

批量更新或删除数据后，会在数据文件中产生大量的删除标记，查询过程中标记删除的数据也是需要扫描的。故多次批量更新/删除后，标记删除的数据量过大会严重影响查询的性能。建议在批量更新/删除业务会反复执行的场景下，定期执行VACUUM FULL以保持查询性能。

3.2.4.4 查看数据

- 使用系统表pg_tables查询数据库所有表的信息。
  ```
  gaussdb=# SELECT * FROM pg_tables;
  ```
- 使用gsql的\d+命令查询表的属性。
  ```
  gaussdb=# \d+ customer_t1;
  ```
- 执行如下命令查询表customer_t1的数据量。
  ```
  gaussdb=# SELECT count(*) FROM customer_t1;
  ```
- 执行如下命令查询表customer_t1的所有数据。
  ```
  gaussdb=# SELECT * FROM customer_t1;
  ```
- 执行如下命令只查询字段c_customer_sk的数据。
  ```
  gaussdb=# SELECT c_customer_sk FROM customer_t1;
  ```
- 执行如下命令过滤字段c_customer_sk的重复数据。
  ```
  gaussdb=# SELECT DISTINCT( c_customer_sk ) FROM customer_t1;
  ```
- 执行如下命令查询字段c_customer_sk为3869的所有数据。
  ```
  gaussdb=# SELECT * FROM customer_t1 WHERE c_customer_sk = 3869;
  ```
- 执行如下命令按照字段c_customer_sk进行排序。
  ```
  gaussdb=# SELECT * FROM customer_t1 ORDER BY c_customer_sk;
  ```

3.2.4.5 删除表中数据

在使用表的过程中，可能会需要删除已过期的数据，删除数据必须从表中整行的删除。

SQL不能直接访问独立的行，只能通过声明被删除行匹配的条件进行。如果表中有一个主键，用户可以指定准确的行。用户可以删除匹配条件的一组行或者一次删除表中的所有行。

使用DELETE命令删除行，如果删除表customer_t1中所有c_customer_sk为3869的记录。

```
gaussdb=# DELETE FROM customer_t1 WHERE c_customer_sk = 3869;
```

如果执行如下命令之一，会删除表中所有的行。

```
gaussdb=# DELETE FROM customer_t1;
```

或：

```
gaussdb=# TRUNCATE TABLE customer_t1;
```

说明

全表删除的场景下，建议使用TRUNCATE，不建议使用DELETE。

删除创建的表。

```
gaussdb=# DROP TABLE customer_t1;
```

3.2.5 查看系统表

除了创建的表以外，数据库还包含很多系统表。这些系统表包含数据库安装信息以及GaussDB上运行的各种查询和进程的信息。可以通过查询系统表来获取有关数据库的信息。

“**查看系统表和系统视图**”中每个表的说明指出了表是对所有用户可见还是只对初始化用户可见。以初始化用户身份登录才能查询只对初始化用户可见的表。

GaussDB提供了以下类型的系统表和系统视图：

- 兼容PostgreSQL的系统表和系统视图
  这类系统表和系统视图具有PG前缀。
- GaussDB新增的系统表和系统视图
  这类系统表和系统视图具有GS前缀。

查看数据库中包含的表

在public Schema下新建表格。

```
gaussdb=#  CREATE TABLE public.search_table_t1(a int);
CREATE TABLE
gaussdb=#  CREATE TABLE public.search_table_t2(b int);
CREATE TABLE
gaussdb=#  CREATE TABLE public.search_table_t3(c int);
CREATE TABLE
gaussdb=#  CREATE TABLE public.search_table_t4(d int);
CREATE TABLE
gaussdb=#  CREATE TABLE public.search_table_t5(e int);
CREATE TABLE
```

在PG_TABLES系统表中查看public Schema中包含的前缀为search_table的表。

```
gaussdb=#  SELECT distinct(tablename) FROM pg_tables WHERE SCHEMANAME = 'public' AND
TABLENAME LIKE 'search_table%';
```

结果如下：

```
   tablename
-----------------
 search_table_t1
 search_table_t2
 search_table_t3
 search_table_t4
 search_table_t5
(5 rows)
```

查看数据库用户

通过PG_USER可以查看数据库中所有用户的列表，还可以查看用户ID（USESYSID）和用户权限。

```
SELECT * FROM pg_user;
 usename | usesysid | usecreatedb | usesuper | usecatupd | userepl |   passwd   | valbegin | valuntil |
respool    | parent | spacelimit | useconfig | no
degroup | tempspacelimit | spillspacelimit | usemonitoradmin | useoperatoradmin | usepolicyadmin
---------+----------+-------------+----------+-----------+---------+----------+----------+----------+-------------
+--------+------------+-----------+---
--------+----------------+-----------------+-----------------+------------------+----------------
 omm     |       10 | t           | t        | t         | t       | ******** |          |          | default_pool |      0 |
```



```
|       |
     |           |           | t           | t           | t
```

查看和停止正在运行的查询语句

通过视图**PG_STAT_ACTIVITY**可以查看正在运行的查询语句。方法如下：

步骤1 设置参数track_activities为on。

```
SET track_activities = on;
```

当此参数为on时，数据库系统才会获取当前活动查询的运行信息。

步骤2 查看正在运行的查询语句。以查看正在运行的查询语句所连接的数据库名、执行查询的用户、查询状态及查询对应的PID为例。

```
SELECT datname, usename, state,pid FROM pg_stat_activity;
 datname  | usename | state  |       pid
----------+---------+--------+-----------------
 testdb | Ruby    | active | 140298793514752
 testdb | Ruby    | active | 140298718004992
 testdb | Ruby    | idle   | 140298650908416
 testdb | Ruby    | idle   | 140298625742592
 testdb | omm | active | 140298575406848
(5 rows)
```

如果state字段显示为idle，则表明此连接处于空闲，等待用户输入命令。

如果仅需要查看非空闲的查询语句，则使用如下命令查看：

```
SELECT datname, usename, state, pid FROM pg_stat_activity WHERE state != 'idle';
```

步骤3 若需要取消运行时间过长的查询，通过PG_TERMINATE_BACKEND函数，根据线程ID结束会话，请执行如下命令。

```
SELECT PG_TERMINATE_BACKEND(140298793514752);
```

显示如下信息，表示结束会话成功。

```
PG_TERMINATE_BACKEND
----------------------
 t
(1 row)
```

显示如下信息，表示用户执行了结束当前会话的操作。

```
FATAL:  terminating connection due to administrator command
FATAL:  terminating connection due to administrator command
```

说明

1. gsql客户端使用PG_TERMINATE_BACKEND函数结束当前正在执行会话的后台线程时，如果当前的用户是初始用户，客户端不会退出而是自动重连，即返回“The connection to the server was lost. Attempting reset: Succeeded.”。否则客户端会重连失败，即返回“The connection to the server was lost. Attempting reset: Failed.”。这是因为只有初始用户可以免密登录，普通用户不能免密登录，从而重连失败。
2. 对于使用PG_TERMINATE_BACKEND函数结束非活跃的后台线程时。如果打开了线程池，此时空闲的会话没有线程ID，无法结束会话。非线程池模式下，结束的会话不会自动重连。

```
FATAL:  terminating connection due to administrator command
FATAL:  terminating connection due to administrator command
The connection to the server was lost. Attempting reset: Succeeded.
```

----结束

3.2.6 其他操作

3.2.6.1 创建和管理 schema

背景信息

schema又称作模式。通过管理schema，允许多个用户使用同一数据库而不相互干扰，可以将数据库对象组织成易于管理的逻辑组，同时便于将第三方应用添加到相应的schema下而不引起冲突。管理schema包括：创建schema、使用schema、删除schema、设置schema的搜索路径以及schema的权限控制。

注意事项

- GaussDB包含一个或多个已命名数据库。用户和用户组在数据库范围内是共享的，但是其数据并不共享。任何与服务器连接的用户都只能访问连接请求里声明的数据库。
- 一个数据库可以包含一个或多个已命名的schema，schema又包含表及其他数据库对象，包括数据类型、函数和操作符等。同一对象名可以在不同的schema中使用而不会引起冲突。例如，schema1和schema2都可以包含一个名为mytable的表。
- 和数据库不同，schema不是严格分离的。用户根据其对schema的权限，可以访问所连接数据库的schema中的对象。进行schema权限管理首先需要对数据库的权限控制进行了解。
- 不能创建以PG_为前缀的schema名，该类schema为数据库系统预留的。
- 在每次创建新用户时，系统会在当前登录的数据库中为新用户创建一个同名schema。对于其他数据库，若需要同名schema，则需要用户手动创建。
- 通过未修饰的表名（名称中只含有表名，没有“schema名”）引用表时，系统会通过search_path（搜索路径）来判断该表是哪个schema下的表。pg_temp和pg_catalog始终会作为搜索路径顺序中的前两位，无论二者是否出现在search_path中，或者出现在search_path中的任何位置。search_path（搜索路径）是一个schema名列表，在其中找到的第一个表就是目标表，如果没有找到则报错。（某个表即使存在，如果它的schema不在search_path中，依然会查找失败）在搜索路径中的第一个schema叫做"当前schema"。它是搜索时查询的第一个schema，同时在没有声明schema名时，新创建的数据库对象会默认存放在该schema下。
- 每个数据库都包含一个pg_catalog schema，它包含系统表和所有内置数据类型、函数和操作符。pg_catalog是搜索路径中的一部分，始终在临时表所属的模式后面，并在search_path中所有模式的前面，即具有第二搜索优先级，以确保可以搜索到数据库内置对象。如果用户需要使用和系统内置对象重名的自定义对象时，可以在操作自定义对象时带上自己的模式。

操作步骤

- 创建管理用户及权限schema
  - 执行如下命令创建一个schema。
    ```
    gaussdb=# CREATE SCHEMA myschema;
    ```
    当结果显示为如下信息，则表示成功创建一个名为myschema的schema。
    ```
    CREATE SCHEMA
    ```
    如果需要在模式中创建或者访问对象，其完整的对象名称由模式名称和具体的对象名称组成，中间由符号“.”隔开。例如：myschema.table。
  - 执行如下命令在创建schema时指定owner。
    ```
    gaussdb=# CREATE SCHEMA myschema AUTHORIZATION omm;
    ```


当结果显示为如下信息，则表示成功创建一个属于omm用户，名为myschema的schema。

```
CREATE SCHEMA
```

- 使用schema

  在特定schema下创建对象或者访问特定schema下的对象，需要使用有schema修饰的对象名。该名称包含schema名以及对象名，它们之间用“.”号分开。

  - 执行如下命令在myschema下创建mytable表。

    ```
    gaussdb=# CREATE TABLE myschema.mytable(id int, name varchar(20));
    CREATE TABLE
    ```

    如果在数据库中指定对象的位置，就需要使用有schema修饰的对象名称。

  - 执行如下命令查询myschema下mytable表的所有数据。

    ```
    gaussdb=# SELECT * FROM myschema.mytable;
    id | name
    ----+------
    (0 rows)
    ```

- schema的搜索路径

  可以设置search_path配置参数指定查询对象可用schema的顺序。在搜索路径列出的第一个schema会变成默认的schema。如果在创建对象时不指定schema，则会创建在默认的schema中。

  - 执行如下命令查看搜索路径。

    ```
    gaussdb=# SHOW SEARCH_PATH;
     search_path
    ----------------
     "$user",public
    (1 row)
    ```

  - 执行如下命令将搜索路径设置为myschema, public。首先搜索myschema，然后搜索public。

    ```
    gaussdb=# SET SEARCH_PATH TO myschema, public;
    SET
    ```

- schema的权限控制

  默认情况下，用户只能访问属于自己的schema中的数据库对象。如果需要访问其他schema的对象，则该schema的所有者应该授予他对该schema的usage权限。

  通过将模式的CREATE权限授予某用户，被授权用户就可以在此模式中创建对象。默认情况下，所有角色都拥有在public模式上的usage权限，但是普通用户没有在public模式上的CREATE权限。普通用户能够连接到一个指定数据库并在它的public模式中创建对象是不安全的，如果普通用户具有在public模式上的CREATE权限则建议通过如下语句撤销该权限。

  - 撤销PUBLIC在public模式下创建对象的权限，下面语句中第一个“public”指的是模式，第二个“PUBLIC”指的是所有角色。

    ```
    gaussdb=# REVOKE CREATE ON SCHEMA public FROM PUBLIC;
    REVOKE
    ```

  - 执行如下命令查看现有的schema：

    ```
    gaussdb=# SELECT current_schema();
     current_schema
    ----------------
     myschema
    (1 row)
    ```

  - 执行如下命令创建用户jack，并将myschema的usage权限授予用户jack。

    ```
    gaussdb=# CREATE USER jack IDENTIFIED BY '********';
    CREATE ROLE
    gaussdb=# GRANT USAGE ON schema myschema TO jack;
    GRANT
    ```

  - 将用户jack对于myschema的usage权限收回。


```
gaussdb=# REVOKE USAGE ON schema myschema FROM jack;
REVOKE
```

- 删除schema
  - 当schema为空时，即该schema下没有数据库对象，使用DROP SCHEMA命令进行删除。例如删除名为nullschema的空schema。
    ```
    gaussdb=# DROP SCHEMA IF EXISTS nullschema;
    DROP SCHEMA
    ```
  - 当schema非空时，如果要删除一个schema及其包含的所有对象，需要使用CASCADE关键字。例如删除myschema及该schema下的所有对象。
    ```
    gaussdb=# DROP SCHEMA myschema CASCADE;
    DROP SCHEMA
    ```
  - 执行如下命令删除用户jack。
    ```
    gaussdb=# DROP USER jack;
    DROP ROLE
    ```

3.2.6.2 创建和管理分区表

背景信息

GaussDB数据库支持的分区表为范围分区表、间隔分区表、列表分区表和哈希分区表。

- 范围分区表：将数据基于范围映射到每一个分区，这个范围是由创建分区表时指定的分区键决定的。这种分区方式是最为常用的，并且分区键经常采用日期，例如将销售数据按照月份进行分区。
- 间隔分区表：是一种特殊的范围分区表，相比范围分区表，新增间隔值定义，当插入记录找不到匹配的分区时，可以根据间隔值自动创建分区。
- 列表分区表：将数据中包含的键值分别存储在不同的分区中，依次将数据映射到每一个分区，分区中包含的键值由创建分区表时指定。
- 哈希分区表：将数据根据内部哈希算法依次映射到每一个分区中，包含的分区个数由创建分区表时指定。

分区表和普通表相比具有以下优点：

- 改善查询性能：对分区对象的查询可以仅搜索自己关心的分区，提高检索效率。
- 增强可用性：如果分区表的某个分区出现故障，表在其他分区的数据仍然可用。
- 方便维护：如果分区表的某个分区出现故障，需要修复数据，只修复该分区即可。

普通表若要转成分区表，需要新建分区表，然后把普通表中的数据导入到新建的分区表中。因此在初始设计表时，请根据业务提前规划是否使用分区表。

操作步骤

示例一：使用默认表空间

- 创建分区表（假设用户已创建tpcds schema）
  ```
  gaussdb=# CREATE TABLE tpcds.customer_address
  (
      ca_address_sk       integer                 NOT NULL   ,
      ca_address_id       character(16)           NOT NULL   ,
      ca_street_number    character(10)                      ,
      ca_street_name      character varying(60)              ,
      ca_street_type      character(15)                      ,
      ca_suite_number     character(10)                      ,
      ca_city             character varying(60)              ,
  ```

```
    ca_county           character varying(30)             ,
    ca_state            character(2)                      ,
    ca_zip              character(10)                     ,
    ca_country          character varying(20)             ,
    ca_gmt_offset       numeric(5,2)                      ,
    ca_location_type    character(20)
)

PARTITION BY RANGE (ca_address_sk)
(
        PARTITION P1 VALUES LESS THAN(5000),
        PARTITION P2 VALUES LESS THAN(10000),
        PARTITION P3 VALUES LESS THAN(15000),
        PARTITION P4 VALUES LESS THAN(20000),
        PARTITION P5 VALUES LESS THAN(25000),
        PARTITION P6 VALUES LESS THAN(30000),
        PARTITION P7 VALUES LESS THAN(40000),
        PARTITION P8 VALUES LESS THAN(MAXVALUE)
)
ENABLE ROW MOVEMENT;
```

当结果显示为如下信息，则表示创建成功。

```
CREATE TABLE
```

- 插入数据

  将表tpcds.customer_address的数据插入到表tpcds.web_returns_p2中。

  例如在数据库中创建了一个表tpcds.customer_address的备份表tpcds.web_returns_p2，现在需要将表tpcds.customer_address中的数据插入到表tpcds.web_returns_p2中，则可以执行如下命令。

```
gaussdb=# CREATE TABLE tpcds.web_returns_p2
(
    ca_address_sk       integer                 NOT NULL   ,
    ca_address_id       character(16)           NOT NULL   ,
    ca_street_number    character(10)                      ,
    ca_street_name      character varying(60)              ,
    ca_street_type      character(15)                      ,
    ca_suite_number     character(10)                      ,
    ca_city             character varying(60)              ,
    ca_county           character varying(30)              ,
    ca_state            character(2)                       ,
    ca_zip              character(10)                      ,
    ca_country          character varying(20)              ,
    ca_gmt_offset       numeric(5,2)                       ,
    ca_location_type    character(20)
)

PARTITION BY RANGE (ca_address_sk)
(
        PARTITION P1 VALUES LESS THAN(5000),
        PARTITION P2 VALUES LESS THAN(10000),
        PARTITION P3 VALUES LESS THAN(15000),
        PARTITION P4 VALUES LESS THAN(20000),
        PARTITION P5 VALUES LESS THAN(25000),
        PARTITION P6 VALUES LESS THAN(30000),
        PARTITION P7 VALUES LESS THAN(40000),
        PARTITION P8 VALUES LESS THAN(MAXVALUE)
)
ENABLE ROW MOVEMENT;
CREATE TABLE
gaussdb=# INSERT INTO tpcds.web_returns_p2 SELECT * FROM tpcds.customer_address;
INSERT 0 0
```

- 修改分区表行迁移属性

```
gaussdb=# ALTER TABLE tpcds.web_returns_p2 DISABLE ROW MOVEMENT;
ALTER TABLE
```

- 删除分区

  删除分区P8。


```
gaussdb=# ALTER TABLE tpcds.web_returns_p2 DROP PARTITION P8;
ALTER TABLE
```

- 增加分区

  增加分区P8，范围为 40000<= P8<MAXVALUE。

```
gaussdb=# ALTER TABLE tpcds.web_returns_p2 ADD PARTITION P8 VALUES LESS THAN
(MAXVALUE);
ALTER TABLE
```

- 重命名分区
  - 重命名分区P8为P_9。

```
gaussdb=# ALTER TABLE tpcds.web_returns_p2 RENAME PARTITION P8 TO P_9;
ALTER TABLE
```

  - 重命名分区P_9为P8。

```
gaussdb=# ALTER TABLE tpcds.web_returns_p2 RENAME PARTITION FOR (40000) TO P8;
ALTER TABLE
```

- 查询分区

  查询分区P6。

```
gaussdb=# SELECT * FROM tpcds.web_returns_p2 PARTITION (P6);
gaussdb=# SELECT * FROM tpcds.web_returns_p2 PARTITION FOR (35888);
```

- 删除分区表和表空间

```
gaussdb=# DROP TABLE tpcds.customer_address;
DROP TABLE
gaussdb=# DROP TABLE tpcds.web_returns_p2;
DROP TABLE
```

示例二：使用用户自定义表空间（假设用户已创建tpcds schema）

按照以下方式对范围分区表进行操作。

- 创建表空间

```
gaussdb=# CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace1/tablespace_1';
gaussdb=# CREATE TABLESPACE example2 RELATIVE LOCATION 'tablespace2/tablespace_2';
gaussdb=# CREATE TABLESPACE example3 RELATIVE LOCATION 'tablespace3/tablespace_3';
gaussdb=# CREATE TABLESPACE example4 RELATIVE LOCATION 'tablespace4/tablespace_4';
```

  当结果显示为如下信息，则表示创建成功。

```
CREATE TABLESPACE
```

- 创建分区表

```
gaussdb=# CREATE TABLE tpcds.customer_address
(
    ca_address_sk       integer                 NOT NULL   ,
    ca_address_id       character(16)           NOT NULL   ,
    ca_street_number    character(10)                      ,
    ca_street_name      character varying(60)              ,
    ca_street_type      character(15)                      ,
    ca_suite_number     character(10)                      ,
    ca_city             character varying(60)              ,
    ca_county           character varying(30)              ,
    ca_state            character(2)                       ,
    ca_zip              character(10)                      ,
    ca_country          character varying(20)              ,
    ca_gmt_offset       numeric(5,2)                       ,
    ca_location_type    character(20)
)
TABLESPACE example1

PARTITION BY RANGE (ca_address_sk)
(
        PARTITION P1 VALUES LESS THAN(5000),
        PARTITION P2 VALUES LESS THAN(10000),
        PARTITION P3 VALUES LESS THAN(15000),
        PARTITION P4 VALUES LESS THAN(20000),
        PARTITION P5 VALUES LESS THAN(25000),
```



```
    PARTITION P6 VALUES LESS THAN(30000),
    PARTITION P7 VALUES LESS THAN(40000),
    PARTITION P8 VALUES LESS THAN(MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;
```

当结果显示为如下信息，则表示创建成功。

```
CREATE TABLE
```

- 插入数据

  将表tpcds.customer_address的数据插入到表tpcds.web_returns_p2中。

  例如在数据库中创建了一个表tpcds.customer_address的备份表tpcds.web_returns_p2，现在需要将表tpcds.customer_address中的数据插入到表tpcds.web_returns_p2中，则可以执行如下命令。

```
gaussdb=# CREATE TABLE tpcds.web_returns_p2
(
   ca_address_sk       integer               NOT NULL   ,
   ca_address_id       character(16)           NOT NULL   ,
   ca_street_number    character(10)                     ,
   ca_street_name      character varying(60)             ,
   ca_street_type      character(15)                     ,
   ca_suite_number     character(10)                     ,
   ca_city             character varying(60)             ,
   ca_county           character varying(30)             ,
   ca_state            character(2)                      ,
   ca_zip              character(10)                     ,
   ca_country          character varying(20)             ,
   ca_gmt_offset       numeric(5,2)                      ,
   ca_location_type    character(20)
)
TABLESPACE example1
PARTITION BY RANGE (ca_address_sk)
(
    PARTITION P1 VALUES LESS THAN(5000),
    PARTITION P2 VALUES LESS THAN(10000),
    PARTITION P3 VALUES LESS THAN(15000),
    PARTITION P4 VALUES LESS THAN(20000),
    PARTITION P5 VALUES LESS THAN(25000),
    PARTITION P6 VALUES LESS THAN(30000),
    PARTITION P7 VALUES LESS THAN(40000),
    PARTITION P8 VALUES LESS THAN(MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;
CREATE TABLE
gaussdb=# INSERT INTO tpcds.web_returns_p2 SELECT * FROM tpcds.customer_address;
INSERT 0 0
```

- 修改分区表行迁移属性

```
gaussdb=# ALTER TABLE tpcds.web_returns_p2 DISABLE ROW MOVEMENT;
ALTER TABLE
```

- 删除分区

  删除分区P8。

```
gaussdb=# ALTER TABLE tpcds.web_returns_p2 DROP PARTITION P8;
ALTER TABLE
```

- 增加分区

  增加分区P8，范围为 40000<= P8<MAXVALUE。

```
gaussdb=# ALTER TABLE tpcds.web_returns_p2 ADD PARTITION P8 VALUES LESS THAN
(MAXVALUE);
ALTER TABLE
```

- 重命名分区
  - 重命名分区P8为P_9。

```
gaussdb=# ALTER TABLE tpcds.web_returns_p2 RENAME PARTITION P8 TO P_9;
ALTER TABLE
```


  - 重命名分区P_9为P8。
    ```
    gaussdb=# ALTER TABLE tpcds.web_returns_p2 RENAME PARTITION FOR (40000) TO P8;
    ALTER TABLE
    ```

- 修改分区的表空间
  - 修改分区P6的表空间为example3。
    ```
    gaussdb=#  ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P6 TABLESPACE example3;
    ALTER TABLE
    ```
  - 修改分区P4的表空间为example4。
    ```
    gaussdb=#  ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P4 TABLESPACE example4;
    ALTER TABLE
    ```
- 查询分区

  查询分区P6。
  ```
  gaussdb=# SELECT * FROM tpcds.web_returns_p2 PARTITION (P6);
  gaussdb=# SELECT * FROM tpcds.web_returns_p2 PARTITION FOR (35888);
  ```
- 删除分区表和表空间
  ```
  gaussdb=# DROP TABLE tpcds.customer_address;
  DROP TABLE
  gaussdb=# DROP TABLE tpcds.web_returns_p2;
  DROP TABLE
  gaussdb=# DROP TABLESPACE example1;
  gaussdb=# DROP TABLESPACE example2;
  gaussdb=# DROP TABLESPACE example3;
  gaussdb=# DROP TABLESPACE example4;
  DROP TABLESPACE
  ```

3.2.6.3 创建和管理索引

背景信息

索引可以提高数据的访问速度，但同时也增加了插入、更新和删除操作的处理时间。所以是否要为表增加索引，索引建立在哪些字段上，是创建索引前必须要考虑的问题。需要分析应用程序的业务处理、数据使用、经常被用作查询的条件或者被要求排序的字段来确定是否建立索引。

索引建立在数据库表中的某些列上。因此，在创建索引时，应该仔细考虑在哪些列上创建索引。

- 在经常需要搜索查询的列上创建索引，可以加快搜索的速度。
- 在作为主键的列上创建索引，强制该列的唯一性和组织表中数据的排列结构。
- 在经常使用连接的列上创建索引，可以加快连接的速度。
- 在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，其指定的范围是连续的。
- 在经常需要排序的列上创建索引，因为索引已经排序，这样查询可以利用索引的排序，加快排序查询时间。
- 在经常使用WHERE子句的列上创建索引，加快条件的判断速度。
- 在经常出现关键字ORDER BY、GROUP BY、DISTINCT后面的字段建立索引。

说明

- 索引创建成功后，系统会自动判断何时引用索引。当系统认为使用索引比顺序扫描更快时，就会使用索引。
- 索引创建成功后，必须和表保持同步以保证能够准确地找到新数据，这样就增加了数据操作的负荷。因此请定期删除无用的索引。
- 分区表索引分为LOCAL索引与GLOBAL索引，一个LOCAL索引对应一个具体分区，而GLOBAL索引则对应整个分区表。

- 在开启逻辑复制的场景下，如需创建包含系统列的主键索引，必须将该表的REPLICA IDENTITY属性设置为FULL或是使用USING INDEX指定不包含系统列的、唯一的、非局部的、不可延迟的和仅包括标记为NOT NULL的列的索引。

操作步骤

创建分区表的步骤请参见**创建和管理分区表**。

- 创建索引
  - 创建分区表LOCAL索引tpcds_web_returns_p2_index1，不指定索引分区的名称。

    ```
    gaussdb=# CREATE INDEX tpcds_web_returns_p2_index1 ON tpcds.web_returns_p2
    (ca_address_id) LOCAL;
    ```

    当结果显示为如下信息，则表示创建成功。

    ```
    CREATE INDEX
    ```

  - 创建分区表LOCAL索引tpcds_web_returns_p2_index2，并指定索引分区的名称。

    ```
    gaussdb=# CREATE INDEX tpcds_web_returns_p2_index2 ON tpcds.web_returns_p2
    (ca_address_sk) LOCAL
    (
        PARTITION web_returns_p2_P1_index,
        PARTITION web_returns_p2_P2_index TABLESPACE example3,
        PARTITION web_returns_p2_P3_index TABLESPACE example4,
        PARTITION web_returns_p2_P4_index,
        PARTITION web_returns_p2_P5_index,
        PARTITION web_returns_p2_P6_index,
        PARTITION web_returns_p2_P7_index,
        PARTITION web_returns_p2_P8_index
    ) TABLESPACE example2;
    ```

    当结果显示为如下信息，则表示创建成功。

    ```
    CREATE INDEX
    ```

  - 创建分区表GLOBAL索引tpcds_web_returns_p2_global_index。

    ```
    CREATE INDEX tpcds_web_returns_p2_global_index ON tpcds.web_returns_p2
    (ca_street_number) GLOBAL;
    ```

- 修改索引分区的表空间
  - 修改索引分区*web_returns_p2_P2_index*的表空间为example1。

    ```
    gaussdb=# ALTER INDEX tpcds.tpcds_web_returns_p2_index2 MOVE PARTITION
    web_returns_p2_P2_index TABLESPACE example1;
    ```

    当结果显示为如下信息，则表示修改成功。

    ```
    ALTER INDEX
    ```

  - 修改索引分区*web_returns_p2_P3_index*的表空间为example2。

    ```
    gaussdb=# ALTER INDEX tpcds.tpcds_web_returns_p2_index2 MOVE PARTITION
    web_returns_p2_P3_index TABLESPACE example2;
    ```

    当结果显示为如下信息，则表示修改成功。

    ```
    ALTER INDEX
    ```

- 重命名索引分区

  执行如下命令对索引分区*web_returns_p2_P8_index*重命名*web_returns_p2_P8_index*_new。

  ```
  gaussdb=# ALTER INDEX tpcds.tpcds_web_returns_p2_index2 RENAME PARTITION web_returns_p2_P8_index TO web_returns_p2_P8_index_new;
  ```

  当结果显示为如下信息，则表示重命名成功。

  ```
  ALTER INDEX
  ```

- 查询索引
  - 执行如下命令查询系统和用户定义的所有索引。

    ```
    gaussdb=# SELECT RELNAME FROM PG_CLASS WHERE RELKIND='i' or RELKIND='I';
    ```

  - 执行如下命令查询指定索引的信息。

    ```
    gaussdb=# \di+ tpcds.tpcds_web_returns_p2_index2
    ```

- 删除索引

  ```
  gaussdb=# DROP INDEX tpcds.tpcds_web_returns_p2_index1;
  gaussdb=# DROP INDEX tpcds.tpcds_web_returns_p2_index2;
  ```

  当结果显示为如下信息，则表示删除成功。

  ```
  DROP INDEX
  ```

GaussDB支持4种创建索引的方式请参见**表3-4**。

说明

- 索引创建成功后，系统会自动判断何时引用索引。当系统认为使用索引比顺序扫描更快时，就会使用索引。
- 索引创建成功后，必须和表保持同步以保证能够准确地找到新数据，这样就增加了数据操作的负荷。因此请定期删除无用的索引。

表 3-4 索引方式

索引方式	描述
唯一索引	可用于约束索引属性值的唯一性，或者属性组合值的唯一性。如果一个表声明了唯一约束或者主键，则GaussDB自动在组成主键或唯一约束的字段上创建唯一索引（可能是多字段索引），以实现这些约束。目前，GaussDB只有B-Tree及UBTree可以创建唯一索引。
多字段索引	一个索引可以定义在表中的多个属性上。目前，GaussDB中的B-Tree支持多字段索引。
部分索引	建立在一个表的子集上的索引，这种索引方式只包含满足条件表达式的元组。
表达式索引	索引建立在一个函数或者从表中一个或多个属性计算出来的表达式上。表达式索引只有在查询时使用与创建时相同的表达式才会起作用。

- 创建一个普通表

  ```
  gaussdb=# CREATE TABLE tpcds.customer_address_bak AS TABLE tpcds.customer_address;
  INSERT 0 0
  ```

- 创建普通索引

  如果对于tpcds.customer_address_bak表，需要经常进行以下查询。

  ```
  gaussdb=# SELECT ca_address_sk FROM tpcds.customer_address_bak WHERE ca_address_sk=14888;
  ```


通常，数据库系统需要逐行扫描整个tpcds.customer_address_bak表以查询所有匹配的元组。如果表tpcds.customer_address_bak的规模很大，但满足WHERE条件的只有少数几个（可能是零个或一个），则这种顺序扫描的性能较差。如果让数据库系统在ca_address_sk属性上维护一个索引，用于快速定位匹配的元组，则数据库系统只需要在搜索树上查询少数的几层就可以找到匹配的元组，这将会大幅提高数据查询的性能。同样，在数据库中进行更新和删除操作时，索引也可以提升这些操作的性能。

使用以下命令创建索引。

```
gaussdb=# CREATE INDEX index_wr_returned_date_sk ON tpcds.customer_address_bak
(ca_address_sk);
CREATE INDEX
```

- 创建唯一索引

  在表tpcds.ship_mode_t1上的SM_SHIP_MODE_SK字段上创建唯一索引。

  ```
  gaussdb=# CREATE UNIQUE INDEX ds_ship_mode_t1_index1 ON
  tpcds.ship_mode_t1(SM_SHIP_MODE_SK);
  ```

- 创建多字段索引

  如果用户需要经常查询表tpcds.customer_address_bak中ca_address_sk是5050，且ca_street_number小于1000的记录，使用以下命令进行查询。

  ```
  gaussdb=# SELECT ca_address_sk,ca_address_id FROM tpcds.customer_address_bak WHERE
  ca_address_sk = 5050 AND ca_street_number < 1000;
  ```

  使用以下命令在字段ca_address_sk和ca_street_number上定义一个多字段索引。

  ```
  gaussdb=# CREATE INDEX more_column_index ON
  tpcds.customer_address_bak(ca_address_sk ,ca_street_number );
  CREATE INDEX
  ```

- 创建部分索引

  如果只需要查询ca_address_sk为5050的记录，可以创建部分索引来提升查询效率。

  ```
  gaussdb=# CREATE INDEX part_index ON tpcds.customer_address_bak(ca_address_sk) WHERE
  ca_address_sk = 5050;
  CREATE INDEX
  ```

- 创建表达式索引

  假如经常需要查询ca_street_number小于1000的信息，执行如下命令进行查询。

  ```
  gaussdb=# SELECT * FROM tpcds.customer_address_bak WHERE trunc(ca_street_number) < 1000;
  ```

  可以为上述查询创建表达式索引：

  ```
  gaussdb=# CREATE INDEX para_index ON tpcds.customer_address_bak (trunc(ca_street_number));
  CREATE INDEX
  ```

- 删除tpcds.customer_address_bak表。

  ```
  gaussdb=# DROP TABLE tpcds.customer_address_bak;
  DROP TABLE
  ```

3.2.6.4 创建和管理视图

背景信息

当用户对数据库中的一张或者多张表的某些字段的组合感兴趣，而又不想每次键入这些查询时，用户就可以定义一个视图，以便解决此问题。

视图与基本表不同，不是物理上实际存在的，是一个虚拟表。数据库中仅存放视图的定义，而不存放视图对应的数据，这些数据仍存放在原来的基本表中。若基本表中的数据发生变化，从视图中查询出的数据也随之改变。视图就像一个窗口，透过它可以看到数据库中用户感兴趣的数据及变化。视图每次被引用的时候都会运行一次。

管理视图

- 创建视图

  执行如下命令创建新视图MyView，其中tpcds.*web_returns*为已经创建的、包含*名为wr_refunded_cash*整型字段的用户表。

  ```
  gaussdb=# CREATE OR REPLACE VIEW MyView AS SELECT * FROM tpcds.web_returns WHERE trunc(wr_refunded_cash) > 10000;
  CREATE VIEW
  ```

  说明

  CREATE VIEW中的OR REPLACE可有可无，当存在OR REPLACE时，表示若以前存在该视图进行替换。

- 查询视图

  执行如下命令查询MyView视图。

  ```
  gaussdb=# SELECT * FROM MyView;
  ```

- 查看当前用户下的视图

  ```
  gaussdb=# SELECT * FROM my_views;
  ```

- 查看所有视图

  ```
  gaussdb=# SELECT * FROM adm_views;
  ```

- 查看某视图的具体信息

  执行如下命令查询MyView视图的详细信息。

  ```
  gaussdb=# \d+ MyView
                      View "PG_CATALOG.MyView"
   Column  |          Type          | Modifiers | Storage  | Description
  ----------+------------------------+-----------+----------+-------------
   USERNAME | CHARACTER VARYING(64) |           | extended |
  View definition:
   SELECT PG_AUTHID.ROLNAME::CHARACTER VARYING(64) AS USERNAME
     FROM PG_AUTHID;
  ```

- 删除视图

  执行如下命令删除MyView视图。

  ```
  gaussdb=# DROP VIEW MyView;
  DROP VIEW
  ```

3.2.6.5 创建和管理序列

背景信息

序列Sequence是用来产生唯一整数的数据库对象。序列的值是按照一定规则自增的整数。因为自增所以不重复，因此Sequence具有唯一标识性。这也是Sequence常被用作主键的原因。

通过序列使某字段成为唯一标识符的方法有两种：

- 一种是声明字段的类型为**序列整型**，由数据库在后台自动创建一个对应的Sequence。
- 另一种是使用**CREATE SEQUENCE**自定义一个新的Sequence，然后将nextval('sequence_name')函数读取的序列值，指定为某一字段的默认值，这样该字段就可以作为唯一标识符。

操作步骤

方法一：声明字段类型为序列整型来定义标识符字段。例如：

```
gaussdb=# CREATE TABLE T1
(
```



```
    id    serial,
    name  text
);
```

当结果显示为如下信息，则表示创建成功。

```
CREATE TABLE
```

方法二：创建序列，并通过nextval('sequence_name')函数指定为某一字段的默认值。

1. 创建序列

   ```
   gaussdb=# CREATE SEQUENCE seq1 cache 100;
   ```

   当结果显示为如下信息，则表示创建成功。

   ```
   CREATE SEQUENCE
   ```

2. 指定为某一字段的默认值，使该字段具有唯一标识属性。

   ```
   gaussdb=# CREATE TABLE T2
   (
      id   int not null default nextval('seq1'),
      name text
   );
   ```

   当结果显示为如下信息，则表示默认值指定成功。

   ```
   CREATE TABLE
   ```

3. 指定序列与列的归属关系。

   将序列和一个表的指定字段进行关联。这样，在删除该字段或其所在表的时候会自动删除已关联的序列。

   ```
   gaussdb=# ALTER SEQUENCE seq1 OWNED BY T2.id;
   ```

   当结果显示为如下信息，则表示指定成功。

   ```
   ALTER SEQUENCE
   ```

说明

除了为序列指定cache，方法二所实现的功能基本与方法一类似。但是一旦定义cache，序列将会产生空洞(序列值为不连贯的数值，如：1.4.5)，并且不能保序。另外为某序列指定从属列后，该列删除，对应的sequence也会被删除。虽然数据库并不限制序列只能为一列产生默认值，但最好不要多列共用同一个序列。

当前版本只支持在定义表的时候指定自增列，或者指定某列的默认值为nextval('seqname')，不支持在已有表中增加自增列或者增加默认值为nextval('seqname')的列。

3.2.6.6 创建和管理定时任务

背景信息

当客户在使用数据库过程中，如果白天执行一些耗时比较长的任务（例如：统计数据汇总之类或从其他数据库同步数据的任务），会对正常的业务有性能影响，所以客户经常选择在晚上执行，无形中增加了客户的工作量。因此GaussDB Kernel数据库兼容A数据库中定时任务的功能，可以由客户创建定时任务，当任务时间点到达后可以自动触发任务的执行，从而减少客户运维的工作量。

GaussDB Kernel数据库兼容A定时任务功能主要通过DBE_TASK高级包提供的接口，可以实现定时任务的创建、任务到期自动执行、任务删除和修改任务属性（包括：任务id、任务的关闭开启、任务的触发时间、触发时间间隔和任务内容等）等功能。

定时任务管理

步骤1 创建测试表。

```
gaussdb=# CREATE TABLE test(id int, time date);
```

当结果显示为如下信息，则表示创建成功。

```
CREATE TABLE
```

步骤2 创建自定义存储过程。

```
gaussdb=# CREATE OR REPLACE PROCEDURE PRC_JOB_1()
AS
N_NUM integer :=1;
BEGIN
FOR I IN 1..1000 LOOP
INSERT INTO test VALUES(I,SYSDATE);
END LOOP;
END;
/
```

当结果显示为如下信息，则表示创建成功。

```
CREATE PROCEDURE
```

步骤3 创建任务。

- 新创建的任务（未指定job_id）表示每隔1分钟执行一次存储过程PRC_JOB_1。

```
gaussdb=# call dbe_task.submit('call public.prc_job_1(); ', sysdate, 'interval ''1 minute'''', :a);
 id
-----
1
(1 row)
```

- 指定job_id创建任务，其中job_id可用范围为1~32767。

```
gaussdb=# call dbe_task.id_submit(1,'call public.prc_job_1(); ', sysdate, 'interval ''1 minute'''');
id_submit
---------


(1 row)
```

步骤4 通过视图查看当前用户已创建的任务信息。

```
gaussdb=# select job,dbname,start_date,last_date,this_date,next_date,broken,status,interval,failures,what
from my_jobs;
job | dbname |     start_date      |        last_date           |        this_date           |      next_date      | broken |
status |      interval       | failures |          what
-----+--------+---------------------+----------------------------+----------------------------+---------------------
+--------+--------+---------------------+----------+---------------------------
1 | testdb   | 2017-07-18 11:38:03 | 2017-07-18 13:53:03.607838 | 2017-07-18 13:53:03.607838 | 2017-07-18
13:54:03 | n      | s      | interval '1 minute' |        0 | call public.prc_job_1();
(1 row)
```

步骤5 停止任务。

```
gaussdb=# call dbe_task.finish(1,true);
finish
--------


(1 row)
```

步骤6 启动任务。

```
gaussdb=# call dbe_task.finish(1,false);
finish
--------


(1 row)
```

步骤7 修改任务属性。

- 修改JOB的next_time参数信息。

```
--修改Job1的next_time为1小时以后开始执行。
gaussdb=# call dbe_task.next_time(1, sysdate+1.0/24);
next_time
-----------


(1 row)
```

- 修改JOB的Interval参数信息。

```
--修改Job1的Interval为每隔1小时执行一次。
gaussdb=# call dbe_task.interval(1,'sysdate + 1.0/24');
 interval
----------


(1 row)
```

- 修改JOB的What参数信息。

```
--修改Job1的What为执行SQL语句“insert into public.test values(333, sysdate+5);”
gaussdb=# call dbe_task.content(1,'insert into public.test values(333, sysdate+5);');
 content
---------


(1 row)
```

- 同时修改JOB的Next_date、Interval、What等多个参数信息。

```
gaussdb=# call dbe_task.update(1, 'call public.prc_job_1();', sysdate, 'interval ''1 minute''');
 update
--------


(1 row)
```

步骤8 删除JOB。

```
gaussdb=# call dbe_task.cancel(1);
 cancel
---------


(1 row)
```

步骤9 查看JOB执行情况。

当JOB自动执行时，如果JOB执行失败（即job_status状态值为'f'）时，请联系管理员查看gs_log的运行日志来查看JOB的失败信息。

日志信息如下所示，从失败信息（detail error msg）中可以查看失败的具体错误。

```
LOG:  Execute Job Detail:
        job_id: 1
        what: call public.test();
        start_date: 2017-07-19 23:30:47.401818
        job_status: failed
        detail error msg: relation "test" does not exist
        end_date: 2017-07-19 23:30:47.401818
        next_run_date: 2017-07-19 23:30:56.855827
```

步骤10 JOB的权限控制。

- 当创建一个JOB时，该JOB会和创建该JOB的数据库和用户绑定（即：pg_job系统表新增的JOB记录中的dbname和log_user）。
- 如果当前用户是DBA用户、系统管理员或该JOB的创建用户（即：pg_job中的log_user），那么该用户有权限通过高级包接口remove、change、next_data、what、interval删除或修改JOB的参数信息。否则，会提示当前用户没有权限操作该JOB。
- 如果当前数据库是该JOB创建所属的数据库（即：pg_job系统表中的dbname），那么连接到当前数据库上可以通过高级包接口cancel、update、next_data、content、interval删除或修改JOB的参数信息。

- 当删除JOB所属的数据库（即：pg_job系统表中的dbname）时，系统会关联删除该数据库从属的JOB记录。
- 当删除JOB所属的用户（即：pg_job系统表中的log_user）时，系统会关联删除该用户从属的JOB记录。

步骤11 JOB的并发控制管理。

用户可以通过配置GUC参数**job_queue_processes**调整并发同时执行的JOB数目。

- 当**job_queue_processes**为0时，表示不启用定时任务功能，任何job都不会被执行。
- 当**job_queue_processes**大于0时，表示启用定时任务功能，该值为系统能够并发处理的最大任务数。

由于并行运行的任务数太多会消耗更多的系统资源，因此需要设置系统并发处理的任务数。当前并发的任务数达到**job_queue_processes**时，且此时又有任务到期，那么这些任务本次得不到执行将延期到下一轮询周期。因此，建议用户根据每个任务的执行时长合理的设置任务的时间间隔（即submit接口中的interval参数），来避免由于任务执行时间过长而导致下个轮询周期无法正常执行的情况。

注：对于不使用JOB的数据库实例，用户可以在数据库实例安装初始化完成后，通过设置**job_queue_processes**为0来关闭JOB功能，减少系统资源的消耗。

----结束

4 开发设计建议

4.1 开发设计建议概述

开发设计建议约定数据库建模和数据库应用程序开发过程中，应当遵守的设计规范。依据这些规范进行建模，能够更好地契合GaussDB的处理架构，输出更高效的业务SQL代码。

开发设计建议中所陈述的“建议”和“关注”含义如下：

- **建议**：用户应当遵守的设计规则。遵守这些规则，能够保证业务的高效运行；违反这些规则，将导致业务性能的大幅下降或某些业务逻辑错误。
- **关注**：在业务开发过程中用户需要注意的细则。用于标识容易导致用户理解错误的知识点（实际上遵守SQL标准的SQL行为），或者程序中潜在的用户不易感知的默认行为。

4.2 数据库对象命名

数据库对象命名需要满足约束：

- 表标识符长度不超过63个字节。
- 标识符以字母或下划线开头，中间字符可以是字母、数字、下划线、$、#。
- 若标识符被双引号（""）包含，则可以使用合法字符的任意组合，如"123gs_column"。
- 标识符不区分大小写，只有被双引号包含才区分大小写。
- 避免使用保留或者非保留关键字命名数据库对象。

  说明

  可以使用select * from pg_get_keywords()查询GaussDB的关键字，或者在**关键字**章节中查看。
- 避免使用双引号括起来的字符串来定义数据库对象名称，除非需要限制数据库对象名称的大小写。数据库对象名称大小写敏感会使定位问题难度增加。
- 数据库对象命名风格务必保持统一。
  - 增量开发的业务系统或进行业务迁移的系统，建议遵守历史的命名风格。

  - 建议使用多个单词组成，以下划线分割。
  - 数据库对象名称建议能够望文知意，尽量避免使用自定义缩写（可以使用通用的术语缩写进行命名）。例如，在命名中可以使用具有实际业务含义的英文词汇或汉语拼音，但规则应该在数据库实例范围内保持一致。
  - 变量名的关键是要具有描述性，即变量名要有一定的意义，变量名要有前缀标明该变量的类型。
- 表对象的命名应该可以表征该表的重要特征。例如，在表对象命名时区分该表是普通表、临时表还是非日志表：
  - 普通表名按照数据集的业务含义命名。
  - 临时表以“tmp_+后缀”命名。
  - 非日志表以“ul_+后缀”命名。
  - 外表以“f_+后缀”命名。
  - 不创建以redis_为前缀的数据库对象。
  - 不创建以mlog_和以matviewmap_为前缀的数据库对象。
  - 不创建以gs_role_为前缀的数据库对象。
- 表对象命名建议不要超过63字节。如果超过该长度内核会对表名进行截断，从而造成和设置值不一致的现象，且在不同字符集下，可能造成字符被截断，出现预期外的字符。

4.3 数据库对象设计

4.3.1 Database 和 Schema 设计

GaussDB中可以使用Database和Schema实现业务的隔离，区别在于Database的隔离更加彻底，各个Database之间共享资源极少，可实现连接隔离、权限隔离等，Database之间无法直接互访。Schema隔离的方式共用资源较多，可以通过grant与revoke语法便捷地控制不同用户对各Schema及其下属对象的权限。

- 从便捷性和资源共享效率上考虑，推荐使用Schema进行业务隔离。
- 建议系统管理员创建Schema和Database，再授予相关用户对应的权限。

Database 设计建议

- 在实际业务中，根据需要创建新的Database，不建议直接使用数据库实例默认的postgres数据库。
- 一个数据库实例内，用户自定义的Database数量推荐值为3个，不建议超过10个。用户自定义的Database数量过多会导致升级、备份等运维操作的效率降低。
- 为了适应全球化的需求，使数据库编码能够存储与表示绝大多数的字符，建议创建Database的时候使用UTF-8编码。
- 创建Database时，需要重点关注字符集编码（ENCODING）和兼容性（DBCOMPATIBILITY）两个配置项。GaussDB支持A、B、C和PG四种兼容模式，分别表示兼容O语法、MY语法、TD语法和POSTGRES语法，不同兼容模式下的语法行为存在一定差异，默认为A兼容模式。
- Database的owner默认拥有该Database下所有对象的所有权限，包括删除权限。删除权限影响较大，请谨慎使用。

Schema 设计建议

- 实际用户环境中Schema数量不建议超过100个。当数据库中存在大量Schema时，会导致gs_dump等依赖Schema数量的操作性能变慢。
- 如果该用户不具有sysadmin权限或者不是该Schema的owner，要访问Schema下的对象，需要同时给用户授予Schema的usage权限和对象的相应权限。
- 如果要在Schema下创建对象，需要授予操作用户该Schema的CREATE权限。
- Schema的owner默认拥有该Schema下对象的所有权限，包括删除权限。删除权限影响较大，请谨慎使用。

4.3.2 表设计

总体上讲，良好的表设计需要遵循以下原则：

- 减少需要扫描的数据量。通过分区表的剪枝机制可以大幅减少数据的扫描量。
- 尽量减少随机I/O。通过聚簇可以实现热数据的连续存储，将随机I/O转换为连续I/O，从而减少扫描的I/O代价。

选择分区方案

当表中的数据量很大时，应当对表进行分区，一般需要遵循以下原则：

- 使用具有明显区间性的字段进行分区，比如日期、区域等字段上建立分区。
- 分区名称应当体现分区的数据特征。例如，关键字+区间特征。
- 将分区上边界的分区值定义为MAXVALUE，以防止可能出现的数据溢出。

表 4-1 表的分区方式及使用场景

分区方式	描述
Range	表数据通过范围进行分区。
Interval	表数据通过范围进行分区，超出范围的会自动根据间隔创建新的分区。
List	表数据通过指定列按照具体值进行分区。
Hash	表数据通过Hash散列方式进行分区。

典型的分区表定义如下：

```
--创建Range分区表
CREATE TABLE staffS_p1
(
  staff_ID       NUMBER(6) not null,
  FIRST_NAME     VARCHAR2(20),
  LAST_NAME      VARCHAR2(25),
  EMAIL          VARCHAR2(25),
  PHONE_NUMBER   VARCHAR2(20),
  HIRE_DATE      DATE,
  employment_ID  VARCHAR2(10),
  SALARY         NUMBER(8,2),
  COMMISSION_PCT NUMBER(4,2),
  MANAGER_ID     NUMBER(6),
  section_ID     NUMBER(4)
)
```



```
PARTITION BY RANGE (HIRE_DATE)
(
   PARTITION HIRE_19950501 VALUES LESS THAN ('1995-05-01 00:00:00'),
   PARTITION HIRE_19950502 VALUES LESS THAN ('1995-05-02 00:00:00'),
   PARTITION HIRE_maxvalue VALUES LESS THAN (MAXVALUE)
);

--创建Interval分区表，初始两个分区，插入分区范围外的数据会自动新增分区
CREATE TABLE sales
(prod_id NUMBER(6),
 cust_id NUMBER,
 time_id DATE,
 channel_id CHAR(1),
 promo_id NUMBER(6),
 quantity_sold NUMBER(3),
 amount_sold NUMBER(10,2)
)
PARTITION BY RANGE (time_id)
INTERVAL('1 day')
( PARTITION p1 VALUES LESS THAN ('2019-02-01 00:00:00'),
  PARTITION p2 VALUES LESS THAN ('2019-02-02 00:00:00')
);

--创建List分区表
CREATE TABLE test_list (col1 int, col2 int)
partition by list(col1)
(
partition p1 values (2000),
partition p2 values (3000),
partition p3 values (4000),
partition p4 values (5000)
);

--创建Hash分区表
CREATE TABLE test_hash (col1 int, col2 int)
partition by hash(col1)
(
partition p1,
partition p2
);
```

更多的表分区语法信息请参见**CREATE TABLE PARTITION**。

4.3.3 字段设计

选择数据类型

在字段设计时，基于查询效率的考虑，一般需要遵循以下原则：

- 尽量使用高效数据类型。

  选择数值类型时，在满足业务精度的情况下，选择数据类型的优先级从高到低依次为整数、浮点数、NUMERIC。
- 当多个表存在逻辑关系时，表示同一含义的字段应该使用相同的数据类型。
- 对于字符串数据，建议使用变长字符串数据类型，并指定最大长度。请务必确保指定的最大长度大于需要存储的最大字符数，避免出现超出字段定义最大长度的异常报错而导致业务中断现象。除非明确知道数据类型为固定长度字符串，否则，不建议使用CHAR(n)、BPCHAR(n)、NCHAR(n)、CHARACTER(n)。

  关于字符串类型的详细说明，请参见**常用字符串类型介绍**。

常用字符串类型介绍

在进行字段设计时，需要根据数据特征选择相应的数据类型。字符串类型在使用时比较容易混淆，GaussDB中常见的字符串类型请参见**字符类型**。

4.3.4 约束设计

DEFAULT 和 NULL 约束

- 如果能够从业务层面补全字段值，那么，不建议使用DEFAULT约束，避免数据加载时产生不符合预期的结果。
- 给明确不存在NULL值的字段加上NOT NULL约束，优化器会在特定场景下对其进行自动优化。
- 给可以显式命名的约束显式命名。除了NOT NULL和DEFAULT约束外，其他约束都可以显式命名。

唯一约束

- 从命名上明确标识唯一约束，例如，命名为“UNI+构成字段”。

主键约束

- 从命名上明确标识主键约束，例如，将主键约束命名为“PK+字段名”。

检查约束

- 从命名上明确标识检查约束，例如，将检查约束命名为“CK+字段名”。

4.3.5 视图和关联表设计

视图设计

- 除非视图之间存在强依赖关系，否则不建议视图嵌套。
- 视图定义中尽量避免排序操作。

关联表设计

- 表之间的关联字段应该尽量少。
- 关联字段的数据类型应该保持一致。
- 关联字段在命名上，应该可以明显体现出关联关系。例如，采用同样名称来命名。

4.4 工具对接

4.4.1 JDBC 配置

目前，GaussDB相关的第三方工具都是通过JDBC进行连接的，此部分将介绍工具配置时的注意事项。

连接参数

- 第三方工具通过JDBC连接GaussDB时，JDBC向GaussDB发起连接请求，会默认添加以下配置参数，详见JDBC代码ConnectionFactoryImpl类的实现。

```
params = {
          { "user", user },
          { "database", database },
          { "client_encoding", "UTF8" },
          { "DateStyle", "ISO" },
          { "extra_float_digits", "3" },
          { "TimeZone",  createPostgresTimeZone() },
        };
```

  这些参数可能会导致JDBC客户端的行为与gsql客户端的行为不一致，例如，Date数据显示方式、浮点数精度表示、timezone显示。

  如果实际期望和这些配置不符，建议在java连接设置代码中显式设定这些参数。

  通过JDBC连接数据库时，会设置extra_float_digits=3，gsql中设置为extra_float_digits=0，可能会造成同一条数据在JDBC显示和gsql显示的精度不同。

- 对于精度敏感的场景，建议使用numeric类型。
- 通过JDBC连接数据库时，应该保证以下三个时区设置一致：
  - JDBC客户端所在主机的时区。
  - GaussDB数据库实例所在主机的时区。
  - GaussDB数据库实例配置过程中时区。

    说明

    时区设置相关的操作，请联系管理员。

fetchsize

在应用程序中，如果需要使用fetchsize，必须关闭autocommit。开启autocommit，会令fetchsize配置失效。

autocommit

在JDBC向GaussDB申请连接的代码中，建议显式开启autocommit。如果基于性能或者其它方面考虑，需要关闭autocommit时，需要应用程序保证事务的提交。例如，在指定的业务SQL执行完之后做显式提交，特别是客户端退出之前务必保证所有的事务已经提交。

释放连接

- 推荐使用连接池限制应用程序的连接数。每执行一条SQL就连接一次数据库，是一种不好的SQL编写习惯。
- 在应用程序完成作业任务之后，应当及时断开和GaussDB的连接，释放资源。建议在任务中设置session超时时间参数。
- 使用JDBC连接池，在将连接释放给连接池前，需要执行以下操作，重置会话环境。否则，可能会因为历史会话信息导致的对象冲突。
  - 如果在连接中设置了GUC参数，那么在将连接归还连接池之前，必须使用“SET SESSION AUTHORIZATION DEFAULT;RESET ALL;”将连接的状态清空。
  - 如果使用了临时表，那么在将连接归还连接池之前，必须将临时表删除。

CopyManager

在不使用ETL工具，数据入库实时性要求比较高的情况下，建议在开发应用程序时，使用GaussDB JDBC驱动的copyManger接口进行微批导入。

4.5 SQL 编写

DDL

- 在GaussDB中，建议DDL（建表、COMMENT等）操作统一执行。在批处理作业中尽量避免DDL操作，避免大量并发事务对性能的影响。
- 在非日志表（unlogged table）使用完后，立即执行数据清理（TRUNCATE）操作。因为在异常场景下，GaussDB不保证非日志表(unlogged table)数据的安全性。
- 临时表和非日志表的存储方式建议和基表相同。
- 索引字段的总长度不超过50字节。否则，索引大小会膨胀比较严重，带来较大的存储开销，同时索引性能也会下降。
- 不要使用DROP…CASCADE方式删除对象，除非已经明确对象间的依赖关系，以免误删。

数据加载和卸载

- 在INSERT语句中显式设置插入的字段列表。例如：
```
INSERT INTO task(name,id,comment) VALUES ('task1','100','第100个任务');
```
- 在批量数据入库之后，或者数据增量达到一定阈值后，建议对表进行ANALYZE操作，防止统计信息不准确而导致的执行计划劣化。
- 如果要清理表中的所有数据，建议使用TRUNCATE TABLE方式，不要使用DELETE TABLE方式。DELETE TABLE方式删除性能差，且不会释放那些已经删除了的数据占用的磁盘空间。

类型转换

- 在需要数据类型转换（不同数据类型进行比较或转换）时，使用强制类型转换，以防隐式类型转换结果与预期不符。
- 在查询中，对常量要显式指定数据类型，不要试图依赖任何隐式的数据类型转换。
- 若sql_compatibility参数设置为A，在导入数据时，空字符串会自动转化为NULL。如果需要保留空字符串，则需将sql_compatibility参数设置为C。

查询操作

- 除ETL程序外，应该尽量避免向客户端返回大量结果集的操作。如果结果集过大，应考虑业务设计是否合理。
- 使用事务方式执行DDL和DML操作。例如，TRUNCATE TABLE、UPDATE TABLE、DELETE TABLE、DROP TABLE等操作，一旦执行提交则无法恢复。对于这类操作，建议使用事务进行封装，必要时可以进行回滚。
- 在查询编写时，建议明确列出查询涉及的所有字段，不建议使用“SELECT *”语法。一方面基于性能考虑，尽量减少查询输出列，另一方面避免增删字段对前端业务兼容性的影响。

- 在访问表对象时带上Schema前缀，可以避免因Schema切换导致访问到非预期的表。
- 超过3张表或视图进行关联（特别是FULL JOIN）时，执行代价难以估算。建议使用WITH TABLE AS语句创建中间临时表的方式增加SQL语句的可读性。
- 尽量避免使用笛卡尔积和FULL JOIN。这些操作会造成结果集的急剧膨胀，同时其执行性能也会降低。
- NULL值的比较只能使用IS NULL或者IS NOT NULL的方式判断，其他任何形式的逻辑判断都返回NULL。例如：NULL<>NULL、NULL=NULL和NULL<>1返回结果都是NULL，而不是期望的布尔值。
- 需要统计表中所有记录数时，不要使用count(col)来替代count(*)。count(*)会统计NULL值（真实行数），而count(col)不会统计。
- 在执行count(col)时，将“值为NULL”的记录行计数为0。在执行sum(col)时，当所有记录都为NULL时，最终将返回NULL；当不全为NULL时，“值为NULL”的记录行将被计数为0。
- count(多个字段)时，多个字段名必须用圆括号括起来。例如，count((col1,col2,col3))。注意：通过多字段统计行数时，即使所选字段都为NULL，该行也被计数，效果与count(*)一致。
- count(distinct col)用来计算该列不重复的非NULL的数量，NULL将不被计数。
- count(distinct (col1,col2,...))用来统计多列的唯一值数量，当所有统计字段都为NULL时，也会被计数，同时这些记录被认为是相同的。
- 使用连接操作符“||”替换concat函数进行字符串连接。因为concat函数本身需要额外查询类型表和函数表，基础性能较慢；另外concat的输出跟data type有关，生成的执行计划时不能提前计算结果值，导致查询性能严重劣化。
- 使用下面**表1 时间相关的宏**替换now函数来获取当前时间。因为now函数生成的执行计划无法下推，导致查询性能严重劣化。

表 4-2 时间相关的宏

宏名称	描述	示例
CURRENT_DATE	获取当前日期，不包含时分秒。	gaussdb=# SELECT CURRENT_DATE; date ------------ 2018-02-02 (1 row)
CURRENT_TIME	获取当前时间，不包含年月日。	gaussdb=# SELECT CURRENT_TIME; timetz -------------------- 00:39:34.633938+08 (1 row)
CURRENT_TIMESTAMP(n)	获取当前日期和时间，包含年月日时分秒。 **说明** n表示存储的毫秒位数。	gaussdb=# SELECT CURRENT_TIMESTAMP(6); timestamptz ------------------------------- 2018-02-02 00:39:55.231689+08 (1 row)

- 尽量避免标量子查询语句的出现。标量子查询是出现在SELECT语句输出列表中的子查询，在下面例子中，“SELECT COUNT(*) FROM films f WHERE f.did = s.id”部分即为一个标量子查询语句。


```
SELECT id, (SELECT COUNT(*) FROM films f WHERE f.did = s.id) FROM staffs_p1 s;
```

标量子查询往往会导致查询性能严重劣化，在应用开发过程中，应当根据业务逻辑，对标量子查询进行等价转换，将其写为表关联。

- 在WHERE子句中，应当对过滤条件进行排序，把选择读较小（筛选出的记录数较少）的条件排在前面。
- WHERE子句中的过滤条件，尽量符合单边规则。即把字段名放在比较条件的一边，优化器在某些场景下会自动进行剪枝优化。形如col op expression，其中col为表的一个列，op为‘=’、‘>’的等比较操作符，expression为不含列名的表达式。例如，

```
SELECT id, from_image_id, from_person_id, from_video_id FROM face_data WHERE
current_timestamp(6) - time < '1 days'::interval;
```

改写为：

```
SELECT id, from_image_id, from_person_id, from_video_id FROM face_data where time >
current_timestamp(6) - '1 days'::interval;
```

- 尽量避免不必要的排序操作。排序需要耗费大量的内存及CPU，如果业务逻辑许可，可以组合使用ORDER BY和LIMIT，减小资源开销。GaussDB默认按照ASC & NULL LAST进行排序。
- 使用ORDER BY子句进行排序时，显式指定排序方式（ASC/DESC），NULL的排序方式（NULL FIRST/NULL LAST）。
- 不要单独依赖LIMIT子句返回特定顺序的结果集。如果返回部分特定结果集，可以将ORDER BY子句与LIMIT子句组合使用，必要时也可以使用OFFSET跳过特定结果。
- 在保障业务逻辑准确的情况下，建议尽量使用UNION ALL来代替UNION。
- 如果过滤条件只有OR表达式，可以将OR表达式转化为UNION ALL以提升性能。使用OR的SQL语句经常无法优化，导致执行速度变慢。例如，下面语句的转换。

```
SELECT * FROM scdc.pub_menu
WHERE (cdp= 300 AND inline=301) OR (cdp= 301 AND inline=302) OR (cdp= 302 ANDinline=301);
```

转换为：

```
SELECT * FROM scdc.pub_menu
WHERE (cdp= 300 AND inline=301)
union all
SELECT * FROM scdc.pub_menu
WHERE (cdp= 301 AND inline=302)
union all
SELECT * FROM tablename
WHERE (cdp= 302 AND inline=301)
```

- 当IN(val1, val2, val3…)表达式中字段较多时，建议使用IN (VALUES (val1), (val2),(val3)…)语句进行替换。优化器会自动把IN约束转换为非关联子查询，从而提升查询性能。
- 在关联字段不存在NULL值的情况下，使用(NOT) EXIST代替(NOT) IN。例如，在下面查询语句中，当T1.C1列不存在NULL值时，可以先为T1.C1字段添加NOT NULL约束，再进行改写。

```
SELECT * FROM T1 WHERE T1.C1 NOT IN (SELECT T2.C2 FROM T2);
```

可以改写为：

```
SELECT * FROM T1 WHERE NOT EXISTS (SELECT  * FROM T2 WHERE T1.C1=T2.C2);
```

说明

- 如果不能保证T1.C1列的值为NOT NULL的情况下，不能进行上述改写。
- 如果T1.C1为子查询的输出，要根据业务逻辑确认其输出是否为NOT NULL。

- 通过游标进行翻页查询，而不使用LIMIT OFFSET语法，避免多次执行带来的资源开销。游标必须在事务中使用，执行完后务必关闭游标并提交事务。

5 应用程序开发教程

5.1 GaussDB 应用程序开发教程

GaussDB支持通过JDBC、ODBC、libpq、Psycopg、Go、ecpg等接口来连接数据库并进行操作。

JDBC

JDBC（Java Database Connectivity，Java数据库连接）是用于执行SQL语句的Java API，可以为多种关系数据库提供统一访问接口。它允许Java应用程序通过SQL语句来执行数据库操作，包括查询、插入、更新和删除数据等。

以下是JDBC的一些关键特点和用法：

1. 数据库连接管理：JDBC允许应用程序建立与数据库的连接，并管理这些连接的生命周期。
2. SQL执行： 使用JDBC，可以执行SQL查询、更新、删除等操作。这通过在Java代码中构建SQL语句并将其发送到数据库来实现。
3. 事务管理：JDBC支持事务管理，可以通过JDBC API来启动、提交或回滚事务，确保数据库操作的一致性和完整性。
4. 异常处理： JDBC定义了一组异常类来处理数据库操作期间可能发生的各种异常情况，开发人员可以使用try-catch块来捕获和处理这些异常。
5. 批处理操作：JDBC提供了批处理功能，允许一次性执行多个SQL语句，从而提高数据库操作的效率。
6. 元数据访问： 通过JDBC，可以获取数据库的元数据信息，如表结构、列名、数据类型等，从而动态地构建SQL查询或根据数据库结构进行操作。

总的来说，JDBC提供了一个灵活且强大的桥梁，使Java应用程序能够与各种不同的数据库进行交互，从而实现数据的持久化和管理。GaussDB库提供了对JDBC 4.2特性的支持，需要使用JDK1.8版本编译程序代码，不支持JDBC桥接ODBC方式。

基于JDBC开发详细教程请参见**基于JDBC开发**。

ODBC

ODBC（Open Database Connectivity，开放数据库互连）是由Microsoft公司基于X/OPEN CLI提出的以C/C++语言来访问数据库的应用程序编程接口。它提供了一种统一

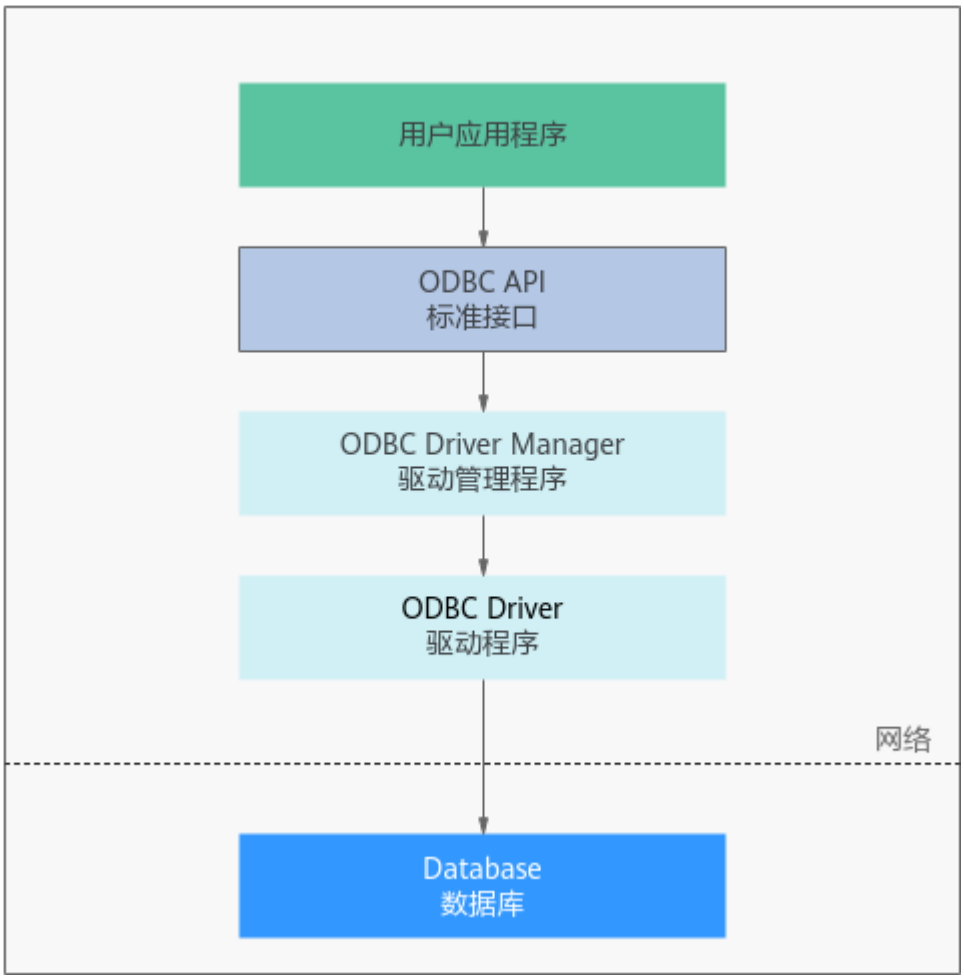
的方法，让应用程序可以访问各种数据库管理系统（DBMS），而不用考虑具体的数据库类型或者操作系统平台。ODBC允许应用程序使用SQL来查询、插入、更新和删除数据库中的数据。应用程序通过ODBC提供的API与数据库进行交互，增强了应用程序的可移植性、扩展性和可维护性。

ODBC的架构包括三个主要组件：应用程序、ODBC驱动管理程序和ODBC驱动程序。应用程序使用ODBC API与ODBC驱动管理程序通信，而ODBC驱动管理程序负责加载和管理ODBC驱动程序。ODBC驱动程序则负责与特定的数据库通信，执行SQL查询并返回结果。ODBC的系统架构请参见图5-1。

总而言之，ODBC提供了一种灵活且跨平台的方法，使得用户可以轻松地将应用程序连接到各种数据库，而无需担心特定数据库系统的细节。

基于ODBC开发详细教程请参见基于ODBC开发。

图 5-1 ODBC 系统架构



GaussDB目前在以下环境中提供对ODBC的支持。

表 5-1 ODBC 支持平台

操作系统	平台
EulerOS 2.5	x86_64位

操作系统	平台
EulerOS 2.9	ARM64位
EulerOS 2.10	x86_64位
EulerOS 2.10	ARM64位
Windows 7	x86_32位
Windows 7	x86_64位
Windows Server 2008	x86_32位
Windows Server 2008	x86_64位
Kylin V10	x86_64位
Kylin V10	ARM64位
UnionTech V20	x86_64位
UnionTech V20	ARM64位

Unix/Linux系统下的驱动程序管理器主要有unixODBC和iODBC，在这选择驱动管理器unixODBC-2.3.7作为连接数据库的组件。

Windows系统自带ODBC驱动程序管理器，在控制面板->管理工具中可以找到数据源（ODBC）选项。

说明

当前数据库ODBC驱动基于开源版本，对于tinyint、smalldatetime、nvarchar、nvarchar2类型，在获取数据类型的时候，可能会出现不兼容的情况。

ODBC相关约束说明：

- ODBC不支持自定义类型，不支持在存储过程中使用自定义类型参数。
- 当数据库开启proc_outparam_override参数时，ODBC无法正常调用带有out参数的存储过程。
- 不支持容灾场景的备机读。

libpq

libpq是GaussDB C应用程序接口。libpq是一套允许客户程序向GaussDB服务器进程发送查询并且获得查询返回值的库函数。同时也是其他几个GaussDB应用接口下面的引擎，如ODBC等依赖的库文件。

基于libpq开发详细教程请参见**基于libpq开发**。

Psycopg

Psycopg是一种用于执行SQL语句的PythonAPI，可以为GaussDB数据库提供统一访问接口，应用程序可基于它进行数据操作。Psycopg2是对libpq的封装，主要使用C语言实现，既高效又安全。它具有客户端游标和服务器端游标、异步通信和通知、支持

“COPY TO/COPY FROM”功能。支持多种类型Python开箱即用，适配GaussDB数据类型。通过灵活的对象适配系统，可以扩展和定制适配。Psycopg2兼容Unicode。

GaussDB数据库提供了对Psycopg2特性的支持，并且支持psycopg2通过SSL模式连接。

基于Psycopg开发详细教程请参见**基于Psycopg开发**。

表 5-2 Psycopg 支持平台

操作系统	平台	Python版本
EulerOS 2.5	• ARM64位 • x86_64位	3.8.5
EulerOS 2.9	• ARM64位 • x86_64位	3.7.4
EulerOS 2.10、Kylin v10、UnionTech20	• ARM64位 • x86_64位	3.7.9
EulerOS 2.11、Suse 12.5	• ARM64位 • x86_64位	3.9.11

须知

psycopg2在编译过程中，会链接（link）GaussDB的openssl，GaussDB的openssl与操作系统自带的openssl可能不兼容。如果遇到不兼容现象，例如提示"version 'OPENSSL_1_1_1f' not found"，请使用环境变量LD_LIBRARY_PATH进行隔离，以避免混用操作系统自带的openssl与GaussDB依赖的openssl。

例如，在执行某个调用psycopg2的应用软件client.py时，将环境变量显性赋予该应用软件：

```
export LD_LIBRARY_PATH=/path/to/gaussdb/libs:$LD_LIBRARY_PATH python client.py
```

其中，/path/to/pyscopg2/lib 表示GaussDB依赖的openssl库所在目录，需根据文件实际存储路径修改。

Go

Go驱动是一个用于连接和操作GaussDB数据库的Go语言驱动。它提供了一些用于连接和操作GaussDB数据库的接口，可以用于执行查询、插入、更新和删除等操作。

以下是Go驱动当前支持的部分功能及其用法：

1. 数据库连接：Go驱动支持通过db.Open连接数据库。连接串支持URL和DSN两种格式。
2. SQL执行与查询：Go驱动提供接口如db.Exec、db.Query等，可以用于SQL的执行与查询 。
3. 事务管理：Go驱动支持Tx接口进行事务管理，其中实现了启动、提交、回滚事务等方法，确保数据库操作的一致性和完整性。

4. 元数据访问： Go驱动提供ColumnType接口用于查询数据库中列属性信息，包括是否可以为空、数据库类型名称、长度和小数位数等信息。
5. 可重用性：Go驱动提供Prepare方法用于预编译SQL，该SQL可以被多次执行，只需改变传入参数，提高数据库可重用性。

基于Go驱动开发详细教程请参见**基于Go驱动开发**。

ecpg

ecpg（embedded SQL C preprocessor for GaussDB Kernel）是一种用于C语言程序的嵌入式SQL预处理器。一个嵌入式SQL程序由一种普通编程语言编写的代码（此处为C语言）和SQL命令共同组成。要构建该程序，源代码（*.pgc）首先通过嵌入式SQL预处理器，将源代码转换成一个普通C语言程序（*.c），然后再通过编译器处理。转换过的ecpg应用通过嵌入式SQL库（ecpglib）调用libpq库中的函数，与GaussDB Kernel服务器使用普通的前端/后端协议通信。

嵌入式SQL程序是插入了数据库相关动作的特殊代码的C语言程序。这种特殊代码形式通常如下：

```
EXEC SQL ...;
```

这些语句在语法上取代了一个C语句，可以出现在全局或者是一个函数中。嵌入式SQL语句遵循普通SQL代码的大小写敏感规则，也允许嵌套的C语言代码风格注释（SQL标准的一部分）。不过，程序的C语言部分遵循C语言程序的标准，不支持嵌套注释。

基于ecpg开发详细教程请参见**基于ecpg开发**。

5.2 开发规范

如果用户在APP的开发过程中，使用了连接池机制，那么需要遵循如下规范：

- 如果在连接中设置了GUC参数，那么在将连接归还连接池之前，必须使用“SET SESSION AUTHORIZATION DEFAULT;RESET ALL;”将连接的状态清空。
- 如果使用了临时表，那么在将连接归还连接池之前，必须将临时表删除。

否则，连接池里面的连接就是有状态的，会对用户后续使用连接池进行操作的正确性带来影响。

应用程序开发驱动兼容性说明如**表5-3**所示：

表 5-3 兼容性说明

驱动	兼容性说明
JDBC、Go	驱动前向兼容数据库，若需使用驱动与数据库同步增加的新特性，须升级数据库。
ODBC、libpq、Psycopg、ecpg	驱动须与数据库版本配套。

须知

- behavior_compat_options='proc_outparam_override' 重载参数仅在A兼容模式可用。
- 原则上，兼容性参数应在创建数据库后就设置，不应在使用过程中来回切换。
- 涉及使用以下场景的特性需要配合将JDBC驱动升级到503.1及以上的配套版本：
  - 开启s2兼容性参数，设置sessiontimezone的合法性校验。

在多线程环境下使用驱动：

JDBC驱动程序是非线程安全的，无法保证连接上的方法同步。由调用者来同步对驱动程序的调用。

5.3 驱动包获取

获取驱动包

下载版本的发布包，如表5-4所示。

表 5-4 驱动包下载列表

版本	下载地址
V2.0-3.x	驱动包 驱动包校验包

为了防止软件包在传递过程或存储期间被恶意篡改，下载软件包时需下载对应的校验包对软件包进行校验，校验方法如下：

1. 上传软件包和软件包校验包到虚拟机（Linux操作系统）的同一目录下。
2. 执行如下命令，校验软件包完整性。

   cat *GaussDB_driver.zip.sha256* **| sha256sum --check**

   如果回显OK，则校验通过。

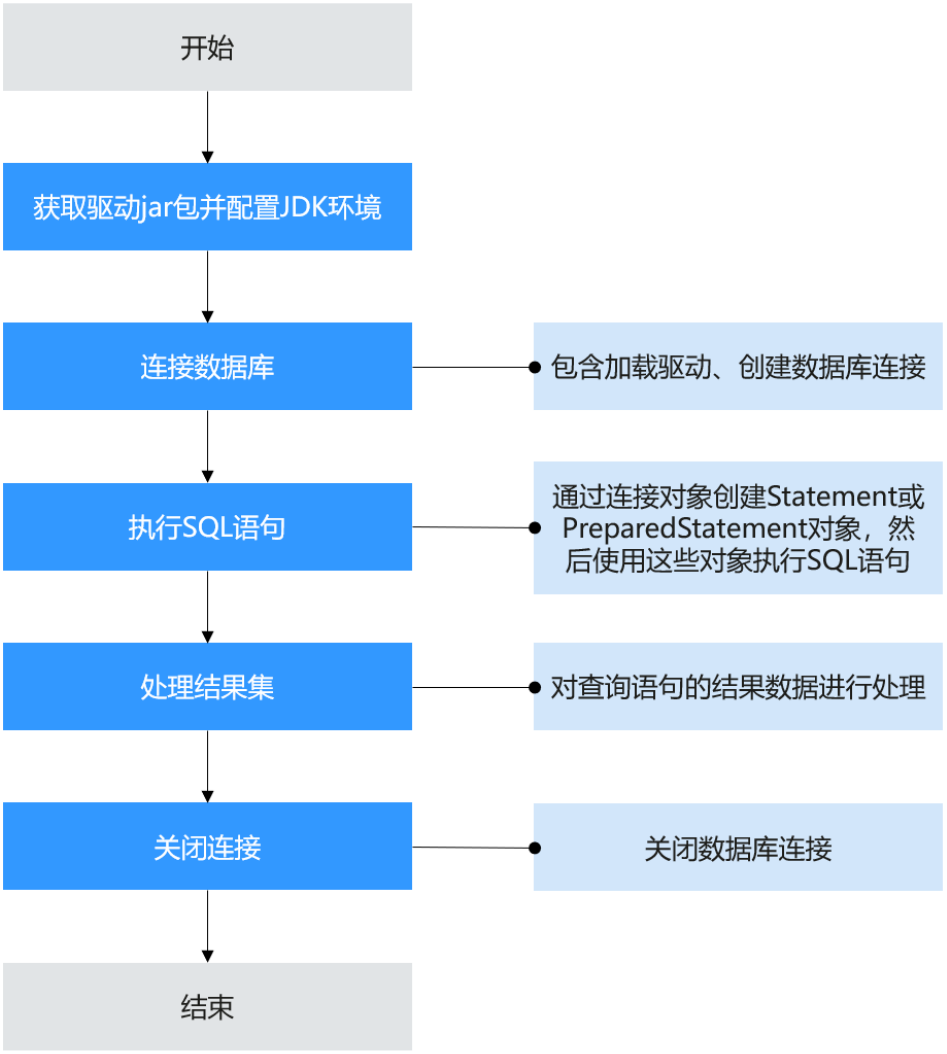
   ```
   GaussDB_driver.zip: OK
   ```

5.4 基于 JDBC 开发

5.4.1 开发流程

采用JDBC开发应用程序的流程，可分为获取驱动jar包并配置JDK环境、连接数据库、执行SQL语句、处理结果集、关闭连接等几个部分，请参见图5-2。

图 5-2 采用 JDBC 开发应用程序的流程



5.4.2 开发步骤

5.4.2.1 获取驱动 jar 包并配置 JDK 环境

环境准备包括获取驱动jar包、配置JDK环境。

JDBC 包

从发布包中获取驱动jar包，包名为GaussDB-Kernel_数据库版本号_操作系统_64bit_Jdbc.tar.gz。解压后JDBC的驱动jar包：

- opengaussjdbc.jar：主类名为“com.huawei.opengauss.jdbc.Driver”，数据库连接的url前缀为“jdbc:opengauss”，推荐使用此驱动包。如果遇到同一JVM进程内需要同时访问PG及GaussDB的场景，请使用此驱动包。
- gsjdbc4.jar：主类名为“org.postgresql.Driver”，数据库连接的url前缀为“jdbc:postgresql”，该驱动包适用于从PG数据库迁移业务的场景，驱动类和加载路径与迁移前保持一致，但接口支持情况不完全一致，未支持的接口需要业务侧进行调整。

- gscejdbc.jar：主类名为“com.huawei.gaussdb.jdbc.Driver”，数据库连接的url前缀为“jdbc:gaussdb”，此驱动包打包了密态数据库需要加载的加解密相关的依赖库，密态场景推荐使用此驱动包，目前仅支持EulerOS操作系统。使用gscejdbc.jar驱动包时，需要先设置环境变量LD_LIBRARY_PATH。具体使用方式参见《特性指南》中“设置密态等值查询 > 使用JDBC操作密态数据库”章节。
- gsjdbc200.jar：主类名为“com.huawei.gauss200.jdbc.Driver”，数据库连接的url前缀为“jdbc:gaussdb”，该驱动包适用于从Gauss200迁移业务的场景，驱动类和加载路径与迁移前保持一致，但接口支持情况不完全一致，未支持的接口需要业务侧进行调整。

须知

- 各驱动包只是驱动类加载路径和url前缀不同，接口功能上相同。
- jdbc发布件jar包按照架构分类，gscejdbc.jar包必须与对应的部署环境一致才能使用，其他jar包无需与部署环境一致。
- 不能使用gsjdbc4的驱动包操作PG数据库，虽然部分版本能够建连成功，但部分接口行为与PG JDBC不同，可能导致未知错误。
- 不能使用PG的驱动包操作GaussDB数据库，虽然部分版本能够建连成功，但部分接口行为与GaussDB JDBC不同，可能导致未知错误。

配置 JDK 环境

客户端需配置JDK1.8。JDK是跨平台的，支持Windows、Linux等多种平台。以Windows为例，配置方法如下。

步骤1 在DOS窗口（Windows下的命令提示符）输入以下命令查看JDK版本。

```
java -version
```

确认是否已安装JDK1.8。如果未安装JDK，请从官方网站下载安装包并安装。

步骤2 根据如下步骤配置系统环境变量。

1. 右键单击“我的电脑”，选择“属性”。
2. 在“系统”页面左侧导航栏单击“高级系统设置”。
3. 在“系统属性”页面，“高级”页签上单击“环境变量”。
4. 在“环境变量”页面上，“系统变量”区域单击“新建”或“编辑”配置系统变量。变量说明如表5-5所示。

表 5-5 变量说明

变量名	操作	变量值
JAVA_HOME	– 若存在，则单击“编辑”。 – 若不存在，则单击“新建”。	JAVA的安装目录。 例如：C:\Program Files\Java\jdk1.8.0_131

变量名	操作	变量值
Path	单击“编辑”。	– 若配置了JAVA_HOME，则在变量值的最前面加上：%JAVA_HOME%\bin – 若未配置JAVA_HOME，则在变量值的最前面加上JAVA安装的全路径：C:\Program Files\Java\jdk1.8.0_131\bin
CLASSPATH	单击“新建”。	%JAVA_HOME%\lib;%JAVA_HOME%\lib\tools.jar

----结束

5.4.2.2 连接数据库

在获取驱动jar包并配置JDK环境的情况下，GaussDB支持用户通过3种方式连接数据库，本章节主要介绍如何使用这3种方式连接数据库。

5.4.2.2.1 连接方式介绍

连接数据库的方式通常是根据安全性、性能、可扩展性和实现复杂度等因素进行选择的。连接方式介绍如表5-6所示。

表 5-6 连接方式介绍

连接方式	优点	缺点	选择原则
非加密方式	• 简单易实现，无需额外的加密和解密操作。 • 性能较高，因为不需要进行加密/解密操作，减少了数据传输的开销。	• 安全性较低，数据在传输过程中容易被窃听或篡改。 • 不适合传输敏感信息，可能违反隐私保护法规。	只适用于内部网络或对安全性要求不高的环境。
SSL方式	• 提供了加密传输，数据在传输过程中得到保护，安全性较高。 • 支持客户端/服务器身份验证，增强了安全性。	• 配置和管理相对复杂，需要证书的颁发和更新。 • 可能会增加一些性能开销，特别是在加密和解密大量数据时。	适用于对数据传输安全性要求较高的环境，如金融、医疗等行业。

连接方式	优点	缺点	选择原则
UDS方式	• 适用于本地通信，不经过网络，传输速度快。 • 不需要额外的网络开销，不容易受到网络攻击。	• 仅限于在同一台主机上的进程间通信，无法用于远程连接。	适用于本地通信的场景，如本地服务和应用之间的数据交换。

5.4.2.2.2 连接参数参考

info参数连接的所有属性名称对大小写敏感。常用的属性如表5-7所示。

表 5-7 info 参数的连接属性

属性名称	属性说明	属性值
PGDBNAME	表示数据库名称（url中无需配置该参数，自动从Properties文件中解析）。	属性类型：String
PGHOST	主机IP地址（url中无需配置该参数，自动从Properties文件中解析），同时支持IPv4和IPv6。	属性类型：String
PGPORT	主机端口号（url中无需配置该参数，自动从Properties文件中解析）。	属性类型：Integer
user	表示创建连接的数据库用户。	属性类型：String
password	表示数据库用户的密码。	属性类型：String
driverInfoMode	控制驱动描述信息的输出模式。	属性类型：String 取值范围：postgresql、gaussdb。 • postgresql：表示输出postgresql相关的驱动描述信息。 • gaussdb：表示输出gaussdb相关的驱动描述信息。 默认值：postgresql

属性名称	属性说明	属性值
enable_ce	表示密态等值查询的能力。	属性类型：Integer 取值范围： ● 1：表示JDBC支持密态等值查询基本能力。 ● 其他值：表示不支持密态等值查询基本能力。 默认值：该属性不设置则为空，表示不支持密态等值查询。
refreshClientEncryption	指定密态数据库是否支持客户端缓存刷新。	属性类型：String 取值范围： ● 1或NULL：表示密态数据库支持客户端缓存刷新。 ● 其他字符串：表示密态数据库不支持客户端缓存刷新。 默认值：NULL
loggerLevel	指定日志记录级别。	属性类型：String 取值范围：目前支持4种级别（不区分大小写）：OFF、INFO、DEBUG、TRACE。 ● OFF：表示关闭日志。 ● INFO、DEBUG和TRACE：表示记录的日志信息详细程度不同。 默认值：该属性不设置则为空，与设置为INFO等效。
loggerFile	用于指定日志输出路径（目录和文件名）。一般要求明确指定日志目录和文件名。 若只指定文件名，未指定目录则日志生成在客户端运行程序目录；若不配置或配置的路径不存在，则日志会默认通过流输出。此参数已废弃，不再生效，如需使用可通过java.util.logging 属性文件或系统属性进行配置。	属性类型：String
logger	表示JDBC Driver要使用的日志输出框架。JDBC Driver支持对接用户应用程序使用的日志输出框架。目前支持的第三方日志输出框架只有基于Slf4j-API的日志框架。具体使用方式，请参见**日志管理**。	属性类型：String 取值范围：Slf4JLogger、JdkLogger ● 如果缺省，则JDBC Driver使用JdkLogger。 ● 采用基于Slf4j-API第三方日志框架。

属性名称	属性说明	属性值
allowEncodingChanges	设置该参数值为true进行字符集类型更改，配合参数characterEncoding设置字符集，二者使用“&”分隔。 例： allowEncodingChanges=true&characterEncoding=UTF8。	属性类型：Boolean 取值范围： ● true：表示更改字符集类型。 ● false：表示不更改字符集类型。 默认值：false
characterEncoding	需要配合allowEncodingChanges使用，allowEncodingChanges为false的情况下，characterEncoding仅允许设置为UTF8；allowEncodingChanges为true的情况下，characterEncoding可在取值范围内进行设置。	属性类型：String 取值范围：UTF8、GBK、LATIN1、GB18030 **说明** 当连接到字符集为GB18030_2022的数据库时，characterEncoding设置为GB18030_2022不生效，会默认使用UTF8，需要设置为GB18030才能正常解析服务端的GB18030_2022字符。 默认值：UTF8
currentSchema	设置当前连接的schema，在search-path（搜索路径）中指定要设置的schema。如果schema名包含除字母、数字、下划线之外的特殊字符，需要在schema名上加引号，注意加引号后schema名大小写敏感。如需配置多个schema，要用逗号“,”进行分隔。 例如： currentSchema=schema_a,"schema-b","schema/c"	属性类型：String 缺省：如果未设置，则默认schema为连接使用的用户名。
loadBalanceHosts	指定是否使用负载均衡功能。 **须知** 集中式场景下，如果使用此参数需要保证业务中没有写操作。	属性类型：Boolean 取值范围： ● true：表示使用洗牌算法从候选主机中随机选择一个主机建立连接。 ● false：表示顺序连接url中指定的多个主机。 默认值：false
hostRecheckSeconds	JDBC尝试连接主机后会保存主机状态：连接成功或连接失败。在hostRecheckSeconds时间内保持可信，超过则状态失效。	属性类型：Integer 属性单位：s 取值范围：0 ~ 2147483647 默认值：10

属性名称	属性说明	属性值
ssl	表示以SSL方式连接数据库。包括NonValidatingFactory通道和使用证书的2种SSL连接方式。	属性类型：Boolean 取值范围： ● true：表示以SSL方式连接数据库。 ● false：表示不以SSL方式连接数据库。 默认值：该属性不设置则为空，与false等效。
sslmode	SSL认证方式。	属性类型：String 取值范围：disable、allow、prefer、require、verify-ca、verify-full。 ● disable：不使用SSL安全连接。 ● allow：如果数据库服务器要求使用，则可以使用SSL安全加密连接，但不验证数据库服务器的真实性。 ● prefer：如果数据库支持，那么首选使用SSL连接，但不验证数据库服务器的真实性。 ● require：只尝试SSL连接，不会检查服务器证书是否由受信任的CA签发，且不会检查服务器主机名与证书中的主机名是否一致。 ● verify-ca：只尝试SSL连接，并且验证服务器是否具有由可信任的证书机构签发的证书。 ● verify-full：只尝试SSL连接，并且验证服务器是否具有由可信任的证书机构签发的证书，以及验证服务器主机名是否与证书中的一致。 默认值：该属性不设置则为空，与require等效。
sslcert	提供客户端证书文件的完整路径。证书类型为End Entity。	属性类型：String 默认值：该属性不设置则为空，读取用户根目录。
sslkey	提供密钥文件的完整路径，使用时需将该密钥转换为DER格式。 `openssl pkcs8 -topk8 -outform DER -in client.key -out client.key.pk8 -nocrypt`	属性类型：String 默认值：该属性不设置则为空，读取用户根目录。

属性名称	属性说明	属性值
sslrootcert	SSL根证书的文件名。根证书的类型为CA。	属性类型：String 默认值：该属性不设置则为空，读取用户根目录。
sslpassword	SSL密码，提供给ConsoleCallbackHandler使用。	属性类型：String
sslpasswordcallback	SSL密码提供者的类名。	属性类型：String 默认值：com.huawei.opengauss.jdbc.ssl.jdbc4.LibPQFactory.ConsoleCallbackHandler
sslfactory	指定SSLSocketFactory在建立SSL连接时用的类名。	属性类型：String 默认值：该属性不设置则为空。
sslprivatekeyfactory	指定实现私钥解密方法的接口com.huawei.opengauss.jdbc.ssl.PrivateKeyFactory的实现类的完整限定类名。如果不提供，首先尝试默认的jdk私钥解密算法，如果无法解密，则使用com.huawei.opengauss.jdbc.ssl.BouncyCastlePrivateKeyFactory，用户需要提供bcpkix-jdk15on.jar包，版本建议1.65以上。	属性类型：String 默认值：该属性不设置则为空，会尝试使用默认的jdk私钥解密算法进行解密。
sslfactoryarg	此值是sslfactory类的构造函数的可选参数（不推荐使用）。	属性类型：String 默认值：该属性不设置则为空。
sslhostnameverifier	主机名验证程序的类名。接口实现javax.net.ssl.HostnameVerifier，默认使用com.huawei.opengauss.jdbc.ssl.PGjdbcHostnameVerifier。	属性类型：String 默认值：该属性不设置则为空，默认使用com.huawei.opengauss.jdbc.ssl.PGjdbcHostnameVerifier。

属性名称	属性说明	属性值
loginTimeout	指建立数据库连接的等待时间。当url配置多IP时，若获取连接花费的时间超过此值，则连接失败，不再尝试后续IP。 loginTimeout时间包括**connectTimeout**和**socketTimeoutInConnecting**，计算方式为：loginTimeout=(connectTimeout+连接认证时间+初始化语句执行时间)*节点数量。 **须知** • 此参数设置后对于多IP而言，时间是尝试连接IP的时间，可能会出现因为设置的值较小导致后面的IP无法连接的问题。例如设置了三个IP，如果loginTimeout为5s，但前两个IP建连总共用了5s，第三个IP会无法进行连接。 • 当CPU、内存、I/O负载中的任意一项接近100%时，会出现连接慢的现象，可能会导致连接时间超过阈值的问题，可通过以下方式进行问题排查： 1. 登录连接慢的物理机或通过管理工具查询资源负载：可通过top命令等确认CPU使用率；通过free命令确认内存使用情况；通过iostat命令确认I/O负载；此外还可以通过cm_agent中的监控日志，以及数据库运维平台中的监测记录进行检查。 2. 针对短时间内大量慢查询导致的峰值负载场景，可通过[数据库服务器的端口号+1]端口连接，查询pg_stat_activity视图；针对慢查询，可以使用系统函数**•pg_terminate_backend(pid int)**进行查杀会话。 3. 针对业务量长期超负载情况（即无明显慢查询，或慢查询查杀后但新的查询依然会变成慢查询），应考虑降低业务负载、增加数据库资源的方式进行优化。	属性类型：Integer 属性单位：s（秒） 取值范围：0 ~ 2147483647。0表示已禁用，timeout不生效。 默认值：0 建议：配置后每次建连都会开启一个异步线程，在连接数较多的情况可能会导致客户端压力增大，可以根据业务酌情调整此参数，建议调整为max(connectTimeout, socketTimeoutInConnecting) * 节点数，防止在网络异常情况且第n个IP为主的情况下无法连接。

属性名称	属性说明	属性值
connectTimeout	用于连接服务器操作系统的超时值。如果连接到服务器操作系统花费的时间超过此值，则连接断开。当url配置多IP时，表示连接单个IP的超时时间。	属性类型：Integer 属性单位：s（秒） 取值范围：0 ~ 2147483647。0表示已禁用，timeout不生效。 默认值：0
socketTimeout	用于socket读取操作的超时值。如果从服务器读取数据流所花费的时间超过此值，则连接关闭。如果不配置该参数，在数据库进程异常情况下，会导致客户端出现长时间等待，建议根据业务可以接受的SQL执行时间进行配置。	属性类型：Integer 属性单位：s（秒） 取值范围：0 ~ 2147483647。0表示已禁用，timeout不生效。 默认值：0
socketTimeoutInConnecting	用于控制建连阶段socket读取操作的超时值。在建连阶段，如果从服务器读取数据流超过此阈值，则尝试查找下一个节点建连。	属性类型：Integer 属性单位：s（秒） 取值范围：0 ~ 2147483647。0表示已禁用，timeout不生效。 默认值：5
statementTimeout	用于控制connection中statement执行时间的超时值。如果statement执行时间超过此值，则取消该statement执行。	属性类型：Integer 属性单位：ms（毫秒） 取值范围：0 ~ 2147483647。0表示已禁用，timeout不生效。 默认值：0
cancelSignalTimeout	发送取消命令的消息本身可能会阻塞，用于控制取消命令的“connect超时”和“socket超时”。如果取消命令超过指定时间未响应，会中断该连接，减少占用客户端资源。	属性类型：Integer 属性单位：s（秒） 取值范围：0 ~ 2147483647。0表示已禁用，timeout不生效。 默认值：10
tcpKeepAlive	TCP保活探测功能的控制开关。	属性类型：Boolean 取值范围： ● true：表示启用TCP保活探测功能。 ● false：表示禁用TCP保活探测功能。 默认值：false

属性名称	属性说明	属性值
logUnclosedConnections	客户端可能由于未调用Connection对象的close()方法而泄漏Connection对象。最终这些对象将被回收，并且调用finalize()方法。	属性类型：Boolean 取值范围： ● true：表示如果用户没有调用close()方法，finalize()方法将关闭Connection对象。 ● false：表示如果用户没有调用close()方法，将不会关闭Connection对象。 默认值：false
assumeMinServerVersion	该参数设置要连接的服务器版本。 如assumeMinServerVersion高于或者等于9.0，可以在建立连接时减少相关包的发送，客户端不会发送请求将float精度设置为3（即维持原来设置的float精度2）。	属性类型：String 默认值：该属性不设置则为空。
ApplicationName	表示正在连接的JDBC驱动名称。在数据库主节点上查询PG_STAT_ACTIVITY系统视图可以看到正在连接的客户端信息，application_name字段表示JDBC驱动名称。	属性类型：String 默认值：PostgreSQL JDBC Driver
connectionExtraInfo	表示驱动是否将当前驱动的部署路径、进程属主用户、url连接配置信息上报到数据库。	属性类型：Boolean 取值范围： ● true：表示JDBC驱动会将当前驱动的部署路径、进程属主用户、url连接配置信息上报到数据库中，记录在connection_info参数里，同时可以在PG_STAT_ACTIVITY和PGXC_STAT_ACTIVITY中查询到。 ● false：表示JDBC驱动不会将当前驱动的部署路径、进程属主用户、url连接配置信息上报到数据库中。 默认值：false

属性名称	属性说明	属性值
autosave	如果查询失败，指定驱动程序应该执行的操作。	属性类型：String 取值范围：always、never、conservative。 ● always：表示JDBC驱动程序在每次查询之前设置一个保存点，并在失败时回滚到该保存点。 ● never：表示无保存点。 ● conservative：表示每次查询都会设置保存点，但是只会在“statement XXX无效”等情况下回滚并重试。 默认值：never
protocolVersion	连接协议版本号。	属性类型：Integer 取值范围：目前仅支持1和3。 ● 1：表示连接V1服务端。 ● 3：表示连接V5服务端。 默认值：该属性不设置则为空，与设置为3等效。
prepareThreshold	该值决定PreparedStatement对象在执行多少次以后使用服务端已经解析好的statement。 **说明** 不建议通过JDBC，并且使用prepareStatement方法执行涉及密码的语句（例如：CREATE USER user_name WITH PASSWORD '********'；），原因是执行达到prepareThreshold指定次数时，数据库将SQL语句进行缓存，基于安全因素不会将密码进行缓存，该prepareStatement方法再次执行时会报错“Password must contain at least 8 characters.”。	属性类型：Integer 取值范围：0 ~ 2147483647 默认值：5。意味着在执行同一个PreparedStatement对象时，在第五次及以上执行时不再向服务端发送parse消息对statement进行解析，而使用之前在服务端已经解析好的statement。
preparedStatementCacheQueries	确定每个连接中缓存PreparedStatement对象所生成query的最大数量。	属性类型：Integer 取值范围：0 ~ 2147483647。0表示禁用缓存。 默认值：256。若在prepareStatement()调用中使用超过256个不同的查询，则最近最少使用的查询缓存将被丢弃。 建议：根据业务需要进行调整。配合prepareThreshold使用。

属性名称	属性说明	属性值
preparedStatementCacheSizeMiB	确定每个连接中缓存PreparedStatement对象所生成query的最大值。	属性类型：Integer 属性单位：MB 取值范围：0 ~ 2147483647。0表示禁用缓存。 默认值：5。若缓存了超过5MB的query，则最近最少使用的查询缓存将被丢弃。
databaseMetadataCacheFields	指定每个连接可缓存的字段的最大个数。	属性类型：Integer 取值范围：0 ~ 2147483647。0表示禁用缓存。 默认值：65536
databaseMetadataCacheFieldsMiB	指定每个连接可缓存的字段的最大值。	属性类型：Integer 属性单位：MB 取值范围：0 ~ 2147483647。0表示禁用缓存。 默认值：5
stringtype	指定调用java.sql.PreparedStatement#setString方法时传给数据库的参数类型。	属性类型：String 取值范围：unspecified、varchar。 ● varchar：表示参数将作为varchar类型发送给服务器。 ● unspecified：表示参数将不指定类型发送到服务器，服务器将尝试推断适当的类型。 默认值：varchar
batchMode	表示是否使用batch模式连接。 **说明** 配置batchMode=on时，执行批量插入/批量修改操作，每一列的数据类型以第一条数据指定的类型为准，若数据类型混用可能会导致报错或者插入的数据异常。	属性类型：String 取值范围： ● on：表示开启batch模式，可以提升批量更新的性能。设置batchMode=on执行成功的返回结果为[count, 0, 0...0]，数组第一个元素为批量影响的总条数。 ● off：表示关闭batch模式。设置batchMode=off执行成功的返回结果为[1, 1, 1...1]，数组各元素对应单次修改的影响条数。 默认值：on 建议：如果本身业务框架（例如hibernate）在批量更新时会检测返回值，可以通过调整此参数来解决。

属性名称	属性说明	属性值
fetchsize	设置数据库连接所创建statement的默认fetchsize。 确定一次fetch在ResultSet中读取的行数，与defaultRowFetchSize功能等价。如果fetchsize和defaultRowFetchSize同时设置，以fetchsize为准。	属性类型：Integer 取值范围：0 ~ 2147483647 默认值：0。表示一次从数据库获取所有结果。 建议：用户根据自身的业务查询数据数量和客户端机器内存情况来配置此参数，设置fetchsize时要关闭自动提交模式（autocommit=false），否则会导致fetchsize无法生效。
defaultRowFetchSize	确定一次fetch在ResultSet中读取的行数。限制每次访问数据库时读取的行数可以避免不必要的内存消耗，从而避免OutOfMemoryException。	属性类型：Integer 取值范围：0 ~ 2147483647 默认值：0。表示一次从数据库获取所有结果。
reWriteBatchedInserts	表示批量插入时，是否重写SQL语句。使用该参数时，要求设置batchMode=off。	属性类型：Boolean 取值范围： ● true：表示批量插入时，可将N条插入语句合并为一条：INSERT INTO TABLE_NAME VALUES (values1, ..., valuesN), ..., (values1, ..., valuesN); ● false：表示批量插入时，不重写SQL语句。 默认值：false
unknownLength	某些GaussDB类型（例如TEXT）没有明确定义的长度，当通过ResultSetMetaData.getColumnDisplaySize和ResultSetMetaData.getPrecision等函数返回关于这些类型的数据时，此参数指定未知长度类型的长度。	属性类型：Integer 取值范围：0 ~ 2147483647 默认值：2147483647

属性名称	属性说明	属性值
uppercaseAttributeName	该属性开启后会将获取元数据的接口的查询结果转为大写。主要适用场景为数据库中存储元数据全为小写，但要使用大写的元数据作为出参和入参。 涉及到的接口请参见：**6.3.4.3-java.sql.DatabaseMetaData**、**6.3.4.7-java.sql.ResultSetMetaData**。 **说明** uppercaseAttributeName参数开启后，如果数据库中有小写、大写和大小写混合的元数据，只能查询出小写部分的元数据，并以大写的形式输出，使用前请务必确认元数据的存储是否全为小写以避免数据出错。	属性类型：Boolean 取值范围： • true表示将获取元数据的接口的查询结果转为大写。 • false表示以内核GUC参数uppercase_attribute_name配置为准。 默认值：false
binaryTransfer	是否启用二进制格式发送和接收数据。	属性类型：Boolean 取值范围： • true：表示启用。 • false：表示不启用。 默认值：false
binaryTransferEnable	启用二进制传输的类型列表。以逗号分隔，例如：binaryTransferEnable=Integer4_ARRAY,Integer8_ARRAY。 OID名称和编号二选一。比如：OID名称为BLOB，编号为88，可以进行如下配置：binaryTransferEnable=BLOB 或 binaryTransferEnable=88。	属性类型：String 默认值：该属性不设置则为空。
binaryTransferDisable	禁用二进制传输的类型列表。以逗号分隔，OID名称和编号二选一。如果binaryTransferDisable与binaryTransferEnable有相同的value，则会禁用。	属性类型：String 默认值：该属性不设置则为空。
blobMode	设置setBinaryStream()方法为不同类型的数据赋值。建议从A、B迁移的系统将该值设定为on，从PG迁移的系统设定为off。	属性类型：String 取值范围： • on：表示为blob类型数据赋值。 • off：表示为bytea类型数据赋值。 默认值：on

属性名称	属性说明	属性值
socketFactory	创建与服务器socket连接的类的名称。该类必须实现接口“javax.net.SocketFactory”，并定义无参或单String参数的构造函数。	属性类型：String
socketFactoryArg	此值是上面提供的socketFactory类的构造函数的可选参数，不推荐使用。	属性类型：String
receiveBufferSize	该值用于设置连接流上的SO_RCVBUF。	属性类型：Integer 属性单位：字节 取值范围：-1 ~ 2147483647 默认值：-1。表示不设置缓冲区大小。
sendBufferSize	该值用于设置连接流上的SO_SNDBUF。	属性类型：Integer 属性单位：字节 取值范围：-1 ~ 2147483647 默认值：-1。表示不设置缓冲区大小。
preferQueryMode	指定执行查询的模式。	属性类型：String 取值范围：simple、extended、extendedForPrepared、extendedCacheEverything。 ● simple：表示只发送Q消息，仅支持文本模式，不支持parse与bind； ● extended：表示支持parse、bind和execute； ● extendedForPrepared：表示只有Prepared Statement对象使用扩展查询，Statement对象只使用简单查询； ● extendedCacheEverything：表示会缓存每个Statement对象所生成的query。 默认值：extended

属性名称	属性说明	属性值
targetServerType	指定要连接的服务器类型。该参数识别主备数据节点是通过查询url连接串中，数据节点是否允许写操作来实现的。	属性类型：String 取值范围：any、master、slave、preferSlave、clusterMainNode。 ● master表示依次尝试连接url中配置的IP，直到能够连接到数据库实例中的主节点，如果找不到将抛出异常。 ● slave表示依次尝试连接url中配置的IP，直到能够连接到数据库实例中的备节点，如果找不到将抛出异常。 ● preferSlave表示尝试连接到url中的备数据节点（如果有可用节点），否则连接到主数据节点。 ● any表示尝试连接到url中的任何一个数据节点。 ● clusterMainNode表示尝试连接到url中的主节点或首备节点（容灾主节点），如果找不到将抛出异常。 默认值：any 查询语句：SELECT local_role, db_state FROM pg_stat_get_stream_replications(); 建议：有写操作的业务建议配置master，以保证主备切换后能正常连接主节点，但是要注意在主备倒换过程中备机没有完全升主的时候无法正常建连，导致业务语句无法正常执行。
priorityServers	指定url上配置的前n个节点作为主数据库实例被优先连接。应用于流式容灾场景。 例如：jdbc:opengauss://host1:port1,host2:port2,host3:port3,host4:port4/database?priorityServers=2。即表示host1与host2为主数据库实例节点，host3与host4为容灾数据库实例节点。	属性类型：Integer 取值范围：大于0且小于url上配置的DN数量。 默认值：NULL

属性名称	属性说明	属性值
forceTargetServerSlave	此值用于控制是否开启强制连接备机功能，并在数据库实例发生主备切换时，禁止已存在的连接在升主备机上继续使用。	属性类型：Boolean 取值范围： ● false表示不开启强制连接备机功能。 ● true表示开启强制连接备机功能。 默认值：false
traceInterfaceClass	获取traceId方法接口com.huawei.opengauss.jdbc.log.Tracer的实现类的完整限定类名。	属性类型：String 默认值：NULL
use_boolean	设置extended模式下setBoolean()方法绑定的oid类型。	属性类型：Boolean 取值范围： ● false：表示绑定int2类型。 ● true：表示绑定bool类型。 默认值：false
allowReadOnly	设置是否允许连接开启只读模式。	属性类型：Boolean 取值范围： ● true：允许设置只读模式。 ● false：不允许设置只读模式，此时调用connection.setReadOnly(true)不生效，仍可以修改数据。 默认值：true
TLSCiphersSupperted	设置支持的TLS加密套件。	属性类型：String 默认值：TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384

属性名称	属性说明	属性值
stripTrailingZeros	设置是否去除numeric类型小数点后的0，仅对ResultSet.getObject(int columnIndex)生效。	属性类型：Boolean 取值范围： ● true：表示去除numeric类型小数点后的0。 ● false：表示不去除numeric类型小数点后的0。 默认值：false
enableTimeZone	指定是否启用客户端时区设置。	属性类型：Boolean 取值范围： ● true：表示启用客户端时区设置（获取JVM时区指定数据库时区）。 ● false：表示不启用客户端时区设置（使用数据库时区）。 默认值：true
oracleCompatible	用户控制驱动接口的A兼容特性。	属性类型：String 取值范围： ● true：表示开启驱动侧所有的A兼容特性。 ● false：表示关闭驱动侧所有的A兼容特性。 ● "tag1,tag2,tag3"：配置一个或多个tag，tag之间用逗号隔开，表示开启驱动侧部分的A兼容特性，每个tag对应一个A兼容特性。当前支持的tag有： – getProcedureColumns：DatabaseMetaData#getProcedureColumns接口的行为兼容A行为。 – batchInsertAffectedRows：reWriteBatchedInserts开启后，执行批量插入接口Statement#executeBatch的返回结果兼容A行为。 默认值：false
printSqlInLog	指定异常信息或日志中是否输出SQL语句。	属性类型：Boolean 取值范围： ● true：表示启用。 ● false：表示禁用。 默认值：true

属性名称	属性说明	属性值
setFloat	表示调用setFloat或者setObject指定类型为float时，传递给内核的Oid是否为float4的Oid。	属性类型：Boolean 取值范围： true：表示传递给内核的Oid为float4。 false：表示传递给内核的Oid为float8。 默认值：false

JDBC驱动支持数据库DDL操作修改表结构后正常执行。

jdbc.version_num仅用来在建连时向内核传递JDBC版本号，不做其他用途，请勿使用。jdbc.version_num从503.1版本开始支持（此前使用jdbc.version参数完成该功能，后续废弃），jdbc.version_num使用规范如下：

jdbc.version_num	版本	范围	使用规则	内核判断	备注
502xx	505.2	50200-50299	版本内必须先使用最小可用值	>=502xx最新设置值	相关兼容性改动必须同步合入更高版本分支
501xx	505.1	50100-50199	版本内必须先使用最小可用值	>=501xx最新设置值	相关兼容性改动必须同步合入更高版本分支
500xx	505.0	50000-50099	版本内必须先使用最小可用值	>=500xx最新设置值	相关兼容性改动必须同步合入更高版本分支
301xx	503.1	30100-30199	版本内必须先使用最小可用值	>=301xx最新设置值	相关兼容性改动必须同步合入更高版本分支
300xx	503.0	30000-30099	版本内必须先使用最小可用值	>=300xx最新设置值	相关兼容性改动必须同步合入更高版本分支

5.4.2.2.3 以非加密方式连接

JDBC以非加密方式连接数据库，首先要加载驱动，然后再创建数据库连接。因此本章节主要介绍加载驱动方式、创建数据库连接的接口、采用不同接口进行非加密连接。

加载驱动方式介绍

加载驱动（以opengaussjdbc.jar为例）有两种方法：

- 在代码中创建连接之前在任意位置装载：
  ```
  Class.forName("com.huawei.opengauss.jdbc.Driver");
  ```
- 在JVM启动时参数传递，指定驱动名称，适用于DOS窗口或者Linux上执行Java代码。jdbctest为测试用例程序的名称。
  ```
  java -Djdbc.drivers=com.huawei.opengauss.jdbc.Driver jdbctest;
  ```

  说明

  - 当使用gsjdbc4.jar时，上面的Driver类名相应修改为“org.postgresql.Driver”。
  - 由于GaussDB在JDBC的使用上与PG的使用方法保持兼容，所以同时在同一进程内使用两个JDBC的驱动的时候，可能会造成类名冲突。
  - 相比于PG驱动，GaussDB JDBC驱动主要做了以下特性的增强：
    - 支持SHA256加密方式登录。
    - 支持对接实现sf4j接口的第三方日志框架。
    - 支持容灾切换。

创建数据库连接的接口介绍

JDBC提供了三种接口，用于创建数据库连接。url、info、user、password参数描述，请参见**表5-8**。

接口一：DriverManager.getConnection(String url)。该方式需要把数据库用户名、密码写在url中，有一定的安全风险，因此不推荐使用。

接口二：DriverManager.getConnection(String url, String user, String password)。具体请参见**采用接口二连接数据库**。

接口三：DriverManager.getConnection(String url, Properties info)。具体请参见**采用接口三连接数据库**。

表 5-8 数据库连接参数

参数	描述
url	opengaussjdbc.jar数据库连接描述符。格式如下： ● jdbc:opengauss: ● jdbc:opengauss:database ● jdbc:opengauss://host/database ● jdbc:opengauss://host:port/database ● jdbc:opengauss://host:port/database?param1=value1&param2=value2 ● jdbc:opengauss://host1:port1,host2:port2/database?param1=value1&param2=value2 **说明** ● 使用gsjdbc200.jar时，将“jdbc:opengauss”修改为“jdbc:gaussdb” ● database为要连接的数据库名称。数据库名称缺省则与用户名一致。 ● host为数据库服务器名称或IP地址。 由于安全原因，数据库主节点禁止数据库内部其他节点无认证接入。如果要在数据库内部访问数据库主节点，请将JDBC程序部署在数据库主节点所在机器，host使用“127.0.0.1”。否则可能会出现“FATAL: Forbid remote connection with trust method!”错误。 建议业务系统单独部署在数据库外部，否则可能会影响数据库运行性能。 缺省情况下，连接服务器为localhost。 ● port为数据库服务器端口。 缺省情况下，会尝试连接到5431端口的database。 ● param为参数名称，即数据库连接属性。参数可以配置在urlL中，以“?”开始配置，以“=”给参数赋值，以“&”作为不同参数的间隔。 ● value为参数值，即数据库连接属性值。 ● 连接时需配置connectTimeout、socketTimeout，如果未配置，默认为0，即不会超时。在DN与客户端出现网络故障时，客户端一直未收到DN侧ACK确认报文，会启动超时重传机制，不断地进行重传。当重传次数达到默认的15次后才会报超时错误，会导致RTO时间很高。 ● 建议使用JDBC标准接口建立连接时，确保url格式的合法性，不合法的url会导致连接失败，且报错信息中包含原始url字符串，可能造成敏感信息泄漏。 ● url中不允许配置密码、凭证等敏感信息，建议通过Properties连接属性来设置敏感信息，参见**采用接口三连接数据库**。
info	数据库连接属性。全量参数请参见**连接参数参考**。
user	数据库用户。
password	数据库用户的密码。

采用接口二连接数据库

以下以opengaussjdbc.jar为例，使用DriverManager.getConnection(String url, String user, String password)接口创建数据库连接，步骤如下：

步骤1 导入java.sql.Connection和java.sql.DriverManager。

java.sql.Connection是数据库连接接口，通过java.sql.DriverManager的getConnection() 方法让应用程序连接到数据库。此外，用户需要根据实际的应用场景，再导入其他的接口和类，具体请参见**JDBC接口参考**。

```
import java.sql.Connection;
import java.sql.DriverManager;
```

步骤2 指定数据库sourceURL（$ip、$port、database需要用户自行修改）、用户名和密码。

用户名和密码直接写到代码中有很大的安全风险，建议在环境变量中存放。

```
String sourceURL = "jdbc:opengauss://$ip:$port/database";
String userName = System.getenv("EXAMPLE_USERNAME_ENV");
String password = System.getenv("EXAMPLE_PASSWORD_ENV");
```

步骤3 加载驱动。

1. 在代码运行工具（如IDE）中添加opengaussjdbc.jar包。
2. 执行以下命令加载数据库驱动程序“com.huawei.opengauss.jdbc.Driver”。

```
String driver = "com.huawei.opengauss.jdbc.Driver";
Class.forName(driver);
```

步骤4 创建数据库连接。

调用DriverManager.getConnection(String url, String user, String password)，进行数据库连接。

```
Connection conn = DriverManager.getConnection(sourceURL, userName, password);
```

----结束

采用接口三连接数据库

以下以opengaussjdbc.jar为例，使用DriverManager.getConnection(String url, Properties info)接口创建数据库连接，步骤如下：

步骤1 导入java.sql.Connection、java.sql.DriverManager、java.util.Properties。

java.util.Properties的setProperty()方法，用于设置Properties对象的属性值。此外，用户需要根据实际的应用场景，再导入其他的接口和类，具体请参见**JDBC接口参考**。

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.Properties;
```

步骤2 指定数据库sourceURL（$ip、$port、database需要用户自行修改）、用户名和密码。

用户名和密码直接写到代码中有很大的安全风险，建议在环境变量中存放。

```
String sourceURL = "jdbc:opengauss://$ip:$port/database";
String userName = System.getenv("EXAMPLE_USERNAME_ENV");
String password = System.getenv("EXAMPLE_PASSWORD_ENV");
```

步骤3 创建Properties对象，并将userName和password设置为该对象的属性值。

```
Properties info = new Properties();
info.setProperty("user", userName);
info.setProperty("password", password);
```

步骤4 加载驱动。

1. 在代码运行工具（如IDE）中添加opengaussjdbc.jar包。
2. 执行以下命令加载数据库驱动程序“com.huawei.opengauss.jdbc.Driver”。

```
String driver = "com.huawei.opengauss.jdbc.Driver";
Class.forName(driver);
```


步骤5 创建数据库连接。

调用DriverManager.getConnection(String url, Properties info)，进行数据库连接。

```
Connection conn = DriverManager.getConnection(sourceURL, info);
```

----结束

5.4.2.2.4 以 SSL 方式连接

用户通过JDBC连接GaussDB服务器时，可以通过开启SSL加密客户端和服务器之间的通讯，为敏感数据在Internet上的传输提供一种安全保障手段。以SSL方式连接数据库有使用NonValidatingFactory通道、使用证书认证2种方式。其中使用证书认证的连接方式，客户端和服务端双方会互相验证身份。本小节中，连接数据库采用的是接口三：DriverManager.getConnection(String url, Properties info)。

方式一：使用 NonValidatingFactory 通道

前置条件：用户已经获取服务端所需要的证书和私钥文件，并完成服务端配置。

> **说明**
>
> 关于证书生成和获取，请联系管理员。服务端配置证书的具体操作，请联系管理员。

以下以opengaussjdbc.jar为例，使用NonValidatingFactory通道的方式连接数据库的步骤如下：

步骤1 导入java.sql.Connection、java.sql.DriverManager、java.util.Properties。

此外，用户需要根据实际的应用场景，再导入其他的接口和类，详见**JDBC接口参考**。

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.Properties;
```

步骤2 指定数据库sourceURL（$ip、$port、database需要用户自行修改）、用户名和密码。

用户名和密码直接写到代码中有很大的安全风险，建议在环境变量中存放。

```
String sourceURL = "jdbc:opengauss://$ip:$port/database";
Properties urlProps = new Properties();
urlProps.setProperty("user", System.getenv("EXAMPLE_USERNAME_ENV"));
urlProps.setProperty("password", System.getenv("EXAMPLE_PASSWORD_ENV"));
```

步骤3 设置ssl属性为true，使用NonValidatingFactory通道。

```
urlProps.setProperty("ssl", "true");
urlProps.setProperty("sslfactory","com.huawei.opengauss.jdbc.ssl.NonValidatingFactory");
```

步骤4 加载驱动。

1. 在代码运行工具（如IDE）中添加opengaussjdbc.jar包。
2. 执行以下命令加载数据库驱动程序“com.huawei.opengauss.jdbc.Driver”。

```
Class.forName("com.huawei.opengauss.jdbc.Driver");
```

步骤5 创建数据库连接。

调用DriverManager.getConnection(String url, Properties info)，进行数据库连接。

```
Connection conn = DriverManager.getConnection(sourceURL,urlProps);
```

----结束

方式二：使用证书认证

前置条件：用户已经获取服务端所需要的证书和私钥文件，并完成服务端配置；同时已经获取客户端所需要的client.crt客户端证书、cacert.pem根证书、client.key.pk8客户端私钥文件，以下**步骤3**介绍如何将证书和私钥文件配置在客户端。

> **说明**
>
> 关于证书生成和获取，请联系管理员。服务端配置证书的具体操作，请联系管理员。

以下以opengaussjdbc.jar为例，使用客户端配置证书的方式连接数据库的步骤如下：

步骤1 导入java.sql.Connection、java.sql.DriverManager、java.util.Properties。

此外，用户需要根据实际的应用场景，再导入其他的接口和类，详见**JDBC接口参考**。

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.Properties;
```

步骤2 指定数据库sourceURL（$ip、$port、database需要用户自行修改）、用户名和密码。

用户名和密码直接写到代码中有很大的安全风险，建议在环境变量中存放。

```
String sourceURL = "jdbc:opengauss://$ip:$port/database";
Properties urlProps = new Properties();
urlProps.setProperty("user", System.getenv("EXAMPLE_USERNAME_ENV"));
urlProps.setProperty("password", System.getenv("EXAMPLE_PASSWORD_ENV"));
```

步骤3 设置ssl属性为true，在客户端配置client.crt客户端证书、cacert.pem根证书、client.key.pk8客户端私钥。

```
urlProps.setProperty("ssl", "true");
urlProps.setProperty("sslcert", "client.crt");
urlProps.setProperty("sslrootcert", "cacert.pem");
urlProps.setProperty("sslkey", "client.key.pk8");
```

> **说明**
>
> 客户端私钥文件使用前，需要将client.key转化为client.key.pk8格式：
>
> ```
> * openssl pkcs8 -topk8 -outform DER -in client.key -out client.key.pk8 -nocrypt
> * openssl pkcs8 -topk8 -inform PEM -in client.key -outform DER -out client.key.der -v1 PBE-MD5-DES
> * openssl pkcs8 -topk8 -inform PEM -in client.key -outform DER -out client.key.der -v1 PBE-SHA1-3DES
> ```
>
> 以上算法由于安全级别较低，不推荐使用。
>
> 如果客户需要采用更高级别的私钥加密算法，启用bouncycastle或者其他第三方私钥解密密码包后可以使用的私钥加密算法如下：
>
> ```
> * openssl pkcs8 -in client.key -topk8  -outform DER -out client.key.der -v2 AES128
> * openssl pkcs8 -in client.key -topk8  -outform DER -out client.key.der -v2 aes-256-cbc -iter 1000000
> * openssl pkcs8 -in client.key -topk8 -out client.key.der  -outform Der -v2 aes-256-cbc -v2prf hmacWithSHA512
> * 启用bouncycastle：使用jdbc的项目引入依赖：bcpkix-jdk15on.jar和bcprov-ext-jdk15on.jar包，版本建议：1.65以上。
> ```

步骤4 配置sslmode。

sslmode设置值：require、verify-ca、verify-full，参数介绍详见**sslmode**。客户根据应用场景选择一种即可。

```
/* 设置sslmode为require */
urlProps.setProperty("sslmode", "require");
/* 设置sslmode为verify-ca */
urlProps.setProperty("sslmode", "verify-ca");
/* 设置sslmode为verify-full（Linux下验证） */
urlProps.setProperty("sslmode", "verify-full");
```

步骤5 加载驱动。

1. 在代码运行工具（如IDE）中添加opengaussjdbc.jar包。
2. 执行以下命令加载数据库驱动程序“com.huawei.opengauss.jdbc.Driver”。

```
Class.forName("com.huawei.opengauss.jdbc.Driver");
```

步骤6 创建数据库连接。

调用DriverManager.getConnection(String url, Properties info)，进行数据库连接。

```
Connection conn = DriverManager.getConnection(sourceURL,urlProps);
```

----结束

5.4.2.2.5 以 UDS 方式连接

Unix domain socket用于同一主机上不同进程间的数据交换，通过添加junixsocket获取套接字工厂使用。

前置条件：引用junixsocket-core-XXX.jar、junixsocket-common-XXX.jar、junixsocket-native-common-XXX.jar，XXX为版本号，引用的这些jar包版本号需要一致。

以下以opengaussjdbc.jar为例，以UDS方式连接数据库的步骤如下：

步骤1 导入java.sql.Connection、java.sql.DriverManager、java.util.Properties。

此外，用户需要根据实际的应用场景，再导入其他的接口和类，详见**JDBC接口参考**。

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.Properties;
```

步骤2 指定数据库的用户名和密码。

用户名和密码直接写到代码中有很大的安全风险，建议在环境变量中存放。并将用户名和密码设置为Properties对象的属性值。

```
String userName = System.getenv("EXAMPLE_USERNAME_ENV");
String password = System.getenv("EXAMPLE_PASSWORD_ENV");
Properties properties = new Properties();
properties.setProperty("user", userName);
properties.setProperty("password", password);
```

步骤3 加载驱动。

1. 在代码运行工具（如IDE）中添加opengaussjdbc.jar包。
2. 执行以下命令加载数据库驱动程序“com.huawei.opengauss.jdbc.Driver”。

```
String driver = "com.huawei.opengauss.jdbc.Driver";
Class.forName(driver);
```

步骤4 指定数据库的$ip、$port、database、socketFactory和socketFactoryArg。

$ip和$port需要用户自行修改，连接主机名database必须设置为“localhost”，socketFactory设置为org.newsclub.net.unix.AFUNIXSocketFactory$FactoryArg，socketFactoryArg设置为*[path-to-the-unix-socket]*，实现以UDS方式连接数据库。socketFactory和socketFactoryArg参数的具体说明，详见**socketFactory**和**socketFactoryArg**。

```
Connection conn = DriverManager.getConnection("jdbc:opengauss://$ip:$port/localhost?
socketFactory=org.newsclub.net.unix" +
        ".AFUNIXSocketFactory$FactoryArg&socketFactoryArg=[path-to-the-unix-socket]",properties);
System.out.println("Connection Successful!");
```


说明

socketFactoryArg参数配置根据真实路径进行配置，与GUC参数unix_socket_directory的值保持一致。

----结束

5.4.2.3 执行 SQL 语句

本小节中，**执行普通SQL语句**创建customer_t1表，**执行预处理插入语句**批量插入数据，**执行预处理更新语句**更新数据，同时演示**创建和调用存储过程**。

执行普通 SQL 语句

应用程序通过执行SQL语句来操作数据库，支持对XML类型数据进行SELECT、UPDATE、INSERT、DELETE等操作。

前置条件是已经连接数据库，连接对象为conn。执行普通SQL语句，创建customer_t1表的步骤如下：

步骤1 调用Connection接口的createStatement方法创建语句对象stmt。

```
Statement stmt = conn.createStatement();
```

步骤2 调用Statement接口的executeUpdate方法执行SQL语句。

```
int rc = stmt.executeUpdate("CREATE TABLE customer_t1(c_customer_sk INTEGER, c_customer_name VARCHAR(32));");
```

步骤3 调用Statement接口的close方法关闭语句对象stmt。

```
stmt.close();
```

----结束

说明

- 数据库中收到的一次执行请求（不在事务块中），如果含有多条语句，将会被打包成一个事务，事务块中不支持vacuum操作。如果其中有一个语句失败，那么整个请求都将会被回滚。
- 使用Statement执行多语句时应以“;”作为各语句间的分隔符，存储过程、函数和匿名块不支持多语句执行。当preferQueryMode=simple，语句执行不进行解析逻辑，此场景下无法使用“;”作为多语句间的分隔符。
- “/”可用作创建单个存储过程、函数、匿名块、包体的结束符。当preferQueryMode=simple，语句执行不进行解析逻辑，此场景下无法使用“/”作为结束符。
- 由于JDBC会对prepareStatement中的SQL语句进行缓存，可能导致内存膨胀，如果JVM内存较小，建议调整preparedStatementCacheSizeMiB或者preparedStatementCacheQueries。

执行预处理插入语句

用一条预处理语句处理多条相似的数据，数据库只创建一次执行计划，节省了语句的编译和优化时间。

前置条件是执行以上的普通SQL语句，已经创建customer_t1表。执行预处理语句，批量插入数据的步骤如下：

步骤1 调用Connection接口的prepareStatement方法创建预处理语句对象pst。

```
PreparedStatement pst = conn.prepareStatement("INSERT INTO customer_t1 VALUES (?,?)");
```


步骤2 针对每条数据要调用对应接口设置参数，以及调用addBatch将SQL语句添加到批处理中。

```
for (int i = 0; i < 3; i++) {
    pst.setInt(1, i);
    pst.setString(2, "data " + i);
    pst.addBatch();
}
```

步骤3 调用PreparedStatement接口的executeBatch方法执行批处理。

```
pst.executeBatch();
```

步骤4 调用PreparedStatement接口的close方法关闭预处理语句对象pst。

```
pst.close();
```

----结束

说明

在实际的批处理过程中，通常不终止批处理程序的执行，否则会降低数据库的性能。因此在批处理程序时，应该关闭自动提交功能，每几行提交一次。关闭自动提交功能的语句为：

```
conn.setAutoCommit(false);
```

执行预处理更新语句

预编译语句是只编译和优化一次，可以通过设置不同的参数值多次使用。由于已经预先编译好，后续使用会减少执行时间。因此，如果多次执行一条语句，请选择使用预编译语句。

前置条件是执行以上的预处理语句，customer_t1表已经批量插入数据。执行预编译SQL语句对数据进行更新的步骤如下：

步骤1 调用Connection接口的prepareStatement方法创建预编译语句对象pstmt。

```
PreparedStatement pstmt = conn.prepareStatement("UPDATE customer_t1 SET c_customer_name = ? WHERE c_customer_sk = 1");
```

步骤2 调用PreparedStatement接口的setString方法设置参数。

```
pstmt.setString(1, "new Data");
```

步骤3 调用PreparedStatement接口的executeUpdate方法执行预编译SQL语句。

```
int rowcount = pstmt.executeUpdate();
```

步骤4 调用PreparedStatement接口的close方法关闭预编译语句对象pstmt。

```
pstmt.close();
```

----结束

须知

prepareStatement设置绑定参数后，最终会构建成一个B报文或U报文，在下一步执行SQL语句时发给服务端。但是B报文或U报文有最大长度限制（不能超过1023MB），如果一次绑定数据过大，可能因报文过长导致异常。因此prepareStatement设置绑定参数时需要注意评估和控制绑定数据的大小，避免出现超出报文上限要求的现象。

创建和调用存储过程

GaussDB支持通过JDBC调用存储过程，前置条件是数据库建连完成、连接对象为conn。

创建存储过程testproc如下：

```
//在数据库中创建如下存储过程，它带有out参数。
CREATE OR REPLACE procedure testproc
(
    psv_in1 in integer,
    psv_in2 in integer,
    psv_inout inout integer
)
AS
BEGIN
    psv_inout := psv_in1 + psv_in2 + psv_inout;
END;
/
```

调用存储过程testproc如下：

步骤1 调用Connection的prepareCall方法创建调用语句对象cstmt。

```
CallableStatement cstmt = conn.prepareCall("{? = CALL testproc(?,?,?)}");
```

步骤2 调用CallableStatement的setInt方法设置参数。

```
cstmt.setInt(2, 50);
cstmt.setInt(1, 20);
cstmt.setInt(3, 90);
```

步骤3 调用CallableStatement的registerOutParameter方法注册输出参数。

```
cstmt.registerOutParameter(4, Types.INTEGER);  //注册out类型的参数，类型为整型。
```

步骤4 调用CallableStatement的execute方法执行SQL语句。

```
cstmt.execute();
```

步骤5 调用CallableStatement的getInt方法获取out输出参数。

```
int out = cstmt.getInt(4);
```

步骤6 调用CallableStatement的close方法关闭调用语句对象cstmt。

```
cstmt.close();
```

----结束

说明

- 一些JDBC驱动程序提供命名参数的方法来设置参数。命名参数的方法允许根据名称而不是顺序来设置参数，若参数有默认值，则可以不用指定参数值就可以使用此参数的默认值。即使存储过程中参数的顺序发生了变更，也不必修改应用程序。目前GaussDB数据库的JDBC驱动程序不支持此方法。
- GaussDB数据库不支持带有输出参数的函数，也不支持存储过程和函数参数默认值。
- conn.prepareCall("{? = CALL TESTPROC(?,?,?)}")，执行存储过程绑定参数时，可以按照占位符的顺序绑定参数，注册第一个参数为出参。也可以按照存储过程中的参数顺序绑定参数，注册第四个参数为出参，上述用例为此场景，注册第四个参数为出参。
- 当游标作为存储过程的返回值时，如果使用JDBC调用该存储过程，返回的游标将不可用。
- 存储过程不能和普通SQL在同一条语句中执行。
- 存储过程中inout类型参数必须注册出参。

创建和调用存储过程（入参为复合数据类型）

以下用例展示A兼容模式下，入参为复合数据类型的存储过程创建和调用情况。前置条件是数据库建连完成、连接对象为conn。

创建存储过程test_proc如下：

```
// 在数据库创建复合数据类型。
CREATE TYPE compfoo AS (f1 int, f3 text);
```



```
// 在数据库中创建table类型。
create type compfoo_table is table of compfoo;
// 在数据库中创建如下存储过程，它带有out参数。
create or replace procedure test_proc
(
    psv_in in compfoo,
    table_in in compfoo_table,
    psv_out out compfoo,
    table_out out compfoo_table
)
as
begin
    psv_out := psv_in;
    table_out:=compfoo_table();
    table_out.extend(table_in.count);
    for i in 1..table_in.count loop
        table_out(i):=table_in(i);
    end loop;
end;
/
```

调用存储过程test_proc如下：

步骤1 设置参数behavior_compat_options='proc_outparam_override'后，调用Connection的prepareCall方法创建调用语句对象cs。

```
Statement statement = conn.createStatement();
statement.execute("SET behavior_compat_options='proc_outparam_override'");
CallableStatement cs = conn.prepareCall("{ CALL test_proc(?,?,?,?) }");
```

步骤2 调用CallableStatement的set方法设置参数。

```
PGobject pGobject = new PGobject();
pGobject.setType("public.compfoo"); // 设置复合类型名，格式为“schema.typename”。
pGobject.setValue("(1,demo)"); // 绑定复合类型值，格式为“(value1,value2)”。
cs.setObject(1, pGobject);
pGobject = new PGobject();
pGobject.setType("public.compfoo_table"); // 设置Table类型名，格式为"schema.typename"。
pGobject.setValue("{\"(10,demo10)\",\"(11,demo111)\"}"); //绑定Table类型值，格式为
"{\"(value1,value2)\",\"(value1,value2)\",...}"。
cs.setObject(2, pGobject);
```

步骤3 调用CallableStatement的registerOutParameter方法注册输出参数。

```
// 注册out类型的参数，类型为复合类型,格式为“schema.typename”。
cs.registerOutParameter(3, Types.STRUCT, "public.compfoo");
// 注册out类型的参数，类型为Table类型,格式为“schema.typename”。
cs.registerOutParameter(4, Types.ARRAY, "public.compfoo_table");
```

步骤4 调用CallableStatement的execute方法执行SQL语句。

```
cs.execute();
```

步骤5 调用CallableStatement的getObject方法获取输出参数。

```
// 返回结构是自定义类型。
PGobject result = (PGobject)cs.getObject(3);  // 获取out参数
result.getValue(); // 获取复合类型字符串形式值。
result.getArrayValue(); //获取复合类型数组形式值，以复合数据类型字段顺序排序。
result.getStruct(); //获取复合类型子类型名，按创建顺序排序。
result.getAttributes(); //返回自定义类型每列组成类型的对象，对于array类型和table类型返回的是PgArray,对于
自定义类型，封装的是PGobject，对于其他类型数据存储方式为字符串类型。
// 返回结果是Table类型。
PgArray pgArray = (PgArray) cs.getObject(4);
ResultSet rs = pgArray.getResultSet();
while (rs.next()) {
  rs.getObject(2);// table类型每行的数据构建成的对象。
}
```


须知

如果出参的table类型组成为自定义类型，例如create type compfoo_table is table of compfoo，此时接收到的返回对象为PgArray，在通过rs.getObject(2)遍历获取到的组成对象也为PgArray，此时无法获取到组成它的compfoo类型对应的每列数据，需要通过getPGobject()获取到PgObject再操作获取。

步骤6 调用CallableStatement的close方法关闭调用语句对象cs。

```
cs.close();
```

----结束

说明

- A兼容模式开启参数后，调用存储过程必须使用{call proc_name(?,?,?)}形式调用，调用函数必须使用{? = call func_name(?,?)}形式调用（等号左侧的“？”为函数返回值的占位符，用于注册函数返回值）。
- 参数behavior_compat_options='proc_outparam_override'变更后，业务需要重新建立连接，否则无法正确调用存储过程和函数。
- 函数和存储过程中包含复合类型时，参数的绑定与注册需要使用schema.typename形式。

5.4.2.4 处理结果集

执行SQL语句后，需要对结果集进行处理。因此本章节主要分为这几部分内容：设置结果集类型、在结果集中定位、获取结果集中光标的位置、获取结果集中的数据。

设置结果集类型

不同类型的结果集有各自的应用场景，应用程序需要根据实际情况选择相应的结果集类型。在执行SQL语句过程中，需要先创建相应的语句对象，而部分创建语句对象的方法提供了设置结果集类型的功能。java.sql.Connection接口提供的创建语句对象的三种方法如下：

```
//创建一个Statement对象，该对象将生成具有给定类型和并发性的ResultSet对象。
createStatement(int resultSetType, int resultSetConcurrency);

//创建一个PreparedStatement对象，该对象将生成具有给定类型和并发性的ResultSet对象。
prepareStatement(String sql, int resultSetType, int resultSetConcurrency);

//创建一个CallableStatement对象，该对象将生成具有给定类型和并发性的ResultSet对象。
prepareCall(String sql, int resultSetType, int resultSetConcurrency);
```


表 5-9 结果集类型

参数	描述
resultSetType	表示结果集的类型，具体有三种类型： • ResultSet.TYPE_FORWARD_ONLY：ResultSet只能向前移动。是缺省值。 • ResultSet.TYPE_SCROLL_SENSITIVE：在修改后重新滚动到修改所在行，可以看到修改后的结果。 • ResultSet.TYPE_SCROLL_INSENSITIVE：对可修改例程所做的编辑不进行显示。 **说明** 结果集从数据库中读取了数据之后，即使类型是ResultSet.TYPE_SCROLL_SENSITIVE，也不会看到由其他事务在这之后引起的改变。调用ResultSet的refreshRow()方法，可进入数据库并从其中取得当前游标所指记录的最新数据。
resultSetConcurrency	表示结果集的并发，具体有两种类型： • ResultSet.CONCUR_READ_ONLY：如果不从结果集中的数据建立一个新的更新语句，不能对结果集中的数据进行更新。 • ResultSet.CONCUR_UPDATEABLE：可改变的结果集。对于可滚动的结果集，可对结果集进行适当的改变。

在结果集中定位

ResultSet对象具有指向其当前数据行的光标。最初，光标被置于第一行之前，next方法将光标移动到下一行。因为该方法在ResultSet对象没有下一行时返回false，所以可以在while循环中使用它来迭代结果集。但对于可滚动的结果集，JDBC驱动程序提供更多的定位方法，使ResultSet指向特定的行。定位方法如表5-10所示。

表 5-10 在结果集中定位的方法

方法	描述
next()	把ResultSet向下移动一行。
previous()	把ResultSet向上移动一行。
beforeFirst()	把ResultSet定位到第一行之前。
afterLast()	把ResultSet定位到最后一行之后。
first()	把ResultSet定位到第一行。
last()	把ResultSet定位到最后一行。
absolute(int row)	把ResultSet移动到参数指定的行数。
relative(int rows)	rows为正数表示把ResultSet向下移动rows行，rows为负数表示把ResultSet向上移动（-rows）行。

获取结果集中光标的位置

对于可滚动的结果集，可调用定位方法来改变光标的位置。JDBC驱动程序提供了获取结果集中光标所处位置的方法。获取光标位置的方法如表5-11所示。

表 5-11 获取结果集光标的位置

方法	描述
isFirst()	是否在第一行。
isLast()	是否在最后一行。
isBeforeFirst()	是否在第一行之前。
isAfterLast()	是否在最后一行之后。
getRow()	获取当前在第几行。

获取结果集中的数据

ResultSet对象提供了丰富的方法，以获取结果集中的数据。获取数据常用的方法如表5-12所示，其他方法请参见JDK官方文档。

表 5-12 ResultSet 对象的常用方法

方法	描述
getInt(int columnIndex)	按列标获取int型数据。
getInt(String columnLabel)	按列名获取int型数据。
getString(int columnIndex)	按列标获取String型数据。
getString(String columnLabel)	按列名获取String型数据。
getDate(int columnIndex)	按列标获取Date型数据
getDate(String columnLabel)	按列名获取Date型数据。

5.4.2.5 关闭数据库连接

在使用数据库连接完成相应的数据操作后，需要关闭数据库连接。

关闭数据库连接可以直接调用close方法。

```
conn.close();
```

说明

很多数据库类如Connection、Statement和ResultSet都有close()方法，在使用完对象后应把它们关闭。Connection对象的关闭将间接关闭所有与它关联的Statement对象，Statement对象的关闭将间接关闭ResultSet对象。

5.4.3 典型应用开发示例

本章的Java代码示例均使用opengaussjdbc.jar包。

5.4.3.1 不同场景下配置连接参数

说明

以下示例场景中node代表“host:port”，host为数据库服务器名称或IP地址，port为数据库服务器端口。

容灾场景

某客户有两套数据库实例，其中A数据库实例为生产数据库实例，B数据库实例为容灾数据库实例。当客户执行容灾切换时，A数据库实例将降为容灾数据库实例，B数据库实例将升为生产数据库实例。此时为了避免修改配置文件导致的应用重启或重新发版，客户可在初始配置文件时，即将A、B数据库实例写入连接串中。此时在主数据库实例不可连接时，驱动将尝试对容灾数据库实例建连。例如A数据库实例为{node1,node2,node3}。B数据库实例为{node4,node5,node6}。

url可参考如下进行配置：

```
jdbc:opengauss://node1,node2,node3,node4,node5,node6/database?priorityServers=3
```

如果想要能连接主集群的同时，可以连接到主集群内的主机，需要同时配置targetServerType=master，url可以参考如下进行配置：

```
jdbc:opengauss://node1,node2,node3,node4,node5,node6/database?
priorityServers=3&targetServerType=master
```

负载均衡场景

某客户存在一套集中式数据库实例，包含1主2备三个节点{node1，node2，node3}，其中node1为主节点，node2、node3为备节点。

客户希望同一应用程序上建立的连接，较为均匀的分布在三个节点上，则url可参考如下进行配置：

```
jdbc:opengauss://node1,node2,node3/database?loadBalanceHosts=true
```

注意

使用loadBalanceHosts时，若连接建立在备DN上，将无法执行写操作。如果业务需要执行写操作，请勿配置该参数。

自动寻主场景

某客户存在一套集中式数据库实例，包含1主2备三个节点{node1，node2，node3}，其中node1为主节点，node2、node3为备节点。

客户希望应用连接能建立在主DN上，并在发生主备切换时，自动选择新的主节点建连，则url可参考如下进行配置：

```
jdbc:opengauss://node1,node2,node3/database?targetServerType=master
```


日志诊断场景

某客户在使用中出现数据导入慢或出现一些难以分析的异常报错时，可通过开启trace日志进行诊断，url可参考如下进行配置：

```
jdbc:opengauss://node1/database?loggerLevel=trace
```

高性能场景

某客户对于相同SQL可能多次执行，仅是传参不同的情况，为了提升执行效率，可开启**prepareThreshold**参数，避免重复生成执行计划，url可参考如下进行配置：

```
jdbc:opengauss://node1/database?prepareThreshold=5
```

某客户一次查询1000万数据，为避免同时返回造成内存溢出，可使用**defaultRowFetchSize**，url可参考如下进行配置：

```
jdbc:opengauss://node1/database?defaultRowFetchSize=50000
```

某客户需要批量插入1000万数据，为提升效率，可使用**batchMode**，url可参考如下进行配置：

```
jdbc:opengauss://node1/database?batchMode=on
```

大小写转换场景

在GaussDB中元数据默认存储为小写，如果从元数据默认存储为大写的数据库迁移至GaussDB时，大写的元数据会变为小写。如果原业务中涉及到大写元数据的处理，可以开启此参数，但是不建议通过这种方式来解决问题，最好通过修改业务编码来解决。如果一定要使用，请务必确认当前数据库中的元数据是否全为小写，以避免出现问题。

```
jdbc:opengauss://node1/database?uppercaseAttributeName=true
```

对于DatabaseMetaData中涉及的接口，按照入参直接调用即可，对于ResultSetMetaData中涉及的接口调用方法如下所示：

```
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM test_supper");
ResultSetMetaData rsmd = rs.getMetaData();
for (int i = 1; i <= rsmd.getColumnCount(); i++) {
    System.out.println(rsmd.getColumnLabel(i) + "    " + rsmd.getColumnName(i));
}
```

5.4.3.2 创建和调用存储过程

本示例演示如何连接数据库、创建和调用存储过程。

```
// 以下用例以opengaussjdbc.jar为例。
// 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放（密码应密文存放，使用时解密），确保安全。
// 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量（环境变量名称请根据自身情况进行设置）EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
// $ip、$port、database需要用户自行修改。

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.CallableStatement;
import java.sql.Types;
```



```
public class DBTest {

 // 创建数据库连接。
 public static Connection GetConnection(String username, String passwd) {
   String driver = "com.huawei.opengauss.jdbc.Driver";
   String sourceURL = "jdbc:opengauss://$ip:$port/database";
   Connection conn = null;
   try {
     // 加载数据库驱动。
     Class.forName(driver);
   } catch (Exception e) {
     e.printStackTrace();
     return null;
   }

   try {
     //创建数据库连接。
     conn = DriverManager.getConnection(sourceURL, username, passwd);
     System.out.println("Connection succeed!");
   } catch (Exception e) {
     e.printStackTrace();
     return null;
   }

   return conn;
 };


 // 创建存储过程。
 public static void CreateCallable(Connection conn) {
   Statement stmt = null;
   try {
     stmt = conn.createStatement();
     // 创建存储过程，返回三个输入值的和。
     stmt.execute("create or replace procedure testproc \n" +
         "(\n" +
         "    psv_in1 in integer,\n" +
         "    psv_in2 in integer,\n" +
         "    psv_inout inout integer\n" +
         ")\n" +
         "as\n" +
         "begin\n" +
         "    psv_inout := psv_in1 + psv_in2 + psv_inout;\n" +
         "end;\n" +
         "/");
   } catch (SQLException e) {
     throw new RuntimeException(e);
   } finally {
     if (stmt != null) {
       try {
         stmt.close();
       } catch (SQLException e) {
         throw new RuntimeException(e);
       }
     }
   }
 }

 // 调用存储过程。
 public static void ExecCallableSQL(Connection conn) {
   CallableStatement cstmt = null;
   try {
     cstmt=conn.prepareCall("{? = CALL TESTPROC(?,?,?)}");
     cstmt.setInt(2, 50);
     cstmt.setInt(1, 20);
     cstmt.setInt(3, 90);
     cstmt.registerOutParameter(4, Types.INTEGER);  //注册out类型的参数，类型为整型。
     cstmt.execute();
     int out = cstmt.getInt(4);  //获取out参数
```



```
      System.out.println("The CallableStatment TESTPROC returns:"+out);
      cstmt.close();
    } catch (SQLException e) {
      if (cstmt != null) {
        try {
          cstmt.close();
        } catch (SQLException e1) {
          e1.printStackTrace();
        }
      }
      e.printStackTrace();
    }
  }


  /**
   * 主程序，逐步调用各静态方法。
   * @param args
  */
  public static void main(String[] args) {
    // 创建数据库连接。
    String userName = System.getenv("EXAMPLE_USERNAME_ENV");
    String password = System.getenv("EXAMPLE_PASSWORD_ENV");
    Connection conn = GetConnection(userName, password);

    // 创建存储过程。
    CreateCallable(conn);

    // 调用存储过程。
    ExecCallableSQL(conn);

    // 关闭数据库连接。
    try {
      conn.close();
    } catch (SQLException e) {
      e.printStackTrace();
    }

  }

}
```

上述示例的运行结果为：

```
Connection succeed!
The CallableStatment TESTPROC returns:160
```

5.4.3.3 获取函数返回值

JDBC调用函数时获取返回值，以下示例展示返回值类型为bit和float8两种数据类型，其他数据类型可参考本示例。

```
// 以下用例以opengaussjdbc.jar为例。
// 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放（密码应密文存放，使用时解密），确保安全。
// 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量（环境变量名称请根据自身情况进行设置）EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
// $ip、$port、database需要用户自行修改。

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.sql.CallableStatement;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.sql.Types;

public class Type {
    public static void main(String[] args) throws SQLException {
```

```
        String driver = "com.huawei.opengauss.jdbc.Driver";
        String username = System.getenv("EXAMPLE_USERNAME_ENV");
        String passwd = System.getenv("EXAMPLE_PASSWORD_ENV");
        String sourceURL = "jdbc:opengauss://$ip:$port/database";
        Connection conn = null;

        try {
            // 加载数据库驱动。
            Class.forName(driver);
        } catch (Exception e) {
            e.printStackTrace();
        }

        try {
            // 以非加密方式创建数据库连接。
            conn = DriverManager.getConnection(sourceURL, username, passwd);
            System.out.println("Connection succeed!");
        } catch (Exception e) {
            e.printStackTrace();
        }

        // 建表。
        String createsql = "create table if not exists t_bit(col_bit bit, col_bit1 int)";
        Statement stmt = conn.createStatement();
        stmt.execute(createsql);
        stmt.close();
        // bit类型使用示例，注意此处bit类型取值范围[0,1]。
        Statement st = conn.createStatement();
        String sqlstr = "create or replace function fun_1()\n" + "returns bit AS $$\n"
            + "select col_bit from t_bit limit 1;\n" + "$$\n" + "LANGUAGE SQL;";
        st.execute(sqlstr);
        CallableStatement c = conn.prepareCall("{ ? = call fun_1() }");
        // 注册输出类型，位串类型。
        c.registerOutParameter(1, Types.BIT);
        c.execute();
        // 使用Boolean类型获取结果。
        System.out.println(c.getBoolean(1));

        // float8类型使用示例。
        st.execute("create table if not exists t_float(col1 float8,col2 int)");
        PreparedStatement pstm = conn.prepareStatement("insert into t_float values(?)");
        pstm.setDouble(1, 123456.123);
        pstm.execute();
        pstm.close();

        // 函数返回值为float8的使用示例。
        st.execute(
            "create or replace function func_float() " + "return float8 " + "as declare " + "var1 float8; " + "begin
"
                + " select col1 into var1 from t_float limit 1; " + " return var1; " + "end;");
        CallableStatement cs = conn.prepareCall("{? = call func_float()}");
        cs.registerOutParameter(1, Types.DOUBLE);
        cs.execute();
        System.out.println(cs.getDouble(1));
        st.close();

        // 关闭数据库连接。
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

上述示例的运行结果为：

```
Connection succeed!
false
123456.123
```


5.4.3.4 批量查询

此示例主要使用setFetchSize调整客户端内存使用，原理是通过数据库游标来分批获取服务端数据，但会加大网络交互，可能会损失部分性能。由于游标事务内有效，故需要先关闭自动提交事务，最后执行手动提交事务。

```
// 以下用例以opengaussjdbc.jar为例。
// 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放（密码应密文存放，使用时解密），确保安全。
// 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量（环境变量名称请根据自身情况进行设置）EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
// $ip、$port、database需要用户自行修改。

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.PreparedStatement;

public class Batch {
    public static void main(String[] args) throws SQLException {
        String driver = "com.huawei.opengauss.jdbc.Driver";
        String username = System.getenv("EXAMPLE_USERNAME_ENV");
        String passwd = System.getenv("EXAMPLE_PASSWORD_ENV");
        String sourceURL = "jdbc:opengauss://$ip:$port/database";
        Connection conn = null;

        try {
            // 加载数据库驱动。
            Class.forName(driver);
        } catch (Exception e) {
            e.printStackTrace();
        }

        try {
            // 以非加密方式创建数据库连接。
            conn = DriverManager.getConnection(sourceURL, username, passwd);
            System.out.println("Connection succeed!");
        } catch (Exception e) {
            e.printStackTrace();
        }

        // 关闭自动提交。
        conn.setAutoCommit(false);

        // 建表。
        Statement st = conn.createStatement();
        st.execute("create table mytable (cal1 int);");

        // 表中插入200行数据。
        PreparedStatement pstm = conn.prepareStatement("insert into mytable values (?)");
        for (int i = 0; i < 200; i++) {
            pstm.setInt(1, i + 1);
            pstm.addBatch();
        }
        pstm.executeBatch();
        conn.commit();
        pstm.close();

        // 打开游标，每次获取50行数据。
        st.setFetchSize(50);
        int fetchCount = 0;
        ResultSet rs = st.executeQuery("SELECT * FROM mytable");
        while (rs.next()) {
            fetchCount++;
        }
        System.out.println(fetchCount == 200);
        conn.commit();
```



```
        rs.close();

        // 关闭服务器游标。
        st.setFetchSize(0);
        fetchCount = 0;
        rs = st.executeQuery("SELECT * FROM mytable");
        while (rs.next()) {
            fetchCount++;
        }
        System.out.println(fetchCount == 200);
        conn.commit();
        rs.close();

        // 关闭语句对象、关闭数据库连接。
        st.close();
        conn.close();

    }
}
```

上述示例的运行结果为：

```
Connection succeed!
true
true
```

执行完毕后可执行如下命令恢复自动提交事务。

```
conn.setAutoCommit(true);
```

5.4.3.5 应用层 SQL 重试

当主数据库节点故障且10s未恢复时，GaussDB会将对应的备数据库节点升主，使数据库正常运行。备升主期间正在运行的作业会失败，备升主后启动的作业不会再受影响。如果要做到数据库节点主备切换过程中，上层业务不感知，可参考此示例构建业务层SQL重试机制。

```
// 以下用例以opengaussjdbc.jar为例。
// 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放（密码应密文存放，使用时解密），确保安全。
// 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量（环境变量名称请根据自身情况进行设置）EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
// $ip、$port、database需要用户自行修改。

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

class ExitHandler extends Thread {
    private Statement cancel_stmt = null;

    public ExitHandler(Statement stmt) {
        super("Exit Handler");
        this.cancel_stmt = stmt;
    }
    public void run() {
        System.out.println("exit handle");
        try {
            this.cancel_stmt.cancel();
        } catch (SQLException e) {
            System.out.println("cancel query failed.");
            e.printStackTrace();
        }
    }
}
```

```
public class SQLRetry {
  //创建数据库连接。
  public static Connection GetConnection(String username, String passwd) {
    String driver = "com.huawei.opengauss.jdbc.Driver";
    String sourceURL = "jdbc:opengauss://$ip:$port/database";
    Connection conn = null;
    try {
      //加载数据库驱动。
      Class.forName(driver).newInstance();
    } catch (Exception e) {
      e.printStackTrace();
      return null;
    }

    try {
      //创建数据库连接。
      conn = DriverManager.getConnection(sourceURL, username, passwd);
      System.out.println("Connection succeed!");
    } catch (Exception e) {
      e.printStackTrace();
      return null;
    }

    return conn;
}

  //执行普通SQL语句，创建jdbc_test1表。
  public static void CreateTable(Connection conn) {
    Statement stmt = null;
    try {
      stmt = conn.createStatement();


      Runtime.getRuntime().addShutdownHook(new ExitHandler(stmt));

      //执行普通SQL语句。
      int rc2 = stmt
         .executeUpdate("DROP TABLE if exists jdbc_test1;");

      int rc1 = stmt
         .executeUpdate("CREATE TABLE jdbc_test1(col1 INTEGER, col2 VARCHAR(10));");

      stmt.close();
    } catch (SQLException e) {
      if (stmt != null) {
        try {
          stmt.close();
        } catch (SQLException e1) {
          e1.printStackTrace();
        }
      }
      e.printStackTrace();
    }
  }

  //执行预处理语句，批量插入数据。
  public static void BatchInsertData(Connection conn) {
    PreparedStatement pst = null;

    try {
      //生成预处理语句。
      pst = conn.prepareStatement("INSERT INTO jdbc_test1 VALUES (?,?)");
      for (int i = 0; i < 100; i++) {
       //添加参数。
        pst.setInt(1, i);
        pst.setString(2, "data " + i);
        pst.addBatch();
      }
```

```
    //执行批处理。
    pst.executeBatch();
    pst.close();
  } catch (SQLException e) {
    if (pst != null) {
      try {
        pst.close();
      } catch (SQLException e1) {
      e1.printStackTrace();
      }
    }
    e.printStackTrace();
  }
}

//执行预编译语句，更新数据。
private static boolean QueryRedo(Connection conn){
  PreparedStatement pstmt = null;
  boolean retValue = false;
  try {
    pstmt = conn
        .prepareStatement("SELECT col1 FROM jdbc_test1 WHERE col2 = ?");

        pstmt.setString(1, "data 10");
        ResultSet rs = pstmt.executeQuery();

        while (rs.next()) {
            System.out.println("col1 = " + rs.getString("col1"));
        }
        rs.close();

    pstmt.close();
     retValue = true;
   } catch (SQLException e) {
    System.out.println("catch..... retValue " + retValue);
    if (pstmt != null) {
      try {
       pstmt.close();
     } catch (SQLException e1) {
       e1.printStackTrace();
      }
    }
    e.printStackTrace();
  }

   System.out.println("finesh.....");
  return retValue;
}

//查询语句，执行失败重试，重试次数可配置。
public static void ExecPreparedSQL(Connection conn) throws InterruptedException {
      int maxRetryTime = 10;
      int time = 0;
      String result = null;
      do {
          time++;
          try {
System.out.println("time:" + time);
boolean ret = QueryRedo(conn);
if(ret == false){
 System.out.println("retry, time:" + time);
 Thread.sleep(10000);
 QueryRedo(conn);
}
          } catch (Exception e) {
              e.printStackTrace();
          }
      } while (null == result && time < maxRetryTime);
```



```
  }

  /**
   * 主程序，逐步调用各静态方法。
   * @param args
   * @throws InterruptedException
   */
  public static void main(String[] args) throws InterruptedException {
    //创建数据库连接。
    String userName = System.getenv("EXAMPLE_USERNAME_ENV");
    String password = System.getenv("EXAMPLE_PASSWORD_ENV");
    Connection conn = GetConnection(userName, password);

    //创建表。
    CreateTable(conn);

    //批量插入数据。
    BatchInsertData(conn);

    //执行预编译语句，更新数据。
    ExecPreparedSQL(conn);

    //关闭数据库连接。
    try {
      conn.close();
    } catch (SQLException e) {
      e.printStackTrace();
    }

  }

}
```

上述示例的运行结果为：

```
Connection succeed!
time:1
col1 = 10
finesh.....
time:2
col1 = 10
finesh.....
time:3
col1 = 10
finesh.....
time:4
col1 = 10
finesh.....
time:5
col1 = 10
finesh.....
time:6
col1 = 10
finesh.....
time:7
col1 = 10
finesh.....
time:8
col1 = 10
finesh.....
time:9
col1 = 10
finesh.....
time:10
col1 = 10
finesh.....
exit handle
```

5.4.3.6 通过本地文件导入导出数据

在使用JAVA语言基于GaussDB进行二次开发时，可以使用CopyManager接口，通过流方式，将数据库中的数据导出到本地文件或者将本地文件导入数据库中，文件支持CSV、TEXT等格式。

代码运行的前提条件：在数据库中创建表migration_table和migration_table_1，并在migration_table表中插入数据。

```
// 以下用例以opengaussjdbc.jar为例。
// 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放（密码应密文存放，使用时解密），确保安全。
// 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量（环境变量名称请根据自身情况进行设置）EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
// $ip、$port、database需要用户自行修改。

import java.sql.Connection;
import java.sql.DriverManager;
import java.io.IOException;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.sql.SQLException;
import com.huawei.opengauss.jdbc.copy.CopyManager;
import com.huawei.opengauss.jdbc.core.BaseConnection;

public class Copy{

    public static void main(String[] args)
    {
     String urls = new String("jdbc:opengauss://$ip:$port/database"); //数据库url
     String username = System.getenv("EXAMPLE_USERNAME_ENV");             //用户名
     String password = System.getenv("EXAMPLE_PASSWORD_ENV");             //密码
     String tablename = new String("migration_table"); //定义表信息
     String tablename1 = new String("migration_table_1"); //定义表信息
     String driver = "com.huawei.opengauss.jdbc.Driver";
     Connection conn = null;

     try {
          Class.forName(driver);
          conn = DriverManager.getConnection(urls, username, password); //以非加密方式连接数据库
        } catch (ClassNotFoundException e) {
            e.printStackTrace(System.out);
        } catch (SQLException e) {
            e.printStackTrace(System.out);
        }

     // 将SELECT * FROM migration_table查询结果导出到本地文件d:/data.txt
     try {
    copyToFile(conn, "d:/data.txt", "(SELECT * FROM migration_table)");
  } catch (SQLException e) {

 e.printStackTrace();
  } catch (IOException e) {

 e.printStackTrace();
  }
     //将d:/data.txt中的数据导入到migration_table_1中。
     try {
     copyFromFile(conn, "d:/data.txt", tablename1);
  } catch (SQLException e) {
        e.printStackTrace();
} catch (IOException e) {

 e.printStackTrace();
}

     // 将migration_table_1中的数据导出到本地文件d:/data1.txt
     try {
```



```
    copyToFile(conn, "d:/data1.txt", tablename1);
 } catch (SQLException e) {

e.printStackTrace();
 } catch (IOException e) {


e.printStackTrace();
}
   }
// 使用copyIn把数据从文件中导入数据库，
public static void copyFromFile(Connection connection, String filePath, String tableName)
      throws SQLException, IOException {

   FileInputStream fileInputStream = null;

   try {
      CopyManager copyManager = new CopyManager((BaseConnection)connection);
      fileInputStream = new FileInputStream(filePath);
      copyManager.copyIn("COPY " + tableName + " FROM STDIN", fileInputStream);
   } finally {
      if (fileInputStream != null) {
         try {
            fileInputStream.close();
         } catch (IOException e) {
            e.printStackTrace();
         }
      }
   }
}

// 使用copyOut把数据从数据库中导出到文件中
public static void copyToFile(Connection connection, String filePath, String tableOrQuery)
       throws SQLException, IOException {

    FileOutputStream fileOutputStream = null;

    try {
       CopyManager copyManager = new CopyManager((BaseConnection)connection);
       fileOutputStream = new FileOutputStream(filePath);
       copyManager.copyOut("COPY " + tableOrQuery + " TO STDOUT", fileOutputStream);
    } finally {
       if (fileOutputStream != null) {
          try {
             fileOutputStream.close();
          } catch (IOException e) {
             e.printStackTrace();
          }
       }
    }
 }
}
```

上述示例的运行结果为：本地d盘两个文件data.txt和data1.txt、数据库表migration_table_1，和数据库表migration_table数据相同。

5.4.3.7 从 MY 迁移数据

下面示例演示如何通过CopyManager从MY向GaussDB进行数据迁移。

代码运行的前提条件：MY和GaussDB数据库分别创建migration_table表。MY数据库的migration_table表预先插入数据。

```
// 以下用例以opengaussjdbc.jar为例。
// 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放（密码应密文存放，使用时解密），确保安全。
// 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量（环境变量名称请根据自身情况进行设置）EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
```



```
// $ip、$port、database需要用户自行修改。

import java.io.StringReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import com.huawei.opengauss.jdbc.copy.CopyManager;
import com.huawei.opengauss.jdbc.core.BaseConnection;

public class Migration{

    public static void main(String[] args) {
        String url = new String("jdbc:opengauss://$ip:$port/database"); //数据库url
        String user = System.getenv("EXAMPLE_USERNAME_ENV");            //GaussDB用户名
        String pass = System.getenv("EXAMPLE_PASSWORD_ENV");            //GaussDB密码
        String tablename = new String("migration_table"); //定义表信息
        String delimiter = new String("|");              //定义分隔符
        String encoding = new String("UTF8");            //定义字符集
        String driver = "com.huawei.opengauss.jdbc.Driver";
        StringBuffer buffer = new StringBuffer();       //定义存放格式化数据的缓存

        try {
            //获取源数据库查询结果集
            ResultSet rs = getDataSet();

            //遍历结果集，逐行获取记录
            //将每条记录中各字段值，按指定分隔符分割，由换行符结束，拼成一个字符串
            //把拼成的字符串，添加到缓存buffer
            while (rs.next()) {
                buffer.append(rs.getString(1) + delimiter
                        + rs.getString(2) + delimiter
                        + rs.getString(3) + delimiter
                        + rs.getString(4)
                        + "\n");
            }
            rs.close();

            try {
                //以非加密方式连接数据库
                Class.forName(driver);
                Connection conn = DriverManager.getConnection(url, user, pass);
                BaseConnection baseConn = (BaseConnection) conn;
                baseConn.setAutoCommit(false);

                //初始化表信息
                String sql = "Copy " + tablename + " from STDIN DELIMITER " + "'" + delimiter + "'" + "
ENCODING " + "'" + encoding + "'";

                //提交缓存buffer中的数据
                CopyManager cp = new CopyManager(baseConn);
                StringReader reader = new StringReader(buffer.toString());
                cp.copyIn(sql, reader);
                baseConn.commit();
                reader.close();
                baseConn.close();
            } catch (ClassNotFoundException e) {
                e.printStackTrace(System.out);
            } catch (SQLException e) {
                e.printStackTrace(System.out);
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    //********************************
```



```
    // 从源数据库返回查询结果集
    //**********************************
    private static ResultSet getDataSet() {
        ResultSet rs = null;
        try {
            Class.forName("com.MySQL.jdbc.Driver").newInstance();
            String userName = System.getenv("EXAMPLE_USERNAME_ENV");
            String password = System.getenv("EXAMPLE_PASSWORD_ENV");
            Connection conn = DriverManager.getConnection("jdbc:MySQL://$ip:$port/database?
useSSL=false&allowPublicKeyRetrieval=true", userName, password);
            Statement stmt = conn.createStatement();
            rs = stmt.executeQuery("select * from migration_table");
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return rs;
    }
}
```

上述示例的运行结果为：迁移后，GaussDB数据库的migration_table表和MY的migration_table表的数据一致。

5.4.3.8 逻辑复制

下面示例演示如何通过JDBC接口使用逻辑复制功能的过程。

配置选项

针对逻辑复制的配置选项，除了参考《特性指南》的“逻辑复制 > 逻辑解码”章节中的配置选项外，还有专门给JDBC等流式解码工具增加的配置项，如下所示：

1. 解码线程并行度

   通过配置选项parallel-decode-num，指定并行解码的Decoder线程数量。其取值范围为int型的1~20，取1表示按照原有的串行逻辑进行解码，取其余值即为开启并行解码。默认值为1。当该选项配置为1时，禁止配置以下选项：解码格式选项decode-style、批量发送选项sending-batch和并行解码队列长度选项parallel-queue-size。

2. 解码格式

   通过配置选项decode-style，指定解码格式。其取值为char型的字符'j'、't'或'b'，分别代表json格式、text格式及二进制格式。默认值为'b'，即二进制格式解码。该选项仅允许并行解码时设置，且二进制格式解码仅在并行解码场景下支持。对于json和text格式解码，在批量发送的解码结果中，每条解码语句的前4字节组成的uint32代表该条语句总字节数（不包含该uint32类型占用的4字节，0代表本批次解码结束），8字节uint64代表相应lsn（begin对应first_lsn，commit对应end_lsn，其他场景对应该条语句的lsn）。

说明

二进制格式编码规则如下所示：

1. 前4字节代表接下来到语句级别分隔符字母P（不含）或者该批次结束符F（不含）的解码结果的总字节数，该值如果为0代表本批次解码结束。
2. 接下来8字节uint64代表相应lsn（begin对应first_lsn，commit对应end_lsn，其他场景对应该条语句的lsn）。
3. 接下来1字节的字母有5种B/C/I/U/D，分别代表begin/commit/insert/update/delete。
4. 当第3步字母为B时：
    1. 接下来的8字节uint64代表CSN。
    2. 接下来的8字节uint64代表first_lsn。
    3. 【可选项】接下来的1字节字母如果为T，则代表后面4字节uint32表示该事务commit时间戳长度，再后面等同于该长度的字符为时间戳字符串。
    4. 【可选项】接下来的1字节字母如果为N，则代表后面4字节uint32表示该事务用户名的长度，再后面等同于该长度的字符为事务的用户名。
    5. 之后仍可能有解码语句，接下来会有1字节字母P或F作为语句间的分隔符，P代表本批次仍有解码的语句，F代表本批次解码完成。
5. 当第3步字母为C时：
    1. 【可选项】接下来1字节字母如果为X，则代表后面的8字节uint64表示xid。
    2. 【可选项】接下来1字节字母如果为T，则代表后面的4字节uint32表示时间戳长度，再后面等同于该长度的字符为时间戳字符串。
    3. 因为批量发送日志时，一个COMMIT日志解码之后可能仍有其他事务的解码结果，接下来的1字节字母如果为P则表示该批次仍需解码，如果为F则表示该批次解码结束。
6. 当第3步字母为I/U/D时：
    1. 接下来的2字节uint16代表schema名的长度。
    2. 按照上述长度读取schema名。
    3. 接下来的2字节uint16代表table名的长度。
    4. 按照上述长度读取table名。
    5. 【可选项】接下来1字节字母如果为N代表为新元组，如果为O代表为旧元组，这里先发送新元组。
        1. 接下来的2字节uint16代表该元组需要解码的列数，记为attrnum。
        2. 以下流程重复attrnum次。
            1. 接下来2字节uint16代表列名的长度。
            2. 按照上述长度读取列名。
            3. 接下来4字节uint32代表当前列类型的OID。
            4. 接下来4字节uint32代表当前列的值（以字符串格式存储）的长度，如果为0xFFFFFFFF则表示NULL，如果为0则表示长度为0的字符串。
            5. 按照上述长度读取列值。
    6. 因为之后仍可能有解码语句，接下来的1字节字母如果为P则表示该批次仍需解码，如果为F则表示该批次解码结束。

3. 限制仅备机解码

   通过配置选项standby-connection，指定是否限制仅备机解码。其取值为Boolean型（可用0或1表示），取true（或1）代表限制仅允许连接备机解码，连接主机解码时会报错退出。取false（或0）时不做限制。默认值为false（0）。

4. 批量发送

   通过配置选项sending-batch，指定是否批量发送。其取值范围为int型的0或1，取0表示逐条发送解码结果，取1表示解码结果累积到达1MB则批量发送解码结果。默认值为0。该选项仅允许并行解码时设置。开启批量发送的场景中，当解码格式为'j'或't'时，在原来的每条解码语句之前会附加一个uint32类型，表示本条解

码结果长度（长度不包含当前的uint32类型），以及一个uint64类型，表示当前解码结果对应的lsn。

5. 并行解码队列长度

   通过配置选项parallel-queue-size，指定并行逻辑解码线程间进行交互的队列长度。取值范围[2，1024]，且必须为2的幂数，默认值为128。队列长度和解码过程的内存使用量正相关。

6. 逻辑解码内存阈值

   逻辑解码内存阈值通过配置选项max-txn-in-memory指定单个事务解码中间结果缓存的内存阈值，单位为MB。并行解码模式下，该参数已废弃，不生效。串行解码模式下，取值范围为[0，100]，默认值为0，表示不管控内存使用。

   通过配置选项max-reorderbuffer-in-memory指定所有事务解码中间结果缓存的内存阈值，单位为GB。串行解码模式下，取值范围为[0，100]，默认值为0，表示不管控内存使用。并行解码模式下，取值范围为[1，max_process_memory总量的50%]，默认值为1与max_process_memory/1048576*10%的较大值，其中1048576为kB到GB的单位转换。当超过内存阈值时，解码过程将出现解码中间结果写临时文件的现象，影响逻辑解码的性能。临时文件的大小与事务修改的数据量成正比。如果临时文件过多，可能会导致磁盘进入只读状态的风险。

7. 逻辑解码发送超时阈值

   通过配置选项sender-timeout指定内核与客户端的心跳超时阈值。当该时间段内没有收到客户端任何消息，逻辑解码将主动停止，并断开和客户端的连接。单位为毫秒（ms），取值范围为[0, 2147483647]，默认值取决于GUC参数logical_sender_timeout配置。设置为0，表示逻辑解码不会主动断开和客户端的连接；如果设置过小，例如1ms，则可能存在解码任务中断的风险。

8. 逻辑解码用户黑名单选项

   使用逻辑解码用户黑名单，逻辑解码结果将过滤黑名单中用户的事务操作。当前相关选项如下：

   a. exclude-userids：指定黑名单用户的OID，多个OID通过逗号分隔，不校验用户OID是否存在。

   b. exclude-users：指定黑名单用户名称，多个名称通过逗号分隔，通过dynamic-resolution设置是否动态解析识别用户名称。若解码报错用户不存在而出现中断，在确定日志产生时刻不存在对应的黑名单用户，可以通过配置dynamic-resolution为true或者从用户黑名单中删除报错用户名称来启动解码继续获取逻辑日志。

   c. dynamic-resolution：是否动态解析黑名单用户名字，默认值为true。设置为false时，当解码检测到黑名单exclude-users中用户不存在时，将报错并退出逻辑解码。设置为true时，当解码检测到黑名单exclude-users中用户不存在时继续解码。

9. 事务逻辑日志输出选项

   a. include-xids：事务的BEGIN逻辑日志是否输出事务ID，默认值为true。

   b. include-timestamp：事务的BEGIN逻辑日志是否输出事务提交时间，默认值为false。

   c. include-user：事务的BEGIN逻辑日志是否输出事务的用户名字，默认值为false。事务的用户名字特指授权用户（执行事务对应会话的登录用户），它在事务的整个执行过程中不会发生变化。

10. JDBC默认设置逻辑解码连接的socketTimeout=10s，备机解码在主机压力大的时候可能会导致连接超时关闭，可以通过配置withStatusInterval(10000,TimeUnit.MILLISECONDS)，调整超时时间解决。

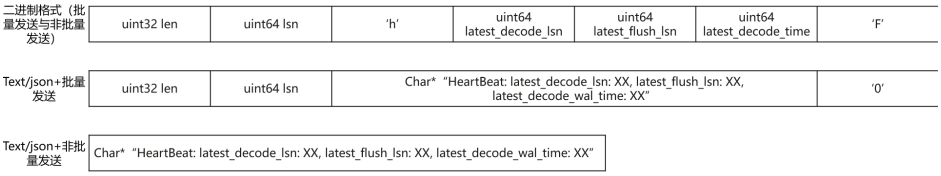
11. 心跳日志输出选项

    enable-heartbeat：是否输出心跳日志，默认值为false。

    说明

    以下对心跳日志进行解析，具体格式见下图：

    - 二进制格式首先是字符'h'，表示消息是心跳日志。
    - 之后是心跳日志内容。第一个8字节uint64代表LSN，表示发送心跳逻辑日志时读取的WAL日志结束位置；第二个8字节uint64代表LSN，表示发送心跳逻辑日志时刻已经落盘的WAL日志的位置；第三个8字节uint64代表时间戳（从1970年1月1日开始），表示最新解码到的事务日志或检查点日志的产生时间戳。
    - 关于消息结束符：如果是二进制格式，则为字符'F'。如果格式为text或者json且为批量发送则结束符为'0'，否则没有结束符。消息内容采用大端字节序进行数据传输。



在并行解码的标准场景下（16核CPU、内存128G、网络带宽 > 200MBps、表的列数为10~100、单行数据量0.1KB~1KB、DML操作以insert为主、不涉及落盘事务即单个事务中语句数量小于4096、parallel-decode-num为8、解码格式为'b'且开启批量发送功能），解码性能（以xlog消耗量为标准）不低于100MB/s。为保证解码性能达标以及尽量降低对业务的影响，一台备机上应尽量仅建立一个并行解码连接，保证CPU、内存、带宽资源充足。

示例

须知

逻辑复制类PGReplicationStream为非线程安全类，并发调用可能导致数据异常。

代码运行的前提条件：

1. 添加JDBC用户机器IP（假设IP为10.11.12.34）到复制数据权限的白名单里，命令如下：

   ```
   gs_guc reload -Z datanode -N all -I all -h 'host replication all 10.11.12.34/32 sha256'
   ```

2. 将wal_level参数设置为logical，设置方法请联系管理员处理。
3. 创建表t1和t2，并且对该表进行DDL或DML操作。

```
// 以下用例以opengaussjdbc.jar为例。
// 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放（密码应密文存放，使用时解密），确保安全。
// 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量（环境变量名称请根据自身情况进行设置）EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
// $ip、$port、database需要用户自行修改。


import com.huawei.opengauss.jdbc.PGProperty;
import com.huawei.opengauss.jdbc.jdbc.PgConnection;
import com.huawei.opengauss.jdbc.replication.LogSequenceNumber;
import com.huawei.opengauss.jdbc.replication.PGReplicationStream;

import java.nio.ByteBuffer;
import java.sql.DriverManager;
```

```
import java.util.Properties;
import java.util.concurrent.TimeUnit;

public class LogicalReplicationDemo {
    private static PgConnection conn = null;
    public static void main(String[] args) {
        String driver = "com.huawei.opengauss.jdbc.Driver";
        //此处配置数据库IP以及端口，这里的端口为haPort，通常默认是所连接DN的port+1端口
        String sourceURL = "jdbc:opengauss://$ip:$port/database";

        //默认逻辑复制槽的名称是：replication_slot
        //测试模式：创建逻辑复制槽
        int TEST_MODE_CREATE_SLOT = 1;
        //测试模式：开启逻辑复制（前提条件是逻辑复制槽已经存在）
        int TEST_MODE_START_REPL = 2;
        //测试模式：删除逻辑复制槽
        int TEST_MODE_DROP_SLOT = 3;
        //开启不同的测试模式
        int testMode = TEST_MODE_START_REPL;

        try {
            Class.forName(driver);
        } catch (Exception e) {
            e.printStackTrace();
            return;
        }

        try {
            Properties properties = new Properties();
            PGProperty.USER.set(properties, System.getenv("EXAMPLE_USERNAME_ENV"));
            PGProperty.PASSWORD.set(properties, System.getenv("EXAMPLE_PASSWORD_ENV"));
     //对于逻辑复制，以下三个属性是必须配置项
            PGProperty.ASSUME_MIN_SERVER_VERSION.set(properties, "9.4");
            PGProperty.REPLICATION.set(properties, "database");
            PGProperty.PREFER_QUERY_MODE.set(properties, "simple");
            conn = (PgConnection) DriverManager.getConnection(sourceURL, properties);
            System.out.println("connection success!");

            if(testMode == TEST_MODE_CREATE_SLOT){
                conn.getReplicationAPI()
                        .createReplicationSlot()
                        .logical()
                        .withSlotName("replication_slot") //这里字符串如包含大写字母则会自动转化为小写字母
                        .withOutputPlugin("mppdb_decoding")
                        .make();
            }else if(testMode == TEST_MODE_START_REPL) {
                //开启此模式前需要创建复制槽
                LogSequenceNumber waitLSN = LogSequenceNumber.valueOf("6F/E3C53568"); //LSN需要用户根
据实际情况进行修改
                PGReplicationStream stream = conn
                        .getReplicationAPI()
                        .replicationStream()
                        .logical()
                        .withSlotName("replication_slot")
                        .withSlotOption("include-xids", true)
                        .withSlotOption("skip-empty-xacts", true)
                        .withStartPosition(waitLSN)
                        .withSlotOption("parallel-decode-num", 10) //解码线程并行度
                        .withSlotOption("white-table-list", "public.t1,public.t2") //白名单列表
                        // .withSlotOption("standby-connection", true) //强制备机解码
                        .withSlotOption("decode-style", "t") //解码格式
                        .withSlotOption("sending-batch", 0) //批量发送解码结果
                        .withSlotOption("max-txn-in-memory", 100) //单个解码事务落盘内存阈值为100MB
                        .withSlotOption("max-reorderbuffer-in-memory", 2) //正在处理的解码事务落盘内存阈值为
2GB
                        .withSlotOption("exclude-users", "userA") //不返回用户userA执行事务的逻辑日志
                        .withSlotOption("include-user", false) //事务BEGIN逻辑日志不携带用户名
                        .withSlotOption("enable-heartbeat", true) // 开启心跳日志
                        .start();
```



```
                while (true) {
                    ByteBuffer byteBuffer = stream.readPending();

                    if (byteBuffer == null) {
                        TimeUnit.MILLISECONDS.sleep(10L);
                        continue;
                    }

                    int offset = byteBuffer.arrayOffset();
                    byte[] source = byteBuffer.array();
                    int length = source.length - offset;
                    System.out.println(new String(source, offset, length));

                    //如果需要flush lsn，根据业务实际情况调用以下接口，该接口会触发数据库复制槽落盘，对服务端解码性能有一定影响，建议调用间隔大于10s。
                    //LogSequenceNumber lastRecv = stream.getLastReceiveLSN();
                    //stream.setFlushedLSN(lastRecv);
                    //stream.forceUpdateStatus();

                }
            }else if(testMode == TEST_MODE_DROP_SLOT){
                conn.getReplicationAPI()
                        .dropReplicationSlot("replication_slot");
            }
        } catch (Exception e) {
            e.printStackTrace();
            return;
        } finally {
            try {
                conn.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

text格式（即't'格式）解码结果示例如下：

```
BEGIN CSN: 2014 first_lsn: 0/2816A28
table public t1 INSERT: a[integer]:1 b[integer]:2 c[text]:'hello'
COMMIT XID: 15504
BEGIN CSN: 2015 first_lsn: 0/2816C20
table public t1 UPDATE: old-key: a[integer]:1 b[integer]:2 c[text]:'hello' new-tuple: a[integer]:1 b[integer]:5 c[text]:'hello'
COMMIT XID: 15505
BEGIN CSN: 2016 first_lsn: 0/2816D60
table public t1 DELETE: a[integer]:1 b[integer]:5 c[text]:'hello'
COMMIT XID: 15506
```

json格式（即'j'格式）解码结果示例如下：

```
BEGIN CSN: 2014 first_lsn: 0/2816A28
{"table_name":"public.t1","op_type":"INSERT","columns_name":["a","b","c"],"columns_type":["integer","integer","text"],"columns_val":["1","2","'hello'"],"old_keys_name":[],"old_keys_type":[],"old_keys_val":[]}
COMMIT XID: 15504
BEGIN CSN: 2015 first_lsn: 0/2816C20
{"table_name":"public.t1","op_type":"UPDATE","columns_name":["a","b","c"],"columns_type":["integer","integer","text"],"columns_val":["1","5","'hello'"],"old_keys_name":["a","b","c"],"old_keys_type":["integer","integer","text"],"old_keys_val":["1","2","'hello'"]}
COMMIT XID: 15505
BEGIN CSN: 2016 first_lsn: 0/2816D60
{"table_name":"public.t1","op_type":"DELETE","columns_name":[],"columns_type":[],"columns_val":[],"old_keys_name":["a","b","c"],"old_keys_type":["integer","integer","text"],"old_keys_val":["1","5","'hello'"]}
COMMIT XID: 15506
```

5.4.4 JDBC 接口参考

JDBC接口是一套提供给用户的API方法，本节将对部分常用接口做具体描述，若涉及其他接口可参考JDK1.8（软件包）/JDBC 4.2中相关内容。

5.4.4.1 java.sql.Connection

java.sql.Connection是数据库连接接口。

表 5-13 对 java.sql.Connection 接口的支持情况

方法名	描述	返回值类型	throws	支持JDBC 4
abort(Executor executor)	终止打开的连接。	void	SQLException	Yes
clearWarnings()	清除为此Connection对象报告的所有警告。	void	SQLException	Yes
close()	立即释放此Connection对象的数据库和JDBC资源，而不是等待它们自动释放。	void	SQLException	Yes
commit()	使自上次提交/回滚以来所做的所有更改永久化，并释放此Connection对象当前持有的任何数据库锁。仅当禁用了自动提交模式时，才应使用此方法。	void	SQLException	Yes
createArrayOf(String typeName, Object[] elements)	用于创建Array对象的工厂方法。	Array	SQLException	Yes
createBlob()	构造一个实现Blob接口的对象。返回的对象最初不包含数据。Blob接口的setBinaryStream和setBytes方法可以用于向Blob添加数据。	Blob	SQLException	Yes

方法名	描述	返回值类型	throws	支持JDBC 4
createClob()	构造一个实现Clob接口的对象。返回的对象最初不包含数据。Clob接口的setAsciiStream、setCharacterStream和setString方法可以用于向Clob添加数据。	Clob	SQLException	Yes
createSQLXML()	构造一个实现SQLXML接口的对象。返回的对象最初不包含数据。可以使用SQLXML接口的createXmlStreamWriter对象和setString方法向SQLXML对象添加数据。	SQLXML	SQLException	Yes
createStatement()	创建一个用于将SQL语句发送到数据库的语句对象。	Statement	SQLException	Yes
createStatement(int resultSetType, int resultSetConcurrency)	创建一个语句对象，该对象将生成具有给定类型和并发性的ResultSet对象。	Statement	SQLException	Yes
createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)	创建一个语句对象，该对象将生成具有给定类型和并发性的ResultSet对象。	Statement	SQLException	Yes

方法名	描述	返回值类型	throws	支持JDBC 4
getAutoCommit()	检索此连接对象的当前自动提交模式。	Boolean	SQLException	Yes
getCatalog()	检索此Connection对象的当前目录名称。	String	SQLException	Yes
getClientInfo()	返回一个列表，其中包含驱动程序支持的每个客户端信息属性的名称和当前值。如果客户端信息属性尚未设置且没有默认值，则该属性的值可能为null。	Properties	SQLException	Yes
getClientInfo(String name)	返回由name指定的客户端信息属性的值。如果尚未设置指定的客户端信息属性，并且没有默认值，此方法可能会返回null。如果驱动程序不支持指定的客户端信息属性名称，此方法也将返回null。	String	SQLException	Yes
getHoldability()	检索使用此Connection对象创建的ResultSet对象的当前保持性。	int	SQLException	Yes

方法名	描述	返回值类型	throws	支持JDBC 4
getMetaData()	检索包含有关此Connection对象表示连接的数据库的元数据的DatabaseMetaData对象。元数据包括有关数据库表、其支持的SQL语法、其存储过程、此连接的功能等的信息。	DatabaseMetaData	SQLException	Yes
getNetworkTimeout()	检索驱动程序等待数据库请求完成的毫秒数。如果超过限制，将引发SQLException。	int	SQLException	Yes
getSchema()	检索此Connection对象的当前架构名称。	String	SQLException	Yes
getTransactionIsolation()	检索此Connection对象的当前事务隔离级别。	int	SQLException	Yes
getTypeMap()	检索与此Connection对象关联的Map对象。除非应用程序添加了条目，否则返回的类型映射将为空。	Map<String,Class<?>>	SQLException	Yes

方法名	描述	返回值类型	throws	支持JDBC 4
getWarnings()	检索此Connection对象上的调用报告的第一个警告。如果有多个警告，则后续警告将链接到第一个警告，并可以通过对先前检索的警告调用方法SQLWarning.getNextWarning来检索。	SQLWarning	SQLException	Yes
isClosed()	检索此Connection对象是否已关闭。如果已对连接调用了方法关闭，或者发生了某些致命错误，则连接将关闭。只有在调用closection.Close方法之后调用此方法时，才保证返回true。	Boolean	SQLException	Yes
isReadOnly()	检索此Connection对象是否处于只读模式。	Boolean	SQLException	Yes
isValid(int timeout)	如果连接尚未关闭并且仍然有效，则返回true。驱动程序应提交对连接的查询，或使用其他机制，以肯定地验证连接在调用此方法时仍然有效。	boolean	SQLException	Yes

方法名	描述	返回值类型	throws	支持JDBC 4
nativeSQL(String sql)	将给定的SQL语句转换为系统的本机SQL语法。驱动程序可以在发送JDBC SQL语法之前将其转换为其系统的本机SQL语法。此方法返回驱动程序本应发送的语句的本机形式。	String	SQLException	Yes
prepareCall(String sql)	创建用于调用数据库存储过程的CallableStatement对象。CallableStatement对象提供了用于设置其IN和OUT参数的方法，以及用于执行对存储过程的调用的方法。	CallableStatement	SQLException	Yes
prepareCall(String sql, int resultSetType, int resultSetConcurrency)	创建一个CallableStatement对象，该对象将生成具有给定类型和并发性的ResultSet对象。此方法与上面的准备呼叫方法相同，但它允许覆盖默认的结果集类型和并发。	CallableStatement	SQLException	Yes

方法名	描述	返回值类型	throws	支持JDBC 4
prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	创建一个CallableStatement对象，该对象将生成具有给定类型和并发性的ResultSet对象。此方法与上面的准备呼叫方法相同，但它允许覆盖默认的结果集类型、结果集并发类型和保持性。	CallableStatement	SQLException	Yes
prepareStatement(String sql)	创建一个用于将参数化SQL语句发送到数据库的准备语句对象。	PreparedStatement	SQLException	Yes
prepareStatement(String sql, int autoGeneratedKeys)	创建一个默认的准备语句对象，该对象能够检索自动生成的键。给定的常量告诉驱动程序，它是否应使自动生成的密钥可供检索。如果SQL语句不是INSERT语句，也不是能够返回自动生成键的SQL语句，则将忽略此参数。	PreparedStatement	SQLException	Yes

方法名	描述	返回值类型	throws	支持JDBC 4
prepareStatement(String sql, int[] columnIndexes)	创建一个默认的准备语句对象，该对象能够返回由给定数组指定的自动生成的键。此数组包含目标表中包含应可用的自动生成键的列的索引。如果SQL语句不是INSERT语句，也不是能够返回自动生成键的SQL语句，驱动程序将忽略数组。	PreparedStatement	SQLException	Yes
prepareStatement(String sql, int resultSetType, int resultSetConcurrency)	创建一个准备语句对象，该对象将生成具有给定类型和并发性的ResultSet对象。此方法与上面的准备语句方法相同，但它允许覆盖默认的结果集类型和并发。	PreparedStatement	SQLException	Yes
prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	创建一个准备语句对象，该对象将生成具有给定类型、并发性和保持性的ResultSet对象。	PreparedStatement	SQLException	Yes

方法名	描述	返回值类型	throws	支持JDBC 4
prepareStatement(String sql, String[] columnNames)	创建一个默认的准备语句对象，该对象能够返回由给定数组指定的自动生成的键。此数组包含目标表中包含应返回的自动生成键的列的名称。如果SQL语句不是INSERT语句，也不是能够返回自动生成键的SQL语句，驱动程序将忽略数组。	PreparedStatement	SQLException	Yes
releaseSavepoint(Savepoint savepoint)	从当前事务中删除指定的Savepoint和后续Savepoint对象。删除保存点后对保存点的任何引用都将导致引发SQLException。	void	SQLException	Yes
rollback()	撤销在当前事务中所做的所有更改，并释放此Connection对象当前持有的任何数据库锁。仅当禁用自动提交模式时，才应使用此方法。	void	SQLException	Yes
rollback(Savepoint savepoint)	撤销设置给定Savepoint对象后所做的所有更改。仅当禁用自动提交时，才应使用此方法。	void	SQLException	Yes

方法名	描述	返回值类型	throws	支持JDBC 4
setAutoCommit(boolean autoCommit)	将此连接的自动提交模式设置为给定状态。如果连接处于自动提交模式，则其所有SQL语句都将作为单个事务执行和提交。否则，其SQL语句将分组为事务，这些事务由对方法提交或方法回滚的调用终止。默认情况下，新连接处于自动提交模式。	void	SQLException	Yes
setClientInfo(Properties properties)	设置连接的客户端信息属性的值。Properties对象包含要设置的客户端信息属性的名称和值。属性列表中包含的客户端信息属性集替换连接上的当前客户端信息属性集。如果当前在连接上设置的属性不在属性列表中，则将清除该属性。	void	SQLClientInfoException	Yes
setClientInfo(String name,String value)	将name指定的客户端info属性的值设置为value指定的值。	void	SQLClientInfoException	Yes

方法名	描述	返回值类型	throws	支持JDBC 4
setHoldability(int holdability)	将使用此Connection对象创建的ResultSet对象的默认保持性更改为给定的保持性。ResultSet对象的默认保持性可以通过调用DatabaseMetaData.getResultSetHoldability来确定。	void	SQLException	Yes
setNetworkTimeout(Executor executor, int milliseconds)	设置连接或从连接创建的对象等待数据库回复任何一个请求的最长时间。如果任何请求仍未应答，则等待方法将返回SQLException，并且从Connection创建的Connection或对象将标记为已关闭。除关闭、isCabled或Connection.isValid方法外，对对象的任何后续使用都将导致SQLException。	void	SQLException	Yes
setReadOnly(boolean readOnly)	将此连接置于只读模式，作为驱动程序的提示，以启用数据库优化。	void	SQLException	Yes

方法名	描述	返回值类型	throws	支持JDBC 4
setSavepoint()	在当前事务中创建一个未命名的保存点，并返回表示它的新保存点对象。	Savepoint	SQLException	Yes
setSavepoint(String name)	在当前事务中使用给定名称创建保存点，并返回表示它的新保存点对象。	Savepoint	SQLException	Yes
setSchema(String schema)	将给定的架构名称设置为访问。	void	SQLException	Yes
setTransactionIsolation(int level)	尝试将此Connection对象的事务隔离级别更改为给定的级别。接口Connection中定义的常量是可能的事务隔离级别。	void	SQLException	Yes
setTypeMap(Map<String,Class<?>> map)	安装给定的TypeMap对象作为此Connection对象的类型映射。类型映射将用于SQL结构化类型和不同类型的自定义映射。	void	SQLException	Yes

须知

1. 接口内部默认使用自动提交模式，若通过setAutoCommit(false)关闭自动提交模式，将会导致后面执行的语句受到显式事务包裹，数据库中不支持事务中执行的语句不能在此模式下执行。
2. 不支持全密态数据库使用setClientInfo("send_token", null)传输密钥，使用setClientInfo("clear_token", null)销毁密钥。
3. 创建PrepareStatement类时，在SQL语句中使用returning语句，returning子句不生效。

5.4.4.2 java.sql.CallableStatement

java.sql.CallableStatement是存储过程执行接口。

表 5-14 对 java.sql.CallableStatement 的支持情况

方法名	返回值类型	支持JDBC 4
getArray(int parameterIndex)	Array	Yes
getBigDecimal(int parameterIndex)	BigDecimal	Yes
getBlob(int parameterIndex)	Blob	Yes
getBoolean(int parameterIndex)	Boolean	Yes
getByte(int parameterIndex)	byte	Yes
getBytes(int parameterIndex)	byte[]	Yes
getClob(int parameterIndex)	Clob	Yes
getDate(int parameterIndex)	Date	Yes
getDate(int parameterIndex, Calendar cal)	Date	Yes
getDouble(int parameterIndex)	double	Yes
getFloat(int parameterIndex)	float	Yes
getInt(int parameterIndex)	int	Yes
getLong(int parameterIndex)	long	Yes
getObject(int parameterIndex)	Object	Yes
getObject(int parameterIndex, Class<T> type)	Object	Yes
getShort(int parameterIndex)	short	Yes
getSQLXML(int parameterIndex)	SQLXML	Yes
getString(int parameterIndex)	String	Yes
getNString(int parameterIndex)	String	Yes
getTime(int parameterIndex)	Time	Yes

方法名	返回值类型	支持JDBC 4
getTime(int parameterIndex, Calendar cal)	Time	Yes
getTimestamp(int parameterIndex)	Timestamp	Yes
getTimestamp(int parameterIndex, Calendar cal)	Timestamp	Yes
registerOutParameter(int parameterIndex, int type)	void	Yes
registerOutParameter(int parameterIndex, int sqlType, int type)	void	Yes
wasNull()	Boolean	Yes

说明

- 不允许含有OUT参数的语句执行批量操作。
- 以下方法是从java.sql.Statement继承而来：close、execute、executeQuery、executeUpdate、getConnection、getResultSet、getUpdateCount、isClosed、setMaxRows、setFetchSize。
- 以下方法是从java.sql.PreparedStatement继承而来：addBatch、clearParameters、execute、executeQuery、executeUpdate、getMetaData、setBigDecimal、setBoolean、setByte、setBytes、setDate、setDouble、setFloat、setInt、setLong、setNull、setObject、setString、setTime、setTimestamp。
- registerOutParameter(int parameterIndex, int sqlType, int type)方法仅用于注册复合数据类型，其他类型不支持。

5.4.4.3 java.sql.DatabaseMetaData

java.sql.DatabaseMetaData是数据库对象定义接口。

表 5-15 对 java.sql.DatabaseMetaData 的支持情况

方法名	返回值类型	支持JDBC 4
allProceduresAreCallable()	boolean	Yes
allTablesAreSelectable()	boolean	Yes
autoCommitFailureClosesAllResultSets()	boolean	Yes
dataDefinitionCausesTransactionCommit()	boolean	Yes

方法名	返回值类型	支持JDBC 4
dataDefinitionIgnoredInTransactions()	boolean	Yes
deletesAreDetected(int type)	boolean	Yes
doesMaxRowSizeIncludeBlobs()	boolean	Yes
generatedKeyAlwaysReturned()	boolean	Yes
getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)	ResultSet	Yes
getCatalogs()	ResultSet	Yes
getCatalogSeparator()	String	Yes
getCatalogTerm()	String	Yes
getClientInfoProperties()	ResultSet	Yes
getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)	ResultSet	Yes
getConnection()	Connection	Yes
getCrossReference(String parentCatalog, String parentSchema, String parentTable, String foreignCatalog, String foreignSchema, String foreignTable)	ResultSet	Yes
getDefaultTransactionIsolation()	int	Yes
getExportedKeys(String catalog, String schema, String table)	ResultSet	Yes
getExtraNameCharacters()	String	Yes
getFunctionColumns(String catalog, String schemaPattern, String functionNamePattern, String columnNamePattern)	ResultSet	Yes

方法名	返回值类型	支持JDBC 4
getFunctions(String catalog, String schemaPattern, String functionNamePattern)	ResultSet	Yes
getIdentifierQuoteString()	String	Yes
getImportedKeys(String catalog, String schema, String table)	ResultSet	Yes
getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)	ResultSet	Yes
getMaxBinaryLiteralLength()	int	Yes
getMaxCatalogNameLength()	int	Yes
getMaxCharLiteralLength()	int	Yes
getMaxColumnNameLength()	int	Yes
getMaxColumnsInGroupBy()	int	Yes
getMaxColumnsInIndex()	int	Yes
getMaxColumnsInOrderBy()	int	Yes
getMaxColumnsInSelect()	int	Yes
getMaxColumnsInTable()	int	Yes
getMaxConnections()	int	Yes
getMaxCursorNameLength()	int	Yes
getMaxIndexLength()	int	Yes
getMaxLogicalLobSize()	default long	Yes
getMaxProcedureNameLength()	int	Yes
getMaxRowSize()	int	Yes
getMaxSchemaNameLength()	int	Yes
getMaxStatementLength()	int	Yes
getMaxStatements()	int	Yes
getMaxTableNameLength()	int	Yes

方法名	返回值类型	支持JDBC 4
getMaxTablesInSelect()	int	Yes
getMaxUserNameLength()	int	Yes
getNumericFunctions()	String	Yes
getPrimaryKeys(String catalog, String schema, String table)	ResultSet	Yes
getPartitionTablePrimaryKeys(String catalog, String schema, String table)	ResultSet	Yes
getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)	ResultSet	Yes
getProcedures(String catalog, String schemaPattern, String procedureNamePattern)	ResultSet	Yes
getProcedureTerm()	String	Yes
getSchemas()	ResultSet	Yes
getSchemas(String catalog, String schemaPattern)	ResultSet	Yes
getSchemaTerm()	String	Yes
getSearchStringEscape()	String	Yes
getSQLKeywords()	String	Yes
getSQLStateType()	int	Yes
getStringFunctions()	String	Yes
getSystemFunctions()	String	Yes
getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)	ResultSet	Yes
getTimeDateFunctions()	String	Yes
getTypeInfo()	ResultSet	Yes

方法名	返回值类型	支持JDBC 4
getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)	ResultSet	Yes
getURL()	String	Yes
getVersionColumns(String catalog, String schema, String table)	ResultSet	Yes
insertsAreDetected(int type)	boolean	Yes
locatorsUpdateCopy()	boolean	Yes
othersDeletesAreVisible(int type)	boolean	Yes
othersInsertsAreVisible(int type)	boolean	Yes
othersUpdatesAreVisible(int type)	boolean	Yes
ownDeletesAreVisible(int type)	boolean	Yes
ownInsertsAreVisible(int type)	boolean	Yes
ownUpdatesAreVisible(int type)	boolean	Yes
storesLowerCaseIdentifiers()	boolean	Yes
storesMixedCaseIdentifiers()	boolean	Yes
storesUpperCaseIdentifiers()	boolean	Yes
supportsBatchUpdates()	boolean	Yes
supportsCatalogsInDataManipulation()	boolean	Yes
supportsCatalogsInIndexDefinitions()	boolean	Yes
supportsCatalogsInPrivilegeDefinitions()	boolean	Yes
supportsCatalogsInProcedureCalls()	boolean	Yes
supportsCatalogsInTableDefinitions()	boolean	Yes

方法名	返回值类型	支持JDBC 4
supportsCorrelatedSubqueries()	boolean	Yes
supportsDataDefinitionAndDataManipulationTransactions()	boolean	Yes
supportsDataManipulationTransactionsOnly()	boolean	Yes
supportsGetGeneratedKeys()	boolean	Yes
supportsMixedCaseIdentifiers()	boolean	Yes
supportsMultipleOpenResults()	boolean	Yes
supportsNamedParameters()	boolean	Yes
supportsOpenCursorsAcrossCommit()	boolean	Yes
supportsOpenCursorsAcrossRollback()	boolean	Yes
supportsOpenStatementsAcrossCommit()	boolean	Yes
supportsOpenStatementsAcrossRollback()	boolean	Yes
supportsPositionedDelete()	boolean	Yes
supportsPositionedUpdate()	boolean	Yes
supportsRefCursors()	boolean	Yes
supportsResultSetConcurrency(int type, int concurrency)	boolean	Yes
supportsResultSetType(int type)	boolean	Yes
supportsSchemasInIndexDefinitions()	boolean	Yes
supportsSchemasInPrivilegeDefinitions()	boolean	Yes
supportsSchemasInProcedureCalls()	boolean	Yes
supportsSchemasInTableDefinitions()	boolean	Yes
supportsSelectForUpdate()	boolean	Yes

方法名	返回值类型	支持JDBC 4
supportsStatementPooling()	boolean	Yes
supportsStoredFunctionsUsingCallSyntax()	boolean	Yes
supportsStoredProcedures()	boolean	Yes
supportsSubqueriesInComparisons()	boolean	Yes
supportsSubqueriesInExists()	boolean	Yes
supportsSubqueriesInIns()	boolean	Yes
supportsSubqueriesInQuantifieds()	boolean	Yes
supportsTransactionIsolationLevel(int level)	boolean	Yes
supportsTransactions()	boolean	Yes
supportsUnion()	boolean	Yes
supportsUnionAll()	boolean	Yes
updatesAreDetected(int type)	boolean	Yes
getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)	ResultSet	Yes
getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)	ResultSet	Yes
getTableTypes()	ResultSet	Yes
getUserName()	String	Yes
isReadOnly()	boolean	Yes
nullsAreSortedHigh()	boolean	Yes
nullsAreSortedLow()	boolean	Yes
nullsAreSortedAtStart()	boolean	Yes
nullsAreSortedAtEnd()	boolean	Yes
getDatabaseProductName()	String	Yes
getDatabaseProductVersion()	String	Yes

方法名	返回值类型	支持JDBC 4
getDriverName()	String	Yes
getDriverVersion()	String	Yes
getDriverMajorVersion()	int	Yes
getDriverMinorVersion()	int	Yes
usesLocalFiles()	boolean	Yes
usesLocalFilePerTable()	boolean	Yes
supportsMixedCaseIdentifiers()	boolean	Yes
storesUpperCaseIdentifiers()	boolean	Yes
storesLowerCaseIdentifiers()	boolean	Yes
supportsMixedCaseQuotedIdentifiers()	boolean	Yes
storesUpperCaseQuotedIdentifiers()	boolean	Yes
storesLowerCaseQuotedIdentifiers()	boolean	Yes
storesMixedCaseQuotedIdentifiers()	boolean	Yes
supportsAlterTableWithAddColumn()	boolean	Yes
supportsAlterTableWithDropColumn()	boolean	Yes
supportsColumnAliasing()	boolean	Yes
nullPlusNonNullIsNull()	boolean	Yes
supportsConvert()	boolean	Yes
supportsConvert(int fromType, int toType)	boolean	Yes
supportsTableCorrelationNames()	boolean	Yes
supportsDifferentTableCorrelationNames()	boolean	Yes
supportsExpressionsInOrderBy()	boolean	Yes
supportsOrderByUnrelated()	boolean	Yes

方法名	返回值类型	支持JDBC 4
supportsGroupBy()	boolean	Yes
supportsGroupByUnrelated()	boolean	Yes
supportsGroupByBeyondSelect()	boolean	Yes
supportsLikeEscapeClause()	boolean	Yes
supportsMultipleResultSets()	boolean	Yes
supportsMultipleTransactions()	boolean	Yes
supportsNonNullableColumns()	boolean	Yes
supportsMinimumSQLGrammar()	boolean	Yes
supportsCoreSQLGrammar()	boolean	Yes
supportsExtendedSQLGrammar()	boolean	Yes
supportsANSI92EntryLevelSQL()	boolean	Yes
supportsANSI92IntermediateSQL()	boolean	Yes
supportsANSI92FullSQL()	boolean	Yes
supportsIntegrityEnhancementFacility()	boolean	Yes
supportsOuterJoins()	boolean	Yes
supportsFullOuterJoins()	boolean	Yes
supportsLimitedOuterJoins()	boolean	Yes
isCatalogAtStart()	boolean	Yes
supportsSchemasInDataManipulation()	boolean	Yes
supportsSavepoints()	boolean	Yes
supportsResultSetHoldability(int holdability)	boolean	Yes
getResultSetHoldability()	int	Yes
getDatabaseMajorVersion()	int	Yes
getDatabaseMinorVersion()	int	Yes

方法名	返回值类型	支持JDBC 4
getJDBCMajorVersion()	int	Yes
getJDBCMinorVersion()	int	Yes

说明

uppercaseAttributeName为true时，以下接口会将查询结果转为大写，可转换范围与java中的toUpperCase方法一致。

- public ResultSet getProcedures(String catalog, String schemaPattern, String procedureNamePattern)
- public ResultSet getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)
- public ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)
- public ResultSet getSchemas(String catalog, String schemaPattern)
- public ResultSet getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)
- public ResultSet getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)
- public ResultSet getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)
- public ResultSet getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)
- public ResultSet getPrimaryKeys(String catalog, String schema, String table)
- protected ResultSet getImportedExportedKeys(String primaryCatalog, String primarySchema, String primaryTable, String foreignCatalog, String foreignSchema, String foreignTable)
- public ResultSet getIndexInfo(String catalog, String schema, String tableName, boolean unique, boolean approximate)
- public ResultSet getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)
- public ResultSet getFunctions(String catalog, String schemaPattern, String functionNamePattern)

注意

getPartitionTablePrimaryKeys(String catalog, String schema, String table)接口用于获取分区表含全局索引的主键列，使用示例如下：

```
PgDatabaseMetaData dbmd = (PgDatabaseMetaData)conn.getMetaData();
dbmd.getPartitionTablePrimaryKeys("catalogName", "schemaName", "tableName");
```

5.4.4.4 java.sql.Driver

java.sql.Driver是数据库驱动接口。

表 5-16 对 java.sql.Driver 的支持情况

方法名	返回值类型	支持JDBC 4
acceptsURL(String url)	Boolean	Yes
connect(String url, Properties info)	Connection	Yes
jdbcCompliant()	Boolean	Yes
getMajorVersion()	int	Yes
getMinorVersion()	int	Yes
getParentLogger()	Logger	Yes
getPropertyInfo(String url, Properties info)	DriverPropertyInfo[]	Yes

5.4.4.5 java.sql.PreparedStatement

java.sql.PreparedStatement是预处理语句接口。

表 5-17 对 java.sql.PreparedStatement 的支持情况

方法名	返回值类型	支持JDBC 4
clearParameters()	void	Yes
execute()	Boolean	Yes
executeQuery()	ResultSet	Yes
excuteUpdate()	int	Yes
executeLargeUpdate()	long	No
getMetaData()	ResultSetMetaData	Yes
getParameterMetaData()	ParameterMetaData	Yes
setArray(int parameterIndex, Array x)	void	Yes
setAsciiStream(int parameterIndex, InputStream x, int length)	void	Yes
setBinaryStream(int parameterIndex, InputStream x)	void	Yes

方法名	返回值类型	支持JDBC 4
setBinaryStream(int parameterIndex, InputStream x, int length)	void	Yes
setBinaryStream(int parameterIndex, InputStream x, long length)	void	Yes
setBlob(int parameterIndex, InputStream inputStream)	void	Yes
setBlob(int parameterIndex, InputStream inputStream, long length)	void	Yes
setBlob(int parameterIndex, Blob x)	void	Yes
setCharacterStream(int parameterIndex, Reader reader)	void	Yes
setCharacterStream(int parameterIndex, Reader reader, int length)	void	Yes
setClob(int parameterIndex, Reader reader)	void	Yes
setClob(int parameterIndex, Reader reader, long length)	void	Yes
setClob(int parameterIndex, Clob x)	void	Yes
setDate(int parameterIndex, Date x, Calendar cal)	void	Yes
setNull(int parameterIndex, int sqlType)	void	Yes

方法名	返回值类型	支持JDBC 4
setNull(int parameterIndex, int sqlType, String typeName)	void	Yes
setObject(int parameterIndex, Object x)	void	Yes
setObject(int parameterIndex, Object x, int targetSqlType)	void	Yes
setObject(int parameterIndex, Object x, int targetSqlType, int scaleOrLength)	void	Yes
setSQLXML(int parameterIndex, SQLXML xmlObject)	void	Yes
setTime(int parameterIndex, Time x)	void	Yes
setTime(int parameterIndex, Time x, Calendar cal)	void	Yes
setTimestamp(int parameterIndex, Timestamp x)	void	Yes
setTimestamp(int parameterIndex, Timestamp x, Calendar cal)	void	Yes
setUnicodeStream(int parameterIndex, InputStream x, int length)	void	Yes
setURL(int parameterIndex, URL x)	void	Yes
setBoolean(int parameterIndex, boolean x)	void	Yes
setBigDecimal(int parameterIndex, BigDecimal x)	void	Yes

方法名	返回值类型	支持JDBC 4
setByte(int parameterIndex, byte x)	void	Yes
setBytes(int parameterIndex, byte[] x)	void	Yes
setDate(int parameterIndex, Date x)	void	Yes
setDouble(int parameterIndex, double x)	void	Yes
setFloat(int parameterIndex, float x)	void	Yes
setInt(int parameterIndex, int x)	void	Yes
setLong(int parameterIndex, long x)	void	Yes
setShort(int parameterIndex, short x)	void	Yes
setString(int parameterIndex, String x)	void	Yes
setNString(int parameterIndex, String x)	void	Yes
addBatch()	void	Yes
executeBatch()	int[]	Yes

说明

- addBatch()、execute()必须在clearBatch()之后才能执行。
- 调用executeBatch()方法并不会清除batch。用户必须显式使用clearBatch()清除 。
- 在添加了一个batch的绑定变量后，用户若想重用这些值(再次添加一个batch)，无需再次使用set*()方法。
- 以下方法是从java.sql.Statement继承而来：close、execute、executeQuery、executeUpdate、getConnection、getResultSet、getUpdateCount、isClosed、setMaxRows、setFetchSize。
- executeLargeUpdate()方法必须在JDBC4.2及以上版本使用。

5.4.4.6 java.sql.ResultSet

java.sql.ResultSet是执行结果集接口。

表 5-18 对 java.sql.ResultSet 的支持情况

方法名	返回值类型	支持JDBC 4
absolute(int row)	Boolean	Yes
afterLast()	void	Yes
beforeFirst()	void	Yes
cancelRowUpdates()	void	Yes
clearWarnings()	void	Yes
close()	void	Yes
deleteRow()	void	Yes
findColumn(String columnLabel)	int	Yes
first()	Boolean	Yes
getArray(int columnIndex)	Array	Yes
getArray(String columnLabel)	Array	Yes
getAsciiStream(int columnIndex)	InputStream	Yes
getAsciiStream(String columnLabel)	InputStream	Yes
getBigDecimal(int columnIndex)	BigDecimal	Yes
getBigDecimal(String columnLabel)	BigDecimal	Yes
getBinaryStream(int columnIndex)	InputStream	Yes
getBinaryStream(String columnLabel)	InputStream	Yes
getBlob(int columnIndex)	Blob	Yes
getBlob(String columnLabel)	Blob	Yes
getBoolean(int columnIndex)	Boolean	Yes
getBoolean(String columnLabel)	Boolean	Yes
getByte(int columnIndex)	byte	Yes

方法名	返回值类型	支持JDBC 4
getBytes(int columnIndex)	byte[]	Yes
getByte(String columnLabel)	byte	Yes
getBytes(String columnLabel)	byte[]	Yes
getCharacterStream(int columnIndex)	Reader	Yes
getCharacterStream(String columnLabel)	Reader	Yes
getClob(int columnIndex)	Clob	Yes
getClob(String columnLabel)	Clob	Yes
getConcurrency()	int	Yes
getCursorName()	String	Yes
getDate(int columnIndex)	Date	Yes
getDate(int columnIndex, Calendar cal)	Date	Yes
getDate(String columnLabel)	Date	Yes
getDate(String columnLabel, Calendar cal)	Date	Yes
getDouble(int columnIndex)	double	Yes
getDouble(String columnLabel)	double	Yes
getFetchDirection()	int	Yes
getFetchSize()	int	Yes
getFloat(int columnIndex)	float	Yes
getFloat(String columnLabel)	float	Yes
getInt(int columnIndex)	int	Yes
getInt(String columnLabel)	int	Yes
getLong(int columnIndex)	long	Yes

方法名	返回值类型	支持JDBC 4
getLong(String columnLabel)	long	Yes
getMetaData()	ResultSetMetaData	Yes
getObject(int columnIndex)	Object	Yes
getObject(int columnIndex, Class<T> type)	<T> T	Yes
getObject(int columnIndex, Map<String,Class<?>> map)	Object	Yes
getObject(String columnLabel)	Object	Yes
getObject(String columnLabel, Class<T> type)	<T> T	Yes
getObject(String columnLabel, Map<String,Class<?>> map)	Object	Yes
getRow()	int	Yes
getShort(int columnIndex)	short	Yes
getShort(String columnLabel)	short	Yes
getSQLXML(int columnIndex)	SQLXML	Yes
getSQLXML(String columnLabel)	SQLXML	Yes
getStatement()	Statement	Yes
getString(int columnIndex)	String	Yes
getString(String columnLabel)	String	Yes
getNString(int columnIndex)	String	Yes
getNString(String columnLabel)	String	Yes

方法名	返回值类型	支持JDBC 4
getTime(int columnIndex)	Time	Yes
getTime(int columnIndex, Calendar cal)	Time	Yes
getTime(String columnLabel)	Time	Yes
getTime(String columnLabel, Calendar cal)	Time	Yes
getTimestamp(int columnIndex)	Timestamp	Yes
getTimestamp(int columnIndex, Calendar cal)	Timestamp	Yes
getTimestamp(String columnLabel)	Timestamp	Yes
getTimestamp(String columnLabel, Calendar cal)	Timestamp	Yes
getType()	int	Yes
getWarnings()	SQLWarning	Yes
insertRow()	void	Yes
isAfterLast()	Boolean	Yes
isBeforeFirst()	Boolean	Yes
isClosed()	Boolean	Yes
isFirst()	Boolean	Yes
isLast()	Boolean	Yes
last()	Boolean	Yes
moveToCurrentRow()	void	Yes
moveToInsertRow()	void	Yes
next()	Boolean	Yes
previous()	Boolean	Yes
refreshRow()	void	Yes
relative(int rows)	Boolean	Yes
rowDeleted()	Boolean	Yes

方法名	返回值类型	支持JDBC 4
rowInserted()	Boolean	Yes
rowUpdated()	Boolean	Yes
setFetchDirection(int direction)	void	Yes
setFetchSize(int rows)	void	Yes
updateArray(int columnIndex, Array x)	void	Yes
updateArray(String columnLabel, Array x)	void	Yes
updateAsciiStream(int columnIndex, InputStream x, int length)	void	Yes
updateAsciiStream(String columnLabel, InputStream x, int length)	void	Yes
updateBigDecimal(int columnIndex, BigDecimal x)	void	Yes
updateBigDecimal(String columnLabel, BigDecimal x)	void	Yes
updateBinaryStream(int columnIndex, InputStream x, int length)	void	Yes
updateBinaryStream(String columnLabel, InputStream x, int length)	void	Yes
updateBoolean(int columnIndex, boolean x)	void	Yes
updateBoolean(String columnLabel, boolean x)	void	Yes
updateByte(int columnIndex, byte x)	void	Yes
updateByte(String columnLabel, byte x)	void	Yes
updateBytes(int columnIndex, byte[] x)	void	Yes
updateBytes(String columnLabel, byte[] x)	void	Yes

方法名	返回值类型	支持JDBC 4
updateCharacterStream(int columnIndex, Reader x, int length)	void	Yes
updateCharacterStream(String columnLabel, Reader reader, int length)	void	Yes
updateDate(int columnIndex, Date x)	void	Yes
updateDate(String columnLabel, Date x)	void	Yes
updateDouble(int columnIndex, double x)	void	Yes
updateDouble(String columnLabel, double x)	void	Yes
updateFloat(int columnIndex, float x)	void	Yes
updateFloat(String columnLabel, float x)	void	Yes
updateInt(int columnIndex, int x)	void	Yes
updateInt(String columnLabel, int x)	void	Yes
updateLong(int columnIndex, long x)	void	Yes
updateLong(String columnLabel, long x)	void	Yes
updateNull(int columnIndex)	void	Yes
updateNull(String columnLabel)	void	Yes
updateObject(int columnIndex, Object x)	void	Yes
updateObject(int columnIndex, Object x, int scaleOrLength)	void	Yes
updateObject(String columnLabel, Object x)	void	Yes

方法名	返回值类型	支持JDBC 4
updateObject(String columnLabel, Object x, int scaleOrLength)	void	Yes
updateRow()	void	Yes
updateShort(int columnIndex, short x)	void	Yes
updateShort(String columnLabel, short x)	void	Yes
updateSQLXML(int columnIndex, SQLXML xmlObject)	void	Yes
updateSQLXML(String columnLabel, SQLXML xmlObject)	void	Yes
updateString(int columnIndex, String x)	void	Yes
updateString(String columnLabel, String x)	void	Yes
updateTime(int columnIndex, Time x)	void	Yes
updateTime(String columnLabel, Time x)	void	Yes
updateTimestamp(int columnIndex, Timestamp x)	void	Yes
updateTimestamp(String columnLabel, Timestamp x)	void	Yes
wasNull()	Boolean	Yes

说明

- 一个Statement不能有多个处于“open”状态的ResultSet。
- 用于遍历结果集（ResultSet）的游标（Cursor）在被提交后不能保持“open”的状态。

5.4.4.7 java.sql.ResultSetMetaData

java.sql.ResultSetMetaData是对ResultSet对象相关信息的具体描述。

表 5-19 对 java.sql.ResultSetMetaData 的支持情况

方法名	返回值类型	支持JDBC 4
getCatalogName(int column)	String	Yes
getColumnClassName(int column)	String	Yes
getColumnCount()	int	Yes
getColumnDisplaySize(int column)	int	Yes
getColumnLabel(int column)	String	Yes
getColumnName(int column)	String	Yes
getColumnType(int column)	int	Yes
getColumnTypeName(int column)	String	Yes
getPrecision(int column)	int	Yes
getScale(int column)	int	Yes
getSchemaName(int column)	String	Yes
getTableName(int column)	String	Yes
isAutoIncrement(int column)	boolean	Yes
isCaseSensitive(int column)	boolean	Yes
isCurrency(int column)	boolean	Yes
isDefinitelyWritable(int column)	boolean	Yes
isNullable(int column)	int	Yes
isReadOnly(int column)	boolean	Yes
isSearchable(int column)	boolean	Yes
isSigned(int column)	boolean	Yes
isWritable(int column)	boolean	Yes

说明

uppercaseAttributeName为true时，下面接口会将查询结果转为大写，可转换范围为26个英文字母。

- public String getColumnName(int column)
- public String getColumnLabel(int column)

5.4.4.8 java.sql.Statement

java.sql.Statement是SQL语句接口。

表 5-20 对 java.sql.Statement 的支持情况

方法名	返回值类型	支持JDBC 4
addBatch(String sql)	void	Yes
clearBatch()	void	Yes
clearWarnings()	void	Yes
close()	void	Yes
closeOnCompletion()	void	Yes
execute(String sql)	Boolean	Yes
execute(String sql, int autoGeneratedKeys)	Boolean	Yes
execute(String sql, int[] columnIndexes)	Boolean	Yes
execute(String sql, String[] columnNames)	Boolean	Yes
executeBatch()	Boolean	Yes
executeQuery(String sql)	ResultSet	Yes
executeUpdate(String sql)	int	Yes
executeUpdate(String sql, int autoGeneratedKeys)	int	Yes
executeUpdate(String sql, int[] columnIndexes)	int	Yes
executeUpdate(String sql, String[] columnNames)	int	Yes
getConnection()	Connection	Yes

方法名	返回值类型	支持JDBC 4
getFetchDirection()	int	Yes
getFetchSize()	int	Yes
getGeneratedKeys()	ResultSet	Yes
getMaxFieldSize()	int	Yes
getMaxRows()	int	Yes
getMoreResults()	Boolean	Yes
getMoreResults(int current)	Boolean	Yes
getResultSet()	ResultSet	Yes
getResultSetConcurrency()	int	Yes
getResultSetHoldability()	int	Yes
getResultSetType()	int	Yes
getQueryTimeout()	int	Yes
getUpdateCount()	int	Yes
getWarnings()	SQLWarning	Yes
isClosed()	Boolean	Yes
isCloseOnCompletion()	Boolean	Yes
isPoolable()	Boolean	Yes
setCursorName(String name)	void	Yes
setEscapeProcessing(boolean enable)	void	Yes
setFetchDirection(int direction)	void	Yes
setMaxFieldSize(int max)	void	Yes
setMaxRows(int max)	void	Yes
setPoolable(boolean poolable)	void	Yes
setQueryTimeout(int seconds)	void	Yes
setFetchSize(int rows)	void	Yes

方法名	返回值类型	支持JDBC 4
cancel()	void	Yes
executeLargeUpdate(String sql)	long	No
getLargeUpdateCount()	long	No
executeLargeBatch()	long	No
executeLargeUpdate(String sql, int autoGeneratedKeys)	long	No
executeLargeUpdate(String sql, int[] columnIndexes)	long	No
executeLargeUpdate(String sql, String[] columnNames)	long	No

说明

- 通过setFetchSize可以减少结果集在客户端的内存占用情况。它的原理是通过将结果集打包成游标，然后分段处理，所以会加大数据库与客户端的通信量，会有性能损耗。
- 由于数据库游标是事务内有效，所以，在设置setFetchSize的同时，需要将连接设置为非自动提交模式，setAutoCommit(false)。同时在业务数据需要持久化到数据库中时，在连接上执行提交操作。
- LargeUpdate相关方法必须在JDBC4.2及以上版本使用。

5.4.4.9 javax.sql.ConnectionPoolDataSource

javax.sql.ConnectionPoolDataSource是数据源连接池接口。

表 5-21 对 javax.sql.ConnectionPoolDataSource 的支持情况

方法名	返回值类型	支持JDBC 4
getPooledConnection()	PooledConnection	Yes
getPooledConnection(String user,String password)	PooledConnection	Yes

5.4.4.10 javax.sql.DataSource

javax.sql.DataSource是数据源接口。

表 5-22 对 javax.sql.DataSource 接口的支持情况

方法名	返回值类型	支持JDBC 4
getConneciton()	Connection	Yes
getConnection(String username,String password)	Connection	Yes
getLoginTimeout()	int	Yes
getLogWriter()	PrintWriter	Yes
setLoginTimeout(int seconds)	void	Yes
setLogWriter(PrintWriter out)	void	Yes

5.4.4.11 javax.sql.PooledConnection

javax.sql.PooledConnection是由连接池创建的连接接口。

表 5-23 对 javax.sql.PooledConnection 的支持情况

方法名	返回值类型	支持JDBC 4
addConnectionEventListener (ConnectionEventListener listener)	void	Yes
close()	void	Yes
getConnection()	Connection	Yes
removeConnectionEventListener (ConnectionEventListener listener)	void	Yes

5.4.4.12 javax.naming.Context

javax.naming.Context是连接配置的上下文接口。

表 5-24 对 javax.naming.Context 的支持情况

方法名	返回值类型	支持JDBC 4
bind(Name name, Object obj)	void	Yes
bind(String name, Object obj)	void	Yes
lookup(Name name)	Object	Yes
lookup(String name)	Object	Yes

方法名	返回值类型	支持JDBC 4
rebind(Name name, Object obj)	void	Yes
rebind(String name, Object obj)	void	Yes
rename(Name oldName, Name newName)	void	Yes
rename(String oldName, String newName)	void	Yes
unbind(Name name)	void	Yes
unbind(String name)	void	Yes

5.4.4.13 javax.naming.spi.InitialContextFactory

javax.naming.spi.InitialContextFactory是初始连接上下文工厂接口。

表 5-25 对 javax.naming.spi.InitialContextFactory 的支持情况

方法名	返回值类型	支持JDBC 4
getInitialContext(Hashtable<?,?> environment)	Context	Yes

5.4.4.14 CopyManager

CopyManager是GaussDB JDBC驱动中提供的一个API接口类，用于批量向GaussDB中导入数据。

CopyManager 的继承关系

CopyManager类位于org.postgresql.copy Package中，继承自java.lang.Object类，该类的声明如下：

```
public class CopyManager
extends Object
```

构造方法

public CopyManager(BaseConnection connection)

throws SQLException

常用方法

表 5-26 CopyManager 常用方法

返回值	方法	描述	throws	支持JDBC4
CopyIn	copyIn(String sql)	-	SQLException	Yes
long	copyIn(String sql, InputStream from)	使用COPY FROM STDIN从InputStream中快速向数据库中的表加载数据。	SQLException, IOException	Yes
long	copyIn(String sql, InputStream from, int bufferSize)	使用COPY FROM STDIN从InputStream中快速向数据库中的表加载数据。	SQLException, IOException	Yes
long	copyIn(String sql, Reader from)	使用COPY FROM STDIN从Reader中快速向数据库中的表加载数据。	SQLException, IOException	Yes
long	copyIn(String sql, Reader from, int bufferSize)	使用COPY FROM STDIN从Reader中快速向数据库中的表加载数据。	SQLException, IOException	Yes
CopyOut	copyOut(String sql)	-	SQLException	Yes
long	copyOut(String sql, OutputStream to)	将一个COPY TO STDOUT的结果集从数据库发送到OutputStream类中。	SQLException, IOException	Yes
long	copyOut(String sql, Writer to)	将一个COPY TO STDOUT的结果集从数据库发送到Writer类中。	SQLException, IOException	Yes

5.4.4.15 PGReplicationConnection

PGReplicationConnection是GaussDB JDBC驱动中提供的一个API接口类，用于执行逻辑复制相关的功能。

PGReplicationConnection 的继承关系

PGReplicationConnection是逻辑复制的接口，实现类是PGReplicationConnectionImpl，该类位于org.postgresql.replication Package中，该类的声明如下：

```
public class PGReplicationConnection implements PGReplicationConnection
```

构造方法

public PGReplicationConnection(BaseConnection connection)

常用方法

表 5-27 PGReplicationConnection 常用方法

返回值	方法	描述	throws
ChainedCreateReplicationSlotBuilder	createReplicationSlot()	用于创建逻辑复制槽。只能创建LSN序逻辑复制槽，若需要创建CSN序逻辑复制槽，请参考逻辑复制SQL函数pg_create_logical_replication_slot。	-
void	dropReplicationSlot(String slotName)	用于删除逻辑复制槽。	SQLException,IOException
ChainedStreamBuilder	replicationStream()	用户开启逻辑复制。	-

5.4.4.16 PGReplicationStream

PGReplicationStream是GaussDB JDBC驱动中提供的一个API接口类，用于操作逻辑复制流。

PGReplicationStream 的继承关系

PGReplicationStream是逻辑复制的接口，实现类是V3PGReplicationStream，该类位于org.postgresql.core.v3.replication Package中，该类的声明如下：

```
public class V3PGReplicationStream implements PGReplicationStream
```

构造方法

```
public V3PGReplicationStream(CopyDual copyDual, LogSequenceNumber startLSN, long updateIntervalMs, ReplicationType replicationType)
```


常用方法

表 5-28 PGReplicationConnection 常用方法

返回值	方法	描述	throws
void	close()	结束逻辑复制，并释放资源。	SQLException
void	forceUpdateStatus()	强制将上次接收、刷新和应用的 LSN 状态发送到后端。	SQLException
LogSequenceNumber	getLastAppliedLSN()	获取上次主机日志回放的LSN。	-
LogSequenceNumber	getLastFlushedLSN()	获取上次主机刷新的LSN，即当前逻辑解码推进的LSN。	-
LogSequenceNumber	getLastReceiveLSN()	获取上次接收的LSN（针对LSN序复制槽）或CSN（针对CSN序复制槽）。	-
boolean	isClosed()	复制流是否关闭。	-
ByteBuffer	read()	从后端读取下一条WAL记录。如果读取不到，该方法阻塞I/O读。	SQLException
ByteBuffer	readPending()	从后端读取下一条WAL记录。如果读取不到，该方法不阻塞I/O读。	SQLException
void	setAppliedLSN(LogSequenceNumber applied)	设置应用的LSN。	-
void	setFlushedLSN(LogSequenceNumber flushed)	设置刷新的LSN（针对LSN序复制槽）或CSN（针对CSN序复制槽），在下次更新时发送至后端，用于推进服务端LSN（针对LSN序复制槽）或CSN（针对CSN序复制槽）。	-

5.4.4.17 ChainedStreamBuilder

ChainedStreamBuilder是GaussDB JDBC驱动中提供的一个API接口类，用于构建复制流。

ChainedStreamBuilder 的继承关系

ChainedStreamBuilder是逻辑复制的接口，实现类是ReplicationStreamBuilder，该类位于org.postgresql.replication.fluent Package中，该类的声明如下：

```
public class ReplicationStreamBuilder implements ChainedStreamBuilder
```

构造方法

public ReplicationStreamBuilder(final BaseConnection connection)

常用方法

表 5-29 ReplicationStreamBuilder 常用方法

返回值	方法	描述	throws
ChainedLogicalStreamBuilder	logical()	创建逻辑复制流。	-
ChainedPhysicalStreamBuilder	physical()	创建物理复制流。	-

5.4.4.18 ChainedCommonStreamBuilder

ChainedCommonStreamBuilder是GaussDB JDBC驱动中提供的一个API接口类，用于为逻辑和物理复制指定通用参数。

ChainedCommonStreamBuilder 的继承关系

ChainedCommonStreamBuilder是逻辑复制的接口，实现抽象类是AbstractCreateSlotBuilder，该类的继承类是LogicalCreateSlotBuilder，位于org.postgresql.replication.fluent.logical Package中，该类的声明如下：

```
public class LogicalCreateSlotBuilder
    extends AbstractCreateSlotBuilder<ChainedLogicalCreateSlotBuilder>
    implements ChainedLogicalCreateSlotBuilder
```

构造方法

public LogicalCreateSlotBuilder(BaseConnection connection)

常用方法

表 5-30 LogicalCreateSlotBuilder 常用方法

返回值	方法	描述	throws
T	withSlotName(String slotName)	指定复制槽名。	-

返回值	方法	描述	throws
ChainedLogicalCreateSlotBuilder	withOutputPlugin(String outputPlugin)	插件名称，当前支持mppdb_decoding。 mppdb_decoding：一种解码的输出格式，设置后输出内容为JSON格式。输出的结果包含相关数据的属性信息和属性对应的值。	-
void	make()	在数据库中创建具有指定参数的插槽。	SQLException
ChainedLogicalCreateSlotBuilder	self()	返回ChainedLogicalCreateSlotBuilder的实现。	-

5.4.4.19 PGobject

表 5-31 PGobject 常用方法

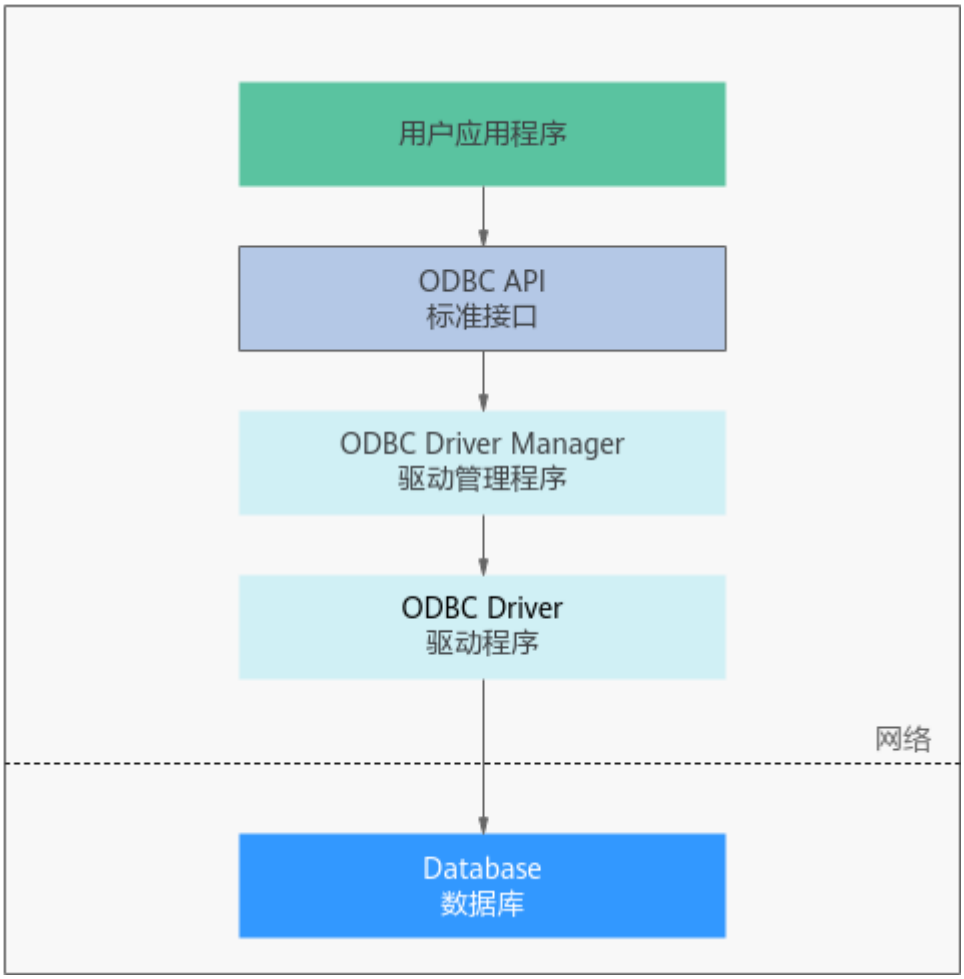
返回值	方法	描述	throws	支持JDBC 4
Object[]	getStruct()	获取复合类型子类型名，按创建顺序排序。	-	Yes
String	getValue()	获取复合类型字符串形式值。	-	Yes
String[]	getArrayValue()	获取复合类型数组形式值，以复合数据类型字段顺序排序。	-	Yes
Object[]	getAttributes()	获取复合类型对象数组形式值（如果组成字段类型为Table类型和Array类型则返回为PgArray，如果组成字段为复合类型则返回PGobject，其他类型返回字符串值）。	SQLException	Yes

5.5 基于 ODBC 开发

ODBC（Open Database Connectivity，开放数据库互连）是由Microsoft公司基于X/OPEN CLI提出的用于访问数据库的应用程序编程接口。应用程序通过ODBC提供的API与数据库进行交互，增强了应用程序的可移植性、扩展性和可维护性。

ODBC的系统结构请参见图5-3。

图 5-3 ODBC 系统结构



GaussDB目前在以下环境中提供对ODBC的支持。

表 5-32 ODBC 支持平台

操作系统	平台
EulerOS 2.5	x86_64位
EulerOS 2.9	ARM64位
EulerOS 2.10	x86_64位
EulerOS 2.10	ARM64位
Windows 7	x86_32位
Windows 7	x86_64位
Windows Server 2008	x86_32位
Windows Server 2008	x86_64位

操作系统	平台
Kylin V10	x86_64位
Kylin V10	ARM64位
UnionTech V20	x86_64位
UnionTech V20	ARM64位

UNIX/Linux系统下的驱动程序管理器主要有unixODBC和iODBC，在这选择驱动管理器unixODBC-2.3.7作为连接数据库的组件。

Windows系统自带ODBC驱动程序管理器，在控制面板->管理工具中可以找到数据源（ODBC）选项。

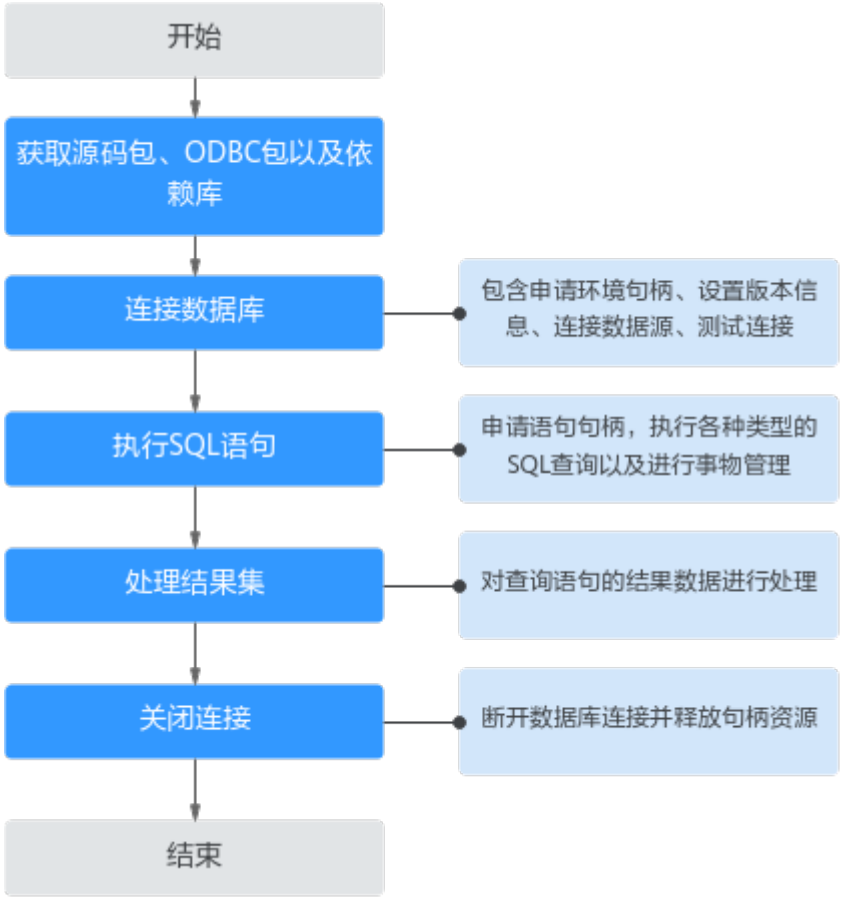
说明

当前数据库ODBC驱动基于开源版本，对于tinyint、smalldatetime、nvarchar、nvarchar2类型，在获取数据类型的时候，可能会出现不兼容的情况。

5.5.1 开发流程

ODBC开发流程如图5-4所示。

图 5-4 ODBC 开发流程图



5.5.2 开发步骤

5.5.2.1 获取源码包、ODBC 包以及依赖库

基于ODBC开发所需的包，依赖库和头文件以及其获取方式如表5-33所示。

表 5-33 ODBC 应用程序开发环境准备

所需资源	获取方式
unixODBC源码包	• unixODBC源码包获取参考地址：**https://www.unixodbc.org/unixODBC-2.3.7.tar.gz**。 • MD5文件下载地址：**https://www.unixodbc.org/unixODBC-2.3.7.tar.gz.md5**。 下载后需使用MD5文件对unixODBC源码包进行完整性校验（查看MD5值，对比MD5值是否与源码包一致）。
Linux系统下的ODBC包以及依赖库	从发布包中获取，包名为GaussDB-Kernel_数据库版本号_操作系统版本号_64bit_Odbc.tar.gz。Linux环境下，开发应用程序要用到unixODBC提供的头文件（sql.h、sqlext.h等）和库libodbc.so。这些头文件和库可从unixODBC-2.3.7的源码包中获得。
Windows系统下的ODBC包以及依赖库	从发布包中获取，包名为GaussDB-Kernel_数据库版本号_Windows_X86_Odbc.tar.gz（32位）和GaussDB-Kernel_数据库版本号_Windows_X64_Odbc.tar.gz（64位）。Windows环境下，开发应用程序用到的相关头文件和库文件由系统自带。

5.5.2.2 连接数据库

5.5.2.2.1 Linux 下配置数据源

ODBC连接数据库之前需要准备好所需资源。连接数据库是通过配置ODBC数据源，使用ODBC API或者相应的驱动程序，实现应用程序与数据库之间的通信和交互。本节介绍如何在Linux系统下配置数据源并连接到数据库。

操作步骤

步骤1 安装unixODBC（默认unixODBC源码包已在环境准备中获取）。如果机器上已经安装了其他版本的unixODBC，可以直接覆盖安装。

以unixODBC-2.3.7版本为例，在客户端执行如下命令安装unixODBC。

```
tar zxvf unixODBC-2.3.7.tar.gz
cd unixODBC-2.3.7

./configure --enable-gui=no #如果要在ARM服务器上编译，请追加一个configure参数： --build=aarch64-unknown-linux-gnu
make
#安装可能需要root权限
make install
```

说明

- 目前不支持unixODBC-2.2.1版本。
- 默认安装到“/usr/local”目录下，生成数据源文件到 “/usr/local/etc”目录下，库文件生成在“/usr/local/lib”目录。
- 通过编译带有--enable-fastvalidate=yes选项的unixODBC来获得更高性能。但此选项可能会导致向ODBC API传递无效句柄的应用程序发生故障，而不是返回SQL_INVALID_HANDLE错误。

步骤2 替换客户端GaussDB驱动程序。

将GaussDB-Kernel_数据库版本号_操作系统版本号_64bit_Odbc.tar.gz解压。解压后会得到两个文件夹：lib与odbc，在odbc文件夹中还会有一个lib文件夹。将解压后得到的/lib文件夹与/odbc/lib文件夹中的所有动态库都复制到“/usr/local/lib”目录下。

步骤3 配置数据源。

1. 配置ODBC驱动文件。

   在“/usr/local/etc/odbcinst.ini”文件中追加以下内容。

   ```
   [GaussMPP]
   Driver64=/usr/local/lib/psqlodbcw.so
   setup=/usr/local/lib/psqlodbcw.so
   ```

   odbcinst.ini文件中的配置参数说明如表5-34所示。

   表 5-34 odbcinst.ini 文件配置参数

参数	描述	示例
[DriverName]	驱动器名称，对应数据源DSN中的驱动名。	[GaussMPP]
Driver64	驱动动态库的路径。	Driver64=/usr/local/lib/psqlodbcw.so
setup	驱动安装路径，与Driver64中动态库的路径一致。	setup=/usr/local/lib/psqlodbcw.so

2. 配置数据源文件。

   在“/usr/local/etc/odbc.ini”文件中追加以下内容。

   ```
   [gaussdb]
   Driver=GaussMPP
   Servername=127.0.0.1 #数据库Server IP
   Database=postgres  #数据库名
   Username=omm  #数据库用户名
   Password=  #数据库用户密码
   Port=8000  #数据库侦听端口
   Sslmode=allow
   ```

   odbc.ini文件配置参数说明如表5-35所示。

   表 5-35 odbc.ini 文件配置参数

参数	描述	示例
[DSN]	数据源的名称。	[gaussdb]

参数	描述	示例
Driver	驱动名，对应odbcinst.ini中的DriverName。	Driver=GaussMPP
Servername	服务器的IP地址。可配置多个IP地址。	Servername=127.0.0.1
Database	要连接的数据库的名称。	Database=postgres
Username	数据库用户名称。	Username=omm
Password	数据库用户密码。 **说明** ODBC驱动本身已经对内存密码进行过清理，以保证用户密码在连接后不会再在内存中保留。 但是如果配置了此参数，由于UnixODBC对数据源文件等进行缓存，可能导致密码长期保留在内存中。 推荐在应用程序连接时，将密码传递给相应API，而非写在数据源配置文件中。同时连接成功后，应当及时清理保存密码的内存段。 **须知** 配置文件中填写密码时，需要遵循http规则： 1. 字符应当采用URL编码规范，如"!"应写作"%21"，"%"应写作"%25"，因此应当特别注意字符。 2. "+"会被替换为空格""。	Password=********
Port	服务器的端口号。	Port=8000
Sslmode	是否启用 SSL 连接。 **说明** 关于Sslmode的选项的允许值，具体信息如**表5-36**所示。	Sslmode=allow

参数	描述	示例
Debug	是否启用调试模式。 取值范围：0 ~ INT_MAX – 设置为0时表示不开启。 – 设置为大于0时表示将会打印gsqlodbc驱动的mylog，日志生成目录为/tmp/。 默认值为0。	Debug=1
UseServerSidePrepare	是否开启数据库端扩展查询协议。 取值范围：0，1 – 取值为0表示不开启。 – 取值为1表示开启。 默认值为1。	UseServerSidePrepare=1
UseBatchProtocol	是否开启批量查询协议（打开可提高DML性能）。 取值范围：0，1 – 取值为0时，不使用批量查询协议（主要用于与早期数据库版本通信兼容）。 – 取值为1，并且数据库support_batch_bind参数存在且为on时，将打开批量查询协议。 默认值为1。	UseBatchProtocol=1
ForExtensionConnector	此开关控制着savepoint是否发送，savepoint相关问题可以注意此开关， 取值范围：0，1 – 取值为0时表示发送savepoint。 – 取值为1时表示不发送savepoint。 默认值为1。	ForExtensionConnector=1

参数	描述	示例
ConnectionExtraInfo	GUC参数connection_info中显示驱动部署路径和进程属主用户的开关。 取值范围：0，1 – 取值为0时表示不打开此开关。 – 取值为1时表示打开此开关。 默认值为0。 **说明** 默认值为0。当设置为1时，ODBC驱动会将当前驱动的部署路径、进程属主用户上报到数据库中，记录在GUC参数connection_info里，同时可以在PG_STAT_ACTIVITY中查询到。	ConnectionExtraInfo=1
BoolsAsChar	是否将布尔值作为字符处理。 取值范围：0，1 – 取值为0时表示Bools值将会映射为SQL_BIT。 – 取值为1时表示Bools值将会映射为SQL_CHAR。 默认值为1。	BoolsAsChar = 1
RowVersioning	是否在更新一行数据时，允许应用检测数据有没有被其他用户进行修改。 取值范围：0，1 – 取值为0时表示不允许应用检测。 – 取值为1时表示允许应用检测。 默认值为0。	RowVersioning=1

参数	描述	示例
ShowSystemTables	是否将默认系统表格视为普通SQL表格。 取值范围：0，1 – 取值为0时驱动不会将默认系统表格视为普通SQL表格。 – 取值为1时驱动将默认系统表格视为普通SQL表格。 默认值为0。	ShowSystemTables=1
MaxCacheQueries	控制每个连接缓存的预编译语句个数。 取值范围：0 ~ 4096 默认值为0。 **说明** 如果设置为0，则不开启客户端预编译语句缓存池。设置为大于4096的值会限制为4096。如果执行过的语句个数超过MaxCacheQueries设置的上限，则淘汰最近最少使用的语句。	MaxCacheQueries=128
MaxCacheSizeMiB	控制每个连接缓存的预编译语句总大小，在MaxCacheQueries大于0时生效。 取值范围：0 ~ 4096 默认值为1。 **说明** 如果缓存的语句总长度大于MaxCacheSizeMiB则淘汰最近最少使用的语句。单位为MB，设置为大于4096的值会限制为4096。	MaxCacheSizeMiB=10

参数	描述	示例
TcpUserTimeout	在支持TCP_USER_TIMEOUT套接字选项的操作系统上，指定传输的数据在TCP连接被强制关闭之前可以保持未确认状态的最大时长。 取值范围：0 ~ INT_MAX 默认值为0。 **说明** 0表示使用系统缺省。通过Unix域套接字做的连接忽略这个参数。单位为毫秒。	TcpUserTimeout=5000

表 5-36 Sslmode 的可选项及其描述

Sslmode	是否会启用SSL加密	描述
disable	否	不使用SSL安全连接。
allow	可能	如果数据库服务器要求使用，则可以使用SSL安全加密连接，但不验证数据库服务器的真实性。
prefer	可能	如果数据库支持，那么首选使用SSL安全加密连接，但不验证数据库服务器的真实性。
require	是	必须使用SSL安全连接，但是只做了数据加密，并不验证数据库服务器的真实性。
verify-ca	是	必须使用SSL安全连接，并且验证数据库是否具有可信证书机构签发的证书。
verify-full	是	必须使用SSL安全连接，在verify-ca的验证范围之外，同时验证数据库所在主机的主机名是否与证书内容一致。GaussDB不支持此模式。

说明

用户通过ODBC连接GaussDB服务器时，可以通过开启SSL加密客户端和服务器之间的通讯。在使用SSL时，默认用户已经获取了服务端和客户端所需要的证书和私钥文件，关于证书等文件的获取请参考Openssl相关文档和命令。

步骤4 在客户端配置环境变量。

```
vim ~/.bashrc
```

在配置文件中追加以下内容。


```
export LD_LIBRARY_PATH=/usr/local/lib/:$LD_LIBRARY_PATH
export ODBCSYSINI=/usr/local/etc
export ODBCINI=/usr/local/etc/odbc.ini
```

步骤5 执行如下命令使设置生效。

```
source ~/.bashrc
```

步骤6 测试连接。

安装后/usr/bin下面会存放生成的二进制，可执行isql -v gaussdb（数据源名称）命令。

- 如果显示如下信息，表明配置正确，连接成功。

```
+---------------------------------------+
| Connected!                            |
|                                       |
| sql-statement                         |
| help [tablename]                      |
| quit                                  |
|                                       |
+---------------------------------------+
```

- 若显示ERROR信息，则表明配置错误。请检查上述配置是否正确。

说明

目前通过ODBC连接数据库时，会如下设置内核参数：

```
SET extra_float_digits = 2;
SET DateStyle = 'ISO';
```

这些参数可能会导致ODBC客户端的行为与gsql客户端的行为不一致，例如，Date数据显示方式、浮点数精度表示。如果实际期望和这些配置不符，建议在ODBC应用代码中显式设定这些参数。

----结束

常见问题处理

- [UnixODBC][Driver Manager]Can't open lib 'xxx/xxx/psqlodbcw.so' : file not found.

  此问题的可能原因：

  - odbcinst.ini文件中配置的路径不正确

    确认的方法：执行ls命令查询错误信息中的路径，以确保该psqlodbcw.so文件存在，同时具有执行权限。

  - psqlodbcw.so的依赖库不存在，或者不在系统环境变量中

    确认的方法：执行ldd命令查询错误信息中的路径，如果是缺少libodbc.so.1等UnixODBC的库，那么按照“操作步骤”中的方法重新配置UnixODBC，并确保它的安装路径下的lib目录添加到了LD_LIBRARY_PATH中。如果是缺少其他库，请将ODBC驱动包中的lib目录添加到LD_LIBRARY_PATH中。如果缺少其他标准库，请自行安装。

- [UnixODBC]connect to server failed: no such file or directory

  此问题的可能原因：

  - 配置了错误的/不可达的数据库地址，或者端口

    请检查数据源配置中的Servername及Port配置项。

  - 服务器侦听不正确

    如果确认Servername及Port配置正确，请根据“操作步骤”中数据库服务器的相关配置，确保数据库侦听了合适的网卡及端口。

- 防火墙及网闸设备

  请确认防火墙设置，将数据库的通信端口添加到可信端口中。

  如果有网闸设备，请确认相关的设置。

- [unixODBC]The password-stored method is not supported.

  此问题的可能原因：

  数据源中未配置Sslmode配置项。

  解决办法：

  请配置该选项至allow或以上选项。此配置的更多信息，请参见**表5-36**。

- Server common name "xxxx" does not match host name "xxxxx"

  此问题的可能原因：

  使用了SSL加密的“verify-full”选项，驱动程序会验证证书中的主机名与实际部署数据库的主机名是否一致。

  解决办法：

  碰到此问题可以使用“verify-ca”选项，不再校验主机名，或者重新生成一套与数据库所在主机名相同的CA证书。

- Driver's SQLAllocHandle on SQL_HANDLE_DBC failed

  此问题的可能原因：

  可执行文件（比如UnixODBC的isql，以下都以isql为例）与数据库驱动（psqlodbcw.so）依赖于不同的ODBC的库版本：libodbc.so.1或者libodbc.so.2。此问题可以通过如下方式确认：

  ```
  ldd `which isql` | grep odbc
  ldd psqlodbcw.so | grep odbc
  ```

  这时，如果输出的libodbc.so最后的后缀数字不同或者指向不同的磁盘物理文件，那么基本就可以断定是此问题。isql与psqlodbcw.so都会要求加载libodbc.so，这时如果它们加载的是不同的物理文件，便会导致两套完全同名的函数列表，同时出现在同一个可见域里（UnixODBC的libodbc.so.*的函数导出列表完全一致），产生冲突，无法加载数据库驱动。

  解决办法：

  确定一个要使用的UnixODBC，卸载另外一个（比如卸载库版本号为.so.2的UnixODBC），然后将剩下的.so.1的库，新建一个同名但是后缀为.so.2的软链接，便可解决此问题。

- FATAL: Forbid remote connection with trust method!

  由于安全原因，数据库主节点禁止数据库内部其他节点无认证接入。

  如果要在数据库内部访问数据库主节点，请将ODBC程序部署在数据库主节点所在机器，服务器地址使用"127.0.0.1"。建议业务系统单独部署在数据库外部，否则可能会影响数据库运行性能。

- [unixODBC][Driver Manager]Invalid attribute value

  有可能是unixODBC的版本并非推荐版本，建议通过“odbcinst --version”命令排查环境中的unixODBC版本。

- authentication method 10 not supported.

  使用开源客户端碰到此问题，可能原因：

  数据库中存储的密码校验只存储了SHA256格式哈希，而开源客户端只识别MD5校验，双方校验方法不匹配报错。

说明

- 数据库并不存储用户密码，只存储用户密码的哈希码。
- 当用户更新用户密码或者新建用户时，数据库会同时存储两种格式的哈希码，这时将兼容开源的认证协议。
- 当旧版本升级到新版本时，由于哈希的不可逆性，所以数据库无法还原用户密码，进而生成新格式的哈希，所以仍然只保留了SHA256格式的哈希，导致仍然无法使用MD5做密码认证。
- MD5加密算法安全性低，存在安全风险，建议使用更安全的加密算法。

要解决该问题，可以更新用户密码（请参见**ALTER USER**），或者新建一个用户（请参见**CREATE USER**），赋予同等权限，使用新用户连接数据库。

- unsupported frontend protocol 3.51: server supports 1.0 to 3.0

  目标数据库版本过低，或者目标数据库为开源数据库。

  请使用对应版本的数据库驱动连接目标数据库。

- FATAL: GSS authentication method is not allowed because XXXX user password is not disabled.

  目标数据库主节点的pg_hba.conf里配置了当前客户端IP使用"gss"方式来做认证，该认证算法不支持用作客户端的身份认证，请修改到"sha256"后再试。

- isql：error while loading shared libraries:xxx

  环境缺少该动态库，需要自行安装对应的库。

5.5.2.2.2 Windows 下配置数据源

Windows操作系统自带ODBC数据源管理器，无需用户手动安装管理器便可直接进行配置。

操作步骤

步骤1 替换客户端GaussDB驱动程序。

根据需要，将包名为GaussDB-Kernel_数据库版本号_Windows_X64_Odbc.tar.gz的64位驱动或包名为GaussDB-Kernel_数据库版本号_Windows_X86_Odbc.tar.gz的32位驱动解压后，单击gsqlodbc.exe进行驱动安装。

步骤2 打开驱动管理器。

在配置数据源时，请使用ODBC版本对应的ODBC驱动管理器（如果使用64位ODBC驱动，必须要使用64位的ODBC驱动管理器，假设操作系统安装盘符为C盘，如果是其他盘符，请对路径做相应修改）。

- 如果需要在64位操作系统使用32位ODBC驱动请使用：C:\Windows\SysWOW64\odbcad32.exe，请勿直接使用“控制面板 > 管理工具 > 数据源(ODBC)”。

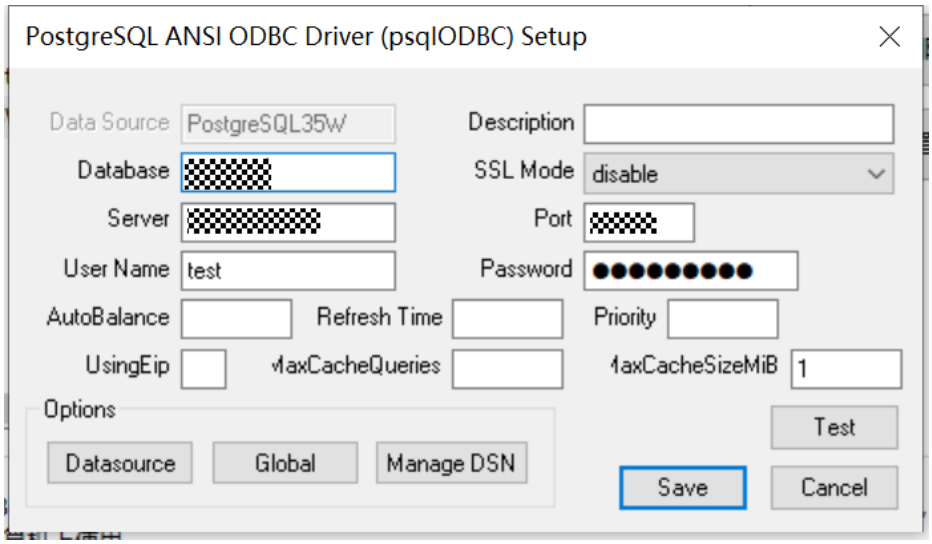
  说明

  WOW64的全称是"Windows 32-bit on Windows 64-bit"，C:\Windows\SysWOW64\存放的是64位系统上的32位运行环境。而C:\Windows\System32\存放的是与操作系统一致的运行环境，具体的技术信息请查阅Windows的相关技术文档。

- 32位操作系统请使用：C:\Windows\System32\odbcad32.exe，或者单击“计算机 > 控制面板 > 管理工具 > 数据源(ODBC)”打开驱动管理器。
- 64位操作系统请使用：控制面板 > 管理工具 > 数据源(ODBC) 打开驱动管理。

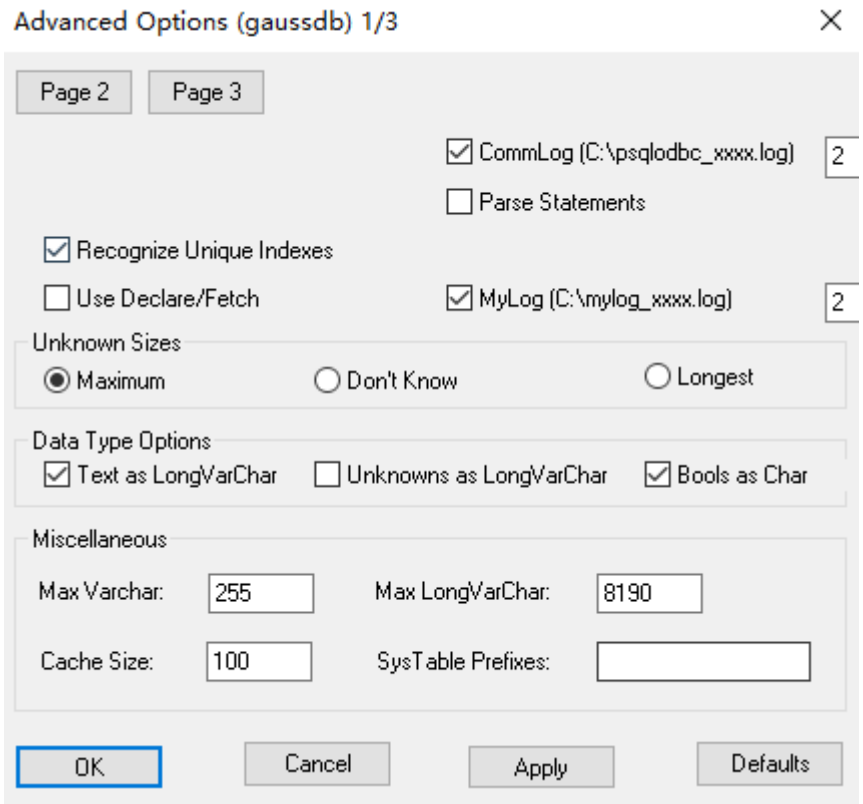
步骤3 配置数据源。

在打开的驱动管理器上，选择“用户DSN > 添加 > PostgreSQL Unicode”，然后进行配置：



参数说明请参见**Linux下配置数据源**文件参数配置。

其中单击Datasource可以选择配置是否打印日志：



须知

此界面上配置的用户名及密码信息，将会被记录在Windows注册表中，再次连接数据库时不再需要输入认证信息。但是出于安全考虑，建议在单击"Save"按钮保存配置信息前，清空相关敏感信息，在使用ODBC的连接API时，再传入所需的用户名、密码信息。

步骤4 SSL模式。

将**步骤3**中设置窗口的“SSL Mode”选项调整至“require”。

表 5-37 Sslmode 的可选项及其描述

Sslmode	是否会启用SSL加密	描述
disable	否	不使用SSL安全连接。
allow	可能	如果数据库服务器要求使用，则可以使用SSL安全加密连接，但不验证数据库服务器的真实性。
prefer	可能	如果数据库支持，那么首选使用SSL安全加密连接，但不验证数据库服务器的真实性。
require	是	必须使用SSL安全连接，但是只做了数据加密，并不验证数据库服务器的真实性。
verify-ca	是	必须使用SSL安全连接，并且验证数据库是否具有可信证书机构签发的证书。当前windows ODBC不支持cert方式认证。
verify-full	是	必须使用SSL安全连接，在verify-ca的验证范围之外，同时验证数据库所在主机的主机名是否与证书内容一致。当前windows ODBC不支持cert方式认证。

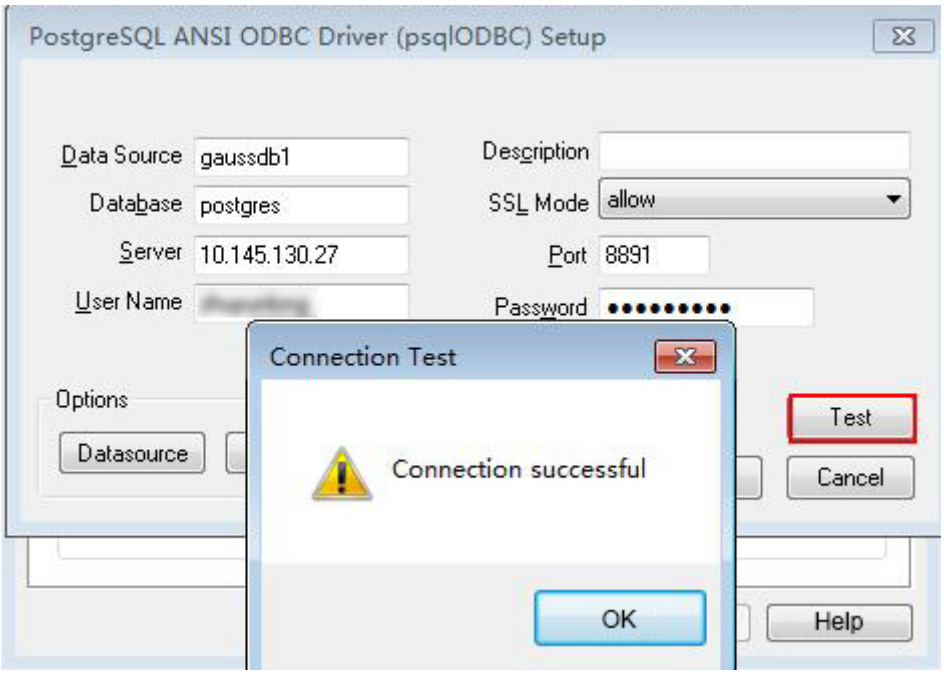
说明

用户通过ODBC连接GaussDB服务器时，可以通过开启SSL加密客户端和服务器之间的通讯。在使用SSL时，默认用户已经获取了服务端和客户端所需要的证书和私钥文件，关于证书等文件的获取请参考Openssl相关文档和命令。

步骤5 测试连接。

单击Test进行测试。

- 如果显示如下，则表明配置正确，连接成功。



- 若显示ERROR信息，则表明配置错误。请重新检查上述配置是否正确。

说明

- 目前通过ODBC连接数据库时，会设置以下内核参数：

```
SET extra_float_digits = 2;
SET DateStyle = 'ISO';
```

这些参数可能会导致ODBC客户端的行为与gsql客户端的行为不一致，例如，Date数据显示方式、浮点数精度表示。如果实际期望和这些配置不符，建议在ODBC应用代码中显式设定这些参数。

----结束

常见问题处理

- connect to server failed: no such file or directory

  此问题可能的原因：

  - 配置了错误的/不可达的数据库地址，或者端口

    请检查数据源配置中的Server及Port配置项。
  - 服务器侦听不正确

    如果确认Server及Port配置正确，请根据“操作步骤”中数据库服务器的相关配置，确保数据库侦听了合适的网卡及端口。
  - 防火墙及网闸设备

    请确认防火墙设置，将数据库的通信端口添加到可信端口中。

    如果有网闸设备，请确认相关的设置。
- The password-stored method is not supported.

  此问题可能原因：

  数据源中未配置Sslmode配置项，请调整此项至allow或以上级别，允许SSL连接，此选项的更多说明，请参见**表5-37**。
- authentication method 10 not supported.

  使用开源客户端碰到此问题，可能原因：

数据库中存储的密码校验只存储了SHA256格式哈希，而开源客户端只识别MD5校验，双方校验方法不匹配报错。

说明

- 数据库并不存储用户密码，只存储用户密码的哈希码。
- 当用户更新用户密码或者新建用户时，数据库会同时存储两种格式的哈希码，这时将兼容开源的认证协议。
- 当旧版本升级到新版本时，由于哈希的不可逆性，所以数据库无法还原用户密码，进而生成新格式的哈希，所以仍然只保留了SHA256格式的哈希，导致仍然无法使用MD5做密码认证。
- MD5加密算法安全性低，存在安全风险，建议使用更安全的加密算法。

要解决该问题，可以更新用户密码（请参见**ALTER USER**），或者新建一个用户（请参见**CREATE USER**），赋予同等权限，使用新用户连接数据库。

- unsupported frontend protocol 3.51: server supports 1.0 to 3.0

  目标数据库版本过低，或者目标数据库为开源数据库。请使用对应版本的数据库驱动连接目标数据库。

- FATAL: GSS authentication method is not allowed because XXXX user password is not disabled.

  目标DN的pg_hba.conf里配置了当前客户端IP使用"gss"方式来做认证，该认证算法不支持用作客户端的身份认证，请修改到"sha256"后再试。

5.5.2.2.3 连接数据库涉及的 API

在测试连接数据库成功后，ODBC API提供了一组函数来连接数据库，如**表5-38**所示。

表 5-38 相关 API 说明

功能	API
申请句柄资源	**SQLAllocHandle**：申请句柄资源，可替代如下函数： • **SQLAllocEnv**：申请环境句柄 • **SQLAllocConnect**：申请连接句柄 • **SQLAllocStmt**：申请语句句柄
设置环境属性	**SQLSetEnvAttr**
设置连接属性	**SQLSetConnectAttr**
设置语句属性	**SQLSetStmtAttr**

以开发源程序DBtest.c为例（完整示例请参考**获取和处理数据库中的数据**）：

```
// DBtest.c (compile with: libodbc.so)
// 程序头文件和全局变量请参考完整示例

// 申请环境句柄。
V_OD_erg = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&V_OD_Env);
if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
{
   printf("Error AllocHandle\n");
   exit(0);
```

```
}

// 设置版本信息（环境属性）。
SQLSetEnvAttr(V_OD_Env, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0);

// 申请连接句柄 。
V_OD_erg = SQLAllocHandle(SQL_HANDLE_DBC, V_OD_Env, &V_OD_hdbc);
if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
{
   SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
   exit(0);
}

// 获取用户名和用户密码。
char *userName;
userName = getenv("EXAMPLE_USERNAME_ENV");
char *password;
password = getenv("EXAMPLE_PASSWORD_ENV");

// 设置连接属性。
SQLSetConnectAttr(V_OD_hdbc, SQL_ATTR_AUTOCOMMIT,(SQLPOINTER *)SQL_AUTOCOMMIT_ON, 0);

// 连接数据库，这里的userName与password分别表示连接数据库的用户名和用户密码。
// 如果odbc.ini文件中已经配置了用户名密码，那么这里可以留空（""）；但是不建议这么做，因为一旦odbc.ini权限管理不善，将导致数据库用户密码泄露。
V_OD_erg = SQLConnect(V_OD_hdbc, (SQLCHAR*) "gaussdb", SQL_NTS,
                      (SQLCHAR*) userName, SQL_NTS,  (SQLCHAR*) password, SQL_NTS);
if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
{
  printf("Error SQLConnect %d\n",V_OD_erg);
  SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
  exit(0);
}
printf("Connected !\n");
```

5.5.2.3 执行 SQL 语句

为了帮助用户实现与数据库的交互，ODBC提供执行SQL语句的相关API如表5-39所示。

表 5-39 相关 API 说明

功能	API
设置语句属性	SQLSetStmtAttr
为执行SQL语句做准备	SQLPrepare
执行一条准备好的SQL语句	SQLExecute
绑定SQL语句的参数标志和缓冲区	SQLBindParameter
直接执行SQL语句	SQLExecDirect

说明

- ODBC为应用程序与数据库的中心层，负责把应用程序发出的SQL指令传到数据库当中，自身并不解析SQL语法。故在应用程序中写入带有保密信息的SQL语句时（如明文密码），保密信息会被暴露在驱动日志中。
- 数据库中收到的一次执行请求（不在事务块中），如果含有多条语句，将会被打包成一个事务，如果其中有一个语句失败，那么整个请求都将会被回滚。

示例如下（完整示例请参考**获取和处理数据库中的数据**）：

```
// 设置语句属性。
SQLSetStmtAttr(V_OD_hstmt, SQL_ATTR_QUERY_TIMEOUT, (SQLPOINTER *)3, 0);

// 申请语句句柄。
SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);

// 直接执行SQL语句。
SQLExecDirect(V_OD_hstmt, "drop table IF EXISTS customer_t1", SQL_NTS);
SQLExecDirect(V_OD_hstmt, "CREATE TABLE customer_t1(c_customer_sk INTEGER, c_customer_name
VARCHAR(32));", SQL_NTS);
SQLExecDirect(V_OD_hstmt, "insert into customer_t1 values(25, 'li')", SQL_NTS);

// 准备执行。
SQLPrepare(V_OD_hstmt, "insert into customer_t1 values(?)", SQL_NTS);

// 绑定参数。
SQLBindParameter(V_OD_hstmt,1, SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER, 0, 0,
            &value, 0, NULL);

// 执行准备好的语句。
SQLExecute(V_OD_hstmt);      SQLExecDirect(V_OD_hstmt, "select c_customer_sk from customer_t1",
SQL_NTS);
```

5.5.2.4 处理结果集

ODBC处理结果集是从数据库中获取数据并将其提供给应用程序进行处理，作用包括但不限于：检索数据、数据展示、数据处理、数据传输和业务逻辑实现等。

ODBC提供处理结果集的相关API如**表5-40**所示。

表 5-40 相关 API 说明

功能	API
绑定缓冲区到结果集的列中	**SQLBindCol**
结果集中取行集	**SQLFetch**
返回结果集中某一列的数据	**SQLGetData**
获取结果集中列的描述信息	**SQLColAttribute**
查看最近一次操作错误信息	**SQLGetDiagRec**

示例如下（完整示例请参考**获取和处理数据库中的数据**）：

```
// 在执行完成SQL语句后，获取结果集某一列的属性。
SQLColAttribute(V_OD_hstmt,1,SQL_DESC_TYPE,typename,100,NULL,NULL);
printf("SQLColAtrribute %s\n",typename);

// 绑定结果集。
SQLBindCol(V_OD_hstmt,1,SQL_C_SLONG, (SQLPOINTER)&V_OD_buffer,150,
```

```
      (SQLLEN *)&V_OD_err);

// 通过SQLFetch取结果集中数据。
V_OD_erg=SQLFetch(V_OD_hstmt);

// 通过SQLGetData获取并返回数据。
while(V_OD_erg != SQL_NO_DATA)
{
  SQLGetData(V_OD_hstmt,1,SQL_C_SLONG,(SQLPOINTER)&V_OD_id,0,NULL);
  printf("SQLGetData ----ID = %d\n",V_OD_id);
  V_OD_erg=SQLFetch(V_OD_hstmt);
};
printf("Done !\n");
```

5.5.2.5 关闭连接

ODBC关闭数据库连接，释放资源相关API如表5-41所示。

表 5-41 相关 API 说明

功能	API
断开与数据源的连接	SQLDisconnect
释放句柄资源	SQLFreeHandle：释放句柄资源，可替代如下函数： • SQLFreeEnv：释放环境句柄 • SQLFreeConnect：释放连接句柄 • SQLFreeStmt：释放语句句柄

示例如下（完整示例请参考获取和处理数据库中的数据）：

```
// 断开数据源连接并释放句柄资源。
SQLFreeHandle(SQL_HANDLE_STMT,V_OD_hstmt);
SQLDisconnect(V_OD_hdbc);
SQLFreeHandle(SQL_HANDLE_DBC,V_OD_hdbc);
SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
```

5.5.3 典型应用开发示例

5.5.3.1 典型应用场景配置

日志诊断场景

ODBC日志分为unixODBC驱动管理器日志和psqlODBC驱动端日志。前者可以用于追溯应用程序API的执行是否成功，后者是底层实现过程中的一些DFX日志，用来帮助定位问题。

unixODBC日志需要在odbcinst.ini文件中配置：

```
[ODBC]
Trace=Yes
TraceFile=/path/to/odbctrace.log

[GaussMPP]
```



```
Driver64=/usr/local/lib/psqlodbcw.so
setup=/usr/local/lib/psqlodbcw.so
```

gsqlODBC日志只需要在odbc.ini加上如下内容：

```
[gaussdb]
Driver=GaussMPP
Servername=10.10.0.13 #数据库Server IP
...
Debug=1 #打开驱动端debug日志
```

说明

unixODBC日志将会生成在TraceFile配置的路径下，psqlODBC会在系统/tmp/下生成mylog_xxx.log。

主备切换自动寻主

数据库实例配备一主多备DN时，将所有DN的IP全部写入配置文件中，ODBC将会自动寻找主DN建立连接。当发生主备切换时，ODBC也可与新的主DN建立连接。

ODBC 相关约束说明

- ODBC不支持备机读。
- ODBC不支持自定义类型，不支持在存储过程中使用自定义类型参数。

5.5.3.2 获取和处理数据库中的数据

说明

- Windows环境下ODBC应用代码可以使用MinGW（Minimalist GNU for Windows）编译器进行编译。编译命令如下：
  ```
  gcc  odbctest.c -o odbctest -lodbc32
  ```
  执行命令为：
  ```
  ./odbctest.exe
  ```
- 在Linux环境下ODBC应用代码可以使用GCC（GNU Compiler Collection）编译器进行编译。编译命令如下：
  ```
  gcc odbctest.c -o odbctest -lodbc
  ```
  执行命令为：
  ```
  ./odbctest
  ```
  如果编译找不到sql.h或者API接口， 尝试手动连接unixodbc的头文件和动态库，即：
  ```
  gcc -I /home/omm/unixodbc/include -L /home/omm/unixodbc/lib odbctest.c -o odbctest -lodbc
  ```

此示例完整演示如何通过ODBC获取和处理GaussDB中的数据。

前提条件：数据源已配置成功。Linux系统请参考**Linux下配置数据源**；Windows系统请参考**Windows下配置数据源**。

```
// DBtest.c (compile with: libodbc.so)
// 本示例以用户名和密码保存在环境变量中为例，运行此示例前请先在本地环境中设置环境变量（环境变量名称请根据需求进行设置）EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
#ifdef WIN32
#include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
SQLHENV      V_OD_Env;      // Handle ODBC environment
SQLHSTMT     V_OD_hstmt;    // Handle statement
SQLHDBC      V_OD_hdbc;     // Handle connection
```

```
char          typename[100];
SQLINTEGER    value = 100;
SQLINTEGER    V_OD_erg,V_OD_buffer,V_OD_err,V_OD_id;
int main(int argc,char *argv[])
{
    // 1. 申请环境句柄。
    V_OD_erg = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&V_OD_Env);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
    {
        printf("Error AllocHandle\n");
        exit(0);
    }

    // 2. 设置版本信息（环境属性）。
    SQLSetEnvAttr(V_OD_Env, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0);

    // 3. 申请连接句柄。
    V_OD_erg = SQLAllocHandle(SQL_HANDLE_DBC, V_OD_Env, &V_OD_hdbc);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
    {
        SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
        exit(0);
    }

    // 获取用户名和用户密码。
    char *userName;
    userName = getenv("EXAMPLE_USERNAME_ENV");
    char *password;
    password = getenv("EXAMPLE_PASSWORD_ENV");

    // 4. 设置连接属性。
    SQLSetConnectAttr(V_OD_hdbc, SQL_ATTR_AUTOCOMMIT,(SQLPOINTER *)SQL_AUTOCOMMIT_ON, 0);

    // 5. 连接数据库，这里的userName与password分别表示连接数据库的用户名和用户密码。
    // 如果odbc.ini文件中已经配置了用户名密码，那么这里可以留空（""）；但是不建议这么做，因为一旦
odbc.ini权限管理不善，将导致数据库用户密码泄露。
    V_OD_erg = SQLConnect(V_OD_hdbc, (SQLCHAR*) "gaussdb", SQL_NTS,
                          (SQLCHAR*) userName, SQL_NTS,  (SQLCHAR*) password, SQL_NTS);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
    {
        printf("Error SQLConnect %d\n",V_OD_erg);
        SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
        exit(0);
    }
    printf("Connected !\n");

    // 6. 设置语句属性。
    SQLSetStmtAttr(V_OD_hstmt, SQL_ATTR_QUERY_TIMEOUT, (SQLPOINTER *)3,0);

    // 7. 申请语句句柄。
    SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);

    // 8. 直接执行SQL语句。
    SQLExecDirect(V_OD_hstmt, "drop table IF EXISTS customer_t1", SQL_NTS);
    SQLExecDirect(V_OD_hstmt, "CREATE TABLE customer_t1(c_customer_sk INTEGER, c_customer_name
VARCHAR(32));", SQL_NTS);
    SQLExecDirect(V_OD_hstmt, "insert into customer_t1 values(25,'li')", SQL_NTS);

    // 9. 准备执行。
    SQLPrepare(V_OD_hstmt,"insert into customer_t1 values(?)",SQL_NTS);

    // 10. 绑定参数。
    SQLBindParameter(V_OD_hstmt,1,SQL_PARAM_INPUT,SQL_C_SLONG,SQL_INTEGER,0,0,
                     &value,0,NULL);

    // 11. 执行准备好的语句。
    SQLExecute(V_OD_hstmt);
    SQLExecDirect(V_OD_hstmt,"select c_customer_sk from customer_t1",SQL_NTS);
```



```
    // 12. 获取结果集某一列的属性。
    SQLColAttribute(V_OD_hstmt,1, SQL_DESC_TYPE,typename,100, NULL, NULL);
    printf("SQLColAtrribute %s\n",typename);

    // 13. 绑定结果集。
    SQLBindCol(V_OD_hstmt,1,SQL_C_SLONG, (SQLPOINTER)&V_OD_buffer,150,
            (SQLLEN *)&V_OD_err);

    // 14. 通过SQLFetch取结果集中数据。
    V_OD_erg=SQLFetch(V_OD_hstmt);

    // 15. 通过SQLGetData获取并返回数据。
    while(V_OD_erg != SQL_NO_DATA)
    {
        SQLGetData(V_OD_hstmt,1,SQL_C_SLONG,(SQLPOINTER)&V_OD_id,0,NULL);
        printf("SQLGetData ----ID = %d\n",V_OD_id);
        V_OD_erg=SQLFetch(V_OD_hstmt);
    };
    printf("Done !\n");

    // 16. 断开数据源连接并释放句柄资源。
    SQLFreeHandle(SQL_HANDLE_STMT,V_OD_hstmt);
    SQLDisconnect(V_OD_hdbc);
    SQLFreeHandle(SQL_HANDLE_DBC,V_OD_hdbc);
    SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
    return(0);
}
```

运行结果如下：

```
Connected !
SQLColAtrribute
SQLGetData ----ID = 25
SQLGetData ----ID = 100
Done!
```

5.5.3.3 批量绑定

前提条件：数据源已配置成功。Linux系统请参考**Linux下配置数据源**；Windows系统请参考**Windows下配置数据源**。

```
/*************************************************************************
* 请在数据源中打开UseBatchProtocol，同时指定数据库中参数support_batch_bind为on
* CHECK_ERROR的作用是检查并打印错误信息。
* 此示例将与用户交互式获取DSN、模拟的数据量，忽略的数据量，并将最终数据入库到test_odbc_batch_insert
中。
*************************************************************************/
#ifdef WIN32
#include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
#include <string.h>
void Exec(SQLHDBC hdbc, SQLCHAR* sql)
{
    SQLRETURN retcode;                  // Return status
    SQLHSTMT hstmt = SQL_NULL_HSTMT;    // Statement handle
    SQLCHAR     loginfo[2048];

    // 分配语句句柄。
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLAllocHandle(SQL_HANDLE_STMT) failed");
        return;
    }
```



```
    // 准备语句。
    retcode = SQLPrepare(hstmt, (SQLCHAR*) sql, SQL_NTS);
    sprintf((char*)loginfo, "SQLPrepare log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLPrepare(hstmt, (SQLCHAR*) sql, SQL_NTS) failed");
        return;
    }

    // 执行语句。
    retcode = SQLExecute(hstmt);
    sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLExecute(hstmt) failed");
        return;
    }

    // 释放句柄。
    retcode = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
    sprintf((char*)loginfo, "SQLFreeHandle stmt log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLFreeHandle(SQL_HANDLE_STMT, hstmt) failed");
        return;
    }
}
int main ()
{
    SQLHENV  henv  = SQL_NULL_HENV;
    SQLHDBC  hdbc  = SQL_NULL_HDBC;
    long     int      batchCount = 1000; // 批量绑定的数据量
    SQLLEN   rowsCount = 0;
    int      ignoreCount = 0;           // 批量绑定的数据中，不要入库的数据量
    int      i = 0;
    SQLRETURN   retcode;
    SQLCHAR     dsn[1024] = {'\0'};
    SQLCHAR     loginfo[2048];
    do
    {
        if (ignoreCount > batchCount)
        {
            printf("ignoreCount(%d) should be less than batchCount(%d)\n", ignoreCount, batchCount);
        }
    }while(ignoreCount > batchCount);

    // 分配环境句柄。
    retcode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLAllocHandle failed");
        goto exit;
    }

    // 设置ODBC版本。
    retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
                                    (SQLPOINTER*)SQL_OV_ODBC3, 0);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLSetEnvAttr failed");
        goto exit;
    }

    // 分配连接。
    retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLAllocHandle failed");
        goto exit;
```



```
    }

    // 设置登录超时。
    retcode = SQLSetConnectAttr(hdbc, SQL_LOGIN_TIMEOUT, (SQLPOINTER)5, 0);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLSetConnectAttr failed");
        goto exit;
    }

    // 设置自动提交。
    retcode = SQLSetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT,
                                    (SQLPOINTER)(1), 0);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLSetConnectAttr failed");
        goto exit;
    }

    // 连接到数据库。
    sprintf(loginfo, "SQLConnect(DSN:%s)", dsn);
    retcode = SQLConnect(hdbc, (SQLCHAR*) "gaussdb", SQL_NTS,
                            (SQLCHAR*) NULL, 0, NULL, 0);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLConnect failed");
        goto exit;
    }

    // 初始化表信息。
    Exec(hdbc, "drop table if exists test_odbc_batch_insert");
    Exec(hdbc, "create table test_odbc_batch_insert(id int primary key, col varchar2(50))");
    // 下面的代码根据用户输入的数据量，构造出将要入库的数据:
    {
        SQLRETURN retcode;
        SQLHSTMT hstmtinesrt = SQL_NULL_HSTMT;
        SQLCHAR      *sql = NULL;
        SQLINTEGER   *ids  = NULL;
        SQLCHAR      *cols = NULL;
        SQLLEN       *bufLenIds = NULL;
        SQLLEN       *bufLenCols = NULL;
        SQLUSMALLINT *operptr = NULL;
        SQLUSMALLINT *statusptr = NULL;
        SQLULEN      process = 0;

        // 这里是按列构造，每个字段的内存连续存放在一起。
        ids = (SQLINTEGER*)malloc(sizeof(ids[0]) * batchCount);
        cols = (SQLCHAR*)malloc(sizeof(cols[0]) * batchCount * 50);

        // 这里是每个字段中，每一行数据的内存长度。
        bufLenIds = (SQLLEN*)malloc(sizeof(bufLenIds[0]) * batchCount);
        bufLenCols = (SQLLEN*)malloc(sizeof(bufLenCols[0]) * batchCount);

        // 该行是否需要被处理，SQL_PARAM_IGNORE 或 SQL_PARAM_PROCEED
        operptr = (SQLUSMALLINT*)malloc(sizeof(operptr[0]) * batchCount);
        memset(operptr, 0, sizeof(operptr[0]) * batchCount);

        // 该行的处理结果。
        // 注：由于数据库中处理方式是同一语句隶属同一事务中，所以如果出错，那么待处理数据都将是出错的，并不会部分入库。
        statusptr = (SQLUSMALLINT*)malloc(sizeof(statusptr[0]) * batchCount);
        memset(statusptr, 88, sizeof(statusptr[0]) * batchCount);
        if (NULL == ids || NULL == cols || NULL == bufLenCols || NULL == bufLenIds)
        {
            fprintf(stderr, "FAILED:\tmalloc data memory failed\n");
            goto exit;
        }
        for (i = 0; i < batchCount; i++)
        {
```



```
        ids[i] = i;
        sprintf(cols + 50 * i, "column test value %d", i);
        bufLenIds[i] = sizeof(ids[i]);
        bufLenCols[i] = strlen(cols + 50 * i);
        operptr[i] = (i < ignoreCount) ? SQL_PARAM_IGNORE : SQL_PARAM_PROCEED;
    }

    // 分配语句句柄。
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmtinesrt);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLAllocHandle failed");
        goto exit;
    }

    // 准备语句。
    sql = (SQLCHAR*)"insert into test_odbc_batch_insert values(?, ?)";
    retcode = SQLPrepare(hstmtinesrt, (SQLCHAR*) sql, SQL_NTS);
    sprintf((char*)loginfo, "SQLPrepare log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLPrepare failed");
        goto exit;
    }
    retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAMSET_SIZE, (SQLPOINTER)batchCount,
sizeof(batchCount));

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLSetStmtAttr failed");
        goto exit;
    }
    retcode = SQLBindParameter(hstmtinesrt, 1, SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER,
sizeof(ids[0]), 0,&(ids[0]), 0, bufLenIds);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLBindParameter failed");
        goto exit;
    }
    retcode = SQLBindParameter(hstmtinesrt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 50, 50,
cols, 50, bufLenCols);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLBindParameter failed");
        goto exit;
    }
    retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAMS_PROCESSED_PTR, (SQLPOINTER)&process,
sizeof(process));

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLSetStmtAttr failed");
        goto exit;
    }
    retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAM_STATUS_PTR, (SQLPOINTER)statusptr,
sizeof(statusptr[0]) * batchCount);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLSetStmtAttr failed");
        goto exit;
    }
    retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAM_OPERATION_PTR, (SQLPOINTER)operptr,
sizeof(operptr[0]) * batchCount);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLSetStmtAttr failed");
        goto exit;
    }
    retcode = SQLExecute(hstmtinesrt);
    sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);
```



```
    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLExecute(hstmtinesrt) failed");
        goto exit;
        retcode = SQLRowCount(hstmtinesrt, &rowsCount);

        if (!SQL_SUCCEEDED(retcode)) {
            printf("SQLRowCount failed");
            goto exit;
        }
        if (rowsCount != (batchCount - ignoreCount))
        {
            sprintf(loginfo, "(batchCount - ignoreCount)(%d) != rowsCount(%d)", (batchCount -
ignoreCount), rowsCount);

            if (!SQL_SUCCEEDED(retcode)) {
                printf("SQLExecute failed");
                goto exit;
            }
        }
        else
        {
            sprintf(loginfo, "(batchCount - ignoreCount)(%d) == rowsCount(%d)", (batchCount -
ignoreCount), rowsCount);

            if (!SQL_SUCCEEDED(retcode)) {
                printf("SQLExecute failed");
                goto exit;
            }
        }

        // 检查返回的行数。
        if (rowsCount != process)
        {
            sprintf(loginfo, "process(%d) != rowsCount(%d)", process, rowsCount);

            if (!SQL_SUCCEEDED(retcode)) {
                printf("SQLExecute failed");
                goto exit;
            }
        }
        else
        {
            sprintf(loginfo, "process(%d) == rowsCount(%d)", process, rowsCount);

            if (!SQL_SUCCEEDED(retcode)) {
                printf("SQLExecute failed");
                goto exit;
            }
        }
        for (i = 0; i < batchCount; i++)
        {
            if (i < ignoreCount)
            {
                if (statusptr[i] != SQL_PARAM_UNUSED)
                {
                    sprintf(loginfo, "statusptr[%d](%d) != SQL_PARAM_UNUSED", i, statusptr[i]);

                    if (!SQL_SUCCEEDED(retcode)) {
                        printf("SQLExecute failed");
                        goto exit;
                    }
                }
            }
            else if (statusptr[i] != SQL_PARAM_SUCCESS)
            {
                sprintf(loginfo, "statusptr[%d](%d) != SQL_PARAM_SUCCESS", i, statusptr[i]);

                if (!SQL_SUCCEEDED(retcode)) {
                    printf("SQLExecute failed");
```

```
                    goto exit;
                }
            }
        }
        retcode = SQLFreeHandle(SQL_HANDLE_STMT, hstmtinesrt);
        sprintf((char*)loginfo, "SQLFreeHandle hstmtinesrt");

        if (!SQL_SUCCEEDED(retcode)) {
            printf("SQLFreeHandle failed");
            goto exit;
        }
    }
  }
exit:
   (void) printf ("\nComplete.\n");

   // 断开连接。
   if (hdbc != SQL_NULL_HDBC) {
      SQLDisconnect(hdbc);
      SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
   }
   // 释放环境句柄。
   if (henv != SQL_NULL_HENV)
      SQLFreeHandle(SQL_HANDLE_ENV, henv);
   return 0;
}
```

运行结果如下：

```
Complete.
```

数据库中查询后显示部分结果如下：

```
 id |          col
----+-----------------------
  0 | column test value 0
  1 | column test value 1
  2 | column test value 2
  3 | column test value 3
  4 | column test value 4
  5 | column test value 5
  6 | column test value 6
  7 | column test value 7
  8 | column test value 8
.....
```

5.5.3.4 高性能绑定类型

需要进行大量数据插入时，有如下建议供用户参考和设置：

- 需要设置批量绑定：odbc.ini配置文件中设置UseBatchProtocol=1、数据库设置support_batch_bind=on。
- ODBC程序绑定类型要和数据库中类型一致。
- 客户端字符集和数据库字符集一致。
- 事务改成手动提交模式。

odbc.ini配置文件：

```
[gaussdb]
Driver=GaussMPP
Servername=10.10.0.13 #数据库Server IP
...
UseBatchProtocol=1 #默认打开
ConnSettings=set client_encoding=UTF8 #设置客户端字符编码，保证和server端一致
```

绑定类型示例：

前提条件：数据源已配置成功。Linux系统请参考**Linux下配置数据源**；Windows系统请参考**Windows下配置数据源**。

```
#ifdef WIN32
#include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
#include <string.h>
#include <sys/time.h>

#define MESSAGE_BUFFER_LEN  128
SQLHANDLE   h_env = NULL;
SQLHANDLE   h_conn = NULL;
SQLHANDLE   h_stmt = NULL;
void print_error()
{
    SQLCHAR     Sqlstate[SQL_SQLSTATE_SIZE+1];
    SQLINTEGER  NativeError;
    SQLCHAR     MessageText[MESSAGE_BUFFER_LEN];
    SQLSMALLINT TextLength;
    SQLRETURN   ret = SQL_ERROR;

    ret = SQLGetDiagRec(SQL_HANDLE_STMT, h_stmt, 1, Sqlstate, &NativeError, MessageText,
MESSAGE_BUFFER_LEN, &TextLength);
    if ( SQL_SUCCESS == ret)
    {
        printf("\n STMT ERROR-%05d %s", NativeError, MessageText);
        return;
    }

    ret = SQLGetDiagRec(SQL_HANDLE_DBC, h_conn, 1, Sqlstate, &NativeError, MessageText,
MESSAGE_BUFFER_LEN, &TextLength);
    if ( SQL_SUCCESS == ret)
    {
        printf("\n CONN ERROR-%05d %s", NativeError, MessageText);
        return;
    }

    ret = SQLGetDiagRec(SQL_HANDLE_ENV, h_env, 1, Sqlstate, &NativeError, MessageText,
MESSAGE_BUFFER_LEN, &TextLength);
    if ( SQL_SUCCESS == ret)
    {
        printf("\n ENV ERROR-%05d %s", NativeError, MessageText);
        return;
    }

    return;
}

/* 期盼函数返回SQL_SUCCESS */
#define RETURN_IF_NOT_SUCCESS(func) \
{\
    SQLRETURN ret_value = (func);\
    if (SQL_SUCCESS != ret_value)\
    {\
        print_error();\
        printf("\n failed line = %u: expect SQL_SUCCESS, but ret = %d", __LINE__, ret_value);\
        return SQL_ERROR; \
    }\
}

/* 期盼函数返回SQL_SUCCESS */
#define RETURN_IF_NOT_SUCCESS_I(i, func) \
{\
    SQLRETURN ret_value = (func);\
    if (SQL_SUCCESS != ret_value)\
```



```
    {\
        print_error();\
        printf("\n failed line = %u (i=%d): : expect SQL_SUCCESS, but ret = %d", __LINE__, (i), ret_value);\
        return SQL_ERROR; \
    }\
}

/* 期盼函数返回SQL_SUCCESS_WITH_INFO */
#define RETURN_IF_NOT_SUCCESS_INFO(func) \
{\
    SQLRETURN ret_value = (func);\
    if (SQL_SUCCESS_WITH_INFO != ret_value)\
    {\
        print_error();\
        printf("\n failed line = %u: expect SQL_SUCCESS_WITH_INFO, but ret = %d", __LINE__, ret_value);\
        return SQL_ERROR; \
    }\
}

/* 期盼数值相等 */
#define RETURN_IF_NOT(expect, value) \
if ((expect) != (value))\
{\
    printf("\n failed line = %u: expect = %u, but value = %u", __LINE__, (expect), (value)); \
    return SQL_ERROR;\
}

/* 期盼字符串相同 */
#define RETURN_IF_NOT_STRCMP_I(i, expect, value) \
if (( NULL == (expect) ) || (NULL == (value)))\
{\
    printf("\n failed line = %u (i=%u): input NULL pointer !", __LINE__, (i)); \
    return SQL_ERROR; \
}\
else if (0 != strcmp((expect), (value)))\
{\
    printf("\n failed line = %u (i=%u): expect = %s, but value = %s", __LINE__, (i), (expect), (value)); \
    return SQL_ERROR;\
}


/* prepare + execute SQL语句 */
int execute_cmd(SQLCHAR *sql)
{
    if ( NULL == sql )
    {
        return SQL_ERROR;
    }

    if ( SQL_SUCCESS != SQLPrepare(h_stmt, sql, SQL_NTS))
    {
        return SQL_ERROR;
    }

    if ( SQL_SUCCESS != SQLExecute(h_stmt))
    {
        return SQL_ERROR;
    }

    return SQL_SUCCESS;
}
/* execute + commit 句柄 */
int commit_exec()
{
    if ( SQL_SUCCESS != SQLExecute(h_stmt))
    {
        return SQL_ERROR;
    }
```



```
    /* 手动提交 */
    if ( SQL_SUCCESS != SQLEndTran(SQL_HANDLE_DBC, h_conn, SQL_COMMIT))
    {
        return SQL_ERROR;
    }

    return SQL_SUCCESS;
}

int begin_unit_test()
{
    SQLINTEGER    ret;

    /* 申请环境句柄 */
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &h_env);
    if ((SQL_SUCCESS != ret) && (SQL_SUCCESS_WITH_INFO != ret))
    {
        printf("\n begin_unit_test::SQLAllocHandle SQL_HANDLE_ENV failed ! ret = %d", ret);
        return SQL_ERROR;
    }

    /* 进行连接前必须要先设置版本号 */
    if (SQL_SUCCESS != SQLSetEnvAttr(h_env, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)SQL_OV_ODBC3,
0))
    {
        print_error();
        printf("\n begin_unit_test::SQLSetEnvAttr SQL_ATTR_ODBC_VERSION failed ! ret = %d", ret);
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
        return SQL_ERROR;
    }

    /* 申请连接句柄 */
    ret = SQLAllocHandle(SQL_HANDLE_DBC, h_env, &h_conn);
    if (SQL_SUCCESS != ret)
    {
        print_error();
        printf("\n begin_unit_test::SQLAllocHandle SQL_HANDLE_DBC failed ! ret = %d", ret);
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
        return SQL_ERROR;
    }

    /* 建立连接 */
    ret = SQLConnect(h_conn, (SQLCHAR*) "gaussdb", SQL_NTS,
                            (SQLCHAR*) NULL, 0, NULL, 0);
    if (SQL_SUCCESS != ret)
    {
        print_error();
        printf("\n begin_unit_test::SQLConnect  failed ! ret = %d", ret);
        SQLFreeHandle(SQL_HANDLE_DBC, h_conn);
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
        return SQL_ERROR;
    }

    /* 申请语句句柄 */
    ret = SQLAllocHandle(SQL_HANDLE_STMT, h_conn, &h_stmt);
    if (SQL_SUCCESS != ret)
    {
        print_error();
        printf("\n begin_unit_test::SQLAllocHandle SQL_HANDLE_STMT failed ! ret = %d", ret);
        SQLFreeHandle(SQL_HANDLE_DBC, h_conn);
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
        return SQL_ERROR;
    }

    return SQL_SUCCESS;
}

void end_unit_test()
{
```



```
    /* 释放语句句柄 */
    if (NULL != h_stmt)
    {
        SQLFreeHandle(SQL_HANDLE_STMT, h_stmt);
    }

    /* 释放连接句柄 */
    if (NULL != h_conn)
    {
        SQLDisconnect(h_conn);
        SQLFreeHandle(SQL_HANDLE_DBC, h_conn);
    }

    /* 释放环境句柄 */
    if (NULL != h_env)
    {
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
    }

    return;
}

int main()
{
    /* begin test */
    if (begin_unit_test() != SQL_SUCCESS)
    {
        printf("\n begin_test_unit failed.");
        return SQL_ERROR;
    }
    /* 句柄配置同前面用例 */
    int         i = 0;
    SQLCHAR*    sql_drop   = "drop table if exists test_bindnumber_001";
    SQLCHAR*    sql_create = "create table test_bindnumber_001("
                             "f4 number, f5 number(10, 2)"
                             ")";
    SQLCHAR*    sql_insert = "insert into test_bindnumber_001 values(?, ?)";
    SQLCHAR*    sql_select = "select * from test_bindnumber_001";
    SQLLEN      RowCount;
    SQL_NUMERIC_STRUCT st_number;
    SQLCHAR     getValue[2][MESSAGE_BUFFER_LEN];

    /* step 1. 建表 */
    RETURN_IF_NOT_SUCCESS(execute_cmd(sql_drop));
    RETURN_IF_NOT_SUCCESS(execute_cmd(sql_create));

    /* step 2.1 通过SQL_NUMERIC_STRUCT结构绑定参数 */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));

    /* 第一行: 1234.5678 */
    memset(st_number.val, 0, SQL_MAX_NUMERIC_LEN);
    st_number.precision = 8;
    st_number.scale = 4;
    st_number.sign = 1;
    st_number.val[0] = 0x4E;
    st_number.val[1] = 0x61;
    st_number.val[2] = 0xBC;

    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0, NULL));

    /* 关闭自动提交 */
    SQLSetConnectAttr(h_conn, SQL_ATTR_AUTOCOMMIT, (SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0);

    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);
```



```
    /* 第二行: 12345678 */
    memset(st_number.val, 0, SQL_MAX_NUMERIC_LEN);
    st_number.precision = 8;
    st_number.scale = 0;
    st_number.sign = 1;
    st_number.val[0] = 0x4E;
    st_number.val[1] = 0x61;
    st_number.val[2] = 0xBC;

    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 0, &st_number, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 0, &st_number, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* 第三行: 12345678 */
    memset(st_number.val, 0, SQL_MAX_NUMERIC_LEN);
    st_number.precision = 0;
    st_number.scale = 4;
    st_number.sign = 1;
    st_number.val[0] = 0x4E;
    st_number.val[1] = 0x61;
    st_number.val[2] = 0xBC;

    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);


    /* step 2.2 第四行通过SQL_C_CHAR字符串绑定参数 */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    SQLCHAR*    szNumber = "1234.5678";
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_NUMERIC, strlen(szNumber), 0, szNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_NUMERIC, strlen(szNumber), 0, szNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* step 2.3 第五行通过SQL_C_FLOAT绑定参数 */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    SQLREAL     fNumber = 1234.5678;
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_FLOAT,
SQL_NUMERIC, sizeof(fNumber), 4, &fNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_FLOAT,
SQL_NUMERIC, sizeof(fNumber), 4, &fNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* step 2.4 第六行通过SQL_C_DOUBLE绑定参数 */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    SQLDOUBLE    dNumber = 1234.5678;
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_DOUBLE,
SQL_NUMERIC, sizeof(dNumber), 4, &dNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_DOUBLE,
SQL_NUMERIC, sizeof(dNumber), 4, &dNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);
```



```
    SQLBIGINT   bNumber1 = 0xFFFFFFFFFFFFFFFF;
    SQLBIGINT   bNumber2 = 12345;

    /* step 2.5 第七行通过SQL_C_SBIGINT绑定参数 */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_SBIGINT,
SQL_NUMERIC, sizeof(bNumber1), 4, &bNumber1, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_SBIGINT,
SQL_NUMERIC, sizeof(bNumber2), 4, &bNumber2, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* step 2.6 第八行通过SQL_C_UBIGINT绑定参数 */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_UBIGINT,
SQL_NUMERIC, sizeof(bNumber1), 4, &bNumber1, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_UBIGINT,
SQL_NUMERIC, sizeof(bNumber2), 4, &bNumber2, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    SQLLEN  lNumber1 = 0xFFFFFFFFFFFFFFFF;
    SQLLEN  lNumber2 = 12345;

    /* step 2.7 第九行通过SQL_C_LONG绑定参数 */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_NUMERIC, sizeof(lNumber1), 0, &lNumber1, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_NUMERIC, sizeof(lNumber2), 0, &lNumber2, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* step 2.8 第十行通过SQL_C_ULONG绑定参数 */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_ULONG,
SQL_NUMERIC, sizeof(lNumber1), 0, &lNumber1, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_ULONG,
SQL_NUMERIC, sizeof(lNumber2), 0, &lNumber2, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    SQLSMALLINT sNumber = 0xFFFF;

    /* step 2.9 第十一行通过SQL_C_SHORT绑定参数 */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_SHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_SHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* step 2.10 第十二行通过SQL_C_USHORT绑定参数 */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_USHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_USHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    SQLCHAR cNumber = 0xFF;
```



```
    /* step 2.11 第十三行通过SQL_C_TINYINT绑定参数 */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_TINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_TINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* step 2.12 第十四行通过SQL_C_UTINYINT绑定参数 */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_UTINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_UTINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* 用字符串类型统一进行期盼 */
    SQLCHAR*    expectValue[14][2] = {{"1234.5678",             "1234.57"},
                                      {"12345678",              "12345678"},
                                      {"0",                     "0"},
                                      {"1234.5678",             "1234.57"},
                                      {"1234.5677",             "1234.57"},
                                      {"1234.5678",             "1234.57"},
                                      {"-1",                    "12345"},
                                      {"18446744073709551615",  "12345"},
                                      {"-1",                    "12345"},
                                      {"4294967295",            "12345"},
                                      {"-1",                    "-1"},
                                      {"65535",                 "65535"},
                                      {"-1",                    "-1"},
                                      {"255",                   "255"},
                                      };

    RETURN_IF_NOT_SUCCESS(execute_cmd(sql_select));
    while ( SQL_NO_DATA != SQLFetch(h_stmt))
    {
        RETURN_IF_NOT_SUCCESS_I(i, SQLGetData(h_stmt, 1, SQL_C_CHAR, &getValue[0],
MESSAGE_BUFFER_LEN, NULL));
        RETURN_IF_NOT_SUCCESS_I(i, SQLGetData(h_stmt, 2, SQL_C_CHAR, &getValue[1],
MESSAGE_BUFFER_LEN, NULL));
        i++;
    }
    printf("\nComplete!\n");
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(i, RowCount);
    SQLCloseCursor(h_stmt);
    /* step final. 删除表还原环境 */
    RETURN_IF_NOT_SUCCESS(execute_cmd(sql_drop));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    end_unit_test();
}
```

说明

上述用例中定义了number列，调用SQLBindParameter接口时，绑定SQL_NUMERIC会比SQL_LONG性能更高。因为如果是char，在数据库服务端插入数据时需要进行数据类型转换，从而引发性能瓶颈。

运行结果如下：

```
Complete!
```


5.5.4 ODBC 接口参考

ODBC接口是一套提供给用户的API函数，本节将对部分常用接口做具体描述，若涉及其他接口可参考msdn（网址：https://msdn.microsoft.com/en-us/library/windows/desktop/ms714177(v=vs.85).aspx）中ODBC Programmer's Reference项的相关内容。

5.5.4.1 SQLAllocEnv

在ODBC 3.x版本中，ODBC 2.x的函数SQLAllocEnv已被SQLAllocHandle代替。有关详细信息请参见**SQLAllocHandle**。

5.5.4.2 SQLAllocConnect

在ODBC 3.x版本中，ODBC 2.x的函数SQLAllocConnect已被SQLAllocHandle代替。有关详细信息请参见**SQLAllocHandle**。

5.5.4.3 SQLAllocHandle

功能描述

分配环境、连接、语句或描述符的句柄，它替代了ODBC 2.x函数SQLAllocEnv、SQLAllocConnect及SQLAllocStmt。

原型

```
SQLRETURN SQLAllocHandle(SQLSMALLINT   HandleType,
                         SQLHANDLE     InputHandle,
                         SQLHANDLE     *OutputHandlePtr);
```

参数

表 5-42 SQLAllocHandle 参数

关键字	参数说明
HandleType	由SQLAllocHandle分配的句柄类型。必须为下列值之一： • SQL_HANDLE_ENV（环境句柄） • SQL_HANDLE_DBC（连接句柄） • SQL_HANDLE_STMT（语句句柄） • SQL_HANDLE_DESC（描述句柄） 申请句柄顺序为，先申请环境句柄，再申请连接句柄，最后申请语句句柄。后申请的句柄依赖它前面申请的句柄。

关键字	参数说明
InputHandle	将要分配的新句柄的类型。InputHandle参数用于指定创建句柄的父句柄，以建立句柄之间的层次关系。不同类型的句柄有不同的层次关系，InputHandle参数用于指定这种关系。 • 如果HandleType为SQL_HANDLE_ENV，则该值为SQL_NULL_HANDLE，表示没有父句柄。 • 如果HandleType为SQL_HANDLE_DBC，则它一定是一个环境句柄，表示该连接句柄是在该环境下创建的。 • 如果HandleType为SQL_HANDLE_STMT或SQL_HANDLE_DESC，则它一定是一个连接句柄，表示该语句句柄是在该连接下创建的。
OutputHandlePtr	**输出参数**：OutputHandlePtr是一个指向SQLHANDLE类型的指针，用于返回分配的新句柄。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。

注意事项

当分配的句柄并非环境句柄时，如果SQLAllocHandle返回的值为SQL_ERROR，则它会将OutputHandlePtr的值设置为SQL_NULL_HDBC、SQL_NULL_HSTMT或SQL_NULL_HDESC。之后，通过调用带有适当参数的**SQLGetDiagRec**，其中HandleType和Handle被设置为IntputHandle的值，可得到相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

请参见：**典型应用开发示例**

5.5.4.4 SQLAllocStmt

在ODBC 3.x版本中，ODBC 2.x的函数SQLAllocStmt已被SQLAllocHandle代替。有关详细信息请参见**SQLAllocHandle**。

5.5.4.5 SQLBindCol

功能描述

将应用程序数据缓冲区绑定到结果集的列中。

原型

```
SQLRETURN SQLBindCol(SQLHSTMT       StatementHandle,
                     SQLUSMALLINT   ColumnNumber,
```

```
SQLSMALLINT    TargetType,
SQLPOINTER     TargetValuePtr,
SQLLEN     BufferLength,
SQLLEN     *StrLen_or_IndPtr);
```

参数

表 5-43 SQLBindCol 参数

关键字	参数说明
StatementHandle	语句句柄。
ColumnNumber	要绑定结果集的列号。起始列号为0，以递增的顺序计算列号，第0列是书签列。若未设置书签页，则起始列号为1。
TargetType	缓冲区中C数据类型的标识符。
TargetValuePtr	**输出参数**：指向与列绑定的数据缓冲区的指针。SQLFetch函数返回这个缓冲区中的数据。如果此参数为一个空指针，则StrLen_or_IndPtr是一个有效值。
BufferLength	TargetValuePtr指向缓冲区的长度，以字节为单位。
StrLen_or_IndPtr	**输出参数**：缓冲区的长度或指示器指针。若为空值，则未使用任何长度或指示器值。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。

注意事项

当SQLBindCol返回SQL_ERROR或SQL_SUCCESS_WITH_INFO时，通过调用**SQLGetDiagRec**函数，并将HandleType和Handle参数设置为SQL_HANDLE_STMT和StatementHandle，可得到一个相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

请参见：**典型应用开发示例**

5.5.4.6 SQLBindParameter

功能描述

将一条SQL语句中的一个参数标志和一个缓冲区绑定起来。

原型

```
SQLRETURN SQLBindParameter(SQLHSTMT       StatementHandle,
                           SQLUSMALLINT   ParameterNumber,
                           SQLSMALLINT    InputOutputType,
                           SQLSMALLINT    ValuetType,
                           SQLSMALLINT    ParameterType,
                           SQLULEN    ColumnSize,
                           SQLSMALLINT    DecimalDigits,
                           SQLPOINTER     ParameterValuePtr,
                           SQLLEN     BufferLength,
                           SQLLEN     *StrLen_or_IndPtr);
```

参数

表 5-44 SQLBindParameter

关键词	参数说明
StatementHandle	语句句柄。
ParameterNumber	参数序号，起始为1，依次递增。
InputOutputType	输入输出参数类型。可以是SQL_PARAM_INPUT、SQL_PARAM_OUTPUT、SQL_PARAM_INPUT_OUTPUT。
ValueType	参数的C数据类型。可以是SQL_C_CHAR、SQL_C_LONG、SQL_C_DOUBLE等。
ParameterType	参数的SQL数据类型。可以是SQL_CHAR、SQL_INTEGER、SQL_DOUBLE等。
ColumnSize	相应参数标记的列或表达式的大小。
DecimalDigits	相应参数标记的列或表达式的十进制数字。
ParameterValuePtr	指向存储参数数据缓冲区的指针。
BufferLength	ParameterValuePtr指向缓冲区的长度，以字节为单位。
StrLen_or_IndPtr	缓冲区的长度或指示器指针。若为空值，则未使用任何长度或指示器值。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。

注意事项

当SQLBindParameter返回SQL_ERROR或SQL_SUCCESS_WITH_INFO时，通过调用**SQLGetDiagRec**函数，并将HandleType和Handle参数设置为SQL_HANDLE_STMT和

StatementHandle，可得到一个相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

请参见：**典型应用开发示例**

5.5.4.7 SQLColAttribute

功能描述

返回结果集中一列的描述符信息。

原型

```
SQLRETURN SQLColAttibute(SQLHSTMT        StatementHandle,
                         SQLUSMALLINT    ColumnNumber,
                         SQLUSMALLINT    FieldIdentifier,
                         SQLPOINTER      CharacterAtrributePtr,
                         SQLSMALLINT     BufferLength,
                         SQLSMALLINT     *StringLengthPtr,
                         SQLLEN          *NumericAttributePtr);
```

参数

表 5-45 SQLColAttribute 参数

关键字	参数说明
StatementHandle	语句句柄。
ColumnNumber	要检索字段的列号，起始为1，依次递增。
FieldIdentifier	IRD中ColumnNumber行的字段。
CharacterAttributePtr	**输出参数**：一个缓冲区指针，返回FieldIdentifier字段值。
BufferLength	• 如果FieldIdentifier是一个ODBC定义的字段，而且CharacterAttributePtr指向一个字符串或二进制缓冲区，则此参数为该缓冲区的长度。 • 如果FieldIdentifier是一个ODBC定义的字段，而且CharacterAttributePtr指向一个整数，则会忽略该字段。
StringLengthPtr	**输出参数**：缓冲区指针，存放*CharacterAttributePtr中字符类型数据的字节总数，对于非字符类型，忽略BufferLength的值。
NumericAttributePtr	**输出参数**：指向一个整型缓冲区的指针，返回IRD中ColumnNumber行FieldIdentifier字段的值。

返回值

- SQL_SUCCESS：表示调用正确。

- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。

注意事项

当SQLColAttribute返回SQL_ERROR或SQL_SUCCESS_WITH_INFO时，通过调用**SQLGetDiagRec**函数，并将HandleType和Handle参数设置为SQL_HANDLE_STMT和StatementHandle，可得到一个相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

请参见：**典型应用开发示例**

5.5.4.8 SQLConnect

功能描述

在驱动程序和数据源之间建立连接。连接上数据源之后，可以通过连接句柄访问到所有有关连接数据源的信息，包括程序运行状态、事务处理状态和错误信息。

原型

```
SQLRETURN  SQLConnect(SQLHDBC        ConnectionHandle,
                      SQLCHAR        *ServerName,
                      SQLSMALLINT    NameLength1,
                      SQLCHAR        *UserName,
                      SQLSMALLINT    NameLength2,
                      SQLCHAR        *Authentication,
                      SQLSMALLINT    NameLength3);
```

参数

表 5-46 SQLConnect 参数

关键字	参数说明
ConnectionHandle	连接句柄，通过SQLAllocHandle获得。
ServerName	要连接数据源的名称。
NameLength1	ServerName的长度。
UserName	数据源中数据库用户名。
NameLength2	UserName的长度。
Authentication	数据源中数据库用户密码。
NameLength3	Authentication的长度。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。
- SQL_STILL_EXECUTING：表示语句正在执行。

注意事项

当调用SQLConnect函数返回SQL_ERROR或SQL_SUCCESS_WITH_INFO时，通过调用**SQLGetDiagRec**函数，并将HandleType和Handle参数设置为SQL_HANDLE_DBC和ConnectionHandle，可得到一个相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

请参见：**典型应用开发示例**

5.5.4.9 SQLDisconnect

功能描述

关闭一个与特定连接句柄相关的连接。

原型

```
SQLRETURN SQLDisconnect(SQLHDBC     ConnectionHandle);
```

参数

表 5-47 SQLDisconnect 参数

关键字	参数说明
ConnectionHandle	连接句柄，通过SQLAllocHandle获得。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。

注意事项

当调用SQLDisconnect函数返回SQL_ERROR或SQL_SUCCESS_WITH_INFO时，通过调用**SQLGetDiagRec**函数，并将HandleType和Handle参数设置为SQL_HANDLE_DBC和

ConnectionHandle，可得到一个相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

请参见：**典型应用开发示例**

5.5.4.10 SQLExecDirect

功能描述

使用参数的当前值，执行一条准备好的语句。对于一次只执行一条SQL语句，SQLExecDirect是最快的执行方式。

原型

```
SQLRETURN SQLExecDirect(SQLHSTMT         StatementHandle,
                        SQLCHAR          *StatementText,
                        SQLINTEGER       TextLength);
```

参数

表 5-48 SQLExecDirect 参数

关键字	参数说明
StatementHandle	语句句柄，通过SQLAllocHandle获得。
StatementText	要执行的SQL语句，不支持一次执行多条语句。
TextLength	StatementText的长度。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_NEED_DATA：表示在执行SQL语句前没有提供足够的参数。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。
- SQL_STILL_EXECUTING：表示语句正在执行。
- SQL_NO_DATA：表示SQL语句不返回结果集。

注意事项

当调用SQLExecDirect函数返回SQL_ERROR或SQL_SUCCESS_WITH_INFO时，通过调用**SQLGetDiagRec**函数，并将HandleType和Handle参数设置为SQL_HANDLE_STMT和StatementHandle，可得到一个相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

请参见：**典型应用开发示例**

5.5.4.11 SQLExecute

功能描述

如果语句中存在参数标记，SQLExecute函数使用参数标记参数的当前值，执行一条准备好的SQL语句。

原型

```
SQLRETURN SQLExecute(SQLHSTMT    StatementHandle);
```

参数

表 5-49 SQLExecute 参数

关键字	参数说明
StatementHandle	要执行语句的句柄。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_NEED_DATA：表示在执行SQL语句前没有提供足够的参数。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_NO_DATA：表示SQL语句不返回结果集。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。
- SQL_STILL_EXECUTING：表示语句正在执行。

注意事项

当SQLExecute函数返回SQL_ERROR或SQL_SUCCESS_WITH_INFO时，可通过调用**SQLGetDiagRec**函数，并将HandleType和Handle参数设置为SQL_HANDLE_STMT和StatementHandle，可得到一个相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

请参见：**典型应用开发示例**

5.5.4.12 SQLFetch

功能描述

从结果集中取下一个行集的数据，并返回所有被绑定列的数据。

原型

```
SQLRETURN SQLFetch(SQLHSTMT    StatementHandle);
```

参数

表 5-50 SQLFetch 参数

关键字	参数说明
StatementHandle	语句句柄，通过SQLAllocHandle获得。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_NO_DATA：表示SQL语句不返回结果集。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。
- SQL_STILL_EXECUTING：表示语句正在执行。

注意事项

当调用SQLFetch函数返回SQL_ERROR或SQL_SUCCESS_WITH_INFO时，通过调用**SQLGetDiagRec**函数，并将HandleType和Handle参数设置为SQL_HANDLE_STMT和StatementHandle，可得到一个相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

请参见：**典型应用开发示例**

5.5.4.13 SQLFreeStmt

在ODBC 3.x版本中，ODBC 2.x的函数SQLFreeStmt已被SQLFreeHandle代替。有关详细信息请参见**SQLFreeHandle**。

5.5.4.14 SQLFreeConnect

在ODBC 3.x版本中，ODBC 2.x的函数SQLFreeConnect已被SQLFreeHandle代替。有关详细信息请参见**SQLFreeHandle**。

5.5.4.15 SQLFreeHandle

功能描述

释放与指定环境、连接、语句或描述符相关联的资源，它替代了ODBC 2.x函数SQLFreeEnv、SQLFreeConnect及SQLFreeStmt。

原型

```
SQLRETURN SQLFreeHandle(SQLSMALLINT   HandleType,
                        SQLHANDLE     Handle);
```

参数

表 5-51 SQLFreeHandle 参数

关键字	参数说明
HandleType	SQLFreeHandle要释放的句柄类型。必须为下列值之一： • SQL_HANDLE_ENV • SQL_HANDLE_DBC • SQL_HANDLE_STMT • SQL_HANDLE_DESC 如果HandleType不是其中之一，SQLFreeHandle返回SQL_INVALID_HANDLE。
Handle	要释放的句柄。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。

注意事项

如果SQLFreeHandle返回SQL_ERROR，句柄仍然有效。

示例

请参见：**典型应用开发示例**

5.5.4.16 SQLFreeEnv

在ODBC 3.x版本中，ODBC 2.x的函数SQLFreeEnv已被SQLFreeHandle代替。有关详细信息请参见**SQLFreeHandle**。

5.5.4.17 SQLPrepare

功能描述

准备一个将要进行的SQL语句。

需要注意的是，ODBC发送准备好语句不支持内核复用计划，会导致每次执行都需要重新生成计划，导致CPU占用率高。如果业务对计划复用有需求建议优先使用JDBC作为客户端。

原型

```
SQLRETURN SQLPrepare(SQLHSTMT      StatementHandle,
                     SQLCHAR       *StatementText,
                     SQLINTEGER    TextLength);
```

参数

表 5-52 SQLPrepare 参数

关键字	参数说明
StatementHandle	语句句柄。
StatementText	SQL文本串。
TextLength	StatementText的长度。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。
- SQL_STILL_EXECUTING：表示语句正在执行。

注意事项

当SQLPrepare返回的值为SQL_ERROR或SQL_SUCCESS_WITH_INFO时，通过调用**SQLGetDiagRec**函数，并将HandleType和Handle参数分别设置为SQL_HANDLE_STMT和StatementHandle，可得到一个相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

请参见：**典型应用开发示例**

5.5.4.18 SQLGetData

功能描述

SQLGetData返回结果集中某一列的数据。可以多次调用它来部分检索不定长度的数据。

原型

```
SQLRETURN SQLGetData(SQLHSTMT        StatementHandle,
                     SQLUSMALLINT    Col_or_Param_Num,
                     SQLSMALLINT     TargetType,
                     SQLPOINTER      TargetValuePtr,
                     SQLLEN          BufferLength,
                     SQLLEN          *StrLen_or_IndPtr);
```

参数

表 5-53 SQLGetData 参数

关键字	参数说明
StatementHandle	语句句柄，通过SQLAllocHandle获得。
Col_or_Param_Num	要返回数据的列号。结果集的列按增序从1开始编号。书签列的列号为0。
TargetType	TargetValuePtr缓冲中的C数据类型的类型标识符。若TargetType为SQL_ARD_TYPE，驱动使用ARD中SQL_DESC_CONCISE_TYPE字段的类型标识符。若为SQL_C_DEFAULT，驱动根据源的SQL数据类型选择缺省的数据类型。
TargetValuePtr	**输出参数**：指向返回数据所在缓冲区的指针。
BufferLength	TargetValuePtr所指向缓冲区的长度。
StrLen_or_IndPtr	**输出参数**：指向缓冲区的指针，在此缓冲区中返回长度或标识符的值。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_NO_DATA：表示SQL语句不返回结果集。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。
- SQL_STILL_EXECUTING：表示语句正在执行。

注意事项

当调用SQLGetData函数返回SQL_ERROR或SQL_SUCCESS_WITH_INFO时，通过调用**SQLGetDiagRec**函数，并将HandleType和Handle参数分别设置为

SQL_HANDLE_STMT和StatementHandle，可得到一个相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

请参见：**典型应用开发示例**

5.5.4.19 SQLGetDiagRec

功能描述

返回诊断记录的多个字段的当前值，其中诊断记录包含错误、警告及状态信息。

原型

```
SQLRETURN  SQLGetDiagRec(SQLSMALLINT    HandleType,
                         SQLHANDLE      Handle,
                         SQLSMALLINT    RecNumber,
                         SQLCHAR        *SQLState,
                         SQLINTEGER     *NativeErrorPtr,
                         SQLCHAR        *MessageText,
                         SQLSMALLINT    BufferLength
                         SQLSMALLINT    *TextLengthPtr);
```

参数

表 5-54 SQLGetDiagRec 参数

关键字	参数说明
HandleType	句柄类型标识符，它说明诊断所要求的句柄类型。必须为下列值之一： • SQL_HANDLE_ENV • SQL_HANDLE_DBC • SQL_HANDLE_STMT • SQL_HANDLE_DESC
Handle	诊断数据结构的句柄，其类型由HandleType来指出。如果HandleType是SQL_HANDLE_ENV，Handle可以是共享的或非共享的环境句柄。
RecNumber	指出应用从查找信息的状态记录。状态记录从1开始编号。
SQLState	**输出参数**：指向缓冲区的指针，该缓冲区存储着有关RecNumber的五字符的SQLSTATE码。
NativeErrorPtr	**输出参数**：指向缓冲区的指针，该缓冲区存储着本地的错误码。
MessageText	指向缓冲区的指针，该缓冲区存储着诊断信息文本串。
BufferLength	MessageText的长度。

关键字	参数说明
TextLengthPtr	**输出参数**：指向缓冲区的指针，返回MessageText中的字节总数。如果返回字节数大于BufferLength，则MessageText中的诊断信息文本被截断成BufferLength减去NULL结尾字符的长度。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。

注意事项

SQLGetDiagRec不发布自己的诊断记录。用下列返回值来报告自己的执行结果：

- SQL_SUCCESS：函数成功返回诊断信息。
- SQL_SUCCESS_WITH_INFO：MessageText太小以致不能容纳所请求的诊断信息。没有诊断记录生成。
- SQL_INVALID_HANDLE：由HandleType和Handle所指出的句柄是不合法句柄。
- SQL_ERROR：RecNumber小于等于0或BufferLength小于0。

如果调用ODBC函数返回SQL_ERROR或SQL_SUCCESS_WITH_INFO，可调用SQLGetDiagRec返回诊断信息SQLSTATE值，SQLSTATE值如**表2 SQLSTATE值**所示。

表 5-55 SQLSTATE 值

SQLSTATE	错误	描述
HY000	一般错误	未定义特定的SQLSTATE所产生的错误。
HY001	内存分配错误	驱动程序不能分配所需要的内存来支持函数的执行或完成。
HY008	取消操作	调用SQLCancel取消执行语句后，依然在StatementHandle上调用函数。
HY010	函数系列错误	在为执行中的所有数据参数或列发送数据前就调用了执行函数。
HY013	内存管理错误	不能处理函数调用，可能由当前内存条件差引起。
HYT01	连接超时	数据源响应请求之前，连接超时。
IM001	驱动程序不支持此函数	调用了StatementHandle相关的驱动程序不支持的函。

示例

请参见：**典型应用开发示例**

5.5.4.20 SQLSetConnectAttr

功能描述

设置控制连接各方面的属性。

原型

```
SQLRETURN SQLSetConnectAttr(SQLHDBC      ConnectionHandle,
                            SQLINTEGER   Attribute,
                            SQLPOINTER   ValuePtr,
                            SQLINTEGER   StringLength);
```

参数

表 5-56 SQLSetConnectAttr 参数

关键字	参数说明
ConnectionHandle	连接句柄。
Attribute	设置属性。
ValuePtr	指向对应Attribute的值。依赖于Attribute的值，ValuePtr是32位无符号整型值，或指向以空结束的字符串，二进制缓冲区，或者驱动定义值。如果ValuePtr参数是驱动程序指定值，ValuePtr可能是有符号的整数。
StringLength	如果ValuePtr指向字符串或二进制缓冲区，则这个参数是*ValuePtr长度，如果ValuePtr指向整型，忽略StringLength。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。

注意事项

当SQLSetConnectAttr的返回值为SQL_ERROR或SQL_SUCCESS_WITH_INFO时，通过借助SQL_HANDLE_DBC的HandleType和ConnectionHandle的Handle，调用**SQLGetDiagRec**可得到相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

请参见：**典型应用开发示例**

5.5.4.21 SQLSetEnvAttr

功能描述

设置控制环境各方面的属性。

原型

```
SQLRETURN SQLSetEnvAttr(SQLHENV      EnvironmentHandle,
                        SQLINTEGER   Attribute,
                        SQLPOINTER   ValuePtr,
                        SQLINTEGER   StringLength);
```

参数

表 5-57 SQLSetEnvAttr 参数

关键字	参数说明
EnvironmentHandle	环境句柄。
Attribute	需设置的环境属性，可为如下值： • SQL_ATTR_ODBC_VERSION：指定ODBC版本。 • SQL_CONNECTION_POOLING：连接池属性。 • SQL_OUTPUT_NTS：指明驱动器返回字符串的形式。
ValuePtr	指向对应Attribute的值。依赖于Attribute的值，ValuePtr可能是32位整型值，或为以空结束的字符串。
StringLength	如果ValuePtr指向字符串或二进制缓冲区，则这个参数是*ValuePtr长度，如果ValuePtr指向整型，忽略StringLength。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。

注意事项

当SQLSetEnvAttr的返回值为SQL_ERROR或SQL_SUCCESS_WITH_INFO时，通过借助SQL_HANDLE_ENV的HandleType和EnvironmentHandle的Handle，调用**SQLGetDiagRec**可得到相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

请参见：**典型应用开发示例**

5.5.4.22 SQLSetStmtAttr

功能描述

设置相关语句的属性。

原型

```
SQLRETURN SQLSetStmtAttr(SQLHSTMT      StatementHandle,
                         SQLINTEGER    Attribute,
                         SQLPOINTER    ValuePtr,
                         SQLINTEGER    StringLength);
```

参数

表 5-58 SQLSetStmtAttr 参数

关键字	参数说明
StatementHandle	语句句柄。
Attribute	需设置的属性。
ValuePtr	指向对应Attribute的值。依赖于Attribute的值，ValuePtr可能是32位无符号整型值，或指向以空结束的字符串，二进制缓冲区，或者驱动定义值。如果ValuePtr参数是驱动程序指定值，ValuePtr可能是有符号的整数。
StringLength	如果ValuePtr指向字符串或二进制缓冲区，则这个参数是*ValuePtr长度，如果ValuePtr指向整型，忽略StringLength。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。

注意事项

当SQLSetStmtAttr的返回值为SQL_ERROR或SQL_SUCCESS_WITH_INFO时，通过借助SQL_HANDLE_STMT的HandleType和StatementHandle的Handle，调用**SQLGetDiagRec**可得到相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

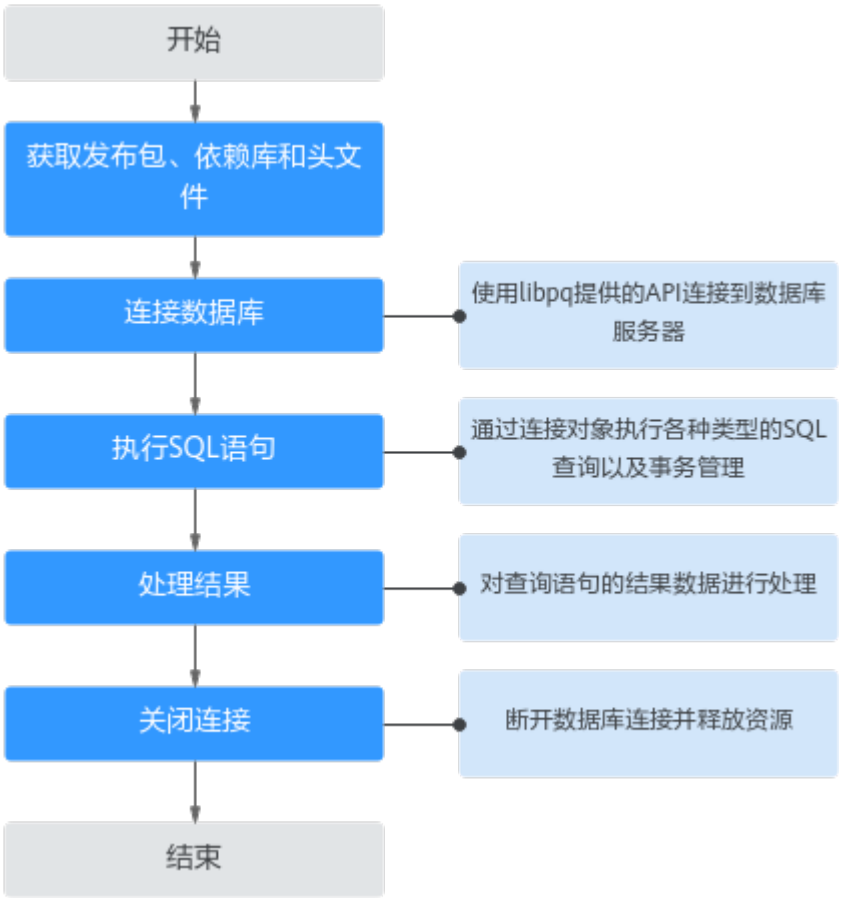
请参见：**典型应用开发示例**

5.6 基于 libpq 开发

5.6.1 开发流程

libpq开发流程如**图5-5**所示。

图 5-5 libpq 开发流程图



5.6.2 开发步骤

5.6.2.1 获取发布包、依赖库和头文件

libpq依赖的库和头文件从发布包中获取，包名为GaussDB-Kernel_数据库版本号_操作系统版本号_64bit_Libpq.tar.gz。其中include文件夹下的头文件为所需的头文件，lib文件夹中为所需的libpq库文件。使用libpq的程序必须包括头文件“libpq-fe.h”并且必须与libpq库连接。

说明

除libpq-fe.h外，include文件夹下默认还存在头文件postgres_ext.h、gs_thread.h、gs_threadlocal.h，这三个头文件是libpq-fe.h的依赖文件。

5.6.2.2 连接数据库

连接到数据库是使用libpq开发应用程序的第一步。此时用户可以使用**PQconnectdb**或**PQsetdbLogin**函数来建立与数据库服务器的连接。这些函数将返回一个连接对象，用户需要保存这个连接对象，以便后续的数据库操作。

以开发源程序testlibpq.c为例（完整示例请参考**数据库建连、执行SQL并返回结果**）：

```
/*
 * 说明： testlibpq.c源程序，提供libpq基本且常见的使用场景。
 * 使用libpq提供的PQconnectdb、PQexec、PQntuples、PQfinish等接口实现数据库建连，执行sql，获取返回结果以及资源清理。
 * 头文件，自定义函数请参考完整示例。
 */

/* 此处user、passwd等变量应从环境变量或配置文件读取，环境变量需用户自己按需配置；非环境变量情况下可直接赋值字符串 */
const char conninfo[1024];
PGconn     *conn;
PGresult   *res;
int         nFields;
int         i,j;
char        *passwd = getenv("EXAMPLE_PASSWD_ENV");
char        *port = getenv("EXAMPLE_PORT_ENV");
char        *host = getenv("EXAMPLE_HOST_ENV");
char        *username = getenv("EXAMPLE_USERNAME_ENV");
char        *dbname = getenv("EXAMPLE_DBNAME_ENV");

/*
 * 用户在命令行上提供了conninfo字符串的值时使用该值
 * 否则环境变量或者所有其它连接参数
 * 都使用缺省值。
 */
if (argc > 1)
    strcpy(conninfo, argv[1]);
else
    sprintf(conninfo,
        "dbname=%s port=%s host=%s application_name=test connect_timeout=5 sslmode=allow user=%s password=%s",
        dbname, port, host, username, passwd);

/* 连接数据库 */
conn = PQconnectdb(conninfo);

/* 检查后端连接成功建立 */
if (PQstatus(conn) != CONNECTION_OK)
{
    fprintf(stderr, "Connection to database failed: %s",
            PQerrorMessage(conn));
    exit_nicely(conn);
}
```

5.6.2.3 执行 SQL 语句

通过连接对象，使用**PQexec**函数执行SQL查询语句。可以执行各种类型的SQL查询，例如SELECT查询、插入数据、更新数据、删除数据等。但如果同时执行多个SQL语句作为一个事务，应该使用事务控制功能，例如执行BEGIN、COMMIT、ROLLBACK 等SQL语句来控制事务的开始、提交和回滚。同时也需要注意执行SQL查询后的错误处理。

示例如下（完整示例请参考**数据库建连、执行SQL并返回结果**）：

```
/*
 * 连接成功后
 * 测试实例涉及游标的使用时候必须使用事务块
 * 把全部放在一个 "select * from pg_database"
 * PQexec() 里，过于简单，不推荐使用
 */

/* 开始一个事务块 */
res = PQexec(conn, "BEGIN");
if (PQresultStatus(res) != PGRES_COMMAND_OK)
{
    fprintf(stderr, "BEGIN command failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}

/*
 * 在结果不需要的时候PQclear PGresult，以避免内存泄漏
 */
PQclear(res);

/*
 * 从系统表 pg_database（数据库的系统目录）里抓取数据
 */
res = PQexec(conn, "DECLARE myportal CURSOR FOR select * from pg_database");
if (PQresultStatus(res) != PGRES_COMMAND_OK)
{
    fprintf(stderr, "DECLARE CURSOR failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}
PQclear(res);

res = PQexec(conn, "FETCH ALL in myportal");
if (PQresultStatus(res) != PGRES_TUPLES_OK)
{
    fprintf(stderr, "FETCH ALL failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}
```

5.6.2.4 处理结果集

libpq提供**PQnfields**、**PQntuples**、**PQfname**等函数来帮助用户对执行SELECT查询后的结果进行适当的解析和处理。

示例如下（完整示例请参考**数据库建连、执行SQL并返回结果**）：

```
/* 打印属性名称 */
nFields = PQnfields(res);
for (i = 0; i < nFields; i++)
    printf("%-15s", PQfname(res, i));
printf("\n\n");

/* 打印行 */
for (i = 0; i < PQntuples(res); i++)
{
    for (j = 0; j < nFields; j++)
        printf("%-15s", PQgetvalue(res, i, j));
    printf("\n");
}

/* 释放结果对象的内存以避免内存泄漏 */
PQclear(res);
```


5.6.2.5 关闭连接

libpq通常使用**PQfinish**函数来关闭与数据库的连接。如果你的应用程序建立了多个连接，则需要确保每个连接都被正确关闭。

示例如下（完整示例请参考**数据库建连、执行SQL并返回结果**）：

```
/* 关闭入口 ... 不用检查错误 ... */
res = PQexec(conn, "CLOSE myportal");
PQclear(res);

/* 结束事务 */
res = PQexec(conn, "END");
PQclear(res);

/* 关闭数据库连接并清理 */
PQfinish(conn);
```

5.6.3 典型应用开发示例

5.6.3.1 数据库建连、执行 SQL 并返回结果

说明

gcc编译libpq源程序，需要通过-I *directory*选项，提供头文件的安装位置（有些时候编译器会查找缺省的目录，因此可以忽略这些选项）。如：

```
gcc -I (头文件所在目录) -L (libpq库所在目录) -o testlibpq testlibpq.c -lpq
```

执行命令为：

```
./testlibpq.c
```

如果要使用制作文件(makefile)，向CPPFLAGS、LDFLAGS、LIBS变量中增加如下选项：

```
CPPFLAGS += -I (头文件所在目录)
LDFLAGS += -L (libpq库所在目录)
LIBS += -lpq
例如：
CPPFLAGS += -I$(GAUSSHOME)/include/libpq
LDFLAGS += -L$(GAUSSHOME)/lib
```

```
/*
 * testlibpq.c
 * 说明： testlibpq.c源程序，提供libpq基本且常见的使用场景。
 * 使用libpq提供的PQconnectdb、PQexec、PQntuples、PQfinish等接口实现数据库建连，执行sql，获取返回结果以及资源清理。
 */
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>
#include <string.h>

static void
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

int main(int argc, char **argv)
{
    /* 此处user、passwd等变量应从环境变量或配置文件读取，环境变量需用户自己按需配置；非环境变量情况下可直接赋值字符串 */
    const char conninfo[1024];
    PGconn     *conn;
    PGresult   *res;
    int         nFields;
    int         i,j;
    char        *passwd = getenv("EXAMPLE_PASSWD_ENV");
```



```
    char        *port = getenv("EXAMPLE_PORT_ENV");
    char        *host = getenv("EXAMPLE_HOST_ENV");
    char        *username = getenv("EXAMPLE_USERNAME_ENV");
    char        *dbname = getenv("EXAMPLE_DBNAME_ENV");

    /*
     * 用户在命令行上提供了conninfo字符串的值时使用该值
     * 否则环境变量或者所有其它连接参数
     * 都使用缺省值。
     */
    if (argc > 1)
        strcpy(conninfo, argv[1]);
    else
        sprintf(conninfo,
            "dbname=%s port=%s host=%s application_name=test connect_timeout=5 sslmode=allow user=%s
password=%s",
            dbname, port, host, username, passwd);

    /* 连接数据库 */
    conn = PQconnectdb(conninfo);

    /* 检查后端连接成功建立 */
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s",
                PQerrorMessage(conn));
        exit_nicely(conn);
    }

    /*
     * 连接成功后
     * 测试实例涉及游标的使用时候必须使用事务块
     * 把全部放在一个 "select * from pg_database"
     * PQexec() 里，过于简单，不推荐使用
     */

    /* 开始一个事务块 */
    res = PQexec(conn, "BEGIN");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "BEGIN command failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    /*
     * 在结果不需要的时候PQclear PGresult，以避免内存泄漏
     */
    PQclear(res);

    /*
     * 从系统表 pg_database（数据库的系统目录）里抓取数据
     */
    res = PQexec(conn, "DECLARE myportal CURSOR FOR select * from pg_database");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "DECLARE CURSOR failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }
    PQclear(res);

    res = PQexec(conn, "FETCH ALL in myportal");
    if (PQresultStatus(res) != PGRES_TUPLES_OK)
    {
        fprintf(stderr, "FETCH ALL failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }
```



```
    /* 打印属性名称 */
    nFields = PQnfields(res);
    for (i = 0; i < nFields; i++)
        printf("%-15s", PQfname(res, i));
    printf("\n\n");

    /* 打印行 */
    for (i = 0; i < PQntuples(res); i++)
    {
        for (j = 0; j < nFields; j++)
            printf("%-15s", PQgetvalue(res, i, j));
        printf("\n");
    }

    /* 释放结果对象的内存以避免内存泄漏 */
    PQclear(res);

    /* 关闭入口 ... 不用检查错误 ... */
    res = PQexec(conn, "CLOSE myportal");
    PQclear(res);

    /* 结束事务 */
    res = PQexec(conn, "END");
    PQclear(res);

    /* 关闭数据库连接并清理 */
    PQfinish(conn);

    return 0;
}
```

示例运行结果如下，其中“user_name”表示数据库管理员用户名，根据具体使用环境会发生变化：

```
datname        datdba        encoding       datcollate     datctype       datistemplate  datallowconn
datconnlimit   datlastsysoid  datfrozenxid   dattablespace  datcompatibilitydatacl        datfrozenxid64
datminmxid     dattimezone    dattype

template_pdb   10            7              en_US.UTF-8    en_US.UTF-8    t              t              -1
12837          0             1663           A                             3              2              PRC            P
templatea      10            7              en_US.UTF-8    en_US.UTF-8    t              f              -1
12837          0             1663           A              {=c/user_name,user_name=CTc/user_name}41372
2              PRC           D
template1      10            7              en_US.UTF-8    en_US.UTF-8    t              t              -1
12837          0             1663           A              {=c/user_name,user_name=CTc/user_name}40414
2              PRC           D
templatem      10            7              en_US.UTF-8    en_US.UTF-8    t              t              -1
12837          0             1663           M              {=c/user_name,user_name=CTc/user_name}55146
2              PRC           D
template0      10            7              en_US.UTF-8    en_US.UTF-8    t              f              -1
12837          0             1663           A              {=c/user_name,user_name=CTc/user_name}39935
2              PRC           D
postgres       10            7              en_US.UTF-8    en_US.UTF-8    f              t              -1
12837          0             1663           A                             40893          2              PRC            D
```

5.6.3.2 执行预备语句

```
/*
 * testlibpq2.c 测试PQprepare
 * PQprepare： 创建一个给定参数的预备语句，用于PQexecPrepared执行预备语句。
 */
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>
#include <string.h>
int main(int argc, char * argv[])
{
    /* 此处user、passwd等变量应从环境变量或配置文件读取，环境变量需用户自己按需配置；非环境变量情况下
```



```
可直接赋值字符串 */
    PGconn *conn;
    PGresult * res;
    ConnStatusType pgstatus;
    char connstr[1024];
    char cmd_sql[2048];
    int nParams = 0;
    int paramLengths[5];
    int paramFormats[5];
    Oid paramTypes[5];
    char * paramValues[5];
    int i, cnt;
    char cid[32];
    int k;
    char *passwd = getenv("EXAMPLE_PASSWD_ENV");
    char *port = getenv("EXAMPLE_PORT_ENV");
    char *hostaddr = getenv("EXAMPLE_HOST_ENV");
    char *username = getenv("EXAMPLE_USERNAME_ENV");
    char *dbname = getenv("EXAMPLE_DBNAME_ENV");

    /* PQconnectdb连接数据库, 详细的连接信息为connstr */
    sprintf(connstr,
            "hostaddr=%s dbname=%s port=%s user=%s password=%s",
             hostaddr, dbname, port, username, passwd);
    conn = PQconnectdb(connstr);
    pgstatus = PQstatus(conn);
    if (pgstatus == CONNECTION_OK)
    {
        printf("Connect database success!\n");
    }
    else
    {
        printf("Connect database fail:%s\n",PQerrorMessage(conn));
        return -1;
    }

    /* 创建表t01 */
    res = PQexec(conn, "DROP TABLE IF EXISTS t01;CREATE TABLE t01(a int, b int);INSERT INTO t01
values(1, 23);");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        printf("Command failed: %s.\n", PQerrorMessage(conn));
        PQfinish(conn);
        return -1;
    }

    /* cmd_sql查询 */
    sprintf(cmd_sql, "SELECT b FROM t01 WHERE a = $1");
    /* cmd_sql中$1对应的参数 */
    paramTypes[0] = 23;
    /* PQprepare创建一个给定参数的预备语句 */
    res = PQprepare(conn,
                    "pre_name",
                    cmd_sql,
                    1,
                    paramTypes);
   if( PQresultStatus(res) != PGRES_COMMAND_OK )
    {
        printf("Failed to prepare SQL : %s\n: %s\n",cmd_sql, PQerrorMessage(conn));
        PQfinish(conn);
        return -1;
    }
    PQclear(res);
    paramValues[0] = cid;
    for (k=0; k<2; k++)
    {
        sprintf(cid, "%d", 1);
        paramLengths[0] = 6;
        paramFormats[0] = 0;
```



```
    /* 执行预备语句 */
    res = PQexecPrepared(conn,
                         "pre_name",
                         1,
                         paramValues,
                         paramLengths,
                         paramFormats,
                         0);
    if( (PQresultStatus(res) != PGRES_COMMAND_OK ) && (PQresultStatus(res) != PGRES_TUPLES_OK))
    {
        printf("%s\n",PQerrorMessage(conn));
        PQclear(res);
        PQfinish(conn);
        return -1;
    }
    cnt = PQntuples(res);
    printf("return %d rows\n", cnt);
    for (i=0; i<cnt; i++)
    {
        printf("row %d: %s\n", i, PQgetvalue(res, i, 0));
    }
    PQclear(res);
  }
  /* 执行结束，关闭连接 */
  PQfinish(conn);
  return 0;
}
```

示例运行结果如下：

```
Connect database success!
NOTICE:  table "t01" does not exist, skipping
return 1 rows
row 0: 23
return 1 rows
row 0: 23
```

5.6.3.3 绑定参数并返回二进制结果

```
/*
 * testlibpq3.c
 * 测试PQexecParams
 * PQexecParams： 执行一个绑定参数的命令，并以二进制格式请求查询结果。
 * 在运行这个例子之前，用下面的命令填充一个数据库
 *
 *
 * CREATE TABLE test1 (i int4, t text);
 *
 * INSERT INTO test1 values (2, 'ho there');
 *
 * 期望的输出如下
 *
 *
 * tuple 0: got
 *  i = (4 bytes) 2
 *  t = (8 bytes) 'ho there'
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <libpq-fe.h>

/* for ntohl/htonl */
#include <netinet/in.h>
#include <arpa/inet.h>

static void
```



```
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

/*
 * 这个函数打印查询结果，这些结果是二进制格式，从上面的
 * 注释里面创建的表中抓取出来的
 */
static void
show_binary_results(PGresult *res)
{
    int         i;
    int         i_fnum,
                t_fnum;

    /* 使用 PQfnumber 来避免对结果中的字段顺序进行假设 */
    i_fnum = PQfnumber(res, "i");
    t_fnum = PQfnumber(res, "t");

    for (i = 0; i < PQntuples(res); i++)
    {
        char       *iptr;
        char       *tptr;
        int         ival;

        /* 获取字段值（忽略可能为空的可能） */
        iptr = PQgetvalue(res, i, i_fnum);
        tptr = PQgetvalue(res, i, t_fnum);

        /*
         * INT4 的二进制表现形式是网络字节序
         * 建议转换成本地字节序
         */
        ival = ntohl(*((uint32_t *) iptr));

        /*
         * TEXT 的二进制表现形式是文本，因此libpq能够给它附加一个字节零
         * 把它看做 C 字符串
         *
         */

        printf("tuple %d: got\n", i);
        printf(" i = (%d bytes) %d\n",
               PQgetlength(res, i, i_fnum), ival);
        printf(" t = (%d bytes) '%s'\n",
               PQgetlength(res, i, t_fnum), tptr);
        printf("\n\n");
    }
}

int
main(int argc, char **argv)
{
    /* 此处user、passwd等变量应从环境变量或配置文件读取，环境变量需用户自己按需配置；非环境变量情况下
可直接赋值字符串 */
    const char conninfo[1024];
    PGconn     *conn;
    PGresult   *res;
    const char *paramValues[1];
    int         paramLengths[1];
    int         paramFormats[1];
    uint32_t    binaryIntVal;
    char        *passwd = getenv("EXAMPLE_PASSWD_ENV");
    char        *port = getenv("EXAMPLE_PORT_ENV");
    char        *hostaddr = getenv("EXAMPLE_HOST_ENV");
    char        *username = getenv("EXAMPLE_USERNAME_ENV");
    char        *dbname = getenv("EXAMPLE_DBNAME_ENV");
```



```
    /*
     * 如果用户在命令行上提供了参数，
     * 那么使用该值为conninfo 字符串；否则
     * 使用环境变量或者缺省值。
     */
    if (argc > 1)
        strcpy(conninfo, argv[1]);
    else
        sprintf(conninfo,
            "dbname=%s port=%s host=%s application_name=test connect_timeout=5 sslmode=allow user=%s
password=%s",
            dbname, port, hostaddr, username, passwd);

    /* 和数据库建立连接 */
    conn = PQconnectdb(conninfo);

    /* 检查与服务器的连接是否成功建立 */
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s",
                PQerrorMessage(conn));
        exit_nicely(conn);
    }

    res = PQexec(conn, "drop table if exists test1;CREATE TABLE test1 (i int4, t text);");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "command failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    PQclear(res);

    res = PQexec(conn, "INSERT INTO test1 values (2, 'ho there');");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "command failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    PQclear(res);

    /* 把整数值 "2" 转换成网络字节序 */
    binaryIntVal = htonl((uint32_t) 2);

    /* 为 PQexecParams 设置参数数组 */
    paramValues[0] = (char *) &binaryIntVal;
    paramLengths[0] = sizeof(binaryIntVal);
    paramFormats[0] = 1;        /* 二进制 */

    /* PQexecParams执行一个绑定参数的命令 */
    res = PQexecParams(conn,
                       "SELECT * FROM test1 WHERE i = $1::int4",
                       1,       /* 一个参数 */
                       NULL,    /* 让后端推导参数类型 */
                       paramValues,
                       paramLengths,
                       paramFormats,
                       1);      /* 要求二进制结果 */

    if (PQresultStatus(res) != PGRES_TUPLES_OK)
    {
        fprintf(stderr, "SELECT failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }
```



```
    /* 显示二进制结果 */
    show_binary_results(res);

    PQclear(res);

    /* 关闭与数据库的连接并清理 */
    PQfinish(conn);

    return 0;
}
```

示例运行结果如下：

```
tuple 0: got
 i = (4 bytes) 2
 t = (8 bytes) 'ho there'
```

5.6.4 libpq 接口参考

5.6.4.1 数据库连接控制函数

数据库连接控制函数控制与数据库服务器的连接。一个应用程序一次可以与多个服务器建立连接，如一个客户端连接多个数据库的场景。每个连接都是用一个从函数PQconnectdb、PQconnectdbParams或PQsetdbLogin获得的PGconn对象表示。注意，这些函数总是返回一个非空的对象指针，除非内存分配失败，会返回一个空的指针。连接建立的接口保存在PGconn对象中，可以调用PQstatus函数来检查返回值查看连接是否成功。

5.6.4.1.1 PQconnectdbParams

功能描述

与数据库服务器建立一个新的连接。

原型

```
PGconn* PQconnectdbParams(const char* const* keywords, const char* const* values, int expand_dbname);
```

参数

表 5-59 PQconnectdbParams 参数

关键字	参数说明
keywords	定义为一个字符串的数组，每个都成为一个关键字。
values	给每个关键字一个值。
expand_dbname	当expand_dbname非零时，允许将dbname的关键字值看做一个连接字符串。只有第一个出现的dbname是这样，随后的dbname值作为纯数据库名处理。

返回值

PGconn *：指向包含连接的对象指针，内存在函数内部申请。

注意事项

该函数用从两个NULL结束的数组中的参数打开一个新的数据库连接。与PQsetdbLogin不同，该函数可以不必更换函数签名（名字）就可以扩展参数集，所以建议应用程序中使用该函数（或者类似的非阻塞变种PQconnectStartParams和PQconnectPoll）。

5.6.4.1.2 PQconnectdb

功能描述

与数据库服务器建立一个新的连接。

原型

```
PGconn* PQconnectdb(const char* conninfo);
```

参数

表 5-60 PQconnectdb 参数

关键字	参数说明
conninfo	连接字符串，字符串中的字段请参见**连接参数说明**章节。

返回值

PGconn *：指向包含连接的对象指针，内存在函数内部申请。

注意事项

- 该函数用从一个字符串conninfo的参数与数据库打开一个新的连接。
- 传入的参数可以为空，表明使用所有缺省的参数，或者可以包含一个或更多个用空白间隔的参数设置，或者还可以包含一个URL。

示例

请参见**典型应用开发示例**章节。

5.6.4.1.3 PQconninfoParse

功能描述

根据连接，返回已解析的连接选项。

原型

```
PQconninfoOption* PQconninfoParse(const char* conninfo, char** errmsg);
```


参数

表 5-61

关键字	参数说明
conninfo	被传递的字符串。可以为空，这样将会使用默认参数。也可以包含由空格分隔的一个或多个参数设置，还可以包含一个URL。
errmsg	错误信息。

返回值

PQconninfoOption类型指针。

5.6.4.1.4 PQconnectStart

功能描述

与数据库服务器建立一次非阻塞的连接。

原型

```
PGconn* PQconnectStart(const char* conninfo);
```

参数

表 5-62

关键字	参数说明
conninfo	连接信息字符串。可以为空，这样将会使用默认参数。也可以包含由空格分隔的一个或多个参数设置，还可以包含一个URL。

返回值

PGconn类型指针。

5.6.4.1.5 PQerrorMessage

功能描述

返回连接上的错误信息。

原型

```
char* PQerrorMessage(const PGconn* conn);
```


参数

表 5-63

关键字	参数说明
conn	连接句柄。

返回值

char类型指针。

5.6.4.1.6 PQsetdbLogin

功能描述

与数据库服务器建立一个新的连接。

原型

```
PGconn* PQsetdbLogin(const char* pghost, const char* pgport, const char* pgoptions, const char* pgtty,
const char* dbName, const char* login, const char* pwd);
```

参数

表 5-64 PQsetdbLogin 参数

关键字	参数说明
pghost	要连接的主机名，详见**连接参数说明**描述的host字段。
pgport	主机服务器的端口号，详见**连接参数说明**描述的port字段。
pgoptions	添加命令行选项以在运行时发送到服务器，详见**连接参数说明**描述的options字段。
pgtty	忽略（该选项声明服务器日志的输出方向）。
dbName	要连接的数据库名，详见**连接参数说明**描述的dbname字段。
login	要连接的用户名，详见**连接参数说明**描述的user字段。
pwd	如果服务器要求密码认证，所用的密码，详见**连接参数说明**描述的password字段。

返回值

PGconn *：指向包含连接的对象指针，内存在函数内部申请。

注意事项

- 该函数为PQconnectdb前身，参数个数固定，未定义参数被调用时使用缺省值，若需要给固定参数设置缺省值，则可赋值NULL或者空字符串。
- 若dbName中包含“=”或连接URL的有效前缀，则该dbName被看做一个conninfo字符串并传递至PQconnectdb中，其余参数与PQconnectdbParams保持一致。

5.6.4.1.7 PQfinish

功能描述

关闭与服务器的连接，同时释放被PGconn对象使用的存储器。

原型

```
void PQfinish(PGconn* conn);
```

参数

表 5-65 PQfinish 参数

关键字	参数说明
conn	指向包含连接的对象指针。

注意事项

若PQstatus判断服务器连接尝试失败，应用程序调用PQfinish释放被PGconn对象使用的存储器，PQfinish调用后PGconn指针不可再次使用。

示例

请参见**典型应用开发示例**章节。

5.6.4.1.8 PQreset

功能描述

重置与服务器的通讯端口。

原型

```
void PQreset(PGconn* conn);
```


参数

表 5-66 PQreset 参数

关键字	参数说明
conn	指向包含连接的对象指针。

注意事项

此函数将关闭与服务器的连接并且试图与同一个服务器重建新的连接，并使用所有前面使用过的参数。该函数在连接异常后进行故障恢复时很有用。

5.6.4.1.9 PQstatus

功能描述

返回连接的状态。

原型

```
ConnStatusType PQstatus(const PGconn *conn);
```

参数

表 5-67 PQstatus 参数

关键字	参数说明
conn	指向包含连接的对象指针。

返回值

ConnStatusType：连接状态的枚举，包括：

```
CONNECTION_STARTED
等待进行连接。

CONNECTION_MADE
连接成功；等待发送。

CONNECTION_AWAITING_RESPONSE
等待来自服务器的响应。

CONNECTION_AUTH_OK
已收到认证；等待后端启动结束。

CONNECTION_SSL_STARTUP
协商SSL加密。

CONNECTION_SETENV
协商环境驱动的参数设置。

CONNECTION_OK
连接正常。
```



```
CONNECTION_BAD
连接故障。
```

注意事项

状态可以是多个值之一。但是，在异步连接过程之外只能看到其中两个：CONNECTION_OK和CONNECTION_BAD。与数据库的良好连接状态为CONNECTION_OK，与数据库连接失败状态为CONNECTION_BAD。通常，“正常”状态将一直保持到PQfinish，但通信失败可能会导致状态过早变为CONNECTION_BAD。在这种情况下，应用程序可以尝试通过调用进行恢复PQreset。

示例

请参见**典型应用开发示例**章节。

5.6.4.2 数据库执行语句函数

与数据库服务器的连接成功建立，便可以使用这里描述的函数执行SQL查询和命令。

5.6.4.2.1 PQclear

功能描述

释放与PGresult相关联的存储空间，任何不再需要的查询结果都应该用PQclear释放。

原型

```
void PQclear(PGresult* res);
```

参数

表 5-68 PQclear 参数

关键字	参数说明
res	包含查询结果的对象指针。

注意事项

PGresult不会自动释放，当提交新的查询时它并不消失，甚至断开连接后也不会。要删除它，必须调用PQclear，否则会有内存泄漏。

示例

请参见**典型应用开发示例**章节。

5.6.4.2.2 PQexec

功能描述

向服务器提交一条命令并等待结果。

原型

```
PGresult* PQexec(PGconn* conn, const char* query);
```

参数

表 5-69 PQexec 参数

关键字	参数说明
conn	指向包含连接的对象指针。
query	需要执行的查询字符串。

返回值

PGresult：包含查询结果的对象指针。

注意事项

应该调用PQresultStatus函数来检查任何错误的返回值（包括空指针的值，在这种情况下它将返回PGRES_FATAL_ERROR）。使用PQerrorMessage获取有关错误的更多信息。

须知

命令字符串可以包括多个SQL命令（用分号分隔）。在一个PQexec调用中发送的多个查询是在一个事务里处理的，除非在查询字符串里有明确的BEGIN/COMMIT命令把整个字符串分隔成多个事务。请注意，返回的PGresult结构只描述字符串里执行的最后一条命令的结果，如果有一个命令失败，那么字符串处理的过程就会停止，并且返回的PGresult会描述错误条件。

示例

请参见**典型应用开发示例**章节。

5.6.4.2.3 PQexecParams

功能描述

执行一个绑定参数的命令。

原型

```
PGresult* PQexecParams(PGconn* conn,
                       const char* command,
                       int nParams,
                       const Oid* paramTypes,
                       const char* const* paramValues,
                       const int* paramLengths,
                       const int* paramFormats,
                       int resultFormat);
```


参数

表 5-70 PQexecParams 参数

关键字	参数说明
conn	连接句柄。
command	SQL文本串。
nParams	绑定参数的个数
paramTypes	绑定参数的类型。
paramValues	绑定参数的值。
paramLengths	参数长度。
paramFormats	参数格式（文本或二进制）。
resultFormat	返回结果格式（文本或二进制）。

返回值

PGresult类型指针。

5.6.4.2.4 PQexecParamsBatch

功能描述

执行一个批量绑定参数的命令。

原型

```
PGresult* PQexecParamsBatch(PGconn* conn,
                            const char* command,
                            int nParams,
                            int nBatch,
                            const Oid* paramTypes,
                            const char* const* paramValues,
                            const int* paramLengths,
                            const int* paramFormats,
                            int resultFormat);
```

参数

表 5-71 PQexecParamsBatch 参数

关键字	参数说明
conn	连接句柄。
command	SQL文本串。
nParams	绑定参数的个数

关键字	参数说明
nBatch	批量操作数。
paramTypes	绑定参数的类型。
paramValues	绑定参数的值。
paramLengths	参数长度。
paramFormats	参数格式（文本或二进制）。
resultFormat	返回结果格式（文本或二进制）。

返回值

PGresult类型指针。

5.6.4.2.5 PQexecPrepared

功能描述

发送一个请求来用给定参数执行一个预备语句，并且等待结果。

原型

```
PGresult* PQexecPrepared(PGconn* conn,
                         const char* stmtName,
                         int nParams,
                         const char* const* paramValues,
                         const int* paramLengths,
                         const int* paramFormats,
                         int resultFormat);
```

参数

表 5-72 PQexecPrepared 参数

关键字	参数说明
conn	连接句柄。
stmtName	*stmt*名称，可以用""或者NULL来引用未命名语句，否则它必须是一个现有预备语句的名字。
nParams	参数个数。
paramValues	参数的实际值。
paramLengths	参数的实际数据长度。
paramFormats	参数的格式（文本或二进制）。
resultFormat	结果的格式（文本或二进制）。

返回值

PGresult类型指针。

5.6.4.2.6 PQexecPreparedBatch

功能描述

发送一个请求来用给定的批量参数执行一个预备语句，并且等待结果。

原型

```
PGresult* PQexecPreparedBatch(PGconn* conn,
                              const char* stmtName,
                              int nParams,
                              int nBatchCount,
                              const char* const* paramValues,
                              const int* paramLengths,
                              const int* paramFormats,
                              int resultFormat);
```

参数

表 5-73 PQexecPreparedBatch 参数

关键字	参数说明
conn	连接句柄。
stmtName	*stmt*名称，可以用""或者NULL来引用未命名语句，否则它必须是一个现有预备语句的名字。
nParams	参数个数。
nBatchCount	批量数。
paramValues	参数的实际值。
paramLengths	参数的实际数据长度。
paramFormats	参数的格式（文本或二进制）。
resultFormat	结果的格式（文本或二进制）。

返回值

PGresult类型指针。

5.6.4.2.7 PQprepare

功能描述

用给定的参数提交请求，创建一个预备语句，然后等待结束。

原型

```
PGresult* PQprepare(PGconn* conn, const char* stmtName, const char* query, int nParams, const Oid* paramTypes);
```

参数

表 5-74 PQprepare 参数

关键字	参数说明
conn	指向包含连接的对象指针。
stmtName	需要执行的prepare名称。
query	需要执行的查询字符串。
nParams	参数个数。
paramTypes	声明参数类型的数组。

返回值

PGresult：包含查询结果的对象指针。

注意事项

- PQprepare创建一个为PQexecPrepared执行用的预备语句，本特性支持命令的重复执行，不需要每次都进行解析和规划。PQprepare仅在协议3.0及以后的连接中支持，使用协议2.0时，PQprepare将失败。
- 该函数从查询字符串创建一个名为stmtName的预备语句，该查询字符串必须包含一个SQL命令。stmtName可以是""来创建一个未命名的语句，在这种情况下，任何预先存在的未命名的语句都将被自动替换，否则，如果在当前会话中已经定义了语句名称，那么这就是一个错误。如果使用了任何参数，那么在查询中将它们称为$1,$2等。nParams是在paramTypes[]数组中预先指定类型的参数的数量。（当nParams为0时，数组指针可以为NULL），paramTypes[]通过OID指定要分配给参数符号的数据类型。如果paramTypes为NULL，或者数组中的任何特定元素为零，服务器将按照对非类型化字面字符串的相同方式为参数符号分配数据类型。另外，查询可以使用数字高于nParams的参数符号，还将推断这些符号的数据类型。

须知

通过执行SQLPREPARE语句，还可以创建与PQexecPrepared一起使用的预备语句。此外，虽然没有用于删除预备语句的libpq函数，但是SQL DEALLOCATE语句可用于此目的。

示例

请参见**典型应用开发示例**章节。

5.6.4.3 异步命令处理函数

PQexec函数对普通的同步应用里提交命令已经足够使用。但是它却有几个缺陷，而这些缺陷可能对某些用户很重要。

- PQexec等待命令结束，而应用可能还有其它的工作要做（比如维护用户界面等），此时并不希望PQexec阻塞应用。
- 因为客户端应用在等待结果的时候是处于挂起状态的，所以应用很难判断它是否该尝试结束正在进行的命令。
- PQexec只能返回一个PGresult结构。如果提交的命令字符串包含多个SQL命令，除了最后一个PGresult以外都会被PQexec丢弃。
- PQexec总是收集命令的整个结果，将其缓存在一个PGresult中。虽然这为应用简化了错误处理逻辑，但是对于包含多行的结果是不切实际的。

不想受到这些限制的应用可以改用下面的函数，这些函数也是构造PQexec的函数：PQsendQuery和PQgetResult。PQsendQueryParams、PQsendPrepare、PQsendQueryPrepared也可以和PQgetResult一起使用。

5.6.4.3.1 PQsendQuery

功能描述

向服务器提交一个命令而不等待结果。如果查询成功发送则返回1，否则返回0。

原型

```
int PQsendQuery(PGconn* conn, const char* query);
```

参数

表 5-75 PQsendQuery 参数

关键字	参数说明
conn	指向包含连接的对象指针。
query	需要执行的查询字符串。

返回值

int：执行结果为1表示成功，0表示失败，失败原因存到conn->errorMessage中。

注意事项

在成功调用PQsendQuery后，调用PQgetResult一次或者多次获取结果。PQgetResult返回空指针表示命令已执行完成，否则不能再次调用PQsendQuery（在同一连接上）。

5.6.4.3.2 PQsendQueryParams

功能描述

给服务器提交一个命令和分隔的参数，而不等待结果。

原型

```
int PQsendQueryParams(PGconn* conn, const char* command, int nParams, const Oid* paramTypes, const char* const* paramValues, const int* paramLengths, const int* paramFormats, int resultFormat);
```

参数

表 5-76 PQsendQueryParams 参数

关键字	参数说明
conn	指向包含连接的对象指针。
command	需要执行的查询字符串。
nParams	参数个数。
paramTypes	参数类型。
paramValues	参数值。
paramLengths	参数长度。
paramFormats	参数格式。
resultFormat	结果的格式。

返回值

int：执行结果为1表示成功，0表示失败，失败原因存到conn->errorMessage中。

注意事项

该函数等效于PQsendQuery，只是查询参数可以和查询字符串分开声明。函数的参数处理和PQexecParams类似，它不能在2.0版本的协议连接上工作，并且它只允许在查询字符串里出现一条命令。

5.6.4.3.3 PQsendPrepare

功能描述

发送一个请求，创建一个给定参数的预备语句，而不等待结束。

原型

```
int PQsendPrepare(PGconn* conn, const char* stmtName, const char* query, int nParams, const Oid* paramTypes);
```


参数

表 5-77 PQsendPrepare 参数

关键字	参数说明
conn	指向包含连接的对象指针。
stmtName	需要执行的prepare名称。
query	需要执行的查询字符串。
nParams	参数个数。
paramTypes	声明参数类型的数组。

返回值

int：执行结果为1表示成功，0表示失败，失败原因存到conn->errorMessage中。

注意事项

该函数为PQprepare的异步版本：如果能够分派请求，则返回1，否则返回0。调用成功后，调用PQgetResult判断服务端是否成功创建了preparedStatement。函数的参数与PQprepare一样处理。与PQprepare一样，它也不能在2.0协议的连接上工作。

5.6.4.3.4 PQsendQueryPrepared

功能描述

发送一个请求执行带有给出参数的预备语句，不等待结果。

原型

```
int PQsendQueryPrepared(PGconn* conn, const char* stmtName, int nParams, const char* const*
paramValues, const int* paramLengths, const int* paramFormats, int resultFormat);
```

参数

表 5-78 PQsendQueryPrepared 参数

关键字	参数说明
conn	指向包含连接信息的对象指针。
stmtName	需要执行的prepare名称。
nParams	参数个数。
paramValues	参数值。
paramLengths	参数长度。
paramFormats	参数格式。

关键字	参数说明
resultFormat	结果的格式。

返回值

int：执行结果为1表示成功，0表示失败，失败原因存到conn->errorMessage中。

注意事项

该函数类似于PQsendQueryParams，但是要执行的命令是通过命名一个预先准备的语句来指定的，而不是提供一个查询字符串。该函数的参数与PQexecPrepared一样处理。和PQexecPrepared一样，它也不能在2.0协议的连接上工作。

5.6.4.3.5 PQflush

功能描述

尝试将任何排队的输出数据刷新到服务器。

原型

```
int PQflush(PGconn* conn);
```

参数

表 5-79 PQflush 参数

关键字	参数说明
conn	指向包含连接信息的对象指针。

返回值

int：如果成功（或者如果发送队列为空），则返回0；如果由于某种原因失败，则返回-1；如果发送队列中的所有数据都发送失败，则返回1。（此情况只有在连接为非阻塞时才能发生），失败原因存到conn->errorMessage中。

注意事项

在非阻塞连接上发送任何命令或数据之后，调用PQflush。如果返回1，则等待套接字变为读或写就绪。如果为写就绪状态，则再次调用PQflush。如果已经读到，调用PQconsumeInput，然后再次调用PQflush。重复，直到PQflush返回0。（必要检查读就绪并使用PQconsumeInput耗尽输入，因为服务器可能会阻止尝试向客户端发送数据（例如NOTICE消息），并且在客户端读取它的数据之前不会读取客户端的数据。）一旦PQflush返回0，等待套接字准备好，然后按照上面描述读取响应。

5.6.4.4 取消查询处理中函数

客户端应用可以使用本节描述的函数，要求取消一个仍在被服务器处理的命令。

5.6.4.4.1 PQgetCancel

功能描述

创建一个数据结构，其中包含取消通过特定数据库连接发出的命令所需的信息。

原型

```
PGcancel* PQgetCancel(PGconn* conn);
```

参数

表 5-80 PQgetCancel 参数

关键字	参数说明
conn	指向包含连接信息的对象指针。

返回值

PGcancel：指向包含cancel信息对象的指针。

注意事项

PQgetCancel创建一个给定PGconn连接对象的PGcancel对象。如果给定的conn是NULL或无效连接，它将返回NULL。PGcancel对象是一个不透明的结构，应用程序不能直接访问它，它只能传递给PQcancel或PQfreeCancel。

5.6.4.4.2 PQfreeCancel

功能描述

释放PQgetCancel创建的数据结构。

原型

```
void PQfreeCancel(PGcancel* cancel);
```

参数

表 5-81 PQfreeCancel 参数

关键字	参数说明
cancel	指向包含cancel信息的对象指针。

注意事项

PQfreeCancel释放一个由前面的PQgetCancel创建的数据对象。

5.6.4.4.3 PQcancel

功能描述

要求服务器放弃处理当前命令。

原型

```
int PQcancel(PGcancel* cancel, char* errbuf, int errbufsize);
```

参数

表 5-82 PQcancel 参数

关键字	参数说明
cancel	指向包含cancel信息的对象指针。
errbuf	出错时保存错误信息的buffer。
errbufsize	保存错误信息的buffer大小。

返回值

int：执行结果为1表示成功，0表示失败，失败原因存到errbuf中。

注意事项

- 成功发送并不保证请求将产生任何效果。如果取消有效，当前命令将提前终止并返回错误结果。如果取消失败（例如，因为服务器已经处理完命令），无返回结果。
- 如果errbuf是信号处理程序中的局部变量，则可以安全地从信号处理程序中调用PQcancel。就PQcancel而言，PGcancel对象是只读的，因此它也可以从一个线程中调用，这个线程与操作PGconn对象线程是分离的。

5.6.4.5 数据库结果处理函数

5.6.4.5.1 PQgetvalue

功能描述

返回一个PGresult的一行的单一域值。行和列号从 0 开始。调用者不应该直接释放该结果。它将在相关的PGresult句柄被传递给PQclear之后被释放。

原型

```
char *PQgetvalue(const PGresult* res, int tup_num, int field_num);
```


参数

表 5-83 PQgetvalue 参数

关键字	参数说明
res	操作结果句柄。
tup_num	行数。
field_num	列数。

返回值

对于文本格式的数据，PQgetvalue返回的值是该域值的一种空值结束的字符串表示。

对于二进制格式的数据，该值是由该数据类型的typsend和typreceive函数决定的二进制表示。

如果该域值为空，则返回一个空串。

示例

请参见**典型应用开发示例**章节。

5.6.4.5.2 PQnfields

功能描述

返回查询结果中每一行的列（域）数。

原型

```
int PQnfields(const PGresult* res);
```

参数

表 5-84 PQnfields 参数

关键字	参数说明
res	操作结果句柄。

返回值

int类型数字。

示例

请参见**典型应用开发示例**章节。

5.6.4.5.3 PQntuples

功能描述

返回查询结果中的行（元组）数。

原型

```
int PQntuples(const PGresult* res);
```

须知

PQntuples返回一个整数结果，在 32 位操作系统上大型的结果集可能使返回值溢出。

参数

表 5-85 PQntuples 参数

关键字	参数说明
res	操作结果句柄。

返回值

int类型数字。

示例

请参见**典型应用开发示例**章节。

5.6.4.5.4 PQfname

功能描述

返回与给定列号相关联的列名。列号从 0 开始。调用者不应该直接释放该结果。它将在相关的PGresult句柄被传递给PQclear之后被释放。

原型

```
char *PQfname(const PGresult* res, int field_num);
```

参数

表 5-86 PQfname 参数

关键字	参数说明
res	操作结果句柄。

关键字	参数说明
field_num	列数。

返回值

char类型指针。

示例

请参见**典型应用开发示例**章节。

5.6.4.5.5 PQresultStatus

功能描述

返回命令的结果状态。

原型

```
ExecStatusType PQresultStatus(const PGresult* res);
```

参数

表 5-87 PQresultStatus 参数

关键字	参数说明
res	包含查询结果的对象指针。

返回值

PQresultStatus：命令执行结果的枚举，包括：

```
PQresultStatus可以返回下面数值之一：
PGRES_EMPTY_QUERY
发送给服务器的字串是空的。

PGRES_COMMAND_OK
成功完成一个不返回数据的命令。

PGRES_TUPLES_OK
成功执行一个返回数据的查询（比如SELECT或者SHOW）。

PGRES_COPY_OUT
（从服务器）Copy Out（复制出）数据传输开始。

PGRES_COPY_IN
Copy In（复制入）（到服务器）数据传输开始。

PGRES_BAD_RESPONSE
服务器的响应无法理解。

PGRES_NONFATAL_ERROR
发生了一个非致命错误（通知或者警告）。
```

```
PGRES_FATAL_ERROR
发生了一个致命错误。

PGRES_COPY_BOTH
复制入/出（到和从服务器）数据传输开始。这个特性当前只用于流复制， 所以这个状态不会在普通应用中发生。

PGRES_SINGLE_TUPLE
PGresult包含一个来自当前命令的结果元组。 这个状态只在查询选择了单行模式时发生
```

注意事项

- 请注意，恰好检索到零行的SELECT命令仍然显示PGRES_TUPLES_OK。PGRES_COMMAND_OK用于永远不能返回行的命令（插入或更新，不带返回子句等）。PGRES_EMPTY_QUERY响应可能表明客户端软件存在bug。
- 状态为PGRES_NONFATAL_ERROR的结果永远不会由PQexec或其他查询执行函数直接返回，此类结果将传递给通知处理程序。

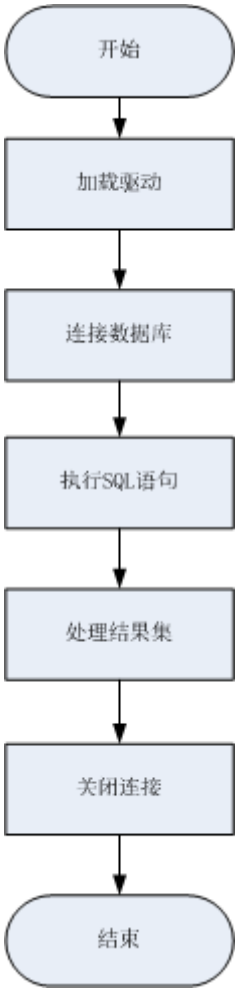
示例

请参见**典型应用开发示例**章节。

5.7 基于 Psycopg 开发

5.7.1 开发流程

图 5-6 采用 Psycopg2 开发应用程序的流程



5.7.2 开发步骤

步骤1 准备相关驱动和依赖库。可以从发布包中获取，包名为GaussDB-Kernel_数据库版本号_操作系统版本号_64bit_Python.tar.gz。

解压后有两个文件夹：

- psycopg2：psycopg2库文件。
- lib：lib库文件。

步骤2 加载驱动。

- 在使用驱动之前，需要做如下操作：
  a. 先解压版本对应驱动包。
  ```
  tar zxvf xxxx-Python.tar.gz
  ```
  b. 使用root用户将psycopg2复制到python安装目录下的site-packages文件夹下。
  ```
  su root
  cp psycopg2 $(python3 -c 'import site; print(site.getsitepackages()[0])') -r
  ```

c. 修改psycopg2目录权限为755。
```
chmod 755 $(python3 -c 'import site; print(site.getsitepackages()[0])')/psycopg2 -R
```
d. 将psycopg2目录添加到环境变量$PYTHONPATH，并使之生效。
```
export PYTHONPATH=$(python3 -c 'import site; print(site.getsitepackages()[0])'):$PYTHONPATH
```
e. 对于非数据库用户，需要将解压后的lib目录，配置在LD_LIBRARY_PATH中。
```
export LD_LIBRARY_PATH=path/to/lib:$LD_LIBRARY_PATH
```

- 在创建数据库连接之前，需要先加载如下数据库驱动程序：
```
import  psycopg2
```

步骤3 连接数据库。

非SSL方式连接数据库：

1. 使用psycopg2.connect函数获得connection对象。
2. 使用connection对象创建cursor对象。

SSL方式连接数据库：

用户通过psycopy2连接GaussDB服务器时，可以通过开启SSL加密客户端和服务器之间的通讯。在使用SSL时，默认用户已经获取了服务端和客户端所需要的证书和私钥文件，关于证书等文件的获取请参见Openssl相关文档和命令。

1. 使用*.ini文件（python的configparser包可以解析这种类型的配置文件）保存数据库连接的配置信息。
2. 在连接选项中添加SSL连接相关参数：sslmode、sslcert、sslkey、sslrootcert。
   a. sslmode：可选项见表5-88。
   b. sslcert：客户端证书路径。
   c. sslkey：客户端密钥路径。
   d. sslrootcert：根证书路径。
3. 使用psycopg2.connect函数获得connection对象。
4. 使用connection对象创建cursor对象。

注意

使用SSL安全连接数据库，需保证所使用的python解释器为生成动态链接库（.so）文件的方式编译，可通过如下步骤确认python解释器的连接方式。

1. 在python解释器命令行中输入import ssl，导入SSL。
2. 执行ps ux查询python解释器运行的pid（假设pid为******）。
3. 在python解释器命令行中执行pmap -p ****** | grep ssl，查看返回结果中是否包含libssl.so的相关路径。如果有，则python解释器为动态链接方式编译。

表 5-88 sslmode 的可选项及其描述

sslmode	是否会启用SSL加密	描述
disable	否	不使用SSL安全连接。
allow	可能	如果数据库服务器要求使用，则可以使用SSL安全加密连接，但不验证数据库服务器的真实性。

sslmode	是否会启用SSL加密	描述
prefer	可能	如果数据库支持，那么首选使用SSL连接，但不验证数据库服务器的真实性。
require	是	必须使用SSL安全连接，但是仅进行数据加密，而并不验证数据库服务器的真实性。
verify-ca	是	必须使用SSL安全连接，并且校验服务端CA有效性。
verify-full	是	必须使用SSL安全连接，目前GaussDB暂不支持。

步骤4 执行SQL语句。

1. 构造操作语句，使用%s作为占位符，执行时psycopg2会用参数值智能替换掉占位符。可以添加RETURNING子句，来得到自动生成的字段值。
2. 使用cursor.execute方法来操作一行SQL语句，使用cursor.executemany方法来操作多行SQL语句。

步骤5 处理结果集。

1. cursor.fetchone()：这种方法提取的查询结果集的下一行，返回一个序列，没有数据可用时则返回空。
2. cursor.fetchall()：这种方法获取所有查询结果（剩余）行，返回一个列表。空行时则返回空列表。

说明

对于数据库特有数据类型，如tinyint类型，查询结果中相应字段为字符串形式。

步骤6 关闭连接。

在使用数据库连接完成相应的数据操作后，需要关闭数据库连接。关闭数据库连接可以直接调用其close方法，如connection.close()。

注意

此方法关闭数据库连接，并不自动调用commit()。如果只是关闭数据库连接而不调用commit()方法，那么所有更改将会丢失。

----结束

5.7.3 示例：常用操作

```
import psycopg2
import os

# 从环境变量中获取用户名和密码。
user = os.getenv('user')
password = os.getenv('password')

# 创建连接对象。
conn=psycopg2.connect(database="database", user=user, password=password, host="localhost", port=port)
cur=conn.cursor() #创建指针对象。
```

```
# 创建连接对象（SSl连接）。
conn = psycopg2.connect(dbname="database", user=user, password=password, host="localhost", port=port,
        sslmode="verify-ca", sslcert="client.crt",sslkey="client.key",sslrootcert="cacert.pem")
注意： 如果sslcert、sslkey、sslrootcert没有填写，默认取当前用户.postgresql目录下对应的client.crt、
client.key、root.crt


# 创建表。
cur.execute("CREATE TABLE student(id integer,name varchar,sex varchar);")

# 插入数据。
cur.execute("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",(1,'Aspirin','M'))
cur.execute("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",(2,'Taxol','F'))
cur.execute("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",(3,'Dixheral','M'))

# 批量插入数据。
stus = ((4,'John','M'),(5,'Alice','F'),(6,'Peter','M'))
cur.executemany("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",stus)

# 获取结果。
cur.execute('SELECT * FROM student')
results=cur.fetchall()
print (results)

# 提交操作。
conn.commit()

# 插入一条数据。
cur.execute("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",(7,'Lucy','F'))

# 回退操作。
conn.rollback()

# 关闭连接。
cur.close()
conn.close()

# psycopg2常用连接方式。
conn = psycopg2.connect(dbname="dbname", user=user, password=password, host="localhost", port=port)
conn = psycopg2.connect(f"dbname=dbname user={user} password={password}  host=localhost port=port")

# 使用日志。
import logging
import psycopg2
from psycopg2.extras import LoggingConnection
import os

# 从环境变量中获取用户名和密码。
user = os.getenv('user')
password = os.getenv('password')

logging.basicConfig(level=logging.DEBUG) # 日志级别。
logger = logging.getLogger(__name__)

db_settings = {
    "user": user,
    "password": password,
    "host": "localhost",
    "database": "dbname",
    "port": port
}

# LoggingConnection默认记录所有SQL，可自行实现filter过滤不需要的或敏感的SQL，下面给出了简单的过滤
password相关SQL的示例。
class SelfLoggingConnection(LoggingConnection):

    def filter(self, msg, curs):
        if db_settings['password'] in msg.decode():
```



```
            return b'queries containing the password will not be recorded'
        return msg

conn = psycopg2.connect(connection_factory=SelfLoggingConnection, **db_settings)
conn.initialize(logger)
```

⚠ 注意

- LoggingConnection默认记录所有SQL信息，且不会对敏感信息进行脱敏，可通过filter函数自行定义输出的SQL内容。
- 日志功能是psycopg2为了方便开发者显性调试全量SQL而提供的额外功能，默认情况下不需要使用。该功能会在pyscopg2执行SQL语句前打印SQL语句，但是，需要在debug日志级别下才会输出。该功能不是默认功能，只是在有特殊需要的时候才使用，没有特别需求，不建议使用。详情参考：https://www.psycopg.org/docs/extras.html?highlight=loggingconnection

5.7.4 Psycopg 接口参考

Psycopg接口是一套提供给用户的API方法，本节将对部分常用接口做具体描述。

5.7.4.1 psycopg2.connect()

功能描述

此方法创建新的数据库会话并返回新的connection对象。

原型

```
import os
conn=psycopg2.connect(dbname="test",user=os.getenv('user'),password=os.getenv('password'),host="127.0.0.1",port="5432")
```

参数

表 5-89 psycopg2.connect 参数

关键字	参数说明
dbname	数据库名称。
user	用户名。
password	密码。
host	数据库IP地址，可指定多IP，IP间以逗号隔开，默认为UNIX socket类型。
port	连接端口号，默认为5432。host为多IP时，如端口号相同，指定一个端口号。否则，端口号与IP一一对应，以逗号隔开。
sslmode	ssl模式，ssl连接时用。
sslcert	客户端证书路径，ssl连接时用。

关键字	参数说明
sslkey	客户端密钥路径，ssl连接时用。
sslrootcert	根证书路径，ssl连接时用。
hostaddr	数据库IP地址。
connect_timeout	客户端连接超时时间。
client_encoding	客户端编码格式。
application_name	application_name的参数值。
fallback_application_name	application_name参数的回退值。
keepalives	控制是否客户端TCP保持连接，默认为1，表示打开；值为0时，表示关闭。若UNIX域套接字连接，则忽略。
options	连接开始时发送给服务器的命令行选项。
keepalives_idle	控制向服务器发送keepalive消息之前不活动的描述，若keepalive被禁用，则忽略此参数。
keepalives_interval	控制未得到服务器确认的keepalive消息应重新传输的描述，若keepalive被禁用，则忽略此参数。
keepalives_count	控制客户端与服务端连接断开之前可能丢失的tcp保持连接的数量。
replication	确认连接使用的是复制协议而不是普通协议。
requiressl	支持sslmode设置。
sslcompression	ssl压缩。设置为1，则通过ssl连接发送的数据将被压缩；设置为0，则禁用压缩。若没有建立ssl的连接，则忽略此参数。
sslcrl	证书吊销列表文件路径，验证ssl服务端证书是否可用。
requirepeer	指定服务器的操作系统用户名。
target_session_attrs	设定连接的主机的类型。主机的类型和设定的值一致时才能连接成功。指定多IP时才会校验此参数。target_session_attrs的设置规则如下： ● any：可以对所有类型的主机进行连接。 ● read-write：当连接的主机允许可读可写时，才进行连接。 ● read-only：仅对可读的主机进行连接。 ● primary（默认值）：仅对主备系统中的主机能进行连接。 ● standby：仅对主备系统中的备机进行连接。 ● prefer-standby：首先尝试找到一个备机进行连接。如果对hosts列表的所有机器都连接失败，那么尝试“any”模式进行连接。

关键字	参数说明
tcp_user_timeout	在支持TCP_USER_TIMEOUT套接字选项的操作系统上，指定传输的数据在TCP连接被强制关闭之前可以保持未确认状态的最大时长。0值表示使用系统缺省。通过Unix域套接字做的连接忽略这个参数。
rw_timeout	设置客户端连接读写超时时间。 当libpq侧触发超时且连接关闭时，其下发给数据库侧正在运行的业务会被强制终止。该能力受GUC参数check_disconnect_query控制，设置为on表示支持该能力，设置为off表示不支持该能力。

返回值

connection对象（连接数据库实例的对象）。

示例

请参见**示例：常用操作**。

5.7.4.2 connection.cursor()

功能描述

此方法用于返回新的cursor对象。

原型

```
cursor(name=None, cursor_factory=None, scrollable=None, withhold=False)
```

参数

表 5-90 connection.cursor 参数

关键字	参数说明
name	cursor名称，默认为None。
cursor_factory	用于创造非标准cursor，默认为None。
scrollable	设置SCROLL选项，默认为None。
withhold	设置HOLD选项，默认为False。

返回值

cursor对象（用于整个数据库使用Python编程的cursor）。

示例

请参见**示例：常用操作**。

5.7.4.3 cursor.execute(query,vars_list)

功能描述

此方法执行被参数化的SQL语句（即占位符，而不是SQL文字）。psycopg2模块支持用%s标志的占位符。

原型

```
cursor.execute(query,vars_list)
```

参数

表 5-91 cursor.execute 参数

关键字	参数说明
query	待执行的SQL语句。
vars_list	变量列表，匹配query中%s占位符。

返回值

无

示例

请参见**示例：常用操作**。

5.7.4.4 cursor.executemany(query,vars_list)

功能描述

此方法执行SQL命令所有参数序列或序列中的SQL映射。

原型

```
cursor.executemany(query,vars_list)
```

参数

表 5-92 cursor.executemany 参数

关键字	参数说明
query	待执行的SQL语句。

关键字	参数说明
vars_list	变量列表，匹配query中%s占位符。

返回值

无

示例

请参见**示例：常用操作**。

5.7.4.5 connection.commit()

功能描述

此方法将当前挂起的事务提交到数据库。

注意

默认情况下，Psycopg在执行第一个命令之前打开一个事务，如果不调用commit()，任何数据操作的效果都将丢失。

原型

```
connection.commit()
```

参数

无

返回值

无

示例

请参见**示例：常用操作**。

5.7.4.6 connection.rollback()

功能描述

此方法回滚当前挂起事务。

注意

执行关闭连接“close()”而不先提交更改“commit()”将导致执行隐式回滚。

原型

```
connection.rollback()
```

参数

无。

返回值

无。

示例

请参见**示例：常用操作**。

5.7.4.7 cursor.fetchone()

功能描述

此方法提取查询结果集的下一行，并返回一个元组。

原型

```
cursor.fetchone()
```

参数

无。

返回值

单个元组，为结果集的第一条结果，当没有更多数据可用时，返回为“None”。

示例

请参见**示例：常用操作**。

5.7.4.8 cursor.fetchall()

功能描述

此方法获取查询结果的所有（剩余）行，并将它们作为元组列表返回。

原型

```
cursor.fetchall()
```

参数

无。

返回值

元组列表，为结果集的所有结果。空行时则返回空列表。

示例

请参见**示例：常用操作**。

5.7.4.9 cursor.close()

功能描述

此方法关闭当前连接的游标。

原型

```
cursor.close()
```

参数

无。

返回值

无。

示例

请参见**示例：常用操作**。

5.7.4.10 connection.close()

功能描述

此方法关闭数据库连接。

注意

此方法关闭数据库连接，并不自动调用commit()。如果只是关闭数据库连接而不调用commit()方法，那么所有更改将会丢失。

原型

```
connection.close()
```

参数

无。

返回值

无。

示例

请参见**示例：常用操作**。

5.8 基于 Go 驱动开发

5.8.1 Go 驱动环境搭建

环境类

- **Go环境配置**

  用户需要在环境变量中配置以下参数：

  - GO111MODULE：用户使用在线导入的方式安装Go驱动时需要设置GO111MODULE为on。如果不希望进行go mod工程的改造，需将GO111MODULE设置为off，并手动下载依赖包。依赖包与驱动根目录和业务代码保持同级。
  - GOPROXY：用户使用在线导入时需配置包含Go驱动包的路径。
  - 用户可以根据自己场景参数配置Go其他相关环境变量。

  通过go env查看Go环境变量配置结果，并且查看Go版本是否在1.13或以上。

- **Go驱动安装**
  - 从发布包中获取Go驱动包。包名为GaussDB-Kernel_数据库版本号_*操作系统版本号*_64bit_Go.tar.gz。解压后为Go驱动源码包。
  - 进入Go驱动代码根路径，执行go mod tidy下载相关依赖，需要在环境变量中配置GOPATH=${Go驱动依赖包存放路径}。
  - 若依赖已下载至本地，可以在go.mod里面添加一行“通过replace将Go驱动包替换为本地Go驱动包地址”，表示代码里面所有的import Go驱动包都是走本地路径， 同时依赖也不会从代理里下载。

⚠ 注意

- 数据库提供的Go驱动包依赖Go 1.13及以上版本。
- 通过go mod tidy下载相关依赖时，可能会下载某个依赖的低版本，如果依赖的低版本存在漏洞，可以通过更改go.mod文件中对应依赖的版本号的方式，更新依赖到漏洞修复后的版本进行规避风险。

驱动类

在创建数据库连接时，需要传入数据库驱动名称“gaussdb”。

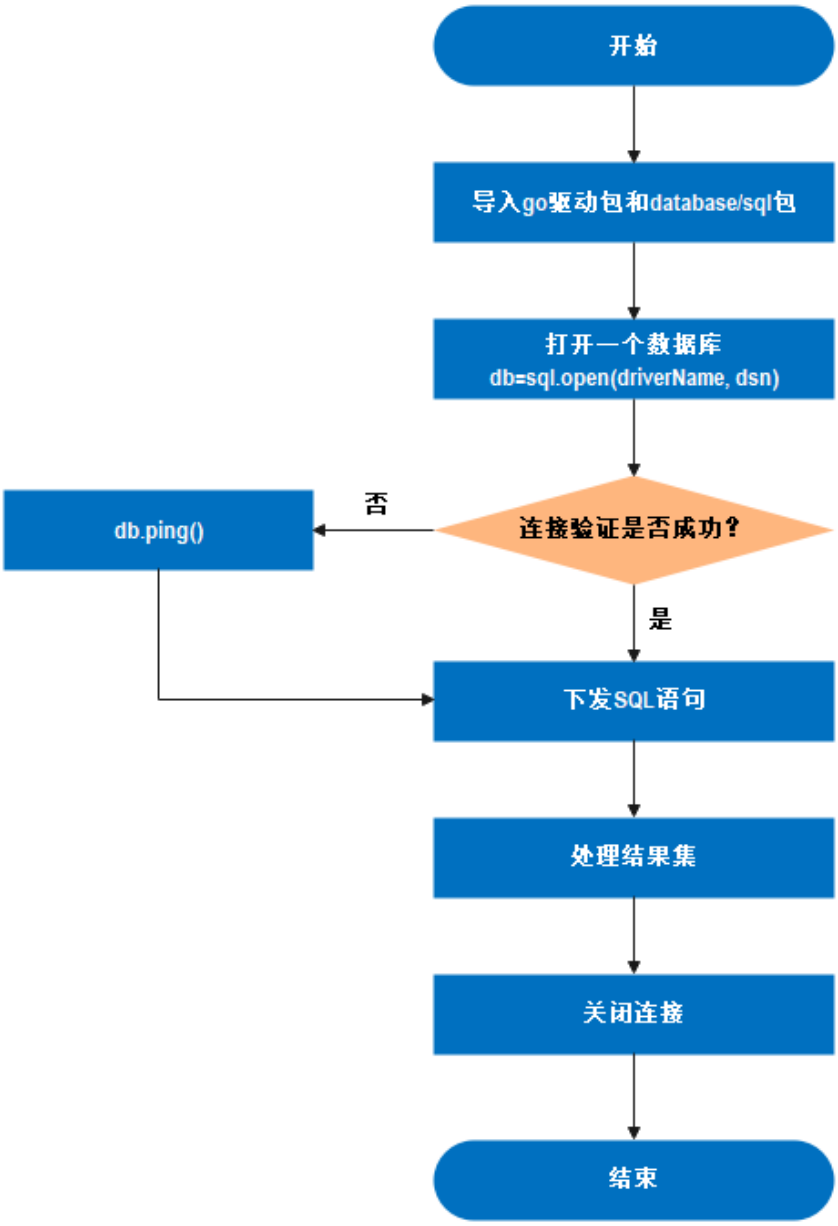
须知

数据库的Go驱动当前不适配业界成熟的ORM框架（比如xorm）。

5.8.2 开发流程

数据库Go驱动遵循Go语言第三方库的规则，只需在应用程序中import驱动，并将驱动代码放入GOPATH路径。

图 5-7 采用 Go 开发应用程序的流程图



5.8.3 连接数据库

使用Go驱动时，调用Go sql的标准接口open创建数据库连接，返回一个连接对象，传入驱动名称和描述字符串。

函数原型

Go驱动提供了如下的方法用于生成一个数据库连接对象。

```
func Open(driverName, dataSourceName string) (*DB, error)
```

参数说明：

- driverName为驱动的名称，数据库的驱动名称为"opengauss"，兼容"postgres。
- dataSourceName为连接的数据源，支持DSN和URL两种：
  - DSN格式：key1 = value1 key2 = value2 …，每组关键字间使用空格隔开，等号左右的空格是可选的。
  - URL格式：driverName://[userspec@][hostspec][/dbname][?paramspec]

    其中，driverName为驱动名称，数据库的驱动名称为“opengauss”，兼容“postgres”，“postgresql”。

    userspec表示user[:password]，需要注意的是使用URL进行连接时，密码中不可包含URL串中的分隔符。如果密码中包含分隔符的话，建议采用DSN格式。

    ```
    hostspec表示[host][:port][,…]
    ```

    dbname为数据库名称，不允许使用初始化用户进行远程登录。

    paramspec为name=value[&…]。

须知

- 在DSN格式中，对于多IP的场景：
  - 当num(ip) = num(port)时，ip和port是一一对应匹配。
  - 当num(ip) > num(port)时，无法匹配到port的ip均与第一个port匹配。例如，host = ip1, ip2, ip3 port = port1, port2的匹配情况为ip1:port1, ip2:port2, ip3:port1。
  - 当num(ip) < num(port)时，则多余的port被舍弃，即使用不到。例如host = ip1, ip2, ip3 port = port1, port2, port3, port4的匹配情况为ip1:port1, ip2:port2, ip3:port3。
- 在URL格式中，对于多IP的场景：
  - URL串中ip:port必须成对出现，即num(ip) = num(port)，并以逗号隔开。例如：opengauss://user:password@ip1:port1, ip2:port2, ip3:port3/postgres。
  - URL串中仅包含多ip，port由环境变量指定或采用默认值5432。例如：opengauss://user:password@ip1, ip2, ip3/postgres，如果设置环境变量PGPORT = "port1, port2"，其匹配情况为ip1:port1, ip2:port2, ip3:port1；未设置环境变量，其匹配情况为ip1:5432,ip2:5432,ip3:5432。

参数

表 5-93 数据库连接参数

参数名称	参数说明
host	主机IP地址，也可通过环境变量${PGHOST}来指定。

port	主机服务器的端口号，也可通过环境变量${PGPORT}来指定。
dbname	数据库名，也可通过环境变量${PGDATABASE}来指定。
user	要连接的用户名，也可通过环境变量${PGUSER}来指定。
password	要连接用户对应的连接密码。
connect_timeout	用于连接服务器操作的超时值，也可通过环境变量${PGCONNECT_TIMEOUT}来指定。
sslmode	启用SSL加密的方式，也可通过环境变量${PGSSLMODE}来指定。 参数取值范围： ● disable：不使用SSL安全连接。 ● allow：如果数据库服务器要求使用，则可以使用SSL安全加密连接，但不验证数据库服务器的真实性。 ● prefer：如果数据库支持，那么首选使用SSL安全加密连接，但不验证数据库服务器的真实性。 ● require：必须使用SSL安全连接，但是只做了数据加密，而并不验证数据库服务器的真实性。 ● verify-ca：必须使用SSL安全连接，并验证服务器是否具有由可信任的证书机构签发的证书。 ● verify-full：必须使用SSL安全连接，并且验证服务器是否具有由可信任的证书机构签发的证书，以及验证服务器主机名是否与证书中的一致
sslkey	指定用于客户端证书的密钥位置，如果需要走SSL连接，并且该参数未指定，可通过设置环境变量${PGSSLKEY}来指定。
sslcert	指定客户端SSL证书的文件名，或者通过设置环境变量${PGSSLCERT}来指定。
sslrootcert	指定一个包含SSL证书机构（CA）证书的文件名称，或者通过设置环境变量${PGSSLROOTCERT}来指定。
sslcrl	指定SSL证书撤销列表（CRL）的文件名。列在这个文件中的证书如果存在，在尝试认证该服务器证书时会被拒绝，从而连接失败。也可通过环境变量${PGSSLCRL}来指定。
sslpassword	指定对密钥解密成明文的密码短语，当指定该参数的时候表示sslkey是一个加密存储的文件，当前sslkey支持des/aes加密方式。 **说明** DES加密算法安全性低，存在安全风险，建议使用更安全的加密算法。

disable_prepared_binary_result	字符串类型，若设置为yes，表示此连接在从预准备语句接收查询结果时不应使用二进制格式。该参数仅用于调试。 取值范围：yes/no。
binary_parameters	字符串类型，该参数表示是否始终以二进制形式发送[]byte。取值范围：yes/no。若该参数设置为yes，建议绑定参数按照[]byte绑定，可以减少内部类型转换。
target_session_attrs	指定数据库的连接类型，该参数用于识别主备节点，也可通过环境变量${PGTARGETSESSIONATTRS}来指定。默认值为“any”，共有六种：any、master、slave、preferSlave、read-write、read-only。 ● any：尝试连接URL连接串中的任何一个数据节点。 ● master：尝试连接到URL连接串中的主节点，如果找不到就抛出异常。 ● slave：尝试连接到URL连接串中的备节点，如果找不到就抛出异常。 ● preferSlave：尝试连接到URL连接串中的备数据节点（如果有可用的话），否则连接到主数据节点。 ● read-write：读写模式，表示只能连接主节点。 ● read-only：只读模式，表示只能连接备节点。
loggerLevel	日志级别，打印相关调试信息，也可通过环境变量${PGLOGGERLEVEL}来指定。 支持trace/debug/info/warn/error/none，级别从高到低。
application_name	设置正在使用连接的GO驱动的名称。缺省值为go-driver，该参数不建议用户配置。
RuntimeParams	要在连接上设置为会话默认值的运行时参数。例如参数名search_path,application_name,timezone等。各参数的详细介绍参见客户端连接缺省设置，可通过show语法查看参数是否设置成功。
enable_ce	密态数据库开关。enable_ce=1表示go驱动支持密态等值查询基本能力。

示例一：

```
package main
//依赖包根据环境中依赖包路径设置。
import (
 "database/sql"
 "fmt"
 _ "gitee.com/opengauss/openGauss-connector-go-pq"
 "log"
```



```
 "strings"
 "time"
)
// 以下代码以单ip:port为例，本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量（环境变量名称请根据自身情况进行设置）。
func main() {
 hostip := os.Getenv("GOHOSTIP")   //GOHOSTIP为写入环境变量的IP地址。
 port := os.Getenv("GOPORT")       //GOPORT为写入环境变量的port。
 usrname := os.Getenv("GOUSRNAME") //GOUSRNAME为写入环境变量的用户名。
 passwd := os.Getenv("GOPASSWD")   //GOPASSWDW为写入环境变量的用户密码。
 str := "host=" + hostip + " port=" + port + " user=" + usrname + " password=" + passwd + " dbname=postgres sslmode=disable" // DSN连接串。
 //str := "opengauss://" + usrname + ":" + passwd + "@" + hostip + ":" + port + "/postgres?sslmode=disable"  // URL连接串。
 db, err:= sql.Open("opengauss", str)
 if err != nil {
  log.Fatal(err)
 }
 defer db.Close()

 err = db.Ping()
 if err != nil {
  log.Fatal(err)
 }

 sqls := []string {
  "drop table if exists testExec",
  "create table testExec(f1 int, f2 varchar(20), f3 number, f4 timestamptz, f5 boolean)",
  "insert into testExec values(1, 'abcdefg', 123.3, '2022-02-08 10:30:43.31 +08', true)",
  "insert into testExec values(:f1, :f2, :f3, :f4, :f5)",
 }

 inF1 := []int{2, 3, 4, 5, 6}
 intF2 := []string{"hello world", "华为", "北京2022冬奥会", "nanjing", "研究所"}
 intF3 := []float64{641.43, 431.54, 5423.52, 665537.63, 6503.1}
 intF4 := []time.Time{
  time.Date(2022, 2, 8, 10, 35, 43, 623431, time.Local),
  time.Date(2022, 2, 10, 19, 11, 54, 353431, time.Local),
  time.Date(2022, 2, 12, 6, 11, 15, 636431, time.Local),
  time.Date(2022, 2, 14, 4, 51, 22, 747653, time.Local),
  time.Date(2022, 2, 16, 13, 45, 55, 674636, time.Local),
 }
 intF5 := []bool{false, true, false, true, true}

 for _, s := range sqls {
  if strings.Contains(s, ":f") {
   for i, _ := range inF1 {
    _, err := db.Exec(s, inF1[i], intF2[i], intF3[i], intF4[i], intF5[i])
    if err != nil {
     log.Fatal(err)
    }
   }
  } else {
   _, err = db.Exec(s)
   if err != nil {
    log.Fatal(err)
   }
  }
 }

 var f1 int
 var f2 string
 var f3 float64
 var f4 time.Time
 var f5 bool
 err = db.QueryRow("select * from testExec").Scan(&f1, &f2, &f3, &f4, &f5)
 if err != nil {
  log.Fatal(err)
 } else {
```



```
 fmt.Printf("f1:%v, f2:%v, f3:%v, f4:%v, f5:%v\n", f1, f2, f3, f4, f5)
 }

 row, err :=db.Query("select * from testExec where f1 > :1", 1)
 if err != nil {
  log.Fatal(err)
 }
 defer row.Close()

 for row.Next() {
  err = row.Scan(&f1, &f2, &f3, &f4, &f5)
  if err != nil {
   log.Fatal(err)
  } else {
   fmt.Printf("f1:%v, f2:%v, f3:%v, f4:%v, f5:%v\n", f1, f2, f3, f4, f5)
  }
 }
}
```

示例二：

```
package main
//依赖包根据环境中依赖包路径设置。
import (
 "context"
 "database/sql"
 "fmt"
 _ "gitee.com/opengauss/openGauss-connector-go-pq"
 "log"
 "strings"
 "time"
)
// 以下代码以多ip:port为例，本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量（环境变量名称请根据自身情况进行设置）。
func main() {
 ctx := context.Background()
 ctx2SecondTimeout, cancelFunc2SecondTimeout := context.WithTimeout(ctx, 2 * time.Second)
 defer cancelFunc2SecondTimeout()

 hostip1 := os.Getenv("GOHOSTIP1")   //GOHOSTIP1为写入环境变量的IP地址。
 hostip2 := os.Getenv("GOHOSTIP2")   //GOHOSTIP2为写入环境变量的IP地址。
 hostip3 := os.Getenv("GOHOSTIP3")   //GOHOSTIP3为写入环境变量的IP地址。
 port1 := os.Getenv("GOPORT1")       //GOPORT1为写入环境变量的port。
 port2 := os.Getenv("GOPORT2")       //GOPORT2为写入环境变量的port。
 usrname := os.Getenv("GOUSRNAME") //GOUSRNAME为写入环境变量的用户名。
 passwd := os.Getenv("GOPASSWD")   //GOPASSWDW为写入环境变量的用户密码。

 str := "host="+hostip1+","+hostip2+","+hostip3+" port="+port1+","+port2+" user="+usrname+"
password="+passwd+" dbname=postgres sslmode=disable" // DSN连接串。
 //str := "opengauss://"+usrname+":"+passwd+"@"+hostip1+":"+port1+","+hostip2+":"+port2+","+hostip3+"/
postgres?sslmode=disable"  // URL连接串。
 db, err:= sql.Open("opengauss", str)
 if err != nil {
  log.Fatal(err)
 }
 defer db.Close()

 // Ping database connection with 2 second timeout
 err = db.PingContext(ctx2SecondTimeout)
 if err != nil {
  log.Fatal(err)
 }

 sqls := []string {
  "drop table if exists testExecContext",
  "create table testExecContext(f1 int, f2 varchar(20), f3 number, f4 timestamptz, f5 boolean)",
  "insert into testExecContext values(1, 'abcdefg', 123.3, '2022-02-08 10:30:43.31 +08', true)",
  "insert into testExecContext values(:f1, :f2, :f3, :f4, :f5)",
 }
```



```
inF1 := []int{2, 3, 4, 5, 6}
intF2 := []string{"hello world", "华为", "北京2022冬奥会", "nanjing", "研究所"}
intF3 := []float64{641.43, 431.54, 5423.52, 665537.63, 6503.1}
    intF4 := []time.Time{
        time.Date(2022, 2, 8, 10, 35, 43, 623431, time.Local),
        time.Date(2022, 2, 10, 19, 11, 54, 353431, time.Local),
        time.Date(2022, 2, 12, 6, 11, 15, 636431, time.Local),
        time.Date(2022, 2, 14, 4, 51, 22, 747653, time.Local),
        time.Date(2022, 2, 16, 13, 45, 55, 674636, time.Local),
    }
    intF5 := []bool{false, true, false, true, true}

    for _, s := range sqls {
        if strings.Contains(s, ":f") {
            for i, _ := range inF1 {
                _, err := db.ExecContext(ctx2SecondTimeout, s, inF1[i], intF2[i], intF3[i], intF4[i], intF5[i])
                if err != nil {
                    log.Fatal(err)
                }
            }
        } else {
            _, err = db.ExecContext(ctx2SecondTimeout, s)
            if err != nil {
                log.Fatal(err)
            }
        }
    }

    var f1 int
    var f2 string
    var f3 float64
    var f4 time.Time
    var f5 bool
    err = db.QueryRowContext(ctx2SecondTimeout, "select * from testExecContext").Scan(&f1, &f2, &f3, &f4,
&f5)
    if err != nil {
        log.Fatal(err)
    } else {
        fmt.Printf("f1:%v, f2:%v, f3:%v, f4:%v, f5:%v\n", f1, f2, f3, f4, f5)
    }

    row, err :=db.QueryContext(ctx2SecondTimeout, "select * from testExecContext where f1 > :1", 1)
    if err != nil {
        log.Fatal(err)
    }
    defer row.Close()

    for row.Next() {
        err = row.Scan(&f1, &f2, &f3, &f4, &f5)
        if err != nil {
            log.Fatal(err)
        } else {
            fmt.Printf("f1:%v, f2:%v, f3:%v, f4:%v, f5:%v\n", f1, f2, f3, f4, f5)
        }
    }
}
```

5.8.4 连接数据库（以 SSL 方式）

数据库的Go驱动支持SSL连接数据库，当开启SSL模式后，如果Go驱动采用SSL方式连接数据库服务端时，Go驱动默认走TLS 1.3标准协议，支持的tls版本最低为1.2。本小节主要介绍应用程序通过Go驱动如何采用SSL的方式对客户端进行配置（服务端配置请联系管理员）。在使用本小节所描述的方法前，默认用户已经获取了服务端和客户端所需要的证书和私钥文件，关于证书等文件的获取请参见Openssl相关文档和命令。

说明

基于SSL的证书认证方式不需要在连接串里面指定用户密码。

客户端配置

上传证书文件，将在服务端配置（服务端配置请联系管理员）操作中生成的文件client.key、client.crt、cacert.pem放置在客户端。

示例一：

```
package main
//依赖包根据环境中依赖包路径设置。
import (
 "database/sql"
 "fmt"
 _ "gitee.com/opengauss/openGauss-connector-go-pq"
 "log"
)
// 以双向认证为例，本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量（环境变量名称请根据自身情况进行设置）。
func main() {
 hostip := os.Getenv("GOHOSTIP")   //GOHOSTIP为写入环境变量的IP地址。
 port := os.Getenv("GOPORT")       //GOPORT为写入环境变量的port。
 usrname := os.Getenv("GOUSRNAME") //GOUSRNAME为写入环境变量的用户名。
 passwd := os.Getenv("GOPASSWD")   //GOPASSWDW为写入环境变量的用户密码。
 sslpasswd := os.Getenv("GOSSLPASSWD") //GOSSLPASSWDW为写入环境变量的密码短语。
 dsnStr := "host=" + hostip + " port=" + port + " user=" + usrname + " password=" + passwd + "
sslcert=certs/client.crt sslkey=certs/client.key sslpassword=" + sslpasswd
 parameters := []string {
  " sslmode=require",
  " sslmode=verify-ca sslrootcert=certs/cacert.pem",
 }

 for _, param := range parameters {
  db, err:= sql.Open("opengauss", dsnStr+param)
  if err != nil {
   log.Fatal(err)
  }

  var f1 int
  err = db.QueryRow("select 1").Scan(&f1)
  if err != nil {
   log.Fatal(err)
  } else {
   fmt.Printf("RESULT: select 1: %d\n", f1)
  }

  db.Close()
 }
}
```

示例二：

```
package main
//依赖包根据环境中依赖包路径设置。
import (
 "database/sql"
 _ "gitee.com/opengauss/openGauss-connector-go-pq"
 "log"
 "strings"
)
// 以验证sslpassword为主，本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量（环境变量名称请根据自身情况进行设置）。
func main() {
 hostip := os.Getenv("GOHOSTIP")   //GOHOSTIP为写入环境变量的IP地址。
 port := os.Getenv("GOPORT")       //GOPORT为写入环境变量的port。
```



```
 usrname := os.Getenv("GOUSRNAME") //GOUSRNAME为写入环境变量的用户名。
 passwd := os.Getenv("GOPASSWD")   //GOPASSWDW为写入环境变量的用户密码。
 dsnStr := "host=" + hostip + " port=" + port + " user=" + usrname + " password=" + passwd + "
dbname=postgres"
  sslpasswd := os.Getenv("GOSSLPASSWD") //GOSSLPASSWDW为写入环境变量的密码短语。
 connStrs := []string {
  " sslmode=verify-ca sslcert=certs/client_rsa.crt sslkey=certs/client_rsa.key sslpassword=" + sslpasswd + "
sslrootcert=certs/cacert_rsa.pem",
  " sslmode=verify-ca sslcert=certs/client_ecdsa.crt sslkey=certs/client_ecdsa.key sslpassword=" + sslpasswd +
" sslrootcert=certs/cacert_ecdsa.pem",
 }
 for _, connStr := range connStrs {
  db, err := sql.Open("opengauss", dsnStr + connStr)
  if err != nil {
   log.Fatal(err)
  }
  var f1 int
  err = db.QueryRow("select 1").Scan(&f1)
  if err != nil {
   if !strings.HasPrefix(err.Error(), "connect failed.") {
    log.Fatal(err)
   }
  }
  db.Close()
 }
}
```

5.8.5 Go 接口参考

5.8.5.1 sql.Open 接口

sql.Open接口如下表所示。

方法	描述	返回值
Open(driverName, dataSourceName string)	根据给定的数据库驱动以及驱动专属的数据源来打开一个数据库。	*DB, error

参数driverName和dataSourceName详解请参见**连接数据库**。

5.8.5.2 type DB

type DB如下表所示。

方法	描述	返回值
(db *DB)Begin()	开启一个事务，事务的隔离级别由驱动决定。	*Tx, error
(db *DB)BeginTx(ctx context.Context, opts *TxOptions)	开启一个给定事务隔离级别的事务，给定的上下文会一直使用到事务提交或回滚为止。若上下文被取消，那么sql包将会对事务进行回滚。	*Tx, error

(db *DB)Close()	关闭数据库并释放所有已打开的资源。	error
(db *DB)Exec(query string, args ...interface{})	执行一个不返回数据行的操作。	Result, error
(db *DB)ExecContext(ctx context.Context, query string, args ...interface{})	在给定上下文中，执行一个不返回数据行的操作。	Result, error
(db *DB)Ping()	检查数据库连接是否仍然有效，并在有需要时建立一个连接。	error
(db *DB)PingContext(ctx context.Context)	在给定上下文中，检查数据库连接是否仍然有效，并在有需要时建立一个连接。	error
(db *DB)Prepare(query string)	为以后的查询或执行创建一个预备语句。	*Stmt, error
(db *DB)PrepareContext(ctx context.Context, query string)	在给定的上下文中，为以后的查询或执行创建一个预备语句。	*Stmt, error
(db *DB)Query(query string, args ...interface{})	执行一个查询并返回多个数据行。	*Rows, error
(db *DB)QueryContext(ctx context.Context, query string, args ...interface{})	在给定的上下文中，执行一个查询并返回多个数据行。	*Rows, error
(db *DB)QueryRow(query string, args ...interface{})	执行一个只返回一个数据行的查询。	*Row
(db *DB)QueryRowContext(ctx context.Context, query string, args ...interface{})	在给定上下文中，执行一个只返回一个数据行的查询。	*Row

须知

1. Query类接口Query()、QueryContext()、QueryRow()、QueryRowContext()通常用于查询语句，如SELECT语句。操作语句使用Exec()接口执行，若非查询语句通过Query类接口执行，则执行结果可能与预期不符，因此不建议使用Query类接口执行非查询语句，例如UPDATE/INSERT等。
2. 使用Query类接口执行查询语句的结果需要通过**type Rows**中Next()接口获取，若不通过Next()接口获取，可能会产生不可预期的错误。

参数说明

参数	参数说明
ctx	表示给定的上下文。
query	被执行的SQL语句。
args	被执行SQL语句需要绑定的参数。支持按位置绑定和按名称绑定，详情见如下**示例**。
opts	事务隔离级别和事务访问模式，其中事务隔离级别（opts.Isolation）支持范围为sql.LevelReadUncommitted,sql.LevelReadCommitted,sql.LevelRepeatableRead,sql.LevelSerializable。事务访问模式（opts.ReadOnly）支持范围为true（read only）和false（read write）。

示例

```
//本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量（环境变量名称请根据自身情况进行设置）。
package main
/* Go驱动依赖包位置根据配置的go.mod设置。 */
import (
 "database/sql"
 _ "gitee.com/opengauss/openGauss-connector-go-pq"
 "log"
)
func main() {
 hostip := os.Getenv("GOHOSTIP")   //GOHOSTIP为写入环境变量的IP地址。
 port := os.Getenv("GOPORT")       //GOPORT为写入环境变量的port。
 usrname := os.Getenv("GOUSRNAME") //GOUSRNAME为写入环境变量的用户名。
 passwd := os.Getenv("GOPASSWD")   //GOPASSWDW为写入环境变量的用户密码。
 str := "host=" + hostip + " port=" + port + " user=" + usrname + " password=" + passwd + "
dbname=postgres sslmode=disable"
 db, err:= sql.Open("opengauss", str)
 if err != nil {
  log.Fatal(err)
 }
 defer db.Close()

 err = db.Ping()
 if err != nil {
  log.Fatal(err)
 }

       _, err = db.Exec("drop table if exists testuser.test")
       _, err = db.Exec("create table test(id int, name char(10))")
```

```
 // 按位置绑定
 _, err = db.Exec("insert into test(id, name) values(:1, :2)", 1, "张三")
 if err != nil {
  log.Fatal(err)
 }

 // 按名称绑定
 _, err = db.Exec("insert into test(id, name) values(:id, :name)", sql.Named("id", 1), sql.Named("name", "张三
"))
 if err != nil {
  log.Fatal(err)
 }
}
```

5.8.5.3 type Stmt

type Stmt如下表所示。

方法	描述	返回值
(s *Stmt)Close()	关闭给定的预处理语句。	error

(s *Stmt)Exec(args ...interface{})	使用给定的参数执行预处理语句，并返回一个Result值。支持PBE 特性。PBE，即Prepare-Bind-Execute，是发送和执行查询的一种方式，CN可通过复杂查询协议接收PBE报文执行语句。 **说明** 1. 预编译语句的占位符可以为“$”也可以是“?”。 2. 预编译语句的占位符数量由数据库限制，当表字段超过数据库限制或者与当前表字段数目不匹配时，由服务端返回错误。 3. PBE特性支持增、删、改操作，批量操作时 U 报文最大长度限制 为1GB - 1B，即0x3fffffff字节。超出长度限制会报bind message length XXX too long. This can be caused by very large or incorrect length specifications on InputStream parameters错误。 4. 插入一条数据的时候，使用PBE对比单插接口(conn.simpleExec，简单执行语句)有大幅的性能下降，建议优先使用单插接口而不是PBE。 5. 驱动底层错误处理重构，PBE性能对比原来下降不到5%。	Result, error
(s *Stmt)ExecContext(ctx context.Context, args ...interface{})	在给定的上下文中，使用给定的参数执行预处理语句，并返回一个Result值。	Result, error
(s *Stmt)Query(args ...interface{})	使用给定的参数执行预处理语句，并以*Rows形式返回查询结果。	*Rows, error
(s *Stmt)QueryContext(ctx context.Context, args ...interface{})	在给定的上下文中，使用给定的参数执行预处理语句，并以*Rows形式返回查询结果。	*Rows, error
(s *Stmt)QueryRow(args ...interface{})	使用给定的参数执行预处理语句，并返回一个*Row作为结果。	*Row

(s *Stmt)QueryRowContext (ctx context.Context, args ...interface{})	在给定的上下文中，使用给定的参数执行预处理语句，并返回一个*Row作为结果。	*Row

参数说明

参数	参数说明
ctx	表示给定的上下文。
query	被执行的SQL语句。
args	被执行SQL语句需要绑定的参数。支持按位置绑定和按名称绑定，详情见DB类型中的**示例**。

须知

1. Query类接口Query()、QueryContext()、QueryRow()、QueryRowContext()通常用于查询语句，如SELECT语句。操作语句使用Exec()接口执行，若非查询语句通过Query类接口执行，则执行结果可能与预期不符，因此不建议使用Query类接口执行非查询语句，例如UPDATE/INSERT等。
2. 使用Query类接口执行查询语句的结果需要通过**type Rows**中Next()接口获取，若不通过Next()接口获取，可能会产生不可预期的错误。

5.8.5.4 type Tx

type Tx如下表所示。

方法	描述	返回值
(tx *Tx)Commit()	提交事务。	error
(tx *Tx)Exec(query string, args ...interface{})	执行一个不返回数据行的操作。	Result, error
(tx *Tx)ExecContext(ctx context.Context, query string, args ...interface{})	在给定上下文中，执行一个不返回数据行的操作。	Result, error
(tx *Tx)Prepare(query string)	为以后的查询或执行创建一个预备语句。返回的语句将在事务中执行，并且一旦事务被提交或回滚就不能再使用。	*Stmt, error

(tx *Tx)PrepareContext(ctx context.Context, query string)	为以后的查询或执行创建一个预备语句。返回的语句将在事务中执行，并且一旦事务被提交或回滚就不能再使用。 给定的上下文将用于预备阶段，而不是事务执行阶段。该方法返回的语句将在事务上下文中执行。	*Stmt, error
(tx *Tx)Query(query string, args ...interface{})	执行一个返回数据行的查询。	*Rows, error
(tx *Tx)QueryContext(ctx context.Context, query string, args ...interface{})	在给定上下文中，执行一个返回数据行的查询。	*Rows, error
(tx *Tx)QueryRow(query string, args ...interface{})	执行一个只返回一个数据行的查询。	*Row
(tx *Tx)QueryRowContext(ctx context.Context, query string, args ...interface{})	在给定上下文中，执行一个只返回一个数据行的查询。	*Row
(tx *Tx) Rollback()	事务回滚。	error

(tx *Tx)Stmt(stmt *Stmt)	为已有的语句返回一个事务专用的预备语句。
	示例：
	`str, err := db.Prepare("insert into t1 values(:1, :2)")`
	`tx, err := db.Begin()`
	`res, err := tx.Stmt(str).Exec(1, "aaa")`
		*Stmt

(tx *Tx)StmtContext(ctx context.Context, stmt *Stmt)	在给定上下文中，为已有的语句返回一个事务专用的预备语句。	*Stmt

须知

1. Query类接口Query()、QueryContext()、QueryRow()、QueryRowContext()通常用于查询语句，如SELECT语句。操作语句使用Exec()接口执行，若非查询语句通过Query类接口执行，则执行结果可能与预期不符，因此不建议使用Query类接口执行非查询语句，例如UPDATE/INSERT等。
2. 使用Query类接口执行查询语句的结果需要通过**type Rows**中Next()接口获取，若不通过Next()接口获取，可能会产生不可预期的错误。

参数说明

参数	参数说明
ctx	表示给定的上下文。
query	被执行的SQL语句。
args	被执行SQL语句需要绑定的参数。支持按位置绑定和按名称绑定，详情DB类型中的**示例**。
stmt	已有的预处理语句，一般指prepare语句返回的预处理语句。

5.8.5.5 type Rows

type Rows如下表所示。

方法	描述	返回值
(rs *Rows)Close()	关闭Rows，停止对数据集的迭代。	error
(rs *Rows)ColumnTypes()	返回列信息。	[]*ColumnType, error
(rs *Rows)Columns()	返回各个列的名字。	[]string, error
(rs *Rows)Err()	返回迭代过程中出现的任何错误。	error
(rs *Rows)Next()	为Scan方法准备好下一个待读取的数据行。如果有进一步的结果集，则返回true，否则返回false。	boolean
(rs *Rows)Scan(dest ...interface{})	将当前被迭代数据行的各个列复制到dest指向的值中。	error
(rs *Rows)NextResultSet()	判断是否有进一步的结果集	boolean

参数说明

参数	参数说明
dest	查询列需要被复制到该参数指向的值

5.8.5.6 type Row

type Row如下所示。

方法	描述	返回值

(r *Row)Scan(dest ...interface{})	将当前数据行中的列复制到dest指向的值中。	error
(r *Row)Err()	返回执行过程中出现的错误。	error

参数说明

参数	参数说明
dest	查询列需要被复制到该参数指向的值

5.8.5.7 type ColumnType

type ColumnType如下表所示。

方法	描述	返回值
(ci *ColumnType)DatabaseTypeName()	返回列类型的数据库系统名称。返回空字符串表示该驱动类型名字并未被支持。	error
(ci *ColumnType)DecimalSize()	返回小数类型的范围和精度。 返回值ok的值为false时，说明给定的类型不可用或者不支持。	precision, scale int64, ok boolean
(ci *ColumnType)Length()	返回数据列类型长度。返回值ok的值为false时，说明给定的类型不存在长度。	length int64, ok boolean
(ci *ColumnType)ScanType()	返回一种Go类型，该类型能够在Rows.scan进行扫描时使用。	reflect.Type
(ci *ColumnType)Name()	返回数据列的名字。	string

5.8.5.8 type Result

type Result如下表所示。

方法	描述	返回值
(res Result)RowsAffected()	返回insert、delete、update、select、move、fetch和copy操作受影响的行数。	int64, error

5.9 基于 ecpg 开发

ecpg(embedded SQL C preprocessor for GaussDB Kernel）是一种用于C语言程序的嵌入式SQL预处理器。一个嵌入式SQL程序由一种普通编程语言编写的代码（此处为C语言）和SQL命令共同组成。要构建该程序，源代码（*.pgc）首先通过嵌入式SQL预处理器，将源代码转换成一个普通C语言程序（*.c），然后再通过编译器处理。转换过的ecpg应用通过嵌入式SQL库（ecpglib）调用libpq库中的函数，与GaussDB Kernel服务器使用普通的前端/后端协议通信。

嵌入式SQL程序是插入了数据库相关动作的特殊代码的C语言程序。这种特殊代码形式通常如下：

```
EXEC SQL ...;
```

这些语句在语法上取代了一个C语句，可以出现在全局或者是一个函数中。嵌入式SQL语句遵循普通SQL代码的大小写敏感规则，也允许嵌套的C语言代码风格注释（SQL标准的一部分）。不过，程序的C语言部分遵循C语言程序的标准，不支持嵌套注释。

5.9.1 开发流程

图 5-8 ecpg 整体开发流程

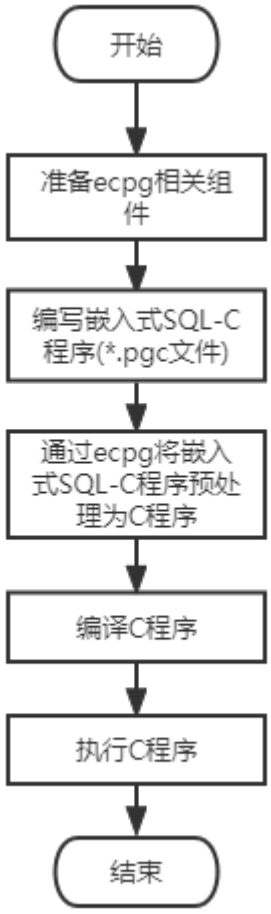


图 5-9 嵌入式 SQL-C 程序开发流程

5.9.2 ecpg 组件介绍

- ecpg支持平台

  表 5-94 ecpg 支持平台

操作系统	平台
EulerOS V2.0SP5	x86_64位
EulerOS V2.0SP9	ARM64位

- ecpg组件
  - ecpg：用于对嵌入式SQL-C进行预处理的可执行二进制文件。
  - libecpg：为ecpg提供连接、执行SQL、事务等实现的动态库，包括libecpg.so、libecpg.so.6和libecpg.so.6.4，在C语言程序编译执行时通过“-lecpg”参数引用。
  - libpgtypes：ecpg提供的用于实现数值、日期、时间戳、区间类型数据操作运算的动态库，包括libpgtypes.so、libecpg.so.6和libecpg.so.6.4，在C语言程序编译执行时通过“-lpgtypes”参数引用。
- ecpg组件的获取路径

- ecpg二进制获取路径：$GAUSSHOME/bin
- ecpg依赖动态库路径：$GAUSSHOME/lib
- ecpg所需头文件路径：$GAUSSHOME/include/ecpg

5.9.3 ecpg 预处理以及编译执行

准备嵌入式SQL-C源程序，以.pgc为后缀名，ecpg负责将其转换成可被编译器编译的C语言程序。

生成的C语言程序被编译器编译为可执行文件，运行该可执行文件实现客户端程序访问数据库。示例请参见**常用示例**章节。

- ecpg预处理以及编译处理过程
  a. 预处理：ecpg -I $GAUSSHOME/include -o test.c test.pgc

     ecpg预处理的参数选项如下：

     ```
     ecpg [OPTION]...
     ```

     其中OPTION参数选项如下：

     - -o OUTFILE：预处理嵌入式SQL-C程序将结果写入OUTFILE，OUTFILE为C语言文件。
     - -I DIRECTORY：头文件的搜索路径。
     - -c：预处理嵌入式SQL-C程序自动生成C语言文件。
     - --version：查看ecpg当前版本。

  b. 编译：gcc -I $GAUSSHOME/include/ecpg -I $GAUSSHOME/include -I $GAUSSHOME/include/postgresql/server/ -L $GAUSSHOME/lib -lecpg -lrt -lpq -lpgtypes -lpthread test_ecpg.c -o test_ecpg

  c. 执行：./test

须知

- ecpg作为编译预处理工具，若在预处理或编译过程中出现找不到头文件或者函数实现的报错信息，可以根据需要指定头文件，或者链接动态库。
- ecpg需要gcc、ld等编译预处理工具，建议gcc使用7.3.0版本。
- 使用ecpg开发应用程序所依赖的其他动态库和头文件，常见的位于$GAUSSHOME/include/libpq, $GAUSSHOME/include。
- 编译过程中常见的动态库依赖：-lpq、-lpq_ce、-lpthread。若开发过程中需要使用libpq通信库，则需要连接-lpq和-lpq_ce。若开发过程中需要使用多线程连接，则需要连接-lpthread。

5.9.4 管理数据库连接

本章节介绍如何建立以及切换数据库连接。

5.9.4.1 连接数据库

使用如下语句连接数据库：


```
EXEC SQL CONNECT TO target [AS connection-name] [USER user-name];
```

target可以通过如下方法声明，斜体部分为变量，请根据实际情况进行修改：

- *dbname*[@*hostname*][:*port*]
- tcp:postgresql://*hostname*[:*port*][/*dbname*][?*options*]
- unix:postgresql://*hostname*[:*port*][/*dbname*][?*options*]
- 一个包含上述形式之一的SQL字符串

声明连接用户名的方法有以下方式：

- *username*/*password*
- *username* SQLIDENTIFIED BY *password*
- *username* USING *password*

如上所述，参数username以及password可以是一个SQL标识符、一个SQL字符串或一个对字符变量的引用。

connection_name表示连接名，如果一个程序只使用一个连接，则可以省略它。最近打开的连接成为当前连接。

示例如下所示：

```
#include <stdlib.h>
EXEC SQL CONNECT TO mydb@sql.mydomain.com;

EXEC SQL CONNECT TO unix:postgresql://sql.mydomain.com/mydb AS myconnection USER username;

EXEC SQL BEGIN DECLARE SECTION;
/* 此处target、user、passwd应从环境变量或配置文件读取，环境变量需用户自己按需配置；非环境变量情况下可直接赋值字符串 */
const char *target = getenv("EXAMPLE_TARGET_ENV");
const char *user = getenv("EXAMPLE_USERNAME_ENV");
const char *passwd = getenv("EXAMPLE_PASSWD_ENV");
EXEC SQL END DECLARE SECTION;
...
EXEC SQL CONNECT TO :target USER :user USING :passwd;
/* 或者 EXEC SQL CONNECT TO :target USER :user/:passwd; */
```

完整示例，请参见**CONNECT**连接语法示例。

说明

- 最后一种形式引用了字符变量，在**宿主变量**中将介绍如何在SQL语句中引用C语言变量。
- 连接目标的格式未在SQL标准中说明，因此若要开发可移植的应用，可使用上述最后一个例子的方法将连接目标字符串封装在某个变量里。

须知

- 若连接语句中指定了ip-port，则必须指定*username*/*password*，该规则由GaussDB Kernel内核通信认证所决定。若不指定ip-port，则通过本地$PGPORT(UDS协议)进行通信。
- 若客户连接时使用SSL安全协议，则需要使用tcp:postgresql://*hostname*[:*port*][/*dbname*][?*options*]连接格式，在options选项中配置sslmode=disable\require。

5.9.4.2 管理连接

嵌入式SQL程序中的SQL语句默认是在当前连接（最近打开的那一个）上执行。如果一个应用需要管理多个连接，那么有以下两种方法。

- 方法1：为每个SQL语句明确选择一个连接：
  ```
  EXEC SQL AT connection-name SELECT ...;
  ```
  适合于应用程序需以混合顺序使用多个连接的情况。

  如果应用程序创建多个执行线程，它们不能共享同一个连接，必须明确控制对连接的访问（利用互斥量）或者每个线程使用一个唯一连接。
- 方法2：执行一个语句来切换连接：
  ```
  EXEC SQL SET CONNECTION connection-name;
  ```
  适用于许多语句在同一个连接上执行的情况。

管理连接示例如下：

```
#include <stdio.h>
EXEC SQL BEGIN DECLARE SECTION;
   char dbname[1024];
EXEC SQL END DECLARE SECTION;

int main()
{
    EXEC SQL CONNECT TO testdb1 AS con1 USER testuser;
    EXEC SQL CONNECT TO testdb2 AS con2 USER testuser;
    EXEC SQL CONNECT TO testdb3 AS con3 USER testuser;

    /* 这个查询将在最近打开的数据库 "testdb3" 中执行 */
    EXEC SQL SELECT current_database() INTO :dbname;
    printf("current=%s (should be testdb3)\n", dbname);

    /* 使用 "AT" 在 "testdb2" 中运行一个查询 */
    EXEC SQL AT con2 SELECT current_database() INTO :dbname;
    printf("current=%s (should be testdb2)\n", dbname);

    /* 切换当前连接到 "testdb1" */
    EXEC SQL SET CONNECTION con1;

    EXEC SQL SELECT current_database() INTO :dbname;
    printf("current=%s (should be testdb1)\n", dbname);

    EXEC SQL DISCONNECT ALL;
    return 0;
}
```

示例输出：

```
current=testdb3 (should be testdb3)
current=testdb2 (should be testdb2)
current=testdb1 (should be testdb1)
```

须知

- 多线程模式下不支持不同线程使用同一连接名，每个线程连接名唯一。
- 连接的建立和关闭需要在同一进程或线程进行。

5.9.5 执行 SQL 命令

嵌入式SQL命令格式为EXEC SQL [Command]，在嵌入的SQL应用中可以运行GaussDB Kernel支持的常见标准SQL语句，或者ecpg提供的扩展SQL语句。当前不支持存储过程、package、匿名块、闪回等特性语法。

5.9.5.1 执行 SQL 语句

步骤1 创建一个表：

```
EXEC SQL CREATE TABLE foo (a int, b varchar);
```

步骤2 插入一行：

```
EXEC SQL INSERT INTO foo VALUES (5, 'abc');
```

步骤3 删除一行：

```
EXEC SQL DELETE FROM foo WHERE a = 5;
```

步骤4 更新表数据：

```
EXEC SQL UPDATE foo SET b = 'gdp' WHERE a = 7;
```

步骤5 单行查询数据：

```
EXEC SQL SELECT a INTO :var_a FROM foo WHERE b = 'def';
```

----结束

完整使用示例：

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

int main ()
{
    ECPGdebug (1, stderr);

    EXEC SQL BEGIN DECLARE SECTION;
    int var_a;
    EXEC SQL END DECLARE SECTION;

    EXEC SQL CONNECT TO postgres;
    // 创建一个表
    EXEC SQL CREATE TABLE foo (a int, b varchar);
    // 插入数据
    EXEC SQL INSERT INTO foo VALUES (5, 'abc');
    EXEC SQL INSERT INTO foo VALUES (6, 'def');
    EXEC SQL INSERT INTO foo VALUES (7, 'ghi');

    // 删除一行
    EXEC SQL DELETE FROM foo WHERE a = 5;
    // 更新表数据
    EXEC SQL UPDATE foo SET b = 'gdp' WHERE a = 7;
    // 单行查询表数据
    EXEC SQL SELECT a INTO :var_a FROM foo WHERE b = 'def';
    // 打印查询结果
    printf("select res is %d\n", var_a);

    EXEC SQL DISCONNECT;

    return 0;
}
```

5.9.5.2 使用游标

使用游标可以检索出多行的结果集，应用程序必须声明一个游标并且从游标中抓取每一行数据。

步骤1 声明一个游标：

```
EXEC SQL DECLARE c CURSOR FOR select * from tb1;
```

步骤2 打开游标：


```
EXEC SQL OPEN c;
```

步骤3 从游标中抓取一行数据：

```
EXEC SQL FETCH 1 in c into :a, :str;
```

步骤4 关闭游标：

```
EXEC SQL CLOSE c;
```

----结束

更多游标的使用细节请参见**DECLARE**，关于FETCH命令的细节请参见**FETCH**。

完整使用示例：

```
#include <string.h>
#include <stdlib.h>

int main(void)
{
exec sql begin declare section;
    int *a = NULL;
    char *str = NULL;
exec sql end declare section;

    int count = 0;
    exec sql connect to postgres ;
    exec sql set autocommit to off;
    exec sql begin;
    exec sql drop table if exists tb1;
    exec sql create table tb1(id int, info text);
    exec sql insert into tb1 (id, info) select generate_series(1, 100000), 'test';
    exec sql select count(*) into :a from tb1;
    printf ("a is %d\n", *a);
    exec sql commit;

    // 定义游标
    exec sql declare c cursor for select * from tb1;
    // 打开游标
    exec sql open c;
    exec sql whenever not found do break;
    while(1) {
        // 抓取数据
        exec sql fetch 1 in c into :a, :str;
        count++;
        if (count == 100000) {
            printf("Fetch res: a is %d, str is %s", *a, str);
        }
    }
    // 关闭游标
    exec sql close c;
    exec sql set autocommit to on;
    exec sql drop table tb1;
    exec sql disconnect;

    ECPGfree_auto_mem();
    return 0;
```

5.9.5.3 事务管理

在ecpg缺省模式下，语句只有在EXEC SQL COMMIT发出的时候才被提交，嵌入的SQL接口也支持事务的自动提交（通过EXEC SQL SET AUTOCOMMIT TO ON语句设置自动提交）。在自动提交模式下，每条命令都是自动提交的，除非它们包围在一个明确的事务块里。自动提交模式可以用EXEC SQL SET AUTOCOMMIT TO OFF语句关闭。

常见事务管理命令如下：

- EXEC SQL COMMIT：提交正在进行的事务。

- EXEC SQL ROLLBACK：回滚正在进行的事务。
- EXEC SQL SET AUTOCOMMIT TO ON：启动自动提交模式。
- EXEC SQL SET AUTOCOMMIT TO OFF：关闭自动提交模式，缺省模式。

5.9.5.4 预备语句

当传递给SQL语句的值在编译时未知或者同一语句将被使用多次时，可以使用预备语句。

- 使用命令PREPARE准备语句。对于未知的值使用占位符"?"：
```
EXEC SQL PREPARE stmt1 FROM "SELECT oid, datname FROM pg_database WHERE oid = ?";
```
- 如果一个语句返回单行，应用程序可以在PREPARE执行语句之后调用EXECUTE，同时使用USING子句为占位符提供实际值：
```
EXEC SQL EXECUTE stmt1 INTO :dboid, :dbname USING 1;
```
- 如果一个语句返回多行，应用程序可以使用基于预备语句声明的游标。为了绑定输入参数，必须使用USING子句打开游标：
```
EXEC SQL PREPARE stmt1 FROM "SELECT oid,datname FROM pg_database WHERE oid > ?";
EXEC SQL DECLARE foo_bar CURSOR FOR stmt1;
/* 当结果集达到最后时，跳出while循环 */
EXEC SQL WHENEVER NOT FOUND DO BREAK;
EXEC SQL OPEN foo_bar USING 100;
...
while (1)
{
    EXEC SQL FETCH NEXT FROM foo_bar INTO :dboid, :dbname;
    ...
}
EXEC SQL CLOSE foo_bar;
```
- 当不再需要预备语句的时候，应释放语句：
```
EXEC SQL DEALLOCATE PREPARE name;
```

5.9.5.5 嵌入式 SQL 命令

5.9.5.5.1 ALLOCATE DESCRIPTOR

功能描述

分配一个新命名的SQL描述符区域。

语法格式

```
ALLOCATE DESCRIPTOR name
```

参数说明

name

SQL描述符名称。大小写敏感，是一个SQL标识或者一个宿主变量。

示例

```
EXEC SQL ALLOCATE DESCRIPTOR mydesc;
```

相关链接

DEALLOCATE DESCRIPTOR，**GET DESCRIPTOR**，**SET DESCRIPTOR**

5.9.5.5.2 CONNECT

功能描述

在客户端和SQL服务器之间建立连接。

语法格式

```
CONNECT TO connection_target [ AS connection_name ] [ USER connection_user ]
```

参数说明

- **connection_target**

  以下列形式之一指定连接的目标服务器：

  - [*database_name*] [@*host*] [:*port*]：通过TCP/IP连接。
  - unix:postgresql://*host* [:*port*] / [*database_name*] [? *connection_option*]：通过Unix域套接字连接。
  - tcp:postgresql://*host* [:*port*] / [*database_name*] [? *connection_option*]：通过TCP/IP连接。
  - SQL string constant：包含上述形式之一的值。

- **connection_name**

  用于该连接的一个可选标识符，可以在其他命令中引用它。可以是一个SQL标识符或者一个宿主变量。

- **connection_user**

  用于数据库连接的用户名。

  使用*user_name*/*password*、*user_name* SQLIDENTIFIED BY *password*或者*user_name* USING *password*之一，这个参数也能指定用户名和密码。

  用户名和密码可以是SQL标识符、字符串常量或者宿主变量。

说明

上述参数中斜体部分为变量，请根据实际情况进行修改。

示例

指定连接参数变体的示例：

```
EXEC SQL CONNECT TO "connectdb" AS main;
EXEC SQL CONNECT TO "connectdb" AS second;
EXEC SQL CONNECT TO 'connectdb' AS main;
EXEC SQL CONNECT TO REGRESSDB1 as main;
EXEC SQL CONNECT TO connectdb AS :id;
EXEC SQL CONNECT TO connectdb AS main USER connectuser/connectdb;
EXEC SQL CONNECT TO connectdb AS main USER connectuser USING "connectdb";
EXEC SQL CONNECT TO connectdb AS main;
EXEC SQL CONNECT TO tcp:postgresql://localhost/connectdb USER connectuser IDENTIFIED BY connectpw;
EXEC SQL CONNECT TO tcp:postgresql://localhost:$PORT/connectdb USER connectuser SQLIDENTIFIED BY
connectpw;
EXEC SQL CONNECT TO unix:postgresql://localhost/connectdb USER connectuser SQLIDENTIFIED BY
"connectpw";
EXEC SQL CONNECT TO unix:postgresql://localhost/connectdb USER connectuser USING "connectpw";
```

连接语法使用示例：

```
#include <stdlib.h>
#include <string.h>
#include <stdlib.h>
```



```
#include <stdio.h>

int main(void)
{
// 宿主变量定义，定义连接串所需的database、password等字段，实际值应从环境变量或配置文件读取，环境变量需用户自己按需配置；非环境变量情况下可直接赋值字符串。
exec sql begin declare section;
    const int max_str_len = 200;
    char db[max_str_len] = getenv("EXAMPLE_DATABASENAME_ENV");
    char pw[max_str_len] = getenv("EXAMPLE_PASSWD_ENV");
    char new_pw[max_str_len] = getenv("EXAMPLE_NEW_PASSWD_ENV");
exec sql end declare section;

    // 打印调试日志。
    ECPGdebug(1, stderr);

    // 连接语句涉及数据库、用户、密码。需提前创建好并有相关操作权限。

    // 连接方式：EXEC SQL CONNECT TO [ database_name ][ @host ][ :port ] [ USER connection_user ]
    // case1: 使用默认的本地连接方式，连接数据库为postgres库。
    exec sql connect to postgres;
    // case2：使用默认的本地连接方式，连接数据库为postgres库, 连接别名为conn1。
    exec sql connect to postgres as conn1;
    // case3：使用ip+port方式(localhost数据库监听的本地地址，$PORT为数据库监听端口)，连接数据库为connectdb库, 指定数据库别名，指定用户密码。
    exec sql connect to connectdb@localhost:$PORT as conn2 user connectuser using :pw;
    // case4：使用ip+port方式(127.0.0.1数据库监听的本地地址，$PORT为数据库监听端口)，连接数据库为connectdb库, 指定数据库别名，指定用户密码。
    exec sql connect to connectdb@127.0.0.1:$PORT as conn3 user connectuser sqlidentified by :pw;
    // case5： 关闭数据库连接。
    exec sql disconnect postgres;
    exec sql disconnect conn1;
    exec sql disconnect conn2;
    exec sql disconnect conn3;

    // 连接方式：EXEC SQL CONNECT TO <tcp|unix>:<gaussdb|postgresql>://host [ :port ]/[ database_name ] [ ?connection_option ]
    // case1: 通过宿主变量pw、db方式，替换url变量。
    strcpy(pw, new_pw);
    strcpy(db, "tcp:postgresql://localhost/connectdb");
    exec sql connect to :db user connectuser using :pw;
    // case2: 其中127.0.0.1为数据库监听ip,connectdb为数据库database。
    exec sql connect to tcp:postgresql://127.0.0.1/connectdb as conn4 user connectuser using :pw;
    // case3:其中127.0.0.1为数据库监听ip,connectdb为数据库database,connect_timeout=14为连接串配置参数。
    exec sql connect to tcp:gaussdb://localhost/connectdb?connect_timeout=14 as conn5 user connectuser sqlidentified by :pw;
    // case4: 关闭所有连接。
    exec sql close all;

    // 连接数据库，并执行业务
    exec sql connect to tcp:postgresql://127.0.0.1/connectdb as conn4 user connectuser using :pw;
    exec sql set autocommit = on;
    exec sql create table t1(a int);
    exec sql insert into t1 values(1),(2);
    exec sql select a from t1 where a > 1;
    exec sql drop table t1;
    exec sql disconnect current;
    return 0;
}
```

使用宿主变量指定连接参数的示例：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
EXEC SQL BEGIN DECLARE SECTION;
    /* 此处dbname、user、pwd应从环境变量或配置文件读取，环境变量需用户自己按需配置；非环境变量情况下可直接赋值字符串 */
```



```
    char *dbname     = getenv("EXAMPLE_DBNAME_ENV");    /* 数据库名 */
    char *user       = getenv("EXAMPLE_USERNAME_ENV");  /* 连接用户名 */
    char *pwd        = getenv("EXAMPLE_PASSWD_ENV"); /* 密码 */
    char *connection = "tcp:postgresql://localhost:$PORT/testdb";  /* 连接字符串 */
    char ver[256];                  /* 存储版本字符串的缓冲区 */
EXEC SQL END DECLARE SECTION;

    ECPGdebug(1, stderr);
    EXEC SQL CONNECT TO :dbname;
    EXEC SQL SELECT pg_catalog.set_config('search_path', '', false); EXEC SQL COMMIT;
    EXEC SQL SELECT version() INTO :ver;
    EXEC SQL DISCONNECT;

    printf("version: %s\n", ver);
    EXEC SQL CONNECT TO :connection USER :user USING :pwd;
    EXEC SQL SELECT pg_catalog.set_config('search_path', '', false); EXEC SQL COMMIT;
    EXEC SQL SELECT version() INTO :ver;
    EXEC SQL DISCONNECT;

    printf("version: %s\n", ver);
    return 0;
}
```

相关链接

DISCONECT，**SET CONNECTION**

5.9.5.5.3 DEALLOCATE DESCRIPTOR

功能描述

释放SQL描述符区域。

语法格式

```
DEALLOCATE DESCRIPTOR name
```

参数说明

name

SQL描述符名称。大小写敏感，是一个SQL标识符或一个宿主变量。

示例

```
DEALLOCATE DESCRIPTOR mydesc;
```

相关链接

ALLOCATE DESCRIPTOR，**GET DESCRIPTOR**，**SET DESCRIPTOR**

5.9.5.5.4 DECLARE

功能描述

声明一个游标用于迭代预备语句的结果集。该命令与SQL命令的DECLARE在语义上略有不同，后者执行查询并准备结果集以便检索，而嵌入式SQL命令只是将一个名称声明为“循环变量”并用于在查询的结果集上迭代，实际在使用OPEN命令打开游标时执行。

语法格式

```
DECLARE cursor_name [ BINARY ]  [ NO SCROLL ] CURSOR [ { WITH | WITHOUT } HOLD ] FOR
prepared_name
DECLARE cursor_name [ BINARY ]  [ NO SCROLL ] CURSOR [ { WITH | WITHOUT } HOLD ] FOR query
```

参数说明

- **cursor_name**
  游标名称，大小写敏感。可以是一个SQL标识符或者一个宿主变量。
- **prepared_name**
  预备查询的名称。可以是一个SQL标识符或者一个宿主变量。
- **query**
  提供游标要返回的行的SELECT命令。

说明

游标选项的含义请参见**DECLARE**。

示例

声明用于查询的游标示例：

```
EXEC SQL DECLARE C CURSOR FOR SELECT * FROM My_Table;
EXEC SQL DECLARE C CURSOR FOR SELECT Item1 FROM T;
EXEC SQL DECLARE cur1 CURSOR FOR SELECT version();
```

声明用于预备语句的游标示例：

```
EXEC SQL PREPARE stmt1 AS SELECT version();
EXEC SQL DECLARE cur1 CURSOR FOR stmt1;
```

相关链接

OPEN

5.9.5.5.5 DESCRIBE

功能描述

检索预准备语句中包含的结果列的元数据信息。

语法格式

```
DESCRIBE [ OUTPUT ] prepared_name USING  SQL  DESCRIPTOR descriptor_name
DESCRIBE [ OUTPUT ] prepared_name INTO SQL  DESCRIPTOR descriptor_name
DESCRIBE [ OUTPUT ] prepared_name INTO sqlda_name
```

参数说明

- **prepared_name**
  预备语句名称可以是一个SQL标识符或者宿主变量。
- **descriptor_name**
  描述符名称，大小写敏感。可以是SQL标识符或者宿主变量。
- **sqlda_name**
  SQLDA变量名称，详细使用请参见**SQLDA**。

示例

```
EXEC SQL ALLOCATE DESCRIPTOR mydesc;
EXEC SQL PREPARE stmt1 FROM :sql_stmt;
EXEC SQL DESCRIBE stmt1 INTO SQL DESCRIPTOR mydesc;
EXEC SQL GET DESCRIPTOR mydesc VALUE 1 :charvar = NAME;
EXEC SQL DEALLOCATE DESCRIPTOR mydesc;
```

相关链接

ALLOCATE DESCRIPTOR，**GET DESCRIPTOR**

5.9.5.5.6 DISCONECT

功能描述

关闭一个（或所有）与数据库的连接。

语法格式

```
DISCONNECT connection_name
DISCONNECT [ CURRENT ]
DISCONNECT DEFAULT
DISCONNECT ALL
```

参数说明

- **connection_name**

  由CONNECT命令建立的数据库连接名称。

- **current**

  关闭“当前的”连接，它可以是最近打开的连接或者是由SET CONNECTION命令设置的连接。如果没有参数被传给DISCONNECT命令，它作为默认值。

- **default**

  关闭默认连接。

- **all**

  关闭所有打开的连接。

示例

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
    /* 需要提前创建testdb库 */
    EXEC SQL CONNECT TO testdb AS DEFAULT USER testuser;
    EXEC SQL CONNECT TO testdb AS con1 USER testuser;
    EXEC SQL CONNECT TO testdb AS con2 USER testuser;
    EXEC SQL CONNECT TO testdb AS con3 USER testuser;
    EXEC SQL DISCONNECT CURRENT;  /* 关闭 con3          */
    EXEC SQL DISCONNECT DEFAULT;  /* 关闭 DEFAULT       */
    EXEC SQL DISCONNECT ALL;      /* 关闭 con2 以及 con1 */
    return 0;
}
```


相关链接

CONNECT，SET CONNECTION

5.9.5.5.7 EXECUTE IMMEDIATE

功能描述

预备并且执行动态指定的SQL语句，不检索结果行。

语法格式

```
EXECUTE IMMEDIATE string
```

参数说明

string

包含要被执行的SQL语句的C字符串或者宿主变量。

示例

使用EXECUTE IMMEDIATE和名为command的宿主变量执行INSERT语句：

```
sprintf(command, "INSERT INTO test (name, amount, letter) VALUES ('db: ''r1''', 1, 'f')");
EXEC SQL EXECUTE IMMEDIATE :command;
```

5.9.5.5.8 GET DESCRIPTOR

功能描述

检索查询结果集的信息，并且将它存储到宿主变量中。在使用该命令将信息传递给宿主语言变量之前通常使用FETCH或者SELECT填充标识符区域。该命令有两种形式：

- 检索描述符的“头部”项，适用于全面查看结果集。
- 列号作为附加参数，检索特定列的信息。

语法格式

```
GET DESCRIPTOR descriptor_name VALUE column_number :cvariable = descriptor_item [, ... ]
GET DESCRIPTOR descriptor_name:cvariable = descriptor_header_item [, ... ]
```

参数说明

- **descriptor_name**
  描述符名称。
- **descriptor_header_item**
  标识要检索哪一个头部项信息。当前仅支持用于得到结果集中列数的COUNT。
- **column_number**
  关于被检索的列数信息。计数从1开始。
- **descriptor_item**
  标记识别检索列的信息项。
- **cvariable**
  宿主变量将接收从描述符区域检索的数据。

示例

检索结果集中列数：

```
EXEC SQL GET DESCRIPTOR d :d_count = COUNT;
```

在第一列中检索数据长度：

```
EXEC SQL GET DESCRIPTOR d VALUE 1 :d_returned_octet_length = RETURNED_OCTET_LENGTH;
```

检索作为字符串第二列的数据主体：

```
EXEC SQL GET DESCRIPTOR d VALUE 2 :d_data = DATA;
```

执行SELECT current_database();并且显示列数、列数据长度和列数据的完整过程示例：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
EXEC SQL BEGIN DECLARE SECTION;
    int  d_count;
    char d_data[1024];
    int  d_returned_octet_length;
EXEC SQL END DECLARE SECTION;
    EXEC SQL CONNECT TO testdb AS con1 USER testuser;
    EXEC SQL SELECT pg_catalog.set_config('search_path', '', false); EXEC SQL COMMIT;
    EXEC SQL ALLOCATE DESCRIPTOR d;

    /* 描述、打开一个游标，并且分配一个描述符给该游标  */
    EXEC SQL DECLARE cur CURSOR FOR SELECT current_database();
    EXEC SQL OPEN cur;
    EXEC SQL FETCH NEXT FROM cur INTO SQL DESCRIPTOR d;

    /* 得到全部列的数量 */
    EXEC SQL GET DESCRIPTOR d :d_count = COUNT;
    printf("d_count = %d\n", d_count);

    /* 得到一个返回列的长度 */
    EXEC SQL GET DESCRIPTOR d VALUE 1 :d_returned_octet_length = RETURNED_OCTET_LENGTH;
    printf("d_returned_octet_length = %d\n", d_returned_octet_length);

    /* 将返回的列取出成一个字符串 */
    EXEC SQL GET DESCRIPTOR d VALUE 1 :d_data = DATA;
    printf("d_data = %s\n", d_data);

    /* 关闭 */
    EXEC SQL CLOSE cur;
    EXEC SQL COMMIT;

    EXEC SQL DEALLOCATE DESCRIPTOR d;
    EXEC SQL DISCONNECT ALL;
    return 0;
}
```

该示例执行结果为：

```
d_count              = 1
d_returned_octet_length = 6
d_data               = testdb
```

相关链接

ALLOCATE DESCRIPTOR，**DEALLOCATE DESCRIPTOR**，**SET DESCRIPTOR**

5.9.5.5.9 OPEN

功能描述

打开一个游标，并将实际值选择性地绑定到游标声明中的占位符。该游标必须事先使用DECLARE命令声明过。执行OPEN命令会触发在服务器上开始执行查询。

语法格式

```
OPEN cursor_name
OPEN cursor_name USING value [, ... ]
OPEN cursor_name USING SQL DESCRIPTOR descriptor_name
```

参数说明

- **cursor_name**

  被打开的游标的名称。可以是一个SQL标识符或者一个宿主变量。

- **value**

  被绑定到游标中一个占位符的值。可以是一个SQL常量、一个宿主变量或者一个带有指示符的宿主变量。

- **descriptor_name**

  包含要绑定到游标中占位符的值的描述符名称。可以是一个SQL标识符或者一个宿主变量。

示例

```
EXEC SQL OPEN a;
EXEC SQL OPEN d USING 1, 'test';
EXEC SQL OPEN c1 USING SQL DESCRIPTOR mydesc;
EXEC SQL OPEN :curname1;
```

相关链接

DECLARE

5.9.5.5.10 PREPARE

功能描述

准备用于执行的语句。

语法格式

```
PREPARE name FROM string
```

参数说明

- **name**

  预备查询标识符。

- **string**

  包含预备语句的文本C字符串或者宿主变量，预备语句包含SELECT、INSERT、UPDATE或者DELETE命令之一。

示例

```
char *stmt = "SELECT * FROM test1 WHERE a = ? AND b = ?";
EXEC SQL ALLOCATE DESCRIPTOR outdesc;
EXEC SQL PREPARE foo FROM :stmt;
EXEC SQL EXECUTE foo USING SQL DESCRIPTOR indesc INTO SQL DESCRIPTOR outdesc;
```

须知

ecpg提供的prepare语句不直接等同于内核提供的prepare语法，举例如下：

GaussDB Kernel内核语法：

```
PREPARE name [ ( data_type [, ...] ) ] AS statement
```

嵌入式SQL语句：

```
EXEC SQL PREPARE I（int, int） AS INSERT INTO T VALUES ( $1, $2 );
EXEC SQL EXECUTE I(1, 2);
```

执行上述语句出现报错时，提示too few arguments on。ecpg提供动态SQL解决PREPARE name [(data_type [, ...])] AS statement语法场景的使用。

利用动态SQL语法规则解决上述问题：

```
EXEC SQL PREPARE I AS INSERT INTO T VALUES ( $1, $2 );
EXEC SQL EXECUTE I using 1, 2;
```

5.9.5.5.11 SET AUTOCOMMIT

功能描述

设置当前数据库会话的自动提交行为。默认情况下，嵌入式SQL程序不自动提交，因此需要显式地发出COMMIT。这个命令可以把会话改成自动提交模式，这样每一个单独的语句都会被隐式提交。

语法格式

```
SET AUTOCOMMIT { = | TO } { ON | OFF }
```

5.9.5.5.12 SET CONNECTION

功能描述

设置一个数据库连接。

语法格式

```
SET CONNECTION [ TO | = ] connection_name
```

参数说明

- **connection_name**

  通过CONNECT命令创建数据库连接名字。

示例

```
EXEC SQL SET CONNECTION TO con2;
EXEC SQL SET CONNECTION = con1;
```


相关链接

CONNECT，DISCONECT

5.9.5.5.13 SET DESCRIPTOR

功能描述

在SQL描述符区域中设置信息，描述符区域通常用于绑定预备查询执行中的参数。该命令有两种形式：

- 适用于描述符“头部”，它独立于特定的数据。
- 为由数字标识的特定数据赋值。

语法格式

```
SET DESCRIPTOR descriptor_name descriptor_header_item = value [, ... ]
SET DESCRIPTOR descriptor_name VALUE number descriptor_item = value [, ...]
```

参数说明

- **descriptor_name**
  描述符名称。
- **descriptor_header_item**
  标记识别设置的头部信息项。目前仅仅支持COUNT设置描述符项数。
- **number**
  设置的描述符项数。计数从1开始。
- **descriptor_item**
  标记识别在描述符中的项信息。当前仅支持DATA、TYPE和LENGTH。
- **value**
  存储到描述符项中的值。可以是SQl常量或者宿主变量。

示例

```
EXEC SQL SET DESCRIPTOR indesc COUNT = 1;
EXEC SQL SET DESCRIPTOR indesc VALUE 1 DATA = 2;
EXEC SQL SET DESCRIPTOR indesc VALUE 1 DATA = :val1;
EXEC SQL SET DESCRIPTOR indesc VALUE 2 INDICATOR = :val1, DATA = 'some string';
EXEC SQL SET DESCRIPTOR indesc VALUE 2 INDICATOR = :val2null, DATA = :val2;
```

相关链接

ALLOCATE DESCRIPTOR，DEALLOCATE DESCRIPTOR，GET DESCRIPTOR

5.9.5.5.14 TYPE

功能描述

定义一个新的数据类型。当运行带有-c选项的ecpg的时候，仅仅标识该命令。

语法格式

```
TYPE type_name IS ctype
```

参数说明

type_name

数据类型名称。

ctype

C语言数据类型说明。

示例

```
EXEC SQL TYPE customer IS
    struct
    {
        varchar name[50];
        int     phone;
    };

EXEC SQL TYPE cust_ind IS
    struct ind
    {
        short   name_ind;
        short   phone_ind;
    };

EXEC SQL TYPE c IS char reference;
EXEC SQL TYPE ind IS union { int integer; short smallint; };
EXEC SQL TYPE intarray IS int[AMOUNT];
EXEC SQL TYPE str IS varchar[BUFFERSIZ];
EXEC SQL TYPE string IS char[11];
```

使用EXEC SQL TYPE的示例（注意以下示例使用时在ecpg预处理阶段需要添加-c参数）：

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

EXEC SQL WHENEVER SQLERROR SQLPRINT;
EXEC SQL TYPE tt IS
    struct
    {
        varchar v[256];
        int     i;
    };
EXEC SQL TYPE tt_ind IS
    struct ind {
        short   v_ind;
        short   i_ind;
    };

int main(void)
{
EXEC SQL BEGIN DECLARE SECTION;
    tt t;
    tt_ind t_ind;
EXEC SQL END DECLARE SECTION;
     EXEC SQL CONNECT TO testdb AS con1;
     EXEC SQL SELECT pg_catalog.set_config('search_path', '', false); EXEC SQL COMMIT;
     EXEC SQL SELECT current_database(), 256 INTO :t:t_ind LIMIT 1;

     printf("t.v = %s\n", t.v.arr);
     printf("t.i = %d\n", t.i);

     printf("t_ind.v_ind = %d\n", t_ind.v_ind);
     printf("t_ind.i_ind = %d\n", t_ind.i_ind);

     EXEC SQL DISCONNECT con1;
```



```
    return 0;
}
```

该示例输出为：

```
t.v = testdb
t.i = 256
t_ind.v_ind = 0
t_ind.i_ind = 0
```

5.9.5.5.15 VAR

功能描述

将新的C数据类型分配给宿主变量。宿主变量必须预先在声明段声明。

说明

- 对于VAR的用法需要谨慎。使用VAR语句后数据类型的变化可能会导致内存地址无效，从而导致数据变量无效，出现无法成功赋值的场景。
- 若在宿主变量声明段中确定好数据类型，则无须使用VAR语句。

语法格式

```
VAR varname IS ctype
```

参数说明

- **varname**
  C语言变量名称。
- **ctype**
  C语言类型说明。

示例

```
EXEC SQL BEGIN DECLARE SECTION;
   short a;
EXEC SQL END DECLARE SECTION;
EXEC SQL VAR a IS int;
```

5.9.5.5.16 WHENEVER

功能描述

定义一个行为，它会在SQL执行异常时（行未找到、SQL告警或错误）被调用。

语法格式

```
WHENEVER { NOT FOUND | SQLERROR | SQLWARNING } action
```

参数说明

参数描述请参见**设置回调**章节。

示例

```
EXEC SQL WHENEVER NOT FOUND CONTINUE;
EXEC SQL WHENEVER NOT FOUND DO BREAK;
```

```
EXEC SQL WHENEVER SQLWARNING SQLPRINT;
EXEC SQL WHENEVER SQLWARNING DO warn();
EXEC SQL WHENEVER SQLERROR sqlprint;
EXEC SQL WHENEVER SQLERROR SQLCALL print2();
EXEC SQL WHENEVER SQLERROR DO handle_error("select");
EXEC SQL WHENEVER SQLERROR DO sqlnotice(NULL, NONO);
EXEC SQL WHENEVER SQLERROR DO sqlprint();
EXEC SQL WHENEVER SQLERROR GOTO error_label;
EXEC SQL WHENEVER SQLERROR STOP;
```

使用WHENEVER NOT FOUND BREAK来处理结果集的循环：

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

int main(void)
{
    EXEC SQL CONNECT TO testdb AS con1;
    EXEC SQL SELECT pg_catalog.set_config('search_path', '', false); EXEC SQL COMMIT;
    EXEC SQL ALLOCATE DESCRIPTOR d;
    EXEC SQL DECLARE cur CURSOR FOR SELECT current_database(), 'hoge', 256;
    EXEC SQL OPEN cur;

    /* 当到达结果集末尾时，跳出循环 */
    EXEC SQL WHENEVER NOT FOUND DO BREAK;

    while (1)
    {
        EXEC SQL FETCH NEXT FROM cur INTO SQL DESCRIPTOR d;
        ...
    }
     EXEC SQL CLOSE cur;
     EXEC SQL COMMIT;
     EXEC SQL DEALLOCATE DESCRIPTOR d;
     EXEC SQL DISCONNECT ALL;
     return 0;
}
```

5.9.6 查询结果集

- 返回单行结果的SELECT语句可以直接使用EXEC SQL执行，请参见**执行SQL语句**章节。

  示例：

  ```
  /* 首先建立一个表并插入数据 */
  EXEC SQL CREATE TABLE test_table (number1 integer, number2 integer);
  EXEC SQL INSERT INTO test_table (number1, number2) VALUES (2, 1);

  /* 查询结果为单行,:num 为宿主变量 */
  EXEC SQL SELECT number1 INTO :num FROM test_table WHERE number2 = 1;
  ```

- 若要处理多行结果集，则必须使用游标，请参见**使用游标**章节（特殊情况下，应用程序可以一次取出多行结果写入到数组类型的宿主变量中，请参见**使用非初级类型的宿主变量**章节）。

  示例：

  ```
    /* 首先建立一个表并插入数据 */
    EXEC SQL CREATE TABLE test_table (number1 integer, number2 integer);
    EXEC SQL INSERT INTO test_table (number1, number2) VALUES (2, 1);
    EXEC SQL INSERT INTO test_table (number1, number2) VALUES (3, 1);
    EXEC SQL INSERT INTO test_table (number1, number2) VALUES (4, 1);
    EXEC SQL INSERT INTO test_table (number1, number2) VALUES (5, 1);

  /* 定义宿主变量 */
  EXEC SQL BEGIN DECLARE SECTION;
    int v1;
    int v2;
  EXEC SQL END DECLARE SECTION;
  ```

```
    /* 声明游标 */
    EXEC SQL DECLARE test_bar CURSOR FOR SELECT number1, number2 FROM test_table ORDER BY
number1;
    /* 打开游标 */
    EXEC SQL OPEN test_bar;
    /* 当游标到达结果集末尾时跳出循环 */
    EXEC SQL WHENEVER NOT FOUND DO BREAK;
    /* 获取查询结果集 */
    while(1)
    {
        EXEC SQL FETCH NEXT FROM test_bar INTO :v1, :v2;
        printf("number1 = %d, number2 = %d\n",v1,v2);
    }
    /* 关闭游标 */
    EXEC SQL CLOSE test_bar;
```

5.9.7 关闭数据库连接

数据库使用完成后关闭数据库连接。

使用如下语句关闭连接：

```
EXEC SQL DISCONNECT [connection];
```

connection可通过如下方法声明：

- connection-name（连接名）
- default（缺省）
- current（当前）
- all（所有）

5.9.8 宿主变量

本节详细介绍如何在C语言程序和嵌入式SQL程序之间使用宿主变量传递数据。在嵌入式SQL-C程序中，将C语言作为宿主语言，将EXEC SQL [Command]语句认为是宿主语言的嵌入式SQL，因此将C语言程序中用于嵌入式SQL语句的变量称为宿主变量。

5.9.8.1 概述

在嵌入式SQL中进行C语言程序和SQL语句之间的数据传递不需要把数据粘贴到语句中，只需要在SQL语句里写上C语言变量的名称，前缀加一个冒号即可。示例如下：

```
EXEC SQL INSERT INTO sometable VALUES (:v1, 'foo', :v2);
```

这个语句引用了两个C语言变量：v1和v2，并且使用一个普通的SQL字串文本，这表明一条SQL语句内并不限制只使用某一种数据。

5.9.8.2 声明段

要实现嵌入式SQL-C程序和数据库间的数据交互（例如：从SQL-C程序把查询语句中的参数传递给数据库，或者从数据库向嵌入式SQL-C程序传回数据），需要在特殊的标记段里面声明包含此数据的C语言变量，以便预处理器能够识别。

标记段以下面的代码开始：

```
EXEC SQL BEGIN DECLARE SECTION;
```

以下面的代码结束：

```
EXEC SQL END DECLARE SECTION;
```

在此之间，必须有常规的C语言变量声明，比如：


```
int   x = 4;
char  foo[16], bar[16];
```

须知

- 标记段代码开始和结束之间声明的宿主变量类型必须为当前支持的数据类型，请参见表5-95。
- 可以隐式地创建一个声明段声明变量：EXEC SQL int i = 4。
- 不在SQL命令里使用的变量可以在特殊的声明段外面声明。
- 结构体或者联合体的定义也必须在DECLARE段中列出，否则预处理器就无法处理这些类型。

5.9.8.3 检索查询

对于常用的检索查询，嵌入式SQL提供了常规命令SELECT和FETCH的特殊变体。这些命令使用特殊的INTO子句，用以指定检索出来的数值存储在哪些宿主变量里。SELECT用于返回单行的查询，FETCH用于使用游标返回多行的查询。

- 使用SELECT

```
/*
 * 假定有这个表：
 * CREATE TABLE test1 (a int, b varchar(50));
 */
EXEC SQL BEGIN DECLARE SECTION;
    int v1;
    VARCHAR v2;
EXEC SQL END DECLARE SECTION;

    ...

    EXEC SQL SELECT a, b INTO :v1, :v2 FROM test;
```

INTO子句出现在选择列表和FROM子句之间。选择列表和INTO后面列表的元素（也叫目标列表）个数必须相同。

- 使用FETCH

```
EXEC SQL BEGIN DECLARE SECTION;
    int v1;
    VARCHAR v2;
EXEC SQL END DECLARE SECTION;
    ...

    EXEC SQL DECLARE foo CURSOR FOR SELECT a, b FROM test;
    ...
    do
    {
        ...
        EXEC SQL FETCH NEXT FROM foo INTO :v1, :v2;
        ...
    } while (...);
```

这里的INTO子句出现在所有SQL子句后面。

5.9.8.4 类型映射

当ecpg应用程序在GaussDB Kernel服务器和C语言程序之间交换值时（例如：从服务器检索查询结果或者执行带有输入参数的SQL语句），在GaussDB Kernel数据类型和宿主语言变量类型（具体的C语言数据类型）之间需要进行值的转换。有两种数据类型可以使用：简单的GaussDB Kernel数据类型，如integer和text，可以直接被应用程序

读取和写入。其他GaussDB Kernel数据类型，如timestamp和numeric，只能通过特殊库函数进行访问，请参见**ecpg接口参考**章节。

表 5-95 GaussDB Kernel 数据类型和 C 变量类型之间的映射

GaussDB Kernel数据类型	宿主变量数据类型
smallint	short
integer	int
bigint	long long int
boolean	boolean
character(*n*), varchar(n), text	char[n+1], VARCHAR[n+1]
double precision	double
real	float
smallserial	short
serial	int
bigserial	long long int
oid	unsigned int
name	char[NAMEDATALEN]
date	date [a]
timestamp	timestamp [a]
interval	interval [a]
decimal	decimal [a]
numeric	numeric [a]

说明

[a]这种类型可以通过**访问特殊数据类型**访问。

须知

- 当前仅支持对于C语言的基本数据类型的使用或者组合，不支持C++语言中string数据类型用作宿主变量数据类型。
- 当前ecpg仅对GaussDB Kernel SQL的常用数据类型做映射，具体支持项请参见**表5-95**。

5.9.8.5 处理字符串

处理SQL字符串数据类型（例如：varchar、text），有两种方式来声明宿主变量：

1. 方式一：使用char[]（一个char字符串），C语言程序中处理字符数据最常见的方式。

```
EXEC SQL BEGIN DECLARE SECTION;
    char str[50];
EXEC SQL END DECLARE SECTION;
```

注意字符串必须控制长度，如果上述示例的宿主变量用作存放查询结果且查询命令返回的字符串长度超过49字节，那么将会发生缓冲区溢出。

2. 方式二：使用VARCHAR类型，ecpg提供的一种特殊类型。在一个VARCHAR类型数组上的定义会被转变成一个struct类型。如下声明：

```
VARCHAR var[180];
```

会被转变成：

```
struct varchar_var
{
    int len;
    char arr[180];
} var;
```

要在一个VARCHAR宿主变量中存储一个字符串，该宿主变量必须被声明为包含零字节为终止符长度的字符串。字段arr存放以零字节为终止符的字符串，字段len保存存储在arr中的字符串的长度，计算长度时不包括终止符。当宿主变量被用于一个查询的输入时，如果strlen(arr)和len结果不同，将使用较短的那一个。

⚠ 注意

- VARCHAR可以被写成大写或小写形式，但是不能大小写混合。
- char和VARCHAR类型宿主变量也可以保存其他SQL类型的值，它们将被存储为字符串形式。

5.9.8.6 使用非初级类型的宿主变量

非初级类型的宿主变量包括数组、typedef、结构体和指针类型的宿主变量。

- 数组

  有两种将数组作为宿主变量的情况。第一种情况是在char[]或者VARCHAR[]中存储一些文本字符串。第二种情况是可在检索多行查询结果时不使用游标。如果不使用数组，则处理多行查询结果时必须使用游标以及FETCH命令。但是如果使用数组类型作为宿主变量，则一次可以检索多行。数组长度需能容纳查询结果的所有行，否则可能会发生缓冲区溢出。

  下面示例为扫描pg_database系统表并且显示所有可用数据库的OID和名称：

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
EXEC SQL BEGIN DECLARE SECTION;
    int dbid[8];
    char dbname[8][16];
    int i;
EXEC SQL END DECLARE SECTION;

    memset(dbname, 0, sizeof(char)* 16 * 8);
    memset(dbid, 0, sizeof(int) * 8);
    * 连接到testdb，需提前创建上testdb库 */
    EXEC SQL CONNECT TO testdb;
    /* 一次检索多行到数组中。 */
```

```
    EXEC SQL SELECT oid,datname INTO :dbid, :dbname FROM pg_database;
    for (i = 0; i < 8; i++)
        printf("oid=%d, dbname=%s\n", dbid[i], dbname[i]);
    EXEC SQL COMMIT;
    EXEC SQL DISCONNECT ALL;
    return 0;
}
```

示例输出（具体值取决于本地环境）：

```
oid=1, dbname=template1
oid=11510, dbname=template0
oid=11511, dbname=postgres
oid=313780, dbname=testdb
oid=0, dbname=
oid=0, dbname=
oid=0, dbname=
```

- 结构体

  结构体成员变量可用来匹配查询结果列的名称，该结构能在一个宿主变量中处理多列值。

  以下示例从pg_database系统表以及使用pg_database_size()函数检索可用数据库的OID、名称和尺寸。在此示例中，按照结构体的成员名匹配SELECT结果的每一列，从而不必把多个宿主变量放在FETCH语句中。

```
EXEC SQL BEGIN DECLARE SECTION;
    typedef struct
    {
        int oid;
        char datname[65];
        long long int size;
    } dbinfo_t;

    dbinfo_t dbval;
EXEC SQL END DECLARE SECTION;
    memset(&dbval, 0, sizeof(dbinfo_t));

    EXEC SQL DECLARE cur1 CURSOR FOR SELECT oid, datname, pg_database_size(oid) AS size FROM
pg_database;
    EXEC SQL OPEN cur1;

    /* 在达到结果集末尾时，跳出 while 循环 */
    EXEC SQL WHENEVER NOT FOUND DO BREAK;

    while (1)
    {
        /* 将多列取到一个结构体中。 */
        EXEC SQL FETCH FROM cur1 INTO :dbval;
        /* 打印该结构体的成员。 */
        printf("oid=%d, datname=%s, size=%lld\n", dbval.oid, dbval.datname, dbval.size);
    }
    EXEC SQL CLOSE cur1;
```

示例输出（具体值取决于本地环境）：

```
oid=1, datname=template1, size=4324580
oid=11510, datname=template0, size=4243460
oid=11511, datname=postgres, size=4324580
oid=313780, datname=testdb, size=8183012
```

结构体宿主变量可将查询结果的部分列转化成结构体字段，其他的查询结果列可以被分配给其它宿主变量。上述的示例也可以使用结构体外部的size宿主变量重新构造：

```
EXEC SQL BEGIN DECLARE SECTION;
    typedef struct
    {
        int oid;
        char datname[65];
    } dbinfo_t;

    dbinfo_t dbval;
```



```
    long long int size;
EXEC SQL END DECLARE SECTION;

    memset(&dbval, 0, sizeof(dbinfo_t));

    EXEC SQL DECLARE cur1 CURSOR FOR SELECT oid, datname, pg_database_size(oid) AS size FROM
pg_database;
    EXEC SQL OPEN cur1;

    /* 在达到结果集末尾时，跳出 while 循环 */
    EXEC SQL WHENEVER NOT FOUND DO BREAK;
    while (1)
    {
        /* 将多列取到一个结构中。 */
        EXEC SQL FETCH FROM cur1 INTO :dbval, :size;
        /* 打印该结构的成员。 */
        printf("oid=%d, datname=%s, size=%lld\n", dbval.oid, dbval.datname, size);
    }

    EXEC SQL CLOSE cur1;
```

- typedef

  使用typedef关键字将新类型映射到已有类型：

  ```
  EXEC SQL BEGIN DECLARE SECTION;
  typedef char mychartype[40];
  typedef long serial_t;
  EXEC SQL END DECLARE SECTION;
  ```

  也可以使用如下命令：

  ```
  EXEC SQL TYPE serial_t IS long;
  ```

  这种声明不需要写在一个声明段中。

- 指针

  可以声明常见类型的指针：

  ```
  EXEC SQL BEGIN DECLARE SECTION;
  int   *intp;
  char **charp;
  EXEC SQL END DECLARE SECTION;
  ```

5.9.8.7 访问特殊数据类型

ecpg支持numeric、decimal、date、timestamp和interval数据类型。由于这些数据类型的内部结构较为复杂，无法被映射到初级数据类型的宿主变量，因此应用程序通过声明特殊类型的宿主变量以及使用pgtypes库中的函数处理这些特殊类型。pgtypes库中的接口函数请参见**ecpg接口参考**章节。

- timestamp,date

  首先，程序必须包含timestamp类型的头文件：

  ```
  #include <pgtypes_timestamp.h>
  ```

  其次，在声明部分中声明类型为timestamp的宿主变量：

  ```
  EXEC SQL BEGIN DECLARE SECTION;
    timestamp ts;
  EXEC SQL END DECLARE SECTION;
  ```

  读取值到宿主变量之后，使用pgtypes库函数进行处理。如下示例中，使用PGTYPESTimestamp_to_asc()函数将timestamp值转换成文本形式：

  ```
  EXEC SQL SELECT now()::timestamp INTO :ts;
  printf("ts = %s\n", PGTYPESTimestamp_to_asc(ts));
  ```

  示例输出如下：

  ```
  ts = 2022-06-27 18:03:56.949343
  ```

  另外，date类型可以用相同的方式处理。程序必须包括pgtypes_data.h头文件，并声明一个date类型的宿主变量并用PGTYPEdata_to_asc()函数将其转换成文本形式。

- interval

  interval类型的处理与timestamp和date类型类似，不同的是interval类型变量的存储空间需要分配在堆内存中。

  示例如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <pgtypes_interval.h>
int main(void)
{
EXEC SQL BEGIN DECLARE SECTION;
    interval *in;
EXEC SQL END DECLARE SECTION;
    /* 连接数据库testdb, 需提前创建好testdb库 */
EXEC SQL CONNECT TO testdb;
    in = PGTYPESinterval_new();
    EXEC SQL SELECT '1 min'::interval INTO :in;
    printf("interval = %s\n", PGTYPESinterval_to_asc(in));
    PGTYPESinterval_free(in);
    EXEC SQL COMMIT;
    EXEC SQL DISCONNECT ALL;
    return 0;
}
```

- numeric,decimal

  numeric和decimal类型的处理类似于interval类型，需要定义一个指针，并在堆上分配一些内存空间并且使用pgtypes库函数访问该变量。

  示例如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <pgtypes_numeric.h>
EXEC SQL WHENEVER SQLERROR STOP;
int main(void)
{
EXEC SQL BEGIN DECLARE SECTION;
    numeric *num;
    numeric *num2;
    decimal *dec;
EXEC SQL END DECLARE SECTION;

    /* 连接数据库testdb, 需提前创建好testdb库 */
EXEC SQL CONNECT TO testdb;

    num = PGTYPESnumeric_new();
    dec = PGTYPESdecimal_new();

    EXEC SQL SELECT 12.345::numeric(4,2), 23.456::decimal(4,2) INTO :num, :dec;
    printf("numeric = %s\n", PGTYPESnumeric_to_asc(num, 0));
    printf("numeric = %s\n", PGTYPESnumeric_to_asc(num, 1));
    printf("numeric = %s\n", PGTYPESnumeric_to_asc(num, 2));
    /*转换十进制到数值型以显示十进制值*/
    num2 = PGTYPESnumeric_new();
    PGTYPESnumeric_from_decimal(dec, num2);
    printf("decimal = %s\n", PGTYPESnumeric_to_asc(num2, 0));
    printf("decimal = %s\n", PGTYPESnumeric_to_asc(num2, 1));
    printf("decimal = %s\n", PGTYPESnumeric_to_asc(num2, 2));
    PGTYPESnumeric_free(num2);
    PGTYPESdecimal_free(dec);
    PGTYPESnumeric_free(num);

    EXEC SQL COMMIT;
    EXEC SQL DISCONNECT ALL;
    return 0;
}
```


5.9.8.8 处理非初级 SQL 数据类型

本节介绍如何处理ecpg应用中非标量以及用户定义的SQL级别的数据类型。注意此处和**使用非初级类型的宿主变量**章节中介绍的对于非初级类型的宿主变量的处理不同。

- 数组

  ecpg不直接支持多维SQL级别数组。一维SQL数组可以被映射到C语言数组类型的宿主变量，反之亦然。但是在创建语句时，如果ecpg并不知道列的类型，则将无法检查C语言数组是否是SQL数组的输入。因此在处理SQL语句的输出时，ecpg需要检查两者是否都是数组。

  如果查询语句仅仅访问一个数组的元素，那么用一个能被映射到该元素类型的宿主变量即可。例如：假设列的类型是integer数组，可以使用int类型的宿主变量。如果元素类型是varchar或text，可以使用char[]或VARCHAR[]类型的宿主变量。

  示例如下：

  ```
  CREATE TABLE t3 (
      ii integer[]
  );
  testdb=> SELECT * FROM t3;
       ii
  -------------
   {1,2,3,4,5}
  (1 row)
  ```

  如下示例检索数组的第四个元素并且把它存储到一个int类型的宿主变量中：

  ```
  EXEC SQL BEGIN DECLARE SECTION;
      int ii;
  EXEC SQL END DECLARE SECTION;

      EXEC SQL DECLARE cur1 CURSOR FOR SELECT ii[4] FROM t3;
      EXEC SQL OPEN cur1;

      EXEC SQL WHENEVER NOT FOUND DO BREAK;

      while (1)
      {
          EXEC SQL FETCH FROM cur1 INTO :ii ;
          printf("ii=%d\n", ii);
      }
      EXEC SQL CLOSE cur1;
  ```

  示例输出：

  ```
  ii=4
  ```

  若要把多个数组元素映射到一个数组类型宿主变量中，数组列的每一个元素以及宿主变量数组的每一个元素都需单独处理。例如：

  ```
  EXEC SQL BEGIN DECLARE SECTION;
      int ii_a[8];
  EXEC SQL END DECLARE SECTION;

      EXEC SQL DECLARE cur1 CURSOR FOR SELECT ii[1], ii[2], ii[3], ii[4] FROM t3;
      EXEC SQL OPEN cur1;

      EXEC SQL WHENEVER NOT FOUND DO BREAK;
      while (1)
      {
          EXEC SQL FETCH FROM cur1 INTO :ii_a[0], :ii_a[1], :ii_a[2], :ii_a[3];
          ...
      }
  ```

  注意：

  ```
  EXEC SQL BEGIN DECLARE SECTION;
      int ii_a[8];
  EXEC SQL END DECLARE SECTION;

      EXEC SQL DECLARE cur1 CURSOR FOR SELECT ii FROM t3;
      EXEC SQL OPEN cur1;
  ```



```
EXEC SQL WHENEVER NOT FOUND DO BREAK;

while (1)
{
    /* 错误 */
    EXEC SQL FETCH FROM cur1 INTO :ii_a;
    ...
}
```

无法将一个数组类型的查询结果直接映射到数组类型的宿主变量。

- 组合类型

  ecpg不直接支持组合类型，例如：在struct中声明成员变量为date数据类型。但是可以单独访问每一个属性或者使用外部字符串表达。

  对于如下示例，可以单独访问每一个属性：

```
CREATE TYPE comp_t AS (intval integer, textval varchar(32));
CREATE TABLE t4 (compval comp_t);
INSERT INTO t4 VALUES ( (256, 'PostgreSQL') );
```

  如下示例程序单独检索类型comp_t的每一个属性：

```
EXEC SQL BEGIN DECLARE SECTION;
    int intval;
    varchar textval[33];
EXEC SQL END DECLARE SECTION;

    /* 将组合类型列的每一个元素放在 SELECT 列表中。 */
    EXEC SQL DECLARE cur1 CURSOR FOR SELECT (compval).intval, (compval).textval FROM t4;
    EXEC SQL OPEN cur1;

    EXEC SQL WHENEVER NOT FOUND DO BREAK;
    while (1)
    {
        /* 将组合类型列的每一个元素取到主变量中。 */
        EXEC SQL FETCH FROM cur1 INTO :intval, :textval;
        printf("intval=%d, textval=%s\n", intval, textval.arr);
    }
    EXEC SQL CLOSE cur1;
```

  FETCH命令中从SQL语句接收并存储的结果宿主变量可以被集中在一个结构体中。结构体类型的宿主变量的详情请参见**处理字符串**章节。如下示例中两个宿主变量intval和textval变成comp_t结构体的成员，并且该结构体在FETCH命令中被指定：

```
EXEC SQL BEGIN DECLARE SECTION;
    typedef struct
    {
        int intval;
        varchar textval[33];
    } comp_t;
    comp_t compval;
EXEC SQL END DECLARE SECTION;

    /* 将组合类型列的每一个元素放在 SELECT 列表中。 */
    EXEC SQL DECLARE cur1 CURSOR FOR SELECT (compval).intval, (compval).textval FROM t4;
    EXEC SQL OPEN cur1;

    EXEC SQL WHENEVER NOT FOUND DO BREAK;
    while (1)
    {
        /* 将 SELECT 列表中的所有值放入一个结构。 */
        EXEC SQL FETCH FROM cur1 INTO :compval;
        printf("intval=%d, textval=%s\n", compval.intval, compval.textval.arr);
    }
    EXEC SQL CLOSE cur1;
```

  虽然在FETCH命令中使用了结构体，但SELECT子句中的属性名依然要依次指定，为了更加方便，可以使用一个*来处理该组合类型值的所有属性，示例如下：

```
...
EXEC SQL DECLARE cur1 CURSOR FOR SELECT (compval).* FROM t4;
```



```
EXEC SQL OPEN cur1;

EXEC SQL WHENEVER NOT FOUND DO BREAK;

while (1)
{
    /* 将 SELECT 列表中的所有值放入一个结构。 */
    EXEC SQL FETCH FROM cur1 INTO :compval;
    printf("intval=%d, textval=%s\n", compval.intval, compval.textval.arr);
}
...
```

即便ecpg无法解析组合类型本身，也可以通过上述方法将组合类型映射到结构体中。

- 用户定义的基础类型

  ecpg使用宿主变量来存放查询结果时，只支持ecpg提供的数据类型，不支持自定义的数据类型。对于通过CREATE TYPE创建的数据类型，无法通过宿主变量找到映射关系。

  用户自定义类型可以使用外部字符串表达以及char[]或VARCHAR[]类型的宿主变量来处理。

  数据类型complex的外部字符串表达是(%lf,%lf)，被定义在函数complex_in()内。如下示例把复杂类型值(1,1)和(3,3)插入到列a和b，并且查询它们的值。

```
EXEC SQL BEGIN DECLARE SECTION;
    varchar a[64];
    varchar b[64];
EXEC SQL END DECLARE SECTION;
    EXEC SQL INSERT INTO test_complex VALUES ('(1,1)', '(3,3)');
    EXEC SQL DECLARE cur1 CURSOR FOR SELECT a, b FROM test_complex;
    EXEC SQL OPEN cur1;

    EXEC SQL WHENEVER NOT FOUND DO BREAK;

    while (1)
    {
        EXEC SQL FETCH FROM cur1 INTO :a, :b;
        printf("a=%s, b=%s\n", a.arr, b.arr);
    }
    EXEC SQL CLOSE cur1;
```

  示例输出：

```
a=(1,1), b=(3,3)
```

5.9.9 执行动态 SQL 语句

在大多数情况下，应用程序执行的SQL语句在编写应用程序时必须是已知的。但是在某些情况下，SQL语句是在运行时构造好的，或由外部源提供的。这种情况下不能将SQL语句直接嵌入到C语言源代码中，但是动态SQL语句支持通过一个字符串变量调用所提供的SQL语句。

5.9.9.1 执行没有结果集的语句

执行EXECUTE IMMEDIATE命令示例如下：

```
EXEC SQL BEGIN DECLARE SECTION;
    const char *stmt = "CREATE TABLE test1 (...);";
EXEC SQL END DECLARE SECTION;
    EXEC SQL EXECUTE IMMEDIATE :stmt;
```

EXECUTE IMMEDIATE可以用于不返回结果集的SQL语句，比如：DDL、INSERT、UPDATE和DELETE语句。但不能用这种方式执行检索数据的语句，比如：SELECT语句。

5.9.9.2 执行具有输入参数的语句

准备一个普通语句，通过替换参数（在想要替换参数的地方输入问号）执行它的特定版本。使用EXECUTE语句通过USING子句给定参数执行准备语句。示例如下：

```
EXEC SQL BEGIN DECLARE SECTION;
   const char *stmt = "INSERT INTO test1 VALUES(?, ?);";
EXEC SQL END DECLARE SECTION;
   /* PREPARE 准备一个语句用于执行 */
   EXEC SQL PREPARE mystmt FROM :stmt;
   ...
   /* 单引号为有效字符，若用字符串需用双引号 */
   EXEC SQL EXECUTE mystmt USING 42, 'foobar';

   /* 当不再需要预备语句时，应该释放它 */
   EXEC SQL DEALLOCATE PREPARE name;
```

5.9.9.3 执行带有结果集的语句

执行具有单独结果集的SQL语句，可以使用EXECUTE。若要保存结果，则增加INTO子句。示例如下：

```
EXEC SQL BEGIN DECLARE SECTION;
   const char *stmt = "SELECT a, b, c FROM test1 WHERE a > ?";
   int v1, v2;
   VARCHAR v3[50];
EXEC SQL END DECLARE SECTION;

   EXEC SQL PREPARE mystmt FROM :stmt;
   ...
   EXEC SQL EXECUTE mystmt INTO :v1, :v2, :v3 USING 37;
```

说明

EXECUTE命令支持INTO子句和USING子句。

如果一个查询可能返回多个结果行，则应使用游标，游标详情请参见**使用游标**章节，示例如下：

```
EXEC SQL BEGIN DECLARE SECTION;
   char dbaname[128];
   char datname[128];
   char *stmt = "SELECT u.usename as dbaname, d.datname "
                "  FROM pg_database d, pg_user u "
                "  WHERE d.datdba = u.usesysid";
EXEC SQL END DECLARE SECTION;

   EXEC SQL CONNECT TO testdb AS con1 USER testuser;

   EXEC SQL PREPARE stmt1 FROM :stmt;

   EXEC SQL DECLARE cursor1 CURSOR FOR stmt1;
   EXEC SQL OPEN cursor1;

   EXEC SQL WHENEVER NOT FOUND DO BREAK;

   while (1)
   {
      EXEC SQL FETCH cursor1 INTO :dbaname,:datname;
      printf("dbaname=%s, datname=%s\n", dbaname, datname);
   }
   EXEC SQL CLOSE cursor1;

   EXEC SQL COMMIT;
   EXEC SQL DISCONNECT ALL;
```

5.9.10 错误处理

有两种非互斥的方法可以处理嵌入式SQL程序的异常情况和告警：

- 使用WHENEVER命令设置回调，处理告警和错误条件。
- 通过sqlca获取错误或者告警的详细信息，进行相应处理。

5.9.10.1 设置回调

设置回调操作，当告警或者错误发生时，直接执行具体操作进行处理，设置回调命令如下：

```
EXEC SQL WHENEVER condition action;
```

condition取值范围：

- SQLERROR：当在SQL语句执行期间发生错误时，调用指定操作。
- SQLWARNING：当在SQL语句执行期间发生告警时，调用指定操作。
- NOT FOUND：当SQL语句检索或者影响零行，则调用指定操作。

action取值范围：

- CONTINUE：忽略回调错误条件，继续执行，通常可以用来停止break包含条件，为缺省值。
- GOTO label/GO TO label：跳转到指定标签（使用C语言goto语句）。
- SQLPRINT：输出消息到标准错误。
- STOP：调用exit(1)，终止程序。
- DO BREAK：执行C语句break，只有在循环中或者switch语句中使用。

示例如下：

```
/* 当出现一个告警时它打印一个消息，发生一个错误时中止程序。 */
EXEC SQL WHENEVER SQLWARNING SQLPRINT;
EXEC SQL WHENEVER SQLERROR STOP;
```


须知

- 语句EXEC SQL WHENEVER是SQL预处理器的一个指令，而非一个C语言语句。不管C语言程序的流程如何，该语句设置的错误或告警动作都适用于位于处理程序设置点之后的嵌入式SQL语句，除非第一个EXEC SQL WHENEVER语句和导致错误或告警情况发生的SQL语句之间为同一个情况设置了不同的动作。因此下面的两个C语言程序都不会得到预期的效果：

```
/*
 * 错误
 */
int main(int argc, char *argv[])
{
    ...
    if (verbose) {
        EXEC SQL WHENEVER SQLWARNING SQLPRINT;
    }
    ...
    EXEC SQL SELECT ...;
    ...
}
/*
 * 错误
 */
int main(int argc, char *argv[])
{
    ...
    set_error_handler();
    ...
    EXEC SQL SELECT ...;
    ...
}
static void set_error_handler(void)
{
    EXEC SQL WHENEVER SQLERROR STOP;
}
```

- 当使用DO BREAK时只能用于while/for/switch场景，且用完需要使用CONTINUE语句忽略。

5.9.10.2 sqlca

嵌入式SQL接口提供了sqlca（SQL通信区）的全局变量。sqlca包含告警和错误信息。如果在语句执行期间发生多个告警和错误，那么sqlca将只保存最后一个信息。在一个多线程的程序中，每一个线程会自动得到它的sqlca副本。

数据结构如下：

```
struct
{
    char sqlcaid[8];
    long sqlabc;
    long sqlcode;
    struct
    {
        int sqlerrml;
        char sqlerrmc[SQLERRMC_LEN];
    } sqlerrm;
    char sqlerrp[8];
    long sqlerrd[6];
    char sqlwarn[8];
    char sqlstate[5];
} sqlca;
```

如果SQL语句没有发生错误，则sqlca.sqlcode为0，sqlca.sqlstate为"00000"。如果发生了告警或者错误，那么sqlca.sqlcode是负数并且sqlca.sqlstate不同于"00000"。SQLSTATE与SQLCODE的具体值请参见**SQLSTATE与SQLCODE**。

如果SQL语句正确执行，那么sqlca.sqlerrd[1]包含被处理行的OID，并且sqlca.sqlerrd[2]包含被处理或返回的行数。

在发生错误或告警时，sqlca.sqlerrm.sqlerrmc将包含描述该错误的字符串。sqlca.sqlerrm.sqlerrml包含存储在sqlca.sqlerrm.sqlerrmc中错误消息的长度（strlen()的结果）。注意：一些消息可能无法适应定长的sqlerrmc数组，它们将被截断。

在发生告警时，sqlca.sqlwarn[2]被设置为W。

sqlcaid、sqlabc、sqlerrp、sqlwarn以及sqlerrd的剩余元素目前未包含有用的信息。

示例如下：

```
/* 整合WHENEVER和sqlca实现错误处理 */
EXEC SQL WHENEVER SQLERROR SQLCALL print_sqlca();

void print_sqlca()
{
    fprintf(stderr, "==== sqlca ====\n");
    fprintf(stderr, "sqlcode: %ld\n", sqlca.sqlcode);
    fprintf(stderr, "sqlerrm.sqlerrml: %d\n", sqlca.sqlerrm.sqlerrml);
    fprintf(stderr, "sqlerrm.sqlerrmc: %s\n", sqlca.sqlerrm.sqlerrmc);
    fprintf(stderr, "sqlerrd: %ld %ld %ld %ld %ld %ld\n", sqlca.sqlerrd[0],sqlca.sqlerrd[1],sqlca.sqlerrd[2],
                                                          sqlca.sqlerrd[3],sqlca.sqlerrd[4],sqlca.sqlerrd[5]);
    fprintf(stderr, "sqlwarn: %d %d %d %d %d %d %d %d\n", sqlca.sqlwarn[0], sqlca.sqlwarn[1],
sqlca.sqlwarn[2],
                                                          sqlca.sqlwarn[3], sqlca.sqlwarn[4], sqlca.sqlwarn[5],
                                                          sqlca.sqlwarn[6], sqlca.sqlwarn[7]);
    fprintf(stderr, "sqlstate: %5s\n", sqlca.sqlstate);
    fprintf(stderr, "===============\n");
}
```

输出结果形如（此处是一个拼写表名错误）：

```
==== sqlca ====
sqlcode: -400
sqlerrm.sqlerrml: 49
sqlerrm.sqlerrmc: relation "pg_databasep" does not exist on line 38
sqlerrd: 0 0 0 0 0 0
sqlwarn: 0 0 0 0 0 0 0 0
sqlstate: 42P01
===============
```

5.9.10.3 SQLSTATE 与 SQLCODE

SQLSTATE是一个由五个字符组成的数组。这五个字符包含数字或大写字母，它表示多种错误或告警情况的代码。SQLSTATE具有一种层次模式：前两个字符表示情况的总体分类，后三个字符表示总体情况的子类。例如：代码00000表示成功状态。

SQLCODE是一个简单的整数形式。值为0表示成功，一个正值表示带附加信息的成功，一个负值表示错误。SQL标准只定义了正值+100，它表示上一个命令返回或者影响了零行，且没有特定的负值。

表 5-96 SQLSTATE 与 SQLCODE 对应关系表

SQLCODE值	SQLSTATE值	含义
0 (ECPG_NO_ERROR)	SQLSTATE 00000	表示没有错误。

SQLCODE值	SQLSTATE值	含义

100 (ECPG_NOT_FOUND)	SQLSTATE 02000	一种无害情况，它表示上一个命令检索或者处理了零行，或者已到达游标的末尾。
		在循环中处理游标时，可以使用这个代码来检测何时中止该循环，示例如下：
		`while (1)`
		`{`
		`  EXEC SQL FETCH ... ;`
		`  if (sqlca.sqlcode == ECPG_NOT_FOUND)`
		`  break;`
		`}`
		实际上WHENEVER NOT FOUND DO BREAK也会在内部这样做，所以一般不会直接使用这种方法。

-12 (ECPG_OUT_OF_MEMORY)	SQLSTATE YE001	虚拟内存已被耗尽，数字值被定义为-ENOMEM。
-200 (ECPG_UNSUPPORTED)	SQLSTATE YE000	预处理器产生了一些该库无法识别的内容。
-201 (ECPG_TOO_MANY_ARGUMENTS)	SQLSTATE 07001 或 07002	表示命令指定的宿主变量数量超过该命令预期。
-202 (ECPG_TOO_FEW_ARGUMENTS)	SQLSTATE 07001 或 07002	表示命令指定的宿主变量数量低于该命令的预期。
-203 (ECPG_TOO_MANY_MATCHES)	SQLSTATE 21000	表示一个查询已经返回了多行，但是该语句只准备存储一个结果行。
-204 (ECPG_INT_FORMAT)	SQLSTATE 42804	宿主变量是类型int而数据库中的数据是一种不同的类型并且含有不能被解释为int的值。该库使用strtol()进行转换。
-205 (ECPG_UINT_FORMAT)	SQLSTATE 42804	宿主变量是类型unsigned int而数据库中的数据是一种不同的类型并且含有不能被解释为unsigned int的值。该库使用strtoul()进行转换。
-206 (ECPG_FLOAT_FORMAT)	SQLSTATE 42804	宿主变量是类型float而数据库中的数据是另一种类型并且含有不能被解释为float的值。该库使用strtod()进行转换。
-207 (ECPG_NUMERIC_FORMAT)	SQLSTATE 42804	宿主变量是类型numeric而数据库中的数据是另一种类型并且含有不能被解释为numeric的值。

SQLCODE值	SQLSTATE值	含义
-208 (ECPG_INTERVAL_FORMAT)	SQLSTATE 42804	宿主变量是类型interval而数据库中的数据是另一种类型并且含有一个不能被解释为interval的值。
-209 (ECPG_DATE_FORMAT)	SQLSTATE 42804	宿主变量是类型date而数据库中的数据是另一种类型并且含有不能被解释为date的值。
-210 (ECPG_TIMESTAMP_FORMAT)	SQLSTATE 42804	宿主变量是类型timestamp而数据库中的数据是另一种类型并且含有不能被解释为timestamp的值。
-211 (ECPG_CONVERT_BOOL)	SQLSTATE 42804	宿主变量是类型boolean而数据库中的数据既不是't'也不是'f'。
-212 (ECPG_EMPTY)	SQLSTATE YE000	发送给SQL服务器的语句是空值（通常在一个嵌入式SQL程序中不会发生，因此它可能指向一个内部错误）。
-213 (ECPG_MISSING_INDICATOR)	SQLSTATE 22002	返回了一个空值并且没有提供空值指示符。
-214 (ECPG_NO_ARRAY)	SQLSTATE 42804	在要求一个数组的地方使用了一个普通变量。
-215 (ECPG_DATA_NOT_ARRAY)	SQLSTATE 42804	在一个要求数组值的地方数据库返回了一个普通变量。
-216 (ECPG_ARRAY_INSERT)	SQLSTATE 42804	该值不能被插入到数组中。
-220 (ECPG_NO_CONN)	SQLSTATE 08003	程序尝试访问一个不存在的连接。
-221 (ECPG_NOT_CONN)	SQLSTATE YE000	程序尝试访问一个存在的连接但是它没有打开（这是一个内部错误）。
-230 (ECPG_INVALID_STMT)	SQLSTATE 26000	尝试使用的语句还没有被准备好。
-239 (ECPG_INFORMIX_DUPLICATE_KEY)	SQLSTATE 23505	重复键错误，违背唯一约束。
-240 (ECPG_UNKNOWN_DESCRIPTOR)	SQLSTATE 33000	没有找到指定的描述符，尝试使用的语句还没有被准备好。

SQLCODE值	SQLSTATE值	含义
-241 (ECPG_INVALID_DESCRIPTOR_INDEX)	SQLSTATE 07009	指定的描述符超出范围。
-242 (ECPG_UNKNOWN_DESCRIPTOR_ITEM)	SQLSTATE YE000	请求了一个非法的描述符（这是一个内部错误）。
-243 (ECPG_VAR_NOT_NUMERIC)	SQLSTATE 07006	在执行一个动态语句期间，数据库返回了一个numeric值而宿主变量不是numeric类型的。
-244 (ECPG_VAR_NOT_CHAR)	SQLSTATE 07006	在执行一个动态语句期间，数据库返回了一个非numeric值而宿主变量是numeric类型的。
-284 (ECPG_INFORMIX_SUBSELECT_NOT_ONE)	SQLSTATE 21000	子查询的结果不是单一行。
-400 (ECPG_PGSQL)	-	SQL服务器导致了某个错误。该消息包含来自SQL服务器的错误消息。
-401 (ECPG_TRANS)	SQLSTATE 08007	SQL服务器通知用户不能启动、提交或回滚事务。
-402 (ECPG_CONNECT)	SQLSTATE 08001	无法建立数据库连接。
-403 (ECPG_DUPLICATE_KEY)	SQLSTATE 23505	重复键错误，违背唯一约束。
-404 (ECPG_SUBSELECT_NOT_ONE)	SQLSTATE 21000	子查询的结果不是单一行。
-602 (ECPG_WARNING_UNKNOWN_PORTAL)	SQLSTATE 34000	指定了一个非法的游标名。
-603 (ECPG_WARNING_IN_TRANSACTION)	SQLSTATE 25001	事务正在进行。
-604 (ECPG_WARNING_NO_TRANSACTION)	SQLSTATE 25P01	没有活动（正在进行）的事务。
-605 (ECPG_WARNING_PORTAL_EXISTS)	SQLSTATE 42P03	指定了一个现有的游标名。

注意

- ecpg为嵌入式SQL新增加的SQLSTATE码有：22002、07001、07002、07006、07009、33000、42601、42804、42P03、YE000、YE001。其余SQLSTATE码沿用内核SQLSTATE码。
- SQLSCODE为-400表示ecpg检测到内核服务器返回错误，其SQLSTATE为内核相应错误的SQLSTATE。

5.9.11 预处理指令

本节介绍ecpg提供的预处理指令，用于处理宏定义、文件包含和条件编译的程序指令。

5.9.11.1 包含文件

将外部文件包含到嵌入SQL程序中，可使用如下语句：

```
EXEC SQL INCLUDE filename;
EXEC SQL INCLUDE <filename>;
EXEC SQL INCLUDE "filename";
```

说明

- ecpg预处理器按照如下顺序搜索文件：
  1. 当前目录
  2. /usr/local/include
  3. GaussDB Kernel的目录，在编译时定义
  4. /usr/include
- EXEC SQL INCLUDE "filename"语句仅搜索当前目录。
- 在每一个目录中，预处理器首先按照给定的文件名搜索，若没找到则会追加.h到文件名后重试（除非指定的文件名已有该后缀）。
- 文件名是大小写敏感的。

5.9.11.2 ifdef、ifndef、else、elif 和 endif 指令

ecpg提供了ifdef、ifndef、else、elif和endif条件编译指令。在预处理时，按照不同的条件去编译程序的不同部分，使用时，需要添加EXEC SQL前缀关键字。

示例如下：

```
EXEC SQL ifndef TZVAR;
EXEC SQL SET TIMEZONE TO 'GMT';
EXEC SQL elif TZNAME;
EXEC SQL SET TIMEZONE TO TZNAME;
EXEC SQL else;
EXEC SQL SET TIMEZONE TO TZVAR;
EXEC SQL endif;
```

5.9.11.3 define 和 undef 指令

嵌入式 SQL具有类似于C语言中#define指令：

```
EXEC SQL DEFINE name;
EXEC SQL DEFINE name value;
EXEC SQL UNDEF name;
```

示例如下：

```
/* 定义名称 */
EXEC SQL DEFINE HAVE_FEATURE;

/* 定义常量 */
EXEC SQL DEFINE MYNUMBER 12;
EXEC SQL DEFINE MYSTRING 'abc';

/* 使用 UNDEF 移除定义 */
EXEC SQL UNDEF MYNUMBER;
```

在嵌入式SQL程序中也可以使用C语言版本的#define和#undef。区别在于定义的值会在哪里被计算，如果使用EXEC SQL DEFINE，那么ecpg预处理阶段会计算这些定义并替换值。如下示例，ecpg对其进行替换并且编译器不会解析名为MYNUMBER的任何名称或标识符：

```
EXEC SQL DEFINE MYNUMBER 12;
...
EXEC SQL UPDATE Tbl SET col = MYNUMBER;
```

须知

不能把#define用于一个将要在嵌入式SQL查询中使用的变量，因为在这种情况下嵌入式SQL预编译器不能看到这个声明。

5.9.12 使用库函数

- ECPGdebug(int on, FILE *stream)：若函数第一个参数为非0，则开启调试日志，第二个参数表示要打印日志的标准输出流。调试日志在标准输出流上执行，日志包含所有输入的SQL语句以及来自GaussDB Kernel服务器的结果。

  示例：

  ```
  #include <stdio.h>
  #include <stdlib.h>
  #include <string.h>
  #include "sqlca.h"

  int main()
  {
      ECPGdebug(1, stderr);
      EXEC SQL CONNECT postgres;
      EXEC SQL SET AUTOCOMMIT TO ON;
      EXEC SQL CREATE TABLE T1(a int);
      return (0);
  }
  ```

- ECPGget_PGconn(const char *connection_name)：返回由给定名称标识的数据库连接句柄。如果connection_name设置为NULL，则返回当前连接句柄。如果没有连接句柄可以被识别，则该函数返回NULL。

  示例：

  ```
  #include <stdio.h>
  #include <stdlib.h>
  #include <string.h>
  #include "sqlca.h"

  int main()
  {
      ECPGdebug(1, stderr);
      EXEC SQL CONNECT TO postgres as con1;
      EXEC SQL SET AUTOCOMMIT TO ON;
      EXEC SQL DROP TABLE IF EXISTS T1;
      PGconn     *conn;
      conn = ECPGget_PGconn("con1");
  ```

```
    printf("conn = %p\n", conn);
    conn = ECPGget_PGconn(NULL);
    printf("conn = %p\n", conn);
    EXEC SQL CREATE TABLE T1(a int);
    return (0);
}
```

- ECPGtransactionStatus(const char *connection_name)：返回connection_name连接的当前事务状态。可能的返回值包括：

```
    PQTRANS_IDLE,    /* connection idle，连接空闲 */
    PQTRANS_ACTIVE,  /* command in progress，命令正在执行 */
    PQTRANS_INTRANS, /* idle, within transaction block，空闲，在事务块里 */
    PQTRANS_INERROR, /* idle, within failed transaction，空闲，在失败的事务里 */
    PQTRANS_UNKNOWN  /* cannot determine status，未知状态 */
```

  示例：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "sqlca.h"

int main()
{
    ECPGdebug(1, stderr);
    EXEC SQL CONNECT TO postgres as con1;
    EXEC SQL DROP TABLE IF EXISTS T1;
    int a = ECPGtransactionStatus("con1");
    printf("%d\n", a);
    EXEC SQL CREATE TABLE T1(a int);
    EXEC SQL COMMIT;
    return (0);
}
```

- ECPGfree_auto_mem()：释放为输出型宿主变量申请的所有内存，在程序结束时调用（return\exit）。

  示例：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "sqlca.h"

int main()
{
  EXEC SQL BEGIN DECLARE SECTION;
  int *ip1=0;
  char **cp2=0;
  int *ipointer1=0;
  int *ipointer2=0;
  int colnum;
  EXEC SQL END DECLARE SECTION;
  int i;

  ECPGdebug(1, stderr);

  EXEC SQL WHENEVER SQLERROR DO sqlprint();
  EXEC SQL CONNECT TO REGRESSDB1;

  EXEC SQL SET DATESTYLE TO postgres;

  EXEC SQL CREATE TABLE test (a int, b text);
  EXEC SQL INSERT INTO test VALUES (1, 'one');
  EXEC SQL INSERT INTO test VALUES (2, 'two');
  EXEC SQL INSERT INTO test VALUES (NULL, 'three');
  EXEC SQL INSERT INTO test VALUES (NULL, NULL);

  EXEC SQL ALLOCATE DESCRIPTOR mydesc;
  EXEC SQL SELECT * INTO SQL DESCRIPTOR mydesc FROM test;
  EXEC SQL GET DESCRIPTOR mydesc :colnum=COUNT;
  EXEC SQL GET DESCRIPPTOR mydesc VALUE 1 :ip1=DATA, :ipointer1=INDICATOR;
```



```
  EXEC SQL GET DESCRIPTOR mydesc VALUE 2 :cp2=DATA, :ipointer2=INDICATOR;

  printf("Result (%d columns):\n", colnum);
  for (i=0;i < sqlca.sqlerrd[2];++i)
  {
    if (ipointer1[i]) printf("NULL, ");
    else printf("%d, ",ip1[i]);

    if (ipointer2[i]) printf("NULL, ");
    else printf("'%s', ",cp2[i]);
    printf("\n");
  }
  ECPGfree_auto_mem();
  printf("\n");

  EXEC SQL DEALLOCATE DESCRIPTOR mydesc;
  EXEC SQL ROLLBACK;
  EXEC SQL DISCONNECT;
  return 0;
}
```

5.9.13 SQL 描述符区域

SQL描述符区域是一种处理SELECT、FETCH或者DESCRIBE语句结果的高级方法。SQL描述符区域把一行数据里的数据和元数据项组合到一个数据结构中。ecpg提供了两种使用描述符区域的方法：命名SQL描述符区域和SQLDA。

5.9.13.1 命名 SQL 描述符区域

一个命名SQL描述符区域由一个头部以及一个或多个条目描述符区域构成。头部包含与整个描述区域相关的信息，而条目描述符区域则描述结果行中的某一列。

- 在使用SQL描述符区域之前，需要分配一个SQL描述符区域：
  ```
  EXEC SQL ALLOCATE DESCRIPTOR identifier;
  ```
- 当不再需要这个描述符区域时，应及时释放：
  ```
  EXEC SQL DEALLOCATE DESCRIPTOR identifier;
  ```
- 要使用一个描述符区域，需要使用INTO子句声明：
  ```
  EXEC SQL FETCH NEXT FROM mycursor INTO SQL DESCRIPTOR mydesc;
  ```
  如果结果集为空，该描述符区域仍会包含查询的元数据。
- 对于还没有执行的预备查询，可以使用DESCRIBE得到其结果集的元数据：
  ```
  EXEC SQL BEGIN DECLARE SECTION;
    char *sql_stmt = "SELECT * FROM table1";
  EXEC SQL END DECLARE SECTION;

    EXEC SQL PREPARE stmt1 FROM :sql_stmt;
    EXEC SQL DESCRIBE stmt1 INTO SQL DESCRIPTOR mydesc;
  ```
  在DESCRIBE和FETCH语句中，INTO和USING关键词的使用相似：它们产生结果集以及一个描述符区域的元数据。
- 从头部检索一个描述符区域的值并且将其存储到一个宿主变量中：
  ```
  EXEC SQL GET DESCRIPTOR name :hostvar = field;
  ```
- 当前只定义了一个头部描述符区域COUNT，它存放描述符区域的条目（即结果集中包含多少列），宿主变量为一个整数类型，需从条目描述符区域中得到一个具体值：
  ```
  EXEC SQL GET DESCRIPTOR name VALUE num :hostvar = field;
  ```
  num可以是一个字符整数或者一个包含整数的宿主变量。可能的类型如下：
  - CARDINALITY（整数）：结果集中的行数
  - DATA：实际的数据项（这个范围的实际数据类型取决于查询）

  - DATETIME_INTERVAL_CODE（整数）：当TYPE是9时，DATETIME_INTERVAL_CODE将具有以下值之一：1表示DATE，2表示TIME，3表示TIMESTAMP，4表示TIME WITH TIME ZONE，5表示TIMESTAMP WITH TIME ZONE。
  - INDICATOR（整数）：指示符（表示一个空值或者一个值截断）
  - LENGTH（整数）：以字符计的数据长度
  - NAME(string)：列名
  - OCTET_LENGTH（整数）：以字节计的数据字符表达的长度
  - PRECISION（整数）：精度（用于类型numeric）
  - RETURNED_LENGTH（整数）：以字符计的数据长度
  - RETURNED_OCTET_LENGTH（整数）：以字节计的数据字符表达的长度
  - SCALE（整数）：比例（用于类型numeric）
  - TYPE（整数）：列的数据类型的数字编码
- 要检索字段数值并且把它存储到一个宿主变量里，使用如下命令：
  ```
  EXEC SQL GET DESCRIPTOR mydesc  VALUE num :hostvar = field
  ```
  num可以是一个字符整数或者一个包含整数的宿主变量。可 能的字段有:
  - DATA
  - 实际数据项（这个字段的数据类型依赖于这个查询）
  - NAME(string)
  - 字段名称
- 手动建立一个描述符区域为一个查询或游标提供输入参数，使用如下命令：
  ```
  EXEC SQL SET DESCRIPTOR name VALUE numfield = :hostvar;
  ```
- 在一个FETCH语句中检索多行记录且用数组类型的宿主变量来存储数据，示例如下：
  ```
  EXEC SQL BEGIN DECLARE SECTION;
     int id[5];
  EXEC SQL END DECLARE SECTION;
     EXEC SQL FETCH 5 FROM mycursor INTO SQL DESCRIPTOR mydesc;
     EXEC SQL GET DESCRIPTOR mydesc VALUE 1 :id = DATA;
  ```

5.9.13.2 SQLDA

SQLDA是一个C语言结构体，用来存放一个查询的结果集，一个结构体存储一个结果集的记录。

```
EXEC SQL include sqlda.h;
sqlda_t         *mysqlda;
EXEC SQL FETCH 3 FROM mycursor INTO DESCRIPTOR mysqlda;
```

注意SQL关键词被省略了，**命名SQL描述符区域**章节中关于INTO和USING关键词的用例在一定条件下也适用于这里。在一个DESCRIBE语句中，如果使用了INTO关键词，则DESCRIPTOR关键词可以省略。

```
EXEC SQL DESCRIBE prepared_statement INTO mysqlda;
```

- 使用SQLDA的步骤：
  a. 准备一个查询，并且为它声明一个游标。
  b. 为结果行声明SQLDA。
  c. 为输入参数声明SQLDA，并且初始化参数和分配内存。
  d. 打开具有输入SQLDA的游标
  e. 从游标中抓取行，并且将它们存储到输出SQLDA中。

f. 从输出SQLDA中读取值到宿主变量中。

g. 关闭游标。

h. 释放为SQLDA分配的内存。

- SQLDA的数据结构类型有三种：sqlda_t、sqlvar_t以及struct sqlname。

  a. sqlda_t结构

  sqlda_t的定义如下：

```
struct sqlda_struct
{
    char            sqldaid[8];
    long            sqldabc;
    short           sqln;
    short           sqld;
    struct sqlda_struct *desc_next;
    struct sqlvar_struct sqlvar[1];
};
typedef struct sqlda_struct sqlda_t;
```

  结构体成员的含义如下：

  - sqldaid：它包含一个字符串"SQLDA"。
  - sqldabc：它包含已分配空间的尺寸（以字节计）。
  - sqln：当它被传递给使用USING关键词的OPEN、DECLARE或者EXECUTE语句时，它包含一个参数化查询实例的输入参数的数目。在它被用作SELECT、EXECUTE或FETCH语句的输出时，它的值和sqld一样。
  - sqld：它包含一个结果集中域的数量。
  - desc_next：如果查询返回不止一个记录，那么会返回一个SQLDA结构体链表，desc_next指向下一个SQLDA结构体的指针。
  - sqlvar：这是结果集中的列组。

  b. sqlvar_t结构

  结构类型sqlvar_t保存一个列值和元数据（例如：类型、长度）。该类型的定义如下：

```
struct sqlvar_struct
{
    short          sqltype;
    short          sqllen;
    char          *sqldata;
    short         *sqlind;
    struct sqlname sqlname;
};
typedef struct sqlvar_struct sqlvar_t;
```

  结构体成员的含义如下：

  - sqltype：包含该域的类型标识符。
  - sqllen：包含域的二进制长度，例如：ECPGt_int是4字节。
  - sqldata：指向数据。数据格式请参见**类型映射**章节。
  - sqlind：指向空指示符。0表示非空，-1表示空。
  - sqlname：域的名称。

  c. struct sqlname结构

一个struct sqlname结构保存一个列名。它被当作sqlvar_t结构的一个成员。该结构的定义如下：

```
#define NAMEDATALEN 64
struct sqlname
{
    short       length;
    char        data[NAMEDATALEN];
};
```

结构体成员的含义如下：

- length：包含域名称的长度。
- data：包含实际的域名称。

- 使用一个SQLDA检索一个结果集。

  通过一个SQLDA检索一个查询结果集的一般步骤：

  a. 声明一个sqlda_t结构来接收结果集。
  b. 执行FETCH/EXECUTE/DESCRIBE命令来处理一个已声明SQLDA的查询。
  c. 通过查看sqlda_t结构的成员sqln来检查结果集中记录的数量。
  d. 从sqlda_t结构体的成员sqlvar[0]、sqlvar[1]等中得到每一列的值。
  e. 沿着sqlda_t结构的成员desc_next指针到达下一行（sqlda_t）。
  f. 根据需要重复上述步骤。

  示例如下：

```
/* 声明一个sqlda_t结构来接收结果集。*/
sqlda_t *sqlda1;
/* 接下来，指定一个命令中的SQLDA。这是一个FETCH命令的例子。*/
EXEC SQL FETCH NEXT FROM cur1 INTO DESCRIPTOR sqlda1;
/* 运行一个循环顺着链表来检索行。*/
sqlda_t *cur_sqlda;
for (cur_sqlda = sqlda1;
     cur_sqlda != NULL;
     cur_sqlda = cur_sqlda->desc_next)
{
    ...
}
/* 在循环内部，运行另一个循环来检索行中每一列的数据（sqlvar_t结构）。*/
for (i = 0; i < cur_sqlda->sqld; i++)
{
    sqlvar_t v = cur_sqlda->sqlvar[i];
    char *sqldata = v.sqldata;
    short sqllen  = v.sqllen;
    ...
}
/* 要得到一列的值，应检查sqlvar_t结构的成员sqltype的值。然后，根据列类型切换到一种合适的方法从sqlvar域中复制数据到一个主变量。*/
char var_buf[1024];
switch (v.sqltype)
{
    case ECPGt_char:
        memset(&var_buf, 0, sizeof(var_buf));
        memcpy(&var_buf, sqldata, (sizeof(var_buf) <= sqllen ? sizeof(var_buf) - 1 : sqllen));
        break;

    case ECPGt_int:
        memcpy(&intval, sqldata, sqllen);
        snprintf(var_buf, sizeof(var_buf), "%d", intval);
        break;
    ...
}
```

- 使用一个SQLDA传递查询参数。

  使用一个SQLDA传递输入参数给一个预备查询的一般步骤：

a. 创建一个预备查询（预备语句）。
b. 声明一个sqlda_t结构体作为SQLDA。
c. 为SQLDA分配内存区域。
d. 在分配好的内存中设置（复制）输入值。
e. 打开一个在SQLDA上声明的游标。

示例如下：

```
/* 首先，创建一个预备语句。 */
EXEC SQL BEGIN DECLARE SECTION;
    char query[1024] = "SELECT d.oid, * FROM pg_database d, pg_stat_database s WHERE d.oid =
s.datid AND (d.datname = ? OR d.oid = ?)";
EXEC SQL END DECLARE SECTION;
    EXEC SQL PREPARE stmt1 FROM :query;

    /* 接下来为一个SQLDA分配内存，并且在sqlda_t结构的sqln成员变量中设置输入参数的数量。
     * 当预备查询要求两个或多个输入参数时，应用必须分配额外的内存空间，空间的大小为 (参数数目 -
1) * sizeof(sqlvar_t)。
     * 这里的例子展示了为两个输入参数分配内存空间。
     */
    sqlda_t *sqlda2;
    sqlda2 = (sqlda_t *) malloc(sizeof(sqlda_t) + sizeof(sqlvar_t));
    memset(sqlda2, 0, sizeof(sqlda_t) + sizeof(sqlvar_t));
    sqlda2->sqln = 2; /* 输入变量的数目 */
    /* 内存分配之后，把参数值存储到sqlvar[]数组（当 SQLDA 在接收结果集时，这也是用来检索列值的数
组）。
     * 在这个例子中，输入参数是"postgres"（字符串类型）和1（整数类型）。*/
    sqlda2->sqlvar[0].sqltype = ECPGt_char;
    sqlda2->sqlvar[0].sqldata = "postgres";
    sqlda2->sqlvar[0].sqllen  = 8;
    int intval = 1;
    sqlda2->sqlvar[1].sqltype = ECPGt_int;
    sqlda2->sqlvar[1].sqldata = (char *) &intval;
    sqlda2->sqlvar[1].sqllen  = sizeof(intval);
    /* 通过打开一个游标并且说明之前已经建立好的 SQLDA，输入参数被传递给预备语句。*/
    EXEC SQL OPEN cur1 USING DESCRIPTOR sqlda2;
    /* 最后，用完输入 SQLDA 后必须显式地释放已分配的内存空间，这与用于接收查询结果的 SQLDA 不
同。*/
    free(sqlda2);
```

5.9.14 常用示例

ecpg 常用示例代码

```
#include <locale.h>
#include <string.h>
#include <stdlib.h>

exec sql whenever sqlerror sqlprint;
exec sql include sqlca;

int main(void)
{
EXEC SQL BEGIN DECLARE SECTION;
    char *temp_str = (char *)malloc(11);
EXEC SQL END DECLARE SECTION;

    ECPGdebug(1, stderr);

    exec sql connect to postgres;

    /* 打开自动提交，以下执行exec sql时不用手动commit */
    exec sql set autocommit = on;
    exec sql drop table if exists test_t;
    /* 建表，插入数据 */
    exec sql create table test_t(f float, i int, a int[10], mstr char(10));
```



```
    exec sql insert into test_t(f, i, a, mstr) values(1.01,1,'{0,1,2,3,4,5,6,7,8,9}', 'China');

    /* 关闭自动提交，以下插入数据的sql语句需要手动commit才能提交 */
    exec sql set autocommit = off;
    exec sql insert into test_t(f, i, a, mstr) values(2.01,2,'{0,1,2,3,4,5,6,7,8,9}', 'USA');
    exec sql commit;

    exec sql insert into test_t(f, i, a, mstr) values(3.01,3,'{0,1,2,3,4,5,6,7,8,9}', 'AUS');
    exec sql insert into test_t(f, i, a, mstr) values(4.01,4,'{0,1,2,3,4,5,6,7,8,9}', 'JAP');
    exec sql commit;

EXEC SQL BEGIN DECLARE SECTION;
    int a[10] = {9,8,7,6,5,4,3,2,1,0};
    int id = 6;
EXEC SQL END DECLARE SECTION;

    /* 从宿主变量取数据插入到表中，宿主变量的类型与表定义的类型一致 */
    strcpy(temp_str, "RUS");
    exec sql insert into test_t(f, i, a, mstr) values(5.01,5,:a,:temp_str);
    exec sql commit;

    exec sql set autocommit = on;
    exec sql begin;
    exec sql insert into test_t(f, i, a, mstr) values(6.01,:id,:a,'SIG');
    exec sql commit;
    exec sql set autocommit = off;

exec sql begin declare section;
    float ff;
    char tmp_text[25] = "klmnopqrst";
exec sql end declare section;

    exec sql set autocommit = on;
    exec sql begin work;

    printf("Found ff=%f tmp_text=%10.10s\n", ff, tmp_text);

    /* 条件查询语句示例 */
    exec sql select f, mstr into :ff,:tmp_text from test_t where f > (select f from test_t where i = 4 or i < 0)
order by a limit 1;
    printf("Found ff=%f tmp_text=%10.10s\n", ff, tmp_text);

    exec sql select f, mstr into :ff,:tmp_text from test_t where mstr = 'JAP' order by i;
    printf("Found ff=%f tmp_text=%10.10s\n", ff, tmp_text);

    exec sql select f, mstr into :ff,:tmp_text from  test_t order by i DESC limit 1;
    printf("Found ff=%f tmp_text=%10.10s\n", ff, tmp_text);

    exec sql select f, mstr into :ff,:tmp_text from  test_t order by mstr limit 1;
    printf("Found ff=%f tmp_text=%10.10s\n", ff, tmp_text);

    exec sql select count(f), a into :ff,:tmp_text from test_t where i > 2 group by a limit 1;
    printf("Found ff=%f tmp_text=%20.30s\n", ff, tmp_text);

    exec sql select count(f), a into :ff,:tmp_text from test_t where i > 3 group by a order by a limit 1;
    printf("Found ff=%f tmp_text=%20.30s\n", ff, tmp_text);

    exec sql select sum(f), a into :ff,:tmp_text from test_t where i > 2  group by a order by a limit 1;
    printf("Found ff=%f tmp_text=%20.30s\n", ff, tmp_text);

    exec sql select distinct a into :tmp_text from test_t order by a limit 1;

    exec sql drop table test_t;

    exec sql commit;
    /* 释放连接，释放为宿主变量分配的内存 */
    exec sql disconnect;
    free(temp_str);
```



```
    return 0;
}
```

pgtypes 库函数示例代码

示例一：使用库函数对时间和日期类型进行不同操作。具体使用方式请参见**使用库函数**章节。

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <limits.h>
#include <pgtypes_date.h>
#include <pgtypes_timestamp.h>

char *dates[] = { "19990108foobar",
                  "19990108 foobar",
                  "1999-01-08 foobar",
                  "January 8, 1999",
                  "1999-01-08",
                  "1/8/1999",
                  "1/18/1999",
                  "01/02/03",
                  "1999-Jan-08",
                  "Jan-08-1999",
                  "08-Jan-1999",
                  "99-Jan-08",
                  "08-Jan-99",
                  "08-Jan-06",
                  "Jan-08-99",
                  "19990108",
                  "990108",
                  "1999.008",
                  "J2451187",
                  "January 8, 99 BC",
                  NULL
                 };

/* 不可与libc的“times”冲突 */
static char *times[] = { "0:04",
                         "1:59 PDT",
                         "13:24:40 -8:00",
                         "13:24:40.495+3",
                          NULL
                        };

char *intervals[] = { "1 minute",
                      "1 12:59:10",
                      "2 day 12 hour 59 minute 10 second",
                      "1 days 12 hrs 59 mins 10 secs",
                      "1 days 1 hours 1 minutes 1 seconds",
                      "1 year 59 mins",
                      "1 year 59 mins foobar",
                      NULL
                    };

int main(void)
{
 exec sql begin declare section;
    date date1;
    timestamp ts1, ts2;
    char *text;
    interval *i1;
    date *dc;
 exec sql end declare section;

    int i, j;
    char *endptr;
```

```
ECPGdebug(1, stderr);

/* 从文本中解析一个时间戳并将日期转换成字符串 */
ts1 = PGTYPEStimestamp_from_asc("2003-12-04 17:34:29", NULL);
text = PGTYPEStimestamp_to_asc(ts1);

printf("timestamp: %s\n", text);
free(text);

/* 从时间戳中抽取日期部分 */
date1 = PGTYPESdate_from_timestamp(ts1);
dc = PGTYPESdate_new();
*dc = date1;
/* 返回一个日期变量的文本表达 */
text = PGTYPESdate_to_asc(*dc);
printf("Date of timestamp: %s\n", text);
free(text);
PGTYPESdate_free(dc);

for (i = 0; dates[i]; i++)
{
    bool err = false;
    /* 从日期的文本表达解析一个日期 */
    date1 = PGTYPESdate_from_asc(dates[i], &endptr);
    if (date1 == INT_MIN) {
        err = true;
    }
    /* 返回一个日期变量的文本表达 */
    text = PGTYPESdate_to_asc(date1);
    printf("Date[%d]: %s (%c - %c)\n",
           i, err ? "-" : text,
           endptr ? 'N' : 'Y',
           err ? 'T' : 'F');
    free(text);
    if (!err)
    {
        for (j = 0; times[j]; j++)
        {
            int length = strlen(dates[i])+ 1+ strlen(times[j])+ 1;
            char* t = (char *)malloc(length);
            sprintf(t, "%s %s", dates[i], times[j]);
            /* 从文本中解析一个时间戳并将日期转换成字符串 */
            ts1 = PGTYPEStimestamp_from_asc(t, NULL);
            text = PGTYPEStimestamp_to_asc(ts1);
            if (i != 19 || j != 3)
                printf("TS[%d,%d]: %s\n",i, j, errno ? "-" : text);
            free(text);
            free(t);
       }
    }
}

/* 从文本中解析一个时间戳 */
ts1 = PGTYPEStimestamp_from_asc("2004-04-04 23:23:23", NULL);

for (i = 0; intervals[i]; i++)
{
    interval *ic;
    /* 从文本中解析一个区间 */
    i1 = PGTYPESinterval_from_asc(intervals[i], &endptr);
    if (*endptr)
        printf("endptr set to %s\n", endptr);
    if (!i1)
    {
         printf("Error parsing interval %d\n", i);
         continue;
     }
     /* 把一个区间变量加到时间戳变量上 */
     j = PGTYPEStimestamp_add_interval(&ts1, i1, &ts2);
```



```
        if (j < 0)
            continue;
        /* 将一个区间类型变量转换成文本格式 */
        text = PGTYPESinterval_to_asc(i1);
        printf("interval[%d]: %s\n", i, text ? text : "-");
        free(text);

        /* 返回一个已分配区间变量的指针 */
        ic = PGTYPESinterval_new();
        /* 复制一个区间类型的变量 */
        PGTYPESinterval_copy(i1, ic);
        /* 将一个区间类型变量转换成文本格式 */
        text = PGTYPESinterval_to_asc(i1);
        printf("interval_copy[%d]: %s\n", i, text ? text : "-");
        free(text);
        /* 释放已经分配区间变量的内存 */
        PGTYPESinterval_free(ic);
        PGTYPESinterval_free(i1);
    }

    return (0);
}
```

示例二：使用pgtypes库函数对numeric类型进行不同操作。

```
#include <stdio.h>
#include <stdlib.h>
#include <pgtypes_numeric.h>
#include <pgtypes_error.h>
#include <decimal.h>

char* nums[] = { "2E394", "-2", ".794", "3.44", "592.49E21", "-32.84e4",
                 "2E-394", ".1E-2", "+.0", "-592.49E-07", "+32.84e-4",
                 ".500001", "-.5000001",
                 "12345678901234567890123456678.91", /* 30个数位应转换为十进制 */
                 "12345678901234567890123456678.921", /* 31个数位的数字不应转为十进制 */
                 "not a number",
                 NULL
                };

static void check_errno(void);

int main(void)
{
    char *text="error\n";
    char *endptr;
    numeric *num, *nin;
    decimal *dec;
    long l;
    int i, j, k, q, r, count = 0;
    double d;
    numeric **numarr = (numeric **) calloc(1, sizeof(numeric));

    ECPGdebug(1, stderr);

    for (i = 0; nums[i]; i++)
    {
        /* 返回由malloc分配的字符串的指针，它包含numeric类型nums[i]的字符串表达 */
        num = PGTYPESnumeric_from_asc(nums[i], &endptr);
        if (!num) check_errno();
        if (endptr != NULL)
        {
            printf("endptr of %d is not NULL\n", i);
            if (*endptr != '\0')
                printf("*endptr of %d is not \\0\n", i);
        }
        if (!num) continue;

        numarr = (numeric **)realloc(numarr, sizeof(numeric *) * (count + 1));
        numarr[count++] = num;
```



```
/* 返回由malloc分配的字符串的指针，它包含numeric类型num的字符串表达 */
text = PGTYPESnumeric_to_asc(num, -1);
if (!text) check_errno();
printf("num[%d,1]: %s\n", i, text); free(text);
text = PGTYPESnumeric_to_asc(num, 0);
if (!text) check_errno();
 printf("num[%d,2]: %s\n", i, text); free(text);
text = PGTYPESnumeric_to_asc(num, 1);
if (!text) check_errno();
 printf("num[%d,3]: %s\n", i, text); free(text);
text = PGTYPESnumeric_to_asc(num, 2);
if (!text) check_errno();
printf("num[%d,4]: %s\n", i, text); free(text);

/* 请求一个指向新分配的numeric变量的指针 */
nin = PGTYPESnumeric_new();
text = PGTYPESnumeric_to_asc(nin, 2);
if (!text) check_errno();
printf("num[%d,5]: %s\n", i, text); free(text);

/* 将一个numeric类型的变量转换为长整型 */
r = PGTYPESnumeric_to_long(num, &l);
if (r) check_errno();
printf("num[%d,6]: %ld (r: %d)\n", i, r?0L:l, r);
if (r == 0)
{
   /* 把一个长整型变量转换为一个numeric变量 */
   r = PGTYPESnumeric_from_long(l, nin);
   if (r) check_errno();
   /* 返回由malloc分配的字符串的指针，它包含numeric类型nin的字符串表达 */
   text = PGTYPESnumeric_to_asc(nin, 2);
   /* 比较两个numeric变量 */
   q = PGTYPESnumeric_cmp(num, nin);
   printf("num[%d,7]: %s (r: %d - cmp: %d)\n", i, text, r, q);
   free(text);
}

/* 将一个numeric类型的变量转换成整数 */
r = PGTYPESnumeric_to_int(num, &k);
if (r) check_errno();
printf("num[%d,8]: %d (r: %d)\n", i, r?0:k, r);
if (r == 0)
{
    /* 把一个整数变量转换成一个numeric变量 */
    r = PGTYPESnumeric_from_int(k, nin);
    if (r) check_errno();
    /* 返回由malloc分配的字符串的指针，它包含numeric类型nin的字符串表达 */
    text = PGTYPESnumeric_to_asc(nin, 2);
    q = PGTYPESnumeric_cmp(num, nin);
    printf("num[%d,9]: %s (r: %d - cmp: %d)\n", i, text, r, q);
    free(text);
}

if (i != 6)
{
     /* 将一个numeric类型的变量转换成双精度类型 */
     r = PGTYPESnumeric_to_double(num, &d);
     if (r) check_errno();
     printf("num[%d,10]: %g (r: %d)\n", i, r?0.0:d, r);
 }

/* 请求一个指向新分配的numeric变量的指针*/
dec = PGTYPESdecimal_new();
/* 将一个decimal类型的变量转换成numeric */
r = PGTYPESnumeric_to_decimal(num, dec);
if (r) check_errno();
printf("num[%d,11]: - (r: %d)\n", i, r);
if (r == 0)
{
```



```
        /* 将一个decimal类型的变量转换成numeric */
        r = PGTYPESnumeric_from_decimal(dec, nin);
        if (r) check_errno();
        /* 返回由malloc分配的字符串的指针，它包含numeric类型nin的字符串表达 */
        text = PGTYPESnumeric_to_asc(nin, 2);
        /* 比较两个numeric变量 */
        q = PGTYPESnumeric_cmp(num, nin);
        printf("num[%d,12]: %s (r: %d - cmp: %d)\n", i, text, r, q);
        free(text);
    }

    /* 释放numeric变量的内存 */
    PGTYPESdecimal_free(dec);
    PGTYPESnumeric_free(nin);
    printf("\n");
}

for (i = 0; i < count; i++)
{
   for (j = 0; j < count; j++)
   {
      /* 请求一个指向新分配的numeric变量的指针 */
      numeric* a = PGTYPESnumeric_new();
      numeric* s = PGTYPESnumeric_new();
      numeric* m = PGTYPESnumeric_new();
      numeric* d = PGTYPESnumeric_new();
      /*  把两个numeric变量相加放到第三个numeric变量中 */
      r = PGTYPESnumeric_add(numarr[i], numarr[j], a);
      if (r)
      {
         check_errno();
         printf("r: %d\n", r);
       }
      else
      {
          /* 返回由malloc分配的字符串的指针，它包含numeric类型a的字符串表达 */
          text = PGTYPESnumeric_to_asc(a, 10);
          printf("num[a,%d,%d]: %s\n", i, j, text);
          free(text);
      }
      /* 把两个numeric变量相减并且把结果返回到第三个numeric变量 */
      r = PGTYPESnumeric_sub(numarr[i], numarr[j], s);
      if (r)
      {
          check_errno();
          printf("r: %d\n", r);
      }
      else
      {
          /* 返回由malloc分配的字符串的指针，它包含numeric类型s的字符串表达 */
          text = PGTYPESnumeric_to_asc(s, 10);
          printf("num[s,%d,%d]: %s\n", i, j, text);
          free(text);
      }
      /* 把两个numeric变量相乘并且把结果返回到第三个numeric变量 */
      r = PGTYPESnumeric_mul(numarr[i], numarr[j], m);
      if (r)
      {
          check_errno();
          printf("r: %d\n", r);
      }
      else
      {
          /* 返回由malloc分配的字符串的指针，它包含numeric类型m的字符串表达 */
          text = PGTYPESnumeric_to_asc(m, 10);
          printf("num[m,%d,%d]: %s\n", i, j, text);
          free(text);
      }
      /* 把两个numeric变量相除并且把结果返回到第三个numeric变量 */
```



```
            r = PGTYPESnumeric_div(numarr[i], numarr[j], d);
            if (r)
            {
                check_errno();
                printf("r: %d\n", r);
            }
            else
            {
                /* 返回由malloc分配的字符串的指针，它包含numeric类型d的字符串表达 */
                text = PGTYPESnumeric_to_asc(d, 10);
                printf("num[d,%d,%d]: %s\n", i, j, text);
                free(text);
            }

            /* 释放一个numeric变量的内存 */
            PGTYPESnumeric_free(a);
            PGTYPESnumeric_free(s);
            PGTYPESnumeric_free(m);
            PGTYPESnumeric_free(d);
        }
    }

    for (i = 0; i < count; i++)
    {
        /* 返回由malloc分配的字符串的指针，它包含numeric类型numarr[i]的字符串表达 */
        text = PGTYPESnumeric_to_asc(numarr[i], -1);
        printf("%d: %s\n", i, text);
        free(text);
        /* 释放内存 */
        PGTYPESnumeric_free(numarr[i]);
    }
    free(numarr);

    return (0);
}

/* 错误处理 */
static void
check_errno(void)
{
    switch(errno)
    {
        case 0:
        printf("(no errno set) - ");
        break;
        case PGTYPES_NUM_OVERFLOW:
        printf("(errno == PGTYPES_NUM_OVERFLOW) - ");
        break;
        case PGTYPES_NUM_UNDERFLOW:
        printf("(errno == PGTYPES_NUM_UNDERFLOW) - ");
        break;
        case PGTYPES_NUM_BAD_NUMERIC:
        printf("(errno == PGTYPES_NUM_BAD_NUMERIC) - ");
        break;
        case PGTYPES_NUM_DIVIDE_ZERO:
        printf("(errno == PGTYPES_NUM_DIVIDE_ZERO) - ");
        break;
        default:
        printf("(unknown errno (%d))\n", errno);
        printf("(libc: (%s)) ", strerror(errno));
        break;
    }
}
```

5.9.15 ecpg 与 Pro*C 兼容性对比

ecpg是GaussDB提供的一种用于C语言程序的嵌入式SQL预处理器，与A数据库Pro*C预处理器在编译执行命令、语法、嵌入式语句等行为和语义上存在差异。

ecpg与Pro*C的相关使用差异对比：

- 目前ecpg不支持EXEC SQL CONTEXT ALLOCATE、EXEC SQL CONTEXT USE、EXEC SQL CONTEXT FREE。

  说明

  ecpg当前不支持CONTEXT申请、使用、释放操作，ecpg有独立的内存管理机制。多线程模式下，ecpg在每个线程中独立地建立连接、执行SQL语句以及相关资源的释放。这一使用方式与Pro*C多线程程模式下每个线程各自进行CONTEXT相关申请与释放的处理逻辑一致。

- 目前ecpg不支持EXEC SQL COMMIT WORK RELEASE。

  说明

  在ecpg中，当业务语句执行COMMIT之后，并没有RELEASE选项，需要通过调用EXEC SQL DISCONNECT、EXEC SQL CLOSE等命令来实现相关资源的释放。Pro*C中EXEC SQL COMMIT带有RELEASE选项。用于释放程序持有的所有连接、游标等资源信息。

- 目前ecpg不支持EXEC SQL ENABLE THREAD。

  说明

  ecpg编译选项中开启宏定义，在main函数的.pgc文件中定义（define）ENABLE_THREAD_SAFETY。

- 目前ecpg不支持存储过程、Package、匿名块、闪回等特性语法。

5.9.16 ecpg 接口参考

ecpg接口参考主要介绍pgtypes库提供的用户在嵌入式SQL-C源码程序中可使用的数据类型相关接口。pgtypes库将SQL数据类型映射到C语言数据类型，并提供一些接口实现其基本功能和运算。

5.9.16.1 区间类型

表5-97列出了ecpg提供的区间类型(interval)数据的常用接口：

表 5-97 区间类型常用接口

API接口	接口描述	说明
interval* PGTYPESinterval_new(void)	返回一个已分配的区间变量的指针。	该函数在堆上创建interval变量，返回值为interval*类型。
void PGTYPESinterval_free(interval* inval)	释放已经分配区间变量的内存。	释放PGTYPESinterval_new函数创建的interval*类型变量。

API接口	接口描述	说明
interval* PGTYPESinterval_from_asc(char* str, char** endptr)	解析文本表示的区间。	该函数解析输入字符串str并且返回一个已分配的区间变量的指针。目前ecpg解析整个字符串并且当前不支持把第一个非法字符的地址存储在*endptr中。可以把endptr设置为NULL。
char* PGTYPESinterval_to_asc(interval* span)	将一个区间类型的变量转换成它的文本表达。	该函数将span指向的区间变量转换成一个char*。
int PGTYPESinterval_copy(interval* intvlsrc, interval* intvldest)	复制一个区间类型的变量。	该函数将intvlsrc指向的区间变量复制到intvldest指向的区间变量。

示例

请参见**常用示例**章节。

5.9.16.2 数值类型

表5-98列出了ecpg提供的数值类型(numeric\decimal)数据的常用接口：

表 5-98 数值类型常用接口

API接口	接口描述	说明
numeric* PGTYPESnumeric_new(void)	请求一个指向新分配的numeric变量的指针。	该函数在堆上创建numeric变量，返回值为numeric*类型。
decimal* PGTYPESdecimal_new(void)	请求一个新分配的decimal变量的指针。	该函数在堆上创建decimal变量，返回值为decimal*类型。
void PGTYPESnumeric_free(numeric* var)	释放一个numeric类型变量的内存。	该函数释放通过PGTYPESnumeric_new函数创建的numeric*类型变量。
void PGTYPESdecimal_free(decimal*)	释放一个decimal类型变量的内存。	该函数释放通过PGTYPESdecimal_new函数创建的decimal*类型变量。

API接口	接口描述	说明
numeric* PGTYPESnumeric_from_asc(char* str, char** endptr)	从字符串中解析一个numeric类型。	有效格式如：-2，.794, +3.44，592.49E07或者-32.84e-4。如果值解析成功，则返回一个有效指针，否则返回空指针。ecpg支持解析完整的字符串，目前不支持存储在*endptr中的第一无效字符的地址，可以设置endptr为空。
char* PGTYPESnumeric_to_asc(numeric* num, int dscale)	返回由malloc分配的字符串的指针，它包含numeric类型num的字符串表达。	numeric值将被使用dscale小数位打印，必要时会取整。
int PGTYPESnumeric_add(numeric* var1, numeric* var2, numeric* result)	将两个numeric变量相加放到第三个numeric变量中。	该函数把变量var1和var2相加放到结果变量result中。成功时该函数返回0，出错时返回-1。
int PGTYPESnumeric_sub(numeric* var1, numeric* var2, numeric* result)	将两个numeric变量相减并且把结果返回到第三个numeric变量。	该函数把变量var2从变量var1中减除。该操作的结果被存储在变量result中。成功时该函数返回0，出错时返回-1。
int PGTYPESnumeric_mul(numeric* var1, numeric* var2, numeric* result)	将两个numeric变量相乘并且把结果返回到第三个numeric变量。	该函数把变量var1和var2相乘。该操作的结果被存储在变量result中。成功时该函数返回0，出错时返回-1。
int PGTYPESnumeric_div(numeric* var1, numeric* var2, numeric* result)	将两个numeric变量相除并且把结果返回到第三个numeric变量。	该函数用变量var2除变量var1。该操作的结果被存储在变量result中。成功时该函数返回0，出错时返回-1。
int PGTYPESnumeric_cmp(numeric* var1, numeric* var2)	比较两个numeric变量。	该函数比较两个numeric变量。错误时会返回INT_MAX。成功时，该函数返回三种可能结果之一： ● var1大于var2则返回1。 ● 如果var1小于var2则返回-1。 ● 如果var1和var2相等则返回0。

API接口	接口描述	说明
int PGTYPESnumeric_from_int(signed int int_val, numeric* var)	将一个整数变量转换成一个numeric变量。	该函数接受一个有符号整型变量并且把它存储在numeric变量var中。成功时返回0，失败时返回-1。
int PGTYPESnumeric_from_long(signed long int long_val, numeric* var)	将一个长整型变量转换成一个numeric变量。	该函数接受一个有符号长整型变量并且把它存储在numeric变量var中。成功时返回0，失败时返回-1。
int PGTYPESnumeric_copy(numeric* src, numeric* dst)	将一个numeric变量复制到另一个中。	该函数把src指向的变量的值复制到dst指向的变量。成功时该函数返回0，出错时返回-1。
int PGTYPESnumeric_from_double(double d, numeric* dst)	将一个双精度类型的变量转换成一个numeric变量。	该函数接受一个双精度类型的变量并且把结果存储在dst指向的变量中。成功时该函数返回0，出错时返回-1。
int PGTYPESnumeric_to_double(numeric* nv, double* dp)	将一个numeric类型的变量转换成双精度类型。	该函数将nv指向的变量中的numeric值转换成dp指向的双精度变量。成功时该函数返回0，出错时返回-1（包括溢出）。溢出时，全局变量errno将被额外地设置成PGTYPES_NUM_OVERFLOW。
int PGTYPESnumeric_to_int(numeric* nv, int* ip)	将一个numeric类型的变量转换成整数类型。	该函数将nv指向的变量的numeric值转换成ip指向的整数变量。成功时该函数返回0，出错时返回-1（包括溢出）。溢出时，全局变量errno将被额外地设置成PGTYPES_NUM_OVERFLOW。
int PGTYPESnumeric_to_long(numeric* nv, long* ip)	将一个numeric类型的变量转换成长整型类型。	该函数将nv指向的变量的numeric值转换成ip指向的长整型变量。成功时该函数返回0，出错时返回-1（包括溢出）。溢出时，全局变量errno将被额外地设置成PGTYPES_NUM_OVERFLOW。

API接口	接口描述	说明
int PGTYPESnumeric_to_decimal(numeric* src, decimal* dst)	将一个numeric类型的变量转换成decimal类型。	该函数将src指向的变量的numeric值转换成dst指向的decimal变量。成功时该函数返回0，出错时返回-1（包括溢出）。溢出时，全局变量errno将被额外地设置成PGTYPES_NUM_OVERFLOW。
int PGTYPESnumeric_from_decimal(decimal* src, numeric* dst)	将一个decimal类型的变量转换成numeric类型。	该函数将src指向的变量的decimal值转换成dst指向的numeric变量。成功时该函数返回0，出错时返回-1（包括溢出）。

示例

请参见**常用示例**章节。

5.9.16.3 日期类型

表5-99列出了ecpg提供的日期类型(date)数据的常用接口：

表 5-99 日期类型常用接口

API接口	接口描述	说明
date* PGTYPESdate_new(void)	返回一个已分配的date变量的指针。	该函数在堆上创建date变量，返回值为date*类型。
void PGTYPESdate_free(date*)	释放已经分配date变量的内存。	释放通过PGTYPESdate_free函数创建的date*类型变量。
date PGTYPESdate_from_asc(char* str, char** endptr)	从日期的文本表达解析一个日期。	该函数接收一个C的字符串str以及一个指向C字符串的指针endptr。ecpg将文本表达的日期解析为字符串形式。当前不支持将第一个非法字符的地址存储在*endptr中，可以把endptr设置为NULL。 注意该函数假定日期格式按照MDY进行格式化，并且当前在ecpg中没有变体可以改变这种格式。

API接口	接口描述	说明
char* PGTYPESdate_to_asc(date dDate)	返回一个日期变量的文本表达。	该函数接收日期dDate作为它的唯一参数。以YYYY-MM-DD格式输出。
date PGTYPESdate_from_timestamp(timestamp dt)	从一个时间戳中抽取日期部分。	该函数接收一个时间戳作为它的唯一参数并且从这个时间戳返回抽取的日期部分。
void PGTYPESdate_julmdy(date jd, int* mdy)	从一个日期类型变量中抽取日、月和年的值。	该函数接收日期jd以及一个指向有3个整数值的数组mdy的指针。变量名就表明了顺序：mdy[0]表示月份，mdy[1]表示日，而mdy[2]表示年。
void PGTYPESdate_mdyjul(int * mdy, date* jdate)	使用指定的整型数值创建日期。	这个函数接收3个整数（mdy）组成的数组作为其第一个参数，三个整数分别用来表示日、月和年。第二个参数是一个指向日期类型变量的指针，它被用来保存操作的结果。
int PGTYPESdate_dayofweek(date d)	为一个日期值返回表示它是星期几的数字。	这个函数接收日期变量d作为它唯一的参数并且返回一个整数说明这个日期是星期几。
void PGTYPESdate_today(date * d)	得到当前日期。	该函数接收一个指向日期变量（d）的指针并且把该参数设置为当前日期。 ● 0：星期日 ● 1：星期一 ● 2：星期二 ● 3：星期三 ● 4：星期四 ● 5：星期五 ● 6：星期六

API接口	接口描述	说明
int PGTYPESdate_defmt_asc(date* d, const char* fmt, char* str)	使用一个格式掩码把一个C的char*返回串转换成一个日期类型的值。	该函数接收一个用来保存操作结果的指向日期值的指针（d）、用于解析日期的格式掩码（fmt）以及包含日期文本表达的C char*串（str）。该函数期望文本表达匹配格式掩码。不需要字符串和格式掩码的一一映射。该函数只分析相继顺序并且查找表示年份位置的文字yy或者yyyy、表示月份位置的mm以及表示日位置的dd。注意不能使用中文年月日，例如，七月八日。 合法输入示例如下（fmt,str）： yy/mm/dd 在2525年，7月28号人类仍存活 dd-mm-yy 2525年7月28号 yy/mm/dd 1994, February 3rd
int PGTYPESdate_fmt_asc(dat dDate, const char* fmtstring, char* outbuf)	使用一个格式掩码将一个日期类型的变量转换成它的文本表达。	该函数接收要转换的日期（dDate）、格式掩码（fmtstring）以及将要保存日期的文本表达的字符串（outbuf）。 成功时，返回0；如果发生错误，则返回一个负值。 合法输入示例如下： 112359 //mmddyy 59/11/23 //yy/mm/dd Nov.23,1959 //mmm.dd,yyyy Mon,Nov.23,1959 //ddd,mmm.dd,yyyy 格式说明： • dd：一个月中的第几天。 • mm：一年中的第几个月。 • yy：两位数的年份。 • yyyy：四位数的年份。 • ddd：星期几的名称。 • mmm：月份的名称。

示例

请参见**常用示例**章节。

5.9.16.4 时间戳类型

表5-100列出了ecpg提供的时间戳(timestamp)数据的常用接口：

表 5-100 时间戳类型常用接口

API接口	接口描述	说明
timestamp PGTYPEStimestamp_from_asc(char *str, char **endptr)	从文本解析一个时间戳并放到一个时间戳变量中。	该函数接收一个解析(str)字符串和指向C char*(endptr)指针。成功时函数返回解析的时间戳，产生错误时返回PGTYPESInvalidTimestamp，并且设置errno为PGTYPES_TS_BAD_TIMESTAMP。 PGTYPEStimestamp_from_asc的合法输入如下所示： 1999-01-08 04:05:06 January 8 04:05:06 1999 PST 1999-Jan-08 04:05:06.789-8 1999-01-08 04:05:06.789（忽略了时区指示符） J2451187 04:05-08:00 1999-01-08 04:05:00（忽略了时区指示符）
char *PGTYPEStimestamp_to_asc(timestamp tstamp)	将日期转换成char*字符串。	该函数接收时间戳tstamp作为它的唯一参数并且返回一个分配好的包含该时间戳文本表达式的字符串。结果必须用PGTYPESchar_free()释放。
void PGTYPEStimestamp_current(timestamp *ts)	获取当前时间戳。	该函数获取当前时间戳，并且将它保存到ts指向的时间戳变量中。
int PGTYPEStimestamp_fmt_asc(timestamp *ts, char *output, int str_len, char *fmtstr)	使用一个格式掩码将一个时间戳变量转换成一个 C char* 。	该函数接收一个指向时间戳的指针（ts）、一个指向输出缓冲区的指针（output）、为输出缓冲区分配的最大长度（str_len）以及用于转换的格式掩码（fmtstr）作为参数。 成功时，该函数返回0；如果有错误发生，则返回一个负值。

API接口	接口描述	说明
int PGTYPEStimestamp_sub(timestamp *ts1, timestamp *ts2, interval *iv)	从一个时间戳中减去另一个时间戳并且把结果保存在一个区间类型的变量中。	该函数将从ts1指向的时间戳变量中减去ts2指向的时间戳变量，并且将把结果存储在iv指向的区间变量中。 成功时，该函数返回0；发生错误时则返回一个负值。
int PGTYPEStimestamp_defmt_asc(char *str, char *fmt, timestamp *d)	用一个格式掩码从时间戳的文本表达解析其值。	该函数接收一个放在变量str中的时间戳文本表达以及放在变量fmt中要使用的格式掩码。结果将被存放在d指向的变量中。 如果格式掩码fmt是NULL，该函数将回退到使用默认的格式掩码%Y-%m-%d %H:%M:%S。
int PGTYPEStimestamp_add_interval(timestamp *tin, interval *span, timestamp *tout)	把一个interval变量加到一个时间戳变量上。	该函数接收一个指向时间戳变量的指针tin以及一个指向interval变量的指针span。它把interval加到时间戳上，然后将结果时间戳保存在tout指向的变量中。 成功时该函数返回0，如果发生错误则返回一个负值。
int PGTYPEStimestamp_sub_interval(timestamp* tin, interval* span, timestamp* tout)	从一个timestamp时间戳变量中减去interval变量。	该函数从tin指向的时间戳变量中减去span指向的interval变量，然后把结果保存在tout指向的变量中。 成功时该函数返回0，如果发生错误则返回一个负值。

示例

请参见**常用示例**章节。

5.10 附录

5.10.1 JDBC

5.10.1.1 数据类型映射关系

数据类型、JAVA变量类型以及JDBC类型索引关系如下：

兼容模式	Gauss Kernel数据类型	JAVA变量类型	JDBC类型索引
A/B	oid	java.lang.Long	java.sql.Types.BIGINT
A/B	numeric	java.math.BigDecimal	java.sql.Types.NUMERIC
A/B	tinyint	java.lang.Integer	java.sql.Types.TINYINT
A/B	smallint	java.lang.Integer	java.sql.Types.SMALLINT
A/B	bigint	java.lang.Long	java.sql.Types.BIGINT
A/B	float4	java.lang.Float	java.sql.Types.REAL
A/B	float8	java.lang.Double	java.sql.Types.DOUBLE
A/B	char	java.lang.String	java.sql.Types.CHAR
A/B	character	java.lang.String	java.sql.Types.CHAR
A/B	bpchar	java.lang.String	java.sql.Types.CHAR
A/B	character varying	java.lang.String	java.sql.Types.VARCHAR
A/B	varchar	java.lang.String	java.sql.Types.VARCHAR
A/B	text	java.lang.String	java.sql.Types.VARCHAR
A/B	name	java.lang.String	java.sql.Types.VARCHAR
A/B	bytea	byte[]	java.sql.Types.BINARY
A/B	blob	java.sql.Blob	java.sql.Types.BLOB
A/B	clob	java.sql.Clob	java.sql.Types.CLOB
A/B	boolean	java.lang.Boolean	java.sql.Types.BIT
B	date	java.sql.Date	java.sql.Types.DATE
A/B	time	java.sql.Time	java.sql.Types.TIME
A/B	timetz	java.sql.Time	java.sql.Types.TIME
A/B	timestamp	java.sql.Timestamp	java.sql.Types.TIMESTAMP
A/B	smalldatetime	java.sql.Timestamp	java.sql.Types.TIMESTAMP
A/B	timestamptz	java.sql.Timestamp	java.sql.Types.TIMESTAMP

兼容模式	Gauss Kernel数据类型	JAVA变量类型	JDBC类型索引
A/B	refcursor	java.sql.ResultSet	java.sql.Types.REF_CURSOR java.sql.Types.OTHER

5.10.1.2 日志管理

GaussDB JDBC驱动程序支持使用日志记录来帮助解决在应用程序中使用GaussDB JDBC驱动程序时遇到的问题。GaussDB JDBC支持如下两种日志管理方式：

- 对接应用程序使用的SLF4J日志框架。
- 对接应用程序使用的JdkLogger日志框架。

SLF4J和JdkLogger是业界Java应用程序日志管理的主流框架，描述应用程序如何使用这些框架超出了本文范围，用户请参考对应的官方文档（SLF4J：**http://www.slf4j.org/manual.html**，JdkLogger：**https://docs.oracle.com/javase/8/docs/technotes/guides/logging/overview.html**）。

方式一：对接应用程序的 SLF4J 日志框架

在建立连接时，url配置logger=Slf4JLogger。此方式支持日志管控，SLF4J可通过文件中的相关配置实现强大的日志管控功能，建议使用此方式进行日志管理。

可采用Log4j或Log4j2来实现SLF4J。当采用Log4j实现SLF4J，需要添加如下jar包（*区分版本）：log4j-*.jar、slf4j-api-*.jar、slf4j-log4*-*.jar。同时指定log4j.properties配置文件的路径（相对路径或者绝对路径均可）。

若采用Log4j2实现SLF4J，需要添加如下jar包（*区分版本）：log4j-api-*.jar、log4j-core-*.jar、log4j-slf4j18-impl-*.jar、slf4j-api-*-alpha1.jar。同时指定log4j2.xml配置文件的路径（相对路径或者绝对路径均可）。

须知

此方式依赖SLF4J的通用API接口，如org.slf4j.LoggerFactory.getLogger(String name)、org.slf4j.Logger.debug(String var1)、org.slf4j.Logger.info(String var1)、org.slf4j.Logger.warn(String warn)、org.slf4j.Logger.warn(String warn)等，若以上接口发生变更，日志将无法打印。

示例：

```
//注意：调用时需要指定log4j.properties或者log4j2.xml配置文件路径。

//使用log4j.properties文件来配置日志记录器的属性，例如日志级别、输出目标等。通过调用PropertyConfigurator的configure方法并传递log4j.properties文件的路径，将这些属性加载到应用程序中。
//PropertyConfigurator.configure("log4j.properties");

//创建Log4j2配置源，指定配置文件log4j2.xml的路径。并初始化日志系统。
//ConfigurationSource source = new ConfigurationSource(new FileInputStream("log4j2.xml"));
//Configurator.initialize(null, source);
```

```
public static Connection GetConnection(String username, String passwd){

   String sourceURL = "jdbc:opengauss://$ip:$port/database?logger=Slf4JLogger";
   Connection conn = null;

   try{
      //创建连接
      conn = DriverManager.getConnection(sourceURL,username,passwd);
      System.out.println("Connection succeed!");
   }catch (Exception e){
      e.printStackTrace();
      return null;
   }
   return conn;
}
```

log4j.properties配置文件：

```
log4j.logger.com.huawei.opengauss.jdbc=ALL, log_gsjdbc

# 默认文件输出配置
log4j.appender.log_gsjdbc=org.apache.log4j.RollingFileAppender
log4j.appender.log_gsjdbc.Append=true
log4j.appender.log_gsjdbc.File=gsjdbc.log
log4j.appender.log_gsjdbc.Threshold=TRACE
log4j.appender.log_gsjdbc.MaxFileSize=10MB
log4j.appender.log_gsjdbc.MaxBackupIndex=5
log4j.appender.log_gsjdbc.layout=org.apache.log4j.PatternLayout
log4j.appender.log_gsjdbc.layout.ConversionPattern=%d %p %t %c - %m%n
log4j.appender.log_gsjdbc.File.Encoding = UTF-8
```

log4j2.xml配置文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration status="OFF">
   <appenders>
      <Console name="Console" target="SYSTEM_OUT">
         <PatternLayout pattern="%d %p %t %c - %m%n"/>
      </Console>
      <File name="FileTest" fileName="test.log">
         <PatternLayout pattern="%d %p %t %c - %m%n"/>
      </File>
      <!--JDBC Driver日志文件输出配置，支持日志回卷，设定日志大小超过10MB时，创建新的文件，新文件的
命名格式为：年-月-日-文件编号-->
      <RollingFile name="RollingFileJdbc" fileName="gsjdbc.log" filePattern="%d{yyyy-MM-dd}-%i.log">
         <PatternLayout pattern="%d %p %t %c - %m%n"/>
         <Policies>
            <SizeBasedTriggeringPolicy size="10 MB"/>
         </Policies>
      </RollingFile>
   </appenders>
   <loggers>
      <root level="all">
         <appender-ref ref="Console"/>
         <appender-ref ref="FileTest"/>
      </root>
      <!--指定JDBC Driver日志，级别为：all，可查看所有日志，输出到gsjdbc.log文件中-->
      <logger name="com.huawei.opengauss.jdbc" level="all" additivity="false">
         <appender-ref ref="RollingFileJdbc"/>
      </logger>
   </loggers>
</configuration>
```


须知

- 在配置文件中，可以指定JDBC与上层业务的配置路径，将JDBC日志与业务日志分别存储到不同文件。
- 同一个项目中配置多个GaussDB数据源时，如果存在部分业务配置了logger=Slf4JLogger，部分业务未配置，会互相影响日志配置。建议这种情况下连接串统一配置。

方式二：对接应用程序使用的 JdkLogger 日志框架

两种方式：采用logging.properties配置文件、直接在示例代码中进行配置。

- 默认的Java日志记录框架将其配置存储在名为logging.properties的文件中。Java会在Java安装目录的文件夹中安装全局配置文件。logging.properties文件也可以创建并与单个项目一起存储。

  logging.properties配置文件：

  ```
  # 指定处理程序为文件。
  handlers= java.util.logging.FileHandler

  # 指定默认全局日志级别。
  .level= ALL

  # 指定日志输出管控标准。
  java.util.logging.FileHandler.level=ALL
  java.util.logging.FileHandler.pattern = gsjdbc.log
  java.util.logging.FileHandler.limit = 500000
  java.util.logging.FileHandler.count = 30
  java.util.logging.FileHandler.formatter = java.util.logging.SimpleFormatter
  java.util.logging.FileHandler.append=false
  ```

- 直接在示例代码中进行配置：

  ```
  System.setProperty("java.util.logging.FileHandler.pattern","jdbc.log");
  FileHandler fileHandler = new FileHandler(System.getProperty("java.util.logging.FileHandler.pattern"));
  Formatter formatter = new SimpleFormatter();
  fileHandler.setFormatter(formatter);
  Logger logger = Logger.getLogger("com.huawei.opengauss.jdbc");
  logger.addHandler(fileHandler);
  logger.setLevel(Level.ALL);
  logger.setUseParentHandlers(false);
  ```

链路跟踪功能

GaussDB JDBC驱动程序提供了应用到数据库的链路跟踪功能，用于将数据库端离散的SQL和应用程序的请求关联起来。该功能需要应用开发者实现com.huawei.opengauss.jdbc.log.Tracer接口类，并在url中指定接口实现类的全类名。

url示例：

```
String url = "jdbc:opengauss://$ip:$port/database?traceInterfaceClass=xxx.xxx.xxx.OpenGaussTraceImpl";
```

com.huawei.opengauss.jdbc.log.Tracer接口类定义如下：

```
public interface Tracer {
// Retrieves the value of traceId.
String getTraceId();
}
```

com.huawei.opengauss.jdbc.log.Tracer接口实现类示例：

```
import com.huawei.opengauss.jdbc.log.Tracer;

public class OpenGaussTraceImpl implements Tracer {
    private static MDC mdc = new MDC();
```

```
    private final String TRACE_ID_KEY = "traceId";

    public void set(String traceId) {
        mdc.put(TRACE_ID_KEY, traceId);
    }

    public void reset() {
        mdc.clear();
    }

    @Override
    public String getTraceId() {
        return mdc.get(TRACE_ID_KEY);
    }
}
```

上下文映射示例，用于存放不同请求生成的traceId。

```
import java.util.HashMap;

public class MDC {
    static final private ThreadLocal<HashMap<String, String>> threadLocal = new ThreadLocal<>();

    public void put(String key, String val) {
        if (key == null || val == null) {
            throw new IllegalArgumentException("key or val cannot be null");
        } else {
            if (threadLocal.get() == null) {
                threadLocal.set(new HashMap<>());
            }
            threadLocal.get().put(key, val);
        }
    }

    public String get(String key) {
        if (key == null) {
            throw new IllegalArgumentException("key cannot be null");
        } else if (threadLocal.get() == null) {
            return null;
        } else {
            return threadLocal.get().get(key);
        }
    }

    public void clear() {
        if (threadLocal.get() == null) {
            return;
        } else {
            threadLocal.get().clear();
        }
    }
}
```

业务使用traceId示例，前置条件需要先建表test_trace_id (id int,name varchar(20))。

```
String traceId = UUID.randomUUID().toString().replaceAll("-", "");
OpenGaussTraceImpl openGaussTrace = new OpenGaussTraceImpl();
openGaussTrace.set(traceId);
Connection con = DriverManager.getConnection(url, user, password);
pstm = con.prepareStatement("SELECT * FROM test_trace_id WHERE id = ?");
pstm.setInt(1, 1);
pstm.execute();
pstm = con.prepareStatement("INSERT INTO test_trace_id VALUES(?,?)");
pstm.setInt(1, 2);
pstm.setString(2, "test");
pstm.execute();
openGaussTrace.reset();
```


说明

- 使用链路跟踪功能时，应用层链路功能由业务保证。
- 应用必须向JDBC暴露获取traceId的接口，并将该接口实现类配置到JDBC连接字符串中。
- 同一请求的不同SQL使用的traceId须相同。
- 应用传入的traceId不能超过32字节，否则多余字节将被截断。

5.10.1.3 常见问题处理

5.10.1.3.1 batchMode 设置错误

问题现象

设置url参数batchMode=on且reWriteBatchedInserts=true，使用JDBC批量插入数据，提示绑定参数数量与语句需要的参数数量不一致：

```
bind message supplies * parameters, but prepared statement "" requires *
```

示例：

```
// conn是已经创建的Connection对象，创建该connection的url参数包含
&batchMode=on&reWriteBatchedInserts=true
// 批量绑定参数后执行，绑定参数数量会与改写后的INSERT语句的栏位数不匹配，提示异常。
// java.sql.BatchUpdateException: bind message supplies 3 parameters, but prepared statement "" requires 6

PreparedStatement stmt = conn.prepareStatement("INSERT INTO test_tbl VALUES (?, ?, ?)");

stmt.setInt(1, 1);
stmt.setString(2, "aaa");
stmt.setString(3, "bbbb");
stmt.addBatch();

stmt.setInt(1, 1);
stmt.setString(2, "aaa");
stmt.setString(3, "bbbb");
stmt.addBatch();

stmt.setInt(1, 1);
stmt.setString(2, "aaa");
stmt.setString(3, "bbbb");
stmt.addBatch();

stmt.executeBatch();
```

原因分析

将参数reWriteBatchedInserts设置为true时，批量语句会将多条SQL合并为一条，导致语句中预留参数栏位数发生变化。如果batchMode=on，会按照合并前的SQL绑定参数，导致绑定参数数量与语句需要的参数数量不一致。

处理方法

reWriteBatchedInserts设置为true时，将batchMode设置为off。

5.10.1.3.2 Hibernate 框架插入数据开启校验时报错

问题现象

客户从A模式数据库迁移到GaussDB数据库，表结构使用DRS工具进行迁移，迁移后客户原业务代码不可用，Hibernate框架校验表结构报错。

```
Schema-validation: wrong column type encountered in column [execute_time] in table [abnormal_event]; found [int8 (Types#BIGINT)], but expecting [int4 (Types#INTEGER)]
```

示例：

- 原A模式数据库中Student表结构为如下格式：
  ```
  id number(15,0)
  sid number(15,0)
  age number(10,0)
  ```
- 使用DRS工具迁移到GaussDB数据库后，Student表结构为如下格式：
  ```
  id bigint
  sid bigint
  age Integer
  ```
- 当实际数据库中Student实体类信息如下时：
  ```
  Long id
  Long sid
  Long age  --Hibernate框架校验表结构报错
  ```

hibernate.cfg.xml配置文件：

```
<hibernate-configuration>
    <session-factory>
        <!--GaussDB连接信息-->
        <property name="connection.driver_class">com.huawei.opengauss.jdbc.Driver</property>
        <property name="connection.url">jdbc:opengauss://x.x.x.x:xx/postgres?currentSchema=public</
property>
        <property name="connection.username">xxxxxx</property>
        <property name="connection.password">xxxxxx</property>

        <!-- 以下为可选配置 -->

        <!--是否支持方言 -->
        <!--在pgsql兼容模式下，必须有如下配置-->
         <property name="dialect">org.hibernate.dialect.PostgreSQL92Dialect</property>
        <!--在A数据库兼容模式下，推荐换如下配置-->
<!--        <property name="dialect">org.hibernate.dialect.OracleDialect</property>-->

        <property name="met"></property>

        <!--执行CURD时是否打印sql语句  -->
        <property name="show_sql">true</property>
        <!--自动建表-->
<!--        <property name="hbm2ddl.auto">create</property>-->
        <!--插入数据时开启校验-->
        <property name="hbm2ddl.auto">validate</property>
        <!--关闭校验-->
<!--        <property name="hbm2ddl.auto">none</property>-->

        <!-- 资源注册 (实体类映射文件)-->
        <mapping resource="student.xml"/>
    </session-factory>
</hibernate-configuration>
```

插入数据的主函数：

```
// 创建要测试的对象
Student student = new Student();
student.setId(20L);
student.setAge(222L);
student.setSid(222L);
```



```
// 开启事务,基于session得到
Configuration conf = new Configuration().configure();
SessionFactory sessionFactory = conf.buildSessionFactory();
Session session = sessionFactory.openSession();
Transaction transaction = session.beginTransaction();

// 通过session保存数据
session.save(student);

// 提交事务
transaction.commit();

// 操作完毕,关闭session连接对象
session.close();
```

原因分析

1. A模式数据库能够插入成功并通过校验：Hibernate框架在校验数据时，通过判断插入数据类型java.lang.Long的TypeCode，与表结构的number类型的TypeCode进行对比，结果不一致，随后会把插入数据的long类型的sqlType与表结构的Type作比较，将long类型转成number(19,0)（对应关系由org.hibernate.dialect.OracleDialect进行维护），因为表结构的type是number，number(19,0)以number开头，所以校验可以通过。使用A模式数据库时，业务代码涉及整数类型的数据，可以使用java.lang.Long类型插入。
2. GaussDB数据库校验不同，关闭校验后数据插入成功：Hibernate框架在插入数据做校验时，业务代码是java.lang.Long类型，和表数据类型bigint的TypeCode是对应的，所以不报错。但是表数据类型是Integer时无法对应，首先判断业务代码java.lang.Long类型的TypeCode，与表数据Integer的TypeCode比较，结果不一致，之后根据org.hibernate.dialect.PostgreSQLDialect对应关系，会将long类型转换为int8，将表数据类型Integer转换为int4，无法对应，所以校验失败。使用GaussDB数据库时，业务代码涉及整数类型的数据，必须使用表类型对应的Java整型。

处理方法

可使用以下方式进行问题处理：

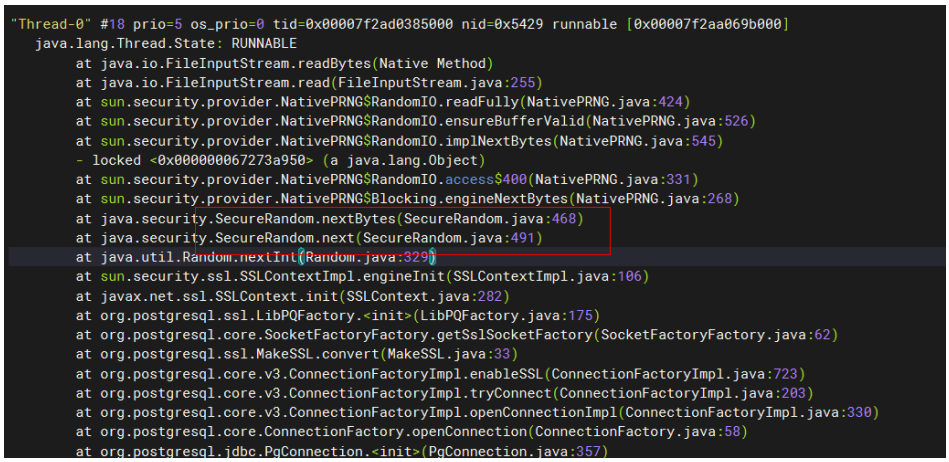
- 关闭校验功能。
- 客户修改业务代码，针对不同的表数据类型，采用对应的Java整型。

5.10.1.3.3 使用 SSL 方式建连报错或阻塞

问题现象

JDBC使用SSL方式建立连接时，会在客户端获取强随机数，建立连接过程中可能出现不同场景的报错信息。

场景1：如下报错：



场景2：建连阻塞。如果连接串中配置了loginTimeout后，会报Connection attempt timed out，如果不配置该参数，会一直阻塞。

原因分析

客户端环境随机数产生的速度太慢，无法满足产品要求，熵源不足，导致服务启动失败。当前已知在一些Linux环境中存在此问题。

处理方法

- 方法1：启动客户端环境中的haveged服务，增加系统熵池熵值以提高读取随机数的速度。

  启动命令为：

  ```
  systemctl start haveged
  ```

- 方法2：调整客户端jdk配置。

  打开$JAVA_PATH/jre/lib/security/java.security文件，修改以下两个配置项：

  ```
  securerandom.source=file:/dev/./urandom
  securerandom.strongAlgorithms=NativePRNGNonBlocking:SUN
  ```

  说明

  方法2的本质是在获取强随机数时，使用伪随机数代替，减少需要消耗的熵值。会影响客户端所有使用该jdk的应用，在获取强随机数时会使用伪随机数代替。

5.10.2 libpq

5.10.2.1 连接参数说明

表 5-101 连接参数

参数	描述
host	要连接的主机名。如果主机名以斜杠开头，则它声明使用Unix域套接字通讯而不是TCP/IP通讯，该值就是套接字文件所存储的目录。如果没有声明host，那么默认是与位于/tmp目录（或者安装GaussDB的时候声明的套接字目录）里面的Unix-域套接字连接。在没有Unix域套接字的机器上，默认与localhost连接。 接受以“,”分割的字符串来指定多个主机名，支持指定多个主机名。
hostaddr	与之连接的主机的IP地址，是标准的IPv4地址格式，比如，172.28.40.9。如果机器支持IPv6，那么也可以使用IPv6的地址。如果声明了一个非空的字符串，那么使用TCP/IP通讯机制。 接受以“,”分割的字符串来指定多个IP地址，支持指定多个IP地址。 使用hostaddr取代host可以让应用避免一次主机名查找，这一点对于那些有时间约束的应用来说可能是非常重要的。不过，GSSAPI或SSPI认证方法要求主机名（host）。因此，应用下面的规则： 1. 如果声明了不带hostaddr的host那么就强制进行主机名查找。 2. 如果声明中没有host，hostaddr的值给出服务器网络地址。如果认证方法要求主机名，那么连接尝试将失败。 3. 如果同时声明了host和hostaddr，那么hostaddr的值作为服务器网络地址。host的值将被忽略，除非认证方法需要它，在这种情况下它将被用作主机名。 **须知** • 如果host不是网络地址hostaddr处的服务器名，那么认证很有可能失败。 • 如果主机名（host）和主机地址都没有，那么libpq将使用一个本地的Unix域套接字进行连接，或者是在没有Unix域套接字的机器上，它将尝试与localhost连接。
port	主机服务器的端口号，或者在Unix域套接字连接时的套接字扩展文件名。 接受以“,”分割的字符串来指定多个端口号，支持指定多个端口号。
user	要连接的用户名，缺省是与运行该应用的用户操作系统名同名的用户。
dbname	数据库名，缺省和用户名相同。
password	如果服务器要求密码认证，所用的密码。
connect_timeout	连接的最大等待时间，以秒计（用十进制整数字符串书写），0或者不声明表示无穷。不建议把连接超时的值设置小于2秒。
client_encoding	为这个连接设置client_encoding配置参数。除了对应的服务器选项接受的值，还可以使用auto从客户端中的当前环境中确定正确的编码（Unix系统上是LC_CTYPE环境变量）。

参数	描述
tty	忽略（以前，该参数指定了发送服务器调试输出的位置）。
options	添加命令行选项以在运行时发送到服务器。
application_name	为application_name配置参数指定一个值，表明当前用户身份。
fallback_application_name	为**application_name**配置参数指定一个后补值。如果通过一个连接参数或PGAPPNAME环境变量没有为application_name给定一个值，将使用这个值。在一般工具程序中，若设置一个默认名，但不希望这个默认名被用户覆盖，可以通过指定一个后补值来实现。
keepalives	控制客户端侧的TCP保持激活是否使用。缺省值是1，意思为打开，但是如果不想要保持激活，可以更改为0，意思为关闭。通过Unix域套接字做的连接忽略这个参数。
keepalives_idle	在TCP应该发送一个保持激活的信息给服务器之后，控制不活动的秒数。0值表示使用系统缺省。通过Unix域套接字做的连接或者如果禁用了保持激活则忽略这个参数。
keepalives_interval	在TCP保持激活信息没有被应该传播的服务器承认之后，控制秒数。0值表示使用系统缺省。通过Unix域套接字做的连接或者如果禁用了保持激活则忽略这个参数。
keepalives_count	控制TCP发送保持激活信息的次数。0值表示使用系统缺省。通过Unix域套接字做的连接或者如果禁用了保持激活则忽略这个参数。
tcp_user_timeout	在支持TCP_USER_TIMEOUT套接字选项的操作系统上，指定传输的数据在TCP连接被强制关闭之前可以保持未确认状态的最大时长。0值表示使用系统缺省。通过Unix域套接字做的连接忽略这个参数。
rw_timeout	设置客户端连接读写超时时间。
sslmode	启用SSL加密的方式： • disable：不使用SSL安全连接。 • allow：如果数据库服务器要求使用，则可以使用SSL安全加密连接，但不验证数据库服务器的真实性。 • prefer：如果数据库支持，那么首选使用SSL安全加密连接，但不验证数据库服务器的真实性。 • require：必须使用SSL安全连接，但是只做了数据加密，而并不验证数据库服务器的真实性。 • verify-ca：必须使用SSL安全连接，当前windows ODBC不支持cert方式认证。 • verify-full：必须使用SSL安全连接，当前windows ODBC不支持cert方式认证。

参数	描述
sslcompression	如果设置为1（默认），SSL连接之上传送的数据将被压缩（这要求OpenSSL版本为0.9.8或更高）。如果设置为0，压缩将被禁用（这要求OpenSSL版本为1.0.0或更高）。如果建立的是一个没有SSL的连接，这个参数会被忽略。如果使用的OpenSSL版本不支持该参数，它也会被忽略。压缩会占用CPU时间，但是当瓶颈为网络时可以提高吞吐量。如果CPU性能是限制因素，禁用压缩能够改进响应时间和吞吐量。
sslcert	这个参数指定客户端SSL证书的文件名。如果没有建立SSL连接，这个参数会被忽略。
sslkey	这个参数指定用于客户端证书的密钥位置。它能够指定一个从外部“引擎”（引擎是OpenSSL的可载入模块）得到的密钥。一个外部引擎说明应该由一个冒号分隔的引擎名称以及一个引擎相关的关键标识符组成。如果没有建立SSL连接，这个参数会被忽略。
sslrootcert	这个参数指定一个包含SSL证书机构（CA）证书的文件名称。如果该文件存在，服务器的证书将被验证是由这些机构之一签发。
sslcrl	这个参数指定SSL证书撤销列表（CRL）的文件名。列在这个文件中的证书如果存在，在尝试认证该服务器证书时会被拒绝。
requirepeer	这个参数指定服务器的操作系统用户，例如requirepeer=postgres。当建立一个Unix域套接字连接时，如果设置了这个参数，客户端在连接开始时检查服务器进程是否运行在指定的用户名之下。如果发现不是，该连接会被一个错误中断。这个参数能被用来提供与TCP/IP连接上SSL证书相似的服务器认证（注意，如果Unix域套接字在/tmp或另一个公共可写的位置，任何用户能启动一个在那里侦听的服务器。使用这个参数来保证所连接的是一个由可信用户运行的服务器）。这个选项只在实现了peer认证方法的平台上支持。
krbsrvname	当用GSSAPI认证时，要使用的Kerberos服务名。为了让Kerberos认证成功，这必须匹配在服务器配置中指定的服务名。
gsslib	用于GSSAPI认证的GSS库。只用在Windows上。设置为gssapi可强制libpq用GSSAPI库来代替默认的SSPI进行认证。
service	用于附加参数的服务名。它指定保持附加连接参数的pg_service.conf中的一个服务名。这允许应用只指定一个服务名，这样连接参数能被集中维护。
authtype	不再使用“authtype”，因此将其标记为“不显示”。将其保留在数组中，以免拒绝旧应用程序中的conninfo字符串，这些应用程序可能仍在尝试设置它。
remote_nodename	指定连接本地节点的远端节点名称。
localhost	指定在一个连接通道中的本地地址。
localport	指定在一个连接通道中的本地端口。

参数	描述
fencedUdfRPCMode	控制fenced UDF RPC协议是使用unix域套接字或特殊套接字文件名。缺省值是0，意思为关闭，使用unix domain socket模式，文件类型为“.s.PGSQL.%d”；但是要使用fenced udf ，文件类型为.s.fencedMaster_unixdomain，可以更改为1，意思为开启。
replication	这个选项决定是否该连接应该使用复制协议而不是普通协议。这是PostgreSQL的复制连接以及pg_basebackup之类的工具在内部使用的协议，但也可以被第三方应用使用。支持下列值，大小写无关： ● true、on、yes、1：连接进入到物理复制模式。 ● database：连接进入到逻辑复制模式，连接到dbname参数中指定的数据库。 ● false、off、no、0：该连接是一个常规连接，这是默认行为。 在物理或者逻辑复制模式中，仅能使用简单查询协议。
backend_version	传递到远端的后端版本号。
prototype	设置当前协议级别，默认：PROTO_TCP。
enable_ce	控制是否允许客户端连接全密态数据库。默认值为0，不开启密态功能。如果需要开启密态等值查询基本能力，则修改为1。。
connection_info	Connection_info是一个包含driver_name、driver_version、driver_path和os_user的json字符串。 如果不为NULL，使用connection_info忽略connectionExtraInf。 如果为NULL，生成与libpq相关的连接信息字符串，当connectionExtraInf为false时connection_info只有driver_name和driver_version。
connectionExtraInf	设置connection_info是否存在扩展信息，默认值为0，如果包含其他信息，则需要设置为1。
target_session_attrs	设定连接的主机的类型。主机的类型和设定的值一致时才能连接成功。指定多IP时才会校验此参数。target_session_attrs的设置规则如下： ● any：可以对所有类型的主机进行连接。 ● read-write：当连接的主机允许可读可写时，才进行连接。 ● read-only：仅对可读的主机进行连接。 ● primary（默认值）：仅对主备系统中的主机能进行连接。 ● standby：仅对主备系统中的备机进行连接。 ● prefer-standby：首先尝试找到一个备机进行连接。如果对hosts列表的所有机器都连接失败，那么尝试“any”模式进行连接。

5.10.3 日志输出相关参数介绍

用户可以根据自己的需要，通过修改实例数据目录下的postgresql.conf文件中特定的配置参数来控制日志的输出，从而更好地了解数据库的运行状态。

可调整的配置参数请参见表5-102。

表 5-102 配置参数

参数名称	描述	取值范围	备注
client_min_messages	配置发送到客户端信息的级别。	• DEBUG5 • DEBUG4 • DEBUG3 • DEBUG2 • DEBUG1 • LOG • NOTICE • WARNING • ERROR • FATAL • PANIC 默认值：NOTICE。	设置级别后，发送到客户端的信息包含所设级别及以下所有低级别会发送的信息。级别越低，发送的信息越少。
log_min_messages	配置写到服务器日志里信息的级别。	• DEBUG5 • DEBUG4 • DEBUG3 • DEBUG2 • DEBUG1 • INFO • NOTICE • WARNING • ERROR • LOG • FATAL • PANIC 默认值：WARNING。	指定某一级别后，写到日志的信息包含所有更高级别会输出的信息。级别越高，服务器日志的信息越少。

参数名称	描述	取值范围	备注
log_min_error_statement	配置写到服务器日志中错误SQL语句的级别。	• DEBUG5 • DEBUG4 • DEBUG3 • DEBUG2 • DEBUG1 • INFO • NOTICE • WARNING • ERROR • FATAL • PANIC 缺省值：ERROR。	所有导致一个特定级别（或者更高级别）错误的SQL语句都将记录在服务器日志中。 只有系统管理员可以修改该参数。
log_min_duration_statement	配置语句执行持续的最短时间。如果某个语句的持续时间大于或者等于设置的毫秒数，则会在日志中记录该语句及其持续时间。打开这个选项可以方便地跟踪需要优化的查询。	INT类型。 默认值：30min。 单位：毫秒。	设置为-1表示关闭这个功能。 只有系统管理员可以修改该参数。
log_connections/ log_disconnections	配置是否在每次会话连接或结束时向服务器日志里打印一条信息。	• on：每次会话连接或结束时向日志里打印一条信息。 • off：每次会话连接或结束时不向日志里打印信息。 默认值：off。	-
log_duration	配置是否记录每个已完成语句的持续时间。	• on：记录每个已完成语句的持续时间。 • off：不记录已完成语句的持续时间。 默认值：off。	只有系统管理员可以修改该参数。

参数名称	描述	取值范围	备注
log_statement	配置日志中记录哪些SQL语句。	• none：不记录任何SQL语句。 • ddl：记录数据定义语句。 • mod：记录数据定义语句和数据操作语句。 • all ：记录所有语句。 默认值： none。	只有系统管理员可以修改该参数。
log_hostname	配置是否记录主机名。	• on：记录主机名。 • off：不记录主机名。 默认值：off。	缺省时，连接日志只记录所连接主机的IP地址。打开这个选项会同时记录主机名。 该参数同时影响查看审计结果、**GS_SESSION_MEMORY_DETAIL**、**PG_STAT_ACTIVITY**和GUC参数log_line_prefix。

上表有关参数级别的说明请参见**表5-103**。

表 5-103 日志级别参数说明

级别	说明
DEBUG[1-5]	提供开发人员使用的信息。5级为最高级别，依次类推，1级为最低级别。
INFO	提供用户隐含要求的信息。如在VACUUM VERBOSE过程中的信息。
NOTICE	提供可能对用户有用的信息。如长标识符的截断，作为主键一部分创建的索引。
WARNING	提供给用户的警告。如在事务块范围之外的COMMIT。
ERROR	报告导致当前命令退出的错误。
LOG	报告一些管理员感兴趣的信息。如检查点活跃性。
FATAL	报告导致当前会话终止的原因。
PANIC	报告导致所有会话退出的原因。

6 SQL 调优指南

SQL调优的唯一目的是“资源利用最大化”，即CPU、内存、磁盘IO三种资源利用最大化。所有调优手段都是围绕资源使用开展的。所谓资源利用最大化是指SQL语句尽量高效，节省资源开销，以最小的代价实现最大的效益。比如做典型点查询的时候，可以用seqscan+filter（即读取每一条元组和点查询条件进行匹配）实现，也可以通过indexscan实现，显然indexscan可以以更小的代价实现相同的效果。

根据硬件资源和客户的业务特征确定合理的数据库部署方案和表定义是数据库在多数情况下满足性能要求的基础。下文的调优说明假设您已根据“软件安装”指引在安装过程中按照合理的数据库方案完成了安装，且已经根据“开发设计建议”的指引进行了数据库设计。

6.1 Query 执行流程

SQL引擎从接受SQL语句到执行SQL语句需要经历的步骤如图6-1和表6-1所示。其中，红色字体部分为DBA可以介入实施调优的环节。

图 6-1 SQL 引擎执行查询类 SQL 语句的流程

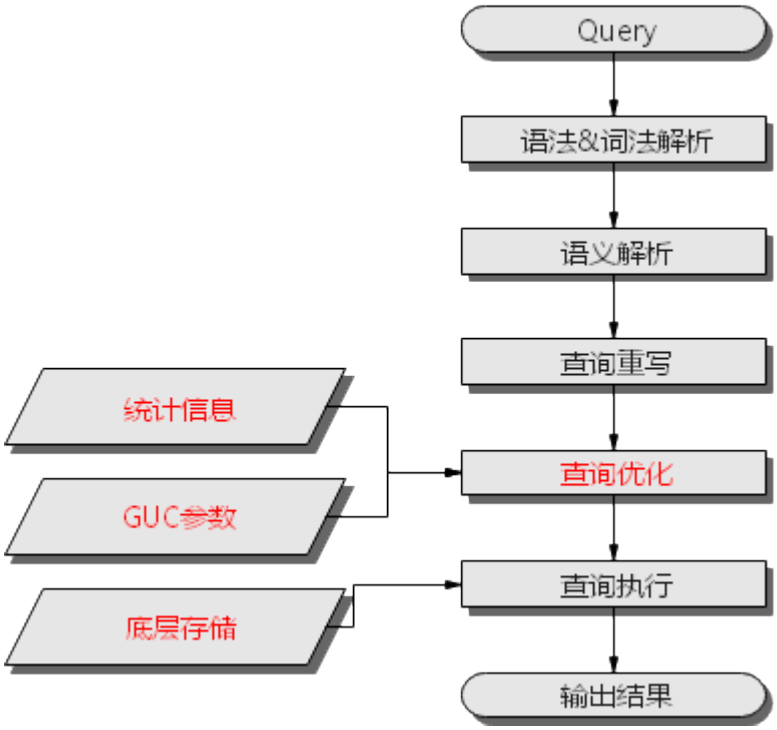


表 6-1 SQL 引擎执行查询类 SQL 语句的步骤说明

步骤	说明
1、语法&词法解析	按照约定的SQL语句规则，把输入的SQL语句从字符串转化为格式化结构(Stmt)。
2、语义解析	将“语法&词法解析”输出的格式化结构转化为数据库可以识别的对象。
3、查询重写	根据规则把“语义解析”的输出等价转化为执行上更为优化的结构。
4、查询优化	根据“查询重写”的输出和数据库内部的统计信息规划SQL语句具体的执行方式，也就是执行计划。统计信息和GUC参数对查询优化（执行计划）的影响，请参见**调优手段之统计信息**和**调优手段之GUC参数**。
5、查询执行	根据“查询优化”规划的执行路径执行SQL查询语句。底层存储方式的选择合理性，将影响查询执行效率。

调优手段之统计信息

GaussDB优化器是典型的基于代价的优化 (Cost-Based Optimization，简称CBO)。在这种优化器模型下，数据库根据表的元组数、字段宽度、NULL记录比率、distinct值、MCV值、HB值等表的特征值，以及一定的代价计算模型，计算出每一个执行步骤的不同执行方式的输出元组数和执行代价(cost)，进而选出整体执行代价最小/首元组返回代价最小的执行方式进行执行。这些特征值就是统计信息。从上面描述可以看出统计信息是查询优化的核心输入，准确的统计信息将帮助优化器选择最合适的查询规

划，一般来说通过analyze语法收集整个表或者表的若干个字段的统计信息，周期性地运行ANALYZE，或者在对表的大部分内容做了更改之后马上运行它是个好习惯。

说明

DDL可能会导致统计信息发生变化，进而导致计划跳变。当表上做了DDL操作后，应注意统计信息是否需要重新收集。

调优手段之 GUC 参数

查询优化的主要目的是为查询语句选择高效的执行方式。

如下SQL语句:

```
select count(1)
from customer inner join store_sales on (ss_customer_sk = c_customer_sk);
```

在执行customer inner join store_sales的时候，GaussDB支持Nested Loop、Merge Join和Hash Join三种不同的Join方式。优化器会根据表customer和表store_sales的统计信息估算结果集的大小以及每种Join方式的执行代价，然后对比选出执行代价最小的执行计划。

正如前面所说，执行代价计算都是基于一定的模型和统计信息进行估算，当因为某些原因代价估算不能反映真实的cost的时候，就需要通过GUC参数设置的方式让执行计划倾向更优规划。例如：random_page_cost参数表示优化器计算一次非顺序抓取磁盘页面的开销，该参数默认值为4。当机器磁盘随机读取的速度较快时，比如SSD设备，可以将该参数的值适当调小，更改后，索引扫描的代价降低，生成计划时更倾向于选择索引扫描的方式。

调优手段之 SQL 重写

除了上述干预SQL引擎所生成执行计划的执行性能外，根据数据库的SQL执行机制以及大量的实践发现，有些场景下，在保证客户业务SQL逻辑的前提下，通过一定规则由DBA重写SQL语句，可以大幅度地提升SQL语句的性能。

这种调优场景对DBA的要求比较高，需要对客户业务有足够的了解，同时也需要扎实的SQL语句基本功，后续会介绍几个常见的SQL改写场景。

6.2 SQL 执行计划介绍

6.2.1 SQL 执行计划概述

SQL执行计划是一个节点树，显示GaussDB执行一条SQL语句时执行的详细步骤。每一个步骤为一个数据库运算符。

使用EXPLAIN命令可以查看优化器为每个查询生成的具体执行计划。EXPLAIN给每个执行节点都输出一行，显示基本的节点类型和优化器为执行这个节点预计的开销值。如下所示。

```
gaussdb=#  explain select * from t1,t2 where t1.c1 = t2.c2;
                          QUERY PLAN
------------------------------------------------------------------------
 Hash Join  (cost=23.73..341.30 rows=16217 width=180)
   Hash Cond: (t1.c1 = t2.c2)
   ->  Seq Scan on t1  (cost=0.00..122.17 rows=5317 width=76)
   ->  Hash  (cost=16.10..16.10 rows=610 width=104)
```



```
      ->  Seq Scan on t2  (cost=0.00..16.10 rows=610 width=104)
(5 rows)
```

- 最底层节点是表扫描节点，它扫描表并返回原始数据行。不同的表访问模式有不同的扫描节点类型：顺序扫描、索引扫描等。最底层节点的扫描对象也可能是非表行数据（不是直接从表中读取的数据），如VALUES子句和返回行集的函数，它们有自己的扫描节点类型。
- 如果查询需要连接、聚集、排序、或者对原始行做其它操作，那么就会在扫描节点之上添加其它节点。 并且这些操作通常都有多种方法，因此在这些位置也有可能出现不同的执行节点类型。
- 第一行(最上层节点)是执行计划总执行开销的预计。这个数值就是优化器试图最小化的数值。

执行计划显示格式

GaussDB对执行计划提供了normal、pretty、summary、run四种显示格式：

- normal：代表使用默认的打印格式。
- pretty：代表使用GaussDB改进后的新显示格式。新的格式层次清晰，计划包含了plan node id，性能分析简单直接。
- summary：是在pretty的基础上增加了对打印信息的分析。
- run：在summary的基础上，将统计的信息输出到csv格式的文件中，以便于进一步分析。

pretty格式执行计划示例：

```
gaussdb=#  explain select * from t1,t2 where t1.c1=t2.c2;
 id |       operation         | E-rows | E-width |     E-costs
----+-------------------------+--------+---------+-----------------
  1 | ->  Hash Join (2,3)     |  23091 |      16 | 58.353..355.674
  2 |    ->  Seq Scan on t1   |   2149 |       8 | 0.000..31.490
  3 |    ->  Hash             |   2149 |       8 | 31.490..31.490
  4 |       ->  Seq Scan on t2 |   2149 |       8 | 0.000..31.490
(4 rows)

 Predicate Information (identified by plan id)
-----------------------------------------------
   1 --Hash Join (2,3)
       Hash Cond: (t1.c1 = t2.c2)
(2 rows)
```

通过设置GUC参数explain_perf_mode，可以显示不同格式的执行计划。下文的用例默认显示pretty格式。

执行计划显示信息

除了设置不同的执行计划显示格式外，还可以通过不同的EXPLAIN用法，显示不同详细程度的执行计划信息。常见有如下几种，关于更多用法请参见**EXPLAIN**语法说明。

- EXPLAIN *statement*： 只生成执行计划，不实际执行。其中statement代表SQL语句。
- EXPLAIN ANALYZE *statement*：生成执行计划，进行执行，并显示执行的概要信息。显示中加入了实际的运行时间统计，包括在每个规划节点内部花掉的总时间(以毫秒计)和它实际返回的行数。
- EXPLAIN PERFORMANCE *statement*：生成执行计划，进行执行，并显示执行期间的全部信息。

为了测量运行时在执行计划中每个节点的开销，EXPLAIN ANALYZE或EXPLAIN PERFORMANCE会在当前查询执行上增加性能分析的开销。在一个查询上运行EXPLAIN ANALYZE或EXPLAIN PERFORMANCE有时会比普通查询明显花费更多的时间。超出的时间多少取决于查询本身复杂程度和使用的平台。

因此，当定位SQL运行慢问题时，如果SQL长时间运行未结束，建议通过EXPLAIN命令查看执行计划，进行初步定位。如果SQL可以运行出来，则推荐使用EXPLAIN ANALYZE或EXPLAIN PERFORMANCE查看执行计划及其实际的运行信息，以便更准确地定位问题原因。

6.2.2 执行计划详解

6.2.2.1 执行计划

以如下SQL语句为例：

```
SELECT * FROM t1, t2 WHERE t1.c1 = t2.c2;
```

执行EXPLAIN的输出为：

```
gaussdb=#  EXPLAIN SELECT * FROM t1,t2 WHERE t1.c1 = t2.c2;
                           QUERY PLAN
-----------------------------------------------------------------------
 Hash Join  (cost=23.73..341.30 rows=16217 width=180)
   Hash Cond: (t1.c1 = t2.c2)
   ->  Seq Scan on t1  (cost=0.00..122.17 rows=5317 width=76)
   ->  Hash  (cost=16.10..16.10 rows=610 width=104)
         ->  Seq Scan on t2  (cost=0.00..16.10 rows=610 width=104)
(5 rows)
```

执行计划层级解读（纵向）：

1. 第一层：Seq Scan on t2

   表扫描算子，用Seq Scan的方式扫描表t2。这一层的作用是把表t2的数据从buffer或者磁盘上读上来输送给上层节点参与计算。

2. 第二层：Hash

   Hash算子，作用是把下层计算输送上来的算子计算hash值，为后续hash join操作做数据准备。

3. 第三层：Seq Scan on t1

   表扫描算子，用Seq Scan的方式扫描表t1。这一层的作用是把表t1的数据从buffer或者磁盘上读上来输送给上层节点参与hash join计算。

4. 第四层：Hash Join

   join算子，主要作用是将t1表和t2表的数据通过hash join的方式连接，并输出结果数据。

6.2.2.2 执行信息

以如下SQL语句在pretty模式下的执行结果为例：

```
gaussdb=# select sum(t2.c1) from t1,t2 where t1.c1=t2.c2 group by t1.c2;
```

执行EXPLAIN PERFORMANCE输出为：

```
gaussdb=# explain performance select sum(t2.c1) from t1,t2 where t1.c1=t2.c2 group by t1.c2;
 id |            operation             | A-time | A-rows | E-rows | E-distinct | Peak Memory | A-width | E-width |
E-costs
----+----------------------------------+--------+--------+--------+------------+-------------+---------+---------
```

```
+------------------
  1 | ->  HashAggregate                | 0.574 |      0 |    200 |           | 29KB       |        |       8 |
396.113..398.113
  2 |    ->  Hash Join (3,4)           | 0.358 |      0 |  18915 | 200, 200  | 12KB       |        |       8 |
53.763..301.538
  3 |       ->  Seq Scan on public.t1  | 0.037 |      1 |   1945 |           | 22KB       |        |       8 | 0.000..29.450
  4 |       ->  Hash                   | 0.038 |      0 |   1945 |           | 264KB      |        |       8 | 29.450..29.450
  5 |         ->  Seq Scan on public.t2 | 0.029 |     30 |   1945 |           | 22KB       |        |       8 | 0.000..29.450
(5 rows)

 Predicate Information (identified by plan id)
-------------------------------------------------
  2 --Hash Join (3,4)
        Hash Cond: (t1.c1 = t2.c2)
(2 rows)

    Memory Information (identified by plan id)
-------------------------------------------------------
  1 --HashAggregate
        Peak Memory: 29KB, Estimate Memory: 64MB
  2 --Hash Join (3,4)
        Peak Memory: 12KB, Estimate Memory: 64MB
  3 --Seq Scan on public.t1
        Peak Memory: 22KB, Estimate Memory: 64MB
  4 --Hash
        Peak Memory: 264KB
 Buckets: 32768  Batches: 1  Memory Usage: 0kB
  5 --Seq Scan on public.t2
        Peak Memory: 22KB, Estimate Memory: 64MB
(11 rows)

 Targetlist Information (identified by plan id)
-------------------------------------------------
  1 --HashAggregate
        Output: sum(t2.c1), t1.c2
        Group By Key: t1.c2
  2 --Hash Join (3,4)
        Output: t1.c2, t2.c1
  3 --Seq Scan on public.t1
        Output: t1.c1, t1.c2, t1.c3
  4 --Hash
        Output: t2.c1, t2.c2
  5 --Seq Scan on public.t2
        Output: t2.c1, t2.c2
(11 rows)

                              Datanode Information (identified by plan id)
-----------------------------------------------------------------------------------------------------------------
  1 --HashAggregate
        (actual time=0.574..0.574 rows=0 loops=1)
        (Buffers: shared hit=2)
        (CPU: ex c/r=0, ex row=0, ex cyc=527797, inc cyc=8385141377087373)
  2 --Hash Join (3,4)
        (actual time=0.358..0.358 rows=0 loops=1)
        (Buffers: shared hit=2)
        (CPU: ex c/r=-8385141375712241, ex row=1, ex cyc=-8385141375712241, inc cyc=8385141376559576)
  3 --Seq Scan on public.t1
        (actual time=0.037..0.037 rows=1 loops=1)
        (Buffers: shared hit=1)
        (CPU: ex c/r=8385141375728512, ex row=1, ex cyc=8385141375728512, inc cyc=8385141375728512)
  4 --Hash
        (actual time=0.038..0.038 rows=0 loops=1)
        (Buffers: shared hit=1)
        (CPU: ex c/r=0, ex row=0, ex cyc=-251554241295571040, inc cyc=8385141376543305)
  5 --Seq Scan on public.t2
        (actual time=0.019..0.029 rows=30 loops=1)
        (Buffers: shared hit=1)
        (CPU: ex c/r=8664646089070478, ex row=30, ex cyc=259939382672114336, inc
cyc=259939382672114336)
```

```
(20 rows)

      ====== Query Summary =====
----------------------------------------
 Datanode executor start time: 0.180 ms
 Datanode executor run time: 0.590 ms
 Datanode executor end time: 0.051 ms
 Planner runtime: 0.366 ms
 Query Id: 844424930141239
 Total runtime: 0.866 ms
(6 rows)
```

上述示例中显示执行信息分为以下6个部分：

1. 以表格的形式将计划显示出来，包含有11个字段，分别是：id、operation、A-time、A-rows、E-rows、E-distinct、Peak Memory、E-memory、A-width、E-width和E-costs。其中计划类字段（id、operation以及E开头字段）的含义与执行EXPLAIN时的含义一致，详见**执行计划**小节中的说明。A-time、A-rows、E-distinct、Peak Memory、A-width的含义说明如下：
   - A-time：当前算子执行完成时间。
   - A-rows：表示当前算子的实际输出元组数。
   - E-distinct：表示hashjoin算子的distinct估计值。
   - Peak Memory：此算子在执行时使用的内存峰值。
   - A-width：表示当前算子每行元组的实际宽度，仅对于重内存使用算子会显示，包括：(Vec)HashJoin、(Vec)HashAgg、(Vec) HashSetOp、(Vec)Sort、(Vec)Materialize算子等，其中(Vec)HashJoin计算的宽度是其右子树算子的宽度，会显示在其右子树上。
2. Predicate Information (identified by plan id)：

   这一部分主要显示的是静态信息，即在整个计划执行过程中不会变的信息，主要是一些join条件和一些filter信息。
3. Memory Information (identified by plan id)：

   这一部分显示的是整个计划中会将内存的使用情况打印出来的算子的内存使用信息，主要是Hash、Sort算子，包括算子峰值内存（peak memory），控制内存（control memory），估算内存使用（operator memory），执行时实际宽度（width），内存使用自动扩展次数（auto spread num），是否提前下盘（early spilled），以及下盘信息，包括重复下盘次数（spill Time(s)），内外表下盘分区数（inner/outer partition spill num），下盘文件数（temp file num），下盘数据量及最小和最大分区的下盘数据量（written disk IO [min, max] ）。
4. Targetlist Information (identified by plan id)：

   这一部分显示的是每一个算子输出的目标列。
5. DataNode Information (identified by plan id)：

   这一部分会将各个算子的执行时间、CPU、buffer的使用情况全部打印出来。
6. ====== Query Summary =====：

   这一部分主要打印总的执行时间和网络流量，包括了初始化和结束阶段的最大最小执行时间，以及当前语句执行时系统可用内存、语句估算内存等信息。

6.2.3 算子详解

6.2.3.1 关键字概述

执行计划中的主要关键字概述：

1. 表访问方式
   - Seq Scan

     全表顺序扫描。
   - Index Scan

     优化器决定使用两步的规划：最底层的规划节点访问一个索引，找出匹配索引条件的行的位置，然后上层规划节点真实地从表中抓取出那些行。独立地抓取数据行比顺序地读取它们的开销高很多，但是因为并非所有表的页面都被访问了，这么做实际上仍然比一次顺序扫描开销要少。使用两层规划的原因是，上层规划节点在读取索引标识出来的行位置之前，会先将它们按照物理位置排序，这样可以最小化独立抓取的开销。

     如果在WHERE里面使用的好几个字段上都有索引，那么优化器可能会使用索引的AND或OR的组合。但是这么做要求访问两个索引，因此与只使用一个索引，而把另外一个条件只当作过滤器相比，这个方法未必是更优。

     索引扫描可以分为以下几类，它们之间的差异在于索引的排序机制。

     - Bitmap Index Scan

       使用位图索引抓取数据页。
     - Index Scan using index_name

       使用简单索引搜索，该方式按照索引键的顺序在索引表中抓取数据。该方式最常用于在大数据量表中只抓取少量数据的情况，或者通过ORDER BY条件匹配索引顺序的查询，以减少排序时间。
     - Index-Only Scan

       当需要的所有信息都包含在索引中时，仅索引扫描便可获取所有数据，不需要引用表。
   - Bitmap Heap Scan

     从其他操作创建的位图中读取页面，过滤掉不符合条件的行。位图堆扫描可避免随机I/O，加快读取速度。
   - TID Scan

     通过TupleID扫描表。
   - Index Ctid Scan

     通过Ctid上的索引对表进行扫描。
   - CTE Scan

     CTE对子查询的操作进行评估并将查询结果临时存储，相当于一个临时表。CTE Scan算子对该临时表进行扫描。
   - Foreign Scan

     从远程数据源读取数据。
   - Function Scan

     获取函数返回的结果集，将它们作为从表中读取的行并返回。
   - Sample Scan

     查询并返回采样数据。
   - Subquery Scan

     读取子查询的结果。
   - Values Scan

作为VALUES命令的一部分读取常量。

- WorkTable Scan

  工作表扫描。在操作中间阶段读取，通常是使用WITH RECURSIVE声明的递归操作。

2. 表连接方式
   - Nested Loop

     嵌套循环，适用于被连接的数据子集较小的查询。在嵌套循环中，外表驱动内表，外表返回的每一行都要在内表中检索找到它匹配的行，因此整个查询返回的结果集不能太大（不能大于10000），要把返回子集较小的表作为外表，而且在内表的连接字段上建议要有索引。

   - (Sonic) Hash Join

     哈希连接，适用于数据量大的表的连接方式。优化器使用两个表中较小的表，利用连接键在内存中建立hash表，然后扫描较大的表并探测散列，找到与散列匹配的行。Sonic和非Sonic的Hash Join的区别在于所使用hash表结构不同，不影响执行的结果集。

   - Merge Join

     归并连接，通常情况下执行性能差于哈希连接。如果源数据已经被排序过，在执行归并连接时，并不需要再排序，此时归并连接的性能优于哈希连接。

3. 运算符
   - sort

     对结果集进行排序。

   - filter

     EXPLAIN输出显示WHERE子句当作一个"filter"条件附属于顺序扫描计划节点。这意味着规划节点为它扫描的每一行检查该条件，并且只输出符合条件的行。预计的输出行数降低了，因为有WHERE子句。不过，扫描仍将必须访问所有 10000 行，因此开销没有降低，实际上它还增加了一些（确切的说，通过10000 * cpu_operator_cost）以反映检查WHERE条件的额外CPU时间。

   - LIMIT

     LIMIT限定了执行结果的输出记录数。如果增加了LIMIT，那么不是所有的行都会被检索到。

   - Append

     合并子操作的结果。

   - Aggregate

     将查询行产生的结果进行组合。可以是GROUPBY、UNION、SELECT DISTINCT子句等函数的组合。

   - BitmapAnd

     位图的AND操作，通过该操作组成匹配更复杂条件的位图。

   - BitmapOr

     位图的OR操作，通过该操作组成匹配更复杂条件的位图。

   - Gather

     将并行线程的数据汇总。

   - Group

     对行进行分组，以进行GROUP BY操作。

   - GroupAggregate

聚合GROUP BY操作的预排序行。

- Hash

  对查询行进行散列操作，以供父查询使用。通常用于执行JOIN操作。

- HashAggregate

  使用哈希表聚合GROUP BY的结果行。

- Merge Append

  以保留排序顺序的方式对子查询结果进行组合，可用于组合表分区中已排序的行。

- ProjectSet

  对返回的结果集执行函数。

- Recursive Union

  对递归函数的所有步骤进行并集操作。

- SetOp

  集合运算，如INTERSECT或EXCEPT。

- Unique

  从有序的结果集中删除重复项。

- HashSetOp

  一种用于 INTERSECT 或 EXCEPT 等集合操作的策略，它使用 Append 来避免预排序的输入。

- LockRows

  锁定有问题的行以阻止其他查询写入，但允许读。

- Materialize

  将子查询的结果存储在内存里，以方便父查询快速访问获取。

- Result

  在不进行扫描的情况下返回一个值。

- WindowAgg

  窗口聚合函数，一般由OVER语句触发。

- Merge

  归并操作。

- StartWith Operator

  层次查询算子，用于执行递归查询操作。

- Rownum

  对查询结果的行编号进行条件过滤。通常出现在rownum子句里。

- Index Cond

  索引扫描条件。

- Unpivot

  转置算子。

4. 分区剪枝相关信息

   - Iterations

     分区迭代算子对一级分区的迭代次数。如果显示PART则为动态剪枝场景。

     例如：Iterations: 4表示迭代算子需要遍历4个一级分区。Iterations: PART表示遍历一级分区个数需要由分区键上的参数条件决定。

- Selected Partitions

  一级分区剪枝的结果，m..n表示m到n号分区被剪枝选中，多个不连续的分区由逗号连接。

  例如：Selected Partitions: 2..4,7 表示2、3、4、7四个分区被选中。
- Sub Iterations

  分区迭代算子对二级分区的迭代次数。如果显示PART则为动态剪枝场景。

  例如：Sub Iterations: 4表示迭代算子需要遍历4个二级分区。Iterations: PART表示遍历二级分区个数需要由分区键上的参数条件决定。
- Selected Subpartitions

  二级分区被剪枝的结果，由一级分区序号:二级分区序号。

  例如：Selected Subpartitions: 2:1 3:2 表示第二个一级分区的1号二级分区和第三个一级分区的2号二级分区被选中。Selected Subpartitions: ALL表示所有二级分区均被选中。

5. 其他关键字
   - Partitioned

     对具体分区的操作。
   - Partition Iterator

     分区迭代器，通常代表子查询是对分区的操作。
   - InitPlan

     非相关子计划。

6.2.3.2 表访问方式

6.2.3.2.1 Seq Scan

算子说明

Seq Scan算子是所有扫描算子中具有普适性的一种，这个算子本质上的原理为对表按某个方向（前向/后向）进行顺序扫描，然后返回符合筛选条件的所有行。

典型场景

- 表无索引，需要对表进行扫描操作。
- 表有索引，但需要对表大部分数据进行扫描操作。

示例

示例1：表无索引，需要对表进行扫描操作。

```
-- 数据准备。
gaussdb=# CREATE TABLE t1 (c1 number, c2 number, c3 number);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(generate_series(1,100), 2, 3);
INSERT 0 100
-- 执行结果。
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE c1 = 2;
                        QUERY PLAN
-----------------------------------------------------
 Seq Scan on t1  (cost=0.00..18.10 rows=3 width=96)
   Filter: (c1 = 2::numeric)
(2 rows)
```


上述示例中，Seq Scan算子输出信息如表6-2所示。

表 6-2 Seq Scan 算子输出信息

信息名称	含义
Seq Scan	算子的名称。
Filter	该算子的过滤谓词，示例中的过滤条件为c1列的值等于2。在查询执行时，满足这些条件的行会被包含在最终的结果集中。

示例2：表有索引，但需要对表大部分数据进行扫描操作。

```
-- 数据准备。
gaussdb=# CREATE TABLE t1(c1 number, c2 number, c3 number);
CREATE TABLE
gaussdb=# CREATE INDEX idx_c1 on t1(c1);
CREATE INDEX
gaussdb=# INSERT INTO t1 VALUES(generate_series(1, 100000), 2, 3);
INSERT 0 100000
-- 执行结果。
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE c1 <= 80000;
                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on t1  (cost=0.00..783.86 rows=9356 width=96)
   Filter: (c1 <= 80000::numeric)
(2 rows)
```

上述示例中，Seq Scan算子输出信息如表6-3所示。

表 6-3 Seq Scan 算子输出信息

信息名称	含义
Seq Scan	算子的名称。
Filter	该算子的过滤谓词，示例中的过滤条件为c1列的值小于等于80000。在查询执行时，满足这些条件的行会被包含在最终的结果集中。

6.2.3.2.2 Index Scan

算子说明

在索引扫描中，数据库使用语句指定的索引列，通过遍历索引树来检索行。数据库为一个值扫描索引时，发生 n 次 I/O 就能找到其要查找的值，其中 n 即 B-tree索引的高度。Index Scan通常用于检索表数据，数据库以轮流方式先读取索引块，找到对应的索引键值，然后通过索引键对应的tid去读取相应的表元组。在数据量大，但是查询结果集较小的场景下，Index Scan的效率往往高于Seq Scan。

典型场景

- 查询某个表中的特定行：当查询语句中包含WHERE子句时，如果WHERE子句中的条件可以通过索引列进行匹配，那么GaussDB就可能会使用Index Scan来查找符合条件的行。
- 排序：当查询语句中包含ORDER BY子句时，如果ORDER BY子句中的列可以通过索引进行排序，那么GaussDB就可能会使用Index Scan来进行排序操作。
- 聚合：当查询语句中包含GROUP BY子句时，如果GROUP BY子句中的列可以通过索引进行分组，那么GaussDB就可能会使用Index Scan来进行聚合操作。
- 连接：当查询语句中包含JOIN操作时，如果JOIN操作中的列可以通过索引进行匹配，那么GaussDB就可能会使用Index Scan来进行连接操作。

示例

示例1：WHERE子句中的条件可以通过索引列进行匹配。

```
-- 数据准备。
gaussdb=# CREATE TABLE test (c1 int, c2 int);
CREATE TABLE
gaussdb=# CREATE INDEX c1_idx ON test (c1);
CREATE INDEX
gaussdb=# INSERT INTO test SELECT generate_series(1, 1000000), random()::integer;
INSERT 0 1000000
gaussdb=# INSERT INTO test SELECT generate_series(1, 1000000), random()::integer;
INSERT 0 1000000
gaussdb=# INSERT INTO test SELECT generate_series(1, 1000000), random()::integer;
INSERT 0 1000000
gaussdb=# EXPLAIN SELECT /*+ indexscan(test c1_idx) */ * FROM test WHERE c1 > 990000;
                            QUERY PLAN
 [Bypass]
 Index Scan using c1_idx on test  (cost=0.00..344.17 rows=9767 width=8)
   Index Cond: (c1 > 990000)
(3 rows)
```

上述示例中，Index Scan算子输出信息如下所示。

信息名称	含义
Index Scan	算子的名称。
Index Cond	该算子的过滤谓词，示例中的过滤条件为c1列的值大于990000。在查询执行时，满足这些条件的行会被包含在最终的结果集中。

示例2：ORDER BY子句中的列可以通过索引进行排序。

```
-- 数据准备，同上。
gaussdb=# EXPLAIN SELECT /*+ indexscan(test c1_idx) */ * FROM test ORDER BY c1;
                                QUERY PLAN
---------------------------------------------------------------------------------
 [Bypass]
 Index Scan using c1_idx on test  (cost=0.00..3815878.82 rows=2451905 width=8)
(2 rows)
```

示例3：GROUP BY子句中的列可以通过索引进行分组。

```
-- 数据准备，同上。
gaussdb=#  EXPLAIN SELECT /*+ indexscan(test c1_idx) */ c1 FROM test GROUP BY c1;
```



```
                                  QUERY PLAN
-------------------------------------------------------------------------------------------
 Group  (cost=0.00..102640.59 rows=200 width=4)
   Group By Key: c1
   ->  Index Only Scan using c1_idx on test  (cost=0.00..96510.82 rows=2451905 width=4)
(3 rows)
```

6.2.3.2.3 Index Only Scan

算子说明

Index Only Scan是GaussDB中的一种查询优化技术，它可以通过只扫描索引而不需要访问表数据来提高查询性能。在执行查询时，如果查询条件只涉及到表的某个索引列，就可以使用Index Only Scan来优化查询。Index Only Scan会直接扫描索引，从而减少了I/O操作和CPU开销，提高了查询性能。

典型场景

只需要查询索引列的值，而不需要访问表中的其他列。例如，查询一个表中的某个列的最大值或最小值，或者查询一个列的不同值的数量。

示例

示例：目标列中仅含索引列。

```
-- 数据准备。
gaussdb=# CREATE TABLE test2 (a int, b int);
CREATE TABLE
gaussdb=# CREATE INDEX test2_idx ON test2 (a, b);
CREATE INDEX
-- 随机插入1000条数据。
gaussdb=# INSERT INTO test2 VALUES(generate_series(1, 1000), generate_series(1, 1000));
INSERT 0 1000
-- 执行结果
gaussdb=# EXPLAIN SELECT a, b FROM test2 WHERE a = 10 AND b = 20;
                              QUERY PLAN
-------------------------------------------------------------------------------
 [Bypass]
 Index Only Scan using test2_idx on test2  (cost=0.00..4.27 rows=1 width=8)
   Index Cond: ((a = 10) AND (b = 20))
(3 rows)
```

上述示例中，Index Only Scan算子输出信息如下所示。

信息名称	含义
Index Only Scan	算子的名称。
Index Cond	该算子的过滤谓词，示例中的过滤条件为a列的值等于10并且b的值等于20。在查询执行时，满足这些条件的行会被包含在最终的结果集中。

6.2.3.2.4 Bitmap

算子说明

1. Bitmap Index Scan，位图索引扫描，该算子的作用为用索引扫描的方式，将过滤出的符合条件的元组的tid形成一张位图并向上返回。多个表扫描出的多个结果位图之间可再做 BITMAPAND或者BITMAPOR操作，相比于Index Scan还需要对表与表之间筛选出来的元组进行比对，可减少过滤时间。
2. Bitmap Heap Scan，位图堆扫描，该算子的下层通常有Bitmap Index Scan，作用为通过扫描下层算子返回的位图（bitmap）来判断哪些元组符合条件并取出然后向上返回。当优化器通过代价估算选择采用Bitmap Index Scan的扫描方式，且需要返回符合条件的元组时，通常上层会配套使用Bitmap Heap Scan来通过扫描位图过滤选取元组并返回。
3. BitmapOr，位图或操作，该算子作用为将下层的bitmap进行或操作，返回操作后的位图。
4. BitmapAnd，位图与操作，该算子作用为将下层的bitmap进行与操作，返回操作后的位图。

使用Bitmap Index Scan和Bitmap Heap Scan的前提条件为：GUC参数enable_bitmapscan设置为on。

典型场景

多个表做关联，且在一个或多个表的关联列上有可用的索引，部分或全部表的表数据足够大。

示例

示例：多表连接带索引。

```
-- 数据准备，创建3个表，数据均为numeric类型，随机填充数据，为每个表的每个属性创建索引。
gaussdb=# SET enable_default_ustore_table = on;
SET
gaussdb=# CREATE TABLE t_btm_test_1(a numeric,b numeric,c numeric);
CREATE TABLE
gaussdb=# CREATE TABLE t_btm_test_2(a numeric,b numeric,c numeric);
CREATE TABLE
gaussdb=# CREATE TABLE t_btm_test_3(a numeric,b numeric,c numeric);
CREATE TABLE
gaussdb=# DECLARE
gaussdb-#     n1 numeric := 100;
gaussdb-#     n2 numeric := 0;
gaussdb-#     n3 numeric := 100;
gaussdb-# BEGIN
gaussdb$#     WHILE n1 > 0 LOOP
gaussdb$#         n2 := 0;
gaussdb$#         WHILE n2 < 100 LOOP
gaussdb$#             n3 := 100;
gaussdb$#             WHILE n3 > 0 LOOP
gaussdb$#                 INSERT INTO t_btm_test_1 VALUES(n1, n2, n3);
gaussdb$#                 INSERT INTO t_btm_test_2 VALUES(n1, n3, n2);
gaussdb$#                 INSERT INTO t_btm_test_3 VALUES(n2, n1, n3);
gaussdb$#                 n3 := n3 - 1;
gaussdb$#             END LOOP;
gaussdb$#             n2 := n2 + 1;
gaussdb$#         END LOOP;
gaussdb$#         n1 := n1 - 1;
gaussdb$#     END LOOP;
gaussdb$# END;
gaussdb$# /
```



```
ANONYMOUS BLOCK EXECUTE
gaussdb=# CREATE INDEX idx_btm_test_11 ON t_btm_test_1 USING UBTREE (a);
CREATE INDEX
gaussdb=# CREATE INDEX idx_btm_test_12 on t_btm_test_1 USING UBTREE (b);
CREATE INDEX
gaussdb=# CREATE INDEX idx_btm_test_13 on t_btm_test_1 USING UBTREE (c);
CREATE INDEX
gaussdb=# CREATE INDEX idx_btm_test_21 on t_btm_test_2 USING UBTREE(a);
CREATE INDEX
gaussdb=# CREATE INDEX idx_btm_test_22 on t_btm_test_2 USING UBTREE (b);
CREATE INDEX
gaussdb=# CREATE INDEX idx_btm_test_23 on t_btm_test_2 USING UBTREE (c);
CREATE INDEX
gaussdb=# CREATE INDEX idx_btm_test_31 on t_btm_test_3 USING UBTREE (a);
CREATE INDEX
gaussdb=# CREATE INDEX idx_btm_test_32 on t_btm_test_3 USING UBTREE (b);
CREATE INDEX
gaussdb=# CREATE INDEX idx_btm_test_33 on t_btm_test_3 USING UBTREE (c);
CREATE INDEX
-- 执行结果。
gaussdb=# SET enable_bitmapscan = on;
SET
gaussdb=#  EXPLAIN SELECT /*+ hashjoin(t_btm_test_1 t_btm_test_2 t_btm_test_3)
BitmapScan(t_btm_test_2 idx_btm_test_21 idx_btm_test_22) BitmapScan(t_btm_test_3 idx_btm_test_31 )*/ *
FROM t_btm_test_1 , t_btm_test_2 , t_btm_test_3  WHERE t_btm_test_1.a = t_btm_test_2.b AND
t_btm_test_2.a = t_btm_test_3.c AND t_btm_test_3.a = t_btm_test_1.c AND t_btm_test_1.a = 1 AND
((t_btm_test_2.a = 1 AND t_btm_test_3.a = 1) OR (t_btm_test_2.a = 3 AND t_btm_test_3.a = 3));
                                                                    QUERY
PLAN
------------------------------------------------------------------------------------------------------------------------------
--------------------------------------
 Hash Join  (cost=7091.42..7475.66 rows=869 width=42)
   Hash Cond: (t_btm_test_1.c = t_btm_test_3.a)
   ->  Index Scan using idx_btm_test_11 on t_btm_test_1  (cost=0.00..350.86 rows=9661 width=14)
         Index Cond: (a = 1::numeric)
   ->  Hash  (cost=7091.31..7091.31 rows=9 width=28)
         ->  Hash Join  (cost=1794.78..7091.31 rows=9 width=28)
               Hash Cond: (t_btm_test_3.c = t_btm_test_2.a)
               Join Filter: (((t_btm_test_2.a = 1::numeric) AND (t_btm_test_3.a = 1::numeric)) OR
((t_btm_test_2.a = 3::numeric) AND (t_btm_test_3.a = 3::numeric)))
               ->  Bitmap Heap Scan on t_btm_test_3  (cost=439.65..4824.65 rows=20296 width=14)
                     Recheck Cond: ((a = 1::numeric) OR (a = 3::numeric))
                     ->  BitmapOr  (cost=439.65..439.65 rows=20400 width=0)
                           ->  Bitmap Index Scan on idx_btm_test_31  (cost=0.00..224.50 rows=10700 width=0)
                                 Index Cond: (a = 1::numeric)
                           ->  Bitmap Index Scan on idx_btm_test_31  (cost=0.00..205.00 rows=9700 width=0)
                                 Index Cond: (a = 3::numeric)
               ->  Hash  (cost=1352.48..1352.48 rows=212 width=14)
                     ->  Bitmap Heap Scan on t_btm_test_2  (cost=657.66..1352.48 rows=212 width=14)
                           Recheck Cond: ((b = 1::numeric) AND ((a = 1::numeric) OR (a = 3::numeric)))
                           ->  BitmapAnd  (cost=657.66..657.66 rows=213 width=0)
                                 ->  Bitmap Index Scan on idx_btm_test_22  (cost=0.00..217.25 rows=10267 width=0)
                                       Index Cond: (b = 1::numeric)
                                 ->  BitmapOr  (cost=440.10..440.10 rows=20733 width=0)
                                       ->  Bitmap Index Scan on idx_btm_test_21  (cost=0.00..210.00 rows=9833
width=0)
                                             Index Cond: (a = 1::numeric)
                                       ->  Bitmap Index Scan on idx_btm_test_21  (cost=0.00..230.00 rows=10900
width=0)
                                             Index Cond: (a = 3::numeric)
(26 rows)
```

上述示例中，Bitmap Heap Scan算子输出信息如下所示。

信息名称	含义
Bitmap Heap Scan	算子的名称。

信息名称	含义
Recheck Cond	在执行索引扫描后，数据库重新检查过滤的谓词，示例中Recheck Cond: ((a = 1::numeric) OR (a = 3::numeric))的过滤条件为，a列的值等于1或者等于3。在查询执行时，满足这些条件的行会被包含在最终的结果集中。

上述示例中，Bitmap Index Scan算子输出信息如下所示。

信息名称	含义
Bitmap Index Scan	算子的名称。
Index Cond	该算子的过滤谓词，在查询执行时，满足这些条件的行会被包含在最终的结果集中。

上述示例中，BitmapOr & BitmapAnd算子输出信息如下所示。

信息名称	含义
BitmapOr	算子的名称，代表将下层返回的bitmap进行or运算，通过该操作组成匹配更复杂条件的位图。
BitmapAnd	算子的名称，代表将下层返回的bitmap进行and运算，通过该操作组成匹配更复杂条件的位图。

6.2.3.2.5 Tid Scan

算子说明

行号扫描，该算子主要利用行号（ctid）过滤元组并返回。ASTORE场景下，数据按行存储在HEAP PAGE中，在B-tree 索引中除了存储字段的value，还会存储对应的行号，因此GaussDB中支持通过行号进行快速检索。

行号的写法为：

```
（page_number，item_number）     -- page_number从0开始编号，item_number从1开始编号
```

通过给定的ctid值找到在磁盘或缓存中对应的元组。

典型场景

前提条件：GUC参数enable_tidscan设置为on。

对于支持Tid Scan的表，给定合法有效的行号，可对数据进行快速检索。当前GaussDB会选择Tid Scan的场景只有WHERE条件为"="的查询，不支持in的条件查询。

示例

示例：查询条件中含有关于tid的筛选条件。

```
-- 数据准备，创建包含3个属性的表，随机填充数据。
gaussdb=# CREATE TABLE t_tid_test(a numeric,b numeric,c numeric);
CREATE TABLE
gaussdb=# DECLARE
gaussdb-#     n1 numeric := 10;
gaussdb-#     n2 numeric := 0;
gaussdb-#     n3 numeric := 100;
gaussdb-# BEGIN
gaussdb$#     WHILE n1 > 0 LOOP
gaussdb$#         n2 := 0;
gaussdb$#         WHILE n2 < 100 LOOP
gaussdb$#             n3 := 100;
gaussdb$#             WHILE n3 > 0 LOOP
gaussdb$#                 INSERT INTO t_tid_test VALUES(n1, n2, n3);
gaussdb$#                 n3 := n3 - 1;
gaussdb$#             END LOOP;
gaussdb$#             n2 := n2 + 1;
gaussdb$#         END LOOP;
gaussdb$#         n1 := n1 - 1;
gaussdb$#     END LOOP;
gaussdb$# END;
gaussdb$# /
ANONYMOUS BLOCK EXECUTE
-- 执行结果
gaussdb=# SET enable_tidscan = on;
SET
gaussdb=# EXPLAIN ANALYZE SELECT * FROM t_tid_test WHERE ctid = '(0,10)'::tid;
                                          QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Tid Scan on t_tid_test  (cost=0.00..4.01 rows=1 width=14) (actual time=0.016..0.016 rows=1 loops=1)
   TID Cond: (ctid = '(0,10)'::tid)
 Total runtime: 0.077 ms
(3 rows)
```

上述示例中，Tid Scan算子输出信息如下所示。

信息名称	含义
Tid Scan	算子的名称。
TID Cond	该算子的过滤谓词，在查询执行时，满足这些条件的行会被包含在最终的结果集中。

6.2.3.2.6 Cte Scan

算子说明

CTE（Common Table Expression）是一种临时表达式，Cte Scan用于扫描CTE表达式生成的临时表。

在GaussDB中，可以通过使用with关键字来指定一个或多个CTE，然后在后续的查询中多次使用。

典型场景

当一个查询结果集需要在后续的查询中多次使用的时候，可以考虑使用CTE来避免多次计算。

示例

示例：带WITH语句的无法改写为子查询的CTE。

```
-- 数据准备。
gaussdb=# CREATE TABLE employees(deptid integer, salary number);
CREATE TABLE
gaussdb=# CREATE TABLE managers(deptid integer);
CREATE TABLE
-- 执行结果。
gaussdb=# EXPLAIN WITH table_b AS (SELECT deptid,sum(salary) dept_salary FROM employees GROUP BY
deptid)
gaussdb-# SELECT m.deptid, b.dept_salary FROM managers m,table_b b,table_b c
gaussdb-# WHERE m.deptid = b.deptid AND  m.deptid = c.deptid;
                                QUERY PLAN
-------------------------------------------------------------------------------------
 Hash Join  (cost=46.32..113.37 rows=2402 width=36)
   Hash Cond: (m.deptid = b.deptid)
   CTE table_b
     ->  HashAggregate  (cost=28.57..30.57 rows=200 width=36)
           Group By Key: employees.deptid
           ->  Seq Scan on employees  (cost=0.00..22.38 rows=1238 width=36)
   ->  Seq Scan on managers m  (cost=0.00..34.02 rows=2402 width=4)
   ->  Hash  (cost=13.25..13.25 rows=200 width=40)
         ->  Hash Join  (cost=6.50..13.25 rows=200 width=40)
               Hash Cond: (b.deptid = c.deptid)
               ->  CTE Scan on table_b b  (cost=0.00..4.00 rows=200 width=36)
               ->  Hash  (cost=4.00..4.00 rows=200 width=4)
                     ->  CTE Scan on table_b c  (cost=0.00..4.00 rows=200 width=4)
(13 rows)
```

6.2.3.2.7 Foreign Scan

算子说明

在GaussDB中，Foreign Scan是一种用于访问外部数据源的扫描器。它可以将外部数据源中的数据作为关系型数据库中的表来处理，从而实现对外部数据源的查询和操作。在GaussDB中，Foreign Scan可以通过扩展API来实现。用户可以编写自己的扩展程序，以实现对外部数据源的访问和操作。同时，GaussDB还提供了一些常用的扩展程序，如FDW（Foreign Data Wrapper），可以用于访问其他关系型数据库中的数据。

典型场景

- 访问外部数据源：当需要访问外部数据源时，可以使用Foreign Scan来读取外部数据源中的数据。
- 数据集成：当需要将多个数据源中的数据进行集成时，可以使用Foreign Scan来将多个数据源中的数据作为关系型数据库中的表来处理。
- 数据迁移：当需要将数据从一个数据源迁移到另一个数据源时，可以使用Foreign Scan来读取源数据源中的数据，并将其插入到目标数据源中。

示例

示例：访问外部数据源。


```
-- 数据准备。
gaussdb=# SET enable_extension = true;
gaussdb=# CREATE EXTENSION file_fdw;
gaussdb=# CREATE SERVER foo FOREIGN DATA WRAPPER file_fdw;
gaussdb=# CREATE FOREIGN TABLE ft (a int, b text) SERVER foo OPTIONS ("format" 'csv', "filename" '/
gaussdb/username/data.csv', "null" 'NULL');
-- 执行结果。
gaussdb=# EXPLAIN SELECT * FROM ft;
                        QUERY PLAN
------------------------------------------------------------
 Foreign Scan on ft  (cost=0.00..138.00 rows=1280 width=36)
   Foreign File: /gaussdb/username/data.csv
(2 rows)
```

上述示例中，Foreign Scan算子输出信息如下所示。

信息名称	含义
Foreign Scan	算子的名称。
Foreign File	外部数据源文件。

6.2.3.2.8 Function Scan

算子说明

Function Scan用于执行函数并从函数返回的结果集中获取元组。

典型场景

当查询需要调用一个函数并返回结果集时，Function Scan算子可以执行函数调用并从函数返回的结果集中获取元组。

示例

示例：调用系统函数。

```
gaussdb=# EXPLAIN SELECT * FROM generate_series(2,4);
                              QUERY PLAN
------------------------------------------------------------------------
 Function Scan on generate_series  (cost=0.00..10.00 rows=1000 width=4)
(1 row)
```

6.2.3.2.9 Sample Scan

算子说明

Sample Scan是一种扫描表的方式，它可以在查询过程中随机地从表中抽取一部分数据进行查询，而不是扫描整个表。Sample Scan配合TABLESAMPLE关键字使用，使用方式为[TABLESAMPLE sampling_method (argument [, ...]) [REPEATABLE (seed)]]。这种技术通常用于大型表或者需要随机抽样的查询场景，可以提高查询效率。

典型场景

当查询时使用tablesample语法，sampling_method使用system或bernoulli 。

示例

示例：采样查询

```
-- 数据准备
gaussdb=# create table t1 (c1 number, c2 number, c3 number);
CREATE TABLE
gaussdb=# insert into t1 values(generate_series(1,100), 2, 3);
INSERT 0 100
-- 执行结果
gaussdb=# explain select c1 from t1 tablesample system(10);
                        QUERY PLAN
-----------------------------------------------------
 Sample Scan on t1  (cost=0.00..272.00 rows=10000 width=6)
   Sampling: system (10::real)
(2 rows)
```

上述示例中，Sample Scan算子输出信息如下所示。

信息名称	含义
Sample Scan	算子的名称。
Sampling	算子采样的方式，示例中采用system 作为采样方法，采样的比例为10%

6.2.3.2.10 SubQuery Scan

算子说明

当执行一个包含子查询的语句时，如果优化器RBO没有对它进行优化，它会先执行子查询的查询计划树，然后将子查询的结果传递给上层查询。

典型场景

当语句中包含子查询的时候，会生成SubQuery Scan算子从子查询中获取元组。

示例

示例：查询中带无法下推的子查询。

```
-- 数据准备。
gaussdb=# CREATE TABLE t1(c1 number, c2 number, c3 number);
CREATE TABLE
gaussdb=# CREATE TABLE t2(c1 number, c2 number, c3 number);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(generate_series(1,10000), (random() * 10)::integer,(random() *
10)::integer );
INSERT 0 10000
gaussdb=# INSERT INTO t2 VALUES(generate_series(1,10000), (random() * 10)::integer,(random() *
10)::integer );
INSERT 0 10000
-- 执行结果。
gaussdb=#  EXPLAIN SELECT * FROM t1 WHERE c2 > (SELECT avg(c2) FROM t2 WHERE t2.c1 = t1.c1);
                              QUERY PLAN
------------------------------------------------------------------------------
 Hash Join  (cost=87.87..198.64 rows=886 width=96)
   Hash Cond: (t1.c1 = subquery."?column?")
   Join Filter: (t1.c2 > subquery.avg)
   ->  Seq Scan on t1  (cost=0.00..67.58 rows=2658 width=96)
```



```
  ->  Hash  (cost=85.37..85.37 rows=200 width=64)
        ->  Subquery Scan on subquery  (cost=80.87..85.37 rows=200 width=64)
              ->  HashAggregate  (cost=80.87..83.37 rows=200 width=64)
                    Group By Key: t2.c1
                    ->  Seq Scan on t2  (cost=0.00..67.58 rows=2658 width=64)
(9 rows)
```

6.2.3.2.11 Values Scan

算子说明

读取 VALUES 子句中指定的值列表，并将其作为一组虚拟行返回给查询计划器。这些虚拟行可以被其他扫描器或操作符使用，例如 HashJoin 或 MergeJoin。

典型场景

该算子提供了一种快速且简便的方法来指定一组值，无需从表中读取数据。这在测试或调试期间，或需要插入少量数据时非常有用。

示例

示例：插入语句中带VALUES。

```
-- 数据准备。
gaussdb=# CREATE TABLE test_b (c1 number);
CREATE TABLE
-- 执行结果。
gaussdb=# EXPLAIN INSERT INTO test_b(c1) VALUES ('1'),('2');
                          QUERY PLAN
---------------------------------------------------------------------
 Insert on test_b  (cost=0.00..0.03 rows=2 width=12)
   ->  Values Scan on "*VALUES*"  (cost=0.00..0.03 rows=2 width=12)
(2 rows)
```

6.2.3.2.12 WorkTable Scan

算子说明

WorkTable Scan是一种基于内存的查询优化技术，它可以将查询结果缓存在内存中，以提高查询性能。当GaussDB执行一个查询语句时，它会将查询结果存储在一个临时表中，然后使用WorkTable Scan来扫描这个临时表，以获取查询结果。WorkTable Scan的优点是可以减少磁盘I/O操作，提高查询性能。但是，它也有一些缺点，比如会消耗大量的内存空间，可能会导致内存溢出等问题。因此，在使用WorkTable Scan时，需要根据具体情况进行调整和优化。

典型场景

- 大表查询：当查询涉及到大表时，WorkTable Scan可以将查询结果分成多个工作单元，每个工作单元处理一部分数据，从而提高查询效率。
- 多表关联查询：当查询需要关联多个表时，WorkTable Scan可以将每个表的数据分成多个工作单元，每个工作单元处理一部分数据，从而减少关联操作的数据量，提高查询效率。
- 分区表查询：当查询涉及到分区表时，WorkTable Scan可以将每个分区的数据分成多个工作单元，每个工作单元处理一部分数据，从而提高查询效率。
- 大数据量统计查询：当查询需要对大量数据进行统计时，WorkTable Scan可以将数据分成多个工作单元，每个工作单元处理一部分数据，从而提高查询效率。

示例

示例：Recursive Union的表不带索引

```
-- 数据准备。
gaussdb=# CREATE TABLE t9(a int);
CREATE TABLE
gaussdb=# INSERT INTO t9 VALUES(1);
INSERT 0 1
-- 执行结果。
gaussdb=# EXPLAIN WITH RECURSIVE tt AS (
gaussdb=# SELECT a FROM t9
gaussdb=# UNION ALL
gaussdb=# SELECT a + 1 FROM tt WHERE a < 10)
gaussdb=# SELECT * FROM tt;
                              QUERY PLAN
------------------------------------------------------------------------------
CTE Scan on tt (cost=7288.14..8937.58 rows=82472 width=4)
  CTE tt
  -> Recursive Union (cost=0.00..7288.14 rows=82472 width=4)
    -> Seq Scan on t9 (cost=0.00..34.02 rows=2402 width=4)
    -> WorkTable Scan on tt (cost=0.00..560.47 rows=8007 width=4)
         Filter: (a < 10)
(6 rows)
```

上述示例中，WorkTable Scan算子输出信息如下所示。

信息名称	含义
WorkTable Scan	算子的名称。
Filter	算子的过滤条件，示例中为a小于10。

6.2.3.3 表连接方式

6.2.3.3.1 Nested Loop Join

算子说明

嵌套循环连接（Nested Loop Join）是最简单的连接方法，也是所有关系数据库系统中都会实现的连接操作。这种方法的基本思想是“把两个表中的数据两两比较，看是否满足连接条件”。

在GaussDB中，Nested Loop Join的工作原理是，对于外部表（Outer Table）中的每一行，扫描内部表（Inner Table），查找符合连接条件的行。这类似于两个嵌套的循环，外部循环遍历外部表，内部循环遍历内部表，因此得名 。

Nested Loop Join的时间复杂度是O(n*m), 其中n和m分别代表两个表的行数，如果内部表可以用索引来扫描，那么时间复杂度可以降低到O(nlogm)。

典型场景

- 当内部表或外部表（或者两者都）非常小的时候。
- 当内部表能够根据连接条件快速定位到满足条件的行时，例如内部表的连接字段已经建了索引。
- Nested Loop Join对连接的条件没有限制，任何连接条件Nested Loop Join都可以执行。

示例

示例：两个小表之间做Join。

```
-- 数据准备。
gaussdb=# CREATE TABLE employee(id int, deptid int);
CREATE TABLE
gaussdb=# INSERT INTO employee VALUES(1, 1), (2,1),(3,2),(4, 1), (5,2);
INSERT 0 5
gaussdb=# CREATE TABLE manager(id int, deptid int);
CREATE TABLE
gaussdb=# INSERT INTO manager VALUES(1,1), (2,2),(3,1),(4,2);
INSERT 0 4
-- 执行结果。
gaussdb=# EXPLAIN SELECT * FROM employee e JOIN manager m ON e.deptid < m.deptid;
                              QUERY PLAN
------------------------------------------------------------------------------
 Nested Loop  (cost=0.00..69341.37 rows=1539400 width=16)
   Join Filter: (e.deptid < m.deptid)
   ->  Seq Scan on employee e  (cost=0.00..31.49 rows=2149 width=8)
   ->  Materialize  (cost=0.00..42.23 rows=2149 width=8)
         ->  Seq Scan on manager m  (cost=0.00..31.49 rows=2149 width=8)
(5 rows)
```

上述示例中，Nested Loop Join输出信息如下所示。

信息名称	含义
Nested Loop	算子的名称。
Join Filter	算子join的连接谓词，示例中条件为e.deptid 小于m.deptid，在查询执行时，满足这些条件的行会被包含在最终的结果集中。

6.2.3.3.2 Hash Join

算子说明

哈希连接（Hash Join）是一种高效的连接方法，它依赖于哈希技术。在进行哈希连接时，GaussDB会先选取两个表中的一个（通常是小表），接下来根据连接条件，建立一个哈希表。哈希表的键是小表的连接字段，值是小表的其他字段。然后，对于大表中的每一行，计算连接字段的哈希值，并在哈希表中查找是否有匹配的行。

Hash Join的时间复杂度为O(n+m), 其中n和m分别代表两个表的行数。然而，如果内部表过大，以至于哈希表无法完全放入内存，则可能需要额外的磁盘I/O操作，这会导致性能降低。

典型场景

- 当两个表的行数差距很大，并且进行连接操作。
- 当两个表的连接字段是等值连接（比如使用=运算符）。

示例

示例：两个表的行数差距很大，并且进行连接操作。

```
-- 数据准备。
gaussdb=# CREATE TABLE t1(c1 number,c2 number,c3 number);
```



```
CREATE TABLE
gaussdb=# CREATE TABLE t2(c1 number,c2 number,c3 number);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(generate_series(1,100000), 2 ,3);
INSERT 0 100000
gaussdb=# INSERT INTO t2 VALUES(generate_series(1,100), 2, 3);
INSERT 0 100
-- 执行结果。
gaussdb=# EXPLAIN SELECT * FROM t1 JOIN t2 ON t1.c1 = t2.c2;
                          QUERY PLAN
------------------------------------------------------------------
 Hash Join  (cost=24.58..1823.14 rows=90944 width=192)
   Hash Cond: (t1.c1 = t2.c2)
   ->  Seq Scan on t1  (cost=0.00..713.69 rows=28069 width=96)
   ->  Hash  (cost=16.48..16.48 rows=648 width=96)
         ->  Seq Scan on t2  (cost=0.00..16.48 rows=648 width=96)
(5 rows)
```

上述示例中，Hash Join算子输出信息如下所示。

信息名称	含义
Hash Join	算子的名称。
Hash Cond	算子Hash join的连接谓词，示例中条件为t1.c1等于t2.c2，在查询执行时，满足这些条件的行会被包含在最终的结果集中。
Hash	内表创建hash table的算子。

6.2.3.3.3 Merge Join

算子说明

合并连接（Merge Join）是一种高效的连接方法，它依赖于排序操作。在进行合并连接时，GaussDB会对两个表的连接字段进行排序，然后同步扫描两个表，寻找匹配的行。 Merge Join的时间复杂度为O(n+m), 其中n和m分别代表两个表的行数。然而，如果需要排序操作，这个排序操作的时间复杂度可能会达到max(O(logn), O(logm))，这通常会比直接的Merge Join操作更加耗时。 在GaussDB中，优化器更倾向于选择Hash Join，即使需要连接的两张表已经经过排序。

典型场景

- 当两个表的大小接近时。
- 当两个表的连接字段已经被排序或者已经有序时，比如通过索引保持排序。
- 当连接条件除了等值连接（比如使用=运算符）之外，还包括范围查询（比如使用<、<=、>、>=运算符）。

示例

示例：两表做连接操作，且两表大小接近。

```
-- 数据准备。
gaussdb=# CREATE TABLE t1(id int,info text);
CREATE TABLE
gaussdb=# CREATE TABLE t2(id int,info text);
```



```
CREATE TABLE
gaussdb=# INSERT INTO t2 SELECT generate_series(1,100000),'bill'||generate_series(1,100000);
INSERT 0 100000
gaussdb=# INSERT INTO t1 SELECT generate_series(1,100000),'bill'||generate_series(1,100000);
INSERT 0 100000
-- 执行结果。
gaussdb=# EXPLAIN SELECT /*+ MERGEJOIN(t2 t1) */ * FROM t2 JOIN t1 ON (t1.id=t2.id);
                            QUERY PLAN
------------------------------------------------------------------------------
 Merge Join  (cost=19421.64..21421.64 rows=100000 width=26)
   Merge Cond: (t2.id = t1.id)
   ->  Sort  (cost=9710.82..9960.82 rows=100000 width=13)
         Sort Key: t2.id
         ->  Seq Scan on t2  (cost=0.00..1406.00 rows=100000 width=13)
   ->  Sort  (cost=9710.82..9960.82 rows=100000 width=13)
         Sort Key: t1.id
         ->  Seq Scan on t1  (cost=0.00..1406.00 rows=100000 width=13)
(8 rows)
```

上述示例中，Merge Join算子输出信息如下所示。

信息名称	含义
Merge Join	算子的名称。
Merge Cond	算子Merge join的连接谓词，示例中条件为t2.id等于t1.id，在查询执行时，满足这些条件的行会被包含在最终的结果集中。

6.2.3.4 运算符

6.2.3.4.1 Sort

算子说明

对底层节点返回的元组进行排序。Sort算子的作用是将查询结果按照指定的排序规则进行排序，然后返回有序的结果集。

典型场景

- 当查询语句中包含order by子句时，GaussDB会在执行计划中选择sort算子来进行排序操作。
- 采用MergeJoin来进行连接操作。

示例

示例：查询语句中包含ORDER BY子句。

```
-- 数据准备。
gaussdb=# CREATE TABLE student(id integer, class_id integer, grade number);
CREATE TABLE
gaussdb=# INSERT INTO student VALUES(generate_series(1,50), 1, floor(100 * random()));
INSERT 0 50
gaussdb=# INSERT INTO student VALUES(generate_series(51,100), 2, floor(100 * random()));
INSERT 0 50
gaussdb=# INSERT INTO student VALUES(generate_series(101,150), 3, floor(100 * random()));
INSERT 0 50
```



```
gaussdb=# INSERT INTO student VALUES(generate_series(151,200), 3, floor(100 * random()));
INSERT 0 50
-- 执行结果。
gaussdb=# EXPLAIN SELECT * FROM student ORDER BY grade;
                        QUERY PLAN
------------------------------------------------------------------
 Sort  (cost=10.64..11.14 rows=200 width=12)
   Sort Key: grade
   ->  Seq Scan on student  (cost=0.00..3.00 rows=200 width=12)
(3 rows)
```

上述示例中，Sort算子输出信息如下所示。

信息名称	含义
Sort	算子的名称。
Sort Key	Sort算子排序的依据关键字。示例中为grade。

6.2.3.4.2 Limit

算子说明

Limit算子限定了执行结果的输出记录数。如果增加了Limit算子，那么不是所有的行都会被检索到。

典型场景

使用带有Limit关键字的查询语句。

示例

示例：使用带有limit关键字的查询语句。

```
-- 数据准备。
gaussdb=# CREATE TABLE t1 ( id int, number int);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES (generate_series(1,50), 1);
INSERT 0 50
gaussdb=# EXPLAIN SELECT * FROM t1 LIMIt 10 OFFSET 20;
                      QUERY PLAN
-------------------------------------------------------------
 Limit  (cost=0.29..0.44 rows=10 width=8)
   ->  Seq Scan on t1  (cost=0.00..31.49 rows=10 width=8)
(2 rows)
```

6.2.3.4.3 Append

算子说明

Append用于多个关系集合的追加，同时处理包含一个或多个子计划的链表。

典型场景

- 带UNION的SQL语句场景。

- 带UNION ALL的SQL语句场景。

示例

示例1：带UNION的SQL语句场景。

```
-- 数据准备。
gaussdb=# CREATE TABLE t1(c1 number, c2 number, c3 number);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(generate_series(1, 100), 2, 3);
INSERT 0 100
gaussdb=# CREATE TABLE t2(c1 number, c2 number, c3 number);
CREATE TABLE
gaussdb=# INSERT INTO t2 VALUES(generate_series(1, 100), 2, 3);
INSERT 0 100
-- 执行结果。
gaussdb=# EXPLAIN SELECT * FROM t1 UNION SELECT * FROM t2;
                          QUERY PLAN
---------------------------------------------------------------------
 HashAggregate  (cost=31.57..39.05 rows=748 width=85)
   Group By Key: t1.c1, t1.c2, t1.c3
   ->  Append  (cost=0.00..25.96 rows=748 width=85)
         ->  Seq Scan on t1  (cost=0.00..2.00 rows=100 width=15)
         ->  Seq Scan on t2  (cost=0.00..16.48 rows=648 width=96)
(5 rows)
```

示例2：带UNION ALL的SQL语句场景。

```
-- 数据准备同上。
-- 执行结果。
gaussdb=# EXPLAIN SELECT * FROM t1 UNION ALL SELECT * FROM t2;
                          QUERY PLAN
---------------------------------------------------------------------
 Result  (cost=0.00..4.00 rows=200 width=15)
   ->  Append  (cost=0.00..4.00 rows=200 width=15)
         ->  Seq Scan on t1  (cost=0.00..2.00 rows=100 width=15)
         ->  Seq Scan on t2  (cost=0.00..2.00 rows=100 width=15)
(4 rows)
```

6.2.3.4.4 Agg

算子说明

Agg算子是用于执行聚集计算的算子, 支持3种策略处理：普通聚集（不分组只做聚集）、排序聚集和哈希聚集。排序聚集和哈希聚集因为涉及到分组，需要和group by搭配一起使用。排序聚集和哈希聚集的差别在于，排序聚集的输入必须是有序的，而哈希聚集则不关注输入的顺序性。必须说明即便是输入排好序的情况下，也不一定会选择排序聚集，因为哈希聚集可能会有更优的执行性能。

典型场景

- 普通聚集： 只涉及到聚集计算，一般用于对表做总数统计，或者对某一列做行数或特性统计；在执行计划中体现是Aggregate关键字。
- 排序聚集：输入元组是已经排好序的，或分组键恰好是有序列，则可能会选择排序聚集；在执行计划中体现是GroupAggregate关键字。
- 哈希聚集：业务中目前涉及的大部分场景，以及数据无序情况下的聚集计算；在执行计划中体现是HashAggregate关键字。

示例

示例1：普通聚集。


```
-- 数据准备。
gaussdb=# CREATE TABLE student(id integer, class_id integer, grade number);
CREATE TABLE
gaussdb=# INSERT INTO student VALUES(generate_series(1,50), 1, floor(100 * random()));
INSERT 0 50
gaussdb=# INSERT INTO student VALUES(generate_series(51,100), 2, floor(100 * random()));
INSERT 0 50
gaussdb=# INSERT INTO student VALUES(generate_series(101,150), 3, floor(100 * random()));
INSERT 0 50
gaussdb=# INSERT INTO student VALUES(generate_series(151,200), 3, floor(100 * random()));
INSERT 0 50
-- 执行结果。
gaussdb=# EXPLAIN SELECT count(*) FROM student;
                           QUERY PLAN
-----------------------------------------------------------------
 Aggregate  (cost=24.59..24.60 rows=1 width=8)
   ->  Seq Scan on student  (cost=0.00..21.67 rows=1167 width=0)
(2 rows)
```

示例2：排序聚集。

```
-- 数据准备同上。
-- 执行结果。
gaussdb=# SET enable_hashagg = off;
SET
gaussdb=# EXPLAIN SELECT id, count(class_id) FROM student GROUP BY id ORDER BY id;
                            QUERY PLAN
------------------------------------------------------------------------
 GroupAggregate  (cost=10.64..14.14 rows=200 width=16)
   Group By Key: id
   ->  Sort  (cost=10.64..11.14 rows=200 width=8)
         Sort Key: id
         ->  Seq Scan on student  (cost=0.00..3.00 rows=200 width=8)
(5 rows)
```

示例3：哈希聚集。

```
-- 数据准备同上。
-- 执行结果。
gaussdb=# EXPLAIN SELECT id, avg(grade) FROM student GROUP BY id;
                            QUERY PLAN
------------------------------------------------------------------
 HashAggregate  (cost=27.51..30.01 rows=200 width=36)
   Group By Key: id
   ->  Seq Scan on student  (cost=0.00..21.67 rows=1167 width=36)
(3 rows)
```

上述示例中，Agg算子输出信息如下所示。

信息名称	含义
Aggregate	普通的聚集算子名称，代表不分组只做聚集。
GroupAggregate	排序聚集算子名称，代表输入元组是已经排好序的，或分组键恰好是有序列。
HashAggregate	哈希聚集算子名称，大部分场景，以及数据无序情况下的聚集计算。

6.2.3.4.5 Group

算子说明

Group算子用于处理Group By子句，对下层排序元组进行分组操作，返回结果是按分组键分组后的结果。

典型场景

分组操作：查询某列有多少个不同的值，其作用类似于DISTINCT。

示例

示例：查询语句中包含GROUP BY子句

```
-- 数据准备。
gaussdb=# CREATE TABLE student(id integer, class_id integer, grade number);
CREATE TABLE
gaussdb=# INSERT INTO student VALUES(generate_series(1,50), 1, floor(100 * random()));
INSERT 0 50
gaussdb=# INSERT INTO student VALUES(generate_series(51,100), 2, floor(100 * random()));
INSERT 0 50
gaussdb=# INSERT INTO student VALUES(generate_series(101,150), 3, floor(100 * random()));
INSERT 0 50
gaussdb=# INSERT INTO student VALUES(generate_series(151,200), 3, floor(100 * random()));
INSERT 0 50
-- 执行结果。
gaussdb=# SET enable_hashagg = off;
SET
gaussdb=# EXPLAIN SELECT class_id FROM student GROUP BY class_id ORDER BY class_id;
                           QUERY PLAN
---------------------------------------------------------------------------
 Group  (cost=10.64..11.64 rows=3 width=4)
   Group By Key: class_id
   ->  Sort  (cost=10.64..11.14 rows=200 width=4)
         Sort Key: class_id
         ->  Seq Scan on student  (cost=0.00..3.00 rows=200 width=4)
(5 rows)
```

上述示例中，Group算子输出信息如下所示。

信息名称	含义
Group	算子的名称。
Group By Key	分组的关键字，示例中以class_id为关键字分组。

6.2.3.4.6 MergeAppend

算子说明

MergeAppend用于多个有序关系集合的追加，操作类似于Append，只是通过归并的方式对有序关系的集合进行加速运算。MergeAppend以保留排序顺序的方式对子查询结果进行组合，可用于组合表分区中已排序的行。因此，与普通Append不同，MergeAppend在执行操作之前需要确保输入的 m_plan 是有序的。通常，在MergeAppend操作之前（即执行计划的子树中）会出现排序算子 Sort。

典型场景

当分区扫描路径为Index或Index Only，且分区剪枝结果大于1，并且满足以下条件时。

所有分区索引均为有效的B-tree索引；SQL查询含Limit子句；分区扫描时，不存在带Filter的分区表查询语句。

当GUC参数sql_beta_feature = 'disable_merge_append_partition'时，不再生成MergeAppend路径。

示例

```
-- 数据准备。
gaussdb=# CREATE TABLE test_range_pt (a INT, b INT, c INT)
gaussdb-# PARTITION BY RANGE(a)
gaussdb-# (
gaussdb(#     PARTITION p1 VALUES LESS THAN (2000),
gaussdb(#     PARTITION p2 VALUES LESS THAN (3000),
gaussdb(#     PARTITION p3 VALUES LESS THAN (4000),
gaussdb(#     PARTITION p4 VALUES LESS THAN (5000),
gaussdb(#     PARTITION p5 VALUES LESS THAN (MAXVALUE)
gaussdb(# )ENABLE ROW MOVEMENT;
CREATE TABLE
gaussdb=# INSERT INTO test_range_pt VALUES
(generate_series(1,10000),generate_series(1,10000),generate_series(1,10000));
INSERT 0 10000
gaussdb=# CREATE INDEX idx_range_b ON test_range_pt(b) LOCAL;
CREATE INDEX
gaussdb=# ANALYZE test_range_pt;
ANALYZE
-- 执行结果。
gaussdb=# EXPLAIN ANALYZE SELECT * FROM test_range_pt WHERE b >10 AND b < 5000 ORDER BY b
LIMIT 10;
                                                  QUERY
PLAN
--------------------------------------------------------------------------------------------------------------------------------
------------------------------------
 Limit  (cost=0.06..1.02 rows=10 width=12) (actual time=1.089..1.128 rows=10 loops=1)
   ->  Result  (cost=0.06..480.32 rows=10 width=12) (actual time=1.076..1.113 rows=10 loops=1)
         ->  Merge Append  (cost=0.06..480.32 rows=10 width=12) (actual time=1.073..1.106 rows=10 loops=1)
               Sort Key: b
               ->  Partition Iterator  (cost=0.00..44.61 rows=998 width=12) (actual time=0.300..0.323 rows=10
loops=1)
                     Iterations: 1
                     ->  Partitioned Index Scan using idx_range_b on test_range_pt  (cost=0.00..44.61 rows=998
width=12) (actual time=0.261..0.280 rows=10 loops=1)
                           Index Cond: ((b > 10) AND (b < 5000))
                           Selected Partitions:  1
               ->  Partition Iterator  (cost=0.00..44.61 rows=998 width=12) (actual time=0.220..0.220 rows=1
loops=1)
                     Iterations: 1
                     ->  Partitioned Index Scan using idx_range_b on test_range_pt  (cost=0.00..44.61 rows=998
width=12) (actual time=0.186..0.186 rows=1 loops=1)
                           Index Cond: ((b > 10) AND (b < 5000))
                           Selected Partitions:  2
               ->  Partition Iterator  (cost=0.00..44.61 rows=998 width=12) (actual time=0.212..0.212 rows=1
loops=1)
                     Iterations: 1
                     ->  Partitioned Index Scan using idx_range_b on test_range_pt  (cost=0.00..44.61 rows=998
width=12) (actual time=0.183..0.183 rows=1 loops=1)
                           Index Cond: ((b > 10) AND (b < 5000))
                           Selected Partitions:  3
               ->  Partition Iterator  (cost=0.00..44.61 rows=998 width=12) (actual time=0.212..0.212 rows=1
loops=1)
                     Iterations: 1
                     ->  Partitioned Index Scan using idx_range_b on test_range_pt  (cost=0.00..44.61 rows=998
```

```
width=12) (actual time=0.183..0.183 rows=1 loops=1)
                        Index Cond: ((b > 10) AND (b < 5000))
                        Selected Partitions:  4
              ->  Partition Iterator  (cost=0.00..44.61 rows=998 width=12) (actual time=0.117..0.117 rows=0
loops=1)
                    Iterations: 1
                    ->  Partitioned Index Scan using idx_range_b on test_range_pt  (cost=0.00..44.61 rows=998
width=12) (actual time=0.089..0.089 rows=0 loops=1)
                        Index Cond: ((b > 10) AND (b < 5000))
                        Selected Partitions:  5
 Total runtime: 2.585 ms
(30 rows)
```

6.2.3.4.7 SetOp

算子说明

SetOp算子用于将两个或多个查询结果合并成一个结果集。SetOp算子包括INTERSECT和EXCEPT。

典型场景

- INTERSECT：返回两个查询结果的交集，即两个结果集中都存在的行。
- INTERSECT ALL：返回两个查询结果的交集，包括重复的行。
- EXCEPT：返回第一个查询结果中存在，但第二个查询结果中不存在的行。
- EXCEPT ALL：返回第一个查询结果中存在，但第二个查询结果中不存在的行，包括重复的行。

示例

示例1：INTERSECT。

```
-- 数据准备
gaussdb=# CREATE TABLE t(a int, b int, c int);
CREATE TABLE
gaussdb=# INSERT INTO t VALUES(generate_series(1, 10), generate_series(601, 610), generate_series(901,
910));
INSERT 0 10
-- 执行结果。
gaussdb=# EXPLAIN SELECT * FROM t INTERSECT SELECT * FROM t;
                                QUERY PLAN
-------------------------------------------------------------------------------
 HashSetOp Intersect  (cost=0.00..126.97 rows=217 width=12)
   ->  Append  (cost=0.00..97.80 rows=3890 width=12)
         ->  Subquery Scan on "*SELECT* 1"  (cost=0.00..48.90 rows=1945 width=12)
               ->  Seq Scan on t  (cost=0.00..29.45 rows=1945 width=12)
         ->  Subquery Scan on "*SELECT* 2"  (cost=0.00..48.90 rows=1945 width=12)
               ->  Seq Scan on t  (cost=0.00..29.45 rows=1945 width=12)
(6 rows)
```

上述示例中，SetOp算子输出信息如下所示。

信息名称	含义
SetOp	算子的名称。
Intersect	合并结果集的方式，示例中Intersect为返回两个查询结果的交集，即两个结果集中都存在的行。

示例2：INTERSECT ALL。

```
-- 数据准备同上。
-- 执行结果。
gaussdb=# EXPLAIN SELECT * FROM t INTERSECT ALL SELECT * FROM t;
                                QUERY PLAN
---------------------------------------------------------------------------------
 HashSetOp Intersect All  (cost=0.00..126.97 rows=1945 width=12)
   ->  Append  (cost=0.00..97.80 rows=3890 width=12)
         ->  Subquery Scan on "*SELECT* 1"  (cost=0.00..48.90 rows=1945 width=12)
               ->  Seq Scan on t  (cost=0.00..29.45 rows=1945 width=12)
         ->  Subquery Scan on "*SELECT* 2"  (cost=0.00..48.90 rows=1945 width=12)
               ->  Seq Scan on t  (cost=0.00..29.45 rows=1945 width=12)
(6 rows)
```

上述示例中，SetOp算子输出信息如下所示。

SetOp	算子的名称。
Intersect All	合并结果集的方式，示例中Intersect All为返回两个查询结果的交集，包括重复的行。

示例3：EXCEPT。

```
-- 数据准备同上。
-- 执行结果。
gaussdb=# EXPLAIN SELECT * FROM t EXCEPT SELECT * FROM t;
                                QUERY PLAN
---------------------------------------------------------------------------------
 HashSetOp Except  (cost=0.00..126.97 rows=217 width=12)
   ->  Append  (cost=0.00..97.80 rows=3890 width=12)
         ->  Subquery Scan on "*SELECT* 1"  (cost=0.00..48.90 rows=1945 width=12)
               ->  Seq Scan on t  (cost=0.00..29.45 rows=1945 width=12)
         ->  Subquery Scan on "*SELECT* 2"  (cost=0.00..48.90 rows=1945 width=12)
               ->  Seq Scan on t  (cost=0.00..29.45 rows=1945 width=12)
(6 rows)
```

上述示例中，SetOp算子输出信息如下所示。

信息名称	含义
SetOp	算子的名称。
Except	合并结果集的方式，示例中Except为返回第一个查询结果中存在，但第二个查询结果中不存在的行。

示例4：EXCEPT ALL。

```
-- 数据准备同上。
-- 执行结果。
gaussdb=# EXPLAIN SELECT * FROM t EXCEPT ALL SELECT * FROM t;
                                QUERY PLAN
---------------------------------------------------------------------------------
 HashSetOp Except All  (cost=0.00..126.97 rows=1945 width=12)
   ->  Append  (cost=0.00..97.80 rows=3890 width=12)
```



```
      ->  Subquery Scan on "*SELECT* 1"  (cost=0.00..48.90 rows=1945 width=12)
            ->  Seq Scan on t  (cost=0.00..29.45 rows=1945 width=12)
      ->  Subquery Scan on "*SELECT* 2"  (cost=0.00..48.90 rows=1945 width=12)
            ->  Seq Scan on t  (cost=0.00..29.45 rows=1945 width=12)
(6 rows)
```

上述示例中，SetOp算子输出信息如下所示。

信息名称	含义
SetOp	算子的名称。
Except All	合并结果集的方式，示例中Except All为返回第一个查询结果中存在，但第二个查询结果中不存在的行，包括重复的行。

6.2.3.4.8 RecursiveUnion

算子说明

RecursiveUnion算子用于处理递归调用的UNION语句，该类语句通常出现在CTE表达式中。常见的语法逻辑为：有一个初始输入集作为递归过程的初始数据，然后开始进行递归调用得到输出，最后将本次递归调用的输出作为下次递归调用的输入，循环调用得到最终的输出。

典型场景

带RecursiveUnion的SQL语句。

示例

示例：带RecursiveUnion的SQL语句。

```
-- 数据准备。
gaussdb=# CREATE TABLE t1(c1 number, c2 number, c3 number);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(generate_series(1, 100), 2, 3);
INSERT 0 100
-- 执行结果。
gaussdb=# EXPLAIN WITH RECURSIVE t1(n) AS (
gaussdb(# VALUES(1)
gaussdb(# UNION ALL
gaussdb(# SELECT n+1 FROM t1 WHERE n < 100)
gaussdb-# SELECT sum(n) FROM t1;
                              QUERY PLAN
------------------------------------------------------------------------------
 Aggregate  (cost=3.65..3.66 rows=1 width=12)
   CTE t1
     ->  Recursive Union  (cost=0.00..2.96 rows=31 width=4)
           ->  Values Scan on "*VALUES*"  (cost=0.00..0.01 rows=1 width=4)
           ->  WorkTable Scan on t1  (cost=0.00..0.23 rows=3 width=4)
                 Filter: (n < 100)
   ->  CTE Scan on t1  (cost=0.00..0.62 rows=31 width=4)
(7 rows)
```

6.2.3.4.9 Unique

算子说明

对下层的数据进行去重处理。在执行过程中，它将会遍历所有输入的数据，对其中的重复记录进行筛选，只保留唯一的记录。

典型场景

关闭enable_hashagg参数，使用带distinct查询。

示例

示例：使用带DISTINCT查询。

```
-- 数据准备。
gaussdb=# CREATE TABLE t1 (id INT , number INT);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(generate_series(1,50), 1);
INSERT 0 50
gaussdb=# INSERT INTO t1 VALUES(generate_series(1,50), 2);
INSERT 0 50
-- 执行结果。
gaussdb=# SET enable_hashagg = off;
SET
gaussdb=# EXPLAIN SELECT DISTINCT t1.id FROM t1;
                           QUERY PLAN
-----------------------------------------------------------------
 Unique  (cost=150.43..161.18 rows=200 width=4)
   ->  Sort  (cost=150.43..155.80 rows=2149 width=4)
         Sort Key: id
         ->  Seq Scan on t1  (cost=0.00..31.49 rows=2149 width=4)
(4 rows)
```

6.2.3.4.10 LockRows

算子说明

LockRows算子用于锁定查询结果集中的行，以防止其他事务对这些行进行修改或删除。

典型场景

- 事务中使用SELECT ... FOR SHARE/UPDATE锁定行，防止其他事务对这些行进行修改或删除。
- 使用FOR SHARE锁定行时，当前事务和其他事务都无法修改或删除锁定行。
- 使用FOR UPDATE锁定行时，除了当前事务，其他事务都无法修改或删除锁定行。

示例

示例：SELECT FOR UPDATE语法。

```
-- 数据准备
gaussdb=# CREATE TABLE t(a int, b int, c int);
CREATE TABLE
gaussdb=# INSERT INTO t VALUES(generate_series(1, 10), generate_series(601, 610), generate_series(901,
910));
INSERT 0 10
```



```
-- 执行结果。
gaussdb=# EXPLAIN SELECT * FROM t WHERE a = 1 FOR UPDATE;
                        QUERY PLAN
-----------------------------------------------------------
 LockRows  (cost=0.00..34.41 rows=10 width=18)
   ->  Seq Scan on t  (cost=0.00..34.31 rows=10 width=18)
         Filter: (a = 1)
(3 rows)
```

6.2.3.4.11 Materialize

算子说明

Materialize算子用于缓存子节点返回的结果，对子查询结果进行保存。对于需要重复多次扫描的子节点（特别是扫描结果每次都相同时）可以减少执行代价。

典型场景

- 当查询语句涉及子查询，需要多次查询同一批数据时，优化器会选择Materialize算子来缓存子查询的结果，从而大大减少扫描的执行时间。子查询常存在于关键字包括IN、ANY、ALL、EXISTS等子句中。
- 连接操作选择Nest Loop作为连接算子。

示例

示例：带ALL的子查询。

```
-- 数据准备。
gaussdb=# CREATE TABLE student(id integer, class_id integer, grade number);
CREATE TABLE
gaussdb=# CREATE TABLE class_table(class_id integer);
CREATE TABLE
gaussdb=# INSERT INTO student VALUES(generate_series(1,50), 1, floor(100 * random()));
INSERT 0 50
gaussdb=# INSERT INTO student VALUES(generate_series(51,100), 2, floor(100 * random()));
INSERT 0 50
gaussdb=# INSERT INTO student VALUES(generate_series(101,150), 3, floor(100 * random()));
INSERT 0 50
gaussdb=# INSERT INTO student VALUES(generate_series(151,200), 3, floor(100 * random()));
INSERT 0 50
gaussdb=# INSERT INTO class_table VALUES(1),(2),(3),(4);
INSERT 0 4
-- 执行结果。
gaussdb=# EXPLAIN SELECT * FROM student WHERE class_id >= all (SELECT class_id FROM class_table);
                              QUERY PLAN
-------------------------------------------------------------------------------
 Seq Scan on student  (cost=0.00..5206.50 rows=100 width=12)
   Filter: (SubPlan 1)
   SubPlan 1
     ->  Materialize  (cost=0.00..46.03 rows=2402 width=4)
           ->  Seq Scan on class_table  (cost=0.00..34.02 rows=2402 width=4)
(5 rows)
```

6.2.3.4.12 Result

算子说明

Result用于处理仅需要一次计算的条件表达式（2 > 1）或者insert中仅有一个VALUES子句，从而控制流程是否可以提前返回，不需要进行后续操作。

典型场景

- Result节点用于优化常量条件表达式的查询，条件表达不依赖于扫描的数据，比如：select * from t1 where 1 > 2。
- 当SELECT中没有FROM子句或者INSERT语句中只有一个VALUES子句时，执行器会直接计算SELECT的投影属性或者使用VALUES子句构造元组。

示例

示例1：Result节点用于优化常量条件表达式的查询。

```
-- 数据准备。
gaussdb=# CREATE TABLE t1(c1 number, c2 number, c3 number);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(generate_series(1, 100), 2, 3);
INSERT 0 100
-- 执行结果。
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE 1 > 2;
                        QUERY PLAN
------------------------------------------------------------
 Result  (cost=0.00..2.00 rows=1 width=15)
   One-Time Filter: false
   ->  Seq Scan on t1  (cost=0.00..2.00 rows=1 width=15)
(3 rows)
```

上述示例中，Result 算子输出信息如下所示。

信息名称	含义
Result	算子的名称。
One-Time Filter	过滤的时候用的常量表达式恒为false，因此只执行一次表达式即可。

示例2：SELECT中没有FROM子句。

```
-- 数据准备同上。
-- 执行结果。
gaussdb=# EXPLAIN SELECT dbe_raw.bit_and('1','2');
                QUERY PLAN
-------------------------------------------
 Result  (cost=0.00..0.01 rows=1 width=0)
(1 row)
```

示例3：INSERT语句中只有一个VALUES子句。

```
-- 数据准备同上。
-- 执行结果。
gaussdb=# EXPLAIN INSERT INTO t1 VALUES(1, 2, 3);
                   QUERY PLAN
-------------------------------------------------
 [Bypass]
 Insert on t1  (cost=0.00..0.01 rows=1 width=0)
   ->  Result  (cost=0.00..0.01 rows=1 width=0)
(3 rows)
```


6.2.3.4.13 WindowAgg

算子说明

WindowAgg算子用于处理元组窗口聚合，WindowAgg算子与Agg算子在功能上类似，实现的模式也相似。主要的区别在于，WindowAgg算子处理的元组限定于同一个窗口内，而Agg算子处理的元组是“整个表”（GROUP BY划分）。

典型场景

查询语句中包含窗口函数，如：row_number() OVER (PARTITION BY xxx)、avg(xxx) OVER (PARTITION BY xxx)。

示例

示例：带Agg(xxx) OVER(PARTITION BY xxx)的SQL语句。

```
-- 数据准备。
gaussdb=# CREATE TABLE t(a int, b int, c int);
CREATE TABLE
gaussdb=# INSERT INTO t VALUES(generate_series(1, 10), generate_series(601, 610), generate_series(901, 910));
INSERT 0 10
-- 执行结果。
gaussdb=# EXPLAIN SELECT a, avg(b) OVER(partition by a) FROM t;
                          QUERY PLAN
---------------------------------------------------------------------
 WindowAgg  (cost=135.70..169.74 rows=1945 width=8)
   ->  Sort  (cost=135.70..140.56 rows=1945 width=8)
         Sort Key: a
         ->  Seq Scan on t  (cost=0.00..29.45 rows=1945 width=8)
(4 rows)
```

6.2.3.4.14 StartWith Operator

算子说明

层次查询算子，用于执行递归查询操作。层次查询的执行流程是：

- 由START WITH区域的条件选择初始的数据集，把初始的数据集设为工作集。
- 只要工作集不为空，会用工作集的数据作为输入，查询下一轮的数据，过滤条件由CONNECT BY区域指定。其中，PRIOR关键字表示当前记录。
- 把步骤2中筛选出来的数据集，设为工作集，返回第二步重复操作。

同时，数据库为每一条选出来的数据添加下述的伪列，方便用户了解数据在递归或者树状结构中的位置。可以根据CONNECT BY中的条件，建立对应的索引，来提高START WITH语句的性能。

典型场景

使用层次查询。

示例

示例：使用层次查询。

```
-- 数据准备。
gaussdb=# CREATE TABLE area (id INT,name VARCHAR(25),parent_id INT);
```



```
CREATE TABLE
gaussdb=# INSERT INTO area VALUES (1,'中国',NULL);
INSERT 0 1
gaussdb=# INSERT INTO area VALUES (2,'北京市',1);
INSERT 0 1
gaussdb=# INSERT INTO area VALUES (3,'朝阳区',2);
INSERT 0 1
gaussdb=# INSERT INTO area VALUES (4,'陕西省',1);
INSERT 0 1
gaussdb=# INSERT INTO area VALUES (5,'西安市',4);
INSERT 0 1
gaussdb=# INSERT INTO area VALUES (6,'雁塔区',5);
INSERT 0 1
gaussdb=# INSERT INTO area VALUES (7,'未央区',5);
INSERT 0 1
-- 执行结果。
gaussdb=# EXPLAIN SELECT level, name FROM area START WITH (id = 1) CONNECT BY PRIOR id =
parent_id;
                                QUERY PLAN
-------------------------------------------------------------------------------------------
 CTE Scan on tmp_reuslt  (cost=294.67..295.55 rows=44 width=72)
   CTE tmp_reuslt
     ->  StartWith Operator  (cost=0.00..294.67 rows=44 width=76)
           Start With pseudo atts: array_key_1
           ->  Recursive Union  (cost=0.00..294.67 rows=44 width=76)
                 ->  Seq Scan on area  (cost=0.00..19.64 rows=4 width=76)
                       Filter: (id = 1)
                 ->  Hash Join  (cost=27.35..27.42 rows=4 width=76)
                       Hash Cond: (tmp_reuslt.id = public.area.parent_id)
                       ->  WorkTable Scan on tmp_reuslt  (cost=0.00..0.02 rows=1 width=4)
                       ->  Hash  (cost=17.71..17.71 rows=771 width=76)
                             ->  Seq Scan on area  (cost=0.00..17.71 rows=771 width=76)
(12 rows)
```

上述示例中，StartWith算子输出信息如下所示。

信息名称	含义
StartWith	算子的名称。
Start With pseudo atts	递归的关键列号。Start With pseudo atts: array_key_1表示被prior标记的条件是原表的第1列，以此列进行递归。

6.2.3.4.15 Rownum

算子说明

Rownum算子会生成伪列，它返回一个数字，表示从查询中获取结果的行编号。第一行的Rownum为1。可以使用Rownum关键字对查询结果的行编号进行条件过滤。通常出现在Rownum子句里。

使用Rownum有一定的约束条件：

- Rownum不可作为别名，以免SQL语句出现歧义。
- 创建索引时不可使用Rownum。
- 创建表时默认值不可为Rownum。
- Where子句中不可使用Rownum的别名。

- 在插入数据时不可使用Rownum。
- 在无表查询中不可以使用Rownum。
- Rownum不能用于Limit子句。
- Rownum不能用于EXECUTE语句的参数。
- Upsert语句不支持Rownum用做Update子句更新。

典型场景

条件过滤语句中包含Rownum的过滤条件。

示例

示例：条件过滤语句中包含Rownum的过滤条件。

```
-- 数据准备。
gaussdb=# CREATE TABLE t1 ( id INT , number INT );
CREATE TABLE
gaussdb=# INSERT INTo t1 VALUES(generate_series(1,50), 1);
INSERT 0 50
gaussdb=# INSERT INTO t1 VALUES(generate_series(1,50), 2);
INSERT 0 50
-- 执行结果。
gaussdb=# EXPLAIN SELECT t1.id FROM t1 WHERE rownum > 10;
                        QUERY PLAN
------------------------------------------------------------
 Rownum  (cost=0.00..2.50 rows=150 width=4)
   Filter: (ROWNUM > 10)
   ->  Seq Scan on t1  (cost=0.00..2.50 rows=150 width=4)
(3 rows)
```

上述示例中，Rownum 算子输出信息如下所示。

信息名称	含义
Rownum	算子的名称。
Filter	过滤的rownum的谓词，示例中的ROWNUM > 10表示，过滤的条件为显示rownum大于10的行。

6.2.3.4.16 Unpivot

算子说明

转置算子。用于将行转换为列。这个操作可以将一张表中的多列数据转换为两列，其中一列是原始表中的列名，另一列是对应的值。

典型场景

查询使用unpivot转置的表。

示例

示例：查询使用unpivot转置的表。


```
-- 数据准备。
gaussdb=# CREATE TABLE t1 (id int, number int, grade int);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(generate_series(1,100), 1, 2);
INSERT 0 100
-- 执行结果。
gaussdb=# EXPLAIN SELECT *  FROM t1 UNPIVOT (v1 FOR v2 in (id,number,grade));
                                QUERY PLAN
-------------------------------------------------------------------------------------
 Subquery Scan on __unnamed_unpivot_subquery__  (cost=0.00..87.80 rows=5806 width=36)
   Filter: (__unnamed_unpivot_subquery__.v1 IS NOT NULL)
   ->  Unpivot  (cost=0.00..29.45 rows=5835 width=36)
         ->  Seq Scan on t1  (cost=0.00..29.45 rows=1945 width=12)
(4 rows)
```

6.2.3.5 分区剪枝相关信息

6.2.3.5.1 Partition Iterator

算子说明

表分区技术（Table-Partitioning）通过将非常大的表或者索引从逻辑上切分为更小、更易管理的逻辑单元（分区），能够减小用户对表查询、变更等语句操作的影响范围；能够让用户通过分区键（Partition Key）快速定位到数据所在的分区，从而避免在数据库中对大表的全量扫描，能够在不同的分区上并发进行DDL、DML操作。GaussDB支持三种分区策略： 范围分区、哈希分区、列表分区。范围分区基于二分binary-search实现，复杂度为O(logN)；哈希分区和列表分区基于key-partOid哈希表实现，复杂度为O(1)。

典型场景

- 使用范围分区创建二级分区表。
- 使用哈希分区创建二级分区表。
- 使用列表分区创建二级分区表。

示例

示例1：使用范围分区创建二级分区表。

```
-- 数据准备。
gaussdb=# CREATE TABLE t1 (c1 int, c2 int) PARTITION BY RANGE (c1) SUBPARTITION BY RANGE (c1)
gaussdb-# (
gaussdb(#     PARTITION p1 VALUES LESS THAN(10)
gaussdb(#     (
gaussdb(#         SUBPARTITION subp1_1 VALUES LESS THAN(5),
gaussdb(#         SUBPARTITION subp1_2 VALUES LESS THAN(10)
gaussdb(#     ),
gaussdb(#     PARTITION p2 VALUES LESS THAN(20)
gaussdb(#     (
gaussdb(#         SUBPARTITION subp2_1 VALUES LESS THAN(15),
gaussdb(#         SUBPARTITION subp2_2 VALUES LESS THAN(20)
gaussdb(#     ),
gaussdb(#     PARTITION p3 VALUES LESS THAN(MAXVALUE)
gaussdb(#     (
gaussdb(#         SUBPARTITION subp3_1 VALUES LESS THAN(30),
gaussdb(#         SUBPARTITION subp3_2 VALUES LESS THAN(MAXVALUE)
gaussdb(#     )
gaussdb(# );
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(generate_series(1,100,2), 1);
INSERT 0 50
```



```
gaussdb=# INSERT INTO t1 VALUES(generate_series(2,101,2), 2);
INSERT 0 50
-- 执行结果。
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE c1 < 15 AND c2 <= 1;
                            QUERY PLAN
------------------------------------------------------------------------------
 Partition Iterator  (cost=0.00..33.23 rows=239 width=8)
   Iterations: 2, Sub Iterations: 3
   ->  Partitioned Seq Scan on t1  (cost=0.00..33.23 rows=239 width=8)
         Filter: ((c1 < 15) AND (c2 <= 1))
         Selected Partitions:  1..2
         Selected Subpartitions:  1:1..2  2:1
(6 rows))
```

上述示例中，Partition Iterator算子输出信息如下所示。

信息名称	含义
Partition Iterator	算子的名称。
Iterations	分区迭代算子对一级分区的迭代次数。如果显示PART则为动态剪枝场景。示例1中Iterations: 2 表示迭代算子需要遍历2个一级分区。Iterations: PART表示遍历一级分区个数需要由分区键上的参数条件决定。
Sub Iterations	分区迭代算子对二级分区的迭代次数。如果显示PART则为动态剪枝场景。 示例1中Sub Iterations: 3表示迭代算子需要遍历3个二级分区。Iterations: PART表示遍历二级分区个数需要由分区键上的参数条件决定。
Filter	该算子的过滤谓词，示例中的过滤条件为C1列的值小于15，且c2列的值小于等于1。在查询执行时，满足这些条件的行会被包含在最终的结果集中。
Partitioned Seq Scan	分区表的扫描方式。
Selected Partitions	一级分区剪枝的结果。示例1中Selected Partitions: 1..2 代表1、2分区被选中。
Selected Subpartitions	二级分区剪枝的结果。示例1中Selected Subpartitions: 1:1..2 2:1 代表第一个一级分区的1、2号两个子分区 和 第二个一级分区的第1号分区被选中。Selected Subpartitions: ALL表示所有二级分区均被选中。

示例2：使用哈希分区创建二级分区表。

```
-- 数据准备。
gaussdb=# CREATE TABLE sales (
gaussdb(#     id INT,
gaussdb(#     number INT
gaussdb(# )
```



```
gaussdb-# PARTITION BY HASH (number)
gaussdb-# SUBPARTITION BY HASH (id)
gaussdb-# (
gaussdb(#     PARTITION p0
gaussdb(# (
gaussdb(#     SUBPARTITION sp0_0,
gaussdb(#         SUBPARTITION sp0_1
gaussdb(# ),
gaussdb(#     PARTITION p1
gaussdb(# (
gaussdb(#     SUBPARTITION sp1_0,
gaussdb(#         SUBPARTITION sp1_1
gaussdb(# )
gaussdb(# );
CREATE TABLE
gaussdb=# INSERT INTO sales VALUES(generate_series(1,50), generate_series(1,50));
INSERT 0 50
-- 执行结果。
gaussdb=# EXPLAIN SELECT * FROM sales WHERE id < 15 AND number <= 50;
                            QUERY PLAN
------------------------------------------------------------------------------
 Partition Iterator  (cost=0.00..42.23 rows=239 width=8)
   Iterations: 2, Sub Iterations: 4
   ->  Partitioned Seq Scan on sales  (cost=0.00..42.23 rows=239 width=8)
         Filter: ((id < 15) AND ("number" <= 50))
         Selected Partitions:  1..2
         Selected Subpartitions:  ALL
(6 rows)
```

上述示例中，Partition Iterator算子输出信息如下所示。

信息名称	含义
Partition Iterator	算子的名称。
Iterations	分区迭代算子对一级分区的迭代次数。如果显示PART则为动态剪枝场景。示例2中Iterations: 2 表示迭代算子需要遍历2个一级分区。Iterations: PART表示遍历一级分区个数需要由分区键上的参数条件决定。
Sub Iterations	分区迭代算子对二级分区的迭代次数。如果显示PART则为动态剪枝场景。 示例2中Sub Iterations: 4表示迭代算子需要遍历4个二级分区。Iterations: PART表示遍历二级分区个数需要由分区键上的参数条件决定。
Filter	该算子的过滤谓词，示例2中的过滤条件为id列的值小于15，且number列的值小于等于50。在查询执行时，满足这些条件的行会被包含在最终的结果集中。
Partitioned Seq Scan	分区表的扫描方式。
Selected Partitions	一级分区剪枝的结果。示例2中Selected Partitions: 1..2 代表1、2分区被选中。

信息名称	含义
Selected Subpartitions	二级分区剪枝的结果。示例2中Selected Subpartitions: ALL表示所有二级分区均被选中。

示例3：使用列表分区创建二级分区表。

```
-- 数据准备。
gaussdb=# CREATE TABLE list_partitioned_table (
gaussdb(#     id INT,
gaussdb(#     category INT
gaussdb(# )
gaussdb-# PARTITION BY LIST (category)
gaussdb-# SUBPARTITION BY LIST (id)
gaussdb-# (
gaussdb(#     PARTITION p0 VALUES (1, 3, 5, 7)
gaussdb(# (
gaussdb(#     SUBPARTITION sp0_0 VALUES (0),
gaussdb(#         SUBPARTITION sp0_1 VALUES (1)
gaussdb(# ),
gaussdb(#     PARTITION p1 VALUES (2, 4, 6, 8)
gaussdb(# (
gaussdb(#     SUBPARTITION sp1_0 VALUES (0),
gaussdb(#         SUBPARTITION sp1_1 VALUES (1)
gaussdb(# )
gaussdb(# );
CREATE TABLE
gaussdb=# INSERT INTO list_partitioned_table VALUES(0,generate_series(1,8,2));
INSERT 0 4
gaussdb=# INSERT INTO list_partitioned_table VALUES(1,generate_series(2,7,2));
INSERT 0 3
-- 执行结果。
gaussdb=# EXPLAIN SELECT * FROM list_partitioned_table WHERE id = 0 AND category <= 5;
                                  QUERY PLAN
-------------------------------------------------------------------------------------
 Partition Iterator  (cost=0.00..42.23 rows=4 width=8)
   Iterations: 2, Sub Iterations: 2
   ->  Partitioned Seq Scan on list_partitioned_table  (cost=0.00..42.23 rows=4 width=8)
         Filter: ((category <= 5) AND (id = 0))
         Selected Partitions:  1..2
         Selected Subpartitions:  1:1  2:1
(6 rows)
```

上述示例中，Partition Iterator算子输出信息如下所示。

信息名称	含义
Partition Iterator	算子的名称。
Iterations	分区迭代算子对一级分区的迭代次数。如果显示PART则为动态剪枝场景。示例3中Iterations: 2 表示迭代算子需要遍历2个一级分区。Iterations: PART表示遍历一级分区个数需要由分区键上的参数条件决定。

信息名称	含义
Sub Iterations	分区迭代算子对二级分区的迭代次数。如果显示PART则为动态剪枝场景。 示例3中Sub Iterations: 2表示迭代算子需要遍历2个二级分区。Iterations: PART表示遍历二级分区个数需要由分区键上的参数条件决定。
Filter	该算子的过滤谓词，示例2中的过滤条件为category列的值小于等于5，且id列的值等于0。在查询执行时，满足这些条件的行会被包含在最终的结果集中。
Partitioned Seq Scan	分区表的扫描方式。
Selected Partitions	一级分区剪枝的结果。示例3中Selected Partitions: 1..2 代表1、2分区被选中。
Selected Subpartitions	二级分区剪枝的结果。示例3中Selected Subpartitions: 1:1 2:1 代表第一个一级分区的1号子分区 和 第二个一级分区的第1号子分区被选中。Selected Subpartitions: ALL表示所有二级分区均被选中。

6.2.3.6 其他关键字

6.2.3.6.1 InitPlan

算子说明

InitPlan是GaussDB的子计划的一部分。GaussDB中子查询计划可分为相关子计划和非相关子计划，相关子计划是指子查询依赖外部查询的行，不可独立于外部查询执行，非相关子计划则相反。在GaussDB中，SubPlan或InitPlan都可以叫做子计划，是相对于整个计划而言可以相对独立执行的部分，一般由不能提升的子计划生成。SubPlan主要是相关子计划生成的，InitPlan则是非相关子计划生成的。SubPlan是在主查询执行期间运行的，在主查询的每一行上重新执行一次，而InitPlan是在主查询执行之前运行的，结果是一次性的，它们在查询开始时计算一次，然后缓存起来在整个查询执行期间重用，所以InitPlan效率会更高。

典型场景

select子计划中不依赖外部查询，又不能进行提升。

示例

示例：SELECT子计划中不依赖外部查询，又不能进行提升。

```
-- 数据准备。
gaussdb=# CREATE TABLE init_table (id int, grade int, time int);
CREATE TABLE
```



```
gaussdb=# INSERT INTO init_table VALUES(generate_series(1,10000), (random() * 10)::integer,(random() *
10)::integer );
INSERT 0 10000
gaussdb=# CREATE TABLE t1(grade int);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES( 10),( 11);
INSERT 0 2
gaussdb=# CREATE TABLE t2(a int, b int, c int);
CREATE TABLE
gaussdb=# INSERT INTO t2 VALUES(1, 2, 3 );
INSERT 0 1
-- 执行结果。
gaussdb=# EXPLAIN SELECT * FROM init_table  WHERE init_table.grade  IN  (SELECT grade FROM t2) AND
init_table.grade != (SELECT * FROM t1);
                          QUERY PLAN
------------------------------------------------------------------------
 Seq Scan on init_table  (cost=34.02..130297.99 rows=3755 width=12)
   Filter: ((grade <> $2) AND (SubPlan 1))
   InitPlan 2 (returns $2)
     ->  Seq Scan on t1  (cost=0.00..34.02 rows=2402 width=4)
   SubPlan 1
     ->  Seq Scan on t2  (cost=0.00..29.45 rows=1945 width=0)
(6 rows)
```

上述示例中，输出信息如下所示。

信息名称	含义
InitPlan	非相关子计划。示例中InitPlan 2表示，子计划2为非相关子计划。
returns	返回的结果。示例中returns $2为将InitPlan的结果存在$2中。
SubPlan	相关子计划。示例中SubPlan 1表示，子计划1为相关子计划。

6.2.3.6.2 Stream

算子说明

Stream是GaussDB的SMP（对称多处理）中使用的一种技术，SMP采用多线程并行算法，在算子内并行执行，充分利用现代服务器单机多核的特点，提高执行效率。SMP分为计划生成与执行两部分：

- SMP计划生成。一阶段计划生成：在路径生成阶段，加入并行路径，最终根据代价，决定所选择的计划。两阶段计划生成：第一步生成原有的串行计划，第二步在将串行计划改造成适应并行的计划 。
- SMP执行过程。为并行执行线程之间进行数据分配、交换和汇总。

SMP的执行依赖Stream算子，不同类型的Stream算子配合完成SMP的执行过程。Stream算子的集中式类型主要分为：

- Local Gather： DN内数据分配，实现DN内部并行线程的数据汇总。
- Local Redistribute： DN内数据分配，在DN内部各线程之间，按照分布键进行数据重分布。
- Local Broadcast： DN内数据分配，将数据广播到DN内部的每个线程。

- Local RoundRobin ： DN内数据分配，在DN内部各线程之间实现数据轮询分发。

GaussDB通过broadcast与redistribute对数据进行多线程处理，然后通过gather汇总数据并处理返回。

典型场景

设置query_dop大于1的查询并行度。

示例

示例：设置query_dop大于1的查询并行度

```
-- 数据准备。
gaussdb=# CREATE TABLE t1 ( id int, number int);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES (generate_series(1, 500000), 1);
INSERT 0 500000
gaussdb=# CREATE TABLE t2 (id int, number int);
CREATE TABLE
gaussdb=# INSERT INTO t2 VALUES (generate_series(250000, 750000), 1);
INSERT 0 500001
gaussdb=# SET query_dop = 4;
SET
-- 执行结果。
gaussdb=#  EXPLAIN SELECT * FROM t1,t2  WHERE t1.id = t2.id;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Streaming(type: LOCAL GATHER dop: 1/4)  (cost=14971.50..33394.63 rows=408830 width=16)
   ->  Hash Join  (cost=14971.50..32097.07 rows=408830 width=16)
         Hash Cond: (t1.id = t2.id)
         ->  Streaming(type: LOCAL REDISTRIBUTE dop: 4/4)  (cost=0.00..15303.62 rows=500000 width=8)
               ->  Seq Scan on t1  (cost=0.00..4725.50 rows=500000 width=8)
         ->  Hash  (cost=13693.90..13693.90 rows=408830 width=8)
               ->  Streaming(type: LOCAL REDISTRIBUTE dop: 4/4)  (cost=0.00..13693.90 rows=408830 width=8)
                     ->  Seq Scan on t2  (cost=0.00..4497.57 rows=408830 width=8)
(8 rows)
gaussdb=#  explain select * from t1,t2  where t1.id > 250000;
                                      QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Streaming(type: LOCAL GATHER dop: 1/4)  (cost=2518.31..643988111.08 rows=102235709270 width=16)
   ->  Nested Loop  (cost=2518.31..319509541.62 rows=102235709270 width=16)
         ->  Streaming(type: BROADCAST dop: 4/4)  (cost=2518.31..18197.05 rows=1000276 width=8)
               ->  Seq Scan on t1  (cost=2518.31..5038.00 rows=250069 width=8)
                     Filter: (id > 250000)
         ->  Materialize  (cost=0.00..5008.61 rows=408830 width=8)
               ->  Seq Scan on t2  (cost=0.00..4497.57 rows=408830 width=8)
(7 rows)
```

上述示例中，Stream算子输出信息如下所示。

信息名称	含义
Streaming	算子的名称。
Type	Stream算子的类型。Gather 代表并行线程的数据汇总。Redistribute 代表各线程之间，按照分布键进行数据重分布。Broadcast代表数据广播到DN内部的每个线程。
dop	Stream算子的并行度。示例中 dop: 4/4 代表使用了全部4个线程。

6.3 调优流程

对慢SQL语句进行分析，通常包括以下步骤：

操作步骤

步骤1 收集SQL中涉及到的所有表的统计信息。在数据库中，统计信息是优化器生成计划的源数据。没有收集统计信息或者统计信息陈旧往往会造成执行计划严重劣化，从而导致性能问题。从经验数据来看，10%左右性能问题是因为没有收集统计信息。具体请参见**更新统计信息**。

步骤2 通过查看执行计划来查找原因。如果SQL长时间运行未结束，通过EXPLAIN命令查看执行计划，进行初步定位。如果SQL可以运行出结果来，则推荐使用EXPLAIN ANALYZE或EXPLAIN PERFORMANCE查看执行计划及实际运行情况，以便更精准地定位问题原因。有关执行计划的详细介绍请参见**SQL执行计划介绍**。

步骤3 **审视和修改表定义**。

步骤4 针对EXPLAIN或EXPLAIN PERFORMANCE信息，定位SQL慢的具体原因以及改进措施，具体参见**典型SQL调优点**。

步骤5 通常情况下，有些SQL语句可以通过查询重写转换成等价的，或特定场景下等价的语句。重写后的语句比原语句更简单，且可以简化某些执行步骤达到提升性能的目的。查询重写方法在各个数据库中基本是通用的。**经验总结：SQL语句改写规则**介绍了几种常用的通过改写SQL进行调优的方法。

步骤6 如果使用上述常规手段无法分析慢SQL根因的场景，还可以通过使用plan trace特性来分析慢SQL根因，具体请参见**PLAN TRACE使用介绍**。

----结束

6.4 更新统计信息

在数据库中，统计信息是优化器生成计划的源数据。没有收集统计信息或者统计信息陈旧往往会造成执行计划严重劣化，从而导致性能问题。

背景信息

ANALYZE语句可收集与数据库中表内容相关的统计信息，统计结果存储在系统表PG_STATISTIC中。查询优化器会使用这些统计数据，以生成最有效的执行计划。

建议在执行了大批量插入/删除操作后，例行对表或全库执行ANALYZE语句更新统计信息。目前默认收集统计信息的采样比例是30000行（即：guc参数default_statistics_target默认设置为100），如果表的总行数超过一定行数（大于1600000），建议设置guc参数default_statistics_target为-2，即按2%收集样本估算统计信息。

对于在批处理脚本或者存储过程中生成的中间表，也需要在完成数据生成之后显式的调用ANALYZE。

对于表中多个列有相关性且查询中有同时基于这些列的条件或分组操作的情况，可尝试收集多列统计信息，以便查询优化器可以更准确地估算行数，并生成更有效的执行计划。

操作步骤

使用以下命令更新某个表或者整个database的统计信息。

```
ANALYZE tablename;                    --更新单个表的统计信息
ANALYZE;                              --更新全库的统计信息
```

使用以下命令进行多列统计信息相关操作。

```
ANALYZE tablename ((column_1, column_2));                    --收集tablename表的column_1、column_2列的多列统计信息

ALTER TABLE tablename ADD STATISTICS ((column_1, column_2));    --添加tablename表的column_1、column_2列的多列统计信息声明
ANALYZE tablename;                                           --收集单列统计信息，并收集已声明的多列统计信息

ALTER TABLE tablename DELETE STATISTICS ((column_1, column_2)); --删除tablename表的column_1、column_2列的多列统计信息或其声明
```

须知

在使用ALTER TABLE tablename ADD STATISTICS语句添加了多列统计信息声明后，系统并不会立刻收集多列统计信息，而是在下次对该表或全库进行ANALYZE时，进行多列统计信息的收集。

如果想直接收集多列统计信息，请使用ANALYZE命令进行收集。

说明

使用EXPLAIN查看各SQL的执行计划时，如果发现某个表SEQ SCAN的输出中rows=10，rows=10是系统给的默认值，有可能该表没有进行ANALYZE，需要对该表执行ANALYZE。

6.5 审视和修改表定义

6.5.1 审视和修改表定义概述

好的表定义至少需要达到以下几个目标：

1. **减少扫描数据量**。通过分区的剪枝机制可以实现该点。
2. **尽量减少随机I/O**。通过聚簇可以实现该点。

表定义在数据库设计阶段创建，在SQL调优过程中进行审视和修改。

6.5.2 使用分区表

分区表是把逻辑上的一张表根据某种方案分成几张物理块进行存储。这张逻辑上的表称之为分区表，物理块称之为分区。分区表是一张逻辑表，不存储数据，数据实际是存储在分区上的。分区表和普通表相比具有以下优点：

1. 改善查询性能：对分区对象的查询可以仅搜索自己关心的分区，提高检索效率。
2. 增强可用性：如果分区表的某个分区出现故障，表在其他分区的数据仍然可用。
3. 方便维护：如果分区表的某个分区出现故障，需要修复数据，只修复该分区即可。

GaussDB支持的分区表为一级分区表和二级分区表，其中一级分区表包括范围分区表、间隔分区表、列表分区表、哈希分区表四种，二级分区表包括范围分区、列表分区、哈希分区两两组合的九种。

- 范围分区表：将数据基于范围映射到每一个分区。这个范围是由创建分区表时指定的分区键决定的。分区键经常采用日期，例如将销售数据按照月份进行分区。
- 间隔分区表：是一种特殊的范围分区表，相比范围分区表，新增间隔值定义，当插入记录找不到匹配的分区时，可以根据间隔值自动创建分区。
- 列表分区表：将数据中包含的键值分别存储在不同的分区中，依次将数据映射到每一个分区，分区中包含的键值由创建分区表时指定。
- 哈希分区表：将数据根据内部哈希算法依次映射到每一个分区中，包含的分区个数由创建分区表时指定。
- 二级分区表：由范围分区、列表分区、哈希分区任意组合得到的分区表，其一级分区和二级分区均可以使用前面三种定义方式。

6.5.3 选择数据类型

高效数据类型，主要包括以下三方面：

1. **尽量使用执行效率比较高的数据类型**

   一般来说整型数据运算(包括“=”、“>”、“<”、“≧”、“≦”、“≠”等常规的比较运算，以及group by)的效率比字符串、浮点数要高。

2. **尽量使用短字段的数据类型**

   长度较短的数据类型不仅可以减小数据文件的大小，提升I/O性能；同时也可以减小相关计算时的内存消耗，提升计算性能。比如对于整型数据，如果可以用smallint就尽量不用int，如果可以用int就尽量不用bigint。

3. **使用一致的数据类型**

   表关联列尽量使用相同的数据类型。如果表关联列数据类型不同，数据库必须动态地转化为相同的数据类型进行比较，这种转换会带来一定的性能开销。

6.6 典型 SQL 调优点

SQL调优是一个不断分析与尝试的过程：试跑Query，判断性能是否满足要求；如果不满足要求，则通过**查看执行计划**分析原因并进行针对性优化；然后重新试跑和优化，直到满足性能目标。

6.6.1 SQL 自诊断

用户在执行查询或者执行INSERT/DELETE/UPDATE/CREATE TABLE AS语句时，可能会遇到性能问题。这种情况下，通过查询**PG_CONTROL_GROUP_CONFIG**，**GS_SESSION_MEMORY_DETAIL**视图的warning字段可以获得对应查询可能导致性能问题的告警信息，为性能调优提供参考。

SQL自诊断的告警类型与GUC参数**resource_track_level**的设置有关系。如果**resource_track_level**设置为query，则可以诊断多列/单列统计信息未收集和SQL不下推的告警。如果**resource_track_level**设置为operator，则可以诊断所有的告警场景。

SQL自诊断的诊断范围与GUC参数**resource_track_cost**的设置有关系。当SQL的代价大于**resource_track_cost**时，SQL才会被诊断。SQL的代价可以通过explain来确认。

SQL自诊断功能受enable_analyze_check参数影响，使用前应确认该开关已打开。

执行语句较多时，可能会由于内存管控导致部分数据无法收集，可以尝试将instr_unique_sql_count设置值调高。

告警场景

目前支持对多列/单列统计信息未收集导致性能问题的场景上报告警。

如果存在单列或者多列统计信息未收集，则上报相关告警。调优方法可以参考**更新统计信息**和**统计信息调优**。

告警信息示例：

整表的统计信息未收集：

```
Statistic Not Collect:
    schema_test.t1
```

单列统计信息未收集：

```
Statistic Not Collect:
    schema_test.t2(c1,c2)
```

多列统计信息未收集:

```
Statistic Not Collect:
    schema_test.t3((c1,c2))
```

单列和多列统计信息未收集：

```
Statistic Not Collect:
    schema_test.t4(c1,c2)    schema_test.t4((c1,c2))
```

规格约束

1. 告警字符串长度上限为2048。如果告警信息超过这个长度（例如存在大量未收集统计信息的超长表名，列名等信息）则不告警，只上报warning：
   ```
   WARNING, "Planner issue report is truncated, the rest of planner issues will be skipped"
   ```
2. 如果query存在limit节点（即查询语句中包含limit），则不会上报limit节点以下的Operator级别的告警。

6.6.2 子查询调优

子查询背景介绍

应用程序通过SQL语句来操作数据库时会使用大量的子查询，这种写法比直接对两个表做连接操作在结构上和思路上更清晰，尤其是在一些比较复杂的查询语句中，子查询有更完整、更独立的语义，会使SQL对业务逻辑的表达更清晰更容易理解，因此得到了广泛的应用。

GaussDB根据子查询在SQL语句中的位置把子查询分成了子查询、子链接两种形式。

- 子查询SubQuery：对应于查询解析树中的范围表RangeTblEntry，更通俗一些指的是出现在FROM语句后面的独立的SELECT语句。
- 子链接SubLink：对应于查询解析树中的表达式，更通俗一些指的是出现在where/on子句、targetlist里面的语句。

  综上，对于查询解析树而言，SubQuery的本质是范围表、而SubLink的本质是表达式。针对SubLink场景而言，由于SubLink可以出现在约束条件、表达式中，按照GaussDB对sublink的实现，sublink可以分为以下几类：

- exist_sublink：对应EXIST、NOT EXIST语句
- any_sublink：对应op ANY(select…)语句，其中OP可以是“<”、“>”“=”操作符，IN/NOT IN (select...)也属于这一类。
- all_sublink：对应op ALL(select…)语句，其中OP可以是“<”、“>”“=”操作符
- rowcompare_sublink：对应record op(select …)语句
- expr_sublink：对应(SELECT with single targetlist item ...)语句
- array_sublink：对应ARRAY(select…)语句
- cte_sublink：对应with query(…)语句

其中的sublink为exist_sublink、any_sublink，在GaussDB的优化引擎中对其应用场景做了优化（子链接提升）。另外，expr_sublink也可以提升，但是由于SQL语句中子查询的使用的灵活性，会带来SQL子查询过于复杂造成性能问题。如果希望关闭expr_sublink的提升优化，可以通过guc参数rewrite_rule来设置 。子查询从大类上来看，分为非相关子查询和相关子查询：

- **非相关子查询None-Correlated SubQuery**

  子查询的执行不依赖于外层父查询的任何属性值。这样子查询具有独立性，可独自求解，形成一个子查询计划先于外层的查询求解。

  例如：

```
gaussdb=# select t1.c1,t1.c2
from t1
where t1.c1 in (
   select c2
   from t2
   where t2.c2 IN (2,3,4)
);
                          QUERY PLAN
---------------------------------------------------------------
 Hash Join
   Hash Cond: (t1.c1 = t2.c2)
   ->  Seq Scan on t1
         Filter: (c1 = ANY ('{2,3,4}'::integer[]))
   ->  Hash
         ->  HashAggregate
               Group By Key: t2.c2
               ->  Seq Scan on t2
                     Filter: (c2 = ANY ('{2,3,4}'::integer[]))
(9 rows)
```

- **相关子查询Correlated-SubQuery**

  子查询的执行依赖于外层父查询的一些属性值（如下列示例t2.c1 = t1.c1条件中的t1.c1）作为内层查询的一个AND-ed条件。这样的子查询不具备独立性，需要和外层查询按分组进行求解。

  例如：

```
gaussdb=# select t1.c1,t1.c2
from t1
where t1.c1 in (
   select c2
   from t2
   where t2.c1 = t1.c1 AND t2.c2 in (2,3,4)
);
                          QUERY PLAN
---------------------------------------------------------------------------
 Seq Scan on t1
   Filter: (SubPlan 1)
   SubPlan 1
     ->  Seq Scan on t2
           Filter: ((c1 = t1.c1) AND (c2 = ANY ('{2,3,4}'::integer[])))
(5 rows)
```


GaussDB 对 SubLink 的优化

针对SubLink的优化策略主要是让内层的子查询提升(pullup)，能够和外表直接做关联查询。判断子查询是否存在性能风险，可以通过explain查询语句查看Sublink的部分是否被转换成SubPlan的执行计划。

例如：

```
gaussdb=# explain select t1.c1, t1.c2 from t1 where t1.c1 in(select c2 from t2 where t2.c1 = t1.c1);
          QUERY PLAN
--------------------------------
 Seq Scan on t1
   Filter: (SubPlan 1)
   SubPlan 1
     ->  Seq Scan on t2
           Filter: (c1 = t1.c1)
(5 rows)
```

- **目前GaussDB支持的Sublink-Release场景**
  - IN-Sublink无相关条件
    - 不能包含上一层查询的表中的列（可以包含更高层查询表中的列）。
    - 不能包含易变函数。

    例如：

    ```
    gaussdb=# EXPLAIN SELECT t1.c1, t1.c2 FROM t1 WHERE t1.c1 IN(SELECT c2 FROM t2 WHERE
    t2.c1 = t1.c1);
              QUERY PLAN
    ---------------------------------
     Seq Scan on t1
       Filter: (SubPlan 1)
       SubPlan 1
         ->  Seq Scan on t2
               Filter: (c1 = t1.c1)
    (5 rows)
    ```

  - Exist-Sublink包含相关条件

    Where子句中必须包含上一层查询的表中的列，子查询的其它部分不能含有上层查询的表中的列。其它限制如下。

    - 子查询必须有from子句。
    - 子查询不能含有with子句。
    - 子查询不能含有聚集函数。
    - 子查询里不能包含集合操作、排序、limit、windowagg、having操作。
    - 不能包含易变函数。

    例如下面查询语句：

    ```
    gaussdb=# EXPLAIN (COSTS OFF) SELECT t1.c1, t1.c2 FROM t1 WHERE exists (SELECT c2 FROM
    t2 WHERE t2.c1 = t1.c1);
            QUERY PLAN
    -------------------------------
     Hash Semi Join
       Hash Cond: (t1.c1 = t2.c1)
       ->  Seq Scan on t1
       ->  Hash
             ->  Seq Scan on t2
    (5 rows)
    ```

    如上打印的执行计划应替换成下面的执行计划：


```
          QUERY PLAN
-----------------------------------
 Hash Join
   Hash Cond: (t1.c1 = t2.c1)
   ->  Seq Scan on t1
   ->  Hash
         ->  HashAggregate
               Group By Key: t2.c1
               ->  Seq Scan on t2
(7 rows)
```

- 包含聚集函数的等值相关子查询的提升

  子查询的where条件中必须含有来自上一层的列，而且此列必须和子查询本层涉及表中的列做相等判断，且这些条件必须用and连接。其它地方不能包含上层的列。其它限制条件如下。

  - 子查询中where条件包含的表达式（列名）必须是表中的列。
  - 子查询的Select关键字后，必须有且仅有一个输出列，此输出列必须是聚集函数（如max），并且聚集函数的参数(t2.c2)不能是来自外层表(t1)中的列。聚集函数不能是count。

    例如，下列示例可以提升。

    ```
    select * from t1 where c1 >(
          select max(t2.c1) from t2 where t2.c1=t1.c1
    );
    ```

    下列示例不能提升，因为子查询没有聚集函数。

    ```
    select * from t1 where c1 >(
          select  t2.c1 from t2 where t2.c1=t1.c1
    );
    ```

    下列示例不能提升，因为子查询有两个输出列。

    ```
    select * from t1 where (c1,c2) >(
          select  max(t2.c1),min(t2.c2) from t2 where t2.c1=t1.c1
    );
    ```

  - 子查询必须是from子句。
  - 子查询中不能有groupby、having、集合操作。
  - 子查询只能是inner join。

    例如：下列示例不能提升。

    ```
    select * from t1 where c1 >(
          select max(t2.c1) from t2 full join t3 on (t2.c2=t3.c2) where t2.c1=t1.c1
    );
    ```

  - 子查询的targetlist中不能包含返回set的函数。
  - 子查询的where条件中必须含有来自上一层的列，而且此列必须和子查询层涉及表中的列做相等判断，且这些条件必须用and连接。其它地方不能包含上层的上层中的列。例如：下列示例中的最内层子链接可以提升。

    ```
    select * from t3 where t3.c1=(
          select t1.c1
          from t1 where c1 >(
                select max(t2.c1) from t2 where t2.c1=t1.c1
    ));
    ```

    基于上面的示例，再加一个条件，则不能提升，因为最内侧子查询引用了上上层中的列。示例如下：

    ```
    select * from t3 where t3.c1=(
          select t1.c1
          from t1 where c1 >(
    ```



```
            select max(t2.c1) from t2 where t2.c1=t1.c1 and t3.c1>t2.c2

    ));
```

- 提升OR子句中的SubLink

  当WHERE过滤条件中有OR连接的EXIST相关SubLink，

  例如：

  ```
  select a, c from t1
  where t1.a = (select avg(a) from t3 where t1.b = t3.b) or
  exists (select * from t4 where t1.c = t4.c);
  ```

  将OR连接的EXIST相关子查询OR子句的提升过程：

  i. 提取where条件中，or子句中的opExpr。为：t1.a = (select avg(a) from t3 where t1.b = t3.b)

  ii. 这个op操作中包含subquery，判断是否可以提升，如果可以提升，重写subquery为：select avg(a), t3.b from t3 group by t3.b，生成not null条件t3.b is not null，并将这个opexpr用这个not null条件替换。此时SQL变为：

  ```
  select a, c
  from t1 left join (select avg(a) avg, t3.b from t3 group by t3.b)  as t3 on (t1.a = avg and t1.b = t3.b)
  where t3.b is not null or exists (select * from t4 where t1.c = t4.c);
  ```

  iii. 再次提取or子句中的exists sublink，exists (select * from t4 where t1.c = t4.c)，判断是否可以提升，如果可以提升，转换subquery为：select t4.c from t4 group by t4.c生成NotNull条件t4.c is not null提升查询，SQL变为：

  ```
  select t1.a, t1.c from t1 left join (select avg(a) avg, t3.b from t3 group by t3.b) as t3 on (t1.a = avg and t1.b = t3.b) left join (select t5.c from t5 group by t5.c) as t5 on (t1.c = t5.c) where t3.b is not null or t5.c is not null;
  ```

- **目前GaussDB不支持的Sublink-Release场景**

  除了以上场景之外都不支持Sublink提升，因此关联子查询会被计划成SubPlan +Broadcast的执行计划，当inner表的数据量较大时则会产生性能风险。

  如果相关子查询中跟外层的两张表做join，那么无法提升该子查询，需要通过将父SQL创建成with子句，然后再跟子查询中的表做相关子查询。

  例如：

  ```
  select distinct t1.a, t2.a
  from t1 left join t2 on t1.a=t2.a and not exists (select a,b from test1 where test1.a=t1.a and test1.b=t2.a);
  ```

  改写为

  ```
  with temp as
  (
        select * from (select t1.a as a, t2.a as b from t1 left join t2 on t1.a=t2.a)

  )
  select distinct a,b
  from temp
  where not exists (select a,b from test1 where temp.a=test1.a and temp.b=test1.b);
  ```

  - 出现在targetlist里的相关子查询无法提升（不含count）

    例如：

    ```
    gaussdb=# explain (costs off)
    select (select c2 from t2 where t1.c1 = t2.c1) ssq, t1.c2
    from t1
    where t1.c2 > 10;
    ```

    执行计划为：

    ```
    gaussdb=# explain (costs off)
    select (select c2 from t2 where t1.c1 = t2.c1) ssq, t1.c2
    ```



```
from t1
where t1.c2 > 10;
          QUERY PLAN
--------------------------------
 Seq Scan on t1
   Filter: (c2 > 10)
   SubPlan 1
     ->  Seq Scan on t2
           Filter: (t1.c1 = c1)
(5 rows)
```

由于相关子查询出现在targetlist（查询返回列表）里，对于t1.c1=t2.c1不匹配的场景仍然需要输出值，因此使用right-outerjoin关联t2&t1，以确保t1.c1=t2.c1在不匹配时，子SSQ能够返回不匹配的补空值。

说明

SSQ和CSSQ的解释如下：

- SSQ：ScalarSubQuery一般指返回1行1列scalar值的sublink，简称SSQ。
- CSSQ：Correlated-ScalarSubQuery和SSQ相同不过是指包含相关条件的SSQ。

上述SQL语句可以改写为：

```
with ssq as
(
   select * from t1 where t1.c2 >10
)
select t2.c2,ssq.c2 from t2 right join ssq on ssq.c1 = t2.c1;
```

改写后的执行计划为：

```
          QUERY PLAN
---------------------------------
 Hash Right Join
   Hash Cond: (t2.c1 = t1.c1)
   ->  Seq Scan on t2
   ->  Hash
         ->  Seq Scan on t1
               Filter: (c2 > 10)
(6 rows)
```

可以看到出现在SSQ返回列表里的相关子查询SSQ，已经被提升成Right Join，从而避免当内表t2较大时出现SubPlan计划导致性能变差。

- 出现在targetlist里的相关子查询无法提升（带count）

  例如：

```
select (select count(*) from t2 where t2.c1=t1.c1) cnt, t1.c1, t3.c1
from t1,t3
where t1.c1=t3.c1 order by cnt, t1.c1;
```

  执行计划为

```
                QUERY PLAN
-------------------------------------------
 Sort
   Sort Key: ((SubPlan 1)), t1.c1
   ->  Hash Join
         Hash Cond: (t1.c1 = t3.c1)
         ->  Seq Scan on t1
         ->  Hash
               ->  Seq Scan on t3
         SubPlan 1
           ->  Aggregate
                 ->  Seq Scan on t2
                       Filter: (c1 = t1.c1)
(11 rows)
```

  由于相关子查询出现在targetlist（查询返回列表）里，对于t1.c1=t2.c1不匹配的场景仍然需要输出值，因此使用left-outerjoin关联T1&T2确保t1.c1=t2.c1在不匹配时子SSQ能够返回不匹配的补空值，但是这里带了count

语句及时在t1.c1=t2.t1不匹配时需要输出0，因此可以使用一个case-when NULL then 0 else count(*)来代替。

上述SQL语句可以改写为：

```
with ssq as
(
   select count(*) cnt, c1 from t2 group by c1
)
select case when
          ssq.cnt is null then 0
          else ssq.cnt
      end cnt, t1.c1, t3.c1
from t1 left join ssq on ssq.c1 = t1.c1,t3
where t1.c1 = t3.c1
order by ssq.cnt, t1.c1;
```

改写后的执行计划为

```
              QUERY PLAN
-----------------------------------------
 Sort
   Sort Key: ssq.cnt, t1.c1
   CTE ssq
     ->  HashAggregate
           Group By Key: t2.c1
           ->  Seq Scan on t2
   ->  Hash Join
         Hash Cond: (t1.c1 = t3.c1)
         ->  Hash Left Join
               Hash Cond: (t1.c1 = ssq.c1)
               ->  Seq Scan on t1
               ->  Hash
                     ->  CTE Scan on ssq
         ->  Hash
               ->  Seq Scan on t3
(15 rows)
```

- 相关条件为不等值场景

  例如：

  ```
  select t1.c1, t1.c2
  from t1
  where t1.c1 = (select agg() from t2.c2 > t1.c2);
  ```

  对于非等值相关条件的SubLink目前无法提升，从语义上可以通过做2次join（一次CorrelationKey，一次rownum自关联）达到提升改写的目的。

  改写方案有两种。

  - 子查询改写方式

    ```
    select t1.c1, t1.c2
    from t1, (
       select t1.rowid, agg() aggref
       from t1,t2
       where t1.c2 > t2.c2 group by t1.rowid
    ) dt /* derived table */
    where t1.rowid = dt.rowid AND t1.c1 = dt.aggref;
    ```

  - CTE改写方式

    ```
    WITH dt as
    (
       select t1.rowid, agg() aggref
       from t1,t2
       where t1.c2 > t2.c2 group by t1.rowid
    )
    select t1.c1, t1.c2
    from t1, dt
    where t1.rowid = dt.rowid AND
    t1.c1 = dt.aggref;
    ```


须知

- 对于AGG类型为count(*)时需要进行CASE-WHEN对没有match的场景补0处理，非COUNT(*)场景NULL处理。
- CTE改写方式如果有sharescan支持性能上能够更优。

更多优化示例

示例：修改select语句，将子查询修改为和主表的join，或者修改为可以提升的subquery，但是在修改前后需要保证语义的正确性。

```
gaussdb=# explain (costs off) select * from t1 where t1.c1 in (select t2.c1 from t2 where t1.c1 = t2.c2);
          QUERY PLAN
-------------------------------
 Seq Scan on t1
   Filter: (SubPlan 1)
   SubPlan 1
     ->  Seq Scan on t2
           Filter: (t1.c1 = c2)
(5 rows)
```

上面事例计划中存在一个subPlan，为了消除这个subPlan可以修改语句为：

```
gaussdb=# explain (costs off) select * from t1 where exists (select t2.c1 from t2 where t1.c1 = t2.c2 and
t1.c1 = t2.c1);
               QUERY PLAN
-----------------------------------------
 Hash Join
   Hash Cond: (t1.c1 = t2.c2)
   ->  Seq Scan on t1
   ->  Hash
         ->  HashAggregate
               Group By Key: t2.c2, t2.c1
               ->  Seq Scan on t2
                     Filter: (c2 = c1)
(8 rows)
```

从计划可以看出，subPlan消除了，计划变成了两个表的hash join，这样会大大提高执行效率。

6.6.3 统计信息调优

统计信息调优介绍

GaussDB是基于代价估算生成的最优执行计划。优化器需要根据analyze收集的统计信息进行行数估算和代价估算，因此统计信息对优化器行数估算和代价估算起着至关重要的作用。通过analyze收集全局统计信息，主要包括：pg_class表中的relpages和reltuples；pg_statistic表中的stadistinct、stanullfrac、stanumbersN、stavaluesN、histogram_bounds等。

实例分析 1：未收集统计信息导致查询性能差

在很多场景下，由于查询中涉及到的表或列没有收集统计信息，会对查询性能有很大的影响。

表结构如下所示：

```
CREATE TABLE LINEITEM
(
```

```
L_ORDERKEY         BIGINT        NOT NULL
, L_PARTKEY        BIGINT        NOT NULL
, L_SUPPKEY        BIGINT        NOT NULL
, L_LINENUMBER     BIGINT        NOT NULL
, L_QUANTITY       DECIMAL(15,2) NOT NULL
, L_EXTENDEDPRICE  DECIMAL(15,2) NOT NULL
, L_DISCOUNT       DECIMAL(15,2) NOT NULL
, L_TAX            DECIMAL(15,2) NOT NULL
, L_RETURNFLAG     CHAR(1)       NOT NULL
, L_LINESTATUS     CHAR(1)       NOT NULL
, L_SHIPDATE       DATE          NOT NULL
, L_COMMITDATE     DATE          NOT NULL
, L_RECEIPTDATE    DATE          NOT NULL
, L_SHIPINSTRUCT   CHAR(25)      NOT NULL
, L_SHIPMODE       CHAR(10)      NOT NULL
, L_COMMENT        VARCHAR(44)   NOT NULL
);

CREATE TABLE ORDERS
(
O_ORDERKEY        BIGINT        NOT NULL
, O_CUSTKEY       BIGINT        NOT NULL
, O_ORDERSTATUS   CHAR(1)       NOT NULL
, O_TOTALPRICE    DECIMAL(15,2) NOT NULL
, O_ORDERDATE     DATE NOT NULL
, O_ORDERPRIORITY CHAR(15)      NOT NULL
, O_CLERK         CHAR(15)      NOT NULL
, O_SHIPPRIORITY  BIGINT        NOT NULL
, O_COMMENT       VARCHAR(79)   NOT NULL
);
```

查询语句如下所示：

```
explain verbose select
count(*) as numwait
from
lineitem l1,
orders
where
o_orderkey = l1.l_orderkey
and o_orderstatus = 'F'
and l1.l_receiptdate > l1.l_commitdate
and not exists (
select
*
from
lineitem l3
where
l3.l_orderkey = l1.l_orderkey
and l3.l_suppkey <> l1.l_suppkey
and l3.l_receiptdate > l3.l_commitdate
)
order by
numwait desc;
```

当出现该问题时，可以通过如下方法确认查询中涉及到的表或列有没有做过analyze收集统计信息。

1. 通过explain verbose执行query分析执行计划时会提示WARNING信息，如下所示：

   ```
   WARNING:Statistics in some tables or columns(public.lineitem.l_receiptdate,
   public.lineitem.l_commitdate, public.lineitem.l_orderkey, public.lineitem.l_suppkey,
   public.orders.o_orderstatus, public.orders.o_orderkey) are not collected.
   HINT:Do analyze for them in order to generate optimized plan.
   ```

2. 可以通过在pg_log目录下的日志文件中查找以下信息来确认是当前执行的query是否由于没有收集统计信息导致查询性能变差。

   ```
   2017-06-14 17:28:30.336 CST 140644024579856 20971684 [BACKEND] LOG:Statistics in some tables
   or columns(public.lineitem.l_receiptdate, public.lineitem.l_commitdate, public.lineitem.l_orderkey,
   ```



```
public.linei
tem.l_suppkey, public.orders.o_orderstatus, public.orders.o_orderkey) are not collected.
2017-06-14 17:28:30.336 CST 140644024579856 20971684 [BACKEND] HINT:Do analyze for them in order to generate optimized plan.
```

当通过以上方法查看到哪些表或列没有做analyze，可以通过对WARNING或日志中上报的表或列做analyze来解决由于未收集统计信息导致查询变慢的问题。

6.6.4 算子级调优

算子级调优介绍

一个查询语句要经过多个算子步骤才会输出最终的结果。由于个别算子耗时过长导致整体查询性能下降的情况比较常见。这些算子是整个查询的瓶颈算子。通用的优化手段是EXPLAIN ANALYZE/PERFORMANCE命令查看执行过程的瓶颈算子，然后进行针对性优化。

如下面的执行过程信息中，Hashagg算子的执行时间占总时间的：(51016-13535)/56476 ≈66%，此处Hashagg算子就是这个查询的瓶颈算子，在进行性能优化时应当优先考虑此算子的优化。

```
 id |                                operation                                |        A-time        | A-rows | E-rows |       Peak Memory       | E-memory | A-width | E-width |   E-costs
----+-------------------------------------------------------------------------+----------------------+--------+--------+-------------------------+----------+---------+---------+-------------
  1 | ->  Row Adapter                                                         | 56476.397            |  10000000 |   237060 | 19KB                    |          |         |      20 | 20933222.75
  2 |   ->  Vector Streaming (type: GATHER)                                   | 55664.220            |  10000000 |   237060 | 243KB                   |          |         |      20 | 20933222.75
  3 |     ->  Vector Hash Aggregate                                           | [55124.685,55132.180] |  10000000 |   237060 | [29349KB, 29441KB]      |  16MB    | [20,20] |      20 | 20918406.50
  4 |       ->  Vector Streaming(type: REDISTRIBUTE)                          | [52519.781,53709.779] | 339364604 |  4856184 | [1219KB, 1219KB]        |  1MB     |         |      20 | 10461210.85
  5 |         ->  Vector Hash Aggregate                                       | [35675.636,51016.424] | 339364604 |  4856184 | [732850KB, 746894KB]    |  16MB    | [20,20] |      20 | 10457195.65
  6 |           ->  Vector Partition Iterator                                 | [9035.202,13565.884]  | 970000000 | 935838097 | [9KB, 9KB]              |  1MB     |         |      20 | 10195891.68
  7 |             ->  Partitioned CStore Scan on xuji.e_mp_day_energy_csv_1 | [9015.645,13535.346]  | 970000000 | 935838097 | [845KB, 845KB]          |  1MB     |         |      20 | 10195891.68
(7 rows)
```

算子级调优示例

示例1：基表扫描时，对于点查或者范围扫描等过滤大量数据的查询，如果使用SeqScan全表扫描会比较耗时，可以在条件列上建立索引选择IndexScan进行索引扫描提升扫描效率。

```
gaussdb=#  explain (analyze on, costs off) select * from t1 where c1=10004;
 id |     operation      |  A-time  | A-rows | Peak Memory | A-width
----+--------------------+----------+--------+-------------+---------
  1 | ->  Seq Scan on t1 | 2053.069 |      7 | 64KB        |
(1 row)

 Predicate Information (identified by plan id)
-----------------------------------------------
   1 --Seq Scan on t1
         Filter: (c1 = 10004)
         Rows Removed by Filter: 110000
(3 rows)
gaussdb=#  create index idx on t1(c1);
CREATE INDEX
gaussdb=#  explain (analyze on, costs off) select * from t1 where c1=10004;
 id |           operation            | A-time | A-rows | Peak Memory | A-width
----+--------------------------------+--------+--------+-------------+---------
  1 | ->  Index Scan using idx on t1 | 0.227  |      7 | 77KB        |
(1 row)

 Predicate Information (identified by plan id)
-----------------------------------------------
   1 --Index Scan using idx on t1
         Index Cond: (c1 = 10004)
(2 rows)
```

上述例子中，全表扫描返回7条数据，过滤掉大量数据，在c1列上建立索引后，使用IndexScan扫描效率显著提高，从2秒降低到0.2毫秒。

示例2：如果从执行计划中看，两表join选择了NestLoop，而实际行数比较大时，NestLoop Join可能执行比较慢。如下的例子中NestLoop耗时27秒，如果设置参数

enable_mergejoin=off关掉Merge Join，同时设置参数enable_nestloop=off关掉NestLoop，让优化器选择HashJoin，则Join耗时降低至2秒。

```
gaussdb=#  explain analyze select count(*) from t1,t2 where t1.c1=t2.c2;
 id |          operation          |  A-time   |  A-rows  | E-rows | Peak Memory | A-width | E-width |         E-
costs
----+-----------------------------+-----------+----------+--------+-------------+---------+---------
+-------------------------
  1 | ->  Aggregate               | 27544.545 |        1 |      1 | 15KB        |         |       8 | 1046289.218..1046289.228
  2 |    ->  Nested Loop (3,4)    | 27544.028 |     1992 |   3133 | 8KB         |         |       0 | 0.000..1046281.385
  3 |       ->  Seq Scan on t1    | 2043.884  |   110007 | 118967 | 61KB        |         |       4 | 0.000..157587.670
  4 |       ->  Materialize       | 6877.119  | 54783486 |    498 | 65KB        |         |       4 | 0.000..11.470
  5 |          ->  Seq Scan on t2 | 0.430     |      498 |    498 | 57KB        |         |       4 | 0.000..8.980
(5 rows)
```

设置参数后：

```
gaussdb=#  set enable_mergejoin=off;
SET
gaussdb=#  set enable_nestloop=off;
SET
gaussdb=#  explain analyze select count(*) from t1,t2 where t1.c1=t2.c2;
 id |          operation          |  A-time  | A-rows | E-rows | Peak Memory | A-width | E-width |        E-
costs
----+-----------------------------+----------+--------+--------+-------------+---------+---------+-------------------------
  1 | ->  Aggregate               | 2103.025 |      1 |      1 | 15KB        |         |       8 | 158088.164..158088.174
  2 |    ->  Hash Join (3,4)      | 2102.536 |   1992 |   3133 | 10KB        |         |       0 | 15.205..158080.331
  3 |       ->  Seq Scan on t1    | 2063.595 | 110007 | 118967 | 61KB        |         |       4 | 0.000..157587.670
  4 |       ->  Hash              | 0.753    |    498 |    498 | 296KB       |         |       4 | 8.980..8.980
  5 |          ->  Seq Scan on t2 | 0.480    |    498 |    498 | 57KB        |         |       4 | 0.000..8.980
(5 rows)
```

示例3：通常情况下Agg选择HashAgg性能较好，如果大结果集选择了Sort +GroupAgg，则需要设置enable_sort=off，HashAgg耗时优于Sort+GroupAgg。

```
gaussdb=#  explain analyze select count(*) from t1 group by c1;
 id |        operation         |  A-time  | A-rows | E-rows | Peak Memory | A-width | E-width |        E-costs
----+--------------------------+----------+--------+--------+-------------+---------+---------+-------------------------
  1 | ->  GroupAggregate       | 2417.004 |  60000 |  18909 | 17KB        |         |      12 |
167616.708..168698.051
  2 |    ->  Sort              | 2304.329 | 200016 | 118967 | 26466KB     |         |       4 | 167616.708..167914.126
  3 |       ->  Seq Scan on t1 | 2125.464 | 200016 | 118967 | 58KB        |         |       4 | 0.000..157587.670
(3 rows)
```

设置参数后：

```
gaussdb=#  set enable_sort=off;
SET
gaussdb=#  explain analyze select count(*) from t1 group by c1;
 id |       operation        |  A-time  | A-rows | E-rows | Peak Memory | A-width | E-width |        E-costs
----+------------------------+----------+--------+--------+-------------+---------+---------+-------------------------
  1 | ->  HashAggregate      | 2324.062 |  60000 |  39912 | 10066KB     |         |       4 | 159297.545..159696.665
  2 |    ->  Seq Scan on t1 | 2131.319 | 200016 | 193303 | 58KB        |         |       4 | 0.000..158331.030
(2 rows)
```

6.7 经验总结：SQL 语句改写规则

根据数据库的SQL执行机制以及大量的实践，总结发现：通过一定的规则调整SQL语句，在保证结果正确的基础上，能够提高SQL执行效率。如果遵守这些规则，常常能够大幅度提升业务查询效率。

- **使用union all代替union**

  union在合并两个集合时会执行去重操作，而union all则直接将两个结果集合并、不执行去重。执行去重会消耗大量的时间，因此，在一些实际应用场景中，如果通过业务逻辑已确认两个集合不存在重叠，可用union all替代union以便提升性能。

- **join列增加非空过滤条件**

  若join列上的NULL值较多，则可以加上is not null过滤条件，以实现数据的提前过滤，提高join效率。

- **not in转not exists**

  not in语句需要使用nestloop anti join来实现，而not exists则可以通过hash anti join来实现。在join列不存在null值的情况下，not exists和not in等价。因此在确保没有null值时，可以通过将not in转换为not exists，通过生成hash join来提升查询效率。

  如下所示，如果t2.d2字段中没有null值(t2.d2字段在表定义中not null)查询可以修改为

  ```
  SELECT * FROM t1 WHERE  NOT EXISTS (SELECT * FROM t2 WHERE t1.c1=t2.d2);
  ```

  产生的计划如下：

  ```
  QUERY PLAN
  ------------------------------
  Hash Anti Join
  Hash Cond: (t1.c1 = t2.d2)
  ->  Seq Scan on t1
  ->  Hash
  ->  Seq Scan on t2
  (5 rows)
  ```

- **选择hashagg。**

  查询中GROUP BY语句如果生成了groupagg+sort的plan性能会比较差，可以通过加大work_mem的方法生成hashagg的plan，因为不用排序而提高性能。

- **尝试将函数替换为case语句。**

  GaussDB函数调用性能较低，如果出现过多的函数调用导致性能下降很多，可以根据情况把可下推函数的函数改成CASE表达式。

- **避免对索引使用函数或表达式运算。**

  对索引使用函数或表达式运算会停止使用索引转而执行全表扫描。

- **尽量避免在where子句中使用“!=”或“<”“>”操作符、null值判断、or连接、参数隐式转换。**

- 如果where条件中出现了 >= 和 <= 同一个值，由于当前不支持范围等价类推导，尽量把条件改为 = 查询。

  如：SELECT * FROM t1 WHERE c1 >= 1 AND c1 <= 1 修改为SELECT * FROM t1 WHERE c1 = 1。

  对于范围查询，优化器在计算选择率时误差相对等值查询较大，所以尽可能把范围查询改为等值查询。

- **对复杂SQL语句进行拆分。**

  对于过于复杂并且不易通过以上方法调整性能的SQL可以考虑拆分的方法，把SQL中某一部分拆分成独立的SQL并把执行结果存入临时表，拆分常见的场景包括但不限于：

  - 作业中多个SQL有同样的子查询，并且子查询数据量较大。
  - Plan cost计算不准，导致子查询hash bucket太小，比如实际数据1000W行，hash bucket只有1000。
  - 函数（如substr,to_number）导致大数据量子查询选择度计算不准。

6.8 SQL 调优关键参数调整

本节将介绍影响GaussDB SQL调优性能的关键数据库主节点配置参数，配置请联系管理员处理。

表 6-4 数据库主节点配置参数

参数/参考值	描述
enable_nestloop=on	控制查询优化器对嵌套循环连接（Nest Loop Join）类型的使用。当设置为“on”后，优化器优先使用Nest Loop Join；当设置为“off”后，优化器在存在其他方法时将优先选择其他方法。 **说明** 如果只需要在当前数据库连接（即当前Session）中临时更改该参数值，则只需要在SQL语句中执行如下命令： `SET enable_nestloop to off;` 此参数默认设置为“on”，但实际调优中应根据情况选择是否关闭。一般情况下，在三种join方式（Nested Loop、Merge Join和Hash Join）里，Nested Loop适合小数据量或者有索引的场景，Hash Join适合大数据分析场景。
enable_bitmapscan=on	控制查询优化器对位图扫描规划类型的使用。设置为“on”，表示使用；设置为“off”，表示不使用。 **说明** 如果只需要在当前数据库连接（即当前Session）中临时更改该参数值，则只需要在SQL语句中执行命令如下命令： `SET enable_bitmapscan to off;` bitmapscan扫描方式适用于“where a > 1 and b > 1”且a列和b列都有索引这种查询条件，但有时其性能不如indexscan。因此，现场调优如发现查询性能较差且计划中有bitmapscan算子，可以关闭bitmapscan，看性能是否有提升。
enable_hashagg=on	控制优化器对Hash聚集规划类型的使用。
enable_hashjoin=on	控制优化器对Hash连接规划类型的使用。
enable_mergejoin=on	控制优化器对融合连接规划类型的使用。
enable_indexscan=on	控制优化器对索引扫描规划类型的使用。
enable_indexonlyscan=on	控制优化器对仅索引扫描规划类型的使用。
enable_seqscan=on	控制优化器对顺序扫描规划类型的使用。完全消除顺序扫描是不可能的，但是关闭这个变量会让优化器在存在其他方法的时候优先选择其他方法。

参数/参考值	描述
enable_sort=on	控制优化器使用的排序步骤。该设置不可能完全消除明确的排序，但是关闭这个变量可以让优化器在存在其他方法的时候优先选择其他方法。
rewrite_rule	控制优化器是否启用LAZYAGG/MAGICSET/PARTIALPUSH/DISABLEREP/UNIQUECHECK/INTARGETLIST/PREDPUSH/PREDPUSHFORCE/PREDPUSHNORMAL/DISABLE_PULLUP_EXPR_SUBLINK/SUBLINK_PULLUP_ENHANCEDDISABLE_PULLUP_NOT_IN_SUBLINK/DISABLE_ROWNUM_PUSHDOWN/DISABLE_WINDOWAGG_PUSHDOWN重写规则。
sql_beta_feature	控制优化器是否启用。SEL_SEMI_POISSON/SEL_EXPR_INSTR/PARAM_PATH_GEN/RAND_COST_OPT/PARAM_PATH_OPT/PAGE_EST_OPT/CANONICAL_PATHKEY/PREDPUSH_SAME_LEVEL/PARTITION_FDW_ON/DISABLE_BITMAP_COST_WITH_LOSSY_PAGES/ENABLE_UPSERT_EXECUTE_GPLAN/DISABLE_FASTPATH_INSERT测试功能。
var_eq_const_selectivity	控制优化器是否使用直方图计算整型常量的选择率。
partition_page_estimation	控制分区表页面是否通过剪枝结果进行页面估算优化，只包括分区表和local索引页面，不包括全局索引页面。估算公式为： 估算后页面 = 分区表总页面 *（剪枝后分区数 / 总分区数）。
partition_iterator_elimination	控制分区表在分区剪枝结果为一个分区时，是否通过消除分区迭代算子来提升执行效率。

参数/参考值	描述
enable_functional_dependency	控制函数依赖统计信息的使用。 设置为“on”，开启两个功能： 1. 执行ANALYZE生成的多列统计信息包含函数依赖统计信息。 2. 计算选择率会使用函数依赖统计信息。 设置为“off”，此两个功能不生效： 1. 执行ANALYZE生成的多列统计信息不包含函数依赖统计信息。 2. 计算选择率不会使用函数依赖统计信息。 **说明** 函数依赖（Functional Dependency）的概念来自于关系数据库范式（Normal Form），表示属性间的函数关系。函数依赖统计信息，对此概念进行了扩展，表示满足函数关系的数据量占总数据量的比例。函数依赖统计信息是多列统计信息的一种，可以用于提升选择率估算的准确率。 函数依赖统计信息适用于形如“where a = 1 and b = 1”的格式，要求a和b均是同一个表的属性，约束条件为等式约束，约束条件用AND连接，约束条件至少为两个。
enable_seqscan_fusion	控制seqscan底噪消除是否打开。
enable_inner_unique_opt	控制优化器对Inner Unique优化的使用。

6.9 使用 Plan Hint 进行调优

6.9.1 Plan Hint 调优概述

Plan Hint为用户提供了直接影响执行计划生成的手段，用户可以通过指定join顺序，join、scan方法，指定结果行数，等多个手段来进行执行计划的调优，以提升查询的性能。

GaussDB还提供了SQL PATCH功能，在不修改业务语句的前提下通过创建SQL PATCH的方式使得Hint生效。

功能描述

Plan Hint在SELECT、INSERT、UPDATE、DELETE、MERGE等关键字后通过如下形式指定：

```
/*+ <plan hint>*/
```

可以同时指定多个hint，之间使用空格分隔。hint只能hint当前层的计划，对于子查询计划的hint，需要在子查询的select关键字后指定hint。

例如：

```
select /*+ <plan_hint1> <plan_hint2> */ * from t1, (select /*+ <plan_hint3> */ * from t2) where 1=1;
```


其中<plan_hint1>，<plan_hint2>为外层查询的hint，<plan_hint3>为内层子查询的hint。

检查Plan Hint调优的效果可以借助explain语法进行分析。通过explain可以查看使用Plan Hint后目标SQL的计划，对比计划是否符合要求以验证Plan Hint的效果。explain有多种计划展示的模式，通过explain_perf_mode进行控制。本节的示例一般通过设置explain_perf_mode为pretty模式来展示计划，展示较全的计划相关信息。部分示例设置explain_perf_mode为normal模式以精简输出信息。

须知

如果在视图定义（CREATE VIEW）时指定hint，则在该视图每次被应用时会使用该hint。

当使用random plan功能（参数plan_mode_seed不为0）时，查询指定的plan hint不会被使用。

支持范围

当前版本Plan Hint支持的范围如下，后续版本会进行增强。

- 指定Join顺序的hint - leading hint。
- 指定Join方式的hint，仅支持除semi/anti join，unique plan之外的常用hint。
- 指定结果集行数的hint。
- 指定Stream方式的hint。
- 指定Scan方式的hint，仅支持常用的tablescan，indexscan和indexonlyscan的hint。
- 指定子链接块名的hint。
- 指定子查询不展开的hint。
- 指定内表物化的hint。
- 指定Bitmapscan的hint。
- 指定Agg方法的hint。

注意事项

不支持Sort、Setop和Subplan的hint。

示例

本章节大部分示例使用下述语句，便于Plan Hint支持的各方法作对比，示例语句及不带hint的原计划如下所示：

```
create table t1(c1 int, c2 int, c3 int);
create table t2(c1 int, c2 int, c3 int);
create table t3(c1 int, c2 int, c3 int);
create index it1 on t1(c1,c2);
create index it2 on t2(c1,c2);
create index it3 on t1(c3,c2);
create table store
(
    s_store_sk              integer              not null,
    s_store_id              char(16)             not null,
```



```
    s_rec_start_date          date                        ,
    s_rec_end_date            date                        ,
    s_closed_date_sk          integer                     ,
    s_store_name              varchar(50)                 ,
    s_number_employees        integer                     ,
    s_floor_space             integer                     ,
    s_hours                   char(20)                    ,
    s_manager                 varchar(40)                 ,
    s_market_id               integer                     ,
    s_geography_class         varchar(100)                ,
    s_market_desc             varchar(100)                ,
    s_market_manager          varchar(40)                 ,
    s_division_id             integer                     ,
    s_division_name           varchar(50)                 ,
    s_company_id              integer                     ,
    s_company_name            varchar(50)                 ,
    s_street_number           varchar(10)                 ,
    s_street_name             varchar(60)                 ,
    s_street_type             char(15)                    ,
    s_suite_number            char(10)                    ,
    s_city                    varchar(60)                 ,
    s_county                  varchar(30)                 ,
    s_state                   char(2)                     ,
    s_zip                     char(10)                    ,
    s_country                 varchar(20)                 ,
    s_gmt_offset              decimal(5,2)                ,
    s_tax_precentage          decimal(5,2)                ,
    primary key (s_store_sk)
);
create table store_sales
(
    ss_sold_date_sk           integer                     ,
    ss_sold_time_sk           integer                     ,
    ss_item_sk                integer               not null,
    ss_customer_sk            integer                     ,
    ss_cdemo_sk               integer                     ,
    ss_hdemo_sk               integer                     ,
    ss_addr_sk                integer                     ,
    ss_store_sk               integer                     ,
    ss_promo_sk               integer                     ,
    ss_ticket_number          integer               not null,
    ss_quantity               integer                     ,
    ss_wholesale_cost         decimal(7,2)                ,
    ss_list_price             decimal(7,2)                ,
    ss_sales_price            decimal(7,2)                ,
    ss_ext_discount_amt       decimal(7,2)                ,
    ss_ext_sales_price        decimal(7,2)                ,
    ss_ext_wholesale_cost     decimal(7,2)                ,
    ss_ext_list_price         decimal(7,2)                ,
    ss_ext_tax                decimal(7,2)                ,
    ss_coupon_amt             decimal(7,2)                ,
    ss_net_paid               decimal(7,2)                ,
    ss_net_paid_inc_tax       decimal(7,2)                ,
    ss_net_profit             decimal(7,2)                ,
    primary key (ss_item_sk, ss_ticket_number)
);
create table store_returns
(
    sr_returned_date_sk       integer                     ,
    sr_return_time_sk         integer                     ,
    sr_item_sk                integer               not null,
    sr_customer_sk            integer                     ,
    sr_cdemo_sk               integer                     ,
    sr_hdemo_sk               integer                     ,
    sr_addr_sk                integer                     ,
    sr_store_sk               integer                     ,
    sr_reason_sk              integer                     ,
    sr_ticket_number          integer               not null,
    sr_return_quantity        integer                     ,
```

```
    sr_return_amt             decimal(7,2)               ,
    sr_return_tax             decimal(7,2)               ,
    sr_return_amt_inc_tax     decimal(7,2)               ,
    sr_fee                    decimal(7,2)               ,
    sr_return_ship_cost       decimal(7,2)               ,
    sr_refunded_cash          decimal(7,2)               ,
    sr_reversed_charge        decimal(7,2)               ,
    sr_store_credit           decimal(7,2)               ,
    sr_net_loss               decimal(7,2)               ,
    primary key (sr_item_sk, sr_ticket_number)
);
create table customer
(
    c_customer_sk             integer               not null,
    c_customer_id             char(16)              not null,
    c_current_cdemo_sk        integer                       ,
    c_current_hdemo_sk        integer                       ,
    c_current_addr_sk         integer                       ,
    c_first_shipto_date_sk    integer                       ,
    c_first_sales_date_sk     integer                       ,
    c_salutation              char(10)                      ,
    c_first_name              char(20)                      ,
    c_last_name               char(30)                      ,
    c_preferred_cust_flag     char(1)                       ,
    c_birth_day               integer                       ,
    c_birth_month             integer                       ,
    c_birth_year              integer                       ,
    c_birth_country           varchar(20)                   ,
    c_login                   char(13)                      ,
    c_email_address           char(50)                      ,
    c_last_review_date        char(10)                      ,
    primary key (c_customer_sk)
);
create table promotion
(
    p_promo_sk                integer               not null,
    p_promo_id                char(16)              not null,
    p_start_date_sk           integer                       ,
    p_end_date_sk             integer                       ,
    p_item_sk                 integer                       ,
    p_cost                    decimal(15,2)                 ,
    p_response_target         integer                       ,
    p_promo_name              char(50)                      ,
    p_channel_dmail           char(1)                       ,
    p_channel_email           char(1)                       ,
    p_channel_catalog         char(1)                       ,
    p_channel_tv              char(1)                       ,
    p_channel_radio           char(1)                       ,
    p_channel_press           char(1)                       ,
    p_channel_event           char(1)                       ,
    p_channel_demo            char(1)                       ,
    p_channel_details         varchar(100)                  ,
    p_purpose                 char(15)                      ,
    p_discount_active         char(1)                       ,
    primary key (p_promo_sk)
);
create table customer_address
(
    ca_address_sk             integer               not null,
    ca_address_id             char(16)              not null,
    ca_street_number          char(10)                      ,
    ca_street_name            varchar(60)                   ,
    ca_street_type            char(15)                      ,
    ca_suite_number           char(10)                      ,
    ca_city                   varchar(60)                   ,
    ca_county                 varchar(30)                   ,
    ca_state                  char(2)                       ,
    ca_zip                    char(10)                      ,
    ca_country                varchar(20)                   ,
```

```
    ca_gmt_offset             decimal(5,2)                ,
    ca_location_type          char(20)                    ,
    primary key (ca_address_sk)
);
create table item
(
    i_item_sk                 integer               not null,
    i_item_id                 char(16)              not null,
    i_rec_start_date          date                        ,
    i_rec_end_date            date                        ,
    i_item_desc               varchar(200)                ,
    i_current_price           decimal(7,2)                ,
    i_wholesale_cost          decimal(7,2)                ,
    i_brand_id                integer                     ,
    i_brand                   char(50)                    ,
    i_class_id                integer                     ,
    i_class                   char(50)                    ,
    i_category_id             integer                     ,
    i_category                char(50)                    ,
    i_manufact_id             integer                     ,
    i_manufact                char(50)                    ,
    i_size                    char(20)                    ,
    i_formulation             char(20)                    ,
    i_color                   char(20)                    ,
    i_units                   char(10)                    ,
    i_container               char(10)                    ,
    i_manager_id              integer                     ,
    i_product_name            char(50)                    ,
    primary key (i_item_sk)
);
explain
select i_product_name product_name
,i_item_sk item_sk
,s_store_name store_name
,s_zip store_zip
,ad2.ca_street_number c_street_number
,ad2.ca_street_name c_street_name
,ad2.ca_city c_city
,ad2.ca_zip c_zip
,count(*) cnt
,sum(ss_wholesale_cost) s1
,sum(ss_list_price) s2
,sum(ss_coupon_amt) s3
FROM   store_sales
,store_returns
,store
,customer
,promotion
,customer_address ad2
,item
WHERE  ss_store_sk = s_store_sk AND
ss_customer_sk = c_customer_sk AND
ss_item_sk = i_item_sk and
ss_item_sk = sr_item_sk and
ss_ticket_number = sr_ticket_number and
c_current_addr_sk = ad2.ca_address_sk and
ss_promo_sk = p_promo_sk and
i_color in ('maroon','burnished','dim','steel','navajo','chocolate') and
i_current_price between 35 and 35 + 10 and
i_current_price between 35 + 1 and 35 + 15
group by i_product_name
,i_item_sk
,s_store_name
,s_zip
,ad2.ca_street_number
,ad2.ca_street_name
,ad2.ca_city
,ad2.ca_zip
;
```

```
 HashAggregate  (cost=18.09..18.10 rows=1 width=776)
   Group By Key: item.i_product_name, item.i_item_sk, store.s_store_name, store.s_zip, ad2.ca_street_number,
ad2.ca_street_name, ad2.ca_city, ad2.ca_zip
   ->  Nested Loop  (cost=0.00..18.06 rows=1 width=776)
         ->  Nested Loop  (cost=0.00..17.37 rows=1 width=416)
               ->  Nested Loop  (cost=0.00..17.08 rows=1 width=420)
                     ->  Nested Loop  (cost=0.00..16.67 rows=1 width=420)
                           ->  Nested Loop  (cost=0.00..16.26 rows=1 width=262)
                                 Join Filter: (item.i_item_sk = store_sales.ss_item_sk)
                                 ->  Nested Loop  (cost=0.00..15.46 rows=2 width=216)
                                       ->  Seq Scan on item  (cost=0.00..11.16 rows=1 width=208)
                                             Filter: ((i_current_price >= 35::numeric) AND (i_current_price <= 45::numeric)
AND (i_current_price >= 36::numeric) AND (i_current_price <= 50::numeric) AN
D (i_color = ANY ('{maroon,burnished,dim,steel,navajo,chocolate}'::bpchar[])))
                                       ->  Index Only Scan using store_returns_pkey on store_returns  (cost=0.00..4.29
rows=2 width=8)
                                             Index Cond: (sr_item_sk = item.i_item_sk)
                                 ->  Index Scan using store_sales_pkey on store_sales  (cost=0.00..0.38 rows=1
width=62)
                                       Index Cond: ((ss_item_sk = store_returns.sr_item_sk) AND (ss_ticket_number =
store_returns.sr_ticket_number))
                           ->  Index Scan using store_pkey on store  (cost=0.00..0.40 rows=1 width=166)
                                 Index Cond: (s_store_sk = store_sales.ss_store_sk)
                     ->  Index Scan using customer_pkey on customer  (cost=0.00..0.40 rows=1 width=8)
                           Index Cond: (c_customer_sk = store_sales.ss_customer_sk)
               ->  Index Only Scan using promotion_pkey on promotion  (cost=0.00..0.28 rows=1 width=4)
                     Index Cond: (p_promo_sk = store_sales.ss_promo_sk)
         ->  Index Scan using customer_address_pkey on customer_address ad2  (cost=0.00..0.68 rows=1
width=368)
               Index Cond: (ca_address_sk = customer.c_current_addr_sk)
(23 rows)
```

6.9.2 指定 Hint 所处于的查询块 Queryblock

功能描述

该功能允许用户在Hint中通过@queryblock来实现查询块级别的Hint控制，可以指定Hint生效的查询块，比如在外层查询块指定内层查询块的Hint。

语法格式

在Hint的参数最开始加入可缺省的@queryblock, Hint_SEPC为某Hint。

```
Hint_SEPC([@queryblock])
```

参数说明

Hint_SEPC为hint名，@queryblock可缺省，若缺省表示在Hint声明的当前查询块生效。若@queryblock缺省之后导致Hint_SPEC无参，则Hint不需要使用括号，直接写成Hint_SPEC，而非Hint_SPEC()。下面分别从queryblock的命名 和 Hint生效的方式给出例子。部分Hint无法仅在最外层生效，且不支持通过@queryblock方式指定，具体参见各自Hint的语法说明。

- 查询块QueryBlock的命名：

  每个查询块，都需要给出一个名称，以实现对Hint的精确指定。命名方式有两种，用户指定和系统默认指定。

  a. 用户可以通过使用blockname的Hint实现对于查询块名称的指定，具体可以参考**子链接块名的hint**。

  b. 若系统对于查询块没有指定默认别名，则系统会自动按照处理的顺序生成默认块名。一般情况下，每个查询块的默认别名由其所在的查询块名的首3个字

母、"$"、查询块的编号组成，比如第一个select查询块的别名为sel$1。 在pretty 模式下，加入blockname 开关的explain方式可以查看对于查询中每个表的处理算子，所在的查询块名。

```
gaussdb=# set explain_perf_mode = pretty;
SET
gaussdb=# explain (blockname on,costs off) select * from t1, (select c1 from t2 group by c1) sub1
where t1.c1 = sub1.c1;
 id |             operation              | Query Block
----+------------------------------------+-------------
  1 | ->  Hash Join (2,3)                | sel$1
  2 |    ->  Seq Scan on t1@"sel$1"      | sel$1
  3 |    ->  Hash                        |
  4 |       ->  HashAggregate            | sel$2
  5 |          ->  Seq Scan on t2@"sel$2" | sel$2
(5 rows)
```

可以看到t2的扫描在sel$2的查询块中。

- @queryblock对于查询块的指定：

  对于上述例子，修改t2中indexscan的方式。

  ```
  select /*+indexscan(@sel$2 t2) tablescan(t1)*/ * from t1, (select c1 from t2 group by c1) sub1 where
  t1.c1 = sub1.c1;
  ```

  indexscan 和 tablescan都为扫描方式的Hint, 扫描方式相关的Hint可以参考**SCAN方式的Hint**。 通过在sel$1的查询块中指定 indexscan(@sel$2 t2)的hint，可以将该hint移至查询块sel$2中，对t2生效。若后续改写时查询块sel$2被提升至sel$1，则该Hint也会一起被提升至sel$1,继续对t2生效。

  ```
  gaussdb=#  explain (blockname on,costs off) select /*+indexscan(@sel$2 t2) tablescan(t1)*/ * from t1,
  (select c1 from t2 group by c1) sub1 where t1.c1 = sub1.c1;
   id |                    operation                    | Query Block
  ----+-------------------------------------------------+-------------
    1 | ->  Hash Join (2,3)                             | sel$1
    2 |    ->  Seq Scan on t1@"sel$1"                   | sel$1
    3 |    ->  Hash                                     |
    4 |       ->  Group                                 | sel$2
    5 |          ->  Index Only Scan using it2 on t2@"sel$2" | sel$2
  (5 rows)
  ```

⚠ 注意

有时候，优化器阶段的查询重写会展开一些查询块，导致计划在explain中不显示相关查询块。Hint指定查询块是根据优化器阶段之前查询块名字进行指定。当意图获知名字的查询块可能会在计划阶段被展开时，可以加入 no_expand的hint(参见**指定子查询不展开的Hint**)，让其不被展开。

1. 查询块sel$2是简单查询，优化器后续处理时进行查询改写，t1提升至sel$1进行处理，因此计划中没有显示在sel$2查询块的操作。

```
gaussdb=# explain (blockname on,costs off) select * from t2, (select c1 from t1 where t1.c3 = 2) sub1 where t2.c1 = sub1.c1;
 id |              operation               | Query Block
----+--------------------------------------+-------------
  1 | ->  Hash Join (2,3)                  | sel$1
  2 |   ->  Seq Scan on t2@"sel$1"         | sel$1
  3 |   ->  Hash                           |
  4 |     ->  Bitmap Heap Scan on t1@"sel$2" | sel$1
  5 |       ->  Bitmap Index Scan using it3 |
(5 rows)
```

2. 查询块sel$2是简单查询，优化器后续处理时因为no_expand跳过查询改写，t1还在原查询块处理。

```
gaussdb=# explain (blockname on,costs off) select * from t2, (select /*+ no_expand*/ c1 from t1 where t1.c3 = 2) sub1 where t2.c1 = sub1.c1;
 id |              operation               | Query Block
----+--------------------------------------+-------------
  1 | ->  Hash Join (2,3)                  | sel$1
  2 |   ->  Seq Scan on t2@"sel$1"         | sel$1
  3 |   ->  Hash                           |
  4 |     ->  Bitmap Heap Scan on t1@"sel$2" | sel$2
  5 |       ->  Bitmap Index Scan using it3 |
(5 rows)
```

3. 通过no_expand知道t1处于sel$2查询块后，可以通过@sel$2进行Hint的查询块指定。

```
gaussdb=# explain (blockname on,costs off) select/*+ indexscan(@sel$2 t1)*/ * from t2, (select c1 from t1 where t1.c3 = 2) sub1 where t2.c1 = sub1.c1;
 id |                operation                 | Query Block
----+------------------------------------------+-------------
  1 | ->  Hash Join (2,3)                      | sel$1
  2 |   ->  Seq Scan on t2@"sel$1"             | sel$1
  3 |   ->  Hash                               |
  4 |     ->  Index Scan using it3 on t1@"sel$2" | sel$1
(4 rows)
```

4. view中查询块的编号需要取决于具体使用该view时的语句顺序。因此在创建view中应该避免使用hint指定查询块的功能，否则行为不可控。

```
gaussdb=# create view v1 as select/*+ no_expand */ c1 from t1 where c1 in (select /*+ no_expand */ c1 from t2 where t2.c3=4 );
CREATE VIEW
gaussdb=# explain (blockname on,costs off) select  * from v1;
 id |                operation                 | Query Block
----+------------------------------------------+-------------
  1 | ->  Seq Scan on t1@"sel$2"               | sel$2
  2 |   ->  Seq Scan on t2@"sel$3"  [1, SubPlan 1] | sel$3
(2 rows)

 Predicate Information (identified by plan id)
-----------------------------------------------
   1 --Seq Scan on t1@"sel$2"
         Filter: (hashed SubPlan 1)
   2 --Seq Scan on t2@"sel$3"
         Filter: (c3 = 4)
(4 rows)
```

此时v1中的语句分属于sel$2和sel$3。

5. 部分Hint无法只能在最外层生效，且不支持通过@queryblock方式指定，具体参见各自Hint的语法说明。

6.9.3 Hint 可以指定表的查询块名和 schema 名

功能描述

由于在一个查询中，允许在不同查询块使用相同表名，同时不同schema可以有相同表名，因此Hint在指定查询中某个表table时允许指定其所属于的查询块名queryblock和schema名，避免歧义。 该指定方法支持所有需要指定表名的Hint。

语法格式

Hint指定某个表table，通过“.”指定schema，通过“@”queryblock指定查询块名。 schema和queryblock可缺省。

```
[schema.]relname[@queryblock]
```

参数说明

- relname为查询中表table的名字，表有别名时，需要优先使用别名alias，此时relname=alias。当表名中有特殊符号，比如“@”、“.”时，relname需要用"" 括起来，以避免和查询块和schema名的声明重合。比如表名relnametest@1，需要写做 "relnametest@1"。
- schema为表所处的schema，可缺省，缺省时Hint不区分schema对relname进行查找。
- queryblock为表所处的queryblock，可缺省，缺省时Hint不区分queryblock对relname进行查找。

示例

1. sel$2的t1被提升至sel$1，存在t1指代不清的问题。

```
gaussdb=# explain(blockname on,costs off) select /*+ indexscan(t1)*/ * from t1, (select c2 from t1
where c1=1) tt1 where t1.c1 = tt1.c2;
WARNING:  Error hint: IndexScan(t1), relation name "t1" is ambiguous.
...
```

2. 指定t1@sel$2,可以发现在sel$2的t1上进行了indexscan，Index Cond: (c1 = 1)。

```
gaussdb=# explain(blockname on,costs off) select /*+ indexscan(t1@sel$2)*/ * from t1, (select c2 from
t1 where c1=1) tt1 where t1.c1 = tt1.c2;
 id |                   operation                    | Query Block
----+------------------------------------------------+-------------
  1 | ->  Hash Join (2,3)                            | sel$1
  2 |    ->  Seq Scan on t1@"sel$1"                  | sel$1
  3 |    ->  Hash                                    |
  4 |       ->  Index Only Scan using it1 on t1@"sel$2" | sel$1
(4 rows)

 Predicate Information (identified by plan id)
------------------------------------------------
  1 --Hash Join (2,3)
        Hash Cond: (public.t1.c1 = public.t1.c2)
  4 --Index Only Scan using it1 on t1@"sel$2"
        Index Cond: (c1 = 1)
(4 rows)
```


6.9.4 Join 顺序的 Hint

功能描述

指明join的顺序，包括内外表顺序。

语法格式

- 仅指定join顺序，不指定内外表顺序。

```
leading([@queryblock] join_table_list)
```

- 同时指定join顺序和内外表顺序，内外表顺序仅在最外层生效。

```
leading([@queryblock] (join_table_list))
```

参数说明

join_table_list为表示表join顺序的hint字符串，可以包含当前层的任意个表（别名），或对于子查询提升的场景，也可以包含子查询的hint别名，同时任意表可以使用括号指定优先级，表之间使用空格分隔。

@queryblock 见**指定Hint所处于的查询块Queryblock**，可省略，表示在当前查询块生效。

须知

表只能用单个字符串表示，不能带schema。

表如果存在别名，需要优先使用别名来表示该表。

join table list中指定的表需要满足以下要求，否则会报语义错误。

- list中的表必须在当前层或提升的子查询中存在。
- list中的表在当前层或提升的子查询中必须是唯一的。如果不唯一，需要使用不同的别名进行区分。
- 同一个表只能在list里出现一次。
- 如果表存在别名，则list中的表需要使用别名。

例如：

leading(t1 t2 t3 t4 t5)表示：t1、t2、t3、t4、t5先join，五表join顺序及内外表不限。

leading((t1 t2 t3 t4 t5))表示：t1和t2先join，t2做内表；再和t3join，t3做内表；再和t4join，t4做内表；再和t5join，t5做内表。

leading(t1 (t2 t3 t4) t5)表示：t2、t3、t4先join，内外表不限；再和t1，t5join，内外表不限。

leading((t1 (t2 t3 t4) t5))表示：t2、t3、t4先join，内外表不限；在最外层，t1再和t2、t3、t4的join表join，t1为外表，再和t5join，t5为内表。

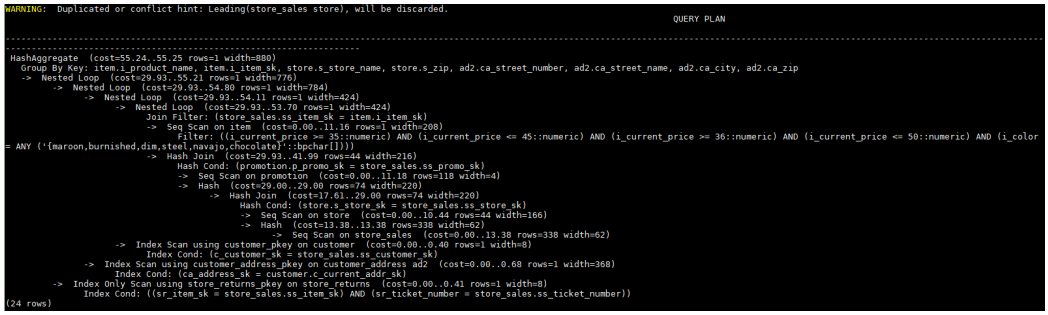
leading((t1 (t2 t3) t4 t5)) leading((t3 t2))表示：t2，t3先join，t2做内表；然后再和t1join，t2，t3的join表做内表；然后再依次跟t4，t5做join，t4，t5做内表。

示例

对**示例**中原语句使用如下hint:

```
explain
select /*+ leading((((((store_sales store) promotion) item) customer) ad2) store_returns) leading((store store_sales))*/ i_product_name product_name ...
```

该hint表示：表之间的join关系是：store_sales和store先join， store_sales做内表，然后依次跟promotion、item, customer、ad2、store_returns做join。生成计划如下所示：



图中计划顶端warning的提示详见**Hint的错误、冲突及告警**的说明。

6.9.5 Join 方式的 Hint

功能描述

指明Join使用的方法，可以为Nested Loop，Hash Join和Merge Join。

语法格式

```
[no] nestloop|hashjoin|mergejoin([@queryblock] table_list)
```

参数说明

- @queryblock 见**指定Hint所处于的查询块Queryblock**，可省略，表示在当前查询块生效。
- **no**表示hint的join方式不使用。
- **table_list**为表示hint表集合的字符串，该字符串中的表与**join_table_list**相同，只是中间不允许出现括号指定join的优先级。

例如：

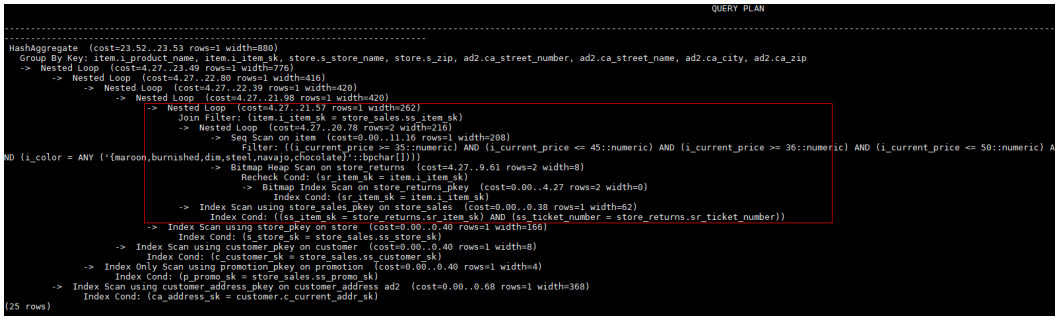
no nestloop(t1 t2 t3)表示：生成t1、t2、t3三表连接计划时，不使用nestloop。三表连接计划可能是t2 t3先join，再跟t1 join，或t1 t2先join，再跟t3 join。此hint只hint最后一次join的join方式，对于两表连接的方法不hint。如果需要，可以单独指定，例如：任意表均不允许nestloop连接，且希望t2 t3先join，则增加hint：no nestloop(t2 t3)。

示例

对**示例**中原语句使用如下hint:

```
explain
select /*+ nestloop(store_sales store_returns item) */ i_product_name product_name ...
```

该hint表示：生成store_sales，store_returns和item三表的结果集时，最后的两表关联使用nestloop。生成计划如下所示：



6.9.6 行数的 Hint

功能描述

指明中间结果集的大小，支持绝对值和相对值的hint。

语法格式

```
rows( [@queryblock] table_list #|+|-|* const)
```

参数说明

- @queryblock 见**指定Hint所处于的查询块Queryblock**，可省略，表示在当前查询块生效。
- **“#”、“+”、“-”、“*”**，进行行数估算hint的四种操作符号。#表示直接使用后面的行数进行hint。“+”、“-”、“*”表示对原来估算的行数进行加、减、乘操作，运算后的行数最小值为1行。table_list为hint对应的单表或多表join结果集，与**Join方式的Hint**中**table_list**相同。
- **const**可以是任意非负数，支持科学计数法。

例如：

rows(t1 #5)表示：指定t1表的结果集为5行。

rows(t1 t2 t3 *1000)表示：指定t1、t2、t3 join完的结果集的行数乘以1000。

建议

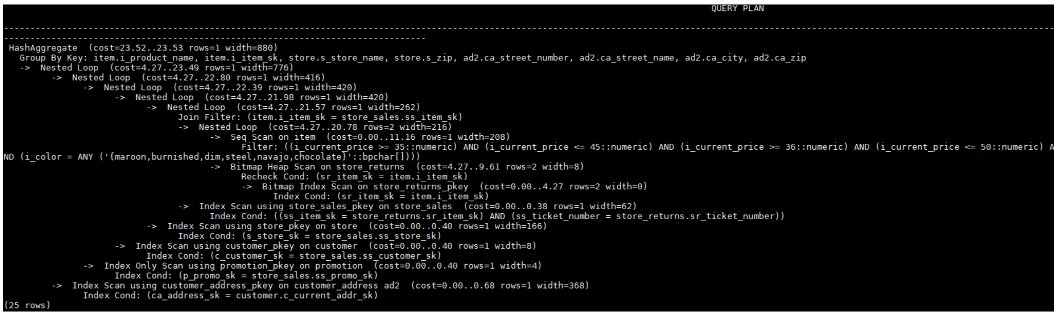
- 推荐使用两个表*的hint。对于两个表的采用*操作符的hint，只要两个表出现在join的两端，都会触发hint。例如：设置hint为rows(t1 t2 * 3)，对于(t1 t3 t4)和(t2 t5 t6)join时，由于t1和t2出现在join的两端，所以其join的结果集也会应用该hint规则乘以3。
- rows hint支持在单表、多表、function table及subquery scan table的结果集上指定hint。

示例

对**示例**中原语句使用如下hint：

```
explain
select /*+ rows(store_sales store_returns *50) */ i_product_name product_name ...
```

该hint表示：store_sales，store_returns关联的结果集估算行数在原估算行数基础上乘以50。生成计划如下所示：



```
                                                QUERY PLAN
--------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=23.52..23.53 rows=1 width=880)
   Group By Key: item.i_product_name, item.i_item_sk, store.s_store_name, store.s_zip, ad2.ca_street_number, ad2.ca_street_name, ad2.ca_city, ad2.ca_zip
   ->  Nested Loop  (cost=4.27..23.49 rows=1 width=776)
         ->  Nested Loop  (cost=4.27..22.80 rows=1 width=416)
               ->  Nested Loop  (cost=4.27..22.39 rows=1 width=420)
                     ->  Nested Loop  (cost=4.27..21.98 rows=1 width=420)
                           ->  Nested Loop  (cost=4.27..21.57 rows=1 width=262)
                                 Join Filter: (item.i_item_sk = store_sales.ss_item_sk)
                                 ->  Nested Loop  (cost=4.27..20.78 rows=2 width=216)
                                       ->  Seq Scan on item  (cost=0.00..11.16 rows=1 width=208)
                                             Filter: ((i_current_price >= 35::numeric) AND (i_current_price <= 45::numeric) AND (i_current_price >= 36::numeric) AND (i_current_price <= 50::numeric) A
ND (i_color = ANY ('{maroon,burnished,dim,steel,navajo,chocolate}'::bpchar[])))
                                       ->  Bitmap Heap Scan on store_returns  (cost=4.27..9.61 rows=2 width=8)
                                             Recheck Cond: (sr_item_sk = item.i_item_sk)
                                             ->  Bitmap Index Scan on store_returns_pkey  (cost=0.00..4.27 rows=2 width=0)
                                                   Index Cond: (sr_item_sk = item.i_item_sk)
                                 ->  Index Scan using store_sales_pkey on store_sales  (cost=0.00..0.38 rows=1 width=62)
                                       Index Cond: ((ss_item_sk = store_returns.sr_item_sk) AND (ss_ticket_number = store_returns.sr_ticket_number))
                           ->  Index Scan using store_pkey on store  (cost=0.00..0.40 rows=1 width=166)
                                 Index Cond: (s_store_sk = store_sales.ss_store_sk)
                     ->  Index Scan using customer_pkey on customer  (cost=0.00..0.40 rows=1 width=8)
                           Index Cond: (c_customer_sk = store_sales.ss_customer_sk)
               ->  Index Only Scan using promotion_pkey on promotion  (cost=0.00..0.40 rows=1 width=4)
                     Index Cond: (p_promo_sk = store_sales.ss_promo_sk)
         ->  Index Scan using customer_address_pkey on customer_address ad2  (cost=0.00..0.68 rows=1 width=368)
               Index Cond: (ca_address_sk = customer.c_current_addr_sk)
(25 rows)
```

6.9.7 Stream 方式的 Hint

功能描述

在并行的执行计划中，指定stream的使用方法，可以为broadcast或者redistribute，表示对数据进行广播或者重分布。

语法格式

```
[no] broadcast|redistribute( [@queryblock] table_list)
```

参数说明

- @queryblock 见**指定Hint所处于的查询块Queryblock**，可省略，表示在当前查询块生效。
- broadcast和redistribute。
  - no表示hint的stream方式不使用。
  - table_list为进行stream操作的单表或多表，多个表之间使用空格分隔。例如：broadcast(t1)，broadcast(t1 t2)。

示例

```
CREATE TABLE stream_t1(a int, b int);
INSERT INTO stream_t1 VALUES(generate_series(1, 1000000), generate_series(1, 1000000));
ANALYZE stream_t1;
CREATE TABLE stream_t2(a int, b int);
INSERT INTO stream_t2 VALUES(generate_series(1, 10000), generate_series(1, 10000));
ANALYZE stream_t2;
SET query_dop = 4;
EXPLAIN (COSTS OFF) SELECT/*+ BROADCAST(stream_t1)*/ * FROM stream_t1 JOIN stream_t2 ON
(stream_t1.a = stream_t2.a);
                    QUERY PLAN
---------------------------------------------------------------
 Streaming(type: LOCAL GATHER dop: 1/4)
   ->  Hash Join
         Hash Cond: (stream_t1.a = stream_t2.a)
         ->  Streaming(type: BROADCAST dop: 4/4)
               ->  Seq Scan on stream_t1
         ->  Hash
               ->  Streaming(type: LOCAL ROUNDROBIN dop: 4/1)
                     ->  Seq Scan on stream_t2
(8 rows)
-- 指定stream_t2进行broadcast的执行计划
EXPLAIN (COSTS OFF) SELECT/*+ BROADCAST(stream_t2)*/ * FROM stream_t1 JOIN stream_t2 ON
(stream_t1.a = stream_t2.a);
```

```
                      QUERY PLAN
-----------------------------------------------------------
 Streaming(type: LOCAL GATHER dop: 1/4)
   ->  Hash Join
         Hash Cond: (stream_t1.a = stream_t2.a)
         ->  Seq Scan on stream_t1
         ->  Hash
               ->  Streaming(type: BROADCAST dop: 4/1)
                     ->  Seq Scan on stream_t2
(7 rows)
-- 表示使用将stream_t2的数据进行broadcast之后再和stream_t1进行join。样例开启了4并发，此时broadcast将广播一张表至其他线程进行并行hash join，由于stream_t2表大小比stream_t1表小，广播t2可以带来更低的性能开销。
```

注：只有在生成并行的执行计划的时候，stream hint才会生效。

6.9.8 Scan 方式的 Hint

功能描述

指明scan使用的方法，可以是tablescan、indexscan和indexonlyscan。

语法格式

```
[no] tablescan|indexscan|indexonlyscan( [@queryblock] table [index])
```

参数说明

- **no**表示hint的scan方式不使用。
- @queryblock 见**指定Hint所处于的查询块Queryblock**，可省略，表示在当前查询块生效。
- **table**表示hint指定的表，只能指定一个表，如果表存在别名应优先使用别名进行hint。
- **index**表示使用indexscan或indexonlyscan的hint时，指定的索引名称，当前只能指定一个。

说明

- 对于indexscan或indexonlyscan，只有hint的索引属于hint的表时，才能使用该hint。
- scan hint支持在行存表、hdfs内外表、、子查询表上指定。对于hdfs内表，由主表和delta表组成，delta表对用户不可见，故hint仅作用在主表上。
- indexonlyscan的计划也能够被indexscan的hint产生，但indexonly的hint只能产生indexonly的计划。
- indexscan兼容indexonlyscan时可能带来一些计划变化，使用cost_model_version进行逃生，通过cost_model_version可以控制是否兼容，在大于2或者等于0时生效。

示例

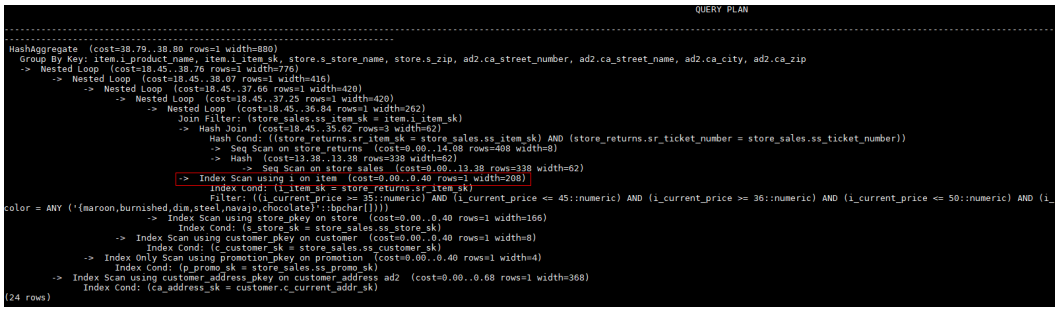
为了hint使用索引扫描，需要首先在表item的i_item_sk列上创建索引，名称为i。

```
create index i on item(i_item_sk);
```

对**示例**中原语句使用如下hint:

```
explain
select /*+ indexscan(item i) */ i_product_name product_name ...
```

该hint表示：item表使用索引i进行扫描。生成计划如下所示：



在集中式环境下，使用gsi hint会报不支持告警。示例如下。

```
-- 创建表
CREATE TABLE gsi_test(a int, b int);
CREATE INDEX gsi_test_idx on gsi_test(a);
-- 使用索引
EXPLAIN SELECT /*+  GSI(gsi_test gsi_test_idx) */ * FROM gsi_test where b = 1;
WARNING:  LINE 1: unsupport distributed hint at 'gsi'
                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on gsi_test  (cost=0.00..36.86 rows=11 width=8)
   Filter: (b = 1)
(2 rows)
```

在集中式环境下，使用gsitable hint会报不支持告警。示例如下。

```
EXPLAIN SELECT /*+ GSITABLE(gsi_test gsi_test_idx) */ * FROM gsi_test where b = 1;
WARNING:  LINE 1: unsupport distributed hint at 'gsitable'
                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on gsi_test  (cost=0.00..36.86 rows=11 width=8)
   Filter: (b = 1)
(2 rows)
```

6.9.9 子链接块名的 hint

功能描述

指明子链接块的名称。

语法格式

```
blockname ( [@queryblock] table)
```

参数说明

- @queryblock 见**指定Hint所处于的查询块Queryblock**，可省略，表示在当前查询块生效。
- **table**表示为该子链接块hint的别名的名称。

说明

- **blockname hint**仅在对应的子链接块没有提升时才会被上层查询使用。目前支持的子链接提升包括IN子链接提升、EXISTS子链接提升和包含Agg等值相关子链接提升。该hint通常会和前面章节提到的hint联合使用。
- 对于FROM关键字后的子查询，则需要使用子查询的别名进行hint，blockname hint不会被用到。
- 如果子链接中含有多个表，则提升后这些表可与外层表以任意优化顺序连接，hint也不会被用到。

示例

```
explain select /*+nestloop(store_sales tt) */ * from store_sales where ss_item_sk in (select /*
+blockname(tt)*/ i_item_sk from item group by 1);
```

该hint表示：子链接的别名为tt，提升后与上层的store_sales表关联时使用nestloop。生成计划如下所示：

```
                                    QUERY PLAN
-----------------------------------------------------------------------------------------
 Nested Loop  (cost=10.53..68.39 rows=169 width=212)
   ->  HashAggregate  (cost=10.53..10.95 rows=42 width=4)
         Group By Key: item.i_item_sk
         ->  Seq Scan on item  (cost=0.00..10.42 rows=42 width=4)
   ->  Index Scan using store_sales_pkey on store_sales  (cost=0.00..1.34 rows=2 width=212)
         Index Cond: (ss_item_sk = item.i_item_sk)
(6 rows)
```

注意

当blockname的hint使用@queryblock进行指定，而不是在当前查询块直接生效时,比如blockname(@sel$2 new_qb_name)。 其他Hint无法通过@new_qb_name进行指定。此时new_qb_name只作为子链接的名字，可以使用Hint进行运算指定。

- 通过blockname(@sel$2 bn2)以@sel$2的方式进行块名bn2的指定，此时TableScan(@bn2 t2)无法通过@bn2找到该queryblock，而得通过@sel$2的方式进行指定。bn3使用Hint blockname(bn3)在当前查询块直接生效，改变默认查询块的名字，因此tablescan(@bn3 t3@bn3)可以通过@bn3进行指定。

  ```
  gaussdb=# explain select /*+ blockname(@sel$2 bn2) tablescan(@bn2 t2) tablescan(@sel$2 t2@bn2)
  indexscan(@sel$2 t2@sel$2) tablescan(@bn3 t3@bn3)*/ c2 from t1 where c1 in ( select /*+ */t2.c1
  from t2 where t2.c2 = 1 group by 1) and c3 in ( select /*+ blockname(bn3)*/t3.c3 from t3 where t3.c2
  = 1 group by 1);
  WARNING:  hint: TableScan(@bn2 t2) does not match any query block
  WARNING:  Error hint: TableScan(@"sel$2" t2@bn2), relation name "t2@bn2" is not found.
  ```

- 通过blockname(@sel$2 bn2)以@sel$2的方式进行子链接块名bn2的指定。当该子链接被提升时，可以通过hashjoin(t1 bn2)对提升后的子链接的运算进行指定。

  ```
  gaussdb=# explain select /*+ blockname(@sel$2 bn2) hashjoin(t1 bn2) nestloop(t1 bn3) nestloop(t1
  sel$3)*/ c2 from t1 where c1 in ( select /*+ */t2.c1 from t2 where t2.c2 = 1 group by 1)  and c3 in
  ( select /*+ blockname(bn3)*/t3.c3 from t3 where t3.c2 = 1 group by 1);
  WARNING:  Duplicated or conflict hint: NestLoop(t1 "sel$3"), will be discarded.
  ```

6.9.10 Hint 的错误、冲突及告警

Plan Hint的结果会体现在计划的变化上，可以通过explain来查看变化。

Hint中的错误不会影响语句的执行，只是不能生效，该错误会根据语句类型以不同方式提示用户。对于explain语句，hint的错误会以warning形式显示在界面上，对于非explain语句，会以debug1级别日志显示在日志中，关键字为PLANHINT。

hint的错误分为以下类型：

- 语法错误

  语法规则树归约失败，会报错，指出出错的位置。

  例如：hint关键字错误，leading hint或join hint指定2个表以下，其它hint未指定表等。一旦发现语法错误，则立即终止hint的解析，所以此时只有错误前面的解析完的hint有效。

  例如：

  ```
  leading((t1 t2)) nestloop(t1) rows(t1 t2 #10)
  ```


nestloop(t1)存在语法错误，则终止解析，可用hint只有之前解析的leading((t1 t2))。

- 语义错误
  - 表不存在，存在多个，或在leading或join中出现多次，均会报语义错误。
  - scanhint中的index不存在，会报语义错误。
  - 另外，如果子查询提升后，同一层出现多个名称相同的表，且其中某个表需要被hint，hint会存在歧义，无法使用，需要为相同表增加别名规避。
- hint重复或冲突

  如果存在hint重复或冲突，只有第一个hint生效，其它hint均会失效，会给出提示。
  - hint重复是指，hint的方法及表名均相同。例如：nestloop(t1 t2) nestloop(t1 t2)。
  - hint冲突是指，table list一样的hint，存在不一样的hint，hint的冲突仅对于每一类hint方法检测冲突。

    例如：nestloop (t1 t2) hashjoin (t1 t2)，则后面与前面冲突，此时hashjoin的hint失效。注意：nestloop(t1 t2)和no mergejoin(t1 t2)不冲突。

    须知

    leading hint中的多个表会进行拆解。例如：leading ((t1 t2 t3))会拆解成：leading((t1 t2)) leading(((t1 t2) t3))，此时如果存在leading((t2 t1))，则两者冲突，后面的会被丢弃。（例外：指定内外表的hint若与不指定内外表的hint重复，则始终丢弃不指定内外表的hint。）
- 子链接提升后hint失效

  子链接提升后的hint失效，会给出提示。通常出现在子链接中存在多个表连接的场景。提升后，子链接中的多个表不再作为一个整体出现在join中。
- hint未被使用
  - 非等值join使用hashjoin hint或mergejoin hint。
  - 不包含索引的表使用indexscan hint或indexonlyscan hint。
  - 通常只有在索引列上使用过滤条件才会生成相应的索引路径，全表扫描将不会使用索引，因此使用indexscan hint或indexonlyscan hint将不会使用。
  - indexonlyscan只有输出和谓词条件列仅包含索引列才会使用，否则指定时hint不会被使用。
  - 多个表存在等值连接时，仅尝试有等值连接条件的表的连接，此时没有关联条件的表之间的路径将不会生成，所以指定相应的leading，join，rows hint将不使用，例如：t1 t2 t3表join，t1和t2, t2和t3有等值连接条件，则t1和t3不会优先连接，leading(t1 t3)不会被使用。
  - 如果子链接未被提升，则blockname hint不会被使用。

6.9.11 优化器 GUC 参数的 Hint

功能描述

设置本次查询执行内生效的查询优化相关GUC参数。hint的推荐使用场景可以参考各guc参数的说明，此处不作赘述。

语法格式

```
set( [@queryblock] param value)
```

参数说明

- @queryblock 见**指定Hint所处于的查询块Queryblock**，可省略，表示在当前查询块生效，该hint只在指定为最外层的queryblock时才会生效。
- **param**表示参数名。
- **value**表示参数的取值。
- 目前支持使用Hint设置生效的参数有
  - 布尔类：

    enable_bitmapscan、enable_hashagg、enable_hashjoin、enable_indexscan、enable_indexonlyscan、enable_material、enable_mergejoin、enable_nestloop、enable_index_nestloop、enable_seqscan、enable_sort、enable_tidscan、partition_iterator_elimination、partition_page_estimation、enable_functional_dependency、var_eq_const_selectivity、enable_inner_unique_opt
  - 整型类：

    query_dop
  - 浮点类：

    cost_weight_index、default_limit_rows、seq_page_cost、random_page_cost、cpu_tuple_cost、cpu_index_tuple_cost、cpu_operator_cost、effective_cache_size

说明

- 设置不在白名单中的参数，参数取值不合法，或hint语法错误时，不会影响查询执行的正确性。使用explain(verbose on)执行可以看到hint解析错误的报错提示。
- GUC参数的hint只在最外层查询生效，子查询内的GUC参数hint不生效。
- 视图定义内的GUC参数hint不生效。
- CREATE TABLE ... AS ... 查询最外层的GUC参数hint可以生效。

6.9.12 Custom Plan 和 Generic Plan 选择的 Hint

功能描述

对于以PBE方式执行的查询语句和DML语句，优化器会基于规则、代价、参数等因素选择生成Custom Plan或Generic Plan执行。用户可以通过use_cplan/use_gplan的hint指定使用哪种计划执行方式。

语法格式

- 指定使用Custom Plan：
  ```
  use_cplan
  ```
- 指定使用Generic Plan：
  ```
  use_gplan
  ```


说明

- 对于非PBE方式执行的SQL语句，设置本hint不会影响执行方式。
- 本Hint的优先级仅高于基于代价的选择和plan_cache_mode参数，即plan_cache_mode无法强制选择执行方式的语句本hint也无法生效。

示例

强制使用Custom Plan

```
create table t (a int, b int, c int);
prepare p as select /*+ use_cplan */ * from t where a = $1;
explain execute p(1);
```

计划如下。可以看到过滤条件为入参的实际值，即此计划为Custom Plan。

```
                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on t  (cost=0.00..34.31 rows=10 width=12)
   Filter: (a = 1)
(2 rows)
```

强制使用Generic Plan

```
deallocate p;
prepare p as select /*+ use_gplan */ * from t where a = $1;
explain execute p(1);
```

计划如下。可以看到过滤条件为待填充的入参，即此计划为Generic Plan。

```
                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on t  (cost=0.00..34.31 rows=10 width=12)
   Filter: (a = $1)
(2 rows)
```

6.9.13 指定子查询不展开的 Hint

功能描述

数据库在对查询进行逻辑优化时通常会将可以提升的子查询提升到上层来避免嵌套执行，但对于某些本身选择率较低且可以使用索引过滤访问页面的子查询，嵌套执行不会导致性能下降过多，而提升之后扩大了查询路径的搜索范围，可能导致性能变差。对于此类情况，可以使用no_expand Hint进行调试。大多数情况下不建议使用此hint。

语法格式

```
no_expand[(@queryblock)]
```

参数说明

[(@queryblock)] 见**指定Hint所处于的查询块Queryblock**，可省略，表示在当前查询块生效，当不指定时，no_expand没有括号"()"。

示例

正常的查询执行：

```
explain select * from t1 where t1.c1 in (select t2.c1 from t2);
```

计划：

```
                                QUERY PLAN
--------------------------------------------------------------------------
 Hash Join  (cost=38.81..92.58 rows=972 width=12)
   Hash Cond: (t1.c1 = t2.c1)
   ->  Seq Scan on t1  (cost=0.00..29.45 rows=1945 width=12)
   ->  Hash  (cost=36.31..36.31 rows=200 width=4)
         ->  HashAggregate  (cost=34.31..36.31 rows=200 width=4)
               Group By Key: t2.c1
               ->  Seq Scan on t2  (cost=0.00..29.45 rows=1945 width=4)
(7 rows)
```

加入no_expand：

```
explain select * from t1 where t1.c1 in (select /*+ no_expand*/ t2.c1 from t2);
```

计划：

```
                          QUERY PLAN
---------------------------------------------------------------
 Seq Scan on t1  (cost=34.31..68.62 rows=972 width=12)
   Filter: (hashed SubPlan 1)
   SubPlan 1
     ->  Seq Scan on t2  (cost=0.00..29.45 rows=1945 width=4)
(4 rows)
```

6.9.14 指定不使用全局计划缓存的 Hint

功能描述

全局计划缓存打开时，可以通过no_gpc Hint来强制单个查询语句不在全局共享计划缓存，只保留会话生命周期的计划缓存。

语法格式

```
no_gpc
```

说明

本参数仅在enable_global_plancache=on时对PBE执行的语句生效。

示例

```
gaussdb=#  deallocate all;
DEALLOCATE ALL
gaussdb=#  prepare p1 as insert /*+ no_gpc*/ into t1 select c1,c2 from t2 where c1=$1;
PREPARE
gaussdb=#  execute p1(3);
INSERT 0 1
gaussdb=#  select * from dbe_perf.global_plancache_status where schema_name='public' order by 1,2;
 nodename | query | refcount | valid | databaseid | schema_name | params_num | func_id | pkg_id | stmt_id
----------+-------+----------+-------+------------+-------------+------------+---------+--------+---------
(0 rows)
```


dbe_perf.global_plancache_status视图中无结果即没有计划被全局缓存。

6.9.15 同层参数化路径的 Hint

功能描述

通过predpush_same_level、nestloop_index Hint来指定同层表或物化视图之间参数化路径生成。

语法格式

```
predpush_same_level(src, dest)
predpush_same_level(src1 src2 ..., dest)
[no] nestloop_index([@queryblock] dest[, index_list]) -- 索引方式
[no] nestloop_index([@queryblock] dest[,(src1 src2 ...)]) -- 表名方式
```

说明

predpush_same_level参数仅在rewrite_rule中的predpushforce选项打开时生效。

nestloop_index对rewrite_rule不做要求。

参数说明

- **no**表示hint的参数化路径方式不使用。
- @queryblock 见**指定Hint所处于的查询块Queryblock**，可省略，表示在当前查询块生效。
- dest为参数化路径的目标表，即索引所在的表。
- src为参数路径的参数表。
- index_list为参数化路径使用的索引序列，为空格隔开的字符串。

示例

1. nestloop_index示例：

- 在t1表上传入t2,t3表的t2.c1和t3.c2进行索引扫描（参数化路径）：

```
gaussdb=# explain (costs off) select /*+nestloop_index(t1,(t2 t3)) */* from t1,t2,t3 where t1.c1 = t2.c1
and t1.c2 = t3.c2;
                        QUERY PLAN
-----------------------------------------------------------
 Nested Loop
   ->  Seq Scan on t3
   ->  Nested Loop
         ->  Seq Scan on t2
         ->  Index Scan using it1 on t1
               Index Cond: ((c1 = t2.c1) AND (c2 = t3.c2))
(6 rows)
```

- 在t1表上的it1上进行索引扫描（参数化路径）：

```
gaussdb=# explain (costs off) select /*+NestLoop_Index(t1,it1) */* from t1,t2 where t1.c1 = t2.c1;
            QUERY PLAN
---------------------------------
 Nested Loop
   ->  Seq Scan on t2
   ->  Index Scan using it1 on t1
         Index Cond: (c1 = t2.c1)
(4 rows)
```

2.predpush_same_level示例：

- 准备参数：

```
gaussdb=# set rewrite_rule = 'predpushforce';
SET
```

- 执行语句查看计划：

```
gaussdb=# explain select * from t1, t2 where t1.c1 = t2.c1;
                           QUERY PLAN
-----------------------------------------------------------------

 Hash Join  (cost=53.76..301.54 rows=18915 width=24)
   Hash Cond: (t1.c1 = t2.c1)
   ->  Seq Scan on t1  (cost=0.00..29.45 rows=1945 width=12)
   ->  Hash  (cost=29.45..29.45 rows=1945 width=12)
         ->  Seq Scan on t2  (cost=0.00..29.45 rows=1945 width=12)
(5 rows)
```

- 可以看到t1.c1 = t2.c1条件过滤在Join上面，此时可以通过predpush_same_level(t1, t2)将条件下推至t2的扫描算子上：

```
gaussdb=# explain select /*+predpush_same_level(t1, t2)*/ * from t1, t2 where t1.c1 = t2.c1;
                            QUERY PLAN
------------------------------------------------------------------

 Nested Loop  (cost=0.00..1143.20 rows=18915 width=24)
   ->  Seq Scan on t1  (cost=0.00..29.45 rows=1945 width=12)
   ->  Index Scan using it2 on t2  (cost=0.00..0.47 rows=10 width=12)
         Index Cond: (c1 = t1.c1)
(4 rows)
```

须知

- 可以指定多个src，但是所有的src必须在同一个条件中。
- 如果指定的src和dest条件不存在，或该条件不符合参数化路径要求，则本hint不生效。
- 如果dest扫描算子上存在stream算子，则本hint不生效。

6.9.16 设置慢 SQL 管控规则的 Hint

功能描述

针对想要进行执行时间/资源管控的SQL语句，设置其被标记为慢SQL的执行时间，最大执行时间，最大IOPS上限。

语法格式

```
wlmrule("time_limit,max_execute_time,max_iops")
```

说明

本参数仅在enable_thread_pool=on时对非sysadmin/monitoradmin用户执行的select类型的语句生效。

- time_limit：SQL语句被标记为慢SQL的执行时长，取值为0-INT_MAX。
- max_execute_time：SQL语句的最大执行时间，执行时间超过该时长后被强制cancel退出，取值为0-INT_MAX。当max_execute_time小于或等于time_limit时，该规则不生效。
- max_iops：SQL语句被标记为慢SQL后最大iops上限，仅在use_workload_manager=on时生效。iops限制采用逻辑IO管控，iops定义请参考io_control_unit定义。取值范围为：Low，Medium，High，None，0-INT_MAX。

示例

```
select /*+ wlmrule("100,500,1") */ * from t2 order by b limit 1;
```

表示指定当前语句被标记为慢SQL的执行时长为100ms，最大执行时间为500ms，最大iops上限为1。

6.9.17 自适应计划选择的 Hint

功能描述

对于以PBE方式执行的查询语句和DML语句，用户可以通过在查询中加choose_adaptive_gplan hint触发自适应计划选择。

语法格式

针对查询开启自适应计划选择：

```
choose_adaptive_gplan
```

说明

- 对于非PBE方式执行的SQL语句，设置本hint不会影响执行方式。
- 本Hint的生效依赖GUC参数enable_cachedplan_mgr开启，否则不生效。

示例

```
prepare k as select /*+ choose_adaptive_gplan */ * from t1 where c1=$1 and c2=$2 and c3=$3 and c4=$4;
```

6.9.18 为子计划结果进行物化的 Hint

功能描述

为子计划结果进行物化，暂存查询记录。只在insert语句应用。

在使用INSERT INTO … SELECT语句插入大量数据且有多行重复值时，因索引需多次对比而导致执行时间过长。使用此HINT对子计划的结果进行物化，暂存查询记录，减少索引比较次数，缩短语句执行时间。

语法格式

```
material_subplan
```

示例

建表并插入数据：

```
create table test_mt_sub(a int, b int) with(storage_type = ustore);
create index on test_mt_sub(a);
create table test_src_mt_sub(a int, b int);
insert into test_src_mt_sub values(generate_series(1,10), generate_series(1,100000));
```

正常的insert into...select语句：

```
insert into test_mt_sub select /*+ nestloop(test_src_mt_sub t1)*/ * from test_src_mt_sub where
notexists(select 1 from test_mt_sub t1 where t1.a = test_src_mt_sub.a);
```

执行计划：

```
                              QUERY PLAN
-----------------------------------------------------------------------
```



```
 Insert on test_mt_sub
   ->  Nested Loop Anti Join
         ->  Seq Scan on test_src_mt_sub
         ->  Index Only Scan using test_mt_sub_a_idx on test_mt_sub t1
               Index Cond: (a = test_src_mt_sub.a)
(5 rows)
```

使用material_subplan hint 算子：

```
insert /*+ material_subplan*/ into test_mt_sub select /*+ nestloop(test_src_mt_sub t1)*/ * from
test_src_mt_sub where not exists(select 1 from test_mt_sub t1 where t1.a = test_src_mt_sub.a);
```

执行计划为：

```
                              QUERY PLAN
-------------------------------------------------------------------------------
 Insert on test_mt_sub
   ->  Materialize
         ->  Nested Loop Anti Join
               ->  Seq Scan on test_src_mt_sub
               ->  Index Only Scan using test_mt_sub_a_idx on test_mt_sub t1
                     Index Cond: (a = test_src_mt_sub.a)
(6 rows)
```

6.9.19 支持 bitmapscan 的 hint

功能描述

支持在目标表上使用指定的索引生成bitmapscan路径，在原优化器可生成路径的基础上选中符合HINT的路径。

语法格式

```
[no] bitmapscan([@queryblock] table [index_list])
```

参数说明：

- **no**表示hint的scan方式不使用。
- @queryblock 见**指定Hint所处于的查询块Queryblock**，可省略，表示在当前查询块生效。
- table为bitmapscan的目标表。
- index_list为bitmapscan使用的索引。

示例

```
gaussdb=# explain(costs off)  select /*+ BitmapScan(t1 it1 it3)*/* from t1 where  (t1.c1 = 5 or t1.c2=6) or
(t1.c3=3 or t1.c2=7);
                          QUERY PLAN
------------------------------------------------------------
 Bitmap Heap Scan on t1
   Recheck Cond: ((c1 = 5) OR (c2 = 6) OR (c3 = 3) OR (c2 = 7))
   ->  BitmapOr
         ->  Bitmap Index Scan on it1
               Index Cond: (c1 = 5)
         ->  Bitmap Index Scan on it3
               Index Cond: (c2 = 6)
         ->  Bitmap Index Scan on it3
               Index Cond: (c3 = 3)
         ->  Bitmap Index Scan on it3
               Index Cond: (c2 = 7)
(11 rows)
```


说明

bitmapscan仅会根据已有的index路径组合bitmapscan路径时优先选择符合要求的路径，因为索引路径构造空间巨大，优化器存在剪枝，若参与的index路径无法生成，则无法构造。

6.9.20 连接时内表物化的 Hint

功能描述

实现在指定连接的inner表时，对内表进行物化。

语法格式

```
[no] materialize_inner([@queryblock] inner_table_list)
```

参数说明

- **no**表示hint的物化方式不使用。
- @queryblock 见**指定Hint所处于的查询块Queryblock**，可省略，表示在当前查询块生效。
- inner_table_list: 执行连接操作时，希望被物化的内表序列，为空格隔开的字符串。

示例

t1表作为内表物化，(t1 t2)的结果作为连接的内表被物化。

```
gaussdb=# explain (costs off) select /*+materialize_inner(t1) materialize_inner(t1 t2)*/ * from t1,t2,t3
where  t1.c3 = t2.c3 and t2.c2=t3.c2 and t1.c2=5;
                        QUERY PLAN
-----------------------------------------------------------
 Nested Loop
   Join Filter: (t2.c2 = t3.c2)
   ->  Seq Scan on t3
   ->  Materialize
         ->  Nested Loop
               Join Filter: (t1.c3 = t2.c3)
               ->  Seq Scan on t2
               ->  Materialize
                     ->  Bitmap Heap Scan on t1
                           Recheck Cond: (c2 = 5)
                           ->  Bitmap Index Scan on it3
                                 Index Cond: (c2 = 5)
(12 rows)
```

6.9.21 指定 agg 算法的 Hint

功能描述

在进行agg算法时可以指定agg的方法。

语法格式

```
use_hash_agg[(@queryblock)], use_sort_agg[(@queryblock)]
```


参数说明

- @queryblock 见**指定Hint所处于的查询块Queryblock**，可省略，表示在当前查询块生效，当不指定时，hint没有括号"()"。

示例

1. 使用hash聚集。

```
gaussdb=# explain (costs off) select c1 from t2 where c1 in( select /*+ use_hash_agg */ t1.c1 from t1,t3 where t1.c1=t3.c1 group by 1);
                 QUERY PLAN
---------------------------------------------
 Hash Semi Join
   Hash Cond: (t2.c1 = t1.c1)
   ->  Seq Scan on t2
   ->  Hash
         ->  HashAggregate
               Group By Key: t1.c1
               ->  Hash Join
                     Hash Cond: (t1.c1 = t3.c1)
                     ->  Seq Scan on t1
                     ->  Hash
                           ->  Seq Scan on t3
(11 rows)
```

2. 使用use_sort_agg聚集，mergejoin有序。

```
gaussdb=# explain (costs off) select c1 from t2 where c1 in( select /*+ use_sort_agg */ t1.c1 from t1,t3 where t1.c1=t3.c1 group by 1);
                    QUERY PLAN
-----------------------------------------------------------
 Hash Semi Join
   Hash Cond: (t2.c1 = t1.c1)
   ->  Seq Scan on t2
   ->  Hash
         ->  Group
               Group By Key: t1.c1
               ->  Merge Join
                     Merge Cond: (t1.c1 = t3.c1)
                     ->  Index Only Scan using it1 on t1
                     ->  Sort
                           Sort Key: t3.c1
                           ->  Seq Scan on t3
(12 rows)
```

6.10 PLAN TRACE 使用介绍

须知

1. 该特性是数据库内核开发人员对慢SQL深度分析使用的特性，不建议非内核开发人员使用。
2. 该特性开启后，会在执行DML期间记录优化器相关信息到系统表中，这样会导致原本的读事务变成了写事务，从而导致要求必须在读事务里执行的函数无法执行，例如函数pg_create_logical_replication_slot等。

使用plan trace特性可以查看查询计划的优化过程。在plan trace中可以看到计划中路径的计算过程、路径的选择与淘汰过程等关键信息，以达到帮助分析慢SQL根因的目的。使用该特性有两种方式。

```
--准备表
CREATE TABLE tb_a(c1 int);
```



```
CREATE TABLE tb_b(c1 int);
CREATE INDEX tb_a_idx_c1 ON tb_a(c1);
CREATE INDEX idx_b ON tb_b(c1);
```

方式一：使用guc参数enable_plan_trace启用plan trace特性，操作步骤如下：

步骤1 打开plan trace guc开关：

```
set enable_plan_trace = on;
```

步骤2 执行业务sql。例如业务sql如下：

```
select * from tb_a a, tb_b b where a.c1 = b.c1 and a.c1=1;
```

步骤3 通过视图gs_my_plan_trace查看自己新生成的plan trace。

```
select * from gs_my_plan_trace order by modifydate limit 1;
```

由于plan trace的记录一般比较大，如果使用gsql连接数据库，建议使用\x命令将gsql的查询结果展示方式改为Expanded方式。

由于plan trace通常比较大，这里只给出本示例执行结果的关键trace的部分片段，如下所示：

片段1：在trace中，可以看到当前执行的sql和计划。

```
query_id      | 69e138356181711a21de1211f639892b
query         | select * from tb_a a, tb_b b where a.c1 = b.c1 and a.c1=1;
unique_sql_id | 3388945134
plan          | Datanode Name: datanode
              | Nested Loop  (cost=0.00..10.75 rows=144 width=8)
              |   ->  Index Only Scan using tb_a_idx_c1 on tb_a a  (cost=0.00..4.46 rows=12 width=4)
              |         Index Cond: (c1 = '***')
              |   ->  Materialize  (cost=0.00..4.52 rows=12 width=4)
              |         ->  Index Only Scan using idx_b on tb_b b  (cost=0.00..4.46 rows=12 width=4)
              |               Index Cond: (c1 = '***')
```

片段2：在trace中，可以看到当前sql使用的关键guc参数。

```
plan_trace    | [key_guc]
              | enable_pbe_optimization=1
              | plan_cache_mode=0
              | random_page_cost=4.000
              | enable_hashjoin=1
              | enable_mergejoin=1
              | enable_nestloop=1
              | enable_seqscan=1
              | effective_cache_size=16385
              | work_mem=65536
              | default_statistics_target=100
              | cost_param=0
              | =[key_guc]=
```

片段3：在trace中，可以看到当前sql的查询计划的路径代价的计算过程。

```
              | [btcostestimate]
              | cal: num_sa_scans,1.000000
              | cal: num_index_tuples=btree_selectivity * index_tuples,48.629630,0.004863,10000.000000
              | cal: num_index_tuples = rint(num_index_tuples / num_sa_scans),49.000000
              |
              | [adt_genericcostestimate]
              | input: loop_count,1.000000 num_index_tuples,49.000000 index_total_pages,37.000000
              | cal: num_sa_scans,1.000000 idx_local_tupls,10000.000000
              | cal: index_selectivity,0.004863
              | cal: num_index_pages=ceil(num_index_tuples/idx_local_tupls * index_total_pages),1.000000
              | cal: num_scans=num_sa_scans * loop_count,1.000000
              | cal: index_total_cost=num_index_pages * spc_random_page_cost,4.000000
              | cal: index_total_cost += num_index_tuples * num_sa_scans * (cpu_index_tuple_cost +
qual_op_cost),4.367500
              | cal: index_total_cost += num_sa_scans * 100.0 * cpu_operator_cost,4.617500
```



```
        | =[adt_genericcostestimate]=
        | =[btcostestimate]=
```

片段4：在trace中，可以看到基表路径的淘汰过程：1. 老路径被淘汰；2. 老路径被淘汰的原因；3. 新路径的相关信息。

```
        | An old path is removed with cost = 881.806443 .. 932.541443;  rows = 49.000000
        | The old path and the comparison results are:
        | {
        |         old pathid=00000005     Cost = NewBetter        |       PathKeys = Equal        |         BMS =
Equal          |         Rows = Equal
        | }
        | A new path is accepted with cost = 284.629750 .. 341.718970;  rows = 49.000000
        | The detail information of the new path:
        | {
        |         HashJoin(1:tb_a  2:tb_b ) pathid=00000011 hasparam=0 rows=49 multiple=1.000000
tuples=0.00 rpages=0.00 ipages=0.00 selec=0.00000000 ml=0 iscost=1 lossy=0 uidx=0)  dop=1
cost=284.63..341.72 hint 0 trace_id=#3##4##11#          clauses: a.id = b.id(norm_
```

片段5：在trace中，可以看到join路径的淘汰过程：1. 老路径被淘汰；2. 老路径淘汰的原因；3. 新路径的相关信息。

```
        | An old path is removed with cost = 4.629750 .. 7591.045220;  rows = 49.000000
        | The old path and the comparison results are:
        | {
        |         old pathid=00000008     Cost = Equal            |       PathKeys = Equal        |         BMS =
Equal          |         Rows = Equal
        |         Small fuzzy factor is used!
        | }
        | A new path is accepted with cost = 4.629750 .. 7566.167720;  rows = 49.000000
        | The detail information of the new path:
        | {
        |         NestLoop(1:tb_a  2:tb_b ) pathid=00000014 hasparam=0 rows=49 multiple=1.000000
tuples=0.00 rpages=0.00 ipages=0.00 selec=0.00000000 ml=0 iscost=1 lossy=0 uidx=0)  dop=1
cost=4.63..7566.17 hint 0 trace_id=#4##13##14#          clauses: a.id = b.id(norm_
        |         Small fuzzy factor is used!
```

trace内容非常多，但都比较通俗易懂，在这里不一一展开，请使用该特性的人员自行分析。

----结束

方式二：使用系统函数gs_plan_trace_watch_sqlid启用plan trace特性，操作步骤如下：

步骤1 通过系统表dbe_perf.statement获得感兴趣的sql 的 unique sql id，例如使用如下sql来获取unique sql id：

```
select * from dbe_perf.statement where query like '%tb_a%';
```

获取到的结果如下中的unique_sql_id字段：

```
node_name           | datanode1
node_id             | 0
user_name           | qiumc
user_id             | 10
unique_sql_id       | 1921680825
query               | select * from tb_a a, tb_b b where a.id=b.id and a.c1=?;
n_calls             | 3
min_elapse_time     | 8880
max_elapse_time     | 12371
total_elapse_time   | 32036
```

步骤2 具有sysadmin权限的用户调用gs_plan_trace_watch_sqlid函数侦听感兴趣的unique sql id。示例如下：

```
select gs_plan_trace_watch_sqlid(1921680825);
```


步骤3 如果感兴趣的unique sql id没有开始生成plan trace，则该unique sql id会被保存在一个内存的列表中，且可以通过函数gs_plan_trace_show_sqlids()来查看当前待收集plan trace的unique sql id列表，例如示例sql如下：

```
select gs_plan_trace_show_sqlids();
```

该sql的执行结果结果如下：

```
-[ RECORD 1 ]-------------+------------
gs_plan_trace_show_sqlids | 1921680825,
```

步骤4 如果此时执行一次sql：

```
select * from tb_a a, tb_b b where a.id=b.id and a.c1=1;
```

同样可以针对该sql生成plan trace记录。

须知

只有具有sysadmin/opradmin/monadmin权限的用户才可以调用gs_plan_trace_watch_sqlid、gs_plan_trace_show_sqlids这两个函数。如果普通用户执行了管理员侦听的unique sql id的sql，则可以使用视图gs_my_plan_trace来查看自己生成的plan trace。

----结束

须知

plan trace通常比较大，需要用户及时清理，否则会占用大量的磁盘空间，用户可以使用gs_plan_trace_delete函数来删除自己生成的plan trace。

例如使用sql：

```
select gs_plan_trace_delete(TIMESTAMPTZ '2023-01-10 17:16:42.652543+08')
```

可以删除当前用户的小于等于2023-01-10 17:16:42.652543+08时间的所有plan trace，从而达到每个用户都可以清理自己plan trace数据的目的。

6.11 使用 SQL PATCH 进行调优

SQL PATCH主要设计给DBA、运维人员及其他需要对SQL进行调优的角色使用，用户通过其他运维视图或定位手段识别到业务语句存在计划不优导致的性能问题时，可以通过创建SQL PATCH对业务语句进行基于Hint的调优。目前支持行数、扫描方式、连接方式、连接顺序、PBE custom/generic计划选择、语句级参数设置、参数化路径的Hint。此外，对于部分由特定语句触发系统内部问题导致系统可服务性受损的语句，在不对业务语句变更的情况下，也可以通过创建用于单点规避的SQL PATCH，对问题场景提前报错处理，避免更大的损失。

特性约束

1. 仅支持针对Unique SQL ID添加补丁，如果存在Unique SQL ID冲突，用于Hint调优的SQL PATCH可能影响性能，但不影响语义正确性。
2. 仅支持不改变SQL语义的Hint作为PATCH，不支持SQL改写。
3. 不支持逻辑备份、恢复。

4. 不支持创建时校验PATCH合法性，如果PATCH的Hint存在语法或语义错误，不影响查询正确执行。
5. 仅初始用户、运维管理员、监控管理员、系统管理员用户有权限执行。
6. 库之间不共享，创建SQL PATCH时需要连接目标库。
7. 配置集中式备机可读时，需要指定主机执行SQL PATCH创建/修改/删除函数调用，备机执行报错。
8. SQL PATCH同步给备机存在一定延迟，待备机回放相关日志后PATCH生效。
9. 限制在存储过程内的SQL PATCH和全局的SQL PATCH不允许同时存在。
10. 使用PREPARE + EXECUTE语法执行的预编译语句执行不支持使用SQL PATCH。
11. SQL PATCH不建议在数据库中长期使用，只应该作为临时规避方法。遇到内核问题所导致的特定语句触发数据库服务不可用问题，以及使用Hint进行调优的场景，需要尽快修改业务或升级内核版本解决问题。并且升级后由于Unique SQL ID生成方法可能变化，可能导致规避方法失效。
12. 当前，除DML语句之外，其他SQL语句（如CREATE TABLE等）的Unique SQL ID是对语句文本直接哈希生成的，所以对于此类语句，SQL PATCH对大小写、空格、换行等敏感，即不同的文本的语句，即使语义相同，仍然需要对应不同的SQL PATCH。对于DML，则同一个SQL PATCH可以对不同入参的语句生效，并且忽略大小写和空格。

示例

SQL PATCH的实现当前基于Unique SQL ID，所以需要打开相关的运维参数才可以生效（enable_resource_track = on，instr_unique_sql_count > 0），Unique SQL ID在WDR报告和慢SQL视图中都可以获取到，在创建SQL PATCH时需要指定Unique SQL ID，对于存储过程内的SQL则需要设置参数 instr_unique_sql_track_type = 'all' 后在dbe_perf.statement_history视图中查询Unique SQL ID。

下面给出简单的使用样例。

场景一：使用SQL PATCH对特定语句进行Hint调优。

```
gaussdb=# create table hint_t1(a int, b int, c int);
CREATE TABLE
gaussdb=# create index on hint_t1(a);
CREATE INDEX
gaussdb=# insert into hint_t1 values(1,1,1);
INSERT 0 1
gaussdb=# set track_stmt_stat_level = 'L1,L1'; --打开FullSQL统计信息
SET
gaussdb=# set explain_perf_mode = normal;
SET
gaussdb=# select * from hint_t1 t1 where t1.a = 1; --执行SQL语句
 a | b | c
---+---+---
 1 | 1 | 1
(1 row)
gaussdb=# \x --切换扩展显示模式，便于观察计划
Expanded display is on.
gaussdb=# select unique_query_id, query, query_plan from dbe_perf.statement_history where query like
'%hint_t1%'; --获取查询计划和Unique SQL ID
-[ RECORD 1 ]---+-----------------------------------------------------------------------------
unique_query_id | 2311517824
query           | select * from hint_t1 t1 where t1.a = ?;
query_plan      | Datanode Name: sgnode
                | Bitmap Heap Scan on hint_t1 t1  (cost=4.33..14.88 rows=10 width=12)
                |   Recheck Cond: (a = '***')
                |   ->  Bitmap Index Scan on hint_t1_a_idx  (cost=0.00..4.33 rows=10 width=0)
                |         Index Cond: (a = '***')
```



```
              |
              |
gaussdb=# \x
Expanded display is off.
gaussdb=# select * from dbe_sql_util.create_hint_sql_patch('patch1', 2311517824, 'indexscan(t1)'); -- 对指定
的Unique SQL ID指定Hint Patch
 create_hint_sql_patch
-----------------------
 t
(1 row)

gaussdb=# explain select * from hint_t1 t1 where t1.a = 1; -- 通过explain可以确认Hint是否生效
NOTICE:  Plan influenced by SQL hint patch
                                QUERY PLAN
-------------------------------------------------------------------------------
 [Bypass]
 Index Scan using hint_t1_a_idx on hint_t1 t1  (cost=0.00..32.43 rows=10 width=12)
   Index Cond: (a = 1)
(3 rows)
gaussdb=# select * from hint_t1 t1 where t1.a = 1; -- 再次执行语句
 a | b | c
---+---+---
 1 | 1 | 1
(1 row)
gaussdb=# \x
Expanded display is on.
gaussdb=# select unique_query_id, query, query_plan from dbe_perf.statement_history where query like
'%hint_t1%'; -- 可以看到新的执行记录计划已改变
-[ RECORD 1 ]---+-----------------------------------------------------------------------------------------------------
unique_query_id | 2311517824
query           | select * from hint_t1 t1 where t1.a = ?;
query_plan      | Datanode Name: sgnode
              | Bitmap Heap Scan on hint_t1 t1  (cost=4.33..15.70 rows=10 p-time=0 p-rows=0 width=12)
              |   Recheck Cond: (a = '***')
              |   ->  Bitmap Index Scan on hint_t1_a_idx  (cost=0.00..4.33 rows=10 p-time=0 p-rows=0 width=0)
              |         Index Cond: (a = '***')
              |
              |
-[ RECORD 2 ]---+-----------------------------------------------------------------------------------------------------
unique_query_id | 2311517824
query           | select * from hint_t1 t1 where t1.a = ?;
query_plan      | Datanode Name: sgnode
              | Index Scan using hint_t1_a_idx on hint_t1 t1  (cost=0.00..8.27 rows=1 p-time=0 p-rows=0
width=12)
              |   Index Cond: (a = '***')
              |
              |
```

场景二：使用SQL PATCH对特定语句进行提前报错规避。

```
gaussdb=# select * from dbe_sql_util.drop_sql_patch('patch1'); -- 删去patch1
 drop_sql_patch
----------------
 t
(1 row)
gaussdb=# select * from dbe_sql_util.create_abort_sql_patch('patch2', 2311517824); 对该语句的Unique SQL
ID创建Abort Patch
 create_abort_sql_patch
------------------------
 t
(1 row)

gaussdb=# select * from hint_t1 t1 where t1.a = 1; -- 再次执行语句会提前报错
ERROR:  Statement 2578396627 canceled by abort patch patch2
```

场景三：针对存储过程内的SQL语句创建SQLpatch。

```
gaussdb=# create table test_proc_patch(a int,b int);
CREATE TABLE
gaussdb=# insert into test_proc_patch values(1,2);
```



```
INSERT 0 1
gaussdb=# create index test_a on test_proc_patch(a);
CREATE INDEX
gaussdb=# create procedure mypro() as num int;
gaussdb$# begin
gaussdb$# select b into num from test_proc_patch where a = 1;
gaussdb$# end;
gaussdb$# /
CREATE PROCEDURE
gaussdb=# set track_stmt_stat_level = 'L0,L1';  -- 打开记录统计信息
SET
gaussdb=# select b from test_proc_patch where a = 1;
 b
---
 2
(1 row)
gaussdb=# call mypro();
 mypro
-------

(1 row)
gaussdb=# set track_stmt_stat_level = 'OFF,L0';    -- 暂时关闭记录统计信息
SET
gaussdb=# select unique_query_id, query, query_plan, parent_unique_sql_id from
dbe_perf.statement_history where query like '%call mypro();%' or query like '%test_proc_patch%';
 unique_query_id |                  query                  |                        query_plan                         |
parent_unique_sql_id
-----------------+-----------------------------------------
+-----------------------------------------------------------------+----------------------
     2859505004 | select b from test_proc_patch where a = ?; | Datanode Name:
sgnode                                  +|                  0
                |                                         | Seq Scan on test_proc_patch  (cost=0.00..1.01 rows=1 width=4)+|
                |                                         |   Filter: (a = '***')                                          +|
                |                                         |                                                               +|
                |                                         |                                                                |
     3460545602 | call mypro();                                | Datanode Name: sgnode
+|                 0
                |                                         | Function Scan on mypro  (cost=0.25..0.26 rows=1 width=4)     +|
                |                                         |                                                               +|
                |                                         |                                                                |
     2859505004 | select b from test_proc_patch where a = ?; | Datanode Name:
sgnode                                  +|                    3460545602
                |                                         | Seq Scan on test_proc_patch  (cost=0.00..1.01 rows=1 width=4)+|
                |                                         |   Filter: (a = '***')                                          +|
                |                                         |                                                               +|
                |                                         |                                                                |
(3 rows)

-- 根据parentid可以调用重载函数限制存储过程内生效
gaussdb=# select * from
dbe_sql_util.create_hint_sql_patch('patch1',2859505004,3460545602,'indexscan(test_proc_patch)');
 create_hint_sql_patch
-----------------------
 t
(1 row)

gaussdb=# select patch_name,unique_sql_id,parent_unique_sql_id,enable,abort,hint_string from
gs_sql_patch where patch_name = 'patch1';  -- 确认SQLpatch记录没有错误
 patch_name | unique_sql_id |  parent_unique_sql_id | enable | abort |       hint_string
------------+---------------+-----------------------+--------+-------+--------------------------
 patch1     |    2859505004 |            3460545602 | t      | f     | indexscan(test_proc_patch)
(1 row)
gaussdb=# set track_stmt_stat_level = 'L0,L1';  -- 打开记录统计信息
gaussdb=# select b from test_proc_patch where a = 1;
 b
---
 2
(1 row)
gaussdb=# call mypro();
```



```
 mypro
-------

(1 row)
gaussdb=# select unique_query_id, query, query_plan, parent_unique_sql_id from
dbe_perf.statement_history where query like '%test_proc_patch%' order by start_time;
 unique_query_id |                 query                 |                                query_plan
| parent_unique_sql_id
-----------------+---------------------------------------
+-------------------------------------------------------------------------------+----------------------
     2859505004 | select b from test_proc_patch where a = ?; | Datanode Name:
sgnode                                          +|                  0
               |                                 | Seq Scan on test_proc_patch  (cost=0.00..1.01 rows=1
width=4)             +|
               |                                 |   Filter: (a = '***')                                  +|
               |                                 |                                                      +|
               |                                 |                                                       |
     2859505004 | select b from test_proc_patch where a = ?; | Datanode Name:
sgnode                                          +|          3460545602
               |                                 | Seq Scan on test_proc_patch  (cost=0.00..1.01 rows=1
width=4)             +|
               |                                 |   Filter: (a = '***')                                  +|
               |                                 |                                                      +|
               |                                 |                                                       |
     2859505004 | select b from test_proc_patch where a = ?; | Datanode Name:
sgnode                                          +|                  0
               |                                 | Seq Scan on test_proc_patch  (cost=0.00..1.01 rows=1
width=4)             +|
               |                                 |   Filter: (a = '***')                                  +|
               |                                 |                                                      +|
               |                                 |                                                       |
     2859505004 | select b from test_proc_patch where a = ?; | Datanode Name:
sgnode                                          +|          3460545602
               |                                 | Index Scan using test_a on test_proc_patch  (cost=0.00..8.27
rows=1 width=4)+|
               |                                 |   Index Cond: (a = '***')                                +|
               |                                 |                                                      +|
               |                                 |                                                       |
(4 rows)
```

相关链接

SQL PATCH相关系统表、接口函数见下表。

表 6-5 SQL PATCH 相关系统表、接口函数介绍

名称		说明
系统表	GS_SQL_PATCH	GS_SQL_PATCH系统表存储所有SQL_PATCH的状态信息。
接口函数 DBE_SQL_UTIL Schema	DBE_SQL_UTIL.create_hint_sql_patch	create_hint_sql_patch是用于创建调优SQL PATCH的接口函数，返回执行是否成功。
	DBE_SQL_UTIL.create_abort_sql_patch	create_abort_sql_patch是用于创建避险SQL PATCH的接口函数，返回执行是否成功。
	DBE_SQL_UTIL.drop_sql_patch	drop_sql_patch是用于在当前建连的CN上删除SQL PATCH的接口函数，返回执行是否成功。

名称		说明
	DBE_SQL_UTIL.enable_sql_patch	enable_sql_patch是用于在当前建连的CN上开启SQL PATCH的接口函数，返回执行是否成功。
	DBE_SQL_UTIL.disable_sql_patch	disable_sql_patch是用于在当前建连的CN上禁用SQL PATCH的接口函数，返回执行是否成功。
	DBE_SQL_UTIL.show_sql_patch	show_sql_patch是用于显示给定patch_name对应的SQL PATCH的接口函数，返回运行结果。
	DBE_SQL_UTIL.create_hint_sql_patch	create_hint_sql_patch是用于创建调优SQL PATCH的接口函数，返回执行是否成功。本函数是原函数的重载函数，支持通过parent_unique_sql_id值限制hint patch的生效范围。
	DBE_SQL_UTIL.create_abort_sql_patch	create_abort_sql_patch是用于创建避险SQL PATCH的接口函数，返回执行是否成功。本函数是原函数的重载函数，支持通过parent_unique_sql_id值限制abort patch的生效范围。

6.12 实际调优案例

6.12.1 案例：调整查询重写 GUC 参数 rewrite_rule

rewrite_rule包含了多个查询重写规则：magicset、uniquecheck、intargetlist、predpush等。下面简要说明其中重要的几个规则的使用场景：

案例环境准备

为了便于规则的使用场景演示，需准备建表语句如下：

```
--清理环境。
DROP SCHEMA IF EXISTS rewrite_rule_guc_test CASCADE;
CREATE SCHEMA rewrite_rule_guc_test;
SET current_schema=rewrite_rule_guc_test;

--创建测试表。
CREATE TABLE t(c1 INT, c2 INT, c3 INT, c4 INT);
CREATE TABLE t1(c1 INT, c2 INT, c3 INT, c4 INT);
CREATE TABLE t2(c1 INT, c2 INT, c3 INT, c4 INT);
```

目标列子查询提升参数 intargetlist

通过将目标列中子查询提升，转为JOIN，往往可以极大提升查询性能。举例如下查询：

```
gaussdb=#  set rewrite_rule='none';
SET
```

```
gaussdb=#  EXPLAIN (verbose on, costs off) SELECT c1,(SELECT avg(c2) FROM t2 WHERE t2.c2=t1.c2)
FROM t1 WHERE t1.c1<100 ORDER BY t1.c2;
                QUERY PLAN
-----------------------------------------------
 Sort
   Output: t1.c1, ((SubPlan 1)), t1.c2
   Sort Key: t1.c2
   ->  Seq Scan on public.t1
         Output: t1.c1, (SubPlan 1), t1.c2
         Filter: (t1.c1 < 100)
         SubPlan 1
           ->  Aggregate
                 Output: avg(t2.c2)
                 ->  Seq Scan on public.t2
                       Output: t2.c1, t2.c2
                       Filter: (t2.c2 = t1.c2)
(12 rows)
```

由于目标列中的相关子查询(select avg(c2) from t2 where t2.c2=t1.c2)无法提升的缘故，导致每扫描t1的一行数据，就会触发子查询的一次执行，效率低下。如果打开intargetlist参数会把子查询提升转为JOIN，来提升查询的性能：

```
gaussdb=#  set rewrite_rule='intargetlist';
SET
gaussdb=# explain (verbose on, costs off) select c1,(select avg(c2) from t2 where t2.c2=t1.c2) from t1
where t1.c1<100 order by t1.c2;
                QUERY PLAN
-----------------------------------------------
 Sort
   Output: t1.c1, (avg(t2.c2)), t1.c2
   Sort Key: t1.c2
   ->  Hash Left Join
         Output: t1.c1, (avg(t2.c2)), t1.c2
         Hash Cond: (t1.c2 = t2.c2)
         ->  Seq Scan on public.t1
               Output: t1.c1, t1.c2
               Filter: (t1.c1 < 100)
         ->  Hash
               Output: (avg(t2.c2)), t2.c2
               ->  HashAggregate
                     Output: avg(t2.c2), t2.c2
                     Group By Key: t2.c2
                     ->  Seq Scan on public.t2
                           Output: t2.c2
(16 rows)
```

提升无 agg 的子查询 uniquecheck

子链接提升需要保证对于每个条件只有一行输出，对于有agg的子查询可以自动提升，对于无agg的子查询如：

```
select t1.c1 from t1 where t1.c1 = (select t2.c1 from t2 where t1.c1=t2.c2) ;
```

重写为：

```
select t1.c1 from t1 join (select t2.c1 from t2 where t2.c1 is not null group by t2.c1(unique check)) tt(c1) on
tt.c1=t1.c1;
```

需注意，上述SQL中的unique check表示t2.c1需要进行检查，非正常SQL表达，该SQL无法直接执行。为了保证语义等价，子查询tt必须保证对于每个group by t2.c1只能有一行输出。打开uniquecheck查询重写参数保证可以提升并且等价，如果在运行时输出了多于一行的数据，就会报错。

```
gaussdb=# set rewrite_rule='uniquecheck';
SET
gaussdb=#  explain verbose select t1.c1 from t1 where t1.c1 = (select t2.c1 from t2 where t1.c1=t2.c1);
                              QUERY PLAN
```



```
-------------------------------------------------------------------------------------
 Hash Join  (cost=43.36..104.40 rows=2149 distinct=[200, 200] width=4)
   Output: t1.c1
   Hash Cond: (t1.c1 = subquery."?column?")
   ->  Seq Scan on public.t1  (cost=0.00..31.49 rows=2149 width=4)
         Output: t1.c1, t1.c2
   ->  Hash  (cost=40.86..40.86 rows=200 width=8)
         Output: subquery."?column?", subquery.c1
         ->  Subquery Scan on subquery  (cost=36.86..40.86 rows=200 width=8)
               Output: subquery."?column?", subquery.c1
               ->  HashAggregate  (cost=36.86..38.86 rows=200 width=4)
                     Output: t2.c1, t2.c1
                     Group By Key: t2.c1
                     Filter: (t2.c1 IS NOT NULL)
                     Unique Check Required
                     ->  Seq Scan on public.t2  (cost=0.00..31.49 rows=2149 width=4)
                           Output: t2.c1
(16 rows)
```

注意：因为分组group by t2.c1 unique check发生在过滤条件tt.c1=t1.c1之前，可能导致原来不报错的查询重写之后报错。举例：

有t1,t2表，其中的数据为：

```
gaussdb=#  select * from t1 order by c2;
 c1 | c2
----+----
  1 |  1
  2 |  2
  3 |  3
(3 rows)
gaussdb=#  select * from t2 order by c2;
 c1 | c2
----+----
  1 |  1
  2 |  2
  3 |  3
  4 |  4
  4 |  4
  5 |  5
(6 rows)
```

分别关闭和打开uniquecheck参数对比，打开之后报错。

```
gaussdb=#  select t1.c1 from t1 where t1.c1 = (select t2.c1 from t2 where t1.c1=t2.c2) ;
 c1
----
  1
  2
  3
(3 rows)
gaussdb=#  set rewrite_rule='uniquecheck';
SET
gaussdb=#  select t1.c1 from t1 where t1.c1 = (select t2.c1 from t2 where t1.c1=t2.c2) ;
ERROR:  more than one row returned by a subquery used as an expression
```

消除子查询中的聚集运算参数 lazyagg

消除子查询中的聚集运算，以此提高查询效率。举例：

```
gaussdb=# set rewrite_rule =none;
SET
gaussdb=# EXPLAIN (costs off) SELECT t.c2, sum(cc) FROM (SELECT c2, sum(c3) AS cc FROM t1 GROUP BY
c2) s1, t WHERE s1.c2=t.c2 GROUP BY t.c2 ORDER BY 1,2;
                    QUERY PLAN
-----------------------------------------------------
 Sort
   Sort Key: t.c2, (sum(s1.cc))
```



```
  ->  HashAggregate
        Group By Key: t.c2
        ->  Hash Join
              Hash Cond: (t.c2 = s1.c2)
              ->  Seq Scan on t
              ->  Hash
                    ->  Subquery Scan on s1
                          ->  HashAggregate
                                Group By Key: t1.c2
                                ->  Seq Scan on t1
(12 rows)
```

子查询与外层查询存在同样的group by条件，两层聚集运算可能导致查询效率低下，打开lazyagg参数，消除子查询中的聚集运算，提升查询性能：

```
gaussdb=# set rewrite_rule = lazyagg;
SET
gaussdb=# explain (costs off) select t.b, sum(cc) from (select b, sum(c) as cc from t1 group by b) s1, t
where s1.b=t.b group by t.b order by 1,2;
              QUERY PLAN
---------------------------------------
 Sort
   Sort Key: t.b, (sum((t1.c)::bigint))
   ->  HashAggregate
         Group By Key: t.b
         ->  Hash Join
               Hash Cond: (t1.b = t.b)
               ->  Seq Scan on t1
               ->  Hash
                     ->  Seq Scan on t
(9 rows)
```

从主查询下推条件到子查询参数 magicset

将带有聚集算子的子查询提前和主查询进行关联。举例：

```
gaussdb=# set rewrite_rule = none;
SET
gaussdb=# EXPLAIN (costs off) SELECT t1 FROM t1 WHERE t1.c2 = 10 AND t1.c3 < (SELECT sum(c3) FROM
t2 WHERE t1.c1 = t2.c1);
              QUERY PLAN
---------------------------------------
 Hash Join
   Hash Cond: (t2.c1 = t1.c1)
   Join Filter: (t1.c3 < (sum(t2.c3)))
   ->  HashAggregate
         Group By Key: t2.c1
         ->  Seq Scan on t2
   ->  Hash
         ->  Seq Scan on t1
               Filter: (c2 = 10)
(9 rows)
```

先针对子查询的关联字段进行分组聚集，再和主查询进行关联，减少相关子链接的重复扫描，提升查询效率，修改重写参数后，计划改变：

```
gaussdb=# set rewrite_rule = magicset;
SET
gaussdb=# explain (costs off) SELECT t1 FROM t1 WHERE t1.c2 = 10 AND t1.c3 < (SELECT sum(c3) FROM
t2 WHERE t1.c1 = t2.c1);
                          QUERY PLAN
---------------------------------------------------------------------------
 Hash Join
   Hash Cond: (t2.c1 = rewrite_rule_guc_test.t1.c1)
   Join Filter: (rewrite_rule_guc_test.t1.c3 < (sum(t2.c3)))
   ->  HashAggregate
         Group By Key: t2.c1
         ->  Hash Join
```



```
            Hash Cond: (t2.c1 = rewrite_rule_guc_test.t1.c1)
            ->  Seq Scan on t2
            ->  Hash
                  ->  HashAggregate
                        Group By Key: rewrite_rule_guc_test.t1.c1
                        ->  Seq Scan on t1
                              Filter: (c2 = 10)
  ->  Hash
        ->  Seq Scan on t1
              Filter: (c2 = 10)
(16 rows)
```

6.12.2 案例：建立合适的索引

现象描述

查询与销售部所有员工的信息：

```
--建表。
CREATE TABLE staffs (staff_id NUMBER(6) NOT NULL, first_name VARCHAR2(20), last_name
VARCHAR2(25), employment_id VARCHAR2(10), section_id NUMBER(4), state_name VARCHAR2(10), city
VARCHAR2(10));
CREATE TABLE sections(section_id NUMBER(4), place_id NUMBER(4), section_name VARCHAR2(20));
CREATE TABLE states(state_id NUMBER(4));
CREATE TABLE places(place_id NUMBER(4), state_id  NUMBER(4));

--优化前查询。
EXPLAIN SELECT staff_id,first_name,last_name,employment_id,state_name,city
FROM staffs,sections,states,places
WHERE sections.section_name='Sales'
AND staffs.section_id = sections.section_id
AND sections.place_id = places.place_id
AND places.state_id = states.state_id
ORDER BY staff_id;

--创建索引。
CREATE INDEX loc_id_pk ON places(place_id);
CREATE INDEX state_c_id_pk ON states(state_id);

--优化后查询。
EXPLAIN SELECT staff_id,first_name,last_name,employment_id,state_name,city
FROM staffs,sections,states,places
WHERE sections.section_name='Sales'
AND staffs.section_id = sections.section_id
AND sections.place_id = places.place_id
AND places.state_id = states.state_id
ORDER BY staff_id;
```

优化分析

在优化前，没有创建places.place_id和states.state_id索引，执行计划如下：

```
                                    QUERY PLAN
-------------------------------------------------------------------------------------------------
 Sort  (cost=125.25..126.34 rows=438 width=254)
   Sort Key: staffs.staff_id
   ->  Hash Join  (cost=64.91..106.03 rows=438 width=254)
         Hash Cond: (states.state_id = places.state_id)
         ->  Seq Scan on states  (cost=0.00..29.45 rows=1945 width=12)
         ->  Hash  (cost=64.35..64.35 rows=45 width=266)
               ->  Hash Join  (cost=33.09..64.35 rows=45 width=266)
                     Hash Cond: (places.place_id = sections.place_id)
                     ->  Seq Scan on places  (cost=0.00..25.13 rows=1513 width=24)
                     ->  Hash  (cost=33.02..33.02 rows=6 width=266)
                           ->  Hash Join  (cost=19.16..33.02 rows=6 width=266)
                                 Hash Cond: (staffs.section_id = sections.section_id)
                                 ->  Seq Scan on staffs  (cost=0.00..12.76 rows=276 width=266)
```



```
                        ->  Hash  (cost=19.11..19.11 rows=4 width=24)
                              ->  Seq Scan on sections  (cost=0.00..19.11 rows=4 width=24)
                                    Filter: ((section_name)::text = 'Sales'::text)
(16 rows)
```

建议在places.place_id和states.state_id列上建立2个索引（参考**现象描述**），执行计划如下：

```
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Sort  (cost=107.40..108.49 rows=438 width=254)
   Sort Key: staffs.staff_id
   ->  Hash Join  (cost=35.37..88.18 rows=438 width=254)
         Hash Cond: (sections.section_id = staffs.section_id)
         ->  Nested Loop  (cost=19.16..66.85 rows=292 width=12)
               ->  Hash Join  (cost=19.16..50.27 rows=30 width=24)
                     Hash Cond: (places.place_id = sections.place_id)
                     ->  Seq Scan on places  (cost=0.00..25.13 rows=1513 width=24)
                     ->  Hash  (cost=19.11..19.11 rows=4 width=24)
                           ->  Seq Scan on sections  (cost=0.00..19.11 rows=4 width=24)
                                 Filter: ((section_name)::text = 'Sales'::text)
               ->  Index Only Scan using state_c_id_pk on states  (cost=0.00..0.45 rows=10 width=12)
                     Index Cond: (state_id = places.state_id)
         ->  Hash  (cost=12.76..12.76 rows=276 width=266)
               ->  Seq Scan on staffs  (cost=0.00..12.76 rows=276 width=266)
(15 rows)
```

6.12.3 案例：增加 JOIN 列非空条件

```
SELECT * FROM join_a a JOIN join_b b ON a.b = b.b;
```

执行计划如下：

```
                                                                QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=58.35..14677.69 rows=1074607 width=16) (actual time=23.374..23.384 rows=10 loops=1)
   Hash Cond: (a.b = b.b)
   ->  Seq Scan on join_a a  (cost=0.00..2248.10 rows=100010 width=8) (actual time=0.495..12.551
rows=100010 loops=1)
   ->  Hash  (cost=31.49..31.49 rows=2149 width=8) (actual time=0.614..0.614 rows=1000 loops=1)
         Buckets: 32768  Batches: 1  Memory Usage: 40kB
         ->  Seq Scan on join_b b  (cost=0.00..31.49 rows=2149 width=8) (actual time=0.009..0.183 rows=1000
loops=1)
 Total runtime: 23.716 ms
(7 rows)
```

优化分析

1. 分析执行计划可知，在顺序扫描阶段耗时较多。
2. 建议在语句中手动添加JOIN列的非空判断，修改后的语句如下所示。

   ```
   SELECT
    *
   SELECT * FROM join_a a JOIN join_b b ON a.b = b.b where a.b IS NOT NULL;
   ```

   执行计划如下：

   ```
                                                                   QUERY PLAN
   ------------------------------------------------------------------------------------------------------------------------------------------
    Hash Join  (cost=58.22..14560.97 rows=1063762 width=16) (actual time=13.237..13.247 rows=10
   loops=1)
      Hash Cond: (a.b = b.b)
      ->  Seq Scan on join_a a  (cost=0.00..2248.10 rows=99510 width=8) (actual time=12.417..12.422
   rows=10 loops=1)
            Filter: (b IS NOT NULL)
            Rows Removed by Filter: 100000
      ->  Hash  (cost=31.49..31.49 rows=2138 width=8) (actual time=0.566..0.566 rows=1000 loops=1)
            Buckets: 32768  Batches: 1  Memory Usage: 40kB
            ->  Seq Scan on join_b b  (cost=0.00..31.49 rows=2138 width=8) (actual time=0.011..0.229
   ```



```
rows=1000 loops=1)
              Filter: (b IS NOT NULL)
 Total runtime: 13.556 ms
(10 rows)
```

6.12.4 案例：改建分区表

现象描述

如下简单SQL语句查询， 性能瓶颈点在normal_date的Scan上。

```
                                                  QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------
 Seq Scan on normal_date  (cost=0.00..259.00 rows=30 width=12) (actual time=0.100..3.466 rows=30
loops=1)
   Filter: (("time" >= '2022-09-01 00:00:00'::timestamp without time zone) AND ("time" <= '2022-10-01
00:00:00'::timestamp without time zone))
   Rows Removed by Filter: 9970
 Total runtime: 3.587 ms
(4 rows)
```

优化分析

从业务层确认表数据(在time字段上)有明显的日期特征，符合分区表的特征。重新规划normal_date表的表定义：字段time为分区键、月为间隔单位定义分区表normal_date_part。修改后结果如下，性能提升近10倍。

```
                                                  QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------
 Partition Iterator  (cost=0.00..480.00 rows=30 width=12) (actual time=0.038..0.085 rows=30 loops=1)
   Iterations: 2
   ->  Partitioned Seq Scan on normal_date_part  (cost=0.00..480.00 rows=30 width=12) (actual
time=0.049..0.063 rows=30 loops=2)
         Filter: (("time" >= '2022-09-01 00:00:00'::timestamp without time zone) AND ("time" <= '2022-10-01
00:00:00'::timestamp without time zone))
         Rows Removed by Filter: 31
         Selected Partitions:  3..4
 Total runtime: 0.360 ms
(7 rows)
```

6.12.5 案例：改写 SQL 消除子查询

现象描述

```
select
    1,
    (select count(*) from normal_date n where n.id = a.id) as GZCS
from normal_date a;
```

此SQL性能较差，查看发现执行计划中存在SubPlan，具体如下：

```
                                                  QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------
 Seq Scan on normal_date a  (cost=0.00..888118.42 rows=5129 width=4) (actual time=2.394..22194.907
rows=10000 loops=1)
   SubPlan 1
     ->  Aggregate  (cost=173.12..173.12 rows=1 width=8) (actual time=22179.496..22179.942 rows=10000
loops=10000)
           ->  Seq Scan on normal_date n  (cost=0.00..173.11 rows=1 width=0) (actual
time=11279.349..22159.608 rows=10000 loops=10000)
                 Filter: (id = a.id)
```

```
                Rows Removed by Filter: 99990000
 Total runtime: 22196.415 ms
(7 rows)
```

优化说明

此优化的核心就是消除子查询。分析业务场景发现*a.id*不为null，那么从SQL语义出发，可以等价改写SQL为：

```
select
count(*)
from normal_date n, normal_date a
where n.id = a.id
group by  a.id;
计划如下：
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------
----
 HashAggregate  (cost=480.86..532.15 rows=5129 width=12) (actual time=21.539..24.356 rows=10000
loops=1)
   Group By Key: a.id
   ->  Hash Join  (cost=224.40..455.22 rows=5129 width=4) (actual time=6.402..13.484 rows=10000 loops=1)
         Hash Cond: (n.id = a.id)
         ->  Seq Scan on normal_date n  (cost=0.00..160.29 rows=5129 width=4) (actual time=0.087..1.459
rows=10000 loops=1)
         ->  Hash  (cost=160.29..160.29 rows=5129 width=4) (actual time=6.065..6.065 rows=10000 loops=1)
               Buckets: 32768  Batches: 1  Memory Usage: 352kB
               ->  Seq Scan on normal_date a  (cost=0.00..160.29 rows=5129 width=4) (actual time=0.046..2.738
rows=10000 loops=1)
 Total runtime: 26.844 ms
(9 rows)
```

说明

为了保证改写的等效性，在*normal_date.id*加了*not null*约束。

6.12.6 案例：改写 SQL 消除 in-clause

现象描述

in-clause/any-clause是常见的SQL语句约束条件，有时in或any后面的clause都是常量，类似于：

```
select count(1) from calc_empfyc_c1_result_tmp_t1 where ls_pid_cusr1 in (‘20120405’, ‘20130405’);
```

或者

```
select count(1) from calc_empfyc_c1_result_tmp_t1 where ls_pid_cusr1 in any(‘20120405’,
‘20130405’);
```

但是也有一些如下的特殊用法：

```
SELECT * FROM test1 t1, test2 t2 WHERE t1.a = any(values(t2.a),(t2.b));
```

其中，a、b为t2中的两列，“t1.a = any(values(t2.ba,(t2.b))”等价于“t1.a = t2.a or t1.a = t2.b”。

因此join-condition实质上是一个不等式，这种非等值的join操作必须使用nestloop来连接，对应执行计划如下：

```
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------
---
 Nested Loop  (cost=0.00..138614.38 rows=2309100 width=16) (actual time=0.152..19225.483 rows=1000
loops=1)
```



```
  Join Filter: (SubPlan 1)
  Rows Removed by Join Filter: 999000
  ->  Seq Scan on test1 t1  (cost=0.00..31.49 rows=2149 width=8) (actual time=0.021..3.309 rows=1000 loops=1)
  ->  Materialize  (cost=0.00..42.23 rows=2149 width=8) (actual time=0.331..1265.810 rows=1000000 loops=1000)
        ->  Seq Scan on test2 t2  (cost=0.00..31.49 rows=2149 width=8) (actual time=0.013..0.268 rows=1000 loops=1)
  SubPlan 1
    ->  Values Scan on "*VALUES*"  (cost=0.00..0.03 rows=2 width=4) (actual time=2890.741..7372.739 rows=1999000 loops=1000000)
 Total runtime: 19227.328 ms
(9 rows)
```

优化说明

测试发现由于两表结果集过大，导致nestloop耗时过长，超过一小时未返回结果，因此性能优化的关键是消除nestloop，让join使用更高效的hashjoin。从语义等价的角度消除any-clause，SQL改写如下：

```
SELECT
*
FROM (
    SELECT * FROM test1 t1, test2 t2 WHERE t1.a = t2.a
    UNION
    SELECT * FROM test1 t1, test2 t2 WHERE t1.a = t2.b
);
```

优化后的SQL查询由两个等值join的子查询构成，而每个子查询都可以使用更适合此场景的hashjoin。优化后的执行计划如下

```
                                                    QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
---
 HashAggregate  (cost=1634.99..2096.81 rows=46182 width=16) (actual time=6.369..6.772 rows=1000 loops=1)
   Group By Key: t1.a, t1.b, t2.a, t2.b
   ->  Append  (cost=58.35..1173.17 rows=46182 width=16) (actual time=0.833..3.414 rows=2000 loops=1)
         ->  Hash Join  (cost=58.35..355.67 rows=23091 width=16) (actual time=0.832..1.590 rows=1000 loops=1)
               Hash Cond: (t1.a = t2.a)
               ->  Seq Scan on test1 t1  (cost=0.00..31.49 rows=2149 width=8) (actual time=0.015..0.156 rows=1000 loops=1)
               ->  Hash  (cost=31.49..31.49 rows=2149 width=8) (actual time=0.531..0.531 rows=1000 loops=1)
                     Buckets: 32768  Batches: 1  Memory Usage: 40kB
                    ->  Seq Scan on test2 t2  (cost=0.00..31.49 rows=2149 width=8) (actual time=0.010..0.199 rows=1000 loops=1)
         ->  Hash Join  (cost=58.35..355.67 rows=23091 width=16) (actual time=0.694..1.421 rows=1000 loops=1)
               Hash Cond: (t1.a = t2.b)
               ->  Seq Scan on test1 t1  (cost=0.00..31.49 rows=2149 width=8) (actual time=0.010..0.160 rows=1000 loops=1)
               ->  Hash  (cost=31.49..31.49 rows=2149 width=8) (actual time=0.524..0.524 rows=1000 loops=1)
                     Buckets: 32768  Batches: 1  Memory Usage: 40kB
                    ->  Seq Scan on test2 t2  (cost=0.00..31.49 rows=2149 width=8) (actual time=0.008..0.177 rows=1000 loops=1)
 Total runtime: 7.759 ms
(16 rows)
```

7 SQL 参考

7.1 SQL

什么是 SQL

SQL是用于访问和处理数据库的标准计算机语言。

SQL提供了各种任务的语句，包括：

- 查询数据。
- 在表中插入、更新和删除行。
- 创建、替换、更改和删除对象。
- 控制对数据库及其对象的访问。
- 保证数据库的一致性和完整性。

SQL语言由用于处理数据库和数据库对象的命令和函数组成。该语言还会强制实施有关数据类型、表达式和文本使用的规则。因此在**SQL参考**章节，除了SQL语法参考外，还介绍了有关数据类型、表达式、函数和操作符等信息。

SQL 发展简史

SQL发展简史如下：

- 1986年，ANSI X3.135-1986，ISO/IEC 9075:1986，SQL-86
- 1989年，ANSI X3.135-1989，ISO/IEC 9075:1989，SQL-89
- 1992年，ANSI X3.135-1992，ISO/IEC 9075:1992，SQL-92
- 1999年，ISO/IEC 9075:1999，SQL:1999
- 2003年，ISO/IEC 9075:2003，SQL:2003
- 2011年，ISO/IEC 9075:2011，SQL:2011
- 2016年，ISO/IEC 9075:2016，SQL:2016
- 2019年，ISO/IEC 9075:2019，SQL:2019

GaussDB 支持的 SQL 标准

GaussDB默认支持SQL:2016的大部分特性。

7.2 关键字

SQL里有保留关键字和非保留关键字之分。根据标准，保留关键字绝不能用做其他标识符。非保留关键字只是在特定的环境里有特殊的含义，而在其他环境里是可以用做标识符的。

须知

1. 目前“非保留”关键字在作为数据库对象的标识符时存在如下限制：
   1. 不支持直接作为列别名使用，即类似SELECT 1 ABORT的用法会导致错误。
   2. 对于ENTITYESCAPING、NOENTITYESCAPING以及WELLFORMED关键字，不带双引号时不支持作为表名、列名、表别名、列别名以及函数名的标识符。
   3. 不带双引号的RAW关键字不支持作为表名和函数名的标识符。
   4. 不带双引号的SET关键字不支持作为表别名的标识符，即类似SELECT * FROM T1 SET的用法均会导致错误。
   5. 不带双引号的BEGIN、BY、CLOSE、CURSOR、DECLARE、DELETE、EXECUTE、FUNCTION、IF、IMMEDIATE、INSERT、LOOP、MOVE、OF、REF、RELEASE、RETURN、SAVEPOINT、STRICT、TYPE以及UPDATE等关键字不支持作为变量名使用。
   6. 用SYS_REFCURSOR关键字作为数据库对象的标识符时，如果不附带双引号，则创建名为REFCURSOR的数据库对象，如果附带了双引号，则创建名为SYS_REFCURSOR的数据库对象。
2. 与“非保留”关键字类似，“非保留（不能是函数或类型）”关键字不支持直接作为列别名使用。
3. 对于不带有双引号的“保留”关键字CURRENT_TIMESTAMP而言，不允许作为函数名。

标识符命名规范

标识符的命名需要遵守如下规范：

- 标识符需要为字母（a-z）、下划线（_）、数字（0-9）或美元符号（$）。
- 标识符必须以字母（a-z）或下划线（_）开头。

  说明

  - 此命名规范为建议项，非强制项。
  - 特殊情况下可以使用双引号规避特殊字符报错。

SQL 关键字

表 7-1 SQL 关键字

关键字	GaussDB	SQL:1999	SQL-92
ABORT	非保留	-	-
ABS	-	非保留	-
ABSOLUTE	非保留	保留	保留
ACCESS	非保留	-	-
ACCOUNT	非保留	-	-
ACTION	非保留	保留	保留
ADA	-	非保留	非保留
ADD	非保留	保留	保留
ADMIN	非保留	保留	-
AFTER	非保留	保留	-
AGGREGATE	非保留	保留	-
ALGORITHM	非保留	-	-
ALIAS	-	保留	-
ALL	保留	保留	保留
ALLOCATE	-	保留	保留
ALSO	非保留	-	-
ALTER	非保留	保留	保留
ALWAYS	非保留	-	-
ANALYSE	保留	-	-
ANALYZE	保留	-	-
AND	保留	保留	保留
ANY	保留	保留	保留
APP	非保留	-	-
APPEND	非保留	-	-
ARCHIVE	非保留	-	-
ARE	-	保留	保留
ARRAY	保留	保留	-
AS	保留	保留	保留

关键字	GaussDB	SQL:1999	SQL-92
ASC	保留	保留	保留
ASENSITIVE	-	非保留	-
ASSERTION	非保留	保留	保留
ASSIGNMENT	非保留	非保留	-
ASYMMETRIC	保留	非保留	-
AT	非保留	保留	保留
ATOMIC	-	非保留	-
ATTRIBUTE	非保留	-	-
AUDIT	非保留	-	-
AUTHID	保留	-	-
AUTHORIZATION	保留（可以是函数或类型）	保留	保留
AUTO_INCREMENT	非保留	-	-
AUTOEXTEND	非保留	-	-
AUTOMAPPED	非保留	-	-
AVG	-	非保留	保留
BACKWARD	非保留	-	-
BAD_PATH	非保留	-	-
BARRIER	非保留	-	-
BEFORE	非保留	保留	-
BEGIN	非保留	保留	保留
BEGIN_NON_ANOYBLOCK	非保留	-	-
BETWEEN	非保留（不能是函数或类型）	非保留	保留
BIGINT	非保留（不能是函数或类型）	-	-
BINARY	保留（可以是函数或类型）	保留	-
BINARY_DOUBLE	非保留（不能是函数或类型）	-	-
BINARY_INTEGER	非保留（不能是函数或类型）	-	-

关键字	GaussDB	SQL:1999	SQL-92
BIT	非保留（不能是函数或类型）	保留	保留
BIT_LENGTH	-	非保留	保留
BITVAR	-	非保留	-
BLANKS	非保留	-	-
BLOB	非保留	保留	-
BODY	非保留	-	-
BOOLEAN	非保留（不能是函数或类型）	保留	-
BOTH	保留	保留	保留
BREADTH	-	保留	-
BUCKETCNT	非保留（不能是函数或类型）	-	-
BUCKETS	保留	-	-
BY	非保留	保留	保留
BYTEAWITHOUTORDER	非保留（不能是函数或类型）	-	-
BYTEAWITHOUTORDERWITHEQUAL	非保留（不能是函数或类型）	-	-
C	-	非保留	非保留
CACHE	非保留	-	-
CALL	非保留	保留	-
CALLED	非保留	非保留	-
CANCELABLE	非保留	-	-
CARDINALITY	-	非保留	-
CASCADE	非保留	保留	保留
CASCADED	非保留	保留	保留
CASE	保留	保留	保留
CAST	保留	保留	保留
CATALOG	非保留	保留	保留
CATALOG_NAME	-	非保留	非保留
CHAIN	非保留	非保留	-

关键字	GaussDB	SQL:1999	SQL-92
CHANGE	非保留	-	-
CHAR	非保留（不能是函数或类型）	保留	保留
CHAR_LENGTH	-	非保留	保留
CHARACTER	非保留（不能是函数或类型）	保留	保留
CHARACTER_LENGTH	-	非保留	保留
CHARACTER_SET_CATALOG	-	非保留	非保留
CHARACTER_SET_NAME	-	非保留	非保留
CHARACTER_SET_SCHEMA	-	非保留	非保留
CHARACTERISTICS	非保留	-	-
CHARACTERSET	非保留	-	-
CHARSET	非保留	-	-
CHECK	保留	保留	保留
CHECKED	-	非保留	-
CHECKPOINT	非保留	-	-
CLASS	非保留	保留	-
CLASS_ORIGIN	-	非保留	非保留
CLEAN	非保留	-	-
CLIENT	非保留	-	-
CLIENT_MASTER_KEY	非保留	-	-
CLIENT_MASTER_KEYS	非保留	-	-
CLOB	非保留	保留	-
CLOSE	非保留	保留	保留
CLUSTER	非保留	-	-
COALESCE	非保留（不能是函数或类型）	非保留	保留
COBOL	-	非保留	非保留
COLLATE	保留	保留	保留

关键字	GaussDB	SQL:1999	SQL-92
COLLATION	保留（可以是函数或类型）	保留	保留
COLLATION_CATALOG	-	非保留	非保留
COLLATION_NAME	-	非保留	非保留
COLLATION_SCHEMA	-	非保留	非保留
COLUMN	保留	保留	保留
COLUMN_ENCRYPTION_KEY	非保留	-	-
COLUMN_ENCRYPTION_KEYS	非保留	-	-
COLUMN_NAME	-	非保留	非保留
COLUMNS	非保留	-	-
COMMAND_FUNCTION	-	非保留	非保留
COMMAND_FUNCTION_CODE	-	非保留	-
COMMENT	非保留	-	-
COMMENTS	非保留	-	-
COMMIT	非保留	保留	保留
COMMITTED	非保留	非保留	非保留
COMPACT	保留（可以是函数或类型）	-	-
COMPATIBLE_ILLEGAL_CHARS	非保留	-	-
COMPILE	非保留	-	-
COMPLETE	非保留	-	-
COMPLETION	非保留	保留	-
COMPRESS	非保留	-	-
CONCURRENTLY	保留（可以是函数或类型）	-	-
CONDITION	非保留	-	-
CONDITION_NUMBER	-	非保留	非保留
CONFIGURATION	非保留	-	-
CONNECT	非保留	保留	保留

关键字	GaussDB	SQL:1999	SQL-92
CONNECTION	非保留	保留	保留
CONNECTION_NAME	-	非保留	非保留
CONSTANT	非保留	-	-
CONSTRAINT	保留	保留	保留
CONSTRAINT_CATALOG	-	非保留	非保留
CONSTRAINT_NAME	-	非保留	非保留
CONSTRAINT_SCHEMA	-	非保留	非保留
CONSTRAINTS	非保留	保留	保留
CONSTRUCTOR	-	保留	-
CONTAINING	非保留	-	-
CONTAINS	-	非保留	-
CONTENT	非保留	-	-
CONTINUE	非保留	保留	保留
CONVERSION	非保留	-	-
CONVERT	非保留	非保留	保留
COORDINATOR	非保留	-	-
COORDINATORS	非保留	-	-
COPY	非保留	-	-
CORRESPONDING	-	保留	保留
COST	非保留	-	-
COUNT	-	非保留	保留
CREATE	保留	保留	保留
CROSS	保留（可以是函数或类型）	保留	保留
CSN	保留（可以是函数或类型）	-	-
CSV	非保留	-	-
CUBE	非保留	保留	-
CURRENT	非保留	保留	保留
CURRENT_CATALOG	保留	-	-
CURRENT_DATE	保留	保留	保留

关键字	GaussDB	SQL:1999	SQL-92
CURRENT_PATH	-	保留	-
CURRENT_ROLE	保留	保留	-
CURRENT_SCHEMA	保留（可以是函数或类型）	-	-
CURRENT_TIME	保留	保留	保留
CURRENT_TIMESTAMP	保留	保留	保留
CURRENT_USER	保留	保留	保留
CURSOR	非保留	保留	保留
CURSOR_NAME	-	非保留	非保留
CYCLE	非保留	保留	-
DATA	非保留	保留	非保留
DATABASE	非保留	-	-
DATAFILE	非保留	-	-
DATANODE	非保留	-	-
DATANODES	非保留	-	-
DATATYPE_CL	非保留	-	-
DATE	非保留（不能是函数或类型）	保留	保留
DATE_FORMAT	非保留	-	-
DATETIME_INTERVAL_CODE	-	非保留	非保留
DATETIME_INTERVAL_PRECISION	-	非保留	非保留
DAY	非保留	保留	保留
DB4AISHOT	非保留	-	-
DBCOMPATIBILITY	非保留	-	-
DBTIMEZONE	保留	-	-
DEALLOCATE	非保留	保留	保留
DEC	非保留（不能是函数或类型）	保留	保留
DECIMAL	非保留（不能是函数或类型）	保留	保留

关键字	GaussDB	SQL:1999	SQL-92
DECLARE	非保留	保留	保留
DECODE	非保留（不能是函数或类型）	-	-
DEFAULT	保留	保留	保留
DEFAULTS	非保留	-	-
DEFERRABLE	保留	保留	保留
DEFERRED	非保留	保留	保留
DEFINED	-	非保留	-
DEFINER	非保留	非保留	-
DELETE	非保留	保留	保留
DELIMITER	非保留	-	-
DELIMITERS	非保留	-	-
DELTA	非保留	-	-
DELTAMERGE	保留（可以是函数或类型）	-	-
DEPTH	-	保留	-
DEREF	-	保留	-
DESC	保留	保留	保留
DESCRIBE	-	保留	保留
DESCRIPTOR	-	保留	保留
DESTROY	-	保留	-
DESTRUCTOR	-	保留	-
DETERMINISTIC	非保留	保留	-
DIAGNOSTICS	-	保留	保留
DICTIONARY	非保留	保留	-
DIRECT	非保留	-	-
DIRECTORY	非保留	-	-
DISABLE	非保留	-	-
DISCARD	非保留	-	-
DISCARD_PATH	非保留	-	-
DISCONNECT	非保留	保留	保留

关键字	GaussDB	SQL:1999	SQL-92
DISPATCH	-	非保留	-
DISTINCT	保留	保留	保留
DISTRIBUTE	非保留	-	-
DISTRIBUTION	非保留	-	-
DO	保留	-	-
DOCUMENT	非保留	-	-
DOMAIN	非保留	保留	保留
DOUBLE	非保留	保留	保留
DROP	非保留	保留	保留
DUMPFILE	非保留	-	-
DUPLICATE	非保留	-	-
DYNAMIC	-	保留	-
DYNAMIC_FUNCTION	-	非保留	非保留
DYNAMIC_FUNCTION_CODE	-	非保留	-
EACH	非保留	保留	-
ELASTIC	非保留	-	-
ELSE	保留	保留	保留
ENABLE	非保留	-	-
ENCLOSED	非保留	-	-
ENCODING	非保留	-	-
ENCRYPTED	非保留	-	-
ENCRYPTED_VALUE	非保留	-	-
ENCRYPTION	非保留	-	-
ENCRYPTION_TYPE	非保留	-	-
END	保留	保留	保留
END-EXEC	-	保留	保留
ENDS	非保留	-	-
ENFORCED	非保留	-	-
ENTITYESCAPING	非保留	-	-

关键字	GaussDB	SQL:1999	SQL-92
ENUM	非保留	-	-
EOL	非保留	-	-
EQUALS	-	保留	-
ERROR	非保留	-	-
ERRORS	非保留	-	-
ESCAPE	非保留	保留	保留
ESCAPED	非保留	-	-
ESCAPING	非保留	-	-
EVALNAME	非保留	-	-
EVENT	非保留	-	-
EVENTS	非保留	-	-
EVERY	非保留	保留	-
EXCEPT	保留	保留	保留
EXCEPTION	-	保留	保留
EXCHANGE	非保留	-	-
EXCLUDE	非保留	-	-
EXCLUDED	保留	-	-
EXCLUDING	非保留	-	-
EXCLUSIVE	非保留	-	-
EXEC	-	保留	保留
EXECUTE	非保留	保留	保留
EXISTING	-	非保留	-
EXISTS	非保留（不能是函数或类型）	非保留	保留
EXPDP	非保留	-	-
EXPIRED_P	-	-	-
EXPLAIN	非保留	-	-
EXTENSION	非保留	-	-
EXTERNAL	非保留	保留	保留
EXTRACT	非保留（不能是函数或类型）	非保留	保留

关键字	GaussDB	SQL:1999	SQL-92
FALSE	保留	保留	保留
FAMILY	非保留	-	-
FAST	非保留	-	-
FEATURES	非保留	-	-
FENCED	保留	-	-
FETCH	保留	保留	保留
FIELDS	非保留	-	-
FILEHEADER	非保留	-	-
FILL_MISSING_FIELDS	非保留	-	-
FILLER	非保留	-	-
FILTER	非保留	-	保留
FINAL	-	非保留	-
FIRST	非保留	保留	保留
FIXED	非保留	-	保留
FLOAT	非保留（不能是函数或类型）	保留	保留
FOLLOWING	非保留	-	-
FOR	保留	保留	保留
FORCE	非保留	-	-
FOREIGN	保留	保留	保留
FORMATTER	非保留	-	-
FORTRAN	-	非保留	非保留
FORWARD	非保留	-	-
FOUND	-	保留	保留
FREE	-	保留	-
FREEZE	保留（可以是函数或类型）	-	-
FROM	保留	保留	保留
FULL	保留（可以是函数或类型）	保留	保留
FUNCTION	非保留	保留	-

关键字	GaussDB	SQL:1999	SQL-92
FUNCTIONS	非保留	-	-
G	-	非保留	-
GENERAL	-	保留	-
GENERATED	非保留	非保留	-
GET	-	保留	保留
GLOBAL	非保留	保留	保留
GO	-	保留	保留
GOTO	-	保留	保留
GRANT	保留	保留	保留
GRANTED	非保留	非保留	-
GREATEST	非保留（不能是函数或类型）	-	-
GROUP	保留	保留	保留
GROUPING	非保留（不能是函数或类型）	保留	-
GROUPPARENT	保留	-	-
HANDLER	非保留	-	-
HAVING	保留	保留	保留
HDFSDIRECTORY	保留（可以是函数或类型）	-	-
HEADER	非保留	-	-
HIERARCHY	-	非保留	-
HOLD	非保留	非保留	-
HOST	-	保留	-
HOUR	非保留	保留	保留
IDENTIFIED	非保留	-	-
IDENTITY	非保留	保留	保留
IF	非保留	-	-
IGNORE	非保留	保留	-
IGNORE_EXTRA_DATA	非保留	-	-
ILIKE	保留（可以是函数或类型）	-	-

关键字	GaussDB	SQL:1999	SQL-92
IMMEDIATE	非保留	保留	保留
IMMUTABLE	非保留	-	-
IMPDP	非保留	-	-
IMPLEMENTATION	-	非保留	-
IMPLICIT	非保留	-	-
IN	保留	保留	保留
INCLUDE	非保留	-	-
INCLUDING	非保留	-	-
INCREMENT	非保留	-	-
INCREMENTAL	非保留	-	-
INDEX	非保留	-	-
INDEXES	非保留	-	-
INDICATOR	-	保留	保留
INFILE	非保留	-	-
INFIX	-	非保留	-
INHERIT	非保留	-	-
INHERITS	非保留	-	-
INITIAL	非保留	-	-
INITIALIZE	-	保留	-
INITIALLY	保留	保留	保留
INITRANS	非保留	-	-
INLINE	非保留	-	-
INNER	保留（可以是函数或类型）	保留	保留
INOUT	非保留（不能是函数或类型）	保留	-
INPUT	非保留	保留	保留
INSENSITIVE	非保留	非保留	保留
INSERT	非保留	保留	保留
INSTANCE	-	非保留	-
INSTANTIABLE	-	非保留	-

关键字	GaussDB	SQL:1999	SQL-92
INSTEAD	非保留	-	-
INT	非保留（不能是函数或类型）	保留	保留
INTEGER	非保留（不能是函数或类型）	保留	保留
INTERNAL	非保留	-	-
INTERSECT	保留	保留	保留
INTERVAL	非保留（不能是函数或类型）	保留	保留
INTO	保留	保留	保留
INVOKER	非保留	非保留	-
IP	非保留	-	-
IS	保留	保留	保留
ISNULL	非保留	-	-
ISOLATION	非保留	保留	保留
ITERATE	-	保留	-
JOIN	保留（可以是函数或类型）	保留	保留
K	-	非保留	-
KEY	非保留	保留	保留
KEY_MEMBER	-	非保留	-
KEY_PATH	非保留	-	-
KEY_STORE	非保留	-	-
KEY_TYPE	-	非保留	-
KILL	非保留	-	-
LABEL	非保留	-	-
LANGUAGE	非保留	保留	保留
LARGE	非保留	保留	-
LAST	非保留	保留	保留
LATERAL	-	保留	-
LC_COLLATE	非保留	-	-
LC_CTYPE	非保留	-	-

关键字	GaussDB	SQL:1999	SQL-92
LEADING	保留	保留	保留
LEAKPROOF	非保留	-	-
LEAST	非保留（不能是函数或类型）	-	-
LEFT	保留（可以是函数或类型）	保留	保留
LENGTH	-	非保留	非保留
LESS	保留	保留	-
LEVEL	非保留	保留	保留
LIKE	保留（可以是函数或类型）	保留	保留
LIMIT	保留	保留	-
LINES	非保留	-	-
LINK	非保留	-	-
LIST	非保留	-	-
LISTEN	非保留	-	-
LNNVL	非保留（不能是函数或类型）	-	-
LOAD	非保留	-	-
LOAD_BAD	非保留	-	-
LOAD_DISCARD	非保留	-	-
LOCAL	非保留	保留	保留
LOCALTIME	保留	保留	-
LOCALTIMESTAMP	保留	保留	-
LOCATION	非保留	-	-
LOCATOR	-	保留	-
LOCK	非保留	-	-
LOG	非保留	-	-
LOGGING	非保留	-	-
LOGIN_ANY	非保留	-	-
LOGIN_FAILURE	非保留	-	-
LOGIN_SUCCESS	非保留	-	-

关键字	GaussDB	SQL:1999	SQL-92
LOGOUT	非保留	-	-
LOOP	非保留	-	-
LOWER	-	非保留	保留
M	-	非保留	-
MAP	-	保留	-
MAPPING	非保留	-	-
MASKING	非保留	-	-
MASTER	非保留	-	-
MATCH	非保留	保留	保留
MATCHED	非保留	-	-
MATERIALIZED	非保留	-	-
MAX	-	非保留	保留
MAXEXTENTS	非保留	-	-
MAXSIZE	非保留	-	-
MAXTRANS	非保留	-	-
MAXVALUE	保留	-	-
MERGE	非保留	-	-
MESSAGE_LENGTH	-	非保留	非保留
MESSAGE_OCTET_LENGTH	-	非保留	非保留
MESSAGE_TEXT	-	非保留	非保留
METHOD	-	非保留	-
MIN	-	非保留	保留
MINEXTENTS	非保留	-	-
MINUS	保留	-	-
MINUTE	非保留	保留	保留
MINVALUE	非保留	-	-
MOD	-	非保留	-
MODE	非保留	-	-
MODEL	非保留	-	-

关键字	GaussDB	SQL:1999	SQL-92
MODIFIES	-	保留	-
MODIFY	保留	保留	-
MODULE	-	保留	保留
MONTH	非保留	保留	保留
MORE	-	非保留	非保留
MOVE	非保留	-	-
MOVEMENT	非保留	-	-
MUMPS	-	非保留	非保留
NAME	非保留	非保留	非保留
NAMES	非保留	保留	保留
NATIONAL	非保留（不能是函数或类型）	保留	保留
NATURAL	保留（可以是函数或类型）	保留	保留
NCHAR	非保留（不能是函数或类型）	保留	保留
NCLOB	-	保留	-
NEW	-	保留	-
NEXT	非保留	保留	保留
NO	非保留	保留	保留
NOCOMPRESS	非保留	-	-
NOCYCLE	保留	-	-
NODE	非保留	-	-
NOENTITYESCAPING	非保留	-	-
NOLOGGING	非保留	-	-
NOMAXVALUE	非保留	-	-
NOMINVALUE	非保留	-	-
NONE	非保留（不能是函数或类型）	保留	-
NOT	保留	保留	保留
NOTHING	非保留	-	-
NOTIFY	非保留	-	-

关键字	GaussDB	SQL:1999	SQL-92
NOTNULL	保留（可以是函数或类型）	-	-
NOWAIT	非保留	-	-
NULL	保留	保留	保留
NULLABLE	-	非保留	非保留
NULLCOLS	非保留	-	-
NULLIF	非保留（不能是函数或类型）	非保留	保留
NULLS	非保留	-	-
NUMBER	非保留（不能是函数或类型）	非保留	非保留
NUMERIC	非保留（不能是函数或类型）	保留	保留
NUMSTR	非保留	-	-
NVARCHAR2	非保留（不能是函数或类型）	-	-
NVL	非保留（不能是函数或类型）	-	-
NVL2	非保留（不能是函数或类型）	-	-
OBJECT	非保留	保留	-
OCTET_LENGTH	-	非保留	保留
OF	非保留	保留	保留
OFF	非保留	保留	-
OFFSET	保留	-	-
OIDS	非保留	-	-
OLD	-	保留	-
ON	保留	保留	保留
ONLY	保留	保留	保留
OPEN	-	保留	保留
OPERATION	-	保留	-
OPERATOR	非保留	-	-
OPTIMIZATION	非保留	-	-

关键字	GaussDB	SQL:1999	SQL-92
OPTION	非保留	保留	保留
OPTIONALLY	非保留	-	-
OPTIONS	非保留	非保留	-
OR	保留	保留	保留
ORDER	保留	保留	保留
ORDINALITY	非保留	保留	-
OUT	非保留（不能是函数或类型）	保留	-
OUTER	保留（可以是函数或类型）	保留	保留
OUTFILE	非保留	-	-
OUTPUT	-	保留	保留
OVER	非保留	-	-
OVERLAPS	保留（可以是函数或类型）	非保留	保留
OVERLAY	非保留（不能是函数或类型）	非保留	-
OVERRIDING	-	非保留	-
OWNED	非保留	-	-
OWNER	非保留	-	-
PACKAGE	非保留	-	-
PACKAGES	非保留	-	-
PAD	-	保留	保留
PARAMETER	-	保留	-
PARAMETER_MODE	-	非保留	-
PARAMETER_NAME	-	非保留	-
PARAMETER_ORDINAL_POSITION	-	非保留	-
PARAMETER_SPECIFIC_CATALOG	-	非保留	-
PARAMETER_SPECIFIC_NAME	-	非保留	-

关键字	GaussDB	SQL:1999	SQL-92
PARAMETER_SPECIFIC_SCHEMA	-	非保留	-
PARAMETERS	-	保留	-
PARSER	非保留	-	-
PARTIAL	非保留	保留	保留
PARTITION	非保留	-	-
PARTITIONS	非保留	-	-
PASCAL	-	非保留	非保留
PASSING	非保留	-	-
PASSWORD	非保留	-	-
PATH	-	保留	-
PCTFREE	非保留	-	-
PER	非保留	-	-
PERCENT	非保留	-	-
PERFORMANCE	保留	-	-
PERM	非保留	-	-
PIVOT	非保留	-	-
PLACING	保留	-	-
PLAN	非保留	-	-
PLANS	非保留	-	-
PLI	-	非保留	非保留
POLICY	非保留	-	-
POOL	非保留	-	-
POSITION	非保留（不能是函数或类型）	非保留	保留
POSTFIX	-	保留	-
PRECEDING	非保留	-	-
PRECISION	非保留（不能是函数或类型）	保留	保留
PREDICT	非保留	-	-
PREFERRED	非保留	-	-

关键字	GaussDB	SQL:1999	SQL-92
PREFIX	非保留	保留	-
PREORDER	-	保留	-
PREPARE	非保留	保留	保留
PREPARED	非保留	-	-
PRESERVE	非保留	保留	保留
PRIMARY	保留	保留	保留
PRIOR	非保留	保留	保留
PRIORER	保留	-	-
PRIVATE	非保留	-	-
PRIVILEGE	非保留	-	-
PRIVILEGES	非保留	保留	保留
PROCEDURAL	非保留	-	-
PROCEDURE	保留	保留	保留
PROFILE	非保留	-	-
PUBLIC	非保留	保留	保留
PUBLISH	非保留	-	-
PURGE	非保留	-	-
QUERY	非保留	-	-
QUOTE	非保留	-	-
RANDOMIZED	非保留	-	-
RANGE	非保留	-	-
RATIO	非保留	-	-
RAW	非保留	-	-
READ	非保留	保留	保留
READS	-	保留	-
REAL	非保留（不能是函数或类型）	保留	保留
REASSIGN	非保留	-	-
REBUILD	非保留	-	-
RECHECK	非保留	-	-

关键字	GaussDB	SQL:1999	SQL-92
RECOVER	非保留	-	-
RECURSIVE	非保留	保留	-
RECYCLEBIN	保留（可以是函数或类型）	-	-
REDISANYVALUE	非保留	-	-
REF	非保留	保留	-
REFERENCES	保留	保留	保留
REFERENCING	-	保留	-
REFRESH	非保留	-	-
REGEXP_LIKE	非保留（不能是函数或类型）	-	-
REINDEX	非保留	-	-
REJECT	保留	-	-
RELATIVE	非保留	保留	保留
RELEASE	非保留	-	-
RELOPTIONS	非保留	-	-
REMOTE	非保留	-	-
REMOVE	非保留	-	-
RENAME	非保留	-	-
REPEATABLE	非保留	非保留	非保留
REPLACE	非保留	-	-
REPLICA	非保留	-	-
RESET	非保留	-	-
RESIZE	非保留	-	-
RESOURCE	非保留	-	-
RESTART	非保留	-	-
RESTRICT	非保留	保留	保留
RESULT	-	保留	-
RETURN	非保留	保留	-
RETURNED_LENGTH	-	非保留	非保留

关键字	GaussDB	SQL:1999	SQL-92
RETURNED_OCTET_LENGTH	-	非保留	非保留
RETURNED_SQLSTATE	-	非保留	非保留
RETURNING	保留	-	-
RETURNS	非保留	保留	-
REUSE	非保留	-	-
REVOKE	非保留	保留	保留
RIGHT	保留（可以是函数或类型）	保留	保留
ROLE	非保留	保留	-
ROLES	非保留	-	-
ROLLBACK	非保留	保留	保留
ROLLUP	非保留	保留	-
ROTATION	非保留	-	-
ROUTINE	-	保留	-
ROUTINE_CATALOG	-	非保留	-
ROUTINE_NAME	-	非保留	-
ROUTINE_SCHEMA	-	非保留	-
ROW	非保留（不能是函数或类型）	保留	-
ROW_COUNT	-	非保留	非保留
ROWNUM	保留	-	-
ROWS	非保留	保留	保留
ROWTYPE	非保留	-	-
RULE	非保留	-	-
SAMPLE	非保留	-	-
SAVEPOINT	非保留	保留	-
SCALE	-	非保留	非保留
SCHEDULE	非保留	-	-
SCHEMA	非保留	保留	保留
SCHEMA_NAME	-	非保留	非保留

关键字	GaussDB	SQL:1999	SQL-92
SCOPE	-	保留	-
SCROLL	非保留	保留	保留
SEARCH	非保留	保留	-
SECOND	非保留	保留	保留
SECTION	-	保留	保留
SECURITY	非保留	非保留	-
SELECT	保留	保留	保留
SELF	-	非保留	-
SENSITIVE	-	非保留	-
SEPARATOR	非保留	-	-
SEQUENCE	非保留	保留	-
SEQUENCES	非保留	-	-
SERIALIZABLE	非保留	非保留	非保留
SERVER	非保留	-	-
SERVER_NAME	-	非保留	非保留
SESSION	非保留	保留	保留
SESSION_USER	保留	保留	保留
SESSIONTIMEZONE	保留	-	-
SET	非保留	保留	保留
SETOF	非保留（不能是函数或类型）	-	-
SETS	非保留	保留	-
SHARE	非保留	-	-
SHIPPABLE	非保留	-	-
SHOW	非保留	-	-
SHRINK	保留	-	-
SHUTDOWN	非保留	-	-
SIBLINGS	非保留	-	-
SIMILAR	保留（可以是函数或类型）	非保留	-
SIMPLE	非保留	非保留	-

关键字	GaussDB	SQL:1999	SQL-92
SIZE	非保留	保留	保留
SKIP	非保留	-	-
SLAVE	非保留	-	-
SLICE	非保留	-	-
SMALLDATETIME	非保留（不能是函数或类型）	-	-
SMALLDATETIME_FORMAT	非保留	-	-
SMALLINT	非保留（不能是函数或类型）	保留	保留
SNAPSHOT	非保留	-	-
SOME	保留	保留	保留
SOURCE	非保留	非保留	-
SPACE	非保留	保留	保留
SPECIFIC	-	保留	-
SPECIFIC_NAME	-	非保留	-
SPECIFICATION	非保留	-	-
SPECIFICTYPE	-	保留	-
SPILL	非保留	-	-
SPLIT	非保留	-	-
SQL	-	保留	保留
SQLCODE	-	-	保留
SQLERROR	-	-	保留
SQLEXCEPTION	-	保留	-
SQLSTATE	-	保留	保留
SQLWARNING	-	保留	-
STABLE	非保留	-	-
STANDALONE	非保留	-	-
START	非保留	保留	-
STARTING	非保留	-	-
STARTS	非保留	-	-

关键字	GaussDB	SQL:1999	SQL-92
STATE	-	保留	-
STATEMENT	非保留	保留	-
STATEMENT_ID	非保留	-	-
STATIC	-	保留	-
STATISTICS	非保留	-	-
STDIN	非保留	-	-
STDOUT	非保留	-	-
STORAGE	非保留	-	-
STORE	非保留	-	-
STORED	非保留	-	-
STRATIFY	非保留	-	-
STRICT	非保留	-	-
STRIP	非保留	-	-
STRUCTURE	-	保留	-
STYLE	-	非保留	-
SUBCLASS_ORIGIN	-	非保留	非保留
SUBLIST	-	非保留	-
SUBPARTITION	非保留	-	-
SUBPARTITIONS	非保留	-	-
SUBSTRING	非保留（不能是函数或类型）	非保留	保留
SUM	-	非保留	保留
SYMMETRIC	保留	非保留	-
SYNONYM	非保留	-	-
SYS_REFCURSOR	非保留	-	-
SYSDATE	保留	-	-
SYSID	非保留	-	-
SYSTEM	非保留	非保留	-
SYSTEM_USER	-	保留	保留
TABLE	保留	保留	保留

关键字	GaussDB	SQL:1999	SQL-92
TABLE_NAME	-	非保留	非保留
TABLES	非保留	-	-
TABLESAMPLE	保留（可以是函数或类型）	-	-
TABLESPACE	非保留	-	-
TARGET	非保留	-	-
TEMP	非保留	-	-
TEMPLATE	非保留	-	-
TEMPORARY	非保留	保留	保留
TERMINATE	-	保留	-
TERMINATED	非保留	-	-
TEXT	非保留	-	-
THAN	非保留	保留	-
THEN	保留	保留	保留
TIME	非保留（不能是函数或类型）	保留	保留
TIME_FORMAT	非保留	-	-
TIMECAPSULE	保留（可以是函数或类型）	-	-
TIMESTAMP	非保留（不能是函数或类型）	保留	保留
TIMESTAMP_FORMAT	非保留	-	-
TIMESTAMPDIFF	非保留（不能是函数或类型）	-	-
TIMEZONE_HOUR	-	保留	保留
TIMEZONE_MINUTE	-	保留	保留
TINYINT	非保留（不能是函数或类型）	-	-
TO	保留	保留	保留
TRAILING	保留	保留	保留
TRANSACTION	非保留	保留	保留
TRANSACTION_ACTIVE	-	非保留	-

关键字	GaussDB	SQL:1999	SQL-92
TRANSACTIONS_COMMITTED	-	非保留	-
TRANSACTIONS_ROLLED_BACK	-	非保留	-
TRANSFORM	非保留	非保留	-
TRANSFORMS	-	非保留	-
TRANSLATE	-	非保留	保留
TRANSLATION	-	保留	保留
TREAT	非保留（不能是函数或类型）	保留	-
TRIGGER	非保留	保留	-
TRIGGER_CATALOG	-	非保留	-
TRIGGER_NAME	-	非保留	-
TRIGGER_SCHEMA	-	非保留	-
TRIM	非保留（不能是函数或类型）	非保留	保留
TRUE	保留	保留	保留
TRUNCATE	非保留	-	-
TRUSTED	非保留	-	-
TSFIELD	非保留	-	-
TSTAG	非保留	-	-
TSTIME	非保留	-	-
TYPE	非保留	非保留	非保留
TYPES	非保留	-	-
UNBOUNDED	非保留	-	-
UNCOMMITTED	非保留	非保留	非保留
UNDER	-	保留	-
UNENCRYPTED	非保留	-	-
UNION	保留	保留	保留
UNIQUE	保留	保留	保留
UNKNOWN	非保留	保留	保留
UNLIMITED	非保留	-	-

关键字	GaussDB	SQL:1999	SQL-92
UNLISTEN	非保留	-	-
UNLOCK	非保留	-	-
UNLOGGED	非保留	-	-
UNNAMED	-	非保留	非保留
UNNEST	-	保留	-
UNPIVOT	非保留	-	-
UNTIL	非保留	-	-
UNUSABLE	非保留	-	-
UPDATE	非保留	保留	保留
UPPER	-	非保留	保留
USAGE	-	保留	保留
USEEOF	非保留	-	-
USER	保留	保留	保留
USER_DEFINED_TYPE_CATALOG	-	非保留	-
USER_DEFINED_TYPE_NAME	-	非保留	-
USER_DEFINED_TYPE_SCHEMA	-	非保留	-
USING	保留	保留	保留
VACUUM	非保留	-	-
VALID	非保留	-	-
VALIDATE	非保留	-	-
VALIDATION	非保留	-	-
VALIDATOR	非保留	-	-
VALUE	非保留	保留	保留
VALUES	非保留（不能是函数或类型）	保留	保留
VARCHAR	非保留（不能是函数或类型）	保留	保留
VARCHAR2	非保留（不能是函数或类型）	-	-

关键字	GaussDB	SQL:1999	SQL-92
VARIABLE	-	保留	-
VARIABLES	非保留	-	-
VARIADIC	保留	-	-
VARYING	非保留	保留	保留
VCGROUP	非保留	-	-
VERBOSE	保留（可以是函数或类型）	-	-
VERIFY	保留	-	-
VERSION	非保留	-	-
VIEW	非保留	保留	保留
VOLATILE	非保留	-	-
WAIT	非保留	-	-
WEAK	非保留	-	-
WELLFORMED	非保留	-	-
WHEN	保留	保留	保留
WHENEVER	-	保留	保留
WHERE	保留	保留	保留
WHITESPACE	非保留	-	-
WINDOW	保留	-	-
WITH	保留	保留	保留
WITHIN	非保留	-	-
WITHOUT	非保留	保留	-
WORK	非保留	保留	保留
WORKLOAD	非保留	-	-
WRAPPER	非保留	-	-
WRITE	非保留	保留	保留
XML	非保留	-	-
XMLATTRIBUTES	非保留（不能是函数或类型）	-	-
XMLCONCAT	非保留（不能是函数或类型）	-	-

关键字	GaussDB	SQL:1999	SQL-92
XMLELEMENT	非保留（不能是函数或类型）	-	-
XMLEXISTS	非保留（不能是函数或类型）	-	-
XMLFOREST	非保留（不能是函数或类型）	-	-
XMLPARSE	非保留（不能是函数或类型）	-	-
XMLPI	非保留（不能是函数或类型）	-	-
XMLROOT	非保留（不能是函数或类型）	-	-
XMLSERIALIZE	非保留（不能是函数或类型）	-	-
XMLTYPE	非保留（不能是函数或类型）	-	-
YEAR	非保留	保留	保留
YES	非保留	-	-
ZONE	非保留	保留	保留

下表所示字段在建表时禁止作为列名。

CTID	XMIN	CMIN	XMAX	CMAX
TABLEOID	XC_NODE_ID	TID	-	GS_TUPLE_UID
TABLEBUCKETID	XC_NODE_HASH	-	-	-

7.3 数据类型

数据类型是数据的一个基本属性，用于区分不同类型的数据。不同的数据类型所占的存储空间不同，能够进行的操作也不相同。数据库中的数据存储在数据表中。数据表中的每一列都定义了数据类型，用户存储数据时，须遵从这些数据类型的属性，否则可能会出错。

GaussDB支持某些数据类型间的隐式转换，具体转换关系请参见**PG_CAST**。

7.3.1 数值类型

表7-2列出了所有的可用类型。数字操作符和相关的内置函数请参见数字操作函数和操作符。

表 7-2 整数类型

名称	描述	存储空间	范围
TINYINT	微整数，别名为INT1。	1字节	0 ~ +255
SMALLINT	小范围整数，别名为INT2。	2字节	-32,768 ~ +32,767
INTEGER	常用的整数，别名为INT4。	4字节	-2,147,483,648 ~ +2,147,483,647
BINARY_INTEGER	常用的整数INTEGER的别名。	4字节	-2,147,483,648 ~ +2,147,483,647
BIGINT	大范围的整数，别名为INT8。	8字节	-9,223,372,036,854,775,808 ~ +9,223,372,036,854,775,807
int16	十六字节的大范围整数，目前不支持用户用于建表等使用。	16字节	-170,141,183,460,469,231,731,687,303,715,884,105,728 ~ +170,141,183,460,469,231,731,687,303,715,884,105,727

示例：

```
--创建具有TINYINT类型数据的表。
gaussdb=# CREATE TABLE int_type_t1
         (
          IT_COL1 TINYINT
         );

--向创建的表中插入数据。
gaussdb=# INSERT INTO int_type_t1 VALUES(10);

--查看数据。
gaussdb=# SELECT * FROM int_type_t1;
 it_col1
---------
 10
(1 row)

--删除表。
gaussdb=# DROP TABLE int_type_t1;
--创建具有TINYINT,INTEGER,BIGINT类型数据的表。
gaussdb=# CREATE TABLE int_type_t2
(
    a TINYINT,
    b TINYINT,
    c INTEGER,
    d BIGINT
);

--插入数据。
```

```
gaussdb=# INSERT INTO int_type_t2 VALUES(100, 10, 1000, 10000);

--查看数据。
gaussdb=# SELECT * FROM int_type_t2;
  a  |b |  c  |   d
-----+----+------+-------
 100 | 10 | 1000 | 10000
(1 row)

--删除表。
gaussdb=# DROP TABLE int_type_t2;
```

说明

- TINYINT、SMALLINT、INTEGER、BIGINT和INT16类型存储各种范围的数字，也就是整数。如果存储超出范围以外的数值将会导致错误。
- 常用的类型是INTEGER，因为它提供了在范围、存储空间、性能之间的最佳平衡。一般只有取值范围确定不超过SMALLINT的情况下，才会使用SMALLINT类型。而只有在INTEGER的范围不够的时候才使用BIGINT，因为INTEGER的处理速度相对快得多。

表 7-3 任意精度类型

名称	描述	存储空间	范围
NUMERIC[(p[,s])], DECIMAL[(p[,s])]	精度p取值范围为[1,1000]，标度s取值范围为[0,p]。 **说明** p为总位数，s为小数位数。	用户声明精度。每四位（十进制位）占用两个字节，然后在整个数据上加上八个字节的额外开销。	未指定精度的情况下，小数点前最大131,072位，小数点后最大16,383位。
NUMBER[(p[,s])]	NUMERIC类型的别名。	用户声明精度。每四位（十进制位）占用两个字节，然后在整个数据上加上八个字节的额外开销。	未指定精度的情况下，小数点前最大131,072位，小数点后最大16,383位。

示例：

```
--创建表。
gaussdb=# CREATE TABLE decimal_type_t1
(
    DT_COL1 DECIMAL(10,4)
);

--插入数据。
gaussdb=# INSERT INTO decimal_type_t1 VALUES(123456.122331);

--查询表中的数据。
gaussdb=# SELECT * FROM decimal_type_t1;
   dt_col1
-------------
 123456.1223
(1 row)

--删除表。
gaussdb=# DROP TABLE decimal_type_t1;
--创建表。
gaussdb=# CREATE TABLE numeric_type_t1
(
    NT_COL1 NUMERIC(10,4)
```



```
);

--插入数据。
gaussdb=# INSERT INTO numeric_type_t1 VALUES(123456.12354);

--查询表中的数据。
gaussdb=# SELECT * FROM numeric_type_t1;
   nt_col1
-------------
 123456.1235
(1 row)

--删除表。
gaussdb=# DROP TABLE numeric_type_t1;
```

说明

- 与整数类型相比，任意精度类型需要更大的存储空间，其存储效率、运算效率以及压缩比效果都要差一些。在进行数值类型定义时，优先选择整数类型。当数值超出整数可表示最大范围时，再选用任意精度类型。
- 使用NUMERIC/DECIMAL进行列定义时，建议指定该列的精度p以及标度s。

表 7-4 序列整型

名称	描述	存储空间	范围
SMALLSERIAL	二字节序列整型。	2字节。	-32,768 ~ +32,767。
SERIAL	四字节序列整型。	4字节。	-2,147,483,648 ~ +2,147,483,647。
BIGSERIAL	八字节序列整型。	8字节。	-9,223,372,036,854,775,808 ~ +9,223,372,036,854,775,807。
LARGESERIAL	默认插入十六字节序列整型，实际数值类型和NUMERIC相同。	变长类型，每四位（十进制位）占用两个字节，然后在整个数据上加上八个字节的额外开销。	小数点前最大131,072位，小数点后最大16,383位。

示例：

```
--创建表。
gaussdb=# CREATE TABLE smallserial_type_tab(a SMALLSERIAL);

--插入数据。
gaussdb=# INSERT INTO smallserial_type_tab VALUES(default);

--再次插入数据。
gaussdb=# INSERT INTO smallserial_type_tab VALUES(default);

--查看数据。
gaussdb=# SELECT * FROM smallserial_type_tab;
 a
```



```
---
 1
 2
(2 rows)

--创建表。
gaussdb=# CREATE TABLE serial_type_tab(b SERIAL);

--插入数据。
gaussdb=# INSERT INTO serial_type_tab VALUES(default);

--再次插入数据。
gaussdb=# INSERT INTO serial_type_tab VALUES(default);

--查看数据。
gaussdb=# SELECT * FROM serial_type_tab;
 b
---
 1
 2
(2 rows)

--创建表。
gaussdb=# CREATE TABLE bigserial_type_tab(c BIGSERIAL);

--插入数据。
gaussdb=# INSERT INTO bigserial_type_tab VALUES(default);

--再次插入数据。
gaussdb=# INSERT INTO bigserial_type_tab VALUES(default);

--查看数据。
gaussdb=# SELECT * FROM bigserial_type_tab;
 c
---
 1
 2
(2 rows)

--创建表。
gaussdb=# CREATE TABLE largeserial_type_tab(c LARGESERIAL);

--插入数据。
gaussdb=# INSERT INTO largeserial_type_tab VALUES(default);

--再次插入数据。
gaussdb=# INSERT INTO largeserial_type_tab VALUES(default);

--查看数据。
gaussdb=# SELECT * FROM largeserial_type_tab;
 c
---
 1
 2
(2 rows)

--删除表。
gaussdb=# DROP TABLE smallserial_type_tab;

gaussdb=# DROP TABLE serial_type_tab;

gaussdb=# DROP TABLE bigserial_type_tab;

gaussdb=# DROP TABLE largeserial_type_tab;
```


说明

SMALLSERIAL、SERIAL、BIGSERIAL和LARGESERIAL类型不是真正的类型，只是为在表中设置唯一标识做的概念上的便利。因此，创建一个整数字段，并且把它的缺省数值安排为从一个序列发生器读取。应用了一个NOT NULL约束以确保NULL不会被插入。在大多数情况下用户可能还希望附加一个UNIQUE或PRIMARY KEY约束避免意外地插入重复的数值，但这个不是自动的。最后，序列发生器将从属于该字段，这样当该字段或表被删除的时候也一并删除它。目前支持在创建表时候指定SERIAL列，也支持在PG兼容模式下的普通表增加SERIAL列。另外临时表也不支持创建SERIAL列。因为SERIAL不是真正的类型，所以也不可以将表中存在的列类型转化为SERIAL。

表 7-5 浮点类型

名称	描述	存储空间	范围
REAL, FLOAT4	单精度浮点数，不精准。	4字节。	-3.402E+38~+3.402E+38，6位十进制数字精度。
DOUBLE PRECISION, FLOAT8	双精度浮点数，不精准。	8字节。	-1.79E+308~+1.79E+308，15位十进制数字精度。
FLOAT[(p)]	浮点数，不精准。精度p取值范围为[1,53]。	4字节或8字节。	根据精度p不同选择REAL或DOUBLE PRECISION作为内部表示。如不指定精度，内部用DOUBLE PRECISION表示。
BINARY_DOUBLE	是DOUBLE PRECISION的别名，为兼容Oracle数据类型。	8字节。	-1.79E+308~+1.79E+308，15位十进制数字精度。
DEC[(p[,s])]	精度p取值范围为[1,1000]，标度s取值范围为[0,p]。	用户声明精度。每四位（十进制位）占用两个字节，然后在整个数据上加上八个字节的额外开销。	在精度和标度指定最大的情况下，小数点前最大131,072位，小数点后最大16,383位。

名称	描述	存储空间	范围
INTEGER[(p[,s])]	精度p取值范围为[1,1000]，标度s取值范围为[0,p]。 在未指定精度和标度的情况下，默认精度p为10，标度s为0。 未指定精度和标度的情况下，该类型映射为INTEGER。指定精度和标度的情况下，该类型映射为NUMERIC。	用户声明精度。每四位（十进制位）占用两个字节，然后在整个数据上加上八个字节的额外开销。	在精度和标度指定最大的情况下，小数点前最大131,072位，小数点后最大16,383位。 未指定精度和标度的情况下，范围是-2,147,483,648 ~ +2,147,483,647。

说明

- 二进制浮点数据类型REAL、FLOAT4、DOUBLE、DOUBLE PRECISION、FLOAT8、FLOAT[(p)]和BINARY_DOUBLE为不精确的数值类型，其内部存储为近似值，因此存储和检索时可能会显示轻微的差异。当用户在使用二进制浮点数据类型时需要注意以下几点：
  - 精确存储和计算：如果需要精确存储和计算（例如货币金额），请改用精确的数据类型（例如numeric）。
  - 复杂计算：若使用不精确的数据类型执行复杂计算以获得重要数据，需要仔细评估其结果。
  - 浮点数比较：比较两个浮点数是否相等的结果可能与预期存在差异。
  - 下溢错误：如果一个浮点数过于接近零，反而无法准确表示，会导致下溢错误。
- 表7-5中描述的p为精度，表示整数位最低可以接受的总位数；s为小数位位数。

示例：

```
--创建表。
gaussdb=# CREATE TABLE float_type_t2
(
    FT_COL1 INTEGER,
    FT_COL2 FLOAT4,
    FT_COL3 FLOAT8,
    FT_COL4 FLOAT(3),
    FT_COL5 BINARY_DOUBLE,
    FT_COL6 DECIMAL(10,4),
    FT_COL7 INTEGER(6,3)
);

--插入数据。
gaussdb=# INSERT INTO float_type_t2 VALUES(10,10.365456,123456.1234,10.3214, 321.321, 123.123654,
123.123654);

--查看数据。
gaussdb=# SELECT * FROM float_type_t2 ;
 ft_col1 | ft_col2 |   ft_col3   | ft_col4 | ft_col5 | ft_col6  | ft_col7
---------+---------+-------------+---------+---------+----------+---------
      10 | 10.3655 | 123456.1234 | 10.3214 | 321.321 | 123.1237 | 123.124
(1 row)

--删除表。
gaussdb=# DROP TABLE float_type_t2;
```

7.3.2 货币类型

货币类型存储带有固定小数精度的货币金额。

表7-6中显示的范围假设有两位小数。可以以任意格式输入，包括整型、浮点型或者典型的货币格式（如“$1,000.00”）。根据区域字符集，输出一般是最后一种形式。

表 7-6 货币类型

名称	描述	存储容量	范围
money	货币金额	8 字节	-92233720368547758.08 ~ +92233720368547758.07

numeric、int和bigint类型的值可以转换为money类型。如果从real和double precision类型转换到money类型，可以先转换为numeric类型，再转换为money类型，例如：

```
gaussdb=# SELECT '12.34'::float8::numeric::money;
 money
--------
 $12.34
(1 row)
```

这种用法是不推荐使用的。浮点数不应该用来处理货币类型，因为小数点的位数可能会导致错误。

money类型的值可以转换为numeric类型而不丢失精度。转换为其他类型可能丢失精度，并且必须通过以下两步来完成：

```
gaussdb=# SELECT '52093.89'::money::numeric::float8;
  float8
----------
 52093.89
(1 row)
```

当一个money类型的值除以另一个money类型的值时，结果是double precision（即一个纯数字，而不是money类型）；在运算过程中货币单位相互抵消。

7.3.3 布尔类型

GaussDB支持的布尔类型请参见表7-7

表 7-7 布尔类型

名称	描述	存储空间	取值
BOOLEAN	布尔类型	1字节。	• true：真 • false：假 • null：未知（unknown）

- “真”值的有效文本值是：
  TRUE、't'、'true'、'y'、'yes'、'1' 、'TRUE'、true、on以及所有非0数值。

- “假”值的有效文本值是：
  FALSE、'f'、'false'、'n'、'no'、'0'、0、'FALSE'、false、off。

使用TRUE和FALSE是比较规范的用法（也是SQL兼容的用法）。

示例

显示用字母t和f输出Boolean值。

```
--创建表。
gaussdb=# CREATE TABLE bool_type_t1
(
    BT_COL1 BOOLEAN,
    BT_COL2 TEXT
);

--插入数据。
gaussdb=# INSERT INTO bool_type_t1 VALUES (TRUE, 'sic est');

gaussdb=# INSERT INTO bool_type_t1 VALUES (FALSE, 'non est');

--查看数据。
gaussdb=# SELECT * FROM bool_type_t1;
 bt_col1 | bt_col2
---------+---------
 t       | sic est
 f       | non est
(2 rows)

gaussdb=# SELECT * FROM bool_type_t1 WHERE bt_col1 = 't';
 bt_col1 | bt_col2
---------+---------
 t       | sic est
(1 row)

--删除表。
gaussdb=# DROP TABLE bool_type_t1;
```

7.3.4 字符类型

GaussDB支持的字符类型请参见**表7-8**。字符串操作符和相关的内置函数请参见**字符处理函数和操作符**。

表 7-8 字符类型

名称	描述	存储空间
CHAR(n) CHARACTER(n) NCHAR(n)	定长字符串，不足补空格。n是指字节长度，如不带精度n，默认精度为1。	最大为10MB。

名称	描述	存储空间
VARCHAR(n) CHARACTER VARYING(n)	变长字符串。PG兼容模式下，n是字符长度。其他兼容模式下，n是指字节长度。	n最大为10485760（即10MB）。 不带n时，最大存储字节长度为1GB-85-4(存储长度参数的空间)-其余列长度，比如表格式为(a int, b varchar, c int)，varchar最大长度为1GB-85-4(存储长度参数的空间)-4(a列int的长度)-4(c列int的长度)=1,073,741,727。详情请参见**变长类型最大存储长度说明示例。**
VARCHAR2(n)	变长字符串。是VARCHAR(n)类型的别名。n是指字节长度。	n最大为10485760（即10MB）。 不带n时，最大存储字节长度为1GB-85-4(存储长度参数的空间)-其余列长度，比如表格式为(a int, b varchar2, c int)，varchar2最大长度为1GB-85-4(存储长度参数的空间)-4(a列int的长度)-4(c列int的长度)=1,073,741,727。详情请参见**变长类型最大存储长度说明示例。**
NVARCHAR2(n)	变长字符串。在SQL_ASCII字符集下，n表示字节；在非SQL_ASCII字符集下，n表示字符。	n最大为10485760（即10MB）。 不带n时，最大存储字节长度为1GB-85-4(存储长度参数的空间)-其余列长度，比如表格式为(a int, b nvarchar2, c int)，nvarchar2最大长度为1GB-85-4(存储长度参数的空间)-4(a列int的长度)-4(c列int的长度)=1,073,741,727。详情请参见**变长类型最大存储长度说明示例。**
TEXT	变长字符串。	最大存储字节长度为1GB-85-4(存储长度参数的空间)-其余列长度，比如表格式为(a int, b text, c int)，text最大长度为1GB-85-4(存储长度参数的空间)-4(a列int的长度)-4(c列int的长度)=1,073,741,727。详情请参见**变长类型最大存储长度说明示例。**

名称	描述	存储空间
CLOB	文本大对象。是TEXT类型的别名。	在astore下，最大为32TB-1，但还需要考虑到列描述头信息的大小，以及列所在元组的大小限制（也小于32TB-1），因此CLOB类型最大值可能小于32TB-1。 在ustore下，最大为1GB-1，但还需要考虑到列描述头信息的大小，以及列所在元组的大小限制（也小于1GB-1），因此CLOB类型最大值可能小于1GB-1。

说明

1. 除了每列的大小限制以外，每个元组的总大小也不可超过1GB-1字节，主要受列的控制头信息、元组控制头信息以及元组中是否存在NULL字段等影响。
2. NCHAR为bpchar类型的别名，VARCHAR2(n)为VARCHAR(n)类型的别名。
3. 超过1GB的clob只有dbe_lob相关高级包支持，系统函数不支持大于1GB clob。
4. 在A兼容性下，默认将接收到的空字符串转换为null。
5. GaussDB最大支持1GB数据传输，函数返回结果字符串最大支持1GB。
6. GaussDB目前所支持的字符集可以参考**CREATE DATABASE**章节中ENCODING部分介绍。

在GaussDB里另外还有两种定长字符类型。在**表7-9**里显示。name类型只用在内部系统表中，作为存储标识符，不建议普通用户使用。该类型长度当前定为64字节（63可用字符加结束符）。类型"char"只用了一个字节的存储空间。它在系统内部主要用于系统表，主要作为简单化的枚举类型使用。

表 7-9 特殊字符类型

名称	描述	存储空间
name	用于对象名的内部类型。	64字节。
"char"	单字节内部类型。	1字节。

示例

- 插入的数据长度超过类型规定的长度的示例。

```
--创建表。
gaussdb=# CREATE TABLE char_type_t1
(
    CT_COL1 CHARACTER(4)
);

--插入数据。
gaussdb=# INSERT INTO char_type_t1 VALUES ('ok');

--查询表中的数据。
gaussdb=# SELECT ct_col1, char_length(ct_col1) FROM char_type_t1;
 ct_col1 | char_length
---------+-------------
 ok      |           4
```

```
(1 row)

--删除表。
gaussdb=# DROP TABLE char_type_t1;

--创建表。
gaussdb=# CREATE TABLE char_type_t2
(
    CT_COL1 VARCHAR(5)
);

--插入数据。
gaussdb=# INSERT INTO char_type_t2 VALUES ('ok');

gaussdb=# INSERT INTO char_type_t2 VALUES ('good');

--插入的数据长度超过类型规定的长度报错。
gaussdb=# INSERT INTO char_type_t2 VALUES ('too long');
ERROR:  value too long for type character varying(5)
CONTEXT:  referenced column: ct_col1

--明确类型的长度，超过数据类型长度后会自动截断。
gaussdb=# INSERT INTO char_type_t2 VALUES ('too long'::varchar(5));

--查询数据。
gaussdb=# SELECT ct_col1, char_length(ct_col1) FROM char_type_t2;
 ct_col1 | char_length
---------+-------------
 ok      |           2
 good    |           4
 too l   |           5
(3 rows)

--删除数据。
gaussdb=# DROP TABLE char_type_t2;
```

- 变长类型最大存储长度说明示例。

  此示例以varchar为例，varchar2/nvarchar/nvarchar2/text同理。

```
-- 创建表，表中有三列，分别为int、varchar、int，根据计算规则，varchar最大存储长度为1GB-85-4-4-4=1073741727。
gaussdb=# CREATE TABLE varchar_maxlength_test1 (a int, b varchar, c int);

-- varchar为1073741728，超过规定长度，插入失败。
gaussdb=# insert into varchar_maxlength_test1 values(1, repeat('a', 1073741728), 1);
ERROR: invalid memory alloc request size 1073741824 in tuplesort.cpp:219

-- varchar为1073741727，长度符合要求，插入成功。
gaussdb=# insert into varchar_maxlength_test1 values(1, repeat('a', 1073741727), 1);

-- 创建表，表中仅varchar一列，根据计算规则，varchar最大存储长度为1GB-85-4=1073741735。
gaussdb=# CREATE TABLE varchar_maxlength_test2 (a varchar);

-- varchar为1073741736，超过规定长度，插入失败。
insert into varchar_maxlength_test2 values(repeat('a', 1073741736));
ERROR:  invalid memory alloc request size 1073741824 in tuplesort.cpp:219

-- varchar为1073741735，长度符合要求，插入成功。
insert into varchar_maxlength_test2 values(repeat('a', 1073741735));
```

7.3.5 二进制类型

GaussDB支持的二进制类型请参见**表7-10**。

表 7-10 二进制类型

名称	描述	存储空间
BLOB	二进制大对象。 目前BLOB支持的外部存取接口仅为： • DBE_LOB.GET_LENGTH • DBE_LOB.READ • DBE_LOB.WRITE • DBE_LOB.WRITE_APPEND • DBE_LOB.COPY • DBE_LOB.ERASE 这些接口详细说明请参见**DBE_LOB**。	在astore下，最大为32TB-1，但还需要考虑到列描述头信息的大小，以及列所在元组的大小限制（也小于32TB-1），因此BLOB类型最大值可能小于32TB-1。 在ustore下，最大为1GB-1，但还需要考虑到列描述头信息的大小，以及列所在元组的大小限制（也小于1GB-1），因此BLOB类型最大值可能小于1GB-1。
RAW	变长的十六进制类型	4字节加上实际的二进制字符串。最大为1GB-1，但还需要考虑到列描述头信息的大小，以及列所在元组的大小限制（也小于1GB-1），因此类型最大值可能小于1GB-1。
BYTEA	变长的二进制字符串。	4字节加上实际的二进制字符串。最大为1GB-1，但还需要考虑到列描述头信息的大小，以及列所在元组的大小限制（也小于1GB-1），因此类型最大值可能小于1GB-1。
BYTEAWITHOUTORDERWITHEQUALCOL	变长的二进制字符串（密态特性新增的类型，如果加密列的加密类型指定为确定性加密，则该列的实际类型为BYTEAWITHOUTORDERWITHEQUALCOL），元命令打印加密表将显示原始数据类型。	4字节加上实际的二进制字符串。最大为1GB减去53字节（即1073741771字节）。
BYTEAWITHOUTORDERCOL	变长的二进制字符串（密态特性新增的类型，如果加密列的加密类型指定为随机加密，则该列的实际类型为BYTEAWITHOUTORDERCOL），元命令打印加密表将显示原始数据类型。	4字节加上实际的二进制字符串。最大为1GB减去53字节（即1073741771字节）。
_BYTEAWITHOUTORDERWITHEQUALCOL	变长的二进制字符串，密态特性新增的类型。	4字节加上实际的二进制字符串。最大为1GB减去53字节（即1073741771字节）。

名称	描述	存储空间
_BYTEAWITHOUTORDERCOL	变长的二进制字符串，密态特性新增的类型。	4字节加上实际的二进制字符串。最大为1GB减去53字节（即1073741771字节）。

说明

- 除了每列的大小限制以外，每个元组的总大小也不可超过1GB-1字节。
- 不支持直接使用BYTEAWITHOUTORDERWITHEQUALCOL、BYTEAWITHOUTORDERCOL、_BYTEAWITHOUTORDERWITHEQUALCOL和_BYTEAWITHOUTORDERCOL类型创建表。
- RAW(n)，n是指字节长度建议值，不会用于校验输入RAW类型的字节长度。
- GaussDB最大支持1GB数据传输，函数返回结果字符串最大支持1GB。

示例:

```
--创建表。
gaussdb=# CREATE TABLE blob_type_t1
(
    BT_COL1 INTEGER,
    BT_COL2 BLOB,
    BT_COL3 RAW,
    BT_COL4 BYTEA
) ;

--插入数据。
gaussdb=# INSERT INTO blob_type_t1 VALUES(10,empty_blob(),
HEXTORAW('DEADBEEF'),E'\\xDEADBEEF');

--查询表中的数据。
gaussdb=# SELECT * FROM blob_type_t1;
 bt_col1 | bt_col2 | bt_col3  |  bt_col4
---------+---------+----------+------------
      10 |         | DEADBEEF | \xdeadbeef
(1 row)

--删除表。
gaussdb=# DROP TABLE blob_type_t1;
```

7.3.6 日期/时间类型

GaussDB支持的日期/时间类型请参见**表7-11**。该类型的操作符和内置函数请参见**时间和日期处理函数和操作符**。

说明

如果其他的数据库时间格式和GaussDB的时间格式不一致，可通过修改配置参数DateStyle的值来保持一致。

表 7-11 日期/时间类型

名称	描述	存储空间
DATE	日期。 ● 最小值：公元前4713年，4713-01-01BC。 ● 最大值：公元5874897年，5874897-12-31AD。 **说明** A兼容性下，数据库将空字符串作为NULL处理，数据类型DATE会被替换为TIMESTAMP(0) WITHOUT TIME ZONE。	4字节（兼容模式A下存储空间大小为8字节）
TIME [(p)] [WITHOUT TIME ZONE]	只用于一日内时间。 p表示小数点后的精度，取值范围为0~6。 ● 最小值：00:00:00。 ● 最大值：24:00:00。	8字节
TIME [(p)] [WITH TIME ZONE]	只用于一日内时间，带时区。 p表示小数点后的精度，取值范围为0~6。 ● 最小值：00:00:00+1559。 ● 最大值：24:00:00。	12字节
TIMESTAMP[(p)] [WITHOUT TIME ZONE]	日期和时间。 p表示小数点后的精度，取值范围为0~6。 ● 最小值：公元前4713年，4713-11-24BC 00:00:00.000000。 ● 最大值：公元294277年，294277-01-09AD 00:00:00.000000。	8字节

名称	描述	存储空间
TIMESTAMP[(p)] [WITH TIME ZONE]	日期和时间，带时区。TIMESTAMP的别名为TIMESTAMPTZ。 p表示小数点后的精度，取值范围为0~6。 ● 最小值：公元前4713年，4713-11-24BC 00:00:00.000000。 ● 最大值：公元294277年，294277-01-09AD 00:00:00.000000。 时区更新：部分国家或地区因为政治、经济、战争等因素经常会更新时区信息， 数据库系统也因此需要同步修改时区文件以确保时间内容的正确性。 GaussDB时区类型目前只涉及timestamp with timezone，当新的时区文件生效时，不会对已有的数据进行变更，新数据会随时区文件信息进行同步调整。	8字节
SMALLDATETIME	日期和时间，不带时区。 精确到分钟，秒位大于等于30秒进一位。 ● 最小值：公元前4713年，4713-11-24BC 00:00:00.000000。 ● 最大值：公元294277年，294277-01-09AD 00:00:00.000000。	8字节
INTERVAL DAY (l) TO SECOND (p)	时间间隔，X天X小时X分X秒。 ● l：天数的精度，取值范围为0~6。兼容性考虑，目前未实现具体功能。 ● p：秒数的精度，取值范围为0~6。小数末尾的零不显示。	16字节

名称	描述	存储空间
INTERVAL [FIELDS] [(p)]	时间间隔。 ● FIELDS：可以是YEAR，MONTH，DAY，HOUR，MINUTE，SECOND，DAY TO HOUR，DAY TO MINUTE，DAY TO SECOND，HOUR TO MINUTE，HOUR TO SECOND，MINUTE TO SECOND。 ● p：秒数的精度，取值范围为0~6，且fields为SECOND，DAY TO SECOND，HOUR TO SECOND或MINUTE TO SECOND时，参数p才有效。小数末尾的零不显示。	12字节
reltime	相对时间间隔。 ● 格式为：X years X mons X days XX:XX:XX。 ● 采用儒略历计时，规定一年为365.25天，一个月为30天，计算输入值对应的相对时间间隔。	4字节
abstime	日期和时间。 ● 格式为：YYYY-MM-DD hh:mm:ss +timezone。 ● 取值范围为1901-12-13 20:45:53 GMT~2038-01-18 23:59:59 GMT，精度为秒。	4字节

说明

在参数a_format_version值为10c和a_format_dev_version值为s1的情况下，缺省的DATE值由以下确定：

- 年通过SYSDATE返回当年。
- 月通过SYSDATE返回当月。
- 日返回01（月份的第一天）。
- 小时，分钟，秒都是0。

示例：

```
--创建表。
gaussdb=# CREATE TABLE date_type_tab(coll date);

--插入数据。
gaussdb=# INSERT INTO date_type_tab VALUES (date '12-10-2010');

--查看数据。
gaussdb=# SELECT * FROM date_type_tab;
       coll
---------------------
```



```
 2010-12-10 00:00:00
(1 row)

--删除表。
gaussdb=# DROP TABLE date_type_tab;

--创建表。
gaussdb=# CREATE TABLE time_type_tab (da time without time zone ,dai time with time zone,dfgh
timestamp without time zone,dfga timestamp with time zone, vbg smalldatetime);

--插入数据。
gaussdb=# INSERT INTO time_type_tab VALUES ('21:21:21','21:21:21 pst','2010-12-12','2013-12-11
pst','2003-04-12 04:05:06');

--查看数据。
gaussdb=# SELECT * FROM time_type_tab;
    da    |     dai     |        dfgh         |          dfga          |         vbg
----------+-------------+---------------------+------------------------+---------------------
 21:21:21 | 21:21:21-08 | 2010-12-12 00:00:00 | 2013-12-11 16:00:00+08 | 2003-04-12 04:05:00
(1 row)

--删除表。
gaussdb=# DROP TABLE time_type_tab;

--创建表。
gaussdb=# CREATE TABLE day_type_tab (a int,b INTERVAL DAY(3) TO SECOND (4));

--插入数据。
gaussdb=# INSERT INTO day_type_tab VALUES (1, INTERVAL '3' DAY);

--查看数据。
gaussdb=# SELECT * FROM day_type_tab;
 a |   b
---+--------
 1 | 3 days
(1 row)

--删除表。
gaussdb=# DROP TABLE day_type_tab;

--创建表。
gaussdb=# CREATE TABLE year_type_tab(a int, b interval year (6));

--插入数据。
gaussdb=# INSERT INTO year_type_tab VALUES(1,interval '2' year);

--查看数据。
gaussdb=# SELECT * FROM year_type_tab;
 a |    b
---+---------
 1 | 2 years
(1 row)

--删除表。
gaussdb=# DROP TABLE year_type_tab;
```

日期输入

日期和时间的输入几乎可以是任何合理的格式，包括ISO-8601格式、SQL-兼容格式、传统POSTGRES格式或者其它的格式。系统支持按照日、月、年的顺序自定义日期输入。如果把DateStyle参数设置为MDY就按照“月-日-年”解析，设置为DMY就按照“日-月-年”解析，设置为YMD就按照“年-月-日”解析。

日期的文本输入需要加单引号包围，语法如下：

type [(p)] 'value'

可选的精度声明中的p是一个整数，表示在秒域中小数部分的位数。表7-12显示了date类型的输入格式。

表 7-12 日期输入格式

例子	描述
1999-01-08	ISO 8601格式（建议格式），任何方式下都是1999年1月8日。
January 8, 1999	在任何datestyle输入模式下都无歧义。
1/8/1999	有歧义，在MDY模式下是1月8日，在DMY模式下是8月1日。
1/18/1999	MDY模式下是1月18日，其它模式下被拒绝。
01/02/03	• MDY模式下的2003年1月2日。 • DMY模式下的2003年2月1日。 • YMD模式下的2001年2月3日。
1999-Jan-08	任何模式下都是1月8日。
Jan-08-1999	任何模式下都是1月8日。
08-Jan-1999	任何模式下都是1月8日。
99-Jan-08	YMD模式下是1月8日，否则错误。
08-Jan-99	一月八日，除了在YMD模式下是错误的之外。
Jan-08-99	一月八日，除了在YMD模式下是错误的之外。
19990108	ISO 8601格式，任何模式下都是1999年1月8日。
990108	ISO 8601格式，任何模式下都是1999年1月8日。
1999.008	年和年里的第几天。
J2451187	儒略日。
January 8, 99 BC	公元前99年。

示例：

```
--创建表。
gaussdb=# CREATE TABLE date_type_tab(coll date);

--插入数据。
gaussdb=# INSERT INTO date_type_tab VALUES (date '12-10-2010');

--查看数据。
gaussdb=# SELECT * FROM date_type_tab;
        coll
---------------------
 2010-12-10
(1 row)

--查看日期格式。
gaussdb=# SHOW datestyle;
```



```
 DateStyle
-----------
 ISO, MDY
(1 row)

--设置日期格式。
gaussdb=# SET datestyle='YMD';
SET

--插入数据。
gaussdb=# INSERT INTO date_type_tab VALUES(date '2010-12-11');

--查看数据。
gaussdb=# SELECT * FROM date_type_tab;
        coll
---------------------
 2010-12-10
 2010-12-11
(2 rows)

--删除表。
gaussdb=# DROP TABLE date_type_tab;
```

时间

时间类型包括time [(p)] without time zone和time [(p)] with time zone。如果只写time等效于time without time zone。

如果在time without time zone类型的输入中声明了时区，则会忽略这个时区。

时间输入类型的详细信息请参见**表7-13**，时区输入类型的详细信息请参见**表7-14**。

表 7-13 时间输入

例子	描述
05:06.8	ISO 8601
4:05:06	ISO 8601
4:05	ISO 8601
040506	ISO 8601
4:05 AM	与04:05一样，输入小时数必须<= 12
4:05 PM	与16:05一样，输入小时数必须<= 12
04:05:06.789-8	ISO 8601
04:05:06-08:00	ISO 8601
04:05-08:00	ISO 8601
040506-08	ISO 8601
04:05:06 PST	缩写的时区
2003-04-12 04:05:06 America/New_York	用名称声明的时区

表 7-14 时区输入

例子	描述
PST	太平洋标准时间（Pacific Standard Time）
America/New_York	完整时区名称
-8:00	ISO 8601与PST的偏移
-800	ISO 8601与PST的偏移
-8	ISO 8601与PST的偏移

示例：

```
gaussdb=# SELECT time '04:05:06';
   time
----------
 04:05:06
(1 row)

gaussdb=# SELECT time '04:05:06 PST';
   time
----------
 04:05:06
(1 row)

gaussdb=# SELECT time with time zone '04:05:06 PST';
   timetz
-------------
 04:05:06-08
(1 row)
```

特殊值

GaussDB支持几个特殊值，在读取的时候将被转换成普通的日期/时间值，请参考**表 7-15**。

表 7-15 特殊值

输入字符串	适用类型	描述
epoch	date，timestamp	1970-01-01 00:00:00+00 （Unix系统零时）
infinity	timestamp	比任何其他时间戳都晚
-infinity	timestamp	比任何其他时间戳都早
now	date，time，timestamp	当前事务的开始时间
today	date，timestamp	今日午夜
tomorrow	date，timestamp	明日午夜
yesterday	date，timestamp	昨日午夜

输入字符串	适用类型	描述
allballs	time	00:00:00.00 UTC

示例：

```
--创建表。
gaussdb=# CREATE TABLE realtime_type_special(col1 varchar(20), col2 date, col3 timestamp, col4 time);

--插入数据。
gaussdb=# INSERT INTO realtime_type_special VALUES('epoch', 'epoch', 'epoch', NULL);
gaussdb=# INSERT INTO realtime_type_special VALUES('now', 'now', 'now', 'now');
gaussdb=# INSERT INTO realtime_type_special VALUES('today', 'today', 'today', NULL);
gaussdb=# INSERT INTO realtime_type_special VALUES('tomorrow', 'tomorrow', 'tomorrow', NULL);
gaussdb=# INSERT INTO realtime_type_special VALUES('yesterday', 'yesterday', 'yesterday', NULL);

--查看数据。
gaussdb=# SELECT * FROM realtime_type_special;
   col1    |        col2         |            col3             |      col4
-----------+---------------------+-----------------------------+-----------------
 epoch     | 1970-01-01 00:00:00 | 1970-01-01 00:00:00         |
 now       | 2023-02-27 11:38:13 | 2023-02-27 11:38:13.032815 | 11:38:13.032815
 today     | 2023-02-27 00:00:00 | 2023-02-27 00:00:00         |
 tomorrow  | 2023-02-28 00:00:00 | 2023-02-28 00:00:00         |
 yesterday | 2023-02-26 00:00:00 | 2023-02-26 00:00:00         |
(5 rows)

gaussdb=# SELECT * FROM realtime_type_special WHERE col3 < 'infinity';
   col1    |        col2         |            col3             |      col4
-----------+---------------------+-----------------------------+-----------------
 epoch     | 1970-01-01 00:00:00 | 1970-01-01 00:00:00         |
 now       | 2023-02-27 11:38:13 | 2023-02-27 11:38:13.032815 | 11:38:13.032815
 today     | 2023-02-27 00:00:00 | 2023-02-27 00:00:00         |
 tomorrow  | 2023-02-28 00:00:00 | 2023-02-28 00:00:00         |
 yesterday | 2023-02-26 00:00:00 | 2023-02-26 00:00:00         |
(5 rows)

gaussdb=# SELECT * FROM realtime_type_special WHERE col3 > '-infinity';
   col1    |        col2         |            col3             |      col4
-----------+---------------------+-----------------------------+-----------------
 epoch     | 1970-01-01 00:00:00 | 1970-01-01 00:00:00         |
 now       | 2023-02-27 11:38:13 | 2023-02-27 11:38:13.032815 | 11:38:13.032815
 today     | 2023-02-27 00:00:00 | 2023-02-27 00:00:00         |
 tomorrow  | 2023-02-28 00:00:00 | 2023-02-28 00:00:00         |
 yesterday | 2023-02-26 00:00:00 | 2023-02-26 00:00:00         |
(5 rows)

gaussdb=# SELECT * FROM realtime_type_special WHERE col3 > 'now';
   col1   |        col2         |        col3         | col4
----------+---------------------+---------------------+------
 tomorrow | 2023-02-28 00:00:00 | 2023-02-28 00:00:00 |
(1 row)

gaussdb=# SELECT * FROM realtime_type_special WHERE col3 = 'today';
 col1  |        col2         |        col3         | col4
-------+---------------------+---------------------+------
 today | 2023-02-27 00:00:00 | 2023-02-27 00:00:00 |
(1 row)

gaussdb=# SELECT * FROM realtime_type_special WHERE col3 = 'tomorrow';
   col1   |        col2         |        col3         | col4
----------+---------------------+---------------------+------
 tomorrow | 2023-02-28 00:00:00 | 2023-02-28 00:00:00 |
(1 row)

gaussdb=# SELECT * FROM realtime_type_special WHERE col3 > 'yesterday';
```

```
  col1   |        col2         |            col3            |      col4
---------+---------------------+----------------------------+-----------------
 now      | 2023-02-27 11:38:13 | 2023-02-27 11:38:13.032815 | 11:38:13.032815
 today    | 2023-02-27 00:00:00 | 2023-02-27 00:00:00        |
 tomorrow | 2023-02-28 00:00:00 | 2023-02-28 00:00:00        |
(3 rows)

gaussdb=# SELECT TIME 'allballs';
   time
----------
 00:00:00
(1 row)

--删除表。
gaussdb=# DROP TABLE realtime_type_special;
```

时间段输入

reltime的输入方式可以采用任何合法的时间段文本格式，包括数字形式（含负数和小数）及时间形式，其中时间形式的输入支持SQL标准格式、ISO-8601格式、POSTGRES格式等。另外，文本输入需要加单引号。

时间段输入的详细信息请参考**表7-16**。

表 7-16 时间段输入

输入示例	输出结果	描述
60	2 mons	采用数字表示时间段，默认单位是day，可以是小数或负数。特别的，负数时间段，在语义上，可以理解为“早于多久”。
31.25	1 mons 1 days 06:00:00	
-365	-12 mons -5 days	
1 years 1 mons 8 days 12:00:00	1 years 1 mons 8 days 12:00:00	采用POSTGRES格式表示时间段，可以正负混用，不区分大小写，输出结果为将输入时间段计算并转换得到的简化POSTGRES格式时间段。
-13 months -10 hours	-1 years -25 days -04:00:00	
-2 YEARS +5 MONTHS 10 DAYS	-1 years -6 mons -25 days -06:00:00	
P-1.1Y10M	-3 mons -5 days -06:00:00	采用ISO-8601格式表示时间段，可以正负混用，不区分大小写，输出结果为将输入时间段计算并转换得到的简化POSTGRES格式时间段。
-12H	-12:00:00	

示例：

```
--创建表。
gaussdb=# CREATE TABLE reltime_type_tab(col1 character(30), col2 reltime);

--插入数据。
gaussdb=# INSERT INTO reltime_type_tab VALUES ('90', '90');
gaussdb=# INSERT INTO reltime_type_tab VALUES ('-366', '-366');
gaussdb=# INSERT INTO reltime_type_tab VALUES ('1975.25', '1975.25');
gaussdb=# INSERT INTO reltime_type_tab VALUES ('-2 YEARS +5 MONTHS 10 DAYS', '-2 YEARS +5
```

```
MONTHS 10 DAYS');
gaussdb=# INSERT INTO reltime_type_tab VALUES ('30 DAYS 12:00:00', '30 DAYS 12:00:00');
gaussdb=# INSERT INTO reltime_type_tab VALUES ('P-1.1Y10M', 'P-1.1Y10M');

--查看数据。
gaussdb=# SELECT * FROM reltime_type_tab;
              col1              |                col2
--------------------------------+-----------------------------------
 90                             | 3 mons
 -366                           | -1 years -18:00:00
 1975.25                        | 5 years 4 mons 29 days
 -2 YEARS +5 MONTHS 10 DAYS     | -1 years -6 mons -25 days -06:00:00
 30 DAYS 12:00:00               | 1 mon 12:00:00
 P-1.1Y10M                      | -3 mons -5 days -06:00:00
(6 rows)

--删除表。
gaussdb=# DROP TABLE reltime_type_tab;
```

7.3.7 几何类型

GaussDB支持的几何类型请参见**表7-17**。最基本的类型：点，是其它类型的基础。

表 7-17 几何类型

名称	存储空间	说明	表现形式
point	16字节	平面中的点	(x,y)
lseg	32字节	（有限）线段	((x1,y1),(x2,y2))
box	32字节	矩形	((x1,y1),(x2,y2))
path	16+16n字节	闭合路径（与多边形相似）	((x1,y1),...)
path	16+16n字节	开放路径	[(x1,y1),...]
polygon	40+16n字节	多边形（与闭合路径相似）	((x1,y1),...)
circle	24 字节	圆	<(x,y),r>（圆心和半径）

GaussDB提供了一系列的函数和操作符用来进行各种几何计算，如拉伸、转换、旋转、计算相交等。详细信息请参考**几何函数和操作符**。

点

点是几何类型的基本二维构造单位。用下面语法描述point的数值：

```
( x , y )
x , y
```

x和y是用浮点数表示的点的坐标，点的数值类型为float8类型。

点输出使用第一种语法。

示例：

```
gaussdb=# select point(1.1, 2.2);
   point
-----------
```



```
 (1.1,2.2)
(1 row)
```

线段

线段（lseg）是用一对点来代表的。用下面的语法描述lseg的数值：

```
[ ( x1 , y1 ) , ( x2 , y2 ) ]
( ( x1 , y1 ) , ( x2 , y2 ) )
( x1 , y1 ) , ( x2 , y2 )
x1 , y1   ,   x2 , y2
```

(x1,y1)和(x2,y2)表示线段的端点，点的数值类型为float8类型。

线段输出使用第一种语法。

示例：

```
gaussdb=# select lseg(point(1.1, 2.2), point(3.3, 4.4));
        lseg
-----------------------
 [(1.1,2.2),(3.3,4.4)]
(1 row)
```

矩形

矩形是用一对对角点来表示的。用下面的语法描述box的值：

```
( ( x1 , y1 ) , ( x2 , y2 ) )
( x1 , y1 ) , ( x2 , y2 )
x1 , y1   ,   x2 , y2
```

(x1,y1)和(x2,y2)表示矩形的一对对角点，点的数值类型为float8类型。

矩形的输出使用第二种语法。

任何两个对角都可以出现在输入中，但按照该种顺序，右上角和左下角的值会被重新排序以存储。

示例：

```
gaussdb=# select box(point(1.1, 2.2), point(3.3, 4.4));
        box
----------------------
 (3.3,4.4),(1.1,2.2)
(1 row)
```

路径

路径由一系列连接的点组成。路径可能是开放的，也就是认为列表中第一个点和最后一个点没有连接，也可能是闭合的，这时认为第一个和最后一个点连接起来。

用下面的语法描述path的数值：

```
[ ( x1 , y1 ) , ... , ( xn , yn ) ]
( ( x1 , y1 ) , ... , ( xn , yn ) )
( x1 , y1 ) , ... , ( xn , yn )
( x1 , y1   , ... ,   xn , yn )
x1 , y1   , ... ,   xn , yn
```

点表示组成路径的线段的端点，点的数值类型为float8类型。方括号（[]）表明一个开放的路径，圆括号（()）表明一个闭合的路径。当最外层的括号被省略，如在第三至第五语法，会假定一个封闭的路径。

路径的输出使用第一种或第二种语法输出。

示例：

```
gaussdb=# select path(polygon '((0,0),(1,1),(2,0))');
        path
---------------------
 ((0,0),(1,1),(2,0))
(1 row)
```

多边形

多边形由一系列点代表（多边形的顶点）。多边形可以认为与闭合路径一样，但是存储方式不一样而且有自己的一套支持函数。

用下面的语法描述polygon的数值：

```
( ( x1 , y1 ) , ... , ( xn , yn ) )
( x1 , y1 ) , ... , ( xn , yn )
( x1 , y1   , ... ,   xn , yn )
x1 , y1   , ... ,   xn , yn
```

点表示多边形的顶点，点的数值类型为float8类型。

多边形输出使用第一种语法。

示例：

```
gaussdb=# select polygon(box '((0,0),(1,1))');
         polygon
---------------------------
 ((0,0),(0,1),(1,1),(1,0))
(1 row)
```

圆

圆由一个圆心和半径标识。用下面的格式描述circle的数值：

```
< ( x , y ) , r >
( ( x , y ) , r )
( x , y ) , r
x , y   , r
```

(x,y)表示圆心，r表示半径，点的数值类型为float8类型。

圆的输出用第一种格式。

示例：

```
gaussdb=# select circle(point(0,0),1);
  circle
----------
 <(0,0),1>
(1 row)
```

7.3.8 网络地址类型

GaussDB提供用于存储IPv4、MAC地址的数据类型。

用这些数据类型存储网络地址比用纯文本类型好，因为这些类型提供输入错误检查和特殊的操作和功能（请参见**网络地址函数和操作符**）。

表 7-18 网络地址类型

名称	存储空间	描述
cidr	7或19字节	IPv4网络
inet	7或19字节	IPv4主机和网络
macaddr	6字节	MAC地址

cidr

cidr（无类别域间路由，Classless Inter-Domain Routing）类型，保存一个IPv4网络地址。声明网络格式为address/y，address表示IPv4地址，y表示子网掩码的二进制位数。如果省略y，则掩码部分使用已有类别的网络编号系统进行计算，但要求输入的数据已经包括了确定掩码所需的所有字节。具体请参见**表7-19**

表 7-19 cidr 类型输入举例

cidr输入	cidr输出	abbrev（cidr）
192.168.100.128/25	192.168.100.128/25	192.168.100.128/25
192.168/24	192.168.0.0/24	192.168.0/24
192.168/25	192.168.0.0/25	192.168.0.0/25
192.168.1	192.168.1.0/24	192.168.1/24
192.168	192.168.0.0/24	192.168.0/24
10.1.2	10.1.2.0/24	10.1.2/24
10.1	10.1.0.0/16	10.1/16
10	10.0.0.0/8	10/8
10.1.2.3/32	10.1.2.3/32	10.1.2.3/32

示例：

```
gaussdb=# CREATE TABLE cidr_test(id int, c cidr);
CREATE TABLE
gaussdb=# INSERT INTO cidr_test VALUES (1, '192.168.100.128/25');
INSERT 0 1
gaussdb=# INSERT INTO cidr_test VALUES (2, '192.168/24');
INSERT 0 1
gaussdb=# INSERT INTO cidr_test VALUES (3, '192.168/25');
INSERT 0 1
gaussdb=# INSERT INTO cidr_test VALUES (4, '192.168.1');
INSERT 0 1
gaussdb=# INSERT INTO cidr_test VALUES (5, '192.168');
INSERT 0 1
gaussdb=# INSERT INTO cidr_test VALUES (6, '10.1.2');
INSERT 0 1
gaussdb=# INSERT INTO cidr_test VALUES (7, '10.1');
INSERT 0 1
gaussdb=# INSERT INTO cidr_test VALUES (8, '10');
```

```
INSERT 0 1
gaussdb=# INSERT INTO cidr_test VALUES (9, '2001:4f8:3:ba::/64');
INSERT 0 1
gaussdb=# INSERT INTO cidr_test VALUES (10, '2001:4f8:3:ba:2e0:81ff:fe22:d1f1/128');
INSERT 0 1
gaussdb=# INSERT INTO cidr_test VALUES (11, '::ffff:127.0.0.0/120');
INSERT 0 1
gaussdb=# INSERT INTO cidr_test VALUES (12, '::ffff:127.0.0.0/128');
INSERT 0 1
gaussdb=# SELECT * FROM cidr_test ORDER BY id;
 id |                  c
----+---------------------------------------
  1 | 192.168.100.128/25
  2 | 192.168.0.0/24
  3 | 192.168.0.0/25
  4 | 192.168.1.0/24
  5 | 192.168.0.0/24
  6 | 10.1.2.0/24
  7 | 10.1.0.0/16
  8 | 10.0.0.0/8
  9 | 2001:4f8:3:ba::/64
 10 | 2001:4f8:3:ba:2e0:81ff:fe22:d1f1/128
 11 | ::ffff:127.0.0.0/120
 12 | ::ffff:127.0.0.0/128
(12 rows)

gaussdb=# DROP TABLE cidr_test;
DROP TABLE
```

inet

inet类型在一个数据区域内保存主机的IPv4地址，以及一个可选子网。主机地址中网络地址的位数表示子网（“子网掩码”）。如果子网掩码是32并且地址是IPv4，则这个值不表示任何子网，只表示一台主机。

该类型的输入格式是address/y，address表示IPv4地址，y是子网掩码的二进制位数。如果省略/y，则子网掩码对IPv4是32，所以该值表示只有一台主机。如果该值表示只有一台主机，/y将不会显示。

inet和cidr类型之间的基本区别是inet接受子网掩码，而cidr不接受。

示例：

```
gaussdb=# CREATE TABLE inet_test(id int, i inet);
CREATE TABLE
gaussdb=# INSERT INTO inet_test VALUES (1, '192.168.100.128/25');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (2, '192.168.100.128');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (3, '192.168.1.0/24');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (4, '192.168.1.0/25');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (5, '192.168.1.255/24');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (6, '192.168.1.255/25');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (7, '10.1.2.3/8');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (8, '11.1.2.3/16');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (9, '12.1.2.3/24');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (10, '13.1.2.3/32');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (11, '2001:4f8:3:ba::/64');
INSERT 0 1
```



```
gaussdb=# INSERT INTO inet_test VALUES (12, '2001:4f8:3:ba:2e0:81ff:fe22:d1f1/128');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (13, '::ffff:127.0.0.0/120');
INSERT 0 1
gaussdb=# INSERT INTO inet_test VALUES (14, '::ffff:127.0.0.0/128');
INSERT 0 1
gaussdb=# SELECT * FROM inet_test ORDER BY id;
 id |                 i
----+-----------------------------------
  1 | 192.168.100.128/25
  2 | 192.168.100.128
  3 | 192.168.1.0/24
  4 | 192.168.1.0/25
  5 | 192.168.1.255/24
  6 | 192.168.1.255/25
  7 | 10.1.2.3/8
  8 | 11.1.2.3/16
  9 | 12.1.2.3/24
 10 | 13.1.2.3
 11 | 2001:4f8:3:ba::/64
 12 | 2001:4f8:3:ba:2e0:81ff:fe22:d1f1
 13 | ::ffff:127.0.0.0/120
 14 | ::ffff:127.0.0.0
(14 rows)

gaussdb=# DROP TABLE inet_test;
DROP TABLE
```

macaddr

macaddr类型存储MAC地址，也就是以太网卡硬件地址（尽管MAC地址还用于其它用途）。可以接受下列格式：

```
'08:00:2b:01:02:03'
'08-00-2b-01-02-03'
'08002b:010203'
'08002b-010203'
'0800.2b01.0203'
'08002b010203'
```

以上示例都表示同一个地址。对于数据位a到f，大小写均可。输出时都是以第一种形式展示。

示例：

```
gaussdb=# CREATE TABLE macaddr_test(id int, m macaddr);
CREATE TABLE
gaussdb=# INSERT INTO macaddr_test VALUES (1, '08:00:2b:01:02:03');
INSERT 0 1
gaussdb=# INSERT INTO macaddr_test VALUES (2, '08-00-2b-01-02-03');
INSERT 0 1
gaussdb=# INSERT INTO macaddr_test VALUES (3, '08002b:010203');
INSERT 0 1
gaussdb=# INSERT INTO macaddr_test VALUES (4, '08002b-010203');
INSERT 0 1
gaussdb=# INSERT INTO macaddr_test VALUES (5, '0800.2b01.0203');
INSERT 0 1
gaussdb=# INSERT INTO macaddr_test VALUES (6, '08002b010203');
INSERT 0 1
gaussdb=# SELECT * FROM macaddr_test ORDER BY id;
 id |         m
----+-------------------
  1 | 08:00:2b:01:02:03
  2 | 08:00:2b:01:02:03
  3 | 08:00:2b:01:02:03
  4 | 08:00:2b:01:02:03
  5 | 08:00:2b:01:02:03
  6 | 08:00:2b:01:02:03
```



```
(6 rows)

gaussdb=# DROP TABLE macaddr_test;
DROP TABLE
```

7.3.9 位串类型

位串就是一串1和0的字符串。它们可以用于存储位掩码。

GaussDB支持两种位串类型：bit(n)和bit varying(n)，这里的n是一个正整数，n最大取值为83886080，相当于10M的容量。

bit类型的数据必须准确匹配长度n，如果存储的数据长度不匹配都会报错。bit varying类型的数据是最长为n的变长类型，长度超过n时会被拒绝。一个没有长度的bit等效于bit(1)，没有长度的bit varying表示没有长度限制。

说明

- 如果用户明确地把一个位串值转换成bit(n)，则此位串右边的内容将被截断或者在右边补齐零，直到刚好n位，而不会抛出任何错误。
- 如果用户明确地把一个位串数值转换成bit varying(n)，且它超过了n位，则它的右边将被截断。
- 使用ADMS平台8.1.3-200驱动版本及之前版本时，写入bit类型需要用::bit varying进行类型转换，否则可能出现异常报错。

```
--创建表。
gaussdb=# CREATE TABLE bit_type_t1
(
   BT_COL1 INTEGER,
   BT_COL2 BIT(3),
   BT_COL3 BIT VARYING(5)
) ;

--插入数据。
gaussdb=# INSERT INTO bit_type_t1 VALUES(1, B'101', B'00');

--插入数据的长度不符合类型的标准会报错。
gaussdb=# INSERT INTO bit_type_t1 VALUES(2, B'10', B'101');
ERROR:  bit string length 2 does not match type bit(3)
CONTEXT:  referenced column: bt_col2

--将不符合类型长度的数据进行转换。
gaussdb=# INSERT INTO bit_type_t1 VALUES(2, B'10'::bit(3), B'101');

--查看数据。
gaussdb=# SELECT * FROM bit_type_t1;
 bt_col1 | bt_col2 | bt_col3
---------+---------+---------
       1 | 101     | 00
       2 | 100     | 101
(2 rows)

--删除表。
gaussdb=# DROP TABLE bit_type_t1;
```

7.3.10 UUID 类型

UUID数据类型用来存储RFC 4122，ISO/IEF 9834-8:2005以及相关标准定义的通用唯一标识符（UUID）。这个标识符是一个由算法产生的128位标识符，确保它不可能使用相同算法在已知的模块中产生相同的标识符。

UUID是一个小写十六进制数字的序列，由连字符分成几组，一组8位数字+三组4位数字+一组12位数字，总共32个数字代表128位，标准的UUID示例如下：


```
a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
```

GaussDB同样支持以其他方式输入：大写字母和数字、由花括号包围的标准格式、省略部分或所有连字符、在任意一组四位数字之后加一个连字符。示例：

```
A0EEBC99-9C0B-4EF8-BB6D-6BB9BD380A11
{a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11}
a0eebc999c0b4ef8bb6d6bb9bd380a11
a0ee-bc99-9c0b-4ef8-bb6d-6bb9-bd38-0a11
```

一般是以标准格式输出。

示例

```
-- 创建表
gaussdb=# CREATE TABLE uuid_test(id int, test uuid);

-- 插入数据，使用示例格式插入数据
gaussdb=# INSERT INTO uuid_test VALUES(1, 'A0EEBC99-9C0B-4EF8-BB6D-6BB9BD380A11');
gaussdb=# INSERT INTO uuid_test VALUES(2, '{a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11}');
gaussdb=# INSERT INTO uuid_test VALUES(3, 'a0eebc999c0b4ef8bb6d6bb9bd380a11');
gaussdb=# INSERT INTO uuid_test VALUES(4, 'a0ee-bc99-9c0b-4ef8-bb6d-6bb9-bd38-0a11');

-- 查看数据，输出时以标准格式输出
gaussdb=# SELECT * FROM uuid_test;
 id |                 test
----+--------------------------------------
  1 | a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
  2 | a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
  3 | a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
  4 | a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
(4 rows)
```

7.3.11 JSON/JSONB 类型

JSON(JavaScript Object Notation)数据，可以是单独的一个标量，也可以是一个数组，也可以是一个键值对象，其中数组和对象可以统称容器(container)：

- 标量(scalar)：单一的数字、bool、string、null都可以称作标量。
- 数组(array)：[]结构，里面存放的元素可以是任意类型的JSON，并且不要求数组内所有元素都是同一类型。
- 对象(object)：{}结构，存储key:value的键值对，其键只能是用""包裹起来的字符串，值可以是任意类型的JSON，对于重复的键，按最后一个键值为准。

GaussDB存在两种数据类型JSON和JSONB，可以用来存储JSON数据。其中JSON是对输入的字符串的完整拷贝，使用时再去解析，所以它会保留输入的空格、重复键以及顺序等；JSONB数据以解析的二进制格式存储，它在解析时会删除语义无关的细节和重复的键，对键值也会进行排序，使用时无需再次解析。

因此可以发现，两者其实都是JSON，它们接受相同的字符串作为输入。它们实际的主要差别是效率。JSON数据类型存储输入文本的精确拷贝，处理函数必须在每个执行上重新解析； 而JSONB数据以解析的二进制格式存储， 由于添加了解析机制，因此在输入上稍微慢些，但是在处理上明显更快， 因为不需要重新解析。同时由于JSONB类型存在解析后的格式归一化等操作，同等的语义下只会有一种格式，因此可以更好更强大的支持很多其他额外的操作，比如按照一定的规则进行大小比较等。JSONB也支持索引，这也是一个明显的优势。

输入格式

输入必须是一个符合JSON数据格式的字符串，此字符串用单引号''声明。

null (null-json)：仅null，全小写。

```
gaussdb=# SELECT 'null'::json;   -- suc
 json
------
 null
(1 row)

gaussdb=# SELECT 'NULL'::jsonb;  -- err
ERROR:  invalid input syntax for type json
```

数字 (num-json)：正负整数、小数、0，支持科学计数法。

```
gaussdb=# SELECT '1'::json;
json
------
 1
(1 row)

gaussdb=# SELECT '-1.5'::json;
 json
------
 -1.5
(1 row)

gaussdb=# SELECT '-1.5e-5'::jsonb, '-1.5e+2'::jsonb;
   jsonb   | jsonb
-----------+-------
 -0.000015 | -150
(1 row)

gaussdb=# SELECT '001'::json, '+15'::json, 'NaN'::json;  -- 不支持多余的前导0，正数的+号，以及NaN和
infinity。
ERROR:  invalid input syntax for type json
```

布尔(bool-json)：仅true、false，全小写。

```
gaussdb=# SELECT 'true'::json;
 json
------
 true
(1 row)

gaussdb=# SELECT 'false'::jsonb;
 jsonb
-------
 false
(1 row)
```

字符串(str-json)：必须是加双引号的字符串。

```
gaussdb=# SELECT '"a"'::json;
 json
------
 "a"
(1 row)

gaussdb=# SELECT '"abc"'::jsonb;
 jsonb
-------
 "abc"
(1 row)
```

数组(array-json)：使用中括号[]包裹，满足数组书写条件。数组内元素类型可以是任意合法的JSON，且不要求类型一致。

```
gaussdb=# SELECT '[1, 2, "foo", null]'::json;
        json
--------------------
 [1, 2, "foo", null]
```



```
(1 row)

gaussdb=# SELECT '[]'::json;
 json
------
 []
(1 row)

gaussdb=# SELECT '[1, 2, "foo", null, [[]], {}]'::jsonb;
              jsonb
--------------------------------
 [1, 2, "foo", null, [[]], {}]
(1 row)
```

对象(object-json)：使用大括号{}包裹，键必须是满足JSON字符串规则的字符串，值可以是任意合法的JSON。

```
gaussdb=# SELECT '{}'::json;
 json
------
 {}
(1 row)

gaussdb=# SELECT '{"a": 1, "b": {"a": 2,  "b": null}}'::json;
                json
------------------------------------
 {"a": 1, "b": {"a": 2,  "b": null}}
(1 row)

gaussdb=# SELECT '{"foo": [true, "bar"], "tags": {"a": 1, "b": null}}'::jsonb;
                        jsonb
------------------------------------------------------
 {"foo": [true, "bar"], "tags": {"a": 1, "b": null}}
(1 row)
```

⚠ 注意

- 区分 'null'::json 和 null::json 是两个不同的概念，类似于字符串 str="null" 和 str=null。
- 对于数字，当使用科学计数法的时候，jsonb类型会将其展开，而json会精准拷贝输入。

JSONB 高级特性

- 注意事项
  - 不支持作为分区键。
  - 不支持外表。

JSON和JSONB的主要差异在于存储方式上的不同，JSONB存储的是解析后的二进制，能够体现JSON的层次结构，更方便直接访问等，因此JSONB会有很多JSON所不具有的高级特性。

- 格式归一化
  - 对于输入的object-json字符串，解析成jsonb二进制后，会天然的丢弃语义上无关紧要的细节，比如空格：

```
gaussdb=# SELECT '   [1, " a ", {"a"   :1    }]  '::jsonb;
       jsonb
----------------------
 [1, " a ", {"a": 1}]
(1 row)
```

  - 对于object-json，会删除重复的键值，只保留最后一个出现的，如：

```
gaussdb=# SELECT '{"a" : 1, "a" : 2}'::jsonb;
  jsonb
----------
 {"a": 2}
(1 row)
```

  - 对于object-json，键值会重新进行排序，排序规则：长度长的在后、长度相等则ascii码大的在后，如：

```
gaussdb=# SELECT '{"aa" : 1, "b" : 2, "a" : 3}'::jsonb;
          jsonb
---------------------------
{"a": 3, "b": 2, "aa": 1}
(1 row)
```

- 大小比较

  由于经过了格式归一化，保证了同一种语义下的jsonb只会有一种存在形式，因此按照制定的规则，可以比较大小。

  - 首先比较类型：object-jsonb > array-jsonb > bool-jsonb > num-jsonb > str-jsonb > null-jsonb
  - 同类型则比较内容：
    - str-jsonb类型：依据text比较的方法，使用数据库默认排序规则进行比较，返回值正数代表大于，负数代表小于，0表示相等。
    - num-jsonb类型：数值比较。
    - bool-jsonb类型：true > false。
    - array-jsonb类型：长度长的 > 长度短的，长度相等则依次比较每个元素。
    - object-jsonb类型：长度长的 > 长度短的，长度相等则依次比较每个键值对，先比较键，再比较值。

  ⚠ 注意

  object-jsonb类型内比较时，使用的是格式整理后的最终结果进行比较，因此相对于直接的输入未必会很直观。

- 创建索引、主外键
  - BTREE索引

    jsonb类型支持创建btree索引，支持创建主键、外键。

- 包含存在

  查询一个JSON之中是否包含某些元素，或者某些元素是否存在于某个JSON中是jsonb的一个重要能力。

```
-- 简单的标量/原始值只包含相同的值。
gaussdb=# SELECT '"foo"'::jsonb @> '"foo"'::jsonb;
 ?column?
----------
 t
(1 row)

-- 左侧数组包含了右侧字符串。
gaussdb=# SELECT '[1, "aa", 3]'::jsonb ? 'aa';
 ?column?
```

```
----------
 t
(1 row)

-- 左侧数组包含了右侧的数组所有元素，顺序、重复不重要。
gaussdb=# SELECT '[1, 2, 3]'::jsonb @> '[1, 3, 1]'::jsonb;
 ?column?
----------
 t
(1 row)

-- 左侧object-json包含了右侧object-json的所有键值对。
gaussdb=# SELECT '{"product": "PostgreSQL", "version": 9.4, "jsonb":true}'::jsonb @>
'{"version":9.4}'::jsonb;
 ?column?
----------
 t
(1 row)

-- 左侧数组并没有包含右侧的数组所有元素，因为左侧数组的三个元素为1、2、[1,3]，右侧的为1、3。
gaussdb=# SELECT '[1, 2, [1, 3]]'::jsonb @> '[1, 3]'::jsonb;
 ?column?
----------
 f
(1 row)

gaussdb=# SELECT '{"foo": {"bar": "baz"}}'::jsonb @> '{"bar": "baz"}'::jsonb;
 ?column?
----------
 f
(1 row)
```

相关的操作符请参见**JSON/JSONB函数和操作符**。

- 函数和操作符

  json/jsonb类型相关支持的函数和操作符请参见**JSON/JSONB函数和操作符**。

7.3.12 HLL 数据类型

HLL（HyperLoglog）是统计数据集中唯一值个数的高效近似算法。它有着计算速度快、节省空间的特点，不需要直接存储集合本身，而是存储一种名为HLL的数据结构。每当有新数据加入进行统计时，只需要把数据经过哈希计算并插入到HLL中，最后根据HLL就可以得到结果。

HLL与其他算法的比较请参见**表7-20**。

表 7-20 HLL 与其他算法比较

项目	Sort算法	Hash算法	HLL
时间复杂度	O(nlogn)	O(n)	O(n)
空间复杂度	O(n)	O(n)	log(logn)
误差率	0	0	≈0.8%
所需存储空间	原始数据大小	原始数据大小	默认规格下最大16KB

HLL在计算速度和所占存储空间上都占优势。在时间复杂度上，Sort算法需要排序至少O(nlogn)的时间，虽说Hash算法和HLL一样扫描一次全表O(n)的时间就可以得出结

果，但是存储空间上，Sort算法和Hash算法都需要先把原始数据存起来再进行统计，会导致存储空间消耗巨大，而对HLL来说不需要存原始数据，只需要维护HLL数据结构，故占用空间有很大的压缩，默认规格下HLL数据结构的最大空间约为16KB。

须知

- 当前默认规格下可计算最大distinct值的数量约为1.1e+15个，误差率为0.8%。用户应注意如果计算结果超过当前规格下distinct最大值会导致计算结果误差率变大，或导致计算结果失败并报错。
- 用户在首次使用该特性时，应该对业务的distinct value做评估，选取适当的配置参数并做验证，以确保精度符合要求：
  - 当前默认参数下，可以计算的distinct值为1.1e+15，如果计算得到的distinct值为NaN，需要调整log2m，或者采用其他算法计算distinct值。
  - 虽然hash算法存在极低的hash collision概率，但是建议用户在首次使用时，选取2-3个hash seed验证，如果得到的distinct value相差不大，则可以从该组seed中任选一个作为hash seed。

HLL中主要的数据结构，请参见**表7-21**。

表 7-21 HyperLogLog 中主要数据结构

数据类型	功能描述
hll	hll头部为27字节长度字段，默认规格下数据段长度0~16KB，可直接计算得到distinct值。

创建HLL数据类型时，可以支持0~4个参数入参，具体的参数含义与参数规格同函数hll_empty一致。第一个参数为log2m，表示分桶数的对数值，取值范围10~16；第二个参数为log2explicit，表示Explicit模式的阈值大小，取值范围0~12；第三个参数为log2sparse，表示Sparse模式的阈值大小，取值范围0~14；第四个参数为duplicatecheck，表示是否启用duplicatecheck，取值范围为0~1。当入参输入值为-1时，会采用默认值设定HLL的参数。可以通过\d或\d+查看HLL类型的参数。

说明

创建HLL数据类型时，根据入参的行为不同，结果不同：

- 创建HLL类型时对应入参不输入或输入-1，采用默认值设定对应的HLL参数。
- 输入合法范围的入参，对应HLL参数采用输入值。
- 输入不合法范围的入参，创建HLL类型报错。

```
-- 创建hll类型的表，不指定入参。
gaussdb=# create table t1 (id integer, set hll);
gaussdb=# \d t1
      Table "public.t1"
 Column |  Type   | Modifiers
--------+---------+-----------
 id     | integer |
 set    | hll     |

-- 创建hll类型的表，指定前两个入参，后两个采用默认值。
gaussdb=# create table t2 (id integer, set hll(12,4));
gaussdb=# \d t2
```



```
         Table "public.t2"
 Column |      Type      | Modifiers
--------+----------------+-----------
 id     | integer        |
 set    | hll(12,4,12,0) |

--创建hll类型的表，指定第三个入参，其余采用默认值。
gaussdb=# create table t3(id int, set hll(-1,-1,8,-1));
gaussdb=# \d t3
         Table "public.t3"
 Column |      Type      | Modifiers
--------+----------------+-----------
 id     | integer        |
 set    | hll(14,10,8,0) |

--创建hll类型的表，指定入参不合法报错。
gaussdb=# create table t4(id int, set hll(5,-1));
ERROR:  log2m = 5 is out of range, it should be in range 10 to 16, or set -1 as default

--删除已创建的hll类型的表。
gaussdb=# drop table t1,t2,t3;
DROP TABLE
```

说明

对含有HLL类型的表插入HLL对象时，HLL类型的设定参数须同插入对象的设定参数一致，否则报错。

```
-- 创建带有hll类型的表。
gaussdb=# create table t1(id integer, set hll(14));

-- 向表中插入hll对象，参数一致，成功。
gaussdb=# insert into t1 values (1, hll_empty(14,-1));

-- 向表中插入hll对象，参数不一致，失败。
gaussdb=# insert into t1(id, set) values (1, hll_empty(14,5));
ERROR:  log2explicit does not match: source is 5 and dest is 10

--删除表。
gaussdb=# drop table t1;
```

HLL的应用场景。

- 场景1：“Hello World”

  通过下面的示例说明如何使用hll数据类型：

  ```
  -- 创建带有hll类型的表。
  gaussdb=# create table helloworld (id integer, set hll);

  -- 向表中插入空的hll。
  gaussdb=# insert into helloworld(id, set) values (1, hll_empty());

  -- 把整数经过哈希计算加入到hll中。
  gaussdb=# update helloworld set set = hll_add(set, hll_hash_integer(12345)) where id = 1;

  -- 把字符串经过哈希计算加入到hll中。
  gaussdb=# update helloworld set set = hll_add(set, hll_hash_text('hello world')) where id = 1;

  -- 得到hll中的distinct值。
  gaussdb=# select hll_cardinality(set) from helloworld where id = 1;
   hll_cardinality
  -----------------
                 2
  (1 row)

  -- 删除表。
  gaussdb=#  drop table helloworld;
  ```

- 场景2：“网站访客数量统计”

  通过下面的示例说明hll如何统计在一段时间内访问网站的不同用户数量：


```
-- 创建原始数据表，表示某个用户在某个时间访问过网站。
gaussdb=# create table facts (
        date            date,
        user_id         integer
);

-- 构造数据，表示一天中有哪些用户访问过网站。
gaussdb=# insert into facts values ('2019-02-20', generate_series(1,100));
gaussdb=# insert into facts values ('2019-02-21', generate_series(1,200));
gaussdb=# insert into facts values ('2019-02-22', generate_series(1,300));
gaussdb=# insert into facts values ('2019-02-23', generate_series(1,400));
gaussdb=# insert into facts values ('2019-02-24', generate_series(1,500));
gaussdb=# insert into facts values ('2019-02-25', generate_series(1,600));
gaussdb=# insert into facts values ('2019-02-26', generate_series(1,700));
gaussdb=# insert into facts values ('2019-02-27', generate_series(1,800));

-- 创建表并指定列为hll。
gaussdb=# create table daily_uniques (
    date            date UNIQUE,
    users           hll
);

-- 根据日期把数据分组，并把数据插入到hll中。
gaussdb=# insert into daily_uniques(date, users)
    select date, hll_add_agg(hll_hash_integer(user_id))
    from facts
    group by 1;

-- 计算每一天访问网站不同用户数量。
gaussdb=# select date, hll_cardinality(users) from daily_uniques order by date;
    date    | hll_cardinality
------------+------------------
 2019-02-20 |              100
 2019-02-21 | 200.217913059312
 2019-02-22 |  301.76494508014
 2019-02-23 | 400.862858326446
 2019-02-24 | 502.626933349694
 2019-02-25 | 601.922606454213
 2019-02-26 | 696.602316769498
 2019-02-27 | 798.111731634412
(8 rows)

-- 计算在2019.02.20到2019.02.26一周中有多少不同用户访问过网站。
gaussdb=# select hll_cardinality(hll_union_agg(users)) from daily_uniques where date >=
'2019-02-20'::date and date <= '2019-02-26'::date;
 hll_cardinality
------------------

 696.602316769498
(1 row)

-- 计算昨天访问过网站而今天没访问网站的用户数量。
gaussdb=# SELECT date, (#hll_union_agg(users) OVER two_days) - #users AS lost_uniques FROM
daily_uniques WINDOW two_days AS (ORDER BY date ASC ROWS 1 PRECEDING);
--默认兼容性（O兼容性）结果如
下：
       date         | lost_uniques
--------------------+--------------
 2019-02-20 00:00:00 |            0
 2019-02-21 00:00:00 |            0
 2019-02-22 00:00:00 |            0
 2019-02-23 00:00:00 |            0
 2019-02-24 00:00:00 |            0
 2019-02-25 00:00:00 |            0
 2019-02-26 00:00:00 |            0
 2019-02-27 00:00:00 |            0
(8 rows)

-- 删除表。
```



```
gaussdb=# drop table facts;
gaussdb=# drop table daily_uniques;
```

- 场景3：“插入数据不满足hll数据结构要求”

  当用户给hll类型的字段插入数据的时候，必须保证插入的数据满足hll数据结构要求，如果解析后不满足就会报错。如下示例中： 插入数据'E\\1234'时，该数据不满足hll数据结构要求，不能解析成功因此失败报错。

```
gaussdb=# create table test(id integer, set hll);
gaussdb=# insert into test values(1, 'E\\1234');
ERROR:  not a hll type, size=6 is not enough
gaussdb=# drop table test;
```

7.3.13 范围类型

范围类型是表达某种元素类型（称为范围的subtype）的一个值的范围的数据类型。例如，timestamp的范围可以被用来表达一个会议室被保留的时间范围。在这种情况下，数据类型是tsrange（“timestamp range”的简写），而timestamp是其subtype。subtype必须具有一种总体的顺序，这样对于元素值是在一个范围值之内、之前还是之后就已经是明确的了。

范围类型非常有用，因为它们可以表达一种单一范围值中的多个元素值，并且可以很清晰地表达诸如范围重叠等概念，如用于时间安排的时间和日期范围，以及价格范围、仪器的量程等场景都是可以表示的。

内建范围类型

有下列内建范围类型：

- int4range — integer的范围
- int8range — bigint的范围
- numrange — numeric的范围
- tsrange — 不带时区的 timestamp的范围
- tstzrange — 带时区的 timestamp的范围
- daterange — date的范围

此外，用户可以定义自己的范围类型，详见**CREATE TYPE**。

例子

```
CREATE TABLE reservation (room int, during tsrange);
INSERT INTO reservation VALUES (1108, '[2010-01-01 14:30, 2010-01-01 15:30)');
-- 包含
SELECT int4range(10, 20) @> 3;
 ?column?
----------
 f
(1 row)
-- 重叠
SELECT numrange(11.1, 22.2) && numrange(20.0, 30.0);
 ?column?
----------
 t
(1 row)
-- 抽取上界
SELECT upper(int8range(15, 25));
 upper
-------
    25
(1 row)
```



```
-- 计算交集
SELECT int4range(10, 20) * int4range(15, 25);
 ?column?
----------
 [15,20)
(1 row)
-- 范围为空吗?
SELECT isempty(numrange(1, 5));
 isempty
---------
 f
(1 row)
DROP TABLE reservation;
```

范围类型上的操作符和函数的完整列表可见**范围函数和操作符**。

包含和排除边界

每一个非空范围都有两个界限，下界和上界。上下界之间的所有点都被包括在范围内。一个包含界限意味着边界点本身也被包括在范围内，而一个排除边界意味着边界点不被包括在范围内。

在一个范围的文本形式中，一个包含下界被表达为“[”而一个排除下界被表达为“(”。同样，一个包含上界被表达为“]”而一个排除上界被表达为“)”（详见**范围输入/输出**）。

函数lower_inc和upper_inc分别测试一个范围值的上下界。

无限（无界）范围

一个范围的下界可以被忽略，意味着所有小于上界的值都被包括在范围中。 同样，如果范围的上界被忽略，那么所有比下界大的值都被包括在范围中。如果上下界都被忽略，该元素类型的所有值都被认为在该范围中。 规定缺失的包括界限自动转换为排除。 用户可以认为这些缺失值为 +/- 无穷大，但它们是特殊范围类型值，并且被视为超出任何范围元素类型的 +/- 无穷大值。

具有“infinity”概念的元素类型可以用它们作为显式边界值。例如，在时间戳范围，[today,infinity)不包括特殊的timestamp值infinity，尽管 [today,infinity] 包括它，就好比 [today,) 和 [today,]。

函数lower_inf和upper_inf分别测试一个范围的无限上下界。

范围输入/输出

一个范围值的输入必须遵循下列模式之一：

```
(lower-bound,upper-bound)
(lower-bound,upper-bound]
[lower-bound,upper-bound)
[lower-bound,upper-bound]
empty
```

一个范围值的输出必须遵循下列模式之一：

```
[lower-bound,upper-bound)
empty
```

圆括号或方括号指示上下界是否为排除的或者包含的。注意最后一个模式是empty，它表示一个空范围（一个不包含点的范围）。

*lower-bound*可以是作为subtype的合法输入的一个字符串，或者是空表示没有下界。同样，*upper-bound*可以是作为 subtype 的合法输入的一个字符串，或者是空表示没有上界。

每个界限值可以使用"（双引号）字符引用。如果界限值包含圆括号、方括号、逗号、双引号或反斜线时，这样做是必须的，否则那些字符会被认作范围语法的一部分。要把一个双引号或反斜线放在一个被引用的界限值中，就在它前面放一个反斜线（还有，在一个双引号引用的界限值中的一对双引号表示一个双引号字符，这与 SQL 字符串中的单引号规则类似）。此外，用户可以避免引用或者使用反斜线转义来保护所有数据字符，否则它们会被当做范围语法的一部分。还有，要写一个是空字符串的界限值，则可以写成""，因为什么都不写表示一个无限界限。

范围值前后允许有空格，但是圆括号或方括号之间的任何空格会被当做上下界值的一部分（取决于元素类型，它可能是也可能不是有意义的）。

例子：

```
--包括3，不包括7之间的所有点。
gaussdb=# SELECT '[3,7)'::int4range;
 int4range
-----------
 [3,7)
(1 row)
--既不包括3也不包括7之间的所有点。
gaussdb=# SELECT '(3,7)'::int4range;
 int4range
-----------
 [4,7)
(1 row)
--只包括单独一个点4。
gaussdb=# SELECT '[4,4]'::int4range;
 int4range
-----------
 [4,5)
(1 row)
--不包括点（并且将被标准化为 '空'）。
gaussdb=# SELECT '[4,4)'::int4range;
 int4range
-----------
 empty
(1 row)
```

构造范围

每一种范围类型都有一个与其同名的构造器函数。使用构造器函数常常比写一个范围文字常数更方便，因为它避免了对界限值的额外引用。构造器函数接受两个或三个参数。两个参数的形式以标准的形式构造一个范围（下界是包含的，上界是排除的），而三个参数的形式按照第三个参数指定的界限形式构造一个范围。第三个参数必须是下列字符串之一：“()”、“(]”、“[)”或者“[]”。例如：

```
--完整形式是：下界、上界以及指示界限包含性/排除性的文本参数。
gaussdb=# SELECT numrange(1.0, 14.0, '(]');
  numrange
------------
 (1.0,14.0]
(1 row)
--如果第三个参数被忽略，则假定为 '[)'。
gaussdb=# SELECT numrange(1.0, 14.0);
  numrange
------------
 [1.0,14.0)
(1 row)
--尽管这里指定了 '(]'，显示时该值将被转换成标准形式，因为int8range是一种离散范围类型。
gaussdb=# SELECT int8range(1, 14, '(]');
```



```
 int8range
-----------
 [2,15)
(1 row)
--为一个界限使用NULL导致范围在那一边是无界的。
gaussdb=# SELECT numrange(NULL, 2.2);
 numrange
----------
 (,2.2)
(1 row)
```

离散范围类型

一种范围的元素类型具有一个良定义的“步长”，例如integer或date。在这些类型中，如果两个元素之间没有合法值，它们可以被说成是相邻。这与连续范围相反，连续范围中总是（或者几乎总是）可以在两个给定值之间标识其他元素值。例如，numeric类型之上的一个范围就是连续的，timestamp上的范围也是（尽管timestamp具有有限的精度，并且在理论上可以被当做离散的，但是可以认为它是连续的，因为通常并不关心它的步长）。

另一种考虑离散范围类型的方法是对每一个元素值都有一个清晰的“下一个”或“上一个”值。了解了这种思想之后，通过选择原来给定的下一个或上一个元素值来取代它，就可以在一个范围界限的包含和排除表达之间转换。例如，在一个整数范围类型中，[4,8]和(3,9)表示相同的值集合，但是对于 numeric 上的范围就不是这样。

一个离散范围类型应该具有一个正规化函数，它知道元素类型期望的步长。正规化函数负责把范围类型的相等值转换成具有相同的表达，特别是与包含或者排除界限一致。如果没有指定一个正规化函数，那么具有不同格式的范围将总是会被当作不等，即使它们实际上是表达相同的一组值。

内建的范围类型int4range、int8range和daterange都使用一种正规的形式，该形式包括下界并且排除上界，也就是[)。不过，用户定义的范围类型可以使用其他形式。

定义新的范围类型

用户可以定义自己的范围类型。这样做最常见的原因是为了使用内建范围类型中提供的subtype上没有的范围。例如，要创建一个subtype float8的范围类型：

```
CREATE TYPE floatrange AS RANGE (
    subtype = float8,
    subtype_diff = float8mi
);
SELECT '[1.234, 5.678]'::floatrange;
  floatrange
---------------
 [1.234,5.678]
(1 row)
DROP TYPE floatrange;
```

因为float8没有有意义的“步长”，在这个例子中没有定义一个正规化函数。

定义自己的范围类型也允许用户指定使用一个不同的子类型B-树操作符类或者集合，以便更改排序顺序来决定哪些值会落入到给定的范围中。

如果subtype被认为是具有离散值而不是连续值，CREATE TYPE命令应当指定一个canonical函数。正规化函数接收一个输入的范围值，并且必须返回一个可能具有不同界限和格式的等价的范围值。对于两个表示相同值集合的范围（例如[1, 7]和[1, 8)），正规的输出必须一样。选择哪一种表达作为正规的没有关系，只要两个具有不同格式的等价值总是能被映射到具有相同格式的相同值就行。除了调整包含/排除界限格式外，假使期望的补偿比subtype能够存储的要大，一个正规化函数可能会舍入边界值。例如，一个timestamp之上的范围类型可能被定义为具有一个一小时的步长，这

样正规化函数可能需要对不是一小时的倍数的界限进行舍入，或者可能直接抛出一个错误。

subtype差异函数采用两个subtype输入值，并且返回表示为一个float8值的差（即*X*减*Y*）。在上面的例子中，可以使用常规float8减法操作符之下的函数。但是对于任何其他subtype，可能需要某种类型转换。还可能需要一些关于如何把差异表达为数字的创新型想法。为了最大的可扩展性，subtype_diff函数应该同意选中的操作符类和排序规则所蕴含的排序顺序，也就是说，只要它的第一个参数根据排序顺序大于第二个参数，它的结果就应该是正值。

subtype_diff函数相关示例：

```
CREATE FUNCTION time_subtype_diff(x time, y time) RETURNS float8 AS 'SELECT EXTRACT(EPOCH FROM
(x - y))' LANGUAGE sql STRICT IMMUTABLE;
CREATE TYPE timerange AS RANGE (
    subtype = time,
    subtype_diff = time_subtype_diff
 );
SELECT '[11:10, 23:00]'::timerange;
      timerange
----------------------
 [11:10:00,23:00:00]
(1 row)
DROP TYPE timerange;
DROP FUNCTION time_subtype_diff;
```

更多关于创建范围类型的信息请参考**CREATE TYPE**。

索引

此外，B-树索引可以在范围类型的表列上创建。对于这些索引类型，基本上唯一有用的范围操作就是等值。使用相应的< 和 >操作符，对于范围值定义有一种 B-树排序顺序，但是该顺序相当任意并且在真实世界中通常不怎么有用。范围类型的 B-树支持主要是为了允许在查询内部进行排序，而不是创建真正的索引。

7.3.14 对象标识符类型

GaussDB在内部使用对象标识符（OID）作为各种系统表的主键。系统不会给用户创建的表增加一个OID系统字段，OID类型代表一个对象标识符。

目前OID类型用一个四字节的无符号整数实现。因此不建议在创建的表中使用OID字段做主键。

表 7-22 对象标识符类型

名称	引用	描述	示例
OID	-	数字化的对象标识符。	564182
CID	-	命令标识符。它是系统字段cmin和cmax的数据类型。命令标识符是32位的量。	-
XID	-	事务标识符。它是系统字段xmin和xmax的数据类型。事务标识符是64位的量。	-

名称	引用	描述	示例
TID	-	行标识符。它是系统表字段ctid的数据类型。行ID是一对数值（块号，块内的行索引），它标识该行在其所在表内的物理位置。	-
REGCONFIG	pg_ts_config	文本搜索配置。	english
REGDICTIONARY	pg_ts_dict	文本搜索字典。	simple
REGOPER	pg_operator	操作符名。	-
REGOPERATOR	pg_operator	带参数类型的操作符。	*(integer,integer)或-(NONE,integer)
REGPROC	pg_proc	函数名称。	sum
REGPROCEDURE	pg_proc	带参数类型的函数。	sum(int4)
REGCLASS	pg_class	关系名。	pg_type
REGTYPE	pg_type	数据类型名。	integer

OID类型：主要作为数据库系统表中字段使用。

示例：

```
gaussdb=# SELECT oid FROM pg_class WHERE relname = 'pg_type';
 oid
------
 1247
(1 row)
```

OID别名类型REGCLASS：主要用于对象OID值的简化查找。

示例：

```
gaussdb=# SELECT attrelid,attname,atttypid,attstattarget FROM pg_attribute WHERE attrelid =
'pg_type'::REGCLASS;
 attrelid |   attname    | atttypid | attstattarget
----------+--------------+----------+---------------
     1247 | xc_node_id   |       23 |             0
     1247 | tableoid     |       26 |             0
     1247 | cmax         |       29 |             0
     1247 | xmax         |       28 |             0
     1247 | cmin         |       29 |             0
     1247 | xmin         |       28 |             0
     1247 | oid          |       26 |             0
     1247 | ctid         |       27 |             0
     1247 | typname      |       19 |            -1
     1247 | typnamespace |       26 |            -1
     1247 | typowner     |       26 |            -1
     1247 | typlen       |       21 |            -1
     1247 | typbyval     |       16 |            -1
     1247 | typtype      |       18 |            -1
```

```
  1247 | typcategory    |       18 |          -1
  1247 | typispreferred |       16 |          -1
  1247 | typisdefined   |       16 |          -1
  1247 | typdelim       |       18 |          -1
  1247 | typrelid       |       26 |          -1
  1247 | typelem        |       26 |          -1
  1247 | typarray       |       26 |          -1
  1247 | typinput       |       24 |          -1
  1247 | typoutput      |       24 |          -1
  1247 | typreceive     |       24 |          -1
  1247 | typsend        |       24 |          -1
  1247 | typmodin       |       24 |          -1
  1247 | typmodout      |       24 |          -1
  1247 | typanalyze     |       24 |          -1
  1247 | typalign       |       18 |          -1
  1247 | typstorage     |       18 |          -1
  1247 | typnotnull     |       16 |          -1
  1247 | typbasetype    |       26 |          -1
  1247 | typtypmod      |       23 |          -1
  1247 | typndims       |       23 |          -1
  1247 | typcollation   |       26 |          -1
  1247 | typdefaultbin  |      194 |          -1
  1247 | typdefault     |       25 |          -1
  1247 | typacl         |     1034 |          -1
  1247 | typelemmod     |       23 |          -1
(39 rows)
```

7.3.15 伪类型

GaussDB数据类型中包含一系列特殊用途的类型，这些类型按照类别被称为伪类型。伪类型不能作为字段的数据类型，但是可以用于声明函数的参数或者结果类型。

当一个函数不仅是简单地接受并返回某种SQL数据类型的情况下伪类型是很有用的。表7-23列出了所有的伪类型。

表 7-23 伪类型

名称	描述
any	表示函数接受任何输入数据类型。
anyelement	表示函数接受任何数据类型。
anyarray	表示函数接受任意数组数据类型。
anynonarray	表示函数接受任意非数组数据类型。
anyenum	表示函数接受任意枚举数据类型。
anyrange	表示函数接受任意范围数据类型。
cstring	表示函数接受或者返回一个空结尾的C字符串。
internal	表示函数接受或者返回一种服务器内部的数据类型。
language_handler	声明一个过程语言调用句柄返回language_handler。
fdw_handler	声明一个外部数据封装器返回fdw_handler。
record	标识函数返回一个未声明的行类型。
trigger	声明一个触发器函数返回trigger。

名称	描述
void	表示函数不返回数值。
opaque	一个已经过时的类型，以前用于所有上面这些用途。

声明用C编写的函数（不管是内置的还是动态装载的）都可以接受或者返回任何这样的伪数据类型。当伪类型作为参数类型使用时，用户需要保证函数的正常运行。

用过程语言编写的函数只能使用实现语言允许的伪类型。目前，过程语言都不允许使用作为参数类型的伪类型，并且只允许使用void和record作为结果类型。一些多态的函数还支持使用anyelement、anyarray、anynonarray、anyenum和anyrange类型。

每一个被声明为anyelement的位置（参数或返回值）都允许具有任意特定的实际数据类型，但是在任何给定的查询中必须全部是相同的实际类型。

伪类型internal用于声明那种只能在数据库系统内部调用的函数，这些函数不能直接在SQL查询里调用。如果函数至少有一个internal类型的参数，则不能从SQL里调用它。建议不要创建任何声明返回internal的函数，除非它至少有一个internal类型的参数。

示例：

```
--创建表。
gaussdb=# CREATE TABLE t1 (a int);

--插入两条数据。
gaussdb=# INSERT INTO t1 values(1),(2);

--创建函数showall()。
gaussdb=# CREATE OR REPLACE FUNCTION showall() RETURNS SETOF record
AS $$ SELECT count(*) from t1; $$
LANGUAGE SQL;

--调用函数showall()。
gaussdb=# SELECT showall();
 showall
---------
 (2)
(1 row)

--删除函数。
gaussdb=# DROP FUNCTION showall();

--删除表。
gaussdb=# DROP TABLE t1;
```

7.3.16 XML 类型

XML数据类型可以被用来存储XML数据。它的内部格式和TEXT类型相同，它比直接在一个TEXT域中存储XML数据的优势在于：XML类型数据支持基于LIBXML2提供的标准XML操作函数及XML规范性的检查。

XML类型可以存储格式良好的遵循XML标准定义的“文档”、以及“内容”片段，它是通过引用更宽泛的“DOCUMENT NODE” XQUERY和XPATH数据模型来定义的。大致上说，这意味着内容片段中可以有多于一个的顶层元素或字符节点。表达式XMLVALUE IS DOCUMENT可以被用来评估一个特定的XML值是一个完整文档或者仅仅是一个文档片段。

XML解析器把XML文档转换为XML DOM对象。DOM（DOCUMENT OBJECT MODEL文档对象模型）定义了访问和操作文档的标准方法。XML DOM（XML DOCUMENT

OBJECT MODEL）定义了访问和操作XML文档的标准方法。XML DOM把XML文档作为树结构来查看。所有元素可以通过DOM树来访问。可以修改或删除它们的内容，并创建新的元素。元素，它们的文本，以及它们的属性，都被认为是节点。

XML底层使用和TEXT类型一样的数据结构进行存储，最大为1GB。

示例：

```
gaussdb=# CREATE TABLE xmltest ( id int, data xml );
gaussdb=# INSERT INTO xmltest VALUES (1, 'one');
gaussdb=# INSERT INTO xmltest VALUES (2, 'two');
gaussdb=# SELECT * FROM xmltest ORDER BY 1;
 id | data
----+--------------------
1 | one
2 | two
(2 rows)
gaussdb=# SELECT xmlconcat(xmlcomment('hello'),
                           xmlelement(NAME qux, 'xml'),
                           xmlcomment('world'));
              xmlconcat
----------------------------------------
 <!--hello--><qux>xml</qux><!--world-->
(1 row)
gaussdb=# DROP TABLE xmltest;
```


说明

- XML类型不支持如下操作：
  - 逻辑表达式and、or、not。
  - 作为非XML操作函数的系统函数的入参。
  - 作为分布键、分区键、二级分区键、外键、主键、唯一约束。
  - XML相关的隐式类型转换，包括字符串和XML类型之间的隐式转换。
  - 数组表达式、行表达式、子查询表达式、TABLE OF和TABLE OF INDEX。
  - 使用XML数据格式列作为普通索引、unique索引、global索引、local索引、部分索引。
  - 比较表达式>、<、>=、<=、=、<>、!=、^=、between and、is distinct from、is not distinct from、<=>。
  - 条件表达式decode、nullif、greatest、least。
  - 作为distinct/group by/order by 参数。
  - 聚合函数sum、max、min、avg、list_agg、corr、covar_pop、covar_samp、stddev、stddev_pop、stddev_samp、var_pop、var_samp、variance、bit_and、bit_or、bool_and、bool_or、every、regr_avgx、regr_avgy、regr_count、regr_intercept、regr_r2、regr_slope、regr_sxx、regr_sxy、regr_syy、rank、spread。
  - 不支持ODBC相关绑定传参接口。
- XML类型支持如下操作：
  - 物理备份恢复。
  - 比较表达式is null、is not null。
  - 条件表达式case、coalesce。
  - 全局临时表和本地临时表。
  - 强制类型转换。
  - 表达式索引。
  - XML类型的输入值需要符合XML格式标准。
  - 支持gs_dump导出和gs_restore导入操作。
  - 并行查询，支持astore和ustore存储引擎。
  - 作为自定义函数的入参、出参、自定义变量和返回值。
  - 作为存储过程的入参、出参、自定义变量和返回值。支持自治事务的存储过程。
  - 字符处理函数quote_literal(string text)（需显式转换为字符类型）、quote_nullable(string text)（需显式转换为字符类型）。
  - 聚集函数count、array_agg、checksum（需显式转换为字符类型）、string_agg（需显式转换为字符类型）。
  - 自定义类型中复合类型涉及XML type时的增删改查，且与普通表中的XML字段一样，需要按照XML的语法插入和修改。
  - 支持JDBC和ODBC对XML数据类型操作，支持对该字段进行select、update、insert、delete，使用SQL语法输入XML值，使用ResultSet类的getSQLXML方法获取XML值。支持JDBC相关绑定传参接口、可使用PreparedStatement预处理语句接口中的setSQLXML方法和ResultSet执行结果集接口中的getSQLXML(int columnIndex)方法。

    调用流程：需要使用java.sql.SQLXML接口类构造XML对象，再设置指定的对象类型为Oid.XML，然后将类型ID和XML值发送到 服务端，从服务端获取到返回结果后，先调用ResultSet.getString，然后通过获取的字符串使用java.sql.SQLXML接口类构造XML对象，此时会再次检查内容是否符合XML标准格式。所以xml值也可以使用ResultSet.getString直接获取XML的字符串对象。

7.3.17 XMLTYPE 类型

XMLTYPE数据类型可以被用来存储XMLTYPE数据。目前，它的内部格式中存储数据是按字符串方式存储的，它比直接在一个TEXT域中存储XML数据的优势在于：XML类型数据支持基于LIBXML2提供的标准XML操作函数及XML规范性的检查。

XMLTYPE类型可以存储格式良好的遵循XML标准定义的“文档”。

XML解析器把XML文档转换为XML DOM对象。DOM（DOCUMENT OBJECT MODEL文档对象模型）定义了访问和操作文档的标准方法。XML DOM（XML DOCUMENT OBJECT MODEL）定义了访问和操作XML文档的标准方法。XML DOM把XML文档作为树结构来查看。所有元素可以通过DOM树来访问。可以修改或删除它们的内容，并创建新的元素。元素，它们的文本，以及它们的属性，都被认为是节点。最大为1GB。

示例：

```
gaussdb=# CREATE TABLE xmltypetest(id int, data xmltype);
gaussdb=# INSERT INTO xmltypetest VALUES (1, '<ss/>');
gaussdb=# INSERT INTO xmltypetest VALUES (2, '<xx/>');
gaussdb=# SELECT * FROM xmltypetest ORDER BY 1;
 id | data
----+-------
  1 | <ss/>
  2 | <xx/>
(2 rows)
gaussdb=# DROP TABLE xmltest;
```


说明

- XMLTYPE类型不支持如下操作：
  - 逻辑表达式and、or、not。
  - 作为非XMLTYPE操作函数的系统函数的入参。
  - 作为分布键、分区键、二级分区键、外键、主键、唯一约束。
  - XMLTYPE相关的隐式类型转换，包括字符串和XMLTYPE类型之间的隐式转换。
  - 数组表达式、行表达式、子查询表达式、TABLE OF和TABLE OF INDEX。
  - 使用XMLTYPE数据格式列作为普通索引、unique索引、global索引、local索引、部分索引。
  - 比较表达式>、<、>=、<=、=、<>、!=、^=、between and、is distinct from、is not distinct from、<=>。
  - 条件表达式decode、nullif、greatest、least。
  - 作为distinct/group by/order by 参数。
  - 聚合函数sum、max、min、avg、list_agg、corr、covar_pop、covar_samp、stddev、stddev_pop、stddev_samp、var_pop、var_samp、variance、bit_and、bit_or、bool_and、bool_or、every、regr_avgx、regr_avgy、regr_count、regr_intercept、regr_r2、regr_slope、regr_sxx、regr_sxy、regr_syy、rank、spread。
  - 不支持ODBC相关绑定传参接口。
- XMLTYPE类型支持如下操作：
  - 物理备份恢复。
  - 比较表达式is null、is not null。
  - 条件表达式case、coalesce。
  - 全局临时表和本地临时表。
  - 强制类型转换。
  - 表达式索引。
  - XMLTYPE类型的输入值需要符合XML格式标准。
  - 支持gs_dump导出和gs_restore导入操作。
  - 并行查询，支持astore和ustore存储引擎。
  - 作为自定义函数的入参、出参、自定义变量和返回值。
  - 作为存储过程的入参、出参、自定义变量和返回值。支持自治事务的存储过程。
  - 字符处理函数quote_literal(string text)（需显式转换为字符类型）、quote_nullable(string text)（需显式转换为字符类型）。
  - 聚集函数count、array_agg、checksum（需显式转换为字符类型）、string_agg（需显式转换为字符类型）。
  - 自定义类型中复合类型涉及XMLTYPE type时的增删改查，且与普通表中的XMLTYPE字段一样，需要按照XMLTYPE的语法插入和修改。
  - 支持对该字段进行select、update、insert、delete，使用SQL语法输入XMLTYPE值。
- 可以创建名称为xmltype的schema，schema下用户可以创建函数，但是不能通过schema.func()这种方式调用schema下定义的函数。

7.3.18 SET 类型

SET类型是一种包含字符串成员的集合类型，在表字段创建时定义。

规格描述

1. SET类型成员个数最大为64个，最小为1个。不能定义为空集。
2. 成员名称长度最大为255个字符，允许使用空字符串作为成员名称。成员名称必须是字符常量，且不能是计算后得到的字符常量，如 SET('a' || 'b', 'c')。
3. 成员名称不能包含逗号，成员名称不能重复。
4. 不支持创建SET类型的数组和域类型。
5. 只有在sql_compatibility参数值为B兼容模式下支持SET类型。
6. 不支持SET类型作为分区表的分区键。
7. DROP TYPE 删除SET类型时，需要使用CASCADE方式删除，且关联的表字段也会被同时删除。
8. 对于USTORE存储方式的表，如果表中包含SET类型的字段，且已经开启回收站功能，表被删除时，不会进入到回收站中，会直接删除。
9. ALTER TABLE 不支持将SET类型字段的数据类型修改为其他SET类型。
10. 表或者SET类型关联的表字段被删除时，或者表字段的SET类型修改为其他类型时，SET数据类型也会被同步删除。
11. 不支持以CREATE TABLE { AS | LIKE } 的方式创建包含SET类型的表。
12. SET类型是随表字段创建的，其名称是组合而成的。如果schema中已经存在同名的数据类型，创建SET类型会失败。
13. SET类型支持与int2、int4、int8、text类型的=、<、>、<、<=、>、>=比较。
14. SET类型支持与int2、int4、int8、float4、float8、numeric、char、varchar、text、nvarchar2数据类型的转换。

注意事项

- SET类型的表字段值必须是SET类型定义的集合的子集。如：

```
-- 先创建B兼容模式的数据库并切换到B兼容数据库。
gaussdb=# CREATE DATABASE b_db dbcompatibility = 'B' encoding = 'utf8' lc_ctype = 'en_US.UTF-8'
lc_collate = 'en_US.UTF-8';
gaussdb=# \c b_db
gaussdb=# CREATE TABLE employee (
  name text,
  site SET('beijing','shanghai','nanjing','wuhan')
);
```

- site字段的值必须是上述集合定义中的子集，可以是空集合，如果提供的值在SET定义中的成员中不存在，会报错。如：

```
gaussdb=# INSERT INTO employee VALUES('zhangsan', 'nanjing,beijing');
INSERT 0 1
gaussdb=# INSERT INTO employee VALUES('zhangsan', 'hangzhou');
ERROR:  invalid input value for set employee_site_set: 'hangzhou'
LINE 1: INSERT INTO employee VALUES('zhangsan', 'hangzhou');
                                                ^
CONTEXT:  referenced column: site
```

- INSERT时无论用户提供的成员值顺序是怎样的，INSERT成功后，查询到的SET类型的值，其成员都是按照定义时的顺序输出的。

```
gaussdb=# SELECT * FROM employee;
   name   |      site
----------+-----------------
 zhangsan | beijing,nanjing
(1 rows)
```

- SET类型是以bitmap的方式存储的。SET类型的成员按照定义时的顺序，赋予不同的值。如：SET('beijing','shanghai','nanjing','wuhan') 的类型，对应的值如下：

表 7-24 SET 成员与其对应的数值

SET成员	成员值	二进制值
'beijing'	1	0001
'shanghai'	2	0010
'nanjing'	4	0100
'wuhan'	8	1000

因此，如果给SET类型的字段赋值为数值时，会转换为对应的子集。如：9对应的二进制值为 1001, 对应的子集是 'beijing,wuhan'。

```
gaussdb=# INSERT INTO employee values('lisi', 9);
INSERT 0 1
gaussdb=# SELECT * FROM employee;
   name   |      site
----------+-----------------
 zhangsan | beijing,nanjing
 lisi     | beijing,wuhan
(2 rows)
gaussdb=# DROP TABLE employee;
-- 切换回原来的数据库，注意postgres为原来的数据库名称。
gaussdb=# \c postgres
-- 删除创建的数据库。
gaussdb=# DROP database b_db;
```

7.3.19 aclitem 类型

aclitem数据类型是用来存储对象权限信息的，它的内部实现是int类型，支持的格式为‘*user1=privs/user2*’。

aclitem[]数据类型为aclitem组成的数组，支持的格式为‘{*user1=privs1/user3*，*user2=privs2/user3*}’。

其中user1，user2和user3为数据库中已存在的用户/角色名，privs为数据库中支持的权限（参见**表12-46**）。

示例：

```
--创建相应用户。
gaussdb=# CREATE USER user1 WITH PASSWORD 'Aa123456789';
gaussdb=# CREATE USER user2 WITH PASSWORD 'Aa123456789';
gaussdb=# CREATE USER omm WITH PASSWORD 'Aa123456789';

--新建一张数据表table_acl，有三个字段，类型分别为int、aclitem、aclitem[]
gaussdb=# CREATE TABLE table_acl (id int,priv aclitem,privs aclitem[]);--向数据表table_acl插入一条内容为(1,'user1=arw/omm','{omm=d/user2,omm=w/omm}')的数据
gaussdb=# INSERT INTO table_acl VALUES (1,'user1=arw/omm','{omm=d/user2,omm=w/omm}');
--向数据表table_acl再插入一条内容为(2,'user1=aw/omm','{omm=d/user2}')的数据
gaussdb=# INSERT INTO table_acl VALUES (2,'user1=aw/omm','{omm=d/user2}');

gaussdb=# SELECT * FROM  table_acl;
 id |     priv      |          privs
----+---------------+-------------------------
  1 | user1=arw/omm | {omm=d/user2,omm=w/omm}
  2 | user1=aw/omm  | {omm=d/user2}
(2 rows)

--删除表和用户。
gaussdb=# DROP USER user1;
```



```
gaussdb=# DROP USER user2;
gaussdb=# DROP USER omm;
gaussdb=# DROP TABLE table_acl;
```

7.3.20 数组类型

数组类型可以用来存储具有相同类型的若干元素。

数组类型的定义

一个数组数据类型一般通过在数组元素的数据类型名称后面加上方括号（[]）来命名。

示例一，创建一个名为sal_emp的表，它有一个表示雇员姓名类型为text的列（name），一个表示雇员季度工资的数组且元素类型为integer的列（pay_by_quarter），一个表示雇员手机号码的数组且元素类型为varchar(11)的列（phone_numbers）：

```
gaussdb=# CREATE TABLE sal_emp (
 name            text,
 pay_by_quarter  integer[],
 phone_numbers   varchar(11)[]
);
gaussdb=# DROP TABLE sal_emp;
```

示例二，其他方式定义一个数组类型，具体定义方法和定义行为参考示例中的注释：

```
gaussdb=# CREATE TABLE sal_emp (
 name             text,
 pay_by_quarter1  integer[][],      -- int类型的二维数组
 pay_by_quarter2  integer[3],       -- int类型的一维数组，尺寸大小为3
 pay_by_quarter3  integer[3][3],    -- int类型的二维数组，每一维尺寸大小为3
 pay_by_quarter4  integer ARRAY,    -- int类型的一维数组
 pay_by_quarter5  integer ARRAY[3]  -- int类型的一维数组，尺寸大小为3
);
gaussdb=# DROP TABLE sal_emp;
```

⚠ 注意

- 数组的维数定义功能并不生效（不影响运行时的行为），建议采用示例一的方式定义数组类型，并且不建议使用多维数组数据。
- 数组的尺寸定义功能并不生效（不影响运行时的行为），建议采用示例一的方式定义数组类型。
- 允许的数组数据维数最大为6。
- 数组元素个数限制如下：
  1. 元素个数最大为134217727个。
  2. 所有元素加起来最大存储空间不超过1GB - 1字节即1073741823字节。

数组构造器

数组构造器是一个能构建数组值的表达式。简单的数组构造器由关键词ARRAY、“[”、用于数组元素值的表达式列表（用逗号分隔）以及最后的“]”组成。示例如下：

```
gaussdb=# SELECT ARRAY[1, 2, 3 + 4];
  array
```



```
---------
 {1,2,7}
(1 row)
```

默认情况下，数组的元素类型是成员表达式的公共类型，使用和UNION或CASE结构（**UNION，CASE和相关构造**）相同的规则决定。可以通过显式类型转换将数组构造为想要的数据类型，示例如下：

```
gaussdb=# SELECT ARRAY[1, 2, 3]::varchar[];
  array
---------
 {1,2,3}
(1 row)

gaussdb=# SELECT ARRAY['a', 'b', 'c'];
  array
---------
 {a,b,c}
(1 row)

gaussdb=# SELECT ARRAY['a', 'b', 'c']::int[];
ERROR:  invalid input syntax for integer: "a"
LINE 1: select ARRAY['a', 'b', 'c']::int[];
                ^
CONTEXT:  referenced column: array

gaussdb=# SELECT ARRAY[1::int, 'b', 'c'];
ERROR:  invalid input syntax for integer: "b"
LINE 1: select ARRAY[1::int, 'b', 'c'];
                             ^
CONTEXT:  referenced column: array
```

除预置的基础类型外，record类型和表类型也可以定义其数组类型，示例：

```
gaussdb=# CREATE TYPE rec IS (c1 int, c2 int);
gaussdb=# SELECT ARRAY[(1, 1), (2, 2)]::rec[];
       array
-------------------
 {"(1,1)","(2,2)"}
(1 row)

gaussdb=# CREATE TABLE tab (c1 int, c2 int);
gaussdb=# SELECT ARRAY[(1, 1), (2, 2)]::tab[];
       array
-------------------
 {"(1,1)","(2,2)"}
(1 row)

gaussdb=# DROP TYPE rec;
gaussdb=# DROP TABLE tab;
```

因为数组必须得有类型，因此在构造一个空数组时，必须明确的将其构造成需要的类型，示例：

```
gaussdb=# SELECT ARRAY[]::int[];
 array
-------
 {}
(1 row)
```

我也可以从子查询的结果中构造一个数组。此时， 数组构造器是关键字ARRAY后跟着用圆括号括起来的子查询，子查询必须只返回一个单独的字段。生成的一维数组将为子查询里每行结果生成一个元素， 元素类型匹配子查询的输出字段。示例：

```
gaussdb=# SELECT ARRAY(select generate_series(1, 6));
    array
---------------
 {1,2,3,4,5,6}
(1 row)
```

多维数组值可以通过嵌套数组构造器的方法来制作。内层构造器中的ARRAY关键字可以省略。比如，下面两个示例是同样的结果：

```
gaussdb=# SELECT ARRAY[ARRAY[1,2], ARRAY[3,4]];
     array
---------------
 {{1,2},{3,4}}
(1 row)

gaussdb=# SELECT ARRAY[[1,2], [3,4]];
     array
---------------
 {{1,2},{3,4}}
(1 row)
```

说明

- 同层的内层构造器必须生成同维的子数组。
- 任何应用于外层ARRAY构造器的类型转换自动的应用到所有的内层构造器。

数组类型的字符串输入

要把一个数组值写成一个文字常数（常量输入），将元素值用花括号包围并用逗号分隔。因此，一个数组常量的一般格式如下：

```
'{ val1 delim val2 delim ... }'
```

上述格式中的delim是元素类型的分隔符，记录在类型的pg_type表的typdelim列中。每个val可以是数组元素类型的一个常量，也可以是一个子数组。示例如下：

```
gaussdb=# SELECT '{1, 2, 3}'::int[] AS RESULT;
 result
---------
 {1,2,3}
(1 row)

gaussdb=# SELECT '{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}'::int[] AS RESULT;
          result
----------------------------
 {{1,2,3},{4,5,6},{7,8,9}}
(1 row)
```

在任意元素值周围可以使用双引号，并且在元素值包含逗号或花括号等一些特殊字符时必须使用双引号。示例如下：

```
gaussdb=# SELECT '{" ", "NULL", null, "\\", "{", "}", ","}'::varchar[] AS RESULT;
               result
-------------------------------------
 {" ","NULL",NULL,"\\","{","}",","}
(1 row)

-- 该示例表示有一个varchar类型的数组，且一共有7个varchar元素，元素依次为：
1、包含一个空格的字符串 2、值为“NULL”的字符串 3、字符串为NULL 4、有一个\字符的字符串
5、有一个{字符的字符串 6、有一个}字符的字符串 7、有一个,字符的字符串
```

说明

- 对于数组字符串常量输入，如果数组元素值是空字符串或者包含花括号、分隔符、双引号、反斜杠、空白或者匹配关键字NULL。则这些元素输入需要使用双引号，在元素值里包含的双引号和反斜杠时需要额外添加一个反斜杠。
- 关键字NULL不区分大小写。
- 输入会自动跳过没有使用双引号的空白。
- 一般不建议使用字符常量的方式构造数组数据，推荐使用ARRAY构造器。

数组类型的字符串输出

一个数组值的输出表现形式由该数组元素类型的输出再加上一些标明该数组结构的修饰组成。这些修饰由围绕在数组值周围的花括号（“{ ”和“}”）加上相邻项之间的分隔字符组成。在多维数组里， 每个维都有自己级别的花括号，并且在同级相邻的花括号项之间包含分隔符。

数组类型数据包含特殊字符（下述说明中的字符），字符串输出示例：

```
gaussdb=# SELECT ARRAY['{', '}', 'hello, world', '"', '\', '  ', NULL] AS RESULT;
                  array
-----------------------------------------------
 {"{","}","hello, world","\"","\\","  ",NULL}
(1 row)
```

说明

对于数组字符串常量输出，如果数组元素值是空字符串或者包含花括号、分隔符、双引号、反斜杠、空白 或者元素为NULL，则这些元素输出时会输出双引号中，双引号和反斜杠则会被反斜杠转义额外输出一个反斜杠。与字符串常量输入相对应。

数组类型的使用

数组类型的使用示例如下：

```
-- 创建有数组类型列的表,并插入一些数据
gaussdb=# CREATE TABLE orders (
name  varchar,
items varchar[]
);
gaussdb=# INSERT INTO orders VALUES('a', ARRAY['苹果', '橘子', '梨']);
gaussdb=# INSERT INTO orders VALUES('b', ARRAY['矿泉水', '可乐', '雪碧']);
gaussdb=# INSERT INTO orders VALUES('c', ARRAY['鼠标', '键盘', '耳机']);
gaussdb=# INSERT INTO orders VALUES('d', '{白菜, 土豆, 茄子}');

-- 查询数据
gaussdb=# SELECT * FROM orders ORDER BY name;
 name |       items
------+--------------------
 a    | {苹果,橘子,梨}
 b    | {矿泉水,可乐,雪碧}
 c    | {鼠标,键盘,耳机}
 d    | {白菜,土豆,茄子}
(4 rows)

-- 访问数组元素
gaussdb=# SELECT items[1] FROM orders ORDER BY name;
 items
--------
 苹果
 矿泉水
 鼠标
 白菜
(4 rows)

-- 访问元素超过范围或者访问下标为NULL时会返回NULL
gaussdb=# SELECT items[4] FROM orders ORDER BY name;
 items
-------




(4 rows)
```



```
gaussdb=# SELECT items[null] FROM orders ORDER BY name;
 items
-------




(4 rows)

-- 访问子数组
gaussdb=# SELECT items[1:2] FROM orders ORDER BY name;
     items
---------------
 {苹果,橘子}
 {矿泉水,可乐}
 {鼠标,键盘}
 {白菜,土豆}
(4 rows)

-- 更新整个数组
gaussdb=# UPDATE orders SET items = ARRAY['香蕉', '西瓜', '草莓'] WHERE name = 'a';
gaussdb=# SELECT items FROM orders WHERE name = 'a';
      items
------------------
 {香蕉,西瓜,草莓}
(1 row)

-- 更新数组的元素
gaussdb=# UPDATE orders SET items[1] = '芒果' WHERE name = 'a';
gaussdb=# SELECT items FROM orders WHERE name = 'a';
      items
------------------
 {芒果,西瓜,草莓}
(1 row)

-- 更新数组的元素片段
gaussdb=# UPDATE orders SET items[1:2] = ARRAY['电脑', '手机'] WHERE name = 'c';
gaussdb=# SELECT items FROM orders WHERE name = 'c';
      items
-----------------
 {电脑,手机,耳机}
(1 row)

-- 添加数组元素，位于原数组最后一个元素和这个新元素之间的未赋值元素为NULL
gaussdb=# UPDATE orders SET items[4] = '显示器' WHERE name = 'c';
gaussdb=# SELECT items FROM orders WHERE name = 'c';
          items
-------------------------
 {电脑,手机,耳机,显示器}
(1 row)

gaussdb=# UPDATE orders SET items[6] = '显示器2' WHERE name = 'c';
gaussdb=# SELECT items FROM orders WHERE name = 'c';
                items
-------------------------------------
 {电脑,手机,耳机,显示器,NULL,显示器2}
(1 row)

gaussdb=# DROP TABLE orders;
```

7.4 常量与宏

GaussDB支持的常量和宏请参见**表7-25**。

表 7-25 常量和宏

参数	描述	示例
CURRENT_CATALOG	当前数据库	testdb=# SELECT CURRENT_CATALOG; current_database ---------- testdb (1 row)
CURRENT_ROLE	当前用户	gaussdb=# SELECT CURRENT_ROLE; current_user ---------- omm (1 row)
CURRENT_SCHEMA	当前数据库模式	gaussdb=# SELECT CURRENT_SCHEMA; current_schema ---------- public (1 row)
CURRENT_USER	当前用户	gaussdb=# SELECT CURRENT_USER; current_user ---------- omm (1 row)
LOCALTIMESTAMP	当前会话时间（无时区）	gaussdb=# SELECT LOCALTIMESTAMP; timestamp ---------- 2015-10-10 15:37:30.968538 (1 row)
NULL	空值	-
SESSION_USER	当前系统用户	gaussdb=# SELECT SESSION_USER; session_user ---------- omm (1 row)
SYSDATE	当前系统日期	gaussdb=# SELECT SYSDATE; sysdate ---------- 2015-10-10 15:48:53 (1 row)
USER	当前用户，此用户为CURRENT_USER的别名。	gaussdb=# SELECT USER; current_user ---------- omm (1 row)

7.5 函数和操作符

操作符可以对一个或多个操作数进行处理，位置上可能处于操作数之前、之后，或两个操作数中间。完成处理之后，返回处理结果。

函数是对一些业务逻辑的封装，以完成特定的功能。函数可以有参数，也可以没有参数。函数是有返回类型的，执行完成后，会返回执行结果。

对于系统函数，用户可以进行修改，但是修改之后系统函数的语义可能会发生改变，从而导致系统控制紊乱。正常情况下不允许用户手工修改系统函数。

说明

- 当GUC参数behavior_compat_options含有'enable_funcname_with_argsname'选项时，投影别名显示完整函数。
- 当GUC参数enable_volatile_match_index设置为ON，且DBCOMPATIBILITY 设置为A时，volatile类型函数可以匹配索引。volatile函数在部分索引下，不确保可以命中索引；在函数执行过程中含有隐式转换时，不确保命中索引。在本就不可以命中索引的场景中，开启此选项，volatile函数依然不能够命中索引。
- 当DBCOMPATIBILITY 设置为A，sql语句中immutable存储过程执行中传入常量或者可以被转换为常量的表达式时（例如immutable函数，但是stable、volatile函数不可以），非每行执行一次。immutable存储过程在入参为行表达式时仍然每行执行一次。immutable存储过程在包含union、union all、order by的sql语句中，执行次数减少但不会只执行一次。

7.5.1 逻辑操作符

常用的逻辑操作符有AND、OR和NOT，其运算结果分别为TRUE、FALSE和NULL，其中NULL代表未知。运算优先级顺序为：NOT>AND>OR。

运算规则请参见表7-26，表中的a和b代表逻辑表达式。

表 7-26 运算规则表

a	b	a AND b的结果	a OR b的结果	NOT a的结果
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
TRUE	NULL	NULL	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE
FALSE	NULL	FALSE	NULL	TRUE
NULL	NULL	NULL	NULL	NULL

说明

- 操作符AND和OR具有交换性，即交换左右两个操作数，不影响其结果。
- 不支持对XML类型数据进行操作。

7.5.2 比较操作符

大部分数据类型都可用比较操作符进行比较，并返回一个布尔类型的值。

比较操作符均为双目操作符，被比较的两个数据类型必须是相同的数据类型或者是可以进行隐式转换的类型。

GaussDB提供的比较操作符请参见表7-27。

表 7-27 比较操作符

操作符	描述
<	小于
>	大于
<=	小于或等于
>=	大于或等于
=	等于
<> 或 !=或^=	不等于

- 比较操作符可以用于所有相关的数据类型。所有比较操作符都是双目操作符，返回布尔类型数值。
- 不等号的计算优先级高于等号。当输入的数据类型不同且无法隐式转换时，比较操作将会失败。例如像1<2<3这样的表达式是非法的，因为布尔值和3之间无法用小于号（<）比较。
- 另外，上述每种操作符在pg_proc系统表中都有对应的函数，如果其对应的函数的属性proleakproof值为f，表示该函数不是防数据泄露的。如果用户只拥有视图权限而不拥有该视图对应表的权限，在查询该视图的时候，可能存在查询计划不是最优的问题。
- 不支持XML类型数据。

7.5.3 字符处理函数和操作符

GaussDB提供的字符处理函数和操作符主要用于字符串与字符串、字符串与非字符串之间的连接，以及字符串的模式匹配操作。注意：字符串处理函数除了length相关函数，其他函数和操作符不支持大于1GBclob作为参数。

- bit_length(string)

  描述：字符串的位数。

  返回值类型：int

  示例：

  ```
  gaussdb=# SELECT bit_length('world');
   bit_length
  ------------
          40
  (1 row)
  ```

- btrim(string text [, characters text])

  描述：从string开头和结尾删除只包含characters中字符（缺省是空白）的最长字符串。

  返回值类型：text

  示例：

  ```
  gaussdb=# SELECT btrim('sring' , 'ing');
   btrim
  -------
   sr
  (1 row)
  ```


- char_length(string)或character_length(string)

  描述：字符串中的字符个数。

  返回值类型：int

  示例：

```
gaussdb=# SELECT char_length('hello');
 char_length
-------------
           5
(1 row)
```

- dump(expr[, return_fmt [, start_position [, length]]])

  描述：返回输入表达式的数据类型代码、字节长度和内部表示形式。return_fmt 指定内部表现形式的进制，start_position 指定从第几个字节开始，length 表示读取的长度。

  返回值类型：text

  说明

  此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s2的情况下有效。

- instr(text,text,int,int)

  描述：instr(string1,string2,int1,int2)返回在string1中从int1位置开始匹配到第int2次string2的位置，第一个int表示开始匹配起始位置，第二个int表示匹配的次数。

  返回值类型：int

  示例：

```
gaussdb=# SELECT instr( 'abcdabcdabcd', 'bcd', 2, 2 );
 instr
--------
      6
(1 row)
```

- instrb(text,text,int,int)

  描述：instrb(string1,string2,int1,int2)返回在string1中从int1位置开始匹配到第int2次string2的位置，第一个int表示开始匹配起始位置，第二个int表示匹配的次数。与instr函数不同的是，instrb固定以字节为单位，不受所使用的字符集影响。

  返回值类型：int

  示例：

```
gaussdb=# SELECT instrb( 'abcdabcdabcd', 'bcd', 2, 2 );
 instrb
--------
      6
(1 row)
```

  说明

  - 此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s1的情况下有效。
  - 参数int1，int2入参若为小数则不会被四舍五入，而是被截断。

- lengthb(text/bpchar)

  描述：获取指定字符串的字节数。

  返回值类型：int

  示例：


```
gaussdb=# SELECT lengthb('hello');
 lengthb
---------
       5
(1 row)
```

- left(str text, n int)

  描述：返回字符串的前n个字符。当n是负数时，返回除最后|n|个字符以外的所有字符。

  返回值类型：text

  示例：

```
gaussdb=# SELECT left('abcde', 2);
 left
------
 ab
(1 row)
```

- length(string bytea, encoding name)

  描述：指定encoding编码格式的string的字符数。在这个编码格式中，string必须是有效的。

  返回值类型：int

  示例：

```
gaussdb=# SELECT length('jose', 'UTF8');
 length
--------
      4
(1 row)
```

  说明

  如果是查询bytea类型的长度，指定utf8编码时，最大长度只能为536870888。

- lpad(string text, length int [, fill text])

  描述：通过填充字符fill（缺省时为空白），把string填充为length长度。如果string已经比length长则将其尾部截断。

  返回值类型：text

  说明

  此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s1的情况下：

  - 参数length表示字符串显示长度，单个字符的显示长度按照A兼容处理。当lpad函数执行过程中出现length剩余长度为1且下一个字符显示长度为2时，在字符串左侧添加一个空格字符。
  - 参数length入参若为小数则不会被四舍五入，而是被截断。

- notlike(x bytea name text, y bytea text)

  描述：比较x和y是否不一致。

  返回值类型：Boolean

  示例：

```
gaussdb=# SELECT notlike(1,2);
   notlike
--------------
          t
(1 row)
gaussdb=# SELECT notlike(1,1);
   notlike
--------------
```



```
          f
(1 row)
```

- octet_length(string)

  描述：字符串中的字节数。

  返回值类型：int

  示例：

```
gaussdb=# SELECT octet_length('jose');
 octet_length
--------------
            4
(1 row)
```

- overlay(string placing string FROM int [for int])

  描述：替换子字符串。FROM int表示从第一个string的第几个字符开始替换，for int表示替换第一个string的字符数目。

  返回值类型：text

  示例：

```
gaussdb=# SELECT overlay('hello' placing 'world' from 2 for 3 );
 overlay
---------
 hworldo
(1 row)
```

- position(substring in string)

  描述：指定子字符串的位置。字符串区分大小写。

  返回值类型：int，字符串不存在时返回0。

  示例：

```
gaussdb=# SELECT position('ing' in 'string');
 position
----------
        4
(1 row)
```

- pg_client_encoding()

  描述：当前客户端编码名称。

  返回值类型：name

  示例：

```
gaussdb=# SELECT pg_client_encoding();
 pg_client_encoding
--------------------
 UTF8
(1 row)
```

- quote_ident(string text)

  描述：返回适用于SQL语句的标识符形式（使用适当的引号进行界定）。只有在必要的时候才会添加引号（字符串包含非标识符字符或者会转换大小写的字符）。返回值中嵌入的引号都写了两次。

  返回值类型：text

  示例：

```
gaussdb=# SELECT quote_ident('hello world');
 quote_ident
--------------
 "hello world"
(1 row)
```

- quote_literal(string text)

描述：返回适用于在SQL语句里当作文本使用的形式（使用适当的引号进行界定）。支持显式转换成字符类型后的XML类型数据。

返回值类型：text

示例：

```
gaussdb=# SELECT quote_literal('hello');
 quote_literal
---------------
 'hello'
(1 row)
```

如果出现如下写法，text文本将进行转义。

```
gaussdb=# SELECT quote_literal(E'O\'hello');
 quote_literal
---------------
 'O''hello'
(1 row)
```

如果出现如下写法，反斜杠会写入两次。

```
gaussdb=# SELECT quote_literal('O\hello');
 quote_literal
---------------
 E'O\\hello'
(1 row)
```

如果参数为NULL，返回空。如果参数可能为null，通常使用函数quote_nullable更适用。

```
gaussdb=# SELECT quote_literal(NULL);
 quote_literal
---------------

(1 row)
```

- quote_literal(value anyelement)

  描述：将给定的值强制转换为text，加上引号作为文本。

  返回值类型：text

  示例：

  ```
  gaussdb=# SELECT quote_literal(42.5);
   quote_literal
  ---------------
   '42.5'
  (1 row)
  ```

  如果出现如下写法，定值将进行转义。

  ```
  gaussdb=# SELECT quote_literal(E'O\'42.5');
   quote_literal
  ---------------
   '0''42.5'
  (1 row)
  ```

  如果出现如下写法，反斜杠会写入两次。

  ```
  gaussdb=# SELECT quote_literal('O\42.5');
   quote_literal
  ---------------
   E'O\\42.5'
  (1 row)
  ```

- quote_nullable(string text)

  描述：返回适用于在SQL语句里当作字符串使用的形式（使用适当的引号进行界定）。

  支持显式转换成字符类型后的XML类型数据。

  返回值类型：text

示例：

```
gaussdb=# SELECT quote_nullable('hello');
 quote_nullable
----------------
 'hello'
(1 row)
```

如果出现如下写法，text文本将进行转义。

```
gaussdb=# SELECT quote_nullable(E'O\'hello');
 quote_nullable
----------------
 'O''hello'
(1 row)
```

如果出现如下写法，反斜杠会写入两次。

```
gaussdb=# SELECT quote_nullable('O\hello');
 quote_nullable
----------------
 E'O\\hello'
(1 row)
```

如果参数为NULL，返回NULL。

```
gaussdb=# SELECT quote_nullable(NULL);
 quote_nullable
----------------
 NULL
(1 row)
```

- quote_nullable(value anyelement)

  描述：将给定的参数值转换为text，加上引号作为文本。

  返回值类型：text

  示例：

  ```
  gaussdb=# SELECT quote_nullable(42.5);
   quote_nullable
  ----------------
   '42.5'
  (1 row)
  ```

  如果出现如下写法，定值将进行转义。

  ```
  gaussdb=# SELECT quote_nullable(E'O\'42.5');
   quote_nullable
  ----------------
   'O''42.5'
  (1 row)
  ```

  如果出现如下写法，反斜杠会写入两次。

  ```
  gaussdb=# SELECT quote_nullable('O\42.5');
   quote_nullable
  ----------------
   E'O\\42.5'
  (1 row)
  ```

  如果参数为NULL，返回NULL。

  ```
  gaussdb=# SELECT quote_nullable(NULL);
   quote_nullable
  ----------------
   NULL
  (1 row)
  ```

- substring_inner(string [from int] [for int])

  描述：截取子字符串，from int表示从第几个字符开始截取，for int表示截取几个字节。

  返回值类型：text

  示例：

```
gaussdb=# SELECT substring_inner('adcde', 2,3);
 substring_inner
-----------------
 dcd
(1 row)
```

- substring(string [from int] [for int])

  描述：截取子字符串，from int表示从第几个字符开始截取，for int表示截取几个字节。

  返回值类型：text

  示例：

```
gaussdb=# SELECT substring('Thomas' from 2 for 3);
 substring
-----------
 hom
(1 row)
```

- substring(string from *pattern*)

  描述：截取匹配POSIX正则表达式的子字符串。如果没有匹配它返回空值，否则返回文本中匹配模式的那部分。

  返回值类型：text

  示例：

```
gaussdb=# SELECT substring('Thomas' from '...$');
 substring
-----------
 mas
(1 row)
gaussdb=# SELECT substring('foobar' from 'o(.)b');
 result
--------
 o
(1 row)
gaussdb=# SELECT substring('foobar' from '(o(.)b)');
 result
--------
 oob
(1 row)
```

  说明

  如果POSIX正则表达式模式包含任何圆括号，那么将返回匹配第一对子表达式（对应第一个左圆括号的）的文本。如果你想在表达式里使用圆括号而又不想导致这个例外，那么你可以在整个表达式外边放上一对圆括号。

- substring(string from *pattern* for *escape*)

  描述：截取匹配SQL正则表达式的子字符串。声明的模式必须匹配整个数据串，否则函数失败并返回空值。为了标识在成功的时候应该返回的模式部分，模式必须包含两个逃逸字符引用的部分，并且逃逸字符后面要添加双引号（"）。匹配这两个标记之间的模式的文本将被返回。

  返回值类型：text

  示例：

```
gaussdb=# SELECT substring('Thomas' from '%#"o_a#"_' for '#');
 substring
-----------
 oma
(1 row)
```

- rawcat(raw,raw)

  描述：字符串拼接函数。

  返回值类型：raw

示例：

```
gaussdb=# SELECT rawcat('ab','cd');
 rawcat
--------
 ABCD
(1 row)
```

- regexp_like(text,text,text)

  描述：正则表达式的模式匹配函数。

  返回值类型：bool

  示例：

```
gaussdb=# SELECT regexp_like('str','[ac]');
 regexp_like
-------------
 f
(1 row)
```

- regexp_substr(string text, pattern text [, position int [, occurrence int [, flags text]]])

  描述：正则表达式的抽取子串函数。与substr功能相似，正则表达式出现多个并列的括号时，也全部处理。

  参数说明：

  - string：用于匹配的源字符串。
  - pattern：用于匹配的正则表达式模式串。
  - position：可选参数，表示从源字符串的第几个字符开始匹配，默认值为1。
  - occurrence：可选参数，表示抽取第几个满足匹配的子串，默认值为1。
  - flags：可选参数，包含零个或多个改变函数匹配行为的单字母标记。flags 支持的选项值及含义描述如**表7-28**所示：

    表 7-28 flags 支持的选项值

选项	描述
'b'	按照无扩展的 BRE 规则匹配。
'c'	大小写敏感匹配。
'e'	按照扩展的 ERE 规则匹配。
'i'	大小写不敏感匹配。
'm'	多行模式匹配。flags 中包含'm' 时，按照多行模式匹配，否则按照单行模式匹配。
'n'	'n'选项的含义和GUC参数**behavior_compat_options**及数据库当前的兼容模式有关： ● 数据库SQL语法兼容模式为A或B，且GUC参数**behavior_compat_options**值包含aformat_regexp_match时，'n'选项表示“.”能够匹配换行符('\n')；flags未指定'n'选项时，“.”不会匹配换行符。 ● 其他情况下，'n'选项和'm'选项的含义一样。
'p'	部分新行敏感的匹配，影响'.'和方括号表达式，和新行敏感的匹配('m'或'n')一样，但是不影响^和$。

选项	描述
'q'	普通字符匹配。
's'	单行模式匹配，含义与'm'、 'n'相反。
't'	紧凑模式匹配，空白符匹配自身。
'w'	逆部分新行匹配，与'p'含义相反。
'x'	宽松模式匹配，忽略空白符。

返回值类型：text

示例：

```
gaussdb=# SELECT regexp_substr('str','[ac]');
 regexp_substr
---------------


(1 row)

gaussdb=# SELECT regexp_substr('foobarbaz', 'b(..)', 3, 2) AS RESULT;
 result
--------
 baz
(1 row)
```

- regexp_count(string text, pattern text [, position int [, flags text]])

  描述：获取满足匹配的子串个数。

  参数说明：

  - string：用于匹配的源字符串。
  - pattern：用于匹配的正则表达式模式串。
  - position：表示从源字符串的第几个字符开始匹配，为可选参数，默认值为1。
  - flags：可选参数，包含零个或多个改变函数匹配行为的单字母标记。flags 支持的选项值及含义描述如**表7-28**所示。

  说明

  此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s1的情况下，以'\'字符为结尾的pattern参数为合法的。

  返回值类型：int

  示例：

  ```
  gaussdb=# SELECT regexp_count('foobarbaz','b(..)', 5) AS RESULT;
  result
  --------
  1
  (1 row)
  ```

- regexp_instr(string text, pattern text [, position int [, occurrence int [, return_opt int [, flags text]]]])

  描述：获取满足匹配条件的子串位置（从1开始）。如果没有匹配的子串，则返回0。

  参数说明：

  - string：用于匹配的源字符串。

- pattern：用于匹配的正则表达式模式串。
- position：可选参数，表示从源字符串的第几个字符开始匹配，默认值为1。
- occurrence：可选参数，表示获取第occurrence个匹配子串的位置，默认值为1。
- return_opt：可选参数，用于控制返回匹配子串的首字符位置还是尾字符位置。取值为0时，返回匹配子串的第一个字符的位置（从1开始计算），取值为大于0的值时，返回匹配子串的尾字符的下一个字符的位置。默认值为0。
- flags：可选参数，包含零个或多个改变函数匹配行为的单字母标记。flags 支持的选项值及含义描述如表7-28所示。

返回值类型：int

示例：

```
gaussdb=# SELECT regexp_instr('foobarbaz','b(..)', 1, 1, 0) AS RESULT;
result
--------
4
(1 row)

gaussdb=# SELECT regexp_instr('foobarbaz','b(..)', 1, 2, 0) AS RESULT;
result
--------
7
(1 row)
```

- regexp_matches(string text, pattern text [, flags text])

  描述：返回string中所有匹配POSIX正则表达式的子字符串。如果pattern不匹配，该函数不返回行。如果模式不包含圆括号子表达式，则每一个被返回的行都是一个单一元素的文本数组，其中包括匹配整个模式的子串。如果模式包含圆括号子表达式，该函数返回一个文本数组，它的第n个元素是匹配模式的第n个圆括号子表达式的子串。

  flags参数为可选参数，包含零个或多个改变函数行为的单字母标记。i表示进行大小写无关的匹配，g表示替换每一个匹配的子字符串而不仅仅是第一个。

  须知

  如果提供了最后一个参数，但参数值是空字符串（''），且数据库SQL兼容模式设置为A的情况下，会导致返回结果为空集。这是因为A兼容模式将''作为NULL处理，避免此类行为的方式有如下几种：

  - 将数据库SQL兼容模式改为C。
  - 不提供最后一个参数，或最后一个参数不为空字符串。

  返回值类型：setof text[]

  示例：

```
gaussdb=# SELECT regexp_matches('foobarbequebaz', '(bar)(beque)');
 regexp_matches
----------------
 {bar,beque}
(1 row)
gaussdb=# SELECT regexp_matches('foobarbequebaz', 'barbeque');
 regexp_matches
----------------
 {barbeque}
(1 row)
gaussdb=#  SELECT regexp_matches('foobarbequebazilbarfbonk', '(b[^b]+)(b[^b]+)', 'g');
regexp_matches
```



```
--------------
 {bar,beque}
 {bazil,barf}
(2 rows)
```

- regexp_split_to_array(string text, pattern text [, flags text])

  描述：用POSIX正则表达式作为分隔符，分隔string。和regexp_split_to_table相同，不过regexp_split_to_array会把它的结果以一个text数组的形式返回。

  返回值类型：text[]

  示例：

```
gaussdb=# SELECT regexp_split_to_array('hello world', E'\\s+');
 regexp_split_to_array
-----------------------
 {hello,world}
(1 row)
```

- regexp_split_to_table(string text, pattern text [, flags text])

  描述：用POSIX正则表达式作为分隔符，分隔string。如果没有与pattern的匹配，该函数返回string。如果至少有一个匹配，对每一个匹配，它都返回从上一个匹配的末尾（或者串的开头）到这次匹配开头之间的文本。当没有更多匹配时，它返回从上一次匹配的末尾到串末尾之间的文本。

  flags参数包含零个或多个改变函数行为的单字母标记。i表示进行大小写无关的匹配。

  返回值类型：setof text

  示例：

```
gaussdb=# SELECT regexp_split_to_table('hello world', E'\\s+');
 regexp_split_to_table
-----------------------
 hello
 world
(2 rows)
```

- repeat(string text, number int)

  描述：将string重复number次。

  返回值类型：text

  示例：

```
gaussdb=# SELECT repeat('Pg', 4);
  repeat
----------
 PgPgPgPg
(1 row)
```

  说明

  由于数据库内存分配机制限制单次内存分配不可超过1GB，因此number最大值不应超过(1G-x)/lengthb(string) - 1。x为头信息长度，通常大于4字节，其具体值在不同的场景下存在差异。

- replace(string text, from text, to text)

  描述：把字符串string里出现的所有子字符串from的内容替换成子字符串to的内容。

  返回值类型：text

  示例：

```
gaussdb=# SELECT replace('abcdefabcdef', 'cd', 'XXX');
    replace
----------------
 abXXXefabXXXef
(1 row)
```


- replace(string, substring)

  描述：删除字符串string里出现的所有子字符串substring的内容。

  string类型：text

  substring类型：text

  返回值类型：text

  示例：

```
gaussdb=# SELECT replace('abcdefabcdef', 'cd');
   replace
----------------
 abefabef
(1 row)
```

- reverse(str)

  描述：返回颠倒的字符串（按字符颠倒）。

  返回值类型：text

  示例：

```
gaussdb=# SELECT reverse('abcde');
 reverse
---------
 edcba
(1 row)
```

- right(str text, n int)

  描述：返回字符串中的后n个字符。当n是负值时，返回除前|n|个字符以外的所有字符。

  返回值类型：text

  示例：

```
gaussdb=# SELECT right('abcde', 2);
 right
-------
 de
(1 row)

gaussdb=# SELECT right('abcde', -2);
 right
-------
 cde
(1 row)
```

- rpad(string text, length int [, fill text])

  描述：使用填充字符fill（缺省时为空白），把string填充到length长度。如果string已经比length长则将其从尾部截断。

  返回值类型：text

  说明

  此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s1的情况下：

  - 参数length表示字符串显示长度，单个字符的显示长度按照O兼容处理。当rpad函数执行过程中出现length剩余长度为1且下一个字符显示长度为2时，在字符串右侧添加一个空格字符。
  - 参数length入参若为小数则不会被四舍五入，而是被截断。

- substrb(text,int,int)

  描述：提取子字符串，第一个int表示提取的起始位置，第二个表示提取几位字符。

返回值类型：text

示例：

```
gaussdb=# SELECT substrb('string',2,3);
 substrb
---------
 tri
(1 row)
```

- substrb(text,int)

  描述：提取子字符串，int表示提取的起始位置。

  返回值类型：text

  示例：

  ```
  gaussdb=# SELECT substrb('string',2);
   substrb
  ---------
   tring
  (1 row)
  ```

- substr(bytea,from,count)

  描述：从参数bytea中抽取子字符串。from表示抽取的起始位置，count表示抽取的子字符串长度。

  返回值类型：text

  示例：

  ```
  gaussdb=# SELECT substr('string',2,3);
   substr
  --------
   tri
  (1 row)
  ```

- string || string

  描述：连接字符串。

  返回值类型：text

  示例：

  ```
  gaussdb=# SELECT 'MPP'||'DB' AS RESULT;
   result
  --------
   MPPDB
  (1 row)
  ```

- string || non-string或non-string || string

  描述：连接字符串和非字符串。

  返回值类型：text

  示例：

  ```
  gaussdb=# SELECT 'Value: '||42 AS RESULT;
    result
  -----------
   Value: 42
  (1 row)
  ```

- split_part(string text, delimiter text, field int)

  描述：根据delimiter分隔string返回生成的第field个子字符串（从出现第一个delimiter的text为基础）。

  返回值类型：text

  示例：

  ```
  gaussdb=# SELECT split_part('abc~@~def~@~ghi', '~@~', 2);
   split_part
  ```



```
------------
 def
(1 row)
```

- strpos(string, substring)

  描述：指定的子字符串的位置。和position(substring in string)一样，不过参数顺序相反。

  返回值类型：int

  示例：

```
gaussdb=# SELECT strpos('source', 'rc');
 strpos
--------
      4
(1 row)
```

- to_hex(number int or bigint)

  描述：把number转换成十六进制表现形式。

  返回值类型：text

  示例：

```
gaussdb=# SELECT to_hex(2147483647);
  to_hex
----------
 7fffffff
(1 row)
```

- translate(string text, from text, to text)

  描述：把在string中包含的任何匹配from中的字符转换为对应的在to中的字符。如果from比to长，删掉在from中出现的额外的字符。

  返回值类型：text

  示例：

```
gaussdb=# SELECT translate('12345', '143', 'ax');
 translate
-----------
 a2x5
(1 row)
```

- length(string)

  描述：获取参数string中字符的数目。

  返回值类型：integer

  示例：

```
gaussdb=# SELECT length('abcd');
 length
--------
      4
(1 row)
```

- lengthb(string)

  描述：获取参数string中字节的数目。与字符集有关，同样的中文字符，在GBK与UTF8中，返回的字节数不同。

  返回值类型：integer

  示例：

```
gaussdb=# SELECT lengthb('Chinese');
 lengthb
---------
       7
(1 row)
```


- substr(string,from)

  描述：

  从参数string中抽取子字符串。

  from表示抽取的起始位置。

  - from为0时，按1处理。
  - from为正数时，抽取从from到末尾的所有字符。
  - from为负数时，抽取字符串的后n个字符，n为from的绝对值。

  返回值类型：text

  示例：

  from为正数时：

  ```
  gaussdb=# SELECT substr('ABCDEF',2);
   substr
  --------
   BCDEF
  (1 row)
  ```

  from为负数时：

  ```
  gaussdb=# SELECT substr('ABCDEF',-2);
   substr
  --------
   EF
  (1 row)
  ```

- substr(string,from,count)

  描述：

  从参数string中抽取子字符串。

  from表示抽取的起始位置。

  count表示抽取的子字符串长度。

  - from为0时，按1处理。
  - from为正数时，抽取从from开始的count个字符。
  - from为负数时，抽取从倒数第n个开始的count个字符，n为from的绝对值。
  - count小于1时，返回null。

  返回值类型：text

  示例：

  from为正数时：

  ```
  gaussdb=# SELECT substr('ABCDEF',2,2);
   substr
  --------
   BC
  (1 row)
  ```

  from为负数时：

  ```
  gaussdb=# SELECT substr('ABCDEF',-3,2);
   substr
  --------
   DE
  (1 row)
  ```

- substrb(string,from)

  描述：该函数和SUBSTR(string,from)函数功能一致，但是计算单位为字节。

  返回值类型：text

  示例：


```
gaussdb=# SELECT substrb('ABCDEF',-2);
 substrb
---------
 EF
(1 row)
```

- substrb(string,from,count)

  描述：该函数和SUBSTR(string,from,count)函数功能一致，但是计算单位为字节。

  返回值类型：bytea

  示例：

```
gaussdb=# SELECT substrb('ABCDEF',2,2);
 substrb
---------
 BC
(1 row)
```

- to_single_byte(char)

  描述：将字符串中所有多字节字符转换为单字节字符。

  返回值类型：text

  说明

  此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s2的情况下有效。

- to_multi_byte(char)

  描述：将字符串中所有单字节字符转换为多字节字符。

  返回值类型：text

  说明

  此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s2的情况下有效。

- trim([leading |trailing |both] [characters] from string)

  描述：从字符串string的开头、结尾或两边删除只包含characters中字符（缺省是一个空白）的最长的字符串。

  返回值类型：text

  示例：

```
gaussdb=# SELECT trim(BOTH 'x' FROM 'xTomxx');
 btrim
-------
 Tom
(1 row)
gaussdb=# SELECT trim(LEADING 'x' FROM 'xTomxx');
 ltrim
-------
 Tomxx
(1 row)
gaussdb=# SELECT trim(TRAILING 'x' FROM 'xTomxx');
 rtrim
-------
 xTom
(1 row)
```

- rtrim(string [, characters])

  描述：从字符串string的结尾删除只包含characters中字符（缺省是个空白）的最长的字符串。

  返回值类型：text

示例：

```
gaussdb=# SELECT rtrim('TRIMxxxx','x');
 rtrim
-------
 TRIM
(1 row)
```

- ltrim(string [, characters])

  描述：从字符串string的开头删除只包含characters中字符（缺省是一个空白）的最长的字符串。

  返回值类型：text

  示例：

  ```
  gaussdb=# SELECT ltrim('xxxxTRIM','x');
   ltrim
  -------
   TRIM
  (1 row)
  ```

- upper(string)

  描述：把字符串转换为大写。

  返回值类型：text

  示例：

  ```
  gaussdb=# SELECT upper('tom');
   upper
  -------
   TOM
  (1 row)
  ```

- lower(string)

  描述：把字符串转换为小写。

  返回值类型：text

  示例：

  ```
  gaussdb=# SELECT lower('TOM');
   lower
  -------
   tom
  (1 row)
  ```

- nls_upper(string [, nlsparam])

  描述：把字符串转换为大写，可以指定排序规则来处理某些国家语言的特殊大写转换规则。nlsparam的格式为'nls_sort=sort_name'，其中sort_name替换为具体的排序规则名。当不输入nlsparam参数时，该函数完全等同于upper。

  返回值类型：text

  示例：

  ```
  gaussdb=# SELECT nls_upper('große');
   nls_upper
  -----------
   GROßE
  (1 row)
  gaussdb=# SELECT nls_upper('große', 'nls_sort = XGerman');
   nls_upper
  -----------
   GROSSE
  (1 row)
  ```

  说明

  此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s2的情况下才支持使用。

- nls_lower(string [, nlsparam])

  描述：把字符串转换为小写，可以指定排序规则来处理某些国家语言的特殊小写转换规则。nlsparam的格式为'nls_sort=sort_name'，其中sort_name替换为具体的排序规则名。当不输入nlsparam参数时，该函数完全等同于lower。

  返回值类型：text

  示例：

```
gaussdb=# SELECT nls_lower('INDIVISIBILITY');
   nls_lower
----------------
 indivisibility
(1 row)
gaussdb=# SELECT nls_lower('INDIVISIBILITY', 'nls_sort = XTurkish');
   nls_lower
----------------
 ındıvısıbılıty
(1 row)
```

  说明

  此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s2的情况下才支持使用。

- instr(string,substring[,position,occurrence])

  描述：从字符串string的position（缺省时为1）所指的位置开始查找并返回第occurrence（缺省时为1）次出现子串substring的位置的值。

  - 当position为0时，返回0。
  - 当position为负数时，从字符串倒数第n个字符往前逆向搜索。n为position的绝对值。

  本函数以字符为计算单位，如一个汉字为一个字符。

  返回值类型：integer

  示例：

```
gaussdb=# SELECT instr('corporate floor','or', 3);
 instr
-------
     5
(1 row)
gaussdb=# SELECT instr('corporate floor','or',-3,2);
 instr
-------
     2
(1 row)
```

- initcap(string)

  描述：将字符串中的每个单词的首字母转换为大写，其他字母转换为小写。

  返回值类型：text

  示例：

```
gaussdb=# SELECT initcap('hi THOMAS');
  initcap
-----------
 Hi Thomas
(1 row)
```

  说明

  此函数在A兼容模式数据库中且开启参数a_format_version值为10c和a_format_dev_version值为s2的情况下，如中文等无大小写区分的字符后接区分大小写的字符首字母会转换为大写，其他字母转换为小写。因此建议开启参数a_format_version值为10c和a_format_dev_version值为s2。

- ascii(string)

  描述：参数string的第一个字符的ASCII码。

  返回值类型：integer

  示例：

  ```
  gaussdb=# SELECT ascii('xyz');
   ascii
  -------
    120
  (1 row)
  ```

- ascii2(string)

  描述：参数string的第一个字符在数据库字符集中的十进制编码。

  返回值类型：bigint

  说明

  此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s2的情况下有效。

- asciistr(string)

  描述：把字符串string中非ASCII字符转换为\XXXX形式，其中XXXX表示UTF-16代码单元。

  返回值类型：text

  说明

  此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s2的情况下有效。

- unistr(string)

  描述：将字符串中的编码序列转换成对应字符，其他字符保持不变。

  返回值类型：text

  说明

  - 此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s2的情况下有效。
  - '\'后必须接4位16进制字符表示编码序列，或者接另一个'\'表示输入单个'\'字符。
  - 入参是时间类型时，时间类型会隐式转换成字符串类型。

- vsize(expr)

  描述：返回输入表达式的字节数。

  返回值类型：int

  说明

  此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s2的情况下有效。

- replace(string varchar, search_string varchar, replacement_string varchar)

  描述：把字符串string中所有子字符串search_string替换成子字符串replacement_string。

  返回值类型：text

  示例：

  ```
  gaussdb=# SELECT replace('jack and jue','j','bl');
     replace
  ```



```
----------------
 black and blue
(1 row)
```

- concat(str1,str2)

  描述：将字符串str1和str2连接并返回。注意，concat会调用data type的输出函数，所以是非immutable的，导致优化器在生成计划的时候不能提前计算结果。如果对性能有要求，建议用 || 替代。

  须知

  - 在sql_compatibility = 'B'的情况下，参数str1或str2为NULL会导致返回结果为NULL。
  - concat函数返回值类型为变长类型，和表中数据比较时，会因为拼接结果丢失字符串长度，导致比较结果不相等。

  返回值类型：text

  示例：

```
gaussdb=# SELECT concat('Hello', ' World!');
   concat
--------------
 Hello World!
(1 row)
gaussdb=# SELECT concat('Hello', NULL);
 concat
--------
 Hello
(1 row)
gaussdb=# CREATE TABLE test_space(c char(10));
CREATE TABLE
gaussdb=# CREATE TABLE test_space VALUES('a');
INSERT 0 1
-- 填充空格后仍然是定长字符串，预期可以查找到结果
gaussdb=# SELECT * FROM test_space WHERE c = 'a ';
     c
------------
 a
(1 row)
-- 拼接结果为变长字符串，比对失败，找不到结果
gaussdb=# SELECT * FROM test_space WHERE c = 'a' || ' ';
 c
---
(0 rows)
```

- chr(integer)

  描述：如果为UTF-8字符集，将输入作为unicode编码，返回一个UTF-8的字符，其他字符集给出ASCII码的字符。

  返回值类型：text

  说明

  此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s1的情况下：参数integer入参若为小数则不会被四舍五入，而是被截断。

  示例：

```
gaussdb=# SELECT chr(65);
 chr
-----
 A
(1 row)
```

```
-- UTF-8字符集的情况下(创建数据库时，已指定数据库字符集为UTF-8，不支持重新设置数据库字符集。)
gaussdb=# SELECT chr(19968);
 chr
-----
 一
(1 row)
```

- chr(cvalue int|bigint)

  描述：将cvalue转换为对应字节序列的字符返回。

  cvalue：cvalue支持类型是可以转换成int或bigint的类型，取值范围为[0, 2^32 - 1]，对应unsigned int的范围，根据输入n的大小返回由1-4个字节组成的字符数组。其中在不同字符集中所返回的字节数组是相同的，但由于编码规则的不同会造成返回字符串的结果依赖于字符集编码。

  当字符集为单字节编码的字符集时，会先将cvalue mod 256后返回一个ASCII码字符。

  注意事项：

  - 当输入的cvalue其中的某个字节为0的时候，输出会在该处截断。
  - 当输入不符合现字符集的编码规则时会报错。
  - 当输入为NULL或者0时返回NULL。

  返回值类型：text

  示例：

```
gaussdb=# SELECT chr(65);
 chr
-----
 A
(1 row)
gaussdb=# SET a_format_version='10c';
gaussdb=# SET a_format_dev_version = 's1';
gaussdb=# SELECT chr(16705);
 chr
-----
 AA
(1 row)

-- 输出被截断
gaussdb=# SELECT chr(4259905);
 chr
-----
 A
(1 row)
```

  说明

  此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s1的情况下，chr函数功能为根据输入大小返回对应字符序列：当前数据库编码字符集为多字节编码字符集时返回值为1-4个字节；当前数据库编码字符集为单字节编码字符集时返回值为将输入值通过mod 256运算后得到的单个字节。否则功能为：若当前数据库编码字符集为UTF-8字符集，则将输入作为unicode编码并返回一个UTF-8字符，若当前数据库编码字符集为其他字符集则返回ASCII码字符。

- regexp_substr(source_char, pattern)

  描述：正则表达式的抽取子串函数。SQL语法兼容A和B的情况下，GUC参数behavior_compat_options的值包含aformat_regexp_match时，. 不能匹配 '\n' 字符；不包含aformat_regexp_match时，. 能够匹配'\n'字符。

  返回值类型：text

  示例：


```
gaussdb=# SELECT regexp_substr('500 Hello World, Redwood Shores, CA', ',[^,]+,')
"REGEXPR_SUBSTR";
  REGEXPR_SUBSTR
-------------------
 , Redwood Shores,
(1 row)
```

- regexp_replace(string, pattern, replacement [,flags])

  描述：替换匹配POSIX正则表达式的子字符串。如果没有匹配pattern，那么返回不加修改的string串。如果有匹配，则返回的string串里面的匹配子串将被replacement串替换掉。

  replacement串可以包含\n，其中\n是1到9，表明string串里匹配模式里第n个圆括号子表达式的子串应该被插入，并且它可以包含\&表示应该插入匹配整个模式的子串。

  可选的flags参数包含零个或多个改变函数行为的单字母标记。flags 支持的选项值及含义描述如**表7-28**所示。

  返回值类型：text

  示例：

```
gaussdb=# SELECT regexp_replace('Thomas', '.[mN]a.', 'M');
 regexp_replace
----------------
 ThM
(1 row)
gaussdb=# SELECT regexp_replace('foobarbaz','b(..)', E'X\\1Y', 'g') AS
RESULT;
   result
-------------
 fooXarYXazY
(1 row)
```

- regexp_replace(string text, pattern text [, replacement text [, position int [, occurrence int [, flags text]]]])

  描述：替换匹配POSIX正则表达式的子字符串。 如果没有匹配pattern，那么返回不加修改的string串。 如果有匹配，则返回的string串里面的匹配子串将被replacement串替换掉。

  参数说明：

  - string：用于匹配的源字符串。
  - pattern：用于匹配的正则表达式模式串。
  - replacement：可选参数，用于替换匹配子串的字符串。如果不给定参数值或者为null，表示用空串替换。
  - position：可选参数，表示从源字符串的第几个字符开始匹配，默认值为1。
  - occurrence：可选参数，表示替换第occurrence个匹配的子串。默认值为1，表示替换匹配到的第一个子串。
  - flags：可选参数，包含零个或多个改变函数匹配行为的单字母标记。flags 支持的选项值及含义描述如**表7-28**所示。

  说明

  此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s1的情况下，occurrence参数默认值为0，表示替换所有匹配到的子串。并且以'\'字符结尾的pattern参数为合法的。

  返回值类型：text

  示例：

```
gaussdb=# SELECT regexp_replace('foobarbaz','b(..)', E'X\\1Y', 2, 2, 'n') AS RESULT;
result
```



```
------------
foobarXazY
(1 row)
```

- concat_ws(sep text, str"any" [, str"any" [, ...]])

  描述：以第一个参数为分隔符，连接第二个以后的所有参数。NULL参数被忽略。

  须知

  - 如果第一个参数值是NULL，会导致返回结果为NULL。
  - 如果第一个参数值是空字符串（''），且数据库SQL兼容模式设置为A的情况下，会导致返回结果为NULL。这是因为A兼容模式将''作为NULL处理，避免此类行为，可以将数据库SQL兼容模式改为B、C或者PG。

  返回值类型：text

  示例：

```
gaussdb=# SELECT concat_ws(',', 'ABCDE', 2, NULL, 22);
 concat_ws
------------
 ABCDE,2,22
(1 row)
```

- nlssort(string text, sort_method text)

  描述：以sort_method指定的排序方式返回字符串在该排序模式下的编码值，该编码值可用于排序，其决定了string在这种排序模式下的先后位置。目前支持的sort_method为'nls_sort=schinese_pinyin_m'和'nls_sort=generic_m_ci'。其中，'nls_sort=generic_m_ci'仅支持纯英文不区分大小写排序。

  string类型：text

  sort_method类型：text

  返回值类型：text

  示例：

```
gaussdb=# CREATE TABLE test(a text);
gaussdb=# INSERT INTO test(a) VALUES ('abC 不');
gaussdb=# INSERT INTO test(a) VALUES ('abC 啊');
gaussdb=# INSERT INTO test(a) VALUES ('abc 啊');
gaussdb=# SELECT * FROM test ORDER BY nlssort(a,'nls_sort=schinese_pinyin_m');
   a
--------
 abc 啊
 abC 啊
 abC 不
(3 rows)
gaussdb=# SELECT * FROM test ORDER BY nlssort(a, 'nls_sort=generic_m_ci');
   a
--------
 abC 啊
 abc 啊
 abC 不
(3 rows)
gaussdb=# DROP TABLE test;
```

- convert(string bytea, src_encoding name, dest_encoding name)

  描述：以dest_encoding指定的目标编码方式转换字符串string。src_encoding指定源编码方式，在该编码下，string必须是合法的。

  返回值类型：bytea

  示例：

```
gaussdb=# SELECT convert('text_in_utf8', 'UTF8', 'GBK');
          convert
----------------------------
 \x746578745f696e5f75746638
(1 row)
```

说明

如果源编码格式到目标编码格式的转换规则不存在，则字符串不进行任何转换直接返回，如GBK和LATIN1之间的转换规则是不存在的，具体转换规则可以通过查看系统表pg_conversion获得。

示例：

```
gaussdb=# SHOW server_encoding;
 server_encoding
-----------------
 LATIN1
(1 row)

gaussdb=# SELECT convert_from('some text', 'GBK');
 convert_from
--------------
 some text
(1 row)

db_latin1=# SELECT convert_to('some text', 'GBK');
      convert_to
----------------------
 \x736f6d652074657874
(1 row)

db_latin1=# SELECT convert('some text', 'GBK', 'LATIN1');
       convert
----------------------
 \x736f6d652074657874
(1 row)
```

- convert_from(string bytea, src_encoding name)

  描述：以数据库的编码方式转换字符串string。

  src_encoding指定源编码方式，在该编码下，string必须是合法的。

  返回值类型：text

  示例：

  ```
  gaussdb=# SELECT convert_from('text_in_utf8', 'UTF8');
   convert_from
  --------------
   text_in_utf8
  (1 row)
  ```

- convert_to(string text, dest_encoding name)

  描述：将字符串转换为dest_encoding的编码格式。

  返回值类型：bytea

  示例：

  ```
  gaussdb=# SELECT convert_to('some text', 'UTF8');
       convert_to
  ----------------------
   \x736f6d652074657874
  (1 row)
  ```

- string [NOT] LIKE pattern [ESCAPE escape-character]

  描述：模式匹配函数。

  如果pattern不包含百分号或者下划线，该模式只代表它本身，这时候LIKE的行为就像等号操作符。在pattern里的下划线（_）匹配任何单个字符；而一个百分号（%）匹配零或多个任何字符。

要匹配下划线或者百分号本身，在pattern里相应的字符必须前导逃逸字符。缺省的逃逸字符是反斜杠，但是用户可以用ESCAPE子句指定一个。要匹配逃逸字符本身，写两个逃逸字符。

返回值类型：Boolean

示例：

```
gaussdb=# SELECT 'AA_BBCC' LIKE '%A@_B%' ESCAPE '@' AS RESULT;
 result
--------
 t
(1 row)
gaussdb=# SELECT 'AA_BBCC' LIKE '%A@_B%' AS RESULT;
 result
--------
 f
(1 row)
gaussdb=# SELECT 'AA@_BBCC' LIKE '%A@_B%' AS RESULT;
 result
--------
 t
(1 row)
```

- REGEXP_LIKE(source_string, pattern [, match_parameter])

  描述：正则表达式的模式匹配函数。

  source_string为源字符串，pattern为正则表达式匹配模式。 match_parameter为匹配选项，可取值为：

  - 'i'：大小写不敏感。
  - 'c'：大小写敏感。
  - 'n'：允许正则表达式元字符“.”匹配换行符。
  - 'm'：将source_string视为多行。

  若忽略match_parameter选项，默认为大小写敏感，“.”不匹配换行符，source_string视为单行。

  返回值类型：Boolean

  示例：

  ```
  gaussdb=# SELECT regexp_like('ABC', '[A-Z]');
   regexp_like
  -------------
   t
  (1 row)
  gaussdb=# SELECT regexp_like('ABC', '[D-Z]');
   regexp_like
  -------------
   f
  (1 row)
  gaussdb=# SELECT regexp_like('ABC', '[a-z]','i');
   regexp_like
  -------------
   t
  (1 row)
  ```

- format(formatstr text [, str"any" [, ...]])

  描述：格式化字符串。

  返回值类型：text

  示例：

  ```
  gaussdb=# SELECT format('Hello %s, %1$s', 'World');
        format
  --------------------
   Hello World, World
  (1 row)
  ```


- md5(string)

  描述：将string使用MD5加密，并以16进制数作为返回值。

  说明

  MD5加密算法安全性低，存在安全风险，不建议使用。

  返回值类型：text

  示例：

```
gaussdb=# SELECT md5('ABC');
               md5
----------------------------------
 902fbdd2b1df0c4f70b4a5d23525e932
(1 row)
```

- sha(string) / sha1(string)

  描述：将string使用SHA1加密，并以16进制数作为返回值，sha和sha1函数功能相同。

  说明

  - SHA1加密算法安全性低，存在安全风险，不建议使用。
  - 该函数仅在GaussDB兼容MY类型时（即sql_compatibility = 'B'）有效，其他类型不支持该函数。

  返回值类型：text

  示例：

```
gaussdb=# SELECT sha('ABC');
                  sha
------------------------------------------
 3c01bdbb26f358bab27f267924aa2c9a03fcfdb8
(1 row)
gaussdb=# SELECT sha1('ABC');
                  sha1
------------------------------------------
 3c01bdbb26f358bab27f267924aa2c9a03fcfdb8
(1 row)
```

- sha2(string，hash_length)

  描述：将string使用SHA2加密，并以16进制数作为返回值。

  hash_length：对应相应的SHA2算法，可选值为 0(SHA-256)、224(SHA-224)、256(SHA-256)、384(SHA-384)、512(SHA-512)，其他值将返回NULL。

  说明

  - SHA224加密算法安全性低，存在安全风险，不建议使用。
  - SHA2函数会在日志中记录哈希的明文，因此不建议用户用该函数加密密钥等敏感信息。
  - 该函数仅在GaussDB兼容B类型时（即sql_compatibility = 'B'）有效，其他类型不支持该函数。

  返回值类型：text

  示例：

```
gaussdb=# SELECT sha2('ABC',224);
                           sha2
----------------------------------------------------------
 107c5072b799c4771f328304cfe1ebb375eb6ea7f35a3aa753836fad
(1 row)
gaussdb=# SELECT sha2('ABC',256);
                              sha2
-----------------------------------------------------------------
```



```
 b5d4045c3f466fa91fe2cc6abe79232a1a57cdf104f7a26e716e0a1e2789df78
(1 row)
gaussdb=# SELECT sha2('ABC',0);
                              sha2
----------------------------------------------------------------
 b5d4045c3f466fa91fe2cc6abe79232a1a57cdf104f7a26e716e0a1e2789df78
(1 row)
```

- decode(string text, format text)

  描述：将二进制数据从文本数据中解码。

  返回值类型：bytea

  示例：

```
gaussdb=# SELECT decode('MTIzAAE=', 'base64');
    decode
--------------
 \x3132330001
(1 row)
```

- similar_escape(pat text, esc text)

  描述：将一个 SQL:2008风格的正则表达式转换为POSIX风格。

  返回值类型：text

  示例：

```
gaussdb=# SELECT similar_escape('\s+ab','2');
 similar_escape
----------------
 ^(?:\\s+ab)$
(1 row)
```

- find_in_set(text, set)

  描述：查找给定成员在集合中的位置，从1开始计数。如果没有找到，返回0。

  返回值类型：int2

  示例：

```
gaussdb=# CREATE DATABASE gaussdb_b WITH DBCOMPATIBILITY 'B';
gaussdb=# \c gaussdb_b
gaussdb_b=# CREATE TABLE employee (
  name text,
  site SET('beijing','shanghai','nanjing','wuhan')
);

gaussdb_b=# INSERT INTO employee values('zhangsan', 'beijing,nanjing');
gaussdb_b=# INSERT INTO employee values('zhangsan2', 'beijing,wuhan');

gaussdb_b=# select site, find_in_set('wuhan', site) from employee;
      site       | find_in_set
-----------------+-------------
 beijing,nanjing |           0
 beijing,wuhan   |           2
(2 rows)

gaussdb_b=# DROP TABLE employee;
gaussdb_b=# \c postgres
gaussdb=# DROP DATABASE gaussdb_b;
```

- encode(data bytea, format text)

  描述：将二进制数据编码为文本数据。

  返回值类型：text

  示例：

```
gaussdb=# SELECT encode(E'123\\000\\001', 'base64');
  encode
----------
```



```
 MTIzAAE=
(1 row)
```

说明

- 若字符串中存在换行符，如字符串由一个换行符和一个空格组成，在GaussDB中LENGTH和LENGTHB的值为2。
- 对于CHAR(n) 类型，GaussDB中n是指字符个数。因此，对于多字节编码的字符集，LENGTHB函数返回的长度可能大于n。
- GaussDB支持多种类型的数据库，目前有4种，分别是A类型，B类型，C类型以及PG类型。在不指定数据库类型时，GaussDB默认是A类型，A的词法分析器与另外三种不一样，在A中空字符串会被当作是NULL。所以，当使用A类型的数据库时，假如上述字符操作函数中有空字符串作为参数，会出现没有输出的情况。例如：

  ```
  gaussdb=# SELECT translate('12345','123','');
  translate
  -----------

  (1 row)
  ```

  这是因为内核在调用相应的函数进行处理前，会判断所输入的参数中是否含有NULL，假如有，则不会调用相应的函数，因此会没有输出。而在PG模式下，字符串的处理方式与postgresql保持一致，因此不会有上述问题产生。

扩展函数和操作符

- pkg_bpchar_opc()

  描述：扩展接口，用于新增bpchar和text或者text和bpchar策略比较操作符，为解决bpchar类型和text类型数据比较，无法命中索引问题。仅系统管理员可以安装扩展。

  须知

  扩展功能为内部使用功能，不建议用户使用。

  示例：

  logs_nchar表、logs_char表、logs_varchar2表和logs_text表如下：

  ```
  /*
  logs_nchar表
  */

  DROP TABLE IF EXISTS logs_nchar;
  CREATE TABLE logs_nchar
  (
     log_id      NCHAR(16),
     log_message VARCHAR2(255) NOT NULL
  );
  DROP INDEX IF EXISTS idx_nchar_logid;
  CREATE INDEX idx_nchar_logid ON logs_nchar(log_id);
  INSERT INTO logs_nchar VALUES('FE306991300002  ', '111');
  INSERT INTO logs_nchar VALUES('FE306991300003  ', '222');
  INSERT INTO logs_nchar VALUES('FE306991300004  ', '222');

  /*
  logs_char表
  */
  DROP TABLE IF EXISTS logs_char;
  CREATE TABLE logs_char
  (
     log_id      CHAR(16),
     log_message VARCHAR2(255) NOT NULL
  );
  ```

```
DROP INDEX IF EXISTS idx_char_logid;
CREATE INDEX idx_char_logid ON logs_char(log_id);
INSERT INTO logs_char VALUES('FE306991300002  ', '111');
INSERT INTO logs_char VALUES('FE306991300003  ', '222');
INSERT INTO logs_char VALUES('FE306991300004  ', '222');

/*
logs_varchar2表
*/
DROP TABLE IF EXISTS logs_varchar2;
CREATE TABLE logs_varchar2
(
   log_id      VARCHAR2(16),
   log_message VARCHAR2(255) NOT NULL
);
DROP INDEX IF EXISTS idx_varchar2_logid;
CREATE INDEX idx_varchar2_logid ON logs_varchar2(log_id);
INSERT INTO logs_varchar2 VALUES('FE306991300002  ', '111');
INSERT INTO logs_varchar2 VALUES('FE306991300003  ', '222');
INSERT INTO logs_varchar2 VALUES('FE306991300004  ', '222');

/*
logs_text表
*/

DROP TABLE IF EXISTS logs_text;
CREATE TABLE logs_text
(
   log_id      text,
   log_message VARCHAR2(255) NOT NULL
);
DROP INDEX IF EXISTS idx_text_logid;
CREATE INDEX idx_text_logid ON logs_text(log_id);
INSERT INTO logs_text VALUES('FE306991300002  ', '111');
INSERT INTO logs_text VALUES('FE306991300003  ', '222');
INSERT INTO logs_text VALUES('FE306991300004  ', '222');
```

bpchar类型和text类型比较时（初始状态，前向兼容）：

```
/*
没有安装扩展时候，nchar和text比较时候，由于没有bpchar和text索引操作符，nchar会隐式转换为text,即定长字符类型转换成变长字符类型，导致执行计划发生了变化，不能命中索引。
*/
gaussdb=# EXPLAIN SELECT * FROM logs_nchar WHERE log_id = RPAD(TRIM('FE306991300002  '),16,'
');
                          QUERY PLAN
---------------------------------------------------------------
 Seq Scan on logs_nchar  (cost=0.00..11.99 rows=1 width=584)
   Filter: ((log_id)::text = 'FE306991300002  '::text)
(2 rows)
/*
表logs_nchar里log_id字段类型是nchar(16)，插入数据为'FE306991300002  '，隐式转换成text，进行比较时，会把后面空格去掉进行比较，即'FE306991300002'='FE306991300002  '，所以不命中数据。
*/
gaussdb=# SELECT * FROM logs_nchar WHERE log_id = RPAD(TRIM('FE306991300002  '),16,' ');
 log_id | log_message
--------+-------------
(0 rows)
```

bpchar类型和text类型比较时（安装pkg_bpchar_opc扩展，与A保持一致）：

```
/*
系统管理员安装pkg_bpchar_opc扩展，数据库增加了bpchar和text类型比较操作符，以及索引相关内容。
*/
gaussdb=# CREATE EXTENSION pkg_bpchar_opc;
CREATE EXTENSION
/*
此时，log_id隐式转换为bpchar类型时，和text类型比较时，能找到比较操作符以及索引信息，能命中索引。
*/
gaussdb=# EXPLAIN SELECT * FROM logs_nchar WHERE log_id = RPAD(TRIM('FE306991300002  '),16,'
');
```



```
                                QUERY PLAN
-------------------------------------------------------------------------------------
 [Bypass]
 Index Scan using idx_nchar_logid on logs_nchar  (cost=0.00..8.27 rows=1 width=584)
   Index Cond: (log_id = 'FE306991300002  '::text)
(3 rows)
gaussdb=# SELECT * FROM logs_nchar WHERE log_id = RPAD(TRIM('FE306991300002  '),16,' ');
      log_id      | log_message
------------------+-------------
 FE306991300002   | 111
(1 row)
gaussdb=# DROP EXTENSION pkg_bpchar_opc;
DROP EXTENSION
```

text类型和bpchar类型比较时（初始状态，前向兼容）：

```
gaussdb=# SELECT * FROM logs_text WHERE log_id = 'FE306991300002  '::bpchar;
 log_id | log_message
--------+-------------
(0 rows)
gaussdb=# EXPLAIN SELECT * FROM logs_text WHERE log_id = 'FE306991300002  '::bpchar;
                                QUERY PLAN
-------------------------------------------------------------------------------------
 [Bypass]
 Index Scan using idx_text1_logid on logs_text  (cost=0.00..8.27 rows=1 width=548)
   Index Cond: (log_id = 'FE306991300002'::text)
(3 rows)
```

text类型和bpchar类型比较时（安装pkg_bpchar_opc扩展，与A保持一致）：

```
gaussdb=# CREATE EXTENSION pkg_bpchar_opc;
CREATE EXTENSION
gaussdb=# SELECT * FROM logs_text WHERE log_id = 'FE306991300002  '::bpchar;
      log_id      | log_message
------------------+-------------
 FE306991300002   | 111
(1 row)
gaussdb=# EXPLAIN SELECT * FROM logs_text WHERE log_id = 'FE306991300002  '::bpchar;
                                QUERY PLAN

-------------------------------------------------------------------
-------------------
 [Bypass]
 Index Scan using idx_text_logid on logs_text  (cost=0.00..8.27
 rows=1 width=548)
   Index Cond: (log_id = 'FE306991300002  '::bpchar)
(3 rows)
gaussdb=# DROP EXTENSION pkg_bpchar_opc;
DROP EXTENSION
```

text类型和bpchar类型比较时（初始状态，前向兼容）：

```
gaussdb=# SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id = t2.log_id;
      log_id      | log_message |      log_id      | log_message
------------------+-------------+------------------+-------------
 FE306991300002   | 111         | FE306991300002   | 111
 FE306991300003   | 222         | FE306991300003   | 222
 FE306991300004   | 222         | FE306991300004   | 222
(3 rows)
gaussdb=# EXPLAIN SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id = t2.log_id;
                              QUERY PLAN
-------------------------------------------------------------------------------
 Hash Join  (cost=12.99..26.15 rows=133 width=1150)
   Hash Cond: ((t1.log_id)::bpchar = t2.log_id)
   ->  Seq Scan on logs_varchar2 t1  (cost=0.00..11.37 rows=137 width=566)
   ->  Hash  (cost=11.33..11.33 rows=133 width=584)
         ->  Seq Scan on logs_char t2  (cost=0.00..11.33 rows=133 width=584)
(5 rows)
gaussdb=# SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id = 'FE306991300002  ';
      log_id      | log_message |      log_id      | log_message
------------------+-------------+------------------+-------------
 FE306991300002   | 111         | FE306991300002   | 111
 FE306991300002   | 111         | FE306991300003   | 222
```



```
 FE306991300002   | 111         | FE306991300004   | 222
(3 rows)
```

text类型和bpchar类型比较时（安装pkg_bpchar_opc扩展，与A保持一致）：

```
gaussdb=# CREATE EXTENSION pkg_bpchar_opc;
CREATE EXTENSION
/*
不推荐写法，由于t1表的log_id的varchar2类型，受安装扩展影响，隐式转换为text类型，和t2表的log_id比较，t2表的log_id类型会由char隐式转换为bpchar类型，此时log_id后面的空格会被数据库去掉，即'FE306991300002'='FE306991300002  '，所以不命中数据。
*/
/*
错误示例：
*/
gaussdb=# SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id = t2.log_id;
 log_id | log_message | log_id | log_message
--------+-------------+--------+-------------
(0 rows)
gaussdb=# explain SELECT * FROM logs_varchar2 t1, logs_char t2
WHERE t1.log_id = t2.log_id;
                              QUERY PLAN

------------------------------------------------------------------
--------------
 Hash Join  (cost=12.99..26.15 rows=133 width=1150)
   Hash Cond: ((t1.log_id)::text = t2.log_id)
   ->  Seq Scan on logs_varchar2 t1  (cost=0.00..11.37 rows=137
 width=566)
   ->  Hash  (cost=11.33..11.33 rows=133 width=584)
         ->  Seq Scan on logs_char t2  (cost=0.00..11.33 rows=1
33 width=584)
(5 rows)
gaussdb=# SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id = 'FE306991300002  ';
      log_id      | log_message |      log_id      | log_message
------------------+-------------+------------------+-------------
 FE306991300002   | 111         | FE306991300002   | 111
 FE306991300002   | 111         | FE306991300003   | 222
 FE306991300002   | 111         | FE306991300004   | 222
(3 rows)
/*
推荐写法，避免t1表的log_id的数据类型转换成text类型，比较时空格被保留，和t2表的log_id比较无法命中数据，将t1表类型强转成没安装扩展前的bpchar类型，即'FE306991300002'='FE306991300002'，所以匹配数据。
*/
/*
正确示例：
*/
gaussdb=# SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id::bpchar = t2.log_id;
      log_id      | log_message |      log_id      | log_message
------------------+-------------+------------------+-------------
 FE306991300002   | 111         | FE306991300002   | 111
 FE306991300003   | 222         | FE306991300003   | 222
 FE306991300004   | 222         | FE306991300004   | 222
(3 rows)
/*
执行计划和没安装扩展前是一致的。
*/
gaussdb=# EXPLAIN SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id::bpchar =
t2.log_id;
                              QUERY PLAN
------------------------------------------------------------------------------
 Hash Join  (cost=12.99..26.15 rows=133 width=1150)
   Hash Cond: ((t1.log_id)::bpchar = t2.log_id)
   ->  Seq Scan on logs_varchar2 t1  (cost=0.00..11.37 rows=137 width=566)
   ->  Hash  (cost=11.33..11.33 rows=133 width=584)
         ->  Seq Scan on logs_char t2  (cost=0.00..11.33 rows=133 width=584)
(5 rows)
gaussdb=# SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id = 'FE306991300002  ';
      log_id      | log_message |      log_id      | log_message
------------------+-------------+------------------+-------------
```



```
 FE306991300002   | 111        | FE306991300002   | 111
 FE306991300002   | 111        | FE306991300003   | 222
 FE306991300002   | 111        | FE306991300004   | 222
(3 rows)
gaussdb=# DROP EXTENSION pkg_bpchar_opc;
DROP EXTENSION

gaussdb=# DROP TABLE logs_nchar;
gaussdb=# DROP TABLE logs_char;
gaussdb=# DROP TABLE logs_varchar2;
gaussdb=# DROP TABLE logs_text;
```

说明

- 为了解决bpchar类型（包含多个后补空格）与text类型做等值匹配操作的时候无法正常匹配数据以及索引问题。
- 涉及ubtree、btree，比较符号包含： >, >=, <, <=, <>。
- 影响面涉及字符类型之间的隐式转换，例如：变长和定长数据类型比较时，变长会优先转换为text类型，而不是最初的bpchar类型。
- 默认不开启pkg_bpchar_opc扩展。检验扩展有没有开启，可以查看系统表pg_extension，有该扩展数据是开启了，没有则是没有开启。关闭扩展时，保持了前向兼容，开启扩展时，保持了与A数据库兼容。扩展功能为内部使用功能，不建议用户使用。

表 7-29 pkg_bpchar_opc 支持的函数说明

函数名称	描述
pg_catalog.bpchar_text_lt	bpchar类型和text类型比较，左边数值是否小于右边的数值。
pg_catalog.bpchar_text_le	bpchar类型和text类型比较，左边数值是否小于等于右边的数值。
pg_catalog.bpchar_text_eq	bpchar类型和text类型比较，左边数值是否等于右边的数值。
pg_catalog.bpchar_text_ge	bpchar类型和text类型比较，左边数值是否大于等于右边的数值。
pg_catalog.bpchar_text_gt	bpchar类型和text类型比较，左边数值是否大于右边的数值。
pg_catalog.bpchar_text_ne	bpchar类型和text类型比较，左边数值是否不等于右边的数值。
pg_catalog.bpchar_text_cmp	bpchar类型和text类型的索引支持比较函数。
pg_catalog.text_bpchar_lt	text类型和bpchar类型比较，左边数值是否小于右边的数值。
pg_catalog.text_bpchar_le	text类型和bpchar类型比较，左边数值是否小于等于右边的数值。
pg_catalog.text_bpchar_eq	text类型和bpchar类型比较，左边数值是否等于右边的数值。

函数名称	描述
pg_catalog.text_bpchar_ge	text类型和bpchar类型比较，左边数值是否大于等于右边的数值。
pg_catalog.text_bpchar_gt	text类型和bpchar类型比较，左边数值是否大于右边的数值。
pg_catalog.text_bpchar_ne	text类型和bpchar类型比较，左边数值是否不等于右边的数值。
pg_catalog.text_bpchar_cmp	text类型和bpchar类型的索引支持比较函数。
pg_catalog.hashbpchartext	bpchar类型和text类型的hash支持比较函数。
pg_catalog.hashtextbpchar	text类型和bpchar类型的hash支持比较函数。

7.5.4 二进制字符串函数和操作符

字符串操作符

SQL定义了一些字符串函数，在这些函数里使用关键字而不是逗号来分隔参数。

- octet_length(string)
  描述：二进制字符串中的字节数。
  返回值类型：int
  示例：

```
gaussdb=# SELECT octet_length(E'jo\\000se'::bytea) AS RESULT;
 result
--------
      5
(1 row)
```

- overlay(string placing string from int [for int])
  描述：替换子串。
  返回值类型：bytea
  示例：

```
gaussdb=# SELECT overlay(E'Th\\000omas'::bytea placing E'\\002\\003'::bytea from 2 for 3) AS
RESULT;
     result
----------------
 \x5402036d6173
(1 row)
```

- position(substring in string)
  描述：特定子字符串的位置。
  返回值类型：int
  示例：

```
gaussdb=# SELECT position(E'\\000om'::bytea in E'Th\\000omas'::bytea) AS RESULT;
 result
--------
```



```
     3
(1 row)
```

- substring(string [from int] [for int])

  描述：截取子串。

  返回值类型：bytea

  示例：

```
gaussdb=# SELECT substring(E'Th\\000omas'::bytea from 2 for 3) AS RESULT;
 result
----------
 \x68006f
(1 row)
```

- substr(string, from int [, for int])

  描述：截取子串。

  返回值类型：bytea

  示例：

```
gaussdb=# SELECT substr(E'Th\\000omas'::bytea,2, 3) as result;
 result
----------
 \x68006f
(1 row)
```

- trim([both] bytes from string)

  描述：从string的开头和结尾删除只包含bytes中字节的最长字符串。

  返回值类型：bytea

  示例：

```
gaussdb=# SELECT trim(E'\\000'::bytea from E'\\000Tom\\000'::bytea) AS RESULT;
 result
----------
 \x546f6d
(1 row)
```

二进制字符串函数

GaussDB也提供了函数调用所使用的常用语法。

- btrim(string bytea,bytes bytea)

  描述：从string的开头和结尾删除只包含bytes中字节的最长的字符串。

  返回值类型：bytea

  示例：

```
gaussdb=# SELECT btrim(E'\\000trim\\000'::bytea, E'\\000'::bytea) AS RESULT;
   result
------------
 \x7472696d
(1 row)
```

- get_bit(string, offset)

  描述：从字符串中抽取位。

  返回值类型：int

  示例：

```
gaussdb=# SELECT get_bit(E'Th\\000omas'::bytea, 45) AS RESULT;
 result
--------
      1
(1 row)
```


- get_byte(string, offset)
  描述：从字符串中抽取字节。
  返回值类型：int
  示例：

```
gaussdb=# SELECT get_byte(E'Th\\000omas'::bytea, 4) AS RESULT;
 result
--------
    109
(1 row)
```

- set_bit(string,offset, newvalue)
  描述：设置字符串中的位。
  返回值类型：bytea
  示例：

```
gaussdb=# SELECT set_bit(E'Th\\000omas'::bytea, 45, 0) AS RESULT;
      result
------------------
 \x5468006f6d4173
(1 row)
```

- set_byte(string,offset, newvalue)
  描述：设置字符串中的字节。
  返回值类型：bytea
  示例：

```
gaussdb=# SELECT set_byte(E'Th\\000omas'::bytea, 4, 64) AS RESULT;
      result
------------------
 \x5468006f406173
(1 row)
```

- rawcmp
  描述：raw数据类型比较函数。
  参数：raw, raw
  返回值类型：integer
- raweq
  描述：raw数据类型比较函数。
  参数：raw, raw
  返回值类型：boolean
- rawge
  描述：raw数据类型比较函数。
  参数：raw, raw
  返回值类型：boolean
- rawgt
  描述：raw数据类型比较函数。
  参数：raw, raw
  返回值类型：boolean
- rawin
  描述：raw数据类型解析函数。
  参数：cstring

返回值类型：bytea

- rawle
  描述：raw数据类型解析函数。
  参数：raw, raw
  返回值类型：boolean
- rawlike
  描述：raw数据类型解析函数。
  参数：raw, raw
  返回值类型：boolean
- rawlt
  描述：raw数据类型解析函数。
  参数：raw, raw
  返回值类型：boolean
- rawne
  描述：比较raw类型是否一样。
  参数：raw, raw
  返回值类型：boolean
- rawnlike
  描述：比较raw类型与模式是否不匹配。
  参数：raw, raw
  返回值类型：boolean
- rawout
  描述：RAW类型的输出接口。
  参数：bytea
  返回值类型：cstring
- rawsend
  描述：转换bytea为二进制类型。
  参数：raw
  返回值类型：bytea
- rawtohex
  描述：raw格式转换为十六进制。
  参数：text
  返回值类型：text

7.5.5 位串函数和操作符

位串操作符

除了常用的比较操作符之外，还可以使用以下的操作符。&，|和#的位串操作数必须等长。在位移的时候，保留原始的位串长度（并以0填充）。

- ||
  描述：位串之间进行连接。

示例：

```
gaussdb=# SELECT B'10001' || B'011' AS RESULT;
 result
----------
 10001011
(1 row)
```

说明

- 单字段内部连续连接操作不建议超过180次。如果超过180次，需拆分为多个连续连接的字符串，在它们之间再执行连接操作。

  例如：str1||str2||str3||str4 拆分为 (str1||str2)||(str3||str4)。
- A兼容模式下位串中包含了空字符串，会忽略空字符串连接其他字符串，其他兼容模式下则会返回空串。

  例如：str1||NULL||str2，A兼容模式下返回str1str2，其他兼容模式下返回NULL。

- &

  描述：位串之间进行“与”操作。

  示例：

  ```
  gaussdb=# SELECT B'10001' & B'01101' AS RESULT;
   result
  --------
   00001
  (1 row)
  ```

- |

  描述：位串之间进行“或”操作。

  示例：

  ```
  gaussdb=# SELECT B'10001' | B'01101' AS RESULT;
   result
  --------
   11101
  (1 row)
  ```

- #

  描述：位串之间如果不一致进行“或”操作。如果两个位串中对应位置都为1或者0，则该位置返回为0。

  示例：

  ```
  gaussdb=# SELECT B'10001' # B'01101' AS RESULT;
   result
  --------
   11100
  (1 row)
  ```

- ~

  描述：位串之间进行“非”操作。

  示例：

  ```
  gaussdb=# SELECT ~B'10001'AS RESULT;
   result
  ----------
   01110
  (1 row)
  ```

- <<

  描述：位串进行左移操作。

  示例：

  ```
  gaussdb=# SELECT B'10001' << 3 AS RESULT;
   result
  ----------
  ```



```
 01000
(1 row)
```

- >>

  描述：位串进行右移操作。

  示例：

```
gaussdb=# SELECT B'10001' >> 2 AS RESULT;
 result
----------
 00100
(1 row)
```

下面的SQL标准函数除了可以用于字符串之外，也可以用于位串：length，bit_length，octet_length，position，substring，overlay。

下面的函数用于位串和二进制字符串：get_bit，set_bit。当用于位串时，这些函数位数从字符串的第一位（最左边）作为0位。

另外，可以在整数和bit之间来回转换。示例：

```
gaussdb=# SELECT 44::bit(10) AS RESULT;
   result
------------
 0000101100
(1 row)

gaussdb=# SELECT 44::bit(3) AS RESULT;
 result
--------
 100
(1 row)

gaussdb=# SELECT cast(-44 as bit(12)) AS RESULT;
    result
--------------
 111111010100
(1 row)

gaussdb=# SELECT '1110'::bit(4)::integer AS RESULT;
 result
--------
     14
(1 row)

gaussdb=# SELECT substring('10101111'::bit(8), 2);
 substring
-----------
 0101111
(1 row)
```

说明

只是转换为“bit”的意思是转换成bit(1)，因此只会转换成整数的最低位。

7.5.6 模式匹配操作符

数据库提供了三种独立的实现模式匹配的方法：SQL LIKE操作符、SIMILAR TO操作符和POSIX-风格的正则表达式。除了这些基本的操作符外，还有一些函数可用于提取或替换匹配子串并在匹配位置分离一个串。

- LIKE

  描述：判断字符串是否能匹配上LIKE后的模式字符串。如果字符串与提供的模式匹配，则LIKE表达式返回为真（NOT LIKE表达式返回假），否则返回为假（NOT LIKE表达式返回真）。

匹配规则：

a. 此操作符只有在它的模式匹配整个串的时候才能成功。如果要匹配在串内任何位置的序列，该模式必须以百分号开头和结尾。

b. 下划线（_）代表（匹配）任何单个字符； 百分号（%）代表任意串的通配符。

c. 要匹配文本里的下划线或者百分号，在提供的模式里相应字符必须前导逃逸字符。逃逸字符的作用是禁用元字符的特殊含义，缺省的逃逸字符是反斜线，也可以用ESCAPE子句指定一个不同的逃逸字符。

d. 要匹配逃逸字符本身，写两个逃逸字符。例如要写一个包含反斜线的模式常量，需要在SQL语句里写两个反斜线。

说明

参数standard_conforming_strings设置为off时，在文串常量中写的任何反斜线都需要被双写。因此，写一个匹配单个反斜线的模式实际上要在语句里写四个反斜线（可以通过用ESCAPE选择一个不同的逃逸字符来避免这种情况，这样反斜线就不再是LIKE的特殊字符了。但仍然是字符文本分析器的特殊字符，所以需要两个反斜线）。

在兼容MYSQL数据模式时，可以通过写ESCAPE ''的方式不选择逃逸字符，这样可以有效地禁用逃逸机制，但是没有办法关闭下划线和百分号在模式中的特殊含义。

e. 关键字ILIKE可以用于替换LIKE，区别是LIKE大小写敏感，ILIKE大小写不敏感。

f. 操作符~~等效于LIKE，操作符~~*等效于ILIKE。

示例：

```
gaussdb=# SELECT 'abc' LIKE 'abc' AS RESULT;
 result
------------
 t
(1 row)
gaussdb=# SELECT 'abc' LIKE 'a%' AS RESULT;
 result
------------
 t
(1 row)
gaussdb=# SELECT 'abc' LIKE '_b_' AS RESULT;
 result
------------
 t
(1 row)
gaussdb=# SELECT 'abc' LIKE 'c' AS RESULT;
 result
------------
 f
(1 row)
```

- SIMILAR TO

  描述：SIMILAR TO操作符根据自己的模式是否匹配给定串而返回真或者假。他和LIKE非常类似，只不过他使用SQL标准定义的正则表达式理解模式。

  匹配规则：

  a. 和LIKE一样，此操作符只有在它的模式匹配整个串的时候才能成功。如果要匹配在串内任何位置的序列，该模式必须以百分号开头和结尾。

  b. 下划线（_）代表（匹配）任何单个字符； 百分号（%）代表任意串的通配符。

  c. SIMILAR TO也支持下面这些从POSIX正则表达式借用的模式匹配元字符。

元字符	含义
|	表示选择（两个候选之一）
*	表示重复前面的项零次或更多次
+	表示重复前面的项一次或更多次
?	表示重复前面的项零次或一次
{m}	表示重复前面的项刚好m次
{m,}	表示重复前面的项m次或更多次
{m,n}	表示重复前面的项至少m次并且不超过n次
()	把多个项组合成一个逻辑项
[...]	声明一个字符类，就像POSIX正则表达式一样

d. 前导逃逸字符可以禁止所有这些元字符的特殊含义。逃逸字符的使用规则和LIKE一样。

正则表达式函数：

支持使用函数**substring(string from pattern for escape)**截取匹配SQL正则表达式的子字符串。

示例：

```
gaussdb=# SELECT 'abc' SIMILAR TO 'abc' AS RESULT;
 result
------------
 t
(1 row)
gaussdb=# SELECT 'abc' SIMILAR TO 'a' AS RESULT;
 result
------------
 f
(1 row)
gaussdb=# SELECT 'abc' SIMILAR TO '%(b|d)%' AS RESULT;
 result
------------
 t
(1 row)
gaussdb=# SELECT 'abc' SIMILAR TO '(b|c)%'  AS RESULT;
 result
------------
 f
(1 row)
```

- POSIX正则表达式

  描述：正则表达式是一个字符序列，定义一个串集合（一个正则集）的缩写。 如果一个串是正则表达式描述的正则集中的一员时， 则说明这个串匹配该正则表达式。 POSIX正则表达式提供了比LIKE和SIMILAR TO操作符更强大的含义。**表7-30**列出了所有可用于POSIX正则表达式模式匹配的操作符。

表 7-30 正则表达式匹配操作符

操作符	描述	例子
~	匹配正则表达式，大小写敏感	'thomas' ~ '.*thomas.*'
~*	匹配正则表达式，大小写不敏感	'thomas' ~* '.*Thomas.*'
!~	不匹配正则表达式，大小写敏感	'thomas' !~ '.*Thomas.*'
!~*	不匹配正则表达式，大小写不敏感	'thomas' !~* '.*vadim.*'

匹配规则：

a. 与LIKE不同，正则表达式允许匹配串里的任何位置，除非该正则表达式显式地挂接在串的开头或者结尾。

b. 除了上文提到的元字符外， POSIX正则表达式还支持下列模式匹配元字符。

元字符	含义
^	表示串开头的匹配
$	表示串末尾的匹配
.	匹配任意单个字符

正则表达式函数：

POSIX正则表达式支持下面函数。

- **substring(string from pattern)**函数提供了抽取一个匹配POSIX正则表达式模式的子串的方法。
- **regexp_count(string text, pattern text [, position int [, flags text]])**函数提供了获取匹配POSIX正则表达式模式的子串数量的功能。
- **regexp_instr(string text, pattern text [, position int [, occurrence int [, return_opt int [, flags text]]]])**函数提供了获取匹配POSIX正则表达式模式子串位置的功能。
- **regexp_substr(string text, pattern text [, position int [, occurrence int [, flags text]]])**函数提供了抽取一个匹配POSIX正则表达式模式的子串的方法。
- **regexp_replace(string, pattern, replacement [,flags])**函数提供了将匹配POSIX正则表达式模式的子串替换为新文本的功能。
- **regexp_matches(string text, pattern text [, flags text])**函数返回一个文本数组，该数组由匹配一个POSIX正则表达式模式得到的所有被捕获子串构成。
- **regexp_split_to_table(string text, pattern text [, flags text])**函数把一个POSIX正则表达式模式当作一个定界符来分离一个串。

- **regexp_split_to_array(string text, pattern text [, flags text])**和regexp_split_to_table类似，是一个正则表达式分离函数，不过它的结果以一个text数组的形式返回。

  说明

  正则表达式分离函数会忽略零长度的匹配，这种匹配发生在串的开头或结尾或者正好发生在前一个匹配之后。这和正则表达式匹配的严格定义是相悖的，后者由regexp_matches实现，但是通常前者是实际中最常用的行为。

示例：

```
gaussdb=#  SELECT 'abc' ~ 'Abc' AS RESULT;
result
--------
 f
(1 row)
gaussdb=# SELECT 'abc' ~* 'Abc' AS RESULT;
 result
--------
 t
(1 row)
gaussdb=# SELECT 'abc' !~ 'Abc' AS RESULT;
 result
--------
 t
(1 row)
gaussdb=# SELECT 'abc'!~* 'Abc' AS RESULT;
 result
--------
 f
(1 row)
gaussdb=# SELECT 'abc' ~ '^a' AS RESULT;
 result
--------
 t
(1 row)
gaussdb=# SELECT 'abc' ~ '(b|d)'AS RESULT;
 result
--------
 t
(1 row)
gaussdb=# SELECT 'abc' ~ '^(b|c)'AS RESULT;
 result
--------
 f
(1 row)
```

虽然大部分的正则表达式搜索都能很快地执行，但是仍可能被人为地处理成需要任意长的时间和任意量的内存。不建议从非安全模式来源接受正则表达式搜索模式，如果必须这样做，建议加上语句超时限制。使用SIMILAR TO模式的搜索具有同样的安全性危险， 因为SIMILAR TO提供了很多和POSIX-风格正则表达式相同的能力。LIKE搜索比其他两种选项简单得多，因此在接受非安全模式来源搜索时要更安全些。

7.5.7 数字操作函数和操作符

数字操作符

- +

  描述：加。

  示例：

```
gaussdb=# SELECT 2+3 AS RESULT;
 result
```



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

- -

  描述：减。

  示例：

  ```
  gaussdb=# SELECT 2-3 AS RESULT;
   result
  --------
     -1
  (1 row)
  ```

- *

  描述：乘。

  示例：

  ```
  gaussdb=# SELECT 2*3 AS RESULT;
   result
  --------
     6
  (1 row)
  ```

- /

  描述：除（除法操作符不会取整）。

  示例：

  ```
  gaussdb=# SELECT 4/2 AS RESULT;
   result
  --------
     2
  (1 row)
  gaussdb=# SELECT 4/3 AS RESULT;
       result
  ------------------
   1.33333333333333
  (1 row)
  ```

- +/-

  描述：正/负。

  示例：

  ```
  gaussdb=# SELECT -2 AS RESULT;
   result
  --------
     -2
  (1 row)
  ```

- %

  描述：模（求余）。

  示例：

  ```
  gaussdb=# SELECT 5%4 AS RESULT;
   result
  --------
     1
  (1 row)
  ```

- @

  描述：绝对值。

  示例：

  ```
  gaussdb=# SELECT @ -5.0 AS RESULT;
   result
  --------
  ```



```
   5.0
(1 row)
```

- ^

  描述：幂（指数运算）。

  示例：

  ```
  gaussdb=# SELECT 2.0^3.0 AS RESULT;
        result
  --------------------
   8.0000000000000000
  (1 row)
  ```

- |/

  描述：平方根。

  示例：

  ```
  gaussdb=# SELECT |/ 25.0 AS RESULT;
   result
  --------
       5
  (1 row)
  ```

- ||/

  描述：立方根。

  示例：

  ```
  gaussdb=# SELECT ||/ 27.0 AS RESULT;
   result
  --------
       3
  (1 row)
  ```

- !

  描述：阶乘。

  示例：

  ```
  gaussdb=# SELECT 5! AS RESULT;
   result
  --------
     120
  (1 row)
  ```

- !!

  描述：阶乘（前缀操作符）。

  示例：

  ```
  gaussdb=# SELECT !!5 AS RESULT;
   result
  --------
     120
  (1 row)
  ```

- &

  描述：二进制AND。

  示例：

  ```
  gaussdb=# SELECT 91&15  AS RESULT;
   result
  --------
     11
  (1 row)
  ```

- |

  描述：二进制OR。

示例：

```
gaussdb=# SELECT 32|3  AS RESULT;
 result
--------
     35
(1 row)
```

- #

  描述：二进制XOR。

  示例：

  ```
  gaussdb=# SELECT 17#5  AS RESULT;
   result
  --------
      20
  (1 row)
  ```

- ~

  描述：二进制NOT。

  示例：

  ```
  gaussdb=# SELECT ~1 AS RESULT;
   result
  --------
     -2
  (1 row)
  ```

- <<

  描述：二进制左移。

  示例：

  ```
  gaussdb=# SELECT 1<<4 AS RESULT;
   result
  --------
     16
  (1 row)
  ```

- >>

  描述：二进制右移。

  示例：

  ```
  gaussdb=# SELECT 8>>2 AS RESULT;
   result
  --------
      2
  (1 row)
  ```

数字操作函数

- abs(x)

  描述：绝对值。

  返回值类型：和输入相同。

  示例：

  ```
  gaussdb=# SELECT abs(-17.4);
   abs
  ------
   17.4
  (1 row)
  ```

- acos(x)

  描述：反余弦。

  返回值类型：double precision

示例：

```
gaussdb=# SELECT acos(-1);
       acos
------------------
 3.14159265358979
(1 row)
```

- asin(x)

  描述：反正弦。

  返回值类型：double precision

  示例：

  ```
  gaussdb=# SELECT asin(0.5);
         asin
  ------------------
   .523598775598299
  (1 row)
  ```

- atan(x)

  描述：反正切。

  返回值类型：double precision

  示例：

  ```
  gaussdb=# SELECT atan(1);
         atan
  ------------------
   .785398163397448
  (1 row)
  ```

- atan2(y, x)

  描述：y/x的反正切。

  返回值类型：double precision

  示例：

  ```
  gaussdb=# SELECT atan2(2, 1);
        atan2
  ------------------
   1.10714871779409
  (1 row)
  ```

- bitand(integer, integer)

  描述：计算两个数字与运算(&)的结果。

  返回值类型：bigint类型数字。

  示例：

  ```
  gaussdb=# SELECT bitand(127, 63);
   bitand
  --------
      63
  (1 row)
  ```

- cbrt(dp)

  描述：立方根。

  返回值类型：double precision

  示例：

  ```
  gaussdb=# SELECT cbrt(27.0);
   cbrt
  ------
     3
  (1 row)
  ```


- ceil(x)
  描述：不小于参数的最小的整数。
  返回值类型：整数。
  示例：

```
gaussdb=# SELECT ceil(-42.8);
 ceil
------
  -42
(1 row)
```

- ceiling(dp or numeric)
  描述：不小于参数的最小整数（ceil的别名）。
  返回值类型：dp or numeric，不考虑隐式类型转换的情况下与输入相同。
  示例：

```
gaussdb=# SELECT ceiling(-95.3);
 ceiling
---------
     -95
(1 row)
```

- cos(x)
  描述：余弦。
  返回值类型：double precision
  示例：

```
gaussdb=# SELECT cos(-3.1415927);
        cos
--------------------
 -.999999999999999
(1 row)
```

- cosh(x)
  描述：双曲余弦。
  返回值类型：dp or numeric，不考虑隐式类型转换的情况下与输入相同。
  示例：

```
gaussdb=# SELECT cosh(4);
        cosh
--------------------
 27.3082328360165
(1 row)
```

  说明

  此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s2的情况下有效。

- cot(x)
  描述：余切。
  返回值类型：double precision
  示例：

```
gaussdb=# SELECT cot(1);
       cot
------------------
 .642092615934331
(1 row)
```

- degrees(dp)
  描述：把弧度转为角度。

返回值类型：double precision

示例：

```
gaussdb=# SELECT degrees(0.5);
     degrees
------------------
 28.6478897565412
(1 row)
```

- div(y numeric, x numeric)

  描述：y除以x的商的整数部分。

  返回值类型：numeric

  示例：

  ```
  gaussdb=# SELECT div(9,4);
   div
  -----
     2
  (1 row)
  ```

- exp(x)

  描述：自然指数。

  返回值类型：dp or numeric，不考虑隐式类型转换的情况下与输入相同。

  示例：

  ```
  gaussdb=# SELECT exp(1.0);
          exp
  --------------------
   2.7182818284590452
  (1 row)
  ```

- floor(x)

  描述：不大于参数的最大整数。

  返回值类型：与输入相同。

  示例：

  ```
  gaussdb=# SELECT floor(-42.8);
   floor
  -------
     -43
  (1 row)
  ```

- int1(in)

  描述：将传入的text参数转换为int1类型值并返回。

  返回值类型：int1

  示例：

  ```
  gaussdb=# SELECT int1('123');
   int1
  ------
   123
  (1 row)
  gaussdb=# SELECT int1('1.1');
   int1
  ------
     1
  (1 row)
  ```

  说明

  - 当sql_compatibility 为 'B'时，非整数数值类型的字符输入，会自动截断或返回数值0。
  - 当sql_compatibility 不为 'B'时，非整数数值类型的字符输入，报错非合法输入。

- int2(in)

  描述：将传入参数转换为int2类型值并返回。

  支持的入参类型包括float4，float8，int16，numeric，text。

  返回值类型：int2

  示例：

```
gaussdb=# SELECT int2('1234');
 int2
------
 1234
(1 row)
gaussdb=# SELECT int2(25.3);
 int2
------
   25
(1 row)
```

  说明

  - 当sql_compatibility 为 'B'时，非整数数值类型的字符输入，会自动截断或返回数值0。
  - 当sql_compatibility 不为 'B'时，非整数数值类型的字符输入，报错非合法输入。

- int4(in)

  描述：将传入参数转换为int4类型值并返回。

  支持的入参类型包括bit、boolean、char、double precision、int16、numeric、real、smallint、text。

  返回值类型：int4

  示例：

```
gaussdb=# SELECT int4('789');
 int4
------
 789
(1 row)
gaussdb=# SELECT int4(99.9);
 int4
------
  100
(1 row)
```

  说明

  - 当sql_compatibility 为 'B'时，非整数数值类型的字符输入，会自动截断或返回数值0。
  - 当sql_compatibility 不为 'B'时，非整数数值类型的字符输入，报错非合法输入。

- int8(in)

  描述：将传入参数转换为int8类型值并返回。支持的入参类型包括：bit、double precision、int16、integer、numeric、oid、real、smallint和text。

  返回值类型：int8

  示例：

```
gaussdb=# SELECT int8('789');
 int8
------
 789
(1 row)
gaussdb=# SELECT int8(99.9);
 int8
------
  99
(1 row)
```


说明

- 当sql_compatibility 为 'B'时，非整数数值类型的字符输入，会自动截断或返回数值0。
- 当sql_compatibility 不为 'B'时，非整数数值类型的字符输入，报错非合法输入。

- float4(in)

  描述：将传入参数转换为float4类型值并返回。支持的入参类型包括：bigint、double precision、int16, integer、numeric、smallint、text。

  返回值类型：float4

  示例：

```
gaussdb=# SELECT float4('789');
 float4
--------
    789
(1 row)

gaussdb=# SELECT float4(99.9);
 float4
--------
   99.9
(1 row)
```

- float8(in)

  描述：将传入参数转换为float8类型值并返回。支持的入参类型包括：bigint、int16、integer、numeric、real、smallint、text。

  返回值类型：float8

  示例：

```
gaussdb=# SELECT float8('789');
 float8
--------
    789
(1 row)

gaussdb=# SELECT float8(99.9);
 float8
--------
   99.9
(1 row)
```

- int16(in)

  描述：将传入参数转换为int16类型值并返回。支持的入参类型包括：bigint、boolean、double precision、integer、numeric、oid、real、smallint、tinyint。

  返回值类型：int16

  示例：

```
gaussdb=# SELECT int16('789');
 int16
--------
    789
(1 row)

gaussdb=# SELECT int16(99.9);
 int16
--------
   100
(1 row)
```

- numeric(in)

  描述：将传入参数转换为numeric类型值并返回。支持的入参类型包括：bigint、boolean、double precision、int16、integer、money、real、smallint。

返回值类型：numeric

示例：

```
gaussdb=# SELECT "numeric"('789');
 numeric
---------
     789
(1 row)

gaussdb=# SELECT "numeric"(99.9);
 numeric
---------
    99.9
(1 row)
```

- oid(in)

  描述：将传入参数转换为oid类型值并返回。支持的入参类型包括：bigint、int16。

  返回值类型：oid

- radians(dp)

  描述：把角度转为弧度。

  返回值类型：double precision

  示例：

  ```
  gaussdb=# SELECT radians(45.0);
      radians
  ------------------
   .785398163397448
  (1 row)
  ```

- random()

  描述：0.0到1.0之间的随机数。

  返回值类型：double precision

  示例：

  ```
  gaussdb=# SELECT random();
       random
  ------------------
   .824823560658842
  (1 row)
  ```

- multiply(x double precision or text, y double precision or text)

  描述：x和y的乘积。

  返回值类型：double precision

  示例：

  ```
  gaussdb=# SELECT multiply(9.0, '3.0');
      multiply
  -------------------
                27
  (1 row)
  gaussdb=# SELECT multiply('9.0', 3.0);
      multiply
  -------------------
                27
  (1 row)
  ```

- ln(x)

  描述：自然对数。

  返回值类型：dp or numeric，不考虑隐式类型转换的情况下与输入相同。

  示例：


```
gaussdb=# SELECT ln(2.0);
        ln
-------------------
 .6931471805599453
(1 row)
```

- log(x)

  描述：以10为底的对数。

  返回值类型：与输入相同。

  示例：

  ```
  gaussdb=# SELECT log(100.0);
        log
  --------------------
   2.0000000000000000
  (1 row)
  ```

- log(b numeric, x numeric)

  描述：以b为底的对数。

  返回值类型：numeric

  示例：

  ```
  gaussdb=# SELECT log(2.0, 64.0);
        log
  --------------------
   6.0000000000000000
  (1 row)
  ```

- mod(x,y)

  描述：x/y的余数（模）。如果x是0，则返回0。

  返回值类型：与参数类型相同。

  示例：

  ```
  gaussdb=# SELECT mod(9,4);
   mod
  -----
     1
  (1 row)
  gaussdb=# SELECT mod(9,0);
   mod
  -----
     9
  (1 row)
  ```

- pi()

  描述：“π”常量。

  返回值类型：double precision

  示例：

  ```
  gaussdb=# SELECT pi();
        pi
  -------------------
   3.14159265358979
  (1 row)
  ```

- power(a double precision, b double precision)

  描述：a的b次幂。

  返回值类型：double precision

  示例：

  ```
  gaussdb=# SELECT power(9.0, 3.0);
        power
  ----------------------
  ```



```
 729.0000000000000000
(1 row)
```

- remainder(x,y)

  描述：x/y的余数，如果y是0，则报错。

  返回值类型：与输入相同（float4、float8或者numeric类型）

  示例：

```
gaussdb=# SELECT remainder(11,4);
 remainder
-----------
        -1
(1 row)
gaussdb=# SELECT remainder(9,0);
ERROR: division by zero
CONTEXT:  referenced column: remainder
```

  说明

  此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s2的情况下有效。

- round(x)

  描述：离输入参数最近的整数。

  返回值类型：与输入相同（double precision或者numeric类型）。

  示例：

```
gaussdb=# SELECT round(42.4);
 round
-------
    42
(1 row)

gaussdb=# SELECT round(42.6);
 round
-------
    43
(1 row)
```

  注意

  float/double类型的输出结果可能会出现-0（trunc、ceil等函数同样会出现此种情形。在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s1的情况下结果为0），例如：

```
gaussdb=# SELECT round(-0.2::float8);
 round
-------
    -0
(1 row)
```

- round(v numeric, s int)

  描述：保留小数点后s位，s后一位进行四舍五入。

  返回值类型：numeric

  示例：

```
gaussdb=# SELECT round(42.4382, 2);
 round
-------
 42.44
(1 row)
```

说明

控制参数s输入为小数时：在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s1的情况下s截断为整数，否则s四舍五入为整数。

在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s1的情况下，round函数支持round(timestamp, text)重载，在以(text, text)或(text, '')为入参调用round函数时会优先选择round(timestamp, text)。

- setseed(dp)

  描述：为随后的random()调用设置种子(-1.0到1.0之间，包含)。

  返回值类型：void

  示例：

```
gaussdb=# SELECT setseed(0.54823);
 setseed
---------

(1 row)
```

- sign(x)

  描述：输出此参数的符号。

  返回值类型：-1表示负数，0表示0，1表示正数。

  示例：

```
gaussdb=# SELECT sign(-8.4);
 sign
------
   -1
(1 row)
```

- sin(x)

  描述：正弦。

  返回值类型：double precision

  示例：

```
gaussdb=# SELECT sin(1.57079);
        sin
-------------------
 .999999999979986
(1 row)
```

- sinh(x)

  描述：双曲正弦。

  返回值类型：dp or numeric，不考虑隐式类型转换的情况下与输入相同。

  示例：

```
gaussdb=# SELECT sinh(4);
        sinh
-------------------
 27.2899171971277
(1 row)
```

说明

此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s2的情况下有效。

- sqrt(x)

  描述：平方根。

  返回值类型：dp or numeric，不考虑隐式类型转换的情况下与输入相同。

  示例：


```
gaussdb=# SELECT sqrt(2.0);
       sqrt
--------------------
 1.414213562373095
(1 row)
```

- tan(x)

  描述：正切。

  返回值类型：double precision

  示例：

```
gaussdb=# SELECT tan(20);
       tan
-------------------
 2.23716094422474
(1 row)
```

- tanh(x)

  描述：双曲正切。

  返回值类型：与输入相同（double precision或者numeric类型）。

  示例：

```
gaussdb=# SELECT tanh(0.1);
                tanh
-------------------------------------------
0.099667994624955817171830508367835218353 89
(1 row)
```

  说明

  此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s2的情况下有效。

- trunc(x)

  描述：截断（取整数部分）。

  返回值类型：与输入相同。

  示例：

```
gaussdb=# SELECT trunc(42.8);
 trunc
-------
    42
(1 row)
```

- trunc(v numeric, s int)

  描述：截断为s位小数。

  返回值类型：numeric

  示例：

```
gaussdb=# SELECT trunc(42.4382, 2);
 trunc
-------
 42.43
(1 row)
```

  说明

  在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为兼容配置项下的有效值时，参数s若入参为小数则不会被四舍五入，而是被截断。

- smgrne(a smgr, b smgr)

  描述：比较两个smgr类型整数是否不相等。

  返回值类型：boolean

- smgreq(a smgr, b smgr)
  描述：比较两个smgr类型整数是否相等。
  返回值类型：boolean
- int1abs
  描述：返回uint8类型数据的绝对值。
  参数：tinyint
  返回值类型：tinyint
- int1and
  描述：返回两个uint8类型数据按位与的结果。
  参数：tinyint, tinyint
  返回值类型：tinyint
- int1cmp
  描述：返回两个uint8类型数据比较的结果，若第一个参数大，则返回1；若第二个参数大，则返回-1；若相等，则返回0。
  参数：tinyint, tinyint
  返回值类型：integer
- int1div
  描述：返回两个uint8类型数据相除的结果，结果为float8类型。
  参数：tinyint, tinyint
  返回值类型：tinyint
- int1eq
  描述：比较两个uint8类型数据是否相等。
  参数：tinyint, tinyint
  返回值类型：boolean
- int1ge
  描述：判断两个uint8类型数据是否第一个参数大于等于第二个参数。
  参数：tinyint, tinyint
  返回值类型：boolean
- int1gt
  描述：无符号1字节整数做大于运算。
  参数：tinyint, tinyint
  返回值类型：boolean
- int1larger
  描述：无符号1字节整数求最大值。
  参数：tinyint, tinyint
  返回值类型：tinyint
- int1le
  描述：无符号1字节整数做小于等于运算。
  参数：tinyint, tinyint
  返回值类型：boolean

- int1lt
  描述：无符号1字节整数做小于运算。
  参数：tinyint, tinyint
  返回值类型：boolean
- int1smaller
  描述：无符号1字节整数求最小算。
  参数：tinyint, tinyint
  返回值类型：tinyint
- int1inc
  描述：无符号1字节整数加一。
  参数：tinyint
  返回值类型：tinyint
- int1mi
  描述：无符号一字节整数做差运算。
  参数：tinyint, tinyint
  返回值类型：tinyint
- int1mod
  描述：无符号一字节整数做取余运算。
  参数：tinyint, tinyint
  返回值类型：tinyint
- int1mul
  描述：无符号一字节整数做乘法运算。
  参数：tinyint, tinyint
  返回值类型：tinyint
- int1ne
  描述：无符号一字节整数不等于运算。
  参数：tinyint, tinyint
  返回值类型：boolean
- int1pl
  描述：无符号一字节整数加法。
  参数：tinyint, tinyint
  返回值类型：tinyint
- int1um
  描述：无符号一字节数取相反数并返回有符号二字节整数。
  参数：tinyint
  返回值类型：smallint
- int1xor
  描述：无符号一字节整数异或操作。
  参数：tinyint, tinyint
  返回值类型：tinyint

- cash_div_int1

  描述：对money类型进行除法运算。

  参数：money, tinyint

  返回值类型：money

- cash_mul_int1

  描述：对money类型进行乘法运算。

  参数：money, tinyint

  返回值类型：money

- int1not

  描述：无符号一字节整数二进制位翻转。

  参数：tinyint

  返回值类型：tinyint

- int1or

  描述：无符号一字节整数或运算。

  参数：tinyint, tinyint

  返回值类型：tinyint

- int1shl

  描述：无符号一字节整数左移指定位数。

  参数：tinyint, integer

  返回值类型：tinyint

- int1shr

  描述：无符号一字节整数右移指定位数。

  参数：tinyint, integer

  返回值类型：tinyint

- width_bucket(op numeric, b1 numeric, b2 numeric, count int)

  描述：返回一个桶，这个桶是在一个有count个桶，上界为b1下界为b2的等深柱图中operand将被赋予的那个桶。

  返回值类型：int

  示例：

  ```
  gaussdb=# SELECT width_bucket(5.35, 0.024, 10.06, 5);
   width_bucket
  --------------
             3
  (1 row)
  ```

- width_bucket(op dp, b1 dp, b2 dp, count int)

  描述：返回一个桶，这个桶是在一个有count个桶，上界为b1下界为b2的等深柱图中operand将被赋予的那个桶。

  返回值类型：int

  示例：

  ```
  gaussdb=# SELECT width_bucket(5.35, 0.024, 10.06, 5);
   width_bucket
  --------------
             3
  (1 row)
  ```


- analyze_tgtype_for_type(n smallint)

  描述：用于解析pg_trigger.tgtype，按位解析n，并返回before each row, after each row, before statement, after statement, instead of中的一个

  返回值类型：varchar2(16)

- analyze_tgtype_for_event(n smallint)

  描述：用于解析pg_trigger.tgtype，按位解析n，并返回insert, update, delete, truncate中的一个或多个

  返回值类型：varchar2(246)

- nanvl(n2, n1)

  描述：输入两个参数，要求为数字类型或可以被隐式转化为数字类型的非数字类型；若第一个参数n2为NaN，返回n1，否则返回n2。

  返回值类型：入参参数中最优先的类型，优先级double precision>float4>numeric。

  示例：

```
gaussdb=# SELECT nanvl('NaN', 1.1);
 nanvl
-------
   1.1
(1 row)
```

  说明

  此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s2的情况下有效。

7.5.8 时间和日期处理函数和操作符

时间日期操作符

时间日期操作符请参见**表7-31**。

说明

要尽量避免在查询中使用 'now'::date、'now'::timestamp、'now'::timestamptz字符串常量强转以及text_date('now')的类似表达式来获取数据库当前时间或者将当前时间值作为函数入参场景，在这些场景下，优化器会提前算出常量时间，造成查询结果不正确。

```
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE b='now'::date;
QUERY PLAN
------------------------------------------------------
Seq Scan on t1  (cost=0.00..13.60 rows=1 width=310)
Filter: ((b)::text = '2024-11-09 15:07:56'::text)
(2 rows)
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE b=text_date('now');
QUERY PLAN
------------------------------------------------------
Seq Scan on t1  (cost=0.00..13.60 rows=1 width=310)
Filter: ((b)::text = '2024-11-09'::text)
(2 rows)
```

推荐使用now()、currenttimestamp()函数作为获取数据库当前时间的方法。

```
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE b=now();
QUERY PLAN
------------------------------------------------------
Seq Scan on t1  (cost=0.00..14.80 rows=1 width=310)
Filter: ((b)::text = (now())::text)
(2 rows)
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE b=text_date(now());
QUERY PLAN
------------------------------------------------------------
Seq Scan on t1  (cost=0.00..16.00 rows=1 width=310)
Filter: ((b)::text = (text_date((now())::text))::text)
(2 rows)
```

用户在使用时间和日期操作符时，对应的操作数请使用明确的类型前缀修饰，以确保数据库在解析操作数的时候能够与用户预期一致，不会产生用户非预期的结果。

比如下面示例没有明确数据类型就会出现异常错误。

```
SELECT date '2001-10-01' - '7' AS RESULT;
```

表 7-31 时间和日期操作符

操作符	示例
+	gaussdb=# SELECT date '2001-9-28' + integer '7' AS RESULT; result ------------ 2001-10-05 (1 row) **说明** 在A兼容模式下，查询结果为2001-10-05 00:00:00。
	gaussdb=# SELECT date '2001-09-28' + interval '1 hour' AS RESULT; result --------------------- 2001-09-28 01:00:00 (1 row)
	gaussdb=# SELECT date '2001-09-28' + time '03:00' AS RESULT; result --------------------- 2001-09-28 03:00:00 (1 row)

操作符	示例
	gaussdb=# SELECT interval '1 day' + interval '1 hour' AS RESULT; result ---------- 1 day 01:00:00 (1 row)
	gaussdb=# SELECT timestamp '2001-09-28 01:00' + interval '23 hours' AS RESULT; result ---------- 2001-09-29 00:00:00 (1 row)
	gaussdb=# SELECT time '01:00' + interval '3 hours' AS RESULT; result ---------- 04:00:00 (1 row)
-	gaussdb=# SELECT date '2001-10-01' - date '2001-09-28' AS RESULT; result -------- 3days (1 row)
	gaussdb=# SELECT date '2001-10-01' - integer '7' AS RESULT; result ---------- 2001-09-24 00:00:00 (1 row)
	gaussdb=# SELECT date '2001-09-28' - interval '1 hour' AS RESULT; result ---------- 2001-09-27 23:00:00 (1 row)
	gaussdb=# SELECT time '05:00' - time '03:00' AS RESULT; result ---------- 02:00:00 (1 row)
	gaussdb=# SELECT time '05:00' - interval '2 hours' AS RESULT; result ---------- 03:00:00 (1 row)
	gaussdb=# SELECT timestamp '2001-09-28 23:00' - interval '23 hours' AS RESULT; result ---------- 2001-09-28 00:00:00 (1 row)
	gaussdb=# SELECT interval '1 day' - interval '1 hour' AS RESULT; result ---------- 23:00:00 (1 row)
	gaussdb=# SELECT timestamp '2001-09-29 03:00' - timestamp '2001-09-27 12:00' AS RESULT; result ---------- 1 day 15:00:00 (1 row)

操作符	示例
*	gaussdb=# SELECT 900 * interval '1 second' AS RESULT; result ---------- 00:15:00 (1 row)
	gaussdb=# SELECT 21 * interval '1 day' AS RESULT; result --------- 21 days (1 row)
	gaussdb=# SELECT double precision '3.5' * interval '1 hour' AS RESULT; result ---------- 03:30:00 (1 row)
/	gaussdb=# SELECT interval '1 hour' / double precision '1.5' AS RESULT; result ---------- 00:40:00 (1 row)

时间/日期函数

- age(timestamp, timestamp)

  描述：将两个参数相减，并以年、月、日作为返回值。若相减值为负，则函数返回亦为负，入参可以都带timezone或都不带timezone。

  返回值类型：interval

  示例：

  ```
  gaussdb=# SELECT age(timestamp '2001-04-10', timestamp '1957-06-13');
           age
  --------------------------
   43 years 9 mons 27 days
  (1 row)
  ```

- age(timestamp)

  描述：当前SQL执行开始时刻的系统时间和参数相减，入参可以带或者不带timezone。

  返回值类型：interval

  示例：

  ```
  gaussdb=# SELECT age(timestamp '1957-06-13');
           age
  --------------------------
   60 years 2 mons 18 days
  (1 row)
  ```

- clock_timestamp()

  描述：返回当前函数被调用时的系统时间的时间戳。volatile函数，每次扫描都会取最新的时间戳，因此在一次查询中每次调用结果不相同。

  返回值类型：timestamp with time zone

  示例：

  ```
  gaussdb=# SELECT clock_timestamp();
        clock_timestamp
  ```



```
--------------------------------
 2017-09-01 16:57:36.636205+08
(1 row)
```

- current_date

  描述：返回当前本条SQL启动的系统时间的日期。

  返回值类型：date

  示例：

  ```
  gaussdb=# SELECT current_date;
     date
  ------------
   2017-09-01
  (1 row)
  ```

  说明

  此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s2的情况下返回值类型为timestamp。

- current_time

  描述：当前事务的开始时刻的系统时间。

  返回值类型：time with time zone

  示例：

  ```
  gaussdb=# SELECT current_time;
        timetz
  --------------------
   16:58:07.086215+08
  (1 row)
  ```

- current_timestamp

  描述：返回的结果为当前SQL启动的系统时间。在PL/SQL中，简单的赋值语句如：time1 := current_timestamp，被认为是表达式，所以会返回上一条SQL语句启动时间。

  返回值类型：timestamp with time zone

  示例：

  ```
  gaussdb=# SELECT current_timestamp;
        pg_systimestamp
  -------------------------------
   2017-09-01 16:58:19.22173+08
  (1 row)
  ```

- current_timestamp(precision)

  描述：返回的结果为当前事务启动的系统时间，并将结果的微秒圆整为指定小数位。

  返回值类型：timestamp with time zone

  示例：

  ```
  gaussdb=# SELECT current_timestamp(1);
        timestamptz
  --------------------------------
   2017-09-01 16:58:19.2+08
  (1 row)
  ```

  说明

  - 此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s2的情况下，precision参数支持numeric类型的整值，否则仅支持int输入。
  - 微秒末位的0不显示。如 2017-09-01 10:32:19.212000 输出显示为 2017-09-01 10:32:19.212。

- pg_systimestamp()

  描述：当前日期和时间(当前语句的开始)。

  返回值类型：timestamp with time zone

  示例：

```
gaussdb=# SELECT pg_systimestamp();
       pg_systimestamp
-------------------------------
 2015-10-14 11:21:28.317367+08
(1 row)
```

- date_part(text, timestamp)

  描述：获取日期/时间值中子域的值，例如年或者小时的值。等效于extract(field from timestamp)。

  timestamp类型：abstime、date、interval、reltime、time with time zone、time without time zone、timestamp with time zone、timestamp without time zone。

  返回值类型：double precision

  示例：

```
gaussdb=# SELECT date_part('hour', timestamp '2001-02-16 20:38:40');
 date_part
-----------
        20
(1 row)
```

- date_part(text, interval)

  描述：获取日期/时间值中子域的值。获取月份值时，如果月份值大于12，则取与12的模。等效于extract(field from timestamp)。

  返回值类型：double precision

  示例：

```
gaussdb=# SELECT date_part('month', interval '2 years 3 months');
 date_part
-----------
         3
(1 row)
```

- date_trunc(text, timestamp)

  描述：截取到参数text指定的精度。

  返回值类型：interval、timestamp with time zone、timestamp without time zone

  示例：

```
gaussdb=# SELECT date_trunc('hour', timestamp  '2001-02-16 20:38:40');
     date_trunc
---------------------
 2001-02-16 20:00:00
(1 row)
```

- trunc(timestamp)

  描述：默认按天截取。

  示例：

```
gaussdb=# SELECT trunc(timestamp  '2001-02-16
20:38:40');
                      trunc
---------------------
2001-02-16 00:00:00
(1 row)
```


- trunc(arg1, arg2)

  描述：截取到arg2指定的精度。

  arg1类型：interval、timestamp with time zone、timestamp without time zone

  arg2类型：text

  返回值类型：interval、timestamp with time zone、timestamp without time zone

  示例：

```
gaussdb=# SELECT trunc(timestamp  '2001-02-16 20:38:40',
'hour');
                trunc
---------------------
2001-02-16 20:00:00
(1 row)
```

- round(arg1, arg2)

  描述：四舍五入到arg2指定的精度。

  arg1类型：timestamp without time zone

  arg2类型：text

  返回值类型：timestamp without time zone

  示例：

```
gaussdb=# SELECT round(timestamp  '2001-02-16 20:38:40',
'hour');
                round
---------------------
2001-02-16 21:00:00
(1 row)
```

  说明

  此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s1的情况下有效。

- daterange(arg1, arg2)

  描述：获取时间边界信息。arg1和arg2的类型为date。

  返回值类型：daterange

  示例：

```
gaussdb=# select daterange('2000-05-06','2000-08-08');
      daterange
-------------------------
 [2000-05-06,2000-08-08)
(1 row)
```

- daterange(arg1, arg2, text)

  描述：获取时间边界信息。arg1和arg2的类型为date，text类型为text。

  返回值类型：daterange

  示例：

```
gaussdb=# SELECT daterange('2000-05-06','2000-08-08','[]');
      daterange
-------------------------
 [2000-05-06,2000-08-09)
(1 row)
```

- extract(field from timestamp)

  描述：获取小时的值。

返回值类型：double precision

示例：

```
gaussdb=# SELECT extract(hour from timestamp '2001-02-16 20:38:40');
 date_part
-----------
        20
(1 row)
```

- extract(field from interval)

  描述：获取月份的值。如果大于12，则取与12的模。

  返回值类型：double precision

  示例：

  ```
  gaussdb=# SELECT extract(month from interval '2 years 3 months');
   date_part
  -----------
           3
  (1 row)
  ```

- isfinite(date)

  描述：判断日期是否为有限值，是则返回t，否则返回f。

  返回值类型：Boolean

  示例：

  ```
  gaussdb=# SELECT isfinite(date '2001-02-16');
   isfinite
  ----------
   t
  (1 row)
  gaussdb=# SELECT isfinite(date 'infinity');
   isfinite
  ----------
   f
  (1 row)
  ```

- isfinite(timestamp)

  描述：判断时间戳是否为有限值，是则返回t，否则返回f。

  返回值类型：Boolean

  示例：

  ```
  gaussdb=# SELECT isfinite(timestamp '2001-02-16 21:28:30');
   isfinite
  ----------
   t
  (1 row)
  gaussdb=# SELECT isfinite(timestamp 'infinity');
   isfinite
  ----------
   f
  (1 row)
  ```

- isfinite(interval)

  描述：判断时间间隔是否为有限值，是则返回t，暂不支持返回f，输入'infinity'会报错。

  返回值类型：Boolean

  示例：

  ```
  gaussdb=# SELECT isfinite(interval '4 hours');
   isfinite
  ----------
   t
  (1 row)
  ```


- justify_days(interval)
  描述：将时间间隔以月（30天为一月）为单位。
  返回值类型：interval
  示例：

```
gaussdb=# SELECT justify_days(interval '35 days');
 justify_days
--------------
 1 mon 5 days
(1 row)
```

- justify_hours(interval)
  描述：将时间间隔以天（24小时为一天）为单位。
  返回值类型：interval
  示例：

```
gaussdb=# SELECT JUSTIFY_HOURS(INTERVAL '27 HOURS');
 justify_hours
----------------
 1 day 03:00:00
(1 row)
```

- justify_interval(interval)
  描述：结合justify_days和justify_hours，调整interval。
  返回值类型：interval
  示例：

```
gaussdb=# SELECT JUSTIFY_INTERVAL(INTERVAL '1 MON -1 HOUR');
 justify_interval
------------------
 29 days 23:00:00
(1 row)
```

- localtime
  描述：当前事务的开始时刻的系统时间。
  返回值类型：time
  示例：

```
gaussdb=# SELECT localtime AS RESULT;
     result
----------------
 16:05:55.664681
(1 row)
```

- localtimestamp
  描述：返回当前本条SQL执行开始时刻的系统日期和时间。
  返回值类型：timestamp
  示例：

```
gaussdb=# SELECT localtimestamp;
         timestamp
----------------------------
 2017-09-01 17:03:30.781902
(1 row)
```

- now()
  描述：当前事务的开始时刻的系统的日期及时间，同一个事务内返回结果相同。
  返回值类型：timestamp with time zone
  示例：

```
gaussdb=# SELECT now();
              now
```



```
-------------------------------
 2017-09-01 17:03:42.549426+08
(1 row)
```

- timenow()

  描述：返回当前本条SQL执行开始时刻的系统日期和时间。

  返回值类型：abstime

  示例：

```
gaussdb=# SELECT timenow();
       timenow
------------------------
 2020-06-23 20:36:56+08
(1 row)
```

- dbtimezone

  描述：当前数据库的时区。

  返回值类型：text

  示例：

```
gaussdb=# SELECT dbtimezone;
       dbtimezone
------------------------
 PRC
(1 row)
```

  说明

  此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s2的情况下有效。

- numtodsinterval(num, interval_unit)

  描述：将数字转换为interval类型。num为numeric类型数字，interval_unit为固定格式字符串（'DAY' | 'HOUR' | 'MINUTE' | 'SECOND'）。

  可以通过设置GUC参数**IntervalStyle**为a，兼容该函数interval输出格式。

  返回值类型：interval

  示例：

```
gaussdb=# SELECT numtodsinterval(100, 'HOUR');
 numtodsinterval
-----------------
 100:00:00
(1 row)

gaussdb=# SET intervalstyle = a;
SET
gaussdb=# SELECT numtodsinterval(100, 'HOUR');
       numtodsinterval
------------------------------
 +000000004 04:00:00.000000000
(1 row)
```

  说明

  此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s2的情况下：当参数interval_unit为 'DAY' 时，参数num超过1000000000会报错。

- numtoyminterval(num, interval_unit)

  描述：将数字转换为interval类型。num为numeric类型数字，interval_unit为固定格式字符串（'YEAR' | 'MONTH'）。

  可以通过设置GUC参数**IntervalStyle**为a，兼容该函数interval输出格式。

  返回值类型：interval

示例：

```
gaussdb=# SELECT numtoyminterval(100, 'MONTH');
 numtoyminterval
-----------------
 8 years 4 mons
(1 row)

gaussdb=# SET intervalstyle = 'a';
SET
gaussdb=# SELECT numtoyminterval(100, 'MONTH');
 numtoyminterval
-----------------
 8-4
(1 row)
```

说明

此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s2的情况下有效。

- new_time(date, timezone1,timezone2)

  描述：当timezone1所表示时区的日期时间为date的时候，返回此时timezone2所表示时区的日期时间值。

  返回值类型：timestamp

  示例：

```
gaussdb=# SELECT new_time('1997-10-10','AST','EST');
      new_time
---------------------
 1997-10-09 23:00:00
(1 row)
gaussdb=# SELECT NEW_TIME(TO_TIMESTAMP ('10-Sep-02 14:10:10.123000','DD-Mon-RR
HH24:MI:SS.FF'), 'AST', 'PST');
        new_time
-------------------------
 2002-09-10 10:10:10.123
(1 row)
```

说明

此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s2的情况下生效。

- sessiontimezone

  描述：当前会话的时区，无入参。

  返回值类型：text。

  示例：

```
gaussdb=# SELECT SESSIONTIMEZONE;
 session_time_zone
-------------------
 PST8PDT
(1 row)
gaussdb=# SELECT LOWER(SESSIONTIMEZONE);
   lower
-----------
 @ 8 hours
(1 row)
```

说明

此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s2的情况下生效。

当set session time zone的值为GMT+08:00/GMT-08:00格式时，正值的偏移量被用于格林威治*以西*的位置，例如GMT+08:00表示西八区，GMT-08:00表示东八区。

- sys_extract_utc(timestamp| timestamptz)

  描述：从具有时区偏移量或时区区域名称的日期时间值中提取UTC（协调世界时-以前称为格林威治平均时间）。如果未指定时区，则日期时间与会话时区关联。入参有timestmp和timestamp两种形式。

  返回值类型：timestamp。

  示例：

```
gaussdb=# SELECT SYS_EXTRACT_UTC(TIMESTAMP '2000-03-28 11:30:00.00');
   sys_extract_utc
---------------------
 2000-03-28 03:30:00
(1 row)
gaussdb=# SELECT SYS_EXTRACT_UTC(TIMESTAMPTZ '2000-03-28 11:30:00.00 -08:00');
   sys_extract_utc
---------------------
 2000-03-28 19:30:00
(1 row)
```

  说明

  此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s2的情况下有效。

- tz_offset('time_zone_name' | '(+/-)hh:mi' | SESSIONTIMEZONE | DBTIMEZONE)

  描述：入参有以上四种形式，返回入参所表示时区的UTC偏移量。

  返回值类型：text。

  示例：

```
gaussdb=# SELECT TZ_OFFSET('US/Pacific');
 tz_offset
-----------
 -08:00
(1 row)
gaussdb=# SELECT TZ_OFFSET(sessiontimezone);
 tz_offset
-----------
 +08:00
(1 row)
```

  说明

  此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s2的情况下生效。

- pg_sleep(seconds)

  描述：服务器线程延迟时间，单位为秒。

  返回值类型：void

  示例：

```
gaussdb=# SELECT pg_sleep(10);
 pg_sleep
----------

(1 row)
```

- statement_timestamp()

  描述：当前日期和时间（当前语句的开始）。

  返回值类型：timestamp with time zone

  示例：

```
gaussdb=# SELECT statement_timestamp();
     statement_timestamp
```



```
-------------------------------
 2017-09-01 17:04:39.119267+08
(1 row)
```

- sysdate

  描述：返回当前本条SQL执行时刻的系统日期和时间。

  返回值类型：timestamp

  示例：

```
gaussdb=# SELECT sysdate;
      sysdate
--------------------
 2017-09-01 17:04:49
(1 row)
```

- current_sysdate

  描述：返回当前本条SQL执行开始时刻的系统日期和时间。

  返回值类型：timestamp

  示例：

```
gaussdb=# SELECT current_sysdate();
  current_sysdate
---------------------
 2023-06-20 20:09:02
(1 row)
```

- timeofday()

  描述：返回当前函数被调用时的系统时间的时间时间戳（像clock_timestamp，但是返回时为text）。

  返回值类型：text

  示例：

```
gaussdb=# SELECT timeofday();
              timeofday
--------------------------------------
 Fri Sep 01 17:05:01.167506 2017 CST
(1 row)
```

- transaction_timestamp()

  描述：当前事务开始的系统的日期及时间。

  返回值类型：timestamp with time zone

  示例：

```
gaussdb=# SELECT transaction_timestamp();
    transaction_timestamp
-------------------------------
 2017-09-01 17:05:13.534454+08
(1 row)
```

- add_months(d,n)

  描述：用于计算时间点d再加上n个月的时间。

  d：timestamp类型的值，以及可以隐式转换为timestamp类型的值。

  n：INTEGER类型的值，以及可以隐式转换为INTEGER类型的值。

  返回值类型：timestamp

  示例：

```
gaussdb=# SELECT add_months(to_date('2017-5-29', 'yyyy-mm-dd'), 11) FROM sys_dummy;
    add_months
---------------------
 2018-04-29 00:00:00
(1 row)
```


说明

此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s1的情况下：

- 当计算结果大于公元9999年时会报错。
- 参数n入参若为小数则不会被四舍五入，而是被截断。

- last_day(d)

  描述：用于计算时间点d当月最后一天的时间。

  返回值类型：timestamp

  示例：

```
gaussdb=# SELECT last_day(to_date('2017-01-01', 'YYYY-MM-DD')) AS cal_result;
     cal_result
---------------------
 2017-01-31 00:00:00
(1 row)
```

- months_between(d1, d2)

  描述：用于计算时间点d1和时间点d2的月份差值，如果两个日期都是月末或天数相同，则返回整数，否则返回值带小数，按31天/月计算。

  返回值类型：numeric

  示例：

```
gaussdb=# SELECT months_between(to_date('2022-10-31', 'yyyy-mm-dd'), to_date('2022-09-30',
'yyyy-mm-dd'));
 months_between
----------------
              1
(1 row)

gaussdb=# SELECT months_between(to_date('2022-10-30', 'yyyy-mm-dd'), to_date('2022-09-30',
'yyyy-mm-dd'));
 months_between
----------------
              1
(1 row)

gaussdb=# SELECT months_between(to_date('2022-10-29', 'yyyy-mm-dd'), to_date('2022-09-30',
'yyyy-mm-dd'));
    months_between
-----------------------
 .96774193548387096774
(1 row)
```

说明

此函数在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s1的情况下有效。

- next_day(x,y)

  描述：用于计算时间点x开始的下一个星期几（y）的时间。

  返回值类型：timestamp

  示例：

```
gaussdb=# SELECT next_day(timestamp '2017-05-25 00:00:00','Sunday')AS cal_result;
     cal_result
---------------------
 2017-05-28 00:00:00
(1 row)
```


说明

- next_day第二个参数支持缩写(不区分大小写)，包括SUN、MON、TUE、WED、THU、THUR、FRI、SAT。

- tinterval(abstime, abstime)

  描述：用两个绝对时间创建时间间隔。

  返回值类型：tinterval

  示例：

```
gaussdb=# call tinterval(abstime 'May 10, 1947 23:59:12', abstime 'Mon May  1 00:30:30 1995');
                    tinterval
----------------------------------------------------
 ["1947-05-10 23:59:12+09" "1995-05-01 00:30:30+08"]
(1 row)
```

- tintervalend(tinterval)

  描述：返回tinterval的结束时间。

  返回值类型：abstime

  示例：

```
gaussdb=# SELECT tintervalend('["Sep 4, 1983 23:59:12" "Oct4, 1983 23:59:12"]');
     tintervalend
-------------------------
 1983-10-04 23:59:12+08
(1 row)
```

- tintervalrel(tinterval)

  描述：计算并返回tinterval的相对时间。

  返回值类型：reltime

  示例：

```
gaussdb=# SELECT tintervalrel('["Sep 4, 1983 23:59:12" "Oct4, 1983 23:59:12"]');
 tintervalrel
--------------
 1 mon
(1 row)
```

- smalldatetime_ge

  描述：判断是否第一个参数大于等于第二个参数。

  参数：smalldatetime, smalldatetime

  返回值类型：boolean

- smalldatetime_cmp

  描述：对比smalldatetime是否相等。

  参数：smalldatetime, smalldatetime

  返回值类型：integer

- smalldatetime_eq

  描述：对比smalldatetime是否相等。

  参数：smalldatetime, smalldatetime

  返回值类型：boolean。

- smalldatetime_gt

  描述：判断是否第一个参数大于第二个参数。

  参数：smalldatetime, smalldatetime

  返回值类型：boolean

- smalldatetime_hash
  描述：计算timestamp对应的哈希值。
  参数：smalldatetime
  返回值类型：integer
- smalldatetime_in
  描述：输入timestamp。
  参数：cstring, oid, integer
  返回值类型：smalldatetime
- smalldatetime_larger
  描述：返回较大的timestamp。
  参数：smalldatetime, smalldatetime
  返回值类型：smalldatetime
- smalldatetime_le
  描述：判断是否第一个参数小于等于第二个参数。
  参数：smalldatetime, smalldatetime
  返回值类型：boolean
- smalldatetime_lt
  描述：判断是否第一个参数小于第二个参数。
  参数：smalldatetime, smalldatetime
  返回值类型：boolean
- smalldatetime_ne
  描述：比较两个timestamp是否不相等。
  参数：smalldatetime, smalldatetime
  返回值类型：boolean
- smalldatetime_out
  描述：timestamp转换为外部形式。
  参数：smalldatetime
  返回值类型：cstring
- smalldatetime_send
  描述：timestamp转换为二进制格式。
  参数：smalldatetime
  返回值类型：bytea
- smalldatetime_smaller
  描述：返回较小的一个smalldatetime。
  参数：smalldatetime, smalldatetime
  返回值类型：smalldatetime
- smalldatetime_to_abstime
  描述：smalldatetime转换为abstime。

参数：smalldatetime
返回值类型：abstime

- smalldatetime_to_time
  描述：smalldatetime转换为time。
  参数：smalldatetime
  返回值类型：time without time zone
- smalldatetime_to_timestamp
  描述：smalldatetime转换为timestamp。
  参数：smalldatetime
  返回值类型：timestamp without time zone
- smalldatetime_to_timestamptz
  描述：smalldatetime转换为timestamptz。
  参数：smalldatetime
  返回值类型：timestamp with time zone
- smalldatetime_to_varchar2
  描述：smalldatetime转换为varchar2。
  参数：smalldatetime
  返回值类型：character varying

说明

获取当前时间有多种方式，请根据实际业务从场景选择合适的接口：

1. 以下接口按照当前事务的开始时刻返回值：
   ```
   CURRENT_DATE
   CURRENT_TIME
   CURRENT_TIME(precision)
   CURRENT_TIMESTAMP(precision)
   LOCALTIME
   LOCALTIMESTAMP
   LOCALTIME(precision)
   LOCALTIMESTAMP(precision)
   transaction_timestamp()
   now()
   ```
   其中CURRENT_TIME和CURRENT_TIMESTAMP(precision)传递带有时区的值；LOCALTIME和LOCALTIMESTAMP传递的值不带时区。CURRENT_TIME、LOCALTIME和LOCALTIMESTAMP可以指定精度参数，这会导致结果在秒字段中四舍五入到小数位数。如果没有精度参数，结果将被给予所能得到的全部精度。

   因为这些函数全部都按照当前事务的开始时刻返回结果，所以函数的值在事务运行的整个期间内都不改变。

   其中transaction_timestamp()等价于CURRENT_TIMESTAMP(precision)，表示当前语句所在事务的开启时间。now()等效于transaction_timestamp()。
2. 以下接口返回当前语句开始时间：
   ```
   statement_timestamp()
   ```
   statement_timestamp()返回当前语句的开始时刻（更准确的说是收到客户端最后一条命令的时间）。statement_timestamp()和transaction_timestamp()在一个事务的第一条命令期间返回值相同，但是在随后的命令中却不一定相同。
3. 以下接口返回函数被调用时的真实当前时间：
   ```
   clock_timestamp()
   timeofday()
   ```
   clock_timestamp()返回真正的当前时间，因此它的值甚至在同一条SQL命令中都会变化。timeofday()和clock_timestamp()相似，timeofday()也返回真实的当前时间，但是它的结果是一个格式化的text串，而不是timestamp with time zone值。

表7-32显示了可以用于截断日期和时间值的模板。

表 7-32 用于日期/时间截断的模式

类别	模式	描述
微秒	MICROSECON	截断日期/时间，精确到微秒（000000 - 999999）
	US	
	USEC	
	USECOND	
毫秒	MILLISECON	截断日期/时间，精确到毫秒（000 - 999）
	MS	
	MSEC	
	MSECOND	
秒	S	截断日期/时间，精确到秒（00 - 59）
	SEC	

类别	模式	描述
	SECOND	
分钟	M	截断日期/时间，精确到分钟（00 - 59）
	MI	
	MIN	
	MINUTE	
小时	H	截断日期/时间，精确到小时（00 - 23）
	HH	
	HOUR	
	HR	
天	D	截断日期/时间，精确到天（01-01 - 12-31）
	DAY	
	DD	
	DDD	
	J	
周	W	截断日期/时间，精确到周（本周的第一天）
	WEEK	
月	MM	截断日期/时间，精确到月（本月的第一天）
	MON	
	MONTH	
季度	Q	截断日期/时间，精确到季度（本季度的第一天）
	QTR	
	QUARTER	
年	Y	截断日期/时间，精确到年（本年的第一天）
	YEAR	
	YR	
	YYYY	
十年	DEC	截断日期/时间，精确到十年（本十年的第一天）
	DECADE	
世纪	C	截断日期/时间，精确到世纪（本世纪的第一天）
	CC	

类别	模式	描述
	CENT	
	CENTURY	
千年	MIL	截断日期/时间，精确到千年（本千年的第一天）
	MILLENNIA	
	MILLENNIUM	

表 7-33 用于时间截断和时间四舍五入的参数

类别	模式	描述
分钟	M	截断或四舍五入日期/时间，精确到分钟（00 - 59）
	MI	
	MIN	
	MINUTE	
小时	H	截断或四舍五入日期/时间，精确到小时（00 - 23）
	HH	
	HOUR	
	HR	
	HH12	
	HH24	
天	DD	截断或四舍五入日期/时间，精确到天（01-01 - 12-31）
	DDD	
	J	
ISO周	IW	截断或四舍五入日期/时间，精确到周（本周的第一天,第一天为周一）
周	DAY	截断或四舍五入日期/时间，精确到周（本周的第一天,第一天为周日）
	DY	
	D	
月周	W	截断或四舍五入日期/时间，精确到周（本周的第一天,第一天为本月第一天的周数）
年周	WW	截断或四舍五入日期/时间，精确到周（本周的第一天,第一天为本年第一天的周数）

类别	模式	描述
月	MM	截断或四舍五入日期/时间，精确到月（本月的第一天）
	MON	
	MONTH	
	RM	
季度	Q	截断或四舍五入日期/时间，精确到季度（本季度的第一天）
	QTR	
	QUARTER	
年	Y	截断或四舍五入日期/时间，精确到年（本年的第一天）
	YEAR	
	YR	
	YYYY	
	SYYYY	
	YYY	
	YY	
	SYEAR	
十年	DEC	截断或四舍五入日期/时间，精确到十年（本十年的第一天）
	DECADE	
世纪	C	截断或四舍五入日期/时间，精确到世纪（本世纪的第一天）
	CC	
	CENT	
	CENTURY	
	SCC	
千年	MIL	截断或四舍五入日期/时间，精确到千年（本千年的第一天）
	MILLENNIA	
	MILLENNIUM	

说明

表7-33中行为仅在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s1的情况下有效。

- timestamp_diff(text, timestamp, timestamp)

  描述：计算两个日期时间之间的差值，截取到参数text指定的精度。

返回值类型：int64

示例：

```
gaussdb=# SELECT timestamp_diff('year','2018-01-01','2020-04-01');
 timestamp_diff
----------------
              2
(1 row)
gaussdb=# SELECT timestamp_diff('month','2018-01-01','2020-04-01');
 timestamp_diff
----------------
             27
(1 row)
gaussdb=# SELECT timestamp_diff('quarter','2018-01-01','2020-04-01');
 timestamp_diff
----------------
              9
(1 row)
gaussdb=# SELECT timestamp_diff('week','2018-01-01','2020-04-01');
 timestamp_diff
----------------
            117
(1 row)
gaussdb=# SELECT timestamp_diff('day','2018-01-01','2020-04-01');
 timestamp_diff
----------------
            821
(1 row)
gaussdb=# SELECT timestamp_diff('hour','2018-01-01 10:10:10','2018-01-01 12:12:12');
 timestamp_diff
----------------
              2
(1 row)
gaussdb=# SELECT timestamp_diff('minute','2018-01-01 10:10:10','2018-01-01 12:12:12');
 timestamp_diff
----------------
            122
(1 row)
gaussdb=# SELECT timestamp_diff('second','2018-01-01 10:10:10','2018-01-01 10:12:12');
 timestamp_diff
----------------
            122
(1 row)
gaussdb=# SELECT timestamp_diff('microsecond','2018-01-01 10:10:10','2018-01-01 10:12:12');
 timestamp_diff
----------------
     122000000
(1 row)
```

TIMESTAMPDIFF

- **TIMESTAMPDIFF(***unit , timestamp_expr1, timestamp_expr2***)**

timestampdiff函数是计算两个日期时间之间(timestamp_expr2-timestamp_expr1)的差值，并以unit形式返回结果。timestamp_expr1，timestamp_expr2必须是一个timestamp、timestamptz、date类型的值表达式。unit表示的是两个日期差的单位。

等效于timestamp_diff(text, timestamp, timestamp)。

说明

该函数仅在GaussDB兼容MY类型时（即dbcompatibility = 'B'）有效，其他类型不支持该函数。

- year

  年份。

```
gaussdb=#  SELECT TIMESTAMPDIFF(YEAR, '2018-01-01', '2020-01-01');
 timestamp_diff
```



```
----------------
              2
(1 row)
```

- quarter

  季度。

```
gaussdb=#  SELECT TIMESTAMPDIFF(QUARTER, '2018-01-01', '2020-01-01');
 timestamp_diff
----------------
              8
(1 row)
```

- month

  月份。

```
gaussdb=#  SELECT TIMESTAMPDIFF(MONTH, '2018-01-01', '2020-01-01');
 timestamp_diff
----------------
             24
(1 row)
```

- week

  星期。

```
gaussdb=#  SELECT TIMESTAMPDIFF(WEEK, '2018-01-01', '2020-01-01');
 timestamp_diff
----------------
            104
(1 row)
```

- day

  天。

```
gaussdb=#  SELECT TIMESTAMPDIFF(DAY, '2018-01-01', '2020-01-01');
 timestamp_diff
----------------
            730
(1 row)
```

- hour

  小时。

```
gaussdb=#  SELECT TIMESTAMPDIFF(HOUR, '2020-01-01 10:10:10', '2020-01-01 11:11:11');
 timestamp_diff
----------------
              1
(1 row)
```

- minute

  分钟。

```
gaussdb=#  SELECT TIMESTAMPDIFF(MINUTE, '2020-01-01 10:10:10', '2020-01-01 11:11:11');
 timestamp_diff
----------------
             61
(1 row)
```

- second

  秒。

```
gaussdb=#  SELECT TIMESTAMPDIFF(SECOND, '2020-01-01 10:10:10', '2020-01-01 11:11:11');
 timestamp_diff
----------------
           3661
(1 row)
```

- microseconds

  秒域（包括小数部分）乘以1,000,000。


```
gaussdb=#  SELECT TIMESTAMPDIFF(MICROSECOND, '2020-01-01 10:10:10.000000', '2020-01-01
10:10:10.111111');
 timestamp_diff
----------------
         111111
(1 row)
```

- timestamp_expr含有时区

```
gaussdb=# SELECT TIMESTAMPDIFF(HOUR,'2020-05-01 10:10:10-01','2020-05-01 10:10:10-03');
 timestamp_diff
----------------
              2
(1 row)
```

EXTRACT

- **EXTRACT(***field* **FROM** *source***)**

extract函数从日期或时间的数值里抽取子域，比如年、小时等。source必须是一个timestamp、time或interval类型的值表达式（类型为date的表达式转换为timestamp，因此也可以用）。field是一个标识符或者字符串，它指定从源数据中抽取的域。extract函数返回类型为double precision的数值。field的取值范围如下所示。

- century

  世纪。

  第一个世纪从0001-01-01 00:00:00 AD开始。这个定义适用于所有使用阳历的国家。没有0世纪，直接从公元前1世纪到公元1世纪。

```
gaussdb=# SELECT EXTRACT(CENTURY FROM TIMESTAMP '2000-12-16 12:21:13');
 date_part
-----------
        20
(1 row)
```

- day
  - 如果source为timestamp，表示月份里的日期（1-31）。

    ```
    gaussdb=# SELECT EXTRACT(DAY FROM TIMESTAMP '2001-02-16 20:38:40');
     date_part
    -----------
            16
    (1 row)
    ```

  - 如果source为interval，表示天数。

    ```
    gaussdb=# SELECT EXTRACT(DAY FROM INTERVAL '40 days 1 minute');
     date_part
    -----------
            40
    (1 row)
    ```

- decade

  年份除以10。

```
gaussdb=# SELECT EXTRACT(DECADE FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
       200
(1 row)
```

- dow

  每周的星期几，星期天（0）到星期六（6）。

```
gaussdb=# SELECT EXTRACT(DOW FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
         5
(1 row)
```


- doy

  一年的第几天（1~365/366）。

```
gaussdb=# SELECT EXTRACT(DOY FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
        47
(1 row)
```

- epoch
  - 如果source为timestamp with time zone，表示自1970-01-01 00:00:00-00 UTC以来的秒数（结果可能是负数）；

    如果source为date和timestamp，表示自1970-01-01 00:00:00-00当地时间以来的秒数；

    如果source为interval，表示时间间隔的总秒数。

```
gaussdb=# SELECT EXTRACT(EPOCH FROM TIMESTAMP WITH TIME ZONE '2001-02-16
20:38:40.12-08');
  date_part
--------------
 982384720.12
(1 row)
gaussdb=# SELECT EXTRACT(EPOCH FROM INTERVAL '5 days 3 hours');
 date_part
-----------
    442800
(1 row)
```

  - 将epoch值转换为时间戳的方法。

```
gaussdb=# SELECT TIMESTAMP WITH TIME ZONE 'epoch' + 982384720.12 * INTERVAL '1
second' AS RESULT;
          result
---------------------------
 2001-02-17 12:38:40.12+08
(1 row)
```

- hour

  小时域（0-23）。

```
gaussdb=# SELECT EXTRACT(HOUR FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
        20
(1 row)
```

- isodow

  一周的第几天（1-7）。

  星期一为1，星期天为7。

  说明

  除了星期天外，都与dow相同。

```
gaussdb=# SELECT EXTRACT(ISODOW FROM TIMESTAMP '2001-02-18 20:38:40');
 date_part
-----------
         7
(1 row)
```

- isoyear

  日期中的ISO 8601标准年（不适用于间隔）。

  每个带有星期一开始的周中包含1月4日的ISO年，所以在年初的1月或12月下旬的ISO年可能会不同于阳历的年。详细信息请参见后续的**week**描述。

```
gaussdb=# SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-01');
 date_part
```



```
-----------
      2005
(1 row)
gaussdb=# SELECT EXTRACT(WEEK FROM TIMESTAMP '2006-01-01 00:00:40');
 date_part
-----------
        52
(1 row)
gaussdb=# SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-02');
 date_part
-----------
      2006
(1 row)
gaussdb=# SELECT EXTRACT(WEEK FROM TIMESTAMP '2006-01-02 00:00:40');
 date_part
-----------
         1
(1 row)
```

- microseconds

  秒域（包括小数部分）乘以1,000,000。

  ```
  gaussdb=# SELECT EXTRACT(MICROSECONDS FROM TIME '17:12:28.5');
   date_part
  -----------
    28500000
  (1 row)
  ```

- millennium

  千年。

  20世纪（19xx年）里面的年份在第二个千年里。第三个千年从2001年1月1日零时开始。

  ```
  gaussdb=# SELECT EXTRACT(MILLENNIUM FROM TIMESTAMP '2001-02-16 20:38:40');
   date_part
  -----------
         3
  (1 row)
  ```

- milliseconds

  秒域（包括小数部分）乘以1000。请注意它包括完整的秒。

  ```
  gaussdb=# SELECT EXTRACT(MILLISECONDS FROM TIME '17:12:28.5');
   date_part
  -----------
     28500
  (1 row)
  ```

- minute

  分钟域（0-59）。

  ```
  gaussdb=# SELECT EXTRACT(MINUTE FROM TIMESTAMP '2001-02-16 20:38:40');
   date_part
  -----------
        38
  (1 row)
  ```

- month

  如果source为timestamp，表示一年里的月份数（1-12）。

  ```
  gaussdb=# SELECT EXTRACT(MONTH FROM TIMESTAMP '2001-02-16 20:38:40');
   date_part
  -----------
         2
  (1 row)
  ```

  如果source为interval，表示月的数目，然后对12取模（0-11）。

  ```
  gaussdb=# SELECT EXTRACT(MONTH FROM INTERVAL '2 years 13 months');
   date_part
  ```

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

- quarter

  该天所在的该年的季度（1-4）。

```
gaussdb=# SELECT EXTRACT(QUARTER FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
         1
(1 row)
```

- second

  秒域，包括小数部分（0-59）。

```
gaussdb=# SELECT EXTRACT(SECOND FROM TIME '17:12:28.5');
 date_part
-----------
      28.5
(1 row)
```

- timezone

  与UTC的时区偏移量，单位为秒。正数对应UTC东边的时区，负数对应UTC西边的时区。

- timezone_hour

  时区偏移量的小时部分。

- timezone_minute

  时区偏移量的分钟部分。

- week

  该天在所在的年份里是第几周。ISO 8601定义一年的第一周包含该年的一月四日（ISO-8601 的周从星期一开始）。换句话说，一年的第一个星期四在第一周。

  在ISO定义里，一月的头几天可能是前一年的第52或者第53周，十二月的后几天可能是下一年第一周。比如，2006-01-01是2005年的第52周，而2006-01-02是2006年的第1周。建议**isoyear**字段和week一起使用以得到一致的结果。

```
gaussdb=# SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-01');
 date_part
-----------
      2005
(1 row)
gaussdb=# SELECT EXTRACT(WEEK FROM TIMESTAMP '2006-01-01 00:00:40');
 date_part
-----------
        52
(1 row)
gaussdb=# SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-02');
 date_part
-----------
      2006
(1 row)
gaussdb=# SELECT EXTRACT(WEEK FROM TIMESTAMP '2006-01-02 00:00:40');
 date_part
-----------
         1
(1 row)
```

- year

  年份域。

```
gaussdb=# SELECT EXTRACT(YEAR FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
```

```
    2001
(1 row)
```

date_part

date_part函数是在传统的Ingres函数的基础上制作的（该函数等效于SQL标准函数extract）：

- **date_part('*field*', *source*)**

这里的field参数必须是一个字符串，而不是一个名称。有效的field与extract一样，详细信息请参见**EXTRACT**。

示例：

```
gaussdb=# SELECT date_part('day', TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
        16
(1 row)
gaussdb=# SELECT date_part('hour', INTERVAL '4 hours 3 minutes');
 date_part
-----------
         4
(1 row)
```

表7-34显示了可以用于格式化日期和时间值的模板。

表 7-34 用于日期/时间格式化的模式

类别	模式	描述
小时	HH	一天的小时数（01-12）
	HH12	一天的小时数（01-12）
	HH24	一天的小时数（00-23）
分钟	MI	分钟（00-59）
秒	SS	秒（00-59）
	FF	微秒（000000-999999）
	FF1	微秒（0-9）
	FF2	微秒（00-99）
	FF3	微秒（000-999）
	FF4	微秒（0000-9999）
	FF5	微秒（00000-99999）
	FF6	微秒（000000-999999）
	SSSSS	午夜后的秒（0-86399）
上、下午	AM或A.M.	上午标识
	PM或P.M.	下午标识

类别	模式	描述
年	Y,YYY	带逗号的年（4和更多位）
	SYYYY	公元前四位年
	YYYY	年（4和更多位）
	YYY	年的后三位
	YY	年的后两位
	Y	年的最后一位
	IYYY	ISO年（4位或更多位）
	IYY	ISO年的最后三位
	IY	ISO年的最后两位
	I	ISO年的最后一位
	RR	年的后两位（可在21世纪存储20世纪的年份）
	RRRR	可接收4位年或两位年。若是两位，则和RR的返回值相同，若是四位，则和YYYY相同。
	• BC或B.C. • AD或A.D.	纪元标识。BC（公元前），AD（公元后）。
月	MONTH	全长大写月份名（空白填充为9字符）
	MON	大写缩写月份名（3字符）
	MM	月份数（01-12）
	RM	罗马数字的月份（I-XII； I=JAN）（大写）
天	DAY	全长大写日期名（空白填充为9字符）
	DY	缩写大写日期名（3字符）
	DDD	一年里的日（001-366）
	DD	一个月里的日（01-31）
	D	一周里的日（1-7； 周日是 1）
周	W	一个月里的周数（1-5）（第一周从该月第一天开始）
	WW	一年里的周数（1-53）（第一周从该年的第一天开始）
	IW	ISO一年里的周数（第一个星期四在第一周里）
世纪	CC	世纪（2位）（21 世纪从 2001-01-01 开始）
儒略日	J	儒略日（自公元前 4712 年 1 月 1 日来的天数）

类别	模式	描述
季度	Q	季度

在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s1的情况下新增用于格式化日期和时间值的模式：

表 7-35 新增用于格式化日期和时间值的模式

类别	模式	描述
世纪	SCC	世纪标识，公元前会显示-
年	SYYYY	返回数字型年，公元前会显示-
	RR	返回日期的2位年份
	RRRR	返回日期的4位年份
	YEAR	返回字符型年
	SYEAR	返回字符型年，公元前会显示-
日期格式	DL	返回指定长日期形式
	DS	返回指定短日期
	TS	返回指定时间格式
秒	FF7	微秒（0000000-9999990）
	FF8	微秒（00000000-99999900）
	FF9	微秒（000000000-999999000）

说明

上表中RR计算年的规则如下：

- 输入的两位年份在00~49之间：
  当前年份的后两位在00~49之间，返回值年份的前两位和当前年份的前两位相同；
  当前年份的后两位在50~99之间，返回值年份的前两位是当前年份的前两位加1。
- 输入的两位年份在50~99之间：
  当前年份的后两位在00~49之间，返回值年份的前两位是当前年份的前两位减1；
  当前年份的后两位在50~99之间，返回值年份的前两位和当前年份的前两位相同。

说明

在A兼容模式数据库中且参数a_format_version值为10c和a_format_dev_version值为s1的情况下：

- to_date, to_timestamp函数支持FX模式（输入和模式严格对应），支持X模式（小数点）。
- 输入模式不能出现超过一次，表示相同信息的模式不能同时出现。如SYYYY和BC不能同时出现。
- 模式大小写不敏感。
- 建议输入和模式之间使用分隔符，否则不保证行为与O完全一致

7.5.9 类型转换函数

类型转换函数

- cash_words(money)

  描述：类型转换函数，将money转换成text。

  示例：

```
gaussdb=# SELECT cash_words('1.23');
           cash_words
-----------------------------------
 One dollar and twenty three cents
(1 row)
```

- cast(x as y)

  描述：类型转换函数，将x转换成y指定的类型。

  示例：

```
gaussdb=# SELECT cast('22-oct-1997' as timestamp);
     timestamp
---------------------
 1997-10-22 00:00:00
(1 row)
```

- cast(x as y [DEFAULT z ON CONVERSION ERROR][,fmt])

  描述：类型转换函数，将x转换成y指定的类型。

  DEFAULT z ON CONVERSION ERROR：可选参数。当尝试将x转换成y指定的类型失败时，则将z转换成y指定的类型。

  fmt：可选参数。当y是以下数据类型时可以指定 fmt 参数：

  int1/int2/int4/int8/int16/float4/float8/numeric: 则可选参数 fmt 的用途与 to_number(expr [,fmt]) 函数相同。

  date/timestamp/timestamp with time zone: 则可选参数 fmt 的用途与 to_date(string [,fmt])/to_timestamp(string [,fmt]) /to_timestamp_tz(string [,fmt]) 函数相同。

  示例：

```
gaussdb=# SELECT cast('22-ocX-1997' as timestamp DEFAULT '22-oct-1997' ON CONVERSION
ERROR, 'DD-Mon-YYYY');
     timestamp
---------------------
 1997-10-22 00:00:00
(1 row)
```

  说明

  在参数a_format_version值为10c和a_format_dev_version值为s1的情况下，才支持 DEFAULT z ON CONVERSION ERROR 及 fmt 语法。

- hextoraw(raw)

  描述：将一个十六进制构成的字符串转换为raw类型。

  返回值类型：raw

  示例：

```
gaussdb=# SELECT hextoraw('7D');
 hextoraw
----------
 7D
(1 row)
```

- numtoday(numeric)

  描述：将数字类型的值转换为指定格式的时间戳。

  返回值类型：timestamp

  示例：

```
gaussdb=# SELECT numtoday(2);
 numtoday
----------
 2 days
(1 row)
```

- rawtohex(string)

  描述：将一个二进制构成的字符串转换为十六进制的字符串。

  结果为输入字符的ACSII码，以十六进制表示。

  返回值类型：varchar

  示例：

```
gaussdb=# SELECT rawtohex('1234567');
    rawtohex
----------------
 31323334353637
(1 row)
```

- rawtohex2(any)

  描述：将一个二进制构成的字符串转换为十六进制的字符串。

  结果为输入字符的ACSII码，以十六进制表示。支持'/'作为正常字符解析，不做转义处理。

  当输入为空时，输出也为空。

  返回值类型：text

  示例：

```
set a_format_version='10c';
SET
gaussdb=# set a_format_dev_version='s2';
SET
gaussdb=# select rawtohex2('12\n?$\123/2');
         rawtohex2
---------------------------
 31325C6E3F245C3132332F32
(1 row)
```

- bit2coding(text)

  描述：读取两个字节长度的字符，按小端存储逻辑保存，然后将每个字符解析成十六进制ASCII码值，最后将整体转换为十进制数。

  返回值类型：int

  示例：

```
gaussdb=# set a_format_version='10c';
SET
```



```
gaussdb=# set a_format_dev_version='s2';
SET
gaussdb=# select bit2coding('1234567890');
 bit2coding
------------
   12849
(1 row)
```

- bit4coding(text)

  描述：读取四个字节长度的字符，按小端存储逻辑保存，然后将每个字符解析成十六进制ASCII码值，最后将整体转换为十进制数。

  返回值类型：int

  示例：

```
gaussdb=# set a_format_version='10c';
SET
gaussdb=# set a_format_dev_version='s2';
SET
gaussdb=# select bit4coding('1234567890');
 bit4coding
------------
 875770417
(1 row)
```

- to_blob(raw)

  描述：将RAW类型转成BLOB类型。

  返回值类型：blob

  示例：

```
gaussdb=# SELECT to_blob('0AADD343CDBBD'::RAW(10));
    to_blob
----------------
 00AADD343CDBBD
(1 row)
```

  说明

  在参数a_format_version值为10c和a_format_dev_version值为s2的情况下，才支持to_blob函数。

- to_bigint(varchar)

  描述：将字符类型转换为bigint类型。

  返回值类型：bigint

  示例：

```
gaussdb=# SELECT to_bigint('123364545554455');
    to_bigint
----------------
 123364545554455
(1 row)
```

- to_binary_double(fmt)

  描述：将expr转换成float8类型的值。

  expr：支持number、float4、float8数据类型，以及可以隐式转换为数值类型的字符串。

  返回值类型：float8

  示例：

```
gaussdb=# SELECT to_binary_double('12345678');
 to_binary_double
------------------
         12345678
(1 row)
```


- to_binary_double(expr, fmt)

  描述：将expr经过指定的fmt匹配后转换成float8类型的数值。

  expr/fmt：支持char、nchar、varchar2、nvarchar2类型的字符串，expr还支持可以隐式转换为字符类型的数值类型。

  返回值类型：float8

  示例：

```
gaussdb=# SELECT to_binary_double('1,2,3', '9,9,9');
 to_binary_double
------------------
              123
(1 row)
```

- to_binary_double(expr default return_value on conversion error)

  描述：将expr转换成float8类型的值，若失败则返回默认值return_value。

  expr：支持number、float4、float8数据类型，以及可以隐式转换为字符串的数值类型。

  注意：expr为非数值、非字符类型会报错。

  返回值类型：float8

  示例：

```
gaussdb=# SELECT to_binary_double(1e2 default 12 on conversion error);
 to_binary_double
------------------
              100
(1 row)

gaussdb=# SELECT to_binary_double('aa' default 12 on conversion error);
 to_binary_double
------------------
               12
(1 row)
```

- to_binary_double(expr default return_value on conversion error, fmt)

  描述：将expr经过指定的fmt匹配后转换成float8类型的数值，若失败则返回默认值return_value。

  expr/fmt：支持char、nchar、varchar2、nvarchar2类型的字符串。expr还支持可以隐式转换为字符串类型的数值类型。

  返回类型：float8

  示例：

```
gaussdb=# SELECT to_binary_double('12-' default 10 on conversion error, '99S');
 to_binary_double
------------------
              -12
(1 row)

gaussdb=# SELECT to_binary_double('aa-' default 12 on conversion error, '99S');
 to_binary_double
------------------
               10
(1 row)
```

  说明

  在参数a_format_version值为10c和a_format_dev_version值为s2的情况下，才支持to_binary_double函数。

- to_binary_float(expr)

  描述：将expr转换成float4类型的值。

expr：支持number、float4、float8数据类型，以及可以隐式转换为数值类型的字符串。

返回值类型：float4

示例：

```
gaussdb=# SELECT to_binary_float('12345678');
 to_binary_float
------------------
     1.23457e+07
(1 row)
```

- to_binary_float(expr, fmt)

  描述：将expr经过指定的fmt匹配后转换成float4类型的数值。

  expr/fmt：支持char、nchar、varchar2、nvarchar2类型的字符串，expr还支持可以隐式转换为字符类型的数值类型。

  返回值类型：float4

  示例：

```
gaussdb=# SELECT to_binary_float('1,2,3', '9,9,9');
 to_binary_float
------------------
             123
(1 row)
```

- to_binary_float(expr default return_value on conversion error)

  描述：将expr转换成float4类型的值，若失败则返回默认值return_value。

  expr：支持number、float4、float8数据类型，以及可以隐式转换为字符串的数值类型。

  注意：expr为非数值、非字符类型会报错。

  返回值类型：float4

  示例：

```
gaussdb=# SELECT to_binary_float(1e2 default 12 on conversion error);
 to_binary_float
------------------
             100
(1 row)

gaussdb=# SELECT to_binary_float('aa' default 12 on conversion error);
 to_binary_float
------------------
              12
(1 row)
```

- to_binary_float(expr default return_value on conversion error, fmt)

  描述：将expr经过指定的fmt匹配后转换成float4类型的数值，若失败则返回默认值return_value。

  expr/fmt：支持char、nchar、varchar2、nvarchar2类型的字符串，expr还支持可以隐式转换为字符类型的数值类型。

  返回类型：float4

  示例：

```
gaussdb=# SELECT to_binary_float('12-' default 10 on conversion error, '99S');
 to_binary_float
------------------
             -12
(1 row)

gaussdb=# SELECT to_binary_float('aa-' default 12 on conversion error, '99S');
 to_binary_float
```



```
------------------
            10
(1 row)
```

说明

在参数a_format_version值为10c和a_format_dev_version值为s2的情况下，才支持to_binary_float函数。

- to_char(datetime/interval [, fmt])

  描述：将一个DATE、TIMESTAMP、TIMESTAMP WITH TIME ZONE或者TIMESTAMP WITH LOCAL TIME ZONE类型的DATETIME或者INTERVAL值按照fmt指定的格式转换为TEXT类型。

  - 可选参数fmt可以为以下几类：日期、时间、星期、季度和世纪。每类都可以有不同的模板，模板之间可以合理组合，常见的模板有：HH、MI、SS、YYYY、MM、DD详情见**表7-34**。
  - 模板可以有修饰词，常用的修饰词是FM，可以用来抑制前导的零或尾随的空白。

  返回值类型：text

  示例：

```
gaussdb=# SELECT to_char(current_timestamp,'HH12:MI:SS');
 to_char
----------
 10:19:26
(1 row)
gaussdb=# SELECT to_char(current_timestamp,'FMHH12:FMMI:FMSS');
 to_char
----------
 10:19:46
(1 row)
```

- to_char(double precision/real, text)

  描述：将浮点类型的值转换为指定格式的字符串。

  返回值类型：text

  示例：

```
gaussdb=# SELECT to_char(125.8::real, '999D99');
 to_char
---------
  125.80
(1 row)
```

- to_char(numeric/smallint/integer/bigint/double precision/real[, fmt])

  描述：将一个整型或者浮点类型的值转换为指定格式的字符串。

  - 可选参数fmt可以为以下几类：十进制字符、“分组”符、正负号和货币符号，每类都可以有不同的模板，模板之间可以合理组合，常见的模板有：9、0、,（千分隔符）、.（小数点），详情见**表7-36**。
  - 模板可以有类似FM的修饰词，但FM不抑制由模板0指定而输出的0。
  - 要将整型类型的值转换成对应16进制值的字符串，使用模板X或x。

  返回值类型：text

  示例：

```
gaussdb=# SELECT to_char(1485,'9,999');
 to_char
---------
  1,485
(1 row)
gaussdb=# SELECT to_char( 1148.5,'9,999.999');
  to_char
```



```
------------
  1,148.500
(1 row)
gaussdb=# SELECT to_char(148.5,'990999.909');
   to_char
-------------
    0148.500
(1 row)
gaussdb=# SELECT to_char(123,'XXX');
 to_char
---------
   7B
(1 row)
```

表 7-36 number 类型 fmt 参数

模式	描述
, (comma)	分组（千）分隔符
. (period)	小数点
$	指定位置输出$
0	带前导零的值
9	带有指定数值位数的值
B	当整数部分是0时返回空格
C	货币符号（使用区域设置）
D	小数点（使用区域设置）
EEEE	科学计数法
G	分组分隔符（使用区域设置）
L	货币符号（使用区域设置）
MI	在指明的位置的负号（如果数字 < 0）
PR	尖括号内负值
RN	罗马数字（输入在 1 和 3999 之间）
S	带符号的数值（使用区域设置）
TM	标准数值与科学计数法
TM9	标准数值与科学计数法
TME	标准数值与科学计数法
U	货币符号（使用区域设置）
V	移动指定位（小数）
PL	在指明的位置的正号（如果数字 > 0）
SG	在指明的位置的正/负号
TH或th	序数后缀

说明

仅在a_format_version和a_format_dev_version参数为：平台版本10c，版本s1 的情况下，支持$、C、TM、TM9、TME、U格式。同时在该参数下，不支持TH、PL、SG格式的fmt。

- to_char(interval, text)

  描述：将时间间隔类型的值转换为指定格式的字符串。

  返回值类型：text

  示例：

```
gaussdb=# SELECT to_char(interval '15h 2m 12s', 'HH24:MI:SS');
 to_char
----------
 15:02:12
(1 row)
```

- to_char(integer, text)

  描述：将整数类型的值转换为指定格式的字符串。

  返回值类型：text

  示例：

```
gaussdb=# SELECT to_char(125, '999');
 to_char
---------
  125
(1 row)
```

- to_char(set)

  描述：将SET类型的值转换为字符串。

  返回值：text

  示例：

```
-- site 是employee的SET类型的字段
gaussdb=# select to_char(site) from employee;
     to_char
-----------------
 beijing,nanjing
 beijing,wuhan
(2 rows)
```

- to_char(numeric, text)

  描述：将数字类型的值转换为指定格式的字符串。

  返回值类型：text

  示例：

```
gaussdb=# SELECT to_char(-125.8, '999D99S');
 to_char
---------
 125.80-
(1 row)
```

- to_char(string)

  描述：将CHAR、VARCHAR、VARCHAR2、CLOB类型转换为TEXT类型。

  如使用该函数对CLOB类型进行转换，且待转换CLOB类型的值超出目标类型的范围，则返回错误。

  返回值类型：text

示例：

```
gaussdb=# SELECT to_char('01110');
 to_char
---------
 01110
(1 row)
```

- to_char(timestamp, text)

  描述：将时间戳类型的值转换为指定格式的字符串。

  返回值类型：text

  示例：

  ```
  gaussdb=# SELECT to_char(current_timestamp, 'HH12:MI:SS');
   to_char
  ----------
   10:55:59
  (1 row)
  ```

  说明

  - to_char函数对于错误的fmt会原样输出，如fmt为FF10，会匹配到FF1进行格式化输出，然后原样输出0。
  - 在参数a_format_version值为10c和a_format_dev_version值为s1的情况下，to_char函数会对错误的fmt进行报错。

- to_nchar (datetime/interval [, fmt])

  描述：将一个DATE、TIMESTAMP、TIMESTAMP WITH TIME ZONE或者TIMESTAMP WITH LOCAL TIME ZONE类型的DATETIME或者INTERVAL值按照fmt指定的格式转换为text类型。

  - 可选参数fmt可以为以下几类：日期、时间、星期、季度和世纪。每类都可以有不同的模板，模板之间可以合理组合，常见的模板有：HH、MI、SS、YYYY、MM、DD详情见**表7-36**。
  - 模板可以有修饰词，常用的修饰词是FM，可以用来抑制前导的零或尾随的空白。

  返回值类型：text

  示例：

  ```
  gaussdb=# SELECT to_nchar(current_timestamp,'HH12:MI:SS');
   to_nchar
  ----------
   10:19:26
  (1 row)
  gaussdb=# SELECT to_nchar(current_timestamp,'FMHH12:FMMI:FMSS');
   to_nchar
  ----------
   10:19:46
  (1 row)
  ```

- to_nchar(double precision/real, text)

  描述：将浮点类型的值转换为指定格式的字符串。

  返回值类型：text

  示例：

  ```
  gaussdb=# SELECT to_nchar(125.8::real, '999D99');
   to_nchar
  ----------
    125.80
  (1 row)
  ```

- to_nchar (numeric/smallint/integer/bigint/double precision/real[, fmt])

  描述：将一个整型或者浮点类型的值转换为指定格式的字符串。

- 可选参数fmt可以为以下几类：十进制字符、“分组”符、正负号和货币符号，每类都可以有不同的模板，模板之间可以合理组合，常见的模板有：9、0、,（千分隔符）、.（小数点），详情见**表7-36**。
- 模板可以有类似FM的修饰词，但FM不抑制由模板0指定而输出的0。
- 要将整型类型的值转换成对应16进制值的字符串，使用模板X或x。

返回值类型：text

示例：

```
gaussdb=# SELECT to_nchar(1485,'9,999');
 to_nchar
----------
  1,485
(1 row)
gaussdb=# SELECT to_nchar( 1148.5,'9,999.999');
  to_nchar
------------
  1,148.500
(1 row)
gaussdb=# SELECT to_nchar(148.5,'990999.909');
   to_nchar
--------------
    0148.500
(1 row)
gaussdb=# SELECT to_nchar(123,'XXX');
 to_nchar
----------
   7B
(1 row)
```

说明

此函数仅在A兼容模式（即sql_compatibility='A'）且参数a_format_version值为10c和a_format_dev_version值为s2的情况下生效。开启参数时支持$、C、TM、TM9、TME、U格式。同时在该参数下，不支持TH、PL、SG格式的fmt。

- to_nchar(interval, text)

  描述：将时间间隔类型的值转换为指定格式的字符串。

  返回值类型：text

  示例：

  ```
  gaussdb=# SELECT to_nchar(interval '15h 2m 12s', 'HH24:MI:SS');
   to_nchar
  ----------
   15:02:12
  (1 row)
  ```

- to_nchar(integer, text)

  描述：将整数类型的值转换为指定格式的字符串。

  返回值类型：text

  示例：

  ```
  gaussdb=# SELECT to_nchar(125, '999');
   to_nchar
  ----------
   125
  (1 row)
  ```

- to_nchar(set)

  描述：将SET类型的值转换为字符串。分布式暂不支持SET数据类型。

  返回值：text

- to_nchar(numeric, text)

描述：将数字类型的值转换为指定格式的字符串。

返回值类型：text

示例：

```
gaussdb=# SELECT to_nchar(-125.8, '999D99S');
 to_nchar
----------
 125.80-
(1 row)
```

- to_nchar (string)

  描述：将CHAR、VARCHAR、VARCHAR2、CLOB类型转换为TEXT类型。

  如使用该函数对CLOB类型进行转换，且待转换CLOB类型的值超出目标类型的范围，则返回错误。

  返回值类型：text

  示例：

```
gaussdb=# SELECT to_nchar('01110');
 to_nchar
----------
 01110
(1 row)
```

- to_nchar(timestamp, text)

  描述：将时间戳类型的值转换为指定格式的字符串。

  返回值类型：text

  示例：

```
gaussdb=# SELECT to_nchar(current_timestamp, 'HH12:MI:SS');
 to_nchar
-----------
 10:55:59
(1 row)
```

- to_clob(char/nchar/varchar/varchar2/nvarchar2/text/raw)

  描述：将RAW类型或者文本字符集类型CHAR、NCHAR、VARCHAR、VARCHAR2、NVARCHAR2、TEXT转成CLOB类型。

  返回值类型：clob

  示例：

```
gaussdb=# SELECT to_clob('ABCDEF'::RAW(10));
 to_clob
---------
 ABCDEF
(1 row)
gaussdb=# SELECT to_clob('hello111'::CHAR(15));
 to_clob
----------
 hello111
(1 row)
gaussdb=# SELECT to_clob('gauss123'::NCHAR(10));
 to_clob
----------
 gauss123
(1 row)
gaussdb=# SELECT to_clob('gauss234'::VARCHAR(10));
 to_clob
----------
 gauss234
(1 row)
gaussdb=# SELECT to_clob('gauss345'::VARCHAR2(10));
 to_clob
----------
```



```
 gauss345
(1 row)
gaussdb=# SELECT to_clob('gauss456'::NVARCHAR2(10));
 to_clob
----------
 gauss456
(1 row)
gaussdb=# SELECT to_clob('World222!'::TEXT);
  to_clob
-----------
 World222!
(1 row)
```

- to_date(text)

  描述：将文本类型的值转换为指定格式的时间戳。目前只支持两类格式。

  - 格式一：无分隔符日期，如20150814，需要包括完整的年月日。
  - 格式二：带分隔符日期，如2014-08-14，分隔符可以是单个任意非数字字符。

  返回值类型：timestamp without time zone

  示例：

  ```
  gaussdb=# SELECT to_date('2015-08-14');
        to_date
  ---------------------
   2015-08-14 00:00:00
  (1 row)
  ```

  说明

  用例执行环境：参数a_format_version值为10c、a_format_dev_version值为s1、nls_timestamp_format值为YYYY-MM-DD HH24:MI:SS。

- to_date(text, text)

  描述：将字符串类型的值转换为指定格式的日期。

  返回值类型：timestamp without time zone

  示例：

  ```
  gaussdb=# SELECT to_date('05 Dec 2000', 'DD Mon YYYY');
        to_date
  ---------------------
   2000-12-05 00:00:00
  (1 row)
  ```

- to_date(text [DEFAULT return_value ON CONVERSION ERROR [, fmt]])

  描述：将字符串text按fmt指定的格式转换成DATE类型的值。不指定fmt时，在参数a_format_version值为10c和a_format_dev_version值为s1的情况下，按参数nls_timestamp_format所指定的格式转换；否则按照固定fmt = 'yyyy-mm-dd hh24-mi-ss'进行转换。

  - text：任何计算结果为CHAR、VARCHAR2、NCHAR、NVARCHAR2、TEXT类型字符串的表达式。输入null，返回null。
  - DEFAULT return_value ON CONVERSION ERROR：可选参数。用于当text转换DATE类型失败时指定返回值return_value。return_value可以是表达式或者绑定的变量，必须可以转换为CHAR、VARCHAR2、NCHAR、NVARCHAR2、TEXT类型或者是null。return_value转换为DATE类型数据的方法与text转换为DATE类型数据相同，如果return_value转换为DATE类型数据失败，该函数报错。
  - fmt：可选参数。指定text的日期时间模型格式。缺省时，text必须符合默认的日期格式。fmt指定为J时，text必须是整数。

  返回值类型：timestamp without time zone

示例：

```
gaussdb=# SELECT to_date('2015-08-14');
      to_date
---------------------
 2015-08-14 00:00:00
(1 row)
gaussdb=# SELECT to_date('05 Dec 2000', 'DD Mon YYYY');
      to_date
---------------------
 2000-12-05 00:00:00
(1 row)
gaussdb=# SET a_format_version='10c';
SET
gaussdb=# SET a_format_dev_version='s1';
SET
gaussdb=# SHOW nls_timestamp_format;
   nls_timestamp_format
----------------------------
 DD-Mon-YYYY HH:MI:SS.FF AM
(1 row)
gaussdb=# SELECT to_date('12-jan-2022' default '12-apr-2022' on conversion error);
      to_date
---------------------
 2022-01-12 00:00:00
(1 row)
gaussdb=# SELECT to_date('12-ja-2022' default '12-apr-2022' on conversion error);
      to_date
---------------------
 2022-04-12 00:00:00
(1 row)
gaussdb=# SELECT to_date('2022-12-12' default '2022-01-01' on conversion error, 'yyyy-mm-dd');
      to_date
---------------------
 2022-12-12 00:00:00
(1 row)
```

⚠ 注意

- 在参数a_format_version值为10c和a_format_dev_version值为s1的情况下，才支持DEFAULT return_value ON CONVERSION ERROR语法。
- 在参数a_format_version值为10c和a_format_dev_version值为s1的情况下，年份的输入超过9999时，系统可能不报错。如to_date('99999-12-12', 'yyyy-mm-dd hh24:mi:ss')结果为9999-09-12 12:00:00。年输入超过9999时，超过4位后面的数字会解析为下一个fmt，to_timestamp同样有此限制。

- to_number (expr [, fmt])

  描述：将expr按指定格式转换为一个NUMBER类型的值。

  类型转换格式请参考**表7-37**。

  转换十六进制字符串为十进制数字时，最多支持16个字节的十六进制字符串转换为无符号数。

  转换十六进制字符串为十进制数字时，格式字符串中不允许出现除'x'或'X'以外的其他字符，否则报错。

  返回值类型：number

  示例：

```
gaussdb=# SELECT to_number('12,454.8-', '99G999D9S');
 to_number
-----------
  -12454.8
(1 row)
```


- to_number(text, text)

  描述：将字符串类型的值转换为指定格式的数字。

  返回值类型：numeric

  示例：

```
gaussdb=# SELECT to_number('12,454.8-', '99G999D9S');
 to_number
-----------
  -12454.8
(1 row)
```

- to_number(expr [DEFAULT return_value ON CONVERSION ERROR [, fmt]])

  描述：将字符串expr根据指定fmt格式转换成numeric类型的值。不指定fmt时，text需要为能直接转换成numeric的字符串, 例: '123' , '1e2'。

  类型转换格式参考：**表7-38**。

  - expr：支持的类型有CHAR、VARCHAR2、NCHAR、NVARCHAR2、TEXT、INT、FLOAT等可以转换成字符串类型的表达式。输入null，返回null。
  - DEFAULT return_value ON CONVERSION ERROR：可选参数。用于当expr转换numeric类型失败时指定返回值return_value。return_value同expr一样可以是任何能转换成字符串的类型。return_value的转换方式和expr相同也是根据fmt格式进行转换，会先校验return_value是否会转换失败，如果失败则该函数报错。
  - fmt：可选参数。指定expr的转换格式。

    任意入参为NULL，则返回NULL。

  返回值类型：numeric

  示例：

```
gaussdb=# SET a_format_version='10c';
gaussdb=# SET a_format_dev_version='s1';

gaussdb=# SELECT to_number('1e2');
 to_number
-----------
       100
(1 row)

gaussdb=# SELECT to_number('123.456');
 to_number
-----------
   123.456
(1 row)

gaussdb=# SELECT to_number('123', '999');
 to_number
-----------
       123
(1 row)

gaussdb=# SELECT to_number('123-', '999MI');
 to_number
-----------
      -123
(1 row)

gaussdb=# SELECT to_number('123' default '456-' on conversion error, '999MI');
 to_number
-----------
      -456
(1 row)
```


说明

在参数a_format_version值为10c和a_format_dev_version值为s1的情况下，才支持DEFAULT return_value ON CONVERSION ERROR语法。

- to_timestamp(double precision)

  描述：把UNIX纪元转换成时间戳。

  返回值类型：timestamp with time zone

  示例：

```
gaussdb=# SELECT to_timestamp(1284352323);
      to_timestamp
------------------------
 2010-09-13 12:32:03+08
(1 row)
```

- to_timestamp(string [,fmt])

  描述：将字符串string按fmt指定的格式转换成时间戳类型的值。不指定fmt时，按参数nls_timestamp_format所指定的格式转换。

  GaussDB的to_timestamp中，

  – 如果输入的年份YYYY=0，系统报错。
  – 如果输入的年份YYYY<0，在fmt中指定SYYYY，则正确输出公元前绝对值n的年份。

  fmt中出现的字符必须与日期/时间格式化的模式相匹配，否则报错。

  返回值类型：timestamp without time zone

  示例：

```
gaussdb=# SHOW nls_timestamp_format;
    nls_timestamp_format
-----------------------------
 DD-Mon-YYYY HH:MI:SS.FF AM
(1 row)

gaussdb=# SELECT to_timestamp('12-sep-2014');
    to_timestamp
---------------------
 2014-09-12 00:00:00
(1 row)
gaussdb=# SELECT to_timestamp('12-Sep-10 14:10:10.123000','DD-Mon-YY HH24:MI:SS.FF');
      to_timestamp
-------------------------
 2010-09-12 14:10:10.123
(1 row)
gaussdb=# SELECT to_timestamp('-1','SYYYY');
      to_timestamp
------------------------
 0001-01-01 00:00:00 BC
(1 row)
gaussdb=# SELECT to_timestamp('98','RR');
    to_timestamp
---------------------
 1998-01-01 00:00:00
(1 row)
gaussdb=# SELECT to_timestamp('01','RR');
    to_timestamp
---------------------
 2001-01-01 00:00:00
(1 row)
```


说明

1. 在参数a_format_version值为10c和a_format_dev_version值为s1的情况下fmt支持FF[7-9]，在FF[7-9]的情况下允许转换string中的对应位置长度小于等于FF紧跟的数字，但最终转换结果长度最大保留6位。
2. 不支持current_timestamp函数返回结果作为string参数。

- to_timestamp(text [DEFAULT return_value ON CONVERSION ERROR [, fmt]])

  描述：将字符串text按fmt指定的格式转换成DATE类型的值。不指定fmt时，在a_format_version和a_format_dev_version参数为：平台版本10c，版本s1 的情况下，按参数nls_timestamp_format所指定的格式转换；否则按照固定fmt = 'yyyy-mm-dd hh24-mi-ss'进行转换。

  – text：任何计算结果为CHAR,VARCHAR2,NCHAR,NVARCHAR2,TEXT类型字符串的表达式。输入null，返回null。
  – DEFAULT return_value ON CONVERSION ERROR：可选参数。用于当text转换DATE类型失败时指定返回值return_value。return_value可以是表达式或者绑定的变量，需可以转换为CHAR、VARCHAR2、NCHAR、NVARCHAR2、TEXT类型或者是null。return_value转换为timestamp类型数据的方法与text转换为timestamp类型数据相同，如果return_value转换为timestamp类型数据失败，该函数报错。
  – fmt：可选参数。指定text的日期时间模型格式。缺省时，text必须符合默认的日期格式。fmt指定为J时，text必须是整数。

  返回值类型：timestamp without time zone

  示例：

```
gaussdb=# SET a_format_version='10c';
SET
gaussdb=# SET a_format_dev_version='s1';
SET
gaussdb=# SELECT to_timestamp('11-Sep-11' DEFAULT '12-Sep-10 14:10:10.123000' ON
CONVERSION ERROR,'DD-Mon-YY HH24:MI:SS.FF');
    to_timestamp
---------------------
 2011-09-11 00:00:00
(1 row)
gaussdb=# SELECT to_timestamp('12-Sep-10 14:10:10.123000','DD-Mon-YY HH24:MI:SSXFF');
      to_timestamp
-------------------------
 2010-09-12 14:10:10.123
(1 row)
```

  说明

  在参数a_format_version值为10c和a_format_dev_version值为s1的情况下，才支持DEFAULT return_value ON CONVERSION ERROR语法。

- to_timestamp(text, text)

  描述：将字符串类型的值转换为指定格式的时间戳。

  返回值类型：timestamp

  示例：

```
gaussdb=# SELECT to_timestamp('05 Dec 2000', 'DD Mon YYYY');
    to_timestamp
---------------------
 2000-12-05 00:00:00
(1 row)
```

- to_timestamp_tz(text [,text])

  描述：将字符串类型的值转换为指定格式带时区的时间戳。

返回值类型：timestamp with time zone

示例：

```
gaussdb=# SELECT to_timestamp_tz('05 Dec 2001', 'DD Mon YYYY');
    to_timestamp_tz
--------------------------
 2001-12-05 00:00:00+08:00
(1 row)
```

说明

此函数在参数a_format_version值为10c和a_format_dev_version值为s1的情况下有效。

- to_timestamp_tz(string [DEFAULT return_value ON CONVERSION ERROR] [,fmt])

  描述：将字符串string按fmt指定的格式转换成带时区时间戳类型的值。不指定fmt时，按参数nls_timestamp_tz_format所指定的格式转换。

  DEFAULT return_value ON CONVERSION ERROR：可选参数。当string转换成timestamp with time zone类型失败时，则将return_value转换timestamp with time zone类型。

  fmt：可选参数。指定string的日期时间模型格式。同to_timestamp函数。

  返回值类型：timestamp with time zone

  示例：

```
gaussdb=# SELECT to_timestamp_tz('05 DeX 2000' DEFAULT '05 Dec 2001' ON CONVERSION ERROR,
'DD Mon YYYY');
    to_timestamp_tz
--------------------------
 2001-12-05 00:00:00+08:00
(1 row)
```

说明

此函数在参数a_format_version值为10c和a_format_dev_version值为s1的情况下有效。

- to_dsinterval(text)

  描述：将字符转换为interval类型。支持SQL兼容格式与ISO格式。

  返回值类型：interval

  示例：

```
gaussdb=# SELECT to_dsinterval('12 1:2:3.456');
    to_dsinterval
----------------------
 12 days 01:02:03.456
(1 row)

gaussdb=# SELECT to_dsinterval('P3DT4H5M6S');
  to_dsinterval
-----------------
 3 days 04:05:06
(1 row)
```

说明

此函数在参数a_format_version值为10c和a_format_dev_version值为s2的情况下有效。

- to_yminterval(text)

  描述：将字符转换为interval类型。支持SQL兼容格式与ISO格式。

  返回值类型：interval

  示例：


```
gaussdb=# SELECT to_yminterval('1-1');
 to_yminterval
-----------------
 1 year 1 mon
(1 row)

gaussdb=# SELECT to_yminterval('P13Y3M4DT4H2M5S');
 to_yminterval
-----------------
 13 years 3 mons
(1 row)
```

说明

此函数在参数a_format_version值为10c和a_format_dev_version值为s2的情况下有效。

表 7-37 数值格式化的模板模式

模式	描述
9	带有指定数值位数的值。
0	带前导零的值。
.（句点）	小数点。
,（逗号）	分组（千）分隔符。
PR	尖括号内负值。
S	带符号的数值（使用区域设置）。
L	货币符号（使用区域设置）。
D	小数点（使用区域设置）。
G	分组分隔符（使用区域设置）。
MI	在指明的位置的负号（如果数字 < 0）。
PL	在指明的位置的正号（如果数字 > 0）。
SG	在指明的位置的正/负号。
RN	罗马数字（输入在 1 和 3999 之间）。
TH或th	序数后缀。
V	移动指定位（小数）。
x或X	16进制转换10进制标识符。

表 7-38 to_number 数值格式化的模板模式

模式	描述
9	匹配一个数字，9的数量可以大于等于expr中对应位置的数字的数量。

模式	描述
0	严格匹配一个数字，0的数量要等于expr中数字的数量。
5	匹配一个0或5的数字。
.（句点）	指定位置的小数点。
,（逗号）	指定位置的分组（千）分隔符，可以在fmt中指定多个逗号。
B	前导空白。
PR	<尖括号>中对应负值。空白对应正值。
S	前导减号（-）的负值，返回带有前导加号（+）的正值。返回带有尾随减号（-）的负值，返回带有尾随加号（+）的正值。
MI	尾随-号的负值，尾随空白的正值。
$	前导美元符号。
L	本地货币符号。
C	指定位置ISO货币符号。
U	双币符号。
D	小数点（使用区域设置）。
G	分组分隔符（ISO标准），可以在fmt中指定多个逗号。
RN / rn	罗马数字（输入在 1 和 3999 之间），to_number不支持该fmt。
V	to_number不支持该fmt。
X / x	十六进制与十进制转换。
TM	to_number不支持。
FM	在fmt中的最开头才能使用，无任何效果。
EEEE	按照科学计数法模型转换。

说明

在参数a_format_version值为10c和a_format_dev_version值为s1的情况下，fmt功能参考该表格，否则参考上表格。其中该表中国际化ISO的fmt功能受参数LC_MONETARY和LC_NUMERIC影响。

- cast_varchar2_to_raw_for_histogram(varchar2)
  描述：将varchar2类型转换为raw类型输出。
  返回值类型：raw
- abstime_text(
  描述：将abstime类型转为text类型输出。

参数：abstime
返回值类型：text

- abstime_to_smalldatetime
  描述：将abstime类型转为smalldatatime类型。
  参数：abstime
  返回值类型：smalldatetime
- bigint_tid
  描述：将bigint转为tid。
  参数：bigint
  返回值类型：tid
- bool_int1
  描述：将boolean转为int1。
  参数：boolean
  返回值类型：tinyint
- bool_int2
  描述：将boolean转为int2。
  参数：boolean
  返回值类型：smallint
- bool_int8
  描述：将boolean转为int8。
  参数：boolean
  返回值类型：bigint
- bpchar_date
  描述：将字符串转为日期。
  参数：character
  返回值类型：date
- bpchar_float4
  描述：将字符串转为float4。
  参数：character
  返回值类型：real
- bpchar_float8
  描述：将字符串转为float8。
  参数：character
  返回值类型：double precision
- bpchar_int4
  描述：将字符串转为int4。
  参数：character
  返回值类型：integer
- bpchar_int8
  描述：将字符串转为int8。

参数：character
返回值类型：bigint

- bpchar_numeric
  描述：将字符串转为numeric。
  参数：character
  返回值类型：numeric
- bpchar_timestamp
  描述：将字符串转为时间戳。
  参数：character
  返回值类型：timestamp without time zone
- bpchar_to_smalldatetime
  描述：将字符串转为smalldatetime。
  参数：character
  返回值类型：smalldatetime
- complex_array_in
  描述：将外部complex_array类型转化为内部anyarray数组类型。
  参数：cstring, oid, int2vector
  返回值类型：anyarray
- date_bpchar
  描述：将date类型转换为bpchar类型。
  参数：date
  返回值类型：character
- date_text
  描述：将date类型转换为text类型。
  参数：date
  返回值类型：text
- date_varchar
  描述：将date类型转换为varchar类型。
  参数：date
  返回值类型：character varying
- f4toi1
  描述：把float4类型强转为uint8类型。
  参数：real
  返回值类型：tinyint
- f8toi1
  描述：把float8类型强转为uint8类型。
  参数：double precision
  返回值类型：tinyint
- float4_bpchar
  描述：float4转换为bpchar。

参数：real
返回值类型：character

- float4_text
  描述：float4转换为text。
  参数：real
  返回值类型：text
- float4_varchar
  描述：float4转换为varchar。
  参数：real
  返回值类型：character varying
- float8_bpchar
  描述：float8转换为bpchar。
  参数：double precision
  返回值类型：character
- float8_interval
  描述：float8转换为interval。
  参数：double precision
  返回值类型：interval
- float8_text
  描述：float8转换为text。
  参数：double precision
  返回值类型：text
- float8_varchar
  描述：float8转换为varchar。
  参数：double precision
  返回值类型：character varying
- i1tof4
  描述：uint8转换为float4。
  参数：tinyint
  返回值类型：real
- i1tof8
  描述：uint8转换为float8。
  参数：tinyint
  返回值类型：double precision
- i1toi2
  描述：uint8转换为int16。
  参数：tinyint
  返回值类型：smallint
- i1toi4
  描述：uint8转换为int32。

参数：tinyint
返回值类型：integer

- i1toi8
  描述：uint8转换为int64。
  参数：tinyint
  返回值类型：bigint
- i2toi1
  描述：int16转换为uint8。
  参数：smallint
  返回值类型：tinyint
- i4toi1
  描述：int32转换为uint8。
  参数：integer
  返回值类型：tinyint
- i8toi1
  描述：int64转换为uint8。
  参数：bigint
  返回值类型：tinyint
- int1_avg_accum
  描述：将第二个uint8类型参数，加入到第一个参数中，一个参数为bigint类型数组。
  参数：bigint[], tinyint
  返回值类型：bigint[]
- int1_bool
  描述：uint8转换为boolean。
  参数：tinyint
  返回值类型：boolean
- int1_bpchar
  描述：uint8转换为bpchar。
  参数：tinyint
  返回值类型：character
- int1_mul_cash
  描述：返回一个int8类型参数和一个cash类型参数的乘积，返回值为cash类型。
  参数：tinyint, money
  返回值类型：money
- int1_numeric
  描述：uint8转换为numeric。
  参数：tinyint
  返回值类型：numeric
- int1_nvarchar2
  描述：uint8转换为nvarchar2。

参数：tinyint
返回值类型：nvarchar2

- int1_text
  描述：uint8转换为text。
  参数：tinyint
  返回值类型：text
- int1_varchar
  描述：uint8转换为varchar。
  参数：tinyint
  返回值类型：character varying
- int1in
  描述：字符串转化为无符号一字节整数。
  参数：cstring
  返回值类型：tinyint
- int1out
  描述：无符号一字节整数转化为字符串。
  参数：tinyint
  返回值类型：cstring
- int1up
  描述：输入整数转化为无符号一字节整数。
  参数：tinyint
  返回值类型：tinyint
- int2_bool
  描述：将有符号二字节整数转化为bool型。
  参数：smallint
  返回值类型：boolean
- int2_bpchar
  描述：将有符号二字节整数转化为BpChar。
  参数：smallint
  返回值类型：character
- int2_text
  描述：有符号二字节整数转化为text类型。
  参数：smallint
  返回值类型：text
- int2_varchar
  描述：有符号二字节整数转化为varchar类型。
  参数：smallint
  返回值类型：character varying
- int4_bpchar
  描述：有符号四字节整数转化为bpchar。

参数：integer
返回值类型：character

- int4_text
  描述：有符号四字节整数转化为text类型。
  参数：integer
  返回值类型：text
- int4_varchar
  描述：有符号四字节整数转化为varchar。
  参数：integer
  返回值类型：character varying
- int8_bool
  描述：有符号八字节整数转化为bool。
  参数：bigint
  返回值类型：boolean
- int8_bpchar
  描述：有符号八字节整数转化为bpchar。
  参数：bigint
  返回值类型：character
- int8_text
  描述：有符号八字节整数转化为text类型。
  参数：bigint
  返回值类型：text
- int8_varchar
  描述：有符号八字节整数转化为varchar。
  参数：bigint
  返回值类型：character varying
- intervaltonum
  描述：将内部数据类型日期转化为numeric类型。
  参数：interval
  返回值类型：numeric
- numeric_bpchar
  描述：numeric转化为bpchar。
  参数：numeric
  返回值类型：character
- numeric_int1
  描述：numeric转化为有符号1字节整数。
  参数：numeric
  返回值类型：tinyint
- numeric_text
  描述：numeric转化为text。

参数：numeric
返回值类型：text

- numeric_varchar
  描述：numeric转化为varchar。
  参数：numeric
  返回值类型：character varying
- nvarchar2in
  描述：将c字符串转化为varchar。
  参数：cstring, oid, integer
  返回值类型：nvarchar2
- nvarchar2out
  描述：将text转化为c字符串。
  参数：nvarchar2
  返回值类型：cstring
- nvarchar2send
  描述：将varchar转化为二进制。
  参数：nvarchar2
  返回值类型：bytea
- oidvectorin_extend
  描述：将字符串转化为oidvector。
  参数：cstring
  返回值类型：oidvector_extend
- oidvectorout_extend
  描述：将oidvector转化为字符串。
  参数：oidvector_extend
  返回值类型：cstring
- oidvectorsend_extend
  描述：将oidvector转化为字符串。
  参数：oidvector_extend
  返回值类型：bytea
- reltime_text
  描述：reltime转换为text。
  参数：reltime
  返回值类型：text
- text_date
  描述：text类型转换为date类型。
  参数：text
  返回值类型：date
- text_float4
  描述：text类型转换为float4类型。

参数：text
返回值类型：real

- text_float8
  描述：text类型转换为float8类型。
  参数：text
  返回值类型：double precision
- text_int1
  描述：text类型转换为int1类型。
  参数：text
  返回值类型：tinyint
- text_int2
  描述：text类型转换为int2类型。
  参数：text
  返回值类型：smallint
- text_int4
  描述：text类型转换为int4类型。
  参数：text
  返回值类型：integer
- text_int8
  描述：text类型转换为int8类型。
  参数：text
  返回值类型：bigint
- text_numeric
  描述：text类型转换为numeric类型。
  参数：text
  返回值类型：numeric
- text_timestamp
  描述：text类型转换为timestamp类型。
  参数：text
  返回值类型：timestamp without time zone
- time_text
  描述：time类型转换为text类型。
  参数：time without time zone
  返回值类型：text
- timestamp_text
  描述：timestamp类型转换为text类型。
  参数：timestamp without time zone
  返回值类型：text
- timestamp_to_smalldatetime
  描述：timestamp类型转换为smalldatetime类型。

参数：timestamp without time zone
返回值类型：smalldatetime

- timestamp_varchar
  描述：timestamp类型转换为varchar类型。
  参数：timestamp without time zone
  返回值类型：character varying
- timestamptz_to_smalldatetime
  描述：timestamptz类型转换为smalldatetime。
  参数：timestamp with time zone
  返回值类型：smalldatetime
- timestampzone_text
  描述：timestampzone类型转换为text类型。
  参数：timestamp with time zone
  返回值类型：text
- timetz_text
  描述：timetz类型转换为text类型。
  参数：time with time zone
  返回值类型：text
- to_integer
  描述：转换为integer类型。
  参数：character varying
  返回值类型：integer
- to_interval
  描述：转换为interval类型。
  参数：character varying
  返回值类型：interval
- to_numeric
  描述：转换为numeric类型。
  参数：character varying
  返回值类型：numeric
- to_nvarchar2
  描述：转换为nvarchar2类型。
  参数：numeric
  返回值类型：nvarchar2
- to_text
  描述：转换为text类型。
  参数：smallint
  返回值类型：text
- to_ts
  描述：转换为ts类型。

参数：character varying
返回值类型：timestamp without time zone

- to_varchar2
  描述：转换为varchar2类型。
  参数：timestamp without time zone
  返回值类型：character varying
- varchar_date
  描述：varchar类型转换为date。
  参数：character varying
  返回值类型：date
- varchar_float4
  描述：varchar类型转换为float4。
  参数：character varying
  返回值类型：real
- varchar_float8
  描述：varchar类型转换为float8。
  参数：character varying
  返回值类型：double precision
- varchar_int4
  描述：varchar类型转换为int4。
  参数：character varying
  返回值类型：integer
- varchar_int8
  描述：varchar类型转换为int8。
  参数：character varying
  返回值类型：bigint
- varchar_numeric
  描述：varchar类型转换为numeric。
  参数：character varying
  返回值类型：numeric
- varchar_timestamp
  描述：varchar类型转换为timestamp。
  参数：character varying
  返回值类型：timestamp without time zone
- varchar2_to_smlldatetime
  描述：varchar2类型转换为smlldatetime。
  参数：character varying
  返回值类型：smalldatetime
- xidout4
  描述：xid输出为4字节数字。

参数：xid32

返回值类型：cstring

- xidsend4

  描述：xid转换为二进制格式。

  参数：xid32

  返回值类型：bytea

- treat(expr AS [JSON | REF] schema.type)

  描述：将expr转化为AS后关键字指定的类型（JSON或输入的用户自定义类型）。

  返回值类型：JSON或输入的用户自定义类型。

  示例：

```
gaussdb=# CREATE TABLE json_doc(data CLOB);
gaussdb=# INSERT INTO json_doc VALUES('{"name":"a"}');
gaussdb=# SELECT treat(data as json) FROM json_doc;
    json
--------------
 {"name":"a"}
(1 row)
gaussdb=# DROP TABLE json_doc;
DROP TABLE
```

- nesttable_to_array(anynesttable)

  描述：将一个无索引的集合类型转换成具有相同元素的数组类型。

  参数：anynesttable

  返回值类型：anyarray

  示例：

```
gaussdb=# create or replace procedure p1 is
          type t1 is table of int;
          v1 t1 := t1(1, 2, 3);
          v2 int[] := cast(v1 as int[]);
        begin
          raise info '%', v2;
        end;
        /
gaussdb=# call p1();
INFO:  {1,2,3}
 p1
----


(1 row)

gaussdb=# CREATE TYPE t1 is table of int;
CREATE TYPE
gaussdb=# SELECT cast(t1(1, 2, 3) as int[]) result;
 result
---------
 {1,2,3}
(1 row)
gaussdb=# DROP PROCEDURE p1;
DROP PROCEDURE

gaussdb=# DROP TYPE t1;
DROP TYPE
```

- indexbytableint_to_array(anyindexbytable)

  描述：将一个索引类型为integer的集合类型转换成具有相同元素的数组类型。

  参数：anyindexbytable

  返回值类型：anyarray

示例：

```
gaussdb=# create or replace package pkg1 is
    type t1 is table of int index by int;
    procedure p1();
 end pkg1;
 /

gaussdb=# create or replace package body pkg1 is
    procedure p1() is
        v1 t1 := t1(1 => 1, 2 => 2, 3 => 3);
        v2 int[];
    begin
        v2 := cast(v1 as int[]);
        raise info '%', v2;
    end;
 end pkg1;
 /

gaussdb=# call pkg1.p1();
INFO:  {1,2,3}
 p1
----

(1 row)

gaussdb=# select indexbytableint_to_array(pkg1.t1(1 => 1, 2 => 2, 3 => 3));
 indexbytableint_to_array
--------------------------
 {1,2,3}
(1 row)

gaussdb=# DROP PACKAGE pkg1;
DROP PACKAGE
```

编码类型转换

- convert_to_nocase(text, text)

  描述：将字符串转换为指定的编码类型。

  返回值类型：bytea

  示例：

```
gaussdb=# SELECT convert_to_nocase('12345', 'GBK');
 convert_to_nocase
-------------------
 \x3132333435
(1 row)
```

7.5.10 几何函数和操作符

几何操作符

- +

  描述：平移。

  示例：

```
gaussdb=# SELECT box '((0,0),(1,1))' + point '(2.0,0)' AS RESULT;
   result
-------------
 (3,1),(2,0)
(1 row)
```

- -

  描述：平移。

示例：

```
gaussdb=# SELECT box '((0,0),(1,1))' - point '(2.0,0)' AS RESULT;
    result
---------------
 (-1,1),(-2,0)
(1 row)
```

- *

  描述：伸展/旋转。

  示例：

```
gaussdb=# SELECT box '((0,0),(1,1))' * point '(2.0,0)' AS RESULT;
   result
-------------
 (2,2),(0,0)
(1 row)
```

- /

  描述：收缩/旋转。

  示例：

```
gaussdb=# SELECT box '((0,0),(2,2))' / point '(2.0,0)' AS RESULT;
   result
-------------
 (1,1),(0,0)
(1 row)
```

- #

  描述：两个图形交面。

  示例：

```
gaussdb=# SELECT box '((1,-1),(-1,1))' # box '((1,1),(-2,-2))' AS RESULT;
 result
---------------
 (1,1),(-1,-1)
(1 row)
```

- #

  描述：图形的路径数目或多边形顶点数。

  示例：

```
gaussdb=# SELECT # path'((1,0),(0,1),(-1,0))' AS RESULT;
 result
--------
      3
(1 row)
```

- @-@

  描述：图形的长度或者周长。

  示例：

```
gaussdb=# SELECT @-@ path '((0,0),(1,0))' AS RESULT;
 result
--------
      2
(1 row)
```

- @@

  描述：图形的中心。

  示例：

```
gaussdb=# SELECT @@ circle '((0,0),10)' AS RESULT;
 result
--------
 (0,0)
(1 row)
```


- <->

  描述：两个图形之间的距离。

  示例：

```
gaussdb=# SELECT circle '((0,0),1)' <-> circle '((5,0),1)' AS RESULT;
 result
--------
      3
(1 row)
```

- &&

  描述：两个图形是否重叠（有一个共同点就为真）。

  示例：

```
gaussdb=# SELECT box '((0,0),(1,1))' && box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- <<

  描述：图形是否全部在另一个图形的左边（没有相同的横坐标）。

  示例：

```
gaussdb=# SELECT circle '((0,0),1)' << circle '((5,0),1)' AS RESULT;
 result
--------
 t
(1 row)
```

- >>

  描述：图形是否全部在另一个图形的右边（没有相同的横坐标）。

  示例：

```
gaussdb=# SELECT circle '((5,0),1)' >> circle '((0,0),1)' AS RESULT;
 result
--------
 t
(1 row)
```

- &<

  描述：图形的最右边是否不超过在另一个图形的最右边。

  示例：

```
gaussdb=# SELECT box '((0,0),(1,1))' &< box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- &>

  描述：图形的最左边是否不超过在另一个图形的最左边。

  示例：

```
gaussdb=# SELECT box '((0,0),(3,3))' &> box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- <<|

  描述：图形是否全部在另一个图形的下边（没有相同的纵坐标）。

  示例：


```
gaussdb=# SELECT box '((0,0),(3,3))' <<| box '((3,4),(5,5))' AS RESULT;
 result
--------
 t
(1 row)
```

- |>>

  描述：图形是否全部在另一个图形的上边（没有相同的纵坐标）。

  示例：

```
gaussdb=# SELECT box '((3,4),(5,5))' |>> box '((0,0),(3,3))' AS RESULT;
 result
--------
 t
(1 row)
```

- &<|

  描述：图形的最上边是否不超过另一个图形的最上边。

  示例：

```
gaussdb=# SELECT box '((0,0),(1,1))' &<| box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- |&>

  描述：图形的最下边是否不超过另一个图形的最下边。

  示例：

```
gaussdb=# SELECT box '((0,0),(3,3))' |&> box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- <^

  描述：图形是否低于另一个图形（允许两个图形有接触）。

  示例：

```
gaussdb=# SELECT box '((0,0),(-3,-3))' <^ box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- >^

  描述：图形是否高于另一个图形（允许两个图形有接触）。

  示例：

```
gaussdb=# SELECT box '((0,0),(2,2))' >^ box '((0,0),(-3,-3))'  AS RESULT;
 result
--------
 t
(1 row)
```

- ?#

  描述：两个图形是否相交。

  示例：

```
gaussdb=# SELECT lseg '((-1,0),(1,0))' ?# box '((-2,-2),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```


- ?-

  描述：图形是否处于水平位置。

  示例：

```
gaussdb=# SELECT ?- lseg '((-1,0),(1,0))' AS RESULT;
 result
--------
 t
(1 row)
```

- ?-

  描述：图形是否水平对齐。

  示例：

```
gaussdb=# SELECT point '(1,0)' ?- point '(0,0)' AS RESULT;
 result
--------
 t
(1 row)
```

- ?|

  描述：图形是否处于竖直位置。

  示例：

```
gaussdb=# SELECT ?| lseg '((-1,0),(1,0))' AS RESULT;
 result
--------
 f
(1 row)
```

- ?|

  描述：图形是否竖直对齐。

  示例：

```
gaussdb=# SELECT point '(0,1)' ?| point '(0,0)' AS RESULT;
 result
--------
 t
(1 row)
```

- ?-|

  描述：两条线是否垂直。

  示例：

```
gaussdb=# SELECT lseg '((0,0),(0,1))' ?-| lseg '((0,0),(1,0))' AS RESULT;
 result
--------
 t
(1 row)
```

- ?||

  描述：两条线是否平行。

  示例：

```
gaussdb=# SELECT lseg '((-1,0),(1,0))' ?|| lseg '((-1,2),(1,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- @>

  描述：图形是否包含另一个图形。

  示例：


```
gaussdb=# SELECT circle '((0,0),2)' @> point '(1,1)' AS RESULT;
 result
--------
 t
(1 row)
```

- <@

  描述：图形是否被包含于另一个图形。

  示例：

```
gaussdb=# SELECT point '(1,1)' <@ circle '((0,0),2)' AS RESULT;
 result
--------
 t
(1 row)
```

- ~=

  描述：两个图形是否相同。

  示例：

```
gaussdb=# SELECT polygon '((0,0),(1,1))' ~= polygon '((1,1),(0,0))' AS RESULT;
 result
--------
 t
(1 row)
```

几何函数

- area(object)

  描述：计算图形的面积。

  返回类型：double precision

  示例：

```
gaussdb=# SELECT area(box '((0,0),(1,1))') AS RESULT;
 result
--------
      1
(1 row)
```

- center(object)

  描述：计算图形的中心。

  返回类型：point

  示例：

```
gaussdb=# SELECT center(box '((0,0),(1,2))') AS RESULT;
 result
---------
 (0.5,1)
(1 row)
```

- diameter(circle)

  描述：计算圆的直径。

  返回类型：double precision

  示例：

```
gaussdb=# SELECT diameter(circle '((0,0),2.0)') AS RESULT;
 result
--------
      4
(1 row)
```

- height(box)

  描述：矩形的竖直高度。

返回类型：double precision

示例：

```
gaussdb=# SELECT height(box '((0,0),(1,1))') AS RESULT;
 result
--------
      1
(1 row)
```

- isclosed(path)

  描述：图形是否为闭合路径。

  返回类型：Boolean

  示例：

  ```
  gaussdb=# SELECT isclosed(path '((0,0),(1,1),(2,0))') AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- isopen(path)

  描述：图形是否为开放路径。

  返回类型：Boolean

  示例：

  ```
  gaussdb=# SELECT isopen(path '[(0,0),(1,1),(2,0)]') AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- length(object)

  描述：计算图形的长度。

  返回类型：double precision

  示例：

  ```
  gaussdb=# SELECT length(path '((-1,0),(1,0))') AS RESULT;
   result
  --------
      4
  (1 row)
  ```

- npoints(path)

  描述：计算路径的顶点数。

  返回类型：int

  示例：

  ```
  gaussdb=# SELECT npoints(path '[(0,0),(1,1),(2,0)]') AS RESULT;
   result
  --------
      3
  (1 row)
  ```

- npoints(polygon)

  描述：计算多边形的顶点数。

  返回类型：int

  示例：

  ```
  gaussdb=# SELECT npoints(polygon '((1,1),(0,0))') AS RESULT;
   result
  --------
      2
  (1 row)
  ```


- pclose(path)
  描述：把路径转换为闭合路径。
  返回类型：path
  示例：

```
gaussdb=# SELECT pclose(path '[(0,0),(1,1),(2,0)]') AS RESULT;
      result
--------------------
 ((0,0),(1,1),(2,0))
(1 row)
```

- popen(path)
  描述：把路径转换为开放路径。
  返回类型：path
  示例：

```
gaussdb=# SELECT popen(path '((0,0),(1,1),(2,0))') AS RESULT;
      result
--------------------
 [(0,0),(1,1),(2,0)]
(1 row)
```

- radius(circle)
  描述：计算圆的半径。
  返回类型：double precision
  示例：

```
gaussdb=# SELECT radius(circle '((0,0),2.0)') AS RESULT;
 result
--------
      2
(1 row)
```

- width(box)
  描述：计算矩形的水平尺寸。
  返回类型：double precision
  示例：

```
gaussdb=# SELECT width(box '((0,0),(1,1))') AS RESULT;
 result
--------
      1
(1 row)
```

几何类型转换函数

- box(circle)
  描述：将圆转换成矩形。
  返回类型：box
  示例：

```
gaussdb=# SELECT box(circle '((0,0),2.0)') AS RESULT;
                                  result
-------------------------------------------------------------------------
 (1.41421356237309,1.41421356237309),(-1.41421356237309,-1.41421356237309)
(1 row)
```

- box(point, point)
  描述：将点转换成矩形。
  返回类型：box

示例：

```
gaussdb=# SELECT box(point '(0,0)', point '(1,1)') AS RESULT;
   result
-------------
 (1,1),(0,0)
(1 row)
```

- box(polygon)

  描述：将多边形转换成矩形。

  返回类型：box

  示例：

```
gaussdb=# SELECT box(polygon '((0,0),(1,1),(2,0))') AS RESULT;
   result
-------------
 (2,1),(0,0)
(1 row)
```

- circle(box)

  描述：矩形转换成圆。

  返回类型：circle

  示例：

```
gaussdb=# SELECT circle(box '((0,0),(1,1))') AS RESULT;
            result
------------------------------
 <(0.5,0.5),0.707106781186548>
(1 row)
```

- circle(point, double precision)

  描述：将圆心和半径转换成圆。

  返回类型：circle

  示例：

```
gaussdb=# SELECT circle(point '(0,0)', 2.0) AS RESULT;
  result
-----------
 <(0,0),2>
(1 row)
```

- circle(polygon)

  描述：将多边形转换成圆。

  返回类型：circle

  示例：

```
gaussdb=# SELECT circle(polygon '((0,0),(1,1),(2,0))') AS RESULT;
                  result
-------------------------------------------
 <(1,0.333333333333333),0.924950591148529>
(1 row)
```

- lseg(box)

  描述：矩形对角线转化成线段。

  返回类型：lseg

  示例：

```
gaussdb=# SELECT lseg(box '((-1,0),(1,0))') AS RESULT;
    result
----------------
 [(1,0),(-1,0)]
(1 row)
```


- lseg(point, point)
  描述：点转换成线段。
  返回类型：lseg
  示例：

```
gaussdb=# SELECT lseg(point '(-1,0)', point '(1,0)') AS RESULT;
    result
----------------
 [(-1,0),(1,0)]
(1 row)
```

- slope(point, point)
  描述：计算两个点构成直线的斜率。
  返回类型: double
  示例：

```
gaussdb=# SELECT slope(point '(1,1)', point '(0,0)') AS RESULT;
 result
--------
      1
(1 row)
```

- path(polygon)
  描述：多边形转换成路径。
  返回类型：path
  示例：

```
gaussdb=# SELECT path(polygon '((0,0),(1,1),(2,0))') AS RESULT;
       result
---------------------
 ((0,0),(1,1),(2,0))
(1 row)
```

- point(double precision, double precision)
  描述：节点。
  返回类型：point
  示例：

```
gaussdb=# SELECT point(23.4, -44.5) AS RESULT;
    result
--------------
 (23.4,-44.5)
(1 row)
```

- point(box)
  描述：矩形的中心。
  返回类型：point
  示例：

```
gaussdb=# SELECT point(box '((-1,0),(1,0))') AS RESULT;
 result
--------
 (0,0)
(1 row)
```

- point(circle)
  描述：圆心。
  返回类型：point
  示例：

```
gaussdb=# SELECT point(circle '((0,0),2.0)') AS RESULT;
 result
```



```
--------
 (0,0)
(1 row)
```

- point(lseg)

  描述：线段的中心。

  返回类型：point

  示例：

```
gaussdb=# SELECT point(lseg '((-1,0),(1,0))') AS RESULT;
 result
--------
 (0,0)
(1 row)
```

- point(polygon)

  描述：多边形的中心。

  返回类型：point

  示例：

```
gaussdb=# SELECT point(polygon '((0,0),(1,1),(2,0))') AS RESULT;
       result
----------------------
 (1,0.333333333333333)
(1 row)
```

- polygon(box)

  描述：矩形转换成4点多边形。

  返回类型：polygon

  示例：

```
gaussdb=# SELECT polygon(box '((0,0),(1,1))') AS RESULT;
         result
---------------------------
 ((0,0),(0,1),(1,1),(1,0))
(1 row)
```

- polygon(circle)

  描述：圆转换成12点多边形。

  返回类型：polygon

  示例：

```
gaussdb=# SELECT polygon(circle '((0,0),2.0)') AS RESULT;

result

------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------
 ((-2,0),(-1.73205080756888,1),(-1,1.73205080756888),(-1.22464679914735e-16,2),
(1,1.73205080756888),(1.73205080756888,1),(2,2.44929359829471e-16),
(1.73205080756888,-0.999999999999999),(1,-1.73205080756888),(3.67394039744206e-16,-2),
(-0.999999999999999,-1.73205080756888),(-1.73205080756888,-1))
(1 row)
```

- polygon(npts, circle)

  描述：圆转换成npts点多边形。

  返回类型：polygon

  示例：

```
gaussdb=# SELECT polygon(12, circle '((0,0),2.0)') AS RESULT;

result
```



```
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 ((-2,0),(-1.73205080756888,1),(-1,1.73205080756888),(-1.22464679914735e-16,2),
(1,1.73205080756888),(1.73205080756888,1),(2,2.44929359829471e-16),
(1.73205080756888,-0.999999999999999),(1,-1.73205080756888),(3.67394039744206e-16,-2),
(-0.999999999999999,-1.73205080756888),(-1.73205080756888,-1))
(1 row)
```

- polygon(path)

  描述：路径转换成多边形。

  返回类型：polygon

  示例：

```
gaussdb=# SELECT polygon(path '((0,0),(1,1),(2,0))') AS RESULT;
      result
--------------------
 ((0,0),(1,1),(2,0))
(1 row)
```

7.5.11 网络地址函数和操作符

cidr 和 inet 操作符

操作符<<，<<=，>>，>>=对子网进行测试。它们只考虑两个地址的网络部分（忽略任何主机部分），然后判断其中一个网络是等于另外一个网络，还是另外一个网络的子网。

- <

  描述：小于。

  示例：

```
gaussdb=# SELECT inet '192.168.1.5' < inet '192.168.1.6' AS RESULT;
 result
--------
 t
(1 row)
```

- <=

  描述：小于或等于。

  示例：

```
gaussdb=# SELECT inet '192.168.1.5' <= inet '192.168.1.5' AS RESULT;
 result
--------
 t
(1 row)
```

- =

  描述：等于。

  示例：

```
gaussdb=# SELECT inet '192.168.1.5' = inet '192.168.1.5' AS RESULT;
 result
--------
 t
(1 row)
```

- >=

  描述：大于或等于。

  示例：


```
gaussdb=# SELECT inet '192.168.1.5' >= inet '192.168.1.5' AS RESULT;
 result
--------
 t
(1 row)
```

- >

  描述：大于。

  示例：

```
gaussdb=# SELECT inet '192.168.1.5' > inet '192.168.1.4' AS RESULT;
 result
--------
 t
(1 row)
```

- <>

  描述：不等于。

  示例：

```
gaussdb=# SELECT inet '192.168.1.5' <> inet '192.168.1.4' AS RESULT;
 result
--------
 t
(1 row)
```

- <<

  描述：包含于。

  示例：

```
gaussdb=# SELECT inet '192.168.1.5' << inet '192.168.1/24' AS RESULT;
 result
--------
 t
(1 row)
```

- <<=

  描述：包含于或等于。

  示例：

```
gaussdb=# SELECT inet '192.168.1/24' <<= inet '192.168.1/24' AS RESULT;
 result
--------
 t
(1 row)
```

- >>

  描述：包含。

  示例：

```
gaussdb=# SELECT inet '192.168.1/24' >> inet '192.168.1.5' AS RESULT;
 result
--------
 t
(1 row)
```

- >>=

  描述：包含或等于。

  示例：

```
gaussdb=# SELECT inet '192.168.1/24' >>= inet '192.168.1/24' AS RESULT;
 result
--------
 t
(1 row)
```


- ~

  描述：位非。

  示例：

```
gaussdb=# SELECT ~ inet '192.168.1.6' AS RESULT;
    result
---------------
 63.87.254.249
(1 row)
```

- &

  描述：两个网络地址的每一位都进行“与”操作。

  示例：

```
gaussdb=# SELECT inet '192.168.1.6' & inet '10.0.0.0' AS RESULT;
 result
---------
 0.0.0.0
(1 row)
```

- |

  描述：两个网络地址的每一位都进行“或”操作。

  示例：

```
gaussdb=# SELECT inet '192.168.1.6' | inet '10.0.0.0' AS RESULT;
   result
-------------
 202.168.1.6
(1 row)
```

- +

  描述：加。

  示例：

```
gaussdb=# SELECT inet '192.168.1.6' + 25 AS RESULT;
    result
--------------
 192.168.1.31
(1 row)
```

- -

  描述：减。

  示例：

```
gaussdb=# SELECT inet '192.168.1.43' - 36 AS RESULT;
   result
-------------
 192.168.1.7
(1 row)
```

- -

  描述：减。

  示例：

```
gaussdb=# SELECT inet '192.168.1.43' - inet '192.168.1.19' AS RESULT;
 result
--------
     24
(1 row)
```

cidr 和 inet 函数

函数abbrev，host，text主要是为了提供可选的显示格式。

- abbrev(inet)
  描述：缩写显示格式文本。
  返回类型：text
  示例：

```
gaussdb=# SELECT abbrev(inet '10.1.0.0/16') AS RESULT;
   result
-------------
 10.1.0.0/16
(1 row)
```

- abbrev(cidr)
  描述：缩写显示格式文本。
  返回类型：text
  示例：

```
gaussdb=# SELECT abbrev(cidr '10.1.0.0/16') AS RESULT;
 result
---------
 10.1/16
(1 row)
```

- broadcast(inet)
  描述：网络广播地址。
  返回类型：inet
  示例：

```
gaussdb=# SELECT broadcast('192.168.1.5/24') AS RESULT;
      result
------------------
 192.168.1.255/24
(1 row)
```

- family(inet)
  描述：抽取地址族，4为IPv4。
  返回类型：int
  示例：

```
gaussdb=# SELECT family('127.0.0.1') AS RESULT;
 result
--------
      4
(1 row)
```

- host(inet)
  描述：将主机地址类型抽出为文本。
  返回类型：text
  示例：

```
gaussdb=# SELECT host('192.168.1.5/24') AS RESULT;
   result
-------------
 192.168.1.5
(1 row)
```

- hostmask(inet)
  描述：为网络构造主机掩码。
  返回类型：inet
  示例：

```
gaussdb=# SELECT hostmask('192.168.23.20/30') AS RESULT;
 result
```



```
---------
 0.0.0.3
(1 row)
```

- masklen(inet)

  描述：抽取子网掩码长度。

  返回类型：int

  示例：

```
gaussdb=# SELECT masklen('192.168.1.5/24') AS RESULT;
 result
--------
     24
(1 row)
```

- netmask(inet)

  描述：为网络构造子网掩码。

  返回类型：inet

  示例：

```
gaussdb=# SELECT netmask('192.168.1.5/24') AS RESULT;
    result
---------------
 255.255.255.0
(1 row)
```

- network(inet)

  描述：抽取地址的网络部分。

  返回类型：cidr

  示例：

```
gaussdb=# SELECT network('192.168.1.5/24') AS RESULT;
     result
----------------
 192.168.1.0/24
(1 row)
```

- set_masklen(inet, int)

  描述：为inet数值设置子网掩码长度。

  返回类型：inet

  示例：

```
gaussdb=# SELECT set_masklen('192.168.1.5/24', 16) AS RESULT;
     result
----------------
 192.168.1.5/16
(1 row)
```

- set_masklen(cidr, int)

  描述：为cidr数值设置子网掩码长度。

  返回类型：cidr

  示例：

```
gaussdb=# SELECT set_masklen('192.168.1.0/24'::cidr, 16) AS RESULT;
     result
----------------
 192.168.0.0/16
(1 row)
```

- text(inet)

  描述：把IP地址和掩码长度抽取为文本。

  返回类型：text

示例：

```
gaussdb=# SELECT text(inet '192.168.1.5') AS RESULT;
    result
----------------
 192.168.1.5/32
(1 row)
```

任何cidr值都能以显式或者隐式的方式转换为inet值，因此上述能够操作inet值的函数也同样能够操作cidr值。inet值也可以转换为cidr值，此时inet子网掩码右侧的所有位都将转换为零，以创建一个有效的cidr值。另外，用户还可以使用常规的类型转换语法将一个文本字符串转换为inet或cidr值。例如：inet(expression)或colname::cidr。

macaddr 函数

函数trunc(macaddr)返回一个MAC地址，该地址的最后三个字节设置为零。

- trunc(macaddr)

  描述：把后三个字节置为零。

  返回类型：macaddr

  示例：

```
gaussdb=# SELECT trunc(macaddr '12:34:56:78:90:ab') AS RESULT;
     result
-------------------
 12:34:56:00:00:00
(1 row)
```

macaddr类型还支持标准关系操作符（>，<=等）用于词法排序，和按位运算符（~，&和|）非，与和或。

7.5.12 文本检索函数和操作符

文本检索操作符

- @@

  描述：tsvector类型的词汇与tsquery类型的词汇是否匹配。

  示例：

```
gaussdb=# SELECT to_tsvector('fat cats ate rats') @@ to_tsquery('cat & rat') AS RESULT;
 result
--------
 t
(1 row)
```

- @@@

  描述：@@的同义词。

  示例：

```
gaussdb=# SELECT to_tsvector('fat cats ate rats') @@@ to_tsquery('cat & rat') AS RESULT;
 result
--------
 t
(1 row)
```

- ||

  描述：连接两个tsvector类型的词汇。

  示例：

```
gaussdb=# SELECT 'a:1 b:2'::tsvector || 'c:1 d:2 b:3'::tsvector AS RESULT;
          result
```



```
----------------------------
 'a':1 'b':2,5 'c':3 'd':4
(1 row)
```

- &&

  描述：将两个tsquery类型的词汇进行“与”操作。

  示例：

```
gaussdb=# SELECT 'fat | rat'::tsquery && 'cat'::tsquery AS RESULT;
          result
----------------------------
 ( 'fat' | 'rat' ) & 'cat'
(1 row)
```

- ||

  描述：将两个tsquery类型的词汇进行“或”操作。

  示例：

```
gaussdb=# SELECT 'fat | rat'::tsquery || 'cat'::tsquery AS RESULT;
          result
----------------------------
 ( 'fat' | 'rat' ) | 'cat'
(1 row)
```

- !!

  描述：tsquery类型词汇的非关系。

  示例：

```
gaussdb=# SELECT !! 'cat'::tsquery AS RESULT;
 result
--------
 !'cat'
(1 row)
```

- @>

  描述：一个tsquery类型的词汇是否包含另一个tsquery类型的词汇。

  示例：

```
gaussdb=# SELECT 'cat'::tsquery @> 'cat & rat'::tsquery AS RESULT;
 result
--------
 f
(1 row)
```

- <@

  描述：一个tsquery类型的词汇是否被包含另一个tsquery类型的词汇。

  示例：

```
gaussdb=# SELECT 'cat'::tsquery <@ 'cat & rat'::tsquery AS RESULT;
 result
--------
 t
(1 row)
```

除了上述的操作符，还为tsvector类型和tsquery类型的数据定义了普通的B-tree比较操作符（=，<等）。

文本检索函数

- get_current_ts_config()

  描述：获取文本检索的默认配置。

  返回类型：regconfig

  示例：


```
gaussdb=# SELECT get_current_ts_config();
 get_current_ts_config
-----------------------
 english
(1 row)
```

- length(tsvector)

  描述：tsvector类型词汇的单词数。

  返回类型：integer

  示例：

```
gaussdb=# SELECT length('fat:2,4 cat:3 rat:5A'::tsvector);
 length
--------
      3
(1 row)
```

- numnode(tsquery)

  描述：tsquery类型的单词加上操作符的数量。

  返回类型：integer

  示例：

```
gaussdb=# SELECT numnode('(fat & rat) | cat'::tsquery);
 numnode
---------
       5
(1 row)
```

- plainto_tsquery([config regconfig ,] query text)

  描述：产生tsquery类型的词汇，并忽略标点。

  返回类型：tsquery

  示例：

```
gaussdb=# SELECT plainto_tsquery('english', 'The Fat Rats');
 plainto_tsquery
-----------------
 'fat' & 'rat'
(1 row)
```

- querytree(query tsquery)

  描述：获取tsquery类型的词汇可加索引的部分。

  返回类型：text

  示例：

```
gaussdb=# SELECT querytree('foo & ! bar'::tsquery);
 querytree
-----------
 'foo'
(1 row)
```

- setweight(tsvector, "char")

  描述：给tsvector类型的每个元素分配权值。

  返回类型：tsvector

  示例：

```
gaussdb=# SELECT setweight('fat:2,4 cat:3 rat:5B'::tsvector, 'A');
            setweight
--------------------------------
 'cat':3A 'fat':2A,4A 'rat':5A
(1 row)
```

- strip(tsvector)

  描述：删除tsvector类型单词中的position和权值。

返回类型：tsvector

示例：

```
gaussdb=# SELECT strip('fat:2,4 cat:3 rat:5A'::tsvector);
       strip
-------------------
 'cat' 'fat' 'rat'
(1 row)
```

- to_tsquery([config regconfig ,] query text)

  描述：标准化单词，并转换为tsquery类型。

  返回类型：tsquery

  示例：

  ```
  gaussdb=# SELECT to_tsquery('english', 'The & Fat & Rats');
    to_tsquery
  ---------------
   'fat' & 'rat'
  (1 row)
  ```

- to_tsvector([config regconfig ,] document text)

  描述：去除文件信息，并转换为tsvector类型。

  返回类型：tsvector

  示例：

  ```
  gaussdb=# SELECT to_tsvector('english', 'The Fat Rats');
     to_tsvector
  -----------------
   'fat':2 'rat':3
  (1 row)
  ```

- to_tsvector_for_batch([config regconfig ,] document text)

  描述：去除文件信息，并转换为tsvector类型。

  返回类型：tsvector

  示例：

  ```
  gaussdb=# SELECT to_tsvector_for_batch('english', 'The Fat Rats');
     to_tsvector
  -----------------
   'fat':2 'rat':3
  (1 row)
  ```

- ts_headline([config regconfig,] document text, query tsquery [, options text])

  描述：高亮显示查询的匹配项。

  返回类型：text

  示例：

  ```
  gaussdb=# SELECT ts_headline('x y z', 'z'::tsquery);
   ts_headline
  --------------
   x y <b>z</b>
  (1 row)
  ```

- ts_rank([weights float4[],] vector tsvector, query tsquery [, normalization integer])

  描述：文档查询排名。

  返回类型：float4

  示例：

  ```
  gaussdb=# SELECT ts_rank('hello world'::tsvector, 'world'::tsquery);
   ts_rank
  ```



```
----------
 .0607927
(1 row)
```

- ts_rank_cd([weights float4[],] vector tsvector, query tsquery [, normalization integer])

  描述：排序文件查询使用覆盖密度。

  返回类型：float4

  示例：

```
gaussdb=# SELECT ts_rank_cd('hello world'::tsvector, 'world'::tsquery);
 ts_rank_cd
------------
          .0
(1 row)
```

- ts_rewrite(query tsquery, target tsquery, substitute tsquery)

  描述：替换目标tsquery类型的单词。

  返回类型：tsquery

  示例：

```
gaussdb=# SELECT ts_rewrite('a & b'::tsquery, 'a'::tsquery, 'foo|bar'::tsquery);
       ts_rewrite
-------------------------
 'b' & ( 'foo' | 'bar' )
(1 row)
```

- ts_rewrite(query tsquery, select text)

  描述：使用SELECT命令的结果替代目标中tsquery类型的单词。

  返回类型：tsquery

  示例：

```
gaussdb=# SELECT ts_rewrite('world'::tsquery, 'select ''world''::tsquery, ''hello''::tsquery');
 ts_rewrite
------------
 'hello'
(1 row)
```

文本检索调试函数

- ts_debug([config regconfig,] document text, OUT alias text, OUT description text, OUT token text, OUT dictionaries regdictionary[], OUT dictionary regdictionary, OUT lexemes text[])

  描述：测试一个配置。

  返回类型：setof record

  示例：

```
gaussdb=# SELECT ts_debug('english', 'The Brightest supernovaes');
                                    ts_debug
-------------------------------------------------------------------------------
 (asciiword,"Word, all ASCII",The,{english_stem},english_stem,{})
 (blank,"Space symbols"," ",{},,)
 (asciiword,"Word, all ASCII",Brightest,{english_stem},english_stem,{brightest})
 (blank,"Space symbols"," ",{},,)
 (asciiword,"Word, all ASCII",supernovaes,{english_stem},english_stem,{supernova})
(5 rows)
```

- ts_lexize(dict regdictionary, token text)

  描述：测试一个数据字典。

  返回类型：text[]

  示例：


```
gaussdb=# SELECT ts_lexize('english_stem', 'stars');
 ts_lexize
-----------
 {star}
(1 row)
```

- ts_parse(parser_name text, document text, OUT tokid integer, OUT token text)

  描述：测试一个解析。

  返回类型：setof record

  示例：

```
gaussdb=# SELECT ts_parse('default', 'foo - bar');
 ts_parse
-----------
 (1,foo)
 (12," ")
 (12,"- ")
 (1,bar)
(4 rows)
```

- ts_parse(parser_oid oid, document text, OUT tokid integer, OUT token text)

  描述：测试一个解析。

  返回类型：setof record

  示例：

```
gaussdb=# SELECT ts_parse(3722, 'foo - bar');
 ts_parse
-----------
 (1,foo)
 (12," ")
 (12,"- ")
 (1,bar)
(4 rows)
```

- ts_token_type(parser_name text, OUT tokid integer, OUT alias text, OUT description text)

  描述：获取分析器定义的记号类型。

  返回类型：setof record

  示例：

```
gaussdb=# SELECT ts_token_type('default');
                        ts_token_type
------------------------------------------------------------
 (1,asciiword,"Word, all ASCII")
 (2,word,"Word, all letters")
 (3,numword,"Word, letters and digits")
 (4,email,"Email address")
 (5,url,URL)
 (6,host,Host)
 (7,sfloat,"Scientific notation")
 (8,version,"Version number")
 (9,hword_numpart,"Hyphenated word part, letters and digits")
 (10,hword_part,"Hyphenated word part, all letters")
 (11,hword_asciipart,"Hyphenated word part, all ASCII")
 (12,blank,"Space symbols")
 (13,tag,"XML tag")
 (14,protocol,"Protocol head")
 (15,numhword,"Hyphenated word, letters and digits")
 (16,asciihword,"Hyphenated word, all ASCII")
 (17,hword,"Hyphenated word, all letters")
 (18,url_path,"URL path")
 (19,file,"File or path name")
 (20,float,"Decimal notation")
 (21,int,"Signed integer")
```



```
 (22,uint,"Unsigned integer")
 (23,entity,"XML entity")
(23 rows)
```

- ts_token_type(parser_oid oid, OUT tokid integer, OUT alias text, OUT description text)

  描述：获取分析器定义的记号类型。

  返回类型：setof record

  示例：

```
gaussdb=# SELECT ts_token_type(3722);
                      ts_token_type
------------------------------------------------------------
 (1,asciiword,"Word, all ASCII")
 (2,word,"Word, all letters")
 (3,numword,"Word, letters and digits")
 (4,email,"Email address")
 (5,url,URL)
 (6,host,Host)
 (7,sfloat,"Scientific notation")
 (8,version,"Version number")
 (9,hword_numpart,"Hyphenated word part, letters and digits")
 (10,hword_part,"Hyphenated word part, all letters")
 (11,hword_asciipart,"Hyphenated word part, all ASCII")
 (12,blank,"Space symbols")
 (13,tag,"XML tag")
 (14,protocol,"Protocol head")
 (15,numhword,"Hyphenated word, letters and digits")
 (16,asciihword,"Hyphenated word, all ASCII")
 (17,hword,"Hyphenated word, all letters")
 (18,url_path,"URL path")
 (19,file,"File or path name")
 (20,float,"Decimal notation")
 (21,int,"Signed integer")
 (22,uint,"Unsigned integer")
 (23,entity,"XML entity")
(23 rows)
```

- ts_stat(sqlquery text, [weights text,] OUT word text, OUT ndoc integer, OUT nentry integer)

  描述：获取tsvector列的统计数据。

  返回类型：setof record

  示例：

```
gaussdb=# SELECT ts_stat('select ''hello world''::tsvector');
  ts_stat
-------------
 (world,1,1)
 (hello,1,1)
(2 rows)
```

7.5.13 JSON/JSONB 函数和操作符

JSON/JSONB数据类型参考**JSON/JSONB类型**，操作符参见**表7-39**、**表7-40**。

表 7-39 JSON/JSONB 通用操作符

操作符	左操作数类型	右操作数类型	返回类型	描述	示例
->	Array-json(b)	int	json(b)	获得array-json元素。下标不存在返回空。	SELECT '[{"a":"foo"},{"b":"bar"},{"c":"baz"}]'::json->2; ?column? -------------- {"c":"baz"} (1 row)
->	object-json(b)	text	json(b)	通过键获得值。不存在则返回空。	SELECT '{"a": {"b":"foo"}}'::json->'a'; ?column? -------------- {"b":"foo"} (1 row)
->>	Array-json(b)	int	text	获得 JSON 数组元素。下标不存在返回空。	SELECT '[1,2,3]'::json->>2; ?column? ---------- 3 (1 row)
->>	object-json(b)	text	text	通过键获得值。不存在则返回空。	SELECT '{"a":1,"b":2}'::json->>'b'; ?column? ---------- 2 (1 row)
#>	container-json(b)	text[]	json(b)	获取在指定路径的JSON 对象，路径不存在则返回空。	SELECT '{"a": {"b":{"c": "foo"}}}'::json #>'{a,b}'; ?column? -------------- {"c": "foo"} (1 row)
#>>	container-json(b)	text[]	text	获取在指定路径的JSON 对象，路径不存在则返回空。	SELECT '{"a":[1,2,3],"b":[4,5,6]}'::json #>>'{a,2}'; ?column? ---------- 3 (1 row)

注意

对于 #> 和 #>> 操作符，当给出的路径无法查找到数据时，不会报错，会返回空。

表 7-40 JSONB 额外支持操作符

操作符	右操作数类型	描述	例子
@>	jsonb	左边的JSON的顶层是否包含右边JSON的顶层所有项。	'{"a":1, "b":2}'::jsonb @> '{"b":2}'::jsonb

操作符	右操作数类型	描述	例子
<@	jsonb	左边的JSON的所有项是否全部存在于右边JSON的顶层。	'{"b":2}'::jsonb <@ '{"a":1, "b":2}'::jsonb
?	text	键/元素的字符串是否存在于 JSON 值的顶层。	'{"a":1, "b":2}'::jsonb ? 'b'
?|	text[]	这些数组字符串中的任何一个是否作为顶层键存在。	'{"a":1, "b":2, "c":3}'::jsonb ?| array['b', 'c']
?&	text[]	是否所有这些数组字符串都作为顶层键存在。	'["a", "b"]'::jsonb ?& array['a', 'b']
=	jsonb	判断两个jsonb的大小关系，同函数jsonb_eq。	/
<>	jsonb	判断两个jsonb的大小关系，同函数jsonb_ne。	/
<	jsonb	判断两个jsonb的大小关系，同函数jsonb_lt。	/
>	jsonb	判断两个jsonb的大小关系，同函数jsonb_gt。	/
<=	jsonb	判断两个jsonb的大小关系，同函数jsonb_le。	/
>=	jsonb	判断两个jsonb的大小关系，同函数jsonb_ge。	/

JSON/JSONB 支持的函数

- array_to_json(anyarray [, pretty_bool])

  描述：返回JSON类型的数组。将一个多维数组组成一个JSON数组。如果pretty_bool为true，将在一维元素之间添加换行符。

  返回类型：json

  示例：

```
gaussdb=# SELECT array_to_json('{{1,5},{99,100}}'::int[]);
  array_to_json
------------------
[[1,5],[99,100]]
(1 row)
```


- row_to_json(record [, pretty_bool])

  描述：返回JSON类型的行。如果pretty_bool为true，将在第一级元素之间添加换行符。

  返回类型：json

  示例：

```
gaussdb=# SELECT row_to_json(row(1,'foo'));
        row_to_json
  ---------------------
   {"f1":1,"f2":"foo"}    (1 row)
```

- json_array_element(array-json, integer)、jsonb_array_element(array-jsonb, integer)

  描述：同操作符`->`, 返回数组中指定下标的元素。

  返回类型：json、jsonb

  示例：

```
gaussdb=# SELECT json_array_element('[1,true,[1,[2,3
]],null]',2);
 json_array_element
 --------------------
 [1,[2,3]]
 (1 row)
```

- json_array_element_text(array-json, integer)、jsonb_array_element_text(array-jsonb, integer)

  描述：同操作符`->>`, 返回数组中指定下标的元素。

  返回类型：text、text

  示例：

```
gaussdb=# SELECT json_array_element_text('[1,true,[1,[2,3]],null]',2);
json_array_element_text
------------------------
 [1,[2,3]]
 (1 row)
```

- json_object_field(object-json, text)、jsonb_object_field(object-jsonb, text)

  描述：同操作符`->`, 返回对象中指定键对应的值。

  返回类型：json、json

  示例：

```
gaussdb=# SELECT json_object_field('{"a": {"b":"foo"}}','a');
 json_object_field
 -------------------
 {"b":"foo"}
 (1 row)
```

- json_object_field_text(object-json, text)、jsonb_object_field_text(object-jsonb, text)

  描述：同操作符`->>`, 返回对象中指定键对应的值。

  返回类型：text、text

  示例：

```
gaussdb=# SELECT json_object_field_text('{"a": {"b":"foo"}}','a');
 json_object_field_text
 -----------------------
 {"b":"foo"}
 (1 row)
```

- json_extract_path(json, VARIADIC text[])、jsonb_extract_path((jsonb, VARIADIC text[])

  描述：等价于操作符`#>`。根据$2所指的路径，查找json，并返回。

返回类型：json、jsonb

示例：

```
gaussdb=# SELECT json_extract_path('{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}', 'f4','f6');
 json_extract_path
-------------------
 "stringy"
(1 row)
```

- json_extract_path_op(json, text[])、jsonb_extract_path_op(jsonb, text[])

  描述：同操作符`#>`。根据$2所指的路径，查找json，并返回。

  返回类型：json、jsonb

  示例：

```
gaussdb=# SELECT json_extract_path_op('{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}', ARRAY['f4','f6']);
 json_extract_path_op
----------------------
 "stringy"
(1 row)
```

- json_extract_path_text(json, VARIADIC text[])、jsonb_extract_path_text((jsonb, VARIADIC text[])

  描述：等价于操作符`#>>`。根据$2所指的路径，查找json，并返回。

  返回类型：text、text

  示例：

```
gaussdb=# SELECT json_extract_path_text('{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}', 'f4','f6');
 json_extract_path_text
------------------------
 "stringy"
(1 row)
```

- json_extract_path_text_op(json, text[])、jsonb_extract_path_text_op(jsonb, text[])

  描述：同操作符`#>>`。根据$2所指的路径，查找json，并返回。

  返回类型：text、text

  示例：

```
gaussdb=# SELECT json_extract_path_text_op('{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}',
ARRAY['f4','f6']);
 json_extract_path_text_op
---------------------------
 "stringy"
(1 row)
```

- json_array_elements(array-json)、jsonb_array_elements(array-jsonb)

  描述：拆分数组，每一个元素返回一行。

  返回类型：json、jsonb

  示例：

```
gaussdb=# SELECT json_array_elements('[1,true,[1,[2,3]],null]');
 json_array_elements
----------------------
 1
 true
 [1,[2,3]]
 null
(4 rows)
```

- json_array_elements_text(array-json)、jsonb_array_elements_text(array-jsonb)

  描述：拆分数组，每一个元素返回一行。

  返回类型：text、text

  示例：


```
gaussdb=# SELECT * FROM  json_array_elements_text('[1,true,[1,[2,3]],null]');
    value
-----------
 1
 true
 [1,[2,3]]

 (4 rows)
```

- json_array_length(array-json)、jsonb_array_length(array-jsonb)

  描述：返回数组长度。

  返回类型：integer

  示例：

```
 gaussdb=# SELECT json_array_length('[1,2,3,{"f1":1,"f2":[5,6]},4,null]');
  json_array_length
 -------------------
                  6
 (1 row)
```

- json_each(object-json)、jsonb_each(object-jsonb)

  描述：将对象的每个键值对拆分转换成一行两列。

  返回类型：setof(key text, value json)、setof(key text, value jsonb)

  示例：

```
 gaussdb=# SELECT * FROM  json_each('{"f1":[1,2,3],"f2":{"f3":1},"f4":null}');
  key |  value
 -----+----------
  f1  | [1,2,3]
  f2  | {"f3":1}
  f4  | null
 (3 rows)
```

- json_each_text(object-json)、jsonb_each_text(object-jsonb)

  描述：将对象的每个键值对拆分转换成一行两列。

  返回类型：setof(key text, value text)、setof(key text, value text)

  示例：

```
 gaussdb=# SELECT * FROM  json_each_text('{"f1":[1,2,3],"f2":{"f3":1},"f4":null}');
  key |  value
 -----+----------
  f1  | [1,2,3]
  f2  | {"f3":1}
  f4  |
 (3 rows)
```

- json_object_keys(object-json)、jsonb_object_keys(object-jsonb)

  描述：返回对象中顶层的所有键。

  返回类型：SETOF text

  示例：

```
 gaussdb=# SELECT json_object_keys('{"f1":"abc","f2":{"f3":"a", "f4":"b"}, "f1":"abcd"}');
  json_object_keys
 ------------------
  f1
  f2
  f1
 (3 rows)
```

- jsonb中会有去重操作

```
 gaussdb=# SELECT jsonb_object_keys('{"f1":"abc","f2":{"f3":"a", "f4":"b"}, "f1":"abcd"}');
  jsonb_object_keys
 -------------------
  f1
  f2
 (2 rows)
```


- json_populate_record(anyelement, object-json [, bool])、jsonb_populate_record(anyelement, object-jsonb [, bool])

  描述：$1必须是一个复合类型的参数。将会把object-json里的每个对键值进行拆分，以键当做列名，与$1中的列名进行匹配查找，并填充到$1的格式中。

  返回类型：anyelement、anyelement

  示例：

```
gaussdb=# CREATE TYPE jpop AS (a text, b int, c bool);
CREATE TYPE
gaussdb=# SELECT * FROM json_populate_record(null::jpop,'{"a":"blurfl","x":43.2}');
   a    | b | c
--------+---+---
 blurfl |   |
(1 row)
gaussdb=#  DROP TYPE jpop;
DROP TYPE
```

```
gaussdb=# SELECT * FROM json_populate_record((1,1,null)::jpop,'{"a":"blurfl","x":43.2}');
   a    | b | c
--------+---+---
 blurfl | 1 |
(1 row)
```

- json_populate_record_set(anyelement, array-json [, bool])、jsonb_populate_record_set(anyelement, array-jsonb [, bool])

  描述：参考上述函数json_populate_record、jsonb_populate_record，对$2数组的每一个元素进行上述参数函数的操作，因此这也要求$2数组的每个元素都是object-json类型的。

  返回类型：setof anyelement、setof anyelement

  示例：

```
gaussdb=# CREATE TYPE jpop AS (a text, b int, c bool);
CREATE TYPE
gaussdb=# SELECT * FROM json_populate_recordset(null::jpop, '[{"a":1,"b":2},{"a":3,"b":4}]');
 a | b | c
---+---+---
 1 | 2 |
 3 | 4 |
(2 rows)
gaussdb=#  DROP TYPE jpop;
DROP TYPE
```

- json_typeof(json)、jsonb_typeof(jsonb)

  描述：检测json类型。

  返回类型：text、text

  示例：

```
gaussdb=# SELECT value, json_typeof(value) FROM (values (json '123.4'), (json '"foo"'), (json 'true'),
(json 'null'), (json '[1, 2, 3]'), (json '{"x":"foo", "y":123}'), (NULL::json))  AS data(value);
         value          | json_typeof
------------------------+-------------
 123.4                  | number
 "foo"                  | string
 true                   | boolean
 null                   | null
 [1, 2, 3]              | array
 {"x":"foo", "y":123} | object
                        |
(7 rows)
```

- json_build_array([VARIADIC "any"])

  描述：从一个可变参数列表构造出一个JSON数组。

  返回类型：array-json

  示例：


```
gaussdb=# SELECT json_build_array('a',1,'b',1.2,'c',true,'d',null,'e',json '{"x": 3, "y": [1,2,3]}','');
                          json_build_array
-------------------------------------------------------------------------
 ["a", 1, "b", 1.2, "c", true, "d", null, "e", {"x": 3, "y": [1,2,3]}, ""]
(1 row)
```

- json_build_object([VARIADIC "any"])

  描述：从一个可变参数列表构造出一个JSON对象，其入参必须为偶数个，两两一组组成键值对。注意键不可为null。

  返回类型：object-json

  示例：

```
gaussdb=# SELECT json_build_object(1,2);
 json_build_object
-------------------
 {"1" : 2}
(1 row)
```

- json_to_record(object-json, bool)

  描述：正如所有返回record的函数一样，调用者必须用一个AS子句显式地定义记录的结构。会将object-json的键值对进行拆分重组，把键当做列名，去匹配填充AS显示指定的记录的结构，bool类型入参表示是否支持对象嵌套，即是否支持json对象的成员的值也是一个json对象，true为支持，false为不支持。

  返回类型：record

  示例：

```
gaussdb=# SELECT * FROM json_to_record('{"a":1,"b":"foo","c":"bar"}',true) AS x(a int, b text, d text);
 a |  b  | d
---+-----+---
 1 | foo |
(1 row)
```

- json_to_recordset(array-json, bool)

  描述：参考函数json_to_record，对数组内每个元素，执行上述函数的操作，因此这要求数组内的每个元素都得是object-json，bool类型入参表示是否支持对象嵌套，即是否支持json对象的成员的值也是一个json对象，true为支持，false为不支持。

  返回类型：setof record

  示例：

```
gaussdb=# SELECT * FROM json_to_recordset(
gaussdb(#   '[{"a":1,"b":"foo","d":false},{"a":2,"b":"bar","c":true}]',
gaussdb(#   false
gaussdb(# ) AS x(a int, b text, c boolean);
 a |  b  | c
---+-----+---
 1 | foo |
 2 | bar | t
(2 rows)
```

- json_object(text[])、json_object(text[], text[])

  描述：从一个文本数组构造一个object-json。这是个重载函数，当入参为一个文本数组的时候，其数组长度必须为偶数，成员被当做交替出现的键值对。两个文本数组的时候，第一个数组认为是键，第二个认为是值，两个数组长度必须相等。键不可为null。

  返回类型：object-json

  示例：

```
gaussdb=# SELECT json_object('{a,1,b,2,3,NULL,"d e f","a b c"}');
                       json_object
-------------------------------------------------------
 {"a" : "1", "b" : "2", "3" : null, "d e f" : "a b c"}
(1 row)
```



```
gaussdb=# SELECT json_object('{a,b,"a b c"}', '{a,1,1}');
              json_object
---------------------------------------
 {"a" : "a", "b" : "1", "a b c" : "1"}
(1 row)
```

- json_agg(any)

  描述：将值聚集为json数组。

  返回类型：array-json

  示例：

```
gaussdb=# SELECT * FROM classes;
 name | score
------+-------
A    |     2
A    |     3
D    |     5
D    |
(4 rows)
gaussdb=# SELECT name, json_agg(score) score FROM classes GROUP BY name ORDER BY name;
name |       score
-----+------------------
A    | [2, 3]
D    | [5, null]
     | [null]
(3 rows)
```

- json_object_agg(any, any)

  描述：将值聚集为json对象。

  返回类型：object-json

  示例：

```
gaussdb=# SELECT * FROM classes;
 name | score
------+-------
A    |     2
A    |     3
D    |     5
D    |
(4 rows)
```

```
gaussdb=# SELECT json_object_agg(name, score) FROM classes GROUP BY name ORDER BY name;
     json_object_agg
--------------------------
 { "A" : 2, "A" : 3 }
 { "D" : 5, "D" : null }
(2 rows)
```

- - jsonb_contained(jsonb, jsonb)

  描述：同操作符 `<@`, 判断$1中的所有元素是否在$2的顶层存在。

  返回类型：bool

  示例：

```
gaussdb=# SELECT jsonb_contained('[1,2,3]', '[1,2,3,4]');
 jsonb_contained
-----------------
 t
(1 row)
```

- - jsonb_contains(jsonb, jsonb)

  描述：同操作符 `@>`, 判断$1中的顶层所有元素是否包含在$2的所有元素。

  返回类型：bool

  示例：

```
gaussdb=# SELECT jsonb_contains('[1,2,3,4]', '[1,2,3]');
 jsonb_contains
----------------
```



```
 t
(1 row)
```

- - jsonb_exists(jsonb, text)

  描述：同操作符 `?`, 字符串$2是否存在$1的顶层以key\elem\scalar的形式存在。

  返回类型：bool

  示例：

```
gaussdb=# SELECT jsonb_exists('["1",2,3]', '1');
 jsonb_exists
--------------
 t
(1 row)
```

- - jsonb_exists_all(jsonb, text[])

  描述：同操作符 `?&`, 字符串数组$2里面，是否所有的元素，都在$1的顶层以key\elem\scalar的形式存在。

  返回类型：bool

  示例：

```
gaussdb=# SELECT jsonb_exists_all('["1","2",3]', '{1, 2}');
 jsonb_exists_all
------------------
 t
(1 row)
```

- - jsonb_exists_any(jsonb, text[])

  描述：同操作符 `?|`, 字符串数组$2里面，是否存在的元素，在$1的顶层以key\elem\scalar的形式存在。

  返回类型：bool

  示例：

```
gaussdb=# SELECT jsonb_exists_any('["1","2",3]', '{1, 2, 4}');
 jsonb_exists_any
------------------
 t
(1 row)
```

- - jsonb_cmp(jsonb, jsonb)

  描述：比较大小，正数代表大于，负数代表小于，0表示相等。

  返回类型：integer

  示例：

```
gaussdb=# SELECT jsonb_cmp('["a", "b"]', '{"a":1, "b":2}');
 jsonb_cmp
-----------
        -1
(1 row)
```

- - jsonb_eq(jsonb, jsonb)

  描述：同操作符 `=`, 比较两个值的大小。

  返回类型：bool

  示例：

```
gaussdb=# SELECT jsonb_eq('["a", "b"]', '{"a":1, "b":2}');
 jsonb_eq
----------
 f
(1 row)
```

- - jsonb_ne(jsonb, jsonb)

  描述：同操作符 `<>`, 比较两个值的大小。

  返回类型：bool

示例：

```
gaussdb=# SELECT jsonb_ne('["a", "b"]', '{"a":1, "b":2}');
 jsonb_ne
----------
 t
(1 row)
```

- - jsonb_gt(jsonb, jsonb)

  描述：同操作符 `>`, 比较两个值的大小。

  返回类型：bool

  示例：

  ```
  gaussdb=# SELECT jsonb_gt('["a", "b"]', '{"a":1, "b":2}');
   jsonb_gt
  ----------
   f
  (1 row)
  ```

- - jsonb_ge(jsonb, jsonb)

  描述：同操作符 `>=`, 比较两个值的大小。

  返回类型：bool

  示例：

  ```
  gaussdb=# SELECT jsonb_ge('["a", "b"]', '{"a":1, "b":2}');
   jsonb_ge
  ----------
   f
  (1 row)
  ```

- - jsonb_lt(jsonb, jsonb)

  描述：同操作符 `<`, 比较两个值的大小。

  返回类型：bool

  示例：

  ```
  gaussdb=# SELECT jsonb_lt('["a", "b"]', '{"a":1, "b":2}');
   jsonb_lt
  ----------
   t
  (1 row)
  ```

- - jsonb_le(jsonb, jsonb)

  描述：同操作符 `<=`, 比较两个值的大小。

  返回类型：bool

  示例：

  ```
  gaussdb=# SELECT jsonb_le('["a", "b"]', '{"a":1, "b":2}');
   jsonb_le
  ----------
   t
  (1 row)
  ```

- - to_json(anyelement)

  描述：把参数转换为`json`。

  返回类型：json

  示例：

  ```
  gaussdb=# SELECT to_json('{1,5}'::text[]);
    to_json
  -----------
   ["1","5"]
  (1 row)
  ```

- - jsonb_hash(jsonb)

  描述：对jsonb进行hash运算。

返回类型：integer

示例：

```
gaussdb=# SELECT jsonb_hash('[1,2,3]');
 jsonb_hash
------------
 -559968547
(1 row)
```

- - 其他函数

  描述：json\jsonb聚集函数所用到的内部函数，功能不过多赘述。

  ```
  json_agg_transfn
  json_agg_finalfn
  json_object_agg_transfn
  json_object_agg_finalfn
  ```

7.5.14 HLL 函数和操作符

哈希函数

- hll_hash_boolean(bool)

  描述：对bool类型数据计算哈希值。

  返回值类型：hll_hashval

  示例：

  ```
  gaussdb=# SELECT hll_hash_boolean(FALSE);
    hll_hash_boolean
  ---------------------
   -5451962507482445012
  (1 row)
  ```

- hll_hash_boolean(bool, int32)

  描述：设置hash seed（即改变哈希策略）并对bool类型数据计算哈希值。

  返回值类型：hll_hashval

  示例：

  ```
  gaussdb=# SELECT hll_hash_boolean(FALSE, 10);
    hll_hash_boolean
  --------------------
   -1169037589280886076
  (1 row)
  ```

- hll_hash_smallint(smallint)

  描述：对smallint类型数据计算哈希值。

  返回值类型：hll_hashval

  示例：

  ```
  gaussdb=# SELECT hll_hash_smallint(100::smallint);
    hll_hash_smallint
  ---------------------
   962727970174027904
  (1 row)
  ```

  说明

  数值大小相同的参数使用不同数据类型的哈希函数计算，最后结果会不一样，因为不同类型哈希函数会选取不同的哈希计算策略。

- hll_hash_smallint(smallint, int32)

  描述：设置hash seed（即改变哈希策略）同时对smallint类型数据计算哈希值。

  返回值类型：hll_hashval

示例：

```
gaussdb=# SELECT hll_hash_smallint(100::smallint, 10);
 hll_hash_smallint
--------------------
 -9056177146160443041
(1 row)
```

- hll_hash_integer(integer)

  描述：对integer类型数据计算哈希值。

  返回值类型：hll_hashval

  示例：

  ```
  gaussdb=# SELECT hll_hash_integer(0);
    hll_hash_integer
  ----------------------
   5156626420896634997
  (1 row)
  ```

- hll_hash_integer(integer, int32)

  描述：对integer类型数据计算哈希值，并设置hashseed（即改变哈希策略）。

  返回值类型：hll_hashval

  示例：

  ```
  gaussdb=# SELECT hll_hash_integer(0, 10);
   hll_hash_integer
  --------------------
   -5035020264353794276
  (1 row)
  ```

- hll_hash_bigint(bigint)

  描述：对bigint类型数据计算哈希值。

  返回值类型：hll_hashval

  示例：

  ```
  gaussdb=# SELECT hll_hash_bigint(100::bigint);
    hll_hash_bigint
  ---------------------
   -2401963681423227794
  (1 row)
  ```

- hll_hash_bigint(bigint, int32)

  描述：对bigint类型数据计算哈希值，并设置hashseed（即改变哈希策略）。

  返回值类型：hll_hashval

  示例：

  ```
  gaussdb=# SELECT hll_hash_bigint(100::bigint, 10);
    hll_hash_bigint
  ---------------------
   -2305749404374433531
  (1 row)
  ```

- hll_hash_bytea(bytea)

  描述：对bytea类型数据计算哈希值。

  返回值类型：hll_hashval

  示例：

  ```
  gaussdb=# SELECT hll_hash_bytea(E'\\x');
   hll_hash_bytea
  ----------------
   0
  (1 row)
  ```


- hll_hash_bytea(bytea, int32)

  描述：对bytea类型数据计算哈希值，并设置hashseed（即改变哈希策略）。

  返回值类型：hll_hashval

  示例：

```
gaussdb=# SELECT hll_hash_bytea(E'\\x', 10);
  hll_hash_bytea
---------------------
 7233188113542599437
(1 row)
```

- hll_hash_text(text)

  描述：对text类型数据计算哈希值。

  返回值类型：hll_hashval

  示例：

```
gaussdb=# SELECT hll_hash_text('AB');
   hll_hash_text
---------------------
 -5666002586880275174
(1 row)
```

- hll_hash_text(text, int32)

  描述：对text类型数据计算哈希值, 并设置hashseed（即改变哈希策略）。

  返回值类型：hll_hashval

  示例：

```
gaussdb=# SELECT hll_hash_text('AB', 10);
hll_hash_text
---------------------
-2215507121143724132
(1 row)
```

- hll_hash_any(anytype)

  描述：对任意类型数据计算哈希值。

  返回值类型：hll_hashval

  示例：

```
gaussdb=# SELECT hll_hash_any(1);
    hll_hash_any
----------------------
 -1316670585935156930
(1 row)

gaussdb=# SELECT hll_hash_any('08:00:2b:01:02:03'::macaddr);
    hll_hash_any
----------------------
 -3719950434455589360
(1 row)
```

- hll_hash_any(anytype, int32)

  描述：对任意类型数据计算哈希值，并设置hashseed（即改变哈希策略）。

  返回值类型：hll_hashval

  示例：

```
gaussdb=# SELECT hll_hash_any(1, 10);
    hll_hash_any
----------------------
 7048553517657992351
(1 row)
```

- hll_hashval_eq(hll_hashval, hll_hashval)

描述：比较两个hll_hashval类型数据是否相等。

返回值类型：bool

示例：

```
gaussdb=# SELECT hll_hashval_eq(hll_hash_integer(1), hll_hash_integer(1));
 hll_hashval_eq
----------------
 t
(1 row)
```

- hll_hashval_ne(hll_hashval, hll_hashval)

  描述：比较两个hll_hashval类型数据是否不相等。

  返回值类型：bool

  示例：

```
gaussdb=# SELECT hll_hashval_ne(hll_hash_integer(1), hll_hash_integer(1));
 hll_hashval_ne
----------------
 f
(1 row)
```

日志函数

hll主要存在三种模式Explicit、Sparse、Full。当数据规模比较小的时候会使用Explicit模式，这种模式下distinct值的计算是没有误差的；随着distinct值越来越多，hll会先后转换为Sparse模式和Full模式，这两种模式在计算结果上没有任何区别，只影响hll函数的计算效率和hll对象的存储空间。下面的函数可以用于查看hll的一些参数。

- hll_print(hll)

  描述：打印hll的一些debug参数信息。

  示例：

```
gaussdb=# SELECT hll_print(hll_empty());
                                  hll_print
-------------------------------------------------------------------------------
 type=1(HLL_EMPTY), log2m=14, log2explicit=10, log2sparse=12, duplicatecheck=0
(1 row)
```

- hll_type(hll)

  描述：查看当前hll的类型。返回值具体含义如下：返回值0，表示HLL_UNINIT，未初始化的hll对象；返回值1，表示HLL_EMPTY，hll空对象；返回值2，表示HLL_EXPLICIT，Explicit模式的hll对象；返回值3，表示HLL_SPARSE，Sparse模式的hll对象；返回值4，表示HLL_FULL，Full模式的hll对象；返回值5，表示HLL_UNDEFINED，不合法的hll对象。

  示例：

```
gaussdb=# SELECT hll_type(hll_empty());
 hll_type
----------
        1
(1 row)
```

- hll_log2m(hll)

  描述：查看当前hll数据结构中的log2m数值，log2m是分桶数的对数值，此值会影响最后hll计算distinct误差率，误差率计算公式为±1.04/√(2 ^ log2m)。当显式指定log2m的取值为10-16之间时，hll会设置分桶数为2^{log2m}。当显示指定log2explicit为-1时，会采用内置默认值。

  示例：

```
gaussdb=# SELECT hll_log2m(hll_empty());
 hll_log2m
```



```
-----------
       14
(1 row)

gaussdb=# SELECT hll_log2m(hll_empty(10));
 hll_log2m
-----------
       10
(1 row)

gaussdb=# SELECT hll_log2m(hll_empty(-1));
 hll_log2m
-----------
       14
(1 row)
```

- hll_log2explicit(hll)

  描述：查看当前hll数据结构中的log2explicit数值。hll通常会由Explicit模式到Sparse模式再到Full模式，这个过程称为promotion hierarchy策略。可以通过调整log2explicit值的大小改变策略，比如log2explicit为0的时候就会跳过Explicit模式而直接进入Sparse模式。当显式指定log2explicit的取值为1-12之间时，hll会在数据段长度超过$2^{log2explicit}$时转为Sparse模式。当显示指定log2explicit为-1时，会采用内置默认值。

  示例：

```
gaussdb=# SELECT hll_log2explicit(hll_empty());
 hll_log2explicit
------------------
              10
(1 row)

gaussdb=# SELECT hll_log2explicit(hll_empty(12, 8));
 hll_log2explicit
------------------
               8
(1 row)

gaussdb=# SELECT hll_log2explicit(hll_empty(12, -1));
 hll_log2explicit
------------------
              10
(1 row)
```

- hll_log2sparse(hll)

  描述：查看当前hll数据结构中的log2sparse数值。hll通常会由Explicit模式到Sparse模式再到Full模式，这个过程称为promotion hierarchy策略。可以通过调整log2sparse值的大小改变策略，比如log2sparse为0的时候就会跳过Sparse模式而直接进入Full模式。当显式指定Sparse的取值为1-14之间时，hll会在数据段长度超过$2^{log2sparse}$时转为Full模式。当显示指定log2sparse为-1时，会采用内置默认值。

  示例：

```
gaussdb=# SELECT hll_log2sparse(hll_empty());
 hll_log2sparse
----------------
            12
(1 row)

gaussdb=# SELECT hll_log2sparse(hll_empty(12, 8, 10));
 hll_log2sparse
----------------
            10
(1 row)

gaussdb=# SELECT hll_log2sparse(hll_empty(12, 8, -1));
```



```
 hll_log2sparse
----------------
             12
(1 row)
```

- hll_duplicatecheck(hll)

  描述：是否启用duplicatecheck，0是关闭，1是开启。默认关闭，对于有较多重复值出现的情况，可以开启以提高效率。当显示指定duplicatecheck为-1时，会采用内置默认值。

  示例：

```
gaussdb=# SELECT hll_duplicatecheck(hll_empty());
 hll_duplicatecheck
--------------------
                  0
(1 row)

gaussdb=# SELECT hll_duplicatecheck(hll_empty(12, 8, 10, 1));
 hll_duplicatecheck
--------------------
                  1
(1 row)

gaussdb=# SELECT hll_duplicatecheck(hll_empty(12, 8, 10, -1));
 hll_duplicatecheck
--------------------
                  0
(1 row)
```

功能函数

- hll_empty()

  描述：创建一个空的hll。

  返回值类型：hll

  示例：

```
gaussdb=# SELECT hll_empty();
                                                  hll_empty
-------------------------------------------------------------
 \x484c4c00000000002b050000000000000000000000000000000000000
(1 row)
```

- hll_empty(int32 log2m)

  描述：创建空的hll并指定参数log2m，取值范围是10到16。若输入-1，则采用内置默认值。

  返回值类型：hll

  示例：

```
gaussdb=# SELECT hll_empty(10);
                                                  hll_empty
-------------------------------------------------------------
 \x484c4c00000000002b040000000000000000000000000000000000000
(1 row)

gaussdb=# SELECT hll_empty(-1);
                                                  hll_empty
-------------------------------------------------------------
 \x484c4c00000000002b050000000000000000000000000000000000000
(1 row)
```

- hll_empty(int32 log2m, int32 log2explicit)

  描述：创建空的hll并依次指定参数log2m、log2explicit。log2explicit取值范围是0到12，0表示直接跳过Explicit模式。该参数可以用来设置Explicit模式的阈值大

小，在数据段长度达到$2^{log2explicit}$后切换为Sparse模式或者Full模式。若输入-1，则log2explicit采用内置默认值。

返回值类型: hll

示例：

```
gaussdb=# SELECT hll_empty(10, 4);
                                                            hll_empty
------------------------------------------------------------------------------------------------------------------
 \x484c4c0000000000130400000000000000000000000000000000000000000000
(1 row)

gaussdb=# SELECT hll_empty(10, -1);
                                                            hll_empty
------------------------------------------------------------------------------------------------------------------
 \x484c4c00000000002b040000000000000000000000000000000000000000000000
(1 row)
```

- hll_empty(int32 log2m, int32 log2explicit, int64 log2sparse)

  描述：创建空的hll并依次指定参数log2m、log2explicit、log2sparse。log2sparse取值范围是0到14，0表示直接跳过Sparse模式。该参数可以用来设置Sparse模式的阈值大小，在数据段长度达到$2^{log2sparse}$后切换为Full模式。若输入-1，则log2sparse采用内置默认值。

  返回值类型：hll

  示例：

  ```
  gaussdb=# SELECT hll_empty(10, 4, 8);
                                                              hll_empty
  ------------------------------------------------------------------------------------------------------------------
   \x484c4c0000000000120400000000000000000000000000000000000000000000
  (1 row)

  gaussdb=# SELECT hll_empty(10, 4, -1);
                                                              hll_empty
  ------------------------------------------------------------------------------------------------------------------
   \x484c4c0000000000130400000000000000000000000000000000000000000000
  (1 row)
  ```

- hll_empty(int32 log2m, int32 log2explicit, int64 log2sparse, int32 duplicatecheck)

  描述：创建空的hll并依次指定参数log2m、log2explicit、log2sparse、duplicatecheck。duplicatecheck取0或者1，表示是否开启该模式，默认情况下该模式会关闭。若输入-1，则duplicatecheck采用内置默认值。

  返回值类型：hll

  示例：

  ```
  gaussdb=# SELECT hll_empty(10, 4, 8, 0);
                                                              hll_empty
  ------------------------------------------------------------------------------------------------------------------
   \x484c4c0000000000120400000000000000000000000000000000000000000000
  (1 row)

  gaussdb=# SELECT hll_empty(10, 4, 8, -1);
                                                              hll_empty
  ------------------------------------------------------------------------------------------------------------------
   \x484c4c0000000000120400000000000000000000000000000000000000000000
  (1 row)
  ```

- hll_add(hll, hll_hashval)

  描述：把hll_hashval加入到hll中。

  返回值类型：hll

  示例：


```
gaussdb=# SELECT hll_add(hll_empty(), hll_hash_integer(1));
                                hll_add
-------------------------------------------------------------------------------
 \x484c4c08000002002b0900000000000000f03f3e2921ff133fbaed3e2921ff133fbaed00
(1 row)
```

- hll_add_rev(hll_hashval, hll)

  描述：把hll_hashval加入到hll中，和hll_add功能一样，只是参数位置进行了交换。

  返回值类型：hll

  示例：

```
gaussdb=# SELECT hll_add_rev(hll_hash_integer(1), hll_empty());
                            hll_add_rev
-------------------------------------------------------------------------------
 \x484c4c08000002002b0900000000000000f03f3e2921ff133fbaed3e2921ff133fbaed00
(1 row)
```

- hll_eq(hll, hll)

  描述：比较两个hll是否相等。

  返回值类型：bool

  示例：

```
gaussdb=# SELECT hll_eq(hll_add(hll_empty(), hll_hash_integer(1)), hll_add(hll_empty(),
hll_hash_integer(2)));
 hll_eq
--------
 f
(1 row)
```

- hll_ne(hll, hll)

  描述：比较两个hll是否不相等。

  返回值类型：bool

  示例：

```
gaussdb=# SELECT hll_ne(hll_add(hll_empty(), hll_hash_integer(1)), hll_add(hll_empty(),
hll_hash_integer(2)));
 hll_ne
--------
 t
(1 row)
```

- hll_cardinality(hll)

  描述：计算hll的distinct值。

  返回值类型：int

  示例：

```
gaussdb=# SELECT hll_cardinality(hll_empty() || hll_hash_integer(1));
 hll_cardinality
-----------------
               1
(1 row)
```

- hll_union(hll, hll)

  描述：把两个hll数据结构union成一个。

  返回值类型：hll

  示例：

```
gaussdb=# SELECT hll_union(hll_add(hll_empty(), hll_hash_integer(1)), hll_add(hll_empty(),
hll_hash_integer(2)));
                                    hll_union
-------------------------------------------------------------------------------------------
```



```
\x484c4c10002000002b0900000000000000004000000000000000000b3ccc49320cca1ae3e2921ff133fbaed00
(1 row)
```

聚合函数

- hll_add_agg(hll_hashval)

  描述：把哈希后的数据按照分组放到hll中。

  返回值类型：hll

  示例：

```
--准备数据
gaussdb=# CREATE TABLE t_id(id int);
gaussdb=# INSERT INTO t_id VALUES(generate_series(1,500));
gaussdb=# CREATE TABLE t_data(a int, c text);
gaussdb=# INSERT INTO t_data SELECT mod(id,2), id FROM t_id;

--创建表并指定列为hll
gaussdb=# CREATE TABLE t_a_c_hll(a int, c hll);

--根据a列GROUP BY对数据分组，把各组数据加到hll中
gaussdb=# INSERT INTO t_a_c_hll SELECT a, hll_add_agg(hll_hash_text(c)) FROM t_data GROUP BY a;

--得到每组数据中hll的Distinct值
gaussdb=# SELECT a, #c AS cardinality FROM t_a_c_hll ORDER BY a;
 a |   cardinality
---+------------------
 0 | 247.862354346299
 1 | 250.908710610377
(2 rows)
```

- hll_add_agg(hll_hashval, int32 log2m)

  描述：把哈希后的数据按照分组放到hll中， 并指定参数log2m，取值范围是10到16。若输入-1或者NULL，则采用内置默认值。

  返回值类型：hll

  示例：

```
gaussdb=# SELECT hll_cardinality(hll_add_agg(hll_hash_text(c), 12)) FROM t_data;
 hll_cardinality
------------------
 497.965240179228
(1 row)
```

- hll_add_agg(hll_hashval, int32 log2m, int32 log2explicit)

  描述：把哈希后的数据按照分组放到hll中，依次指定参数log2m、log2explicit。log2explicit取值范围是0到12，0表示直接跳过Explicit模式。该参数可以用来设置Explicit模式的阈值大小，在数据段长度达到$2^{log2explicit}$后切换为Sparse模式或者Full模式。若输入-1或者NULL，则log2explicit采用内置默认值。

  返回值类型：hll

  示例：

```
gaussdb=# SELECT hll_cardinality(hll_add_agg(hll_hash_text(c), NULL, 1)) FROM t_data;
 hll_cardinality
------------------
 498.496062953313
(1 row)
```

- hll_add_agg(hll_hashval, int32 log2m, int32 log2explicit, int64 log2sparse)

  描述：把哈希后的数据按照分组放到hll中， 依次指定参数log2m、log2explicit、log2sparse。log2sparse取值范围是0到14，0表示直接跳过Sparse模式。该参数可以用来设置Sparse模式的阈值大小，在数据段长度达到$2^{log2sparse}$后切换为Full模式。若输入-1或者NULL，则log2sparse采用内置默认值。

返回值类型：hll

示例：

```
gaussdb=# SELECT hll_cardinality(hll_add_agg(hll_hash_text(c), NULL, 6, 10)) FROM t_data;
 hll_cardinality
------------------
 498.496062953313
(1 row)
```

- hll_add_agg(hll_hashval, int32 log2m, int32 log2explicit, int64 log2sparse, int32 duplicatecheck)

  描述：把哈希后的数据按照分组放到hll中, 依次指定参数log2m、log2explicit、log2sparse、duplicatecheck，duplicatecheck取值范围是0或者1，表示是否开启该模式，默认情况下该模式会关闭。若输入-1或者NULL，则duplicatecheck采用内置默认值。

  返回值类型：hll

  示例：

```
gaussdb=# SELECT hll_cardinality(hll_add_agg(hll_hash_text(c), NULL, 6, 10, -1)) FROM t_data;
 hll_cardinality
------------------
 498.496062953313
(1 row)
```

- hll_union_agg(hll)

  描述：将多个hll类型数据union成一个hll。

  返回值类型：hll

  示例：

```
--将各组中的hll数据union成一个hll，并计算distinct值。
gaussdb=# SELECT #hll_union_agg(c) AS cardinality FROM t_a_c_hll;
   cardinality
------------------
 498.496062953313
(1 row)

--删除表
gaussdb=# DROP TABLE t_id;
gaussdb=# DROP TABLE t_data;
gaussdb=# DROP TABLE t_a_c_hll;
```

说明

注意：当两个或者多个hll数据结构做union的时候，必须要保证其中每一个hll里面的精度参数一样，否则将不可以进行union。同样的约束也适用于函数hll_union(hll,hll)。

废弃函数

由于版本升级，HLL（HyperLogLog）有一些旧的函数废弃，用户可以用类似的函数进行替代。

- hll_schema_version(hll)

  描述：查看当前hll中的schema version。旧版本schema version是常值1，用来进行hll字段的头部校验，重构后的hll在头部增加字段“HLL”进行校验，schema version不再使用。

- hll_regwidth(hll)

  描述：查看hll数据结构中桶的位数大小。旧版本桶的位数regwidth取值1~5，会存在较大的误差，也限制了基数估计上限。 重构后regwidth为固定值6，不再使用regwidth变量。

- hll_expthresh(hll)

  描述：得到当前hll中expthresh大小。采用hll_log2explicit(hll)替代类似功能。

- hll_sparseon(hll)

  描述：是否启用Sparse模式。采用hll_log2sparse(hll)替代类似功能，0表示关闭Sparse模式。

内置函数

HLL（HyperLogLog）有一系列内置函数用于内部对数据进行处理，一般情况下用户不需要熟知这些函数的使用。详情见**表7-41**。

表 7-41 内置函数

函数名称	功能描述
hll_in	以string格式接收hll数据。
hll_out	以string格式发送hll数据。
hll_recv	以bytea格式接收hll数据。
hll_send	以bytea格式发送hll数据。
hll_trans_in	以string格式接收hll_trans_type数据。
hll_trans_out	以string格式发送hll_trans_type数据。
hll_trans_recv	以bytea形式接收hll_trans_type数据。
hll_trans_send	以bytea形式发送hll_trans_type数据。
hll_typmod_in	接收typmod类型数据。
hll_typmod_out	发送typmod类型数据。
hll_hashval_in	接收hll_hashval类型数据。
hll_hashval_out	发送hll_hashval类型数据。
hll_add_trans0	类似于hll_add所提供的功能，初始化时无指定入参，通常在聚合运算的第一阶段DN上使用。
hll_add_trans1	类似于hll_add所提供的功能，初始化时指定一个入参，通常在聚合运算的第一阶段DN上使用。
hll_add_trans2	类似于hll_add所提供的功能，初始化时指定两个入参，通常在聚合运算的第一阶段DN上使用。
hll_add_trans3	类似于hll_add所提供的功能，初始化时指定三个入参，通常在聚合运算的第一阶段DN上使用。
hll_add_trans4	类似于hll_add所提供的功能，初始化时指定四个入参，通常在聚合运算的第一阶段DN上使用。
hll_union_trans	类似hll_union所提供的功能，在聚合运算的第一阶段DN上使用。

函数名称	功能描述
hll_union_collect	类似于hll_union所提供的功能，在聚合运算第二阶段DN上使用，汇总各个DN上的结果。
hll_pack	在聚合运算第三阶段DN上使用，把自定义hll_trans_type类型最后转换成hll类型。
hll	用于hll类型转换成hll类型，根据输入参数会设定指定参数。
hll_hashval	用于bigint类型转换成hll_hashval类型。
hll_hashval_int4	用于int4类型转换成hll_hashval类型。

操作符

- =

  描述：比较hll或hll_hashval的值是否相等。

  返回值类型：bool

  示例：

```
--hll
gaussdb=# SELECT (hll_empty() || hll_hash_integer(1)) = (hll_empty() || hll_hash_integer(1));
column
----------
 t
(1 row)

--hll_hashval
gaussdb=# SELECT hll_hash_integer(1) = hll_hash_integer(1);
 ?column?
----------
 t
(1 row)
```

- <> or !=

  描述：比较hll或hll_hashval是否不相等。

  返回值类型：bool

  示例：

```
--hll
gaussdb=# SELECT (hll_empty() || hll_hash_integer(1)) <> (hll_empty() || hll_hash_integer(2));
 ?column?
----------
 t
(1 row)

--hll_hashval
gaussdb=# SELECT hll_hash_integer(1) <> hll_hash_integer(2);
 ?column?
----------
 t
(1 row)
```

- ||

  描述：可代表hll_add, hll_union, hll_add_rev三个函数的功能。

  返回值类型：hll

  示例：

```
--hll_add
gaussdb=# SELECT hll_empty() || hll_hash_integer(1);
```



```
                              ?column?
-------------------------------------------------------------------------------
 \x484c4c08000002002b0900000000000000f03f3e2921ff133fbaed3e2921ff133fbaed00
(1 row)

--hll_add_rev
gaussdb=# SELECT hll_hash_integer(1) || hll_empty();
                              ?column?
-------------------------------------------------------------------------------
 \x484c4c08000002002b0900000000000000f03f3e2921ff133fbaed3e2921ff133fbaed00
(1 row)

--hll_union
gaussdb=# SELECT (hll_empty() || hll_hash_integer(1)) || (hll_empty() || hll_hash_integer(2));
                                        ?column?
-----------------------------------------------------------------------------------------------

\x484c4c10002000002b0900000000000000004000000000000000000b3ccc49320cca1ae3e2921ff133fba
ed00
(1 row)
```

- #

  描述：计算出hll的distinct值， 同hll_cardinality函数。

  返回值类型：int

  示例：

```
gaussdb=# SELECT #(hll_empty() || hll_hash_integer(1));
 ?column?
----------
        1
(1 row)
```

7.5.15 SEQUENCE 函数

序列函数为用户从序列对象中获取后续的序列值提供了简单的多用户安全的方法。

- nextval(regclass)

  描述：递增序列并返回新值。

  说明

  为了避免从同一个序列获取值的并发事务被阻塞，nextval操作不会回滚；即一旦值被抓取，就认为它已经被用过，并且不会再被返回。即使该操作处于事务中，当事务之后中断，或者如果调用查询结束不使用该值，也是如此。这种情况将在指定值的顺序中留下未使用的“空洞”。因此，GaussDB序列对象不能用于获得“无间隙”序列。

  须知

  nextval函数只能在主机上执行，备机不支持执行此函数。

  返回类型：numeric

  nextval函数有两种调用方式（其中第二种调用方式目前不支持Sequence命名中有特殊字符"."的情况），如下：

```
gaussdb=# CREATE SEQUENCE seqDemo;
--示例1：
gaussdb=# SELECT nextval('seqDemo');
 nextval
---------
       1
(1 row)
--示例2：
```



```
gaussdb=# SELECT seqDemo.nextval;
 nextval
---------
       2
(1 row)
gaussdb=# DROP SEQUENCE seqDemo;
```

- currval(regclass)

  描述：返回当前会话里最近一次nextval返回的数值。如果当前会话还没有调用过指定的sequence的nextval，那么调用currval将会报错。

  返回类型：numeric

  currval函数有两种调用方式（其中第二种调用方式目前不支持Sequence命名中有特殊字符"."的情况），如下：

```
gaussdb=# CREATE SEQUENCE seq1;
gaussdb=# SELECT nextval('seq1');
--示例1：
gaussdb=# SELECT currval('seq1');
 currval
---------
       1
(1 row)
--示例2：
gaussdb=# SELECT seq1.currval;
 currval
---------
       1
(1 row)
gaussdb=# DROP SEQUENCE seq1;
```

- lastval()

  描述：返回当前会话里最近一次nextval返回的数值。这个函数等效于currval，只是它不用序列名为参数，它抓取当前会话里面最近一次nextval使用的序列。如果当前会话还没有调用过nextval，那么调用lastval将会报错。

  返回类型：numeric

  示例：

```
gaussdb=# CREATE SEQUENCE seq1;
gaussdb=# SELECT nextval('seq1');
gaussdb=# SELECT lastval();
 lastval
---------
       1
(1 row)
gaussdb=# DROP SEQUENCE seq1;
```

- setval(regclass, numeric)

  描述：设置序列的当前数值。

  返回类型：numeric

  示例：

```
gaussdb=# CREATE SEQUENCE seqDemo;
gaussdb=# SELECT nextval('seqDemo');
gaussdb=# SELECT setval('seqDemo',5);
 setval
--------
      5
(1 row)
gaussdb=# DROP SEQUENCE seqDemo;
```

- setval(regclass, numeric, Boolean)

  描述：设置序列的当前数值以及is_called标志。

  返回类型：numeric

示例：

```
gaussdb=# CREATE SEQUENCE seqDemo;
gaussdb=# SELECT nextval('seqDemo');
gaussdb=# SELECT setval('seqDemo',5,true);
 setval
--------
      5
(1 row)
gaussdb=# DROP SEQUENCE seqDemo;
```

说明

Setval后当前会话会立刻生效，但如果其他会话有缓存的序列值，只能等到缓存值用尽才能感知Setval的作用。所以为了避免序列值冲突，setval要谨慎使用。

因为序列是非事务的，setval造成的改变不会由于事务的回滚而撤销。

须知

nextval函数只能在主机上执行，备机不支持执行此函数。

- pg_sequence_last_value(sequence_oid oid, OUT cache_value int16, OUT last_value int16)

  描述：获取指定sequence的参数，包含缓存值、当前值。

  返回类型：int16，int16

- gs_get_sequence_last_value(sequence_oid oid, OUT cache_value int16, OUT last_value int16)

  描述：获取指定sequence的参数，包含缓存值、当前值。该函数针对无权限的sequence返回null值。

  返回类型：int16，int16

- last_insert_id()

  描述：获取最近一次为自动增长列成功插入的第一个自动生成的值。

  返回类型：int16

- last_insert_id(int16)

  描述：设置下一次last_insert_id()函数的返回值，并返回此值。若参数为NULL，将下一次last_insert_id()函数的返回值设为0，此函数返回NULL。

  返回值类型：int16

  示例：

```
gaussdb=# create database b_format_db with dbcompatibility = 'b';
gaussdb=# \c b_format_db;
b_format_db=#SELECT last_insert_id(100);
 last_insert_id
----------------
            100
(1 row)
b_format_db=#SELECT last_insert_id();
 last_insert_id
----------------
            100
(1 row)
b_format_db=# \c postgres;
gaussdb=# DROP DATABASE b_format_db;
```

说明

- last_insert_id()和last_insert_id(int16)是会话级别的函数，若当前会话未对自动增长列插入任何数据，last_insert_id()返回值为0。
- last_insert_id()和last_insert_id(int16)仅在参数sql_compatibility='B'时可用。

7.5.16 数组函数和操作符

数组操作符

- =

  描述：两个数组是否相等。

  示例：

```
gaussdb=# SELECT ARRAY[1.1,2.1,3.1]::int[] = ARRAY[1,2,3] AS RESULT ;
 result
--------
 t
(1 row)
```

- <>

  描述：两个数组是否不相等。

  示例：

```
gaussdb=# SELECT ARRAY[1,2,3] <> ARRAY[1,2,4] AS RESULT;
 result
--------
 t
(1 row)
```

- <

  描述：一个数组是否小于另一个数组。

  示例：

```
gaussdb=# SELECT ARRAY[1,2,3] < ARRAY[1,2,4] AS RESULT;
 result
--------
 t
(1 row)
```

- >

  描述：一个数组是否大于另一个数组。

  示例：

```
gaussdb=# SELECT ARRAY[1,4,3] > ARRAY[1,2,4] AS RESULT;
 result
--------
 t
(1 row)
```

- <=

  描述：一个数组是否小于或等于另一个数组。

  示例：

```
gaussdb=# SELECT ARRAY[1,2,3] <= ARRAY[1,2,3] AS RESULT;
 result
--------
 t
(1 row)
```

- >=

  描述：一个数组是否大于或等于另一个数组。

示例：

```
gaussdb=# SELECT ARRAY[1,4,3] >= ARRAY[1,4,3] AS RESULT;
 result
--------
 t
(1 row)
```

- @>

  描述：一个数组是否包含另一个数组。

  示例：

  ```
  gaussdb=# SELECT ARRAY[1,4,3] @> ARRAY[3,1] AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- <@

  描述：一个数组是否被包含于另一个数组。

  示例：

  ```
  gaussdb=# SELECT ARRAY[2,7] <@ ARRAY[1,7,4,2,6] AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- &&

  描述：一个数组是否和另一个数组重叠（有共同元素）。

  示例：

  ```
  gaussdb=# SELECT ARRAY[1,4,3] && ARRAY[2,1] AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- ||

  描述：数组与数组进行连接。

  示例：

  ```
  gaussdb=# SELECT ARRAY[1,2,3] || ARRAY[4,5,6] AS RESULT;
     result
  ---------------
   {1,2,3,4,5,6}
  (1 row)
  gaussdb=# SELECT ARRAY[1,2,3] || ARRAY[[4,5,6],[7,8,9]] AS RESULT;
            result
  -------------------------
   {{1,2,3},{4,5,6},{7,8,9}}
  (1 row)
  ```

- ||

  描述：元素与数组进行连接。

  示例：

  ```
  gaussdb=# SELECT 3 || ARRAY[4,5,6] AS RESULT;
    result
  -----------
   {3,4,5,6}
  (1 row)
  ```

- ||

  描述：数组与元素进行连接。

  示例：

```
gaussdb=# SELECT ARRAY[4,5,6] || 7 AS RESULT;
  result
-----------
 {4,5,6,7}
(1 row)
```

数组比较是使用默认的B-tree比较函数对所有元素逐一进行比较的。多维数组的元素按照行顺序进行访问。如果两个数组的内容相同但维数不等，决定排序顺序的首要因素是维数。

数组函数

- array_append(anyarray, anyelement)

  描述：向数组末尾添加元素，只支持一维数组。

  返回类型：anyarray

  示例：

  ```
  gaussdb=# SELECT array_append(ARRAY[1,2], 3) AS RESULT;
   result
  ---------
   {1,2,3}
  (1 row)
  ```

- array_prepend(anyelement, anyarray)

  描述：向数组开头添加元素，只支持一维数组。

  返回类型：anyarray

  示例：

  ```
  gaussdb=# SELECT array_prepend(1, ARRAY[2,3]) AS RESULT;
   result
  ---------
   {1,2,3}
  (1 row)
  ```

- array_cat(anyarray, anyarray)

  描述：连接两个数组，支持多维数组。

  返回类型：anyarray

  示例：

  ```
  gaussdb=# SELECT array_cat(ARRAY[1,2,3], ARRAY[4,5]) AS RESULT;
     result
  -------------
   {1,2,3,4,5}
  (1 row)

  gaussdb=# SELECT array_cat(ARRAY[[1,2],[4,5]], ARRAY[6,7]) AS RESULT;
        result
  ---------------------
   {{1,2},{4,5},{6,7}}
  (1 row)
  ```

- array_union(anyarray, anyarray)

  描述：连接两个数组，只支持一维数组。有入参为NULL时返回另一个入参。

  返回类型：anyarray

  示例：

  ```
  gaussdb=# SELECT array_union(ARRAY[1,2,3], ARRAY[3,4,5]) AS RESULT;
     result
  -------------
   {1,2,3,3,4,5}
  (1 row)
  ```



```
gaussdb=# SELECT array_union(ARRAY[1,2,3], NULL) AS RESULT;
 result
---------
 {1,2,3}
(1 row)
```

- array_union_distinct(anyarray, anyarray)

  描述：连接两个数组，并去重，只支持一维数组。有入参为NULL时返回另一个入参。

  返回类型：anyarray

  示例：

```
gaussdb=# SELECT array_union_distinct(ARRAY[1,2,3], ARRAY[3,4,5]) AS RESULT;
   result
-------------
 {1,2,3,4,5}
(1 row)

gaussdb=# SELECT array_union_distinct(ARRAY[1,2,3], NULL) AS RESULT;
 result
---------
 {1,2,3}
(1 row)
```

- array_intersect(anyarray, anyarray)

  描述：两个数组取交集，只支持一维数组。有入参为NULL时返回NULL。

  返回类型：anyarray

  示例：

```
gaussdb=# SELECT array_intersect(ARRAY[1,2,3], ARRAY[3,4,5]) AS RESULT;
   result
-------------
 {3}
(1 row)

gaussdb=# SELECT array_intersect(ARRAY[1,2,3], NULL) AS RESULT;
 result
--------

(1 row)
```

- array_intersect_distinct(anyarray, anyarray)

  描述：两个数组取交集，并去重，只支持一维数组。有入参为NULL时返回NULL。

  返回类型：anyarray

  示例：

```
gaussdb=# SELECT array_intersect_distinct(ARRAY[1,2,2], ARRAY[2,2,4,5]) AS RESULT;
   result
-------------
 {2}
(1 row)

gaussdb=# SELECT array_intersect_distinct(ARRAY[1,2,3], NULL) AS RESULT;
 result
--------

(1 row)
```

- array_except(anyarray, anyarray)

  描述：两个数组取差，只支持一维数组。第一个入参为NULL时返回NULL，第二个入参为NULL时返回第一个入参。

  返回类型：anyarray

示例：

```
gaussdb=# SELECT array_except(ARRAY[1,2,3], ARRAY[3,4,5]) AS RESULT;
   result
-------------
 {1,2}
(1 row)

gaussdb=# SELECT array_except(ARRAY[1,2,3], NULL) AS RESULT;
 result
---------
 {1,2,3}
(1 row)

gaussdb=# SELECT array_except(NULL, ARRAY[3,4,5]) AS RESULT;
 result
--------

(1 row)
```

- array_except_distinct(anyarray, anyarray)

  描述：两个数组取差，并去重，只支持一维数组。第一个入参为NULL时返回NULL，第二个入参为NULL时返回第一个入参。

  返回类型：anyarray

  示例：

```
gaussdb=# SELECT array_except_distinct(ARRAY[1,2,2,3], ARRAY[3,4,5]) AS RESULT;
   result
-------------
 {1,2}
(1 row)

gaussdb=# SELECT array_except_distinct(ARRAY[1,2,3], NULL) AS RESULT;
 result
---------
 {1,2,3}
(1 row)

gaussdb=# SELECT array_except_distinct(NULL, ARRAY[3,4,5]) AS RESULT;
 result
--------

(1 row)
```

- array_ndims(anyarray)

  描述：返回数组的维数。

  返回类型：int

  示例：

```
gaussdb=# SELECT array_ndims(ARRAY[[1,2,3], [4,5,6]]) AS RESULT;
 result
--------
      2
(1 row)
```

- array_dims(anyarray)

  描述：返回数组各个维度中的低位下标值和高位下标值。

  返回类型：text

  示例：

```
gaussdb=# SELECT array_dims(ARRAY[[1,2,3], [4,5,6]]) AS RESULT;
   result
------------
 [1:2][1:3]
(1 row)
```


- array_length(anyarray, int)

  描述：返回指定数组维度的长度。int为指定数组维度。

  返回类型：int

  示例：

```
gaussdb=# SELECT array_length(array[1,2,3], 1) AS RESULT;
 result
--------
      3
(1 row)

gaussdb=# SELECT array_length(array[[1,2,3],[4,5,6]], 2) AS RESULT;
 result
--------
      3
(1 row)
```

- array_lower(anyarray, int)

  描述：返回指定数组维数的下界。int为指定数组维度。

  返回类型：int

  示例：

```
gaussdb=# SELECT array_lower('[0:2]={1,2,3}'::int[], 1) AS RESULT;
 result
--------
      0
(1 row)
```

  说明

  如果第一个参数为null开启参数varray_compat后会报错Reference to uninitialized collection，开启前返回NULL。

- array_upper(anyarray, int)

  描述：返回指定数组维数的上界。int为指定数组维度。

  返回类型：int

  示例：

```
gaussdb=# SELECT array_upper(ARRAY[1,8,3,7], 1) AS RESULT;
 result
--------
      4
(1 row)
```

  说明

  如果第一个参数为null开启参数varray_compat后会报错Reference to uninitialized collection，开启前返回NULL。

- array_to_string(anyarray, text [, text])

  描述：使用第一个text作为数组的新分隔符，使用第二个text替换数组值为null的值。

  返回类型：text

  示例：

```
gaussdb=# SELECT array_to_string(ARRAY[1, 2, 3, NULL, 5], ',', '*') AS RESULT;
  result
-----------
 1,2,3,*,5
(1 row)
```

- array_delete(anyarray)

  描述：清空数组中的元素并返回一个同类型的空数组。

返回类型：anyarray

示例：

```
gaussdb=# SELECT array_delete(ARRAY[1,8,3,7]) AS RESULT;
 result
--------
 {}
(1 row)
```

说明

如果第一个参数为null开启参数varray_compat后会报错Reference to uninitialized collection，开启前返回NULL。

- array_deleteidx(anyarray, int)

  描述：从数组中删除指定下标的元素并返回剩余元素组成的数组。

  返回类型：anyarray

  示例：

```
gaussdb=# SELECT array_deleteidx(ARRAY[1,2,3,4,5], 1) AS RESULT;
  result
-----------
 {2,3,4,5}
(1 row)
```

  说明

  - array_deleteidx(anyarray, int)此函数在参数a_format_version值为10c和a_format_dev_version值为s1的情况下被禁用。
  - 在开启varray_compat参数后，如果第一个参数为null会报错Reference to uninitialized collection，第二个参数为null返回原数组，未开启该参数时，参数有一个为null则返回null；若第二个参数小于等于0时开启参数后会报错Subscript outside of limit，开启前返回原数组；若第二个参数大于该数组元素数量时(包括0即空数组)开启参数后会报错Subcript outside of count开启前返回原数组。

- array_extendnull(anyarray, int)

  描述：往数组尾部添加指定个数的null空元素。

  返回类型：anyarray

  示例：

```
gaussdb=# SELECT array_extendnull(ARRAY[1,8,3,7],1) AS RESULT;
    result
--------------
{1,8,3,7,null}
(1 row)
```

  说明

  如果第一个参数为null开启参数varray_compat后会报错Reference to uninitialized collection，开启前返回NULL；如果第二个参数为NULL，开启参数前返回NULL，开启后返回原数组；若第二个参数小于0开启参数后报错numeric or value error开启前返回原数组。

- array_extendnull(anyarray, int, int)

  描述：往数组尾部添加指定个数的指定索引的元素。

  返回类型：anyarray

  示例：

```
-- 设置参数
gaussdb=# set a_format_version = '10c';
gaussdb=# set a_format_dev_version = 's1';

gaussdb=# SELECT array_extendnull(ARRAY[1,8,3,7],2,2) AS RESULT;
```



```
   result
--------------
{1,8,3,7,8,8}
(1 row)

gaussdb=# set a_format_version = '10c';
gaussdb=# set a_format_dev_version = 's1';
```

说明

- array_extendnull(anyarray, int, int)此函数在参数a_format_version值为10c和a_format_dev_version值为s1的情况下有效。
- 如果第一个参数为null开启参数varray_compat后会报错Reference to uninitialized collection，开启前返回NULL；如果第二个参数或者第三个参数为NULL，开启参数前返回NULL，开启后返回原数组。

- array_trim(anyarray, int)

  描述：从数组尾部删除指定个数个元素。

  返回类型：anyarray

  示例：

```
gaussdb=# SELECT array_trim(ARRAY[1,8,3,7],1) AS RESULT;
 result
---------
 {1,8,3}
(1 row)
```

  说明

  如果第一个参数为null开启参数varray_compat后会报错Reference to uninitialized collection，开启前返回NULL；如果第二个参数为NULL，开启参数前返回NULL，开启后返回原数组；如果第二个参数超过数组元素个数(包括0即空数组)时开启参数后会报错Subscipt outside of count，开启前返回空数组，如果第二个参数小于0开启后报错numeric or value error，开启前返回原数组。

- array_exists(anyarray, int)

  描述：检查第二个参数是否是数组的合法下标。

  返回类型：boolean

  示例：

```
gaussdb=# SELECT array_exists(ARRAY[1,8,3,7],1) AS RESULT;
 result
--------
 t
(1 row)
```

- array_next(anyarray, int)

  描述：根据第二个入参返回数组中指定下标元素的下一个元素的下标。

  返回类型：int

  示例：

```
gaussdb=# SELECT array_next(ARRAY[1,8,3,7],1) AS RESULT;
 result
--------
      2
(1 row)
```

  说明

  如果第一个参数为null开启参数varray_compat后会报错Reference to uninitialized collection，开启前返回NULL。

- array_prior(anyarray, int)

描述：根据第二个入参返回数组中指定下标元素的上一个元素的下标。

返回类型：int

示例：

```
gaussdb=# SELECT array_prior(ARRAY[1,8,3,7],2) AS RESULT;
 result
--------
  1
(1 row)
```

说明

如果第一个参数为null开启参数varray_compat后会报错Reference to uninitialized collection，开启前返回NULL。

- string_to_array(text, text [, text])

  描述：使用第二个text指定分隔符，使用第三个可选的text作为NULL值替换模板，如果分隔后的子串与第三个可选的text完全匹配，则将其替换为NULL。

  返回类型：text[]

  示例：

```
gaussdb=# SELECT string_to_array('xx~^~yy~^~zz', '~^~', 'yy') AS RESULT;
    result
---------------
 {xx,NULL,zz}
(1 row)
gaussdb=# SELECT string_to_array('xx~^~yy~^~zz', '~^~', 'y') AS RESULT;
   result
------------
 {xx,yy,zz}
(1 row)
```

- unnest(anyarray)

  描述：扩大一个数组为一组行。

  返回类型：setof anyelement

  示例：

```
gaussdb=# SELECT unnest(ARRAY[1,2]) AS RESULT;
 result
--------
      1
      2
(2 rows)
```

- unnest(anynesttable)

  描述：返回nesttable中的元素集合。

  返回类型：setof anyelement

  约束：不支持tableof类型嵌套tableof类型或者tableof嵌套其他类型再嵌套tableof类型的情况。

  示例：

```
CREATE OR REPLACE PROCEDURE f1()
AS
    TYPE t1 IS TABLE of INT;
    v2 t1 := t1(null, 2, 3, 4, null);
    tmp INT;
    CURSOR c1 IS SELECT * FROM unnest(v2);
BEGIN
OPEN c1;
FOR i IN 1 .. v2.count LOOP
    FETCH c1 INTO tmp;
    IF tmp IS null THEN
```



```
        dbe_output.print_line(i || ': is null');
    ELSE
        dbe_output.print_line(i || ': ' || tmp);
    END IF;
END LOOP;
CLOSE c1;
END;
/

gaussdb=# CALL f1();
1: is null
2: 2
3: 3
4: 4
5: is null
 f1
----

(1 row)
```

- unnest(anyindexbytable)

  描述：返回table of index by类型根据index排序后的元素集合。

  返回类型：setof anyelement

  约束：不支持tableof类型嵌套tableof类型或者tableof嵌套其他类型再嵌套tableof类型的情况。只支持index by int类型，不支持index by varchar类型。

  示例：

```
CREATE OR REPLACE PROCEDURE f1()
AS
    TYPE t1 IS TABLE of INT INDEX BY INT;
    v2 t1 := t1(1=>1, -10=>(-10), 6=>6, 4=>null);
    tmp INT;
    CURSOR c1 IS SELECT * FROM unnest(v2);
BEGIN
OPEN c1;
FOR i IN 1 .. v2.count LOOP
    FETCH c1 INTO tmp;
    IF tmp IS null THEN
        dbe_output.print_line(i || ': is null');
    ELSE
        dbe_output.print_line(i || ': ' || tmp);
    END IF;
END LOOP;
CLOSE c1;
END;
/

gaussdb=# CALL f1();
1: -10
2: 1
3: is null
4: 6
 f1
----

(1 row)
```

在string_to_array中，如果分隔符参数是NULL，输入字符串中的每个字符将在结果数组中变成一个独立的元素。如果分隔符是一个空白字符串，则整个输入的字符串将变为一个元素的数组。否则输入字符串将在每个分隔字符串处分开。

在string_to_array中，如果省略NULL字符串参数或为NULL，将字符串中没有输入内容的子串替换为NULL。

在array_to_string中，如果省略NULL字符串参数或为NULL，运算中将跳过在数组中的任何NULL元素，并且不会在输出字符串中出现。

- _pg_keysequal
  描述：判断两个smallint数组是否相同。
  参数：smallint[], smallint[]
  返回值类型：boolean

  说明

  此函数存在于information_schema命名空间。

7.5.17 范围函数和操作符

范围操作符

- =
  描述：等于。
  示例：

```
gaussdb=# SELECT int4range(1,5) = '[1,4]'::int4range AS RESULT;
 result
--------
 t
(1 row)
```

- <>
  描述：不等于。
  示例：

```
gaussdb=# SELECT numrange(1.1,2.2) <> numrange(1.1,2.3) AS RESULT;
 result
--------
 t
(1 row)
```

- <
  描述：小于。
  示例：

```
gaussdb=# SELECT int4range(1,10) < int4range(2,3) AS RESULT;
 result
--------
 t
(1 row)
```

- >
  描述：大于。
  示例：

```
gaussdb=# SELECT int4range(1,10) > int4range(1,5) AS RESULT;
 result
--------
 t
(1 row)
```

- <=
  描述：小于或等于。
  示例：

```
gaussdb=# SELECT numrange(1.1,2.2) <= numrange(1.1,2.2) AS RESULT;
 result
--------
 t
(1 row)
```


- >=

  描述：大于或等于。

  示例：

  ```
  gaussdb=# SELECT numrange(1.1,2.2) >= numrange(1.1,2.0) AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- @>

  描述：包含范围。

  示例：

  ```
  gaussdb=# SELECT int4range(2,4) @> int4range(2,3) AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- @>

  描述：包含元素。

  示例：

  ```
  gaussdb=# SELECT '[2011-01-01,2011-03-01)'::tsrange @> '2011-01-10'::timestamp AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- <@

  描述：范围包含于。

  示例：

  ```
  gaussdb=# SELECT int4range(2,4) <@ int4range(1,7) AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- <@

  描述：元素包含于。

  示例：

  ```
  gaussdb=# SELECT 42 <@ int4range(1,7) AS RESULT;
   result
  --------
   f
  (1 row)
  ```

- &&

  描述：重叠（有共同点）。

  示例：

  ```
  gaussdb=# SELECT int8range(3,7) && int8range(4,12) AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- <<

  描述：范围值是否比另一个范围值的最小值还小（没有交集）。

  示例：


```
gaussdb=# SELECT int8range(1,10) << int8range(100,110) AS RESULT;
 result
--------
 t
(1 row)
```

- >>

  描述：范围值是否比另一个范围值的最大值还大（没有交集）。

  示例：

```
gaussdb=# SELECT int8range(50,60) >> int8range(20,30) AS RESULT;
 result
--------
 t
(1 row)
```

- &<

  描述：范围值的最大值是否不超过另一个范围值的最大值。

  示例：

```
gaussdb=# SELECT int8range(1,20) &< int8range(18,20) AS RESULT;
 result
--------
 t
(1 row)
```

- &>

  描述：范围值的最小值是否不小于另一个范围值的最小值。

  示例：

```
gaussdb=# SELECT int8range(7,20) &> int8range(5,10) AS RESULT;
 result
--------
 t
(1 row)
```

- -|-

  描述：相邻。

  示例：

```
gaussdb=# SELECT numrange(1.1,2.2) -|- numrange(2.2,3.3) AS RESULT;
 result
--------
 t
(1 row)
```

- +

  描述：并集。

  示例：

```
gaussdb=# SELECT numrange(5,15) + numrange(10,20) AS RESULT;
 result
--------
 [5,20)
(1 row)
```

- *

  描述：交集。

  示例：

```
gaussdb=# SELECT int8range(5,15) * int8range(10,20) AS RESULT;
 result
---------
 [10,15)
(1 row)
```


- -

  描述：差集。

  示例：

```
gaussdb=# SELECT int8range(5,15) - int8range(10,20) AS RESULT;
 result
--------
 [5,10)
(1 row)
```

简单的比较操作符<，>，<=和>=先比较下界，只有下界相等时才比较上界。

<<、>>和-|-操作符当包含空范围时也会返回false；也就是，不认为空范围在其他范围之前或之后。

并集和差集操作符的执行结果无法包含两个不相交的子范围。

范围函数

如果范围是空或者需要的界限是无穷的，lower和upper函数将返回null。lower_inc、upper_inc、lower_inf和upper_inf函数均对空范围返回false。

- numrange(numeric, numeric, [text])

  描述：表示一个范围。

  返回类型：范围元素类型

  示例：

```
gaussdb=# SELECT numrange(1.1,2.2) AS RESULT;
 result
--------
[1.1,2.2)
(1 row)
gaussdb=# SELECT numrange(1.1,2.2, '()') AS RESULT;
 result
--------
(1.1,2.2)
(1 row)
```

- lower(anyrange)

  描述：范围的下界。

  返回类型：范围元素类型

  示例：

```
gaussdb=# SELECT lower(numrange(1.1,2.2)) AS RESULT;
 result
--------
    1.1
(1 row)
```

- upper(anyrange)

  描述：范围的上界。

  返回类型：范围元素类型

  示例：

```
gaussdb=# SELECT upper(numrange(1.1,2.2)) AS RESULT;
 result
--------
    2.2
(1 row)
```

- isempty(anyrange)

  描述：范围是否为空。

返回类型：Boolean

示例：

```
gaussdb=# SELECT isempty(numrange(1.1,2.2)) AS RESULT;
 result
--------
 f
(1 row)
```

- lower_inc(anyrange)

  描述：是否包含下界。

  返回类型：Boolean

  示例：

  ```
  gaussdb=# SELECT lower_inc(numrange(1.1,2.2)) AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- upper_inc(anyrange)

  描述：是否包含上界。

  返回类型：Boolean

  示例：

  ```
  gaussdb=# SELECT upper_inc(numrange(1.1,2.2)) AS RESULT;
   result
  --------
   f
  (1 row)
  ```

- lower_inf(anyrange)

  描述：下界是否为无穷。

  返回类型：Boolean

  示例：

  ```
  gaussdb=# SELECT lower_inf('(,)'::daterange) AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- upper_inf(anyrange)

  描述：上界是否为无穷。

  返回类型：Boolean

  示例：

  ```
  gaussdb=# SELECT upper_inf('(,)'::daterange) AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- elem_contained_by_range(anyelement, anyrange)

  描述：判断元素是否在范围内。

  返回类型：Boolean

  示例：

  ```
  gaussdb=# SELECT elem_contained_by_range('2', numrange(1.1,2.2));
   elem_contained_by_range
  -------------------------
   t
  (1 row)
  ```


7.5.18 聚集函数

聚集函数

- sum(expression)

  描述：所有输入行的expression总和。

  返回类型：

  通常情况下输入数据类型和输出数据类型是相同的，但以下情况会发生类型转换：

  - 对于SMALLINT或INT输入，输出类型为BIGINT。
  - 对于BIGINT输入，输出类型为NUMBER 。
  - 对于浮点数输入，输出类型为DOUBLE PRECISION。

  示例：

```
gaussdb=# CREATE TABLE tab(a int);
CREATE TABLE
gaussdb=# INSERT INTO tab values(1);
INSERT 0 1
gaussdb=# INSERT INTO tab values(2);
INSERT 0 1
gaussdb=# SELECT sum(a) FROM tab;
 sum
-----
   3
(1 row)
```

- max(expression)

  描述：所有输入行中expression的最大值。

  参数类型：任意数组、数值、字符串、日期/时间类型。

  返回类型：与参数数据类型相同

  示例：

```
gaussdb=# CREATE TABLE max_t1(a int, b int);

gaussdb=# INSERT INTO max_t1 VALUES(1,2),(2,3),(3,4),(4,5);

gaussdb=# SELECT MAX(a) FROM max_t1;
 max
-----
   4
(1 row)

gaussdb=# DROP TABLE max_t1;
```

- min(expression)

  描述：所有输入行中expression的最小值。

  参数类型：任意数组、数值、字符串、日期/时间类型。

  返回类型：与参数数据类型相同

  示例：

```
gaussdb=# CREATE TABLE min_t1(a int, b int);

gaussdb=# INSERT INTO min_t1 VALUES(1,2),(2,3),(3,4),(4,5);

gaussdb=# SELECT MIN(a) FROM min_t1;
 min
-----
   1
(1 row)
```



```
gaussdb=# DROP TABLE min_t1;
```

- avg(expression)
  描述：所有输入值的均值（算术平均）。
  返回类型：
  对于任何整数类型输入，结果都是NUMBER类型。
  对于任何浮点输入，结果都是DOUBLE PRECISION类型。
  否则和输入数据类型相同。
  示例：

```
gaussdb=# CREATE TABLE avg_t1(a int, b int);

gaussdb=# INSERT INTO avg_t1 VALUES(1,2),(2,3),(3,4),(4,5);

gaussdb=# SELECT AVG(a) FROM avg_t1;
        avg
--------------------
 2.5000000000000000
(1 row)

gaussdb=# DROP TABLE avg_t1;
```

- count(expression)
  描述：返回表中满足expression不为NULL的行数。
  返回类型：BIGINT
  支持对XML类型数据操作。
  示例：

```
gaussdb=# CREATE TABLE count_t1(a int, b int);

gaussdb=# INSERT INTO count_t1 VALUES (NULL,1),(1,2),(2,3),(3,4),(4,5);

gaussdb=# SELECT COUNT(a) FROM count_t1;
 count
-------
     4
(1 row)

gaussdb=# DROP TABLE count_t1;
```

- count(*)
  描述：返回表中的记录行数。
  返回类型：BIGINT
  支持对XML类型数据操作。
  示例：

```
gaussdb=# CREATE TABLE count_t1(a int, b int);

gaussdb=# INSERT INTO count_t1 VALUES (NULL,1),(1,2),(2,3),(3,4),(4,5);

gaussdb=# SELECT COUNT(*) FROM count_t1;
 count
-------
     5
(1 row)

gaussdb=# DROP TABLE count_t1;
```

- median(expression) [over (query partition clause)]
  描述：返回表达式的中位数，计算时NULL将会被median函数忽略。可以使用distinct关键字排除表达式中的重复记录。输入expression的数据类型可以是数值

类型（包括integer， double，bigint等）， 也可以是interval类型。其他数据类型不支持求取中位数。

返回类型：double或interval类型

示例：

```
SELECT median(id) FROM (values(1), (2), (3), (4), (null)) test(id);
 median
--------
    2.5
(1 row)
```

- array_agg(expression)

  描述：将所有输入值（包括空）连接成一个数组。

  返回类型：参数类型的数组

  支持对XML类型数据操作。

  示例：

  ```
  gaussdb=# CREATE TABLE array_agg_t1(a int, b int);

  gaussdb=# INSERT INTO array_agg_t1 VALUES (NULL,1),(1,2),(2,3),(3,4),(4,5);

  gaussdb=# SELECT ARRAY_AGG(a) FROM array_agg_t1;
    array_agg
  ----------------
   {NULL,1,2,3,4}
  (1 row)

  gaussdb=# DROP TABLE array_agg_t1;
  ```

- string_agg(expression, delimiter)

  描述：将输入值连接成为一个字符串，用分隔符分开。

  返回类型：和参数数据类型相同。

  支持对显式转换成字符类型后的XML类型数据操作。

  示例：

  ```
  gaussdb=# CREATE TABLE string_agg_t1(a int, b int);

  gaussdb=# INSERT INTO string_agg_t1 VALUES (NULL,1),(1,2),(2,3),(3,4),(4,5);

  gaussdb=# SELECT STRING_AGG(a,';') FROM string_agg_t1;
   string_agg
  ------------
   1;2;3;4
  (1 row)

  gaussdb=# DROP TABLE string_agg_t1;
  ```

- listagg(expression [, delimiter]) WITHIN GROUP(ORDER BY order-list)

  描述：将聚集列数据按WITHIN GROUP指定的排序方式排列，并用delimiter指定的分隔符拼接成一个字符串。

  - expression：必选。指定聚集列名或基于列的有效表达式，不支持DISTINCT关键字和VARIADIC参数。
  - delimiter：可选。指定分隔符，可以是字符串常数或基于分组列的确定性表达式，缺省时表示分隔符为空。
  - order-list：必选。指定分组内的排序方式。

  返回类型：text

  示例：

  聚集列是文本字符集类型。


```
gaussdb=# CREATE TABLE listagg_t1(a int, b text);

gaussdb=# INSERT INTO listagg_t1 VALUES (NULL,'a1'),(1,'b2'),(1,'c3'),(2,'d4'),(2,'e5'),(3,'f6');

gaussdb=# SELECT a,LISTAGG(b,';') WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
 a | listagg
---+---------
 1 | b2;c3
 2 | d4;e5
 3 | f6
   | a1
(4 rows)

gaussdb=# DROP TABLE listagg_t1;
```

聚集列是整型。

```
gaussdb=# CREATE TABLE listagg_t1(a int, b int);

gaussdb=# INSERT INTO listagg_t1 VALUES (NULL,1),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT a,LISTAGG(b,';') WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
 a | listagg
---+---------
 1 | 2;3
 2 | 4;5
 3 | 6
   | 1
(4 rows)

gaussdb=# DROP TABLE listagg_t1;
```

聚集列是浮点类型。

```
gaussdb=# CREATE TABLE listagg_t1(a int, b float);

gaussdb=# INSERT INTO listagg_t1 VALUES (NULL,1.111),(1,2.222),(1,3.333),(2,4.444),(2,5.555),
(3,6.666);

gaussdb=# SELECT a,LISTAGG(b,';') WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
 a |       listagg
---+-------------------
 1 | 2.222000;3.333000
 2 | 4.444000;5.555000
 3 | 6.666000
   | 1.111000
(4 rows)

gaussdb=# DROP TABLE listagg_t1;
```

聚集列是时间类型。

```
gaussdb=# CREATE TABLE listagg_t1(a int, b timestamp);

gaussdb=# INSERT INTO listagg_t1 VALUES (NULL,'2000-01-01'),(1,'2000-02-02'),(1,'2000-03-03'),
(2,'2000-04-04'),(2,'2000-05-05'),(3,'2000-06-06');

gaussdb=# SELECT a,LISTAGG(b,';') WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
 a |                  listagg
---+-------------------------------------------
 1 | 2000-02-02 00:00:00;2000-03-03 00:00:00
 2 | 2000-04-04 00:00:00;2000-05-05 00:00:00
 3 | 2000-06-06 00:00:00
   | 2000-01-01 00:00:00
(4 rows)

gaussdb=# DROP TABLE listagg_t1;
```

聚集列是时间间隔类型。

```
gaussdb=# CREATE TABLE listagg_t1(a int, b interval);

gaussdb=# INSERT INTO listagg_t1 VALUES (NULL,'1 days'),(1,'2 days'),(1,'3 days'),(2,'4 days'),(2,'5
```



```
days'),(3,'6 days');

gaussdb=# SELECT a,LISTAGG(b,';') WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
 a |    listagg
---+---------------
 1 | 2 days;3 days
 2 | 4 days;5 days
 3 | 6 days
   | 1 day
(4 rows)

gaussdb=# DROP TABLE listagg_t1;
```

分隔符缺省时，默认为空。

```
gaussdb=# CREATE TABLE listagg_t1(a int, b interval);

gaussdb=# INSERT INTO listagg_t1 VALUES (NULL,'1 days'),(1,'2 days'),(1,'3 days'),(2,'4 days'),(2,'5
days'),(3,'6 days');

gaussdb=# SELECT a,LISTAGG(b) WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
 a |   listagg
---+--------------
 1 | 2 days3 days
 2 | 4 days5 days
 3 | 6 days
   | 1 day
(4 rows)

gaussdb=# DROP TABLE listagg_t1;
```

listagg作为窗口函数时，OVER子句不支持ORDER BY的窗口排序，listagg列为对应分组的有序聚集。

```
gaussdb=# CREATE TABLE listagg_t1(a int, b interval);

gaussdb=# INSERT INTO listagg_t1 VALUES (NULL,'1 days'),(1,'2 days'),(1,'3 days'),(2,'4 days'),(2,'5
days'),(3,'6 days');

gaussdb=# SELECT a,LISTAGG(b) WITHIN GROUP(ORDER BY b) OVER(PARTITION BY a) FROM
listagg_t1;
 a |   listagg
---+--------------
 1 | 2 days3 days
 1 | 2 days3 days
 2 | 4 days5 days
 2 | 4 days5 days
 3 | 6 days
   | 1 day
(6 rows)

gaussdb=# DROP TABLE listagg_t1;
```

- group_concat([DISTINCT | ALL] expression [,expression ...] [ORDER BY { expression [[ASC | DESC | USING operator] | nlssort_expression_clause] [NULLS { FIRST | LAST }] } [,...]] [SEPARATOR str_val])

  描述：参数数量不定，可对多列进行拼接，将聚集列数据按照ORDER BY指定的排序方式排列，并用separator指定的分隔符拼接成一个字符串, 不支持作为窗口函数使用。

  - DISTINCT：可选，表示对每行拼接后结果进行去重。
  - expression：必选，指定聚集列名或基于列的有效表达式。
  - ORDER BY: 可选，后跟可变数量表达式及排序规则。group_concat函数中不支持（ORDER BY + 数字）形式。
  - SEPARATOR子句: 可选，后跟字符或字符串，分组中相邻两行表达式结果使用此分隔符拼接。若不指定，默认使用英文逗号‘,’。

- 当同时指定DISTINCT和ORDER BY时，ORDER BY表达式必须在distinct表达式中，否则报错。
- 使用参数group_concat_max_len限制GROUP_CONCAT最大返回长度，超长截断，目前能返回的最大长度是1073741823。

返回类型：text

示例：

使用separator指定分隔符为';'。

```
gaussdb=# CREATE TABLE group_concat_t1(a int, b int);

gaussdb=# INSERT INTO group_concat_t1 VALUES (NULL,1),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT a,group_concat(b separator ';') FROM group_concat_t1 GROUP BY a ORDER BY a;
 a | group_concat
---+--------------
 1 | 2;3
 2 | 4;5
 3 | 6
   | 1
(4 rows)

gaussdb=# DROP TABLE group_concat_t1;
```

分隔符缺省时，默认为','。

```
gaussdb=# CREATE TABLE group_concat_t1(a int, b int);

gaussdb=# INSERT INTO group_concat_t1 VALUES (NULL,1),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT a,group_concat(a,b) FROM group_concat_t1 GROUP BY a ORDER BY a;
 a | group_concat
---+--------------
 1 | 12,13
 2 | 24,25
 3 | 36
   | 1
(4 rows)

gaussdb=# DROP TABLE group_concat_t1;
```

聚集列是文本字符集类型。

```
gaussdb=# CREATE TABLE group_concat_t1(a int, b text);

gaussdb=# INSERT INTO group_concat_t1 VALUES (NULL,'a1'),(1,'b2'),(1,'c3'),(2,'d4'),(2,'e5'),(3,'f6');

gaussdb=# SELECT a,group_concat(a,b) FROM group_concat_t1 GROUP BY a ORDER BY a;
 a | group_concat
---+--------------
 1 | 1b2,1c3
 2 | 2d4,2e5
 3 | 3f6
   | a1
(4 rows)

gaussdb=# DROP TABLE group_concat_t1;
```

聚集列是整型。

```
gaussdb=# CREATE TABLE group_concat_t1(a int, b int);

gaussdb=# INSERT INTO group_concat_t1 VALUES (NULL,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,group_concat(b) FROM group_concat_t1 GROUP BY a ORDER BY a;
 a | group_concat
---+--------------
 1 | 2,3
 2 | 4,5
 3 | 6
```



```
  | 1
(4 rows)

gaussdb=# DROP TABLE group_concat_t1;
```

聚集列是浮点类型。

```
gaussdb=# CREATE TABLE group_concat_t1(a int, b float);

gaussdb=# INSERT INTO group_concat_t1 VALUES (NULL,1.11),(1,2.22),(1,3.33),(2,4.44),(2,5.55),
(3,6.66);

gaussdb=# SELECT a,group_concat(b) FROM group_concat_t1 GROUP BY a ORDER BY a;
 a | group_concat
---+--------------
 1 | 2.22,3.33
 2 | 4.44,5.55
 3 | 6.66
   | 1.11
(4 rows)

gaussdb=# DROP TABLE group_concat_t1;
```

聚集列是时间类型。

```
gaussdb=# CREATE TABLE group_concat_t1(a int, b timestamp);

gaussdb=# INSERT INTO group_concat_t1 VALUES (NULL,'2000-01-01'),(1,'2000-02-02'),
(1,'2000-03-03'),(2,'2000-04-04'),(2,'2000-05-05'),(3,'2000-06-06');

gaussdb=# SELECT a,group_concat(b) FROM group_concat_t1 GROUP BY a ORDER BY a;
 a |              group_concat
---+-----------------------------------------
 1 | 2000-02-02 00:00:00,2000-03-03 00:00:00
 2 | 2000-04-04 00:00:00,2000-05-05 00:00:00
 3 | 2000-06-06 00:00:00
   | 2000-01-01 00:00:00
(4 rows)

gaussdb=# DROP TABLE group_concat_t1;
```

聚集列是二进制类型。

```
gaussdb=# CREATE TABLE group_concat_t1(a int, b bytea);

gaussdb=# INSERT INTO group_concat_t1 VALUES (NULL,'1'),(1,'2'),(1,'3'),(2,'4'),(2,'5'),(3,'6');

gaussdb=# SELECT a,group_concat(b) FROM group_concat_t1 GROUP BY a ORDER BY a;
 a | group_concat
---+--------------
 1 | \x32,\x33
 2 | \x34,\x35
 3 | \x36
   | \x31
(4 rows)

gaussdb=# DROP TABLE group_concat_t1;
```

聚集列是时间间隔类型。

```
gaussdb=# CREATE TABLE group_concat_t1(a int, b interval);

gaussdb=# INSERT INTO group_concat_t1 VALUES (NULL,'1 days'),(1,'2 days'),(1,'3 days'),(2,'4 days'),
(2,'5 days'),(3,'6 days');

gaussdb=# SELECT a,group_concat(b) FROM group_concat_t1 GROUP BY a ORDER BY a;
 a | group_concat
---+---------------
 1 | 2 days,3 days
 2 | 4 days,5 days
 3 | 6 days
   | 1 day
(4 rows)
```



```
gaussdb=# DROP TABLE group_concat_t1;
```

使用distinct去重。

```
gaussdb=# CREATE TABLE group_concat_t1(a int, b interval);

gaussdb=# INSERT INTO group_concat_t1 VALUES (NULL,'1 days'),(1,'2 days'),(1,'2 days'),(1,'3 days'),
(1,'3 days'),(2,'4 days'),(2,'5 days'),(3,'6 days');

gaussdb=# SELECT a,group_concat(distinct b) FROM group_concat_t1 GROUP BY a ORDER BY a;
 a | group_concat
---+---------------
 1 | 2 days,3 days
 2 | 4 days,5 days
 3 | 6 days
   | 1 day
(4 rows)

gaussdb=# DROP TABLE group_concat_t1;
```

使用ORDER BY排序。

```
gaussdb=# CREATE TABLE group_concat_t1(a int, b interval);

gaussdb=# INSERT INTO group_concat_t1 VALUES (NULL,'1 days'),(1,'2 days'),(1,'3 days'),(2,'4 days'),
(2,'5 days'),(3,'6 days');

gaussdb=# SELECT a,group_concat(b ORDER BY b desc) FROM group_concat_t1 GROUP BY a ORDER
BY a;
 a | group_concat
---+---------------
 1 | 3 days,2 days
 2 | 5 days,4 days
 3 | 6 days
   | 1 day
(4 rows)

gaussdb=# DROP TABLE group_concat_t1;
```

- wm_concat(expression)

  描述：将列数据连接成为一个字符串，用','进行分隔。

  返回类型：和参数数据类型相同

  说明

  wm_concat是A数据库兼容性需求，目前A数据库最新版本已经取消此函数，在A数据库中目前使用listagg函数对功能进行替代。目前此函数功能可使用listagg函数或string_agg进行替代，使用具体方法见上述两函数描述。

- covar_pop(Y, X)

  描述：总体协方差。

  返回类型：double precision

  示例：

```
gaussdb=# CREATE TABLE covar_pop_t1(a int, b int);

gaussdb=# INSERT INTO covar_pop_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);

gaussdb=# SELECT COVAR_POP(a,b) FROM covar_pop_t1;
 covar_pop
-----------
       100
(1 row)

gaussdb=# DROP TABLE covar_pop_t1;
```

- covar_samp(Y, X)

描述：样本协方差。

返回类型：double precision

示例：

```
gaussdb=# CREATE TABLE covar_samp_t1(a int, b int);

gaussdb=# INSERT INTO covar_samp_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);

gaussdb=# SELECT COVAR_SAMP(a,b) FROM covar_samp_t1;
 covar_samp
------------
        125
(1 row)

gaussdb=# DROP TABLE covar_samp_t1;
```

- stddev_pop(expression)

  描述：总体标准差。

  返回类型：对于浮点类型的输入返回double precision，其他输入返回numeric。

  示例：

```
gaussdb=# CREATE TABLE stddev_pop_t1(a int, b int);

gaussdb=# INSERT INTO stddev_pop_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);

gaussdb=# SELECT STDDEV_POP(a) FROM stddev_pop_t1;
     stddev_pop
--------------------
 7.4833147735478828
(1 row)

gaussdb=# DROP TABLE stddev_pop_t1;
```

- stddev_samp(expression)

  描述：样本标准差。

  返回类型：对于浮点类型的输入返回double precision，其他输入返回numeric。

  示例：

```
gaussdb=# CREATE TABLE stddev_samp_t1(a int, b int);

gaussdb=# INSERT INTO stddev_samp_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);

gaussdb=# SELECT STDDEV_SAMP(a) FROM stddev_samp_t1;
    stddev_samp
--------------------
 8.3666002653407555
(1 row)

gaussdb=# DROP TABLE stddev_samp_t1;
```

- var_pop(expression)

  描述：总体方差（总体标准差的平方）。

  返回类型：对于浮点类型的输入返回double precision类型，其他输入返回numeric类型。

  示例：

```
gaussdb=# CREATE TABLE var_pop_t1(a int, b int);

gaussdb=# INSERT INTO var_pop_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);

gaussdb=# SELECT VAR_POP(a) FROM var_pop_t1;
       var_pop
--------------------
 56.0000000000000000
```



```
(1 row)

gaussdb=# DROP TABLE var_pop_t1;
```

- var_samp(expression)

  描述：样本方差（样本标准差的平方）。

  返回类型：对于浮点类型的输入返回double precision类型，其他输入返回numeric类型。

  示例：

```
gaussdb=# CREATE TABLE var_samp_t1(a int, b int);

gaussdb=# INSERT INTO var_samp_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);

gaussdb=# SELECT VAR_SAMP(a) FROM var_samp_t1;
     var_samp
--------------------
 70.0000000000000000
(1 row)

gaussdb=# DROP TABLE var_samp_t1;
```

- bit_and(expression)

  描述：所有非NULL输入值的按位与(AND)，如果全部输入值皆为NULL，那么结果也为NULL 。

  返回类型：和参数数据类型相同。

  示例：

```
gaussdb=# CREATE TABLE bit_and_t1(a int, b int);

gaussdb=# INSERT INTO bit_and_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT BIT_AND(a) FROM bit_and_t1;
 bit_and
---------
       0
(1 row)

gaussdb=# DROP TABLE bit_and_t1;
```

- bit_or(expression)

  描述：所有非NULL输入值的按位或(OR)，如果全部输入值皆为NULL，那么结果也为NULL。

  返回类型：和参数数据类型相同

  示例：

```
gaussdb=# CREATE TABLE bit_or_t1(a int, b int);

gaussdb=# INSERT INTO bit_or_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT BIT_OR(a) FROM bit_or_t1;
 bit_or
--------
      3
(1 row)

gaussdb=# DROP TABLE bit_or_t1;
```

- bool_and(expression)

  描述：如果所有输入值都是真，则为真，否则为假。

  返回类型：bool

  示例：


```
gaussdb=# SELECT bool_and(100 <2500);
 bool_and
----------
 t
(1 row)
```

- bool_or(expression)

  描述：如果所有输入值只要有一个为真，则为真，否则为假。

  返回类型：bool

  示例：

```
gaussdb=# SELECT bool_or(100 <2500);
 bool_or
----------
 t
(1 row)
```

- corr(Y, X)

  描述：相关系数。

  返回类型：double precision

  示例：

```
gaussdb=# CREATE TABLE corr_t1(a int, b int);

gaussdb=# INSERT INTO corr_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT CORR(a,b) FROM corr_t1;
       corr
------------------
 .944911182523068
(1 row)

gaussdb=# DROP TABLE corr_t1;
```

- every(expression)

  描述：等效于bool_and。

  返回类型：bool

  示例：

```
gaussdb=# SELECT every(100 <2500);
 every
-------
 t
(1 row)
```

- regr_avgx(Y, X)

  描述：自变量的平均值 (sum(X)/N)。

  返回类型：double precision

  示例：

```
gaussdb=# CREATE TABLE regr_t1(a int, b int);

gaussdb=# INSERT INTO regr_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT REGR_AVGX(a,b) FROM regr_t1;
 regr_avgx
-----------
         4
(1 row)

gaussdb=# DROP TABLE regr_t1;
```

- regr_avgy(Y, X)

  描述：因变量的平均值 (sum(Y)/N)。

返回类型：double precision

示例：

```
gaussdb=# CREATE TABLE regr_avgy_t1(a int, b int);

gaussdb=# INSERT INTO regr_avgy_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT REGR_AVGY(a,b) FROM regr_avgy_t1;
 regr_avgy
-----------
      1.8
(1 row)

gaussdb=# DROP TABLE regr_avgy_t1;
```

- regr_count(Y, X)

  描述：两个表达式都不为NULL的输入行数。

  返回类型：bigint

  示例：

```
gaussdb=# CREATE TABLE regr_count_t1(a int, b int);

gaussdb=# INSERT INTO regr_count_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT REGR_COUNT(a,b) FROM regr_count_t1;
 regr_count
------------
          5
(1 row)

gaussdb=# DROP TABLE regr_count_t1;
```

- regr_intercept(Y, X)

  描述：根据所有输入的点(X, Y)按照最小二乘法拟合成一个线性方程，然后返回该直线的Y轴截距。

  返回类型：double precision

  示例：

```
gaussdb=# CREATE TABLE regr_intercept_t1(a int, b int);

gaussdb=# INSERT INTO regr_intercept_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT REGR_INTERCEPT(b,a) FROM regr_intercept_t1;
  regr_intercept
------------------
 .785714285714286
(1 row)

gaussdb=# DROP TABLE regr_intercept_t1;
```

- regr_r2(Y, X)

  描述：相关系数的平方。

  返回类型：double precision

  示例：

```
gaussdb=# CREATE TABLE regr_r2_t1(a int, b int);

gaussdb=# INSERT INTO regr_r2_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT REGR_R2(b,a) FROM regr_r2_t1;
     regr_r2
------------------
 .892857142857143
(1 row)
```



```
gaussdb=# DROP TABLE regr_r2_t1;
```

- regr_slope(Y, X)

  描述：根据所有输入的点(X, Y)按照最小二乘法拟合成一个线性方程， 然后返回该直线的斜率。

  返回类型：double precision

  示例：

```
gaussdb=# CREATE TABLE regr_slope_t1(a int, b int);

gaussdb=# INSERT INTO regr_slope_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT REGR_SLOPE(b,a) FROM regr_slope_t1;
    regr_slope
------------------
 1.78571428571429
(1 row)

gaussdb=# DROP TABLE regr_slope_t1;
```

- regr_sxx(Y, X)

  描述：sum(X^2) - sum(X)^2/N （自变量的“平方和”）。

  返回类型：double precision

  示例：

```
gaussdb=# CREATE TABLE regr_sxx_t1(a int, b int);

gaussdb=# INSERT INTO regr_sxx_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT REGR_SXX(b,a) FROM regr_sxx_t1;
 regr_sxx
----------
      2.8
(1 row)

gaussdb=# DROP TABLE regr_sxx_t1;
```

- regr_sxy(Y, X)

  描述：sum(X*Y) - sum(X) * sum(Y)/N （自变量和因变量的“乘方积”）。

  返回类型：double precision

  示例：

```
gaussdb=# CREATE TABLE regr_sxy_t1(a int, b int);

gaussdb=# INSERT INTO regr_sxy_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT REGR_SXY(b,a) FROM regr_sxy_t1;
 regr_sxy
----------
        5
(1 row)

gaussdb=# DROP TABLE regr_sxy_t1;
```

- regr_syy(Y, X)

  描述：sum(Y^2) - sum(Y)^2/N（因变量的"平方和"）。

  返回类型：double precision

  示例：

```
gaussdb=# CREATE TABLE regr_syy_t1(a int, b int);

gaussdb=# INSERT INTO regr_syy_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);
```



```
gaussdb=# SELECT REGR_SYY(b,a) FROM regr_syy_t1;
 regr_syy
----------
       10
(1 row)

gaussdb=# DROP TABLE regr_syy_t1;
```

- stddev(expression)

  描述：stddev_samp的别名。

  返回类型：对于浮点类型的输入返回double precision，其他输入返回numeric。

  示例：

```
gaussdb=# CREATE TABLE stddev_t1(a int, b int);

gaussdb=# INSERT INTO stddev_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT STDDEV(a) FROM stddev_t1;
       stddev
----------------------
 .83666002653407554798
(1 row)

gaussdb=# DROP TABLE stddev_t1;
```

- variance(expexpression,ression)

  描述：var_samp的别名。

  返回类型：对于浮点类型的输入返回double precision类型，其他输入返回numeric类型。

  示例：

```
gaussdb=# CREATE TABLE variance_t1(a int, b int);

gaussdb=# INSERT INTO variance_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT VARIANCE(a) FROM variance_t1;
      variance
----------------------
 .70000000000000000000
(1 row)

gaussdb=# DROP TABLE variance_t1;
```

- delta

  描述：返回当前行和前一行的差值。

  参数：numeric

  返回值类型：numeric

- checksum(expression)

  描述：返回所有输入值的CHECKSUM值。使用该函数可以用来验证GaussDB数据库（不支持GaussDB之外的其他数据库）的备份恢复或者数据迁移操作前后表中的数据是否相同。在备份恢复或者数据迁移操作前后都需要用户通过手工执行SQL命令的方式获取执行结果，通过对比获取的执行结果判断操作前后表中的数据是否相同。

说明

- 对于大表，CHECKSUM函数可能会需要很长时间。
- 如果某两表的CHECKSUM值不同，则表明两表的内容是不同的。由于CHECKSUM函数中使用散列函数不能保证无冲突，因此两个不同内容的表可能会得到相同的CHECKSUM值，存在这种情况的可能性较小。对于列进行的CHECKSUM也存在相同的情况。
- 对于时间类型timestamp, timestamptz和smalldatetime，计算CHECKSUM值时请确保时区设置一致。

– 若计算某列的CHECKSUM值，且该列类型可以默认转为TEXT类型，则expression为列名。
– 若计算某列的CHECKSUM值，且该列类型不能默认转为TEXT类型，则expression为列名::TEXT。
– 若计算所有列的CHECKSUM值，则expression为表名::TEXT。

可以默认转换为TEXT类型的类型包括：char, name, int8, int2, int1, int4, raw, pg_node_tree, float4, float8, bpchar, varchar, nvarchar, nvarchar2, date, timestamp, timestamptz, numeric, smalldatetime，其他类型需要强制转换为TEXT，例如XML类型。

返回类型：numeric。

示例：

表中可以默认转为TEXT类型的某列的CHECKSUM值。

```
gaussdb=# CREATE TABLE checksum_t1(a int, b int);

gaussdb=# INSERT INTO checksum_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT CHECKSUM(a) FROM checksum_t1;
  checksum
-------------
 18126842830
(1 row)

gaussdb=# DROP TABLE checksum_t1;
```

表中不能默认转为TEXT类型的某列的CHECKSUM值。注意此时CHECKSUM参数是列名::TEXT。

```
gaussdb=# CREATE TABLE checksum_t1(a int, b int);

gaussdb=# INSERT INTO checksum_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT CHECKSUM(a::TEXT) FROM checksum_t1;
  checksum
-------------
 18126842830
(1 row)

gaussdb=# DROP TABLE checksum_t1;
```

表中所有列的CHECKSUM值。注意此时CHECKSUM参数是表名::TEXT，且表名前不加Schema。

```
gaussdb=# CREATE TABLE checksum_t1(a int, b int);

gaussdb=# INSERT INTO checksum_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT CHECKSUM(checksum_t1::TEXT) FROM checksum_t1;
  checksum
-------------
 11160522226
(1 row)

gaussdb=# DROP TABLE checksum_t1;
```


- percentile_cont(percentile float)

  描述：对给定的列按照时间序列排序并返回百分位值。

  返回类型：float

  说明

  - percentile为[0,1]之间的小数，精度为浮点值类型，不支持95%写法。
  - 需要与WITHIN GROUP (ORDER BY)结合使用，指定分位值的列，且该列类型需要为数值型。

  示例：

```
gaussdb=# SELECT percentile_cont(0) WITHIN GROUP (ORDER BY value) FROM (VALUES (1),(2)) v(value);
 percentile_cont
-----------------
               1
(1 row)
```

- mode() WITHIN GROUP (ORDER BY value anyelement)

  描述：返回某列中出现频率最高的值，如果多个值频率相同，则返回最小的那个值。排序方式和该列类型的默认排序方式相同。其中value为输入参数，可以为任意类型。

  返回类型：与输入参数类型相同。

  示例：

```
gaussdb=# SELECT mode() WITHIN GROUP (ORDER BY value) FROM (values(1, 'a'), (2, 'b'), (2, 'c')) v(value, tag);
 mode
------
    2
(1 row)
gaussdb=# SELECT mode() WITHIN GROUP (ORDER BY tag) FROM (values(1, 'a'), (2, 'b'), (2, 'c')) v(value, tag);
 mode
------
 a
(1 row)
```

- pivot_func(anyelement)

  描述：返回某列中唯一不为NULL的值，如果有超过两个非NULL值则报错。其中value为输入参数，可以为任意类型。

  返回类型：与输入参数类型相同。

  说明

  该聚合函数主要作为pivot语法内部使用。

  示例：

```
gaussdb=# CREATE TABLE pivot_func_t1(a int, b int);

gaussdb=# INSERT INTO pivot_func_t1 VALUES (NULL,11),(1,2);

gaussdb=# SELECT PIVOT_FUNC(a) FROM pivot_func_t1;
 pivot_func
------------
          1
(1 row)

gaussdb=# DROP TABLE pivot_func_t1;
```


7.5.19 窗口函数

窗口函数

窗口函数与OVER语句一起使用。OVER语句用于对数据进行分组，并对组内元素进行排序。窗口函数用于给组内的值生成序号。

说明

窗口函数中的order by后面必须跟字段名，若order by后面跟数字，该数字会被按照常量处理，因此对目标列没有起到排序的作用。

- RANK()

  描述：RANK函数为各组内值生成跳跃排序序号，其中，相同的值具有相同序号。

  返回值类型：BIGINT

  示例：

```
gaussdb=# CREATE TABLE rank_t1(a int, b int);

gaussdb=# INSERT INTO rank_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,b,RANK() OVER(PARTITION BY a ORDER BY b) FROM rank_t1;
 a | b | rank
---+---+------
 1 | 1 |    1
 1 | 1 |    1
 1 | 2 |    3
 1 | 3 |    4
 2 | 4 |    1
 2 | 5 |    2
 3 | 6 |    1
(7 rows)

gaussdb=# DROP TABLE rank_t1;
```

- ROW_NUMBER()

  描述：ROW_NUMBER函数为各组内值生成连续排序序号，其中，相同的值其序号也不相同。

  返回值类型：BIGINT

  示例：

```
gaussdb=# CREATE TABLE row_number_t1(a int, b int);

gaussdb=# INSERT INTO row_number_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,b,ROW_NUMBER() OVER(PARTITION BY a ORDER BY b) FROM row_number_t1;
 a | b | row_number
---+---+------------
 1 | 1 |          1
 1 | 1 |          2
 1 | 2 |          3
 1 | 3 |          4
 2 | 4 |          1
 2 | 5 |          2
 3 | 6 |          1
(7 rows)

gaussdb=# DROP TABLE row_number_t1;
```

- DENSE_RANK()

  描述：DENSE_RANK函数为各组内值生成连续排序序号，其中，相同的值具有相同序号。

返回值类型：BIGINT

示例：

```
gaussdb=# CREATE TABLE dense_rank_t1(a int, b int);

gaussdb=# INSERT INTO dense_rank_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,b,DENSE_RANK() OVER(PARTITION BY a ORDER BY b) FROM dense_rank_t1;
 a | b | dense_rank
---+---+------------
 1 | 1 |          1
 1 | 1 |          1
 1 | 2 |          2
 1 | 3 |          3
 2 | 4 |          1
 2 | 5 |          2
 3 | 6 |          1
(7 rows)

gaussdb=# DROP TABLE dense_rank_t1;
```

- PERCENT_RANK()

  描述：PERCENT_RANK函数为各组内对应值生成相对序号，即根据公式 (rank - 1) / (totalrows - 1)计算所得的值。其中rank为该值依据RANK函数所生成的对应序号，totalrows为该分组内的总元素个数。

  返回值类型：DOUBLE PRECISION

  示例：

```
gaussdb=# CREATE TABLE percent_rank_t1(a int, b int);

gaussdb=# INSERT INTO percent_rank_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,b,PERCENT_RANK() OVER(PARTITION BY a ORDER BY b) FROM percent_rank_t1;
 a | b |    percent_rank
---+---+-----------------
 1 | 1 |               0
 1 | 1 |               0
 1 | 2 | .666666666666667
 1 | 3 |               1
 2 | 4 |               0
 2 | 5 |               1
 3 | 6 |               0
(7 rows)

gaussdb=# DROP TABLE percent_rank_t1;
```

- CUME_DIST()

  描述：CUME_DIST函数为各组内对应值生成累积分布序号。即根据公式(小于等于当前值的数据行数)/(该分组总行数totalrows)计算所得的相对序号。

  返回值类型：DOUBLE PRECISION

  示例：

```
gaussdb=# CREATE TABLE cume_dist_t1(a int, b int);

gaussdb=# INSERT INTO cume_dist_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,b,CUME_DIST() OVER(PARTITION BY a ORDER BY b) FROM cume_dist_t1;
 a | b | cume_dist
---+---+-----------
 1 | 1 |        .5
 1 | 1 |        .5
 1 | 2 |       .75
 1 | 3 |         1
 2 | 4 |        .5
 2 | 5 |         1
 3 | 6 |         1
```



```
(7 rows)

gaussdb=# DROP TABLE cume_dist_t1;
```

- NTILE(num_buckets integer)

  描述：NTILE函数根据num_buckets integer将有序的数据集合平均分配到num_buckets所指定数量的桶中，并将桶号分配给每一行。分配时应尽量做到平均分配。

  返回值类型：INTEGER

  示例：

```
gaussdb=# CREATE TABLE ntile_t1(a int, b int);

gaussdb=# INSERT INTO ntile_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,b,NTILE(2) OVER(PARTITION BY a ORDER BY b) FROM ntile_t1;
 a | b | ntile
---+---+-------
 1 | 1 |     1
 1 | 1 |     1
 1 | 2 |     2
 1 | 3 |     2
 2 | 4 |     1
 2 | 5 |     2
 3 | 6 |     1
(7 rows)

gaussdb=# DROP TABLE ntile_t1;
```

- LAG(value any [, offset integer [, default any]])

  描述：LAG函数为各组内对应值生成滞后值。即当前值对应的行数往前偏移offset位后所得行的value值作为序号。若经过偏移后行数不存在，则对应结果取为default值。若无指定，在默认情况下，offset取为1，default值取为NULL。default值的类型需要与value值的类型保持一致。

  返回值类型：与参数数据类型相同

  示例：

```
-- 建表并插入数据
gaussdb=# CREATE TABLE ta1 (hire_date date, last_name varchar(20), department_id int);
CREATE TABLE
gaussdb=# INSERT INTO ta1 VALUES('07-DEC-02', 'Raphaely', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 VALUES('24-JUL-05', 'Tobias',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 VALUES('24-DEC-05', 'Baida',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 VALUES('18-MAY-03', 'Khoo', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('15-NOV-06', 'Himuro',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('10-AUG-07', 'Colmenares',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('10-MAY-07', 'yq',  11);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('10-MAY-08', 'zi',  11);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('', 'yq1',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values(null, 'yq2',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('10-DEC-07', 'yq3',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values(null, null,  11);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values(null, null,  11);
INSERT 0 1
```



```
-- 调用LAG,指定offset=3, default=null
gaussdb=# SELECT hire_date, last_name, department_id, lag(hire_date, 3, null)  OVER (PARTITION BY
department_id ORDER BY last_name) AS "NextHired" FROM ta1 ORDER BY department_id;
     hire_date      | last_name  | department_id |      NextHired
--------------------+------------+---------------+---------------------
 2007-05-10 00:00:00 | yq         |            11 |
 2008-05-10 00:00:00 | zi         |            11 |
                     |            |            11 |
                     |            |            11 | 2007-05-10 00:00:00
 2005-12-24 00:00:00 | Baida      |            30 |
 2007-08-10 00:00:00 | Colmenares |            30 |
 2006-11-15 00:00:00 | Himuro     |            30 |
 2003-05-18 00:00:00 | Khoo       |            30 | 2005-12-24 00:00:00
 2002-12-07 00:00:00 | Raphaely   |            30 | 2007-08-10 00:00:00
 2005-07-24 00:00:00 | Tobias     |            30 | 2006-11-15 00:00:00
                     | yq1        |            30 | 2003-05-18 00:00:00
                     | yq2        |            30 | 2002-12-07 00:00:00
 2007-12-10 00:00:00 | yq3        |            30 | 2005-07-24 00:00:00
(13 rows)
```

- LEAD(value any [, offset integer [, default any]])

  描述：LEAD函数为各组内对应值生成提前值。即当前值对应的行数向后偏移offset位后所得行的value值作为序号。若经过向后偏移后行数超过当前组内的总行数，则对应结果取为default值。若无指定，在默认情况下，offset取为1，default值取为NULL。default值的类型需要与value值的类型保持一致。

  返回值类型：与参数数据类型相同。

  示例：

```
-- 建表并插入数据
gaussdb=# CREATE TABLE ta1 (hire_date date, last_name varchar(20), department_id int);
CREATE TABLE
gaussdb=# INSERT INTO ta1 values('07-DEC-02', 'Raphaely', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('24-JUL-05', 'Tobias',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('24-DEC-05', 'Baida',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('18-MAY-03', 'Khoo', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('15-NOV-06', 'Himuro',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('10-AUG-07', 'Colmenares',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('10-MAY-07', 'yq',  11);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('10-MAY-08', 'zi',  11);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('', 'yq1',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values(null, 'yq2',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('10-DEC-07', 'yq3',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values(null, null,  11);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values(null, null,  11);
INSERT 0 1

-- 调用LEAD,指定offset=2
gaussdb=# SELECT hire_date, last_name, department_id, lead(hire_date, 2) OVER (PARTITION BY
department_id ORDER BY last_name) AS "NextHired" FROM ta1 ORDER BY department_id;
     hire_date      | last_name  | department_id |      NextHired
--------------------+------------+---------------+---------------------
 2007-05-10 00:00:00 | yq         |            11 |
 2008-05-10 00:00:00 | zi         |            11 |
                     |            |            11 |
                     |            |            11 |
```



```
2005-12-24 00:00:00 | Baida      |          30 | 2006-11-15 00:00:00
2007-08-10 00:00:00 | Colmenares |          30 | 2003-05-18 00:00:00
2006-11-15 00:00:00 | Himuro     |          30 | 2002-12-07 00:00:00
2003-05-18 00:00:00 | Khoo       |          30 | 2005-07-24 00:00:00
2002-12-07 00:00:00 | Raphaely   |          30 |
2005-07-24 00:00:00 | Tobias     |          30 |
                    | yq1        |          30 | 2007-12-10 00:00:00
                    | yq2        |          30 |
2007-12-10 00:00:00 | yq3        |          30 |
(13 rows)
```

- FIRST_VALUE(value any)

  描述：FIRST_VALUE函数取各组内的第一个值作为返回结果。

  返回值类型：与参数数据类型相同。

  示例：

```
gaussdb=# CREATE TABLE first_value_t1(a int, b int);

gaussdb=# INSERT INTO first_value_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,b,FIRST_VALUE(b) OVER(PARTITION BY a ORDER BY b) FROM first_value_t1;
 a | b | first_value
---+---+-------------
 1 | 1 |           1
 1 | 1 |           1
 1 | 2 |           1
 1 | 3 |           1
 2 | 4 |           4
 2 | 5 |           4
 3 | 6 |           6
(7 rows)

gaussdb=# DROP TABLE first_value_t1;
```

- LAST_VALUE(value any)

  描述：LAST_VALUE函数取各组内的最后一个值作为返回结果。

  返回值类型：与参数数据类型相同。

  示例：

```
gaussdb=# CREATE TABLE last_value_t1(a int, b int);

gaussdb=# INSERT INTO last_value_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,b,LAST_VALUE(b) OVER(PARTITION BY a ORDER BY b) FROM last_value_t1;
 a | b | last_value
---+---+------------
 1 | 1 |          1
 1 | 1 |          1
 1 | 2 |          2
 1 | 3 |          3
 2 | 4 |          4
 2 | 5 |          5
 3 | 6 |          6
(7 rows)

gaussdb=# DROP TABLE last_value_t1;
```

- DELTA

  描述：返回当前行和前一行的差值。

  参数：numeric

  返回值类型：numeric

- NTH_VALUE(value any, nth integer)

  描述：NTH_VALUE函数返回该组内的第nth行作为结果。若该行不存在，则默认返回NULL。

返回值类型：与参数数据类型相同。

示例：

```
gaussdb=# CREATE TABLE nth_value_t1(a int, b int);

gaussdb=# INSERT INTO nth_value_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,b,NTH_VALUE(b, 2) OVER(PARTITION BY a order by b) FROM nth_value_t1;
 a | b | nth_value
---+---+-----------
 1 | 1 |         1
 1 | 1 |         1
 1 | 2 |         1
 1 | 3 |         1
 2 | 4 |
 2 | 5 |         5
 3 | 6 |
(7 rows)

gaussdb=# DROP TABLE nth_value_t1;
```

7.5.20 安全函数

安全函数

- gs_encrypt_aes128(encryptstr,keystr)

  描述：以keystr为加密口令对encryptstr字符串进行加密，返回加密后的字符串。keystr的长度范围为8~16字节，至少包含3种字符（大写字母、小写字母、数字、特殊字符）。

  返回值类型：text

  返回值长度：至少为92字节，不超过4*[(Len+68)/3]字节，其中Len为加密前数据长度（单位为字节）。

  示例：

  ```
  gaussdb=# SELECT gs_encrypt_aes128('MPPDB','Asdf1234');

                              gs_encrypt_aes128
  ----------------------------------------------------------------------------------
  jRLOH2cqywwbAeiQQ9KuujGhJVGfp317wsdKbUC+AcWX7NLZAUPmITDJhuu/
  6164qOrLA8uvCRA60QNX6MF3yOPViWc=
  (1 row)
  ```

  说明

  由于该函数的执行过程需要传入加密口令，为了安全起见，gsql工具不会将包含该函数名字样的SQL记录入执行历史。即无法在gsql里通过上下翻页功能找到该函数的执行历史。

- gs_encrypt(encryptstr,keystr, encrypttype)

  描述：根据encrypttype，以keystr为加密口令对encryptstr字符串进行加密，返回加密后的字符串。keystr的长度范围为8~16字节，至少包含3种字符（大写字母、小写字母、数字、特殊字符），encrypttype可以是aes128或sm4。

  返回值类型：text

  示例：

  ```
  gaussdb=#  SELECT gs_encrypt('MPPDB','Asdf1234','sm4');
         gs_encrypt
   ------------------------------
   ZBzOmaGA4Bb+coyucJ0B8AkIShqc
  (1 row)
  ```


说明

由于该函数的执行过程需要传入加密口令，为了安全起见，gsql工具不会将包含该函数名字样的SQL记录入执行历史。即无法在gsql里通过上下翻页功能找到该函数的执行历史。

- gs_decrypt_aes128(decryptstr,keystr)

  描述：以keystr为解密口令对decrypt字符串进行解密，返回解密后的字符串。解密使用的keystr必须保证与加密时使用的keystr一致才能正常解密。keystr不得为空。

  说明

  此参数需要结合gs_encrypt_aes128加密函数共同使用。

  返回值类型：text

  示例：

```
gaussdb=# SELECT gs_decrypt_aes128('jRLOH2cqywwbAeiQQ9KuujGhJVGfp317wsdKbUC
+AcWX7NLZAUPmITDJhuu/6164qOrLA8uvCRA60QNX6MF3yOPViWc=','Asdf1234');
 gs_decrypt_aes128
-------------------
 MPPDB
(1 row)
```

  说明

  由于该函数的执行过程需要传入解密口令，为了安全起见，gsql工具不会将包含该函数名字样的SQL记录入执行历史；即无法在gsql里通过上下翻页功能找到该函数的执行历史。

- gs_decrypt(decryptstr,keystr,decrypttype)

  描述：根据decrypttype，以keystr为解密口令对decrypt字符串进行解密，返回解密后的字符串。解密使用的decrypttype及keystr必须保证与加密时使用的encrypttype及keystr一致才能正常解密。keystr不得为空。decrypttype可以是aes128或sm4。

  此函数需要结合gs_encrypt加密函数共同使用。

  返回值类型：text

  示例：

```
gaussdb=# SELECT gs_decrypt('ZBzOmaGA4Bb+coyucJ0B8AkIShqc','Asdf1234','sm4');
 gs_decrypt
------------
 MPPDB
(1 row)
```

  说明

  由于该函数的执行过程需要传入解密口令，为了安全起见，gsql工具不会将包含该函数名字样的SQL记录入执行历史；即无法在gsql里通过上下翻页功能找到该函数的执行历史。

- aes_encrypt(str, key_str, init_vector)

  描述：基于AES算法，使用密钥字符串key_str和初始化向量init_vector对字符串str进行加密。

  参数解释：

  - str: 需要被加密的字符串。若str为NULL，函数返回NULL。
  - key_str: 密钥字符串。若key_str为NULL，函数返回NULL。为了安全，对于128bit/192bit/256bit的密钥长度（由块加密模式block_encryption_mode确定，建议用户使用128bit/192bit/256bit的安全随机数作为密钥字符串。
  - init_vector: 为需要它的块加密模式提供初始化变量，长度大于等于16字节（大于16字节的部分会被自动忽略）。str和key_str均不为NULL时，该参数

不可为NULL，否则报错。为了安全，建议用户在OFB模式下，保证每次加密IV值的唯一性；在CBC模式和CFB模式下，保证每次加密的IV值不可被预测。

返回值类型：text

示例：

```
gaussdb=# SELECT aes_encrypt('huwei123','123456vfhex4dyu,vdaladhjsadad','1234567890123456');
 aes_encrypt
-------------
 u*8\x05c?0
(1 row)
```

说明

- 该函数仅在GaussDB兼容MY类型时（即sql_compatibility = 'B'）有效，其他类型不支持该函数。兼容类型在创建数据库的时候设置，详见《开发指南》的“SQL参考 > SQL语法 > CREATE DATABASE”章节。
- 由于该函数的执行过程需要传入解密口令，为了安全起见，gsql工具不会将包含该函数名字样的SQL记录入执行历史；即无法在gsql里通过上下翻页功能找到该函数的执行历史。
- 在存储过程的相关操作中需尽量避免调用该函数，避免敏感参数信息在日志中泄露的风险。同时建议用户在使用包含该函数的存储过程相关操作时，将该函数的参数信息过滤后再提供给外部维护人员定位，日志使用完后请及时删除。
- 在打开debug_print_plan开关的情况下需尽量避免调用该函数，避免敏感参数信息在日志中泄露的风险。同时建议用户在打开debug_print_plan开关生成的日志中对该函数的参数信息进行过滤后再提供给外部维护人员定位，日志使用完后请及时删除。
- 由于SQL_ASCII设置与其他设置表现不同。如果服务器字符集是SQL_ASCII，服务器把字节值0~127根据 ASCII标准解释，而字节值128~255则当作无法解析的字符。如果设置为SQL_ASCII，就不会有编码转换。该函数调用openssl三方库返回的数据的编码为非ASCII数据，因此当数据库服务端字符集设置为SQL_ASCII时，客户端编码也需设置为SQL_ASCII ，否则会报错。因为数据库不会帮助转换或者校验非ASCII字符。

- aes_decrypt(pass_str, key_str, init_vector)

  描述：基于AES算法，使用密钥字符串key_str和初始化向量init_vector对字符串str进行解密。

  参数解释：

  – pass_str: 需要被解密的字符串。若pass_str为NULL，函数返回NULL。

  – key_str: 密钥字符串。若key_str为NULL，函数返回NULL。为了安全，对于128bit/192bit/256bit的密钥长度（由块加密模式block_encryption_mode确定），建议用户使用128bit/192bit/256bit的安全随机数作为密钥字符串。

  – init_vector: 为需要它的块解密模式提供初始化变量，长度大于等于16字节（大于16字节的部分会被自动忽略）。pass_str和key_str均不为NULL时，该参数不可为NULL，否则报错。为了安全，建议用户在OFB模式下，保证每次加密IV值的唯一性；在CBC模式和CFB模式下，保证每次加密的IV值不可被预测。

  返回值类型：text

  示例：

```
gaussdb=# SELECT
aes_decrypt(aes_encrypt('huwei123','123456vfhex4dyu,vdaladhjsadad','1234567890123456'),'123456vf
hex4dyu,vdaladhjsadad','1234567890123456');
 aes_decrypt
-------------
 huwei123
(1 row)
```


说明

- 该函数仅在GaussDB兼容MY类型时（即sql_compatibility = 'B'）有效，其他类型不支持该函数。兼容类型在创建数据库的时候设置，详见《开发指南》的“SQL参考 > SQL语法 > CREATE DATABASE”章节。
- 由于该函数的执行过程需要传入解密口令，为了安全起见，gsql工具不会将包含该函数名字样的SQL记录入执行历史；即无法在gsql里通过上下翻页功能找到该函数的执行历史。
- 在存储过程的相关操作中需尽量避免调用该函数，避免敏感参数信息在日志中泄露的风险。同时建议用户在使用包含该函数的存储过程相关操作时，将该函数的参数信息过滤后再提供给外部维护人员定位，日志使用完后请及时删除。
- 在打开debug_print_plan开关的情况下需尽量避免调用该函数，避免敏感参数信息在日志中泄露的风险。同时建议用户在打开debug_print_plan开关生成的日志中对该函数的参数信息进行过滤后再提供给外部维护人员定位，日志使用完后请及时删除。
- 若想成功解密，需要保证block_encryption_mode，key_str，iv值与加密时一致。
- 由于编码差异，不支持从gsql客户端直接拷贝加密后的数据进行解密，此场景解密出的结果不一定是加密前的字符串。
- 由于SQL_ASCII设置与其他设置表现不同。如果服务器字符集是SQL_ASCII，服务器把字节值0~127根据 ASCII标准解释，而字节值128~255则当作无法解析的字符。如果设置为SQL_ASCII，就不会有编码转换。该函数调用openssl三方库返回的数据的编码为非ASCII数据，因此当数据库服务端字符集设置为SQL_ASCII时，客户端编码也需设置为SQL_ASCII ，否则会报错。因为数据库不会帮助转换或者校验非ASCII字符。

- gs_digest(input_string, hash_algorithm)

  描述：使用指定的哈希算法，对输入的字符串计算哈希，并且以十六进制数作为返回值。

  参数解释：

  – input_string: 需要被计算哈希的字符串，不能为NULL。

  – hash_algorithm: 指定的哈希计算算法，当前支持SHA256，SHA384，SHA512和SM3，支持大写和小写。使用不支持的哈希算法则会报错。

  返回值类型：text

  示例：

  ```
  gaussdb=# SELECT pg_catalog.gs_digest('gaussdb', 'sha256');
                              gs_digest
  ------------------------------------------------------------------
   4dc50d746f4e04f9b446986b34a0050e358fbfb8bc1fba314c54b52a417b0b8e
  (1 row)
  ```

- gs_password_deadline

  描述：显示当前账户密码离过期还距离多少天。

  返回值类型：interval

  示例：

  ```
  gaussdb=# SELECT gs_password_deadline();
    gs_password_deadline
  -------------------------
   83 days 17:44:32.196094
  (1 row)
  ```

- gs_password_notifytime()

  描述：显示账户密码到期前提醒的天数。

  返回值类型：int32

- login_audit_messages(BOOLEAN)

  描述：查看登录用户的登录信息。

返回值类型：元组

示例：

- 查看上一次登录成功的日期、时间和IP等信息。

```
gaussdb=> SELECT * FROM login_audit_messages(true);
 username | database |       logintime        |    mytype     | result | client_conninfo
----------+----------+------------------------+---------------+--------+-----------------
 omm      | testdb | 2020-06-29 21:56:40+08 | login_success | ok     | gsql@[local]
(1 row)
```

- 查看自从上一次登录成功以来登录失败的尝试次数、日期和时间。

```
gaussdb=>  SELECT * FROM login_audit_messages(false);
 username | database |       logintime        |    mytype    | result |  client_conninfo
----------+----------+------------------------+--------------+--------+-------------------
 omm      | testdb | 2020-06-29 21:57:55+08 | login_failed | failed | [unknown]@[local]
 omm      | testdb | 2020-06-29 21:57:53+08 | login_failed | failed | [unknown]@[local]
(2 rows)
```

- login_audit_messages_pid

  描述：查看登录用户的登录信息。与login_audit_messages的区别在于结果基于当前backendid向前查找。所以不会因为同一用户的后续登录，而影响本次登录的查询结果。也就是查询不到该用户后续登录的信息。

  返回值类型：元组

  说明

  在开启线程池的情况下，由于线程切换，同一session中获取到的backendid可能会发生变化，因此会造成多次调用该函数返回值不一致的情况。不建议用户在开启线程池的情况下调用此函数。

  示例：

  - 查看上一次登录成功的日期、时间和IP等信息。

```
gaussdb=> SELECT * FROM login_audit_messages_pid(true);
 username | database |       logintime        |    mytype     | result | client_conninfo |    backendid
----------+----------+------------------------+---------------+--------+-----------------+-----------------
 omm      | testdb | 2020-06-29 21:56:40+08 | login_success | ok     | gsql@[local]    |
139823109633792
(1 row)
```

  - 查看自从上一次登录成功以来登录失败的尝试次数、日期和时间。

```
gaussdb=> SELECT * FROM login_audit_messages_pid(false);
 username | database |       logintime        |    mytype    | result |  client_conninfo  |    backendid
----------+----------+------------------------+--------------+--------+-------------------+-----------------
 omm      | testdb | 2020-06-29 21:57:55+08 | login_failed | failed | [unknown]@[local] |
139823109633792
 omm      | testdb | 2020-06-29 21:57:53+08 | login_failed | failed | [unknown]@[local] |
139823109633792
(2 rows)
```

- inet_server_addr

  描述：显示服务器IP信息。

  返回值类型：inet

  示例：

```
gaussdb=# SELECT inet_server_addr();
 inet_server_addr
------------------
 10.10.0.13
(1 row)
```

  说明

  - 上面是以客户端在10.10.0.50上，服务器端在10.10.0.13上为例。
  - 如果是通过本地连接，使用此接口显示为空。

- inet_client_addr
  描述：显示客户端IP信息。
  返回值类型：inet
  示例：

```
gaussdb=# SELECT inet_client_addr();
 inet_client_addr
------------------
 10.10.0.50
(1 row)
```

  说明

  - 上面是以客户端在10.10.0.50上，服务器端在10.10.0.13上为例。
  - 如果是通过本地连接，使用此接口显示为空。

- pg_query_audit
  描述：查看数据库主节点审计日志。
  返回值类型：record
  函数返回字段如下：

名称	类型	描述
time	timestamp with time zone	操作时间
type	text	操作类型
result	text	操作结果
userid	oid	用户id
username	text	执行操作的用户名
database	text	数据库名称
client_conninfo	text	客户端连接信息
object_name	text	操作对象名称
detail_info	text	执行操作详细信息
node_name	text	节点名称
thread_id	text	线程id
local_port	text	本地端口
remote_port	text	远端端口

- pg_delete_audit
  描述：删除指定时间段的审计日志。
  返回值类型：void
- alldigitsmasking

描述：脱敏策略的内部函数，对所有字符进行脱敏。
参数：col text, letter character default '0'
返回值类型：text

- creditcardmasking
  描述：脱敏策略的内部函数，对所有信用卡信息进行脱敏。
  参数：col text, letter character default 'x'
  返回值类型：text
- randommasking
  描述：脱敏策略的内部函数，使用随机策略。
  参数：col text
  返回值类型：text
- fullemailmasking
  描述：脱敏策略的内部函数，对出现最后一个'.'之前的文本（除'@'之外）进行脱敏。
  参数：col text, letter character default 'x'
  返回值类型：text
- basicemailmasking
  描述：脱敏策略的内部函数，对出现第一个'@'之前的文本进行脱敏。
  参数：col text, letter character default 'x'
  返回值类型：text
- shufflemasking
  描述：脱敏策略的内部函数，对字符进行乱序排列。
  参数：col text
  返回值类型：text

7.5.21 密态函数和操作符

- byteawithoutorderwithequalcolin(cstring)
  描述：将输入转码转化成内部byteawithoutorderwithequalcol形式。
  参数类型：cstring
  返回值类型：byteawithoutorderwithequalcol
- byteawithoutorderwithequalcolout(byteawithoutorderwithequalcol)
  描述：将内部byteawithoutorderwithequalcol类型的数据转码转化为cstring类型。
  参数类型：byteawithoutorderwithequalcol
  返回值类型：cstring
- byteawithoutorderwithequalcolsend(byteawithoutorderwithequalcol)
  描述：将byteawithoutorderwithequalcol类型的数据转码转化为bytea类型。
  参数类型：byteawithoutorderwithequalcol
  返回值类型：bytea
- byteawithoutorderwithequalcolrecv(internal)
  描述：将internal类型的数据转码转化为byteawithoutorderwithequalcol类型。

参数类型：internal

返回值类型：byteawithoutorderwithequalcol

- byteawithoutorderwithequalcoltypmodin(cstring[])

  描述：将cstring[]类型的数据转码转化为byteawithoutorderwithequalcol类型。

  参数类型：cstring[]

  返回值类型：int4

- byteawithoutorderwithequalcoltypmodout(int4)

  描述：将int4类型的数据转码转化为cstring类型。

  参数类型：int4

  返回值类型：cstring

- byteawithoutordercolin(cstring)

  描述：将输入转码转化成内部byteawithoutordercolin形式。

  参数类型：cstring

  返回值类型：byteawithoutordercol

- byteawithoutordercolout(byteawithoutordercol)

  描述：将内部byteawithoutordercol类型的数据转码转化为cstring类型。

  参数类型：byteawithoutordercol

  返回值类型：cstring

- byteawithoutordercolsend(byteawithoutordercol)

  描述：将byteawithoutordercol类型的数据转码转化为bytea类型。

  参数类型：byteawithoutordercol

  返回值类型：bytea

- byteawithoutordercolrecv(internal)

  描述：将internal类型的数据转码转化为byteawithoutordercol类型。

  参数类型：internal

  返回值类型：byteawithoutordercol

- byteawithoutorderwithequalcolcmp(byteawithoutorderwithequalcol, byteawithoutorderwithequalcol)

  描述：比较两个byteawithoutorderwithequalcol类型的数据大小，若第一个参数小于第二个参数，返回-1，若等于，返回0，若大于，则返回1。

  参数类型：byteawithoutorderwithequalcol, byteawithoutorderwithequalcol

  返回值类型：int4

- byteawithoutorderwithequalcolcmpbytear(byteawithoutorderwithequalcol, bytea)

  描述：比较byteawithoutorderwithequalcol和bytea数据大小，若第一个参数小于第二个参数，返回-1，若等于，返回0，若大于，则返回1。

  参数类型：byteawithoutorderwithequalcol, bytea

  返回值类型：int4

- byteawithoutorderwithequalcolcmpbyteal(bytea, byteawithoutorderwithequalcol)

  描述：比较bytea和byteawithoutorderwithequalcol数据大小，若第一个参数小于第二个参数，返回-1，若等于，返回0，若大于，则返回1。

参数类型：bytea, byteawithoutorderwithequalcol

返回值类型：int4

- byteawithoutorderwithequalcoleq(byteawithoutorderwithequalcol, byteawithoutorderwithequalcol)

  描述：比较两个byteawithoutorderwithequalcol类型的数据是否相同，相同则返回true，否则返回false。

  参数类型：byteawithoutorderwithequalcol, byteawithoutorderwithequalcol

  返回值类型：bool

- byteawithoutorderwithequalcoleqbyteal(bytea, byteawithoutorderwithequalcol)

  描述：比较bytea和byteawithoutorderwithequalcol数据是否相同，相同则返回true，否则返回false。

  参数类型：bytea，byteawithoutorderwithequalcol

  返回值类型：bool

- byteawithoutorderwithequalcoleqbytear(byteawithoutorderwithequalcol, bytea)

  描述：比较byteawithoutorderwithequalcol和bytea数据是否相同，相同则返回true，否则返回false。

  参数类型：byteawithoutorderwithequalcol, bytea

  返回值类型：bool

- byteawithoutorderwithequalcolne(byteawithoutorderwithequalcol, byteawithoutorderwithequalcol)

  描述：比较两个byteawithoutorderwithequalcol类型的数据是否不相同，不相同则返回true，否则返回false。

  参数类型：byteawithoutorderwithequalcol, byteawithoutorderwithequalcol

  返回值类型：bool

- byteawithoutorderwithequalcolnebyteal(bytea, byteawithoutorderwithequalcol)

  描述：比较bytea和byteawithoutorderwithequalcol数据是否相同，相同则返回true，否则返回false。

  参数类型：bytea，byteawithoutorderwithequalcol

  返回值类型：bool

- byteawithoutorderwithequalcolnebytear(byteawithoutorderwithequalcol, bytea)

  描述：比较byteawithoutorderwithequalcol和bytea数据是否不相同，相同则返回true，否则返回false。

  参数类型：byteawithoutorderwithequalcol, bytea

  返回值类型：bool

- hll_hash_byteawithoutorderwithequalcol(byteawithoutorderwithequalcol)

  描述：返回byteawithoutorderwithequalcol的hll哈希值。

  参数类型：byteawithoutorderwithequalcol

  返回值类型：hll_hashval

示例

byteawithoutorderwithequalcolin、byteawithoutorderwithequalcolout等密态等值函数为数据库内核中数据类型byteawithoutorderwithequalcol指定的in、out、send、recv等读写格式转换函数，具体可参考bytea类型的byteain、byteaout等函数，但会对本地的cek进行验证，需要密文字段中有本地存在的cekoid才能执行成功。

```
-- 例如存在加密表int_type，int_col2为其加密列

-- 使用非密态客户端连接数据库，查询加密列密文
gaussdb=# SELECT int_col2 FROM int_type;
                                                            int_col2
------------------------------------------------------------------------------------------------------------------------------------------------------

\x01c35301bf421c8edf38c34704bcc82838742917778ccb402a1b7452ad4a6ac7371acc0ac33100000035fe3424919854c86194f1aa5bb4e1ca656e8fc6d05324a1419b69f488bdc3c6
(1 row)

-- 将加密列密文当做byteawithoutorderwithequalcolin入参，格式从cstring输入转码转化成内部byteawithoutorderwithequalcol形式
gaussdb=# SELECT byteawithoutorderwithequalcolin('\x01c35301bf421c8edf38c34704bcc82838742917778ccb402a1b7452ad4a6ac7371acc0ac33100000035fe3424919854c86194f1aa5bb4e1ca656e8fc6d05324a1419b69f488bdc3c6');
                                                 byteawithoutorderwithequalcolin
------------------------------------------------------------------------------------------------------------------------------------------------------

\x01c35301bf421c8edf38c34704bcc82838742917778ccb402a1b7452ad4a6ac7371acc0ac33100000035fe3424919854c86194f1aa5bb4e1ca656e8fc6d05324a1419b69f488bdc3c6
(1 row)
```

由于byteawithoutorderwithequalcolin等的实现会对cek进行查找，并且判断是否为正常加密后的数据类型。

因此如果用户输入数据的格式不是加密后的数据格式，并且在本地不存在对应cek的情况下，会返回错误。

```
gaussdb=# SELECT * FROM byteawithoutorderwithequalcolsend('\x907219912381298461289346129'::byteawithoutorderwithequalcol);
ERROR:  cek with OID 596711794 not found
LINE 1: SELECT * FROM byteawithoutorderwithequalcolsend('\x907219912...
                                                        ^
gaussdb=# SELECT * FROM byteawithoutordercolout('\x90721901999999999999912381298461289346129');
ERROR:  cek with OID 2566986098 not found
LINE 1: SELECT * FROM byteawithoutordercolout('\x9072190199999999999...

gaussdb=# SELECT * FROM byteawithoutorderwithequalcolrecv('\x90721901999999999999912381298461289346129'::byteawithoutorderwithequalcol);
ERROR:  cek with OID 2566986098 not found
                                          ^
gaussdb=# SELECT * FROM byteawithoutorderwithequalcolsend('\x90721901999999999999912381298461289346129'::byteawithoutorderwithequalcol);
ERROR:  cek with OID 2566986098 not found
LINE 1: SELECT * FROM byteawithoutorderwithequalcolsend('\x907219019...
                                                        ^
```

7.5.22 返回集合的函数

序列号生成函数

- generate_series(start, stop)
  描述：生成一个数值序列，从start到stop，步长为1。

参数类型：int、bigint、numeric

返回值类型：setof int、setof bigint、setof numeric（与参数类型相同）

- generate_series(start, stop, step)

  描述：生成一个数值序列，从start到stop，步长为step。

  参数类型：int、bigint、numeric

  返回值类型：setof int、setof bigint、setof numeric（与参数类型相同）

- generate_series(start, stop, step interval)

  描述：生成一个数值序列，从start到stop，步长为step。

  参数类型：timestamp或timestamp with time zone

  返回值类型：setof timestamp或setof timestamp with time zone（与参数类型相同）

如果step是正数且start大于stop，则返回零行。相反，如果step是负数且start小于stop，则也返回零行。如果输入是NULL，同样产生零行。如果step为零则是一个错误。

示例：

```
gaussdb=# SELECT * FROM generate_series(2,4);
 generate_series
-----------------
               2
               3
               4
(3 rows)

gaussdb=# SELECT * FROM generate_series(5,1,-2);
 generate_series
-----------------
               5
               3
               1
(3 rows)

gaussdb=# SELECT * FROM generate_series(4,3);
 generate_series
-----------------
(0 rows)

--这个示例应用于date-plus-integer操作符。
gaussdb=# SELECT current_date + s.a AS dates FROM generate_series(0,14,7) AS s(a);
   dates
------------
 2017-06-02
 2017-06-09
 2017-06-16
(3 rows)

gaussdb=# SELECT * FROM generate_series('2008-03-01 00:00'::timestamp, '2008-03-04 12:00', '10 hours');
   generate_series
---------------------
 2008-03-01 00:00:00
 2008-03-01 10:00:00
 2008-03-01 20:00:00
 2008-03-02 06:00:00
 2008-03-02 16:00:00
 2008-03-03 02:00:00
 2008-03-03 12:00:00
 2008-03-03 22:00:00
 2008-03-04 08:00:00
(9 rows)
```


下标生成函数

- generate_subscripts(array anyarray, dim int)

  描述：生成一系列包括给定数组的下标。

  返回值类型：setof int

- generate_subscripts(array anyarray, dim int, reverse boolean)

  描述：生成一系列包括给定数组的下标。当reverse为真时，该系列则以相反的顺序返回。

  返回值类型：setof int

generate_subscripts是一个为给定数组中的指定维度生成有效下标集的函数。如果数组中没有所请求的维度或者NULL数组，返回零行（但是会给数组元素为空的返回有效下标）。示例：

```
--基本用法。
gaussdb=# SELECT generate_subscripts('{NULL,1,NULL,2}'::int[], 1) AS s;
 s
---
 1
 2
 3
 4
(4 rows)
--unnest一个2D数组。
gaussdb=# CREATE OR REPLACE FUNCTION unnest2(anyarray)
RETURNS SETOF anyelement AS $$
SELECT $1[i][j]
   FROM generate_subscripts($1,1) g1(i),
        generate_subscripts($1,2) g2(j);
$$ LANGUAGE sql IMMUTABLE;

gaussdb=# SELECT * FROM unnest2(ARRAY[[1,2],[3,4]]);
 unnest2
---------
       1
       2
       3
       4
(4 rows)

--删除函数。
gaussdb=# DROP FUNCTION unnest2;
```

7.5.23 条件表达式函数

条件表达式函数

- coalesce(expr1, expr2, ..., exprn)

  描述：

  返回参数列表中第一个非NULL的参数值。

  COALESCE(expr1, expr2) 等价于CASE WHEN expr1 IS NOT NULL THEN expr1 ELSE expr2 END。

  示例：

  ```
  gaussdb=# SELECT coalesce(NULL,'hello');
   coalesce
  ----------
   hello
  (1 row)
  ```

  备注：

  - 如果表达式列表中的所有表达式都等于NULL，则本函数返回NULL。
  - 它常用于在显示数据时用缺省值替换NULL。
  - 和CASE表达式一样，COALESCE不会计算不需要用来判断结果的参数；即在第一个非空参数右边的参数不会被计算。

- decode(base_expr, compare1, value1, Compare2,value2, … default)

  描述：把base_expr与后面的每个compare(n) 进行比较，如果匹配返回相应的value(n)。如果没有发生匹配，则返回default。

  示例：

  ```
  gaussdb=# SELECT decode('A','A',1,'B',2,0);
   case
  ------
   1
  (1 row)
  ```

  备注：

  - 不支持对xml数据类型的操作

- nullif(expr1, expr2)

  描述：当且仅当expr1和expr2相等时，NULLIF才返回NULL，否则它返回expr1。

  nullif(expr1, expr2) 逻辑上等价于CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END。

  示例：

  ```
  gaussdb=# SELECT nullif('hello','world');
   nullif
  --------
   hello
  (1 row)
  ```

  备注：

  - 不支持对xml数据类型的操作。
  - 如果两个参数的数据类型不同，则：
    - 若两种数据类型之间存在隐式转换，则以其中优先级较高的数据类型为基准将另一个参数隐式转换成该类型，转换成功则进行计算，转换失败则返回错误。如：

      ```
      gaussdb=# SELECT nullif('1234'::VARCHAR,123::INT4);
       nullif
      --------
        1234
      (1 row)
      gaussdb=# SELECT nullif('1234'::VARCHAR,'2012-12-24'::DATE);
      ERROR:  invalid input syntax for type timestamp: "1234"
      ```

    - 若两种数据类型之间不存在隐式转换，则返回错误。如：

      ```
      gaussdb=# SELECT nullif(TRUE::BOOLEAN,'2012-12-24'::DATE);
      ERROR:  operator does not exist: boolean = timestamp without time zone
      LINE 1: SELECT nullif(TRUE::BOOLEAN,'2012-12-24'::DATE) FROM sys_dummy;
      ^
      HINT:  No operator matches the given name and argument type(s). You might need to add explicit type casts.
      ```

- nvl(expr1 , expr2)

  描述：

  - 如果expr1为NULL则返回expr2。
  - 如果expr1非NULL，则返回expr1。

  示例：


```
gaussdb=# SELECT nvl('hello','world');
 nvl
-------
 hello
(1 row)
```

备注：参数expr1和expr2可以为任意类型，当NVL的两个参数不属于同类型时，看第二个参数是否可以向第一个参数进行隐式转换。如果可以则返回第一个参数类型，否则返回错误。

- nvl2(expr1 , expr2，expr3)

  描述：

  – 如果expr1为NULL，则返回expr3。
  – 如果expr1非NULL，则返回expr2。

  说明

  此函数在参数a_format_version值为10c和a_format_dev_version值为s1的情况下有效。

  示例：

```
gaussdb=# SELECT nvl2('hello','world','other');
 case
-------
 world
(1 row)
```

  备注：参数expr2和expr3可以为任意类型， 当NVL2的后面两个参数不属于同类型时，看expr3参数是否可以向expr2参数进行隐式转换，如果不能隐式转换，会返回错误。如果第一个参数是数值类型，函数在将本参数和其他参数都转换为numeric类型，然后进行比较，对于不能转换的，提示出错信息；第一个参数是其他类型的，函数将其他参数都转换为第一个参数的类型进行比较，对于不能转换的，提示出错信息。

- greatest(expr1 [, ...])

  描述：获取并返回参数列表中值最大的表达式的值。

  返回值类型：

  示例：

```
gaussdb=# SELECT greatest(1*2,2-3,4-1);
 greatest
----------
        3
(1 row)
gaussdb=# SELECT greatest('HARRY', 'HARRIOT', 'HAROLD');
 greatest
----------
 HARRY
(1 row)
```

  备注：

  不支持对xml数据类型的操作。

  说明

  此函数在参数a_format_version值为10c和a_format_dev_version值为s1的情况下：

  - 如果参数中有任意一个参数的值为null，函数返回null。
  - 如果第一个参数是数值类型，函数将第一个参数和其他参数都转换为numeric类型，然后进行比较，对于不能转换的，提示出错信息；第一个参数是其他类型的，函数将其他参数都转换为第一个参数的类型进行比较，对于不能转换的，提示出错信息。

- least(expr1 [, ...])

  描述：获取并返回参数列表中值最小的表达式的值。

示例：

```
gaussdb=# SELECT least(1*2,2-3,4-1);
 least
-------
    -1
(1 row)
gaussdb=# SELECT least('HARRY','HARRIOT','HAROLD');
 least
--------
 HAROLD
(1 row)
```

备注：

不支持对xml数据类型的操作。

说明

此函数在参数a_format_version值为10c和a_format_dev_version值为s1的情况下：

- 如果参数中有任意一个参数的值为null，函数返回null。
- 如果第一个参数是数值类型，函数将第一个参数和其他参数都转换为numeric类型，然后进行比较，对于不能转换的，提示出错信息；第一个参数是其他类型的，函数将其他参数都转换为第一个参数的类型进行比较，对于不能转换的，提示出错信息。

- EMPTY_BLOB()

  描述：使用EMPTY_BLOB在INSERT或UPDATE语句中初始化一个BLOB变量，取值为NULL。

  返回值类型：BLOB

  示例：

  ```
  --新建表
  gaussdb=# CREATE TABLE blob_tb(b blob,id int);
  --插入数据
  gaussdb=# INSERT INTO blob_tb VALUES (empty_blob(),1);
  --删除表
  gaussdb=# DROP TABLE blob_tb;
  ```

  备注：使用DBE_LOB.GET_LENGTH求得的长度为0。

- EMPTY_CLOB()

  描述：使用EMPTY_CLOB在INSERT或UPDATE语句中初始化一个CLOB变量，取值为空。

  说明

  此函数在参数a_format_version值为10c和a_format_dev_version值为s1的情况下有效。

  返回值类型：CLOB

  示例：

  ```
  --新建表
  gaussdb=# CREATE TABLE clob_tb(c clob,id int);
  --插入数据
  gaussdb=# INSERT INTO clob_tb VALUES (empty_clob(),1);
  --删除表
  gaussdb=# DROP TABLE clob_tb;
  ```

  备注：使用DBE_LOB.GET_LENGTH求得的长度为0。

- lnnvl(condition)

  描述：lnnvl用于某个查询语句的where子句中，如果条件为true就返回false，如果条件为unknown或者false，就返回true。

  condition：必须为逻辑表达式，但不能用于复合条件如and，or或者between。

  返回类型：bool

示例：

```
--新建表
gaussdb=# CREATE TABLE student_demo (name VARCHAR2(20), grade  NUMBER(10,2));
CREATE TABLE

--插入数据
gaussdb=# INSERT INTO student_demo VALUES ('name0',0);
INSERT 0 1
gaussdb=# INSERT INTO student_demo VALUES ('name1',1);
INSERT 0 1
gaussdb=# INSERT INTO student_demo VALUES ('name2',2);
INSERT 0 1

--调用lnnvl
gaussdb=# SELECT * FROM student_demo WHERE LNNVL(name = 'name1');
 name  | grade
-------+-------
 name0 |  0.00
 name2 |  2.00
(2 rows)

--删除表
gaussdb=# DROP TABLE student_demo;
DROP TABLE
```

说明

此函数在参数a_format_version值为10c和a_format_dev_version值为s2的情况下生效。

7.5.24 系统信息函数

会话信息函数

- SYS_CONTEXT()

  描述：返回当前时刻与上下文命名空间'namespace'关联的参数'parameter'的值。

  返回值类型：text

  示例：

  ```
  select SYS_CONTEXT('userenv','NLS_CURRENCY');
   sys_context
  -------------
   $
  (1 row)

  select SYS_CONTEXT('userenv','NLS_DATE_FORMAT');
    sys_context
  ----------------
   ISO, MDY
  (1 row)

  select SYS_CONTEXT('userenv','NLS_DATE_LANGUAGE');
   sys_context
  -------------
   en_US.UTF-8
  (1 row)
  ```

- current_catalog

  描述：当前数据库的名称（在标准SQL中称"catalog"）。

  返回值类型：name

  示例：

  ```
  testdb=# SELECT current_catalog;
   current_database
  ------------------
  ```



```
 testdb
(1 row)
```

- current_database()

  描述：当前数据库的名称。

  返回值类型：name

  示例：

```
testdb=# SELECT current_database();
 current_database
------------------
 testdb
(1 row)
```

- current_query()

  描述：由客户端提交的当前执行语句（可能包含多个声明）。

  返回值类型：text

  示例：

```
gaussdb=# SELECT current_query();
     current_query
-------------------------
 SELECT current_query();
(1 row)
```

- current_schema[()]

  描述：当前模式的名称。

  返回值类型：name

  示例：

```
gaussdb=# SELECT current_schema();
 current_schema
----------------
 public
(1 row)
```

  备注：current_schema返回在搜索路径中第一个顺位有效的模式名。（如果搜索路径为空则返回NULL，没有有效的模式名也返回NULL）。如果创建表或者其他命名对象时没有声明目标模式，则将使用这些对象的模式。

- current_schemas(Boolean)

  描述：搜索路径中的模式名称。

  返回值类型：name[]

  示例：

```
gaussdb=# SELECT current_schemas(true);
   current_schemas
---------------------
 {pg_catalog,public}
(1 row)
```

  备注：

  current_schemas(Boolean)返回搜索路径中所有模式名称的数组。布尔选项决定像pg_catalog这样隐含包含的系统模式是否包含在返回的搜索路径中。

  说明

  搜索路径可以通过运行时设置更改。命令是：

```
SET search_path TO schema [, schema, ...]
```

- current_user

  描述：当前执行环境下的用户名。

返回值类型：name

示例：

```
gaussdb=# SELECT current_user;
 current_user
--------------
 omm
(1 row)
```

备注：current_user是用于权限检查的用户标识。通常，他表示会话用户，但是可以通过**SET ROLE**改变他。在函数执行的过程中随着属性SECURITY DEFINER的改变，其值也会改变。

- definer_current_user

  描述：当前执行环境下的用户名。

  返回值类型：name

  示例：

  ```
  gaussdb=# SELECT definer_current_user();
   definer_current_user
  ----------------------
   omm
  (1 row)
  ```

- pg_current_sessionid()

  描述：当前执行环境下的会话ID。

  返回值类型：text

  示例：

  ```
  gaussdb=# SELECT pg_current_sessionid();
      pg_current_sessionid
  ----------------------------
   1579228402.140190434944768
  (1 row)
  ```

  备注：pg_current_sessionid()是用于获取当前执行环境下的会话ID。其组成结构为：时间戳.会话ID，当线程池模式开启（enable_thread_pool=on）时，会话ID为SessionID；而线程池模式关闭时，会话ID为ThreadID。

- pg_current_sessid

  描述：当前执行环境下的会话ID。

  返回值类型：text

  示例：

  ```
  gaussdb=# select pg_current_sessid();
  pg_current_sessid
  -------------------
  140308875015936
  (1 row)
  ```

  备注：在线程池模式下获得当前会话的会话ID，非线程池模式下获得当前会话对应的后台线程ID。

- pg_current_userid

  描述：当前用户ID。

  返回值类型：text

  ```
  gaussdb=# SELECT pg_current_userid();
  pg_current_userid
  -------------------
  10
  (1 row)
  ```

- working_version_num()

描述：版本序号信息。返回一个系统兼容性有关的版本序号。

返回值类型：int

示例：

```
gaussdb=# SELECT working_version_num();
 working_version_num
---------------------
               92231
(1 row)
```

- tablespace_oid_name()

  描述: 根据表空间oid，查找表空间名称。

  返回值类型：text

  示例：

  ```
  gaussdb=# select tablespace_oid_name(1663);
   tablespace_oid_name
  ---------------------
   pg_default
  (1 row)
  ```

- inet_client_addr()

  描述：连接的远端地址。inet_client_addr返回当前客户端的IP地址。

  说明

  此函数只有在远程连接模式下有效。

  返回值类型：inet

  示例：

  ```
  gaussdb=# SELECT inet_client_addr();
   inet_client_addr
  ------------------
   10.10.0.50
  (1 row)
  ```

- inet_client_port()

  描述：连接的远端端口。inet_client_port返回当前客户端的端口号。

  说明

  此函数只有在远程连接模式下有效。

  返回值类型：int

  示例：

  ```
  gaussdb=# SELECT inet_client_port();
   inet_client_port
  ------------------
             33143
  (1 row)
  ```

- inet_server_addr()

  描述：连接的本地地址。inet_server_addr返回服务器接收当前连接用的IP地址。

  说明

  此函数只有在远程连接模式下有效。

  返回值类型：inet

  示例：

  ```
  gaussdb=# SELECT inet_server_addr();
   inet_server_addr
  ```



```
------------------
 10.10.0.13
(1 row)
```

- inet_server_port()

  描述：连接的本地端口。inet_server_port返回接收当前连接的端口号。如果是通过Unix-domain socket连接的，则所有这些函数都返回NULL。

  说明

  此函数只有在远程连接模式下有效。

  返回值类型：int

  示例：

```
gaussdb=# SELECT inet_server_port();
 inet_server_port
------------------
             8000
(1 row)
```

- pg_backend_pid()

  描述：当前会话连接的服务进程的进程ID。

  返回值类型：int

  示例：

```
gaussdb=# SELECT pg_backend_pid();
 pg_backend_pid
-----------------
 140229352617744
(1 row)
```

- pg_conf_load_time()

  描述：配置加载时间。pg_conf_load_time返回最后加载服务器配置文件的时间戳。

  返回值类型：timestamp with time zone

  示例：

```
gaussdb=# SELECT pg_conf_load_time();
      pg_conf_load_time
-------------------------------
 2017-09-01 16:05:23.89868+08
(1 row)
```

- pg_my_temp_schema()

  描述：会话的临时模式的OID，不存在则为0。

  返回值类型：oid

  示例：

```
gaussdb=# SELECT pg_my_temp_schema();
 pg_my_temp_schema
-------------------
                 0
(1 row)
```

  备注：pg_my_temp_schema返回当前会话中临时模式的OID，如果不存在（没有创建临时表）的话则返回0。如果给定的OID是其它会话中临时模式的OID，pg_is_other_temp_schema则返回true。

- pg_is_other_temp_schema(oid)

  描述：是否为另一个会话的临时模式。

  返回值类型：Boolean

  示例：


```
gaussdb=# SELECT pg_is_other_temp_schema(25356);
 pg_is_other_temp_schema
-------------------------
 f
(1 row)
```

- pg_listening_channels()

  描述：会话正在侦听的信道名称。

  返回值类型：setof text

  示例：

  ```
  gaussdb=# SELECT pg_listening_channels();
   pg_listening_channels
  -----------------------
  (0 rows)
  ```

  备注：pg_listening_channels返回当前会话正在侦听的一组信道名称。

- pg_postmaster_start_time()

  描述：服务器启动时间。pg_postmaster_start_time返回服务器启动时的timestamp with time zone。

  返回值类型：timestamp with time zone

  示例：

  ```
  gaussdb=# SELECT pg_postmaster_start_time();
     pg_postmaster_start_time
  ------------------------------
   2017-08-30 16:02:54.99854+08
  (1 row)
  ```

- pg_get_ruledef(rule_oid)

  描述：获取规则的CREATE RULE命令。

  返回值类型：text

  示例：

  ```
  gaussdb=# select * from pg_get_ruledef(24828);
                                pg_get_ruledef
  ---------------------------------------------------------------------
   CREATE RULE t1_ins AS ON INSERT TO t1 DO INSTEAD INSERT INTO t2 (id) VALUES (new.id);
  (1 row)
  ```

- sessionid2pid()

  描述: 从sessionid中得到pid信息（例如，gs_session_stat中sessid列）。

  返回值类型: int8

  示例：

  ```
  gaussdb=# select sessionid2pid(sessid::cstring) from gs_session_stat limit 2;
    sessionid2pid
  -----------------
   139973107902208
   139973107902208
  (2 rows)
  ```

- session_context('namespace' , 'parameter')

  描述：获取并返回指定namespace下参数parameter的值。

  返回值类型：VARCHAR

  示例：

  ```
  gaussdb=# SELECT session_context('USERENV', 'CURRENT_SCHEMA');
  session_context
  -------------------
    public
  (1 row)
  ```


备注：当前支持的parameter：current_user, current_schema, client_info, ip_address, sessionid, sid.

- pg_trigger_depth()

  描述：触发器的嵌套层次。

  返回值类型：int

  示例：

```
gaussdb=# SELECT pg_trigger_depth();
 pg_trigger_depth
------------------
                0
(1 row)
```

- session_user

  描述：会话用户名。

  返回值类型：name

  示例：

```
gaussdb=# SELECT session_user;
 session_user
--------------
 omm
(1 row)
```

  备注：session_user通常是连接当前数据库的初始用户，不过系统管理员可以用**SET SESSION AUTHORIZATION**修改这个设置。

- user

  描述：等价于current_user。

  返回值类型：name

  示例：

```
gaussdb=# SELECT user;
 current_user
--------------
 omm
(1 row)
```

- getpgusername()

  描述：获取数据库用户名。

  返回值类型：name

  示例：

```
gaussdb=# select getpgusername();
 getpgusername
---------------
 GaussDB_userna
(1 row)
```

- getdatabaseencoding()

  描述：获取数据库编码方式。

  返回值类型：name

  示例：

```
gaussdb=# select getdatabaseencoding();
 getdatabaseencoding
---------------------
 SQL_ASCII
(1 row)
```

- version()

描述：版本信息。version返回一个描述服务器版本信息的字符串。

返回值类型：text

示例：

```
gaussdb=# select version();
version
------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------
gaussdb (GaussDB Kernel 503.1.XXX build fab4f5ea) compiled at 2021-10-24 11:58:22 commit 3086
last mr 6592 release
(1 row)
```

- opengauss_version()

  描述：openGauss版本信息。

  返回值类型：text

  使用示例如下，查询结果中的*x.x.x*请以实际输出为准：

  ```
  gaussdb=# select opengauss_version();
   opengauss_version
  -------------------
   x.x.x
  (1 row)
  ```

- gs_deployment()

  描述：当前系统的部署形态信息。

  返回值类型：text

  示例：

  ```
  gaussdb=#  select gs_deployment();
     gs_deployment
  ---------------------
   BusinessCentralized
  (1 row)
  ```

- get_hostname()

  描述：返回当前节点的hostname。

  返回值类型：text

  示例：

  ```
  gaussdb=# SELECT get_hostname();
   get_hostname
  --------------
   linux-user
  (1 row)
  ```

- get_nodename()

  描述：返回当前节点的名字。

  返回值类型：text

  示例：

  ```
  gaussdb=# SELECT get_nodename();
   get_nodename
  --------------
   datanode1
  (1 row)
  ```

- get_schema_oid(cstring)

  描述：返回查询schema的oid。

  返回值类型：oid

  示例：

  ```
  gaussdb=# SELECT get_schema_oid('public');
   get_schema_oid
  ```



```
----------------
        2200
(1 row)
```

- get_client_info()

  描述：返回客户端信息。

  返回值类型：record

访问权限查询函数

DDL类权限ALTER、DROP、COMMENT、INDEX、VACUUM属于所有者固有的权限，隐式拥有。

以下访问权限查询函数仅表示用户是否具有某对象上的某种对象权限，即返回记录在系统表acl字段中的对象权限拥有情况。

- has_any_column_privilege(user, table, privilege)

  描述：指定用户是否有访问表任何列的权限。

  表 7-42 参数类型说明

参数名	合法入参类型
user	name, oid
table	text, oid
privilege	text

  返回类型：Boolean

- has_any_column_privilege(table, privilege)

  描述：当前用户是否有访问表任何列的权限，合法参数类型见表7-42。

  返回类型：Boolean

  备注：has_any_column_privilege检查用户是否以特定方式访问表的任何列。其参数可能与has_table_privilege类似，除了访问权限类型必须是SELECT、INSERT、UPDATE、COMMENT或REFERENCES的一些组合。

  说明

  拥有表的表级别权限则隐含的拥有该表每列的列级权限，因此如果与has_table_privilege参数相同，has_any_column_privilege总是返回true。但是如果授予至少一列的列级权限也返回成功。

- has_column_privilege(user, table, column, privilege)

  描述：指定用户是否有访问列的权限。

  表 7-43 参数类型说明

参数名	合法入参类型
user	name, oid
table	text, oid

参数名	合法入参类型
column	text, smallint
privilege	text

返回类型：Boolean

- has_column_privilege(table, column, privilege)

  描述：当前用户是否有访问列的权限，合法参数类型见表7-43。

  返回类型：Boolean

  备注：has_column_privilege检查用户是否以特定方式访问一列。其参数类似于has_table_privilege，可以通过列名或属性号添加列。想要的访问权限类型必须是SELECT、INSERT、UPDATE、COMMENT或REFERENCES的一些组合。

  说明

  拥有表的表级别权限则隐含的拥有该表每列的列级权限。

- has_cek_privilege(user, cek, privilege)

  描述：指定用户是否有访问列加密密钥CEK的权限。参数说明如下。

  表 7-44 参数类型说明

参数名	合法入参类型	描述	取值范围
user	name，oid	用户	用户名字或id。
cek	text，oid	列加密密钥	列加密密钥名称或id。
privilege	text	权限	• USAGE：允许使用指定列加密密钥。 • DROP：允许删除指定列加密密钥。

  返回类型：Boolean

- has_cmk_privilege(user, cmk, privilege)

  描述：指定用户是否有访问客户端加密主密钥CMK的权限。参数说明如下。

  表 7-45 参数类型说明

参数名	合法入参类型	描述	取值范围
user	name，oid	用户	用户名字或id。
cmk	text，oid	客户端加密主密钥	客户端加密主密钥名称或id。

参数名	合法入参类型	描述	取值范围
privilege	text	权限	• USAGE：允许使用指定客户端加密主密钥。 • DROP：允许删除指定客户端加密主密钥。

返回类型：Boolean

- has_database_privilege(user, database, privilege)
  描述：指定用户是否有访问数据库的权限。参数说明如下。

  表 7-46 参数类型说明

参数名	合法入参类型
user	name, oid
database	text, oid
privilege	text

  返回类型：Boolean

- has_database_privilege(database, privilege)
  描述：当前用户是否有访问数据库的权限，合法参数类型请参见表7-46。
  返回类型：Boolean
  备注：has_database_privilege检查用户是否能以在特定方式访问数据库。其参数类似has_table_privilege。访问权限类型必须是CREATE、CONNECT、TEMPORARY、ALTER、DROP、COMMENT或TEMP（等价于TEMPORARY）的一些组合。
- has_directory_privilege(user, directory, privilege)
  描述：指定用户是否有访问directory的权限。

  表 7-47 参数类型说明

参数名	合法入参类型
user	name, oid
directory	text, oid
privilege	text

  返回类型：Boolean

- has_directory_privilege(directory, privilege)

描述：当前用户是否有访问directory的权限，合法参数类型请参见表7-47。

返回类型：Boolean

- has_foreign_data_wrapper_privilege(user, fdw, privilege)

  描述：指定用户是否有访问外部数据封装器的权限。

  表 7-48 参数类型说明

参数名	合法入参类型
user	name, oid
fdw	text, oid
privilege	text

  返回类型：Boolean

- has_foreign_data_wrapper_privilege(fdw, privilege)

  描述：当前用户是否有访问外部数据封装器的权限。合法参数类型请参见表7-48。

  返回类型：Boolean

  备注：has_foreign_data_wrapper_privilege检查用户是否能以特定方式访问外部数据封装器。其参数类似has_table_privilege。访问权限类型必须是USAGE。

- has_function_privilege(user, function, privilege)

  描述：指定用户是否有访问函数的权限。

  表 7-49 参数类型说明

参数名	合法入参类型
user	name, oid
function	text, oid
privilege	text

  返回类型：Boolean

- has_function_privilege(function, privilege)

  描述：当前用户是否有访问函数的权限。合法参数类型请参见表7-49。

  返回类型：Boolean

  备注：has_function_privilege检查一个用户是否能以指定方式访问一个函数。其参数类似has_table_privilege。使用文本字符而不是OID声明一个函数时，允许输入的类型和regprocedure数据类型一样（请参考对象标识符类型）。访问权限类型必须是EXECUTE、ALTER、DROP或COMMENT。

- has_language_privilege(user, language, privilege)

  描述：指定用户是否有访问语言的权限。

表 7-50 参数类型说明

参数名	合法入参类型
user	name, oid
language	text, oid
privilege	text

返回类型：Boolean

- has_language_privilege(language, privilege)

  描述：当前用户是否有访问语言的权限。合法参数类型请参见**表7-50**。

  返回类型：Boolean

  备注：has_language_privilege检查用户是否能以特定方式访问一个过程语言。其参数类似has_table_privilege。访问权限类型必须是USAGE。

- has_nodegroup_privilege(user, nodegroup, privilege)

  描述：检查用户是否有数据库节点访问权限。

  返回类型：Boolean

表 7-51 参数类型说明

参数名	合法入参类型
user	name, oid
nodegroup	text, oid
privilege	text

- has_nodegroup_privilege(nodegroup, privilege)

  描述：检查用户是否有数据库节点访问权限。参数与has_table_privilege类似。访问权限类型必须是USAGE、CREATE、COMPUTE、ALTER或DROP。

  返回类型：Boolean

- has_schema_privilege(user, schema, privilege)

  描述：指定用户是否有访问模式的权限。

  返回类型：Boolean

- has_schema_privilege(schema, privilege)

  描述：当前用户是否有访问模式的权限。

  返回类型：Boolean

  备注：has_schema_privilege检查用户是否能以特定方式访问一个模式。其参数类似has_table_privilege。访问权限类型必须是CREATE、USAGE、ALTER、DROP或COMMENT的一些组合。

- has_server_privilege(user, server, privilege)

  描述：指定用户是否有访问外部服务的权限。

  返回类型：Boolean

- has_server_privilege(server, privilege)

  描述：当前用户是否有访问外部服务的权限。

  返回类型：Boolean

  备注：has_server_privilege检查用户是否能以指定方式访问一个外部服务器。其参数类似has_table_privilege。访问权限类型必须是USAGE、ALTER、DROP或COMMENT之一的值。

- has_table_privilege(user, table, privilege)

  描述：指定用户是否有访问表的权限。

  返回类型：Boolean

- has_table_privilege(table, privilege)

  描述：当前用户是否有访问表的权限。

  返回类型：Boolean

  备注：has_table_privilege检查用户是否以特定方式访问表。用户可以通过名称或OID（pg_authid.oid）来指定，public表示为PUBLIC角色，或如果缺省该参数，则使用current_user。该表可以通过名称或者OID声明。如果用名称声明，则在必要时可以用模式进行修饰。如果使用文本字符串来声明所希望的权限类型，这个文本字符串必须是SELECT、INSERT、UPDATE、DELETE、TRUNCATE、REFERENCES、TRIGGER、ALTER、DROP、COMMENT、INDEX或VACUUM之一的值。可以给权限类型添加WITH GRANT OPTION，用来测试权限是否拥有授权选项。也可以用逗号分隔列出的多个权限类型，如果拥有任何所列出的权限，则结果便为true。

  示例：

  ```
  gaussdb=# SELECT has_table_privilege('tpcds.web_site', 'select');
   has_table_privilege
  ---------------------
   t
  (1 row)

  gaussdb=# SELECT has_table_privilege('omm', 'tpcds.web_site', 'select,INSERT WITH GRANT OPTION ');
   has_table_privilege
  ---------------------
   t
  (1 row)
  ```

- has_tablespace_privilege(user, tablespace, privilege)

  描述：指定用户是否有访问表空间的权限。

  返回类型：Boolean

- has_tablespace_privilege(tablespace, privilege)

  描述：当前用户是否有访问表空间的权限。

  返回类型：Boolean

  备注：has_tablespace_privilege检查用户是否能以特定方式访问一个表空间。其参数类似has_table_privilege。访问权限类型必须是CREATE、ALTER、DROP或COMMENT之一的值。

- pg_has_role(user, role, privilege)

  描述：指定用户是否有角色的权限。

  返回类型：Boolean

- pg_has_role(role, privilege)

  描述：当前用户是否有角色的权限。

  返回类型：Boolean

备注：pg_has_role检查用户是否能以特定方式访问一个角色。其参数类似has_table_privilege，除了public不能用做用户名。访问权限类型必须是MEMBER或USAGE的一些组合。 MEMBER表示的是角色中的直接或间接成员关系（也就是SET ROLE的权限），而USAGE表示无需通过SET ROLE也直接拥有角色的使用权限。

- has_any_privilege(user, privilege)

  描述：指定用户是否有某项ANY权限，若同时查询多个权限，只要具有其中一个则返回true。

  返回类型：Boolean

  表 7-52 参数类型说明

参数名	合法入参类型	描述	取值范围
user	name	用户	已存在的用户名。
privilege	text	ANY权限	可选取值： • CREATE ANY TABLE [WITH ADMIN OPTION] • ALTER ANY TABLE [WITH ADMIN OPTION] • DROP ANY TABLE [WITH ADMIN OPTION] • SELECT ANY TABLE [WITH ADMIN OPTION] • INSERT ANY TABLE [WITH ADMIN OPTION] • UPDATE ANY TABLE [WITH ADMIN OPTION] • DELETE ANY TABLE [WITH ADMIN OPTION] • CREATE ANY SEQUENCE [WITH ADMIN OPTION] • CREATE ANY INDEX [WITH ADMIN OPTION] • CREATE ANY FUNCTION [WITH ADMIN OPTION] • EXECUTE ANY FUNCTION [WITH ADMIN OPTION] • CREATE ANY PACKAGE [WITH ADMIN OPTION] • EXECUTE ANY PACKAGE [WITH ADMIN OPTION] • CREATE ANY TYPE [WITH ADMIN OPTION]

模式可见性查询函数

每个函数执行检查数据库对象类型的可见性。对于函数和操作符，如果在前面的搜索路径中没有相同的对象名称和参数的数据类型，则此对象是可见的。对于操作符类，则要同时考虑名称和相关索引的访问方法。

所有这些函数都需要使用OID来标识需要检查的对象。如果用户想通过名称测试对象，则使用OID别名类型（regclass、regtype、regprocedure、regoperator、regconfig或regdictionary）将会很方便。

比如，如果一个表所在的模式在搜索路径中，并且在前面的搜索路径中没有同名的表，则这个表是可见的。它等效于表可以不带明确模式修饰进行引用。比如，要列出所有可见表的名称：

```
gaussdb=# SELECT relname FROM pg_class WHERE pg_table_is_visible(oid);
```

- pg_collation_is_visible(collation_oid)
  描述：该排序是否在搜索路径中可见。
  返回类型：Boolean
- pg_conversion_is_visible(conversion_oid)
  描述：该转换是否在搜索路径中可见。
  返回类型：Boolean
- pg_function_is_visible(function_oid)
  描述：该函数是否在搜索路径中可见。
  返回类型：Boolean
- pg_opclass_is_visible(opclass_oid)
  描述：该操作符类是否在搜索路径中可见。
  返回类型：Boolean
- pg_operator_is_visible(operator_oid)
  描述：该操作符是否在搜索路径中可见。
  返回类型：Boolean
- pg_opfamily_is_visible(opclass_oid)
  描述：该操作符族是否在搜索路径中可见。
  返回类型：Boolean
- pg_table_is_visible(table_oid)
  描述：该表是否在搜索路径中可见。
  返回类型：Boolean
- pg_ts_config_is_visible(config_oid)
  描述：该文本检索配置是否在搜索路径中可见。
  返回类型：Boolean
- pg_ts_dict_is_visible(dict_oid)
  描述：该文本检索词典是否在搜索路径中可见。
  返回类型：Boolean
- pg_ts_parser_is_visible(parser_oid)
  描述：该文本搜索解析是否在搜索路径中可见。
  返回类型：Boolean

- pg_ts_template_is_visible(template_oid)

  描述：该文本检索模板是否在搜索路径中可见。

  返回类型：Boolean

- pg_type_is_visible(type_oid)

  描述：该类型（或域）是否在搜索路径中可见。

  返回类型：Boolean

系统表信息函数

- format_type(type_oid, typemod)

  描述：获取数据类型的SQL名称

  返回类型：text

  备注：format_type通过某个数据类型的OID以及可能的修饰词，返回其SQL名称。如果不知道具体的修饰词，则在修饰词的位置传入NULL。修饰词一般只对有长度限制的数据类型有意义。format_type所返回的SQL名称中包含数据类型的长度值，其大小是：实际存储长度len - sizeof(int32)，单位字节。原因是数据存储时需要32位的空间来存储用户对数据类型的自定义长度信息，即实际存储长度要比用户定义长度多4个字节。在下例中，format_type返回的SQL名称为“character varying(6)”，6表示varchar类型的长度值是6字节，因此该类型的实际存储长度为10字节。

  ```
  gaussdb=# SELECT format_type((SELECT oid FROM pg_type WHERE typname='varchar'), 10);
      format_type
  ----------------------
   character varying(6)
  (1 row)
  ```

- getdistributekey(table_name)

  描述：获取一个hash表的分布列。单机环境下不支持分布，该函数返回为空。

- pg_check_authid(role_oid)

  描述：检查是否存在给定oid的角色名。

  返回类型：Boolean

  示例：

  ```
  gaussdb=# select pg_check_authid(1);
  pg_check_authid
  -----------------
  f
  (1 row)
  ```

- pg_describe_object(catalog_id, object_id, object_sub_id)

  描述：获取数据库对象的描述。

  返回类型：text

  备注：pg_describe_object返回由目录OID，对象OID和一个（或许0个）子对象ID指定的数据库对象的描述。这有助于确认存储在pg_depend系统表中对象的身份。

- pg_get_constraintdef(constraint_oid)

  描述：获取约束的定义。

  返回类型：text

- pg_get_constraintdef(constraint_oid, pretty_bool)

  描述：获取约束的定义。

返回类型：text

备注：pg_get_constraintdef和pg_get_indexdef分别从约束或索引上使用创建命令进行重构。

- pg_get_expr(pg_node_tree, relation_oid)

  描述：反编译表达式的内部形式，假设其中的任何Vars都引用第二个参数指定的关系。

  返回类型：text

- pg_get_expr(pg_node_tree, relation_oid, pretty_bool)

  描述：反编译表达式的内部形式，假设其中的任何Vars都引用第二个参数指定的关系。

  返回类型：text

  备注：pg_get_expr反编译一个独立表达式的内部形式，比如一个字段的缺省值。便于检查系统表的内容。如果表达式可能包含关键字，则指定引用相关的OID作为第二个参数；如果没有关键字，为零即可。

- pg_get_functiondef(func_oid)

  描述：获取函数的定义。

  返回类型：text

  示例：

```
gaussdb=# SELECT * FROM pg_get_functiondef(598);
 headerlines |                  definition
-------------+-----------------------------------------------------
           4 | CREATE OR REPLACE FUNCTION pg_catalog.abbrev(inet)+
             | RETURNS text                                      +
             | LANGUAGE internal                                 +
             | IMMUTABLE STRICT NOT FENCED NOT SHIPPABLE         +
             | AS $function$inet_abbrev$function$                +
             |
(1 row)
```

- pg_get_function_arguments(func_oid)

  描述：获取函数定义的参数列表（带默认值）

  返回类型：text

  备注：pg_get_function_arguments返回一个函数的参数列表，需要在CREATE FUNCTION中使用这种格式。

- pg_get_function_identity_arguments(func_oid)

  描述：获取参数列表来确定一个函数 (不带默认值)

  返回类型：text

  备注：pg_get_function_identity_arguments返回需要的参数列表用来标识函数，这种形式需要在ALTER FUNCTION中使用，并且这种形式省略了默认值。

- pg_get_function_result(func_oid)

  描述：获取函数的RETURNS子句

  返回类型：text

  备注：pg_get_function_result为函数返回适当的RETURNS子句。

- pg_get_indexdef(index_oid)

  描述：获取索引的CREATE INDEX命令

  返回类型：text

  示例：


```
gaussdb=# SELECT * FROM pg_get_indexdef(16416);
                         pg_get_indexdef
-------------------------------------------------------------------------
 CREATE INDEX test3_b_idx ON test3 USING btree (b) TABLESPACE pg_default
(1 row)
```

- pg_get_indexdef(index_oid, dump_schema_only)

  描述：获取索引的CREATE INDEX命令，仅用于dump场景。对于包含local索引的间隔分区表，当dump_schema_only为true时，返回的创建索引语句中不包含自动创建的分区的local索引信息；当dump_schema_only为false时，返回的创建索引语句中包含自动创建的分区的local索引信息。对于非间隔分区表或者不包含local索引的间隔分区分区表，dump_schema_only参数取值不影响函数返回结果。

  返回类型：text

  示例：

```
gaussdb=# CREATE TABLE sales
gaussdb-# (prod_id NUMBER(6),
gaussdb(#  cust_id NUMBER,
gaussdb(#  time_id DATE,
gaussdb(#  channel_id CHAR(1),
gaussdb(#  promo_id NUMBER(6),
gaussdb(#  quantity_sold NUMBER(3),
gaussdb(#  amount_sold NUMBER(10,2)
gaussdb(# )
PARTITION BY RANGE( time_id) INTERVAL('1 day')
gaussdb-# (
gaussdb(#  partition p1 VALUES LESS THAN ('2019-02-01 00:00:00'),
gaussdb(#  partition p2 VALUES LESS THAN ('2019-02-02 00:00:00')
gaussdb(# );
CREATE TABLE
gaussdb=# create index index_sales on sales(prod_id) local (PARTITION idx_p1 ,PARTITION idx_p2);
CREATE INDEX
gaussdb=# -- 插入数据没有匹配的分区，新创建一个分区，并将数据插入该分区
gaussdb=# INSERT INTO sales VALUES(1, 12, '2019-02-05 00:00:00', 'a', 1, 1, 1);
INSERT 0 1
gaussdb=# SELECT oid FROM pg_class WHERE relname = 'index_sales';
  oid
-------
 24632
(1 row)
gaussdb=# SELECT * FROM pg_get_indexdef(24632, true);
                                            pg_get_indexdef
-------------------------------------------------------------------------------------------------------------------------------
---
 CREATE INDEX index_sales ON sales USING btree (prod_id) LOCAL(PARTITION idx_p1, PARTITION
idx_p2)  TABLESPACE pg_default
(1 row)
gaussdb=# SELECT * FROM pg_get_indexdef(24632, false);
                                                  pg_get_indexdef

-------------------------------------------------------------------------------------------------------------------------------
-------------
--------------------
 CREATE INDEX index_sales ON sales USING btree (prod_id) LOCAL(PARTITION idx_p1, PARTITION
idx_p2, PARTITION sys_p1_prod_id_idx)  TA
BLESPACE pg_default
(1 row
```

- pg_get_indexdef(index_oid, column_no, pretty_bool)

  描述：获取索引的CREATE INDEX命令，或者如果column_no不为零，则只获取一个索引字段的定义。

  示例：

```
gaussdb=# SELECT * FROM pg_get_indexdef(16416, 0, false);
                         pg_get_indexdef
-------------------------------------------------------------------------
```



```
 CREATE INDEX test3_b_idx ON test3 USING btree (b) TABLESPACE pg_default
(1 row)
gaussdb=# SELECT * FROM pg_get_indexdef(16416, 1, false);
 pg_get_indexdef
-----------------
 b
(1 row)
```

返回类型：text

备注：pg_get_functiondef为函数返回一个完整的CREATE OR REPLACE FUNCTION语句。

- pg_get_keywords()

  描述：获取SQL关键字和类别列表。

  返回类型：setof record

  备注：pg_get_keywords返回一组关于描述服务器识别SQL关键字的记录。word列包含关键字。catcode列包含一个分类代码：U表示通用的，C表示列名，T表示类型或函数名，或R表示保留。catdesc列包含了一个可能本地化描述分类的字符串。

- pg_get_userbyid(role_oid)

  描述：获取给定OID的角色名。

  返回类型：name

  备注：pg_get_userbyid通过角色的OID抽取对应的用户名。

- pg_check_authid(role_id)

  描述：通过role_id检查用户是否存在。

  返回类型：text

  示例：

  ```
  gaussdb=# SELECT pg_check_authid(20);
  pg_check_authid
  -----------------
  f
  (1 row)
  ```

- pg_get_viewdef(view_name)

  描述：为视图获取底层的SELECT命令。

  返回类型：text

- pg_get_viewdef(view_name, pretty_bool)

  描述：为视图获取底层的SELECT命令，如果pretty_bool为true，行字段可以包含80列。

  返回类型：text

  备注：pg_get_viewdef重构出定义视图的SELECT查询。这些函数大多数都有两种形式，其中带有pretty_bool参数，且参数为true时，是"适合打印"的结果，这种格式更容易读。另一种是缺省的格式，更有可能被将来的不同版本用同样的方法解释。如果是用于转储，那么尽可能避免使用适合打印的格式。给pretty-print参数传递false生成的结果和没有这个参数的变种生成的结果完全一样。

- pg_get_viewdef(view_oid)

  描述：为视图获取底层的SELECT命令。

  返回类型：text

- pg_get_viewdef(view_oid, pretty_bool)

  描述：为视图获取底层的SELECT命令，如果pretty_bool为true，行字段可以包含80列。

返回类型：text

- pg_get_viewdef(view_oid, wrap_column_int)

  描述：为视图获取底层的SELECT命令；行字段被换到指定的列数，打印是隐含的。

  返回类型：text

- pg_get_tabledef(table_oid)

  描述：根据table_oid获取表定义。

  示例：

```
gaussdb=# SELECT * FROM pg_get_tabledef(16384);
                  pg_get_tabledef
-----------------------------------------------------
 SET search_path = public;                          +
 CREATE  TABLE t1 (                                 +
        c1 bigint DEFAULT nextval('serial'::regclass)+
 )                                                  +
 WITH (orientation=row, compression=no)             +
 TO GROUP group1;
(1 row)
```

  返回类型：text

- pg_get_tabledef(table_name)

  描述：根据table_name获取表定义。

  示例：

```
gaussdb=# SELECT * FROM pg_get_tabledef('t1');
                  pg_get_tabledef
-----------------------------------------------------
 SET search_path = public;                          +
 CREATE  TABLE t1 (                                 +
        c1 bigint DEFAULT nextval('serial'::regclass)+
 )                                                  +
 WITH (orientation=row, compression=no)             +
 TO GROUP group1;
(1 row)
```

  返回类型：text

  备注：pg_get_tabledef重构出表定义的CREATE语句，包含了表定义本身、索引信息、comments信息。对于表对象依赖的group、schema、tablespace、server等信息，需要用户自己去创建，表定义里不会有这些对象的创建语句。

- pg_options_to_table(reloptions)

  描述：获取存储选项名称/值对的集合。

  返回类型：setof record

  备注：pg_options_to_table当通过pg_class.reloptions或pg_attribute.attoptions时返回存储选项名称/值对（option_name/option_value）的集合。

- pg_tablespace_databases(tablespace_oid)

  描述：获取在指定的表空间中有对象的数据库OID集合。

  返回类型：setof oid

  备注：pg_tablespace_databases允许检查表空间的状况，返回在该表空间中保存了对象的数据库OID集合。如果这个函数返回数据行，则该表空间就是非空的，因此不能删除。要显示该表空间中的特定对象，用户需要连接pg_tablespace_databases标识的数据库与查询pg_class系统表。

- pg_tablespace_location(tablespace_oid)

  描述：获取表空间所在的文件系统的路径。

  返回类型：text

- pg_typeof(any)

  描述：获取任何值的数据类型。

  返回类型：regtype

  备注：pg_typeof返回传递给他的值的数据类型OID。这可能有助于故障排除或动态构造SQL查询。声明此函数返回regtype，这是一个OID别名类型（请参考**对象标识符类型**）；这意味着它是一个为了比较而显示类型名称的OID。

  示例：

  ```
  gaussdb=# SELECT pg_typeof(33);
   pg_typeof
  -----------
   integer
  (1 row)

  gaussdb=# SELECT typlen FROM pg_type WHERE oid = pg_typeof(33);
   typlen
  --------
       4
  (1 row)
  ```

- collation for (any)

  描述：获取参数的排序。

  返回类型：text

  备注：表达式collation for返回传递给他的值的排序。

  示例：

  ```
  gaussdb=# SELECT collation for (description) FROM pg_description LIMIT 1;
   pg_collation_for
  ------------------
   "default"
  (1 row)
  ```

  值可能是引号括起来的并且模式限制的。如果没有为参数表达式排序，则返回一个null值。如果参数不是排序的类型，则抛出一个错误。

- pg_extension_update_paths(name)

  描述：返回指定扩展的版本更新路径。

  返回类型：text(source text), text(path text), text(target text)

- pg_get_serial_sequence(tablename, colname)

  描述：获取对应表名和列名上的序列。

  返回类型：text

  示例：

  ```
  gaussdb=# SELECT * FROM pg_get_serial_sequence('t1', 'c1');
   pg_get_serial_sequence
  ------------------------
   public.serial
  (1 row)
  ```

- pg_sequence_parameters(sequence_oid)

  描述：获取指定sequence的参数，包含起始值，最小值和最大值，递增值等。

  返回类型：int16, int16, int16, int16, Boolean

  示例：

  ```
  gaussdb=# SELECT * FROM pg_sequence_parameters(16420);
   start_value | minimum_value |     maximum_value     | increment | cycle_option
  -------------+---------------+-----------------------+-----------+--------------
           101 |             1 | 9223372036854775807 |         1 | f
  (1 row)
  ```


- gs_get_sequence_parameters(sequence_oid)

  描述：获取指定sequence的参数，包含起始值，最小值和最大值，递增值等。该函数针对无权限的sequence返回null值。

  返回类型：int16, int16, int16, int16, Boolean

  示例：

```
gaussdb=# SELECT * FROM gs_get_sequence_parameters(16420);
 start_value | minimum_value |     maximum_value     | increment | cycle_option
-------------+---------------+-----------------------+-----------+--------------
         101 |             1 | 9223372036854775807 |         1 | f
(1 row)
```

注释信息函数

- col_description(table_oid, column_number)

  描述：获取一个表字段的注释。

  返回类型：text

  备注：col_description返回一个表中字段的注释，通过表OID和字段号来声明。

- obj_description(object_oid, catalog_name)

  描述：获取一个数据库对象的注释。

  返回类型：text

  备注：带有两个参数的obj_description返回一个数据库对象的注释，该对象是通过其OID和其所属的系统表名称声明。比如，obj_description(123456,'pg_class')将返回OID为123456的表的注释。只带一个参数的obj_description只要求对象OID。

  obj_description不能用于表字段，因为字段没有自己的OID。

- obj_description(object_oid)

  描述：获取一个数据库对象的注释。

  返回类型：text

- shobj_description(object_oid, catalog_name)

  描述：获取一个共享数据库对象的注释。

  返回类型：text

  备注：shobj_description和obj_description差不多，不同之处仅在于前者用于共享对象。一些系统表是通用于GaussDB中所有数据库的全局表，因此这些表的注释也是全局存储的。

事务 ID 和快照

内部事务ID类型（xid）是64位。这些函数使用的数据类型txid_snapshot，存储在特定时刻事务ID可见性的信息。其组件描述在**表7-53**。

表 7-53 快照组件

名称	描述
xmin	最早的事务ID（txid）仍然活动。所有较早事务将是已经提交可见的，或者是直接回滚。
xmax	作为尚未分配的txid。所有大于或等于此txids的都是尚未开始的快照时间，因此不可见。

名称	描述
xip_list	当前快照中活动的txids。这个列表只包含在xmin和xmax之间活动的txids；有可能活动的txids高于xmax。介于大于等于xmin、小于xmax，并且不在这个列表中的txid，在这个时间快照已经完成的，因此按照提交状态查看他是可见还是回滚。这个列表不包含子事务的txids。

txid_snapshot的文本表示为：xmin:xmax:xip_list。

示例：10:20:10,14,15意思为：xmin=10, xmax=20, xip_list=10, 14, 15。

以下的函数在一个输出形式中提供服务器事务信息。这些函数的主要用途是为了确定在两个快照之间有哪个事务提交。

- txid_current()
  描述：获取当前事务ID。
  返回类型：bigint
- gs_txid_oldestxmin()
  描述：获取当前最小事务id的值oldesxmin。
  返回类型：bigint
- txid_current_snapshot()
  描述：获取当前快照。
  返回类型：txid_snapshot
- txid_snapshot_xip(txid_snapshot)
  描述：在快照中获取正在进行的事务ID。
  返回类型：setof bigint
- txid_snapshot_xmax(txid_snapshot)
  描述：获取快照的xmax。
  返回类型：bigint
- txid_snapshot_xmin(txid_snapshot)
  描述：获取快照的xmin。
  返回类型：bigint
- txid_visible_in_snapshot(bigint, txid_snapshot)
  描述：在快照中事务ID是否可见（不使用子事务ID）。
  返回类型：Boolean
- get_local_prepared_xact()
  描述：获取当前节点两阶段残留事务信息，包括事务id，两阶段gid名称，prepared的时间，owner的oid，database的oid及当前节点的node_name。
  返回类型：xid, text, timestamptz, oid, oid，text
- get_remote_prepared_xacts()
  描述：获取所有远程节点两阶段残留事务信息，包括事务id，两阶段gid名称，prepared的时间，owner的名称，database的名称及node_name。
  返回类型：xid, text, timestamptz, name, name，text

- global_clean_prepared_xacts(text, text)

  描述：并发清理两阶段残留事务，仅GaussDB分布式场景下gs_clean工具可以调用清理，其他用户调用均返回false。

  返回类型：Boolean

- gs_get_next_xid_csn()

  描述：返回全局所有节点上的next_xid和next_csn值。

  返回值如下：

  表 7-54 gs_get_next_xid_csn 返回参数说明

字段名	描述
nodename	节点名称。
next_xid	当前节点下一个事务id号。
next_csn	当前节点下一个csn号。

- pg_control_system()

  描述：返回系统控制文件状态。

  返回类型：SETOF record

- pg_control_checkpoint()

  描述：返回系统检查点状态。

  返回类型：SETOF record

- pv_builtin_functions

  描述：查看所有内置系统函数信息。

  参数：nan

  返回值类型：proname name, pronamespace oid, proowner oid, prolang oid, procost real, prorows real, provariadic oid, protransform regproc, proisagg boolean, proiswindow boolean, prosecdef boolean, proleakproof boolean, proisstrict boolean, proretset boolean, provolatile "char", pronargs smallint, pronargdefaults smallint, prorettype oid, proargtypes oidvector, proallargtypes integer[], proargmodes "char"[], proargnames text[], proargdefaults pg_node_tree, prosrc text, probin text, proconfig text[], proacl aclitem[], prodefaultargpos int2vector, fencedmode boolean, proshippable boolean, propackage boolean, oid oid

- pv_thread_memory_detail

  描述：返回各线程的内存信息。

  参数：nan

  返回值类型：threadid text, tid bigint, thrdtype text, contextname text, level smallint, parent text, totalsize bigint, freesize bigint, usedsize bigint

- pg_relation_compression_ratio

  描述：查询表压缩率，默认返回1.0。

  参数：text

  返回值类型：real

- pg_relation_with_compression
  描述：查询表是否压缩。
  参数：text
  返回值类型：boolean
- pg_stat_file_recursive
  描述：列出路径下所有文件。
  参数：location text
- pg_shared_memory_detail
  描述：返回所有已产生的共享内存上下文的使用信息，各列描述请参考**GS_SHARED_MEMORY_DETAIL**。
  参数：nan
  返回值类型：contextname text, level smallint, parent text, totalsize bigint, freesize bigint, usedsize bigint
- get_gtm_lite_status
  描述：返回GTM上的backupXid和csn号，用来支持问题定位，GTM-FREE模式下不支持使用本系统函数，集中式不支持该函数。
- gs_stat_get_wlm_plan_operator_info
  描述：从内部哈希表中获取算子计划信息。
  参数：oid
  返回值类型：datname text, queryid int8, plan_node_id int4, startup_time int8, total_time int8, actual_rows int8, max_peak_memory int4, query_dop int4, parent_node_id int4, left_child_id int4, right_child_id int4, operation text, orientation text, strategy text, options text, condition text, projection text
- pg_stat_get_partition_tuples_hot_updated
  描述：返回给定分区id的分区热更新元组数的统计。
  参数：oid
  返回值类型：bigint
- gs_session_memory_detail_tp
  描述：返回会话的内存使用情况，参考gs_session_memory_detail。
  参数：nan
  返回值类型：sessid text, sesstype text, contextname text, level smallint, parent text, totalsize bigint, freesize bigint, usedsize bigint
- gs_thread_memory_detail
  描述：返回各线程的内存信息。
  参数：nan
  返回值类型：threadid text, tid bigint, thrdtype text, contextname text, level smallint, parent text, totalsize bigint, freesize bigint, usedsize bigint
- pg_stat_get_wlm_realtime_operator_info
  描述：从内部哈希表中获取实时执行计划算子信息。
  参数：nan
  返回值类型：queryid bigint, pid bigint, plan_node_id integer, plan_node_name text, start_time timestamp with time zone, duration bigint, status text, query_dop integer, estimated_rows bigint, tuple_processed bigint,

min_peak_memory integer, max_peak_memory integer, average_peak_memory integer, memory_skew_percent integer, min_spill_size integer, max_spill_size integer, average_spill_size integer, spill_skew_percent integer, min_cpu_time bigint, max_cpu_time bigint, total_cpu_time bigint, cpu_skew_percent integer, warning text

- pg_stat_get_wlm_operator_info

  描述：从内部哈希表中获取执行计划算子信息。

  参数：nan

  返回值类型：queryid bigint, pid bigint, plan_node_id integer, plan_node_name text, start_time timestamp with time zone, duration bigint, query_dop integer, estimated_rows bigint, tuple_processed bigint, min_peak_memory integer, max_peak_memory integer, average_peak_memory integer, memory_skew_percent integer, min_spill_size integer, max_spill_size integer, average_spill_size integer, spill_skew_percent integer, min_cpu_time bigint, max_cpu_time bigint, total_cpu_time bigint, cpu_skew_percent integer, warning text

- pg_stat_get_wlm_node_resource_info

  该函数当前版本暂不可用。

- pg_stat_get_session_wlmstat

  描述：返回当前会话负载信息。

  参数：pid integer

  返回值类型：datid oid, threadid bigint, sessionid bigint, threadpid integer, usesysid oid, appname text, query text, priority bigint, block_time bigint, elapsed_time bigint, total_cpu_time bigint, skew_percent integer, statement_mem integer, active_points integer, dop_value integer, current_cgroup text, current_status text, enqueue_state text, attribute text, is_plana boolean, node_group text, srespool name

- pg_stat_get_wlm_instance_info

  描述：返回当前实例负载信息。

  参数：nan

  返回值类型：instancename text, timestamp timestamp with time zone, used_cpu integer, free_memory integer, used_memory integer, io_await double precision, io_util double precision, disk_read double precision, disk_write double precision, process_read bigint, process_write bigint, logical_read bigint, logical_write bigint, read_counts bigint, write_counts bigint

- pg_stat_get_wlm_instance_info_with_cleanup

  描述：返回当前实例负载信息，并且保存到系统表中。

  参数：nan

  返回值类型：instancename text, timestamp timestamp with time zone, used_cpu integer, free_memory integer, used_memory integer, io_await double precision, io_util double precision, disk_read double precision, disk_write double precision, process_read bigint, process_write bigint, logical_read bigint, logical_write bigint, read_counts bigint, write_counts bigint

- pg_stat_get_wlm_realtime_session_info

  描述：返回实时会话负载信息。

  参数：nan

返回值类型：nodename text, threadid bigint, block_time bigint, duration bigint, estimate_total_time bigint, estimate_left_time bigint, schemaname text, query_band text, spill_info text, control_group text, estimate_memory integer, min_peak_memory integer, max_peak_memory integer, average_peak_memory integer, memory_skew_percent integer, min_spill_size integer, max_spill_size integer, average_spill_size integer, spill_skew_percent integer, min_dn_time bigint, max_dn_time bigint, average_dn_time bigint, dntime_skew_percent integer, min_cpu_time bigint, max_cpu_time bigint, total_cpu_time bigint, cpu_skew_percent integer, min_peak_iops integer, max_peak_iops integer, average_peak_iops integer, iops_skew_percent integer, warning text, query text, query_plan text, cpu_top1_node_name text, cpu_top2_node_name text, cpu_top3_node_name text, cpu_top4_node_name text, cpu_top5_node_name text, mem_top1_node_name text, mem_top2_node_name text, mem_top3_node_name text, mem_top4_node_name text, mem_top5_node_name text, cpu_top1_value bigint, cpu_top2_value bigint, cpu_top3_value bigint, cpu_top4_value bigint, cpu_top5_value bigint, mem_top1_value bigint, mem_top2_value bigint, mem_top3_value bigint, mem_top4_value bigint, mem_top5_value bigint, top_mem_dn text, top_cpu_dn text

- pg_stat_get_wlm_session_iostat_info

  描述：返回会话负载IO信息。

  参数：nan

  返回值类型：threadid bigint, maxcurr_iops integer, mincurr_iops integer, maxpeak_iops integer, minpeak_iops integer, iops_limits integer, io_priority integer, curr_io_limits integer

- pg_stat_get_wlm_statistics

  描述：返回会话负载统计数据。

  参数：nan

  返回值类型：statement text, block_time bigint, elapsed_time bigint, total_cpu_time bigint, qualification_time bigint, skew_percent integer, control_group text, status text, action text

- adm_hist_snapshot_func()

  描述：返回快照执行时间相关信息，访问该函数需要打开enable_wdr_snapshot参数，并且需要snapshot schema， snapshot表和tables_snap_timestamp表的访问权限。

  参数：nan

  返回值类型：snap_id bigint, dbid oid, begin_interval_time timestamp(3), end_interval_time timestamp(3), flush_elapsed interval day(5) to second(1), begin_interval_time_tz timestamp(3) with time zone, end_interval_time_tz timestamp(3) with time zone

- gs_get_current_version()

  描述：依据当前编译宏返回当前编译模式，返回'P'。

  参数：nan

  返回值类型：char

- gs_get_kernel_info()

  描述：DN节点上的事务相关的信息。

  返回值如下。

表 7-55 gs_get_kernel_info 返回参数说明

名称	类型	描述
node_name	text	节点名。
module	text	模块名。包括： ● XACT（事务模块）。 ● STANDBY（备机模块）。 ● UNDO（undo模块）。 ● HOTPATH（热补丁模块）。
name	text	探查内存态关键数据名字。包括： ● startup_max_xid（进程启动时最大的xid值）。 ● recent_local_xmin（本地活跃事务最小xid值）。 ● recent_global_xmin（全局活跃事务最小xid值）。 ● standby_xmin（备机活跃事务最小xid值）。 ● standby_redo_cleanup_xmin（备机redo时cleanup日志最小xid值）。 ● standby_redo_cleanup_xmin_lsn（备机redo时cleanup日志最小xid的LSN值）。 ● local_csn_min（本地活跃事务最小CSN值）。 ● replication_slot_xmin（复制槽的最小xid值）。 ● replication_slot_catalog_xmin（catalog复制槽最小xid值）。 ● global_recycle_xid（全局undo回收事务的最小xid值）。 ● global_frozen_xid（全局冻结事务的最小的xid值）。 ● hotpatch_additional_info（热补丁预留字段）
value	text	探查内存态关键数据值。

示例：

```
gaussdb=# select * from gs_get_kernel_info();
  node_name   | module |             name              |  value
--------------+--------+-------------------------------+---------
 datanode1    | XACT   | startup_max_xid               | 16488
 datanode1    | XACT   | recent_local_xmin             | 15805
 datanode1    | XACT   | recent_global_xmin            | 15805
 datanode1    | STANDBY | standby_xmin                 | 0
```

```
 datanode1    | STANDBY | standby_redo_cleanup_xmin     | 0
 datanode1    | STANDBY | standby_redo_cleanup_xmin_lsn | 0/0
 datanode1    | XACT    | local_csn_min                 | 6014225
 datanode1    | XACT    | replication_slot_xmin         | 0
 datanode1    | XACT    | replication_slot_catalog_xmin | 0
 datanode1    | UNDO    | global_recycle_xid            | 15805
 datanode1    | XACT    | global_frozen_xid             | 0
 datanode1    | HOTPATH | hotpatch_additional_info      |
(12 row)
```

7.5.25 系统管理函数

7.5.25.1 配置设置函数

配置设置函数是可以用于查询以及修改运行时配置参数的函数。

- current_setting(setting_name)

  描述：当前的设置值。

  返回值类型：text

  备注：current_setting用于以查询形式获取setting_name的当前值。和SQL语句SHOW是等效的。比如：

  ```
  gaussdb=# SELECT current_setting('datestyle');

   current_setting
  -----------------
   ISO, MDY
  (1 row)
  ```

- set_working_grand_version_num_manually(tmp_version)

  描述：通过切换授权版本号来更新和升级数据库的新特性。

  返回值类型：void

- shell_in(type)

  描述： 为shell类型输入路由（那些尚未填充的类型）。

  返回值类型：void

- shell_out(type)

  描述：为shell 类型输出路由（那些尚未填充的类型）。

  返回值类型：void

- set_config(setting_name, new_value, is_local)

  描述：设置参数并返回新值。

  返回值类型：text

  备注：set_config将参数setting_name设置为new_value。如果is_local为true，则new_value将只应用于当前事务。如果希望new_value应用于当前会话，可以使用false，和SQL语句SET是等效的。例如：

  ```
  gaussdb=# SELECT set_config('log_statement_stats', 'off', false);

   set_config
  ------------
   off
  (1 row)
  ```

7.5.25.2 通用文件访问函数

通用文件访问函数提供了对数据库服务器上的文件的本地访问接口。只有数据库目录和log_directory目录里面的文件可以访问。使用相对路径访问数据库目录里面的文

件，以及匹配log_directory配置而设置的路径访问日志文件。只有数据库初始化用户才能使用这些函数。

- pg_ls_dir(dirname text)

  描述：列出目录中的文件。

  返回值类型：setof text

  备注：pg_ls_dir返回指定目录里面的除了特殊项“.”和“..”之外所有名称。

  示例：

```
gaussdb=# SELECT pg_ls_dir('./');
      pg_ls_dir
----------------------
 .postgresql.conf.swp
 postgresql.conf
 pg_tblspc
 PG_VERSION
 pg_ident.conf
 core
 server.crt
 pg_serial
 pg_twophase
 postgresql.conf.lock
 pg_stat_tmp
 pg_notify
 pg_subtrans
 pg_ctl.lock
 pg_xlog
 pg_clog
 base
 pg_snapshots
 postmaster.opts
 postmaster.pid
 server.key.rand
 server.key.cipher
 pg_multixact
 pg_errorinfo
 server.key
 pg_hba.conf
 pg_replslot
 .pg_hba.conf.swp
 cacert.pem
 pg_hba.conf.lock
 global
 gaussdb.state
(32 rows)
```

- pg_read_file(filename text, offset bigint, length bigint)

  描述：返回一个文本文件的内容。

  返回值类型：text

  备注：pg_read_file返回一个文本文件的一部分，从offset开始，最多返回length字节（如果先达到文件结尾，则小于这个数值）。如果offset是负数，则它是相对于文件结尾回退的长度。如果省略了offset和length，则返回整个文件。

  示例：

```
gaussdb=# SELECT pg_read_file('postmaster.pid',0,100);
             pg_read_file
-----------------------------------------
 53078                                  +
 /srv/BigData/hadoop/data1/dbnode+
 1500022474                             +
 8000                                  +
 /var/run/FusionInsight                 +
 localhost                              +
  2
(1 row)
```


- pg_read_binary_file(filename text [, offset bigint, length bigint,missing_ok boolean])

  描述：返回一个二进制文件的内容。

  返回值类型：bytea

  备注：pg_read_binary_file的功能与pg_read_file类似，除了结果的返回值为bytea类型不一致，相应地不会执行编码检查。与convert_from函数结合，这个函数可以用来读取用指定编码的一个文件。

  ```
  gaussdb=# SELECT convert_from(pg_read_binary_file('filename'), 'UTF8');
  ```

- pg_stat_file(filename text)

  描述：返回一个文本文件的状态信息。

  返回值类型：record

  备注：pg_stat_file返回一条记录，其中包含：文件大小、最后访问时间戳、最后更改时间戳、最后文件状态修改时间戳以及标识传入参数是否为目录的Boolean值。典型的用法：

  ```
  gaussdb=# SELECT * FROM pg_stat_file('filename');
  gaussdb=# SELECT (pg_stat_file('filename')).modification;
  ```

  示例：

  ```
  gaussdb=# SELECT convert_from(pg_read_binary_file('postmaster.pid'), 'UTF8');
              convert_from
  ----------------------------------------
   4881                                 +
   /srv/BigData/gaussdb/data1/dbnode+
   1496308688                           +
   25108                                +
   /opt/user/Bigdata/gaussdb/gaussdb_tmp +
   *                                    +
    25108001  43352069                  +

  (1 row)
  gaussdb=# SELECT * FROM pg_stat_file('postmaster.pid');

   size |         access         |      modification      |         change
  | creation | isdir
  ------+------------------------+------------------------+------------------------
  +----------+-------
    117 | 2017-06-05 11:06:34+08 | 2017-06-01 17:18:08+08 | 2017-06-01 17:18:08+08
  |          | f
  (1 row)
  gaussdb=# SELECT (pg_stat_file('postmaster.pid')).modification;
       modification
  ------------------------
   2017-06-01 17:18:08+08
  (1 row)
  ```

7.5.25.3 服务器信号函数

服务器信号函数向其他服务器进程发送控制信号。只有系统管理员有权执行以下函数。

- pg_cancel_backend(pid int)

  描述：取消一个后端线程正在执行的语句。

  返回值类型：Boolean

  备注：pg_cancel_backend向由pid标识的后端进程发送一个查询取消（SIGINT）信号。一个活动的后端进程的PID可以从pg_stat_activity视图的pid字段找到，或者在服务器上用ps列出数据库进程。具有SYSADMIN权限的用户，后端进程所连

接的数据库的属主，后端进程的属主或者继承了内置角色gs_role_signal_backend权限的用户有权使用该函数。

- pg_cancel_session(pid bigint, sessionid bigint)

  描述：线程池模式下，取消一个活跃状态会话正在执行的语句。

  返回值类型：Boolean

  备注：pg_cancel_session的入参可以通过pg_stat_activity中的pid字段和sessionid的字段查询，可以用于清理线程池模式下，活跃状态会话正在执行的语句。当入参pid和sessionid相同，且均为线程id时，功能和pg_cancel_backend相同。

- pg_reload_conf()

  描述：导致所有服务器进程重新装载它们的配置文件。

  返回值类型：Boolean

  备注：pg_reload_conf给服务器发送一个SIGHUP信号，导致所有服务器进程重新装载配置文件。

- pg_rotate_logfile()

  描述：滚动服务器的日志文件。

  返回值类型：Boolean

  备注：pg_rotate_logfile给日志文件管理器发送信号，使之立即切换到一个新的输出文件。这个函数只有在redirect_stderr用于日志输出的时候才有用，否则不会产生日志文件管理器子进程。

- pg_terminate_backend(pid int)

  描述：终止一个后台线程。

  返回值类型：Boolean

  备注：如果成功，函数返回true，否则返回false。具有SYSADMIN权限的用户，后端线程所连接的数据库的属主，后端线程的属主或者继承了内置角色gs_role_signal_backend权限的用户有权使用该函数。若执行该函数后未成功终止目标会话，则会强制关闭该会话和客户端的socket连接。

  须知

  该函数可终止非线程池的线程、活跃状态的线程池线程，但无法终止非活跃状态的线程池线程。

  示例：

  ```
  gaussdb=# SELECT pid from pg_stat_activity;
        pid
  -----------------
   140657876268816
  (1 rows)

  gaussdb=# SELECT pg_terminate_backend(140657876268816);
   pg_terminate_backend
  ----------------------
   t
  (1 row)
  ```

- pg_terminate_session(pid int64, sessionid int64)

  描述：线程池模式下，终止一个后台session。

  返回值类型：Boolean

备注：如果成功，函数返回true，否则返回false。具有SYSADMIN权限的用户、会话所连接的数据库的属主、会话的属主、或者继承了内置角色gs_role_signal_backend权限的用户有权使用该函数。若执行该函数后未成功终止目标会话，则会强制关闭该会话和客户端的socket连接。

须知

当入参pid和sessionid相同，且均为线程id时，该函数可终止非线程池的线程、活跃状态的线程池线程。

当入参pid和sessionid不同时，该函数可终止活跃状态的会话，或关闭非活跃状态会话和客户端的socket连接。

- pg_terminate_active_session_socket(pid int64, sessionid int64)

  描述：关闭一个活跃session和客户端的socket连接。

  返回值类型：Boolean

  备注：如果成功，函数返回true，否则返回false。具有SYSADMIN权限的用户、后端线程所连接的数据库的属主、后端线程的属主或者继承了内置角色gs_role_signal_backend权限的用户有权使用该函数。

7.5.25.4 备份恢复控制函数

备份控制函数

备份控制函数可帮助进行在线备份。

- pg_create_restore_point(name text)

  描述：为执行恢复创建一个命名点。需要管理员角色权限。

  返回值类型：text

  备注：pg_create_restore_point创建了一个可以用作恢复目的、有命名的事务日志记录，并返回相应的事务日志位置。在恢复过程中，recovery_target_name可以通过这个名称定位对应的日志恢复点，并从此处开始执行恢复操作。避免使用相同的名称创建多个恢复点，因为恢复操作将在第一个匹配（恢复目标）的名称上停止。

- pg_current_xlog_location()

  描述：获取当前事务日志的写入位置。

  返回值类型：text

  备注：pg_current_xlog_location使用与前面那些函数相同的格式显示当前事务日志的写入位置。如果是只读操作，不需要系统管理员权限。

- pg_current_xlog_insert_location()

  描述：获取当前事务日志的插入位置。

  返回值类型：text

  备注：pg_current_xlog_insert_location显示当前事务日志的插入位置。插入点是事务日志在某个瞬间的“逻辑终点”，而实际的写入位置则是从服务器内部缓冲区写出时的终点。写入位置是可以从服务器外部检测到的终点，如果要归档部分完成事务日志文件，则该操作即可实现。插入点主要用于服务器调试目的。如果是只读操作，不需要系统管理员权限。

- gs_current_xlog_insert_end_location()

  描述：获取当前事务日志的插入位置。

  返回值类型：text

  备注：gs_current_xlog_insert_end_location显示当前事务日志的实际插入位置。

- pg_start_backup(label text [, fast boolean])

  描述：开始执行在线备份。需要管理员角色、复制的角色或运维管理员角色打开operation_mode。

  返回值类型：text

  备注：pg_start_backup接受一个用户定义的备份标签（通常这是备份转储文件存放地点的名称）。这个函数向数据库的数据目录写入一个备份标签文件，然后以文本方式返回备份的事务日志起始位置。

  ```
  gaussdb=# SELECT pg_start_backup('label_goes_here');
   pg_start_backup
  -----------------
   0/3000020
  (1 row)
  ```

- pg_stop_backup()

  描述：完成执行在线备份。需要管理员角色、复制的角色或运维管理员角色打开operation_mode。

  返回值类型：text

  备注：pg_stop_backup删除pg_start_backup创建的标签文件，并且在事务日志归档区里创建一个备份历史文件。这个历史文件包含给予pg_start_backup的标签、备份的事务日志起始与终止位置、备份的起始和终止时间。返回值是备份的事务日志终止位置。计算出中止位置后，当前事务日志的插入点将自动前进到下一个事务日志文件，结束的事务日志文件可以被立即归档从而完成备份。

- pg_switch_xlog()

  描述：切换到一个新的事务日志文件。需要管理员角色或运维管理员角色打开operation_mode。

  返回值类型：text

  备注：pg_switch_xlog移动到下一个事务日志文件，以允许将当前日志文件归档（假定使用连续归档）。返回值是刚完成的事务日志文件的事务日志结束位置+1。如果从最后一次事务日志切换以来没有活动的事务日志，则pg_switch_xlog不进行移动操作，直接返回当前事务日志文件的开始位置。

- pg_xlogfile_name(location text)

  描述：将事务日志的位置字符串转换为文件名。

  返回值类型：text

  备注：pg_xlogfile_name仅抽取事务日志文件名称。如果给定的事务日志位置恰好位于事务日志文件的交界上，这两个函数都返回前一个事务日志文件的名称。这对于管理事务日志归档来说是非常有利的，因为前一个文件是当前最后一个需要归档的文件。

- pg_xlogfile_name_offset(location text)

  描述：将事务日志的位置字符串转换为文件名并返回在文件中的字节偏移量。

  返回值类型：text,integer

  备注：可以使用pg_xlogfile_name_offset从前述函数的返回结果中抽取相应的事务日志文件名称和字节偏移量。例如：

  ```
  gaussdb=# SELECT * FROM pg_xlogfile_name_offset(pg_stop_backup());
  NOTICE:  pg_stop_backup cleanup done, waiting for required WAL segments to be archived
  ```



```
NOTICE:  pg_stop_backup complete, all required WAL segments have been archived
        file_name         | file_offset
--------------------------+-------------
000000010000000000000003 |         272
(1 row)
```

- pg_xlog_location_diff(location text, location text)

  描述：计算两个事务日志位置之间在字节上的区别。

  返回值类型：numeric

- pg_cbm_start_tracked_location()

  描述：用于查询cbm解析的起始lsn位置。

  返回值类型：text

- pg_cbm_tracked_location()

  描述：用于查询cbm解析到的lsn位置。

  返回值类型：text

- pg_cbm_get_merged_file(startLSNArg text, endLSNArg text)

  描述：用于将指定lsn范围之内的cbm文件合并成一个cbm文件，并返回合并完的cbm文件名。

  返回值类型：text

  备注：必须是系统管理员或运维管理员才能获取cbm合并文件。

- pg_cbm_get_changed_block(startLSNArg text, endLSNArg text)

  描述：用于将指定lsn范围之内的cbm文件合并成一个表，并返回表的各行记录。

  返回值类型：records

  备注：pg_cbm_get_changed_block返回的表字段包含：合并起始的lsn、合并截止的lsn、表空间oid、库oid、表的relfilenode、表的fork number、表是否为系统表、表是否被删除、表是否被创建、表是否被截断、表被截断后的页面数、有多少页被修改以及被修改的页号的列表。

- pg_cbm_recycle_file(targetLSNArg text)

  描述：删除不再使用的cbm文件，并返回删除后的第一条lsn。

  返回值类型：text

- pg_cbm_force_track(targetLSNArg text,timeOut int)

  描述：强制执行一次cbm追踪到指定的xlog位置，并返回实际追踪结束点的xlog位置。

  返回值类型：text

- pg_enable_delay_ddl_recycle()

  描述：开启延迟DDL功能，并返回开启点的xlog位置。需要管理员角色或运维管理员角色打开operation_mode。

  返回值类型：text

- pg_disable_delay_ddl_recycle(barrierLSNArg text, isForce bool)

  描述：关闭延迟DDL功能，并返回本次延迟DDL生效的xlog范围。需要管理员角色或运维管理员角色打开operation_mode。

  返回值类型：records

- pg_enable_delay_xlog_recycle()

  描述：开启延迟xlog回收功能，数据库主节点修复使用。需要管理员角色或运维管理员角色打开operation_mode。

返回值类型：void

- pg_disable_delay_xlog_recycle()

  描述：关闭延迟xlog回收功能，数据库主节点修复使用。需要管理员角色或运维管理员角色打开operation_mode。

  返回值类型：void

- pg_cbm_rotate_file(rotate_lsn text)

  描述：等待cbm解析到rotate_lsn之后，强制切换文件，在build期间调用。

  返回值类型：void。

- gs_roach_stop_backup(backupid text)

  描述：停止一个内部备份工具GaussRoach开启的备份。与pg_stop_backup系统函数类似，但更轻量。

  返回值类型：text，内容为当前日志的插入位置。

- gs_roach_enable_delay_ddl_recycle(backupid name)

  描述：开启延迟DDL功能，并返回开启点的日志位置。与pg_enable_delay_ddl_recycle系统函数类似，但更轻量。并且，通过传入不同的backupid，可以支持并发打开延迟DDL。

  返回值类型：text，内容为返回开启点的日志位置。

- gs_roach_disable_delay_ddl_recycle(backupid text)

  描述：关闭延迟DDL功能，并返回本次延迟DDL生效的日志范围。与pg_enable_delay_ddl_recycle系统函数类似，但更轻量。并且，通过传入不同的backupid，可以支持并发关闭延迟DDL功能。

  返回值类型：records，内容为本次延迟DDL生效的日志范围。

- gs_roach_switch_xlog(request_ckpt bool)

  描述：切换当前使用的日志段文件，并且，如果request_ckpt为true，则触发一个全量检查点。

  返回值类型：text，内容为切段日志的位置。

- gs_block_dw_io(timeout int, identifier text)

  描述：阻塞双写页面刷盘。

  参数说明：

  - timeout

    阻塞时长。

    取值范围：[0, 3600]（秒），0为阻塞时长为0。

  - identifier

    此次操作的标识。

    取值范围：字符串，不支持除大小写字母、数字以及下划线(_)以外的字符。

  返回值类型：bool

  备注：调用该函数的用户需要具有SYSADMIN权限或具有OPRADMIN权限，运维管理员角色须打开operation_mode。

- gs_is_dw_io_blocked()

  描述：查看当前双写页面刷盘是否被阻塞，如果处于阻塞中则返回true。

  返回值类型：bool

  备注：调用该函数的用户需要具有SYSADMIN权限或具有OPRADMIN权限，运维管理员角色须打开operation_mode。

- gs_pitr_advance_last_updated_barrier()

  描述：在PITR模式下，强制推进上次上传到OBS/NAS介质中的全局最大已归档恢复点到当前点，无入参。

  返回值类型：text

  备注：调用该函数的用户需要具有SYSADMIN权限或具有OPRADMIN权限，运维管理员角色须打开operation_mode。同时只能在集中式的主DN上使用才有效。返回值为当前推进到的最新本地最大已归档恢复点。

- gs_pitr_clean_local_barrier_files('delete_timestamp')

  描述：清理本地缓存的barrier记录文件。

  参数范围：delete_timestamp参数类型为text，为linux时间戳，长度为10位。

  返回值类型：text

  备注：调用该函数的用户需要具有SYSADMIN权限或具有OPRADMIN权限，运维管理员角色须打开operation_mode。返回的结果是删除后本地最老barrier文件的开始时间戳。

- gs_get_barrier_lsn(barrier_name text)

  描述：获取备份创建的barrier对应的lsn。

  返回值类型：text

  备注：目前不支持该函数。当前入参仅支持gs_roach_full和gs_roach_inc。调用该函数的用户需要具有SYSADMIN权限或具有OPRADMIN权限，运维管理员角色须打开operation_mode。

- gs_gbr_relation_associated_filenode(schemaName name, tableName name)

  描述：返回与输入table相关的全部索引、sequence、分区、toast、toast index的relfilenode。

  返回值类型：records

  备注：gs_gbr_relation_associated_filenode返回的字段包含文件类型relkind、文件所在namespace、文件对应的relation name、文件所在的database oid、文件所在的tablespace oid以及文件的relfilenode。

- pg_create_physical_replication_slot_extern(slotname text, dummy_standby bool, extra_content text, need_recycle_xlog bool)

  须知

  slotname为归档槽名称，取值范围：字符串，仅支持小写字母、数字以及“_”，“?”，“-”，“.”字符，且不支持“.”或“..”单独作为复制槽名称。建议使用字母字符串作为归档槽名，且长度不能超过64。

  描述：创建OBS/NAS归档槽。slotname为归档槽/恢复槽的slotname ，主备必须使用同一个slotname。dummy_standby是预留参数。extra_content包含了归档槽的一些信息。对于OBS归档槽，其格式为"OBS;obs_server_ip;obs_bucket_name;obs_ak;obs_sk;archive_path;is_recovery;is_vote_replicate"，OBS表示归档槽的归档的介质，obs_server_ip为obs的ip，obs_bucket_name为obs的桶名，obs_ak为obs的ak，obs_sk为obs的sk，archive_path为归档的路径i，is_recovery标志是归档槽还是恢复槽，0表示是归档槽，1表示是恢复槽。is_vote_replicate标志是否是投票副本优先，0表示同步备机归档优先，1表示投票副本归档优先，当前版本该字段为预留字段，暂未适配。对

于NAS归档槽，其格式为"NAS;archive_path;is_recovery;is_vote_replicate"，相比OBS归档槽，缺少了OBS相关的配置信息，其余字段意义相同。

如果是不指定OBS或NAS介质的话，默认指定的是OBS归档槽，其extra_content格式为"obs_server_ip;obs_bucket_name;obs_ak;obs_sk;archive_path;is_recovery;is_vote_replicate"。

need_recycle_xlog标志创建归档槽时是否回收老的归档日志，true表示回收，false表示不回收。

返回值类型：records包含slotname和xlog_position。

备注：调用该函数的用户需要具有SYSADMIN权限或具有REPLICATION权限或继承了内置角色gs_role_replication的权限。目前不支持创建多归档槽。

例如：

创建OBS归档槽：

```
gaussdb=# select * from pg_create_physical_replication_slot_extern('uuid', false,
'OBS;obs_server_ip;obs_bucket_name;obs_ak;obs_sk;gaussdb_uuid/dn1;0;0', false);
 slotname | xlog_position
----------+---------------
 uuid     |
(1 row)
```

创建NAS归档槽：

```
gaussdb=# select * from pg_create_physical_replication_slot_extern('uuid', false, 'NAS;/data/nas/media/
gaussdb_uuid/dn1;0;0',false);
 slotname | xlog_position
----------+---------------
 uuid     |
```

- gs_set_obs_delete_location(delete_location text)

  描述：设置OBS归档日志可删除的位置。delete_location实际为Log Sequence Number（LSN），该位置之前的日志已经完成回放并且落盘，可以在OBS上进行删除。

  返回值类型：xlog_file_name text，表明此次可删除点所在的日志文件名。无论OBS删除是否成功，该值都会正常返回。

  ```
  gaussdb=# select gs_set_obs_delete_location('0/54000000');
   gs_set_obs_delete_location
  ----------------------------
   000000010000000000000054_00
  (1 row)
  ```

- gs_set_obs_delete_location_with_slotname(cstring, cstring)

  描述：设置指定归档槽OBS归档日志可删除的位置。第一个参数实际为Log Sequence Number（LSN），该位置之前的日志已经完成回放并且落盘，可以在OBS上进行删除，第二个参数为归档槽的名称。

  返回值类型：xlog_file_name text，表明此次可删除点所在的日志文件名。无论OBS删除是否成功，该值都会正常返回。

- gs_get_global_barrier_status()

  描述：gs_get_global_barrier_status用以查询主数据库实例已在obs完成归档的最新global barrier。

  返回值类型：text

  global_barrier_id：全局最新barrier ID。

  global_achive_barrier_id：全局最新归档barrier ID。

- gs_get_global_barriers_status()

描述：gs_get_global_barriers_status用以查询主数据库实例已在OBS完成归档的最新global barrier。

返回值类型：text

slot_name：槽位名。

global_barrier_id：全局最新barrier ID。

global_achive_barrier_id：全局最新归档barrier ID。

- gs_set_archive_standby_name_on_primary(text)

  描述：gs_set_archive_standby_name_on_primary用来指定归档备机，设置后将由指定的备机执行归档操作。输入为备机名称，如：dn_6002。

  返回值类型：bool

  true表示设置成功，false表示设置失败。

恢复控制函数

恢复信息函数提供了当前备机状态的信息。这些函数可能在恢复期间或正常运行中执行。

- pg_is_in_recovery()

  描述：如果恢复仍然在进行中则返回true。

  返回值类型：Boolean

- pg_last_xlog_receive_location()

  描述：获取最后接收事务日志的位置并通过流复制将其同步到磁盘。当流复制正在进行时，事务日志将持续递增。如果恢复已完成，则最后一次获取的WAL记录会被静态保持并在恢复过程中同步到磁盘。如果流复制不可用，或还没有开始，这个函数返回NULL。

  返回值类型：text

- pg_last_xlog_replay_location()

  描述：获取最后一个事务日志在恢复时重放的位置。如果恢复仍在进行，事务日志将持续递增。如果已经完成恢复，则将保持在恢复期间最后接收WAL记录的值。如果未进行恢复但服务器正常启动时，则这个函数返回NULL。

  返回值类型：text

- pg_last_xact_replay_timestamp()

  描述：获取最后一个事务在恢复时重放的时间戳。这是为在主节点上生成事务提交或终止WAL记录的时间。如果在恢复时没有事务重放，则这个函数返回NULL。如果恢复仍在进行，则事务日志将持续递增。如果恢复已经完成，则将保持在恢复期间最后接收WAL记录的值。如果服务器无需恢复就已正常启动，则这个函数返回NULL。

  返回值类型：timestamp with time zone

恢复控制函数控制恢复的进程。这些函数可能只在恢复时被执行。

- pg_is_xlog_replay_paused()

  描述：如果恢复暂停则返回true。

  返回值类型：Boolean

- pg_xlog_replay_pause()

  描述：立即暂停恢复。

返回值类型：void

- pg_xlog_replay_resume()

  描述：如果恢复处于暂停状态，则重新启动。

  返回值类型：void

- gs_get_active_archiving_standby()

  描述：查询同一分片内归档备机的信息。返回备机名，备机归档位置和已归档日志个数。

  返回值类型：text，int

- gs_pitr_get_warning_for_xlog_force_recycle()

  描述：查询开启归档后是否因归档槽不推进日志大量堆积导致日志被回收。

  返回值类型：bool

- gs_pitr_clean_history_global_barriers(stop_barrier_timestamp cstring)

  描述：清理指定时间之前所有barrier记录。返回最老的barrier记录。入参为cstring类型，linux时间戳。需要管理员角色或运维管理员角色执行。

  返回值类型：text

- gs_pitr_archive_slot_force_advance(stop_barrier_timestamp cstring)

  描述：强制推进归档槽，并清理不需要的barrier记录。返回新的归档槽位置。入参为cstring类型，linux时间戳。需要管理员角色或运维管理员角色执行。

  返回值类型：text

当恢复暂停时，没有发生数据库更改。如果是在热备里，所有新的查询将看到一致的数据库快照，并且不会有进一步的查询冲突产生，直到恢复继续。

如果不能使用流复制，则暂停状态将无限的延续。当流复制正在进行时，将连续接收WAL记录，最终将填满可用磁盘空间，这个进度取决于暂停的持续时间，WAL生成的速度和可用的磁盘空间。

- gs_recent_barrier_buffer_info(start_time text, end_time text)

  描述：根据用户输入的时间范围，进行相应的barrier信息查询，获取time_stamp、CSN、LSN和standard_time。

  返回值类型：records

  说明：调用该函数的用户需要具有SYSADMIN权限或具有OPRADMIN权限。输入参数start_time和end_time采用“年-月-日 时间”格式，其中时间采用clock格式。查询最大时间跨度为1天，超出跨度约束，根据查询起始时间将结束时间自动转换为极限边界进行查询。例如：

```
gaussdb=# SELECT * FROM gs_recent_barrier_buffer_info('2024-01-15 23:27:50', '2024-01-15
23:28:00');
 timestamp |       lsn        |   csn    |    standard_time
-----------+------------------+----------+---------------------
 1705332470 | 00000000/15FFBBA0 | 41020421 | 2024-01-15 23:27:50
 1705332471 | 00000000/15FFBDF0 | 41020422 | 2024-01-15 23:27:51
 1705332472 | 00000000/15FFC058 | 41020423 | 2024-01-15 23:27:52
 1705332472 | 00000000/15FFC0F8 | 41020424 | 2024-01-15 23:27:52
 1705332473 | 00000000/15FFC348 | 41020425 | 2024-01-15 23:27:53
 1705332474 | 00000000/15FFC598 | 41020426 | 2024-01-15 23:27:54
 1705332475 | 00000000/15FFC638 | 41020427 | 2024-01-15 23:27:55
 1705332476 | 00000000/15FFC888 | 41020428 | 2024-01-15 23:27:56
 1705332476 | 00000000/15FFDC80 | 41020433 | 2024-01-15 23:27:56
 1705332477 | 00000000/15FFDD20 | 41020434 | 2024-01-15 23:27:57
 1705332478 | 00000000/15FFDF70 | 41020435 | 2024-01-15 23:27:58
 1705332479 | 00000000/15FFE1D8 | 41020436 | 2024-01-15 23:27:59
 1705332480 | 00000000/15FFE278 | 41020437 | 2024-01-15 23:28:00
```



```
 1705332480 | 00000000/15FFE4C8 | 41020438 | 2024-01-15 23:28:00
(14 rows)
```

- gs_show_obs_media_files(slot_name cstring, src cstring, offset int32, limit int32)

  描述：根据用户输入的归档槽（slot_name）和OBS目录地址（src），查询OBS文件列表。

  返回值类型：records

  说明：调用该函数的用户需要具有SYSADMIN权限或具有OPRADMIN权限。Offset为查询结果偏移，limit为输出最大行数，查询src下所有文件。例如：

```
gaussdb=# SELECT gs_show_obs_archive_files('ssh','dn1/pg_xlog',0, 5);
                                gs_show_obs_archive_files
-----------------------------------------------------------------------------------------------
 (wstdist_ssh/archive/dn1/pg_xlog/
000000010000000000000007_00_01_00000004_00000002_00000000)
 (wstdist_ssh/archive/dn1/pg_xlog/
000000010000000000000007_00_01_00000103_00000003_00000000)
 (wstdist_ssh/archive/dn1/pg_xlog/
000000010000000000000007_01_01_00000004_00000002_00000000)
 (wstdist_ssh/archive/dn1/pg_xlog/
000000010000000000000007_01_01_00000103_00000003_00000000)
 (wstdist_ssh/archive/dn1/pg_xlog/
000000010000000000000007_02_01_00000004_00000002_00000000)
(5 rows)
```

- gs_upload_obs_media_file(slot_name cstring, src cstring, dest cstring, is_forced bool)

  描述：根据用户输入的归档槽（slot_name）、上传文件原地址（src）、OBS地址（dest）和是否强制上传（is_forced），上传OBS文件。

  返回值类型：void

  说明：调用该函数的用户需要具有SYSADMIN权限或具有OPRADMIN权限。原文件目录仅允许为$GAUSSLOG目录。例如：

```
gaussdb=# SELECT * FROM gs_upload_obs_archive_file('ssh', '/data/gauss/log/stwang/test/
000000010000000000000007_02_01_00000004_00000002_00000000', 'dn1/pg_xlog/
000000010000000000000019_02_01_00000000_00000000_00000003', true);
 gs_upload_obs_archive_file
----------------------------


(1 row)
```

- gs_download_obs_media_file(slot_name cstring, src cstring, dest cstring)

  描述：根据用户输入的归档槽（slot_name）、下载原地址（src）和本地目标地址（dest），下载OBS文件。

  返回值类型：void

  说明：调用该函数的用户需要具有SYSADMIN权限或具有OPRADMIN权限。下载目录仅允许为$GAUSSLOG目录。例如：

```
gaussdb=# SELECT * FROM gs_download_obs_archive_file('ssh','dn1/pg_xlog/
000000010000000000000019_02_01_00000000_00000000_00000003','/data/gauss/log/stwang/test');
 gs_download_obs_archive_file
------------------------------


(1 row)
```

7.5.25.5 双数据库实例容灾控制函数

- gs_streaming_dr_in_switchover()

  描述：基于流式复制的异地容灾解决方案中主数据库实例在执行计划内switchover过程中截断业务的接口。

返回值类型：Boolean，表明此次业务截断是否成功，是否可以正常进行switchover流程。

```
gaussdb=# SELECT * FROM gs_streaming_dr_in_switchover();
 is_in_switchover
------------------
 f
(1 row)
```

7.5.25.6 双数据库实例容灾查询函数

- gs_get_local_barrier_status()

  描述：两地三中心跨Region容灾特性开启后，主数据库实例和灾备数据库实例进行日志同步，通过barrier日志在主数据库实例的落盘，在灾备数据库实例的回放来确定主数据库实例归档日志进度与灾备数据库实例日志回放进度。gs_get_local_barrier_status用于查询灾备数据库实例每个节点当前的日志回放情况。

  返回值类型：text

  barrier_id：灾备数据库实例某节点当前回放到的最新barrier ID。

  barrier_lsn：灾备数据库实例某节点当前回放到的最新barrier ID的Log Sequence Number（LSN）。

  archive_lsn：灾备数据库实例某节点当前已获得归档日志的位置，该参数当前未生效。

  flush_lsn：灾备数据库实例某节点当前已完成刷盘日志位置。

- gs_hadr_has_barrier_creator()

  描述：两地三中心跨Region容灾特性开启后，查询当前节点是否存在barrier_creator线程，存在返回true（需要系统管理员角色）。

  返回值类型：Boolean

  备注：该函数只有在容灾数据库实例启动计划内switchover时使用。

  ```
  gaussdb=# SELECT * FROM gs_hadr_has_barrier_creator();
   has_barrier_creator
  ---------------------
   f
  (1 row)
  ```

- gs_hadr_in_recovery()

  描述：两地三中心跨Region容灾特性开启后，查询当前节点是否处于基于目标barrier的日志恢复中，还在恢复中返回true。只有完成日志恢复，才会启动switchover流程中的灾备数据库实例升为生产数据库实例的步骤（需要系统管理员角色）。

  返回值类型：Boolean

  备注：该函数只有在容灾数据库实例启动计划内switchover时使用。

  ```
  gaussdb=# SELECT * FROM gs_hadr_in_recovery();
   is_in_recovery
  ----------------
   t
  (1 row)
  ```

- gs_streaming_dr_get_switchover_barrier()

  描述：两地三中心跨Region容灾-基于流式复制的解决方案中，查询灾备数据库实例的DN实例是否已接收到switchover barrier日志并完成回放，已完成返回true。灾备数据库实例只有在所有DN实例都完成switchover barrier日志回放，才会启动switchover流程中的灾备数据库实例升为生产数据库实例的步骤（需要系统管理员角色）。

返回值类型：Boolean

备注：该函数只有在流式容灾解决方案中容灾数据库实例启动计划内switchover时使用。

```
gaussdb=# SELECT * FROM gs_streaming_dr_get_switchover_barrier();
 get_switchover_barrier
------------------------
 f
(1 row)
```

- gs_streaming_dr_service_truncation_check()

  描述：两地三中心跨Region容灾-基于流式复制的解决方案中，查询主数据库实例的DN实例是否已完成switchover barrier日志发送，已完成返回true。只有完成日志发送，才会启动switchover流程中的生产数据库实例降为灾备数据库实例的步骤（需要系统管理员角色）。

  返回值类型：Boolean

  备注：该函数只有在容灾数据库实例启动计划内switchover时使用。

```
gaussdb=# SELECT * FROM gs_streaming_dr_service_truncation_check();
 complete_truncation
---------------------
 f
(1 row)
```

- gs_hadr_local_rto_and_rpo_stat()

  描述：显示流式容灾的主数据库实例和备数据库实例日志流控信息（只可在主数据库实例的主DN使用，备DN以及备数据库实例均上不可获取到统计信息）。

  返回值类型：record，具体各个字段的类型和含义如**表7-56**所示。

表 7-56 gs_hadr_local_rto_and_rpo_stat 参数说明

参数	类型	描述
hadr_sender_node_name	text	节点的名称，包含主数据库实例和备数据库实例首备。
hadr_receiver_node_name	text	备数据库实例首备名称。
source_ip	text	主数据库实例主DN IP地址。
source_port	int	主数据库实例主DN通信端口。
dest_ip	text	备数据库实例首备DN IP地址。
dest_port	int	备数据库实例首备DN通信端口。
current_rto	int	流控的信息，当前主备数据库实例的日志rto时间（单位：秒）。
target_rto	int	流控的信息，目标主备数据库实例间的rto时间（单位：秒）。
current_rpo	int	流控的信息，当前主备数据库实例的日志rpo时间（单位：秒）。
target_rpo	int	流控的信息，目标主备数据库实例间的rpo时间（单位：秒）。

参数	类型	描述
rto_sleep_time	int	RTO流控信息，为了达到目标rto，预期主机walsender所需要的睡眠时间（单位：微秒）。
rpo_sleep_time	int	RPO流控信息，为了达到目标rpo，预期主机xlogInsert所需要的睡眠时间（单位：微秒）。

```
gaussdb=# SELECT * FROM gs_hadr_local_rto_and_rpo_stat();
 hadr_sender_node_name | hadr_receiver_node_name |  source_ip   | source_port |   dest_ip    |
dest_port | current_rto | target_rto | current_rpo | target_rpo | rto_sleep_time | rpo_sleep_time
-----------------------+-------------------------+--------------+-------------+--------------+-------------+-------------+------------+-------------+----------
+-------------+------------+-------------+------------+----------------+----------------
 dn_6001_6002_6003     | cluster_b_hadr_dn_6001  | <IP> |       <PORT> | <IP> |     <PORT> |           0
|           0 |           0 |           0 |           0 |              0
(1 row)
```

- gs_hadr_remote_rto_and_rpo_stat()

  描述：显示流式容灾其它非本地数据分片的主数据库实例和备数据库实例日志流控信，集中式部署场景不支持该函数。

7.5.25.7 快照同步函数

快照同步函数是导出当前快照的标识符。

- pg_export_snapshot()

  描述：保存当前的快照并返回它的标识符。

  返回值类型：text

  备注：函数pg_export_snapshot保存当前的快照并返回一个文本字符串标识此快照。这个字符串必须传递给想要导入快照的客户端。可用在set transaction snapshot snapshot_id时导入snapshot，但是应用的前提是该事务设置了SERIALIZABLE或REPEATABLE READ隔离级别。而GaussDB目前是不支持这两种隔离级别的。该函数的输出不可用做set transaction snapshot的输入。

  ```
  gaussdb=# SELECT * FROM pg_export_snapshot();
   pg_export_snapshot
  --------------------
   00000000000070AD-1
  (1 row)
  ```

- pg_export_snapshot_and_csn()

  描述：保存当前的快照并返回它的标识符。比pg_export_snapshot()多返回一列CSN，表示当前快照的CSN。

  返回值类型：text

  ```
  gaussdb=# SELECT * FROM pg_export_snapshot_and_csn();
       snapshot       | csn
  --------------------+------
   0000000000007129-1 | 390D
  (1 row)
  ```

7.5.25.8 数据库对象函数

数据库对象尺寸函数

数据库对象尺寸函数计算数据库对象使用的实际磁盘空间。

- pg_column_size(any)

  描述：存储一个指定的数值需要的字节数（可能压缩过）。

  返回值类型：int

  备注：pg_column_size显示用于存储某个独立数据值的空间。

```
gaussdb=# SELECT pg_column_size(1);
 pg_column_size
----------------
              4
(1 row)
```

- pg_database_size(oid)

  描述：指定OID代表的数据库使用的磁盘空间。

  返回值类型：bigint

- pg_database_size(name)

  描述：指定名称的数据库使用的磁盘空间。

  返回值类型：bigint

  备注：pg_database_size接受一个数据库的OID或者名称，然后返回该对象使用的全部磁盘空间。

  示例：

```
gaussdb=# CREATE DATABASE testdb dbcompatibility 'A';
CREATE DATABASE
gaussdb=# SELECT pg_database_size('testdb');
 pg_database_size
------------------
         51590112
(1 row)
gaussdb=# DROP DATABASE testdb;
DROP DATABASE
```

- pg_relation_size(oid)

  描述：指定OID代表的表或者索引所使用的磁盘空间。

  返回值类型：bigint

- get_db_source_datasize()

  描述：估算当前数据库非压缩态的数据总容量。

  返回值类型：bigint

  备注：调用该函数前需要做analyze。

  示例：

```
gaussdb=# analyze;
ANALYZE
gaussdb=# SELECT get_db_source_datasize();
 get_db_source_datasize
------------------------
            35384925667
(1 row)
```

- pg_relation_size(text)

  描述：指定名称的表或者索引使用的磁盘空间。表名称可以用模式名修饰。

  返回值类型：bigint

- pg_relation_size(relation regclass, fork text)

  描述：指定表或索引的指定分叉树（'main'，'fsm'或'vm'）使用的磁盘空间。

  返回值类型：bigint

- pg_relation_size(relation regclass)

  描述：pg_relation_size(..., 'main')的简写。

  返回值类型：bigint

  备注：pg_relation_size接受一个表、索引、压缩表的OID或者名称，然后返回它们的字节大小。

- pg_partition_size(oid, oid)

  描述：指定OID代表的分区使用的磁盘空间。其中，第一个oid为表的OID，第二个oid为分区的OID。

  返回值类型：bigint

- pg_partition_size(text, text)

  描述：指定名称的分区使用的磁盘空间。其中，第一个text为表名，第二个text为分区名。

  返回值类型：bigint

- pg_partition_indexes_size(oid, oid)

  描述：指定OID代表的分区的索引使用的磁盘空间。其中，第一个oid为表的OID，第二个oid为分区的OID。

  返回值类型：bigint

- pg_partition_indexes_size(text, text)

  描述：指定名称的分区的索引使用的磁盘空间。其中，第一个text为表名，第二个text为分区名。

  返回值类型：bigint

- pg_indexes_size(regclass)

  描述：附加到指定表的索引使用的总磁盘空间。

  返回值类型：bigint

- pg_size_pretty(bigint)

  描述：将以64位整数表示的字节值转换为具有单位的易读格式。

  返回值类型：text

- pg_size_pretty(numeric)

  描述：将以数值表示的字节值转换为具有单位的易读格式。

  返回值类型：text

  备注：pg_size_pretty用于把其他函数的结果格式化成一种易读的格式，可以根据情况使用kB 、MB 、GB 、TB。

- pg_table_size(regclass)

  描述：指定的表使用的磁盘空间，不计索引（但是包含TOAST，自由空间映射和可见性映射）。

  返回值类型：bigint

- pg_tablespace_size(oid)

  描述：指定OID代表的表空间使用的磁盘空间。

  返回值类型：bigint

- pg_tablespace_size(name)

  描述：指定名称的表空间使用的磁盘空间。

  返回值类型：bigint

备注：

pg_tablespace_size接受一个数据库的OID或者名称，然后返回该对象使用的全部磁盘空间。

- pg_total_relation_size(oid)

  描述：指定OID代表的表使用的磁盘空间，包括索引和压缩数据。

  返回值类型：bigint

- pg_total_relation_size(regclass)

  描述：指定的表使用的总磁盘空间，包括所有的索引和TOAST数据。

  返回值类型：bigint

- pg_total_relation_size(text)

  描述：指定名称的表所使用的全部磁盘空间，包括索引和压缩数据。表名称可以用模式名修饰。

  返回值类型：bigint

  备注：pg_total_relation_size接受一个表或者一个压缩表的OID或者名称，然后返回以字节计的数据和所有相关的索引和压缩表的尺寸。

- datalength(any)

  描述：计算一个指定的数据需要的字节数（不考虑数据的管理空间和数据压缩，数据类型转换等情况）。

  返回值类型：int

  备注：datalength用于计算某个独立数据值的空间。

  示例：

```
gaussdb=# SELECT datalength(1);
datalength
------------
4
(1 row)
```

目前支持的数据类型及计算方式见下表：

数据类型			存储空间
数值类型	整数类型	TINYINT	1
		SMALLINT	2
		INTEGER	4
		BINARY_INTEGER	4
		BIGINT	8
	任意精度型	DECIMAL	每4位十进制数占两个字节，小数点前后数字分别计算。
		NUMERIC	每4位十进制数占两个字节，小数点前后数字分别计算。
		NUMBER	每4位十进制数占两个字节，小数点前后数字分别计算。

数据类型			存储空间
	序列整型	SMALLSERIAL	2
		SERIAL	4
		BIGSERIAL	8
		LARGESERIAL	每4位十进制数占两个字节，小数点前后数字分别计算。
	浮点类型	FLOAT4	4
		DOUBLE PRECISION	8
		FLOAT8	8
		BINARY_DOUBLE	8
		FLOAT[(p)]	每4位十进制数占两个字节，小数点前后数字分别计算。
		DEC[(p[,s])]	每4位十进制数占两个字节，小数点前后数字分别计算。
		INTEGER[(p[,s])]	每4位十进制数占两个字节，小数点前后数字分别计算。
布尔类型	布尔类型	BOOLEAN	1
字符类型	字符类型	CHAR	n
		CHAR(n)	n
		CHARACTER(n)	n
		NCHAR(n)	n
		VARCHAR(n)	n
		CHARACTER	字符实际字节数。
		VARYING(n)	字符实际字节数。
		VARCHAR2(n)	字符实际字节数。
		NVARCHAR(n)	字符实际字节数。
		NVARCHAR2(n)	字符实际字节数。
		TEXT	字符实际字节数。
		CLOB	字符实际字节数。
时间类型	时间类型	DATE	8

数据类型			存储空间
		TIME	8
		TIMEZ	12
		TIMESTAMP	8
		TIMESTAMPZ	8
		SMALLDATETIME	8
		INTERVAL DAY TO SECOND	16
		INTERVAL	16
		RELTIME	4
		ABSTIME	4
		TINTERVAL	12

数据库对象位置函数

- pg_relation_filenode(relation regclass)

  描述：指定关系的文件节点数。

  返回值类型：oid

  备注：pg_relation_filenode接受一个表、索引、序列或压缩表的OID或者名称，并且返回当前分配给它的“filenode”数。文件节点是关系使用的文件名称的基本组件。对大多数表来说，结果和pg_class.relfilenode相同，但对确定的系统目录来说，relfilenode为0而且这个函数必须用来获取正确的值。如果传递一个没有存储的关系，比如一个视图，那么这个函数返回NULL。

- pg_relation_filepath(relation regclass)

  描述：指定关系的文件路径名。

  返回值类型：text

  备注：pg_relation_filepath类似于pg_relation_filenode，但是它返回关系的整个文件路径名（相对于数据库的数据目录PGDATA）。

- pg_filenode_relation(tablespace oid, filenode oid)

  描述：获取对应的tablespace和relfilenode所对应的表名。

  返回类型：regclass

- pg_partition_filenode(partition_oid)

  描述：获取到指定分区表的oid锁对应的filenode。

  返回类型：oid

- pg_partition_filepath(partition_oid)

  描述：指定分区的文件路径名。

  返回值类型：text

回收站对象函数

- gs_is_recycle_object(classid, objid, objname)
  描述：判断是否为回收站对象。
  返回值类型：Boolean

7.5.25.9 咨询锁函数

咨询锁函数用于管理咨询锁（Advisory Lock）。

- pg_advisory_lock(key bigint)
  描述：获取会话级别的排他咨询锁。
  返回值类型：void
  备注：pg_advisory_lock锁定应用程序定义的资源，该资源可以用一个64位或两个不重叠的32位键值标识。如果已经有另外的会话锁定了该资源，则该函数将阻塞到该资源可用为止。这个锁是排他的。多个锁定请求将会被压入栈中，因此，如果同一个资源被锁定了三次，它必须被解锁三次以将资源释放给其他会话使用。
- pg_advisory_lock(key1 int, key2 int)
  描述：获取会话级别的排他咨询锁。
  返回值类型：void
  备注：只允许sysadmin对键值对(65535, 65535)加会话级别的排他咨询锁，普通用户无权限。
- pg_advisory_lock(lock_id int4, lock_id int4, datebase_name Name)
  描述：通过传入锁ID和数据库名字，获取指定数据库的排他咨询锁。
  返回值类型：void
- pg_advisory_lock_shared(key bigint)
  描述：获取会话级别的共享咨询锁。
  返回值类型：void
- pg_advisory_lock_shared(key1 int, key2 int)
  描述：获取会话级别的共享咨询锁。
  返回值类型：void
  备注：pg_advisory_lock_shared类似于pg_advisory_lock，不同之处仅在于共享锁会话可以和其他请求共享锁的会话共享资源，但排他锁除外。
- pg_advisory_unlock(key bigint)
  描述：释放会话级别的排他咨询锁。
  返回值类型：Boolean
- pg_advisory_unlock(key1 int, key2 int)
  描述：释放会话级别的排他咨询锁。
  返回值类型：Boolean
  备注：pg_advisory_unlock释放先前取得的排他咨询锁。如果释放成功则返回true。如果实际上并未持有指定的锁，将返回false并在服务器中产生一条SQL警告信息。
- pg_advisory_unlock(lock_id int4, lock_id int4, datebase_name Name)
  描述：通过传入锁ID和数据库名字，释放指定数据库上的排他咨询锁。

返回值类型：Boolean

备注：如果释放成功则返回true；如果未持有锁，则返回false。

- pg_advisory_unlock_shared(key bigint)

  描述：释放会话级别的共享咨询锁。

  返回值类型：Boolean

- pg_advisory_unlock_shared(key1 int, key2 int)

  描述：释放会话级别的共享咨询锁。

  返回值类型：Boolean

  备注：pg_advisory_unlock_shared类似于pg_advisory_unlock，不同之处在于该函数释放的是共享咨询锁。

- pg_advisory_unlock_all()

  描述：释放当前会话持有的所有咨询锁。

  返回值类型：void

  备注：pg_advisory_unlock_all将会释放当前会话持有的所有咨询锁，该函数在会话结束的时候被隐含调用，即使客户端异常地断开连接也是一样。

- pg_advisory_xact_lock(key bigint)

  描述：获取事务级别的排他咨询锁。

  返回值类型：void

- pg_advisory_xact_lock(key1 int, key2 int)

  描述：获取事务级别的排他咨询锁。

  返回值类型：void

  备注：pg_advisory_xact_lock类似于pg_advisory_lock，不同之处在于锁是自动在当前事务结束时释放，而且不能被显式的释放。只允许sysadmin对键值对(65535, 65535)加事务级别的排他咨询锁，普通用户无权限。

- pg_advisory_xact_lock_shared(key bigint)

  描述：获取事务级别的共享咨询锁。

  返回值类型：void

- pg_advisory_xact_lock_shared(key1 int, key2 int)

  描述：获取事务级别的共享咨询锁。

  返回值类型：void

  备注：pg_advisory_xact_lock_shared类似于pg_advisory_lock_shared，不同之处在于锁是在当前事务结束时自动释放，而且不能被显式的释放。

- pg_try_advisory_lock(key bigint)

  描述：尝试获取会话级排他咨询锁。

  返回值类型：Boolean

  备注：pg_try_advisory_lock类似于pg_advisory_lock，不同之处在于该函数不会阻塞以等待资源的释放。它要么立即获得锁并返回true，要么返回false表示目前不能锁定。

- pg_try_advisory_lock(key1 int, key2 int)

  描述：尝试获取会话级排他咨询锁。

  返回值类型：Boolean

备注：只允许sysadmin对键值对(65535, 65535)加会话级别的排他咨询锁，普通用户无权限。

- pg_try_advisory_lock_shared(key bigint)

  描述：尝试获取会话级共享咨询锁。

  返回值类型：Boolean

- pg_try_advisory_lock_shared(key1 int, key2 int)

  描述：尝试获取会话级共享咨询锁。

  返回值类型：Boolean

  备注：pg_try_advisory_lock_shared类似于pg_try_advisory_lock，不同之处在于该函数尝试获得共享锁而不是排他锁。

- pg_try_advisory_xact_lock(key bigint)

  描述：尝试获取事务级别的排他咨询锁。

  返回值类型：Boolean

- pg_try_advisory_xact_lock(key1 int, key2 int)

  描述：尝试获取事务级别的排他咨询锁。

  返回值类型：Boolean

  备注：pg_try_advisory_xact_lock类似于pg_try_advisory_lock，不同之处在于如果得到锁，在当前事务的结束时自动释放，而且不能被显式的释放。只允许sysadmin对键值对(65535, 65535)加事务级别的排他咨询锁，普通用户无权限。

- pg_try_advisory_xact_lock_shared(key bigint)

  描述：尝试获取事务级别的共享咨询锁。

  返回值类型：Boolean

- pg_try_advisory_xact_lock_shared(key1 int, key2 int)

  描述：尝试获取事务级别的共享咨询锁。

  返回值类型：Boolean

  备注：pg_try_advisory_xact_lock_shared类似于pg_try_advisory_lock_shared，不同之处在于如果得到锁，在当前事务结束时自动释放，而且不能被显式的释放。

- lock_cluster_ddl()

  描述：尝试对数据库内所有存活的数据库主节点获取会话级别的排他咨询锁。

  返回值类型：Boolean

  备注：只允许sysadmin调用，普通用户无权限。

- unlock_cluster_ddl()

  描述：尝试对数据库主节点会话级别的排他咨询锁。

  返回值类型：Boolean

  会话级别咨询锁示例：

```
gaussdb=# select pg_advisory_lock(1); --依次获取3个会话级别的咨询锁
 pg_advisory_lock
------------------
(1 row)

gaussdb=# select pg_advisory_lock(1,2);
 pg_advisory_lock
------------------
(1 row)
gaussdb=# select pg_advisory_lock(2,2,'postgres');
```



```
 pg_advisory_lock
------------------
(1 row)

gaussdb=# select * from pg_locks;  --查询pg_locks表，看到新增了3个advisory类型的锁
  locktype   | database | relation | page | tuple | bucket | virtualxid | transactionid | classid | objid |
objsubid | virtualtransaction |       pid       |    sessionid    |       mode       | granted | fastpath |
locktag       | global_sessionid
-------------+----------+----------+------+-------+--------+------------+---------------+---------+-------
+----------+--------------------+-----------------+-----------------+------------------+---------+----------+------
--------------+------------------
 relation   |    12850 |    10652 |      |       |        |            |               |         |       |
4/27611            | 140245958915840 | 140245958915840 | AccessShareLock | t       | t        | 3232:
299c:0:0:0:0 | 0:0#0
 virtualxid |          |          |      |       |        | 4/27611    |               |         |       | 4/27611            |
140245958915840 | 140245958915840 | ExclusiveLock   | t       | t        | 4:6bd
b:0:0:0:7    | 0:0#0
 virtualxid |          |          |      |       |        | 19/16219   |               |         |       | 19/16219
| 140246771033856 | 140246771033856 | ExclusiveLock   | t       | t        | 13:3f
5b:0:0:0:7   | 0:0#0
 virtualxid |          |          |      |       |        | 17/97293   |               |         |       | 17/97293
| 140246750590720 | 140246750590720 | ExclusiveLock   | t       | t        | 11:17
c0d:0:0:0:7  | 0:0#0
 advisory   |    12850 |          |      |       |        |            |               |       0 |     1 |        1 | 4/27611
| 140245958915840 | 140245958915840 | ExclusiveLock   | t       | f        | 3232:
0:1:1:0:b    | 0:0#0
 advisory   |    12850 |          |      |       |        |            |               |       1 |     2 |        2 | 4/27611
| 140245958915840 | 140245958915840 | ExclusiveLock   | t       | f        | 3232:
1:2:2:0:b    | 0:0#0
 advisory   |    12850 |          |      |       |        |            |               |       2 |     2 |        2 | 4/27611
| 140245958915840 | 140245958915840 | ExclusiveLock   | t       | f        | 3232:
2:2:2:0:b    | 0:0#0
(7 rows)

gaussdb=# select pg_advisory_unlock(1);  --依次释放3个会话级别的咨询锁
 pg_advisory_unlock
--------------------
 t
(1 row)

gaussdb=# select pg_advisory_unlock(1,2);
 pg_advisory_unlock
--------------------
 t
(1 row)

gaussdb=# select pg_advisory_unlock(2,2,'postgres');
 pg_advisory_unlock
--------------------
 t
(1 row)

gaussdb=# select * from pg_locks;  --查询pg_locks表，看到3个advisory类型的锁已被释放
  locktype   | database | relation | page | tuple | bucket | virtualxid | transactionid | classid | objid |
objsubid | virtualtransaction |       pid       |    sessionid    |       mode       | granted | fastpath |
locktag       | global_sessionid
-------------+----------+----------+------+-------+--------+------------+---------------+---------+-------
+----------+--------------------+-----------------+-----------------+------------------+---------+----------+------
--------------+------------------
 relation   |    12850 |    10652 |      |       |        |            |               |         |       |
4/27618            | 140245958915840 | 140245958915840 | AccessShareLock | t       | t        | 3232:
299c:0:0:0:0 | 0:0#0
 virtualxid |          |          |      |       |        | 4/27618    |               |         |       | 4/27618            |
140245958915840 | 140245958915840 | ExclusiveLock   | t       | t        | 4:6be
2:0:0:0:7    | 0:0#0
 virtualxid |          |          |      |       |        | 19/16254   |               |         |       | 19/16254
| 140246771033856 | 140246771033856 | ExclusiveLock   | t       | t        | 13:3f
7e:0:0:0:7   | 0:0#0
 virtualxid |          |          |      |       |        | 17/97506   |               |         |       | 17/97506
```

```
| 140246750590720 | 140246750590720 | ExclusiveLock   | t       | t       | 11:17
ce2:0:0:0:7  | 0:0#0
(4 rows)

gaussdb=# select pg_advisory_lock(1); --再次获取会话级别的咨询锁
 pg_advisory_lock
------------------

(1 row)

gaussdb=# select pg_advisory_unlock_all();  --释放所有会话级别的咨询锁
 pg_advisory_unlock_all
------------------------

(1 row)

gaussdb=# select * from pg_locks;  --查询pg_locks表，看到advisory类型的锁已被释放
  locktype  | database | relation | page | tuple | bucket | virtualxid | transactionid | classid | objid |
objsubid | virtualtransaction |       pid       |    sessionid    |      mode       | granted | fastpath |
locktag      | global_sessionid
------------+----------+----------+------+-------+--------+------------+---------------+---------+-------
+----------+--------------------+-----------------+-----------------+-----------------+---------+----------+------
-------------+------------------
 relation   |    12850 |    10652 |      |       |        |            |               |         |      |
4/27629           | 140245958915840 | 140245958915840 | AccessShareLock | t       | t       | 3232:
299c:0:0:0:0 | 0:0#0
 virtualxid |          |          |      |       |        | 4/27629    |               |         |      | 4/27629           |
140245958915840 | 140245958915840 | ExclusiveLock   | t       | t       | 4:6be
d:0:0:0:7    | 0:0#0
 virtualxid |          |          |      |       |        | 19/16369   |               |         |      | 19/16369
| 140246771033856 | 140246771033856 | ExclusiveLock   | t       | t       | 13:3f
f1:0:0:0:7   | 0:0#0
 virtualxid |          |          |      |       |        | 17/98194   |               |         |      | 17/98194
| 140246750590720 | 140246750590720 | ExclusiveLock   | t       | t       | 11:17
f92:0:0:0:7  | 0:0#0
(4 rows)
```

事务级别咨询锁示例：

```
gaussdb=# select pg_advisory_xact_lock_shared(1); --获取事务级别的共享咨询锁
 pg_advisory_xact_lock_shared
------------------------------

(1 row)

gaussdb=# select * from pg_locks; --因单条语句视为一个事务，因此咨询锁已随事务提交自动释放。查询
pg_locks表，看不到事务级咨询锁
  locktype  | database | relation | page | tuple | bucket | virtualxid | transactionid | classid | objid |
objsubid | virtualtransaction |       pid       |    sessionid    |      mode       | granted | fastpath |
locktag      | global_sessionid
------------+----------+----------+------+-------+--------+------------+---------------+---------+-------
+----------+--------------------+-----------------+-----------------+-----------------+---------+----------+------
-------------+------------------
 relation   |    12850 |    10652 |      |       |        |            |               |         |      |
4/27631           | 140245958915840 | 140245958915840 | AccessShareLock | t       | t       | 3232:
299c:0:0:0:0 | 0:0#0
 virtualxid |          |          |      |       |        | 4/27631    |               |         |      | 4/27631           |
140245958915840 | 140245958915840 | ExclusiveLock   | t       | t       | 4:6be
f:0:0:0:7    | 0:0#0
 virtualxid |          |          |      |       |        | 19/16396   |               |         |      | 19/16396
| 140246771033856 | 140246771033856 | ExclusiveLock   | t       | t       | 13:40
0c:0:0:0:7   | 0:0#0
 virtualxid |          |          |      |       |        | 17/98359   |               |         |      | 17/98359
| 140246750590720 | 140246750590720 | ExclusiveLock   | t       | t       | 11:18
037:0:0:0:7  | 0:0#0
(4 rows)

gaussdb=# start transaction;  --显式开启事务块
START TRANSACTION
gaussdb=# select pg_advisory_xact_lock_shared(1);  --获取事务级别的共享咨询锁
 pg_advisory_xact_lock_shared
------------------------------
```

```
(1 row)

gaussdb=# select * from pg_locks;  --查询pg_locks表，新增了一个advisory类型锁
  locktype   | database | relation | page | tuple | bucket | virtualxid | transactionid | classid | objid |
objsubid | virtualtransaction |       pid       |    sessionid    |       mode       | granted | fastpath |
locktag       | global_sessionid
-------------+----------+----------+------+-------+--------+------------+---------------+---------+-------
+----------+--------------------+-----------------+-----------------+------------------+---------+----------+------
--------------+------------------
 relation   |    12850 |    10652 |      |       |        |            |               |         |       |
4/27632            | 140245958915840 | 140245958915840 | AccessShareLock | t       | t        | 3232:
299c:0:0:0:0 | 0:0#0
 virtualxid |          |          |      |       |        | 4/27632    |               |         |       |
4/27632            | 140245958915840 | 140245958915840 | ExclusiveLock   | t       | t        | 4:6bf
0:0:0:0:7    | 0:0#0
 virtualxid |          |          |      |       |        | 19/16399   |               |         |       |
19/16399           | 140246771033856 | 140246771033856 | ExclusiveLock   | t       | t        | 13:40
0f:0:0:0:7   | 0:0#0
 virtualxid |          |          |      |       |        | 17/98377   |               |         |       |
17/98377           | 140246750590720 | 140246750590720 | ExclusiveLock   | t       | t        | 11:18
049:0:0:0:7  | 0:0#0
 advisory   |    12850 |          |      |       |        |            |               |       0 |     1 |
1 | 4/27632            | 140245958915840 | 140245958915840 | ShareLock       | t       | f        | 3232:
0:1:1:0:b    | 0:0#0
(5 rows)

gaussdb=# commit;  --提交事务块
COMMIT
gaussdb=# select * from pg_locks;  --查询pg_locks表，事务级别advisory类型锁已自动释放
  locktype   | database | relation | page | tuple | bucket | virtualxid | transactionid | classid | objid |
objsubid | virtualtransaction |       pid       |    sessionid    |       mode       | granted | fastpath |
locktag       | global_sessionid
-------------+----------+----------+------+-------+--------+------------+---------------+---------+-------
+----------+--------------------+-----------------+-----------------+------------------+---------+----------+------
--------------+------------------
 relation   |    12850 |    10652 |      |       |        |            |               |         |       |
4/27633            | 140245958915840 | 140245958915840 | AccessShareLock | t       | t        | 3232:
299c:0:0:0:0 | 0:0#0
 virtualxid |          |          |      |       |        | 4/27633    |               |         |       |
4/27633            | 140245958915840 | 140245958915840 | ExclusiveLock   | t       | t        | 4:6bf
1:0:0:0:7    | 0:0#0
 virtualxid |          |          |      |       |        | 19/16401   |               |         |       |
19/16401           | 140246771033856 | 140246771033856 | ExclusiveLock   | t       | t        | 13:40
11:0:0:0:7   | 0:0#0
 virtualxid |          |          |      |       |        | 17/98385   |               |         |       |
17/98385           | 140246750590720 | 140246750590720 | ExclusiveLock   | t       | t        | 11:18
051:0:0:0:7  | 0:0#0
(4 rows)
```

尝试获取咨询锁示例：

```
gaussdb=# select pg_try_advisory_lock_shared(1); --在没有其他连接加咨询锁时，尝试获取对象1的共享
咨询锁成功
 pg_try_advisory_lock_shared
-----------------------------
 t
(1 row)

gaussdb=# select pg_advisory_unlock_all();  --释放这个咨询锁
 pg_advisory_unlock_all
------------------------

(1 row)

gaussdb=# select pg_advisory_lock(1); --获取对象1的排他咨询锁
 pg_advisory_lock
------------------

(1 row)
```



```
--保持这个连接不断开，新启动一个连接
gaussdb=# select pg_try_advisory_lock_shared(1); --因已有连接持有排他锁，所以新连接中尝试获取对象1的共享咨询锁失败
 pg_try_advisory_lock_shared
-----------------------------
 f
(1 row)
```

7.5.25.10 逻辑复制函数

说明

使用逻辑复制函数，需要设置GUC参数wal_level为logical。具体配置请参考《特性指南》的“逻辑复制 > 逻辑解码 > 使用SQL函数接口进行逻辑解码”章节。

- pg_create_logical_replication_slot('slot_name', 'plugin_name', 'output_order')

  描述：创建逻辑复制槽。

  参数说明：

  - slot_name

    流复制槽名称。

    取值范围：字符串，仅支持小写字母、数字以及“_”，“?”，“-”，“.”字符，且不支持“.”或“..”单独作为复制槽名称。

  - plugin_name

    插件名称。

    取值范围：字符串，当前支持mppdb_decoding。

  - output_order

    复制槽解码结果输出顺序，该参数为可选参数。

    取值范围：0或1，默认值为0。

    - 0：设为0时，复制槽解码结果按照事务的COMMIT LSN排序。此时复制槽的confirmed_csn为0，此复制槽称为**LSN序复制槽**。
    - 1：设为1时，复制槽解码结果按照事务的CSN排序。此时复制槽的confirmed_csn为非0值，此复制槽称为**CSN序复制槽**。

  返回值类型：name, text

  示例：

  ```
  gaussdb=# SELECT * FROM pg_create_logical_replication_slot('slot_lsn','mppdb_decoding',0);
   slotname | xlog_position
  ----------+---------------
   slot_lsn | 0/6D08B58
  (1 row)

  gaussdb=# SELECT * FROM pg_create_logical_replication_slot('slot_csn','mppdb_decoding',1);
   slotname | xlog_position
  ----------+---------------
   slot_csn | 0/59AD800
  (1 row)
  ```

  备注：第一个返回值表示slot_name，第二个返回值在LSN序复制槽和CSN序复制槽下有不同含义。对于LSN序复制槽，该值为复制槽的confirmed_flush，表示COMMIT LSN小于等于该值的事务之后不会被解码输出；对于CSN序复制槽，该值为复制槽的confirmed_csn，表示CSN小于等于该值的事务之后不会被解码输出。调用该函数的用户需要具有SYSADMIN权限或具有REPLICATION权限或继承了内置角色gs_role_replication的权限。此函数目前只支持在主机调用。

- pg_create_physical_replication_slot('slot_name', 'isDummyStandby')

描述：创建新的物理复制槽。

参数说明：

- slot_name

  流复制槽名称。

  取值范围：字符串，仅支持小写字母、数字以及“_”，“?”，“-”，“.”字符，且不支持“.”或“..”单独作为复制槽名称。

- isDummyStandby

  预留参数。

  类型：bool。

返回值类型：name, text

备注：调用该函数的用户需要具有SYSADMIN权限或具有REPLICATION权限或继承了内置角色gs_role_replication的权限。由于该函数创建的物理复制槽没有restart_lsn，会被认为是无效槽，在做checkpoint时会被自动删除。

- pg_drop_replication_slot('slot_name')

  描述：删除流复制槽。

  参数说明：

  - slot_name

    流复制槽名称。

    取值范围：字符串，仅支持小写字母、数字以及“_”，“?”，“-”，“.”字符，且不支持“.”或“..”单独作为复制槽名称。

  返回值类型：void

  备注：调用该函数的用户需要具有SYSADMIN权限或具有REPLICATION权限或继承了内置角色gs_role_replication的权限。此函数目前只支持在主机调用。

- pg_logical_slot_peek_changes('slot_name', 'upto_lsn', upto_nchanges, 'options_name', 'options_value')

  描述：解码并不推进流复制槽（下次解码可以再次获取本次解出的数据）。

  参数说明：

  - slot_name

    流复制槽名称。

    取值范围：字符串，仅支持小写字母、数字以及“_”，“?”，“-”，“.”字符，且不支持“.”或“..”单独作为复制槽名称。

  - upto_lsn

    在CSN序逻辑复制槽上代表日志的CSN，表示解码直到小于等于此CSN的事务日志解码完毕（可能会解码一个CSN大于指定CSN的事务）；在LSN序复制槽上代表日志的LSN，表示解码直到COMMIT LSN大于等于该LSN的第一个事务解码完毕。

    取值范围：字符串（代表十六进制格式表示的uint64，在正中间用'/'分割，左右两边各为一个uint32，如某个uint32为0则显示0），如'1/2AAFC60'、'0/A060'或'3A/0'。为NULL时表示不对解码截止的日志位置做限制。

  - upto_nchanges

    解码条数（包含begin和commit）。假设一共有三条事务，分别包含3、5、7条记录，如果upto_nchanges为4，那么会解码出前两个事务共8条记录。解码完第二条事务时发现解码条数记录大于等于upto_nchanges，会停止解码。

取值范围：非负整数。

说明

upto_lsn和upto_nchanges中任一参数达到限制，解码都会结束。

- options：此项为可选参数，由一系列options_name和options_value一一对应组成。
  - include-xids
    解码出的data列是否包含xid信息。
    取值范围：0或1，默认值为1。
    - 0：设为0时，解码出的data列不包含xid信息。
    - 1：设为1时，解码出的data列包含xid信息。
  - skip-empty-xacts
    解码时是否忽略空事务信息。
    取值范围：0或1，默认值为0。
    - 0：设为0时，解码时不忽略空事务信息。
    - 1：设为1时，解码时会忽略空事务信息。
  - include-timestamp
    解码信息是否包含commit时间戳。
    取值范围：0或1，默认值为0。
    - 0：设为0时，解码信息不包含commit时间戳。
    - 1：设为1时，解码信息包含commit时间戳。
  - only-local
    是否仅解码本地日志。
    取值范围：0或1，默认值为1。
    - 0：设为0时，解码非本地日志和本地日志。
    - 1：设为1时，仅解码本地日志。
  - force-binary
    是否以二进制格式输出解码结果。
    取值范围：0，默认值为0。
    - 0：设为0时，以文本格式输出解码结果。
  - white-table-list
    白名单参数，包含需要进行解码的schema和表名。
    取值范围：包含白名单中表名的字符串，不同的表以','为分隔符进行隔离；使用'*'来模糊匹配所有情况；schema名和表名间以'.'分隔，不允许存在任意空白符。例：

    ```
    gaussdb=# SELECT * FROM pg_logical_slot_peek_changes('slot1', NULL, 4096, 'white-table-list', 'public.t1,public.t2');
    ```

  - max-txn-in-memory
    内存管控参数，单位为MB，单个事务占用内存大于该值即进行落盘。

取值范围：0~100的整型，默认值为0，即不开启此种管控。

- max-reorderbuffer-in-memory

  内存管控参数，单位为GB，拼接-发送线程中正在拼接的事务总内存（包含缓存）大于该值则对当前解码事务进行落盘。

  取值范围：0~100的整型，默认值为0，即不开启此种管控。

- include-user

  事务的BEGIN逻辑日志是否输出事务的用户名字。

  取值范围：0或1，默认值为0。

  - 0：设为0时，事务的BEGIN逻辑日志不输出事务的用户名字。
  - 1：设为1时，事务的BEGIN逻辑日志输出事务的用户名字。

- exclude-userids

  黑名单用户的OID参数。

  取值范围：指定黑名单用户的OID，多个OID通过','分隔，不校验用户OID是否存在。

- exclude-users

  黑名单用户的名字参数。

  取值范围：指定黑名单用户的名字，多个名字通过','分隔；通过dynamic-resolution设置是否动态解析识别用户名字。若解码报错用户不存在而中断，在确定日志产生时刻不存在对应的黑名单用户，可以通过配置dynamic-resolution成true或者从用户黑名单中删除报错用户名字来启动解码继续获取逻辑日志。

- dynamic-resolution

  是否动态解析黑名单用户名字。

  取值范围：0或1，默认值为1。

  - 0：设为0时，当解码观测到黑名单exclude-users中用户不存在时将会报错并退出逻辑解码。
  - 1：设为1时，当解码观测到黑名单exclude-users中用户不存在时继续解码。

说明

其他配置选项可参考《特性指南》中“逻辑复制 > 逻辑解码 > 逻辑解码选项”章节。

返回值类型：text、xid、text

示例：

```
gaussdb=# SELECT * FROM pg_logical_slot_peek_changes('slot_lsn',NULL,4096,'skip-empty-xacts','on');
 location  | xid  |                                                                    data
-----------+-------
+---------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------
 0/6D0B500 | 46914 | BEGIN 46914
 0/6D0B530 | 46914 | {"table_name":"public.t1","op_type":"INSERT","columns_name":
["a","b"],"columns_type":["integer","integer"],"columns_val":["3","1"],"old_keys_name":
[],"old_keys_type":[],"old_keys_val":[]}
 0/6D0B8B8 | 46914 | COMMIT 46914 (at 2023-02-22 17:29:31.090018+08) CSN 94034528
 0/6D0BB58 | 46915 | BEGIN 46915
 0/6D0BB88 | 46915 | {"table_name":"public.t1","op_type":"INSERT","columns_name":
["a","b"],"columns_type":["integer","integer"],"columns_val":["3","2"],"old_keys_name":
[],"old_keys_type":[],"old_keys_val":[]}
```

```
 0/6D0BF08 | 46915 | COMMIT 46915 (at 2023-02-22 17:31:30.672093+08) CSN 94034568
 0/6D0BF08 | 46916 | BEGIN 46916
 0/6D0BF38 | 46916 | {"table_name":"public.t1","op_type":"INSERT","columns_name":
["a","b"],"columns_type":["integer","integer"],"columns_val":["3","3"],"old_keys_name":
[],"old_keys_type":[],"old_keys_val":[]}
 0/6D0C218 | 46916 | COMMIT 46916 (at 2023-02-22 17:31:34.438319+08) CSN 94034570
(9 rows)

gaussdb=# SELECT * FROM pg_logical_slot_peek_changes('slot_csn',NULL,4096,'skip-empty-xacts','on');
 location  | xid  |                                                   data
-----------+-------
+----------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------
 0/0       | 46914 | BEGIN CSN: 94034528
 0/0       | 46914 | {"table_name":"public.t1","op_type":"INSERT","columns_name":
["a","b"],"columns_type":["integer","integer"],"columns_val":["3","1"],"old_keys_name":
[],"old_keys_type":[],"old_keys_val":[]}
 0/59ADA60 | 46914 | COMMIT 46914 (at 2023-02-22 17:29:31.090018+08) CSN 94034528
 0/59ADA60 | 46915 | BEGIN CSN: 94034568
 0/59ADA60 | 46915 | {"table_name":"public.t1","op_type":"INSERT","columns_name":
["a","b"],"columns_type":["integer","integer"],"columns_val":["3","2"],"old_keys_name":
[],"old_keys_type":[],"old_keys_val":[]}
 0/59ADA88 | 46915 | COMMIT 46915 (at 2023-02-22 17:31:30.672093+08) CSN 94034568
 0/59ADA88 | 46916 | BEGIN CSN: 94034570
 0/59ADA88 | 46916 | {"table_name":"public.t1","op_type":"INSERT","columns_name":
["a","b"],"columns_type":["integer","integer"],"columns_val":["3","3"],"old_keys_name":
[],"old_keys_type":[],"old_keys_val":[]}
 0/59ADA8A | 46916 | COMMIT 46916 (at 2023-02-22 17:31:34.438319+08) CSN 94034570
(9 rows)
```

备注：上述函数示例中，slot_lsn/slot_csn为逻辑复制槽名称。函数返回解码结果，每一条解码结果包含三列，对应上述返回值类型，分别表示LSN（LSN序复制槽）或CSN（CSN序复制槽）位置、xid和解码内容。其中location列代表CSN时，仅当解码到COMMIT日志时才会更新。

调用该函数的用户需要具有SYSADMIN权限或具有REPLICATION权限或继承了内置角色gs_role_replication的权限。

- pg_logical_slot_get_changes('slot_name', 'upto_lsn', upto_nchanges, 'options_name', 'options_value')

  描述：解码并推进流复制槽。

  参数说明：与pg_logical_slot_peek_changes一致，详细内容请参见**•pg_logical_slot_peek_ch...**。

  备注：调用该函数的用户需要具有SYSADMIN权限或具有REPLICATION权限或继承了内置角色gs_role_replication的权限。函数支持在主机或备机调用，在备机调用会同步推进主机上对应的逻辑复制槽。

  说明

  在备机上执行该函数，推进主机对应复制槽时需占用主机的一个walsender。由于逻辑解码功能会为每个逻辑复制槽预留walsender，因此正常场景执行该函数会正常推进主机的逻辑复制槽，如果短时间连续执行该函数会导致通知主机推进失败，且没有报错。

- pg_logical_slot_peek_binary_changes('slot_name', 'upto_lsn', upto_nchanges, 'options_name', 'options_value')

  描述：以二进制格式解码且不推进流复制槽（下次解码可以再次获取本次解出的数据）。

  参数说明：

  – slot_name

    流复制槽名称。

    取值范围：字符串，仅支持小写字母、数字以及“_”，“?”，“-”，“.”字符，且不支持“.”或“..”单独作为复制槽名称。

- upto_lsn

  在CSN序逻辑复制槽上代表日志的CSN，表示解码直到小于等于此CSN的事务日志解码完毕（可能会解码一个CSN大于指定CSN的事务）；在LSN序复制槽上代表日志的LSN，表示解码直到COMMIT LSN大于等于该LSN的第一个事务解码完毕。

  取值范围：字符串（代表十六进制格式表示的uint64，在正中间用'/'分割，左右两边各为一个uint32，如某个uint32为0则显示0），如'1/2AAFC60'、'0/A060'或'3A/0'。为NULL时表示不对解码截止的日志位置做限制。

- upto_nchanges

  解码条数（包含begin和commit）。假设一共有三条事务，分别包含3、5、7条记录，如果upto_nchanges为4，那么会解码出前两个事务共8条记录。解码完第二条事务时发现解码条数记录大于等于upto_nchanges，会停止解码。

  取值范围：非负整数。

  说明

  upto_lsn和upto_nchanges中任一参数达到限制，解码都会结束。

- options：此项为可选参数，由一系列options_name和options_value一一对应组成。

  - include-xids

    解码出的data列是否包含xid信息。

    取值范围：0或1，默认值为1。

    - 0：设为0时，解码出的data列不包含xid信息。
    - 1：设为1时，解码出的data列包含xid信息。

  - skip-empty-xacts

    解码时是否忽略空事务信息。

    取值范围：0或1，默认值为0。

    - 0：设为0时，解码时不忽略空事务信息。
    - 1：设为1时，解码时会忽略空事务信息。

  - include-timestamp

    解码信息是否包含commit时间戳。

    取值范围：0或1，默认值为0。

    - 0：设为0时，解码信息不包含commit时间戳。
    - 1：设为1时，解码信息包含commit时间戳。

  - only-local

    是否仅解码本地日志。

    取值范围：0或1，默认值为1。

    - 0：设为0时，解码非本地日志和本地日志。
    - 1：设为1时，仅解码本地日志。

  - force-binary

    是否以二进制格式输出解码结果。

取值范围：0或1，默认值为0，均以二进制格式输出结果。

- white-table-list

  白名单参数，包含需要进行解码的schema和表名。

  取值范围：包含白名单中表名的字符串，不同的表以','为分隔符进行隔离；使用'*'来模糊匹配所有情况；schema名和表名间以'.'分隔，不允许存在任意空白符。例：select * from pg_logical_slot_peek_binary_changes('slot1', NULL, 4096, 'white-table-list', 'public.t1,public.t2');

- max-txn-in-memory

  内存管控参数，单位为MB，单个事务占用内存大于该值即进行落盘。

  取值范围：0~100的整型，默认值为0，即不开启此种管控。

- max-reorderbuffer-in-memory

  内存管控参数，单位为GB，拼接-发送线程中正在拼接的事务总内存（包含缓存）大于该值则对当前解码事务进行落盘。

  取值范围：0~100的整型，默认值为0，即不开启此种管控。

- include-user

  事务的BEGIN逻辑日志是否输出事务的用户名字。

  取值范围：0或1，默认值为0。

  - 0：设为0时，事务的BEGIN逻辑日志不输出事务的用户名字。
  - 1：设为1时，事务的BEGIN逻辑日志输出事务的用户名字。

- exclude-userids

  黑名单用户的OID参数。

  取值范围：指定黑名单用户的OID，多个OID通过','分隔，不校验用户OID是否存在。

- exclude-users

  黑名单用户的名字参数。

  取值范围：指定黑名单用户的名字，多个名字通过','分隔；通过dynamic-resolution设置是否动态解析识别用户名字。若解码报错用户不存在而中断，在确定日志产生时刻不存在对应的黑名单用户，可以通过配置dynamic-resolution成true或者从用户黑名单中删除报错用户名字来启动解码继续获取逻辑日志。

- dynamic-resolution

  是否动态解析黑名单用户名字。

  取值范围：0或1，默认值为1。

  - 0：设为0时，当解码观测到黑名单exclude-users中用户不存在时将会报错并退出逻辑解码。
  - 1：设为1时，当解码观测到黑名单exclude-users中用户不存在时继续解码。

说明

某些配置选项在函数中仅能配置而不实际生效，可参考《特性指南》中“逻辑复制 > 逻辑解码 > 逻辑解码选项”章节。

返回值类型：text、xid、bytea

备注：函数返回解码结果，每一条解码结果包含三列，对应上述返回值类型，分别表示LSN位置、xid和二进制格式的解码内容。调用该函数的用户需要具有SYSADMIN权限或具有REPLICATION权限或继承了内置角色gs_role_replication的权限。

- pg_logical_slot_get_binary_changes('slot_name', 'upto_lsn', upto_nchanges, 'options_name', 'options_value')

  描述：以二进制格式解码并推进流复制槽。

  参数说明：与pg_logical_slot_peek_binary_changes一致，详细内容请参见**•pg_logical_slot_peek_bi...**。

  备注：调用该函数的用户需要具有SYSADMIN权限或具有REPLICATION权限或继承了内置角色gs_role_replication的权限。此函数目前只支持在主机上调用。

- pg_replication_slot_advance ('slot_name', 'upto_lsn')

  描述：直接推进流复制槽到指定upto_lsn，不输出解码结果。

  参数说明：

  – slot_name

    流复制槽名称。

    取值范围：字符串，仅支持小写字母、数字以及“_”，“?”，“-”，“.”字符，且不支持“.”或“..”单独作为复制槽名称。

  – upto_lsn

    在CSN序逻辑复制槽上代表推进到的日志CSN位置，下次解码时只会输出比该CSN大的事务结果。如果输入的CSN比当前流复制槽记录的confirmed_csn还要小，则直接返回；如果输入的CSN比当前可获取的最新CSN要大或并未输入，则推进到当前可获取的最新CSN。

    在LSN序复制槽上代表推进到的日志LSN位置，下次解码时只会输出提交位置比该LSN大的事务结果。如果输入的LSN比当前流复制槽记录的推进位置还要小，则报错；如果输入的LSN比当前最新物理日志LSN还要大或并未输入，则推进到当前最新物理日志LSN。

    取值范围：字符串（代表十六进制格式表示的uint64，在正中间用'/'分割，左右两边各为一个uint32，如某个uint32为0则显示0），如'1/2AAFC60'、'0/A060'或'3A/0'。为NULL时表示不对解码截止的日志位置做限制。

  返回值类型：name, text

  备注：返回值分别对应slot_name和实际推进到的LSN或CSN。调用该函数的用户需要具有SYSADMIN权限或具有REPLICATION权限或继承了内置角色gs_role_replication的权限。此函数支持在主机调用，或在备机对逻辑复制槽调用。在备机调用会同步推进主机上对应的逻辑复制槽。

  说明

  该函数支持在备机上对逻辑复制槽执行，同步推进主机上对应的逻辑复制槽。在备机上执行该函数，推进主机对应复制槽时需占用主机的一个walsender。由于逻辑解码功能会为每个逻辑复制槽预留walsender，因此正常场景执行该函数会正常推进主机的逻辑复制槽，如果短时间连续执行该函数会导致通知主机推进失败，且没有报错。

- pg_logical_get_area_changes('LSN_start', 'LSN_end', upto_nchanges, 'decoding_plugin', 'xlog_path', 'options_name', 'options_value')

  描述：没有ddl的前提下，指定lsn区间进行解码，或者指定xlog文件进行解码。

说明

约束条件如下：

- 当前网络和硬件环境正常。
- 单条元组大小建议不超过500MB，500MB~1GB之间会报错。
- 不支持数据页复制这种不落xlog的数据找回。
- 调用接口时，要求日志级别wal_level=logical，且只有在wal_level=logical期间产生的日志文件才能被解析，如果使用的xlog文件为非logical级别，则解码内容没有对应的值和类型，无其他影响。如果wal_level未被设置成logical级别，则报错不解码。
- xlog文件只能被完全同构的dn的某个副本解析，且数据库发生没有DDL操作和VACUUM FULL，以确保可以找到数据对应的元信息。
- 需要注意一次不要读入过多xlog文件，指定范围解码不指定文件解码的时候推荐一次一个xlog文件的大小，一般情况下，一个xlog文件解码过程中粗估占用内存为xlog文件大小的2~3倍。
- 不支持VACUUM FULL之前的数据找回。
- 不能解码扩容前的xlog文件。
- 对于update语句解码需要表有主键，否则会导致update语句where里数据为空。
- 此解码方式为根据xlog文本记录数据进行解码，将可解码内容解码出来，不基于事务进行解码，因此不在此xlog里的数据无法解码。
- 从解码点开始，如果未指定解码文件，会先检测从解码开始点到最新redo值之间是否发生ddl，如果发生ddl则全部不解码；如果指定解码文件，会同时检测解码文件的开始点到文件最后可读内容之间，以及数据目录下xlog开始点到最新redo值之间是否发生ddl，检测到一条ddl则对所有表都不解码

备注：打开三权分立时，只有数据库初始用户可以调用；关闭三权分立时，需要具备系统管理员权限。

参数说明：

- LSN_start

  指定开始解码的lsn。

  取值范围：字符串（LSN，格式为xlogid/xrecoff），如'1/2AAFC60'。为NULL时表示不对解码截止的日志位置做限制。

- LSN_end

  指定解码结束的lsn。

  取值范围：字符串（LSN，格式为xlogid/xrecoff），如'1/2AAFC60'。为NULL时表示不对解码截止的日志位置做限制。

- upto_nchanges

  解码条数（包含begin和commit）。假设一共有三条事务，分别包含3、5、7条记录，如果upto_nchanges为4，那么会解码出前两个事务共8条记录。解码完第二条事务时发现解码条数记录大于等于upto_nchanges，会停止解码。

  取值范围：非负整数。

  说明

  LSN和upto_nchanges中任一参数达到限制，解码都会结束。

- decoding_plugin

解码插件，指定解码内容输出格式的so插件。

取值范围：提供mppdb_decoding和sql_decoding两个解码插件。

- xlog_path

解码插件，指定解码文件的xlog绝对路径，文件级别

取值范围：NULL 或者 xlog文件绝对路径的字符串。

- options：此项为可选参数，由一系列options_name和options_value一一对应组成，可以缺省，详见**pg_logical_slot_peek_cha...**。

示例：

```
gaussdb=# CREATE TABLE t1(a int, b int);
CREATE TABLE
gaussdb=# SELECT pg_current_xlog_location();
 pg_current_xlog_location
--------------------------
 0/5ECBCD48
(1 row)

gaussdb=# INSERT INTO t1 VALUES(1,1);
INSERT 0 1
gaussdb=# UPDATE t1 SET b = 2 WHEREa = 1;
UPDATE 1
gaussdb=# DELETE FROM t1;
DELETE 1
gaussdb=# SELECT * FROM pg_logical_get_area_changes('0/5ECBCD48', NULL, NULL, 'sql_decoding',
NULL);
  location  |  xid  |                                  data
------------+-------+-------------------------------------------------------------------------------------
 0/5ECBCD78 | 70718 | insert into public.t1 values (1, 1);
 0/5ECBCEA0 | 70718 | COMMIT 70718 (at 2023-11-01 10:58:51.448885+08) 39319
 0/5ECBCED0 | 70719 | delete from public.t1 where a = 1 and b = 1;insert into public.t1 values (1, 2);
 0/5ECBD028 | 70719 | COMMIT 70719 (at 2023-11-01 10:58:56.487796+08) 39320
 0/5ECBD210 | 70720 | delete from public.t1 where a = 1 and b = 2;
 0/5ECBD338 | 70720 | COMMIT 70720 (at 2023-11-01 10:58:58.856661+08) 39321
(6 rows)

--针对带有生成列的表的场景：
gaussdb=# CREATE TABLE t2(a int, b int generated always as (a + 1) stored);
CREATE TABLE
gaussdb=# SELECT pg_current_xlog_location();
 pg_current_xlog_location
--------------------------
 0/5F62CFE8
(1 row)

gaussdb=# INSERT INTO t2(a) VALUES(1);
INSERT 0 1
gaussdb=# UPDATE t2 SET a = 2 WHERE a = 1;
UPDATE 1
gaussdb=# DELETE FROM t2;
DELETE 1
gaussdb=# SELECT * FROM pg_logical_get_area_changes('0/5F62CFE8', NULL, NULL, 'sql_decoding',
NULL, 'skip-generated-columns', 'on');
  location  |  xid  |                          data
------------+-------+-------------------------------------------------------------------------
 0/5F62D0C8 | 71293 | insert into public.t2 values (1);
 0/5F62D1F0 | 71293 | COMMIT 71293 (at 2023-11-01 11:11:49.452044+08) 39516
 0/5F62D220 | 71294 | delete from public.t2 where a = 1;insert into public.t2 values (2);
 0/5F62D378 | 71294 | COMMIT 71294 (at 2023-11-01 11:11:54.327701+08) 39517
 0/5F62D408 | 71295 | delete from public.t2 where a = 2;
 0/5F62D530 | 71295 | COMMIT 71295 (at 2023-11-01 11:11:58.362057+08) 39518
(6 rows)
```

- pg_get_replication_slots()

  描述：获取复制槽列表。

  返回值类型：text、text、text、oid、boolean、xid、xid、text、boolean、text、xid

  示例：

```
gaussdb=# SELECT * FROM pg_get_replication_slots();
 slot_name |     plugin     | slot_type | datoid | active | xmin | catalog_xmin | restart_lsn |
```



```
dummy_standby | confirmed_flush | confirmed_csn
-----------+----------------+-----------+--------+--------+------+--------------+-------------+---------------
+-----------------+---------------
 dn_6002   |              | physical |      0 | t      |      |              | 0/3622B528  | f           |               |
 dn_6003   |              | physical |      0 | t      |      |              | 0/3622B528  | f           |               |
 slot_lsn  | mppdb_decoding | logical   | 131072 | f      |      |        66658 | 0/36252350  | f           |
0/362523D0      |
 slot_test | mppdb_decoding | logical   | 131072 | f      |      |        66658 | 0/36251718  | f
|               |      10025527
(4 rows)
```

备注：返回值的slot_name代表复制槽名，plugin代表逻辑复制槽对应的输出插件名称，slot_type代表复制槽的类型（physical代表物理复制槽，logical代表逻辑复制槽），datoid代表复制槽所在的数据库OID，active代表复制槽是否为激活状态（f代表未激活，t代表已激活），xmin代表数据库须为复制槽保留的最早事务的事务号，catalog_xmin代表数据库须为逻辑复制槽保留的最早的涉及系统表的事务的事务号，restart_lsn表示复制槽需要的最早xlog的物理位置，dummy_standby是预留参数，confirmed_flush代表客户端确认接收到的日志位置（逻辑复制槽专用），confirmed_csn代表客户端确认接收到的日志中最后一个事务对应的CSN（逻辑复制槽专用）。

须知

在DN上执行查询，LSN序逻辑复制槽的confirmed_csn查询结果为空，CSN序逻辑复制槽的confirmed_flush查询结果为空。

- gs_get_parallel_decode_status()

  描述：监控各个解码线程的读取日志队列和解码结果队列的长度，以便定位并行解码性能瓶颈。

  返回值类型：text、int、text、text、text、int64、int64、TimestampTz

  示例：

```
gaussdb=# SELECT * FROM gs_get_parallel_decode_status();
 slot_name | parallel_decode_num | read_change_queue_length  | decode_change_queue_length |
reader_lsn | working_txn_cnt | working_txn_memory |      decoded_time
-----------+---------------------+---------------------------+----------------------------+------------
+-----------------+--------------------+-----------------------
 slot1     |                   2 | queue0: 1005, queue1: 320 | queue0: 63, queue1: 748    | 0/1DCE2578
|              42 |          192927504 | 2023-01-10 11:18:22+08
(1 row)
```

  备注：返回值的slot_name代表复制槽名，parallel_decode_num代表该复制槽的并行解码线程数，read_change_queue_length列出了每个解码线程读取日志队列的当前长度，decode_change_queue_length列出了每个解码线程解码结果队列的当前长度，reader_lsn表示当前reader线程读取的日志位置，working_txn_cnt表示当前拼接-发送线程中正在拼接的事务个数，working_txn_memory代表拼接-发送线程中拼接事务占用总内存（单位字节），decoded_time代表该复制槽最新解码到的WAL日志时间。

  须知

  decoded_time时间来自检查点日志和事务提交日志，存在一定误差。如果没有解码到任何前述包含时间的日志，则显示“2000-01-01 08:00:00+08”(依照数据库设置的时区而定)。

- gs_get_slot_decoded_wal_time(slot_name)

描述：查看某个执行并行解码的活跃状态的复制槽最新解码的WAL日志时间。

参数说明：

- slot_name：

  要查询的复制槽名称。

  取值范围：字符串，不支持除字母、数字以及（_?-.）以外的字符。

示例：

```
gaussdb=# SELECT * FROM gs_get_slot_decoded_wal_time('replication_slot');
 gs_get_slot_decoded_wal_time
------------------------------
 2023-01-10 11:25:22+08
(1 row)
```

备注：返回一列值代表该复制槽最新解码到的WAL日志时间。

须知

返回的时间来自检查点日志和事务提交日志，存在一定误差。如果没有解码到任何前述包含时间的日志，则显示“2000-01-01 08:00:00+08”(依照数据库设置的时区而定)。查询一个当前不存在的逻辑复制槽的最新解码的WAL日志时间时，返回NULL，在gsql中NULL显示和设置有关，可以使用\pset null 'null'设置。

- gs_logical_parallel_decode_status('slot_name')

  描述：获取执行并行逻辑解码的某一活跃状态的复制槽的解码统计信息，包含26行指标。

  指标定义如下：

  ```
  Record - (stat_id int, stat_name TEXT, value TEXT)
  ```

表 7-57 指标含义

指标名称	说明
slot_name	逻辑解码任务复制槽名称。
reader_lsn	逻辑解码日志位置。
wal_read_total_time	日志模块加载耗时。
wal_wait_total_time	日志模块等待耗时。
parser_total_time	reader线程处理耗时。
decoder_total_time	各decoder线程处理耗时总值。
sender_total_time	sender线程处理耗时。
net_send_total_time	网络发送逻辑日志耗时。
net_wait_total_time	网络等待发送逻辑日志耗时。
net_send_total_bytes	网络发送逻辑日志字节数。
transaction_count	事务数量。
big_transaction_count	大事务数量。

指标名称	说明
max_transaction_tuples	事务操作元组的最大数量。
sent_transaction_count	发送事务数量（本库）。
spill_disk_transaction_count	落盘事务数量。
spill_disk_bytes	累计落盘字节数量，单位：byte。
spill_disk_count	落盘次数。
input_queue_full_count	各decode线程输入队列FULL次数总数。
output_queue_full_count	各decode线程输出队列FULL次数总数。
dml_count	各decode线程解码WAL日志中DML数量（本库）总数。
dml_filtered_count	各decode线程解码过滤WAL日志中DML数量（本库）总数。
toast_count	涉及TOAST表修改行数。
candidate_catalog_xmin	当前逻辑复制槽catalog_xmin候选点
candidate_xmin_lsn	推进catalog_xmin所需要的日志确认接收点
candidate_restart_valid	推进restart_lsn所需要的日志确认接收点
candidate_restart_lsn	当前逻辑复制槽restart_lsn候选点

参数说明：

- slot_name

  流复制槽名称。

  取值范围：字符串，仅支持小写字母、数字以及“_”，“?”，“-”，“.”字符，且不支持“.”或“..”单独作为复制槽名称。

返回值类型：int、text、text

示例：

```
gaussdb=# SELECT * FROM gs_logical_parallel_decode_status('replication_slot');
 stat_id |          stat_name           |      value
---------+------------------------------+------------------
       1 | slot_name                    | replication_slot
       2 | reader_lsn                   | 0/357E180
       3 | wal_read_total_time          | 266694599
       4 | wal_wait_total_time          | 266691307
       5 | parser_total_time            | 39971
       6 | decoder_total_time           | 81216
       7 | sender_total_time            | 48193
       8 | net_send_total_time          | 19388
       9 | net_wait_total_time          | 0
      10 | net_send_total_bytes         | 266897
      11 | transaction_count            | 7
      12 | big_transaction_count        | 1
      13 | max_transaction_tuples       | 4096
      14 | sent_transaction_count       | 7
```

```
    15 | spill_disk_transaction_count | 1
    16 | spill_disk_bytes             | 244653
    17 | spill_disk_count             | 4096
    18 | input_queue_full_count       | 0
    19 | output_queue_full_count      | 0
    20 | dml_count                    | 4097
    21 | dml_filtered_count           | 0
    22 | toast_count                  | 0
    23 | candidate_catalog_xmin       | 17152
    24 | candidate_xmin_lsn           | 0/420A598
    25 | candidate_restart_valid      | 0/420A598
    26 | candidate_restart_lsn        | 0/420A598
(26 rows)
```

备注：按照指标定义，指标应该满足下列约束关系：

wal_read_total_time >= wal_wait_total_time;

transaction_count >= big_transaction_count;

transaction_count >= sent_transaction_count;

transaction_count >= spill_disk_transaction_count;

dml_count >= dml_filtered_count;

dml_count >= toast_count;

如果spill_transaction_count == 0，那么 spill_disk_bytes == 0；

但由于严格保证需要频繁加锁解锁，将对性能造成较大影响，故上面约束关系在极端情况下可能不满足。

transaction_count统计的是所有库的事务数量。

sent_transaction_count统计的是本库的发送事务数量，因为非本库事务将不会被发送。

当传入不存在的slot_name时，函数不报错，返回值为空。

- gs_logical_parallel_decode_reset_status('slot_name')

  描述：重置gs_logical_parallel_decode_status('slot_name')中的指标。

  参数说明：

  – slot_name

    流复制槽名称。

    取值范围：字符串，仅支持小写字母、数字以及“_”，“?”，“-”，“.”字符，且不支持“.”或“..”单独作为复制槽名称。

  返回值类型：text

  示例：

```
gaussdb=# SELECT * FROM gs_logical_parallel_decode_reset_status('replication_slot');
 gs_logical_parallel_decode_reset_status
-----------------------------------------
 OK
(1 row)

gaussdb=# SELECT * FROM gs_logical_parallel_decode_status('replication_slot');
 stat_id |          stat_name           |      value
---------+------------------------------+------------------
       1 | slot_name                    | replication_slot
       2 | reader_lsn                   | 0/357E420
       3 | wal_read_total_time          | 0
       4 | wal_wait_total_time          | 0
       5 | parser_total_time            | 0
       6 | decoder_total_time           | 0
       7 | sender_total_time            | 0
       8 | net_send_total_time          | 0
       9 | net_wait_total_time          | 0
```



```
    10 | net_send_total_bytes         | 0
    11 | transaction_count            | 0
    12 | big_transaction_count        | 0
    13 | max_transaction_tuples       | 0
    14 | sent_transaction_count       | 0
    15 | spill_disk_transaction_count | 0
    16 | spill_disk_bytes             | 0
    17 | spill_disk_count             | 0
    18 | input_queue_full_count       | 0
    19 | output_queue_full_count      | 0
    20 | dml_count                    | 0
    21 | dml_filtered_count           | 0
    22 | toast_count                  | 0
    23 | candidate_catalog_xmin       | 0
    24 | candidate_xmin_lsn           | 0/0
    25 | candidate_restart_valid      | 0/420A598
    26 | candidate_restart_lsn        | 0/420A598
(26 rows)
```

备注：传入不存在的slot_name时，函数不报错，返回值为invalid slot name。

不允许对正在进行observe的复制槽进行reset操作，异常信息按优先级如下：

a. slot_name为空，报错，显示：ERROR: inputString should not be NULL。

b. slot_name不为空，但不存在，不报错，显示：invalid slot name。

c. slot_name不为空，但当前slot_name对应的复制槽正在被采样（observing），不报错，显示can't reset during observing! use gs_logical_decode_stop_observe to stop。

- gs_logical_decode_start_observe('slot_name', window, interval)

  描述：开启某一执行并行解码的活跃状态的逻辑复制槽性能指标采样。

  参数说明：

  – slot_name

    流复制槽名称。

    取值范围：字符串，仅支持小写字母、数字以及“_”，“?”，“-”，“.”字符，且不支持“.”或“..”单独作为复制槽名称。

  – window

    指定采样窗口大小

    取值范围：整数类型,最小2, 最大1024,收集最近interval*window时间内采样数据。

  – interval

    性能监控的间隔，时间间隔类型，秒级。

    取值范围：时间间隔类型，最小1s，最大1min，收集最近interval*window时间内采样数据。

    返回值类型：text

  示例：

```
gaussdb=# SELECT * FROM gs_logical_decode_start_observe('replication_slot',20,5);
 gs_logical_decode_start_observe
---------------------------------
 OK
(1 row)

gaussdb=# SELECT * FROM gs_logical_decode_start_observe('replication_slot',20,5);
 gs_logical_decode_start_observe
---------------------------------
 observe has started!
(1 row)
```

  备注：传入不存在的slot_name时，函数不报错，返回值为invalid slot name。

不允许对已经开启observe的复制槽进行开启observe操作，异常信息按优先级如下：

1. slot_name为空，报错，显示：ERROR: inputString should not be NULL。

2. slot_name不为空，但不存在，不报错，显示：invalid slot name。

3. 若window小于2，则显示：window has to be >= 2。

4. 若window大于1024，则显示：window has to be <= 1024。

5. 若interval小于1S，则显示：sample interval has to be >= 1s。

6. 若interval大于60S，则显示：sample interval has to be <= 60s。

7. slot_name不为空，但当前slot_name对应的复制槽已开启observe，不报错，显示：observe has started!

- gs_logical_decode_stop_observe('slot_name')

  描述：停止逻辑复制性能指标采样。

  参数说明：

  – slot_name

    流复制槽名称。

    取值范围：字符串，仅支持小写字母、数字以及“_”，“?”，“-”，“.”字符，且不支持“.”或“..”单独作为复制槽名称。

  – 返回值类型：text

  示例：

```
gaussdb=# select * from gs_logical_decode_stop_observe('replication_slot');
 gs_logical_decode_stop_observe
--------------------------------
 OK
(1 row)

gaussdb=# select * from gs_logical_decode_stop_observe('replication_slot');
 gs_logical_decode_stop_observe
--------------------------------
 observe not started!
(1 row)
```

  备注：传入不存在的slot_name时，函数不报错，返回值为invalid slot name。

  不允许对已经停止observe的复制槽进行停止observe操作，异常信息按优先级如下：

  1. slot_name为空，报错，显示：ERROR: inputString should not be NULL。

  2. slot_name不为空，但不存在，不报错，显示：invalid slot name。

  3. slot_name不为空，但当前slot_name对应的复制槽已停止observe，不报错，显示：observe not started!

- gs_logical_decode_observe_data('slot_name')

  描述：展示逻辑复制性能指标采样原始数据。

  参数说明：

  – slot_name

    流复制槽名称。

    取值范围：字符串，仅支持小写字母、数字以及“_”，“?”，“-”，“.”字符，且不支持“.”或“..”单独作为复制槽名称。

  – 返回值类型：setof record

  示例：


```
gaussdb=# select * from gs_logical_decode_observe_data('replication_slot');
 slot_name |          sample_time          | reader_lsn | wal_read_total_time | wal_wait_total_time |
parser_total_time | decoder_total_time | sender_total_time | net_send_total_time |
net_send_total_bytes | transaction_count | big_
transaction_count | sent_transaction_count | spill_transaction_count | spill_disk_bytes
-----------+-------------------------------+------------+---------------------+---------------------
+-------------------+--------------------+------------------+--------------------+---------------------
+-------------------+-----
-----------------+-----------------------+-----------------------+------------------
 repl      | 2023-01-12 20:09:40.416798+08 |   49447976 |         776846657 |         776846244
|              65 | {46,11,11,6}       |              56 |               0 |                 0 |                 0 |
               0 |                   0 |                 0 |            0
 repl      | 2023-01-12 20:09:45.416849+08 |   49447976 |         776846657 |         776846244
|              65 | {46,11,11,6}       |              56 |               0 |                 0 |                 0 |
               0 |                   0 |                 0 |            0
 repl      | 2023-01-12 20:09:50.417006+08 |   49447976 |         776846657 |         776846244
|              65 | {46,11,11,6}       |              56 |               0 |                 0 |                 0 |
               0 |                   0 |                 0 |            0
 repl      | 2023-01-12 20:09:55.417057+08 |   49447976 |         776846657 |         776846244
|              65 | {46,11,11,6}       |              56 |               0 |                 0 |                 0 |
               0 |                   0 |                 0 |            0
 repl      | 2023-01-12 20:10:00.417115+08 |   49447976 |         776846657 |         776846244
|              65 | {46,11,11,6}       |              56 |               0 |                 0 |                 0 |
               0 |                   0 |                 0 |            0
 repl      | 2023-01-12 20:10:05.417165+08 |   49447976 |         776846657 |         776846244
|              65 | {46,11,11,6}       |              56 |               0 |                 0 |                 0 |
               0 |                   0 |                 0 |            0
 repl      | 2023-01-12 20:10:10.417217+08 |   49447976 |         776846657 |         776846244
|              65 | {46,11,11,6}       |              56 |               0 |                 0 |                 0 |
               0 |                   0 |                 0 |            0
 repl      | 2023-01-12 20:10:15.417271+08 |   49447976 |         776846657 |         776846244
|              65 | {46,11,11,6}       |              56 |               0 |                 0 |                 0 |
               0 |                   0 |                 0 |            0
 repl      | 2023-01-12 20:10:20.417342+08 |   49448264 |         836085882 |         836085442
|              67 | {46,11,11,8}       |              58 |               0 |                 0 |                 0 |
               0 |                   0 |                 0 |            0
 repl      | 2023-01-12 20:10:25.417433+08 |   49448264 |         836085882 |         836085442
|              67 | {46,11,11,8}       |              58 |               0 |                 0 |                 0 |
               0 |                   0 |                 0 |            0
 repl      | 2023-01-12 20:10:30.417526+08 |   49448264 |         836085882 |         836085442
|              67 | {46,11,11,8}       |              58 |               0 |                 0 |                 0 |
               0 |                   0 |                 0 |            0
 repl      | 2023-01-12 20:10:35.417532+08 |   49448264 |         836085882 |         836085442
|              67 | {46,11,11,8}       |              58 |               0 |                 0 |                 0 |
               0 |                   0 |                 0 |            0
 repl      | 2023-01-12 20:10:40.417644+08 |   49448264 |         836085882 |         836085442
|              67 | {46,11,11,8}       |              58 |               0 |                 0 |                 0 |
               0 |                   0 |                 0 |            0
 repl      | 2023-01-12 20:10:45.417763+08 |   49448264 |         836085882 |         836085442
|              67 | {46,11,11,8}       |              58 |               0 |                 0 |                 0 |
               0 |                   0 |                 0 |            0
(14 rows
```

备注：传入不存在的slot_name时，函数不报错，返回值为空记录。

- gs_logical_decode_observe('slot_name')

  描述：展示逻辑复制性能指标数据。

  参数说明：

  – slot_name

    流复制槽名称。

    取值范围：字符串，仅支持小写字母、数字以及“_”，“?”，“-”，“.”字符，且不支持“.”或“..”单独作为复制槽名称。

  – 返回值类型：setof record

  示例：


```
gaussdb=# select * from gs_logical_decode_observe('replication_slot');
    slot_name     |          sample_time          | logical_decode_rate | wal_read_rate | parser_rate  |
decoder_rate  |  sender_rate  | net_send_rate
------------------+-------------------------------+---------------------+---------------+--------------
+---------------+---------------+---------------
 replication_slot | 2023-01-12 20:16:50.42448+08  | 0.000               | 0.000         | 0.000        |
0.000          | 0.000         | 0.000
 replication_slot | 2023-01-12 20:16:55.424537+08 | 0.000               | 0.000         | 0.000        |
0.000          | 0.000         | 0.000
 replication_slot | 2023-01-12 20:17:00.424641+08 | 0.000               | 0.000         | 0.000        |
0.000          | 0.000         | 0.000
 replication_slot | 2023-01-12 20:17:05.424645+08 | 0.000               | 0.000         | 0.000        |
0.000          | 0.000         | 0.000
 replication_slot | 2023-01-12 20:17:10.424795+08 | 0.000               | 0.000         | 0.000        |
0.000          | 0.000         | 0.000
 replication_slot | 2023-01-12 20:17:15.424848+08 | 0.000               | 0.000         | 0.000        |
0.000          | 0.000         | 0.000
 replication_slot | 2023-01-12 20:17:20.424849+08 | 57.600              | 699029.126    | 96000000.000 |
1152000000.000 | 144000000.000 | 0.000
 replication_slot | 2023-01-12 20:17:25.424959+08 | 0.000               | 699029.126    | 96000000.000 |
1152000000.000 | 144000000.000 | 0.000
 replication_slot | 2023-01-12 20:17:30.42496+08  | 0.000               | 699029.126    | 96000000.000 |
1152000000.000 | 144000000.000 | 0.000
 replication_slot | 2023-01-12 20:17:35.425059+08 | 0.000               | 699029.126    | 96000000.000 |
1152000000.000 | 144000000.000 | 0.000
```

备注：传入不存在的slot_name时，函数不报错，返回值为空记录。若分母为0，则返回最近一次有效数据。若分母不为0，分子为0，则返回0。

指标计算公式：

logical_decode_rate = (reader_lsn1 - reader_lsn2) / (sample_time1 - sample_time2)

wal_read_rate = (reader_lsn1 - reader_lsn2) / (wal_read_total_time1 - wal_read_total_time2) - (wal_wait_total_time1 - wal_wait_total_time2)

parser_rate = (reader_lsn1 - reader_lsn2) / (parser_total_time1 - parser_total_time2)

decoder_rate= (reader_lsn1 - reader_lsn2) / avg(decoder_total_time1[i]-decoder_total_time2[i])

sender_rate = (reader_lsn1 - reader_lsn2) / (sender_total_time1 - sender_total_time2) - (net_send_total_time1 - net_send_total_time2)

sender_rate = (net_sent_bytes1 - net_sent_bytes2) / (net_send_total_time1 - net_send_total_time2) - (net_wait_total_time1 - net_wait_total_time2)

- gs_logical_decode_observe_status('slot_name')

  描述：查询指定逻辑解码任务的监控状态。

  参数说明：

  – slot_name

    流复制槽名称。

    取值范围：字符串，仅支持小写字母、数字以及“_”，“?”，“-”，“.”字符，且不支持“.”或“..”单独作为复制槽名称。

  – 返回值类型：text

  示例：

```
gaussdb=# select * from gs_logical_decode_observe_status('replication_slot');
 gs_logical_decode_observe_status
----------------------------------
 START
(1 row)
```



```
gaussdb=# select * from gs_logical_decode_observe_status('replication_slot');
 gs_logical_decode_observe_status
----------------------------------
 invalid slot name
(1 row)

gaussdb=# select * from gs_logical_decode_stop_observe('replication_slot');
 gs_logical_decode_stop_observe
--------------------------------
 OK
(1 row)

gaussdb=# select * from gs_logical_decode_observe_status('replication_slot');
 gs_logical_decode_observe_status
----------------------------------
 STOP
(1 row)
```

备注：传入不存在的slot_name时，函数不报错，返回值为invalid slot name。

- gs_get_parallel_decode_thread_info()

  描述：返回并行解码的线程信息。

  返回值类型：int64、text、text、int

  示例：

```
gaussdb=# select * from gs_get_parallel_decode_thread_info();
    thread_id    | slot_name | thread_type | seq_number
-----------------+-----------+-------------+------------
 140335364699904 | slot1     | sender      |          1
 140335214098176 | slot1     | reader      |          1
 140335325312768 | slot1     | decoder     |          1
 140335291750144 | slot1     | decoder     |          2
 140335274968832 | slot1     | decoder     |          3
 140335258187520 | slot1     | decoder     |          4
 140335165404928 | slot2     | sender      |          1
 140335022864128 | slot2     | reader      |          1
 140335129818880 | slot2     | decoder     |          1
 140335113037568 | slot2     | decoder     |          2
(10 rows)
```

  备注：返回值thread_id代表线程id，slot_name代表复制槽名，thread_type表示线程种类（共三种，sender代表发送线程、reader代表读取线程、decoder代表解码线程），seq_number代表每个线程在当前复制槽中同种线程的序号。其中sender和reader在每个并行解码连接中均只有一个，因此序号均为1，decoder的序号从1排列到当前复制槽解码并行度。

- pg_replication_origin_create (node_name)

  描述：用给定的外部名称创建一个复制源，并且返回分配给它的内部ID。

  备注：调用该函数的用户需要具有SYSADMIN权限。

  参数说明：

  - node_name

    待创建的复制源的名称。

    取值范围：字符串，不支持除字母、数字以及（_?-.）以外的字符。

  返回值类型：oid

- pg_replication_origin_drop (node_name)

  描述：删除一个以前创建的复制源，包括任何相关的重放进度。

  备注：调用该函数的用户需要具有SYSADMIN权限。

  参数说明：

  - node_name

待删除的复制源的名称。

取值范围：字符串，不支持除字母、数字以及（_?-.）以外的字符。

- pg_replication_origin_oid (node_name)

  描述：根据名称查找复制源并返回内部ID。如果没有发现这样的复制源，则抛出错误。

  备注：调用该函数的用户需要具有SYSADMIN权限。

  参数说明：

  – node_name

    要查找的复制源的名称

    取值范围：字符串，不支持除字母、数字以及（_?-.）以外的字符。

  返回值类型：oid

- pg_replication_origin_session_setup (node_name)

  描述：将当前会话标记为从给定的原点回放，从而允许跟踪回放进度。只能在当前没有选择原点时使用。使用pg_replication_origin_session_reset 命令来撤销。

  备注：调用该函数的用户需要具有SYSADMIN权限。

  参数说明：

  – node_name

    复制源名称。

    取值范围：字符串，不支持除字母、数字以及（_?-.）以外的字符。

- pg_replication_origin_session_reset ()

  描述：取消pg_replication_origin_session_setup()的效果。

  备注：调用该函数的用户需要具有SYSADMIN权限。

- pg_replication_origin_session_is_setup ()

  描述：如果在当前会话中选择了复制源则返回真。

  备注：调用该函数的用户需要具有SYSADMIN权限。

  返回值类型：boolean

- pg_replication_origin_session_progress (flush)

  描述：返回当前会话中选择的复制源的重放位置。

  备注：调用该函数的用户需要具有SYSADMIN权限。

  参数说明：

  – flush

    决定对应的本地事务是否被确保已经刷入磁盘。

    取值范围：boolean

  返回值类型：LSN

- pg_replication_origin_xact_setup (origin_lsn, origin_timestamp)

  描述：将当前事务标记为重放在给定LSN和时间戳上提交的事务。只能在使用pg_replication_origin_session_setup选择复制源时调用。

  备注：调用该函数的用户需要具有SYSADMIN权限。

  参数说明：

  – origin_lsn

    复制源回放位置。

取值范围：LSN

- origin_timestamp

  事务提交时间。

  取值范围：timestamp with time zone

- pg_replication_origin_xact_reset ()

  描述：取消pg_replication_origin_xact_setup()的效果。

  备注：调用该函数的用户需要具有SYSADMIN权限。

- pg_replication_origin_advance (node_name, lsn)

  描述：

  将给定节点的复制进度设置为给定的位置。这主要用于设置初始位置，或在配置更改或类似的变更后设置新位置。

  注意：这个函数的使用不当可能会导致不一致的复制数据。

  备注：调用该函数的用户需要具有SYSADMIN权限。

  参数说明：

  - node_name

    已有复制源名称。

    取值范围：字符串，不支持除字母、数字以及（_?-.）以外的字符。

  - lsn

    复制源回放位置。

    取值范围：LSN

- pg_replication_origin_progress (node_name, flush)

  描述：返回给定复制源的重放位置。

  备注：调用该函数的用户需要具有SYSADMIN权限。

  参数说明：

  - node_name

    复制源名称。

    取值范围：字符串，不支持除字母、数字以及（_?-.）以外的字符。

  - flush

    决定对应的本地事务是否被确保已经刷入磁盘。

    取值范围：boolean

- pg_show_replication_origin_status()

  描述：获取复制源的复制状态。

  备注：调用该函数的用户需要具有SYSADMIN权限。

  返回值类型：

  - local_id：oid，复制源id。
  - external_id：text，复制源名称。
  - remote_lsn：LSN，复制源的lsn位置。
  - local_lsn：LSN，本地的lsn位置。

7.5.25.11 其它函数

- plan_seed()

  描述：获取前一次查询语句的seed值（内部使用）。

  返回值类型：int

- pg_stat_get_env()

  描述：获取当前节点的环境变量信息，仅sysadmin和monitor admin可以访问。

  返回值类型：record

  示例：

```
gaussdb=# select pg_stat_get_env();
                                                  pg_stat_get_env
--------------------------------------------------------------------------------------------------------------------
----------------
 (sgnode,"localhost,XXX.XXX.XXX.XXX",28589,26000,/home/omm,/home/omm/data/
single_node,pg_log)
(1 row)
```

- pg_catalog.plancache_clean()

  描述：清理节点上无人使用的全局计划缓存。

  返回值类型：bool

- pg_catalog.plancache_status()

  描述：显示节点上全局计划缓存的信息，函数返回信息和**GLOBAL_PLANCACHE_STATUS**一致。

  返回值类型：record

- textlen(text)

  描述：提供查询text的逻辑长度的方法。

  返回值类型：int

- threadpool_status()

  描述：显示线程池中工作线程及会话的状态信息。

  返回值类型：record

- get_local_active_session()

  描述：提供当前节点保存在内存中的历史活跃session状态的采样记录。

  返回值类型：record

- pg_stat_get_thread()

  描述：提供当前节点下所有线程的状态信息，sysadmin和monitor admin用户可以查看所有线程信息，普通用户查看本用户的线程信息。

  返回值类型：record

- pg_stat_get_sql_count()

  描述：提供当前节点中用户执行的SELECT/UPDATE/INSERT/DELETE/MERGE INTO语句的计数结果，sysadmin和monitor admin用户可以查看所有用户的信息，普通用户查看本用户的统计信息。

  返回值类型：record

- pg_stat_get_data_senders()

  描述：提供当前活跃的数据复制发送线程的详细信息。

  返回值类型：record

- get_wait_event_info()
  描述：提供wait event事件的具体信息。
  返回值类型：record
- generate_wdr_report(begin_snap_id bigint, end_snap_id bigint, report_type cstring, report_scope cstring, node_name cstring)
  描述：基于两个snapshot生成系统诊断报告。需要在系统库下执行，默认初始化用户或monadmin用户可以访问。只可在系统库中查询到结果，用户库中无法查询。
  返回值类型：record

  表 7-58 generate_wdr_report 参数说明

参数	说明	取值范围
begin_snap_id	生成某段时间内性能诊断报告的开始snapshotid。	-
end_snap_id	结束snapshot的id，默认end_snap_id大于begin_snap_id。	-
report_type	指定生成report的类型。	• summary • detail • all，即同时包含summary 和 detail。
report_scope	指定生成report的范围。	• cluster：数据库级别的信息。 • node：节点级别的信息。
node_name	在report_scope指定为node时，需要把该参数指定为对应节点的名称。（节点名称可以执行select * from pg_node_env;查询）。 在report_scope为cluster时，该值可以省略或者指定为空或NULL。	• cluster：省略/空/NULL • node：GaussDB中的节点名称。

- create_wdr_snapshot()
  描述：手工生成系统诊断快照，该函数需要sysadmin权限。
  返回值类型：text
- kill_snapshot()
  描述：kill后台的WDR snapshot线程，调用该函数的用户需要具有SYSADMIN权限或具有REPLICATION权限或继承了内置角色gs_role_replication的权限。
  返回值类型：void
- capture_view_to_json(text, integer)
  描述：将视图的结果存入GUC: perf_directory所指定的目录，如果is_crossdb为1，则表示对于所有的database都会访问一次view；如果is_crossdb为0，则表示仅对当前database进行一次视图访问。该函数只有sysadmin和monitor admin用户可以执行。

返回值类型：int

- reset_unique_sql(text, text, bigint)

  描述：用来清理数据库节点内存中的Unique SQL（需要sysadmin/monitor admin权限）。

  返回值类型：bool

  表 7-59 reset_unique_sql 参数说明

参数	类型	描述
scope	text	清理范围类型： ● 'GLOBAL'：清理所有的节点，如果是'GLOBAL'，则只可以为主节点执行此函数。 ● 'LOCAL'：清理本节点。
clean_type	text	● 'BY_USERID'：按用户ID来进行清理Unique SQL。 ● 'BY_CNID'：按主节点的ID来进行清理Unique SQL。 ● 'ALL'：全部清理。
clean_value	int8	具体清理type对应的清理值。

- wdr_xdb_query(db_name_str text, query text)

  描述：提供本地跨数据库执行query的能力。例如：在连接到testdb库时，访问test库下的表，只有初始化用户才有权限执行。

  ```
  select col1 from wdr_xdb_query('dbname=test','select col1 from t1') as dd(col1 int);
  ```

  返回值类型：record

- pg_wlm_jump_queue(pid int)

  描述：调整任务到数据库主节点队列的最前端。

  返回值类型：boolean

  – true：成功。
  – false：失败。

- gs_wlm_switch_cgroup(pid int, cgroup text)

  描述：调整作业的优先级到新控制组。

  返回值类型：boolean

  – true：成功。
  – false：失败。

- pv_session_memctx_detail(threadid tid, MemoryContextName text)

描述：将线程tid的MemoryContextName内存上下文信息记录到“$GAUSSLOG/pg_log/${node_name}/dumpmem”目录下的“threadid_timestamp.log”文件中。其中threadid可通过视图GS_SESSION_MEMORY_DETAIL中的sessid后获得。在正式发布的版本中仅接受MemoryContextName为空串（两个单引号表示输入为空串，即''）的输入，此时会记录所有的内存上下文信息，否则不会有任何操作。该函数需要管理员权限的用户才能执行。

返回值类型：boolean

- true：成功。
- false：失败。

- pg_shared_memctx_detail(MemoryContextName text)

  描述：将MemoryContextName内存上下文信息记录到“$GAUSSLOG/pg_log/${node_name}/dumpmem”目录下的“threadid_timestamp.log”文件中。该函数功能仅在DEBUG版本中供内部开发人员和测试人员调试使用，在正式发布版本中调用该函数不会有任何操作。该函数需要管理员权限的用户才能执行。

  返回值类型：boolean

  - true：成功。
  - false：失败。

- local_bgwriter_stat()

  描述：显示本实例的bgwriter线程刷页信息，候选buffer链中页面个数，buffer淘汰信息。

  返回值类型：record

- local_candidate_stat()

  描述：显示本实例的候选buffer链中页面个数，buffer淘汰信息，包含normal buffer pool和segment buffer pool。

  返回值类型：record

- local_ckpt_stat()

  描述：显示本实例的检查点信息和各类日志刷页情况。

  返回值类型：record

- local_double_write_stat()

  描述：显示本实例的双写文件的情况。

  返回值类型：record

表 7-60 local_double_write_stat 参数说明

参数	类型	描述
node_name	text	实例名称。
curr_dwn	int8	当前双写文件的序列号。
curr_start_page	int8	当前双写文件恢复起始页面。
file_trunc_num	int8	当前双写文件复用的次数。
file_reset_num	int8	当前双写文件写满后发生重置的次数。
total_writes	int8	当前双写文件总的I/O次数。

参数	类型	描述
low_threshold_writes	int8	低效率写双写文件的I/O次数（一次I/O刷页数量少于16页面）。
high_threshold_writes	int8	高效率写双写文件的I/O次数（一次I/O刷页数量多于一批，421个页面）。
total_pages	int8	当前刷页到双写文件区的总的页面个数。
low_threshold_pages	int8	低效率刷页的页面个数。
high_threshold_pages	int8	高效率刷页的页面个数。
file_id	int8	当前双写文件的id号。

- local_single_flush_dw_stat()
  描述：显示本实例的单页面淘汰双写文件的情况。
  返回值类型：record
- local_pagewriter_stat()
  描述：显示本实例的刷页信息和检查点信息。
  返回值类型：record
- local_redo_stat()
  描述：显示本实例的备机的当前回放状态。
  返回值类型：record
  备注：返回的回放状态主要包括当前回放位置，回放最小恢复点位置等信息。
- local_recovery_status()
  描述：显示本实例的主机和备机的日志流控信息。
  返回值类型：record
- gs_wlm_node_recover(boolean isForce)
  描述：获取当前内存中记录的TopSQL查询语句级别相关统计信息，当传入的参数不为0时，会将这部分信息从内存中清理掉。
  返回值类型：record
- gs_cgroup_map_ng_conf(group name)
  描述：读取指定逻辑数据库的cgroup配置文件。
  返回值类型：record
- gs_wlm_switch_cgroup(sess_id int8, cgroup name)
  描述：切换指定会话的控制组。
  返回值类型：record
- comm_client_info()
  描述：用于查询单个节点活跃的客户端连接信息。
  返回值类型：setof record
- pg_get_flush_lsn()

描述：返回当前节点flush的xlog位置。

返回值类型：text

- pg_get_sync_flush_lsn()

  描述：返回当前节点多数派flush的xlog位置。

  返回值类型：text

- dbe_perf.get_global_full_sql_by_timestamp(start_timestamp timestamp with time zone, end_timestamp timestamp with time zone)

  描述：获取数据库级的全量SQL(Full SQL)信息。只可在系统库中查询到结果，用户库中无法查询。

  返回值类型：record

  表 7-61 dbe_perf.get_global_full_sql_by_timestamp 参数说明

参数	类型	描述
start_timestamp	timestamp with time zone	SQL启动时间范围的开始时间点。
end_timestamp	timestamp with time zone	SQL启动时间范围的结束时间点。若start_timestamp>=end_timestamp，则函数执行报错。

- dbe_perf.get_global_slow_sql_by_timestamp(start_timestamp timestamp with time zone, end_timestamp timestamp with time zone)

  描述：获取数据库级的慢SQL(Slow SQL)信息。只可在系统库中查询到结果，用户库中无法查询。

  返回值类型：record

  表 7-62 dbe_perf.get_global_slow_sql_by_timestamp 参数说明

参数	类型	描述
start_timestamp	timestamp with time zone	SQL启动时间范围的开始时间点。
end_timestamp	timestamp with time zone	SQL启动时间范围的结束时间点。若start_timestamp>=end_timestamp，则函数执行报错。

- statement_detail_decode(detail text, format text, pretty boolean)

  描述：解析全量/慢SQL语句中的details字段的信息。只可在系统库中查询到结果，用户库中无法查询。

  返回值类型：text

  表 7-63 statement_detail_decode 参数说明

参数	类型	描述
detail	text	SQL语句产生的事件的集合（不可读）。

参数	类型	描述
format	text	解析输出格式，取值为 plaintext。
pretty	boolean	当format为plaintext时，是否以优雅的格式展示： ● true：表示通过“\n”分隔事件。 ● false：表示通过“，”分隔事件。

- pg_control_system()
  描述：返回系统控制文件状态。
  返回类型：SETOF record
- pg_control_checkpoint()
  描述：返回系统检查点状态。
  返回类型：SETOF record
- get_prepared_pending_xid()
  描述：当恢复完成时，返回nextxid。
  参数：nan
  返回值类型：text
- pg_clean_region_info()
  描述：清理regionmap。
  参数：nan
  返回值类型：character varying
- pg_get_replication_slot_name()
  描述：获取slot name。
  参数：nan
  返回值类型：text
- pg_get_running_xacts()
  描述：获取运行中的xact。
  参数：nan
  返回值类型：handle integer、gxid xid、state tinyint、node text、xmin xid、vacuum boolean、timeline bigint、prepare_xid xid、pid bigint、next_xid xid

  表 7-64 pg_get_running_xacts 返回参数说明

名称	类型	描述
handle	integer	事务在GTM对应的句柄。
gxid	xid	事务id号。
state	tinyint	事务状态。 ● 3：prepared。 ● 0：starting。

名称	类型	描述
node	text	节点名称。
xmin	xid	节点上当前数据涉及的最小事务号xmin。
vacuum	boolean	表示当前事务是否是lazy vacuum事务。 ● t（true）：表示是。 ● f（false）：表示否。
timeline	bigint	数据库重启次数。
prepare_xid	xid	处于prepared状态事务的id号，若不在prepared状态，值为0。
pid	bigint	事务对应的线程id。
next_xid	xid	本地活跃事务最小CSN值。

- pg_get_variable_info()
  描述：获取共享内存变量cache。
  参数：nan
  返回值类型：node_name text、nextOid oid、nextXid xid、oldestXid xid、xidVacLimit xid、oldestXidDB oid、lastExtendCSNLogpage xid、startExtendCSNLogpage xid、nextCommitSeqNo xid、latestCompletedXid xid、startupMaxXid xid
- pg_get_xidlimit()
  描述：从共享内存获取事务id信息。
  参数：nan
  返回值类型：nextXid xid、oldestXid xid、xidVacLimit xid、xidWarnLimit xid、xidStopLimit xid、xidWrapLimit xid、oldestXidDB oid
- pg_relation_compression_ratio()
  描述：查询表压缩率，默认返回1.0。
  参数：text
  返回值类型：real
- pg_relation_with_compression()
  描述：查询表是否压缩。
  参数：text
  返回值类型：boolean
- pg_stat_file_recursive()
  描述：列出路径下所有文件。
  参数：location text
  返回值类型：path text、filename text、size bigint、isdir boolean
- pg_stat_get_activity_for_temptable()
  描述：返回临时表相关的后台进程的记录。
  参数：nan

返回值类型：datid oid、timelineid integer、tempid integer、sessionid bigint

- pg_stat_get_activity_ng()
  描述：返回nodegroup相关的后台进程的记录。
  参数：pid bigint
  返回值类型：datid oid、pid bigint、sessionid bigint、node_group text
- pg_stat_get_cgroup_info()
  描述：返回cgroup信息。
  参数：nan
  返回值类型：cgroup_name text、percent integer、usage_percent integer、shares bigint、usage bigint、cpuset text、relpath text、valid text、node_group text
- pg_stat_get_realtime_info_internal()
  描述：返回实时信息，当前该接口已不可用，返回FailedToGetSessionInfo。
  参数：oid、oid, bigint、cstring、oid
  返回值类型：text
- pg_test_err_contain_err()
  描述：测试错误类型和返回信息。
  参数：integer
  返回值类型：void
- get_global_user_transaction()
  描述：返回所有节点上各用户的事务相关信息。
  返回值类型：node_name name、usename name、commit_counter bigint、rollback_counter bigint、resp_min bigint、resp_max bigint、resp_avg bigint、resp_total bigint、bg_commit_counter bigint、bg_rollback_counter bigint、bg_resp_min bigint、bg_resp_max bigint、bg_resp_avg bigint、bg_resp_total bigint
- pg_collation_for()
  描述：返回入参字符串对应的排序规则。
  参数：any（如果是常量必须进行显式类型转换）
  返回值类型：text
- pgxc_unlock_for_sp_database(name Name)
  该函数当前版本暂不可用。
- pgxc_lock_for_sp_database(name Name)
  该函数当前版本暂不可用。
- copy_error_log_create()
  描述：创建COPY FROM容错机制所需要的错误表（public.pgxc_copy_error_log）。
  返回值类型：Boolean

说明

- 此函数会尝试创建public.pgxc_copy_error_log表，表的详细信息请参见**表7-65**。
- 在relname列上创建B-tree索引，并REVOKE ALL on public.pgxc_copy_error_log FROM public对错误表进行权限控制（与COPY语句权限一致）。
- 由于尝试创建的public.pgxc_copy_error_log定义是一张行存表，因此数据库实例上必须支持行存表的创建才能够正常运行此函数，并使用后续的COPY容错功能。需要特别注意的是，enable_hadoop_env这个GUC参数开启后会禁止在数据库实例内创建行存表（GaussDB默认为off）。
- 此函数自身权限为Sysadmin及以上（与错误表、COPY权限一致）。
- 若创建前public.pgxc_copy_error_log表已存在或者copy_error_log_relname_idx索引已存在，则此函数会报错回滚。

表 7-65 错误表 public.pgxc_copy_error_log 信息

列名称	类型	描述
relname	character varying	表名称。以模式名.表名形式显示。
begintime	timestamp with time zone	出现数据格式错误的时间。
filename	character varying	出现数据格式错误的数据源文件名。
lineno	bigint	在数据源文件中，出现数据格式错误的行号。
rawrecord	text	在数据源文件中，出现数据格式错误的原始记录。
detail	text	详细错误信息。

- dynamic_func_control(scope text, function_name text, action text, "{params}" text[])

  描述：动态开启内置的功能，当前仅支持动态开启全量SQL。

  返回值类型：record

表 7-66 dynamic_func_control 参数说明

参数	类型	描述
scope	text	动态开启功能的范围，当前仅支持'LOCAL'。
function_name	text	功能的名称，当前仅支持'STMT'。

参数	类型	描述
action	text	当function_name为'STMT'时，action仅支持TRACK/UNTRACK/LIST/CLEAN： ● TRACK - 开始记录归一化SQL的全量SQL信息。 ● UNTRACK - 取消记录归一化SQL的全量SQL信息。 ● LIST - 列取当前TRACK的归一化SQL的信息。 ● CLEAN - 清理记录当前归一化SQL的信息。
params	text[]	当function_name为'STMT'时，对应不同的action时，对应的params设置如下： ● TRACK - '{"归一化SQLID", "L0/L1/L2"}' ● UNTRACK - '{"归一化SQLID"}' ● LIST - '{}' ● CLEAN - '{}'

- gs_parse_page_bypath(path text, blocknum bigint, relation_type text, read_memory boolean)

  描述：用于解析指定表页面，并返回存放解析内容的路径。

  返回值类型：text

  备注：必须是系统管理员或运维管理员才能执行此函数。

表 7-67 gs_parse_page_bypath 参数说明

参数	类型	描述
path	text	• 对于普通表，相对路径为：tablespace name/database oid/表的relfilenode(物理文件名)。例如：base/16603/16394。 • 对于普通表的visibility map，相对路径为：tablespace name/database oid/表的vm文件。例如：base/16603/16394_vm。 • 对于clog文件，相对路径为：pg_clog目录下的clog文件。例如：000000000000。 • 对于csnlog文件，相对路径为：pg_csnlog目录下的csnlog文件。例如：000000000000。 • 对于undo record文件，相对路径为：undo目录下的undo/UNDOPERSISTENCE/zondid.segno。例如：undo/permanent/00000.0000009。 • 对于undo meta文件，相对路径为：undo目录下的undo/UNDOPERSISTENCE/zondid.meta.segno。例如：undo/permanent/00000.meta.0000004。 • 表文件的相对路径可以通过pg_relation_filepath(table_name text)查找。分区表的路径可以查看pg_partition系统表和调用pg_partition_filepath(partition_oid)。 • 合法的path格式列举： – global/relNode – base/dbNode/relNode – pg_tblspc/spcNode/version_dir/dbNode/relNode
blocknum	bigint	• -1：所有block的信息（强制从磁盘解析）。 • 0~MaxBlockNumber：对应block的信息。 • 对于BTree/UBTree索引，block 0为索引元页面。 • 对于undo record文件，采用逻辑块号，对应文件block大于等于segno*128，小于(segno+1)*128。 • 对于undo meta文件，采用逻辑块号，对应文件block大于等于segno*4，小于（segno+1）*4。

参数	类型	描述
relation_type	text	• heap（astore 表）。 • uheap（ustore 表）。 • btree（BTree 索引）。 • ubtree（UBTree 索引）。 • fsm (astore/ustore堆表空闲空间)。 • segment（段页式，预留参数，暂不支持）。 • vm（astore普通表的visibility map）。 • clog（事务状态日志commit log）。 • csnlog（快照时间戳日志commit sequence number log）。 • undo_slot（事务槽信息）。 • undo_record（undo 记录信息）。
read_memory	boolean	• false：从磁盘文件解析。 • true：首先尝试从共享缓冲区中解析该页面；如果共享缓冲区中不存在，则从磁盘文件解析。

正常使用示例：

```
# 解析BTree索引文件中所有页面的信息内容
# 函数调用前请依据参数说明中的内容确保文件路径真实存在
gaussdb=# select gs_parse_page_bypath('base/16603/16394', -1, 'btree', false);
                      gs_parse_page_bypath
-----------------------------------------------------------------
 /data_dir/1663_16603_16394_-1.page
(1 row)

# 解析vm文件中所有block的可见性结果
gaussdb=# select gs_parse_page_bypath('base/12828/16771_vm', -1, 'vm', false);
                       gs_parse_page_bypath
--------------------------------------------------------------------
 /data_dir/1663_12828_16771_-1_vm.page
(1 row)

# 解析clog文件中0号block的commit log日志
gaussdb=# select gs_parse_page_bypath('000000000000', 0, 'clog', false);
                      gs_parse_page_bypath
-----------------------------------------------------------------
 /data_dir/000000000000.clog
(1 row)
```

异常场景报错示例：

```
# 入参blocknum超出取值范围时报错
gaussdb=# select gs_parse_page_bypath('base/12828/16777', -10, 'heap', false);
ERROR:  Blocknum should be between -1 and 4294967294.
CONTEXT:  referenced column: gs_parse_page_bypath
```

- gs_xlogdump_lsn(start_lsn text, end_lsn text)

  描述：用于解析指定lsn范围之内的XLOG日志，并返回存放解析内容的路径。可以通过pg_current_xlog_location()获取当前XLOG位置。

  返回值类型：text

参数：LSN起始位置，LSN结束位置

备注：必须是系统管理员或运维管理员才能执行此函数。

- gs_xlogdump_xid(c_xid xid)

  描述：用于解析指定xid的XLOG日志，并返回存放解析内容的路径。可以通过txid_current()获取当前事务ID。

  参数：事务ID

  返回值类型：text

  备注：必须是系统管理员或运维管理员才能执行此函数。

- gs_xlogdump_tablepath(path text, blocknum bigint, relation_type text)

  描述：用于解析指定表页面对应的日志，并返回存放解析内容的路径。

  返回值类型：text

  备注：必须是系统管理员或运维管理员才能执行此函数。

  表 7-68 gs_xlogdump_tablepath 参数说明

参数	类型	描述
path	text	• 对于普通表，相对路径为：tablespace name/database oid/表的relfilenode(物理文件名)。例如：base/16603/16394。 • 表文件的相对路径可以通过pg_relation_filepath(table_name text)查找。分区表的路径可以查看pg_partition系统表和调用pg_partition_filepath(partition_oid)。 • 合法的path格式列举： – global/relNode – base/dbNode/relNode – pg_tblspc/spcNode/version_dir/dbNode/relNode
blocknum	bigint	• -1：所有block的信息（强制从磁盘解析）。 • 0~MaxBlockNumber：对应block的信息。
relation_type	text	• heap(astore 表) • uheap(ustore 表) • btree(BTree 索引) • ubtree(UBTree 索引) • segment(段页式，预留参数，暂不支持)

- gs_xlogdump_parsepage_tablepath(path text, blocknum bigint, relation_type text, read_memory boolean)

  描述：用于解析指定表页面和表页面对应的日志，并返回存放解析内容的路径。可以看做一次执行gs_parse_page_bypath和gs_xlogdump_tablepath。该函数执行的前置条件是表文件存在。如果想查看已删除的表的相关日志，请直接调用gs_xlogdump_tablepath。

  返回值类型：text

备注：必须是系统管理员或运维管理员才能执行此函数。

表 7-69 gs_xlogdump_parsepage_tablepath 参数说明

参数	类型	描述
path	text	● 对于普通表，相对路径为：tablespace name/database oid/表的relfilenode(物理文件名)；例如：base/16603/16394 ● 表文件的相对路径可以通过pg_relation_filepath(table_name text)查找。分区表的路径可以查看pg_partition系统表和调用pg_partition_filepath(partition_oid)。 ● 合法的path格式列举： – global/relNode – base/dbNode/relNode – pg_tblspc/spcNode/version_dir/dbNode/relNode
blocknum	bigint	● -1：所有block的信息（强制从磁盘解析）。 ● 0~MaxBlockNumber：对应block的信息。
relation_type	text	● heap(astore 表) ● uheap(ustore 表) ● btree(BTree 索引) ● ubtree(UBTree 索引) ● segment(段页式，预留参数，暂不支持)
read_memory	boolean	● false：从磁盘文件解析。 ● true：首先尝试从共享缓冲区中解析该页面；如果共享缓冲区中不存在，则从磁盘文件解析。

- gs_index_verify(Oid oid, uint32 blkno)
  描述：用于校验UBtree索引页面或者索引树上key的顺序是否正确。
  返回值类型：record

  表 7-70 gs_index_verify 参数说明

参数	类型	描述
oid	Oid	● 索引文件relfilenode，可以通过select relfilenode from pg_class where relname='name'查询，其中name表示对应的索引文件名字。

参数	类型	描述
blkno	uint32	● 0 ： 表示检验整个索引树上所有页面。 ● 大于0：表示校验页面编码等于blkno的索引页面。

- gs_index_recycle_queue(Oid oid, int type, uint32 blkno)
  描述：用于解析UBtree索引回收队列信息。
  返回值类型：record

  表 7-71 gs_index_recycle_queue 参数说明

参数	类型	描述
oid	Oid	● 索引文件relfilenode，可以通过select relfilenode from pg_class where relname='name'查询，其中name表示对应的索引文件名字。
type	int	● 0：表示解析整个待回收队列。 ● 1：表示解析整个空页队列。 ● 2：表示解析单个页面。
blkno	uint32	● 回收队列页面编号，该参数只有在type=2的时候有效，blkno有效取值范围为1~4294967294。

- gs_stat_wal_entrytable(int64 idx)
  描述：用于输出xlog中预写日志插入状态表的内容。
  返回值类型：record

  表 7-72 gs_stat_wal_entrytable 参数说明

参数类型	参数名	类型	描述
输入参数	idx	int64	● -1：查询数组所有元素。 ● 0-最大值：具体某个数组元素内容。
输出参数	idx	uint64	记录对应数组中的下标。
输出参数	endlsn	uint64	记录的LSN标签。
输出参数	lrc	int32	记录对应的LRC。

参数类型	参数名	类型	描述
输出参数	status	uint32	标识当前entry对应的xlog是否已经完全拷贝到wal buffer中： ● 0：非COPIED ● 1：COPIED

- gs_walwriter_flush_position()
  描述：输出预写日志的刷新位置。
  返回值类型：record

  表 7-73 gs_walwriter_flush_position 参数说明

参数类型	参数名	类型	描述
输出参数	last_flush_status_entry	int32	Xlog flush上一个刷盘的tblEntry下标索引。
输出参数	last_scanned_lrc	int32	Xlog flush上一次扫描到的最后一个tblEntry记录的LRC。
输出参数	curr_lrc	int32	WALInsertStatusEntry状态表中LRC最新的使用情况，该LRC表示下一个Xlog记录写入时在WALInsertStatusEntry对应的LRC值。
输出参数	curr_byte_pos	uint64	Xlog记录写入WAL文件，最新分配的位置，下一个xlog记录插入点。
输出参数	prev_byte_size	uint32	上一个xlog记录的长度。
输出参数	flush_result	uint64	当前全局xlog刷盘的位置。
输出参数	send_result	uint64	当前主机上xlog发送位置。
输出参数	shm_rqst_write_pos	uint64	共享内存中记录的XLogCtl中LogwrtRqst请求的write位置。
输出参数	shm_rqst_flush_pos	uint64	共享内存中记录的XLogCtl中LogwrtRqst请求的flush位置。
输出参数	shm_result_write_pos	uint64	共享内存中记录的XLogCtl中LogwrtResult的write位置。
输出参数	shm_result_flush_pos	uint64	共享内存中记录的XLogCtl中LogwrtResult的flush位置。
输出参数	curr_time	text	当前时间。

- gs_walwriter_flush_stat(int operation)

  描述：用于统计预写日志write与sync的次数频率与数据量，以及xlog文件的信息。

  返回值类型：record

  表 7-74 gs_walwriter_flush_stat 参数说明

参数类型	参数名	类型	描述
输入参数	operation	int	● -1：关闭统计开关(默认状态为关闭)。 ● 0：打开统计开关。 ● 1：查询统计信息。 ● 2：重置统计信息。
输出参数	write_times	uint64	Xlog调用write接口的次数。
输出参数	sync_times	uint64	Xlog调用sync接口次数。
输出参数	total_xlog_sync_bytes	uint64	Backend线程请求写入xlog总量统计值。
输出参数	total_actual_xlog_sync_bytes	uint64	调用sync接口实际刷盘的xlog总量统计值。
输出参数	avg_write_bytes	uint32	每次调用XLogWrite接口请求写的xlog量。
输出参数	avg_actual_write_bytes	uint32	实际每次调用write接口写的xlog量。
输出参数	avg_sync_bytes	uint32	平均每次请求sync的xlog量。
输出参数	avg_actual_sync_bytes	uint32	实际每次调用sync刷盘xlog量。
输出参数	total_write_time	uint64	调用write操作总时间统计(单位：us)。
输出参数	total_sync_time	uint64	调用sync操作总时间统计(单位：us)。
输出参数	avg_write_time	uint32	每次调用write接口平均时间(单位：us)。
输出参数	avg_sync_time	uint32	每次调用sync接口平均时间(单位：us)。
输出参数	curr_init_xlog_segno	uint64	当前最新创建的xlog段文件编号。

参数类型	参数名	类型	描述
输出参数	curr_open_xlog_segno	uint64	当前正在写的xlog段文件编号。
输出参数	last_reset_time	text	上一次重置统计信息的时间。
输出参数	curr_time	text	当前时间。

- gs_catalog_attribute_records()

  描述：对于指定的系统表oid，返回该系统表对应的各个字段的定义。仅支持oid小于10000的普通系统表（不支持索引、toast表等）。

  参数：系统表oid

  返回值类型：record

- gs_comm_proxy_thread_status()

  描述：用于在数据库实例配置用户态网络的场景下，代理通信库comm_proxy收发数据包统计。

  参数：nan

  返回值类型：record

  说明

  此函数的查询仅在集中式环境开始部署用户态网络，且comm_proxy_attr参数中enable_dfx配置为true的条件下显示具体信息。其他场景报错不支持查询。

- pg_ls_tmpdir()

  描述：返回默认表空间下临时目录（pgsql_tmp）中每个文件的名称、大小和最后修改时间。

  参数：nan

  返回值类型：record

  备注：必须是系统管理员或者监控管理员才能执行此函数。

参数类型	参数名	类型	描述
输出参数	name	text	文件名称。
输出参数	size	int8	文件大小（单位：byte）。
输出参数	modification	timestamptz	文件最后修改时间。

- pg_ls_tmpdir(oid)

  描述：返回指定表空间下临时目录（pgsql_tmp）中每个文件的名称、大小和最后修改时间。

  参数：oid

  返回值类型：record

备注：必须是系统管理员或者监控管理员才能执行此函数。

参数类型	参数名	类型	描述
输入参数	oid	oid	表空间id。
输出参数	name	text	文件名称。
输出参数	size	int8	文件大小（单位：byte）。
输出参数	modification	timestamptz	文件最后修改时间。

- pg_ls_waldir()

  描述：返回预写日志(WAL)目录中每个文件的名称、大小和最后修改时间。

  参数：nan

  返回值类型：record

  备注：必须是系统管理员或者监控管理员才能执行此函数。

参数类型	参数名	类型	描述
输出参数	name	text	文件名称。
输出参数	size	int8	文件大小（单位：byte）。
输出参数	modification	timestamptz	文件最后修改时间。

- gs_write_term_log(void)

  描述：写入一条日志记录DN节点当前的term值。备DN节点返回false，主DN节点写入成功后返回true。

  返回值类型：Boolean

- gs_stat_space(bool init)

  描述：用于查询UStore中做Insert操作时拓展页面的状态。

  返回值类型：record

参数类型	参数名	类型	描述
输入参数	init	bool	是否重置已统计的数据。
输出参数	access_func	int8	relation_get_buffer_for_utuple接口访问总次数。
输出参数	cache_blk	int8	relation_get_buffer_for_utuple获取缓存次数。
输出参数	cache_succ	int8	relation_get_buffer_for_utuple获取缓存成功次数。

参数类型	参数名	类型	描述
输出参数	nblk_first	int8	relation_get_buffer_for_utuple中第一次获取nblocks-1次数。
输出参数	nblk_first_succ	int8	relation_get_buffer_for_utuple中第一次获取nblocks-1成功次数。
输出参数	nblk_second	int8	relation_get_buffer_for_utuple中第二次获取nblocks-1次数。
输出参数	nblk_second_succ	int8	relation_get_buffer_for_utuple中第二次获取nblocks-1成功次数。
输出参数	fsm_first	int8	第一次访问FSM次数。
输出参数	fsm_first_success	int8	第一次访问FSM成功次数。
输出参数	fsm_rewrite	int8	FSM回写次数。
输出参数	fsm_second	int8	第二次访问FSM次数。
输出参数	fsm_second_sucess	int8	第二次访问FSM成功次数。
输出参数	prune_count	int8	relation_get_buffer_for_utuple中Prune次数。
输出参数	prune_space	int8	relation_get_buffer_for_utuple中Prune总空间。
输出参数	coprune_count	int8	联合清理执行次数。
输出参数	coprune_scan_blocks	int8	联合清理扫描总页数。
输出参数	coprune_prune_count	int8	联合清理Prune次数。
输出参数	coprune_prune_space	int8	联合清理Prune总空间。
输出参数	con_extend_count	int8	并发扩页数量。
输出参数	con_extend_time	int8	并发扩页总时间。
输出参数	single_extend_time	int8	单一扩页次数。

示例：

```
gaussdb=# select * from gs_stat_space(false);
 access_func | cache_blk | cache_succ | nblk_first | nblk_first_succ | nblk_sencond | nblk_sencond_succ |
fsm_first | fsm_first_success | fsm_rewrite | fsm_
second | fsm_second_success | prune_count | prune_space | coprune_count | coprune_scan_blocks |
coprune_prune_count | coprune_prune_space | con_extend_count
 | con_extend_time | single_extend_count
-------------+-----------+------------+------------+-----------------+--------------+-------------------+-----------
+-------------------+-------------+-----
-------+--------------------+-------------+-------------+---------------+--------------------+--------------------
+--------------------+-----------------+-----------------
-+-----------------+--------------------
   10082397 |  10082397 |   10082235 |        88 |            78 |       47021 |            47021 |       162
|              74 |      360996 |
360996 |           270948 |       6711 |          0 |        45497 |            222619 |                0
|                 0 |            3675
 |       25542884 |             26791
(1 row)
```

备注：请重点关注cache_succ此值较小说明系统缓存失效，prune_space 此值较小表示UStore数据页页面清理机制可能存在问题，con_extend_time此值过高可能表明UStore并发拓页时耗时较高。

- gs_index_dump_read(int8 reset, text out_type)

  描述：用于查询索引获取新页面时在循环队列中产生的buffer read信息和索引页面相同key从左到右遍历叶子页面的buffer read信息。

  返回值类型：record

参数类型	参数名	类型	描述
输入参数	reset	int8	● 0：将统计信息reset为初始值0，重新开始统计。 ● 1：直接show出当前的统计信息。
输入参数	out_type	text	● urq：输出循环队列的统计信息。 ● ubtree：输出索引页的统计信息。 ● all：循环队列和索引页的统计信息全部输出。
输出参数	relfilenode	oid	统计到的最大buffer read值对应的索引relfilenode。
输出参数	max_count	int8	最大buffer read值。
输出参数	ave_count	int8	平均buffer read值。

说明

- 该接口当前仅支持USTORE索引表。
- 该接口执行时，先reset清理、将记录全部置为0。再进行查询，直到下次统计到信息值之前，查询记录会一直为0。查询示例如下：

```
gaussdb=# select * from gs_index_dump_read(0, 'all');
relfilenode | max_count | ave_count
--------------+-------------+-------------
|                  |
(1 row)
gaussdb=# select * from gs_index_dump_read(1, 'all');
relfilenode | max_count | ave_count
--------------+-------------+-------------
0 |                0 |                0
0 |                0 |                0
(2 rows)
```

- gs_redo_upage(directory_path text, backup_path text, blocknum bigint, relation_type text, xlog_path text, lsn text)

  描述：用于将备份的特定UStore数据页面重放到指定LSN，并在重放期间校验页面，若检测到页面受损或页面记录受损则告警受损信息，并在重放后落盘页面，返回落盘路径、页面LSN以及受损信息，否则重放至指定LSN并落盘页面后返回，必须是系统管理员或运维管理员才能执行此函数。

  返回值类型：record

参数类型	参数	类型	描述
输入参数	directory_path	text	备份文件所在目录。
输入参数	backup_path	text	备份表文件的相对路径，与备份文件所在目录拼成表文件的完整路径，例如，base/15635/12488，若备份文件不存在，传入null。
输入参数	blocknum	bigint	0~MaxBlockNumber：对应页面的块号。
输入参数	relation_type	text	• uheap（Ustore数据页）。 • ubtree（Ustore索引页）。 • indexurq（Urq页面）。 • undo_record（Undo record页面）。 • undo_slot（Transaction slot页面）。
输入参数	xlog_path	text	归档日志目录的绝对路径。
输入参数	lsn	text	lsn由两个16进制数(各32 bits)组成，中间通过"/"隔开，例如：2/962D1DF8。如果是0则重放到最新版本。
输出参数	output_filename	text	文件的落盘路径与文件名。

参数类型	参数	类型	描述
输出参数	output_lsn	text	最后一次页面重放的LSN。
输出参数	corruption_desc	text	页面受损情况描述。

- dbe_perf.get_full_sql_by_parent_id_and_timestamp(parent_id bigint, start_timestamp timestamp with time zone, end_timestamp timestamp with time zone)

  描述：根据parent_id获取某个时间段内，执行某个存储过程的数据库级的全量SQL及其子语句的记录。只可在系统库中查询到结果，用户库中无法查询。

  返回值类型：record

参数	类型	描述
parent_id	bigint	指定需要检索的存储过程的调用语句的unique_sql_id。
start_timestamp	timestamp with time zone	SQL启动时间范围的开始时间点。
end_timestamp	timestamp with time zone	SQL启动时间范围的结束时间点。

  示例：

```
gaussdb=# CREATE TABLE test(a int,b int);
CREATE TABLE
gaussdb=# INSERT INTO test values(1,1);
INSERT 0 1
gaussdb=# CREATE PROCEDURE mypro1() as num int;
gaussdb$# begin
gaussdb$# INSERT INTO test values(2,2);
gaussdb$# DELETE FROM test where a = 2;
gaussdb$# end;
gaussdb$# /
CREATE PROCEDURE

打开参数，跟踪存储过程子语句。
gaussdb=# SET instr_unique_sql_track_type = 'all';
SET

打开参数，dbe_perf.statement_history表生成全量语句记录。
gaussdb=# SET track_stmt_stat_level = 'L0,L0';
SET

gaussdb=# CALL mypro1();
 mypro1
--------

(1 row)

gaussdb=# SET track_stmt_stat_level = 'off,L0';
SET

gaussdb=# SET instr_unique_sql_track_type = 'top';
```



```
SET

查询关键信息，作为函数参数使用。
gaussdb=# SELECT query,unique_query_id,start_time,finish_time FROM dbe_perf.statement_history;
                query                | unique_query_id |          start_time           |          finish_time
-------------------------------------+-----------------+-------------------------------+-------------------------------
+-------------------------------
 set track_stmt_stat_level = 'L0,L0'; |       636388010 | 2023-06-02 17:40:49.176155+08 | 2023-06-02
17:40:49.176543+08
 call mypro1();                       |       536458473 | 2023-06-02 17:40:59.028144+08 | 2023-06-02
17:40:59.032027+08
 delete from test where a = ?         |       583323884 | 2023-06-02 17:40:59.029955+08 | 2023-06-02
17:40:59.031577+08
 insert into test values(?,?)         |       769279931 | 2023-06-02 17:40:59.029219+08 | 2023-06-02
17:40:59.029947+08
(4 rows)

通过外层语句的unique_query_id，开始时间的起止时间作为参数，查询该时间段内，指定存储过程及其子
语句的信息。
gaussdb=# SELECT query FROM
dbe_perf.get_full_sql_by_parent_id_and_timestamp(536458473,'2023-06-02
17:40:59.028144+08','2023-06-02 17:40:59.032027+08');
            query
------------------------------
 call mypro1();
 delete from test where a = ?
 insert into test values(?,?)
(3 rows)

删除表
gaussdb=# DROP TABLE test;
DROP TABLE

删除存储过程
gaussdb=# DROP PROCEDURE mypro1;
DROP PROCEDURE
```

- gs_xlogdump_bylastlsn(last_lsn text, blocknum bigint, relation_type text)

  描述：传入一个页面LSN以及块号，解析LSN对应的WAL日志，并获取对应块号的last LSN继续解析，直到last LSN为0或者更老版本的WAL日志已被复用回收，并将解析后的日志落盘到指定路径，必须是系统管理员或运维管理员才能执行此函数。本系统函数不支持备机调用。

  返回值类型：text

参数类型	参数名	类型	描述
输入参数	last_lsn	text	解析指定页面的LSN，基于十六进制表示，如12BA/32CDEDDD，LSN可通过页面解析工具（gs_parse_page_bypath等）获取。
输入参数	blocknum	bigint	指定页面的逻辑块号。 参数范围：-1~MaxBlockNumber。 块号指定为-1时表示从WAL日志中获取默认块号。
输入参数	relation_type	text	指定解析页面的类型。 参数范围：uheap、ubtree、heap、btree、undo_record、undo_slot。

参数类型	参数名	类型	描述
输出参数	output_filepath	text	WAL日志解析结果的落盘路径。

示例：

```
# 获取页面LSN信息
# 函数调用前请依据参数说明中的内容确保文件路径真实存在
gaussdb=# select * from gs_parse_page_bypath('base/15833/16768', 0, 'uheap', false);
                        output_filepath
-------------------------------------------------------------
 /data1/database/cluster/primary/data/1663_15833_16768_0.page
(1 row)
gaussdb=# select * from gs_xlogdump_bylastlsn('0/4593570', -1, 'uheap');
                        output_filepath
-----------------------------------------------------------------
 /data1/database/cluster/primary/data/pg_log/dump/4593570_-1.xlog
(1 row)
gaussdb=# select * from gs_xlogdump_bylastlsn('0/4593570', 0, 'ubtree');
ERROR:  The input lsn 0/4593570 related xlog is not ubtree.
```

7.5.25.12 Undo 系统函数

- gs_undo_meta(type, zoneId, location)

  描述：Undo模块元信息。

  参数说明：

  - type(元信息类型)
    - 0：表示UndoZone(Record) 对应的元信息。
    - 1：表示UndoZone(Transaction Slot) 对应的元信息。
    - 2：表示UndoSpace(Record) 对应的元信息。
    - 3：表示UndoSpace(Transaction Slot) 对应的元信息。
  - zoneId(UndoZone编号)
    - -1：表示所有UndoZone的元信息。
    - 0-1024*1024-1：表示对应ZoneID的元信息。
  - location(读取位置）
    - 0：表示从当前内存中读取。
    - 1：表示从物理文件中读取。

  返回值类型：record

  表 7-75 gs_undo_meta 参数说明

参数类型	参数名	类型	描述
输出参数	zoneId	oid	Undo Zone的ID。
输出参数	persistType	oid	持久化级别。

参数类型	参数名	类型	描述
输出参数	insert	text	下一条插入的Undo记录位置。
输出参数	discard	text	普通回收到的Undo记录位置。
输出参数	end	text	强制回收掉的Undo记录位置，小于它的Undo记录已经被回收。
输出参数	used	text	已经使用的Undo空间。
输出参数	lsn	text	修改UndoZone的LSN。
输出参数	pid	oid	UndoZone绑定的进程ID。

- gs_undo_translot(location, zoneId)
  描述：Undo事务槽信息。
  参数说明：
  - location(读取位置)
    - 0：表示从当前内存中读取。
    - 1：表示从物理文件中读取。
  - zoneId(UndoZone编号)
    - -1：表示所有UndoZone的元信息。
    - 0-1024*1024-1：表示对应ZoneID的元信息。

  返回值类型：record

  表 7-76 gs_undo_translot 参数说明

参数类型	参数名	类型	描述
输出参数	groupId	oid	使用的UndoZone ID。
输出参数	xactId	text	事务ID。
输出参数	startUndoPtr	text	Transaction Slot对应事务起始插入Undo记录位置。
输出参数	endUndoPtr	text	Transaction Slot对应事务结束插入Undo记录位置。
输出参数	lsn	text	对应Transaction Slot指针。
输出参数	slot_states	oid	事务状态。 ● 0：表示已经提交。 ● 1：表示正在执行中。 ● 2：表示回滚中。 ● 3：表示回滚完成。

- gs_stat_undo([bool init])
  描述：Undo统计信息。
  返回值类型：record

  表 7-77 gs_stat_undo 参数说明

参数类型	参数名	类型	描述
输入参数	init	bool	可选参数，是否清理统计信息并重新开始统计。
输出参数	curr_used_zone_count	uint32	当前使用的UndoZone数量。
输出参数	top_used_zones	text	前三个使用量最大的UndoZone信息，格式输出为： ● zoneId1：使用大小。 ● zoneId2：使用大小。 ● zoneId3：使用大小。
输出参数	curr_used_undo_size	uint32	当前使用的Undo总空间大小，单位为MB。
输出参数	undo_threshold	uint32	为guc参数undo_space_limit_size * 80%计算的结果,单位为MB。
输出参数	global_recycle_xid	uint64	当前Undo空间回收到的事务xid(小于该xid事务产生的Undo记录都已经被回收)。
输出参数	oldest_xmin	uint64	最老的活跃事务。
输出参数	total_undo_chain_len	int64	所有访问过的Undo链总长度。
输出参数	max_undo_chain_len	int64	最大访问过的Undo链长度。
输出参数	create_undo_file_count	uint32	创建的Undo文件数量统计。
输出参数	discard_undo_file_count	uint32	删除的Undo文件数量统计。
输出参数	info	text	如果入参为false，输出undo_space_limit_size、undo_limit_size_per_transaction 、undo_retention_time 参数的合理化建议。如果入参为true，即需要init（清理统计信息），仅提示‘The statistics have been initialized.’。

示例1：清理undo统计信息

```
gaussdb=# select * from gs_stat_undo(true);
 curr_used_zone_count |    top_used_zones     | curr_used_undo_size | undo_threshold |
global_recycle_xid | oldest_xmin | total_undo_chain_len | max_undo_chain_len | create_undo_file_coun
t | discard_undo_file_count |                 info
----------------------+-----------------------+---------------------+----------------+--------------------
+-------------+----------------------+--------------------+----------------------
--+-------------------------+-----------------------------------
                   3 | 0 : 0, 0 : 0, 0 : 0 |                   1 |         209715 |             15741 |       15741
|                    0 |                   0 |
2 |                       0 | The statistics have been initialized.
(1 row)
```

示例2：输出undo统计信息

```
gaussdb=# select * from gs_stat_undo(false);
 curr_used_zone_count |    top_used_zones     | curr_used_undo_size | undo_threshold |
global_recycle_xid | oldest_xmin | total_undo_chain_len | max_undo_chain_len | create_undo_file_coun
t | discard_undo_file_count |
             info

----------------------+-----------------------+---------------------+----------------+--------------------
+-------------+----------------------+--------------------+----------------------
--+-------------------------
+-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------
-
                   3 | 0 : 0, 0 : 0, 0 : 0 |                   1 |         209715 |             16253 |       16253
|                    0 |                   0 |
2 |                       0 | Based on the statistic info from last initialization, undo_space_limit_size is
recommended to set >= 120953 blocks, undo_limit_size_per_transaction is
recommended to set <= 327150 blocks, undo_retention_time is recommended to set <= 259200
seconds. Since last initialization, max undo space used by a single transaction is 32715 blocks.
(1 row)
```

- gs_undo_record(undoptr)

  描述：Undo记录解析。

  参数说明：

  – undoptr（undo记录指针）

  返回值类型：record

- gs_undo_dump_parsepage_mv(relpath text, blkno bigint, reltype text, rmem boolean)

  描述：解析USTORE数据表磁盘页面的页头信息，每个元组的头部信息，标识位信息以及所有可以查询到undo历史版本信息。

  返回值类型：text

  备注：必须是系统管理员或者运维管理人员才能执行此函数。

  说明

  该接口当前仅支持USTORE数据表。

表 7-78 gs_undo_dump_parsepage_mv 参数说明

参数类型	参数名	类型	描述
输入参数	relpath	text	USTORE表数据文件相对路径，相对路径格式为：tablespace name/database oid/relfilenode，例如base/16603/16384, 表对应数据文件的相对路径查找可以通过pg_relation_filepath('tablename')查询。
输入参数	blkno	bigint	• -1：解析所有block页面。 • 0-MaxBlocNumber：解析指定的block页面。
输入参数	reltype	text	表类型，目前仅支持USTORE数据表，取值为uheap。
输入参数	rmem	boolean	• false • true 目前仅支持false，从磁盘文件上解析对应的页面。
输出参数	output	text	解析结果文件的绝对路径。

- gs_undo_meta_dump_zone(zone_id int, read_memory boolean)
  描述：解析Undo模块中UndoZone的元信息。
  返回值类型：record

  表 7-79 gs_undo_meta_dump_zone 参数说明

参数类型	参数名	类型	描述
输入参数	zone_id	int	UndoZone编号： • -1：查询所有UndoZone。 • 0-1,048,575：查询对应zone_id编号的UndoZone元信息。
输入参数	read_memory	boolean	• true：从当前内存中读取。 • false：从物理文件中读取。
输出参数	zone_id	int	UndoZone编号。

参数类型	参数名	类型	描述
输出参数	persist_type	int	持久化级别： ● 0：普通表。 ● 1：无日志表。 ● 2：临时表。
输出参数	insert	text	下一条插入的Undo记录位置。
输出参数	discard	text	普通回收到的Undo记录位置。
输出参数	forcediscard	text	强制回收掉Undo记录位置，小于它的Undo记录已经被回收。
输出参数	lsn	text	修改Zone的LSN。

- gs_undo_meta_dump_spaces(zone_id int, read_memory boolean)
  描述：解析Undo模块中Undo记录空间、Transaction Slot空间的元信息。
  返回值类型：record

  表 7-80 gs_undo_meta_dump_spaces 参数说明

参数类型	参数名	类型	描述
输入参数	zone_id	int	UndoZone编号： ● -1：查询所有UndoZone。 ● 0-1,048,575：查询对应zone_id编号的UndoZone元信息。
输入参数	read_memory	boolean	● true：从当前内存中读取。 ● false：从物理文件中读取。
输出参数	zone_id	int	UndoZone编号。
输出参数	undorecord_space_tail	text	UndoRecord空间的结尾位置。
输出参数	undorecord_space_head	text	UndoRecord空间的起始位置。
输出参数	undorecord_space_lsn	text	修改UndoRecord空间LSN。
输出参数	undoslot_space_tail	text	Transaction Slot空间的结尾位置。

参数类型	参数名	类型	描述
输出参数	undoslot_space_head	text	Transaction Slot空间的起始位置。
输出参数	undoreslot_space_lsn	text	修改Transaction Slot空间LSN。

- gs_undo_meta_dump_slot(zone_id int, read_memory boolean)
  描述：解析Undo模块中Transaction Slot元信息。
  返回值类型：record

  表 7-81 gs_undo_meta_dump_slot 参数说明

参数类型	参数名	类型	描述
输入参数	zone_id	int	Undo zone编号： ● -1：查询所有UndoZone。 ● 0-1,048,575：查询对应zone_id编号的UndoZone元信息。
输入参数	read_memory	boolean	● true：从当前内存中读取。 ● false：从物理文件中读取。
输出参数	zone_id	int	UndoZone编号。
输出参数	allocate	text	Undo Transaction Slot分配位置。
输出参数	recycle	text	Undo Transaction Slot回收位置。
输出参数	frozen_xid	text	Frozen Xid，用于可见性判断。
输出参数	global_frozen_xid	text	全局最小的Frozen Xid，小于该Xid的事务可见。
输出参数	recycle_xid	text	回收到的Xid，小于该Xid的事务被回收。
输出参数	global_recycle_xid	text	全局最小的Recycle Xid，小于该Xxid的事务被回收。

- gs_undo_translot_dump_slot(zone_id int, read_memory boolean)
  描述：解析UndoZone中的Transaction Slot。
  返回值类型：record

表 7-82 gs_undo_translot_dump_slot 参数说明

参数类型	参数名	类型	描述
输入参数	zone_id	oid	UndoZone编号： ● -1：查询所有UndoZone。 ● 0-1,048,575：查询对应zone_id编号的UndoZone元信息。
输入参数	read_memory	boolean	● true：从当前内存中读取。 ● false：从物理文件中读取。
输出参数	zone_id	text	UndoZone编号。
输出参数	slot_xid	text	事务ID。
输出参数	start_undoptr	text	Transaction Slot对应事务起始插入Undo记录位置。
输出参数	end_undoptr	text	Transaction Slot对应事务结束插入Undo记录位置。
输出参数	slot_ptr	text	Transaction Slot对应的位置。
输出参数	slot_states	oid	事务状态： ● 0：已提交。 ● 1：执行中。 ● 2：回滚中。 ● 3：回滚完成。

- gs_undo_translot_dump_xid(slot_xid xid, read_memory boolean)
  描述：根据xid解析UndoZone中对应的Transaction Slot。
  返回值类型：record

表 7-83 gs_undo_translot_dump_xid 参数说明

参数类型	参数名	类型	描述
输入参数	slot_xid	xid	需要查询的事务ID。
输入参数	read_memory	boolean	● true：从当前内存中读取。 ● false：从物理文件中读取。

参数类型	参数名	类型	描述
输出参数	zone_id	text	UndoZ-one编号。
输出参数	slot_xid	text	事务ID。
输出参数	start_undoptr	text	Transaction Slot对应事务起始插入Undo记录位置。
输出参数	end_undoptr	text	Transaction Slot对应事务结束插入Undo记录位置。
输出参数	slot_ptr	text	Transaction Slot对应的位置。
输出参数	slot_states	oid	事务状态： ● 0 已提交。 ● 1 执行中。 ● 2 回滚中。 ● 3 回滚完成。

- gs_undo_dump_record(undoptr bigint)
  描述：解析Undo记录。
  返回值类型：record

  表 7-84 gs_undo_dump_record 参数说明

参数类型	参数名	类型	描述
输入参数	undoptr	bigint	需要解析的Undo记录起始位置。
输出参数	undoptr	bigint	需要解析的Undo记录起始位置。
输出参数	xactid	text	事务ID。
输出参数	cid	text	Command Id。
输出参数	reloid	text	Relation Oid。
输出参数	relfilenode	text	文件的Relfinode。
输出参数	utype	text	Undo记录类型。

参数类型	参数名	类型	描述
输出参数	blkprev	text	同一个块前一条Undo记录的位置。
输出参数	blockno	text	块号。
输出参数	uoffset	text	Undo记录偏移。
输出参数	prevurp	text	前一条Undo记录位置。
输出参数	payloadlen	text	Undo记录数据部分长度。
输出参数	oldxactid	text	前一个事务ID。
输出参数	partitionoid	text	分区Oid。
输出参数	tablespace	text	表空间。
输出参数	alreadyread_bytes	text	读取到的Undo记录长度。
输出参数	prev_undorec_len	text	前一条Undo记录长度。
输出参数	td_id	text	Transaction Directory的ID。
输出参数	reserved	text	Undo 记录中存储的旧版本元组预留标识位。
输出参数	flag	text	Undo 记录中存储的旧版本元组状态标识。
输出参数	flag2	text	Undo 记录中存储的旧版本元组列数。
输出参数	t_hoff	text	Undo记录数据头的长度。

示例（在Undo记录没有被回收的前提下，入参undoptr可以通过gs_undo_translot_dump_slot函数的出参end_undoptr转换为10进制后进行查询）：

```
gaussdb=# SELECT * FROM gs_undo_dump_record('0000000000000042');
 undoptr |   xactid   | cid | reloid | relfilenode | utype |  blkprev   | blockno | uoffset | prevurp |
payloadlen | oldxactid | partitionoid | tablespace | alreadyread_byt
es | prev_undorec_len | td_id | reserved | flag | flag2 | t_hoff
---------+------------+-----+--------+-------------+-------+------------+---------+---------+---------
+------------+-----------+--------------+------------+----------------
```



```
---+------------------+-------+----------+------+-------+--------
     42 | 1073807360 | 0   | 0      | 108986369   | 0     | 1024786474 | 0       | 0       | 0       | 0         |
0         | 0            | 0          | 36
   | 16390           | -1    | -1       | -1    | -1    | -1
(1 row)
```

- gs_undo_dump_xid(undo_xid xid)

  描述：根据xid解析Undo记录。

  返回值类型：record

  表 7-85 gs_undo_dump_xid 参数说明

参数类型	参数名	类型	描述
输入参数	undo_xid	xid	事务Xid。
输出参数	undoptr	xid	需要解析的Undo记录起始位置。
输出参数	xactid	text	事务ID。
输出参数	cid	text	Command Id。
输出参数	reloid	text	Relation Oid。
输出参数	relfilenode	text	文件的Relfinode。
输出参数	utype	text	Undo记录类型。
输出参数	blkprev	text	同一个块前一条Undo记录的位置。
输出参数	blockno	text	块号。
输出参数	uoffset	text	Undo记录偏移。
输出参数	prevurp	text	前一条Undo记录位置。
输出参数	payloadlen	text	Undo记录数据部分长度。
输出参数	oldxactid	text	前一个事务ID。
输出参数	partitionoid	text	分区Oid。

参数类型	参数名	类型	描述
输出参数	tablespace	text	表空间。
输出参数	alreadyread_bytes	text	读取到的Undo记录长度。
输出参数	prev_undorec_len	text	前一条Undo记录长度。
输出参数	td_id	text	Transaction Directory的ID。
输出参数	reserved	text	Undo 记录中存储的旧版本元组预留标识位。
输出参数	flag	text	Undo 记录中存储的旧版本元组状态标识。
输出参数	flag2	text	Undo 记录中存储的旧版本元组列数。
输出参数	t_hoff	text	Undo记录数据头的长度。

- gs_verify_undo_record(type, start_idx，end_idx, location)

  描述：校验Undo记录，目前只支持磁盘校验模式。仅支持在业务非运行时执行离线校验，校验之前需要手动执行一次checkpoint落盘操作。

  返回值类型：record

  表 7-86 gs_verify_undo_record 参数说明

参数类型	参数名	类型	描述
输入参数	type	text	校验类型： ● 'urp'：校验给定urp区间的所有Undo记录。 ● 'zone'：校验给定Zone区间的所有Zone的所有Undo记录。
输入参数	start_idx	int64	开始位置： ● 'urp'时，表示起始Undo记录位置。 ● type为'zone'时，表示起始UndoZone编号。
输入参数	end_idx	int64	结束位置： ● 'urp'时，表示Undo记录结束位置。 ● 'zone'时，表示结束UndoZone编号。

参数类型	参数名	类型	描述
输入参数	location	bool	● 0：内存校验。 ● 1：磁盘校验。 目前该参数只支持输入为1。
输出参数	zone_id	int64	UndoZone编号。
输出参数	detail	text	校验出错信息。

示例1：校验urp为24的这条Undo记录：

```
gaussdb=# select * from gs_verify_undo_record('urp', 24, 24, 1);
zone_id | detail
--------+--------
(0 rows)
```

示例2：若校验urp时输入的urp是错误的，即不是Undo记录起始位置，可能会返回校验结果为空或校验失败：

```
gaussdb=# select * from gs_verify_undo_record('urp', 35184372173095, 35184372173095, 1);
 zone_id |                          detail
---------+-----------------------------------------------------------------
       2 | verification failed: xid is invalid, urp:35184372173095.
(1 row)
```

示例3：从磁盘中校验zone0到zone2的所有Undo记录：

```
gaussdb=# select * from gs_verify_undo_record('zone', 0, 2, 1);
zone_id | detail
--------+--------
(0 rows)
```

说明

调用此视图如有报错，请联系华为工程师处理。

- gs_verify_undo_slot(type, start_idx，end_idx, location)

  描述：校验Undo事务槽，目前只支持磁盘校验模式。仅支持在业务非运行时执行离线校验，校验之前需要手动执行一次checkpoint落盘操作。

  返回值类型：record

表 7-87 gs_verify_undo_slot 参数说明

参数类型	参数名	类型	描述
输入参数	type	text	校验类型： ● 'zone'：校验给定Zone区间的所有Zone的所有事务槽。
输入参数	start_idx	int64	起始UndoZone编号。
输入参数	end_idx	int64	结束UndoZone编号。

参数类型	参数名	类型	描述
输入参数	location	bool	● 0：内存校验。 ● 1：磁盘校验。 目前该参数只支持输入为1。
输出参数	zone_id	int64	UndoZone编号。
输出参数	detail	text	校验出错信息。

示例：从磁盘中校验zone0到zone2的所有事务槽记录：

```
gaussdb=# select * from gs_verify_undo_slot('zone', 0, 2, 1);
zone_id | detail
--------+--------
(0 rows)
```

说明

调用此视图如有报错，请联系华为工程师处理。

- gs_verify_undo_meta(type, start_idx，end_idx, location)

  描述：校验Undo元信息，目前只支持磁盘校验模式。仅支持在业务非运行时执行离线校验，校验之前需要手动执行一次checkpoint落盘操作。

  返回值类型：record

表 7-88 gs_verify_undo_meta 参数说明

参数类型	参数名	类型	描述
输入参数	type	text	校验类型，type只能设置为'all'。 ● 'all'：校验给定Zone区间的所有Zone的所有Meta信息。
输入参数	start_idx	int64	起始UndoZone编号。
输入参数	end_idx	int64	结束UndoZone。
输入参数	location	bool	● 0：内存校验。 ● 1：磁盘校验。 目前该参数只支持输入为1。
输出参数	zone_id	int64	UndoZone编号。
输出参数	detail	text	校验出错信息。

示例：从磁盘中校验zone0到zone2的所有meta信息记录：

```
gaussdb=# select * from gs_verify_undo_meta('all', 0, 2, 1);
zone_id | detail
--------+--------
(0 rows)
```


说明

调用此视图如有报错，请联系华为工程师处理。

- gs_async_rollback_worker_status()
  描述：活跃异步回滚线程状态监控。
  返回值类型：record

  表 7-89 gs_async_rollback_worker_status 参数说明

参数类型	参数名	类型	描述
输出参数	datid	oid	数据库ID。
输出参数	pid	int64	进程ID。
输出参数	sessionid	int64	会话ID。
输出参数	usesysid	oid	发起该线程的用户ID。
输出参数	state	int32	线程当前状态： 0：未定义。 1：空闲。 2：运行中。
输出参数	rollback_start_time	timestamptz	线程启动时间戳。
输出参数	idx	uint32	异步回滚线程在数组中的下标。
输出参数	xid	uint64	正在回滚的事务Xid。
输出参数	progress	text	该事务的回滚进度（当前已回滚的Undo 记录条数/总Undo 记录条数，以百分比形式显示）。

- gs_async_rollback_xact_status()
  描述：异步回滚任务哈希表监控。
  返回值类型：record

  表 7-90 gs_async_rollback_xact_status 参数说明

参数类型	参数名	类型	描述
输出参数	xid	xid	需要进行异步回滚的事务Xid。
输出参数	start_undoptr	uint64	该事务起始Undo记录指针。
输出参数	end_undoptr	uint64	该事务结束Undo记录指针。
输出参数	dbid	uint32	该事务所在数据库ID。

参数类型	参数名	类型	描述
输出参数	slot_ptr	uint64	该事务对应事务槽的指针。
输出参数	launched	bool	是否有对应的活跃的异步回滚线程。

- gs_undo_recycler_status()
  描述：异步回收线程状态监控。
  返回值类型：record

  表 7-91 gs_undo_recycler_status 参数说明

参数类型	参数名	类型	描述
输出参数	datid	oid	数据库ID。
输出参数	pid	int64	进程ID。
输出参数	sessionid	int64	会话ID。
输出参数	usesysid	oid	发起该线程的用户ID。
输出参数	state	int32	线程当前状态： 0：未定义。 1：空闲。 2：运行中。
输出参数	backend_start	timestamptz	线程启动时间戳。
输出参数	total_recycle_time	uint64	总回收时间。
输出参数	max_recycle_time	uint64	最大回收时间。
输出参数	total_recycle_size	uint64	总回收空间。
输出参数	total_recycle_count	uint64	总回收次数。
输出参数	recycle_sleep_count	uint64	睡眠次数。
输出参数	recycle_sleep_time	uint64	睡眠总时间。
输出参数	max_recycle_sleep_time	uint64	最长睡眠时间。
输出参数	last_recycle_timestamp	uint64	上次成功回收时间戳。

参数类型	参数名	类型	描述
输出参数	last_update_global_recycle_xid_timestamp	uint64	globalRecycleXid上次推进时间戳。

- gs_undo_launcher_status()

  描述：异步回滚发起线程状态监控。

  返回值类型：record

  表 7-92 gs_undo_launcher_status 参数说明

参数类型	参数名	类型	描述
输出参数	datid	oid	数据库ID。
输出参数	pid	int64	进程ID。
输出参数	sessionid	int64	会话ID。
输出参数	usesysid	oid	发起该线程的用户ID。
输出参数	state	int32	线程当前状态： 0：未定义。 1：空闲。 2：运行中。
输出参数	backend_start	timestamptz	线程启动时间戳。
输出参数	total_async_rollback_task_count	uint64	本节点数据库启动后发起异步回滚任务总个数。
输出参数	average_async_rollback_time	uint64	异步回滚任务平均耗时。
输出参数	max_async_rollback_time	uint64	异步回滚任务最长耗时。
输出参数	min_async_rollback_time	uint64	异步回滚任务最短耗时。

7.5.25.13 SQL 限流函数

- gs_add_workload_rule(rule_type, rule_name, databases, start_time, end_time, max_workload, option_val)

  描述：创建一条SQL限流规则，需要具有sysadmin权限的用户才可执行。

  参数：参数介绍请参见**表7-93**。

  返回值类型：int8

表 7-93 gs_add_workload_rule 参数说明

参数名称	类型	描述	取值范围
rule_type	text	限流规则类型，不区分大小写。	“sqlid”：根据Unique SQL ID进行限流； “select”、“insert”、“update”、“delete”、“merge”：根据查询类型和关键字进行限流； “resource”：根据系统资源利用率进行实例级别的限流。
rule_name	name	限流规则名称，用于检索限流规则。	任意字符串，可以为NULL。
databases	name[]	限流规则生效的数据库名称数组，区分大小写。	数据库名列表，必须为已创建的数据库名。可以为NULL，表示所有数据库生效。 目前只有指定rule_type为查询类型时，数据库列表才生效，因为Unique SQL ID本身是与库进行绑定的，其只属于某个库；而根据资源利用率的限流规则是对实例生效的，即对所有库生效。
start_time	timestamptz	限流规则生效的开始时间。	可以为NULL，表示从当前时间开始生效。
end_time	timestamptz	限流规则生效的结束时间。	可以为NULL，表示规则一直生效。
max_workload	int8	限流规则设置的最大并发数。	-

参数名称	类型	描述	取值范围
option_val	text[]	限流规则的补充信息。	与rule_type匹配，具体匹配关系如下： ● “sqlid”：要限流的Unique SQL ID，以及慢SQL管控规则，格式为'{id=1234, time_limit=100, max_execute_time=500, max_iops=1}'，其中id值为Unique SQL ID，为必选项，可通过dbe_perf.statement或者pg_stat_activity视图获取。其他选项非必选，其含义参考慢SQL管控规则的Hint； ● “select”、“insert”、“update”、“delete”、“merge”： 要限流的关键字序列，不区分大小写，可以为NULL； ● “resource”： 要限流的资源阈值，形式为'{cpu-80, memory-70}'，表示触发实例级别限流的操作系统资源阈值，可以为NULL，表示不管资源利用率直接进行限流。

示例：

```
gaussdb=#  select gs_add_workload_rule('sqlid', 'rule for one query', '', now(), '', 20, '{id=32413214}');
 gs_add_workload_rule
----------------------
                    1
(1 row)
gaussdb=#  create database db1;
gaussdb=#  create database db2;
gaussdb=#  select gs_add_workload_rule('select', 'rule for select', '{db1, db2}', '', '', 100, '{tb1, tb2}');
 gs_add_workload_rule
----------------------
                    2
(1 row)
gaussdb=#  select gs_add_workload_rule('resource', 'rule for resource', '{}', '', '', 20, '{cpu-80}');
 gs_add_workload_rule
----------------------
                    3
(1 row)
gaussdb=#  drop database db1;
DROP DATABASE
gaussdb=#  drop database db2;
DROP DATABASE
```

- gs_update_workload_rule(rule_id, rule_name, databases, start_time, end_time, max_workload, option_val)

描述：更新一条SQL限流规则，需要重新设置全部参数，不支持只指定部分参数。需要具有sysadmin权限的用户才可执行。

参数：参数介绍请参见**表7-94**

返回值类型：boolean

表 7-94 gs_update_workload_rule 参数说明

参数名称	类型	描述	取值范围
rule_id	int8	要更新的限流规则ID。	-
rule_name	name	限流规则名称，用于检索限流规则。	任意字符串，可以为NULL。
databases	name[]	限流规则生效的数据库名称数组，区分大小写。	数据库名列表，必须为已创建的数据库名。可以为NULL，表示所有数据库生效。 目前只有指定rule_type为查询类型时，数据库列表才生效，因为Unique SQL ID本身是与库进行绑定的，其只属于某个库；而根据资源利用率的限流规则是对实例生效的，即对所有库生效。
start_time	timestamptz	限流规则生效的开始时间。	可以为NULL，表示从当前时间开始生效。
end_time	timestamptz	限流规则生效的结束时间。	可以为NULL，表示规则一直生效。
max_workload	int8	限流规则设置的最大并发数。	-

参数名称	类型	描述	取值范围
option_val	text[]	限流规则的补充信息。	与rule_type匹配，具体匹配关系如下： ● “sqlid”：要限流的Unique SQL ID，以及慢SQL管控规则，格式为'{id=1234, time_limit=100, max_execute_time=500, max_iops=1}'，其中id值为Unique SQL ID，为必选项，可通过dbe_perf.statement或者pg_stat_activity视图获取。其他选项非必选，其含义参考慢SQL管控规则的Hint； ● “select”、“insert”、“update”、“delete”、“merge”：要限流的关键字序列，不区分大小写，可以为NULL； ● “resource”：要限流的资源阈值，形式为'{cpu-80, memory-70}'，表示触发实例级别限流的操作系统资源阈值，可以为NULL，表示不管资源利用率直接进行限流。

示例：

```
gaussdb=#  create database db1;
gaussdb=#  select gs_update_workload_rule(2, 'rule for select 2', '{db1}', now(), '', 50, '{tb1}');
 gs_update_workload_rule
-------------------------
 t
(1 row)
gaussdb=#  drop database db1;
DROP DATABASE
```

- gs_delete_workload_rule(rule_id int8)

  描述：删除一条SQL限流规则，需要具有sysadmin权限的用户才可执行。

  参数：rule_id，要更新的限流规则ID，类型为int8。

  返回值类型：boolean

  示例：

```
gaussdb=#  select gs_delete_workload_rule(3);
 gs_delete_workload_rule
```

```
-------------------------
 t
(1 row)
```

- gs_get_workload_rule_stat(rule_id)

  描述：查询SQL限流规则拦截SQL的次数，需要具有sysadmin权限的用户才可执行。

  参数：rule_id，要查询的限流规则ID，类型为int8。可以指定rule_id为-1，此时表示查询所有的SQL限流规则。

  返回值类型：

名称	类型	描述
rule_id	int8	SQL限流规则的ID。
validate_count	int8	SQL限流规则拦截SQL的次数。

  示例：

```
gaussdb=#  select * from gs_get_workload_rule_stat(1);
 rule_id | validate_count
---------+----------------
       1 |              0
(1 row)
gaussdb=#  select * from gs_get_workload_rule_stat(-1);
 rule_id | validate_count
---------+----------------
       1 |              0
       2 |              0
(2 rows)
```

7.5.26 统计信息函数

统计信息函数根据访问对象分为两种类型：针对某个数据库进行访问的函数，以数据库中每个表或索引的OID作为参数，标识需要报告的数据库；针对某个服务器进行访问的函数，以一个服务器进程号为参数，其范围从1到当前活跃服务器的数目。

- pg_stat_get_db_conflict_tablespace(oid)

  描述：由于恢复与数据库中删除的表空间发生冲突而取消的查询数。

  返回值类型：bigint

- pg_control_group_config()

  描述：在当前节点上打印cgroup配置。该函数需要SYSADMIN权限的用户才能够执行。

  返回值类型：record

- pg_stat_get_db_stat_reset_time(oid)

  描述：上次重置数据库统计信息的时间。首次连接到每个数据库期间初始化为系统时间。当您在数据库上调用pg_stat_reset以及针对其中的任何表或索引执行pg_stat_reset_single_table_counters时，重置时间都会更新。

  返回值类型：timestamptz

- pg_stat_get_function_total_time(oid)

  描述：该函数花费的总挂钟时间，以微秒为单位。包括花费在此函数调用其它函数上的时间。

返回值类型：bigint

- pg_stat_get_xact_tuples_returned(oid)

  描述：当前事务中参数为表时通过顺序扫描读取的行数，或参数为索引时返回的索引条目数。

  返回值类型：bigint

- pg_lock_status()

  描述：查询打开事务所持有的锁信息，所有用户均可执行该函数。

  返回值类型：返回字段可参考**PG_LOCKS**视图返回字段，该视图是通过查询本函数得到的结果。

- gs_lwlock_status()

  描述：查询数据库系统内所有轻量级锁信息，包括等锁和持锁信息，所有用户均可执行该函数。

  返回值类型：setofrecord

- pg_stat_get_xact_numscans(oid)

  描述：当前事务中参数为表时执行的顺序扫描次数，或参数为索引时执行的索引扫描次数。

  返回值类型：bigint

- pg_stat_get_xact_blocks_fetched(oid)

  描述：当前事务中对表或索引的磁盘块获取请求数。

  返回值类型：bigint

- pg_stat_get_xact_blocks_hit(oid)

  描述：当前事务中对缓存中找到的表或索引的磁盘块获取请求数。

  返回值类型：bigint

- pg_stat_get_xact_function_calls(oid)

  描述：在当前事务中调用该函数的次数。

  返回值类型：bigint

- pg_stat_get_xact_function_self_time(oid)

  描述：在当前事务中仅花费在此函数上的时间，不包括花费在此函数内部调用其它函数上的时间。

  返回值类型：bigint

- pg_stat_get_xact_function_total_time(oid)

  描述：当前事务中该函数所花费的总挂钟时间（以微秒为单位），包括花费在此函数内部调用其它函数上的时间。

  返回值类型：bigint

- pg_stat_get_wal_senders()

  描述：在主机端查询walsender信息。

  返回值类型：setofrecord

  返回字段说明如下：

表 7-95 返回字段说明

字段名称	字段类型	字段说明
pid	bigint	walsender的线程号。
sender_pid	integer	walsender的pid相对的轻量级线程号。
local_role	text	主节点类型。
peer_role	text	备节点类型。
peer_state	text	备节点状态。
state	text	walsender状态。
catchup_start	timestamp with time zone	catchup启动时间。
catchup_end	timestamp with time zone	catchup结束时间。
sender_sent_location	text	主节点发送位置。
sender_write_location	text	主节点落盘位置。
sender_flush_location	text	主节点flush磁盘位置。
sender_replay_location	text	主节点redo位置。
receiver_received_location	text	备节点接收位置。
receiver_write_location	text	备节点落盘位置。
receiver_flush_location	text	备节点flush磁盘位置。
receiver_replay_location	text	备节点redo磁盘位置。
sync_percent	text	同步百分比。
sync_state	text	同步状态。
sync_group	text	同步复制的所属分组。
sync_priority	text	同步复制的优先级。
sync_most_available	text	最大可用模式设置。
channel	text	walsender信道信息。

- get_paxos_replication_info()
  描述：查询Paxos模式下主机或备机的复制状态。
  返回值类型：setofrecord
  返回字段说明如下：

表 7-96 返回字段说明

字段名称	字段类型	字段说明
paxos_write_location	text	已经写入DCF的XLog位置。
paxos_commit_location	text	已经在DCF中达成一致的XLog位置。
local_write_location	text	节点的落盘位置。
local_flush_location	text	节点的flush磁盘位置。
local_replay_location	text	节点的redo磁盘位置。
dcf_replication_info	text	节点的DCF模块信息。

- pg_stat_get_stream_replications()
  描述：查询主备复制状态。
  返回值类型：setofrecord
  返回值说明如下。

  表 7-97 返回值说明

返回参数	返回参数类型	返回参数说明
local_role	text	本地角色。
static_connections	integer	连接统计。
db_state	text	数据库状态。
detail_information	text	详细信息。

- pg_stat_get_db_numbackends(oid)
  描述：处理该数据库活跃的服务器进程数目。
  返回值类型：integer
- pg_stat_get_db_xact_commit(oid)
  描述：数据库中已提交事务的数量。
  返回值类型：bigint
- pg_stat_get_db_xact_rollback(oid)
  描述：数据库中回滚事务的数量。
  返回值类型：bigint
- pg_stat_get_db_blocks_fetched(oid)
  描述：数据库中磁盘块抓取请求的总数。

返回值类型：bigint

- pg_stat_get_db_blocks_hit(oid)
  描述：数据库在缓冲区中找到的磁盘块抓取请求的总数。
  返回值类型：bigint
- pg_stat_get_db_tuples_returned(oid)
  描述：为数据库返回的Tuple数。
  返回值类型：bigint
- pg_stat_get_db_tuples_fetched(oid)
  描述：为数据库中获取的Tuple数。
  返回值类型：bigint
- pg_stat_get_db_tuples_inserted(oid)
  描述：在数据库中插入Tuple数。
  返回值类型：bigint
- pg_stat_get_db_tuples_updated(oid)
  描述：在数据库中更新的Tuple数。
  返回值类型：bigint
- pg_stat_get_db_tuples_deleted(oid)
  描述：数据库中删除Tuple数。
  返回值类型：bigint
- pg_stat_get_db_conflict_lock(oid)
  描述：数据库中锁冲突的数量。
  返回值类型：bigint
- pg_stat_get_db_deadlocks(oid)
  描述：数据库中死锁的数量。
  返回值类型：bigint
- pg_stat_get_numscans(oid)
  描述：如果参数是一个表，则顺序扫描读取的行数目。如果参数是一个索引，则返回索引行的数目。
  返回值类型：bigint
- pg_stat_get_role_name(oid)
  描述：根据用户oid获取用户名。仅SYSADMIN和MONADMIN用户可以访问。
  返回值类型：text
  示例：

```
gaussdb=# select pg_stat_get_role_name(10);
 pg_stat_get_role_name
-----------------------
 aabbcc
(1 row)
```

- pg_stat_get_tuples_returned(oid)
  描述：如果参数是一个表，则顺序扫描读取的行数目。如果参数是一个索引，则返回的索引行的数目。
  返回值类型：bigint

- pg_stat_get_tuples_fetched(oid)
  描述：如果参数是一个表，则位图扫描抓取的行数目。如果参数是一个索引，则用简单索引扫描抓取的行数目。
  返回值类型：bigint
- pg_stat_get_tuples_inserted(oid)
  描述：插入表中行的数量。
  返回值类型：bigint
- pg_stat_get_tuples_updated(oid)
  描述：在表中已更新行的数量。
  返回值类型：bigint
- pg_stat_get_tuples_deleted(oid)
  描述：从表中删除行的数量。
  返回值类型：bigint
- pg_stat_get_tuples_changed(oid)
  描述：该表上一次analyze或autoanalyze之后插入、更新、删除行的总数量。
  返回值类型：bigint
- pg_stat_get_tuples_hot_updated(oid)
  描述：表热更新的行数。
  返回值类型：bigint
- pg_stat_get_live_tuples(oid)
  描述：表活行数。
  返回值类型：bigint
- pg_stat_get_dead_tuples(oid)
  描述：表死行数。
  返回值类型：bigint
- pg_stat_get_blocks_fetched(oid)
  描述：表或者索引的磁盘块抓取请求的数量。
  返回值类型：bigint
- pg_stat_get_blocks_hit(oid)
  描述：在缓冲区中找到的表或者索引的磁盘块请求数目。
  返回值类型：bigint
- pg_stat_get_xact_tuples_fetched(oid)
  描述：事务中扫描的tuple行数。
  返回值类型：bigint
- pg_stat_get_xact_tuples_inserted(oid)
  描述：表相关的活跃子事务中插入的tuple数。
  返回值类型：bigint
- pg_stat_get_xact_tuples_deleted(oid)
  描述：表相关的活跃子事务中删除的tuple数。
  返回值类型：bigint

- pg_stat_get_xact_tuples_hot_updated(oid)

  描述：表相关的活跃子事务中热更新的tuple数。

  返回值类型：bigint

- pg_stat_get_xact_tuples_updated(oid)

  描述：表相关的活跃子事务中更新的tuple数。

  返回值类型：bigint

- pg_stat_get_last_vacuum_time(oid)

  描述：用户在该表上最后一次手动启动清理或者autovacuum线程启动清理的时间。

  返回值类型：timestamptz

- pg_stat_get_last_autovacuum_time(oid)

  描述：autovacuum守护进程在该表上最后一次启动清理的时间。

  返回值类型：timestamptz

- pg_stat_get_vacuum_count(oid)

  描述：用户在该表上启动清理的次数。

  返回值类型：bigint

- pg_stat_get_autovacuum_count(oid)

  描述：autovacuum守护进程在该表上启动清理的次数。

  返回值类型：bigint

- pg_stat_get_last_analyze_time(oid)

  描述：用户在该表上最后一次手动启动分析或者autovacuum线程启动分析的时间。

  返回值类型：timestamptz

- pg_stat_get_last_autoanalyze_time(oid)

  描述：autovacuum守护进程在该表上最后一次启动分析的时间。

  返回值类型：timestamptz

- pg_stat_get_analyze_count(oid)

  描述：用户在该表上启动分析的次数。

  返回值类型：bigint

- pg_stat_get_autoanalyze_count(oid)

  描述：autovacuum守护进程在该表上启动分析的次数。

  返回值类型：bigint

- pg_total_autovac_tuples(bool)

  描述：返回total autovac相关的tuple记录，如nodename、nspname、relname以及各类tuple的IUD信息，入参为：是否查询relation信息。

  返回值类型：setofrecord

  返回参数说明如下。

表 7-98 返回参数说明

返回参数	返回参数类型	返回参数说明
nodename	name	节点名称。
nspname	name	名称空间名称。
relname	name	表、索引、视图等对象名称。
partname	name	分区名称。
n_dead_tuples	bigint	表分区内的死行数。
n_live_tuples	bigint	表分区内的活行数。
changes_since_analyze	bigint	analyze产生改变的数量。

- pg_total_gsi_autovac_tuples(bool)

  描述：集中式不支持。

  返回值类型：setofrecord

- pg_autovac_status(oid)

  描述：返回和autovac状态相关的参数信息，如nodename,nspname,relname,analyze,vacuum设置，analyze/vacuum阈值,analyze/vacuum tuple数等。仅SYSADMIN可以使用该函数。

  返回值类型：setofrecord

  返回值参数说明如下。

  表 7-99 返回值参数说明

返回参数	返回参数类型	返回参数说明
nspname	text	名称空间名称。
relname	text	表、索引、视图等对象名称。
nodename	text	节点名称。
doanalyze	Boolean	是否执行analyze。
anltuples	bigint	analyze tuple数量。
anlthresh	bigint	analyze阈值。
dovacuum	Boolean	是否执行vacuum。
vactuples	bigint	vacuum tuple数量。
vacthresh	bigint	vacuum阈值。

- pg_autovac_timeout(oid)

描述：返回某个表做autovac连续超时的次数，表信息非法或node信息异常返回NULL。

返回值类型：bigint

- pg_stat_get_last_data_changed_time(oid)

  描述：对于在表上的修改insert/update/delete/truncate和在表的分区(partition/subpartition)上的修改exchange/truncate/drop，在该表上最后一次操作的时间，**PG_STAT_ALL_TABLES**视图last_data_changed列的数据是通过该函数求值，在表数量很大的场景中，通过视图获取表数据最后修改时间的性能较差，建议直接使用该函数获取表数据的最后修改时间。入参是表的oid。

  返回值类型：timestamptz

- pg_stat_set_last_data_changed_time(oid)

  描述：手动设置该表上最后一次insert/update/delete, exchange/truncate/drop partition操作的时间。

  返回值类型：void

- pg_backend_pid()

  描述：当前会话的服务器线程的线程ID。

  返回值类型：integer

- pg_stat_get_activity(integer)

  描述：返回一个关于带有特殊PID的后台进程的记录信息，当参数为NULL时，则返回每个活动的后台进程的记录。返回结果不包含connection_info列。初始用户、系统管理员和MONADMIN可以查看所有的数据，普通用户只能查询自己的结果。

  示例：

```
gaussdb=# select * from pg_stat_get_activity(139881386280704);
 datid |       pid       | sessionid | usesysid | application_name | state  |
query                       | waiting |         xact_start          |          query_start          |
    backend_start         |         state_change          | client_addr | client_hostname | client_port | enqueue
|     query_id      |   srespool   | global_sessionid | unique_sql_id | trace_id
-------+-----------------+-----------+----------+------------------+--------
+-----------------------------------------------------+---------+-------------------------------
+--------------------------------+-----
-------------------------+-----------------------------+-------------+-----------------+-------------+---------
+------------------+--------------+-----------------+---------------+----------
 16545 | 139881386280704 |        69 |       10 | gsql             | active | select * from
pg_stat_get_activity(139881386280704); | f       | 2022-01-18 19:43:05.167718+08 | 2022-01-18
19:43:05.167718+08 | 2022
-01-18 19:42:33.513507+08 | 2022-01-18 19:43:05.16773+08 |              |               |          -1 |         |
72620543991624410 | default_pool | 1938253334#69#0  |    3751941862 |
(1 row)
```

返回值类型：setofrecord

返回参数说明如下。

表 7-100 返回参数说明

返回参数	返回参数类型	返回参数说明
datid	oid	用户会话在后台连接到的数据库OID。
pid	bigint	后台线程ID。
sessionid	bigint	会话ID。

返回参数	返回参数类型	返回参数说明
usesysid	oid	登录该后台的用户OID。
application_name	text	连接到该后台的应用名。
state	text	该后台当前总体状态。
query	text	该后台的最新查询。如果state状态是active（活跃的），此字段显示当前正在执行的查询。所有其他情况表示上一个查询。
waiting	Boolean	如果后台当前正等待锁则为true。
xact_start	timestamp with time zone	启动当前事务的时间，如果没有事务是活跃的，则为null。 如果当前查询是首个事务，则这列等同于query_start列。
query_start	timestamp with time zone	开始当前活跃查询的时间，如果state的值不是active，则这个值是上一个查询的开始时间。如果是存储过程、函数、PACKAGE，则查询的是第一个查询时间，不会随着存储过程内语句运行而改变。
backend_start	timestamp with time zone	该过程开始的时间，即当客户端连接服务器时。
state_change	timestamp with time zone	上次状态改变的时间。
client_addr	inet	连接到该后台的客户端的IP地址。如果此字段是null，它表明通过服务器机器上UNIX套接字连接客户端或者这是内部进程，如autovacuum。

返回参数	返回参数类型	返回参数说明
client_hostname	text	客户端的主机名，这个字段是通过client_addr的反向DNS查找得到。这个字段只有在启动log_hostname且使用IP连接时才非空。
client_port	integer	客户端用于与后台通讯的TCP端口号，如果使用Unix套接字，则为-1。
enqueue	text	该字段暂不支持。
query_id	bigint	查询语句的ID。
srespool	name	资源池名字。
global_sessionid	text	全局会话id。
unique_sql_id	bigint	语句的unique sql id。
trace_id	text	驱动传入的trace id，与应用的一次请求相关联。

- pg_stat_get_activity_with_conninfo(integer)

  描述：返回一个关于带有特殊PID的后台进程的记录信息，当参数为NULL时，则返回每个活动的后台进程的记录。初始用户、系统管理员和MONADMIN可以查看所有的数据，普通用户只能查询自己的结果。

  返回值类型：setofrecord

  返回值说明如下。

  表 7-101 返回值说明

返回值	返回值类型	返回值说明
datid	oid	用户会话在后台连接到的数据库OID。
pid	bigint	后台线程ID。
sessionid	bigint	会话ID。
usesysid	oid	登录该后台的用户OID。
application_name	text	连接到该后台的应用名。
state	text	改后台当前总体状态。

返回值	返回值类型	返回值说明
query	text	该后台的最新查询。如果state状态是active（活跃的），此字段显示当前正在执行的查询。所有其他情况表示上一个查询。
waiting	Boolean	如果后台当前正等待锁则为true。
xact_start	timestamp with time zone	启动当前事务的时间，如果没有事务是活跃的，则为null。如果当前查询是首个事务，则这列等同于query_start列。
query_start	timestamp with time zone	开始当前活跃查询的时间，如果state的值不是active，则这个值是上一个查询的开始时间。如果是存储过程、函数、PACKAGE，则查询的是第一个查询时间，不会随着存储过程内语句运行而改变。
backend_start	timestamp with time zone	该过程开始的时间，即当客户端连接服务器时。
state_change	timestamp with time zone	上次状态改变的时间。
client_addr	inet	连接到该后台的客户端的IP地址。如果此字段是null，它表明通过服务器机器上UNIX套接字连接客户端或者这是内部进程，如autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过client_addr的反向DNS查找得到。这个字段只有在启动log_hostname且使用IP连接时才非空。
client_port	integer	客户端用于与后台通讯的TCP端口号，如果使用Unix套接字，则为-1。
enqueue	text	该字段暂不支持。

返回值	返回值类型	返回值说明
query_id	bigint	查询语句的ID。
connection_info	text	json格式字符串，记录当前连接数据库的驱动类型、驱动版本号、当前驱动的部署路径、进程属主用户等信息。
srespool	name	资源池名字。
global_sessionid	text	全局会话ID。
unique_sql_id	bigint	语句的unique sql id。
trace_id	text	驱动传入的trace id，与应用的一次请求相关联。
top_xid	xid	事务的顶层事务ID。
current_xid	xid	事务的当前事务ID。
xlog_quantity	bigint	事务当前使用的xLog量，单位为字节。

- gs_get_explain(integer)

  描述：返回一个关于指定PID的后台线程的运行态计划，PID不能为空。该功能生效的必要条件是GUC参数track_activities为on，支持语句的范围为可以explain且其计划不包含STREAM算子的SQL，详细说明如下：

  - 若GUC参数plan_collect_thresh=-1，则该函数的返回结果一直为空。
  - 若GUC参数plan_collect_thresh=0，若当前SQL执行时间大于等于GUC参数log_min_duration_statement指定值，且计划中所有算子处理的tuple数量和大于等于10000时，开始收集运行态计划；且后续每当所有算子处理tuple数量的和的增量超过10000时，进行一次收集。
  - 若GUC参数plan_collect_thresh>0，会按照该参数指定的阈值进行增量收集运行态计划。

  返回值类型：text，具体各个字段的类型和含义如下：

参数	类型	描述
运行态计划字符串	text	其中计划字符串中的A-rows是算子实时返回行数。

- pg_stat_get_function_calls(oid)

  描述：函数已被调用次数。

  返回值类型：bigint

- pg_stat_get_function_self_time(oid)

  描述：只有在此函数上所花费的时间。此函数调用其它函数所花费的时间被排除在外。

返回值类型：bigint

- pg_stat_get_backend_idset()

  描述：设置当前活动的服务器进程数（从1到活动服务器进程的数量）。

  返回值类型：setofinteger

- pg_stat_get_backend_pid(integer)

  描述：给定的服务器线程的线程ID。

  返回值类型：bigint

- pg_stat_get_backend_dbid(integer)

  描述：给定服务器进程的数据库ID。

  返回值类型：oid

- pg_stat_get_backend_userid(integer)

  描述：给定服务器进程的用户ID，本函数仅系统管理员可调用。

  返回值类型：oid

- pg_stat_get_backend_activity(integer)

  描述：给定服务器进程的当前活动查询，仅在调用者是系统管理员或被查询会话的用户，并且打开track_activities的时候才能获得结果。

  返回值类型：text

- pg_stat_get_backend_waiting(integer)

  描述：如果给定服务器进程在等待某个锁，并且调用者是系统管理员或被查询会话的用户，并且打开track_activities的时候才返回真。

  返回值类型：Boolean

- pg_stat_get_backend_activity_start(integer)

  描述：给定服务器进程当前正在执行的查询的起始时间，仅在调用者是系统管理员或被查询会话的用户，并且打开track_activities的时候才能获得结果。

  返回值类型：timestampwithtimezone

- pg_stat_get_backend_xact_start(integer)

  描述：给定服务器进程当前正在执行的事务的开始时间，但只有当前用户是系统管理员或被查询会话的用户，并且打开track_activities的时候才能获得结果。

  返回值类型：timestampwithtimezone

- pg_stat_get_backend_start(integer)

  描述：给定服务器进程启动的时间，如果当前用户不是系统管理员或被查询的后端的用户，则返回NULL。

  返回值类型：timestampwithtimezone

- pg_stat_get_backend_client_addr(integer)

  描述：连接到给定客户端后端的IP地址。如果是通过Unix域套接字连接的则返回NULL；如果当前用户不是系统管理员或被查询会话的用户，也返回NULL。

  返回值类型：inet

- pg_stat_get_backend_client_port(integer)

  描述：连接到给定客户端后端的TCP端口。如果是通过Unix域套接字连接的则返回-1；如果当前用户不是系统管理员或被查询会话的用户，也返回NULL。

  返回值类型：integer

- pg_stat_get_bgwriter_timed_checkpoints()
  描述：后台写进程开启定时检查点的时间（因为checkpoint_timeout时间已经过期了）。
  返回值类型：bigint
- pg_stat_get_bgwriter_requested_checkpoints()
  描述：后台写进程开启基于后端请求的检查点的时间，因为已经超过了checkpoint_segments或因为已经执行了CHECKPOINT。
  返回值类型：bigint
- pg_stat_get_bgwriter_buf_written_checkpoints()
  描述：在检查点期间后台写进程写入的缓冲区数目。
  返回值类型：bigint
- pg_stat_get_bgwriter_buf_written_clean()
  描述：为日常清理脏块，后台写进程写入的缓冲区数目。
  返回值类型：bigint
- pg_stat_get_bgwriter_maxwritten_clean()
  描述：后台写进程停止清理扫描的时间，因为已经写入了更多的缓冲区（相比bgwriter_lru_maxpages参数声明的缓冲区数）。
  返回值类型：bigint
- pg_stat_get_buf_written_backend()
  描述：后端进程写入的缓冲区数，因为它们需要分配一个新的缓冲区。
  返回值类型：bigint
- pg_stat_get_buf_alloc()
  描述：分配的总缓冲区数。
  返回值类型：bigint
- pg_stat_clear_snapshot()
  描述：清理当前的统计快照，该函数仅SYSADMIN和MONADMIN可以执行。
  返回值类型：void
- pg_stat_reset()
  描述：为当前数据库重置统计计数器为0（需要系统管理员权限）。
  返回值类型：void
- pg_stat_reset_shared(text)
  描述：重置shared cluster每个节点当前数据统计计数器为0（需要系统管理员权限）。
  返回值类型：void
- pg_stat_reset_single_table_counters(oid)
  描述：为当前数据库中的一个表或索引重置统计为0（需要系统管理员权限）。
  返回值类型：void
- pg_stat_reset_single_function_counters(oid)
  描述：为当前数据库中的一个函数重置统计为0（需要系统管理员权限）。
  返回值类型：void
- fenced_udf_process(integer)

描述：查看本地UDF Master和Work进程数。入参为1时查看master进程数，入参为2时查看worker进程数，入参为3时结束所有worker进程。

返回值类型：text

- total_cpu()

  描述：获取当前节点使用的CPU时间，单位是jiffies。

  返回值类型：bigint

- total_memory()

  描述：获取当前节点使用的虚拟内存大小，单位KB。

  返回值类型：bigint

- pg_stat_bad_block(text, int, int, int, int, int, timestamp with time zone, timestamp with time zone)

  描述：获取当前节点自启动后，读取出现Page的损坏信息。

  例: select * from pg_stat_bad_block();

  返回值类型：record

- pg_stat_bad_block_clear()

  描述：清理节点记录的读取出现的Page损坏信息（需要系统管理员权限）。

  返回值类型：void

- gs_respool_exception_info(pool text)

  描述：查看某个资源池关联的查询规则信息。

  返回值类型：record

- gs_control_group_info(pool text)

  描述：查看资源池关联的控制组信息。该函数需要SYSADMIN权限的用户才能够执行。

  返回值类型：record

  返回信息如下：

属性	属性值	描述
name	class_a:workload_a1	class和workload名称。
class	class_a	Class控制组名称。
workload	workload_a1	Workload控制组名称。
type	DEFWD	控制组类型（Top、CLASS、BAKWD、DEFWD、TSWD）。
gid	87	控制组id。
shares	30	占父节点CPU资源的百分比。
limits	0	占父节点CPU核数的百分比。
rate	0	Timeshare中的分配比例。
cpucores	0-3	CPU核心数。

- gs_prepared_statements()

  描述：显示所有会话所有可用的预备语句，该函数需要SYSADMIN权限的用户才能够执行，函数返回信息具体的字段和GS_ALL_PREPARED_STATEMENTS字段一致。

  返回值类型：record

- gs_all_control_group_info()

  描述：查看数据库内所有的控制组信息。

  返回值类型：record

- gs_get_control_group_info()

  描述：查看所有的控制组信息。函数返回信息具体的字段GS_GET_CONTROL_GROUP_INFO字段。该函数需要SYSADMIN权限的用户才能够执行。

  返回值类型：record

- gs_plan_trace_delete(TIMESTAMPTZ)

  描述：删除当前用户的小于等于指定时间max_time的所有plan trace。所有用户都可以使用该函数，且只要该函数在执行过程中没有任何异常，那么该函数将返回t。

  返回值类型：bool

- gs_plan_trace_watch_sqlid(bigint)

  描述：侦听一个要生成plan trace的unique sql id，该id来自dbe_perf.statement系统表的unique_sql_id字段。另外，该函数只能被初始用户和有SYSADMIN/opradmin/MONADMIN权限的用户调用，且只要该函数在执行过程中没有任何异常，那么该函数将返回t。

  返回值类型：bool

须知

1. 在数据库系统中，当前被侦听的unique sql id被存放到一个长度为128的循环数组中，如果调用该函数的速度过于频繁，那么有可能存在被侦听但是没有生成plan trace的unique sql id被覆盖掉。
2. 如果一个unique sql id只被侦听一次，那么该unique sql id只能生成一次plan trace。如果相同的unique sql id被侦听多次，那么该unique sql id会生成多次plan trace。

- gs_plan_trace_show_sqlids()

  描述：查看当前系统中待生成plan trace的unique sql id字符串列表。该函数只能被初始用户和有SYSADMIN/opradmin/MONADMIN权限的用户调用。

  返回值类型：text

须知

1. 系统中有两个待生成plan trace的unique sq id：730834934、730834935，那么使用该函数看到的结果为字符串文本：“730834934,730834935,”。
2. 使用这个函数看不到正在生成plan trace的unique sql id。

- get_instr_workload_info(integer)
  描述：获取数据库主节点上事务量信息，事务时间信息。
  返回值类型：record

属性	属性值	描述
resourcepool_oid	10	资源池的oid(逻辑同负载等价)。
commit_counter	4	前端事务commit数量。
rollback_counter	1	前端事务rollback数量。
resp_min	949	前端事务最小响应时间（单位：微秒）。
resp_max	201891	前端事务最大响应时间（单位：微秒）。
resp_avg	43564	前端事务平均响应时间(单位：微秒)。
resp_total	217822	前端事务总响应时间（单位：微秒）。
bg_commit_counter	910	后端事务commit数量。
bg_rollback_counter	0	后端事务rollback数量。
bg_resp_min	97	后端事务最小响应时间（单位：微秒）。
bg_resp_max	678080687	后端事务最大响应时间（单位：微秒）。
bg_resp_avg	327847884	后端事务平均响应时间（单位：微秒）。
bg_resp_total	298341575300	后端事务总响应时间（单位：微秒）。

- pv_instance_time()
  描述：获取当前节点上各个关键阶段的时间消耗。
  返回值类型：record

Stat_name属性	属性值	描述
DB_TIME	1062385	所有线程端到端的墙上时间（WALL TIME）消耗总和(单位： 微秒)。
CPU_TIME	311777	所有线程CPU时间消耗总和(单位：微秒)。
EXECUTION_TIME	380037	消耗在执行器上的时间总和(单位：微秒)。

Stat_name属性	属性值	描述
PARSE_TIME	6033	消耗在SQL解析上的时间总和(单位：微秒)。
PLAN_TIME	173356	消耗在执行计划生成上的时间总和(单位： 微秒)。
REWRITE_TIME	2274	消耗在查询重写上的时间总和(单位： 微秒)。
PL_EXECUTION_TIME	0	消耗在PL/SQL执行上的时间总和(单位： 微秒)。
PL_COMPILATION_TIME	557	消耗在SQL编译上的时间总和(单位：微秒)。
NET_SEND_TIME	1673	消耗在网络发送上的时间总和(单位： 微秒)。
DATA_IO_TIME	426622	消耗在数据读写上的时间总和(单位： 微秒)。

- DBE_PERF.get_global_instance_time()

  描述：提供整个数据库各个关键阶段的时间消耗，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- get_instr_unique_sql()

  描述：获取当前节点的执行语句（归一化SQL）信息，查询该函数必须具有SYSADMIN权限或者MONADMIN权限。

  返回值类型：record

- reset_unique_sql(text, text, bigint)

  描述：用来清理CN/DN内存中的Unique SQL（需要SYSADMIN/MONADMIN权限）。

  返回值类型：Boolean

  表 7-102 reset_unique_sql 参数说明

参数	类型	描述
scope	text	清理范围类型： 'GLOBAL' - 清理所有的CN/DN节点，如果是'GLOBAL'，则只可以为CN节点执行此函数。 'LOCAL' - 清理本节点。
clean_type	text	'BY_USERID' - 按用户ID来进行清理Unique SQL。 'BY_CNID' - 按CN的ID来进行清理Unique SQL。 'ALL' - 全部清理。

参数	类型	描述
clean_value	int8	具体清理type对应的清理值。如果第二个参数为ALL，则第三个参数不起作用，可以取任意值。

说明

此函数中所说节点指分布式节点，当前GaussDB为集中式数据库，global与local功能一致，取值范围不支持BY_CNID。

- get_instr_wait_event(NULL)

  描述：获取当前节点event等待的统计信息。

  返回值类型：record

- get_instr_user_login()

  描述：获取当前节点的用户登录/退出次数信息，查询该函数必须具有SYSADMIN或者MONADMIN权限。

  返回值类型：record

- get_instr_rt_percentile(integer)

  描述：获取数据库SQL响应时间P80、P95分布信息。

  返回值类型：record

- get_node_stat_reset_time()

  描述：获取当前节点的统计信息重置（重启，主备倒换，数据库删除）时间。

  返回值类型：record

- DBE_PERF.get_global_os_runtime()

  描述：显示当前操作系统运行的状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_os_threads()

  描述：提供整个数据库中所有正常节点下的线程状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_workload_sql_count()

  描述：提供整个数据库中不同负载SELECT，UPDATE，INSERT，DELETE，DDL，DML，DCL计数信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_workload_sql_elapse_time()

  描述：提供整个数据库中不同负载SELECT、UPDATE、INSERT和DELETE，响应时间信息（TOTAL、AVG、MIN、MAX），查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_workload_transaction()

  描述：获取数据库内所有节点上的事务量信息，事务时间信息，查询该函数必须具有MONADMIN权限。

返回值类型：record

- DBE_PERF.get_global_session_stat()

  描述：获取数据库节点上的会话状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

  说明

  状态信息有14项：commit、rollback、sql、table_scan、blocks_fetched、physical_read_operation、shared_blocks_dirtied、local_blocks_dirtied、shared_blocks_read、local_blocks_read、blocks_read_time、blocks_write_time、sort_imemory、sort_idisk。

- DBE_PERF.get_global_session_time()

  描述：提供整个数据库各节点各个关键阶段的时间消耗，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_session_memory()

  描述：汇聚各节点的Session级别的内存使用情况，包含执行作业在数据节点上gaussdb线程和Stream线程分配的所有内存，单位为MB，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_session_memory_detail()

  描述：汇聚各节点的线程的内存使用情况，以MemoryContext节点来统计，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- create_wlm_session_info(int flag)

  描述：将当前内存中记录的TopSQL查询语句级别相关统计信息清理。该函数只有管理员用户可以执行。

  返回值类型：int

- pg_stat_get_wlm_session_info(int flag)

  描述：获取当前内存中记录的TopSQL查询语句级别相关统计信息，当传入的参数不为0时，会将这部分信息从内存中清理掉。该函数只有SYSADMIN和MONADMIN用户可以执行。

  返回值类型：record

- gs_paxos_stat_replication()

  描述：在主机端查询备机信息。

  返回值类型：setofrecord

  返回字段说明如下：

字段名称	字段类型	字段说明
local_role	text	发送日志节点的角色。
peer_role	text	接收日志节点的角色。
local_dcf_role	text	发送日志节点的DCF角色。

peer_dcf_role	text	接收日志节点的DCF角色。
peer_state	text	接收日志节点的状态。
sender_write_location	text	发送日志节点写到xlog buffer的位置。
sender_commit_location	text	发送日志节点DCF日志达成一致性点。
sender_flush_location	text	发送日志节点写到xlog disk的位置。
sender_replay_location	text	发送日志节点replay的位置。
receiver_write_location	text	接收日志节点写到xlog buffer的位置。
receiver_commit_location	text	接收日志节点DCF日志达成一致性点。
receiver_flush_location	text	接收日志节点写到xlog disk的位置。
receiver_replay_location	text	接收日志节点重演xlog的位置。
sync_percent	text	同步百分比。
dcf_run_mode	int4	DCF同步模式。
channel	text	信道信息。

- gs_wlm_get_resource_pool_info(int)

  描述：获取所有用户的资源使用统计信息，入参为int类型可以为任意int值或NULL。

  返回值类型：record
- gs_wlm_get_all_user_resource_info()

  描述：获取所有用户的资源使用统计信息。

  返回值类型：record
- gs_wlm_get_user_info(int)

  描述：获取所有用户的相关信息，入参为int类型，可以为任意int值或NULL。该函数只有SYSADMIN权限的用户可以执行。

  返回值类型：record
- gs_wlm_readjust_user_space(oid)

  描述：修正所有用户的存储空间使用情况。该函数只有管理员用户可以执行。

  返回值类型：record
- gs_wlm_readjust_user_space_through_username(text name)

  描述：修正指定用户的存储空间使用情况。该函数普通用户只能修正自己的使用情况，只有管理员用户可以修正所有用户的使用情况。当name指定为“0000”，表示需要修正所有用户的使用情况。

返回值类型：record

- gs_wlm_readjust_user_space_with_reset_flag(text name, boolean isfirst)

  描述：修正指定用户的存储空间使用情况。入参isfirst为true表示从0开始统计，否则从上一次结果继续统计。该函数普通用户只能修正自己的使用情况，只有管理员用户可以修正所有用户的使用情况。当name指定为“0000”，表示需要修正所有用户的使用情况。

  返回值类型：record

- gs_wlm_session_respool(bigint)

  描述：获取当前所有后台线程的session resource pool相关信息，入参为bigint类型可以为任意bigint值或NULL。

  返回值类型：record

- gs_wlm_get_session_info()

  描述：目前该接口已废弃，暂不可用。

- gs_wlm_get_user_session_info()

  描述：目前该接口已废弃，暂不可用。

- gs_io_wait_status()

  描述：目前该接口不支持单机和集中式，暂不可用。

- global_stat_get_hotkeys_info()

  描述：获取整个数据库实例中热点key的统计情况。目前该接口不支持单机和集中式，暂不可用。

- global_stat_clean_hotkeys()

  描述：清理整个数据库实例中热点key的统计信息。目前该接口不支持单机和集中式，暂不可用。

- DBE_PERF.get_global_session_stat_activity()

  描述：汇聚数据库内各节点上正在运行的线程相关的信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_thread_wait_status()

  描述：汇聚所有节点上工作线程（backend thread）以及辅助线程（auxiliary thread）的阻塞等待情况，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_operator_history_table()

  描述：汇聚当前用户数据库主节点上执行作业结束后的算子相关记录（持久化），查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_operator_history()

  描述：汇聚当前用户数据库主节点上执行作业结束后的算子相关记录，查询该函数必须具有SYSADMIN和MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_operator_runtime()

  描述：汇聚当前用户数据库主节点上执行作业实时的算子相关记录，查询该函数必须具有SYSADMIN和MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_statement_complex_history()

  描述：汇聚当前用户数据库主节点上复杂查询的历史记录，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_statement_complex_history_table()

  描述：汇聚当前用户数据库主节点上复杂查询的历史记录（持久化），查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_statement_complex_runtime()

  描述：汇聚当前用户数据库主节点上复杂查询的实时信息，查询该函数必须具有SYSADMIN和MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_memory_node_detail()

  描述：汇聚所有节点某个数据库节点内存使用情况，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_shared_memory_detail()

  描述：汇聚所有节点已产生的共享内存上下文的使用信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_statio_all_indexes()

  描述：汇聚所有节点当前数据库中的每个索引行，显示特定索引的I/O的统计，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_all_tables()

  描述：显示汇聚各节点数据中每个表（包括TOAST表）的一行的统计信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_all_tables()

  描述：显示各节点数据中每个表（包括TOAST表）的一行的统计信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_local_toastname_and_toastindexname()

  描述：提供本地toast表的name和index和其关联表的对应关系，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_statio_all_indexes()

  描述：统计所有节点当前数据库中的每个索引行，显示特定索引的I/O的统计，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_statio_all_sequences()

  描述：提供命名空间中所有sequences的IO状态信息，查询该函数必须具有MONADMIN权限。

返回值类型：record

- DBE_PERF.get_global_statio_all_tables()

  描述：汇聚各节点的数据库中每个表I/O的统计，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_statio_all_tables()

  描述：统计数据库内数据库中每个表I/O的统计，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_local_toast_relation()

  描述：提供本地toast表的name和其关联表的对应关系，查询该函数必须具有MONADMIN权限

  返回值类型：record

- DBE_PERF.get_global_statio_sys_indexes()

  描述：汇聚各节点的命名空间中所有系统表索引的IO状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_statio_sys_indexes()

  描述：统计各节点的命名空间中所有系统表索引的IO状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_statio_sys_sequences()

  描述：提供命名空间中所有系统表为sequences的IO状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_statio_sys_tables()

  描述：提供各节点的命名空间中所有系统表的IO状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_statio_sys_tables()

  描述：数据库内汇聚命名空间中所有系统表的IO状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_statio_user_indexes()

  描述：各节点的命名空间中所有用户关系表索引的IO状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_statio_user_indexes()

  描述：数据库内汇聚命名空间中所有用户关系表索引的IO状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_statio_user_sequences()

描述：显示各节点的命名空间中所有用户的sequences的IO状态信息，查询该函数必须具有MONADMIN权限。

返回值类型：record

- DBE_PERF.get_global_statio_user_tables()

  描述：显示各节点的命名空间中所有用户关系表的IO状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_statio_user_tables()

  描述：数据库内汇聚命名空间中所有用户关系表的IO状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_all_indexes()

  描述：汇聚所有节点数据库中每个索引的统计信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_all_indexes()

  描述：统计所有节点数据库中每个索引的统计信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_sys_tables()

  描述：汇聚各节点pg_catalog、information_schema模式的所有命名空间中系统表的统计信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_sys_tables()

  描述：统计各节点pg_catalog、information_schema模式的所有命名空间中系统表的统计信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_sys_indexes()

  描述：汇聚各节点pg_catalog、information_schema模式中所有系统表的索引状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_sys_indexes()

  描述：统计各节点pg_catalog、information_schema模式中所有系统表的索引状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_user_tables()

  描述：汇聚所有命名空间中用户自定义普通表的状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_user_tables()

  描述：统计所有命名空间中用户自定义普通表的状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_user_indexes()

  描述：汇聚所有数据库中用户自定义普通表的索引状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_user_indexes()

  描述：统计所有数据库中用户自定义普通表的索引状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_database()

  描述：汇聚所有节点数据库统计信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_database_conflicts()

  描述：统计所有节点数据库统计信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_xact_all_tables()

  描述：汇聚命名空间中所有普通表和toast表的事务状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_xact_all_tables()

  描述：统计命名空间中所有普通表和toast表的事务状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_xact_sys_tables()

  描述：汇聚所有节点命名空间中系统表的事务状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_xact_sys_tables()

  描述：统计所有节点命名空间中系统表的事务状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_xact_user_tables()

  描述：汇聚所有节点命名空间中用户表的事务状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_xact_user_tables()

  描述：统计所有节点命名空间中用户表的事务状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_user_functions()

  描述：汇聚所有节点命名空间中用户定义函数的事务状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_xact_user_functions()

  描述：统计所有节点命名空间中用户定义函数的事务状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_bad_block()

  描述：汇聚所有节点表、索引等文件的读取失败信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_file_redo_iostat()

  描述：统计所有节点表、索引等文件的读取失败信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_file_iostat()

  描述：汇聚所有节点数据文件IO的统计，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_locks()

  描述：汇聚所有节点的锁信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_replication_slots()

  描述：汇聚所有节点上逻辑复制信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.GET_GLOBAL_PARALLEL_DECODE_STATUS()

  描述：汇聚当前节点上的复制槽的并行解码信息，查询该函数必须具有MONADMIN权限。返回值同视图**DBE_PERF.GLOBAL_PARALLEL_DECODE_STATUS**。

  返回值类型：record

- DBE_PERF.GET_GLOBAL_PARALLEL_DECODE_THREAD_INFO()

  描述：汇聚当前节点上的复制槽的并行解码线程信息，查询该函数必须具有MONADMIN权限。返回值同视图**DBE_PREF.GLOBAL_PARALLEL_DECODE_THREAD_INFO**。

  返回值类型：record

- DBE_PERF.get_global_bgwriter_stat()

  描述：汇聚所有节点后端写进程活动的统计信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_replication_stat()

  描述：汇聚各节点日志同步状态信息，如发起端发送日志位置，收端接收日志位置等，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_transactions_running_xacts()

  描述：汇聚各节点运行事务的信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_transactions_running_xacts()

  描述：统计各节点运行事务的信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_transactions_prepared_xacts()

  描述：汇聚各节点当前准备好进行两阶段提交的事务的信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_transactions_prepared_xacts()

  描述：统计各节点当前准备好进行两阶段提交的事务的信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_statement()

  描述：汇聚各节点历史执行语句状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_statement_count()

  描述：汇聚各节点SELECT、UPDATE、INSERT、DELETE，响应时间信息（TOTAL、AVG、MIN、MAX），查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_config_settings()

  描述：汇聚各节点GUC参数配置信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_wait_events()

  描述：汇聚各节点wait events状态信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_statement_responsetime_percentile()

  描述：获取数据库SQL响应时间P80，P95分布信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_user_login()

  描述：统计数据库各节点用户登录/退出次数信息，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_record_reset_time()

  描述：汇聚数据库统计信息重置（重启，主备倒换，数据库删除）时间，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.standby_statement_history(only_slow[, time1, time2])

  描述：备机中用来查询FULL SQL记录的函数，主机通过表statement_history进行查询，备机通过此函数进行查询。查询该函数必须具有MONADMIN权限。

  参数：参见**表7-103**。

返回值类型：record，同表statement_history。

表 7-103 standby_statement_history 参数说明

参数	类型	描述
only_slow	bool	是否仅查询慢SQL。 true为是，相当于 select .. where is_slow_sql = true;。 false或NULL为查询全部SQL，相当于不对is_slow_sql进行过滤。
time1	timestamptz	可选输入，表示查询SQL的finish_time的最小时间。
time2	timestamptz	可选输入，表示查询SQL的finish_time的最大时间。

说明

- 两个时间参数time1、time2的说明：表示查询的SQL的finish_time所属时间段，分别表示起始与终止时间，输入NULL或者不输入表示没有限制, 功能等同于select .. where finish_time between time1 and time2;。
- 备机上该功能的数据并非存在表里，不存在start_time列的索引，推荐使用参数对finish_time进行条件查找。
- 由于备机Full/Slow SQL采用异步下盘方式，所以用户SQL信息存储时刻可能有所滞后，建议客户查询此接口适当扩大时间查询范围。

- DBE_PERF.track_memory_context(context_list text)

  描述：设置需要统计内存申请详细信息的内存上下文。入参为内存上下文的名称，使用“,”分隔，如“ThreadTopMemoryContext, SessionCacheMemoryContext”，注意该内存上下文名称是上下文敏感的。此外，单个内存上下文的长度为63，超过的部分会被截断。而且一次能够统计的内存上下文上限为16个，设置超过16个内存上下文会设置失败。每一次调用该函数都会将上次统计的结果清空，当入参指定为""时，表示取消该统计功能。查询该函数必须具有MONADMIN权限。

  返回值类型：boolean

- DBE_PERF.track_memory_context_detail()

  描述：获取DBE_PERF.track_memory_context函数指定的内存上下文的内存申请详细信息。返回值的定义见视图DBE_PERF.track_memory_context_detail。查询该函数必须具有MONADMIN权限。

  返回值类型：record

- pg_stat_get_mem_mbytes_reserved(tid)

  描述：统计资源管理相关变量值，仅用于定位问题使用。

  参数：线程id。

  返回值类型：text

- gs_wlm_user_resource_info(name text)

  描述：查询具体某个用户的资源限额和资源使用情况。

  返回值类型：record

- pg_stat_get_file_stat()

  描述：通过对数据文件IO的统计，反映数据的IO性能，用以发现IO操作异常等性能问题。

  返回值类型：record

- pg_stat_get_redo_stat()

  描述：用于统计会话线程日志回放情况。

  返回值类型：record

- pg_stat_get_status(int8)

  描述：可以检测当前实例中工作线程（backend thread）以及辅助线程（auxiliary thread）的阻塞等待情况。

  返回值类型：record

- get_local_rel_iostat()

  描述：查询当前节点的数据文件IO状态累计值。

  返回值类型：record

- DBE_PERF.get_global_rel_iostat()

  描述：汇聚所有节点数据文件IO的统计，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.global_threadpool_status()

  描述：显示在所有节点上的线程池中工作线程及会话的状态信息。函数返回信息具体字段**GLOBAL_THREADPOOL_STATUS**字段，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- pv_os_run_info()

  描述：显示当前操作系统运行的状态信息，具体字段信息参考**GS_OS_RUN_INFO**。

  参数：nan

  返回值类型：setof record

- pv_session_stat()

  描述：以会话线程或AutoVacuum线程为单位，统计会话状态信息，具体字段信息参考**GS_SESSION_STAT**。

  参数：nan

  返回值类型：setof record

- pv_session_time()

  描述：用于统计会话线程的运行时间信息，及各执行阶段所消耗时间，具体字段信息参考**GS_SESSION_TIME**。

  参数：nan

  返回值类型：setof record

- pg_stat_get_db_temp_bytes()

  描述：用于统计通过数据库查询写入临时文件的数据总量。计算所有临时文件，不论为什么创建临时文件，而且不管log_temp_files设置。

  参数：oid

返回值类型：bigint

- pg_stat_get_db_temp_files()

  描述：通过数据库查询创建的临时文件数量。计算所有临时文件，不论为什么创建临时文件（比如排序或者哈希），而且不管log_temp_files设置。

  参数：oid

  返回值类型：bigint

- create_wlm_instance_statistics_info()

  描述：将当前实例的历史监控数据进行持久化保存。

  参数：nan

  返回值类型：integer

- remote_candidate_stat()

  描述：用于显示数据库所有实例的检查点信息和各类日志刷页情况（本节点除外），集中式不支持。

  返回值类型：record

- remote_ckpt_stat()

  描述：用于显示数据库所有实例的检查点信息和各类日志刷页情况（本节点除外），集中式不支持。

  返回值类型：record

- remote_single_flush_dw_stat()

  描述：显示数据库所有实例的单页面双写文件的情况(本节点除外)，集中式不支持。

  返回值类型：record

- remote_double_write_stat()

  描述：显示数据库所有实例的双写文件的情况(本节点除外)，集中式不支持。

  返回值类型：record

- remote_pagewriter_stat()

  描述：显示数据库所有实例的刷页信息和检查点信息(本节点除外)，集中式不支持。

  返回值类型：record

- remote_recovery_status()

  描述：显示关于主机和备机的日志流控信息(本节点除外)，集中式不支持。

  返回值类型：record

- remote_redo_stat()

  描述：显示所有实例的日志回放情况(本节点除外)，集中式不支持。

  返回值类型：record

- DBE_PERF.gs_stat_activity_timeout(int)

  描述：获取当前节点上执行时间超过超时阈值的查询作业信息。需要GUC参数track_activities设置为on才能正确返回结果。超时阈值的取值范围是0~2147483，查询该函数必须具有MONADMIN权限。

  返回值类型：setof record

名称	类型	描述
database	name	用户会话连接的数据库名称。
pid	bigint	后台线程ID。
sessionid	bigint	会话ID。
usesysid	oid	登录该后台的用户OID。
application_name	text	连接到该后台的应用名。
query	text	该后台正在执行的查询。
xact_start	timestamptz	启动当前事务的时间。
query_start	timestamptz	开始当前查询的时间。如果是存储过程、函数、PACKAGE，则查询的是第一个查询时间，不会随着存储过程内语句运行而改变。
query_id	bigint	查询语句ID。

- gs_wlm_user_resource_info(name text)

  描述：查询具体某个用户的资源限额和资源使用情况。普通用户只能查询到自己相关的信息，管理员权限的用户可以查看全部用户的信息。

  返回值类型：record

- create_wlm_instance_statistics_info

  描述：将当前实例的历史监控数据进行持久化保存。

  参数：nan

  返回值类型：integer

- create_wlm_operator_info(int flag)

  描述：将当前内存中记录的TopSQL算子级别相关统计信息清理，当传入的参数大于0时，会将这部分信息归档到gs_wlm_operator_info中，否则不会归档。该函数只有SYSADMIN权限的用户可以执行。

  返回值类型：int

- GS_ALL_NODEGROUP_CONTROL_GROUP_INFO(text)

  描述：提供了所有逻辑数据库实例的控制组信息。该函数在调用的时候需要指定要查询逻辑数据库实例的名称。例如要查询'installation'逻辑数据库实例的控制组信息：

  ```
  SELECT * FROM GS_ALL_NODEGROUP_CONTROL_GROUP_INFO('installation')
  ```

  返回值类型：record

  函数返回字段如下：

名称	类型	描述
name	text	控制组的名称。
type	text	控制组的类型。

名称	类型	描述
gid	bigint	控制组ID。
classgid	bigint	Workload所属Class的控制组ID。
class	text	Class控制组。
workload	text	Workload控制组。
shares	bigint	控制组分配的CPU资源配额。
limits	bigint	控制组分配的CPU资源限额。
wdlevel	bigint	Workload控制组层级。
cpucores	text	控制组使用的CPU核的信息。

- gs_total_nodegroup_memory_detail
  描述：返回当前数据库逻辑数据库使用内存的信息，单位为MB得到一个逻辑数据库。
  返回值类型：setof record
- local_redo_time_count()
  描述：返回本节点各个回放线程的各个流程的耗时统计（仅在备机上有有效数据）。
  返回值如下：
  local_redo_time_count返回参数说明。

字段名	描述
thread_name	线程名字。

字段名	描述
step1_total	step1的总时间，每个线程对应的流程如下： ● 极致RTO： redo batch：从队列中获取一条日志。 redo manager：从队列中获取一条日志。 redo worker：从队列中获取一条日志。 trxn manager：从队列中读取一条日志。 trxn worker：从队列中读取一条日志。 read worker：从文件中读取一次xlog page（整体）。 read page worker：从队列中获取一个日志。 startup：从队列中获取一个日志。 ● 并行回放： page redo：从队列中获取一条日志。 startup：读取一条日志。
step1_count	step1的统计次数。

字段名	描述
step2_total	step2的总时间，每个线程对应的流程如下： ● 极致RTO： redo batch：处理日志（整体）。 redo manager：处理日志（整体）。 redo worker：处理日志（整体）。 trxn manager：处理日志（整体）。 trxn worker：处理日志（整体）。 read worker：读取xlog page耗时。 read page worker：生成和发送lsn forwarder。 startup：check stop(是否回放到指定位置）。 ● 并行回放： page redo：处理日志（整体）。 startup：check stop（是否回放到指定位置）。
step2_count	step2的统计次数。
step3_total	step3的总时间，每个线程对应的流程如下： ● 极致RTO： redo batch：更新standbystate。 redo manager：数据日志处理。 redo worker：回放page页日志（整体）。 trxn manager：更新flush lsn。 trxn worker：回放日志处理。 read worker：推进xlog segment。 read page worker：获取一个新的item。 startup：redo delay（延迟回放特性等待时间）。 ● 并行回放： page redo：更新standbystate。 startup：redo delay（延迟回放特性等待时间）。
step3_count	step3的统计次数。

字段名	描述
step4_total	step4的总时间，每个线程对应的流程如下： ● 极致RTO： redo batch：解析xLog。 redo manager：DDL处理。 redo worker：读取数据page页。 trxn manager：同步等待时间。 trxn worker：更新本线程lsn。 read page worker：将日志放入分发线程。 startup：分发（整体）。 ● 并行回放： page redo：undo 日志回放。 startup：分发（整体）。
step4_count	step4的统计次数。
step5_total	step5的总时间，每个线程对应的流程如下： ● 极致RTO： redo batch：分发给redo manager。 redo manager：分发给redo worker。 redo worker：回放数据page页的日志。 trxn manager：分发给trxn worker。 trxn worker：强同步wait时间。 read page worker：更新本线程lsn。 startup：日志decode。 ● 并行回放： page redo：share trxn 日志回放。 startup：日志回放。
step5_count	step5的统计次数。

字段名	描述
step6_total	step6的总时间，每个线程对应的流程如下： ● 极致RTO： redo worker：回放非数据页page日志。 trxn manager：全局lsn更新。 redo manager：数据页面日志存入hash表日志crc校验。 ● 并行回放： page redo：synctrxn日志回放。 startup：强同步等待。
step6_count	step6的统计次数。
step7_total	step7的总时间，每个线程对应的流程如下： ● 极致RTO： redo manager：创建表空间。 redo worker：fsm更新。 ● 并行回放： page redo：single日志回放。
step7_count	step7的统计次数。
step8_total	step8的总时间，每个线程对应的流程如下： ● 极致RTO： redo worker：强同步等待。 ● 并行回放： page redo：all workers do日志回放。
step8_count	step8的统计次数。
step9_total	step9的总时间，每个线程对应的流程如下： ● 极致RTO： redo manager：分发日志给page redo线程。 ● 并行回放： page redo：muliti workers do日志回放。
step9_count	step9的统计次数。

- local_xlog_redo_statics()
  描述：返回本节点已经回放的各个类型的日志统计信息（仅在备机上有有效数据）。

返回值如下：

表 7-104 local_xlog_redo_statics 返回参数说明

字段名	描述
xlog_type	日志类型。
rmid	resource manager id。
info	xlog operation。
num	日志个数。
extra	针对page回放日志和xact日志有有效值。 • 如果是page页回放日志类型则表示从磁盘读取page的个数。 • 如果是xact日志类型则表示删除文件的个数。

- gs_get_shared_memctx_detail(text)

  描述：返回指定内存上下文上的内存申请的详细信息，包含每一处内存申请所在的文件、行号和大小（同一文件同一行大小会做累加）。只支持查询通过pg_shared_memory_detail视图查询出来的内存上下文，入参为内存上下文名称（即pg_shared_memory_detail返回结果的contextname列）。查询该函数必须具有SYSADMIN权限或者MONADMIN权限。

  返回值类型：setof record

名称	类型	描述
file	text	申请内存所在文件的文件名。
line	int8	申请内存所在文件的代码行号。
size	int8	申请的内存大小，同一文件同一行多次申请会做累加。

  说明

  该视图不支持release版本小型化场景。

- gs_get_session_memctx_detail(text)

  描述：返回指定内存上下文上的内存申请的详细信息，包含每一处内存申请所在的文件、行号和大小（同一文件同一行大小会做累加）。仅在线程池模式下生效。只支持查询通过gs_session_memory_context视图查询出来的内存上下文，入参为内存上下文名称（即gs_session_memory_context返回结果的contextname列）。查询该函数必须具有SYSADMIN权限或者MONADMIN权限。

  返回值类型：setof record

名称	类型	描述
file	text	申请内存所在文件的文件名。
line	int8	申请内存所在文件的代码行号。
size	int8	申请的内存大小，单位为byte，同一文件同一行多次申请会做累加。

说明

该视图仅在线程池模式下生效，且该视图不支持release版本小型化场景。

- gs_get_history_memory_detail(cstring)

  描述：查询历史内存快照信息，入参类型为cstring，取值为NULL或内存快照log文件名称：

  a. 若入参为NULL，则显示当前节点所有的内存快照log文件列表。
  b. 若入参为a查询到的列表中的内存快照log名称，则显示该log文件记录的内存快照详细信息。
  c. 若输入其他入参，则会提示入参错误或打开文件失败。

  查询该函数必须具有SYSADMIN权限或者MONADMIN权限。

  返回值类型：text

名称	类型	描述
memory_info	text	内存信息，如果函数入参为NULL，该列显示内存快照文件列表信息；入参为内存快照文件名称，则显示该文件的具体内容。

- gs_stack()

  描述：显示线程调用栈。查询该函数需要有SYSADMIN权限或者MONADMIN权限。

  参数：tid，线程id。tid是可选参数，指定tid参数时，函数返回tid对应线程调用栈；当不指定tid参数时，函数返回所有线程的调用栈。

  返回值：当指定tid时，返回值为text；当不指定tid时，返回值为setof record。

  示例：

  select * from gs_stack(tid)获取指定线程调用栈。

```
gaussdb=# select * from gs_stack(139663481165568);
                          gs_stack
------------------------------------------------------------------
 __poll + 0x2d                                                   +
 WaitLatchOrSocket(Latch volatile*, int, int, long) + 0x29f      +
 WaitLatch(Latch volatile*, int, long) + 0x2e                    +
 JobScheduleMain() + 0x90f                                       +
 int GaussDbThreadMain<(knl_thread_role)9>(knl_thread_arg*) + 0x456+
 InternalThreadFunc(void*) + 0x2d                                +
 ThreadStarterFunc(void*) + 0xa4                                 +
 start_thread + 0xc5                                             +
 clone + 0x6d                                                    +
(1 row)
```

  select * from gs_stack()获取所有线程的调用栈。


```
gaussdb=# select * from gs_stack();
-[ RECORD
1 ]----------------------------------------------------------------------------------------------------------
tid   | 139670364324352
lwtid | 308
stack | __poll + 0x2d
      | CommWaitPollParam::caller(int (*)(pollfd*, unsigned long, int), unsigned long) + 0x34
      | int comm_socket_call<CommWaitPollParam, int (*)(pollfd*, unsigned long,
int)>(CommWaitPollParam*, int (*)(pollfd*, unsigned long
, int)) + 0x28
      | comm_poll(pollfd*, unsigned long, int) + 0xb1
      | ServerLoop() + 0x72b
      | PostmasterMain(int, char**) + 0x314e
      | main + 0x617
      | __libc_start_main + 0xf5
      | 0x55d38f8db3a7
[ RECORD 2 ]----------------------------------------------------------------------------------------------------
tid   | 139664851859200
lwtid | 520
stack | __poll + 0x2d
      | WaitLatchOrSocket(Latch volatile*, int, int, long) + 0x29f
      | SysLoggerMain(int) + 0xc86
      | int GaussDbThreadMain<(knl_thread_role)17>(knl_thread_arg*) + 0x45d
      | InternalThreadFunc(void*) + 0x2d
      | ThreadStarterFunc(void*) + 0xa4
      | start_thread + 0xc5
      | clone + 0x6d
```

- gs_get_thread_memctx_detail(tid,text)

  描述：返回指定内存上下文上的内存申请的详细信息，包含每一处内存申请所在的文件、行号和大小（同一文件同一行大小会做累加）。只支持查询通过gs_thread_memory_context视图查询出来的内存上下文，第一个入参为线程id（即gs_thread_memory_context返回数据的tid列），第二个参数为内存上下文名称（即gs_thread_memory_context返回数据的contextname列）。查询该函数必须具有SYSADMIN权限或者MONADMIN权限。

  返回值类型：setof record

名称	类型	描述
file	text	申请内存所在文件的文件名。
line	int8	申请内存所在文件的代码行号。
size	int8	申请的内存大小，单位为byte，同一文件同一行多次申请会做累加。

  说明

  该视图不支持release版本小型化场景。

- gs_tpworker_execstmt_stat()

  描述：描述语句的运行时信息，SYSADMIN和MONADMIN用户执行则显示全部正在执行的语句的信息，普通用户查询只能查询自己执行的SQL语句的信息。

  返回值类型：setof record

名称	类型	描述
db_oid	oid	用户会话在后台连接到的数据库OID。

名称	类型	描述
db_name	name	用户会话在后台连接到的数据库名称。
threadpool_worker	varchar	线程所属的numagroup和线程的ID，格式如下：numagroup_threadid。
thread_id	bigint	线程ID。
session_id	bigint	会话ID。
query_id	bigint	正在执行的SQL语句的ID。
query_text	text	正在执行的SQL语句内容。
unique_sql_id	bigint	SQL语句生成的唯一id。
client_hostname	text	客户端的主机名，这个字段是通过client_addr的反向DNS查找得到。这个字段只有在启动log_hostname且使用IP连接时才非空。
client_app_name	text	客户端app的名字。
stmt_slow_time_threshold	int	单位毫秒，SQL语句被标记为慢SQL的预设超时时间。
stmt_start_time	timestamp with time zone	语句执行的开始时间。如果是存储过程、函数、PACKAGE，则查询的是第一个查询时间，不会随着存储过程内语句运行而改变。
stmt_elapse_time	int	距离查询开始执行时已经过去的时间。
stmt_control_status	varchar	当前语句状态： ● Waiting：等待状态，session接入未获得线程执行。 ● Running：当前语句正常执行。 ● Control：当前语句进入了资源管控阶段。
stmt_control_rule	text	当前语对应的慢SQL管控规则。
stmt_control_iostat	text	当前语句的iops值和最大iops上限。格式如下： curVal/maxVal
stmt_control_memstat	text	当前字段预留，暂不支持。
stmt_control_cpustat	text	当前字段预留，暂不支持。

名称	类型	描述
stmt_control_netstat	text	当前字段预留，暂不支持。

- gs_tpworker_execslot_stat()

  描述：描述线程的运行时信息，SYSADMIN和MONADMIN用户执行则显示全部线程的信息，普通用户查询只能查询自己执行的SQL语句所在的线程的信息。

  返回值类型：setof record

名称	类型	描述
numagroup	int	当前线程所属的numagroup。
worker_id	int	当前线程的ID。
worker_bind_type	text	线程绑定方式numabind, cpubind, allbind, nobind。
worker_cpu_affinity	text	线程和CPU的亲和性，即线程可以调度的cpu核数范围。
worker_status	varchar	当前线程状态： ● Waiting：等待状态，session接入未获得线程执行。 ● Running：当前语句正常执行。 ● Control：当前语句进入了资源管控阶段。
served_query_id	bigint	正在执行的SQL语句的ID。
served_query_text	text	正在执行的SQL语句内容。

- gs_session_all_settings(sessionid bigint)

  描述：查询本节点上sessionid对应的session的全量GUC参数配置。需要SYSADMIN或者MONADMIN权限执行。

  入参说明：sessionid，会话ID。

  返回值类型：setof record

  函数返回字段说明如下：

名称	类型	描述
name	text	参数名称。
setting	text	参数当前值。
unit	text	参数的隐式单位。

  示例：

```
gaussdb=# select sessionid from pg_stat_activity where usename = 'testuser';
 sessionid
-----------
    788861
(1 row)

gaussdb=# select * from gs_session_all_settings(788861) where name = 'work_mem';
   name   | setting | unit
----------+---------+------
 work_mem | 131072  | kB
(1 row)
```

- gs_session_all_settings()

  描述：查询本节点上所有session的全量GUC参数配置。需要SYSADMIN或者MONADMIN权限执行。

  返回值类型：setof record

名称	类型	描述
sessionid	bigint	会话的ID。
pid	bigint	后端线程的ID。
name	text	参数名称。
setting	text	参数当前值。
unit	text	参数的隐式单位。

  示例：

```
gaussdb=# select * from gs_session_all_settings() where name = 'work_mem';
    sessionid    |       pid       |   name   | setting | unit
-----------------+-----------------+----------+---------+------
 140550214145792 |           96974 | work_mem | 65536   | kB
 140550214145792 |           96971 | work_mem | 65536   | kB
 140549731735296 | 140549731735296 | work_mem | 65536   | kB
 140549764413184 | 140549764413184 | work_mem | 65536   | kB
(4 rows)
```

- gs_local_wal_preparse_statistics()

  描述：查询本节点上日志预解析线程最近一次启动，预解析日志的情况。需要SYSADMIN权限执行。

  返回值类型：setof record

名称	类型	描述
preparser_term	text	最近一次预解析日志得到的最大term值。
preparser_start_time	timestamptz	最近一次预解析启动时间。
preparser_end_time	timestamptz	最近一次预解析结束时间。
preparser_start_location	text	最近一次预解析日志起始位置。

名称	类型	描述
preparser_end_location	text	最近一次预解析日志结束位置。
preparser_total_bytes	int8	最近一次预解析日志量，单位：byte。
preparser_speed	int8	最近一次预解析速度，单位：byte/ms。
is_valid	bool	最近一次预解析结果是否可以用于选主。

示例：

```
gaussdb=# select * from gs_local_wal_preparse_statistics();
 preparser_term |     preparser_start_time      |      preparser_end_time       | preparser_start_location |
preparser_end_location | preparser_total_bytes | preparser_speed | is_valid
----------------+-------------------------------+-------------------------------+--------------------------
+------------------------+----------------------+-----------------+----------
 3107           | 2023-02-01 17:04:23.367946+08 | 2023-02-01 17:04:25.354434+08 | 00000003/
C3EEA660       | 00000004/0BE60738      |            1207394520 |      1207394520 | f
(1 row)
```

- gs_hot_standby_space_info()

  描述：查询standby_read/base_page，standby_read/block_info_meta，standby_read/lsn_info_meta文件夹中的文件总数和总大小。

  返回值类型：setof record

名称	类型	描述
base_page_file_num	xid	bage_page_file的总数量。
base_page_total_size	xid	bage_page_file的总大小。
lsn_info_meta_file_num	xid	lsn_info_meta_file的总数量。
lsn_info_meta_total_size	xid	lsn_info_meta_file的总大小。
block_info_meta_file_num	xid	block_info_meta_file的总数量。
block_info_meta_total_size	xid	block_info_meta_file的总大小。

示例：

```
gaussdb=# select * from gs_hot_standby_space_info();
 base_page_file_num | base_page_total_size | lsn_info_meta_file_num | lsn_info_meta_total_size |
block_info_meta_file_num | block_info_meta_total_size
--------------------+----------------------+------------------------+--------------------------
+--------------------------+---------------------------
                 6 |               163840 |                      6 |                  3136 |                       16
```



```
|                  147456
(1 row)
```

- exrto_file_read_stat()

  描述：查询备机读新增的base page file、lsn info meta file和block info meta file三种类型的文件磁盘访问次数和访问总时延。连接备DN查询，其他情况查询结果为0。

  返回值类型：setof record

名称	类型	描述
lsn_info_page_disk_read_counter	int8	lsn info meta file的磁盘访问次数。
lsn_info_page_disk_read_dur	int8	lsn info meta file的磁盘访问总时延。
blk_info_meta_disk_read_counter	int8	block info meta file的磁盘访问次数。
blk_info_meta_disk_read_dur	int8	block info meta file的磁盘访问总时延。
base_page_read_disk_counter	int8	base page file的磁盘访问次数。
base_page_read_disk_dur	int8	base page file的磁盘访问总时延。

  示例：

```
gaussdb=# SELECT * FROM exrto_file_read_stat();
 lsn_info_page_disk_read_counter | lsn_info_page_disk_read_dur | blk_info_meta_disk_read_counter |
blk_info_meta_disk_read_dur | base_page_read_disk_counter | base_page_read_disk_dur
---------------------------------+-----------------------------+---------------------------------
+-----------------------------+-----------------------------+-------------------------
                          14987 |                       92313 |                           23879 |                      129811
|                           0 |                           0
(1 row)
```

- gs_exrto_recycle_info()

  描述：查询资源回收位置，其中包括每个线程的回收lsn，全局回收的lsn，查询线程最老的快照的lsn。连接备DN查询，其他情况查询结果为0。

  返回值类型：setof record

名称	类型	描述
page_redo_worker_thread_id	text	redo线程的回收lsn位置，其中thread_id为redo线程的线程id。
global_recycle_lsn	text	全局回收位置的lsn。
exrto_snapshot_oldest_lsn	text	查询线程的最老的快照lsn。

示例：

```
gaussdb=# SELECT * FROM gs_exrto_recycle_info();
            thread_id             | recycle_lsn
----------------------------------+-------------
 page_redo_worker_140148895381248 | 0/7B4552E0
 page_redo_worker_140148872312576 | 0/7B4535B8
 global_recycle_lsn               | 0/7B4535B8
 exrto_snapshot_oldest_lsn        | 0/8488E6D0
(4 rows)
```

- gs_stat_get_db_conflict_all(oid)

  入参说明：dbid(oid)为数据库的oid。

  描述：查询发送不同类型回放冲突信号的数量。

  返回值类型：setof record

名称	类型	描述
conflict_all	int8	发送回放冲突信号的总数量。
conflict_tablespace	int8	发送tablespace类型回放冲突信号的数量。
conflict_lock	int8	发送lock类型回放冲突信号的数量。
conflict_snapshot	int8	发送snapshot类型回放冲突信号的数量。
conflict_bufferpin	int8	发送bufferpin类型回放冲突信号的数量。
conflict_startup_deadlock	int8	发送startup_deadlock类型回放冲突信号的数量。
conflict_truncate	int8	发送truncate类型回放冲突信号的数量。
conflict_standby_query_timeout	int8	发送standby_query_timeout类型回放冲突信号的数量。
conflict_force_recycle	int8	发送force_recycle类型回放冲突信号的数量。

示例：

```
gaussdb=# SELECT * FROM gs_stat_get_db_conflict_all(12738);
 conflict_all | conflict_tablespace | conflict_lock | conflict_snapshot | conflict_bufferpin |
conflict_startup_deadlock | conflict_truncate | conflict_standby_query_timeout | conflict_force_recycle
--------------+---------------------+---------------+-------------------+--------------------
+---------------------------+-------------------+-------------------------------+------------------------
           0 |                 0 |             0 |                 0 |                 0 |                          0 |                 0
|                         0 |                     0
(1 row)
```

- gs_redo_stat_info()

  描述：查询回放信息。包括回放线程的buffer命中率、执行unlink_rels文件数量、极致RTO场景下回放线程读取buffer时产生I/O操作的waitevent信息以及wal_read_from_write_buffer的waitevent信息。需要连接备DN查询。

  返回值类型：setof record

名称	类型	描述
buffer_hit_rate	float8	回放线程的buffer命中率。
ddl_unlink_nrels_count	int8	回放DDL操作执行unlink rel文件的数量。
read_buffer_io_counter	int8	极致RTO场景下，回放线程读取buffer时产生I/O操作的waitevent触发次数。
read_buffer_io_total_dur	int8	极致RTO场景下，回放线程读取buffer时产生I/O操作的waitevent总用时。
read_buffer_io_avg_dur	int8	极致RTO场景下，回放线程读取buffer时产生I/O操作的waitevent平均用时。
read_buffer_io_min_dur	int8	极致RTO场景下，回放线程读取buffer时产生I/O操作的waitevent最小用时。
read_buffer_io_max_dur	int8	极致RTO场景下，回放线程读取buffer时产生I/O操作的waitevent最大用时。
read_wal_buf_counter	int8	极致RTO场景下，wal_read_from_write_buffer的waitevent触发次数。
read_wal_buf_total_dur	int8	极致RTO场景下，wal_read_from_write_buffer的waitevent总用时。
read_wal_buf_avg_dur	int8	极致RTO场景下，wal_read_from_write_buffer的waitevent平均用时。
read_wal_buf_min_dur	int8	极致RTO场景下，wal_read_from_write_buffer的waitevent最小用时。
read_wal_buf_max_dur	int8	极致RTO场景下，wal_read_from_write_buffer的waitevent最大用时。

示例：

```
gaussdb=# SELECT * FROM gs_redo_stat_info();
-[ RECORD 1 ]------------+----------
buffer_hit_rate          | 70.5707
ddl_unlink_nrels_count   | 3
read_buffer_io_counter   | 1732
read_buffer_io_total_dur | 2850806
read_buffer_io_avg_dur   | 1645
read_buffer_io_min_dur   | 3
read_buffer_io_max_dur   | 981639
read_wal_buf_counter     | 9779
read_wal_buf_total_dur   | 193612470
read_wal_buf_avg_dur     | 19798
read_wal_buf_min_dur     | 3
read_wal_buf_max_dur     | 1914777
```

- gs_recovery_conflict_waitevent_info()

  描述：查询处理回放冲突的函数的waitevent相关信息。需要连接备DN查询。

  返回值类型：setof record

名称	类型	描述
conflict_lock_counter	int8	处理LOCK类型回放冲突的触发次数。
conflict_lock_total_dur	int8	处理LOCK类型回放冲突的总用时。
conflict_lock_avg_dur	int8	处理LOCK类型回放冲突的平均用时。
conflict_lock_min_dur	int8	处理LOCK类型回放冲突的最小用时。
conflict_lock_max_dur	int8	处理LOCK类型回放冲突的最大用时。
conflict_snapshot_counter	int8	处理SNAPSHOT类型回放冲突的触发次数。
conflict_snapshot_total_dur	int8	处理SNAPSHOT类型回放冲突的总用时。
conflict_snapshot_avg_dur	int8	处理SNAPSHOT类型回放冲突的平均用时。
conflict_snapshot_min_dur	int8	处理SNAPSHOT类型回放冲突的最小用时。
conflict_snapshot_max_dur	int8	处理SNAPSHOT类型回放冲突的最大用时。
conflict_tablespace_counter	int8	处理TABLESPACE类型回放冲突的触发次数。
conflict_tablespace_total_dur	int8	处理TABLESPACE类型回放冲突的总用时。
conflict_tablespace_avg_dur	int8	处理TABLESPACE类型回放冲突的平均用时。
conflict_tablespace_min_dur	int8	处理TABLESPACE类型回放冲突的最小用时。
conflict_tablespace_max_dur	int8	处理TABLESPACE类型回放冲突的最大用时。
conflict_database_counter	int8	处理DATABASE类型回放冲突的触发次数。
conflict_database_total_dur	int8	处理DATABASE类型回放冲突的总用时。
conflict_database_avg_dur	int8	处理DATABASE类型回放冲突的平均用时。
conflict_database_min_dur	int8	处理DATABASE类型回放冲突的最小用时。
conflict_database_max_dur	int8	处理DATABASE类型回放冲突的最大用时。
conflict_truncate_counter	int8	处理TRUNCATE类型回放冲突的触发次数。

名称	类型	描述
conflict_truncate_total_dur	int8	处理TRUNCATE类型回放冲突的总用时。
conflict_truncate_avg_dur	int8	处理TRUNCATE类型回放冲突的平均用时。
conflict_truncate_min_dur	int8	处理TRUNCATE类型回放冲突的最小用时。
conflict_truncate_max_dur	int8	处理TRUNCATE类型回放冲突的最大用时。
conflict_standby_query_timeout_counter	int8	处理STANDBY_QUERY_TIMEOUT类型回放冲突的触发次数。
conflict_standby_query_timeout_total_dur	int8	处理STANDBY_QUERY_TIMEOUT类型回放冲突的总用时。
conflict_standby_query_timeout_avg_dur	int8	处理STANDBY_QUERY_TIMEOUT类型回放冲突的平均用时。
conflict_standby_query_timeout_min_dur	int8	处理STANDBY_QUERY_TIMEOUT类型回放冲突的最小用时。
conflict_standby_query_timeout_max_dur	int8	处理STANDBY_QUERY_TIMEOUT类型回放冲突的最大用时。
conflict_force_recycle_counter	int8	处理FORCE_RECYCLE类型回放冲突的触发次数。
conflict_force_recycle_total_dur	int8	处理FORCE_RECYCLE类型回放冲突的总用时。
conflict_force_recycle_avg_dur	int8	处理FORCE_RECYCLE类型回放冲突的平均用时。
conflict_force_recycle_min_dur	int8	处理FORCE_RECYCLE类型回放冲突的最小用时。
conflict_force_recycle_max_dur	int8	处理FORCE_RECYCLE类型回放冲突的最大用时。

示例：

```
gaussdb=# SELECT * FROM gs_recovery_conflict_waitevent_info();
-[ RECORD 1 ]----------------------------+------
conflict_lock_counter                    | 0
conflict_lock_total_dur                  | 0
conflict_lock_avg_dur                    | 0
conflict_lock_min_dur                    | 0
conflict_lock_max_dur                    | 0
conflict_snapshot_counter                | 0
conflict_snapshot_total_dur              | 0
conflict_snapshot_avg_dur                | 0
conflict_snapshot_min_dur                | 0
conflict_snapshot_max_dur                | 0
conflict_tablespace_counter              | 0
```

```
conflict_tablespace_total_dur          | 0
conflict_tablespace_avg_dur            | 0
conflict_tablespace_min_dur            | 0
conflict_tablespace_max_dur            | 0
conflict_database_counter              | 0
conflict_database_total_dur            | 0
conflict_database_avg_dur              | 0
conflict_database_min_dur              | 0
conflict_database_max_dur              | 0
conflict_truncate_counter              | 6
conflict_truncate_total_dur            | 35872
conflict_truncate_avg_dur              | 5978
conflict_truncate_min_dur              | 5130
conflict_truncate_max_dur              | 7459
conflict_standby_query_timeout_counter   | 0
conflict_standby_query_timeout_total_dur | 0
conflict_standby_query_timeout_avg_dur   | 0
conflict_standby_query_timeout_min_dur   | 0
conflict_standby_query_timeoutmax_dur    | 0
conflict_force_recycle_counter         | 0
conflict_force_recycle_total_dur       | 0
conflict_force_recycle_avg_dur         | 0
conflict_force_recycle_min_dur         | 0
conflict_force_recycle_max_dur         | 0
```

- gs_display_delay_ddl_info()
  描述：查看备机中延迟删除的文件信息。
  返回值类型：setof record

名称	类型	描述
type	INT4	删除操作的对象是表或数据库。
lsn	TEXT	标识特定日志文件记录在此日志文件中的位置。
tablespace	INT4	数据库中用于存储表和索引的物理空间。
database	INT4	该数据库的物理存储位置。
relation	INT4	数据库中的对象，可以是表、视图、索引的物理位置。
bucketid	INT4	指定关系对象所属的bucket。
opt	INT4	压缩表相关属性。
forknum	INT4	主体命名之后的后缀命名，通过主体命名和后缀命名，可以找到唯一的物理文件。

示例：

```
gaussdb=# SELECT * FROM gs_display_delay_ddl_info();
 type | lsn | tablespace | database | relation | bucketid | opt | forknum
------+-----+------------+----------+----------+----------+-----+---------
(0 rows)
```

分区表统计信息函数

- gs_stat_get_partition_stats(oid)
  描述：获取特定分区的统计信息。
  返回值类型：record

- gs_stat_get_xact_partition_stats(oid)
  描述：获取特定分区的事务中统计信息。
  返回值类型：record
- gs_stat_get_all_partitions_stats()
  描述：获取所有分区的统计信息。
  返回值类型：setof record
- gs_stat_get_xact_all_partitions_stats()
  描述：获取所有分区的事务中统计信息。
  返回值类型：setof record
- gs_statio_get_all_partitions_stats()
  描述：获取所有分区的I/O统计信息。
  返回值类型：setof record
  上述五个函数示例：

⚠ 注意

运行时统计信息上报是异步的，且基于UDP协议，后台线程处理可能存在延迟和丢包，此处示例预期仅供参考。

事务外统计信息查询

```
gaussdb=# CREATE TABLE part_tab1
gaussdb-# (
gaussdb(#     a int, b int
gaussdb(# )
gaussdb-# PARTITION BY RANGE(b)
gaussdb-# (
gaussdb(#     PARTITION P1 VALUES LESS THAN(10),
gaussdb(#     PARTITION P2 VALUES LESS THAN(20),
gaussdb(#     PARTITION P3 VALUES LESS THAN(MAXVALUE)
gaussdb(# );
CREATE TABLE
gaussdb=# CREATE TABLE subpart_tab1
gaussdb-# (
gaussdb(#     month_code VARCHAR2 ( 30 ) NOT NULL ,
gaussdb(#     dept_code  VARCHAR2 ( 30 ) NOT NULL ,
gaussdb(#     user_no    VARCHAR2 ( 30 ) NOT NULL ,
gaussdb(#     sales_amt  int
gaussdb(# )
gaussdb-# PARTITION BY RANGE (month_code) SUBPARTITION BY RANGE (dept_code)
gaussdb-# (
gaussdb(#   PARTITION p_201901 VALUES LESS THAN( '201903' )
gaussdb(#   (
gaussdb(#     SUBPARTITION p_201901_a VALUES LESS THAN( '2' ),
gaussdb(#     SUBPARTITION p_201901_b VALUES LESS THAN( '3' )
gaussdb(#   ),
gaussdb(#   PARTITION p_201902 VALUES LESS THAN( '201904' )
gaussdb(#   (
gaussdb(#     SUBPARTITION p_201902_a VALUES LESS THAN( '2' ),
gaussdb(#     SUBPARTITION p_201902_b VALUES LESS THAN( '3' )
gaussdb(#   )
gaussdb(# );
CREATE TABLE
gaussdb=# CREATE INDEX index_part_tab1 ON part_tab1(b) LOCAL
gaussdb-# (
gaussdb(# PARTITION b_index1,
gaussdb(# PARTITION b_index2,
```



```
gaussdb(# PARTITION b_index3
gaussdb(# );
CREATE INDEX
gaussdb=# CREATE INDEX idx_user_no ON subpart_tab1(user_no) LOCAL;
CREATE INDEX
gaussdb=# INSERT INTO part_tab1 VALUES(1, 1);
INSERT 0 1
gaussdb=# INSERT INTO part_tab1 VALUES(1, 11);
INSERT 0 1
gaussdb=# INSERT INTO part_tab1 VALUES(1, 21);
INSERT 0 1
gaussdb=# UPDATE part_tab1 SET a = 2 WHERE b = 1;
UPDATE 1
gaussdb=# UPDATE part_tab1 SET a = 3 WHERE b = 11;
UPDATE 1
gaussdb=# UPDATE /*+ indexscan(part_tab1) */ part_tab1 SET a = 4 WHERE b = 21;
UPDATE 1
gaussdb=# DELETE FROM part_tab1;
DELETE 3
gaussdb=# ANALYZE part_tab1;
ANALYZE
gaussdb=# VACUUM part_tab1;
VACUUM
gaussdb=# INSERT INTO subpart_tab1 VALUES('201902', '1', '1', 1);
INSERT 0 1
gaussdb=# INSERT INTO subpart_tab1 VALUES('201902', '2', '2', 1);
INSERT 0 1
gaussdb=# INSERT INTO subpart_tab1 VALUES('201903', '1', '3', 1);
INSERT 0 1
gaussdb=# INSERT INTO subpart_tab1 VALUES('201903', '2', '4', 1);
INSERT 0 1
gaussdb=# UPDATE subpart_tab1 SET sales_amt = 2 WHERE user_no='1';
UPDATE 1
gaussdb=# UPDATE subpart_tab1 SET sales_amt = 3 WHERE user_no='2';
UPDATE 1
gaussdb=# UPDATE subpart_tab1 SET sales_amt = 4 WHERE user_no='3';
UPDATE 1
gaussdb=# UPDATE /*+ indexscan(subpart_tab1) */ subpart_tab1 SET sales_amt = 5 WHERE
user_no='4';
UPDATE 1
gaussdb=# DELETE FROM subpart_tab1;
DELETE 4
gaussdb=# ANALYZE subpart_tab1;
ANALYZE
gaussdb=# VACUUM subpart_tab1;
VACUUM
gaussdb=# SELECT * FROM gs_stat_all_partitions;
 partition_oid | schemaname |    relname    | partition_name | sub_partition_name | seq_scan |
seq_tup_read | idx_scan | idx_tup_fetch | n_tup_ins | n_tup_upd | n_tup_del | n_tup_hot_upd | n_live_tup
|
 n_dead_tup |          last_vacuum          |     last_autovacuum     |         last_analyze          |
last_autoanalyze     | vacuum_count | autovacuum_count | analyze_count | autoanalyze_count
---------------+------------+--------------+----------------+--------------------+-----------+--------------
+----------+---------------+-----------+-----------+-----------+---------------+------------+
------------+-------------------------------+-----------------------+-------------------------------
+------------------------+--------------+------------------+---------------+--------------------
        16964 | public     | subpart_tab1 | p_201902       | p_201902_b         |       5 |           1 |       4
|            1 |         1 |         1 |         1 |             1 |           0 |
         1 | 2023-05-15 20:36:45.293965+08 | 2000-01-01 08:00:00+08 | 2023-05-15
20:36:44.688861+08 | 2000-01-01 08:00:00+08 |            1 |               0 |             1 |              0
        16963 | public     | subpart_tab1 | p_201902       | p_201902_a         |       5 |           1 |       4
|            0 |         1 |         1 |         1 |             1 |           0 |
         1 | 2023-05-15 20:36:45.291022+08 | 2000-01-01 08:00:00+08 | 2023-05-15
20:36:44.688843+08 | 2000-01-01 08:00:00+08 |            1 |               0 |             1 |              0
        16961 | public     | subpart_tab1 | p_201901       | p_201901_b         |       5 |           1 |       4
|            0 |         1 |         1 |         1 |             1 |           0 |
         1 | 2023-05-15 20:36:45.288037+08 | 2000-01-01 08:00:00+08 | 2023-05-15
20:36:44.688829+08 | 2000-01-01 08:00:00+08 |            1 |               0 |             1 |              0
        16960 | public     | subpart_tab1 | p_201901       | p_201901_a         |       5 |           1 |       4
```



```
|            0 |         1 |         1 |         1 |            1 |         0 |
         1 | 2023-05-15 20:36:45.285311+08 | 2000-01-01 08:00:00+08 | 2023-05-15
20:36:44.688802+08 | 2000-01-01 08:00:00+08 |            1 |             0 |            1 |             0
         16954 | public     | part_tab1    | p3           |                |       2 |            1 |        1 |            1
|         1 |         1 |         1 |            1 |         0 |
         1 | 2023-05-15 20:36:29.490636+08 | 2000-01-01 08:00:00+08 | 2023-05-15
20:36:28.540115+08 | 2000-01-01 08:00:00+08 |            1 |             0 |            1 |             0
         16953 | public     | part_tab1    | p2           |                |       4 |            1 |        1 |            0
|         1 |         1 |         1 |            1 |         0 |
         1 | 2023-05-15 20:36:29.487914+08 | 2000-01-01 08:00:00+08 | 2023-05-15
20:36:28.540098+08 | 2000-01-01 08:00:00+08 |            1 |             0 |            1 |             0
         16952 | public     | part_tab1    | p1           |                |       5 |            1 |        1 |            0
|         1 |         1 |         1 |            1 |         0 |
         1 | 2023-05-15 20:36:29.48536+08  | 2000-01-01 08:00:00+08 | 2023-05-15 20:36:28.540071+08
| 2000-01-01 08:00:00+08 |            1 |               0 |             1 |               0
(7 rows)

gaussdb=# SELECT * FROM gs_statio_all_partitions;
 partition_oid | schemaname |    relname     | partition_name | sub_partition_name | heap_blks_read |
heap_blks_hit | idx_blks_read | idx_blks_hit | toast_blks_read | toast_blks_hit | tidx_blks_read | t
idx_blks_hit
---------------+------------+--------------+----------------+--------------------+----------------+--------------
+---------------+--------------+-----------------+----------------+----------------+--
-------------
         16964 | public     | subpart_tab1 | p_201902       | p_201902_b         |              4 |            8
|             2 |           21 |                 |                |                |

         16963 | public     | subpart_tab1 | p_201902       | p_201902_a         |              4 |            8
|             2 |           21 |                 |                |                |

         16961 | public     | subpart_tab1 | p_201901       | p_201901_b         |              4 |            8
|             2 |           21 |                 |                |                |

         16960 | public     | subpart_tab1 | p_201901       | p_201901_a         |              4 |            8
|             2 |           21 |                 |                |                |

         16954 | public     | part_tab1    | p3           |                |            4 |            8 |            2
|           15 |                 |                |                |

         16953 | public     | part_tab1    | p2           |                |            4 |            8 |            2
|           15 |                 |                |                |

         16952 | public     | part_tab1    | p1           |                |            4 |            8 |            2
|           15 |                 |                |                |

(7 rows)

gaussdb=# SELECT * FROM gs_stat_get_partition_stats(16952);
 partition_oid | seq_scan | seq_tup_read | idx_scan | idx_tup_fetch | n_tup_ins | n_tup_upd | n_tup_del |
n_tup_hot_upd | n_live_tup | n_dead_tup |          last_vacuum          |     last_autovacuum
  |          last_analyze          |     last_autoanalyze     | vacuum_count | autovacuum_count |
analyze_count | autoanalyze_count |    last_data_changed     | heap_blks_read | heap_blks_hit |
idx_blks_re
ad | idx_blks_hit | tup_fetch | block_fetch
---------------+----------+--------------+----------+---------------+-----------+-----------+-----------
+---------------+------------+------------+------------------------------+---------------------
---+------------------------------+-----------------------+--------------+------------------+---------------
+-------------------+-----------------------+----------------+---------------+------------
---+--------------+-----------+-------------
         16952 |        5 |            1 |        1 |            0 |         1 |         1 |         1 |            1 |         0
|          1 | 2023-05-15 20:36:29.48536+08 | 2000-01-01 08:00:00+0
8 | 2023-05-15 20:36:28.540071+08 | 2000-01-01 08:00:00+08 |            1 |               0 |            1
|               0 | 2000-01-01 08:00:00+08 |             4 |             8 |
 2 |           21 |         0 |           12
(1 row)
```

事务内统计信息查询：

```
gaussdb=# BEGIN;
BEGIN
gaussdb=# INSERT INTO part_tab1 VALUES(1, 1);
```



```
INSERT 0 1
gaussdb=# INSERT INTO part_tab1 VALUES(1, 11);
INSERT 0 1
gaussdb=# INSERT INTO part_tab1 VALUES(1, 21);
INSERT 0 1
gaussdb=# UPDATE part_tab1 SET a = 2 WHERE b = 1;
UPDATE 1
gaussdb=# UPDATE part_tab1 SET a = 3 WHERE b = 11;
UPDATE 1
gaussdb=# UPDATE /*+ indexscan(part_tab1) */ part_tab1 SET a = 4 WHERE b = 21;
UPDATE 1
gaussdb=# DELETE FROM part_tab1;
DELETE 3
gaussdb=# INSERT INTO subpart_tab1 VALUES('201902', '1', '1', 1);
INSERT 0 1
gaussdb=# INSERT INTO subpart_tab1 VALUES('201902', '2', '2', 1);
INSERT 0 1
gaussdb=# INSERT INTO subpart_tab1 VALUES('201903', '1', '3', 1);
INSERT 0 1
gaussdb=# INSERT INTO subpart_tab1 VALUES('201903', '2', '4', 1);
INSERT 0 1
gaussdb=# UPDATE subpart_tab1 SET sales_amt = 2 WHERE user_no='1';
UPDATE 1
gaussdb=# UPDATE subpart_tab1 SET sales_amt = 3 WHERE user_no='2';
UPDATE 1
gaussdb=# UPDATE subpart_tab1 SET sales_amt = 4 WHERE user_no='3';
UPDATE 1
gaussdb=# UPDATE /*+ indexscan(subpart_tab1) */ subpart_tab1 SET sales_amt = 5 WHERE
user_no='4';
UPDATE 1
gaussdb=# DELETE FROM subpart_tab1;
DELETE 4
gaussdb=# SELECT * FROM gs_stat_xact_all_partitions;
 partition_oid | schemaname |    relname     | partition_name | sub_partition_name | seq_scan |
seq_tup_read | idx_scan | idx_tup_fetch | n_tup_ins | n_tup_upd | n_tup_del | n_tup_hot_upd
---------------+------------+--------------+----------------+--------------------+----------+--------------
+----------+--------------+----------+----------+----------+--------------
       16964 | public     | subpart_tab1 | p_201902       | p_201902_b         |        4 |          4 |        1
|           2 |        1 |        1 |        1 |            1
       16963 | public     | subpart_tab1 | p_201902       | p_201902_a         |        4 |          4 |        1
|           0 |        1 |        1 |        1 |            1
       16961 | public     | subpart_tab1 | p_201901       | p_201901_b         |        4 |          4 |        1
|           0 |        1 |        1 |        1 |            1
       16960 | public     | subpart_tab1 | p_201901       | p_201901_a         |        4 |          4 |        1
|           0 |        1 |        1 |        1 |            1
       16954 | public     | part_tab1    | p3             |                    |        1 |          1 |        1 |          2
|        1 |        1 |        1 |            1
       16953 | public     | part_tab1    | p2             |                    |        3 |          2 |        0 |          0
|        1 |        1 |        1 |            1
       16952 | public     | part_tab1    | p1             |                    |        4 |          2 |        0 |          0
|        1 |        1 |        1 |            1
(7 rows)

gaussdb=# SELECT * FROM gs_stat_get_xact_partition_stats(16952);
 partition_oid | seq_scan | seq_tup_read | idx_scan | idx_tup_fetch | n_tup_ins | n_tup_upd | n_tup_del |
n_tup_hot_upd | tup_fetch
---------------+----------+--------------+----------+---------------+-----------+-----------+-----------
+---------------+-----------
       16952 |        4 |            2 |        0 |             0 |         1 |         1 |         1 |             1 |         0
(1 row)

-- 删除表
gaussdb=# DROP TABLE part_tab1;
DROP TABLE
gaussdb=# DROP TABLE subpart_tab1;
DROP TABLE
```

- gs_stat_get_partition_analyze_count(oid)
  描述：用户在该分区上启动分析的次数

返回值类型：bigint

- gs_stat_get_partition_autoanalyze_count(oid)
  描述：autovacuum守护进程在该分区上启动分析的次数。
  返回值类型：bigint
- gs_stat_get_partition_autovacuum_count(oid)
  描述：autovacuum守护进程在该分区上启动清理的次数。
  返回值类型：bigint
- gs_stat_get_partition_last_analyze_time(oid)
  描述：用户在该分区上最后一次手动启动分析或者autovacuum线程启动分析的时间。
  返回值类型：timestamptz
- gs_stat_get_partition_last_autoanalyze_time(oid)
  描述：autovacuum守护进程在该分区上最后一次启动分析的时间。
  返回值类型：timestamptz
- gs_stat_get_partition_last_autovacuum_time(oid)
  描述：autovacuum守护进程在该分区上最后一次启动清理的时间。
  返回值类型：timestamptz
- gs_stat_get_partition_last_data_changed_time(oid)
  描述：对于在分区上的修改insert/update/delete/truncate，在该表上最后一次操作的时间。当前暂不支持。
  返回值类型：timestamptz
- gs_stat_get_partition_last_vacuum_time(oid)
  描述：用户在该分区上最后一次手动启动清理或者autovacuum线程启动清理的时间。
  返回值类型：timestamptz
- gs_stat_get_partition_numscans(oid)
  描述：分区顺序扫描读取的行数目。
  返回值类型：bigint
- gs_stat_get_partition_tuples_returned(oid)
  描述：分区顺序扫描读取的行数目。
  返回值类型：bigint
- gs_stat_get_partition_tuples_fetched(oid)
  描述：分区位图扫描抓取的行数目。
  返回值类型：bigint
- gs_stat_get_partition_vacuum_count(oid)
  描述：用户在该分区上启动清理的次数。
  返回值类型：bigint
- gs_stat_get_xact_partition_tuples_fetched(oid)
  描述：事务中扫描的tuple行数。
  返回值类型：bigint

- gs_stat_get_xact_partition_numscans(oid)
  描述：当前事务中分区执行的顺序扫描次数。
  返回值类型：bigint
- gs_stat_get_xact_partition_tuples_returned(oid)
  描述：当前事务中分区通过顺序扫描读取的行数。
  返回值类型：bigint
- gs_stat_get_partition_blocks_fetched(oid)
  描述：分区的磁盘块抓取请求的数量。
  返回值类型：bigint
- gs_stat_get_partition_blocks_hit(oid)
  描述：在缓冲区中找到的分区的磁盘块请求数目。
  返回值类型：bigint
- pg_stat_get_partition_tuples_inserted(oid)
  描述：插入相应表分区中行的数量。
  返回值类型：bigint
- pg_stat_get_partition_tuples_updated(oid)
  描述：在相应表分区中已更新行的数量。
  返回值类型：bigint
- pg_stat_get_partition_tuples_deleted(oid)
  描述：从相应表分区中删除行的数量。
  返回值类型：bigint
- pg_stat_get_partition_tuples_changed(oid)
  描述：该表分区上一次analyze或autoanalyze之后插入、更新、删除行的总数量。
  返回值类型：bigint
- pg_stat_get_partition_live_tuples(oid)
  描述：分区表活行数。
  返回值类型：bigint
- pg_stat_get_partition_dead_tuples(oid)
  描述：分区表死行数。
  返回值类型：bigint
- pg_stat_get_xact_partition_tuples_inserted(oid)
  描述：表分区相关的活跃子事务中插入的tuple数。
  返回值类型：bigint
- pg_stat_get_xact_partition_tuples_deleted(oid)
  描述：表分区相关的活跃子事务中删除的tuple数。
  返回值类型：bigint
- pg_stat_get_xact_partition_tuples_hot_updated(oid)
  描述：表分区相关的活跃子事务中热更新的tuple数。
  返回值类型：bigint

- pg_stat_get_xact_partition_tuples_updated(oid)
  描述：表分区相关的活跃子事务中更新的tuple数。
  返回值类型：bigint
- pg_stat_get_partition_tuples_hot_updated(oid)
  描述：返回给定分区id的分区热更新元组数的统计。
  参数：oid
  返回值类型：bigint

7.5.27 触发器函数

- pg_get_triggerdef(oid)
  描述：获取触发器的定义信息。
  参数：待查触发器的OID。
  返回值类型：text
  示例:

```
-- 创建表tri_insert。
gaussdb=# CREATE TABLE tri_insert (a int, b int);
CREATE TABLE
-- 创建函数trigger_func。
gaussdb=# CREATE FUNCTION trigger_func() RETURNS trigger LANGUAGE plpgsql AS '
gaussdb'# BEGIN
gaussdb'# RAISE NOTICE ''trigger_func(%) called: action = %, when = %, level = %'', TG_ARGV[0],
TG_OP, TG_WHEN, TG_LEVEL;
gaussdb'# RETURN NULL;
gaussdb'# END;';
CREATE FUNCTION
-- 创建触发器before_ins_stmt_trig。
gaussdb=# CREATE TRIGGER before_ins_stmt_trig BEFORE INSERT ON tri_insert
gaussdb-# FOR EACH STATEMENT EXECUTE PROCEDURE trigger_func('before_ins_stmt');
CREATE TRIGGER
-- 创建触发器after_ins_when_trig。
gaussdb=# CREATE TRIGGER after_ins_when_trig AFTER INSERT ON tri_insert
gaussdb-# FOR EACH ROW WHEN (new.a IS NOT NULL) EXECUTE PROCEDURE
trigger_func('after_ins_when');
CREATE TRIGGER
-- 查看表tri_insert的触发器定义信息。
gaussdb=# SELECT pg_get_triggerdef(oid) FROM pg_trigger WHERE tgrelid = 'tri_insert'::regclass;

pg_get_triggerdef
-----------------------------------------------------------------------------------------------------------------------------------
 CREATE TRIGGER before_ins_stmt_trig BEFORE INSERT ON tri_insert FOR EACH STATEMENT
EXECUTE PROCEDURE trigger_func('before_ins_stmt')
 CREATE TRIGGER after_ins_when_trig AFTER INSERT ON tri_insert FOR EACH ROW WHEN ((new.a IS
NOT NULL)) EXECUTE PROCEDURE trigger_func('after_ins_when')
(2 rows)
```

- pg_get_triggerdef(oid, boolean)
  描述：获取触发器的定义信息。
  参数：待查触发器的OID及是否以pretty方式展示。

  说明

  仅在创建trigger时指定WHEN条件的情况下，布尔类型参数才生效。

  返回值类型：text
  示例:

```
-- 查看表tri_insert的触发器定义信息，以非pretty形式。
gaussdb=# SELECT pg_get_triggerdef(oid, false) FROM pg_trigger WHERE tgrelid = 'tri_insert'::regclass;
```



```
pg_get_triggerdef
-----------------------------------------------------------------------------------------------------------------------------------------------------------
 CREATE TRIGGER before_ins_stmt_trig BEFORE INSERT ON tri_insert FOR EACH STATEMENT
EXECUTE PROCEDURE trigger_func('before_ins_stmt')
 CREATE TRIGGER after_ins_when_trig AFTER INSERT ON tri_insert FOR EACH ROW WHEN ((new.a IS
NOT NULL)) EXECUTE PROCEDURE trigger_func('after_ins_when')
(2 rows)

-- 查看表tri_insert的触发器定义信息，以pretty形式。
gaussdb=# SELECT pg_get_triggerdef(oid, true) FROM pg_trigger WHERE tgrelid = 'tri_insert'::regclass;

pg_get_triggerdef
-----------------------------------------------------------------------------------------------------------------------------------------------------------
 CREATE TRIGGER before_ins_stmt_trig BEFORE INSERT ON tri_insert FOR EACH STATEMENT
EXECUTE PROCEDURE trigger_func('before_ins_stmt')
 CREATE TRIGGER after_ins_when_trig AFTER INSERT ON tri_insert FOR EACH ROW WHEN (new.a IS
NOT NULL) EXECUTE PROCEDURE trigger_func('after_ins_when')
(2 rows)
-- 清理表tri_insert。
gaussdb=# DROP TABLE tri_insert CASCADE;
DROP TABLE
-- 清理函数trigger_func。
gaussdb=# DROP FUNCTION trigger_func;
DROP FUNCTION
```

7.5.28 HashFunc 函数

- ora_hash(expression,[seed])

  描述：用于计算给定表达式的哈希值。expression:可输入的类型覆盖字符串，时间类型，数字类型，根据expression进行计算哈希值。seed:可选参数，一个int8值，可以对同一个输入值返回不同的结果, 用于计算带随机数的hash值。

  返回类型：int8类型的哈希值。

  示例：

```
gaussdb=# select ora_hash(123);
 ora_hash
------------
 4089882933
(1 row)
gaussdb=# select ora_hash('123');
 ora_hash
------------
 2034089965
(1 row)
gaussdb=# select ora_hash('sample');
 ora_hash
------------
 1573005290
(1 row)
gaussdb=# select ora_hash(to_date('2012-1-2','yyyy-mm-dd'));
 ora_hash
------------
 1171473495
(1 row)
gaussdb=# select ora_hash(123,234);
 ora_hash
------------
 -9089505052966355682
(1 row)
gaussdb=# select ora_hash('123',234);
 ora_hash
------------
 5742589019960764616
(1 row)
```



```
gaussdb=# select ora_hash('sample',234);
 ora_hash
------------
 -1747984408055821656
(1 row)
gaussdb=# select ora_hash(to_date('2012-1-2','yyyy-mm-dd'),234);
 ora_hash
------------
 -3306025179710572679
(1 row)
```

说明

此函数在参数a_format_version值为10c和a_format_dev_version值为s2的情况下才能生效。

- hash_array(anyarray)

  描述：数组哈希，将数组的元素通过哈希函数得到结果，并返回合并结果。

  参数：数据类型为anyarray。

  返回值类型：integer

  示例：

  ```
  gaussdb=# select hash_array(ARRAY[[1,2,3],[1,2,3]]);
   hash_array
  ------------
   -382888479
  (1 row)
  ```

- hash_numeric(numeric)

  描述：计算Numeric类型的数据的hash值。

  参数：Numeric类型的数据。

  返回值类型：integer

  示例：

  ```
  gaussdb=# select hash_numeric(30);
   hash_numeric
  --------------
    -282860963
  (1 row)
  ```

- hash_range(anyrange)

  描述：计算range的哈希值。

  参数：anyrange类型的数据。

  返回值类型：integer

  示例：

  ```
  gaussdb=# select hash_range(numrange(1.1,2.2));
   hash_range
  ------------
   683508754
  (1 row)
  ```

- hashbpchar(character)

  描述：计算bpchar的哈希值。

  参数：character类型的数据。

  返回值类型：integer

  示例：

  ```
  gaussdb=# select hashbpchar('hello');
   hashbpchar
  -------------
  ```



```
 -1870292951
(1 row)
```

- hashchar(char)

  描述：char和布尔数据转换为哈希值。

  参数：char类型的数据或者bool类型的数据。

  返回值类型：integer

  示例：

```
gaussdb=# select hashbpchar('hello');
 hashbpchar
-------------
 -1870292951
(1 row)

gaussdb=# select hashchar('true');
  hashchar
------------
 1686226652
(1 row)
```

- hashenum(anyenum)

  描述：枚举类型转哈希值。

  参数：anyenum类型的数据。

  返回值类型：integer

  示例：

```
gaussdb=# CREATE TYPE b1 AS ENUM('good', 'bad', 'ugly');
CREATE TYPE
gaussdb=# call hashenum('good'::b1);
  hashenum
------------
 1821213359
(1 row)
```

- hashfloat4(real)

  描述：float4转哈希值。

  参数：real类型的数据。

  返回值类型：integer

  示例：

```
gaussdb=# select hashfloat4(12.1234);
 hashfloat4
------------
 1398514061
(1 row)
```

- hashfloat8(double precision)

  描述：float8转哈希值。

  参数：double precision类型的数据。

  返回值类型：integer

  示例：

```
gaussdb=# select hashfloat8(123456.1234);
 hashfloat8
------------
 1673665593
(1 row)
```

- hashinet(inet)

  描述：inet / cidr转哈希值。

参数：inet类型的数据。

返回值类型：integer

示例：

```
gaussdb=# select hashinet('127.0.0.1'::inet);
  hashinet
-------------
 -1435793109
(1 row)
```

- hashint1(tinyint)

  描述：INT1转哈希值。

  参数：tinyint类型的数据。

  返回值类型：uint32

  示例：

  ```
  gaussdb=# select hashint1(20);
    hashint1
  -------------
   -2014641093
  (1 row)
  ```

- hashint2(smallint)

  描述：INT2转哈希值。

  参数：smallint类型的数据。

  返回值类型：uint32

  示例：

  ```
  gaussdb=# select hashint2(20000);
    hashint2
  ------------
   -863179081
  (1 row)
  ```

7.5.29 提示信息函数

- report_application_error()

  描述：PL执行过程中，可以使用此函数来抛ERROR。

  返回值类型：void

  表 7-105 report_application_error 参数说明

参数	类型	说明	是否必选
log	text	error消息的内容。	是
code	int4	error消息对应的error code，范围为：-20999 ~ -20000。	否

7.5.30 全局临时表函数

- pg_get_gtt_relstats(relOid)

  描述：显示当前会话指定的全局临时表的基本信息。

  参数：全局临时表的OID。

  返回值类型：record

示例:

```
gaussdb=# select * from pg_get_gtt_relstats(74069);
 relfilenode | relpages | reltuples | relallvisible | relfrozenxid | relminmxid
-------------+----------+-----------+---------------+--------------+------------
       74069 |       58 |     13000 |             0 |        11151 |          0
(1 row)
```

- pg_get_gtt_statistics(relOid, attnum, ''::text)

  描述：显示当前会话指定的全局临时表的单列统计信息。

  参数：全局临时表的OID和属性attnum。

  返回值类型：record

  示例:

```
gaussdb=# select * from pg_get_gtt_statistics(74069,1,''::text);
 starelid | starelkind | staattnum | stainherit | stanullfrac | stawidth | stadistinct | stakind1 | stakind2 |
stakind3 | stakind4 | stakind5 | staop1 | staop2 | staop3 | staop4 | staop5 | stanumbers1 | stanumbers2
| stanumbers3 | stanu
mbers4 | stanumbers5
|


stavalues1

                                                                | stavalues2 | stavalues3 | stavalues4 | stavalues5 |
stadndistinct | staextinfo
----------+------------+-----------+------------+-------------+----------+-------------+----------+----------
+----------+----------+----------+--------+--------+--------+--------+--------+-------------+-------------
+-------------+------
-------+-------------
+-----------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+------------+------------+------------
+------------+--------------+------------
   74069 | c         |         1 | f         |          0 |       4 |         -1 |       2 |       3 |       0 |       0 |
0 |     97 |     97 |      0 |      0 |      0 |           | {1}        |            |
      |            |
{1,130,260,390,520,650,780,910,1040,1170,1300,1430,1560,1690,1820,1950,2080,2210,2340,2470,2600,
2730,2860,2990,3120,3250,3380,3510,3640,3770,3900,4030,4160,4290,4420,4550,4680,4810,4940,5070
,5200,5330,5460,5590,57
20,5850,5980,6110,6240,6370,6500,6630,6760,6890,7020,7150,7280,7410,7540,7670,7800,7930,8060,8
190,8320,8450,8580,8710,8840,8970,9100,9230,9360,9490,9620,9750,9880,10010,10140,10270,10400,1
0530,10660,10790,10920,11050,11180,11310,11440,1
1570,11700,11830,11960,12090,12220,12350,12480,12610,12740,12870,13000} |            |
|           |           |            0 |
(1 row)
```

- pg_gtt_attached_pid(relOid)

  描述：显示正在使用指定全局临时表的所有线程pid和会话ID。当线程池开启时，如果正在使用全局临时表的会话为detach状态，pid会显示为0，sessionid为会话的ID。

  参数：全局临时表的OID。

  返回值类型：record

  示例:

```
gaussdb=# select * from pg_gtt_attached_pid(74069);
 relid |       pid       |   sessionid
-------+-----------------+-----------------
 74069 | 139648170456832 | 139648170456832
 74069 | 139648123270912 | 139648123270912
(2 rows)
```

- dbe_perf.get_global_full_sql_by_timestamp(start_timestamp timestamp, end_timestamp timestamp)

描述：获取实例级的全量SQL(Full SQL)信息。
返回值类型：record

表 7-106 dbe_perf.get_global_full_sql_by_timestamp 参数说明

参数	类型	描述
start_timestamp	timestamp	SQL启动时间范围的开始时间点。
end_timestamp	timestamp	SQL启动时间范围的结束时间点。

- dbe_perf.get_global_slow_sql_by_timestamp(start_timestamp timestamp, end_timestamp timestamp)
  描述：获取实例级的慢SQL(Slow SQL)信息。
  返回值类型：record

  表 7-107 dbe_perf.get_global_slow_sql_by_timestamp 参数说明

参数	类型	描述
start_timestamp	timestamp	SQL启动时间范围的开始时间点。
end_timestamp	timestamp	SQL启动时间范围的结束时间点。

- statement_detail_decode(detail text, format text, pretty bool)
  解析全量/慢SQL语句中的details字段的信息。

  表 7-108 statement_detail_decode 参数说明

参数	类型	描述
detail	text	SQL语句产生的事件的集合(不可读)。
format	text	解析输出格式，取值为plaintext或json。
pretty	bool	当format为plaintext时，是否以优雅的格式展示： ● true表示通过“\n”分隔事件。 ● false表示通过“,”分隔事件。

- pg_list_gtt_relfrozenxids()
  描述：显示各会话的冻结事务xid。
  pid=0的行，显示所有会话中最老的冻结事务xid。
  参数：无。
  返回值类型：record

示例:

```
gaussdb=# select * from pg_list_gtt_relfrozenxids();
       pid       | relfrozenxid
-----------------+--------------
 139648123270912 |        11151
 139648170456832 |        11155
               0 |        11151
(3 rows)
```

7.5.31 故障注入系统函数

- gs_fault_inject(int64, text, text, text, text, text)

  描述：该函数不能调用，调用时会报WARNING信息："unsupported fault injection"，并不会对数据库产生任何影响和改变。

  参数：int64注入故障类型（0：CLOG扩展页面，1：读取CLOG页面，2：强制死锁）。

  - text第二个入参在第一入参为2的模式下若为“1”则死锁，其余不死锁；第二个入参在第一入参为0，1时，表示CLOG开始扩展或读取的起始页面号。
  - text第三个入参在第一入参为0，1时，表示扩展或读取的页面个数。
  - text第四到六入参为预留参数。

  返回值类型：int64

7.5.32 AI 特性函数

- gs_index_advise(text)

  描述：针对单条查询语句推荐索引。

  参数：SQL语句字符串

  返回值类型：record

  示例请参见《特性指南》的“DBMind: 数据库自治运维 > DBMind的AI子功能 > Index-advisor：索引推荐 > 单query索引推荐”章节。

- hypopg_create_index(text, [text])

  描述：创建虚拟索引。

  参数：创建索引语句的字符串，创建的虚拟索引的级别（可选）

  返回值类型：record

  示例请参见《特性指南》的“DBMind: 数据库自治运维 > DBMind的AI子功能 > Index-advisor：索引推荐 > 虚拟索引”章节。

- hypopg_display_index([text])

  描述：显示所有创建的虚拟索引信息。

  参数：显示的虚拟索引级别（可选）

  返回值类型：record

  示例请参见《特性指南》的“DBMind: 数据库自治运维 > DBMind的AI子功能 > Index-advisor：索引推荐 > 虚拟索引”章节。

- hypopg_drop_index(oid)

  描述：删除指定的虚拟索引。

  参数：索引的oid

  返回值类型：bool

示例请参见《特性指南》的“DBMind: 数据库自治运维 > DBMind的AI子功能 > Index-advisor：索引推荐 > 虚拟索引”章节。

- hypopg_reset_index([text])

  描述：清除所有虚拟索引。

  参数：清除的虚拟索引级别（可选）

  返回值类型：无

  示例请参见《特性指南》的“DBMind: 数据库自治运维 > DBMind的AI子功能 > Index-advisor：索引推荐 > 虚拟索引”章节。

- hypopg_estimate_size(oid)

  描述：估计指定索引创建所需的空间大小。

  参数：索引的oid

  返回值类型：int8

  示例请参见《特性指南》的“DBMind: 数据库自治运维 > DBMind的AI子功能 > Index-advisor：索引推荐 > 虚拟索引”章节。

- check_engine_status(ip text, port text)

  描述：测试给定的ip和port上是否有predictor engine提供服务。

  参数：predictor engine的ip地址和端口号。

  返回值类型：text

  说明

  该函数当前版本不可用。

- encode_plan_node(optname text, orientation text, strategy text, options text, dop int8, quals text, projection text)

  描述：对入参的计划算子信息进行编码。

  参数：计划算子信息。

  返回值类型：text。

  说明

  该函数为内部功能调用函数，不建议用户直接使用。

- model_train_opt(template text, model text)

  描述：训练给定的查询性能预测模型。

  参数：性能预测模型的模板名和模型名。

  返回值类型：tartup_time_accuracy FLOAT8, total_time_accuracy FLOAT8, rows_accuracy FLOAT8, peak_memory_accuracy FLOAT8

  说明

  该函数当前版本不可用。

- track_model_train_opt(ip text, port text)

  描述：返回给定ip和port predictor engine的训练日志地址。

  参数：predictor engine的ip地址和端口号。

  返回值类型：text

  说明

  该函数当前版本不可用。

- encode_feature_perf_hist(datname text)

  描述：将目标数据库已收集的历史计划算子进行编码。

  参数：数据库名。

  返回值类型：queryid bigint, plan_node_id int, parent_node_id int, left_child_id int, right_child_id int, encode text, startup_time bigint, total_time bigint, rows bigint, peak_memory int

- gather_encoding_info(datname text)

  描述：调用encode_feature_perf_hist，将编码好的数据进行持久化保存。

  参数：数据库名。

  返回值类型：int

- db4ai_predict_by_bool (text, VARIADIC "any")

  描述：获取返回值为布尔型的模型进行模型推断任务。此函数为内部调用函数，建议直接使用语法PREDICT BY进行推断任务。

  参数：模型名称和推断任务的输入列。

  返回值类型：bool

  说明

  不建议用户直接使用。

- db4ai_predict_by_float4(text, VARIADIC "any")

  描述：获取返回值为float4的模型进行模型推断任务。此函数为内部调用函数，建议直接使用语法PREDICT BY进行推断任务。

  参数：模型名称和推断任务的输入列。

  返回值类型：float

  说明

  不建议用户直接使用。

- db4ai_predict_by_float8(text, VARIADIC "any")

  描述：获取返回值为float8的模型进行模型推断任务。此函数为内部调用函数，建议直接使用语法PREDICT BY进行推断任务。

  参数：模型名称和推断任务的输入列。

  返回值类型：float

  说明

  不建议用户直接使用。

- db4ai_predict_by_int32(text, VARIADIC "any")

  描述：获取返回值为int32的模型进行模型推断任务。此函数为内部调用函数，建议直接使用语法PREDICT BY进行推断任务。

  参数：模型名称和推断任务的输入列。

  返回值类型：int

  说明

  不建议用户直接使用。

- db4ai_predict_by_int64(text, VARIADIC "any")

描述：获取返回值为int64的模型进行模型推断任务。此函数为内部调用函数，建议直接使用语法PREDICT BY进行推断任务。

参数：模型名称和推断任务的输入列。

返回值类型：int

说明

不建议用户直接使用。

- db4ai_predict_by_numeric(text, VARIADIC "any")

  描述：获取返回值为numeric的模型进行模型推断任务。此函数为内部调用函数，建议直接使用语法PREDICT BY进行推断任务。

  参数：模型名称和推断任务的输入列。

  返回值类型：numeric

  说明

  不建议用户直接使用。

- db4ai_predict_by_text(text, VARIADIC "any")

  描述：获取返回值为字符型的模型进行模型推断任务。此函数为内部调用函数，建议直接使用语法PREDICT BY进行推断任务。

  参数：模型名称和推断任务的输入列。

  返回值类型：text

  说明

  不建议用户直接使用。

- db4ai_predict_by_float8_array(text, VARIADIC "any")

  描述：获取返回值为字符型的模型进行模型推断任务。此函数为内部调用函数，建议直接使用语法PREDICT BY进行推断任务。

  参数：模型名称和推断任务的输入列。

  返回值类型：text

  说明

  不建议用户直接使用。

- gs_explain_model(text)

  描述：获取返回值为字符型的模型进行模型解析文本化任务。

  参数：模型名称。

  返回值类型：text

- gs_ai_stats_explain(text, text[])

  描述：打印对应表和列上的多列智能统计信息。

  参数：表名称和列名集合。

  返回值类型：text

  示例：

```
gaussdb=# SET enable_ai_stats=1;
gaussdb=# drop table db4ai_bayesnet;
gaussdb=# create table db4ai_bayesnet(attr1 VARCHAR(256), attr2 VARCHAR(256));
gaussdb=# INSERT INTO db4ai_bayesnet SELECT 'x','x' FROM generate_series(1, 10000) AS s(i);
gaussdb=# ANALYZE db4ai_bayesnet((attr1,attr2));
gaussdb=# select gs_ai_stats_explain('db4ai_bayesnet', ARRAY['attr1', 'attr2']);
```


- ai_watchdog_detection_warnings()
  描述：获取AI Watchdog的风险告警信息。
  参数：无。
  返回值类型：record
  示例：
  ```
  gaussdb=# select * from ai_watchdog_detection_warnings();
  ```
- ai_watchdog_monitor_status(int)
  描述：获取AI Watchdog的监控信息。
  参数：返回监控序列的长度上限，取值范围为(0,100)，默认值为10。
  返回值类型：record
  示例：
  ```
  gaussdb=# select * from ai_watchdog_monitor_status();
  ```
- ai_watchdog_parameters()
  描述：获取AI Watchdog的内部参数或状态信息。
  参数：无。
  返回值类型：record
  示例：
  ```
  gaussdb=# select * from ai_watchdog_parameters();
  ```

7.5.33 动态数据脱敏函数

说明

该函数为内部功能调用函数。

- creditcardmasking(col text, letter char default 'x')
  描述：将col字符串后四位之前的数字使用letter替换。
  参数：待替换的字符串、替换字符。
  返回值类型：text
  示例：
  ```
  gaussdb=# select * from creditcardmasking('4511-8454-2178-6551', 'x');
    creditcardmasking
  ---------------------
   xxxx-xxxx-xxxx-6551
  (1 row)
  ```
- basicemailmasking(col text, letter char default 'x')
  描述：将col字符串中第一个'@'之前的字符使用letter替换。
  参数：待替换的字符串、替换字符。
  返回值类型：text
  示例：
  ```
  gaussdb=# select * from basicemailmasking('Alex15@huawei.com','x');
    basicemailmasking
  --------------------
   xxxxxx@huawei.com
  (1 row)
  ```
- fullemailmasking(col text, letter char default 'x')
  描述：将col字符串中出现最后一个'.'之前的字符(除'@'外)使用letter替换。

参数：待替换的字符串、替换字符。

返回值类型：text

示例：

```
gaussdb=# select * from fullemailmasking('Alex15@huawei.com','x');
fullemailmasking
-------------------
 xxxxxx@xxxxxx.com
(1 row)
```

- alldigitsmasking(col text, letter char default '0')

  描述：将col字符串中出现的数字使用letter替换。

  参数：待替换的字符串、替换字符。

  返回值类型：text

  示例：

```
gaussdb=# select * from  alldigitsmasking('abcdef 123456 ui 323 jsfd321 j3k2l3','0');
          alldigitsmasking
--------------------------------------
 abcdef 000000 ui 000 jsfd000 j0k0l0
(1 row)
```

- shufflemasking(col text)

  描述：将col字符串中的字符乱序排列。

  参数：待替换的字符串、替换字符。

  返回值类型：text

  示例：

```
gaussdb=#  select * from  shufflemasking('abcdef 123456 ui 323 jsfd321 j3k2l3');
           shufflemasking
--------------------------------------
  22dc3316 3jb af4e3f135sjl ud2 k32i
(1 row)
```

- randommasking(col text)

  描述：将col字符串中的字符随机化。

  参数：待替换的字符串、替换字符。

  返回值类型：text

  示例：

```
gaussdb=#  select * from  randommasking('abcdef');
 randommasking
---------------
 63d8dc
(1 row)
```

- regexpmasking(col text, reg text, replace_text text, pos INTEGER default 0, reg_len INTEGER default -1)

  描述：将col字符串使用正则表达式替换。

  参数：待替换的字符串、正则表达式、替换的起始位置、替换长度。

  返回值类型：text

  示例：

```
gaussdb=#  select * from regexpmasking('abcdef 123456 ui 323 jsfd321 j3k2l3','[\d+]','0');
            regexpmasking
--------------------------------------
 abcdef 000000 ui 000 jsfd000 j0k0l0
(1 row)
```


7.5.34 层次递归查询函数

层次递归查询语句中可使用以下函数返回连接路径上的相关信息。

- sys_connect_by_path(col, separator)

  描述：仅在层次递归查询中适用，用于返回从根节点到当前行的连接路径。

  参数col为在路径中显示的列的名称，只支持类型为CHAR/VARCHAR/NVARCHAR2/TEXT的列，参数separator为路径节点之间的分隔符。

  返回值类型：text

  示例：

```
gaussdb=# select *, sys_connect_by_path(name, '-') from connect_table start with id = 1 connect by prior id = pid;
 id | pid | name | sys_connect_by_path
----+-----+------+---------------------
  1 |   0 | a    | -a
  2 |   1 | b    | -a-b
  4 |   1 | d    | -a-d
  3 |   2 | c    | -a-b-c
(4 rows)
```

- connect_by_root(col)

  描述：仅在层次递归查询中适用，用于返回当前行最顶层父亲行中某列的值。

  参数col为输出列的名称。仅支持如下类型的列：INT8、INT1、INT2、OID、INT4、BOOL、CHAR、NAME、FLOAT4、FLOAT8、ABSTIME、RELTIME、DATE、CASH、TIME、TIMESTAMP、TIMESTAMPTZ、SMALLDATETIME、UUID、INTERVAL、TIMETZ、INT2VECTOR、CLOB、NVARCHAR2、VARCHAR、TEXT、VECTOR、BPCHAR、RAW、BYTEA、NUMERIC、XID、CID以及TID，且通过强制类型转换为上述类型的操作无法绕开白名单限制，如：connect_by_root(col::text)（其中col不是上述白名单中的类型）。

  返回值类型：即为所指定列col的数据类型。

  示例：

```
gaussdb=# select *, connect_by_root(name) from connect_table start with id = 1 connect by prior id = pid;
 id | pid | name | connect_by_root
----+-----+------+-----------------
  1 |   0 | a    | a
  2 |   1 | b    | a
  4 |   1 | d    | a
  3 |   2 | c    | a
(4 rows)
```

7.5.35 内部函数

GaussDB中下列函数使用了内部数据类型，用户无法直接调用，在此章节列出。

- 选择率计算函数

areajoinsel	areasel	arraycontjoinsel	arraycontsel	contjoinsel	contsel	eqjoinsel
eqsel	iclikejoinsel	iclikesel	icnlikejoinsel	icnlikesel	icregexeqjoinsel	icregexeqsel
icregexnejoinsel	icregexnesel	likejoinsel	likesel	neqjoinsel	neqsel	nlikejoinsel

nlikesel	positionjoinsel	positionsel	regexeqjoinsel	regexeqsel	regexnejoinsel	regexnesel
scalargtjoinsel	scalargtsel	scalarltjoinsel	scalarltsel	tsmatchjoinsel	tsmatchsel	-

- 统计信息收集函数

array_typanalyze	range_typanalyze	ts_typanalyze
local_rto_stat	standby_statement_history_internal	-

- 排序内部功能函数

bpchar_sortsupport	bytea_sortsupport	date_sortsupport	numeric_sortsupport	timestamp_sortsupport

- 内部类型处理函数

abstimerecv	euc_jis_2004_to_utf8	int2recv	line_recv	oidvectorrecv_extend	tidrecv	utf8_to_koi8u
anyarray_recv	euc_jp_to_mic	int2vectorrecv	lseg_recv	path_recv	time_recv	utf8_to_shift_jis_2004
array_recv	euc_jp_to_sjis	int4recv	macaddr_recv	pg_node_tree_recv	time_transform	utf8_to_sjis
ascii_to_mic	euc_jp_to_utf8	int8recv	mic_to_ascii	point_recv	timestamp_recv	utf8_to_uhc
ascii_to_utf8	euc_kr_to_mic	internal_out	mic_to_big5	poly_recv	timestamp_transform	utf8_to_win
big5_to_euc_tw	euc_kr_to_utf8	interval_recv	mic_to_euc_cn	pound_nexttoken	timestamptz_recv	uuid_recv
big5_to_mic	euc_tw_to_big5	interval_transform	mic_to_euc_jp	prsd_nexttoken	timetz_recv	varbit_recv
big5_to_utf8	euc_tw_to_mic	iso_to_koi8r	mic_to_euc_kr	range_recv	tinterval recv	varbit_transform
bit_recv	euc_tw_to_utf8	iso_to_mic	mic_to_euc_tw	rawrecv	tsqueryrecv	varchar_transform

boolrecv	float4recv	iso_to_win1251	mic_to_iso	record_recv	tsvectorrecv	varcharrecv
box_recv	float8recv	iso_to_win866	mic_to_koi8r	regclassrecv	txid_snapshot_recv	void_recv
bpcharrecv	gb18030_to_utf8	iso8859_1_to_utf8	mic_to_latin1	regconfigrecv	uhc_to_utf8	win_to_utf8
btoidsortsupport	gbk_to_utf8	iso8859_to_utf8	mic_to_latin2	regdictionaryrecv	unknownrecv	win1250_to_latin2
byteearecv	-	johab_to_utf8	mic_to_latin3	regoperatorrecv	utf8_to_ascii	win1250_to_mic
byteawithoutorderwithequalcolrecv	gtsvector_compress	json_recv	mic_to_latin4	regoperrecv	utf8_to_big5	win1251_to_iso
cash_recv	gtsvector_consistent	koi8r_to_iso	mic_to_sjis	regprocedurerecv	utf8_to_euc_cn	win1251_to_koi8r
charrecv	gtsvector_decompress	koi8r_to_mic	mic_to_win1250	regprocrecv	utf8_to_euc_jis_2004	win1251_to_mic
cidr_recv	gtsvector_penalty	koi8r_to_utf8	mic_to_win1251	regtyperecv	utf8_to_euc_jp	win1251_to_win866
cidrecv	gtsvector_picksplit	koi8r_to_win1251	mic_to_win866	reltimerecv	utf8_to_euc_kr	win866_to_iso
circle_recv	gtsvector_same	koi8r_to_win866	namerecv	shift_jis_2004_to_euc_jis_2004	utf8_to_euc_tw	win866_to_koi8r
cstring_recv	gtsvector_union	koi8u_to_utf8	ngram_nexttoken	shift_jis_2004_to_utf8	utf8_to_gb18030	win866_to_mic
date_recv	hll_recv	latin1_to_mic	numeric_recv	sjis_to_euc_jp	utf8_to_gbk	win866_to_win1251
domain_recv	hll_trans_recv	latin2_to_mic	numeric_transform	sjis_to_mic	utf8_to_iso8859	xidrecv

euc_cn_to_mic	-	latin2_to_win1250	nvarchar2recv	sjis_to_utf8	utf8_to_iso8859_1	xidrecv4
euc_cn_to_utf8	inet_recv	latin3_to_mic	oidrecv	smalldatetime_recv	utf8_to_johab	xml_recv
euc_jis_2004_to_shift_jis_2004	int1recv	latin4_to_mic	oidvectorrecv	textrecv	utf8_to_koi8r	-
i16toi1	int16	int16_bool	int16eq	int16div	int16ge	int16gt
int16in	int16le	int16lt	int16mi	int16mul	int16ne	int16out
int16pl	int16recv	int16send	numeric_bool	int2vectorin_extend	int2vectorout_extend	int2vectorrecv_extend
int2vectorsend_extend	tdigest_in	tdigest_merge	tdigest_merge_to_one	tdigest_mergep	tdigest_out	-
anyset_out	btint2setcmp	btint4setcmp	btint8setcmp	btsetcmp	btsetint2cmp	btsetint4cmp
btsetint8cmp	btsetsortsupport	float4	float8	hashsetint	hashsettext	int2
int2seteq	int2setge	int2setgt	int2setle	int2setlt	int2setne	int4
int4seteq	int4setge	int4setgt	int4setle	int4setlt	int4setne	int8
int8seteq	int8setge	int8setgt	int8setle	int8setlt	int8setne	set
set_in	set_out	set_recv	set_send	seteq	setge	setgt
setint2eq	setint2ge	setint2gt	setint2le	setint2lt	setint2ne	setint4eq
setint4ge	setint4gt	setint4le	setint4lt	setint4ne	setint8eq	setint8ge
setint8gt	setint8le	setint8lt	setint8ne	setle	setlt	setne
settexteq	settextge	settextgt	settextle	settextlt	settextne	settobpchar
settonumber	settonvarchar2	settotext	settovarchar	textseteq	textsetge	textsetgt

textsetle	textsetlt	textsetne	gb18030_2022_to_utf8	utf8_to_gb18030_2022	array_to_nesttable	array_to_indexby_int_table

- 聚合操作内部函数

array_agg_finalfn	array_agg_transfn	bytea_string_agg_finalfn	bytea_string_agg_transfn	date_list_agg_noarg2_transfn	date_list_agg_transfn	float4_list_agg_noarg2_transfn
float4_list_agg_transfn	float8_list_agg_noarg2_transfn	float8_list_agg_transfn	int2_list_agg_noarg2_transfn	int2_list_agg_transfn	int4_list_agg_noarg2_transfn	int4_list_agg_transfn
int8_list_agg_noarg2_transfn	int8_list_agg_transfn	interval_list_agg_noarg2_transfn	interval_list_agg_transfn	list_agg_finalfn	list_agg_noarg2_transfn	list_agg_transfn
median_float8_finalfn	median_interval_finalfn	median_transfn	mode_final	numeric_list_agg_noarg2_transfn	numeric_list_agg_transfn	ordered_set_transition
percentile_cont_float8_final	percentile_cont_interval_final	string_agg_finalfn	string_agg_transfn	timestamp_list_agg_noarg2_transfn	timestamp_list_agg_transfn	timestamptz_list_agg_noarg2_transfn
timestamptz_list_agg_transfn	checksumtext_agg_transfn	-	-	-	-	-

- 哈希内部功能函数

hashbeginscan	hashbuild	hashbuildempty	hashbulkdelete	hashcostestimate	hashendscan	hashgetbitmap
hashgettuple	hashinsert	hashmarkpos	hashmerge	hashrescan	hashrestrpos	hashvacuumcleanup
hashvarlena	-	-	-	-	-	-

- Btree索引内部功能函数

cbtreebuild	cbtreecanreturn	cbtreecostestimate	cbtreegetbitmap	cbtreegettuple	btbeginscan	btbuild
btbuildempty	btbulkdelete	btcanreturn	btcostestimate	btendscan	btfloat4sortsupport	btfloat8sortsupport
btgetbitmap	btgettuple	btinsert	btint2sortsupport	btint4sortsupport	btint8sortsupport	btmarkpos
btmerge	btnamesortsupport	btrescan	btrestrpos	bttextsortsupport	btvacuumcleanup	cbtreeoptions

- Psort索引内部函数

psortbuild	psortcanreturn	psortcostestimate	psortgetbitmap	psortgettuple

- Ubtree索引内部函数

ubtbeginscan	ubtbuild	ubtbuildempty	ubtbulkdelete	ubtcanreturn
ubtcostestimate	ubtendscan	ubtgetbitmap	ubtgettuple	ubtinsert
ubtmarkpos	ubtmerge	ubtoptions	ubtrescan	ubtrestrpos
ubtvacuumcleanup	-	-	-	-

- plpgsql内部函数

  plpgsql_inline_handler
- 集合相关内部函数

array_indexby_delete	array_indexby_length	array_integer_deleteidx	array_integer_exists	array_integer_first	array_integer_last
array_integer_next	array_integer_prior	array_varchar_deleteidx	array_varchar_exists	array_varchar_first	array_varchar_last
array_varchar_next	array_varchar_prior	-	-	-	-

- 外表相关内部函数

dist_fdw_handler	roach_handler	streaming_fdw_handler	dist_fdw_validator	file_fdw_handler	file_fdw_validator	log_fdw_handler

dblink_fdw_handler	dblink_fdw_validator	-	-	-	-	-

- 主DN远程读取备DN数据页辅助函数

  gs_read_block_from_remote用于读取表文件的页面。默认只有初始化用户可以查看，其余用户需要赋权后才可以使用。

- 主DN远程读取备DN数据文件辅助函数

  gs_read_file_from_remote用于读取指定的文件。gs_repair_file利用gs_read_file_size_from_remote函数获取文件大小后，依赖这个函数将远端文件逐段读取。默认只有初始化用户可以查看，其余用户需要赋权后才可以使用。

  gs_read_file_size_from_remote用于读取指定文件的大小。gs_repair_file函数修复文件时，要先获取远端关于这个文件的大小，用于校验本地文件缺失的文件信息，然后将缺失的文件逐个修复。默认只有初始化用户可以查看，其余用户需要赋权后才可以使用。

- 以备DN增量重建其他备或级联备DN辅助函数

  gs_standby_incremental_filemap_create用于创建备DN增量重建临时filemap文件，用于存储当次增量重建需传输的数据路径与大小。只有初始化用户且application为gs_rewind时可以调用。

  gs_standby_incremental_filemap_insert用于向指定临时filemap文件中插入文件信息，指定文件的路径、传输起始点、单次传输数据长度和rebuild标志位。只有初始化用户且application为gs_rewind时可以调用。

  gs_standby_incremental_filemap_execute用于获取指定临时filemap文件中存储的文件信息并删除指定filemap，用于备DN增量重建的数据传输。只有初始化用户且application为gs_rewind时可以调用。

- AI特性函数

create_snapshot	create_snapshot_internal	prepare_snapshot_internal	prepare_snapshot	manage_snapshot_internal	archive_snapshot	publish_snapshot
purge_snapshot_internal	purge_snapshot	sample_snapshot	-	-	-	-

  – PKG_SERVICE函数

isubmit_on_nodes	submit_on_nodes	-	-	-	-	-

- 其他函数

to_tsvector_for_batch	value_of_percentile	disable_conn	bind_variable	job_update	job_cancel	job_finish
similar_escape	table_skewness（不可用）	timetz_text	time_text	reltime_text	abstime_text	_pg_keysequal
analyze_query (不可用）	analyze_workload（不可用）	ssign_table_type	gs_comm_proxy_thread_status	gs_txid_oldestxmin	-	pg_stat_segment_space_info
remote_segment_space_info	set_cost_params	set_weight_params	start_collect_workload	tdigest_in	tdigest_merge	tdigest_merge_to_one
tdigest_mergep	tdigest_out	-	-	-	-	-

- 视图相关引用函数

  adm_hist_sqlstat_func

  adm_hist_sqlstat_idlog_func

  adm_hist_sqltext_func

- gs_txn_snapshot系统表维护函数

  gs_insert_delete_txn_snapshot用于GTM-Lite模式下维护全局各节点gs_txn_snapshot系统表，只有系统管理员用户才能调用，当前版本调用该函数将返回f，无实际操作。

- xmltype类型相关函数

  isschemavalid

7.5.36 Global SysCache 特性函数

- gs_gsc_table_detail(database_id default NULL, rel_id default NULL)

  描述：查看数据库里全局系统缓存的表元数据。调用该函数的用户需要具有SYSADMIN权限。

  参数：指定需要查看全局系统缓存的数据库和表，database_id默认值NULL或者-1表示所有的数据库，0表示共享表，其他数字表示指定数据库及共享表，rel_id表示指定表的oid，默认值NULL或者-1表示所有的表，其他值表示指定的表，database_id不存在会报错，rel_id不存在结果为空。

  返回值类型：Tuple

  示例:

```
select * from gs_gsc_table_detail(-1) limit 1;
database_oid | database_name | reloid |        relname         | relnamespace | reltype | reloftype |
relowner | relam | relfilenode | reltablespace | relhasindex | relisshared | relkind | relnatts | relhasoids |
relhaspkey | parttype | tdhasuids | attnames  | extinfo
--------------+---------------+---------+-------------------------+--------------+---------+-----------+----------
+-------+-------------+---------------+-------------+-------------+---------+----------+------------+------------
```



```
+----------+-----------+-----------+---------
0 |          |   2676 | pg_authid_rolname_index |          11 |       0 |         0 |       10 |   403 |          0
|         1664 | f          | t          | i       |         1 | f         | f         | n       | f         | 'rolname' |
(1 row)
```

- gs_gsc_catalog_detail(database_id default NULL, rel_id default NULL)

  描述：查看数据库里全局系统缓存的系统表行信息。调用该函数的用户需要具有SYSADMIN权限。

  参数：指定需要查看全局系统缓存的数据库和表，database_id默认值NULL或者-1表示所有的数据库，0表示共享表，其他数字表示指定数据库及共享表，rel_id表示指定表的id，仅包含所有有系统缓存的系统表，默认值NULL或者-1表示所有的表，其他值表示指定的表，database_id不存在会报错，rel_id不存在结果为空。

  返回值类型：Tuple

  示例：

```
--首先通过pg_database获取特定数据库的oid，查询语句一般为：SELECT oid, * FROM pg_database;
--返回元组中通过datname列找到对应的oid列的值，然后执行如下查询，示例获取的oid为16574。
gaussdb=# select * from gs_gsc_catalog_detail(16574, 1260);
 database_id | database_name | rel_id | rel_name  | cache_id |  self  |  ctid  | infomask | infomask2 |
hash_value | refcount
-------------+---------------+--------+-----------+----------+--------+--------+----------+-----------+------------
+----------
           0 |               |   1260 | pg_authid |       10 | (0, 9) | (0, 9) |    10507 |        26 |  531311568 |
10
           0 |               |   1260 | pg_authid |       11 | (0, 4) | (0, 4) |     2313 |        26 |  365368336 |        1
           0 |               |   1260 | pg_authid |       11 | (0, 9) | (0, 9) |    10507 |        26 | 3911517328 |
10
           0 |               |   1260 | pg_authid |       11 | (0, 7) | (0, 7) |     2313 |        26 | 1317799983 |
1
           0 |               |   1260 | pg_authid |       11 | (0, 5) | (0, 5) |     2313 |        26 | 3664347448 |
1
           0 |               |   1260 | pg_authid |       11 | (0, 1) | (0, 1) |     2313 |        26 |  276477273 |        1
           0 |               |   1260 | pg_authid |       11 | (0, 3) | (0, 3) |     2313 |        26 | 2465837659 |
1
           0 |               |   1260 | pg_authid |       11 | (0, 8) | (0, 8) |     2313 |        26 | 3205288035 |
1
           0 |               |   1260 | pg_authid |       11 | (0, 6) | (0, 6) |     2313 |        26 |  131811687 |        1
           0 |               |   1260 | pg_authid |       11 | (0, 2) | (0, 2) |     2313 |        26 | 1226484587 |
1
(10 rows)
```

- gs_gsc_clean(database_id default NULL)

  描述：清理global syscache的缓存，需要注意，正在使用中的数据不会被清理。调用该函数的用户需要具有SYSADMIN权限。

  参数：指定需要清理全局系统缓存的数据库，默认值NULL或者-1表示强制清理所有的数据库全局系统缓存，0表示只淘汰共享表的全局系统缓存，其他数字表示淘汰指定数据库以及共享表的全局系统缓存，database_id不存在会报错。

  返回值类型：bool

  示例：

```
gaussdb=# select * from gs_gsc_clean();
gs_gsc_clean
--------------
t
(1 row)
```

- gs_gsc_dbstat_info(database_id default NULL)

  描述：获取本地节点的GSC的内存统计信息，包括tuple、relation、partition的缓存查询，命中，加载、失效、占用空间信息，DB级别的淘汰信息，线程引用信息，内存占用信息。可以用于定位性能问题，例如当发现hits/searches数组远小

于1时，可能是global_syscache_threshold设置太小，导致查询命中率下降。调用该函数的用户需要具有SYSADMIN权限。

参数：指定需要查看的数据库全局系统缓存统计信息，NULL或者-1表示查看所有的数据库，0表示只查看共享表信息，其他数字表示查看指定的数据库和共享表的信息。不合法的输入值，database_id不存在会报错。

返回值类型：Tuple

示例：

```
gaussdb=# select * from gs_gsc_dbstat_info();
database_id | database_name | tup_searches | tup_hits | tup_miss | tup_count | tup_dead | tup_memory
| rel_searches | rel_hits | rel_mis
s | rel_count | rel_dead | rel_memory | part_searches | part_hits | part_miss | part_count | part_dead |
part_memory | total_memory | swa
pout_count | refcount
-------------+---------------+--------------+----------+----------+-----------+----------+------------
+--------------+----------+--------
--+-----------+----------+------------+---------------+-----------+-----------+------------+-----------
+-------------+--------------+----
-----------+----------
          0 |               |          300 |      235 |       31 |        22 |        2 |       9752 |          598 |      108
|       1
8 |        18 |        0 |     77720 |            0 |         0 |         0 |         0 |         0 |          0 |       752912 |
         0 |        0
       16574 | testdb      |         3368 |     2289 |      329 |       273 |        0 |      92593 |         1113 |
524 |       4
8 |        48 |        0 |    340456 |            0 |         0 |         0 |         0 |         0 |          0 |      4124792 |
         0 |       10
(2 rows)
```

7.5.37 数据损坏检测修复函数

修复主机文件页面的约束概述：

文件类型	文件页面级别	主备机	检测修复
普通行存表(astore，ustore，包括压缩表)，不包括索引	文件&页面	主机	手动检测手动修复。
undo(不包括undo meta)	页面	主机	手动检测手动修复（不包括analyse verify）。
unlogged表的init fork文件	文件	主机	手动检测手动修复。

- gs_verify_data_file(verify_segment bool)

  描述：校验当前实例当前库是否存在文件丢失的情况。校验只包括数据表主文件是否有中间段丢失的情况。默认参数是false。参数设置为true时为预留参数，当前暂不支持。默认只有初始化用户、具有sysadmin属性的用户以及在运维模式下具有运维管理员属性的用户可以查看，其余用户需要赋权后才可以使用。

  返回的结果：

  - rel_oid和rel_name是对应文件的表oid和表名，miss_file_path表示丢失文件的相对路径。

  参数说明：

  - verify_segment

指定文件校验的范围。

取值范围：true和false，默认是false。true为预留参数，当前暂不支持。

返回值类型：record

示例：（仅当发现有异常行时才会输出异常行，否则输出0行。）

```
gaussdb=# select * from gs_verify_data_file();
node_name         | rel_oid |  rel_name    |  miss_file_path
------------------+---------+--------------+------------------
dn_6001_6002_6003 |   16554 |     test     | base/16552/24745
```

- gs_repair_file(tableoid Oid, path text, timeout int)

  描述：根据传入的参数修复文件，仅支持有正常主备连接的主DN使用。参数依据gs_verify_data_file函数返回的oid和路径填写。修复成功返回值为true，修复失败会显示具体失败原因。默认只有在主DN节点上，初始用户、具有sysadmin属性的用户以及在运维模式下具有运维管理员属性的用户可以查看，其余用户需要赋权后才可以使用。

  ⚠ 注意

  1. 当DN实例上存在文件损坏时，进行升主会校验出错，报PANIC退出无法升主，为正常现象。
  2. 当文件存在但是大小为0时，此时不会去修复该文件，若想要修复该文件，需要将为0的文件删除后再修复。
  3. 删除文件需要等文件fd自动关闭后再修复，人工操作可以执行重启进程、主备切换命令。

  参数说明：

  - tableoid

    要修复的文件对应的表oid，依据gs_verify_data_file函数返回的列表中rel_oid一列填写。

    取值范围： Oid，0 - 4294967295。注意：输入负值等都会被强制转成非负整数类型。

  - path

    需要修复的文件路径，依据gs_verify_data_file函数返回的列表中miss_file_path一列填写。

    取值范围：字符串。

  - timeout

    等待备DN回放的时长，修复文件需要等待备DN回放到当前主DN对应的位置，根据备DN回放所需时长设定。

    取值范围：60s - 3600s。

  返回值类型：bool

  示例：（按照gs_verify_data_file的输出填写tablespace和path）

  ```
  gaussdb=# select * from gs_repair_file(16554,'base/16552/24745',360);
  gs_repair_file
  ----------------
  t
  ```

- local_bad_block_info()

  描述：显示本实例页面损坏的情况。从磁盘读取页面，发现页面CRC校验失败时进行记录。默认只有初始化用户、具有sysadmin属性的用户、具有监控管理员属

性的用户以及在运维模式下具有运维管理员属性的用户、以及监控用户可以查看，其余用户需要赋权后才可以使用。

显示信息：file_path是损坏文件的相对路径。block_num是该文件损坏的具体页面号，页面号从0开始。check_time表示发现页面损坏的时间。repair_time表示修复页面的时间。

返回值类型：record

示例：（仅当有损坏记录时输出相关条目，否则输出0行）

```
gaussdb=# select * from local_bad_block_info();
node_name    | spc_node | db_node | rel_node| bucket_node | fork_num | block_num |    file_path     |
check_time             |   repair_time
-----------------+--------+---------+---------+-------------+----------+-----------+------------------
+-------------------------+-------------------------------
dn_6001_6002_6003|  1663 |  16552 |  24745 |       -1    |    0    | 0        | base/16552/24745 |
2022-01-13 20:19:08.385004+08 | 2022-01-13 20:19:08.407314+08
```

- local_clear_bad_block_info()

  描述：清理local_bad_block_info中已修复页面的数据，也就是repair_time不为空的信息。默认只有初始化用户、具有sysadmin属性的用户以及在运维模式下具有运维管理员属性的用户、以及监控用户可以查看，其余用户需要赋权后才可以使用。

  返回值类型：bool

  示例：

  ```
  gaussdb=# select * from local_clear_bad_block_info();
  result
  --------
  t
  ```

- gs_verify_and_tryrepair_page (path text, blocknum oid, verify_mem bool, is_segment bool)

  描述：校验本实例指定页面的情况。默认只有在主DN节点上，使用初始化用户、具有sysadmin属性的用户以及在运维模式下具有运维管理员属性的用户可以查看，其余用户需要赋权后才可以使用。

  返回的结果信息：disk_page_res表示磁盘上页面的校验结果；mem_page_res表示内存中页面的校验结果；is_repair表示在校验的过程中是否触发修复功能，t表示已修复，f表示未修复。

  注意：

  a. 当DN实例上存在页面损坏时，进行升主会校验出错，报PANIC退出无法升主，为正常现象。不支持hashbucket表页面损坏的修复。
  b. 此函数触发的修复仅支持修复内存中的页面，需要在内存页面落盘后物理页面修复才能正式生效。

  参数说明：

  – path

    损坏文件的路径。依据local_bad_block_info中file_path一列填写。如果要对存储类型为USTORE的表进行UNDO页面校验，请直接填写需要校验的UNDO页面路径。

    取值范围：字符串。

  – blocknum

    损坏文件的页号。依据local_bad_block_info中block_num一列填写。如果要对存储类型为USTORE的表进行UNDO页面校验，请直接填写需要校验的UNDO页面的块号。

取值范围：Oid，0 - 4294967295。注意：输入负值等都会被强制转成非负整数类型。

- verify_mem

  指定是否校验内存中的指定页面。设定为false时，只校验磁盘上的页面。设置为true时，校验内存中的页面和磁盘上的页面。如果发现磁盘上页面损坏，会将内存中的页面做一个基本信息校验刷盘，修复磁盘上页面。如果校验内存页面时发现页面不在内存中，会经内存接口读取磁盘上的页面。此过程中如果磁盘页面有问题，则会触发远程读自动修复功能。

  取值范围：bool，true和false。

- is_segment

  是否是段页式表。false表示不是段页式表，true为预留参数值，当前暂不支持。

  取值范围：bool，true和false。

返回值类型：record

示例：（请依据local_bad_block_info的输出传参，否则报错。）

```
gaussdb=# select * from gs_verify_and_tryrepair_page('base/16552/24745',0,false,false);
node_name        |       path       |  blocknum |        disk_page_res         | mem_page_res | is_repair
-----------------+------------------+-----------+------------------------------+--------------+----------
dn_6001_6002_6003 | base/16552/24745 |     0     | page verification succeeded.|              | f
```

- gs_repair_page(path text, blocknum oid, is_segment bool, timeout int)

  描述：修复本实例指定页面，仅支持有正常主备连接的主DN使用。页面修复成功返回true，修复过程中出错会有报错信息提示。默认只有在主DN节点上，使用初始化用户、具有sysadmin属性的用户以及在运维模式下具有运维管理员属性的用户可以查看，其余用户需要赋权后才可以使用。

  注意：当DN实例上存在页面损坏时，进行升主会校验出错，报PANIC退出无法升主，为正常现象。不支持hashbucket表页面损坏的修复。

  参数说明：

  - path

    损坏页面的路径。根据local_bad_block_info中file_path一列设置，或者是gs_verify_and_tryrepair_page函数中path一列设置。

    取值范围：字符串

  - blocknum

    损坏页面的页面号。根据local_bad_block_info中block_num一列设置，或者是gs_verify_and_tryrepair_page函数中blocknum一列设置。

    取值范围：Oid，0 - 4294967295。注意：输入负值等都会被强制转成非负整数类型。

  - is_segment

    是否是段页式表。false表示不是段页式表，true为预留参数值，当前暂不支持。

    取值范围：bool，true或者false。

  - timeout

    等待备DN回放的时长。修复页面需要等待备DN回放到当前主DN对应的位置，根据备DN回放所需时长设定。

    取值范围：60s - 3600s。

  返回值类型：bool

  示例：（请根据local_bad_block_info的输出传参，否则报错。）


```
gaussdb=# select * from gs_repair_page('base/16552/24745',0,false,60);
result
--------
t
```

- gs_edit_page_bypath(path text, blocknum int64, offset int, data text, data_size int, read_backup bool, storage_type text)

  描述：传入目标表文件的路径、块号、偏移量、修改的目标数据以及长度，将目标数据修改到页面对应字段中。其中，read_backup字段控制文件的读取方式，storage_type字段表示文件的存储方式（例如页式存储），并返回修改后落盘的文件路径。为防止误修改操作，该函数不会直接对原页面而是对复制页面进行修改，并将修改后的页面落盘到指定路径。只有系统管理员或者运维模式下的运维管理员才能执行此函数。

  返回值类型：text

表 7-109 gs_edit_page_bypath 参数说明

参数类型	参数名	类型	描述
输入参数	path	text	待修改文件的物理文件路径，结合read_backup字段，既可以是数据库目录下文件的相对路径，也可以是备份等文件的绝对路径。若目标文件不存在或读取失败等，返回相应报错信息。 ● read_backup为false：path路径格式为tablespace name/database oid/表的relfilenode(物理文件名)。例如：base/16603/16394。 ● read_backup为true：path为合法路径，此时由于无法获取到输入文件的其他相关信息，所以需要用户保障输入数据的正确性。 注：仅支持upage、ubtree数据页的编辑修改。不支持创建了表空间的表。由于无法获取到输入文件的其他相关信息，所以需要用户保障输入数据类型的正确性。
输入参数	blocknum	bigint	指定修复页面的块号。 参数范围：0~MaxBlockNumber。 结合read_backup字段，读取指定物理/逻辑块号对应的页面，当指定块号超出范围等，返回相应报错信息。
输入参数	offset	int	修改字段的页内偏移。 参数范围：0~BLCKSZ。 当用户指定小于0或者大于BLCKSZ的值时，使用系统视图返回相应报错信息。

参数类型	参数名	类型	描述
输入参数	data	text	修改的目标值类型。 类型： ● '0x'：表示十六进制。 ● '0b'：表示二进制。 ● '0s'：表示字符串。 其他：当data参数不为上述类型时认为数据数据为十进制字符串。
输入参数	data_size	int	写入的数据长度，单位：字节。 参数范围：1 ~ 8。 当用户指定写入长度小于1或大于8字节，又或者offset+data_size > BLCKSZ时，使用系统视图返回相应报错信息。
输入参数	read_backup	bool	是否从备份目录中读取页面，当该字段为false时，将通过逻辑块号读取目标页面，否则基于物理块号读取页面。
输入参数	storage_type	text	文件的存储方式，当前仅支持页式，可选参数： ● 'page'（页式）。 ● 'segment'（段页式），预留参数，暂不支持。
输出参数	output_msg	text	修改成功时，返回修改后文件落盘的绝对路径，修改后的文件存储于pg_log/dump目录下。若修改失败，则返回失败相关信息。

注：使用示例时请按参数说明传参并使用实际存在的物理路径。

示例1：在base/15808/25075表的0号页面偏移16字节处覆盖写入值为0X1FFF的数据。

```
gaussdb=# select gs_edit_page_bypath('base/15808/25075',0,16,'0x1FFF', 2, false, 'page');
                        gs_edit_page_bypath
-------------------------------------------------------------------
/pg_log_dir/dump/1663_15808_25075_0.editpage

(1 rows)
```

示例2：当输入参数不符合规范时返回对应错误消息。

```
gaussdb=# select gs_edit_page_bypath('base/15808/25075', 0,16,'@1231!', 8, false, 'page');
          gs_edit_page_bypath
---------------------------------------
 Error: the parameter 'data' decode failed.
(1 row)
```

示例3：当需要写入的数据与原始值相同，返回告警信息。

```
gaussdb=# select gs_edit_page_bypath('/pg_log_dir/dump/1663_15808_25075_0.editpage',
0,16,'0x1FFF', 2, true, 'page');
                  gs_edit_page_bypath
-----------------------------------------------------------
 Warning: source buffer is consistent with target buffer.
(1 row)
```


- gs_repair_page_bypath(src_path text, src_blkno int64, dest_path text, dest_blkno int64, storage_type text)

  描述：传入源文件路径以及页面号，将该页面覆盖写入到目标文件指定页面号上，支持基于备机修复主机页面。此外，该视图支持对坏块的初始化操作。

  a. 对目标页面进行覆盖写并同步备机，页式修改对象支持Uheap、Ubtree页面。后续支持Undo Record页面、Undo Slot页面、压缩表，以及Astore页面。不支持系统表文件的修改，也不支持对数据区的修改。

  b. 功能支持将页面覆盖写到目标页面上。覆盖之前会将目标页面备份并落盘到指定目录，支持将备份页面重写回目标页面，在主机对普通表的修改会生成新的WAL日志并同步备机，在备机的修改不会记录WAL日志。

  c. 修复视图仅适用于集中式与分布式的主节点，或者开启备机读场景下的备节点。用户需要系统管理员或者运维模式下的运维管理员权限，所有修改均会记录数据库日志，并且，建议使用前开启系统函数的审计日志，便于记录审计信息。

  d. 修复时源页面与目标页面的LSN必须一致，否则修复失败。

  返回值类型：text

⚠ 注意

调用本系统函数属于高危风险操作，请用户谨慎使用。

参数类型	参数名	类型	描述
输入参数	src_path	text	源文件的路径。支持的路径主要包含以下几类： • 数据文件以及索引文件：pg_log/dump/1663_15808_25075_0.editpage。 • 在主机端指定src_path为'standby'，即从备机端读取页面修复主机。 • 在主机端指定src_path为'init_block'，允许极端场景下跳过坏块。
输入参数	src_blkno	bigint	源页面的物理块号。 参数范围：0~MaxBlockNumber。
输入参数	dest_path	text	目标文件的相对路径。例如：base/15808/25075。
输入参数	dest_blkno	bigint	目标页面的逻辑块号。 参数范围：0~MaxBlockNumber。

参数类型	参数名	类型	描述
输入参数	storage_type	text	目标文件的存储方式，当前仅支持页式，可选参数： • 'page'（页式）。 • 'segment'（段页式，预留参数，暂不支持）。
输出参数	output_msg	text	覆盖写成功时返回目标页面备份的路径，失败时返回报错信息。落盘文件名格式为relfilepath_blocknum_timestamp.repairpage。

注：请按照实际情况根据上表传参并确认物理文件存在。传参有异常或修复失败时将报错。

示例1：输入指定路径下的文件，覆盖写入到目标文件中。

```
gaussdb=# select * from gs_repair_page_bypath('pg_log/dump/1663_15991_16767_0.editpage', 0, 'base/15991/16767', 0, 'page');
                                        output_msg
-------------------------------------------------------------------------------------------
 /pg_log_dir/dump/1663_15991_16767_0_738039702421788.repairpage
(1 row)
```

示例2：从备机端读取页面修复主机。

```
gaussdb=# select * from gs_repair_page_bypath('standby', 0, 'base/15990/16768', 0, 'page');
                                        output_msg
-------------------------------------------------------------------------------------------
/pg_log_dir/dump/1663_15990_16768_0_738040397197907.repairpage
(1 row)
```

示例3：初始化目标页面，支持坏块跳过。

```
gaussdb=# select * from gs_repair_page_bypath('init_block', 0, 'base/15990/16768', 0, 'page');
                                        output_msg
-------------------------------------------------------------------------------------------
/pg_log_dir/dump/1663_15990_16768_0_738040768010281.repairpage
(1 row)
```

- gs_repair_undo_byzone(zone_id int)

  描述：传入待修复Undo Zone的zone_id，对目标Undo Zone的元信息进行修复，并返回修复结果的详细信息；如果没有进行修复则没有输出信息。

  返回值类型：record

  备注：当前函数仅支持在主节点进行调用，修复成功后会通过记录xLog日志同步到备机，且调用者必须是系统管理员或者运维模式下的运维管理员，建议使用前开启系统函数的审计日志，便于记录审计信息。

⚠ 注意

调用本系统函数属于高危风险操作，请用户谨慎使用。

表 7-110 gs_repair_undo_byzone 参数说明

参数类型	参数名	类型	描述
输入参数	zone_id	int	Undo Zone编号： ● -1：修复所有Undo Zone的元信息。 ● 0~1048575：修复对应zone_id编号的Undo Zone的元信息。
输出参数	zone_id	int	Undo Zone编号。
输出参数	repair_detail	text	对应zone_id的Undo Zone元信息的修复结果，修复成功显示"rebuild undo meta succeed."；修复失败显示"rebuild undo meta failed."及其失败原因。

注：执行时根据修复情况，输出为三种情况之一。

示例1：输入的zone_id对应的Undo Zone元信息没有损坏时，预期没有输出。

```
gaussdb=# select * from gs_repair_undo_byzone(4);
 zone_id | repair_detail
---------+---------------
(0 rows)
```

示例2：输入的zone_id对应的Undo Zone元信息修复成功时，显示修复成功的信息。

```
gaussdb=# select * from gs_repair_undo_byzone(78);
 zone_id | repair_detail
---------+---------------
      78 | rebuild undo meta succeed.
(1 row)
```

示例3：输入的zone_id对应的Undo Zone元信息修复失败时，显示修复失败的详细信息。

```
gaussdb=# select * from gs_repair_undo_byzone(0);
 zone_id | repair_detail
---------+---------------
       0 | rebuild undo meta failed. try lock undo zone_id failed.
(1 row)
```

说明

如果待修复的Undo Zone已损坏且zone_id已经被其他活跃线程占用时，调用该修复函数时，占用zone_id的活用线程会自动结束，强制修复已损坏的Undo Zone元信息。

- gs_verify_urq(index_oid oid, partindex_oid oid, blocknum bigint, queue_type text)

  描述：校验索引回收队列（潜在队列/可用队列/单页面）的正确性。

  参数说明：详见**表7-111**。

  返回值类型：record

表 7-111 gs_verify_urq 参数说明

参数类型	参数名	类型	描述
输入参数	index_oid	oid	UBTree索引oid： ● 普通索引：索引oid。 ● 全局索引：GPI oid。 ● local索引：主索引oid。
输入参数	partindex_oid	oid	UBTree分区索引oid： ● 普通索引：0。 ● 全局索引：0。 ● local索引：分区索引oid（一级/二级）。
输入参数	blocknum	bigint	页面号： ● 队列类型为single page时，校验单个页面blocknum的所有元组正确性。取值范围为[0，队列文件大小/8192）。 ● 队列类型为empty queue或free queue时，blocknum是一个无效值。
输入参数	queue_type	text	队列类型： ● empty queue：潜在队列。 ● free queue：可用队列。 ● single page：队列单页面。
输出参数	error_code	text	错误码。
输出参数	detail	text	具体报错及其他关键信息。

示例1：使用示例时请根据参数说明传参，使用实际存在的oid和blocknum，否则将报错。

```
gaussdb=# select * from gs_verify_urq(16387, 0, 1, 'free queue');
 error_code | detail
------------+--------
(0 rows)
```

示例2：使用示例时请根据参数说明传参，使用实际存在的oid和blocknum，否则将报错。

```
gaussdb=# select * from gs_verify_urq(16387, 0, 1, 'empty queue');
      error_code       |
detail
-----------------------
+-----------------------------------------------------------------------------------------------------------------------
 VERIFY_URQ_PAGE_ERROR | invalid urq meta: oid 16387, blkno 1, head_blkno = 1, tail_blkno = 3,
nblocks_upper = 4294967295, nblocks_lower = 1; urq_blocks = 6, index_blocks = 12
(1 row)
```


说明

该接口当前仅支持USTORE索引表。如果索引回收队列校验正常，则该视图不输出错误码和报错详细信息；否则输出错误码和报错详细信息，错误码包含"VERIFY_URQ_PAGE_ERROR"、"VERIFY_URQ_LINK_ERROR"、"VERIFY_URQ_HEAD_MISSED_ERROR"和"VERIFY_URQ_TAIL_MISSED_ERROR"，如出现以上错误码，请联系华为工程师辅助定位。

- gs_urq_dump_stat(index_oid oid, partindex_oid oid)

  描述：查询指定索引回收队列相关信息。

  显示信息：recentGlobalDataXmin和globalFrozenXid是回收队列判断索引页面是否可以被回收时使用的两个oldestxmin，next_xid为下一个最新的事务xid，urq_blocks为回收队列总页面数，以及free queue（可用队列）、empty queue（潜在队列）有效页面里的相关信息。

  参数说明：详见表7-112。

表 7-112 gs_urq_dump_stat 参数说明

参数类型	参数名	类型	描述
输入参数	index_oid	oid	UBTree索引oid： ● 普通索引：索引oid。 ● 全局索引：GPI oid。 ● local索引：主索引oid。
输入参数	partindex_oid	oid	UBTree分区索引oid： ● 普通索引：0。 ● 全局索引：0。 ● local索引：分区索引oid（一级/二级）。
输出参数	result	text	索引回收队列的详细统计信息。

示例：使用示例时请根据实际情况按参数说明传参，使用实际存在的oid，否则将报错。

```
gaussdb=# select * from gs_urq_dump_stat(16387, 0);
                                                   result
---------------------------------------------------------------------------------------------------------------------
 urq stat info: recentGlobalDataXmin = 213156, globalFrozenXid = 213156, next_xid = 214157,
urq_blocks = 6,                    +
   free  queue:    head page blkno = 0 min_xid = 211187 max_xid = 214157, tail page blkno = 0
min_xid = 211187 max_xid = 214157,+
              middle page min_xid = 1152921504606846975 max_xid = 0, valid_pages = 1, valid_items =
6, can_use_item = 3        +
   empty queue:    head page blkno = 1 min_xid = 212160 max_xid = 213160, tail page blkno = 3
min_xid = 213162 max_xid = 214156,+
              middle page min_xid = 1152921504606846975 max_xid = 0, valid_pages = 2, valid_items =
999, can_use_item = 498    +

(1 row)
```

说明

该接口当前仅支持USTORE索引表。

- gs_repair_urq(index_oid oid, partindex_oid oid)

  描述：重建（有损）索引回收队列（潜在队列和可用队列）。删除当前索引的回收队列文件，重新创建一个空的回收队列文件。重建成功显示reinitial the recycle queue of index relation sucessfully。

  参数说明：详见**表7-113**。

  备注：当前函数仅支持在主节点进行调用。

  表 7-113 gs_repair_urq 参数说明

参数类型	参数名	类型	描述
输入参数	index_oid	oid	UBTree索引oid： • 普通索引：索引oid。 • 全局索引：GPI oid。 • local索引：主索引oid。
输入参数	partindex_oid	oid	UBTree分区索引oid： • 普通索引：0。 • 全局索引：0。 • local索引：分区索引oid（一级/二级）。
输出参数	result	text	重建成功显示reinitial the recycle queue of index relation sucessfully。

  示例：使用示例时请根据实际情况按参数说明传参，使用实际存在的oid，否则将报错。

```
gaussdb=# select * from gs_repair_urq(16387, 0);
                              result
-------------------------------------------------------------
 reinitial the recycle queue of index relation sucessfully.
(1 row)
```

  说明

  该接口当前仅支持USTORE索引表。

- gs_get_standby_bad_block_info()

  描述：显示备机上已经检测到但是还未修复的页面。默认只有在备DN节点上，使用初始用户、具有sysadmin权限的用户以及在运维模式下具有运维管理员权限的用户、以及监控用户可以查看，其余用户需要赋权后才可以使用。返回值invalid_type列共有4种类型：NOT_PRESENT（页面不存在）、NOT_INITIALIZED（页面初始化失败）、 LSN_CHECK_ERROR（LSN校验失败）、CRC_CHECK_ERROR（CRC校验失败）。

  返回值类型：record

  示例：（若不存在已检测到但未修复的页面，则输出0行）

```
gaussdb=# select * from gs_get_standby_bad_block_info();
 spc_node | db_node | rel_node | bucket_node | fork_num | block_num |  invalid_type   |
master_page_lsn
----------+---------+----------+-------------+----------+-----------+-----------------+-----------------
```



```
  1663  |  16552 |  24745 |     -1  |   0   |    0    | CRC_CHECK_ERROR | 0/B2009E8
(1 rows)
```

7.5.38 XML 类型函数

以下函数兼容Postgres 9.2。

- xmlparse ({ DOCUMENT | CONTENT } value [wellformed])

  描述：使用函数xmlparse，从字符数据产生XML类型的值。

  参数：数据类型为text。

  返回值类型：XML

  示例：

```
gaussdb=# SELECT XMLPARSE (DOCUMENT '<?xml version="1.0"?><book><title>Manual</
title><chapter>...</chapter></book>');
                    xmlparse
-------------------------------------------------------------
 <book><title>Manual</title><chapter>...</chapter></book>
(1 row)
gaussdb=# SELECT XMLPARSE (CONTENT 'abc<foo>bar</foo><bar>foo</bar>');
         xmlparse
---------------------------------
 abc<foo>bar</foo><bar>foo</bar>
(1 row)
gaussdb=# SELECT XMLPARSE (CONTENT 'abc<foo>bar</foo' wellformed);
    xmlparse
------------------
 abc<foo>bar</foo
(1 row)
```

- xmlserialize({ DOCUMENT | CONTENT } value AS type)

  描述：使用函数xmlserialize，从XML产生一个字符串。

  参数：类型可以是character，character varying或text（或其中某个的变种）。

  返回值类型：XML

  示例：

```
gaussdb=# SELECT XMLSERIALIZE(CONTENT 'good' AS CHAR(10));
 xmlserialize
--------------
 good
(1 row)
gaussdb=# SELECT xmlserialize(DOCUMENT '<head>bad</head>' as text);
  xmlserialize
------------------
 <head>bad</head>
(1 row)
```

  说明

  当一个字符串值在没有通过XMLPARSE或XMLSERIALIZE的情况下，与xml类型进行转换时，具体选择DOCUMENT或CONTENT由“XML option” 会话配置参数决定，这个配置参数可以由标准命令来设置：

```
SET XML OPTION { DOCUMENT | CONTENT };
```

  或使用类似的语法来设置：

```
SET xmloption TO { DOCUMENT | CONTENT };
```

- xmlcomment(text)

  描述：创建一个XML值，并且它包含一个用指定文本作为内容的XML注释。该文本不包含“--”字符且结尾不存在“-”字符、符合XML注释的格式要求。且当参数为空时、结果也为空。

  参数：数据类型为text。

返回值类型：XML

示例：

```
gaussdb=# SELECT xmlcomment('hello');
  xmlcomment
--------------
 <!--hello-->
```

- xmlconcat(xml[, ...])

  描述：将由单个XML值组成的列表串接成一个单独的值，该值包含一个XML的内容片段。其中空值会被忽略，并且只有当所有参数都为空时结果才为空。在兼容A数据库模式下，设置a_format_version值为10c和a_format_dev_version值为s2，增加了校验输入片段是否为非良构xml文本。

  参数：数据类型为XML。

  返回值类型：XML

  示例1：

```
gaussdb=# set xmloption=content;
SET
gaussdb=# select XMLCONCAT(('<?xml version="1.0" encoding="GB2312" standalone="no"?
><bar>foo</bar>'),('<?xml version="1.0" encoding="GB2312" standalone="no" ?><bar>foo</bar>')) ;
                          xmlconcat
----------------------------------------------------------------------
<?xml version="1.0" standalone="no"?><bar>foo</bar><bar>foo</bar>
(1 row)
gaussdb=# select XMLCONCAT('abc>');
 xmlconcat
-----------
 abc>
(1 row)
```

  示例2：兼容A数据库的语法示例。

```
gaussdb=# set a_format_version='10c';
SET
gaussdb=# set a_format_dev_version=s2;
SET
gaussdb=# set xmloption=content;
SET
gaussdb=# select XMLCONCAT(('<?xml version="1.0" encoding="GB2312" standalone="no"?
><bar>foo</bar>'),('<?xml version="1.0" encoding="GB2312" standalone="no" ?><bar>foo</bar>')) ;
                          xmlconcat
----------------------------------------------------------------------
<?xml version="1.0" standalone="no"?><bar>foo</bar><bar>foo</bar>
(1 row)
gaussdb=# select XMLCONCAT('abc>');
ERROR:  invalid XML document
DETAIL:  line 1: Start tag expected, '<' not found
abc>
^
CONTEXT:  referenced column: xmlconcat
```

- xmlelement([ENTITYESCAPING | NOENTITYESCAPING] { [NAME] element_name | EVALNAME element_name } [, xmlattributes([ENTITYESCAPING | NOENTITYESCAPING] value [[AS] attname | AS EVALNAME attname] [, ...])] [, content [[AS] alias] [, ...]])

  描述：使用给定的名称、属性和内容产生一个XML元素。

  返回值类型：XML

  示例：

```
gaussdb=# SELECT xmlelement(name foo);
 xmlelement
------------
 <foo/>
```



```
在A兼容模式下：
gaussdb=# set a_format_version='10c';
SET
gaussdb=# set a_format_dev_version=s2;
SET
1.xmlelement中默认不设置或者设置ENTITYESCAPING关键字时，xmlelement的内容中的保留字符将被转义：
gaussdb=# SELECT xmlelement("entityescaping<>", 'a$><&"b');
                        xmlelement
-----------------------------------------------------------
 <entityescaping<>>a$&gt;&lt;&amp;&quot;b</entityescaping<>>
(1 row)

gaussdb=# SELECT xmlelement(entityescaping "entityescaping<>", 'a$><&"b');
                        xmlelement
-----------------------------------------------------------
 <entityescaping<>>a$&gt;&lt;&amp;&quot;b</entityescaping<>>
(1 row)

2.xmlelement中设置NOENTITYESCAPING关键字时，xmlelement的内容中的保留字符将不会被转义：
gaussdb=# SELECT xmlelement(noentityescaping "entityescaping<>", 'a$><&"b');
                 xmlelement
--------------------------------------------
 <entityescaping<>>a$><&"b</entityescaping<>>
(1 row)

3.xmlelement中对内容使用[as] alias声明别名时，内容值类型必须为xml类型：
gaussdb=# SELECT xmlelement("entityescaping<>", '<abc/>' b);
ERROR:  argument of XMLELEMENT must be type xml, not type unknown
LINE 1: SELECT xmlelement("entityescaping<>", '<abc/>' b);
                                              ^
CONTEXT:  referenced column: xmlelement

gaussdb=# SELECT xmlelement("entityescaping<>", '<abc/>' as b);
ERROR:  argument of XMLELEMENT must be type xml, not type unknown
LINE 1: SELECT xmlelement("entityescaping<>", '<abc/>' as b);
                                              ^
CONTEXT:  referenced column: xmlelement

gaussdb=# SELECT xmlelement("entityescaping<>", xml('<abc/>') b);
                xmlelement
-------------------------------------------
 <entityescaping<>><abc/></entityescaping<>>
(1 row)

gaussdb=# SELECT xmlelement("entityescaping<>", xml('<abc/>') as b);
                xmlelement
-------------------------------------------
 <entityescaping<>><abc/></entityescaping<>>
(1 row)

4.xmlattributes中默认不设置或者设置ENTITYESCAPING关键字时，xmlattributes的内容中的保留字符将被转义：
gaussdb=# SELECT xmlelement("entityescaping<>", xmlattributes('entityescaping<>'
"entityescaping<>"));
                         xmlelement
------------------------------------------------------------
 <entityescaping<> entityescaping<>="entityescaping&lt;&gt;"/>
(1 row)

gaussdb=# SELECT xmlelement(name "entityescaping<>", xmlattributes(entityescaping
'entityescaping<>' "entityescaping<>"));
                         xmlelement
------------------------------------------------------------
 <entityescaping<> entityescaping<>="entityescaping&lt;&gt;"/>
(1 row)

5.xmlattributes中设置NOENTITYESCAPING关键字时，xmlattributes的内容中的保留字符将不会被转义：
gaussdb=# SELECT xmlelement("entityescaping<>", xmlattributes(noentityescaping 'entityescaping<>'
```



```
"entityescaping<>"));
                    xmlelement
-----------------------------------------------------------
 <entityescaping<> entityescaping<>="entityescaping<>"/>
(1 row)
```

说明

1. xmlelement和xmlattributes的name字段赋NULL时，行为与A数据库不一致。
   xmlelement的name字段赋NULL时，结果显示name信息为空，且不显示属性信息。
   xmlattributes的name字段赋NULL时，不显示属性信息。
2. 设置如下两个参数后，xmlelement的内容转义规则为A兼容，未设置时xmlelement的内容转义规则为PG兼容。

   ```
   set a_format_version='10c';
   set a_format_dev_version=s2;
   ```

- xmlforest(content [AS name] [, ...])

  描述：使用给定名称和内容产生一个元素的XML序列。

  返回值类型：XML

  示例：

  ```
  gaussdb=# SELECT xmlforest('abc' AS foo, 123 AS bar);
         xmlforest
  ------------------------------
   <foo>abc</foo><bar>123</bar>
  ```

- xmlpi(name target [, content])

  描述：创建一个XML处理指令。若内容不为空，则内容不能包含字符序列。

  返回值类型：XML

  示例：

  ```
  gaussdb=# SELECT xmlpi(name php, 'echo "hello world";');
           xmlpi
  -----------------------------
   <?php echo "hello world";?>
  ```

- xmlroot(xml, version text | no value [, standalone yes|no|no value])

  描述：修改一个XML值的根节点的属性。如果指定了一个版本，它会替换根节点的版本声明中的值，如果指定了一个独立设置，它会替换根节点的独立声明中的值。

  示例：

  ```
  gaussdb=# SELECT xmlroot('<?xml version="1.1"?><content>abc</content>',version '1.0', standalone
  yes);
                          xmlroot
  ----------------------------------------------------------------
   <?xml version="1.0" standalone="yes"?><content>abc</content>
  (1 row)
  ```

- xmlagg(xml [order_by_clause])

  描述：该函数是一个聚集函数、它将聚集函数调用的输入值串接起来，且支持跨行串接，order_by_clause详见**SELECT**。在兼容A数据库模式下，设置a_format_version值为10c和a_format_dev_version值为s2，数据库xmloption参数默认为content，当xmloption设置为document时，使用换行符串接多行xml。若xml声明中encoding属性值不为默认编码UTF-8时，聚集结果有xml声明。

  参数：XML

  返回值类型：XML

  示例1：

  ```
  gaussdb=# CREATE TABLE xmltest (
            id int,
            data xml
           );
  ```



```
gaussdb=# INSERT INTO xmltest VALUES (1, '<value>one</value>');
INSERT 0 1
gaussdb=# INSERT INTO xmltest VALUES (2, '<value>two</value>');
INSERT 0 1
gaussdb=# SELECT xmlagg(data) FROM xmltest;
              xmlagg
--------------------------------------
 <value>one</value><value>two</value>
(1 row)
```

示例2：兼容A数据库的语法示例。

```
gaussdb=# set xmloption=document;
SET
gaussdb=# SELECT xmlagg(data) FROM xmltest;
      xmlagg
--------------------
 <value>one</value>+
 <value>two</value>
(1 row)
gaussdb=# DELETE FROM XMLTEST;
DELETE 2
gaussdb=# INSERT INTO xmltest VALUES (1, '<?xml version="1.0" encoding="GBK"?><value>one</
value>');
INSERT 0 1
gaussdb=# INSERT INTO xmltest VALUES (2, '<?xml version="1.0" encoding="GBK"?><value>two</
value>');
INSERT 0 1
gaussdb=# SELECT xmlagg(data) FROM xmltest;
                      xmlagg
------------------------------------------------------------
 <?xml version="1.0" encoding="GBK"?><value>one</value>+
 <value>two</value>
(1 row)
gaussdb=# SELECT xmlagg(data order by id desc) FROM xmltest;
                      xmlagg
------------------------------------------------------------
 <?xml version="1.0" encoding="GBK"?><value>two</value>+
 <value>one</value>
(1 row)

gaussdb=# DROP TABLE xmltest;
```

- xmlexists(text passing [BY REF] xml [BY REF])

  描述：评价一个XPath 1.0表达式(第一个参数)，以传递的XML值作为其上下文项。 如果评价的结果产生一个空节点集，该函数返回false，如果产生任何其他值，则返回true。 如果任何参数为空，则函数返回null。 作为上下文项传递的非空值必须是一个XML文档，而不是内容片段或任何非XML值。

  参数：XML

  返回值类型：BOOLEAN

  示例：

  ```
  gaussdb=# SELECT xmlexists('//town[text() = ''Toronto'']' PASSING BY REF '<towns><town>Toronto</
  town><town>Ottawa</town></towns>');
   xmlexists
  ------------
   t
  (1 row)
  ```

- xml_is_well_formed(text)

  描述：检查text是不是正确的XML类型格式、返回值为布尔类型。

  参数：text

  返回值类型：BOOLEAN

  示例：

  ```
  gaussdb=# SELECT xml_is_well_formed('<>');
   xml_is_well_formed
  ```



```
--------------------
 f
(1 row)
```

- xml_is_well_formed_document(text)

  描述：检查text是不是正确的XML类型格式、返回值为布尔类型。

  参数：text

  返回值类型：BOOLEAN

  示例：

```
gaussdb=# SELECT xml_is_well_formed_document('<pg:foo xmlns:pg="http://postgresql.org/
stuff">bar</pg:foo>');
 xml_is_well_formed_document
-----------------------------
 t
(1 row)
```

- xml_is_well_formed_content(text)

  描述：检查text是不是正确的XML类型格式、返回值为布尔类型。

  参数：text

  返回值类型：BOOLEAN

  示例：

```
gaussdb=# select xml_is_well_formed_content('k');
 xml_is_well_formed_content
----------------------------
 t
(1 row)
```

- xpath(xpath, xml [, nsarray])

  描述：在XML类型的数据xml上计算XPath 1.0表达式如：xpath (a text value)。它返回一个XML值的数组，该数组对应于该XPath表达式产生的节点集合。如果该XPath表达式返回一个标量值而不是一个节点集合，将会返回一个单一元素的数组。

  第二个参数必须是一个良构的XML文档。注意，它必须有一个单一根节点元素。

  该函数可选的第三个参数是一个名字空间映射的数组。这个数组应该是一个二维text数组，其第二轴长度等于2（即它应该是一个数组的数组，其中每一个都刚好由2个元素组成）。每个数组项的第一个元素是名字空间的名称（别名），第二个元素是名字空间的URI。并不要求在这个数组中提供的别名和在XML文档本身中使用的那些名字空间相同（换句话说，在XML文档中和在xpath函数环境中，别名都是本地的）。

  返回值类型：XML

  示例：

```
gaussdb=# SELECT xpath('/my:a/text()', '<my:a xmlns:my="http://example.com">test</
my:a>',ARRAY[ARRAY['my', 'http://example.com']]);
 xpath
--------
 {test}
(1 row)
```

- xpath_exists(xpath, xml [, nsarray])

  描述：该函数是xpath函数的一种特殊形式。这个函数不是返回满足XPath 1.0表达式的单一XML值，它返回一个布尔值表示查询是否被满足(具体来说，它是否产生了空节点集以外的任何值)。这个函数等价于标准的XMLEXISTS谓词，不过它还提供了对一个名字空间映射参数的支持。

  返回值类型：BOOLEAN

  示例：


```
gaussdb=# SELECT xpath_exists('/my:a/text()', '<my:a xmlns:my="http://example.com">test</
my:a>',ARRAY[ARRAY['my', 'http://example.com']]);
 xpath_exists
--------------
 t
(1 row)
```

说明

以下XML类型函数示例中，需进行前置数据准备，如下所示：

```
gaussdb=# CREATE SCHEMA testxmlschema;
CREATE SCHEMA
gaussdb=# CREATE TABLE testxmlschema.test1 (a int, b text);
CREATE TABLE
gaussdb=# INSERT INTO testxmlschema.test1 VALUES (1, 'one'), (2, 'two'), (-1, null);
INSERT 0 3
gaussdb=# CREATE DATABASE test;
CREATE DATABASE

--示例执行结束后，可使用如下命令删除上述前置数据：
gaussdb=# DROP DATABASE test;
DROP DATABASE
gaussdb=# DROP TABLE testxmlschema.test1;
DROP TABLE
gaussdb=# DROP SCHEMA testxmlschema;
DROP SCHEMA
```

- query_to_xml(query text, nulls boolean, tableforest boolean, targetns text)

  描述：该函数会将query查询的内容映射成XML模式文档。

  返回值类型：XML

  示例：

```
gaussdb=# SELECT query_to_xml('SELECT * FROM testxmlschema.test1', false, false, '');
                        query_to_xml
-------------------------------------------------------------
 <table xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">+
                                                             +
 <row>                                                       +
   <a>1</a>                                                  +
   <b>one</b>                                                +
 </row>                                                      +
                                                             +
 <row>                                                       +
   <a>2</a>                                                  +
   <b>two</b>                                                +
 </row>                                                      +
                                                             +
 <row>                                                       +
   <a>-1</a>                                                 +
 </row>                                                      +
                                                             +
 </table>                                                    +
(1 row)
```

- query_to_xmlschema(query text, nulls boolean, tableforest boolean, targetns text)

  描述：该函数会将query查询的内容映射成XML文档和XML模式文档，并把两个文档连接在一起。

  返回值类型：XML

  示例：

```
gaussdb=# SELECT query_to_xmlschema('SELECT * FROM testxmlschema.test1', false, false, '');
                                query_to_xmlschema
-----------------------------------------------------------------------------------------------
 <xsd:schema                                                                          +
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">                                     +
                                                                                      +
```



```
<xsd:simpleType name="INTEGER">                                                 +
  <xsd:restriction base="xsd:int">                                              +
    <xsd:maxInclusive value="2147483647"/>                                      +
    <xsd:minInclusive value="-2147483648"/>                                     +
  </xsd:restriction>                                                            +
</xsd:simpleType>                                                               +
                                                                                +
<xsd:simpleType name="UDT.regression.pg_catalog.text">                          +
  <xsd:restriction base="xsd:string">                                           +
  </xsd:restriction>                                                            +
</xsd:simpleType>                                                               +
                                                                                +
<xsd:complexType name="RowType">                                                +
  <xsd:sequence>                                                                +
    <xsd:element name="a" type="INTEGER" minOccurs="0"></xsd:element>           +
    <xsd:element name="b" type="UDT.regression.pg_catalog.text" minOccurs="0"></xsd:element>+
  </xsd:sequence>                                                               +
</xsd:complexType>                                                              +
                                                                                +
<xsd:complexType name="TableType">                                              +
  <xsd:sequence>                                                                +
    <xsd:element name="row" type="RowType" minOccurs="0" maxOccurs="unbounded"/> +
  </xsd:sequence>                                                               +
</xsd:complexType>                                                              +
                                                                                +
<xsd:element name="table" type="TableType"/>                                    +
                                                                                +
</xsd:schema>
(1 row)
```

- query_to_xml_and_xmlschema(query text, nulls boolean, tableforest boolean, targetns text)

  描述：该函数会将query查询的内容映射成XML文档和XML模式文档，并把两个文档连接在一起。

  返回值类型：XML

  示例：

```
gaussdb=# SELECT query_to_xml_and_xmlschema('SELECT * FROM testxmlschema.test1', true, true, '');
                                query_to_xml_and_xmlschema
-------------------------------------------------------------------------------------------
<xsd:schema                                                                     +
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">                               +
                                                                                +
<xsd:simpleType name="INTEGER">                                                 +
  <xsd:restriction base="xsd:int">                                              +
    <xsd:maxInclusive value="2147483647"/>                                      +
    <xsd:minInclusive value="-2147483648"/>                                     +
  </xsd:restriction>                                                            +
</xsd:simpleType>                                                               +
                                                                                +
<xsd:simpleType name="UDT.regression.pg_catalog.text">                          +
  <xsd:restriction base="xsd:string">                                           +
  </xsd:restriction>                                                            +
</xsd:simpleType>                                                               +
                                                                                +
<xsd:complexType name="RowType">                                                +
  <xsd:sequence>                                                                +
    <xsd:element name="a" type="INTEGER" nillable="true"></xsd:element>         +
    <xsd:element name="b" type="UDT.regression.pg_catalog.text" nillable="true"></xsd:element>+
  </xsd:sequence>                                                               +
</xsd:complexType>                                                              +
                                                                                +
<xsd:element name="row" type="RowType"/>                                        +
                                                                                +
</xsd:schema>                                                                   +
                                                                                +
<row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">                     +
                                                                                +
```



```
  <a>1</a>                                                                +
  <b>one</b>                                                               +
</row>                                                                    +
                                                                         +
<row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">                                +
                                                                         +
  <a>2</a>                                                                +
  <b>two</b>                                                               +
</row>                                                                    +
                                                                         +
<row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">                                +
                                                                         +
  <a>-1</a>                                                               +
  <b xsi:nil="true"/>                                                      +
</row>                                                                    +
                                                                         +
(1 row)
```

- cursor_to_xml(cursor refcursor, count int, nulls boolean,tableforest boolean, targetns text)

  描述：该函数会将游标查询的内容映射成XML文档。

  返回值类型：XML

  示例：

```
gaussdb=#CURSOR xc WITH HOLD FOR SELECT * FROM testxmlschema.test1 ORDER BY 1, 2;
DECLARE CURSOR
gaussdb=# SELECT cursor_to_xml('xc'::refcursor, 5, false, true, '');
                    cursor_to_xml
-------------------------------------------------------------
 <row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">+
                                                          +
  <a>-1</a>                                                +
 </row>                                                   +
                                                          +
 <row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">+
                                                          +
  <a>1</a>                                                 +
  <b>one</b>                                               +
 </row>                                                   +
                                                          +
 <row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">+
                                                          +
  <a>2</a>                                                 +
  <b>two</b>                                               +
 </row>                                                   +
                                                          +

(1 row)
```

- cursor_to_xmlschema(cursor refcursor, nulls boolean, tableforest boolean, targetns text)

  描述：该函数会将游标查询的内容映射成XML模式文档。

  返回值类型：XML

  示例：

```
gaussdb=# SELECT cursor_to_xmlschema('xc'::refcursor, true, false, '');
                                cursor_to_xmlschema
-----------------------------------------------------------------------------------------------
 <xsd:schema                                                                  +
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">                                      +
                                                                            +
 <xsd:simpleType name="INTEGER">                                                  +
  <xsd:restriction base="xsd:int">                                                 +
    <xsd:maxInclusive value="2147483647"/>                                         +
    <xsd:minInclusive value="-2147483648"/>                                        +
  </xsd:restriction>                                                           +
 </xsd:simpleType>                                                              +
```



```
                                                                +
<xsd:simpleType name="UDT.regression.pg_catalog.text">                                +
  <xsd:restriction base="xsd:string">                                   +
  </xsd:restriction>                                                  +
</xsd:simpleType>                                                       +
                                                                +
<xsd:complexType name="RowType">                                              +
  <xsd:sequence>                                                    +
    <xsd:element name="a" type="INTEGER" nillable="true"></xsd:element>                 +
    <xsd:element name="b" type="UDT.regression.pg_catalog.text" nillable="true"></xsd:element>+
  </xsd:sequence>                                                   +
</xsd:complexType>                                                      +
                                                                +
<xsd:complexType name="TableType">                                            +
  <xsd:sequence>                                                    +
    <xsd:element name="row" type="RowType" minOccurs="0" maxOccurs="unbounded"/>          +
  </xsd:sequence>                                                   +
</xsd:complexType>                                                      +
                                                                +
<xsd:element name="table" type="TableType"/>                                    +
                                                                +
</xsd:schema>
(1 row)
```

- schema_to_xml(schema name, nulls boolean, tableforest boolean, targetns text)

  描述：该函数会将整个模式的内容映射成XML文档。

  返回值类型：XML

  示例：

```
gaussdb=# SELECT schema_to_xml('testxmlschema', false, true, '');
                        schema_to_xml
-------------------------------------------------------------------
 <testxmlschema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">+
                                                        +
 <test1>                                                  +
                                                        +
   <a>1</a>                                               +
   <b>one</b>                                               +
 </test1>                                                 +
                                                        +
 <test1>                                                  +
                                                        +
   <a>2</a>                                               +
   <b>two</b>                                               +
 </test1>                                                 +
                                                        +
 <test1>                                                  +
                                                        +
   <a>-1</a>                                              +
 </test1>                                                 +
                                                        +
                                                        +
 </testxmlschema>                                               +
(1 row)
```

- schema_to_xmlschema(schema name, nulls boolean, tableforest boolean, targetns text)

  描述：该函数会将整个模式的内容映射成XML模式文档。

  返回值类型：XML

  示例：

```
gaussdb=# SELECT schema_to_xmlschema('testxmlschema', false, true, '');
                                    schema_to_xmlschema
-----------------------------------------------------------------------------------------------------
 <xsd:schema                                                                        +
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">                                           +
```



```
                                                                +
<xsd:simpleType name="INTEGER">                                   +
  <xsd:restriction base="xsd:int">                                +
    <xsd:maxInclusive value="2147483647"/>                          +
    <xsd:minInclusive value="-2147483648"/>                         +
  </xsd:restriction>                                              +
</xsd:simpleType>                                                 +
                                                                +
<xsd:simpleType name="UDT.t1.pg_catalog.text">                    +
  <xsd:restriction base="xsd:string">                             +
  </xsd:restriction>                                              +
</xsd:simpleType>                                                 +
                                                                +
<xsd:complexType name="SchemaType.t1.testxmlschema">                                +
  <xsd:sequence>                                                  +
    <xsd:element name="test1" type="RowType.t1.testxmlschema.test1" minOccurs="0"
maxOccurs="unbounded"/>+
  </xsd:sequence>                                                 +
</xsd:complexType>                                                +
                                                                +
<xsd:element name="testxmlschema" type="SchemaType.t1.testxmlschema"/>
+
                                                                +
</xsd:schema>
(1 row)
```

- schema_to_xml_and_xmlschema(schema name, nulls boolean, tableforest boolean, targetns text)

  描述：该函数会将整个模式的内容映射成XML文档和XML模式文档，并把两个文档连接在一起。

  返回值类型：XML

  示例：

```
 gaussdb=# SELECT schema_to_xml_and_xmlschema('testxmlschema', true, true, 'foo');
                                schema_to_xml_and_xmlschema
-----------------------------------------------------------------------------------------------------------------
 <testxmlschema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="foo"
xsi:schemaLocation="foo #">+
                                                                +
                                                              +
 <xsd:schema                                                    +
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"                                +
    targetNamespace="foo"                                       +
    elementFormDefault="qualified">                               +
                                                                +
 <xsd:simpleType name="INTEGER">                                  +
  <xsd:restriction base="xsd:int">                              +
    <xsd:maxInclusive value="2147483647"/>                        +
    <xsd:minInclusive value="-2147483648"/>                       +
  </xsd:restriction>                                            +
 </xsd:simpleType>                                              +
                                                                +
 <xsd:simpleType name="UDT.t1.pg_catalog.text">                   +
  <xsd:restriction base="xsd:string">                             +
  </xsd:restriction>                                            +
 </xsd:simpleType>                                              +
                                                                +
 <xsd:complexType name="SchemaType.t1.testxmlschema">                                +
  <xsd:sequence>                                                +
    <xsd:element name="test1" type="RowType.t1.testxmlschema.test1" minOccurs="0"
maxOccurs="unbounded"/>   +
  </xsd:sequence>                                               +
 </xsd:complexType>                                             +
                                                                +
 <xsd:element name="testxmlschema"
type="SchemaType.t1.testxmlschema"/>                          +
                                                                +
 </xsd:schema>                                                  +
```



```
                                                                +
<test1>                                                           +
                                                                +
  <a>1</a>                                                          +
  <b>one</b>                                                          +
</test1>                                                          +
                                                                +
<test1>                                                           +
                                                                +
  <a>2</a>                                                          +
  <b>two</b>                                                          +
</test1>                                                          +
                                                                +
<test1>                                                           +
                                                                +
  <a>-1</a>                                                         +
  <b xsi:nil="true"/>                                                 +
</test1>                                                          +
                                                                +
                                                                +
</testxmlschema>                                                        +

(1 row)
```

- database_to_xml(nulls boolean, tableforest boolean, targetns text)

  描述：该函数会将整个数据库的内容映射成XML文档。

  返回值类型：XML

  示例：

```
gaussdb=# SELECT database_to_xml(true, true, 'test');
                          database_to_xml
-------------------------------------------------------------------------------
 <test xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="test">+
                                                                +
 <dbe_x005F_xml>                                                      +
                                                                +
 </dbe_x005F_xml>                                                     +
                                                                +
 <dbe_x005F_xmldom>                                                     +
                                                                +
 </dbe_x005F_xmldom>                                                    +
                                                                +
 <dbe_x005F_xmlparser>                                                  +
                                                                +
 </dbe_x005F_xmlparser>                                                 +
                                                                +
 <public>                                                             +
                                                                +
 </public>                                                            +
                                                                +
 </test>                                                              +

(1 row)
```

- database_to_xmlschema(nulls boolean, tableforest boolean, targetns text)

  描述：该函数会将整个数据库的内容映射成XML模式文档。

  返回值类型：XML

  示例：

```
gaussdb=# SELECT database_to_xmlschema(true, true, 'test');
                             database_to_xmlschema
-----------------------------------------------------------------------------------------
 <xsd:schema                                                                    +
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"                                    +
    targetNamespace="test"                                                    +
    elementFormDefault="qualified">                                             +
                                                                          +
 <xsd:complexType name="CatalogType.test">                                        +
```



```
  <xsd:all>                                                                +
    <xsd:element name="dbe_x005F_xml" type="SchemaType.test.dbe_x005F_xml"/>          +
    <xsd:element name="dbe_x005F_xmldom" type="SchemaType.test.dbe_x005F_xmldom"/>    +
    <xsd:element name="dbe_x005F_xmlparser" type="SchemaType.test.dbe_x005F_xmlparser"/>+
    <xsd:element name="public" type="SchemaType.test.public"/>                       +
  </xsd:all>                                                               +
</xsd:complexType>                                                               +
                                                                         +
<xsd:element name="test" type="CatalogType.test"/>                                 +
                                                                         +
</xsd:schema>
(1 row)
```

- database_to_xml_and_xmlschema(nulls boolean, tableforest boolean, targetns text)

  描述：该函数会将整个模式的内容映射成XML文档和XML模式文档，并把两个文档连接在一起。

  返回值类型：XML

  示例：

```
gaussdb=# SELECT database_to_xml_and_xmlschema(true, true, 'test');
                                database_to_xml_and_xmlschema
-----------------------------------------------------------------------------------------------------
 <test xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="test"
xsi:schemaLocation="test #">+
                                                                                 +
 <xsd:schema                                                                       +
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"                                     +
    targetNamespace="test"                                                          +
    elementFormDefault="qualified">                                                    +
                                                                                 +
 <xsd:complexType name="CatalogType.test">                                             +
  <xsd:all>                                                                       +
    <xsd:element name="dbe_x005F_xml" type="SchemaType.test.dbe_x005F_xml"/>                    +
    <xsd:element name="dbe_x005F_xmldom"
type="SchemaType.test.dbe_x005F_xmldom"/>                 +
    <xsd:element name="dbe_x005F_xmlparser"
type="SchemaType.test.dbe_x005F_xmlparser"/>           +
    <xsd:element name="public" type="SchemaType.test.public"/>                          +
  </xsd:all>                                                                     +
 </xsd:complexType>                                                                    +
                                                                                 +
 <xsd:element name="test" type="CatalogType.test"/>                                     +
                                                                                 +
 </xsd:schema>                                                                      +
                                                                                 +
 <dbe_x005F_xml>                                                                     +
                                                                                 +
 </dbe_x005F_xml>                                                                    +
                                                                                 +
 <dbe_x005F_xmldom>                                                                    +
                                                                                 +
 </dbe_x005F_xmldom>                                                                   +
                                                                                 +
 <dbe_x005F_xmlparser>                                                                 +
                                                                                 +
 </dbe_x005F_xmlparser>                                                                +
                                                                                 +
 <public>                                                                           +
                                                                                 +
 </public>                                                                          +
                                                                                 +
 </test>                                                                           +

(1 row)
```

- table_to_xml(tbl regclass, nulls boolean, tableforest boolean, targetns text)

  描述：该函数会将关系表的内容映射成XML文档。

返回值类型：XML

示例：

```
gaussdb=# SELECT table_to_xml('testxmlschema.test1', false, false, '');
                      table_to_xml
-------------------------------------------------------------
 <test1 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">+
                                                             +
 <row>                                                       +
   <a>1</a>                                                   +
   <b>one</b>                                                  +
 </row>                                                      +
                                                             +
 <row>                                                       +
   <a>2</a>                                                   +
   <b>two</b>                                                  +
 </row>                                                      +
                                                             +
 <row>                                                       +
   <a>-1</a>                                                  +
 </row>                                                      +
                                                             +
 </test1>                                                    +

(1 row)
```

- table_to_xmlschema(tbl regclass, nulls boolean, tableforest boolean, targetns text)

  描述：该函数会将关系表的内容映射成XML模式文档。

  返回值类型：XML

  示例：

```
gaussdb=# SELECT table_to_xmlschema('testxmlschema.test1', false, false, '');
                                                                                                                  table_to_xmlschema
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
 <xsd:schema                                                                                                                                                  +
     xmlns:xsd="http://www.w3.org/2001/XMLSchema">                                                                                                                +
                                                                                                                                                                  +
 <xsd:simpleType name="INTEGER">                                                                                                                                  +
   <xsd:restriction base="xsd:int">                                                                                                                               +
     <xsd:maxInclusive value="2147483647"/>                                                                                                                        +
     <xsd:minInclusive value="-2147483648"/>                                                                                                                       +
   </xsd:restriction>                                                                                                                                            +
 </xsd:simpleType>                                                                                                                                               +
                                                                                                                                                                  +
 <xsd:simpleType name="UDT.regression.pg_catalog.text">                                                                                                           +
   <xsd:restriction base="xsd:string">                                                                                                                            +
   </xsd:restriction>                                                                                                                                            +
 </xsd:simpleType>                                                                                                                                               +
                                                                                                                                                                  +
 <xsd:complexType name="RowType.regression.testxmlschema.test1">
+
   <xsd:sequence>                                                                                                                                                +
     <xsd:element name="a" type="INTEGER" minOccurs="0"></
xsd:element>                                                                                                                                                     +
     <xsd:element name="b" type="UDT.regression.pg_catalog.text" minOccurs="0"></
xsd:element>                                                                                                                                                     +
   </xsd:sequence>                                                                                                                                               +
 </xsd:complexType>                                                                                                                                              +
                                                                                                                                                                  +
 <xsd:complexType name="TableType.regression.testxmlschema.test1">
+
   <xsd:sequence>                                                                                                                                                +
     <xsd:element name="row" type="RowType.regression.testxmlschema.test1" minOccurs="0"
maxOccurs="unbounded"/>+
   </xsd:sequence>                                                                                                                                               +
 </xsd:complexType>                                                                                                                                              +
                                                                                                                                                                  +
 <xsd:element name="test1"
```



```
type="TableType.regression.testxmlschema.test1"/>                                +
                                                                                  +
 </xsd:schema>
(1 row)
```

- table_to_xml_and_xmlschema(tbl regclass, nulls boolean, tableforest boolean, targetns text)

  描述：该函数会将关系表的内容映射成XML文档和XML模式文档，并把两个文档连接在一起。

  返回值类型：XML

  示例：

```
gaussdb=# SELECT table_to_xml_and_xmlschema('testxmlschema.test1', false, false, '');
                                    table_to_xml_and_xmlschema
-----------------------------------------------------------------------------------------------------------
 <test1 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="#">                +
                                                                                  +
 <xsd:schema                                                                      +
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">                                 +
                                                                                  +
 <xsd:simpleType name="INTEGER">                                                  +
  <xsd:restriction base="xsd:int">                                                +
    <xsd:maxInclusive value="2147483647"/>                                        +
    <xsd:minInclusive value="-2147483648"/>                                       +
  </xsd:restriction>                                                              +
 </xsd:simpleType>                                                                +
                                                                                  +
 <xsd:simpleType name="UDT.regression.pg_catalog.text">                           +
  <xsd:restriction base="xsd:string">                                             +
  </xsd:restriction>                                                              +
 </xsd:simpleType>                                                                +
                                                                                  +
 <xsd:complexType name="RowType.regression.testxmlschema.test1">
+
  <xsd:sequence>                                                                  +
    <xsd:element name="a" type="INTEGER" minOccurs="0"></
xsd:element>                                      +
    <xsd:element name="b" type="UDT.regression.pg_catalog.text" minOccurs="0"></
xsd:element>                +
  </xsd:sequence>                                                                 +
 </xsd:complexType>                                                               +
                                                                                  +
 <xsd:complexType name="TableType.regression.testxmlschema.test1">
+
  <xsd:sequence>                                                                  +
    <xsd:element name="row" type="RowType.regression.testxmlschema.test1" minOccurs="0"
maxOccurs="unbounded"/>+
  </xsd:sequence>                                                                 +
 </xsd:complexType>                                                               +
                                                                                  +
 <xsd:element name="test1"
type="TableType.regression.testxmlschema.test1"/>                                +
                                                                                  +
 </xsd:schema>                                                                    +
                                                                                  +
 <row>                                                                            +
  <a>1</a>                                                                        +
  <b>one</b>                                                                      +
 </row>                                                                           +
                                                                                  +
 <row>                                                                            +
  <a>2</a>                                                                        +
  <b>two</b>                                                                      +
 </row>                                                                           +
                                                                                  +
 <row>                                                                            +
  <a>-1</a>                                                                       +
```



```
</row>                                                                  +
                                                                      +
</test1>                                                                +

(1 row)
```

说明

- xpath相关函数仅支持xpath() 和xpath_exists()，由于其使用xpath语言查询XML文档，而这些函数都依赖于libxml2 库，且这个库仅在Xpath1.0提供，所以对XPath的限制为1.0。
- 不支持xquery、xml extension、xslt功能。

7.5.39 XMLTYPE 类型函数

- createxml(varchar2 [,varchar2 ,numeric ,numeric])

  描述：varchar2 类型静态方法创建xmltype类型。

  参数：第一个参数要转换成xmltype的字符串（必传字段），第二个参数为用于使输入符合给定架构的可选架构URL（可选字段，默认为空，暂不生效），第三个参数为指示实例根据给定的XML架构有效的标志（可选字段，默认为0，暂不生效），第四个参数为是否为良构的标识（可选字段，默认为0，暂不生效）。

  返回值类型：xmltype

  示例：

```
gaussdb=# SELECT createxml('<a>123</a>');
 createxml
--------------
 <a>123</a>
(1 row)
```


说明

- 与A数据库差异：在PL/SQL中，createxml允许入参为空串，并返回NULL。
- 字符串encoding只支持UTF-8、GBK、LATIN1~LATIN10，version字段只支持1.x。
- createxml支持以xmltype.createxml()语法方式调用。

  示例：

  ```
  gaussdb=# SELECT xmltype.createxml('<a>123</a>');
   createxml
  ------------
   <a>123</a>
  (1 row)
  ```

- 本章内入参为xmltype的函数支持以xmltype().func()的方式调用，会将前一项返回的xmltype类型当作入参传入后一项的函数内，该语法支持多层嵌套（用户自定义函数入参为xmltype不支持该语法）。

  示例：

  ```
  gaussdb=# select xmltype('<a>123<b>456</b></a>').extract('/a/b').getstringval();
   xmltypefunc
  -------------
   <b>456</b>
  (1 row)
  ```

  上述用例实际效果与函数嵌套一致：

  ```
  gaussdb=# select getstringval(extractxml(xmltype('<a>123<b>456</b></a>'),'/a/b'));
   getstringval
  --------------
   <b>456</b>
  (1 row)
  ```

- 存储过程内支持xmltype类型的变量以a.func()方式调用函数，该语法支持一层嵌套。

  示例：

  ```
  gaussdb=# declare
          a xmltype;
          b varchar2;
  begin
          a:=xmltype('<a>123<b>456</b></a>');
          b:=a.getstringval();
          RAISE NOTICE 'xmltype_str is : %',b;
  end;
  /
  NOTICE:  xmltype_str is : <a>123<b>456</b></a>
  ```

- createxml(clob [,varchar2 ,numeric ,numeric])

  描述：clob类型静态方法创建xmltype类型。

  参数：第一个参数要转换成xmltype的clob对象（必传字段），第二个参数为用于使输入符合给定架构的可选架构URL（可选字段，默认为空，暂不生效），第三个参数为实例根据给定的XML架构有效的标志（可选字段，默认为0，暂不生效），第四个参数为是否为良构的标识（可选字段，默认为0，暂不生效）。

  返回值类型：xmltype

  示例：

  ```
  gaussdb=# declare
   xmltype_clob clob;
   xmltype_obj xmltype;
   xmltype_str varchar2(1000);
  begin
   xmltype_clob := '<a>123</a>';
   xmltype_obj := createxml(xmltype_clob);
   xmltype_str := xmltype_obj.getstringval();
   RAISE NOTICE 'xmltype_str is : %',xmltype_str;
  end;
  /
  NOTICE:  xmltype_str is : <a>123</a>
  ```


说明

clob类型参数入参最大支持1GB-1。

- createxml(blob, numeric [,varchar2 ,numeric ,numeric])

  描述：blob类型静态方法创建xmltype类型。

  参数：第一个参数要转换成xmltype的blob对象（必传字段），第二个参数为输入xml数据的字符集id（必传字段），第三个参数为用于使输入符合给定架构的可选架构URL（可选字段，默认为空，暂不生效），第四个参数为实例根据给定的XML架构有效的标志（可选字段，默认为0，暂不生效），第五个参数为是否为良构的标识（可选字段，默认为0，暂不生效）。

  返回值类型：xmltype

  示例：

```
gaussdb=# declare
 xmltype_blob blob;
 xmltype_obj xmltype;
 xmltype_str varchar2(1000);
begin
 xmltype_blob := xmltype('<a>123</a>').getblobval(7);
 xmltype_obj := createxml(xmltype_blob,7);
 xmltype_str := xmltype_obj.getstringval();
 RAISE NOTICE 'xmltype_str is : %',xmltype_str;
end;
/
NOTICE:  xmltype_str is : <?xml version="1.0" encoding="UTF8"?>
<a>123</a>
```

  说明

  - blob类型参数入参最大支持256MB-1。
  - 字符集id取值范围为1~41。

- getblobval(xmltype, numeric)

  描述：将xmltype类型转化成blob类型，支持xmltype().func()方式调用。

  参数：第一个参数为xmltype类型，第二个参数为要转换的目标字符集的字符集id。

  返回值类型：blob

  示例：

```
gaussdb=# SELECT getblobval(xmltype('<asd/>'),7);
                                   getblobval
--------------------------------------------------------------------------------

3C3F786D6C2076657273696F6E3D22312E302220656E636F64696E673D2255544638223F3E0A3C6173
642F3E
(1 row)
```

  xmltype().func()方式：

```
gaussdb=# select xmltype('<asd/>').getblobVal(7);
                                   xmltypefunc
--------------------------------------------------------------------------------

3C3F786D6C2076657273696F6E3D22312E302220656E636F64696E673D2255544638223F3E0A3C6173
642F3E
(1 row)
```

  说明

  入参xmltype长度最大256MB-1。

- getclobval(xmltype)

  描述：将xmltype类型转化成clob类型，支持xmltype().func()方式调用。

参数：入参为xmltype类型。

返回值类型：clob

示例：

```
gaussdb=# SELECT getclobval(xmltype('<a>123</a>'));
 getclobval
--------------
 <a>123</a>
(1 row)
```

xmltype().func()方式：

```
gaussdb=# SELECT xmltype('<a>123</a>').getclobval();
 xmltypefunc
--------------
 <a>123</a>
(1 row)
```

- getnumberval(xmltype)

  描述：将xmltype类型转化成numeric类型，支持xmltype().func()方式调用。

  参数：入参为xmltype类型。

  返回值类型：numeric

  示例：

  ```
  gaussdb=# SELECT getnumberval(xmltype('<a>123</a>').extract('/a/text()'));
   getnumberval
  --------------
           123
  (1 row)
  ```

  xmltype().func()方式：

  ```
  gaussdb=# SELECT xmltype('<a>123</a>').extract('/a/text()').getnumberval();
   xmltypefunc
  --------------
           123
  (1 row)
  ```

- isfragment(xmltype)

  描述：返回该xmltype类型是片段（1）还是文档（0），支持xmltype().func()方式调用。

  参数：入参为xmltype类型。

  返回值类型：numeric

  示例：

  ```
  gaussdb=# SELECT isfragment(xmltype('<a>123</a>'));
   isfragment
  --------------
            0
  (1 row)
  ```

  xmltype().func()方式：

  ```
  gaussdb=# SELECT xmltype('<a>123</a>').isfragment();
   xmltypefunc
  --------------
            0
  (1 row)
  ```

- xmltype(varchar2 [,varchar2 ,numeric ,numeric])

  描述：varchar2 类型创建xmltype类型。

  参数：第一个参数要转换成xmltype的字符串（必传字段），第二个参数为用于使输入符合给定架构的可选架构URL（可选字段，默认为空，暂不生效），第三个参数为指示实例根据给定的XML架构有效的标志（可选字段，默认为0，暂不生效），第四个参数为是否为良构的标识（可选字段，默认为0，暂不生效）。

  返回值类型：xmltype

示例：

```
gaussdb=# SELECT xmltype('<a>123</a>');
 xmltype
--------------
 <a>123</a>
(1 row)
```

说明

- 与A数据库差异：在PL/SQL中，xmltype允许入参为空串，并返回NULL。
- 字符串encoding只支持UTF-8、GBK、LATIN1~LATIN10，version字段只支持1.x。

- xmltype(clob [,varchar2 ,numeric ,numeric])

  描述：clob类型创建xmltype类型。

  参数：第一个参数要转换成xmltype的clob对象（必传字段），第二个参数为用于使输入符合给定架构的可选架构URL（可选字段，默认为空，暂不生效），第三个参数为实例根据给定的XML架构有效的标志（可选字段，默认为0，暂不生效），第四个参数为是否为良构的标识（可选字段，默认为0，暂不生效）。

  返回值类型：xmltype

  示例：

```
gaussdb=# declare
 xmltype_clob clob;
 xmltype_obj xmltype;
 xmltype_str varchar2(1000);
begin
 xmltype_clob := '<a>123</a>';
 xmltype_obj := xmltype(xmltype_clob);
 xmltype_str := xmltype_obj.getstringval();
 RAISE NOTICE 'xmltype_str is : %',xmltype_str;
end;
/
NOTICE:  xmltype_str is : <a>123</a>
```

  说明

  clob类型参数入参最大支持1GB-1。

- xmltype(blob, numeric [,varchar2 ,numeric ,numeric])

  描述：blob类型创建xmltype类型。

  参数：第一个参数要转换成xmltype的blob对象（必传字段），第二个参数为输入xml数据的字符集id，第三个参数为用于使输入符合给定架构的可选架构URL（可选字段，默认为空，暂不生效），第四个参数为实例根据给定的XML架构有效的标志（可选字段，默认为0，暂不生效），第五个参数为是否为良构的标识（可选字段，默认为0，暂不生效）。

  返回值类型：xmltype

  示例：

```
gaussdb=# declare
 xmltype_blob blob;
 xmltype_obj xmltype;
 xmltype_str varchar2(1000);
begin
 xmltype_blob := getblobval(createxml('<a>123</a>'),7);
 xmltype_obj := xmltype(xmltype_blob,7);
 xmltype_str := xmltype_obj.getstringval();
 RAISE NOTICE 'xmltype_str is : %',xmltype_str;
end;
/
NOTICE:  xmltype_str is : <?xml version="1.0" encoding="UTF8"?>
<a>123</a>
```


说明

- blob类型参数入参最大支持256MB-1。
- 字符集id取值范围为1~41。

- getstringval(xmltype)

  描述：此函数将xmltype转化为字符串。

  参数：需要转换的xmltype。

  返回值类型：varchar2

  getstringval函数有两种调用方式。

  示例1：

```
gaussdb=# SELECT getstringval('<a>123<b>456</b></a>');
    getstringval
----------------------
 <a>123<b>456</b></a>
(1 row)
```

  示例2：调用方式兼容ORA的语法。

```
gaussdb=# SELECT xmltype('<a>123<b>456</b></a>').getstringval();
    xmltypefunc
----------------------
 <a>123<b>456</b></a>
(1 row)
```

- getrootelement(xmltype)

  描述：此函数获取xmltype的根元素。

  参数：需要获取根元素的xmltype。

  返回值类型：varchar2

  getrootelement函数有两种调用方式。

  示例1：

```
gaussdb=# SELECT getrootelement('<a>123<b>456</b></a>');
 getrootelement
----------------
 a
(1 row)
```

  示例2：调用方式兼容ORA的语法。

```
gaussdb=# SELECT xmltype('<a>123<b>456</b></a>').getrootelement();
 xmltypefunc
-------------
 a
(1 row)
```

- getnamespace(xmltype)

  描述：此函数获取xmltype顶层元素的命名空间。

  参数：需要获取命名空间的xmltype。

  返回值类型：varchar2

  getnamespace函数有两种调用方式。

  示例1：

```
gaussdb=# SELECT getnamespace('<c:a xmlns:c="asd">123<d:b xmlns:d="qwe">456</d:b></c:a>');
 getnamespace
--------------
 asd
(1 row)
```

  示例2：调用方式兼容ORA的语法。


```
gaussdb=# SELECT xmltype('<c:a xmlns:c="asd">123<d:b xmlns:d="qwe">456</d:b></
c:a>').getnamespace();
 xmltypefunc
-------------
 asd
(1 row)
```

- existsnode(xmltype, varchar2[, varchar2])

  描述：此函数根据xpath表达式判断在xmltype中是否存在该xml节点，如果存在返回1，否则返回0。

  参数：被查询的xmltype，查询的xpath节点路径，xpath路径的命名空间（在入参有命名空间时，xpath和命名空间都需要定义别名，如示例3）。

  返回值类型：numeric

  existsnode函数有两种调用方式。

  示例1：

  ```
  gaussdb=# SELECT existsnode('<a>123<b>456</b></a>','/a/b');
   existsnode
  ------------
            1
  (1 row)
  ```

  示例2：调用方式兼容ORA的语法。

  ```
  gaussdb=# SELECT xmltype('<a>123<b>456</b></a>').existsnode('/a/b');
   xmltypefunc
  -------------
             1
  (1 row)
  ```

  示例3：

  ```
  gaussdb=# SELECT existsnode('<a:b xmlns:a="asd">123<c>456</c></a:b>','/a:b/c','xmlns:a="asd"');
   existsnode
  ------------
            1
  (1 row)
  ```

  示例4：调用方式兼容ORA的语法。

  ```
  gaussdb=# SELECT xmltype('<a:b xmlns:a="asd">123<c>456</c></a:b>').existsnode('/a:b/
  c','xmlns:a="asd"');
   xmltypefunc
  -------------
             1
  (1 row)
  ```

- extractxml(xmltype, varchar2[, varchar2])

  描述：此函数根据xpath表达式判断在xmltype中是否存在该xml节点，如果存在返回包含该节点的xmltype，如果不存在返回NULL。可以将返回值插入xmltype类型的表中。

  参数：被查询的xmltype，查询的xpath节点路径，xpath路径的命名空间（在入参有命名空间时，xpath和命名空间都需要定义别名，如示例3）。

  返回值类型：xmltype

  extractxml函数有两种调用方式。

  示例1：

  ```
  gaussdb=# SELECT extractxml('<a>123<b>456</b></a>','/a/b');
   extractxml
  ------------
   <b>456</b>
  (1 row)
  ```

  示例2：调用方式兼容ORA的语法。


```
gaussdb=# SELECT xmltype('<a>123<b>456</b></a>').extract('/a/b');
 xmltypefunc
-------------
 <b>456</b>
(1 row)

gaussdb=# SELECT xmltype('<a>123<b>456</b></a>').extractxml('/a/b');
 xmltypefunc
-------------
 <b>456</b>
(1 row)
```

示例3：

```
gaussdb=# SELECT extractxml('<a:b xmlns:a="asd">123<c>456</c></a:b>','/a:b','xmlns:a="asd"');
              extractxml
----------------------------------------
 <a:b xmlns:a="asd">123<c>456</c></a:b>
(1 row)
```

示例4：调用方式兼容ORA的语法。

```
gaussdb=# SELECT xmltype('<a:b xmlns:a="asd">123<c>456</c></a:b>').extract('/a:b','xmlns:a="asd"');
             xmltypefunc
----------------------------------------
 <a:b xmlns:a="asd">123<c>456</c></a:b>
(1 row)

gaussdb=# SELECT xmltype('<a:b xmlns:a="asd">123<c>456</c></a:b>').extractxml('/
a:b','xmlns:a="asd"');
             xmltypefunc
----------------------------------------
 <a:b xmlns:a="asd">123<c>456</c></a:b>
(1 row)
```

7.5.40 Global Plsql Cache 特性函数

- invalidate_plsql_object()，invalidate_plsql_object(schema, objname, objtype);

  描述：将Global Plsql Cache全局缓存中的对象失效掉，仅在enable_global_plsqlcache = on时可用。调用该函数的用户需要具有SYSADMIN权限。

  参数：该函数为重载函数。当无入参时，将所有database内的所有全局缓存对象失效。

  当指定(schema, objname, objtype)三个参数时可将当前database内的指定全局缓存对象失效，其中：schema为对象所属的schema名称；objname为对象名称；objtype为对象类型，对象为package类型时值为'package'，对象为函数或存储过程时值为'function'。

  示例:

  该函数不返回失效结果，可通过gs_glc_memory_detail视图查询，对象未被失效时可在视图中查找到对应的valid状态的行，失效后则没有对应的valid状态行（对象为package时，valid状态后会显示缓存对象为包头或者包体）。

  invalidate_plsql_object所属的schema为pg_catalog，但不指定schema也可调用该函数。

```
--如在视图中可查到函数f3的缓存信息，状态为valid。
gaussdb=# SELECT * FROM gs_glc_memory_detail WHERE type='func';
 contextname | database | schema  | type | status  |       location        |  env   | usedsize
-------------+----------+---------+------+---------+-----------------------+--------+----------
 f3          | testdb   | public  | func | valid   | in_global_hash_table  |      0 |    47584
gaussdb=# SELECT * FROM gs_glc_memory_detail WHERE type='pkg';
 contextname | database | schema  | type |      status      |       location        |  env   | usedsize
-------------+----------+---------+------+------------------+-----------------------+--------+----------
 pkg1        | testdb   | public  | pkg  | valid:spec/body | in_global_hash_table  |      0 |   184176
```



```
--调用函数，指定schema、函数名、类型即可将f3失效，再次查询视图，则f3没有对应的valid状态行。
gaussdb=# SELECT invalidate_plsql_object('public','f3','function');
 invalidate_plsql_object
-------------------------
(1 row)
--如要失效package类型，参数如下所示。
gaussdb=# call pg_catalog.invalidate_plsql_object('public','pkg1','package');
 invalidate_plsql_object
-------------------------
(1 row)

--调用时没有入参，则将失效所有缓存对象。
gaussdb=# SELECT invalidate_plsql_object();
 invalidate_plsql_object
-------------------------
(1 row)
```

7.5.41 其他系统函数

其他系统函数包含两类，兼容PostgreSQL的函数和实现内部功能的函数。这些函数不推荐使用，若需使用，请联系华为技术支持工程师。

兼容 PostgreSQL 的函数和操作符

GaussDB的内建函数和操作符兼容PostgreSQL。其中部分函数为系统内部调用函数（如send、recv等）不支持用户使用；其他函数不推荐使用，若需使用，请联系华为技术支持工程师。

_pg_char_max_length	_pg_char_octet_length	_pg_datetime_precision	_pg_expandarray	_pg_index_position	_pg_interval_type	_pg_numeric_precision
_pg_numeric_precision_radix	_pg_numeric_scale	_pg_truetypid	_pg_truetypmod	abbrev	abs	abstime
abstimeeq	abstimege	abstimegt	abstimein	abstimele	abstimelt	abstimene
abstimeout	abstimerecv	abstimesend	aclcontains	acldefault	aclexplode	aclinsert
aclitemeq	aclitemin	aclitemout	aclremove	acos	age	akeys
any_in	any_out	anyarray_in	anyarray_out	anyarray_recv	anyarray_send	anyelement_in
anyelement_out	anyenum_in	anyenum_out	anynonarray_in	anynonarray_out	anyrange_in	anyrange_out
anytextcat	area	areajoinsel	areasel	array_agg	array_agg_finalfn	array_agg_transfn
array_append	array_cat	array_dims	array_eq	array_fill	array_ge	array_gt

array_in	array_larger	array_le	array_length	array_lower	array_lt	array_ndims
array_ne	array_out	array_prepend	array_recv	array_send	array_smaller	array_to_json
array_to_string	array_typanalyze	array_upper	arraycontained	arraycontains	arraycontjoinsel	arraycontsel
arrayoverlap	ascii	asin	atan	atan2	avals	avg
big5_to_euc_tw	big5_to_mic	big5_to_utf8	bit	bit_and	bit_in	bit_length
bit_or	bit_out	bit_recv	bit_send	bitand	bitcat	bitcmp
biteq	bitge	bitgt	bitle	bitlt	bitne	bitnot
bitor	bitshiftleft	bitshiftright	bittypmodin	bittypmodout	bitxor	bool
bool_and	bool_or	booland_statefunc	booleq	boolge	boolgt	boolin
boolle	boollt	boolne	boolor_statefunc	boolout	boolrecv	boolsend
box	box_above	box_above_eq	box_add	box_below	box_below_eq	box_center
box_contain	box_contain_pt	box_contained	box_distance	box_div	box_eq	box_ge
box_gt	box_in	box_intersect	box_le	box_left	box_lt	box_mul
box_out	box_overabove	box_overbelow	box_overlap	box_overleft	box_overright	box_recv
box_right	box_same	box_send	box_sub	bpchar	bpchar_larger	bpchar_pattern_ge
bpchar_pattern_gt	bpchar_pattern_le	bpchar_pattern_lt	bpchar_smaller	bpchar_sortsupport	bpcharcmp	bpchareq
bpcharge	bpchargt	bpcharicliike	bpcharicnlike	bpcharicregexeq	bpcharicregexne	bpcharin
bpcharle	bpcharlike	bpcharlt	bpcharne	bpcharnlike	bpcharout	bpcharrecv

bpcharregexeq	bpcharregexne	bpcharsend	bpchartypmodin	bpchartypmodout	broadcast	btabstimecmp
btarraycmp	btbeginscan	btboolcmp	btbpchar_pattern_cmp	btbuild	btbuildempty	btbulkdelete
btcanreturn	btcharcmp	btcostestimate	btendscan	btfloat48cmp	btfloat4cmp	btfloat4sortsupport
btfloat84cmp	btfloat8cmp	btfloat8sortsupport	btgetbitmap	btgettuple	btinsert	btint24cmp
btint28cmp	btint2cmp	btint2sortsupport	btint42cmp	btint48cmp	btint4cmp	btint4sortsupport
btint82cmp	btint84cmp	btint8cmp	btint8sortsupport	btmarkpos	btnamecmp	btnamesortsupport
btoidcmp	btoidsortsupport	btoidvectorcmp	btoptions	btrecordcmp	btreltimecmp	btrescan
btrestrpos	btrim	bttext_pattern_cmp	bttextcmp	bttextsortsupport	bttidcmp	bttintervalcmp
btvacuumcleanup	bytea_sortsupport	bytea_string_agg_finalfn	bytea_string_agg_transfn	byteacat	byteacmp	byteaeq
byteage	byteagt	byteain	byteale	bytealike	bytealt	byteane
byteanlike	byteaout	bytearecv	byteasend	cash_cmp	cash_div_cash	cash_div_flt4
cash_div_flt8	cash_div_int2	cash_div_int4	cash_div_int8	cash_eq	cash_ge	cash_gt
cash_in	cash_le	cash_lt	cash_mi	cash_mul_flt4	cash_mul_flt8	cash_mul_int2
cash_mul_int4	cash_mul_int8	cash_ne	cash_out	cash_pl	cash_recv	cash_send
cashlarger	cashsmaller	cbrt	ceil	ceiling	center	char

char_length	character_length	chareq	charge	chargt	charin	charle
charlt	charne	charout	charrecv	charsend	chr	cideq
cidin	cidout	cidr	cidr_in	cidr_out	cidr_recv	cidr_send
cidrecv	cidsend	circle	circle_above	circle_add_pt	circle_below	circle_center
circle_contain	circle_contain_pt	circle_contained	circle_distance	circle_div_pt	circle_eq	circle_ge
circle_gt	circle_in	circle_le	circle_left	circle_lt	circle_mul_pt	circle_ne
circle_out	circle_overabove	circle_overbelow	circle_overlap	circle_overleft	circle_overright	circle_recv
circle_right	circle_same	circle_send	circle_sub_pt	clock_timestamp	close_lb	close_ls
close_lseg	close_pb	close_pl	close_ps	close_sb	close_sl	col_description
concat	concat_ws	contjoinsel	contsel	convert	convert_from	convert_to
corr	cos	cot	count	covar_pop	covar_samp	cstring_in
cstring_out	cstring_recv	cstring_send	cume_dist	current_database	current_query	current_schema
-	current_setting	current_user	currtid	currtid2	currval	-
-	-	-	-	date	date_cmp	date_cmp_timestamp
date_cmp_timestamptz	date_eq	date_eq_timestamp	date_eq_timestamptz	date_ge	date_ge_timestamp	date_ge_timestamptz
date_gt	date_gt_timestamp	date_gt_timestamptz	date_in	date_larger	date_le	date_le_timestamp
date_le_timestamptz	date_lt	date_lt_timestamp	date_lt_timestamptz	date_mi	date_mi_interval	date_mii

date_ne	date_ne_timestamp	date_ne_timestamptz	date_out	date_pl_interval	date_pli	date_recv
date_send	date_smaller	date_sortsupport	daterange_canonical	daterange_subdiff	datetime_pl	datetimetz_pl
dcbrt	decode	defined	degrees	delete	dense_rank	dexp
diagonal	diameter	dispell_init	dispell_lexize	dist_cpoly	dist_lb	dist_pb
dist_pc	dist_pl	dist_ppath	dist_ps	dist_sb	dist_sl	div
dlog1	dlog10	domain_in	domain_recv	dpow	dround	dsimple_init
dsimple_lexize	dsnowball_init	dsnowball_lexize	dsqrt	dsynonym_init	dsynonym_lexize	dtrunc
each	enum_ne	enum_out	enum_range	enum_recv	enum_send	enum_smaller
eqjoinsel	eqsel	euc_cn_to_mic	euc_cn_to_utf8	euc_jis_2004_to_shift_jis_2004	euc_jis_2004_to_utf8	euc_jp_to_mic
euc_jp_to_sjis	euc_jp_to_utf8	euc_kr_to_mic	euc_kr_to_utf8	euc_tw_to_big5	euc_tw_to_mic	euc_tw_to_utf8
every	exist	exists_all	exists_any	exp	factorial	family
fdw_handler_in	fdw_handler_out	fetchval	first_value	float4	float4_accum	float48div
float48eq	float48ge	float48gt	float48le	float48lt	float48mi	float48mul
float48ne	float48pl	float4abs	float4div	float4eq	float4ge	float4gt
float4in	float4larger	float4le	float4lt	float4mi	float4mul	float4ne
float4out	float4pl	float4recv	float4send	float4smaller	float4um	float4up
float8	float8_accum	float8_avg	float8_collect	float8_corr	float8_covar_pop	float8_covar_samp

float8_regr_accum	float8_regr_avgx	float8_regr_avgy	float8_regr_collect	float8_regr_intercept	float8_regr_r2	float8_regr_slope
float8_regr_sxx	float8_regr_sxy	float8_regr_syy	float8_stddev_pop	float8_stddev_samp	float8_var_pop	float8_var_samp
float84div	float84eq	float84ge	float84gt	float84le	float84lt	float84mi
float84mul	float84ne	float84pl	float8abs	float8div	float8eq	float8ge
float8gt	float8in	float8larger	float8le	float8lt	float8mi	float8mul
float8ne	float8out	float8pl	float8recv	float8send	float8smaller	float8um
float8up	floor	flt4_mul_cash	flt8_mul_cash	fmgr_c_validator	fmgr_internal_validator	fmgr_sql_validator
format	format_type	gb18030_to_utf8	gbk_to_utf8	generate_series	generate_subscripts	get_bit
get_byte	get_current_ts_config	-	-	-	-	-
gtsquery_compress	gtsquery_consistent	gtsquery_decompress	gtsquery_penalty	gtsquery_picksplit	gtsquery_same	gtsquery_union
gtsvector_compress	gtsvector_consistent	gtsvector_decompress	gtsvector_penalty	gtsvector_picksplit	gtsvector_same	gtsvector_union
gtsvectorin	gtsvectorout	has_tablespace_privilege	has_type_privilege	hash_aclitem	hashbeginscan	hashbuild
hashbuildempty	hashbulkdelete	hashcostestimate	hashendscan	hashgetbitmap	hashgettuple	hashinsert
hashint2vector	hashint4	hashint8	hashmacaddr	hashmarkpos	hashname	hashoid
hashoidvector	hashoptions	hashrescan	hashrestrpos	hashtext	hashvacuumcleanup	hashvarlena
host	hostmask	iclikejoinsel	iclikesel	icnlikejoinsel	icnlikesel	icregexeqjoinsel

icregexeqsel	icregexnejoinsel	icregexnesel	inet_client_addr	inet_client_port	inet_in	inet_out
inet_recv	inet_send	inet_server_addr	inet_server_port	inetand	inetmi	inetmi_int8
inetnot	inetor	inetpl	initcap	int2_accum	int2_avg_accum	int2_mul_cash
int2_sum	int24div	int24eq	int24ge	int24gt	int24le	int24lt
int24mi	int24mul	int24ne	int24pl	int28div	int28eq	int28ge
int28gt	int28le	int28lt	int28mi	int28mul	int28ne	int28pl
int2abs	int2and	int2div	int2eq	int2ge	int2gt	int2in
int2larger	int2le	int2lt	int2mi	int2mod	int2mul	int2ne
int2not	int2or	int2out	int2pl	int2recv	int2send	int2shl
int2shr	int2smaller	int2um	int2up	int2vectoreq	int2vectorin	int2vectorout
int2vectorrecv	int2vectorsend	int2xor	int4_accum	int4_avg_accum	int4_mul_cash	int4_sum
int42div	int42eq	int42ge	int42gt	int42le	int42lt	int42mi
int42mul	int42ne	int42pl	int48div	int48eq	int48ge	int48gt
int48le	int48lt	int48mi	int48mul	int48ne	int48pl	int4abs
int4and	int4div	int4eq	int4ge	int4gt	int4in	int4inc
int4larger	int4le	int4lt	int4mi	int4mod	int4mul	int4ne
int4not	int4or	int4out	int4pl	int4range	int4range_canonical	int4range_subdiff
int4recv	int4send	int4shl	int4shr	int4smaller	int4um	int4up

int4xor	int8	int8_avg	int8_avg_accum	int8_avg_collect	int8_mul_cash	int8_sum
int8_sum_to_int8	int8_accum	int82div	int82eq	int82ge	int82gt	int82le
int82lt	int82mi	int82mul	int82ne	int82pl	int84div	int84eq
int84ge	int84gt	int84le	int84lt	int84mi	int84mul	int84ne
int84pl	int8abs	int8and	int8div	int8eq	int8ge	int8gt
int8in	int8inc	int8inc_any	int8inc_float8_float8	int8larger	int8le	int8lt
int8mi	int8mod	int8mul	int8ne	int8not	int8or	int8out
int8pl	int8pl_inet	int8range	int8range_canonical	int8range_subdiff	int8recv	int8send
int8shl	int8shr	int8smaller	int8um	int8up	int8xor	integer_pl_date
inter_lb	inter_sb	inter_sl	internal_in	internal_out	interval	interval_accum
interval_avg	interval_cmp	interval_collect	interval_div	interval_eq	interval_ge	interval_gt
interval_hash	interval_in	interval_larger	interval_le	interval_lt	interval_mi	interval_mul
interval_ne	interval_out	interval_pl	interval_pl_date	interval_pl_time	interval_pl_timestamp	interval_pl_timestamptz
interval_pl_timetz	interval_recv	interval_send	interval_smaller	interval_transform	interval_um	intervaltypmodin
intervaltypmodout	intinterval	isexists	ishorizontal	iso_to_koi8r	iso_to_mic	iso_to_win1251
iso_to_win866	iso8859_1_to_utf8	iso8859_to_utf8	isparallel	isperp	isvertical	johab_to_utf8
jsonb_in	jsonb_out	jsonb_recv	jsonb_send	-	-	-
json_in	json_out	json_recv	json_send	justify_days	justify_hours	justify_interval

koi8r_to_iso	koi8r_to_mic	koi8r_to_utf8	koi8r_to_win1251	koi8r_to_win866	koi8u_to_utf8	language_handler_in
language_handler_out	latin1_to_mic	latin2_to_mic	latin2_to_win1250	latin3_to_mic	latin4_to_mic	like_escape
likejoinsel	likesel	line	line_distance	line_eq	line_horizontal	line_in
line_interpt	line_intersect	line_out	line_parallel	line_perp	line_recv	line_send
line_vertical	ln	lo_close	lo_creat	lo_create	lo_export	lo_import
lo_lseek	lo_open	lo_tell	lo_truncate	lo_unlink	log	loread
lower	lower_inc	lower_inf	lowrite	lpad	lseg	lseg_center
lseg_distance	lseg_eq	lseg_ge	lseg_gt	lseg_horizontal	lseg_in	lseg_interpt
lseg_intersect	lseg_le	lseg_length	lseg_lt	lseg_ne	lseg_out	lseg_parallel
lseg_perp	lseg_recv	lseg_send	lseg_vertical	ltrim	macaddr_and	macaddr_cmp
macaddr_eq	macaddr_ge	macaddr_gt	macaddr_in	macaddr_le	macaddr_lt	macaddr_ne
macaddr_not	macaddr_or	macaddr_out	macaddr_recv	macaddr_send	makeaclitem	masklen
max	md5（MD5加密算法安全性低，存在安全风险，建议使用更安全的加密算法）	mic_to_big5	mic_to_euc_cn	mic_to_euc_jp	mic_to_euc_kr	mic_to_euc_tw
mic_to_iso	mic_to_koi8r	mic_to_latin1	mic_to_latin2	mic_to_latin3	mic_to_latin4	mic_to_sjis
mic_to_win1250	mic_to_win1251	mic_to_win866	min	mktinterval	money	mul_d_interval
name	nameeq	namege	namegt	nameiclike	nameicnlike	nameicregexeq

nameicregexne	namein	namele	namelike	namelt	namene	namenlike
nameout	namerecv	nameregexeq	nameregexne	namesend	neqjoinsel	neqsel
network_cmp	network_eq	network_ge	network_gt	network_le	network_lt	network_ne
network_sub	network_subeq	network_sup	network_supeq	nlikejoinsel	nlikesel	numeric
numeric_abs	numeric_accum	numeric_add	numeric_avg	numeric_avg_accum	numeric_avg_collect	numeric_cmp
numeric_collect	numeric_div	numeric_div_trunc	numeric_eq	numeric_exp	numeric_fac	numeric_ge
numeric_gt	numeric_in	numeric_inc	numeric_larger	numeric_le	numeric_ln	numeric_log
numeric_lt	numeric_mod	numeric_mul	numeric_ne	numeric_out	numeric_power	numeric_recv
numeric_send	numeric_smaller	numeric_sortsupport	numeric_sqrt	numeric_stddev_pop	numeric_stddev_samp	numeric_sub
numeric_transform	numeric_uminus	numeric_uplus	numeric_var_pop	numeric_var_samp	numerictypmodin	numerictypmodout
numrange_subdiff	oid	oideq	oidge	oidgt	oidin	oidlarger
oidle	oidlt	oidne	oidout	oidrecv	oidsend	oidsmaller
oidvectoreq	oidvectorge	oidvectorgt	oidvectorin	oidvectorle	oidvectorlt	oidvectorne
oidvectorout	oidvectorrecv	oidvectorsend	oidvectortypes	on_pb	on_pl	on_ppath
on_ps	on_sb	on_sl	opaque_in	opaque_out	ordered_set_transition	overlaps

overlay	path	path_add	path_add_pt	path_center	path_contain_pt	path_distance
path_div_pt	path_in	path_interr	path_length	path_mul_pt	path_n_eq	path_n_ge
path_n_gt	path_n_le	path_n_lt	path_npoints	path_out	path_recv	path_send
path_sub_pt	percentile_cont	percentile_cont_float8_final	percentile_cont_interval_final	pg_char_to_encoding	pg_cursor	pg_encoding_max_length
pg_encoding_to_char	pg_extension_config_dump	-	-	pg_node_tree_in	pg_node_tree_out	pg_node_tree_recv
pg_node_tree_send	pg_prepared_statement	pg_prepared_xact	-	-	pg_show_all_settings	pg_stat_get_bgwriter_stat_reset_time
pg_stat_get_buf_fsync_backend	pg_stat_get_checkpoint_sync_time	pg_stat_get_checkpoint_write_time	pg_stat_get_db_blk_read_time	pg_stat_get_db_blk_write_time	pg_stat_get_db_conflict_all	pg_stat_get_db_conflict_bufferpin
pg_stat_get_db_conflict_snapshot	pg_stat_get_db_conflict_startup_deadlock	pg_switch_xlog	-	pg_timezone_abbrevs	pg_timezone_names	pg_stat_get_wal_receiver
plpgsql_call_handler	plpgsql_inline_handler	plpgsql_validator	point_above	point_add	point_below	point_distance
point_div	point_eq	point_horiz	point_in	point_left	point_mul	point_ne
point_out	point_recv	point_right	point_send	point_sub	point_vert	poly_above
poly_below	poly_center	poly_contain	poly_contain_pt	poly_contained	poly_distance	poly_in
poly_left	poly_npoints	poly_out	poly_overabove	poly_overbelow	poly_overlap	poly_overleft
poly_overright	poly_recv	poly_right	poly_same	poly_send	polygon	position

positionjoinsel	positionsel	postgresql_fdw_validator	pow	power	prsd_end	prsd_headline
prsd_lextype	prsd_nexttoken	prsd_start	pt_contained_circle	pt_contained_poly	-	-
-	quote_ident	quote_literal	quote_nullable	radians	radius	random
range_adjacent	range_after	range_before	range_cmp	range_contained_by	range_contains	range_contains_elem
range_eq	range_ge	range_gt	range_in	range_intersect	range_le	range_lt
range_minus	range_ne	range_out	range_overlaps	range_overleft	range_overright	range_recv
range_send	range_typanalyze	range_union	rank	record_eq	record_ge	record_gt
record_in	record_le	record_lt	record_ne	record_out	record_recv	record_send
regclass	regclassin	regclassout	regclassrecv	regclasssend	regconfigin	regconfigout
regconfigrecv	regconfigsend	regdictionaryin	regdictionaryout	regdictionaryrecv	regdictionarysend	regexeqjoinsel
regexeqsel	regexnejoinsel	regexnesel	regexp_matches	regexp_replace	regexp_split_to_array	regexp_split_to_table
regoperatorin	regoperatorout	regoperatorrecv	regoperatorsend	regoperin	regoperout	regoperrecv
regopersend	regprocedurein	regprocedureout	regprocedurerecv	regprocedureresend	regprocin	regprocout
regprocrecv	regprocsend	regr_avgx	regr_avgy	regr_count	regr_intercept	regr_r2
regr_slope	regr_sxx	regr_sxy	regr_syy	regtypein	regtypeout	regtyperecv
regtypesend	reltime	reltimeeq	reltimege	reltimegt	reltimein	reltimele

reltimelt	reltimene	reltimeout	reltimerecv	reltimesend	repeat	replace
reverse	RI_FKey_cascade_del	RI_FKey_cascade_upd	RI_FKey_check_ins	RI_FKey_check_upd	RI_FKey_noaction_del	RI_FKey_noaction_upd
RI_FKey_restrict_del	RI_FKey_restrict_upd	RI_FKey_setdefault_del	RI_FKey_setdefault_upd	RI_FKey_setnull_del	RI_FKey_setnull_upd	right
round	row_number	row_to_json	rpad	rtrim	scalargtjoinsel	scalargtsel
scalarltjoinsel	scalarltsel	-	schema_to_xml_and_xmlschema	-	session_user	set_bit
set_byte	set_config	set_masklen	shift_jis_2004_to_euc_jis_2004	shift_jis_2004_to_utf8	sjis_to_euc_jp	sjis_to_mic
sjis_to_utf8	smgrin	smgrout	spg_kd_choose	spg_kd_config	spg_kd_inner_consistent	spg_kd_picksplit
spg_quad_choose	spg_quad_config	spg_quad_inner_consistent	spg_quad_leaf_consistent	spg_quad_picksplit	spg_text_choose	spg_text_config
spg_text_inner_consistent	spg_text_leaf_consistent	spg_text_picksplit	spgbeginscan	spgbuild	spgbuildempty	spgbulkdelete
spgcanreturn	spgcostestimate	spgendscan	spggetbitmap	spggettuple	spginsert	spgmarkpos
spgoptions	spgrescan	spgrestrpos	spgvacuumcleanup	stddev	stddev_pop	stddev_samp

string_agg	string_agg_finalfn	string_agg_transfn	strip	sum	suppress_redundant_updates_trigger	-
-	-	tan	text	text_ge	text_gt	text_larger
text_le	text_lt	text_pattern_ge	text_pattern_gt	text_pattern_le	text_pattern_lt	text_smaller
textanycat	textcat	texteq	texticlike	texticnlike	texticregexeq	texticregexne
textin	textlike	textne	textnlike	textout	textrecv	textregexeq
textregexne	textsend	thesaurus_init	thesaurus_lexize	tideq	tidge	tidgt
tidin	tidlarger	tidle	tidlt	tidne	tidout	tidrecv
tidsend	tidsmaller	time	time_cmp	time_eq	time_ge	time_gt
time_hash	time_in	time_larger	time_le	time_lt	time_mi_interval	time_mi_time
time_ne	time_out	time_pl_interval	time_recv	time_send	time_smaller	time_transform
timedate_pl	timemi	timepl	timestamp	timestamp_cmp	timestamp_cmp_date	timestamp_cmp_timestamptz
timestamp_eq	timestamp_eq_date	timestamp_eq_timestamptz	timestamp_ge	timestamp_ge_date	timestamp_ge_timestamptz	timestamp_gt
timestamp_gt_date	timestamp_gt_timestamptz	timestamp_hash	timestamp_in	timestamp_larger	timestamp_le	timestamp_le_date
timestamp_le_timestamptz	timestamp_lt	timestamp_lt_date	timestamp_lt_timestamptz	timestamp_mi	timestamp_mi_interval	timestamp_ne

timestamp_ne_date	timestamp_ne_timestamptz	timestamp_out	timestamp_pl_interval	timestamp_recv	timestamp_send	timestamp_smaller
timestamp_sortsupport	timestamp_transform	timestamptypmodin	timestamptypmodout	timestamptz	timestamptz_cmp	timestamptz_cmp_date
timestamptz_cmp_timestamp	timestamptz_eq	timestamptz_eq_date	timestamptz_eq_timestamp	timestamptz_ge	timestamptz_ge_date	timestamptz_ge_timestamp
timestamptz_gt	timestamptz_gt_date	timestamptz_gt_timestamp	timestamptz_in	timestamptz_larger	timestamptz_le	timestamptz_le_date
timestamptz_le_timestamp	timestamptz_lt	timestamptz_lt_date	timestamptz_lt_timestamp	timestamptz_mi	timestamptz_mi_interval	timestamptz_ne
timestamptz_ne_date	timestamptz_ne_timestamp	timestamptz_out	timestamptz_pl_interval	timestamptz_recv	timestamptz_send	timestamptz_smaller
timestamptztypmodin	timestamptztypmodout	timetypmodin	timetypmodout	timetz	timetz_cmp	timetz_eq
timetz_ge	timetz_gt	timetz_hash	timetz_in	timetz_larger	timetz_le	timetz_lt
timetz_mi_interval	timetz_ne	timetz_out	timetz_pl_interval	timetz_recv	timetz_send	timetz_smaller
timetzdate_pl	timetztypmodin	timetztypmodout	timezone（2069）	timezone（1159）	timezone（2037）	timezone (2070)
timezone (1026)	timezone (2038)	tintervalct	tintervaleq	tintervalge	tintervalgt	tintervalin
tintervalle	tintervalleneq	tintervallenge	tintervallengt	tintervallenle	tintervallenlt	tintervallenne
tintervallt	tintervalne	tintervalout	tintervalov	tintervalrecv	tintervalsame	tintervalsend

tintervalstart	to_ascii（1845）	to_ascii（1847）	to_ascii（1846）	trigger_in	trigger_out	ts_match_qv
ts_match_tq	ts_match_tt	ts_match_vq	ts_rank	ts_rank_cd	ts_rewrite	ts_stat
ts_token_type	ts_typanalyze	tsmatchjoinsel	tsmatchsel	tsq_mcontained	tsq_mcontains	tsquery_and
tsquery_cmp	tsquery_eq	tsquery_ge	tsquery_gt	tsquery_le	tsquery_lt	tsquery_ne
tsquery_not	tsquery_or	tsqueryin	tsqueryout	tsqueryrecv	tsquerysend	tsrange
tsrange_subdiff	tstzrange	tstzrange_subdiff	tsvector_cmp	tsvector_concat	tsvector_eq	tsvector_ge
tsvector_gt	tsvector_le	tsvector_lt	tsvector_ne	tsvector_update_trigger	tsvector_update_trigger_column	tsvectorin
tsvectorout	tsvectorrecv	tsvectorsend	txid_current	txid_current_snapshot	txid_snapshot_in	txid_snapshot_out
txid_snapshot_recv	txid_snapshot_send	txid_snapshot_xip	txid_snapshot_xmax	txid_snapshot_xmin	txid_visible_in_snapshot	uhc_to_utf8
unique_key_recheck	unknownin	unknownout	unknownrecv	unknownsend	unnest	utf8_to_big5
utf8_to_euc_cn	utf8_to_euc_jis_2004	utf8_to_euc_jp	utf8_to_euc_kr	utf8_to_euc_tw	utf8_to_gb18030	utf8_to_gbk
utf8_to_iso8859	utf8_to_iso8859_1	utf8_to_johab	utf8_to_koi8r	utf8_to_koi8u	utf8_to_shift_jis_2004	utf8_to_sjis
utf8_to_uhc	utf8_to_win	uuid_cmp	uuid_eq	uuid_ge	uuid_gt	uuid_hash
uuid_in	uuid_le	uuid_lt	uuid_ne	uuid_out	uuid_recv	uuid_send
var_pop	var_samp	varbit	varbit_in	varbit_out	varbit_recv	varbit_send

varbit_transform	varbitcmp	varbiteq	varbitge	varbitgt	varbitle	varbitlt
varbitne	varbittypmodin	varbittypmodout	varchar	varchar_transform	varcharin	varcharout
varcharrecv	varcharsend	varchartypmodin	varchartypmodout	variance	void_in	void_out
void_recv	void_send	win_to_utf8	win1250_to_latin2	win1250_to_mic	win1251_to_iso	win1251_to_koi8r
win1251_to_mic	win1251_to_win866	win866_to_iso	win866_to_koi8r	win866_to_mic	win866_to_win1251	xideq
xideqint4	xidin	xidout	xidrecv	xidsend	xml	xml_in
-	-	-	xml_out	xml_recv	xml_send	-
-	xmlconcat2	-	xmlvalidate	pg_notify	-	-

实现内部功能的函数

下述列表为GaussDB实现系统内部功能所使用的函数，不推荐使用，若需使用，请联系华为技术支持工程师。

- locktag_decode(locktag text)
  描述：从locktag中解析锁的具体信息。
  返回值类型：text
- smgreq(a smgr, b smgr)
  描述：比较两个smgr是否一样。
  参数：smgr, smgr
  返回值类型：boolean
- smgrne(a smgr, b smgr)
  描述：判断两个smgr是否不一样。
  参数：smgr、smgr
  返回值类型：boolean
- xidin4
  描述：输入4字节的xid。
  参数：cstring
  返回值类型：xid32

- set_hashbucket_info
  描述：设置哈希桶信息。
  参数：text
  返回值类型：boolean
- int1send
  描述：将无符号一字节整数打包放入内部数据缓冲流。
  参数：tinyint
  返回值类型：bytea
- listagg
  描述：list类型agg聚集函数。
  参数：smallint、text
  返回值类型：text
- log_fdw_validator
  描述：验证函数。
  参数：text[]、oid
  返回值类型：void
- nvarchar2typmodin
  描述：获取varchar的typmod信息。
  参数：cstring[]
  返回值类型：integer
- nvarchar2typmodout
  描述：获取varchar的typmod信息，并构造字符串返回。
  参数：integer
  返回值类型：cstring
- read_disable_conn_file
  描述：读取禁止的连接文件。
  参数：nan
  返回值类型：disconn_mode text、disconn_host text、disconn_port text、local_host text、local_port text、redo_finished text
- regex_like_m
  描述：正则匹配，判断字符串是否符合给定的正则表达式。
  参数：text、text
  返回值类型：boolean
- update_pgjob
  描述：更新job。
  参数：bigint、"char"、bigint、timestamp without time zone、timestamp without time zone、timestamp without time zone、timestamp without time zone、timestamp without time zone、smallint、text
  返回值类型：void

- enum_cmp
  描述：枚举类比较函数，用于判断两个枚举类是否相等，以及相对大小。
  参数：anyenum、anyenum
  返回值类型：integer
- enum_eq
  描述：枚举类比较函数，用于实现=符号。
  参数：anyenum、anyenum
  返回值类型：boolean
- enum_first
  描述：返回枚举类中的第一个元素。
  参数：anyenum
  返回值类型：anyenum
- enum_ge
  描述：枚举类比较函数，用于实现>=符号。
  参数：anyenum、anyenum
  返回值类型：boolean
- enum_gt
  描述：枚举类比较函数，用于实现>符号。
  参数：anyenum、anyenum
  返回值类型：boolean
- enum_in
  描述：枚举类比较函数，用于判断元素是否在枚举类中。
  参数：cstring、oid
  返回值类型：anyenum
- enum_larger
  描述：枚举类比较函数，用于实现>符号。
  参数：anyenum、anyenum
  返回值类型：anyenum
- enum_last
  描述：返回枚举类中的最后一个元素。
  参数：anyenum
  返回值类型：anyenum
- enum_le
  描述：枚举类比较函数，用于实现<=符号。
  参数：anyenum、anyenum
  返回值类型：boolean
- enum_lt
  描述：枚举类比较函数，用于实现<符号。

参数：anyenum、anyenum
返回值类型：boolean

- enum_smaller
  描述：枚举类比较函数，用于实现<符号。
  参数：anyenum、anyenum
  返回值类型：boolean
- node_oid_name
  描述：不支持。
  参数：oid
  返回值类型：cstring
- pg_buffercache_pages
  描述：从共享buffer缓存里读取数据。
  参数：nan
  返回值类型：bufferid integer、relfilenode oid、bucketid smallint、reltablespace oid、reldatabase oid、relforknumber smallint、relblocknumber bigint、isdirty boolean、usage_count smallint
- pg_check_xidlimit
  描述：判断nextxid是否>= xidwarnlimit。
  参数：nan
  返回值类型：boolean
- gs_validate_ext_listen_ip
  描述：清理DN实例上连接无效IP的业务。
  参数：详见**表7-114**。
  返回值：bigint pid、text node_name
  注意： 当前环境查询该函数时，第一个入参仅支持“normal”，无返回值。

  表 7-114 GS_VALIDATE_EXT_LISTEN_IP 字段

名称	类型	描述
clear	IN cstring	是否清理，normal表示清理DN实例上连接无效IP的业务。
validate_node_name	IN cstring	指定所要清理IP连接所在DN实例名。
validate_ip	IN cstring	指定所要清理具体的IP。
pid	OUT bigint	被清理的IP连接所在业务线程ID。
node_name	OUT text	被清理的IP连接所在实例名。

- gs_comm_listen_address_ext_info
  描述：显示当前连接listen_address_ext配置扩展IP的DFX信息。
  参数：nan

返回值类型：text node_name、text app、bigint tid、integer lwtid、bigint query_id、integer socket、text remote_ip、text remote_port、text local_ip、text local_port。

注意： 当前不支持查询该函数。

表 7-115 GS_COMM_LISTEN_ADDRESS_EXT_INFO 字段

名称	类型	描述
node_name	OUT text	描述当前实例名。
app	OUT text	描述当前连接DN的客户端。
tid	OUT bigint	描述当前线程的线程号。
lwtid	OUT integer	描述当前线程的轻量级线程号。
query_id	OUT bigint	描述当前线程的查询ID。
socket	OUT integer	描述当前物理连接的socket fd。
remote_ip	OUT text	描述当前连接对端IP。
remote_port	OUT text	描述当前连接对端port。
local_ip	OUT text	描述当前连接本端IP。
local_port	OUT text	描述当前连接本端port。

- gs_get_global_listen_address_ext_info()

  描述：提供查询全局扩展IP配置信息。

  参数：详见**表7-116**。

  返回值类型：text node_name、text host、text port、text ext_listen_ip。

  注意： 当前不支持查询该函数。

  表 7-116 GS_GET_GLOBAL_LISTEN_ADDRESS_EXT_INFO 字段

名称	类型	描述
dn_mode	IN cstring	指定显示的DN范围，null默认查询所有。
node_name	OUT text	DN实例名。
host	OUT text	DN实例侦听IP。
port	OUT text	DN实例侦听port。
ext_listen_ip	OUT text	DN实例侦听扩展IP。

- gs_get_listen_address_ext_info()

  描述：提供查询当前DN实例扩展IP配置信息。

  参数：nan

  返回值类型：text node_name、text host、bigint port、text ext_listen_ip。

注意： 当前不支持查询该函数。

表 7-117 GS_GET_LISTEN_ADDRESS_EXT_INFO

名称	类型	描述
node_name	OUT text	DN实例名。
host	OUT text	DN实例侦听IP。
port	OUT bigint	DN实例侦听port。
ext_listen_ip	OUT text	DN实例侦听扩展IP。

- pg_comm_delay
  描述：展示单个DN的通信库时延状态。
  参数：nan
  返回值类型：text、text、integer、integer、integer、integer
- pg_comm_recv_stream
  描述：展示单个DN上所有的通信库接收流状态。
  参数：nan
  返回值类型：text、bigint、text、bigint、integer、integer、integer、text、bigint、integer、integer、integer、bigint、bigint、bigint、bigint、bigint
- pg_comm_send_stream
  描述：展示单个DN上所有的通信库发送流状态。
  参数：nan
  返回值类型：text、bigint、text、bigint、integer、integer、integer、text、bigint、integer、integer、integer、bigint、bigint、bigint、bigint、bigint
- pg_comm_status
  描述：展示单个DN的通信状态。
  参数：nan
  返回值类型：text、integer、integer、bigint、bigint、bigint、bigint、bigint、integer、integer、integer、integer、integer
- pg_log_comm_status
  描述：在DN上打印一些log。
  参数：nan
  返回值类型：boolean
- pg_parse_clog
  描述：解析clog获取xid的status。
  参数：nan
  返回值类型：xid xid、status text
- pg_pool_ping
  描述：设置PoolerPing。

参数：boolean

返回值类型：SETOF boolean

- pg_resume_bkp_flag

  描述：用于备份恢复获取delay xlong标志。

  参数：slot_name name

  返回值类型：start_backup_flag boolean、to_delay boolean、ddl_delay_recycle_ptr text、rewind_time text

- psortoptions

  描述：返回psort属性。

  参数：text[]、boolean

  返回值类型：bytea

- xideq4

  描述：对比两个xid类型的值是否相等。

  参数：xid32、xid32

  返回值类型：boolean

- xideqint8

  描述：对比xid类型和int8类型的值是否相等。

  参数：xid、bigint

  返回值类型：boolean

- xidlt

  描述：返回xid1 < xid2是否成立。

  参数：xid、xid

  返回值类型：boolean

- xidlt4

  描述：返回xid1 < xid2是否成立。

  参数：xid32、xid32

  返回值类型：boolean

- gs_shutdown_cross_region_walsenders

  描述：中断跨集群流式复制。

  参数：nan

  返回值类型：void

  备注：调用该函数的用户需要具有SYSADMIN权限或具有OPRADMIN权限，运维管理员角色须打开operate_mode。

- is_dblink_in_transaction

  描述：判断当前事务中是否使用了oid对应的DATABASE LINK。

  参数：oid

  返回值类型：boolean

- dblink_has_updatasent

  描述：判断当前事务中是否使用oid对应的DATABASE LINK发送了dml语句且未提交。

  参数：oid

返回值类型：boolean

- get_last_xmin_by_oid
  描述：通过表的oid获取该表所有字段中最大的xmin值。
  参数：oid
  返回值类型：xid
- get_relid_by_relname
  描述：通过表的表名及relnamespace获取表的oid。
  参数：cstring, oid
  返回值类型：oid
- copy_summary_create
  描述：copy_summary_create用于创建gs_copy_summary表。
  参数：无
  返回值类型：text

7.5.42 废弃函数

GaussDB中下列函数在最新版本中已废弃：

- gs_wlm_get_session_info
- gs_wlm_get_user_session_info
- pgxc_get_csn
- pgxc_get_stat_dirty_tables
- pgxc_get_thread_wait_status
- pgxc_gtm_snapshot_status
- pgxc_is_committed
- pgxc_lock_for_backup
- pgxc_lock_for_sp_database
- pgxc_lock_for_transfer
- pgxc_log_comm_status
- pgxc_max_datanode_size
- pgxc_node_str
- pgxc_pool_check
- pgxc_pool_connection_status
- pgxc_pool_reload
- pgxc_prepared_xact
- pgxc_snapshot_status
- pgxc_stat_dirty_tables
- pgxc_unlock_for_sp_database
- pgxc_unlock_for_transfer
- pgxc_version
- array_extend

- prepare_statement_status
- remote_rto_stat
- dbe_perf.global_slow_query_info
- dbe_perf.global_slow_query_info_bytime
- dbe_perf.global_slow_query_history
- pg_stat_get_pooler_status
- pg_stat_get_wlm_node_resource_info
- pg_stat_get_wlm_session_info_internal
- DBE_PERF.get_wlm_controlgroup_ng_config()
- DBE_PERF.get_wlm_user_resource_runtime()
- global_space_shrink
- pg_pool_validate
- gs_stat_ustore
- table_skewness(text)
- table_skewness(text, text, text)
- pv_compute_pool_workload()

7.6 表达式

7.6.1 简单表达式

逻辑表达式

逻辑表达式的操作符和运算规则，请参见**逻辑操作符**。

比较表达式

常用的比较操作符，请参见**比较操作符**。

除比较操作符外，还可以使用以下句式结构：

- BETWEEN操作符
  a BETWEEN x AND y等效于a >= x AND a <= y
  a NOT BETWEEN x AND y等效于a < x OR a > y
- 检查一个值是不是null，可使用：
  expression IS NULL
  expression IS NOT NULL
  或者与之等价的句式结构，但不是标准的：
  expression ISNULL
  expression NOTNULL

须知

- 不要写expression=NULL或expression<>(!=)NULL，因为NULL代表一个未知的值，不能通过该表达式判断两个未知值是否相等。
- XML类型数据仅支持比较表达式 IS NULL、IS NOT NULL。

- is distinct from/is not distinct from
  - is distinct from

    A和B的数据类型、值不完全相同时为true。

    A和B的数据类型、值完全相同时为false。

    将空值视为相同。
  - is not distinct from

    A和B的数据类型、值不完全相同时为false。

    A和B的数据类型、值完全相同时为true。

    将空值视为相同。
- <=> 安全等于操作符

  在 '=' 比较的基础上增加NULL值的比较，在操作符左右值都不为NULL时与 '=' 结果相同。

  A和B的数据类型、值不完全相同时为false。

  A和B的数据类型、值完全相同时为true。

  将空值视为相同。

  说明

  - <=>操作符与 is not distinct from 用法完全相同。
  - 该操作符仅在数据库兼容MY类型时（即sql_compatibility = 'B'）有效，其他类型不支持该操作符。

示例

```
gaussdb=# SELECT 2 BETWEEN 1 AND 3 AS RESULT;
 result
----------
 t
(1 row)

gaussdb=# SELECT 2 >= 1 AND 2 <= 3 AS RESULT;
 result
----------
 t
(1 row)

gaussdb=# SELECT 2 NOT BETWEEN 1 AND 3 AS RESULT;
 result
----------
 f
(1 row)

gaussdb=# SELECT 2 < 1 OR 2 > 3 AS RESULT;
 result
----------
 f
(1 row)
```

```
gaussdb=# SELECT 2+2 IS NULL AS RESULT;
 result
----------
 f
(1 row)

gaussdb=# SELECT 2+2 IS NOT NULL AS RESULT;
 result
----------
 t
(1 row)

gaussdb=# SELECT 2+2 ISNULL AS RESULT;
 result
----------
 f
(1 row)

gaussdb=# SELECT 2+2 NOTNULL AS RESULT;
 result
----------
 t
(1 row)

gaussdb=# SELECT 2+2 IS DISTINCT FROM NULL AS RESULT;
 result
----------
 t
(1 row)

gaussdb=# SELECT 2+2 IS NOT DISTINCT FROM NULL AS RESULT;
 result
----------
 f
(1 row)
gaussdb=#  create database test2 DBCOMPATIBILITY  'B';
CREATE DATABASE
gaussdb=#  \c b_database
b_database=# SELECT 1 <=> 1 AS RESULT;
 result
--------
 t
(1 row)
b_databaseSELECT NULL <=> 1 AS RESULT;
 result
--------
 f
(1 row)

b_databaseSELECT NULL <=> NULL AS RESULT;
 result
--------
 t
(1 row)
b_database=# \c postgres
```

7.6.2 条件表达式

在执行SQL语句时，可通过条件表达式筛选出符合条件的数据。

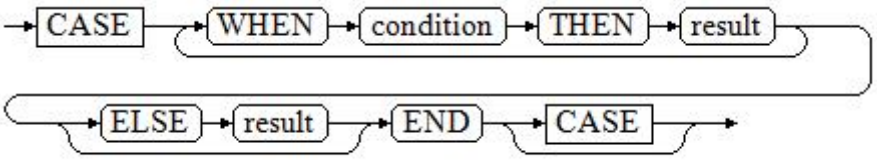
条件表达式主要有以下几种：

- CASE

  CASE表达式是条件表达式，类似于其他编程语言中的CASE语句。

  CASE表达式的语法图请参考**图7-1**。

图 7-1 case::=



CASE子句可以用于合法的表达式中。condition是一个返回BOOLEAN数据类型的表达式：

- 如果结果为真，CASE表达式的结果就是符合该条件所对应的result。
- 如果结果为假，则以相同方式处理随后的WHEN或ELSE子句。
- 如果各WHEN condition都不为真，表达式的结果就是在ELSE子句执行的result。如果省略了ELSE子句且没有匹配的条件，结果为NULL。
- 支持对XML类型数据操作。

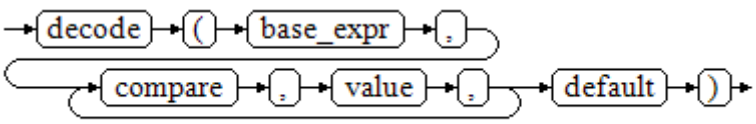
示例：

```
gaussdb=# CREATE TABLE case_when_t1(CW_COL1 INT);

gaussdb=# INSERT INTO case_when_t1 VALUES (1), (2), (3);

gaussdb=# SELECT * FROM case_when_t1;
cw_col1
---------
 1
 2
 3
(3 rows)

gaussdb=# SELECT CW_COL1, CASE WHEN CW_COL1=1 THEN 'one' WHEN CW_COL1=2 THEN 'two'
ELSE 'other' END FROM case_when_t1 ORDER BY 1;
 cw_col1 | case
---------+-------
       1 | one
       2 | two
       3 | other
(3 rows)

gaussdb=# DROP TABLE case_when_t1;
```

- DECODE

  DECODE的语法图请参见**图7-2**。

图 7-2 decode::=



将表达式base_expr与后面的每个compare(n) 进行比较，如果匹配返回相应的value(n)。如果没有发生匹配，则返回default。

支持对XML类型数据操作。

示例请参见**条件表达式函数**。

```
gaussdb=# SELECT DECODE('A','A',1,'B',2,0);
 case
```

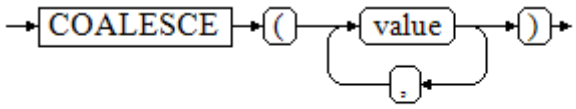


```
------
   1
(1 row)
```

- COALESCE

  COALESCE的语法图请参见**图7-3**。

  图 7-3 coalesce::=

  COALESCE返回它的第一个非NULL的参数值。如果参数都为NULL，则返回NULL。它常用于在显示数据时用缺省值替换NULL。和CASE表达式一样，COALESCE只计算用来判断结果的参数，即在第一个非空参数右边的参数不会被计算。

  支持对XML类型数据操作。

  示例：

```
gaussdb=# CREATE TABLE c_tabl(description varchar(10), short_description varchar(10), last_value
varchar(10)) ;

gaussdb=# INSERT INTO c_tabl VALUES('abc', 'efg', '123');
gaussdb=# INSERT INTO c_tabl VALUES(NULL, 'efg', '123');

gaussdb=# INSERT INTO c_tabl VALUES(NULL, NULL, '123');

gaussdb=# SELECT description, short_description, last_value, COALESCE(description, short_description,
last_value) FROM c_tabl ORDER BY 1, 2, 3, 4;
 description | short_description | last_value | coalesce
-------------+-------------------+------------+----------
 abc         | efg               | 123        | abc
             | efg               | 123        | efg
             |                   | 123        | 123
(3 rows)

gaussdb=# DROP TABLE c_tabl;
```

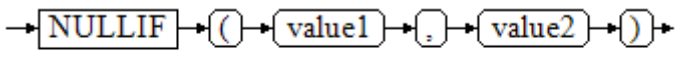
  如果description不为NULL，则返回description的值，否则计算下一个参数short_description；如果short_description不为NULL，则返回short_description的值，否则计算下一个参数last_value；如果last_value不为NULL，则返回last_value的值，否则返回（none）。

```
gaussdb=# SELECT COALESCE(NULL,'Hello World');
   coalesce
---------------
 Hello World
(1 row)
```

- NULLIF

  NULLIF的语法图请参见**图7-4**。

  图 7-4 nullif::=

只有当value1和value2相等时，NULLIF才返回NULL。否则它返回value1。支持对XML类型数据操作。

示例：

```
gaussdb=# CREATE TABLE null_if_t1 (
    NI_VALUE1 VARCHAR(10),
    NI_VALUE2 VARCHAR(10)
);

gaussdb=# INSERT INTO null_if_t1 VALUES('abc', 'abc');
gaussdb=# INSERT INTO null_if_t1 VALUES('abc', 'efg');

gaussdb=# SELECT NI_VALUE1, NI_VALUE2, NULLIF(NI_VALUE1, NI_VALUE2) FROM null_if_t1 ORDER BY 1, 2, 3;

 ni_value1 | ni_value2 | nullif
-----------+-----------+--------
 abc       | abc       |
 abc       | efg       | abc
(2 rows)
gaussdb=# DROP TABLE null_if_t1;
```

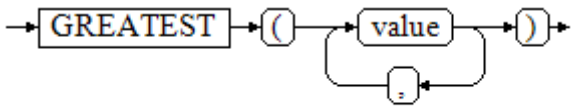
如果value1等于value2则返回NULL，否则返回value1。

```
gaussdb=# SELECT NULLIF('Hello','Hello World');
 nullif
--------
 Hello
(1 row)
```

- GREATEST（最大值），LEAST（最小值）

  GREATEST的语法图请参见**图7-5**。

  图 7-5 greatest::=

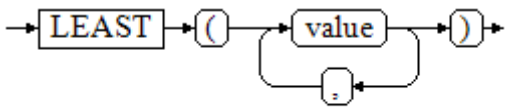
  

  从一个任意数字表达式的列表里选取最大的数值。支持对XML类型数据操作。

  ```
  gaussdb=# SELECT greatest(9000,155555,2.01);
   greatest
  ----------
    155555
  (1 row)
  ```

  LEAST的语法图请参见**图7-6**。

  图 7-6 least::=

  从一个任意数字表达式的列表里选取最小的数值。

  以上的数字表达式必须都可以转换成一个普通的数据类型，该数据类型将是结果类型。

列表中的NULL值将被忽略。只有所有表达式的结果都是NULL的时候，结果才是NULL。

支持对XML类型数据操作。

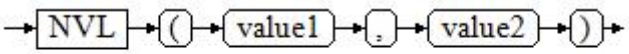
```
gaussdb=# SELECT least(9000,2);
 least
-------
     2
(1 row)
```

示例请参见**条件表达式函数**。

- NVL

  NVL的语法图请参见**图7-7**。

  图 7-7 nvl::=

  如果value1为NULL则返回value2，如果value1非NULL，则返回value1。支持对XML类型数据操作。

  示例：

```
gaussdb=# SELECT nvl(null,1);
nvl
-----
 1
(1 row)
gaussdb=# SELECT nvl ('Hello World',1);
      nvl
---------------
 Hello World
(1 row)
```

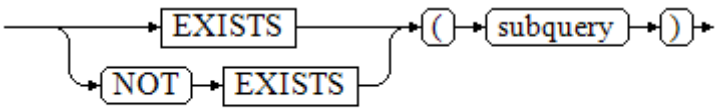
7.6.3 子查询表达式

子查询表达式主要有以下几种：

- EXISTS/NOT EXISTS

  EXISTS/NOT EXISTS的语法图请参见**图7-8**。

  图 7-8 EXISTS/NOT EXISTS::=

  EXISTS的参数是一个任意的SELECT语句，即子查询。系统对子查询进行运算以判断它是否返回行。如果它至少返回一行，则EXISTS结果就为“真”；如果子查询没有返回任何行， EXISTS的结果是“假”。

  这个子查询通常只是运行到能判断它是否可以生成至少一行为止，而不是等到全部结束。

  不支持对XML类型数据操作。

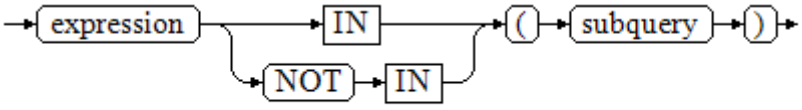
示例：

```
gaussdb=# CREATE TABLE exists_t1(a int, b int);
gaussdb=# INSERT INTO exists_t1 VALUES(1, 2),(2, 3),(3, 4),(4, 5);

gaussdb=# CREATE TABLE exists_t2(a int, c int);
gaussdb=# INSERT INTO exists_t2 VALUES(3, 4),(4, 5),(5, 6),(6, 7);

gaussdb=# SELECT * FROM exists_t1 t1 WHERE EXISTS (SELECT * FROM exists_t2 t2 WHERE t2.a =
t1.a);
 a | b
---+---
 3 | 4
 4 | 5
(2 rows)

gaussdb=# DROP TABLE exists_t1, exists_t2;
```

- IN/NOT IN

  IN/NOT IN的语法请参见图7-9。

  图 7-9 IN/NOT IN::=

  右边是一个圆括弧括起来的子查询，它必须只返回一个字段。左边表达式对子查询结果的每一行进行一次计算和比较。如果找到任何相等的子查询行，则IN结果为“真”。如果没有找到任何相等行，则结果为“假”（包括子查询没有返回任何行的情况）。

  表达式或子查询行里的NULL遵照SQL处理布尔值和NULL组合时的规则。如果两个行对应的字段都相等且非空，则这两行相等；如果任意对应字段不等且非空，则这两行不等；否则结果是未知（NULL）。如果每一行的结果都是不等或NULL，并且至少有一个NULL，则IN的结果是NULL。

  不支持对XML类型数据操作。

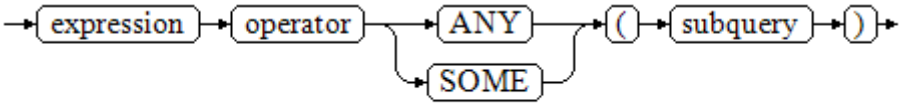
  示例：

  ```
  gaussdb=# CREATE TABLE in_t1(a int, b int);
  gaussdb=# INSERT INTO in_t1 VALUES(1, 2),(2, 3),(3, 4),(4, 5);

  gaussdb=# CREATE TABLE in_t2(a int, c int);
  gaussdb=# INSERT INTO in_t2 VALUES(3, 4),(4, 5),(5, 6),(6, 7);

  gaussdb=# SELECT * FROM in_t1 t1 WHERE t1.a IN (SELECT t2.a FROM in_t2 t2);
   a | b
  ---+---
   3 | 4
   4 | 5
  (2 rows)

  gaussdb=# DROP TABLE in_t1, in_t2;
  ```

- ANY/SOME

  ANY/SOME的语法图请参见图7-10。

图 7-10 any/some::=



右边是一个圆括弧括起来的子查询，它必须只返回一个字段。左边表达式使用operator对子查询结果的每一行进行一次计算和比较，其结果必须是布尔值。如果至少获得一个真值，则ANY结果为“真”。如果全部获得假值，则结果是“假”（包括子查询没有返回任何行的情况）。SOME是ANY的同义词。IN与ANY可以等效替换 。

不支持对XML类型数据操作。

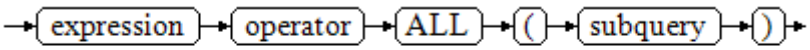
示例：

```
gaussdb=# CREATE TABLE any_t1(a int, b int);
gaussdb=# INSERT INTO any_t1 VALUES(1, 2),(2, 3),(3, 4),(4, 5);

gaussdb=# CREATE TABLE any_t2(a int, c int);
gaussdb=# INSERT INTO any_t2 VALUES(3, 4),(4, 5),(5, 6),(6, 7);

gaussdb=# SELECT * FROM any_t1 t1 WHERE t1.a < ANY(SELECT t2.a FROM any_t2 t2 where t2.a = 3 or t2.a = 4);
 a | b
---+---
 1 | 2
 2 | 3
 3 | 4
(3 rows)

gaussdb=# DROP TABLE any_t1, any_t2;
```

- ALL

  ALL的语法请参见**图7-11**。

  图 7-11 all::=

  右边是一个圆括弧括起来的子查询，它必须只返回一个字段。左边表达式使用operator对子查询结果的每一行进行一次计算和比较，其结果必须是布尔值。如果全部获得真值，ALL结果为“真”（包括子查询没有返回任何行的情况）。如果至少获得一个假值，则结果是“假”。

  不支持对XML类型数据操作。

  示例：

```
gaussdb=# CREATE TABLE all_t1(a int, b int);
gaussdb=# INSERT INTO all_t1 VALUES(1, 2),(2, 3),(3, 4),(4, 5);

gaussdb=# CREATE TABLE all_t2(a int, c int);
gaussdb=# INSERT INTO all_t2 VALUES(3, 4),(4, 5),(5, 6),(6, 7);

gaussdb=# SELECT * FROM all_t1 t1 WHERE t1.a < ALL(SELECT t2.a FROM all_t2 t2 where t2.a = 3 or t2.a = 4);
 a | b
---+---
 1 | 2
```



```
 2 | 3
(2 rows)

gaussdb=# DROP TABLE all_t1, all_t2;
```

7.6.4 数组表达式

IN

expression **IN** *(value [, ...])*

右侧括号中的是一个表达式列表。左侧表达式的结果与表达式列表的内容进行比较。如果列表中的内容符合左侧表达式的结果，则IN的结果为true。如果没有相符的结果，则IN的结果为false。

示例如下：

```
gaussdb=# SELECT 8000+500 IN (10000, 9000) AS RESULT;
 result
----------
 f
(1 row)
```

说明

- 如果表达式结果为null，或者表达式列表不符合表达式的条件且右侧表达式列表返回结果至少一处为空，则IN的返回结果为null，而不是false。这样的处理方式和SQL返回空值的布尔组合规则是一致的。
- 不支持对XML类型数据操作。

NOT IN

expression **NOT IN** *(value [, ...])*

右侧括号中的是一个表达式列表。左侧表达式的结果与表达式列表的内容进行比较。如果在列表中的内容没有符合左侧表达式结果的内容，则NOT IN的结果为true。如果有符合的内容，则NOT IN的结果为false。

示例如下：

```
gaussdb=# SELECT 8000+500 NOT IN (10000, 9000) AS RESULT;
 result
----------
 t
(1 row)
```

说明

- 如果查询语句返回结果为空，或者表达式列表不符合表达式的条件且右侧表达式列表返回结果至少一处为空，则NOT IN的返回结果为null，而不是false。这样的处理方式和SQL返回空值的布尔组合规则是一致的。
- 在所有情况下X NOT IN Y等价于NOT(X IN Y)。
- 不支持对XML类型数据操作。

ANY/SOME (array)

expression operator **ANY** *(array expression)*

expression operator **SOME** *(array expression)*

右侧括号中的是一个数组表达式，它必须产生一个数组值。左侧表达式的结果使用操作符对数组表达式的每一行结果都进行计算和比较，比较结果必须是布尔值。

示例如下：

```
gaussdb=# SELECT 8000+500 < SOME (array[10000,9000]) AS RESULT;
 result
----------
 t
(1 row)
gaussdb=# SELECT 8000+500 < ANY (array[10000,9000]) AS RESULT;
 result
----------
 t
(1 row)
```

说明

- 如果对比结果至少获取一个真值，则ANY的结果为true。
- 如果对比结果没有真值，则ANY的结果为false。
- 如果结果没有真值，并且数组表达式生成至少一个值为null，则ANY的值为NULL，而不是false。这样的处理方式和SQL返回空值的布尔组合规则是一致的。
- SOME是ANY的同义词。
- 不支持对XML类型数据操作。

ALL (array)

expression operator **ALL** *(array expression)*

右侧括号中的是一个数组表达式，它必须产生一个数组值。左侧表达式的结果使用操作符对数组表达式的每一行结果都进行计算和比较，比较结果必须是布尔值。

- 如果所有的比较结果都为真值（包括数组不含任何元素的情况），则ALL的结果为true。
- 如果存在一个或多个比较结果为假值，则ALL的结果为false。
- 如果数组表达式产生一个NULL数组，则ALL的结果为NULL。如果左边表达式的值为NULL ，则ALL的结果通常也为NULL(不严格的比较操作符可能得到不同的结果)。如果右边的数组表达式中包含null元素并且比较结果没有假值，则ALL的结果将是NULL(不严格的比较操作符可能得到不同的结果)， 而不是“真”。这样的处理方式和SQL返回空值的布尔组合规则是一致的。
- 不支持对XML类型数据操作。

```
gaussdb=# SELECT 8000+500 < ALL (array[10000,9000]) AS RESULT;
 result
----------
 t
(1 row)
```

7.6.5 行表达式

语法如下：

```
row_constructor operator row_constructor
```

两边都是一个行构造器，两行值必须具有相同数目的字段，每一行都进行比较，行比较允许使用=，<>，<，<=，>=等操作符，或其中一个相似的语义符。

对于<，<=，>，> =的情况下，行中元素从左到右依次比较，直到遇到一对不相等的元素或者一对为空的元素。如果这对元素中存在至少一个null值，则比较结果是未知的

（null），否则这对元素的比较结果为最终的结果。如果最终没有遇到不相等或者为空的元素，则认为这两行值相等，根据操作符含义判断最终结果。

不支持对XML类型数据操作。

示例：

```
gaussdb=# SELECT ROW(1,2,NULL) < ROW(1,3,0) AS RESULT;
 result
----------
 t
(1 row)

gaussdb=# SELECT (4,5,6) > (3,2,1) AS result;
 result
--------
 t
(1 row)

gaussdb=# SELECT (4,1,1) > (3,2,1) AS result;
 result
--------
 t
(1 row)

gaussdb=# SELECT ('test','data') > ('data','data') AS result;
 result
--------
 t
(1 row)

gaussdb=# SELECT (4,1,1) > (3,2,null) AS result;
 result
--------
 t
(1 row)

gaussdb=# SELECT (null,1,1) > (3,2,1) AS result;
 result
--------

(1 row)

gaussdb=# SELECT (null,5,6) > (null,5,6) AS result;
 result
--------

(1 row)

gaussdb=# SELECT (4,5,6) > (4,5,6) AS result;
 result
--------
 f
(1 row)

gaussdb=# SELECT (2,2,5) >= (2,2,3) AS result;
 result
--------
 t
(1 row)

gaussdb=# SELECT (2,2,1) <= (2,2,3) AS result;
 result
--------
 t
(1 row)
```

=，<>和别的操作符使用略有不同。如果两行值的所有字段都是非空并且符合操作符条件，则认为两行是符合操作符条件的；如果两行值的任意字段为非空并且不符合操作

符条件，则认为两行是不符合操作符条件的；如果两行值的任意字段为空，则比较的结果是未知的（null）。

示例：

```
gaussdb=# SELECT (1,2,3) = (1,2,3) AS result;
 result
--------
 t
(1 row)

gaussdb=# SELECT (1,2,3) <> (2,2,3) AS result;
 result
--------
 t
(1 row)

gaussdb=# SELECT (2,2,3) <> (2,2,null) AS result;
 result
--------

(1 row)

gaussdb=# SELECT (null,5,6) <> (null,5,6) AS result;
 result
--------

(1 row)
```

7.7 伪列

ROWNUM是一个伪列，它返回一个数字，表示从查询中获取结果的行编号。第一行的ROWNUM为1，第二行的为2，以此类推。

使用ROWNUM来限制查询返回的行数，如以下示例所示：

```
gaussdb=# CREATE TABLE Students (name varchar(20), id int) with (STORAGE_TYPE = USTORE);
gaussdb=# INSERT INTO Students VALUES ('Jack', 35);
gaussdb=# INSERT INTO Students VALUES ('Leon', 15);
gaussdb=# INSERT INTO Students VALUES ('James', 24);
gaussdb=# INSERT INTO Students VALUES ('Taker', 81);
gaussdb=# INSERT INTO Students VALUES ('Mary', 25);
gaussdb=# INSERT INTO Students VALUES ('Rose', 64);
gaussdb=# INSERT INTO Students VALUES ('Perl', 18);
gaussdb=# INSERT INTO Students VALUES ('Under', 57);
gaussdb=# INSERT INTO Students VALUES ('Angel', 101);
gaussdb=# INSERT INTO Students VALUES ('Frank', 20);
gaussdb=# INSERT INTO Students VALUES ('Charlie', 40);

--输出表Students前10行数据。
gaussdb=# SELECT * FROM Students WHERE rownum <= 10;
 name  | id
-------+-----
 Jack  |  35
 Leon  |  15
 James |  24
 Taker |  81
 Mary  |  25
 Rose  |  64
 Perl  |  18
 Under |  57
 Angel | 101
 Frank |  20
(10 rows)
```

如果有子句跟在同一查询语句中，则结果输出的行将按照子句重新排序:

```
gaussdb=# SELECT * FROM Students WHERE rownum < 5 order by 1;
 name  | id
```



```
-------+----
 Jack  | 35
 James | 24
 Leon  | 15
 Taker | 81
(4 rows)
```

如果将子句嵌入到子查询中并将条件放在最外层的查询中，则能够在排序后使用ROWNUM条件:

```
gaussdb=# SELECT rownum, * FROM (SELECT * FROM Students order by 1) WHERE rownum <= 2;
 rownum |  name   | id
--------+---------+-----
      1 | Angel   | 101
      2 | Charlie |  40
(2 rows)
```

当ROWNUM大于正整数的值，认为条件始终为false。例如以下所示，该语句不会返回表中任何结果：

```
gaussdb=# SELECT * FROM Students WHERE rownum > 1;
 name | id
------+----
(0 rows)
```

使用ROWNUM指定给表的一定范围的每一行分配值:

```
gaussdb=# SELECT * FROM Students;
  name   | id
---------+-----
 Jack    |  35
 Leon    |  15
 James   |  24
 Taker   |  81
 Mary    |  25
 Rose    |  64
 Perl    |  18
 Under   |  57
 Angel   | 101
 Frank   |  20
 Charlie |  40
(11 rows)

gaussdb=# update Students set id = id + 5 WHERE rownum < 4;
UPDATE 3
gaussdb=# SELECT * FROM Students;
  name   | id
---------+-----
 Jack    |  40
 Leon    |  20
 James   |  29
 Taker   |  81
 Mary    |  25
 Rose    |  64
 Perl    |  18
 Under   |  57
 Angel   | 101
 Frank   |  20
 Charlie |  40
(11 rows)

gaussdb=# DROP TABLE Students;
DROP TABLE
```

使用ROWNUM有一定的约束条件：

- ROWNUM不可作为别名，以免SQL语句出现歧义。
- 创建索引时不可使用ROWNUM。
- 创建表时默认值不可为ROWNUM。

- Where子句中不可使用ROWNUM的别名。
- 在插入数据时不可使用ROWNUM。
- 在无表查询中不可以使用ROWNUM。
- ROWNUM不能用于Limit子句。
- ROWNUM不能用于EXECUTE语句的参数。
- UPSERT语句不支持ROWNUM用做update子句更新。
- 若having子句中含有ROWNUM（且不在聚合函数中）时，group by子句中必须含有ROWNUM（且不在聚合函数中），除非group by子句存在表达式，例如：SELECT a + a FROM t group by a + a having rownum < 5。
- having子句中如果存在ROWNUM条件则不允许having子句下推至扫描节点：

```
gaussdb=# CREATE TABLE test (a int, b int);
CREATE TABLE
gaussdb=# INSERT INTO test SELECT generate_series, generate_series from generate_series(1, 10);
INSERT 0 10

--rownum条件不能下推至seqscan。
gaussdb=# EXPLAIN SELECT a,rownum FROM test group by a,rownum having rownum < 5;
                            QUERY PLAN
------------------------------------------------------------------------
 HashAggregate  (cost=42.23..69.10 rows=2149 width=4)
   Group By Key: a, ROWNUM
   Filter: ((ROWNUM) < 5)
   ->  Rownum  (cost=0.00..31.49 rows=2149 width=4)
         ->  Seq Scan on test  (cost=0.00..31.49 rows=2149 width=4)
(5 rows)
```

- 子查询中如果存在ROWNUM条件则不允许谓词下推至扫描节点：

```
--b<5 不能下推至seqscan。
gaussdb=# EXPLAIN SELECT * FROM (SELECT * FROM test WHERE rownum < 5) WHERE b < 5;
                              QUERY PLAN
-----------------------------------------------------------------------------
 Subquery Scan on __unnamed_subquery__  (cost=0.00..0.01 rows=1 width=8)
   Filter: (__unnamed_subquery__.b < 5)
   ->  Rownum  (cost=0.00..0.00 rows=1 width=8)
         StopKey: (ROWNUM < 5)
         ->  Seq Scan on test  (cost=0.00..31.49 rows=2149 width=8)
(5 rows)

gaussdb=# DROP TABLE test;
DROP TABLE
```

⚠ 注意

不推荐ROWNUM条件用于JOIN ON子句，GaussDB中ROWNUM条件用于JOIN ON子句时在LEFT JOIN、RIGHT JOIN、FULL JOIN场景下和MERGE INTO场景下与其他数据库行为不一致，直接进行业务迁移存在风险。

7.8 类型转换

7.8.1 概述

背景信息

在SQL语言中，每个数据都与一个决定其行为和用法的数据类型相关。GaussDB提供一个可扩展的数据类型系统，该系统比其它SQL实现更具通用性和灵活性。因此，

GaussDB中大多数类型转换是由通用规则来管理的，这种做法允许使用混合类型的表达式。

GaussDB扫描/分析器只将词法元素分解成五个基本种类：整数、浮点数、字符串、标识符和关键字。大多数非数字类型首先表现为字符串。SQL语言的定义允许将常量字符串声明为具体的类型。例如：

```
gaussdb=# SELECT text 'Origin' AS "label", point '(0,0)' AS "value";
 label  | value
--------+-------
 Origin | (0,0)
(1 row)
```

示例中有两个文本常量，类型分别为text和point。如果没有为字符串文本声明类型，则该文本首先被定义成一个unknown类型。

在GaussDB分析器里，有四种基本的SQL结构需要独立的类型转换规则：

- 函数调用

  多数SQL类型系统是建筑在一套丰富的函数上的。函数调用可以有一个或多个参数。因为SQL允许函数重载，所以不能通过函数名直接找到要调用的函数，分析器必须根据函数提供的参数类型选择正确的函数。

- 操作符

  SQL允许在表达式上使用前缀或后缀（单目）操作符，也允许表达式内部使用双目操作符（两个参数）。像函数一样，操作符也可以被重载，因此操作符的选择也和函数一样取决于参数类型。

- 值存储

  INSERT和UPDATE语句将表达式结果存入表中。语句中的表达式类型必须和目标字段的类型一致或者可以转换为一致。

- UNION，CASE和相关构造

  因为联合SELECT语句中的所有查询结果必须在一列里显示出来，所以每个SELECT子句中的元素类型必须相互匹配并转换成一个统一类型。类似地，一个CASE构造的结果表达式必须转换成统一的类型，这样整个case表达式会有一个统一的输出类型。同样的要求也存在于ARRAY构造以及GREATEST和LEAST函数中。

系统表pg_cast存储了有关数据类型之间的转换关系以及如何执行这些转换的信息。详细信息请参见**PG_CAST**。

语义分析阶段会决定表达式的返回值类型并选择适当的转换行为。数据类型的基本类型分类，包括：Boolean，numeric，string，bitstring，datetime，timespan，geometric和network。每种类型都有一种或多种首选类型用于解决类型选择的问题。根据首选类型和可用的隐含转换，就可能保证有歧义的表达式（那些有多个候选解析方案的）得到有效的方式解决。

所有类型转换规则都是建立在下面几个基本原则上的：

- 隐含转换绝不能有奇怪的或不可预见的输出。
- 如果一个查询不需要隐含的类型转换，分析器和执行器不应该进行更多的额外操作。这就是说，任何一个类型匹配、格式清晰的查询不应该在分析器里耗费更多的时间，也不应该向查询中引入任何不必要的隐含类型转换调用。
- 另外，如果一个查询在调用某个函数时需要进行隐式转换，当用户定义了一个有正确参数的函数后，解释器应该选择使用新函数。
- XML类型数据不支持隐式类型转换，包括字符串和XML类型之间的隐式转换。

7.8.2 操作符

操作符类型解析

1. 从系统表pg_operator中选出要考虑的操作符。如果可以找到一个参数类型以及参数个数都一致的操作符，那么这个操作符就是最终使用的操作符。如果找到了多个备选的操作符，将从中选择一个最合适的。
2. 寻找最优匹配。
   a. 抛弃那些输入类型不匹配并且也不能隐式转换成匹配的候选操作符。unknown文本在这种情况下可以转换成任何东西。如果只剩下一个候选项，则用之，否则继续下一步。
   b. 遍历所有候选操作符，保留那些输入类型匹配最准确的。域类型看做和域类型的基本类型相同。如果没有一个操作符能被保留，则保留所有候选。如果只剩下一个候选项，则用之，否则继续下一步。
   c. 遍历所有候选操作符，保留那些需要类型转换时接受(属于输入数据类型的类型范畴的)首选类型位置最多的操作符。如果没有接受首选类型的操作符，则保留所有候选。如果只剩下一个候选项，则用之，否则继续下一步。
   d. 如果有任何输入参数是unknown类型，检查剩余的候选操作符对应参数位置的类型范畴。在每一个能够接受字符串类型范畴的位置使用string类型（这种对字符串的偏爱是合适的，因为unknown文本确实像字符串）。如果所有剩下的候选操作符都接受相同的类型范畴，则选择该类型范畴，否则抛出一个错误（因为在没有更多线索的条件下无法作出正确的选择）。现在抛弃不接受选定的类型范畴的候选操作符，如果任意候选操作符在某个给定的参数位置接受一个首选类型，则抛弃那些在该参数位置接受非首选类型的候选操作符。如果没有一个操作符能被保留，则保留所有候选。如果只剩下一个候选项，则用之，否则继续下一步。
   e. 如果同时有unknown和已知类型的参数，并且所有已知类型的参数都是相同的类型，那么假设unknown参数也是那种类型，并检查哪个候选操作符在unknown参数位置接受那个类型。如果只有一个操作符符合，那么使用它。否则，产生一个错误。

示例

示例1：阶乘操作符类型解析。在系统表中里只有一个阶乘操作符（后缀!），它以bigint作为参数。扫描器给下面查询表达式的参数赋予bigint的初始类型：

```
gaussdb=# SELECT 40 ! AS "40 factorial";

                  40 factorial
-------------------------------------------------
 815915283247897734345611269596115894272000000000
(1 row)
```

分析器对参数做类型转换，查询等效于：

```
gaussdb=# SELECT CAST(40 AS bigint) ! AS "40 factorial";
```

示例2：字符串连接操作符类型分析。一种字符串风格的语法既可以用于字符串也可以用于复杂的扩展类型。未声明类型的字符串将被所有可能的候选操作符匹配。有一个未声明的参数的例子：

```
gaussdb=# SELECT text 'abc' || 'def' AS "text and unknown";
 text and unknown
------------------
 abcdef
(1 row)
```


本例中分析器寻找两个参数都是text的操作符。确实有这样的操作符，两个参数都是text类型。

下面是连接两个未声明类型的值：

```
gaussdb=# SELECT 'abc' || 'def' AS "unspecified";
 unspecified
-------------
 abcdef
(1 row)
```

说明

因为查询中没有声明任何类型，所以本例中对类型没有任何初始提示。因此，分析器查找所有候选操作符，发现既存在接受字符串类型范畴的操作符也存在接受位串类型范畴的操作符。因为字符串类型范畴是首选，所以选择字符串类型范畴的首选类型text作为解析未知类型文本的声明类型。

示例3：绝对值和取反操作符类型分析。GaussDB操作符表里面有几条记录对应于前缀操作符@，它们都用于为各种数值类型实现绝对值操作。其中之一用于float8类型，它是数值类型范畴中的首选类型。因此，在面对unknown输入的时候， GaussDB会使用该类型：

```
gaussdb=# SELECT @ '-4.5' AS "abs";
 abs
-----
 4.5
(1 row)
```

系统在应用选定的操作符之前隐式的转换unknown类型的文字为float8类型。

示例4：数组包含操作符类型分析。这里是解决一个操作符带有一个已知和一个未知类型输入的例子：

```
gaussdb=# SELECT array[1,2] <@ '{1,2,3}' as "is subset";
 is subset
-----------
 t
(1 row)
```

说明

GaussDB操作符表有几条记录对应于中缀操作符<@，但是只有两个可以在左侧接受一个整数数组的操作符是数组包含(anyarray <@ anyarray) 和范围包含(anyelement <@ anyrange)的。因为没有多态的伪类型(参阅**伪类型**)是首选的，所以解析器不能解决这个基础上的歧义。然而最后一个解析规则告诉用户，假设未知类型的文字是和另外一个输入相同的类型，那就是整数数组。现在只有两个操作符中的一个可以匹配，所以选择数组包含。（如果用户选择了范围包含，用户将得到一个错误，因为字符串没有正确的格式成为范围的文字。）

7.8.3 函数

函数类型解析

1. 从系统表pg_proc中选择所有可能被选到的函数。如果使用了一个不带模式修饰的函数名称，那么认为该函数是那些在当前搜索路径中的函数。如果给出一个带修饰的函数名，那么只考虑指定模式中的函数。

   如果搜索路径中找到了多个不同参数类型的函数。将从中选择一个合适的函数。
2. 查找和输入参数类型完全匹配的函数。如果找到一个，则用之。如果输入的实参类型都是unknown类型，则不会找到匹配的函数。
3. 如果未找到完全匹配，请查看该函数是否为一个特殊的类型转换函数。

4. 寻找最优匹配。
   a. 抛弃那些输入类型不匹配并且也不能隐式转换成匹配的候选函数。unknown文本在这种情况下可以转换成任何东西。如果只剩下一个候选项，则用之，否则继续下一步。
   b. 遍历所有候选函数，保留那些输入类型匹配最准确的。此时，域被看作和它们的基本类型相同。如果没有一个函数能准确匹配，则保留所有候选。如果只剩下一个候选项，则用之，否则继续下一步。
   c. 遍历所有候选函数，保留那些需要类型转换时接受首选类型位置最多的函数。如果没有接受首选类型的函数，则保留所有候选。如果只剩下一个候选项，则用之，否则继续下一步。
   d. 如果有任何输入参数是unknown类型，检查剩余的候选函数对应参数位置的类型范畴。在每一个能够接受字符串类型范畴的位置使用string类型（这种对字符串的偏爱是合适的，因为unknown文本确实像字符串）。另外，如果所有剩下的候选函数都接受相同的类型范畴，则选择该类型范畴，否则抛出一个错误（因为在没有更多线索的条件下无法做出正确的选择）。现在抛弃不接受选定的类型范畴的候选函数，然后，如果任意候选函数在那个范畴接受一个首选类型，则抛弃那些在该参数位置接受非首选类型的候选函数。如果没有一个候选符合这些测试则保留所有候选。如果只有一个候选函数符合，则使用它；否则，继续下一步。
   e. 如果同时有unknown和已知类型的参数，并且所有已知类型的参数有相同的类型，假设unknown参数也是这种类型，检查哪个候选函数可以在unknown参数位置接受这种类型。如果正好一个候选符合，那么使用它。否则，产生一个错误。

示例

示例1：圆整函数参数类型解析。只有一个round函数有两个参数（第一个是numeric，第二个是integer）。所以下面的查询自动把第一个类型为integer的参数转换成numeric类型。

```
gaussdb=# SELECT round(4, 4);
 round
--------
 4.0000
(1 row)
```

实际上它被分析器转换成：

```
gaussdb=# SELECT round(CAST (4 AS numeric), 4);
```

因为带小数点的数值常量初始时被赋予numeric类型，因此下面的查询将不需要类型转换，并且可能会略微高效一些：

```
gaussdb=# SELECT round(4.0, 4);
```

示例2：子字符串函数类型解析。有好几个substr函数，其中一个接受text和integer类型。如果用一个未声明类型的字符串常量调用它，系统将选择接受string类型范畴的首选类型（也就是text类型）的候选函数。

```
gaussdb=# SELECT substr('1234', 3);
 substr
--------
     34
(1 row)
```

如果该字符串声明为varchar类型，就像从表中取出来的数据一样，分析器将试着将其转换成text类型：


```
gaussdb=# SELECT substr(varchar '1234', 3);
 substr
--------
     34
(1 row)
```

被分析器转换后实际上变成：

```
gaussdb=# SELECT substr(CAST (varchar '1234' AS text), 3);
```

说明

分析器从pg_cast表中了解到text和varchar是二进制兼容的，意思是说一个可以传递给接受另一个的函数而不需要做任何物理转换。因此，在这种情况下，实际上没有做任何类型转换。

而且，如果以integer为参数调用函数，分析器将试图将其转换成text类型：

```
gaussdb=# SELECT substr(1234, 3);
substr
--------
 34
(1 row)
```

被分析器转换后实际上变成：

```
gaussdb=# SELECT substr(CAST (1234 AS text), 3);
 substr
--------
     34
(1 row)
```

7.8.4 值存储

值存储数据类型解析

1. 查找与目标字段准确的匹配。
2. 试着将表达式直接转换成目标类型。如果已知这两种类型之间存在一个已注册的转换函数，那么直接调用该转换函数即可。如果表达式是一个未知类型文本，该文本字符串的内容将交给目标类型的输入转换过程。
3. 检查目标类型是否有长度转换。长度转换是一个从某类型到自身的转换。如果在pg_cast表里面找到一个，那么在存储到目标字段之前先在表达式上应用。这样的转换函数总是接受一个额外的类型为integer的参数，它接收目标字段的atttypmod值（实际上是其声明长度，atttypmod的解释随不同的数据类型而不同），并且它可能接受一个Boolean类型的第三个参数，表示转换是显式的还是隐式的。转换函数负责施加那些长度相关的语义，比如长度检查或者截断。

示例

character存储类型转换。对一个目标列定义为character(20)的语句，下面的语句显示存储值的长度正确：

```
gaussdb=# CREATE SCHEMA tpcds;
gaussdb=# CREATE TABLE tpcds.value_storage_t1 (
   VS_COL1 CHARACTER(20)
);
gaussdb=# INSERT INTO tpcds.value_storage_t1 VALUES('abcdef');
gaussdb=# SELECT VS_COL1, octet_length(VS_COL1) FROM tpcds.value_storage_t1;
       vs_col1        | octet_length
----------------------+--------------
 abcdef               |           20
(1 row)
)
```



```
gaussdb=# DROP TABLE tpcds.value_storage_t1;
gaussdb=# DROP SCHEMA tpcds;
```

说明

两个unknown文本缺省解析成text，这样就允许||操作符解析成text连接。然后操作符的text结果转换成bpchar("空白填充的字符型"， character类型内部名称)以匹配目标字段类型。从text到bpchar的转换是二进制兼容的，这样的转换是隐含的并且实际上不做任何函数调用。在系统表里找到长度转换函数bpchar(bpchar, integer, Boolean) 并且应用于该操作符的结果和存储的字段长。这个类型相关的函数执行所需的长度检查和额外的空白填充。

7.8.5 UNION，CASE 和相关构造

SQL UNION构造必须把那些可能不太相似的类型匹配起来成为一个结果集。解析算法分别应用于联合查询的每个输出字段。INTERSECT和EXCEPT构造对不相同的类型使用和UNION相同的算法进行解析。CASE、ARRAY、VALUES、GREATEST和LEAST构造也使用同样的算法匹配它的部件表达式并且选择一个结果数据类型。

UNION，CASE 和相关构造解析

- 如果所有输入都是相同的类型，并且不是unknown类型，那么解析成这种类型。
- 如果所有输入都是unknown类型则解析成text类型（字符串类型范畴的首选类型）。否则，忽略unknown输入（例外：Union操作会将一组两个unkown类型解析成text类型，然后继续与其他组进行类型匹配选择）。
- 如果输入不属于同一个类型范畴，则失败。（unknown类型除外）
- 如果输入类型是同一个类型范畴，则选择该类型范畴的首选类型。（例外：Union操作会选择第一个分支的类型作为所选类型。）

  说明

  系统表pg_type中typcategory表示数据类型范畴，typispreferred表示是否是typcategory分类中的首选类型。
- 把所有输入转换为所选的类型（对于字符串保持原有长度）。如果从给定的输入到所选的类型没有隐式转换则失败。
- 若输入中含json、txid_snapshot、sys_refcursor或几何类型，则不能进行union。

对于 case 和 coalesce，在 TD 兼容模式下的处理

- 如果所有输入都是相同的类型，并且不是unknown类型，那么解析成这种类型。
- 如果所有输入都是unknown类型则解析成text类型。
- 如果输入字符串（包括unknown，unknown当text来处理）和数字类型，那么解析成字符串类型，如果是其他不同的类型范畴，则报错。
- 如果输入类型是同一个类型范畴，则选择该类型的优先级较高的类型。
- 把所有输入转换为所选的类型。如果从给定的输入到所选的类型没有隐式转换则失败。

对于 case，在 A 兼容模式下的处理

decode(expr, search1, result1, search2, result2, ..., defresult); 在设置参数sql_beta_feature = a_style_coerce时，按A兼容模式下的处理，将整个表达式最终的返回值类型定为result1的数据类型，或者与result1同类型范畴的更高精度的数据类型。（例如，numeric与int同属数值类型范畴，但numeric比int精度要高，具有更高优先级）。对于CASE WHEN，A兼容性下与默认行为相同。

- 如果所有输入都是相同的类型，并且不是unknown类型，那么解析成这种类型。否则，进入后续步骤。
- 将result1的数据类型置为最终的返回值类型preferType，其所属类型范畴为preferCategory。
- 依次考虑result2、result3直至defresult的数据类型。如果其类型范畴也是preferCategory，即与result1具有相同的类型范畴，则判断其精度（优先级）是否高于preferType，如果高于，则将preferType更新为更高精度的数据类型；如果其类型范畴不是preferCategory，则判断其数据类型是否可以隐式转换为preferType，不可以则报错。
- 将最终preferType记录的数据类型作为整个表达式最终的返回值类型；表达式的结果向此类型进行隐式转换。

注1：

为了兼容一种特殊情况，即表示了超大数字的字符类型向数值类型转换的情况，例如select decode(1, 2, 2, '53465465676465454657567678676')，大数超过了bigint、double等的表示范围。所以，当result1的类型范畴为数值类型，且不满足上述"所有输入都是相同的类型"条件时，将返回值的类型直接置为numeric，以兼容此种特殊情况。

注2：

数值类型的优先级排序：numeric>float8>float4>int8>int4>int2>int1

字符类型的优先级排序：text>varchar=nvarchar2>bpchar>char

日期类型的优先级排序：
timestamptz>timestamp>smalldatetime>date>abstime>timetz>time

日期跨度类型的优先级排序：interval>tinterval>reltime

注3：

ORA兼容模式，当参数set sql_beta_feature的值设置为'a_style_coerce'时，所支持的隐式类型转换见下图，\代表不需要转换，yes表示支持，空白表示不支持：

	bool	int1	int2	int4	int8	float4	float8	numeric	money	char	bpchar	varchar2	nvarchar2	text/clob	raw	blob	date	time	timetz	timestamp	timestamptz	smalldatetime	interval	reltime	abstime
bool	\																								
int1		\	yes	yes	yes	yes	yes	yes		yes	yes	yes	yes	yes											
int2		yes	\	yes	yes	yes	yes	yes		yes	yes	yes	yes	yes											
int4		yes	yes	\	yes	yes	yes	yes		yes	yes	yes	yes	yes											
int8		yes	yes	yes	\	yes	yes	yes		yes	yes	yes	yes	yes											
float4		yes	yes	yes	yes	\	yes	yes		yes	yes	yes	yes	yes											
float8		yes	yes	yes	yes	yes	\	yes		yes	yes	yes	yes	yes											
numeric		yes	yes	yes	yes	yes	yes	\		yes	yes	yes	yes	yes											
money									\																
char		yes	yes	yes	yes	yes	yes	yes		\	yes	yes	yes	yes											
bpchar		yes	yes	yes	yes	yes	yes	yes		yes	\	yes	yes	yes											
varchar2		yes	yes	yes	yes	yes	yes	yes		yes	yes	\	yes	yes	yes										
nvarchar2		yes	yes	yes	yes	yes	yes	yes		yes	yes	yes	\	yes											
text/clob		yes	yes	yes	yes	yes	yes	yes		yes	yes	yes	yes	\											
raw												yes		yes	\	yes									
blob															yes	\									
date												yes	yes	yes			\			yes	yes	yes			yes
time														yes				\	yes						
timetz														yes				yes	\						
timestamp												yes	yes	yes			yes			\	yes	yes			yes
timestamptz														yes			yes			yes	\	yes			yes
smalldatetime												yes		yes			yes			yes	yes	\			yes
interval												yes	yes	yes									\	yes	
reltime														yes									yes	\	
abstime														yes			yes			yes	yes	yes			\

示例

示例1：Union中的待定类型解析。这里，unknown类型文本'b'将被解析成text类型。

```
gaussdb=# SELECT text 'a' AS "text" UNION SELECT 'b';
 text
------
 a
 b
(2 rows)
```

示例2：简单Union中的类型解析。文本1.2的类型为numeric，而且integer类型的1可以隐含地转换为numeric，因此使用这个类型。

```
gaussdb=# SELECT 1.2 AS "numeric" UNION SELECT 1;
 numeric
---------
       1
     1.2
(2 rows)
```

示例3：转置Union中的类型解析。这里，因为类型real不能被隐含转换成integer，但是integer可以隐含转换成real，那么联合的结果类型将是real。

```
gaussdb=# SELECT 1 AS "real" UNION SELECT CAST('2.2' AS REAL);
 real
------
    1
  2.2
(2 rows)
```

示例4：TD模式下，coalesce参数输入int和varchar类型，那么解析成varchar类型。A模式下会报错。

```
--在A模式下，创建A兼容模式的数据库a_1。
gaussdb=# CREATE DATABASE a_1 dbcompatibility = 'A';

--切换数据库为a_1。
gaussdb=# \c a_1

--创建表t1。
a_1=# CREATE TABLE t1(a int, b varchar(10));

--查看coalesce参数输入int和varchar类型的查询语句的执行计划。
a_1=# EXPLAIN SELECT coalesce(a, b) FROM t1;
ERROR:  COALESCE types integer and character varying cannot be matched
LINE 1: EXPLAIN SELECT coalesce(a, b) FROM t1;
                                ^
CONTEXT:  referenced column: coalesce

--删除表。
a_1=# DROP TABLE t1;

--切换数据库为testdb。
a_1=# \c testdb

--在TD模式下，创建TD兼容模式的数据库td_1。
gaussdb=# CREATE DATABASE td_1 dbcompatibility = 'C';

--切换数据库为td_1。
gaussdb=# \c td_1

--创建表t2。
td_1=# CREATE TABLE t2(a int, b varchar(10));

--查看coalesce参数输入int和varchar类型的查询语句的执行计划。
td_1=# EXPLAIN VERBOSE select coalesce(a, b) from t2;
                                  QUERY PLAN
-------------------------------------------------------------------------------
```



```
 Data Node Scan  (cost=0.00..0.00 rows=0 width=0)
   Output: (COALESCE((t2.a)::character varying, t2.b))
   Node/s: All dbnodes
   Remote query: SELECT COALESCE(a::character varying, b) AS "coalesce" FROM public.t2
(4 rows)

--删除表。
td_1=# DROP TABLE t2;

--切换数据库为testdb。
td_1=# \c testdb

--删除A和TD模式的数据库。
gaussdb=# DROP DATABASE a_1;
gaussdb=# DROP DATABASE td_1;
```

示例5：ORA模式下，将整个表达式最终的返回值类型定为result1的数据类型，或者与result1同类型范畴的更高精度的数据类型。

```
--在ORA模式下，创建ORA兼容模式的数据库ora_1。
gaussdb=# CREATE DATABASE ora_1 dbcompatibility = 'A';

--切换数据库为ora_1。
gaussdb=# \c ora_1

--开启Decode兼容性参数。
set sql_beta_feature='a_style_coerce';

--创建表t1。
ora_1=# CREATE TABLE t1(c_int int, c_float8 float8, c_char char(10), c_text text, c_date date);

--插入数据。
ora_1=# INSERT INTO t1 VALUES(1, 2, '3', '4', date '12-10-2010');

--result1类型为char，defresult类型为text，text精度更高，返回值的类型由char更新为text。
ora_1=# SELECT decode(1, 2, c_char, c_text) AS result, pg_typeof(result) FROM t1;
 result | pg_typeof
--------+-----------
 4      | text
(1 row)

--result1类型为int，属于数值类型范畴，返回值的类型置为numeric。
ora_1=# SELECT decode(1, 2, c_int, c_float8) AS result, pg_typeof(result) FROM t1;
 result | pg_typeof
--------+-----------
      2 | numeric
(1 row)

--不存在defresult数据类型向result1数据类型之间的隐式转换，报错处理。
ora_1=# SELECT decode(1, 2, c_int, c_date) FROM t1;
ERROR:  CASE types integer and timestamp without time zone cannot be matched
LINE 1: SELECT decode(1, 2, c_int, c_date) FROM t1;
                                   ^
CONTEXT:  referenced column: c_date

--关闭Decode兼容性参数。
set sql_beta_feature='none';

--删除表。
ora_1=# DROP TABLE t1;
DROP TABLE

--切换数据库为testdb。
ora_1=# \c testdb

--删除ORA模式的数据库。
gaussdb=# DROP DATABASE ora_1;
DROP DATABASE
```


示例6：Union操作会将一组两个unkown类型解析成text类型，然后继续与其他组进行类型匹配选择。

```
--前两个NULL为unkown类型，解析为text类型，然后将此text类型与第三个元素varchar2类型进行匹配选择，最终选择text类型。
gaussdb=# SELECT "text", pg_typeof("text") as type from (SELECT NULL AS "text" UNION ALL SELECT NULL AS "text" UNION ALL SELECT 'a'::varchar2 as "text");
 text | type
------+------
      | text
      | text
 a    | text
(3 rows)
```

7.9 系统操作

GaussDB通过SQL语句执行不同的系统操作，比如：设置变量，显示执行计划和垃圾收集等操作。

设置变量

设置会话或事务中需要使用的各种参数，请参考**SET**。

显示执行计划

显示GaussDB为SQL语句规划的执行计划，请参考**EXPLAIN**。

事务日志检查点

预写式日志（WAL）缺省时在事务日志中每隔一段时间放置一个检查点。CHECKPOINT强制立即进行检查，而不是等到下一次调度时的检查点。请参考**CHECKPOINT**。

垃圾收集

进行垃圾收集以及可选择的对数据库进行分析。请参考**VACUUM**。

收集统计信息

收集与数据库中表内容相关的统计信息。请参考**ANALYZE | ANALYSE**。

设置当前事务的约束检查模式

设置当前事务里的约束检查的特性。请参考**SET CONSTRAINTS**。

关闭当前数据库节点

关闭当前数据库节点，请参考**SHUTDOWN**。

7.10 事务控制

事务是用户定义的一个数据库操作序列，这些操作要么全做要么全不做，是一个不可分割的工作单位。

启动事务

GaussDB通过START TRANSACTION和BEGIN语法启动事务，请参考**START TRANSACTION**和**BEGIN**。

设置事务

GaussDB通过SET TRANSACTION或者SET LOCAL TRANSACTION语法设置事务，请参考**SET TRANSACTION**。

提交事务

GaussDB通过COMMIT或者END可完成提交事务的功能，即提交事务的所有操作，请参考**COMMIT | END**。

回滚事务

回滚是在事务运行的过程中发生了某种故障，事务不能继续执行，系统将事务中对数据库的所有已完成的操作全部撤销。请参考**ROLLBACK**。

说明

数据库中收到的一次执行请求（不在事务块中），如果含有多条语句，将会被打包成一个事务，如果其中有一个语句失败，那么整个请求都将会被回滚。

7.11 DDL 语法一览表

DDL（Data Definition Language数据定义语言），用于定义或修改数据库中的对象。如：表、索引、视图等。

说明

GaussDB不支持数据库主节点不完整时进行DDL操作。例如：数据库中有1个数据库主节点故障时执行新建数据库、表等操作都会失败。

定义客户端加密主密钥

客户端加密主密钥主要用于密态数据库特性，用来加密列加密密钥(cek)。客户端加密主密钥定义主要包括创建客户端加密主密钥以及删除客户端加密主密钥。所涉及的SQL语句，请参考**表7-118**。

表 7-118 客户端加密主密钥定义相关 SQL

功能	相关SQL
创建客户端加密主密钥	CREATE CLIENT MASTER KEY
删除客户端加密主密钥	DROP CLIENT MASTER KEY

定义列加密密钥

列加密密钥主要用于密态数据库特性中，用来加密数据。列加密密钥定义主要包括创建列加密密钥、轮转加密列加密密钥的客户端主密钥以及删除列加密密钥。所涉及的SQL语句，请参考表7-118。

表 7-119 列加密密钥定义相关 SQL

功能	相关SQL
创建列加密密钥	CREATE COLUMN ENCRYPTION KEY
修改列加密密钥指定的客户端主密钥	8.14.191-ALTER COLUMN ENCRYPTION KEY
删除列加密密钥	DROP COLUMN ENCRYPTION KEY

定义数据库

数据库是组织、存储和管理数据的仓库，而数据库定义主要包括：创建数据库、修改数据库属性，以及删除数据库。所涉及的SQL语句，请参考表7-120。

表 7-120 数据库定义相关 SQL

功能	相关SQL
创建数据库	CREATE DATABASE
修改数据库属性	ALTER DATABASE
删除数据库	DROP DATABASE

定义模式

模式是一组数据库对象的集合，主要用于控制对数据库对象的访问。所涉及的SQL语句，请参考表7-121。

表 7-121 模式定义相关 SQL

功能	相关SQL
创建模式	CREATE SCHEMA
修改模式属性	ALTER SCHEMA
删除模式	DROP SCHEMA

定义表空间

表空间用于管理数据对象，与磁盘上的一个目录对应。所涉及的SQL语句，请参考表7-122。

表 7-122 表空间定义相关 SQL

功能	相关SQL
创建表空间	CREATE TABLESPACE
修改表空间属性	ALTER TABLESPACE
删除表空间	DROP TABLESPACE

定义表

表是数据库中的一种特殊数据结构，用于存储数据对象以及对象之间的关系。所涉及的SQL语句，请参考表7-123。

表 7-123 表定义相关 SQL

功能	相关SQL
创建表	CREATE TABLE
修改表属性	ALTER TABLE
删除表	DROP TABLE

定义分区表

分区表是一种逻辑表，数据是由普通表存储的，主要用于提升查询性能。所涉及的SQL语句，请参考表7-124。

表 7-124 分区表定义相关 SQL

功能	相关SQL
创建分区表	CREATE TABLE PARTITION
创建分区	ALTER TABLE PARTITION
修改分区表属性	ALTER TABLE PARTITION
删除分区	ALTER TABLE PARTITION
删除分区表	DROP TABLE

定义索引

索引是对数据库表中一列或多列的值进行排序的一种结构，使用索引可快速访问数据库表中的特定信息。所涉及的SQL语句，请参考表7-125。

表 7-125 索引定义相关 SQL

功能	相关SQL
创建索引	CREATE INDEX
修改索引属性	ALTER INDEX
删除索引	DROP INDEX
重建索引	REINDEX

定义存储过程

存储过程是一组为了完成特定功能的SQL语句集，经编译后存储在数据库中，用户通过指定存储过程的名称并给出参数（如果该存储过程带有参数）来执行它。所涉及的SQL语句，请参考表7-126。

表 7-126 存储过程定义相关 SQL

功能	相关SQL
创建存储过程	CREATE PROCEDURE
删除存储过程	DROP PROCEDURE
重编译存储过程	ALTER PROCEDURE

定义函数

在GaussDB中，它和存储过程类似，也是一组SQL语句集，使用上没有差别。所涉及的SQL语句，请参考表7-127。

表 7-127 函数定义相关 SQL

功能	相关SQL
创建函数	CREATE FUNCTION
修改函数属性	ALTER FUNCTION
删除函数	DROP FUNCTION

定义包

包由包头（package specification）和包体(package body)组成，用来分类管理存储过程和函数，类似于Java、C++等语言中的类。

表 7-128 包定义相关 SQL

功能	相关SQL
创建包	CREATE PACKAGE
删除包	DROP PACKAGE
修改包属性	ALTER PACKAGE

定义视图

视图是从一个或几个基本表中导出的虚表，可用于控制用户对数据访问，请参考表7-129。

表 7-129 视图定义相关 SQL

功能	相关SQL
创建视图	CREATE VIEW
删除视图	DROP VIEW

定义游标

为了处理SQL语句，存储过程进程分配一段内存区域来保存上下文联系。游标是指向上下文区域的句柄或指针。借助游标，存储过程可以控制上下文区域的变化，请参考表7-130。

表 7-130 游标定义相关 SQL

功能	相关SQL
创建游标	CURSOR
移动游标	MOVE
从游标中提取数据	FETCH
关闭游标	CLOSE

定义聚合函数

表 7-131 聚合函数定义相关 SQL

功能	相关SQL
创建一个新的聚合函数	CREATE AGGREGATE
修改聚合函数	ALTER AGGREGATE

功能	相关SQL
删除聚合函数	DROP AGGREGATE

定义数据类型转换

表 7-132 数据类型定义相关 SQL

功能	相关SQL
创建一个新的用户自定义数据类型转换	CREATE CAST
删除用户自定义数据类型转换	DROP CAST

定义插件扩展

须知

扩展功能为内部使用功能，不建议用户使用。

表 7-133 插件扩展定义相关 SQL

功能	相关SQL
创建一个新的插件扩展	CREATE EXTENSION
修改插件扩展	ALTER EXTENSION
删除插件扩展	DROP EXTENSION

定义操作符

表 7-134 操作符定义相关 SQL

功能	相关SQL
创建一个新的操作符	CREATE OPERATOR
修改操作符	ALTER OPERATOR
删除操作符	DROP OPERATOR

定义数据类型

表 7-135 数据类型定义相关 SQL

功能	相关SQL
创建一个新的数据类型	**CREATE TYPE**
修改数据类型	**ALTER TYPE**
删除数据类型	**DROP TYPE**

定义定时任务

表 7-136 定时任务定义相关 SQL

功能	相关SQL
创建一个新的定时任务	**CREATE EVENT**
修改定时任务	**ALTER EVENT**
删除定时任务	**DROP EVENT**

定义 DATABASE LINK 对象

DATABASE LINK是可以操作远程数据库对象，所涉及的SQL语句，请参考**表7-137**。

表 7-137 DATABASE LINK 对象相关 SQL

功能	相关SQL
创建一个新的DATABASE LINK对象	**CREATE DATABASE LINK**
修改DATABASE LINK对象	**ALTER DATABASE LINK**
删除DATABASE LINK对象	**DROP DATABASE LINK**

定义外部数据封装器

表 7-138 外部数据封装器相关 SQL

功能	相关SQL
创建外部数据封装器	**CREATE FOREIGN DATA WRAPPER**
修改外部数据封装器	**ALTER FOREIGN DATA WRAPPER**
删除外部数据封装器	**DROP FOREIGN DATA WRAPPER**

7.12 DML 语法一览表

DML（Data Manipulation Language数据操作语言），用于对数据库表中的数据进行操作。如：插入、更新、查询、删除。

插入数据

插入数据是往数据库表中添加一条或多条记录，请参考**INSERT**。

修改数据

修改数据是修改数据库表中的一条或多条记录，请参考**UPDATE**。

查询数据

数据库查询语句SELECT是用于在数据库中检索适合条件的信息，请参考**SELECT**。

删除数据

GaussDB提供了两种删除表数据的语句：删除表中指定条件的数据，请参考**DELETE**；或删除表的所有数据，请参考**TRUNCATE**。

TRUNCATE快速地从表中删除所有行，它和在每个表上进行无条件的DELETE有同样的效果，不过因为它不做表扫描，因而快得多。在大表上最有用。

拷贝数据

GaussDB提供了在表和文件之间拷贝数据的语句，请参考**COPY**。

锁定表

GaussDB提供了多种锁模式用于控制对表中数据的并发访问，请参考**LOCK**。

调用函数

GaussDB提供了三个用于调用函数的语句，它们在语法结构上没有差别，请参考**CALL**。

操作会话

用户与数据库之间建立的连接称为会话，请参考**表7-139**。

表 7-139 会话相关 SQL

功能	相关SQL
修改会话	**ALTER SESSION**
结束会话	**ALTER SYSTEM KILL SESSION**

7.13 DCL 语法一览表

DCL（Data Control Language数据控制语言），是用来创建用户角色、设置或更改数据库用户或角色权限的语句。

定义角色

角色是用来管理权限的，从数据库安全的角度考虑，可以把所有的管理和操作权限划分到不同的角色上。所涉及的SQL语句，请参考**表7-140**。

表 7-140 角色定义相关 SQL

功能	相关SQL
创建角色	**CREATE ROLE**
修改角色属性	**ALTER ROLE**
删除角色	**DROP ROLE**

定义用户

用户是用来登录数据库的，通过对用户赋予不同的权限，可以方便地管理用户对数据库的访问及操作。所涉及的SQL语句，请参考**表7-141**。

表 7-141 用户定义相关 SQL

功能	相关SQL
创建用户	**CREATE USER**
修改用户属性	**ALTER USER**
删除用户	**DROP USER**

授权

GaussDB提供了针对数据对象和角色授权的语句，请参考**GRANT**。

收回权限

GaussDB提供了收回权限的语句，请参考**REVOKE**。

设置默认权限

GaussDB允许设置应用于将来创建的对象的权限，请参考**ALTER DEFAULT PRIVILEGES**。

关闭当前节点

GaussDB支持使用shutdown命令关闭当前数据库节点，请参考**SHUTDOWN**。

7.14 SQL 语法

7.14.1 SQL 语法格式说明

表 7-142 SQL 语法格式说明

格式	意义
[ ]	表示用“[ ]”括起来的部分是可选的。
...	表示前面的元素可重复出现。
[x | y | ...]	表示从两个或多个选项中选取一个或者不选。
{ x | y | ... }	表示从两个或多个选项中选取一个。
[x | y | ...] [...]	表示可选多个参数或者不选，如果选择多个参数，则参数之间用空格分隔。
[x | y | ...] [,...]	表示可选多个参数或者不选，如果选择多个参数，则参数之间用逗号分隔。
{ x | y | ... } [...]	表示可选多个参数，至少选一个，如果选择多个参数，则参数之间以空格分隔。
{ x | y | ... } [,...]	表示可选多个参数，至少选一个，如果选择多个参数，则参数之间用逗号分隔。

7.14.2 ABORT

功能描述

回滚当前事务并且撤销所有当前事务中所做的更改。

作用等同于**ROLLBACK**，早期SQL有用ABORT，现在推荐使用ROLLBACK。

注意事项

在事务外部执行ABORT语句不会影响事务的执行，但是会抛出一个NOTICE信息。

语法格式

```
ABORT [ WORK | TRANSACTION ] ;
```

参数说明

- **WORK | TRANSACTION**
  可选关键字，除了增加可读性没有其他任何作用。

示例

```
--创建表customer_demographics_t1。
gaussdb=# CREATE TABLE customer_demographics_t1
(
    CD_DEMO_SK                INTEGER               NOT NULL,
    CD_GENDER                 CHAR(1)                       ,
    CD_MARITAL_STATUS         CHAR(1)                       ,
    CD_EDUCATION_STATUS       CHAR(20)                      ,
    CD_PURCHASE_ESTIMATE      INTEGER                       ,
    CD_CREDIT_RATING          CHAR(10)                      ,
    CD_DEP_COUNT              INTEGER                       ,
    CD_DEP_EMPLOYED_COUNT     INTEGER                       ,
    CD_DEP_COLLEGE_COUNT      INTEGER
)
;

--插入记录。
gaussdb=# INSERT INTO customer_demographics_t1 VALUES(1920801,'M', 'U', 'DOCTOR DEGREE', 200,
'GOOD', 1, 0,0);

--开启事务。
gaussdb=# START TRANSACTION;

--更新字段值。
gaussdb=# UPDATE customer_demographics_t1 SET cd_education_status= 'Unknown';

--终止事务，上面所执行的更新会被撤销掉。
gaussdb=# ABORT;

--查询数据。
gaussdb=# SELECT * FROM customer_demographics_t1 WHERE cd_demo_sk = 1920801;
cd_demo_sk | cd_gender | cd_marital_status | cd_education_status  | cd_purchase_estimate | cd_credit_rating
| cd_dep_count | cd_dep_employed_count | cd_dep_college_count
------------+-----------+-------------------+----------------------+----------------------+------------------
+--------------+-----------------------+----------------------
   1920801 | M         | U                 | DOCTOR DEGREE        |                  200 | GOOD             |            1
|                  0 |                     0
(1 row)

--删除表。
gaussdb=# DROP TABLE customer_demographics_t1;
```

相关链接

SET TRANSACTION，**COMMIT | END**，**ROLLBACK**

7.14.3 ALTER AGGREGATE

功能描述

修改一个聚合函数的定义。包括名称、所有者和模式。

注意事项

要使用 ALTER AGGREGATE ，你必须是该聚合函数的所有者。 要改变一个聚合函数的模式，你必须在新模式上有 CREATE 权限。 要改变所有者，你必须是新所有角色的一个直接或间接成员，并且该角色必须在聚合函数的模式上有 CREATE 权限。（这些限制强制了修改该所有者不会做任何通过删除和重建聚合函数不能做的事情。不过，具有SYSADMIN权限用户可以用任何方法任意更改聚合函数的所属关系）。

语法格式

```
ALTER AGGREGATE name ( argtype [ , ... ] ) RENAME TO new_name
ALTER AGGREGATE name ( argtype [ , ... ] ) OWNER TO new_owner
ALTER AGGREGATE name ( argtype [ , ... ] ) SET SCHEMA new_schema
```

参数说明

- **name**

  现有的聚合函数的名称(可以有模式修饰)。
- **argtype**

  聚合函数操作的输入数据类型。要引用一个零参数聚合函数，可以写入*代替输入数据类型列表。
- **new_name**

  聚合函数的新名字。
- **new_owner**

  聚合函数的新所有者。
- **new_schema**

  聚合函数的新模式。

示例

把一个接受integer 类型参数的聚合函数myavg重命名为 my_average：

```
ALTER AGGREGATE myavg(integer) RENAME TO my_average;
```

把一个接受integer 类型参数的聚合函数myavg的所有者改为joe：

```
ALTER AGGREGATE myavg(integer) OWNER TO joe;
```

把一个接受integer 类型参数的聚合函数myavg移动到模式myschema里：

```
ALTER AGGREGATE myavg(integer) SET SCHEMA myschema;
```

兼容性

SQL标准里没有ALTER AGGREGATE语句。

相关链接

CREATE AGGREGATE，**DROP AGGREGATE**

7.14.4 ALTER AUDIT POLICY

功能描述

修改统一审计策略。

注意事项

- 只有poladmin，sysadmin或初始用户才能进行此操作。
- 需要打开enable_security_policy开关统一审计策略才可以生效。

语法格式

添加/删除审计策略中的操作类型。

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name { ADD | REMOVE } { [ privilege_audit_clause ]
[ access_audit_clause ] };
```

修改审计策略中的过滤条件。

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name MODIFY ( filter_group_clause );
```

将审计策略中的过滤条件删除。

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name DROP FILTER;
```

修改审计策略描述。

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name COMMENTS policy_comments;
```

打开或者关闭审计策略。

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name { ENABLE | DISABLE };
```

- privilege_audit_clause：

  审计策略中具体的DDL操作类型及目标资源标签。
  ```
  PRIVILEGES ({ DDL | ALL } [ ON LABEL ( resource_label_name [, ... ] ) ])
  ```
- access_audit_clause：

  审计策略中具体的DML操作类型及目标资源标签。
  ```
  ACCESS ({ DML | ALL } [ ON LABEL ( resource_label_name [, ... ] ) ])
  ```
- filter_group_clause：

  审计策略中的过滤条件。
  ```
  FILTER ON { filter_type ( filter_value [, ... ] ) } [, ... ]
  ```

参数说明

- **policy_name**

  审计策略名称，需要唯一，不可重复。

  取值范围：字符串，要符合**标识符命名规范**。
- **DDL**

  指的是针对数据库执行如下操作时进行审计，目前支持：CREATE、ALTER、DROP、ANALYZE、COMMENT、GRANT、REVOKE、SET、SHOW。
- **DML**

  指的是针对数据库执行如下操作时进行审计，目前支持：SELECT、COPY、DEALLOCATE、DELETE、EXECUTE、INSERT、PREPARE、REINDEX、TRUNCATE、UPDATE。
- **ALL**

  指的是上述DDL或DML中支持的所有对数据库的操作。当形式为{ DDL | ALL }时，ALL指所有DDL操作；当形式为{ DML | ALL }时，ALL指所有DML操作。
- **filter_type**

  指定审计策略的过滤信息，过滤类型包括：IP、ROLES、APP。
- **filter_value**

  指具体过滤信息内容。
- **policy_comments**

用于记录策略相关的描述信息。

- **ENABLE|DISABLE**

  可以打开或关闭统一审计策略。

示例

请参考CREATE AUDIT POLICY的**示例**。

相关链接

CREATE AUDIT POLICY，**DROP AUDIT POLICY**。

7.14.5 ALTER COLUMN ENCRYPTION KEY

功能描述

CMK密钥轮转，轮换加密COLUMN ENCRYPTION KEY的CLIENT MASTER KEY，对COLUMN ENCRYPTION KEY明文进行重加密。

注意事项

- 本语法属于全密态数据库特有语法，当连接数据库服务器时，需打开全密态数据库的开关，才能使用本语法。
- 本语法只能进行CMK轮转，对列加密密钥明文进行重加密，实际上列加密密钥明文不变，不能修改加密列数据的密文。

语法格式

```
ALTER COLUMN ENCRYPTION KEY column_encryption_key_name WITH VALUES ( CLIENT_MASTER_KEY = client_master_key_name );
```

参数说明

- **column_encryption_key_name**

  该参数作为密钥对象名，在同一命名空间下，需满足命名唯一性约束。

  取值范围：字符串，要符合**标识符命名规范**。

- **client_master_key_name**

  指定用于重加密该CEK的新的CMK，取值为：CMK对象名，该CMK对象由CREATE CLIENT MASTER KEY语法创建。与密钥轮转前指定的客户端主密钥不为同一密钥。

须知

国密算法约束：

由于SM2、SM3、SM4等算法属于中国国家密码标准算法，为规避法律风险，需配套使用。如果轮转CEK前使用的CMK是国密算法，则轮转CEK指定的CMK仍必须使用国密算法。

示例

请参考CREATE COLUMN ENCRYPTION KEY的**8.15.63-示例**。

相关链接

CREATE COLUMN ENCRYPTION KEY，**DROP COLUMN ENCRYPTION KEY**

7.14.6 ALTER EVENT

功能描述

修改已创建的定时任务中的参数。

注意事项

- 定时任务相关操作只有sql_compatibility = 'B'时支持。
- 只有定时任务的所有者有权修改对应的定时任务，系统管理员默认拥有修改所有定时任务的权限。
- 可以通过SHOW EVENTS或在PG_JOB表中查看log_user列来获得job的所有者信息。
- 每次修改定时任务成功后，会更新被修改job的所有者为当前用户，若修改定时任务时指定了definer，则更新为被指定的definer。
- definer选项场景限制与**CREATE EVENT**章节中对definer限制场景一致。

须知

系统管理员修改其他用户创建的定时任务后，被修改定时任务的所有者将切换为系统管理员，待执行语句将使用系统管理员的权限执行。

语法格式

```
ALTER
    [DEFINER = user]
EVENT event_name
    [ON SCHEDULE schedule]
    [ON COMPLETION [NOT] PRESERVE]
    [RENAME TO new_event_name]
    [ENABLE | DISABLE | DISABLE ON SLAVE]
    [COMMENT 'string']
    [DO event_body]
```

参数说明

- DEFINER

  定时任务待执行语句在执行时使用的权限。默认情况下使用当前创建定时任务者的权限，当definer被指定时，使用被指定用户user的用户权限。

  definer参数只有具有sysadmin权限的用户有权指定。

- ON SCHEDULE

  定时任务执行时刻。其中schedule子句与**CREATE EVENT**中schedule一致。

- RENAME TO
  更新定时任务名。
- ON COMPLETION [NOT] PRESERVE
  默认情况下，一旦事务处于完成状态，系统表中就会立刻删除该定时任务。用户可以通过设置ON COMPLETION PRESERVE来覆盖默认行为。
- ENABLE | DISABLE | DISABLE ON SLAVE
  创建定时任务后，定时任务默认处于ENABLE状态，即到规定时间立即执行待执行语句。用户可以使用DISABLE关键字，改变定时任务的活动状态。DISABLE ON SLAVE表现与DISABLE一致。
- COMMENT
  用户可以给定时任务添加注释，注释内容在GS_JOB_ATTRIBUTE表中查看。
- DO
  定时任务待执行语句。

示例

```
--创建一个定时任务。
gaussdb=# CREATE TABLE t_ev(num int);

gaussdb=# CREATE EVENT IF NOT EXISTS event_e1 ON SCHEDULE AT sysdate + interval 5 second + interval 33 minute DISABLE DO insert into t_ev values(0);

--修改定时任务状态和待执行语句。
gaussdb=# ALTER EVENT event_e1 ENABLE DO select 1;

--修改定时任务名。
gaussdb=# ALTER EVENT event_e1 RENAME TO event_ee;

--查询定时任务。
gaussdb=# SHOW EVENTS;

--删除定时任务。
gaussdb=# DROP EVENT event_ee;

--删除表。
gaussdb=# DROP TABLE t_ev;
```

相关链接

CREATE EVENT，**DROP EVENT**，**SHOW EVENTS**

7.14.7 ALTER DATABASE

功能描述

修改数据库的属性，包括它的名称、所有者、连接数限制、对象隔离属性等。

注意事项

- 只有数据库的所有者或者被授予了数据库ALTER权限的用户才能执行ALTER DATABASE命令，系统管理员默认拥有此权限。针对所要修改属性的不同，还有以下权限约束：
  - 修改数据库名称，必须拥有CREATEDB权限。
  - 修改数据库所有者，当前用户必须是该DATABASE的所有者或者系统管理员，必须拥有CREATEDB权限，且该用户是新所有者角色的成员。

  - 修改数据库默认表空间，该用户必须拥有新表空间的CREATE权限。这个语句会从物理上将一个数据库原来缺省表空间上的表和索引移至新的表空间。注意不在缺省表空间的表和索引不受此影响。
- 不能重命名当前使用的数据库，如果需要重新命名，须连接至其他数据库上。

语法格式

- 修改数据库的最大连接数。
```
ALTER DATABASE database_name
    [ WITH ] CONNECTION LIMIT connlimit;
```
- 修改数据库名称。
```
ALTER DATABASE database_name
    RENAME TO new_name;
```
- 修改数据库所有者。
```
ALTER DATABASE database_name
    OWNER TO new_owner;
```
- 修改数据库默认表空间。
```
ALTER DATABASE database_name
    SET TABLESPACE new_tablespace;
```

  说明

  如果该数据库中的某些表或对象已经创建在new_tablespace下，则无法将该数据库的默认表空间修改为new_tablespace，执行会报错。
- 修改数据库指定会话参数值。
```
ALTER DATABASE database_name
    SET configuration_parameter { { TO | = } { value | DEFAULT } | FROM CURRENT };
```
- 数据库配置参数重置。
```
ALTER DATABASE database_name RESET
    { configuration_parameter | ALL };
```
- 修改数据库对象隔离属性。
```
ALTER DATABASE database_name [ WITH ] { ENABLE | DISABLE } PRIVATE OBJECT;
```

  说明

  - 修改数据库的对象隔离属性时须连接至该数据库，否则无法更改。
  - 新创建的数据库，对象隔离属性默认是关闭的。当开启数据库对象隔离属性后，普通用户只能查看有权访问的对象（表、函数、视图、字段等）。对象隔离特性对管理员用户不生效，当开启对象隔离特性后，管理员也可以查看到全量的数据库对象。
- 修改数据库时区。
```
ALTER DATABASE database_name SET DBTIMEZONE = time_zone;
```

参数说明

- **database_name**

  需要修改属性的数据库名称。

  取值范围：字符串，要符合**标识符命名规范**。
- **connlimit**

  数据库可以接收的最大并发连接数（管理员用户连接除外）。

  取值范围：[-1, 2^31-1]的整数，建议填写1~50的整数。-1（缺省）表示没有限制。
- **new_name**

  数据库的新名称。

取值范围：字符串，要符合**标识符命名规范**。

- **new_owner**

  数据库的新所有者。

  取值范围：字符串，有效的用户名。

- **new_tablespace**

  数据库新的默认表空间，该表空间为数据库中已经存在的表空间。默认的表空间为pg_default。

  取值范围：字符串，有效的表空间名。

- **configuration_parameter**
  - **value**

    把指定的数据库会话参数值设置为给定的值。如果value是DEFAULT或者RESET，则在新的会话中使用系统的缺省设置。OFF关闭设置。

    取值范围：字符串，

    - DEFAULT
    - OFF
    - RESET
  - **FROM CURRENT**

    取当前会话中的值设置为configuration_parameter的值。

- **time_zone**

  设置database_name的数据库的时区值，需要有对应的数据库的权限。

  取值范围：字符串
  - 系统支持的时区和其相应的缩写
  - -15:59到+15:00
- **FROM CURRENT**

  根据当前会话连接的数据库设置该参数的值。
- **RESET configuration_parameter**

  重置指定的数据库会话参数值。
- **RESET ALL**

  重置全部的数据库会话参数值。

说明

- 修改数据库默认表空间，会将旧表空间中的所有表和索引转移到新表空间中，该操作不会影响其他非默认表空间中的表和索引。
- 修改的数据库会话参数值，将在下一次会话中生效。

示例

请参考CREATE DATABASE的**示例**。

相关链接

CREATE DATABASE，**DROP DATABASE**

7.14.8 ALTER DATABASE LINK

功能描述

修改DATABASE LINK对象。DATABASE LINK详细说明请见**DATABASE LINK**。

注意事项

目前仅支持修改DATABASE LINK对象的用户名和密码。

语法格式

```
ALTER [ PUBLIC ] DATABASE LINK dblink
 { CONNECT TO user IDENTIFIED BY password };
```

参数说明

- **dblink**

  连接名称。

- **user**

  远端被连接数据库用户名。

- **password**

  远端被连接数据库用户密码。

- **PUBLIC**

  连接类型，不加public默认private。

示例

```
--创建拥有系统管理员权限的用户。
gaussdb=#  CREATE USER user1 WITH SYSADMIN PASSWORD '********';
gaussdb=#  SET ROLE user1 PASSWORD '********';

--创建公共dblink。
gaussdb=# CREATE PUBLIC DATABASE LINK public_dblink CONNECT TO 'user1' IDENTIFIED BY '********'
USING (host '192.168.11.11',port '54399',dbname 'db01');

--创建普通用户。
gaussdb=# CREATE USER user2 PASSWORD '********';

--修改dblink对象信息。
gaussdb=# ALTER PUBLIC DATABASE LINK public_dblink CONNECT TO 'user2' IDENTIFIED BY '********';

--删除公共dblink。
gaussdb=# DROP PUBLIC DATABASE LINK public_dblink;

--删除创建出的用户。
gaussdb=# RESET ROLE;
gaussdb=# DROP USER user1;
gaussdb=# DROP USER user2;
```

相关链接

CREATE DATABASE LINK，**DROP DATABASE LINK**

7.14.9 ALTER DEFAULT PRIVILEGES

功能描述

设置应用于将来创建的对象的权限（这不会影响分配到已有对象中的权限）。

注意事项

目前只支持表（包括视图）、序列、函数，类型，密态数据库客户端主密钥和列加密密钥的权限更改。

语法格式

```
ALTER DEFAULT PRIVILEGES
    [ FOR { ROLE | USER } target_role [, ...] ]
    [ IN SCHEMA schema_name [, ...] ]
    abbreviated_grant_or_revoke;
```

- 其中abbreviated_grant_or_revoke子句用于指定对哪些对象进行授权或回收权限。

  ```
  grant_on_tables_clause
    | grant_on_sequences_clause
    | grant_on_functions_clause
    | grant_on_types_clause
    | grant_on_client_master_keys_clause
    | grant_on_column_encryption_keys_clause
    | revoke_on_tables_clause
    | revoke_on_sequences_clause
    | revoke_on_functions_clause
    | revoke_on_types_clause
    | revoke_on_client_master_keys_clause
    | revoke_on_column_encryption_keys_clause
  ```

- 其中grant_on_tables_clause子句用于对表授权。

  ```
  GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | ALTER | DROP |
  COMMENT | INDEX | VACUUM }
      [, ...] | ALL [ PRIVILEGES ] }
      ON TABLES
      TO { [ GROUP ] role_name | PUBLIC } [, ...]
      [ WITH GRANT OPTION ]
  ```

- 其中grant_on_sequences_clause子句用于对序列授权。

  ```
  GRANT { { SELECT | UPDATE | USAGE | ALTER | DROP | COMMENT }
      [, ...] | ALL [ PRIVILEGES ] }
      ON SEQUENCES
      TO { [ GROUP ] role_name | PUBLIC } [, ...]
      [ WITH GRANT OPTION ]
  ```

- 其中grant_on_functions_clause子句用于对函数授权。

  ```
  GRANT { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
      ON FUNCTIONS
      TO { [ GROUP ] role_name | PUBLIC } [, ...]
      [ WITH GRANT OPTION ]
  ```

- 其中grant_on_types_clause子句用于对类型授权。

  ```
  GRANT { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
      ON TYPES
      TO { [ GROUP ] role_name | PUBLIC } [, ...]
      [ WITH GRANT OPTION ]
  ```

- 其中grant_on_client_master_keys_clause子句用于对客户端主密钥授权。

  ```
  GRANT { { USAGE  | DROP } [, ...] | ALL [ PRIVILEGES ] }
      ON CLIENT_MASTER_KEYS
      TO { [ GROUP ] role_name | PUBLIC } [, ...]
      [ WITH GRANT OPTION ]
  ```


- 其中grant_on_column_encryption_keys_clause子句用于对列加密密钥授权。

```
GRANT { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON COLUMN_ENCRYPTION_KEYS
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ]
```

- 其中revoke_on_tables_clause子句用于回收表对象的权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | ALTER | DROP | COMMENT |
INDEX | VACUUM }
    [, ...] | ALL [ PRIVILEGES ] }
    ON TABLES
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

- 其中revoke_on_sequences_clause子句用于回收序列的权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | UPDATE | USAGE | ALTER | DROP | COMMENT }
    [, ...] | ALL [ PRIVILEGES ] }
    ON SEQUENCES
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

- 其中revoke_on_functions_clause子句用于回收函数的权限。

```
REVOKE [ GRANT OPTION FOR ]
    { {EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON FUNCTIONS
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

- 其中revoke_on_types_clause子句用于回收类型的权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON TYPES
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

- 其中revoke_on_client_master_keys_clause子句用于回收客户端主密钥的权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON CLIENT_MASTER_KEYS
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

- 其中revoke_on_column_encryption_keys_clause子句用于回收列加密密钥的权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON COLUMN_ENCRYPTION_KEYS
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

参数说明

- **target_role**

  已有角色的名称。如果省略FOR ROLE/USER，则缺省值为当前角色/用户。

  取值范围：已有角色的名称。

- **schema_name**

  现有模式的名称。

  target_role必须有schema_name的CREATE权限。

  取值范围：现有模式的名称。

- **role_name**

  被授予或者取消权限角色的名称。

取值范围：已存在的角色名称。

须知

如果想删除一个被赋予了默认权限的角色，有必要恢复改变的缺省权限或者使用DROP OWNED BY来为角色脱离缺省的权限记录。

示例

```
--将创建在模式tpcds里的所有表（和视图）的SELECT权限授予每一个用户。
gaussdb=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds GRANT SELECT ON TABLES TO PUBLIC;

--创建用户普通用户jack。
gaussdb=# CREATE USER jack PASSWORD '*********';

--将tpcds下的所有表的插入权限授予用户jack。
gaussdb=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds GRANT INSERT ON TABLES TO jack;

--撤销上述权限。
gaussdb=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds REVOKE SELECT ON TABLES FROM PUBLIC;
gaussdb=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds REVOKE INSERT ON TABLES FROM jack;

--删除用户jack。
gaussdb=# DROP USER jack;
```

相关链接

GRANT，**REVOKE**

7.14.10 ALTER DIRECTORY

功能描述

对directory属性进行修改。

注意事项

- 目前只支持修改directory属主。
- 当enable_access_server_directory=off时，只允许初始用户修改directory属主；当enable_access_server_directory=on时，具有SYSADMIN权限的用户和directory对象的属主可以修改directory，且要求该用户是新属主的成员。

语法格式

```
ALTER DIRECTORY directory_name
    OWNER TO new_owner;
```

参数描述

- **directory_name**

  需要修改的目录名称，范围为已经存在的目录名称。

- **new_owner**

  目录的新属主。

示例

```
--创建目录。
gaussdb=# CREATE OR REPLACE DIRECTORY  dir  as '/tmp/';

--创建用户
gaussdb=# CREATE USER jim PASSWORD '********';

--修改目录的owner。
gaussdb=# ALTER DIRECTORY dir OWNER TO jim;

--删除目录。
gaussdb=# DROP DIRECTORY dir;

--删除用户。
gaussdb=# DROP USER jim;
```

相关链接

CREATE DIRECTORY，DROP DIRECTORY

7.14.11 ALTER FOREIGN DATA WRAPPER

功能描述

修改外部数据封装的定义。

注意事项

- 只有初始用户和系统管理员用户能够修改外部数据封装。
- 只有在support_extended_features=on时才能够成功执行alter语句。

语法格式

- 设置外部数据封装属性
  ```
  ALTER FOREIGN DATA WRAPPER name
  [ HANDLER handler_function | NO HANDLER ]
  [ VALIDATOR validator_function | NO VALIDATOR ]
  [ OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ... ]) ];
  ```
- 设置新的所有者
  ```
  ALTER FOREIGN DATA WRAPPER name OWNER TO new_owner;
  ```
- 设置新的名称
  ```
  ALTER FOREIGN DATA WRAPPER name RENAME TO new_name;
  ```

参数说明

- **name**

  已有外部数据封装的名字。
- **HANDLER handler_function**

  为外部数据封装指定一个新的处理函数。
- **NO HANDLER**

  这个参数用来指定外部数据封装不再拥有处理函数。

> **注意**
>
> 使用外部数据封装但没有handler的外表不能访问。

- **VALIDATOR validator_function**

  为外部数据封装指定一个新的验证函数。

> **注意**
>
> 根据新的验证器，外部数据封装器或依赖的服务器、 用户映射或外部表的已经存在的选项是有可能是无效的。用户在使用外部数据封装之前需要保证这些选项是正确的。 不过，ALTER FOREIGN DATA WRAPPER 命令中指定的任何选项都将使用新的验证函数检查。

- **NO VALIDATOR**

  这个用来指定外部数据封装不再有验证函数。

- **OPTIONS ([ADD | SET | DROP] option ['value'] [, ...])**

  修改外部数据封装的选项。 ADD, SET, 和 DROP 指定表现的动作。如果没有明确指定操作默认是ADD。选项名必须唯一。 使用外部数据封装验证函数时，名字和取值也会被验证。

- **new_owner**

  外部数据封装新的所有者的用户名。

- **new_name**

  外部数据封装的新名称。

示例

```
--修改一个外部数据封装dbi，增加选项foo，删除bar
gaussdb=# ALTER FOREIGN DATA WRAPPER dbi OPTIONS (ADD foo '1', DROP 'bar');

--修改外部数据封装dbi验证器为bob.myvalidator
gaussdb=# ALTER FOREIGN DATA WRAPPER dbi VALIDATOR bob.myvalidator;
```

相关链接

CREATE FOREIGN DATA WRAPPER，DROP FOREIGN DATA WRAPPER

7.14.12 ALTER FUNCTION

功能描述

修改自定义函数的属性或重编译函数。

注意事项

- 只有函数的所有者或者被授予了函数ALTER权限的用户才能执行ALTER FUNCTION命令，系统管理员默认拥有该权限。针对所要修改属性的不同，还有以下权限约束：
  - 如果函数中涉及对临时表相关的操作，则无法使用ALTER FUNCTION。

  - 修改函数的所有者或修改函数的模式，当前用户必须是该函数的所有者或者系统管理员，且该用户是新所有者角色的成员。
  - 只有系统管理员和初始化用户可以将function的schema修改成public。
- 重编译需要设置plpgsql_dependency参数。
- 仅有初始化用户或者创建该存储过程的用户可以修改存储过程为定义者权限的存储过程。
- 当打开三权分立时，对于定义者权限的函数，不允许任何角色修改函数的OWNER。
- 当关闭三权分立时，对于定义者权限的函数，仅初始用户和系统管理员可以修改函数的OWNER，但不允许将函数OWNER修改为运维管理员。
- 只有初始化用户才能修改函数的OWNER为初始化用户。

语法格式

- 修改自定义函数的附加参数。
```
ALTER FUNCTION function_name ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    action [ ... ] [ RESTRICT ];
```
  其中附加参数action子句语法为。
```
{CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT}
 | {IMMUTABLE | STABLE | VOLATILE}
 | {SHIPPABLE | NOT SHIPPABLE}
 | {NOT FENCED | FENCED}
 | [ NOT ] LEAKPROOF
 | { [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER }
 | AUTHID { DEFINER | CURRENT_USER }
 | COST execution_cost
 | ROWS result_rows
 | SET configuration_parameter { { TO | = } { value | DEFAULT }| FROM CURRENT}
 | RESET {configuration_parameter | ALL}
```
- 修改自定义函数的名称。
```
ALTER FUNCTION function_name ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    RENAME TO new_name;
```
- 修改自定义函数的所属者。
```
ALTER FUNCTION function_name ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    OWNER TO new_owner;
```
- 修改自定义函数的模式。
```
ALTER FUNCTION function_name ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    SET SCHEMA new_schema;
```
- 重编译函数
```
ALTER FUNCTION function_name COMPILE;
```

参数说明

- **function_name**

  要修改的函数名称。

  取值范围：已存在的函数名。
- **argmode**

  标识该参数是输入、输出参数。

  取值范围：
  - IN：声明入参。
  - OUT：声明出参。

  - INOUT：声明出入参。
  - VARIADIC：声明数组类型的参数。

- **argname**

  参数名称。

  取值范围：字符串，符合**标识符命名规范**。

- **argtype**

  函数参数的类型。

- **CALLED ON NULL INPUT**

  表明该函数的某些参数是NULL的时候可以按照正常的方式调用。缺省时与指定此参数的作用相同。

- **RETURNS NULL ON NULL INPUT**

  STRICT

  STRICT用于指定如果函数的某个参数是NULL，此函数总是返回NULL。如果声明了这个参数，则如果存在NULL参数时不会执行该函数；而只是自动假设一个NULL结果。

  RETURNS NULL ON NULL INPUT和STRICT的功能相同。

- **IMMUTABLE**

  表示该函数在给出同样的参数值时总是返回同样的结果。

- **STABLE**

  表示该函数不能修改数据库，对相同参数值，在同一次表扫描里，该函数的返回值不变，但是返回值可能在不同SQL语句之间变化。

- **VOLATILE**

  表示该函数值可以在一次表扫描内改变，不会做任何优化。

- **LEAKPROOF**

  表示该函数没有副作用，指出参数只包括返回值。LEAKPROOF只能由系统管理员设置。

- **EXTERNAL**

  （可选）目的是和SQL兼容，这个特性适合于所有函数，而不仅是外部函数。

- **SECURITY INVOKER**

  AUTHID CURRENT_USER

  表明该函数将以调用它的用户的权限执行。缺省时与指定此参数的作用相同。

  SECURITY INVOKER和AUTHID CURRENT_USER的功能相同。

- **SECURITY DEFINER**

  AUTHID DEFINER

  声明该函数将以创建它的用户的权限执行。

  AUTHID DEFINER和SECURITY DEFINER的功能相同。

- **COST execution_cost**

  用来估计函数的执行成本。

  execution_cost以cpu_operator_cost为单位。

  取值范围：正数。

- **ROWS result_rows**

估计函数返回的行数。用于函数返回的是一个集合。

取值范围：正数，默认值是1000行。

- **configuration_parameter**
  - **value**

    把指定的数据库会话参数值设置为给定的值。如果value是DEFAULT或者RESET，则在新的会话中使用系统的缺省设置。OFF关闭设置。

    取值范围：字符串。

    - DEFAULT
    - OFF
    - RESET

    指定默认值。
  - **FROM CURRENT**

    取当前会话中的值设置为configuration_parameter的值。
- **new_name**

  函数的新名称。要修改函数的所属模式，必须拥有新模式的CREATE权限。

  取值范围：字符串，符合**标识符命名规范**。
- **new_owner**

  函数的新所有者。要修改函数的所有者，新所有者必须拥有该函数所属模式的CREATE权限。需要注意注意的是：仅有初始化用户才可以将函数的owner设置为初始化用户。

  取值范围：已存在的用户角色。
- **new_schema**

  函数的新模式。

  取值范围：已存在的模式。

示例

请参见CREATE FUNCTION的**示例**。

重编译示例：

```
--开启依赖功能。
gaussdb=# set behavior_compat_options ='plpgsql_dependency';

--创建函数。
gaussdb=# create or replace function test_func(a int) return int
is
    proc_var int;
gaussdb=# begin
    proc_var := a;
    return 1;
end;
/

--用函数名重编译函数。
gaussdb=# alter procedure test_func compile;

--用函数带类型签名重编译存储过程。
gaussdb=# alter procedure test_func(int) compile;
```


相关链接

CREATE FUNCTION，DROP FUNCTION

7.14.13 ALTER GLOBAL CONFIGURATION

功能描述

新增、修改系统表gs_global_config，增加key-value值。

注意事项

- 仅支持数据库初始用户运行此命令。
- 参数名称不能为weak_password、undostoragetype。

语法格式

```
ALTER GLOBAL CONFIGURATION with(name=value, name=value...);
```

参数说明

- **name**

  参数名称，text类型，不能为weak_password、undostoragetype，除此之外没有限制。
- **value**

  参数值，text类型。

相关链接

DROP GLOBAL CONFIGURATION

7.14.14 ALTER GROUP

功能描述

修改一个用户组的属性。

注意事项

ALTER GROUP是ALTER ROLE的别名，非SQL标准语法，不推荐使用，建议用户直接使用ALTER ROLE替代。

语法格式

- 向用户组中添加用户。

  ```
  ALTER GROUP group_name
    ADD USER user_name [, ... ];
  ```
- 从用户组中删除用户。

  ```
  ALTER GROUP group_name
    DROP USER user_name [, ... ];
  ```
- 修改用户组的名称。

  ```
  ALTER GROUP group_name
    RENAME TO new_name;
  ```


参数说明

请参考ALTER ROLE的**参数说明**。

示例

```
--创建用户组。
gaussdb=# CREATE GROUP super_users WITH PASSWORD "********";

--创建用户。
gaussdb=# CREATE ROLE lche WITH PASSWORD "********";

--创建用户。
gaussdb=# CREATE ROLE jim WITH PASSWORD "********";

--向用户组中添加用户。
gaussdb=# ALTER GROUP super_users ADD USER lche, jim;

--从用户组中删除用户。
gaussdb=# ALTER GROUP super_users DROP USER jim;

--修改用户组的名称。
gaussdb=# ALTER GROUP super_users RENAME TO normal_users;

--删除用户。
gaussdb=# DROP ROLE lche, jim;

--删除用户组。
gaussdb=# DROP GROUP normal_users;
```

相关链接

CREATE GROUP，**DROP GROUP**，**ALTER ROLE**

7.14.15 ALTER INDEX

功能描述

ALTER INDEX用于修改现有索引的定义。

它有几种子形式：

- IF EXISTS

  如果指定的索引不存在，则发出一个notice而不是error。
- RENAME TO

  只改变索引的名称。对存储的数据没有影响。
- SET TABLESPACE

  这个选项会改变索引的表空间为指定表空间，并且把索引相关的数据文件移动到新的表空间里。
- SET ({ STORAGE_PARAMETER = value } [, ...])

  改变索引的一个或多个索引方法特定的存储参数。 需要注意的是索引内容不会被这个命令立即修改，根据参数的不同，可能需要使用REINDEX重建索引来获得期望的效果。
- RESET ({ storage_parameter } [, ...])

  重置索引的一个或多个索引方法特定的存储参数为缺省值。与SET一样，可能需要使用REINDEX来完全更新索引。

- [MODIFY PARTITION index_partition_name] UNUSABLE
  用于设置表或者索引分区上的索引不可用。
- REBUILD [PARTITION index_partition_name]
  用于重建表或者索引分区上的索引。
- RENAME PARTITION
  用于重命名索引分区。
- MOVE PARTITION
  用于修改索引分区的所属表空间。

注意事项

索引的所有者或者拥有索引所在表的INDEX权限的用户或者被授予了ALTER ANY INDEX权限的用户有权限执行此命令，系统管理员默认拥有此权限。

语法格式

- 重命名表索引的名称。
```
ALTER INDEX [ IF EXISTS ] index_name
    RENAME TO new_name;
```
- 修改表索引的所属空间。
```
ALTER INDEX [ IF EXISTS ] index_name
    SET TABLESPACE tablespace_name;
```
- 修改表索引的存储参数。
```
ALTER INDEX [ IF EXISTS ] index_name
    SET ( {storage_parameter = value} [, ... ] );
```
- 重置表索引的存储参数。
```
ALTER INDEX [ IF EXISTS ] index_name
    RESET ( storage_parameter [, ... ] ) ;
```
- 设置表索引或索引分区不可用。
```
ALTER INDEX [ IF EXISTS ] index_name
    [ MODIFY PARTITION index_partition_name ] UNUSABLE;
```
- 重建表索引或索引分区。
```
ALTER INDEX index_name
    REBUILD [ PARTITION index_partition_name ];
```
- 重命名索引分区。
```
ALTER INDEX [ IF EXISTS ] index_name
    RENAME PARTITION index_partition_name TO new_index_partition_name;
```
- 修改索引分区的所属表空间。
```
ALTER INDEX [ IF EXISTS ] index_name
    MOVE PARTITION index_partition_name TABLESPACE new_tablespace;
```

参数说明

- **index_name**
  要修改的索引名。
- **new_name**
  新的索引名。
  取值范围：字符串，且符合**标识符命名规范**。
- **tablespace_name**
  表空间的名称。

取值范围：已存在的表空间。

- **storage_parameter**

  索引方法特定的参数名。ACTIVE_PAGES表示索引的页面数量，可能比实际的物理文件页面少，可以用于优化器调优。目前只对ustore的分区表local索引生效，且会被vacuum、analyze更新（包括auto vacuum）。不建议用户手动设置该参数。

- **value**

  索引方法特定的存储参数的新值。根据参数的不同，这可能是一个数字或单词。

- **new_index_partition_name**

  新索引分区名。

- **index_partition_name**

  索引分区名。

- **new_tablespace**

  新表空间。

示例

请参见CREATE INDEX的**示例**。

相关链接

CREATE INDEX，**DROP INDEX**，**REINDEX**

7.14.16 ALTER LANGUAGE

本版本暂不支持使用该语法。

7.14.17 ALTER MASKING POLICY

功能描述

修改脱敏策略。

注意事项

- 只有poladmin，sysadmin或初始用户才能执行此操作。
- 需要打开enable_security_policy开关脱敏策略才可以生效。

语法格式

- 修改策略描述：
  ```
  ALTER MASKING POLICY policy_name COMMENTS policy_comments;
  ```
- 修改脱敏方式：
  ```
  ALTER MASKING POLICY policy_name [ADD | REMOVE | MODIFY] masking_actions[, ...];
  其中masking_action:
    masking_function ON LABEL(label_name[, ...])
  ```
- 修改脱敏策略生效场景：
  ```
  ALTER MASKING POLICY policy_name MODIFY(FILTER ON FILTER_TYPE(filter_value[, ...])[, ...]);
  ```
- 移除脱敏策略生效场景，使策略对所用场景生效：
  ```
  ALTER MASKING POLICY policy_name DROP FILTER;
  ```


- 修改脱敏策略开启/关闭：
  ```
  ALTER MASKING POLICY policy_name [ENABLE | DISABLE];
  ```

参数说明

- **policy_name**
  脱敏策略名称，需要唯一，不可重复。
  取值范围：字符串，要符合**标识符命名规范**。
- **policy_comments**
  需要为脱敏策略添加或修改的描述信息。
- **masking_function**
  指的是预置的八种脱敏方式或者用户自定义的函数，支持模式。
  maskall不是预置函数，硬编码在代码中，不支持\df展示。
  预置时脱敏方式如下：
  ```
  maskall | randommasking | creditcardmasking | basicemailmasking | fullemailmasking |
  shufflemasking | alldigitsmasking | regexpmasking
  ```
- **label_name**
  资源标签名称。
- **FILTER_TYPE**
  指定脱敏策略的过滤信息，过滤类型包括：IP、ROLES、APP。
- **filter_value**
  指具体过滤信息内容，例如具体的IP，具体的APP名称，具体的用户名。
- **ENABLE|DISABLE**
  可以打开或关闭脱敏策略。若不指定ENABLE|DISABLE，语句默认为ENABLE。

示例

```
--创建dev_mask和bob_mask用户。
gaussdb=# CREATE USER dev_mask PASSWORD '********';
gaussdb=# CREATE USER bob_mask PASSWORD '********';

--创建一个表tb_for_masking。
gaussdb=# CREATE TABLE tb_for_masking(col1 text, col2 text, col3 text);

--创建资源标签标记敏感列col1。
gaussdb=# CREATE RESOURCE LABEL mask_lb1 ADD COLUMN(tb_for_masking.col1);

--创建资源标签标记敏感列col2。
gaussdb=# CREATE RESOURCE LABEL mask_lb2 ADD COLUMN(tb_for_masking.col2);

--对访问敏感列col1的操作创建脱敏策略。
gaussdb=# CREATE MASKING POLICY maskpol1 maskall ON LABEL(mask_lb1);

--为脱敏策略maskpol1添加描述。
gaussdb=# ALTER MASKING POLICY maskpol1 COMMENTS 'masking policy for tb_for_masking.col1';

--修改脱敏策略maskpol1，新增一项脱敏方式。
gaussdb=# ALTER MASKING POLICY maskpol1 ADD randommasking ON LABEL(mask_lb2);

--修改脱敏策略maskpol1，移除一项脱敏方式。
gaussdb=# ALTER MASKING POLICY maskpol1 REMOVE randommasking ON LABEL(mask_lb2);

--修改脱敏策略maskpol1，修改一项脱敏方式。
gaussdb=# ALTER MASKING POLICY maskpol1 MODIFY randommasking ON LABEL(mask_lb1);
```



```
--修改脱敏策略maskpol1使之仅对用户dev_mask和bob_mask,客户端工具为gsql，IP地址为'10.20.30.40',
'127.0.0.0/24'场景生效。
gaussdb=# ALTER MASKING POLICY maskpol1 MODIFY (FILTER ON ROLES(dev_mask, bob_mask),
APP(gsql), IP('10.20.30.40', '127.0.0.0/24'));

--修改脱敏策略maskpol1，使之对所有用户场景生效。
gaussdb=# ALTER MASKING POLICY maskpol1 DROP FILTER;

--禁用脱敏策略maskpol1。
gaussdb=# ALTER MASKING POLICY maskpol1 DISABLE;

--删除脱敏策略。
gaussdb=# DROP MASKING POLICY maskpol1;

-删除资源标签。
gaussdb=# DROP RESOURCE LABEL mask_lb1, mask_lb2;

-删除表tb_for_masking。
gaussdb=# DROP TABLE tb_for_masking;

--删除用户dev_mask和bob_mask。
gaussdb=# DROP USER dev_mask, bob_mask;
```

相关链接

CREATE MASKING POLICY，**DROP MASKING POLICY**。

7.14.18 ALTER MATERIALIZED VIEW

功能描述

更改一个现有物化视图的多个辅助属性。

可用于ALTER MATERIALIZED VIEW的语句形式和动作是ALTER TABLE的一个子集，并且在用于物化视图时具有相同的含义。详见**ALTER TABLE**。

注意事项

- 只有物化视图的所有者有权限执行ALTER TMATERIALIZED VIEW命令，系统管理员默认拥有此权限。
- 不支持更改物化视图结构。

语法格式

- 修改物化视图的所属用户。

```
ALTER MATERIALIZED VIEW [ IF EXISTS ] mv_name
    OWNER TO new_owner;
```

- 修改物化视图的列。

```
ALTER MATERIALIZED VIEW [ IF EXISTS ] mv_name
    RENAME [ COLUMN ] column_name TO new_column_name;
```

- 重命名物化视图。

```
ALTER MATERIALIZED VIEW [ IF EXISTS ] mv_name

    RENAME TO new_name;
```

参数说明

- **mv_name**

  一个现有物化视图的名称，可以用模式修饰。

取值范围：字符串，符合**标识符命名规范**。

- **column_name**
  一个新的或者现有的列的名称。
  取值范围：字符串，符合**标识符命名规范**。
- **new_column_name**
  一个现有列的新名称。
- **new_owner**
  该物化视图的新拥有者的用户名。
- **new_name**
  该物化视图的新名称。

示例

```
--创建表。
gaussdb=# CREATE TABLE my_table (c1 int, c2 int) WITH(STORAGE_TYPE=ASTORE);

--创建全量物化视图。
gaussdb=# CREATE MATERIALIZED VIEW foo AS SELECT * FROM my_table;

--把物化视图foo重命名为bar。
gaussdb=# ALTER MATERIALIZED VIEW foo RENAME TO bar;

--删除全量物化视图。
gaussdb=# DROP MATERIALIZED VIEW bar;

--删除表my_table。
gaussdb=# DROP TABLE my_table;
```

相关链接

CREATE INCREMENTAL MATERIALIZED VIEW，**CREATE MATERIALIZED VIEW**，**DROP MATERIALIZED VIEW**，**REFRESH INCREMENTAL MATERIALIZED VIEW**，**REFRESH MATERIALIZED VIEW**

7.14.19 ALTER OPERATOR

功能描述

修改一个操作符的定义。

注意事项

要使用ALTER OPERATOR，你必须是该操作符的所有者。 要修改所有者，你还必须是新的所有角色的直接或间接成员，并且该成员必须在此操作符的模式上有CREATE权限。（这些限制强制了修改该所有者不会做任何通过删除和重建操作符不能做的事情。不过，具有SYSADMIN权限用户可以以任何方式修改任意操作符的所有权。）

语法格式

- 更改操作符的所有者。
  ```
  ALTER OPERATOR name ( { left_type | NONE } , { right_type | NONE } ) OWNER TO new_owner;
  ```
- 更改操作符的模式。
  ```
  ALTER OPERATOR name ( { left_type | NONE } , { right_type | NONE } ) SET SCHEMA new_schema;
  ```


参数说明

- **name**
  一个现有操作符的名字。
- **left_type**
  操作符的左操作数的数据类型；如果没有左操作数，那么写NONE。
- **right_type**
  操作符的右操作数的数据类型；如果没有右操作数，那么写NONE。
- **new_owner**
  操作符的新所有者。
- **new_schema**
  操作符的新模式名。

示例

改变一个用于text的用户定义操作符a @@ b：

```
gaussdb=# ALTER OPERATOR @@ (text, text) OWNER TO joe;
```

兼容性

SQL 标准里没有ALTER OPERATOR语句。

相关链接

CREATE OPERATOR，**CREATE OPERATOR CLASS**，**DROP OPERATOR**

7.14.20 ALTER PACKAGE

功能描述

修改PACKAGE的属性或重编译包。

注意事项

- 目前仅支持ALTER PACKAGE OWNER功能，系统管理员默认拥有该权限，有以下权限约束：
  - 当前用户必须是该package的所有者或者系统管理员，且该用户是新所有者角色的成员。
- 重编译包需要设置plpgsql_dependency参数。
- 仅有初始化用户可以修改定义者权限的package的owner。
- 当打开三权分立时，即使是系统管理员，也必须拥有用户组权限才能修改package的owner，DEFINER类型package不允许修改所有者。
- 只有初始化用户才能修改package的owner为初始化用户。
- 非三权分立模式下，仅系统管理员以上权限可以修改package所有者，但不允许修改所有者为运维管理员。
- 不允许系统管理员将DEFINER类型的package的所有者改为初始用户或运维管理员。

语法格式

- 修改package的所有者。
  ```
  ALTER PACKAGE package_name OWNER TO new_owner;
  ```
- 重编译包
  ```
  ALTER PACKAGE package_name COMPILE [PACKAGE | BODY | SPECIFICATION];
  ```

参数说明

- **package_name**

  要修改的package名称。

  取值范围：已存在的package名，仅支持修改单个package。

- **new_owner**

  package的新所有者。要修改函数的所有者，新所有者必须拥有该package所属模式的CREATE权限。

  取值范围：已存在的用户角色。

示例

请参见**CREATE PACKAGE**中示例。

重编译示例：

```
--开启依赖功能。
gaussdb=# set behavior_compat_options ='plpgsql_dependency';

--创建函数。
gaussdb=# create or replace package test_pkg as
    pkg_var int := 1;
    procedure test_pkg_proc(var int);
end test_pkg;
/

gaussdb=# create or replace package body test_pkg as
    procedure test_pkg_proc(var int)
is
begin
    pkg_var := 1;
end;
end test_pkg;
/

--重编译包。
gaussdb=# alter package test_pkg compile;
```

相关链接

CREATE PACKAGE，**DROP PACKAGE**

7.14.21 ALTER PROCEDURE

功能描述

修改自定义存储过程的属性或重编译存储过程。

注意事项

- 只有存储过程的所有者或者被授予了存储过程ALTER权限的用户才能执行ALTER PROCEDURE命令，系统管理员默认拥有该权限。针对所要修改属性的不同，还有以下权限约束：
  - 如果存储过程中涉及对临时表相关的操作，则无法使用ALTER PROCEDURE。
  - 修改存储过程的所有者或修改存储过程的模式，当前用户必须是该存储过程的所有者或者系统管理员，且该用户是新所有者角色的成员。
  - 只有系统管理员和初始化用户可以将procedure的schema修改成public。
- 重编译存储过程需要设置plpgsql_dependency参数。
- 仅有初始用户或者创建该存储过程的用户可以修改存储过程为定义者权限的存储过程。
- 当打开三权分立时，对于定义者权限的存储过程，不允许任何角色修改存储过程的owner。
- 当关闭三权分立时，对于定义者权限的存储过程，仅初始用户和系统管理员可以修改存储过程的OWNER，但不允许将存储过程OWNER修改为运维管理员。
- 只有初始用户才能修改存储过程的OWNER为初始用户。

语法格式

- 修改自定义存储过程的附加参数。

```
ALTER PROCEDURE procedure_name ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    action [ ... ] [ RESTRICT ];
```

  其中附加参数action子句语法为。

```
{CALLED ON NULL INPUT  | STRICT}
 | {IMMUTABLE | STABLE | VOLATILE}
 | {SHIPPABLE | NOT SHIPPABLE}
 | {NOT FENCED | FENCED}
 | [ NOT ] LEAKPROOF
 | { [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER }
 | AUTHID { DEFINER | CURRENT_USER }
 | COST execution_cost
 | ROWS result_rows
 | SET configuration_parameter { { TO | = } { value | DEFAULT }| FROM CURRENT}
 | RESET {configuration_parameter | ALL}
```

- 修改自定义存储过程的名称。

```
ALTER PROCEDURE proname ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    RENAME TO new_name;
```

- 修改自定义存储过程的所属者。

```
ALTER PROCEDURE proname ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    OWNER TO new_owner;
```

- 修改自定义存储过程的模式。

```
ALTER PROCEDURE proname ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    SET SCHEMA new_schema;
```

- 重编译存储过程

```
ALTER PROCEDURE procedure_name[( [ { [ argname ] [ argmode ] argtype} [, ...] ] )] COMPILE;
```

参数说明

- **procedure_name**

  要修改的存储过程名称。

  取值范围：已存在的存储过程名。

- **argmode**

  标识该参数是输入、输出参数。

  取值范围：IN/OUT/INOUT/VARIADIC。

- **argname**

  参数名称。

  取值范围：字符串，符合**标识符命名规范**。

- **argtype**

  存储过程参数的类型。

- **CALLED ON NULL INPUT**

  表明该存储过程的某些参数是NULL的时候可以按照正常的方式调用。缺省时与指定此参数的作用相同。

- **IMMUTABLE**

  表示该存储过程在给出同样的参数值时总是返回同样的结果。

- **STABLE**

  表示该存储过程不能修改数据库，对相同参数值，在同一次表扫描里，该函数的返回值不变，但是返回值可能在不同SQL语句之间变化。

- **VOLATILE**

  表示该存储过程值可以在一次表扫描内改变，不会做任何优化。

- **LEAKPROOF**

  表示该存储过程没有副作用，指出参数只包括返回值。LEAKPROOF只能由系统管理员设置。

- **EXTERNAL**

  （可选）目的是和SQL兼容，这个特性适合于所有函数，而不仅是外部函数。

- **SECURITY INVOKER**

  AUTHID CURRENT_USER

  表明该存储过程将以调用它的用户的权限执行。缺省时与指定此参数的作用相同。

  SECURITY INVOKER和AUTHID CURRENT_USER的功能相同。

- **SECURITY DEFINER**

  AUTHID DEFINER

  声明该存储过程将以创建它的用户的权限执行。

  AUTHID DEFINER和SECURITY DEFINER的功能相同。

- **COST execution_cost**

  用来估计存储过程的执行成本。

  execution_cost以cpu_operator_cost为单位。

  取值范围：正数。

- **ROWS result_rows**

  估计存储过程返回的行数。用于存储过程返回的是一个集合。

  取值范围：正数，默认值是1000行。

- **configuration_parameter**
  - **value**

把指定的数据库会话参数值设置为给定的值。如果value是DEFAULT或者RESET，则在新的会话中使用系统的缺省设置。OFF关闭设置。

取值范围：字符串。

- DEFAULT
- OFF
- RESET

指定默认值。

- **FROM CURRENT**

  取当前会话中的值设置为configuration_parameter的值。

- **new_name**

  存储过程的新名称。要修改存储过程的所属模式，必须拥有新模式的CREATE权限。

  取值范围：字符串，符合**标识符命名规范**。

- **new_owner**

  存储过程的新所有者。要修改存储过程的所有者，新所有者必须拥有该存储过程所属模式的CREATE权限。

  取值范围：已存在的用户角色。

- **new_schema**

  存储过程的新模式。

  取值范围：已存在的模式。

示例

请参见CREATE FUNCTION的**示例**。

重编译示例：

```
--开启依赖功能。
gaussdb=# set behavior_compat_options ='plpgsql_dependency';
--创建存储过程。
gaussdb=# create or replace procedure test_proc(a int)
is
    proc_var int;
begin
    proc_var := a;
end;
/
--用存储过程名重编译存储过程。
gaussdb=# alter procedure test_proc compile;

--用存储过程带类型签名重编译存储过程。
gaussdb=# alter procedure test_proc(int) compile;
```

相关链接

CREATE PROCEDURE，**DROP PROCEDURE**

7.14.22 ALTER RESOURCE LABEL

功能描述

修改资源标签。

注意事项

只有POLADMIN、SYSADMIN或初始用户才能执行此操作。

语法格式

```
ALTER RESOURCE LABEL label_name (ADD|REMOVE)
  label_item_list[, ...];
```

- label_item_list：
  ```
  resource_type(resource_path[, ...])
  ```
- resource_type：
  ```
  TABLE | COLUMN | SCHEMA | VIEW | FUNCTION
  ```

参数说明

- **label_name**

  资源标签名称。

  取值范围：字符串，要符合**标识符命名规范**。
- **resource_type**

  指的是要标记的数据库资源类型。
- **resource_path**

  指的是描述具体的数据库资源的路径。

示例

```
--创建基本表table_for_label。
gaussdb=# CREATE TABLE table_for_label(col1 int, col2 text);

--创建资源标签table_label。
gaussdb=# CREATE RESOURCE LABEL table_label ADD COLUMN(table_for_label.col1);

--将col2添加至资源标签table_label中。
gaussdb=# ALTER RESOURCE LABEL table_label ADD COLUMN(table_for_label.col2)

--将资源标签table_label中的一项移除。
gaussdb=# ALTER RESOURCE LABEL table_label REMOVE COLUMN(table_for_label.col1);

--删除资源标签table_label。
gaussdb=# DROP RESOURCE LABEL table_label;

--删除基本表table_for_label。
gaussdb=# DROP TABLE table_for_label;
```

相关链接

CREATE RESOURCE LABEL，**DROP RESOURCE LABEL**。

7.14.23 ALTER RESOURCE POOL

功能描述

修改一个资源池，指定其他控制组。

注意事项

只要用户对当前数据库有ALTER权限，就可以修改资源池。

语法格式

```
ALTER RESOURCE POOL pool_name
    WITH ({MEM_PERCENT= pct | CONTROL_GROUP="group_name" | ACTIVE_STATEMENTS=stmt |
MAX_DOP = dop | MEMORY_LIMIT='memory_size' | io_limits=io_limits | io_priority='io_priority'}[, ... ]);
```

参数说明

- **pool_name**

  资源池名称。

  资源池名称为已创建的资源池。

  取值范围：字符串，要符合**标识符命名规范**。

- **group_name**

  控制组名称。

  说明

  - 设置控制组名称时，语法可以使用双引号，也可以使用单引号。
  - group_name对大小写敏感。
  - 若数据库管理员指定自定义Class组下的Workload控制组，如control_group的字符串为："class1:workload1"；代表此资源池指定到class1控制组下的workload1控制组。也可同时指定Workload控制组的层次，如control_group的字符串为："class1:workload1:1"。
  - 若数据库用户指定Timeshare控制组代表的字符串，即"Rush"、"High"、"Medium"或"Low"其中一种，如control_group的字符串为"High"；代表资源池指定到DefaultClass控制组下的"High" Timeshare控制组。

  取值范围：已创建的控制组。

- **stmt**

  资源池语句执行的最大并发数量。

  取值范围：数值型，-1~2147483647。-1：不限制，0：禁止任何语句执行。

- **dop**

  资源池最大并发度，语句执行时能够创建的最多线程数量。

  取值范围：数值型，1~2147483647。

- **memory_size**

  资源池最大使用内存。

  取值范围：字符串，内容范围1KB~2047GB，单位大小写敏感。

- **mem_percent**

  资源池可用内存占全部内存或者组用户内存使用的比例。

在多租户场景下，组用户和业务用户的mem_percent范围为1-100的整数，默认为20。

在普通场景下，普通用户的mem_percent范围为0-100的整数，默认值为0。

说明

mem_percent和memory_limit同时指定时，只有mem_percent起作用。

- **io_limits**

  资源池每秒可触发IO次数上限。

  以万次为单位计数。

  取值范围：数值型，0-2147483647

- **io_priority**

  IO利用率高达90%时，重消耗IO作业进行IO资源管控时关联的优先级等级。

  包括三档可选：Low、Medium和High。不控制时可设置为None，默认为None。

  取值范围：枚举型，可选项为：None，Low、Medium和High。

  说明

  io_limits和io_priority的设置都仅对复杂作业有效。包括批量导入（INSERT INTO SELECT、COPY FROM、CREATE TABLE AS等），单DN数据量超过500MB的复杂查询和VACUUM FULL等操作。

- **max_workers**

  只用于扩容的接口，表示扩容数据重分布时，表内插入并发度。

- **max_connections**

  最大连接数，用来限制资源池可使用的最大连接数。

  说明

  所有资源池的最大连接数加起来不能超过整个gaussdb进程设置的guc参数max_connections指定的最大连接数。

示例

本示例假定用户已成功创建自定义的class1控制组及其下属的Low、wg1、wg2 三个Workload控制组。

```
--创建一个资源池。
gaussdb=# CREATE RESOURCE POOL pool1 WITH (CONTROL_GROUP="Medium");

--更新一个资源池，其控制组指定为"DefaultClass"组下属的"High" Timeshare Workload控制组。
gaussdb=# ALTER RESOURCE POOL pool1 WITH (CONTROL_GROUP="High");

--更新一个资源池，其控制组指定为"class1"组下属的"Low" Timeshare Workload控制组。
gaussdb=# ALTER RESOURCE POOL pool1 WITH (CONTROL_GROUP="class1:Low");

--更新一个资源池，其控制组指定为"class1"组下属的"wg1" Workload控制组。
gaussdb=# ALTER RESOURCE POOL pool1 WITH (CONTROL_GROUP="class1:wg1");

--更新一个资源池，其控制组指定为"class1"组下属的"wg2" Workload控制组。
gaussdb=# ALTER RESOURCE POOL pool1 WITH (CONTROL_GROUP="class1:wg2:3");

--删除资源池pool1。
gaussdb=# DROP RESOURCE POOL pool1;
```


相关链接

CREATE RESOURCE POOL，DROP RESOURCE POOL

7.14.24 ALTER ROLE

功能描述

修改角色属性。

注意事项

无。

语法格式

- 修改角色的权限。

```
ALTER ROLE role_name [ [ WITH ] option [ ... ] ];
```

其中权限项子句option为。

```
{CREATEDB | NOCREATEDB}
    | {CREATEROLE | NOCREATEROLE}
    | {INHERIT | NOINHERIT}
    | {AUDITADMIN | NOAUDITADMIN}
    | {SYSADMIN | NOSYSADMIN}
    | {MONADMIN | NOMONADMIN}
    | {OPRADMIN | NOOPRADMIN}
    | {POLADMIN | NOPOLADMIN}
    | {USEFT | NOUSEFT}
    | {LOGIN | NOLOGIN}
    | {REPLICATION | NOREPLICATION}
    | {VCADMIN | NOVCADMIN}
    | {PERSISTENCE | NOPERSISTENCE}
    | CONNECTION LIMIT connlimit
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password' [EXPIRED]
    | [ ENCRYPTED | UNENCRYPTED ] IDENTIFIED BY 'password' [ REPLACE 'old_password' | EXPIRED ]
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD { 'password' | DISABLE | EXPIRED }
    | [ ENCRYPTED | UNENCRYPTED ] IDENTIFIED BY { 'password' [ REPLACE 'old_password' ] |
DISABLE }
    | VALID BEGIN 'timestamp'
    | VALID UNTIL 'timestamp'
    | RESOURCE POOL 'respool'
    | PERM SPACE 'spacelimit'
    | PGUSER
```

- 修改角色的名称。

```
ALTER ROLE role_name
    RENAME TO new_name;
```

- 锁定或解锁。

```
ALTER ROLE role_name
    ACCOUNT { LOCK | UNLOCK };
```

- 设置角色的配置参数。

```
ALTER ROLE role_name [ IN DATABASE database_name ]
    SET configuration_parameter {{ TO | = } { value | DEFAULT } | FROM CURRENT};
```

- 重置角色的配置参数。

```
ALTER ROLE role_name
    [ IN DATABASE database_name ] RESET {configuration_parameter|ALL};
```

参数说明

- **role_name**

现有角色名。

取值范围：已存在的角色名，如果角色名中包含大写字母则需要使用双引号括起来。

- **IN DATABASE database_name**

  表示修改角色在指定数据库上的参数。

- **SET configuration_parameter {{ TO | = } { value | DEFAULT } | FROM CURRENT}**

  设置角色的参数。ALTER ROLE中修改的会话参数只针对指定的角色，且在下一次该角色启动的会话中有效。

  取值范围：

  configuration_parameter和value的取值请参见**SET**。

  DEFAULT：表示清除configuration_parameter参数的值，configuration_parameter参数的值将继承本角色新产生的SESSION的默认值。

  FROM CURRENT：取当前会话中的值设置为configuration_parameter参数的值。

- **RESET {configuration_parameter|ALL}**

  清除configuration_parameter参数的值。与SET configuration_parameter TO DEFAULT的效果相同。

  取值范围：ALL表示清除所有参数的值。

- **ACCOUNT LOCK | ACCOUNT UNLOCK**
  - ACCOUNT LOCK：锁定账户，禁止登录数据库。
  - ACCOUNT UNLOCK：解锁账户，允许登录数据库。

- **PGUSER**

  当前版本不允许修改角色的PGUSER属性

- **PASSWORD/IDENTIFIED BY 'password'**

  重置或修改用户密码。除了初始用户外其他管理员或普通用户修改自己的密码需要输入正确的旧密码。只有初始用户、系统管理员（sysadmin）或拥有创建用户（CREATEROLE）权限的用户才可以重置普通用户密码，无需输入旧密码。初始用户可以重置系统管理员的密码，系统管理员不允许重置其他系统管理员的密码。应当使用单引号将用户密码括起来。

- **EXPIRED**

  设置密码失效。只有初始用户、系统管理员（sysadmin）或拥有创建用户（CREATEROLE）权限的用户才可以设置用户密码失效，其中系统管理员也可以设置自己或其他系统管理员密码失效。不允许设置初始用户密码失效。

  密码失效的用户可以登录数据库但不能执行查询操作，只有修改密码或由管理员重置密码后才可以恢复正常查询操作。

其他参数请参见CREATE ROLE的**参数说明**。

示例

请参见CREATE ROLE的**示例**。

相关链接

CREATE ROLE，**DROP ROLE**，**SET ROLE**

7.14.25 ALTER ROW LEVEL SECURITY POLICY

功能描述

对已存在的行访问控制策略（包括行访问控制策略的名称，行访问控制指定的用户，行访问控制的策略表达式）进行修改。

注意事项

表的所有者或管理员用户才能进行此操作。

语法格式

修改已存在行访问控制策略的名称

```
ALTER [ ROW LEVEL SECURITY ] POLICY policy_name ON table_name RENAME TO new_policy_name;
```

修改已存在行访问控制策略的指定用户、策略表达式

```
ALTER [ ROW LEVEL SECURITY ] POLICY policy_name ON table_name
    [ TO { role_name | PUBLIC } [, ...] ]
    [ USING ( using_expression ) ];
```

参数说明

- policy_name

  行访问控制策略名称。

- table_name

  行访问控制策略的表名。

- new_policy_name

  新的行访问控制策略名称。

- role_name

  行访问控制策略应用的数据库用户，可以指定多个用户，PUBLIC表示应用到所有用户。

- using_expression

  行访问控制策略，形式类似于where子句中的布尔型表达式。

示例

```
--创建数据表all_data。
gaussdb=# CREATE TABLE all_data(id int, role varchar(100), data varchar(100));

--创建行访问控制策略，当前用户只能查看用户自身的数据。
gaussdb=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role = CURRENT_USER);
gaussdb=# \d+ all_data
                        Table "public.all_data"
 Column |          Type          | Modifiers | Storage  | Stats target | Description
--------+------------------------+-----------+----------+--------------+-------------
 id     | integer                |           | plain    |              |
 role   | character varying(100) |           | extended |              |
 data   | character varying(100) |           | extended |              |
Row Level Security Policies:
    POLICY "all_data_rls" FOR ALL
      TO public
      USING (((role)::name = "current_user"()))
Has OIDs: no
Options: orientation=row, compression=no
```



```
--创建用户alice、bob。
gaussdb=# CREATE ROLE alice WITH PASSWORD "********";
gaussdb=# CREATE ROLE bob WITH PASSWORD "********";

--修改行访问控制all_data_rls的名称。
gaussdb=# ALTER ROW LEVEL SECURITY POLICY all_data_rls ON all_data RENAME TO all_data_new_rls;

--修改行访问控制策略影响的用户。
gaussdb=# ALTER ROW LEVEL SECURITY POLICY all_data_new_rls ON all_data TO alice, bob;
gaussdb=# \d+ all_data
                        Table "public.all_data"
 Column |          Type          | Modifiers | Storage  | Stats target | Description
--------+------------------------+-----------+----------+--------------+-------------
 id     | integer                |           | plain    |              |
 role   | character varying(100) |           | extended |              |
 data   | character varying(100) |           | extended |              |
Row Level Security Policies:
    POLICY "all_data_new_rls" FOR ALL
      TO alice,bob
      USING (((role)::name = "current_user"()))
Has OIDs: no
Options: orientation=row, compression=no, enable_rowsecurity=true

--修改行访问控制策略表达式。
gaussdb=# ALTER ROW LEVEL SECURITY POLICY all_data_new_rls ON all_data USING (id > 100 AND role =
current_user);
gaussdb=# \d+ all_data
                        Table "public.all_data"
 Column |          Type          | Modifiers | Storage  | Stats target | Description
--------+------------------------+-----------+----------+--------------+-------------
 id     | integer                |           | plain    |              |
 role   | character varying(100) |           | extended |              |
 data   | character varying(100) |           | extended |              |
Row Level Security Policies:
    POLICY "all_data_new_rls" FOR ALL
      TO alice,bob
      USING (((id > 100) AND ((role)::name = "current_user"())))
Has OIDs: no
Options: orientation=row, compression=no, enable_rowsecurity=true

--删除用户alice、bob。
gaussdb=# DROP ROLE alice, bob;

--删除访问控制策略。
gaussdb=# DROP ROW LEVEL SECURITY POLICY all_data_new_rls ON all_data;

--删除数据表all_data。
gaussdb=# DROP TABLE all_data;
```

相关链接

CREATE ROW LEVEL SECURITY POLICY，**DROP ROW LEVEL SECURITY POLICY**

7.14.26 ALTER SCHEMA

功能描述

修改模式属性。

注意事项

- 只有模式的所有者或者被授予了模式ALTER权限的用户有权限执行ALTER SCHEMA命令，系统管理员默认拥有此权限。但要修改模式的所有者，当前用户必须是该模式的所有者或者系统管理员，且该用户是新所有者角色的成员。

- 对于除public以外的系统模式，如pg_catalog、sys等，只允许初始用户修改模式的所有者。修改系统自带模式的名称可能会导致部分功能不可用甚至影响数据库正常运行，默认情况下不允许修改系统自带模式的名称，考虑到前向兼容性，仅允许当系统在启动或升级过程中或参数allow_system_table_mods为on时修改。
- 只有初始用户可以将schema的属主修改为运维管理员，其他用户无法将schema的属主修改为运维管理员。

语法格式

- 修改模式的名称。
```
ALTER SCHEMA schema_name
    RENAME TO new_name;
```
- 修改模式的所有者。
```
ALTER SCHEMA schema_name
    OWNER TO new_owner;
```
- 修改模式的默认字符集和字符序。
```
ALTER SCHEMA schema_name
    [ [DEFAULT] CHARACTER SET | CHARSET [ = ] default_charset ] [ [DEFAULT] COLLATE [ = ] default_collation ];
```

参数说明

- **schema_name**

  现有模式的名称。

  取值范围：已存在的模式名。
- **RENAME TO new_name**

  修改模式的名称。非系统管理员要改变模式的名称，则该用户必须在此数据库上有CREATE权限。

  new_name：模式的新名称。

  须知

  - 模式名不能和当前数据库里其他的模式重名。
  - 模式名不能和当前数据库的初始用户重名。
  - 模式的名称不可以“pg_”开头。
  - 模式的名称不可以“gs_role_”开头。

  取值范围：字符串，要符合**标识符命名规范**。
- **OWNER TO new_owner**

  修改模式的所有者。非系统管理员要改变模式的所有者，该用户还必须是新的所有角色的直接或间接成员， 并且该成员必须在此数据库上有CREATE权限。

  new_owner：模式的新所有者。

  取值范围：已存在的用户名/角色名。
- **default_charset**

  修改模式的默认字符集，单独指定时会将模式的默认字符序设置为指定的字符集的默认字符序。

  仅在sql_compatibility='B'时支持该语法。支持字符集参见**表7-146**。

- **default_collate**

  修改模式的默认字符序，单独指定时会将模式的默认字符集设置为指定的字符序对应的字符集。

  仅在sql_compatibility='B'时支持该语法。支持字符序参见**表7-146**。

示例

```
--创建模式ds。
gaussdb=# CREATE SCHEMA ds;

--将当前模式ds更名为ds_new。
gaussdb=# ALTER SCHEMA ds RENAME TO ds_new;

--创建用户jack。
gaussdb=# CREATE USER jack PASSWORD '********';

--将DS_NEW的所有者修改为jack。
gaussdb=# ALTER SCHEMA ds_new OWNER TO jack;

--将DS_NEW的默认字符集修改为utf8mb4，默认字符序修改为utf8mb4_bin。仅在sql_compatibility='B'时支持该语法。
gaussdb=# ALTER SCHEMA ds_new CHARACTER SET utf8mb4 COLLATE utf8mb4_bin;

--删除用户jack和模式ds_new。
gaussdb=# DROP SCHEMA ds_new;
gaussdb=# DROP USER jack;
```

相关链接

CREATE SCHEMA，**DROP SCHEMA**

7.14.27 ALTER SEQUENCE

功能描述

修改一个现有的序列的参数。

注意事项

- 序列的所有者或者被授予了序列ALTER权限的用户或者被授予了ALTER ANY SEQUENCE权限的用户才能执行ALTER SEQUENCE命令，系统管理员默认拥有该权限。但要修改序列的所有者，当前用户必须是该序列的所有者或者系统管理员，且该用户是新所有者角色的成员。
- 当前版本仅支持修改拥有者、归属列和最大值。若要修改其他参数，可以删除重建，并用Setval函数恢复当前值。
- ALTER SEQUENCE MAXVALUE不支持在事务、函数和存储过程中使用。
- 修改序列的最大值后，会清空该序列在所有会话的cache。
- 如果Sequence被创建时使用了LARGE标识，则ALTER时也需要使用LARGE标识。
- ALTER SEQUENCE会阻塞nextval、setval、currval和lastval的调用。

语法格式

- 修改序列归属列

```
ALTER [ LARGE ] SEQUENCE [ IF EXISTS ] name
    [MAXVALUE maxvalue | NO MAXVALUE | NOMAXVALUE | CACHE cache]
    [ OWNED BY { table_name.column_name | NONE } ] ;
```


- 修改序列的拥有者
  ```
  ALTER [ LARGE ] SEQUENCE [ IF EXISTS ] name OWNER TO new_owner;
  ```

参数说明

- **name**

  将要修改的序列名称。

- **IF EXISTS**

  当序列不存在时使用该选项不会出现错误消息，仅有一个通知。

- **CACHE**

  为了快速访问，而在内存中预先存储序列号的个数。如果没有指定，将保持旧的缓冲值。

- **OWNED BY**

  将序列和一个表的指定字段进行关联。这样，在删除那个字段或其所在表的时候会自动删除已关联的序列。

  如果序列已经和表有关联后，使用这个选项后新的关联关系会覆盖旧的关联。

  关联的表和序列的所有者必须是同一个用户，并且在同一个模式中。

  使用OWNED BY NONE将删除任何已经存在的关联。

- **new_owner**

  序列新所有者的用户名。用户要修改序列的所有者，必须是新角色的直接或者间接成员，并且那个角色必须有序列所在模式上的CREATE权限。

示例

```
--创建一个名为serial的递增序列，从101开始。
gaussdb=# CREATE SEQUENCE serial START 101;

--创建一个表，定义默认值。
gaussdb=# CREATE TABLE T1(C1 bigint default nextval('serial'));

--将序列serial的归属列变为T1.C1。
gaussdb=# ALTER SEQUENCE serial OWNED BY T1.C1;

--删除序列和表。
gaussdb=# DROP SEQUENCE serial cascade;
gaussdb=# DROP TABLE T1;
```

相关链接

CREATE SEQUENCE，**DROP SEQUENCE**

7.14.28 ALTER SERVER

功能描述

增加、修改和删除一个现有server的参数。已有server可以从pg_foreign_server系统表中查询。

注意事项

- 只有SERVER的所有者或者被授予了SERVER的ALTER权限的用户才可以执行ALTER SERVER命令，系统管理员默认拥有该权限。但要修改SERVER的所有者，当前用户必须是该SERVER的所有者或者系统管理员，且该用户是新所有者角色的成员。

- OPTIONS中的敏感字段（如password、secret_access_key）在使用多层引号时，语义和不带引号的场景是不同的，因此不会被识别为敏感字段进行脱敏。

语法格式

- 修改外部服务的参数。

```
ALTER SERVER server_name [ VERSION 'new_version' ]
    [ OPTIONS ( {[ ADD | SET | DROP ] option ['value']} [, ... ] ) ];
```

  在OPTIONS选项里，ADD、SET和DROP指定要执行的操作，未指定时默认为ADD操作。option和value为对应操作的参数。

- 修改外部服务的所有者。

```
ALTER SERVER server_name
  OWNER TO new_owner;
```

- 修改外部服务的名称。

```
ALTER SERVER server_name
  RENAME TO new_name;
```

参数说明

- **server_name**

  所修改的server的名称。

- **new_version**

  修改后server的新版本名称。

- **OPTIONS**

  更改该服务器的选项。ADD、SET和 DROP指定要执行的动作。如果没有显式地指定操作， 将会假定为ADD。选项名称必须唯一，名称和值也会使用该服务器的外部数据包装器库进行验证。

  除了libpq支持的连接参数外，还额外提供以下参数：

  - **fdw_startup_cost**

    执行一个外表扫描时的启动耗时估算。这个值通常包含建立连接、远端对请求的分析和生成计划的耗时。默认值为100。取值范围为大于0的实数。

  - **fdw_typle_cost**

    在远端服务器上对每一个元组进行扫描时的额外消耗。这个值通常表示数据在server间传输的额外消耗。默认值为0.01。取值范围为大于0的实数。

- **new_name**

  修改后server的新名称。

示例

```
--创建my_server。
gaussdb=# CREATE SERVER my_server FOREIGN DATA WRAPPER log_fdw;

--修改外部服务的名称。
gaussdb=# ALTER SERVER my_server
   RENAME TO my_server_1;

--删除my_server_1。
gaussdb=# DROP SERVER my_server_1;
```

相关链接

CREATE SERVER，**DROP SERVER**

7.14.29 ALTER SESSION

功能描述

ALTER SESSION命令用于定义或修改那些对当前会话有影响的条件或参数。修改后的会话参数会一直保持，直到断开当前会话。

注意事项

- 如果执行SET TRANSACTION之前没有执行START TRANSACTION，则事务立即结束，命令无法显示效果。
- 可以用START TRANSACTION里面声明所需要的transaction_mode(s)的方法来避免使用SET TRANSACTION。具体请参见：**START TRANSACTION**。

语法格式

- 设置会话的事务参数。

```
ALTER SESSION SET [ SESSION CHARACTERISTICS AS ] TRANSACTION
   { ISOLATION LEVEL { READ COMMITTED } | { READ ONLY  | READ WRITE } } [, ...] ;
```

- 设置会话的其他运行时参数。

```
ALTER SESSION SET
   {{config_parameter { { TO  | =  }  { value | DEFAULT }
     | FROM CURRENT }}
     | TIME ZONE time_zone
     | CURRENT_SCHEMA schema
     | NAMES encoding_name
     | ROLE role_name PASSWORD 'password'
     | SESSION AUTHORIZATION { role_name PASSWORD 'password' | DEFAULT }
     | XML OPTION { DOCUMENT | CONTENT }
   } ;
```

参数说明

- **config_parameter**

  可设置的运行时参数的名称。可用的运行时参数可以使用SHOW ALL命令查看。

  - **value**

    config_parameter的新值。可以声明为字符串常量、标识符、数字，或者逗号分隔的列表。DEFAULT用于把这些参数设置为它们的缺省值。

    - DEFAULT
    - OFF
    - RESET
    - 用户指定的值：需要满足修改参数的取值限制

  - **FROM CURRENT**

    取当前会话中的值设置为configuration_parameter的值。

- **TIME ZONE timezone**

  用于指定当前会话的本地时区。

  取值范围：有效的本地时区。该选项对应的运行时参数名称为TimeZone，DEFAULT缺省值为PRC。

- **CURRENT_SCHEMA**

schema

CURRENT_SCHEMA用于指定当前的模式。

取值范围：已存在模式名称。如果模式名不存在，会导致CURRENT_SCHEMA值为空。

- **SCHEMA schema**

  同CURRENT_SCHEMA。此处的schema是个字符串。

- **NAMES encoding_name**

  用于设置客户端的字符编码。等价于set client_encoding to encoding_name。

  取值范围：有效的字符编码。该选项对应的运行时参数名称为client_encoding，默认编码为UTF8。

- **role_name**

  取值范围：字符串。要符合**标识符命名规范**。

- **password**

  角色的密码。要求符合密码的命名规则。

- **SESSION AUTHORIZATION**

  当前会话的用户表示符。

- **XML OPTION { DOCUMENT | CONTENT }**

  用于设置XML的解析方式。

  取值范围：CONTENT（缺省）、DOCUMENT

示例

```
--创建模式ds。
gaussdb=# CREATE SCHEMA ds;

--设置模式搜索路径。
gaussdb=# SET SEARCH_PATH TO ds, public;

--设置日期时间风格为传统的POSTGRES风格（日在月前）。
gaussdb=# SET DATESTYLE TO postgres, dmy;

--设置当前会话的字符编码为UTF8。
gaussdb=# ALTER SESSION SET NAMES 'UTF8';

--设置时区为加州伯克利。
gaussdb=# SET TIME ZONE 'PST8PDT';

--设置时区为意大利。
gaussdb=# SET TIME ZONE 'Europe/Rome';

--设置当前模式。
gaussdb=# ALTER SESSION SET CURRENT_SCHEMA TO tpcds;

--设置XML OPTION为DOCUMENT。
gaussdb=# ALTER SESSION SET XML OPTION DOCUMENT;

--创建角色joe，并设置会话的角色为joe。
gaussdb=# CREATE ROLE joe WITH PASSWORD '********';
gaussdb=# ALTER SESSION SET SESSION AUTHORIZATION joe PASSWORD '*********';

--切换到默认用户。
gaussdb=> ALTER SESSION SET SESSION AUTHORIZATION default;

--删除ds模式。
gaussdb=# DROP SCHEMA ds;
```



```
--删除joe。
gaussdb=# DROP ROLE joe;

--开启事务,设置事务级别
gaussdb=# START TRANSACTION;
gaussdb=# ALTER SESSION SET TRANSACTION READ ONLY;
gaussdb=# END;
```

相关链接

SET

7.14.30 ALTER SYNONYM

功能描述

修改SYNONYM对象的属性。

注意事项

- 目前仅支持修改SYNONYM对象的属主。
- 只有系统管理员有权限修改SYNONYM对象的属主信息。
- 新属主必须具有SYNONYM对象所在模式的CREATE权限。

语法格式

```
ALTER SYNONYM synonym_name
    OWNER TO new_owner;
```

参数描述

- **synonym_name**
  待修改的同义词名字，可以带模式名。
  取值范围：字符串，需要符合**标识符命名规范**。
- **new_owner**
  同义词对象的新所有者。
  取值范围：字符串，有效的用户名。

示例

```
--创建系统管理员用户。
gaussdb=# CREATE USER sysadmin WITH SYSADMIN PASSWORD '********';

--切换管理员用户。
gaussdb=# \c - sysadmin

--创建同义词t1。
gaussdb=# CREATE OR REPLACE SYNONYM t1 FOR ot.t1;

--创建新用户u1。
gaussdb=# CREATE USER u1 PASSWORD '********';

--给新用户赋权限。
gaussdb=# GRANT ALL ON SCHEMA sysadmin TO u1;

--修改同义词t1的owner为u1。
gaussdb=# ALTER SYNONYM t1 OWNER TO u1;
```



```
--删除同义词t1。
gaussdb=# DROP SYNONYM t1;

--收回用户u1权限。
gaussdb=# REVOKE ALL ON SCHEMA sysadmin FROM u1;

--删除用户u1。
gaussdb=# DROP USER u1;

--切换到初始用户init_user，请使用真实的初始用户名称替换init_user。
gaussdb=# \c - init_user

--删除用户sysadmin。
gaussdb=# DROP USER sysadmin;
```

相关链接

CREATE SYNONYM，**DROP SYNONYM**

7.14.31 ALTER SYSTEM KILL SESSION

功能描述

ALTER SYSTEM KILL SESSION命令用于结束一个会话。

注意事项

无。

语法格式

```
ALTER SYSTEM KILL SESSION 'session_sid, serial' [ IMMEDIATE ];
```

参数说明

- **session_sid, serial**

  会话的SID和SERIAL（获取方法请参考示例）。可通过pg_stat_activity系统表配合查询当前活跃线程（可见示例），但执行ALTER SYSTEM KILL SESSION命令时线程可能已结束。

  取值范围：通过查看系统表dv_sessions可查看所有会话的SID和SERIAL。

- **IMMEDIATE**

  表明会话将在命令执行后立即结束。

示例

```
--查询会话信息。
gaussdb=# SELECT sa.sessionid AS sid,0::integer AS serial#,ad.rolname AS username FROM
pg_stat_get_activity(NULL) AS sa LEFT JOIN pg_authid ad ON(sa.usesysid = ad.oid)WHERE
sa.application_name <> 'JobScheduler';
       sid        | serial# | username
------------------+---------+----------
 140131075880720 |       0 | omm
 140131025549072 |       0 | omm
 140131073779472 |       0 | omm
 140131071678224 |       0 | omm
 140131125774096 |       0 |
 140131127875344 |       0 |
 140131113629456 |       0 |
 140131094742800 |       0 |
```



```
(8 rows)

--结束SID为140131075880720的会话。
gaussdb=# ALTER SYSTEM KILL SESSION '140131075880720,0' IMMEDIATE;
```

7.14.32 ALTER TABLE

功能描述

修改表，包括修改表的定义、重命名表、重命名表中指定的列、重命名表的约束、设置表的所属模式、添加/更新多个列、打开/关闭行访问控制开关。

注意事项

- 表的所有者、被授予了表ALTER权限的用户或被授予ALTER ANY TABLE权限的用户有权限执行ALTER TABLE命令，系统管理员默认拥有此权限。但要修改表的所有者或者修改表的模式，当前用户必须是该表的所有者或者系统管理员，且该用户是新所有者角色的成员。
- 不能修改分区表的TABLESPACE，但可以修改分区的TABLESPACE。
- 不支持修改存储参数ORIENTATION。
- SET SCHEMA操作不支持修改为系统内部模式，当前仅支持用户模式之间的修改。
- 不支持增加DEFAULT值中包含nextval()表达式的列。
- 不支持对外表、临时表开启行访问控制开关。
- 通过约束名删除PRIMARY KEY约束时，不会删除NOT NULL约束，如果有需要，请手动删除NOT NULL约束。
- 使用JDBC时，支持通过PrepareStatement对DEFAULT值进行参数化设置。
- 如果用ADD COLUMN增加一个字段，那么所有表中现有行都初始化为该字段的缺省值（如果没有声明DEFAULT子句，那么就是 NULL）。

  新增列没有声明DEFAULT值时，默认值为NULL，不会触发全表更新。

  新增列如果有DEFAULT值，必须符合以下所有要求，否则会带来全表更新开销，影响在线业务：

  1. 数据类型为以下类型中的一种：BOOL, BYTEA, SMALLINT, BIGINT, SMALLINT, INTEGER, NUMERIC, FLOAT, DOUBLE PRECISION, CHAR, VARCHAR, TEXT, TIMESTAMPTZ, TIMESTAMP, DATE, TIME, TIMETZ, INTERVAL, SERIAL, BIGSERIAL, SMALLSERIAL；

  2. 新增列的DEFAULT值长度不超过128个字节；

  3. 新增列DEFAULT值不包含易变（volatile）函数；

  4. 新增列设置有DEFAULT值，且DEFAULT值不为NULL。

  如果不确定是否满足条件3，可以查询PG_RPOC系统表中函数的provolatile属性是否为‘v’。

  5. 在开启PG兼容模式下，支持新增列数据类型为SERIAL、BIGSERIAL和SMALLSERIAL，且仅只支持普通表，不支持分区表、临时表、unlogged permanent table、支持astore和ustore存储引擎。
- 使用FIRST | AFTER column_name新增列或修改列，或修改字段的字符集，会带来全表更新开销，影响在线业务。
- 表约束个数不能超过32767个。

语法格式

- 修改表的定义。

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }
    action [, ... ];
ALTER TABLE [ IF EXISTS ] table_name
    ADD ( { column_name data_type [ compress_mode ] [ COLLATE collation ] [ column_constraint
[ ... ] ]} [, ...] );
ALTER TABLE [ IF EXISTS ] table_name
    MODIFY ( { column_name data_type | column_name [ CONSTRAINT constraint_name ] NOT NULL
[ ENABLE ] | column_name [ CONSTRAINT constraint_name ] NULL } [, ...] );
ALTER TABLE [ IF EXISTS ] table_name
    RENAME TO new_table_name;
ALTER TABLE [ IF EXISTS ] { table_name  [*] | ONLY table_name | ONLY ( table_name  )}
    RENAME [ COLUMN ] column_name TO new_column_name;
ALTER TABLE [ IF EXISTS ] { table_name  [*] | ONLY table_name | ONLY ( table_name  )}
    RENAME CONSTRAINT constraint_name TO new_constraint_name;
ALTER TABLE [ IF EXISTS ] table_name
    SET SCHEMA new_schema;
```

其中具体表操作action可以是以下子句之一：

```
column_clause
    | ADD table_constraint [ NOT VALID ]
    | ADD table_constraint_using_index
    | VALIDATE CONSTRAINT constraint_name
    | DROP CONSTRAINT [ IF EXISTS ]  constraint_name [ RESTRICT | CASCADE ]
    | CLUSTER ON index_name
    | SET WITHOUT CLUSTER
    | SET ( {storage_parameter = value} [, ... ] )
    | RESET ( storage_parameter [, ... ] )
    | OWNER TO new_owner
    | SET TABLESPACE new_tablespace
    | SET {COMPRESS|NOCOMPRESS}
    | TO { GROUP groupname | NODE ( nodename [, ... ] ) }
    | ADD NODE ( nodename [, ... ] )
    | DELETE NODE ( nodename [, ... ] )
    | UPDATE SLICE LIKE table_name
    | DISABLE TRIGGER [ trigger_name | ALL | USER ]
    | ENABLE TRIGGER [ trigger_name | ALL | USER ]
    | ENABLE REPLICA TRIGGER trigger_name
    | ENABLE ALWAYS TRIGGER trigger_name
    | ENABLE ROW LEVEL SECURITY
    | DISABLE ROW LEVEL SECURITY
    | FORCE ROW LEVEL SECURITY
    | NO FORCE ROW LEVEL SECURITY

    | REPLICA IDENTITY { DEFAULT | USING INDEX index_name | FULL | NOTHING }
    | AUTO_INCREMENT [ = ] value
    | [ [ DEFAULT ] CHARACTER SET | CHARSET [ = ] default_charset ] [ [ DEFAULT ] COLLATE [ = ]
default_collation ]
    | CONVERT TO CHARACTER SET | CHARSET charset | DEFAULT [ COLLATE collation ]
```


说明

- **ADD table_constraint [NOT VALID]**

  给表增加一个新的约束。

- **ADD table_constraint_using_index**

  根据已有唯一索引为表增加主键约束或唯一约束。

- **VALIDATE CONSTRAINT constraint_name**

  验证一个使用NOT VALID选项创建的检查类约束，通过扫描全表来保证所有记录都符合约束条件。如果约束已标记为有效时，什么操作也不会发生。

- **DROP CONSTRAINT [IF EXISTS] constraint_name [RESTRICT | CASCADE]**

  删除一个表上的约束。

- **CLUSTER ON index_name**

  为将来的CLUSTER（聚簇）操作选择默认索引。实际上并没有重新盘簇化处理该表。

- **SET WITHOUT CLUSTER**

  从表中删除最新使用的CLUSTER索引。这样会影响将来那些没有声明索引的CLUSTER（聚簇）操作。

- **SET ({storage_parameter = value} [, ...])**

  修改表的一个或多个存储参数。当table_name为索引名时，ACTIVE_PAGES表示索引的页面数量，可能比实际的物理文件页面少，可以用于优化器调优。目前只对Ustore的分区表local索引生效，且会被vacuum、analyze更新（包括 auto vacuum）。不建议用户手动设置该参数。

- **RESET (storage_parameter [, ...])**

  重置表的一个或多个存储参数。与SET一样，根据参数的不同可能需要重写表才能获得想要的效果。

- **OWNER TO new_owner**

  将表、序列、视图的属主改变成指定的用户。

- **SET TABLESPACE new_tablespace**

  这种形式将表空间修改为指定的表空间并将相关的数据文件移动到新的表空间。但是表上的所有索引都不会被移动，索引可以通过ALTER INDEX语法的SET TABLESPACE选项来修改索引的表空间。

- **SET {COMPRESS|NOCOMPRESS}**

  修改表的压缩特性。表压缩特性的改变只会影响后续批量插入的数据的存储方式，对已有数据的存储毫无影响。也就是说，表压缩特性的修改会导致该表中同时存在着已压缩和未压缩的数据。行存表不支持压缩。

- **TO { GROUP groupname | NODE (nodename [, ...]) }**

  此语法仅在扩展模式（GUC参数support_extended_features为on时）下可用。该模式谨慎打开，主要供内部扩容工具使用，一般用户不应使用该模式。

- **ADD NODE (nodename [, ...])**

  此语法主要供内部扩容工具使用，一般用户不建议使用。

- **DELETE NODE (nodename [, ...])**

  此语法主要供内部缩容工具使用，一般用户不建议使用。

- **UPDATE SLICE LIKE table_name**

  此语法主要供内部扩缩容工具使用，一般用户不可以使用。

- **DISABLE TRIGGER [trigger_name | ALL | USER]**

  禁用trigger_name所表示的单个触发器，或禁用所有触发器，或仅禁用用户触发器（此选项不包括内部生成的约束触发器，例如，可延迟唯一性和排除约束的约束触发器）。

  应谨慎使用此功能，因为如果不执行触发器，则无法保证原先期望的约束的完整性。

- **| ENABLE TRIGGER [trigger_name | ALL | USER]**

启用trigger_name所表示的单个触发器，或启用所有触发器，或仅启用用户触发器。

- **| ENABLE REPLICA TRIGGER trigger_name**

  触发器触发机制受GUC配置变量**session_replication_role**的影响，当复制角色为“origin”（默认值）或“local”时，将触发简单启用的触发器。

  配置为ENABLE REPLICA的触发器仅在会话处于“replica”模式时触发。

- **| ENABLE ALWAYS TRIGGER trigger_name**

  无论当前复制模式如何，配置为ENABLE ALWAYS的触发器都将触发。

- **| { DISABLE | ENABLE } ROW LEVEL SECURITY**

  开启或关闭表的行访问控制开关。

  当开启行访问控制开关时，如果未在该数据表定义相关行访问控制策略，数据表的行级访问将不受影响；如果关闭表的行访问控制开关，即使定义了行访问控制策略，数据表的行访问也不受影响。详细信息参见**CREATE ROW LEVEL SECURITY POLICY**章节。

- **| { NO FORCE | FORCE } ROW LEVEL SECURITY**

  强制开启或关闭表的行访问控制开关。

  默认情况，表所有者不受行访问控制特性影响，但当强制开启表的行访问控制开关时，表的所有者（不包含系统管理员用户）会受影响。系统管理员可以绕过所有的行访问控制策略，不受影响。

- **REPLICA IDENTITY { DEFAULT | USING INDEX index_name | FULL | NOTHING }**

  在逻辑复制场景下，指定该表的UPDATE和DELETE操作中旧元组的记录级别。

  - DEFAULT记录主键的列的旧值，没有主键则不记录。
  - USING INDEX记录命名索引覆盖的列的旧值，这些值必须是唯一的、不局部的、不可延迟的，并且仅包括标记为NOT NULL的列。
  - FULL记录该行中所有列的旧值。
  - NOTHING不记录有关旧行的信息。

  在逻辑复制场景，解析该表的UPDATE和DELETE操作语句时，解析出的旧元组由以此方法记录的信息组成。对于有主键表该选项可设置为DEFAULT或FULL。对于无主键表该选项需设置为FULL，否则解码时旧元组将解析为空。一般场景不建议设置为NOTHING，旧元组会始终解析为空。

  即使指定DEFAULT或USING INDEX，当前ustore表列的旧值中也可能包含该行所有列的旧值，只有旧值涉及toast该配置选项才会生效。另外针对ustore表，选项NOTHING无效，实际效果等同于FULL。

- **AUTO_INCREMENT [=] value**

  设置自动增长列下一次的自增值。设置的值只有大于当前自增计数器时才会生效。

  value必须是非负整数，且不得大于2^{127}-1。

  此子句仅在参数sql_compatibility='B'时生效。

- **[[DEFAULT] CHARACTER SET | CHARSET [=] default_charset] [[DEFAULT] COLLATE [=] default_collation]**

  修改表的默认字符集和默认字符序为指定的值。修改不会影响表中当前已经存在的列。

  此子句仅在参数sql_compatibility='B'时生效。

- **CONVERT TO CHARACTER SET | CHARSET charset [COLLATE collation]**

  修改表的默认字符集和默认字符序为指定的值，同时将表中的所有字符类型的字段的字符集和字符序设置为指定的值，并将字段里的数据转换为新字符集编码。

– 其中列相关的操作column_clause可以是以下子句之一：

```
ADD [ COLUMN ] column_name data_type [ CHARACTER SET | CHARSET  charset ]
[ compress_mode ] [ COLLATE collation ] [ column_constraint [ ... ] ] [ FIRST | AFTER
column_name ]
| MODIFY column_name data_type
| MODIFY column_name [ CONSTRAINT constraint_name ] NOT NULL [ ENABLE ]
| MODIFY column_name [ CONSTRAINT constraint_name ] NULL
| MODIFY [ COLUMN ] column_name data_type [ CHARACTER SET | CHARSET charset ]
[{[ COLLATE collation ] | [ column_constraint ]} [ ... ] ] [FIRST | AFTER column_name]
```



```
| CHANGE [ COLUMN ] column_name new_column_name data_type [ CHARACTER SET |
CHARSET charset ] [{[ COLLATE collation ] | [ column_constraint ]} [ ... ] ] [FIRST | AFTER
column_name]
| DROP [ COLUMN ] [ IF EXISTS ] column_name [ RESTRICT | CASCADE ]
| ALTER [ COLUMN ] column_name [ SET DATA ] TYPE data_type [ COLLATE collation ] [ USING
expression ]
| ALTER [ COLUMN ] column_name { SET DEFAULT expression | DROP DEFAULT }
| ALTER [ COLUMN ] column_name { SET | DROP } NOT NULL
| ALTER [ COLUMN ] column_name SET STATISTICS [PERCENT] integer
| ADD STATISTICS (( column_1_name, column_2_name [, ...] ))
| DELETE STATISTICS (( column_1_name, column_2_name [, ...] ))
| ALTER [ COLUMN ] column_name SET ( {attribute_option = value} [, ... ] )
| ALTER [ COLUMN ] column_name RESET ( attribute_option [, ... ] )
| ALTER [ COLUMN ] column_name SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }
```

说明

- **ADD [COLUMN] column_name data_type [CHARACTER SET | CHARSET [=] charset] [compress_mode] [COLLATE collation] [column_constraint [...]] [FIRST | AFTER column_name]**

  向表中增加一个新的字段。用ADD COLUMN增加一个字段，所有表中现有行都初始化为该字段的缺省值（如果没有声明DEFAULT子句，值为NULL）。其中FIRST | AFTER column_name表示新增字段到某个位置。

- **ADD ({ column_name data_type [compress_mode] } [, ...])**

  向表中增加多列。

- **MODIFY ({ column_name data_type | column_name [CONSTRAINT constraint_name] NOT NULL [ENABLE] | column_name [CONSTRAINT constraint_name] NULL } [, ...])**

  修改表已存在字段的数据类型。此命令会导致该字段的统计信息清空，建议在修改后重新收集该列的统计信息。

- **MODIFY [COLUMN] column_name data_type [CHARACTER SET | CHARSET charset] [{[COLLATE collation] | [column_constraint]} [...]] [FIRST | AFTER column_name]**

  修改表已存在字段的定义，将用新定义替换字段原定义，原字段上的索引、独立对象约束（例如：主键、唯一键、CHECK约束等）不会被删除。[FIRST | AFTER column_name]语法表示修改字段定义的同时修改字段在表中的位置。

  此语法只能在参数sql_compatibility='B'时使用。不支持外表，不支持修改加密字段，不支持修改分区键字段的数据类型和排序规则，不支持修改规则引用的字段的数据类型和排序规则，不支持修改物化视图引用的字段的数据类型和排序规则。

  被修改数据类型或排序规则的字段如果被一个生成列引用，这个生成列的数据将会重新生成。

  被修改字段若被一些对象依赖（比如：索引、独立对象约束、视图、触发器、行级访问控制策略等），修改字段过程中将会重建这些对象。若被修改后字段定义违反此类对象的约束，修改操作会失败，比如：修改作为视图结果列的字段的数据类型。请修改字段前评估这类影响。

  被修改字段若被一些对象调用（比如：自定义函数、存储过程等），修改字段不会处理这些对象。修改字段完毕后，这些对象有可能出现不可用的情况，请修改字段前评估这类影响。

  修改字段的字符集或字符序会将字段中的数据转换为新的字符集进行编码。

  此子句与上一子句中“MODIFY column_name data_type”部分语法相同，语义功能不同，当GUC参数b_format_behavior_compat_options含有'enable_modify_column'选项时，将按照此子句功能处理。

  此命令会导致该字段的统计信息清空，建议在修改后重新收集该列的统计信息。

- **CHANGE [COLUMN] old_column_name new_column_name data_type [CHARACTER SET | CHARSET charset] [{[COLLATE collation] | [column_constraint]} [...]] [FIRST | AFTER column_name]**

  修改表已存在字段的名称和定义，字段新名称不能是已有字段的名称，将用新名称和定义替换字段原名称和定义原字段上的索引、独立对象约束（例如：主键、唯一键、CHECK约束）等不会被删除。[FIRST | AFTER column_name]语法表示修改字段名称和定义的同时修改字段在表中的位置。

  此语法只能在参数sql_compatibility='B'时使用。不支持外表。不支持修改加密字段，不支持修改分区键字段的数据类型和排序规则，不支持修改规则引用的字段的数据类型和排序规则，不支持修改物化视图引用的字段的数据类型和排序规则

  被修改数据类型或排序规则的字段如果被一个生成列引用，这个生成列的数据将会重新生成。

  被修改字段若被一些对象依赖（比如：索引、独立对象约束、视图、触发器、行级访问控制策略等），修改字段过程中将会重建这些对象。若被修改后字段定义违反此类对象的约束，修改操作会失败，比如：修改作为视图结果列的字段的数据类型。请修改字段前评估这类影响。

被修改字段若被一些对象调用（比如：自定义函数、存储过程等），修改字段不会处理这些对象。修改字段名称后，这些对象有可能出现不可用的情况，请修改字段前评估这类影响。

修改字段的字符集或字符序会将字段中的数据转换为新的字符集进行编码。

- **DROP [COLUMN] [IF EXISTS] column_name [RESTRICT | CASCADE]**

  从表中删除一个字段，和这个字段相关的索引和表约束也会被自动删除。如果任何表之外的对象依赖于这个字段，必须声明CASCADE ，比如视图。

  DROP COLUMN命令并不是物理上把字段删除，而只是简单地把它标记为对SQL操作不可见。随后对该表的插入和更新将在该字段存储一个NULL。因此，删除一个字段是很快的，但是它不会立即释放表在磁盘上的空间，因为被删除了的字段占据的空间还没有回收。这些空间将在执行VACUUM时而得到回收。

- **ALTER [COLUMN] column_name [SET DATA] TYPE data_type [COLLATE collation] [USING expression]**

  改变表字段的数据类型。该字段涉及的索引和简单的表约束将被自动地转换为使用新的字段类型，方法是重新分析最初提供的表达式。

  当字段的原始数据类型和修改后的数据类型二进制兼容时，执行该语句不需要对整表进行重写，其他场景下会进行整表重写。原始数据类型和目标数据类型是否二进制兼容可以在PG_CAST系统表中查看，如果castmethod为'b'则二进制兼容。例如源表中数据类型是text类型，如果转为int类型则会触发表重写，转为clob类型则不会触发表重写。如果表重写被触发，该表上被删除的空间也将被立刻回收。

  此命令会导致该字段的统计信息清空，建议在修改后重新收集该列的统计信息。

- **ALTER [COLUMN] column_name { SET DEFAULT expression | DROP DEFAULT }**

  为一个字段设置或者删除缺省值。请注意缺省值只应用于随后的INSERT命令，它们不会修改表中已经存在的行。也可以为视图创建缺省，这个时候它们是在视图的ON INSERT规则应用之前插入到INSERT句中的。

- **ALTER [COLUMN] column_name { SET | DROP } NOT NULL**

  修改一个字段是否允许NULL值或者拒绝NULL值。如果表在字段中包含非NULL，则只能使用SET NOT NULL。

- **ALTER [COLUMN] column_name SET STATISTICS [PERCENT] integer**

  为随后的ANALYZE操作设置针对每个字段的统计收集目标。目标的范围可以在0到10000之内设置。设置为-1时表示重新恢复到使用系统缺省的统计目标。

- **{ADD | DELETE} STATISTICS ((column_1_name, column_2_name [, ...]))**

  用于添加和删除多列统计信息声明（不实际进行多列统计信息收集），以便在后续进行全表或全库analyze时进行多列统计信息收集。如果关闭GUC参数enable_functional_dependency，每组多列统计信息最多支持32列；如果开启GUC参数enable_functional_dependency，每组多列统计信息最多支持4列。不支持添加/删除多列统计信息声明的表：系统表、外表。

- **{ENABLE | DISABLE } STATISTICS ((column_1_name, column_2_name [, ...]))**

  用于启用和禁用多列统计信息。在开启自动创建统计信息的场景下（需使用GUC参数**auto_statistic_ext_columns**），禁用特定的多列组合，防止被自动创建出来并使用。

- **ALTER [COLUMN] column_name SET ({attribute_option = value} [, ...])**

  ALTER [COLUMN] column_name RESET (attribute_option [, ...])

  设置/重置属性选项。

  目前，属性选项只定义了n_distinct和n_distinct_inherited。n_distinct影响表本身的统计值，而n_distinct_inherited影响表及其继承子表的统计。目前，只支持SET/RESET n_distinct参数，禁止SET/RESET n_distinct_inherited参数。

- **ALTER [COLUMN] column_name SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }**

为一个字段设置存储模式。这个设置控制这个字段是内联保存还是保存在一个附属的表里，以及数据是否要压缩。。SET STORAGE本身并不改变表上的任何东西，只是设置将来的表操作时，建议使用的策略。

- 其中列约束column_constraint为：

```
[ CONSTRAINT constraint_name ]
  { NOT NULL |
    NULL |
    CHECK ( expression ) |
    DEFAULT default_expr  |
    GENERATED ALWAYS AS ( generation_expr ) [STORED] |
    AUTO_INCREMENT |
    UNIQUE [KEY] index_parameters |
    PRIMARY KEY index_parameters |
    ENCRYPTED WITH ( COLUMN_ENCRYPTION_KEY = column_encryption_key,
ENCRYPTION_TYPE = encryption_type_value )
|
    REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH
SIMPLE ]
        [ ON DELETE action ] [ ON UPDATE action ] }    [ DEFERRABLE | NOT DEFERRABLE
| INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

- 其中列的压缩可选项compress_mode为：

```
{ DELTA | PREFIX | DICTIONARY | NUMSTR | NOCOMPRESS }
```

– 其中根据已有唯一索引为表增加主键约束或唯一约束table_constraint_using_index为：

```
[ CONSTRAINT constraint_name ]
  { UNIQUE | PRIMARY KEY } USING INDEX index_name
  [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

– 其中表约束table_constraint为：

```
[ CONSTRAINT [ constraint_name ] ]
  { CHECK ( expression ) |
    UNIQUE [ idx_name ][ USING method ] ( { { column_name | ( expression ) } [ ASC |
DESC ] } [, ... ] ) index_parameters |
    PRIMARY KEY [ USING method ] ( { column_name [ ASC | DESC ] }[, ... ] ) index_parameters
    ) |
    FOREIGN KEY [ idx_name ] ( column_name [, ... ] ) REFERENCES reftable [ ( refcolumn
[, ... ] ) ]
      [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE action ] [ ON UPDATE
action ] }
  [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

其中索引参数index_parameters为：

```
[ WITH ( {storage_parameter = value} [, ... ] ) ]
  [ USING INDEX TABLESPACE tablespace_name ] [BY GLOBAL INDEX]
```

- 重命名表。对名称的修改不会影响所存储的数据。

```
ALTER TABLE [ IF EXISTS ] table_name
  RENAME TO new_table_name;
```

- 重命名表中指定的列。

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name )}
  RENAME [ COLUMN ] column_name TO new_column_name;
```

- 重命名表的约束。

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }
  RENAME CONSTRAINT constraint_name TO new_constraint_name;
```

- 设置表的所属模式。

```
ALTER TABLE [ IF EXISTS ] table_name
  SET SCHEMA new_schema;
```


说明

- 这种形式把表移动到另外一个模式。相关的索引、约束都跟着移动。目前序列不支持改变schema。 若该表拥有序列，需要将序列删除，重建，或者取消拥有关系， 才能将表schema更改成功。
- 要修改一个表的模式，用户必须在新模式上拥有CREATE权限。要把该表添加为一个父表的新子表，用户必须同时又是父表的所有者。要修改所有者，用户还必须是新的所有角色的直接或间接成员，并且该成员必须在此表的模式上有CREATE权限。这些限制规定了该用户不能做出了重建和删除表之外的事情。不过，系统管理员可以以任何方式修改任意表的所有权限。
- 除了RENAME和SET SCHEMA之外所有动作都可以捆绑在一个经过多次修改的列表中并行使用。比如，可以在一个命令里增加几个字段或修改几个字段的类型。对于大表，此种操作带来的效率提升更明显，原因在于只需要对该大表做一次处理。
- 增加一个CHECK或NOT NULL约束将会扫描该表，以保证现有的行符合约束要求。
- 用一个非空缺省值增加一个字段或者改变一个字段的现有类型会重写整个表。对于大表来说，这个操作可能会花很长时间，并且它还临时需要两倍的磁盘空间。

- 添加多个列。

```
ALTER TABLE [ IF EXISTS ] table_name
    ADD ( { column_name data_type [ compress_mode ] [ COLLATE collation ] [ column_constraint [ ... ] ]} [, ...] );
```

- 更新多个列。

```
ALTER TABLE [ IF EXISTS ] table_name
    MODIFY ( { column_name data_type | column_name [ CONSTRAINT constraint_name ] NOT NULL [ ENABLE ] | column_name [ CONSTRAINT constraint_name ] NULL } [, ...] );
```

参数说明

- **IF EXISTS**

  如果不存在相同名称的表，不会抛出一个错误，而会发出一个通知，告知表不存在。

- **table_name [*] | ONLY table_name | ONLY (table_name)**

  table_name是需要修改的表名。

  若声明了ONLY选项，则只有那个表被更改。若未声明ONLY，该表及其所有子表都将会被更改。另外，可以在表名称后面显示的增加*选项来指定包括子表，即表示所有后代表都被扫描，这是默认行为。

- **constraint_name**
  - 在DROP CONSTRAINT操作中表示要删除的现有约束的名称。
  - 在ADD CONSTRAINT操作中表示新增的约束名称。

  须知

  对于新增约束，在B模式数据库下（即sql_compatibility = 'B'）constraint_name为可选项，在其他模式数据库下，必须加上constraint_name。

- **index_name**

  索引名称。

- **idx_name**

  索引名。

须知

在ADD CONSTRAINT操作中：

- index_name仅在B模式数据库下（即sql_compatibility = 'B'）支持，其他模式数据库下不支持。
- 对于外键约束，constraint_name和index_name同时指定时，索引名为constraint_name。
- 对于唯一键约束，constraint_name和index_name同时指定时，索引名以index_name。

- **USING method**

  指定创建索引的方法。

  取值范围参考**参数说明**中的USING method。

须知

在ADD CONSTRAINT操作中：

- USING method仅在B模式数据库下（即sql_compatibility = 'B'）支持，其他模式数据库下不支持。
- 在B模式下，未指定USING method时，对于ASTORE的存储方式，默认索引方法为btree；对于USTORE的存储方式，默认索引方法为ubtree。

- **ASC | DESC**

  ASC表示指定按升序排序（默认）。DESC指定按降序排序。

须知

在ADD CONSTRAINT中，ASC|DESC只在B模式数据库下（即sql_compatibility = 'B'）支持，其他模式数据库不支持。

- **expression**

  创建一个基于该表的一个或多个字段的表达式索引约束，必须写在圆括弧中。

须知

UNIQUE约束中的表达式索引只在B模式数据库下支持（即sql_compatibility = 'B'），其他模式数据库不支持。

- **storage_parameter**

  表的存储参数的名称。

  创建索引新增一个选项：

  – parallel_workers（int类型）

    表示创建索引时起的bgworker线程数量，例如2就表示将会起2个bgworker线程并发创建索引。

    取值范围：[0,32]，0表示关闭并行建索引。

默认值：不设置该参数，表示未开启并行建索引功能。

- hasuids（bool类型）

  默认值：off

  参数开启：更新表元组时，为元组分配表级唯一标识id。

- **new_owner**

  表新拥有者的名称。

- **new_tablespace**

  表所属新的表空间名称。

- **column_name，column_1_name， column_2_name**

  现存的或新字段的名称。

- **data_type**

  新字段的类型，或者现存字段的新类型。

- **compress_mode**

  表字段的压缩可选项。该子句指定该字段优先使用的压缩算法。行存表不支持压缩。

- **charset**

  只在B模式数据库下（即sql_compatibility = 'B'）支持该语法，其他模式数据库不支持。指定表字段的字符集，单独指定时会将字段的字符序设置为指定的字符集的默认字符序。

- **collation**

  字段排序规则（字符序）名称。可选字段COLLATE指定了新字段的排序规则，如果省略，排序规则为新字段的默认类型。排序规则可以使用“select * from pg_collation;”命令从pg_collation系统表中查询，默认的排序规则为查询结果中以default开始的行。

  对于B模式数据库下（即sql_compatibility = 'B'）还支持utf8mb4_bin、utf8mb4_general_ci、utf8mb4_unicode_ci、binary字符序，部分说明见表字段的字符集说明（参见**表7-146**）。

  说明

  - 仅字符类型支持指定字符集，指定为binary字符集或字符序实际是将字符类型转化为对应的二进制类型，若类型映射不存在则报错。当前仅有TEXT类型转化为BLOB的映射。
  - 除binary字符集和字符序外，当前仅支持指定与数据库编码相同的字符集。
  - 未显式指定字段字符集或字符序时，若指定了表的默认字符集或字符序，字段字符集和字符序将从表上继承。若表的默认字符集或字符序不存在，当b_format_behavior_compat_options = 'default_collation'时，字段的字符集和字符序将继承当前数据库的字符集及其对应的默认字符序。
  - 当修改的字符集或字符序对应的字符集与当前字段字符集不同时，会将字段中的数据转换为指定的字符集进行编码。

- **USING expression**

  USING子句声明如何从旧的字段值里计算新的字段值；如果省略，缺省从旧类型向新类型的赋值转换。如果从旧数据类型到新类型没有隐含或者赋值的转换，则必须提供一个USING子句。

说明

ALTER TYPE的USING选项实际上可以声明涉及该行旧值的任何表达式，即它可以引用除了正在被转换的字段之外其他的字段。这样，就可以用ALTER TYPE语法做非常普遍性的转换。因为这个灵活性，USING表达式并没有作用于该字段的缺省值（如果有的话），结果可能不是缺省表达式要求的常量表达式。这就意味着如果从旧类型到新类型没有隐含或者赋值转换的话，即使存在USING子句，ALTER TYPE也可能无法把缺省值转换成新的类型。在这种情况下，应该用DROP DEFAULT先删除缺省，执行ALTER TYPE，然后使用SET DEFAULT增加一个合适的新缺省值。类似的考虑也适用于涉及该字段的索引和约束。

- **NOT NULL | NULL**

  设置列是否允许空值。

- **ENABLE**

  表示启动该约束，缺省时默认启用。

- **integer**

  带符号的整数常值。当使用PERCENT时表示按照表数据的百分比收集统计信息，integer的取值范围为0-100。

- **attribute_option**

  属性选项。

- **PLAIN | EXTERNAL | EXTENDED | MAIN**

  字段存储模式。

  - PLAIN必须用于定长的数值（比如integer）并且是内联的、不压缩的。
  - MAIN用于内联、可压缩的数据。
  - EXTERNAL用于外部保存、不压缩的数据。使用EXTERNAL将令在text和bytea字段上的子字符串操作更快，但付出的代价是增加了存储空间。
  - EXTENDED用于外部的压缩数据，EXTENDED是大多数支持非PLAIN存储的数据的缺省。

- **CHECK (expression)**

  每次将要插入的新行或者将要被更新的行必须使表达式结果为真才能成功，否则会抛出一个异常并且不会修改数据库。

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

  目前，CHECK表达式不能包含子查询也不能引用除当前行字段之外的变量。

- **DEFAULT default_expr**

  给字段指定缺省值。

  缺省表达式的数据类型必须和字段类型匹配。

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

- **COLUMN_ENCRYPTION_KEY = column_encryption_key**

  为ENCRYPTED WITH约束中列加密密钥的名称。

  取值范围：字符串，要符合标识符命名规范。

- **ENCRYPTION_TYPE = encryption_type_value**

  为ENCRYPTED WITH约束中的加密类型，encryption_type_value的值为[DETERMINISTIC | RANDOMIZED]。

- **GENERATED ALWAYS AS (generation_expr) [STORED]**

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

说明

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

- **AUTO_INCREMENT**

  指定列为自动增长列。

  详见：**•AUTO_INCREMENT**。

- **UNIQUE [KEY] index_parameters**

  UNIQUE (column_name [, ...]) index_parameters

  UNIQUE约束表示表里的一个或多个字段的组合必须在全表范围内唯一。

  UNIQUE KEY只能在sql_compatibility='B'时使用，与UNIQUE语义相同。

- **PRIMARY KEY index_parameters**

  PRIMARY KEY (column_name [, ...]) index_parameters

  主键约束表明表中的一个或者一些字段只能包含唯一（不重复）的非NULL值。

- **REFERENCES reftable [(refcolumn)] [MATCH matchtype] [ON DELETE action] [ON UPDATE action] (column constraint)**

  FOREIGN KEY (column_name [, ...]) REFERENCES reftable [(refcolumn [, ...])] [MATCH matchtype] [ON DELETE action] [ON UPDATE action] (table constraint)

  外键约束要求新表中一列或多列构成的组应该只包含、匹配被参考表中被参考字段值。若省略refcolumn，则将使用reftable的主键。被参考列应该是被参考表中的唯一字段或主键。外键约束不能被定义在临时表和永久表之间。

  参考字段与被参考字段之间存在三种类型匹配，分别是：

  - MATCH FULL：不允许一个多字段外键的字段为NULL，除非全部外键字段都是NULL。
  - MATCH SIMPLE（缺省）：允许任意外键字段为NULL。
  - MATCH PARTIAL：目前暂不支持。

  另外，当被参考表中的数据发生改变时，某些操作也会在新表对应字段的数据上执行。ON DELETE子句声明当被参考表中的被参考行被删除时要执行的操作。ON UPDATE子句声明当被参考表中的被参考字段数据更新时要执行的操作。对于ON DELETE子句、ON UPDATE子句的可能动作：

  - NO ACTION（缺省）：删除或更新时，创建一个表明违反外键约束的错误。若约束可推迟，且若仍存在任何引用行，那这个错误将会在检查约束的时候产生。
  - RESTRICT：删除或更新时，创建一个表明违反外键约束的错误。与NO ACTION相同，只是动作不可推迟。
  - CASCADE：删除新表中任何引用了被删除行的行，或更新新表中引用行的字段值为被参考字段的新值。
  - SET NULL：设置引用字段为NULL。
  - SET DEFAULT：设置引用字段为它们的缺省值。
- **DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE**

  设置该约束是否可推迟。
  - DEFERRABLE：可以推迟到事务结尾使用SET CONSTRAINTS命令检查。
  - NOT DEFERRABLE：在每条命令之后马上检查。
  - INITIALLY IMMEDIATE：那么每条语句之后就立即检查它。
  - INITIALLY DEFERRED：只有在事务结尾才检查它。
- **WITH ({storage_parameter = value} [, ...])**

  为表或索引指定一个可选的存储参数。
- **tablespace_name**

  索引所在表空间的名称。
- **COMPRESS|NOCOMPRESS**
  - NOCOMPRESS：如果指定关键字NOCOMPRESS则不会修改表的现有压缩特性。
  - COMPRESS：如果指定COMPRESS关键字，则对该表进行批量插入元组时触发该特性。行存表不支持压缩。
- **new_table_name**

  修改后新的表名称。
- **new_column_name**

  表中指定列修改后新的列名称。
- **new_constraint_name**

  修改后表约束的新名称。
- **new_schema**

  修改后新的模式名称。
- **CASCADE**

  级联删除依赖于被依赖字段或者约束的对象（比如引用该字段的视图）。
- **RESTRICT**

  如果该列还被其他字段或者约束引用，则拒绝删除该列。RESTRICT为CASCADE的缺省选项，如果未指定则为RESTRICT。语句示例如下：

  ```
  alter table <表名>[drop [column] <列名> [cascade | restrict]];
  ```
- **FIRST**

  新增列或修改列到第一位。
- **AFTER column_name**

  新增列或修改列到column_name之后。

说明

- 仅在B模式数据库下（即sql_compatibility = 'B'）支持，其他模式数据库不支持。
- 加密列不支持FIRST | AFTER column_name。
- 有规则依赖的表不支持改变表列的位置（包括新增和修改导致列位置的变化）。
- 外表不支持FIRST | AFTER column_name。
- SET类型的字段不支持修改到指定位置。

- **schema_name**

  表所在的模式名称。

- **[DEFAULT] CHARACTER SET | CHARSET [=] default_charset**

  仅在sql_compatibility='B'时支持该语法。修改表的默认字符集，单独指定时会将表的默认字符序设置为指定的字符集的默认字符序。

- **[DEFAULT] COLLATE [=] default_collation**

  仅在sql_compatibility='B'时支持该语法。修改表的默认字符序，单独指定时会将表的默认字符集设置为指定的字符序对应的字符集。字符序参见**表7-146**。

  说明

  未显式指定表的字符集或字符序时，若指定了模式的默认字符集或字符序，表字符集和字符序将从模式上继承。若模式的默认字符集或字符序不存在，当b_format_behavior_compat_options = "default_collation"时，表的字符集和字符序将继承当前数据库的字符集及其对应的默认字符序。

修改表示例

- **重命名表。**

```
gaussdb=# CREATE TABLE aa(c1 int, c2 int);
gaussdb=# ALTER TABLE IF EXISTS aa RENAME TO test_alt1;
```

- **修改表所属模式。**

```
--创建模式test_schema。
gaussdb=# CREATE SCHEMA test_schema;

--把表test_alt1的所属模式修改为test_schema。
gaussdb=# ALTER TABLE test_alt1 SET SCHEMA test_schema;

--查询表信息。
gaussdb=# SELECT schemaname,tablename FROM pg_tables WHERE tablename = 'test_alt1';
 schemaname  | tablename
-------------+-----------
 test_schema | test_alt1
(1 row)
```

- **修改表的所有者。**

```
--创建用户test_user。
gaussdb=# CREATE USER test_user PASSWORD 'XXXXXXXXXX';

--修改test_alt1表的所有者为test_user。
gaussdb=# ALTER TABLE IF EXISTS test_schema.test_alt1 OWNER TO test_user;

--查看。
gaussdb=# SELECT tablename, schemaname, tableowner FROM pg_tables WHERE tablename =
'test_alt1';
 tablename | schemaname  | tableowner
-----------+-------------+------------
 test_alt1 | test_schema | test_user
(1 row)
```

- **修改表的表空间。**

```
--创建表空间tbs_data1。
gaussdb=# CREATE TABLESPACE tbs_data1 RELATIVE LOCATION 'tablespace1/tbs_data1';
```



```
--修改test_alt1表的空间为tbs_data1。
gaussdb=# ALTER TABLE test_schema.test_alt1 SET TABLESPACE tbs_data1;

--查看。
gaussdb=# SELECT tablename, tablespace FROM pg_tables WHERE tablename = 'test_alt1';
 tablename | tablespace
-----------+------------
 test_alt1 | tbs_data1
(1 row)

--删除。
gaussdb=# DROP TABLE test_schema.test_alt1;
gaussdb=# DROP TABLESPACE tbs_data1;
gaussdb=# DROP SCHEMA test_schema;
gaussdb=# DROP USER test_user;
```

修改列示例

- **修改列名。**

```
--建表。
gaussdb=# CREATE TABLE test_alt2(c1 INT,c2 INT);

--修改列名。
gaussdb=# ALTER TABLE test_alt2 RENAME c1 TO id;
gaussdb=# ALTER TABLE test_alt2 RENAME COLUMN c2 to areaid;

--查看。
gaussdb=# \d test_alt1
   Table "public.test_alt1"
 Column |  Type   | Modifiers
--------+---------+-----------
 id     | integer |
 areaid | integer |
```

- **增加列。**

```
--表test_alt1增加列。
gaussdb=# ALTER TABLE IF EXISTS test_alt2 ADD COLUMN name VARCHAR(20);

--查看。
gaussdb=# \d test_alt2
           Table "public.test_alt1"
  Column  |         Type          | Modifiers
----------+-----------------------+-----------
 id       | integer               |
 areacode | integer               |
 name     | character varying(20) |
```

- **修改列的数据类型。**

```
--修改test_alt1表中name字段的类型。
gaussdb=# ALTER TABLE test_alt1 MODIFY name VARCHAR(50);

--查看。
gaussdb=# \d test_alt1
          Table "public.test_alt2"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 id     | integer               |
 areaid | integer               |
 name   | character varying(50) |
--修改test_alt1表中name字段的类型。
gaussdb=# ALTER TABLE test_alt2 ALTER COLUMN name TYPE VARCHAR(25);

--查看。
gaussdb=# \d test_alt2
          Table "public.test_alt2"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 id     | integer               |
```



```
 areaid | integer               |
 name   | character varying(25) |
```

- **删除列**。

```
--删除test_alt1中areaid字段。
gaussdb=# ALTER TABLE test_alt2 DROP COLUMN areaid;

--查看。
gaussdb=# \d test_alt2
          Table "public.test_alt2"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 id     | integer               |
 name   | character varying(25) |
```

- **修改字段存储模式**。

```
--查看表详细信息。
gaussdb=# \d+ test_alt2
                            Table "public.test_alt2"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+-------------
 id     | integer               |           | plain    |              |
 name   | character varying(25) |           | extended |              |
Has OIDs: no
Options: orientation=row, compression=no, storage_type=USTORE

--修改test_alt2表中name字段的存储模式。
gaussdb=# ALTER TABLE test_alt2 ALTER COLUMN name SET STORAGE PLAIN;

--查看。
gaussdb=# \d+ test_alt2
                           Table "public.test_alt2"
 Column |         Type          | Modifiers | Storage | Stats target | Description
--------+-----------------------+-----------+---------+--------------+-------------
 id     | integer               |           | plain   |              |
 name   | character varying(25) |           | plain   |              |
Has OIDs: no
Options: orientation=row, compression=no, storage_type=USTORE

--删除。
gaussdb=# DROP TABLE test_alt2;
```

- **修改字段到指定位置**。

```
--创建B模式数据库。
gaussdb=# CREATE DATABASE test DBCOMPATIBILITY 'B';

--连接至test数据库并创建表tbl_test。
gaussdb=# \c test
gaussdb=# CREATE TABLE tbl_test(id int, name varchar(20));

--修改tbl_test表中字段name类型，并指定位置到最前面。
gaussdb=# ALTER TABLE tbl_test MODIFY COLUMN name varchar(25) FIRST;

--查看。
gaussdb=# \d tbl_test;
          Table "public.tbl_test"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 name   | character varying(25) |
 id     | integer               |

--修改tbl_test字段name的类型，并指定位置在id字段的后面。
gaussdb=# ALTER TABLE tbl_test MODIFY COLUMN name varchar(10) AFTER id;

--查看。
gaussdb=# \d tbl_test;
          Table "public.tbl_test"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 id     | integer               |
```



```
 name   | character varying(10) |

--删除表tbl_test。
gaussdb=# DROP TABLE tbl_test;

--切换至默认数据库删除数据库test(根据实际情况切换至相应的数据库)。
gaussdb=# \c postgres
gaussdb=# DROP DATABASE test;
```

修改约束示例

- **为列添加非空约束。**

```
--建表。
gaussdb=# CREATE TABLE test_alt3(pid INT, areaid CHAR(5), name VARCHAR(20));

--为pid添加非空约束。
gaussdb=# ALTER TABLE test_alt3 MODIFY pid NOT NULL;

--查看。
gaussdb=# \d test_alt3
          Table "public.test_alt3"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 pid    | integer               | not null
 areaid | character(5)          |
 name   | character varying(20) |
```

- **取消列的非空约束。**

```
gaussdb=# ALTER TABLE test_alt3 MODIFY pid NULL;
--查看。
gaussdb=# \d test_alt3
          Table "public.test_alt3"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 pid    | integer               |
 areaid | character(5)          |
 name   | character varying(20) |
```

- **修改字段默认值。**

```
--修改test_alt1表中id的默认值。
gaussdb=# ALTER TABLE test_alt3 ALTER COLUMN areaid SET DEFAULT '00000';

--查看。
gaussdb=# \d test_alt3
                Table "public.test_alt3"
 Column |         Type          |        Modifiers
--------+-----------------------+--------------------------
 pid    | integer               |
 areaid | character(5)          | default '00000'::bpchar
 name   | character varying(20) |
--删除id的默认值。
gaussdb=# ALTER TABLE test_alt3 ALTER COLUMN areaid DROP DEFAULT;

--查看。
gaussdb=# \d test_alt3
          Table "public.test_alt3"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 pid    | integer               |
 areaid | character(5)          |
 name   | character varying(20) |
```

- **添加表级约束。**
  - 直接添加约束。

```
--给表添加主键约束。
gaussdb=# ALTER TABLE test_alt3 ADD CONSTRAINT pk_test3_pid PRIMARY KEY (pid);

--查看。
gaussdb=# \d test_alt3
```



```
          Table "public.test_alt3"
 Column |          Type          | Modifiers
--------+------------------------+-----------
 pid    | integer                | not null
 areaid | integer                |
 name   | character varying(20) |
Indexes:
    "pk_test3_pid" PRIMARY KEY, btree (pid) TABLESPACE pg_default
```

- 先创建索引然后再添加约束。

```
--建表。
gaussdb=# CREATE TABLE test_alt4(c1 INT, c2 INT);

--建索引。
gaussdb=# CREATE UNIQUE INDEX pk_test4_c1 ON test_alt4(c1);

--添加约束时关联已经创建的索引。
gaussdb=# ALTER TABLE test_alt4 ADD CONSTRAINT pk_test4_c1 PRIMARY KEY USING INDEX
pk_test4_c1;

--查看。
gaussdb=# \d test_alt4
   Table "public.test_alt4"
 Column |  Type   | Modifiers
--------+---------+-----------
 c1     | integer | not null
 c2     | integer |
Indexes:
    "pk_test4_c1" PRIMARY KEY, btree (c1) TABLESPACE pg_default

--删除。
gaussdb=# DROP TABLE test_alt4;
```

- **删除表级约束**。

```
--删除约束。
gaussdb=# ALTER TABLE test_alt3 DROP CONSTRAINT IF EXISTS pk_test3_pid;

--查看。
gaussdb=# \d test_alt3
          Table "public.test_alt3"
 Column |          Type          | Modifiers
--------+------------------------+-----------
 pid    | integer                | not null
 areaid | integer                |
 name   | character varying(20) |

--删除。
gaussdb=# DROP TABLE test_alt3;
```

7.14.33 ALTER TABLE PARTITION

功能描述

修改表分区，包括增加/删除分区、切割/合并分区、清空分区、移动分区表空间、交换分区、重命名分区，以及修改分区属性等。

注意事项

- 只有分区表的所有者或者被授予了分区表ALTER权限的用户有权限执行ALTER TABLE PARTITION命令，系统管理员默认拥有此权限。
- 添加分区的表空间不能是PG_GLOBAL。
- 添加分区的名称不能与该分区表已有分区的名称相同。
- 添加分区的分区键值要和分区表的分区键的类型一致。

- 若添加RANGE分区，添加分区键值要大于分区表中最后一个范围分区的上边界。
- 若添加LIST分区，添加分区键值不能与现有分区键值重复。
- 不支持添加HASH分区。
- 如果目标分区表中已有分区数达到了最大值1048575，则不能继续添加分区。
- 当分区表只有一个分区时，不能删除该分区。
- 选择分区使用PARTITION FOR()，括号里指定值个数应该与定义分区时使用的列个数相同，并且一一对应。
- Value分区表不支持相应的Alter Partition操作。
- 间隔分区表不支持添加分区。
- 哈希分区表不支持切割分区，不支持合成分区，不支持添加和删除分区。
- 删除、切割、合并、清空、交换分区的操作会使Global索引失效，可以申明UPDATE GLOBAL INDEX子句同步更新索引。
- 如果删除、切割、合并、清空、交换分区操作不申明UPDATE GLOBAL INDEX子句，并发的DML业务有可能因为索引不可用而报错。
- 若设置参数enable_gpi_auto_update为on，即使不申明UPDATE GLOBAL INDEX子句，也会自动更新Global索引。

语法格式

修改分区表分区包括修改表分区主语法、修改表分区名称的语法和重置分区ID的语法。

- 修改表分区主语法。

```
ALTER TABLE [ IF EXISTS ] { table_name  [*] | ONLY table_name | ONLY ( table_name  )}
  action [, ... ];
```

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

```
  move_clause |
  exchange_clause |
  row_clause |
  merge_clause |
  modify_clause |
  split_clause |
  add_clause |
  drop_clause |
  truncate_clause
```

  - move_clause子语法用于移动分区到新的表空间。

```
MOVE PARTITION { partion_name | FOR ( partition_value [, ...] ) } TABLESPACE tablespacename
```

  - exchange_clause子语法用于把普通表的数据迁移到指定的分区。

```
EXCHANGE PARTITION { ( partition_name ) | FOR ( partition_value [, ...] ) }
   WITH TABLE {[ ONLY ] ordinary_table_name | ordinary_table_name * | ONLY
( ordinary_table_name )}
   [ { WITH | WITHOUT } VALIDATION ] [ VERBOSE ] [ UPDATE GLOBAL INDEX ]
```

    进行交换的普通表和分区必须满足如下条件：

    - 普通表和分区的列数目相同，对应列的信息严格一致，包括：列名、列的数据类型、列约束、列的Collation信息、列的存储参数、列的压缩信息等。
    - 普通表和分区的表压缩信息严格一致。

- 普通表索引和分区Local索引个数相同，且对应索引的信息严格一致。
- 普通表和分区的表约束个数相同，且对应表约束的信息严格一致。
- 普通表不可以是临时表，分区表只能是范围分区表，列表分区表，哈希分区表或间隔分区表。
- 普通表和分区表上不可以有动态数据脱敏，行访问控制约束。

须知

- 完成交换后，普通表和分区的数据被置换，同时普通表和分区的表空间信息被置换。此时，普通表和分区的统计信息变得不可靠，需要对普通表和分区重新执行analyze。
- 由于非分区键不能建立本地唯一索引，只能建立全局唯一索引，所以如果普通表含有唯一索引时，可能会导致不能交换数据。

  如果需要进行数据交换操作，可以通过创建中间表的方式，先将分区数据插入到中间表，truncate分区，普通表数据插入分区表，drop普通表，重命名中间表的方式完成数据交换操作。
- 如果在普通表/分区表上进行了drop column操作，被删除的列依然物理存在，所以需要保证普通表和分区的被删除列也严格对齐才能交换成功。

– row_clause子语法用于设置分区表的行迁移开关。

```
{ ENABLE | DISABLE } ROW MOVEMENT
```

– merge_clause子语法用于把多个分区合并成一个分区。一个命令中合并的源分区上限为300。

```
MERGE PARTITIONS { partition_name } [, ...] INTO PARTITION partition_name
    [ TABLESPACE tablespacename ] [ UPDATE GLOBAL INDEX ]
```

须知

对于范围分区/间隔分区，MERGE分区要求源分区的范围连续递增，且MERGE后的分区名可以与最后一个源分区名相同；对于列表分区，则源分区无顺序要求，且MERGE后的分区名可以与任一源分区名相同。如果MERGE后的分区名与源分区名相同，视为同一个分区。

须知

USTORE存储引擎表不支持在事务块/存储过程中执行ALTER TABLE MERGE PARTITIONS的操作。

– modify_clause子语法用于设置分区索引是否可用。

```
MODIFY PARTITION partition_name { UNUSABLE LOCAL INDEXES | REBUILD UNUSABLE LOCAL INDEXES }
```

– split_clause子语法用于把一个分区切割成多个分区。

```
SPLIT PARTITION { partition_name | FOR ( partition_value [, ...] ) } { split_point_clause | no_split_point_clause } [ UPDATE GLOBAL INDEX ]
```

须知

- SPLIT后的分区名可以与源分区名相同，但视为不同的分区。

- 范围分区表和间隔分区表指定切割点split_point_clause的语法为：

```
AT ( partition_value ) INTO ( PARTITION partition_name [ TABLESPACE
tablespacename ] , PARTITION partition_name [ TABLESPACE tablespacename ] )
```

  须知

  切割点的大小要位于正在被切割的分区的分区键范围内，指定切割点的方式只能把一个分区切割成两个新分区。

- 范围分区表和间隔分区表不指定切割点no_split_point_clause的语法为：

```
INTO { ( partition_less_than_item [, ...] ) | ( partition_start_end_item [, ...] ) }
```

  须知

  - 不指定切割点的方式，partition_less_than_item指定的第一个新分区的分区键要大于正在被切割的分区的前一个分区（如果存在的话）的分区键，partition_less_than_item指定的最后一个分区的分区键要等于正在被切割的分区的分区键大小。
  - 不指定切割点的方式，partition_start_end_item指定的第一个新分区的起始点（如果存在的话）必须等于正在被切割的分区的前一个分区（如果存在的话）的分区键，partition_start_end_item指定的最后一个分区的终止点（如果存在的话）必须等于正在被切割的分区的分区键。
  - partition_less_than_item支持的分区键个数最多为16，而partition_start_end_item仅支持1个分区键，其支持的数据类型参见**•PARTITION BY RANGE [COLUMNS] (partition_key)**。
  - 在同一语句中partition_less_than_item和partition_start_end_item两者不可同时使用；不同split语句之间没有限制。

- 分区项partition_less_than_item的语法如下，其中最后一个分区可以不写分区范围定义，即VALUES LESS THAN (partition_value)部分，默认继承源分区范围定义的上界值。

```
PARTITION partition_name VALUES LESS THAN ( { partition_value | MAXVALUE }  [, ...] )
   [ TABLESPACE tablespacename ]
```

- 分区项partition_start_end_item的语法如下，其约束参见**START END语法描述**。

```
PARTITION partition_name {
      {START(partition_value) END (partition_value) EVERY (interval_value)} |
      {START(partition_value) END ({partition_value | MAXVALUE})} |
      {START(partition_value)} |
      {END({partition_value | MAXVALUE})}
} [TABLESPACE tablespace_name]
```

- 列表分区表指定切割点split_point_clause的语法如下：

```
VALUES ( partition_value_list ) INTO ( PARTITION partition_name [ TABLESPACE
tablespacename ] , PARTITION partition_name [ TABLESPACE tablespacename ] )
```

须知

切割点必须是源分区的一个非空真子集，指定切割点的方式只能把一个分区切割成两个新分区。

- 列表分区表不指定切割点no_split_point_clause的语法如下，其中最后一个分区不能写分区范围定义，即VALUES (partition_value_list)部分，其范围等于源分区去掉其他子分区后的剩余集合。

```
INTO ( PARTITION partition_name VALUES (partition_value_list) [ TABLESPACE tablespacename ][, ...] )
```

须知

- 最后一个新分区不能写分区范围定义，其范围等于源分区去掉其他子分区后的剩余集合。
- 不指定切割点的方式，每一个新分区都必须是源分区的一个非空真子集，且互不交叉。

– add_clause子语法用于为指定的分区表添加一个或多个分区。

```
ADD {partition_less_than_item | partition_start_end_item| partition_list_item }
```

分区项partition_list_item的语法如下。

```
PARTITION partition_name VALUES (list_values_clause)
    [ TABLESPACE tablespacename ]
```

须知

- partition_list_item支持最多16个分区键，其支持的数据类型参见**•PARTITION BY LIST [COLUMNS] (partition_key)**。
- 间隔/哈希分区表不支持添加分区。

– drop_clause子语法用于删除分区表中的指定分区。

```
DROP PARTITION  { partition_name | FOR (  partition_value [, ...] )  } [ UPDATE GLOBAL INDEX ]
```

须知

- 哈希分区表不支持删除分区。
- 当分区表只有一个分区时，不能删除该分区。

– truncate_clause子语法用于清空分区表中的指定分区。

```
TRUNCATE PARTITION  { partition_name | FOR (  partition_value [, ...] )  } [ UPDATE GLOBAL INDEX ]
```

- 修改表分区名称的语法。

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name  )}
    RENAME PARTITION { partion_name | FOR ( partition_value [, ...] ) } TO partition_new_name;
```

- 重置分区ID的语法。

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name  )} RESET PARTITION;
```

参数说明

- **table_name**

  分区表名。

  取值范围：已存在的分区表名。

- **partition_name**

  分区名。

  取值范围：已存在的分区名。

- **tablespacename**

  指定分区要移动到哪个表空间。

  取值范围：已存在的表空间名。

- **partition_value**

  分区键值。

  通过PARTITION FOR（ partition_value [, ...])子句指定的这一组值，可以唯一确定一个分区。

  取值范围：需要进行操作的分区的分区键的取值范围。

- **UNUSABLE LOCAL INDEXES**

  设置该分区上的所有索引不可用。

- **REBUILD UNUSABLE LOCAL INDEXES**

  重建该分区上的所有索引。

- **{ ENABLE | DISABLE } ROW MOVEMET**

  行迁移开关。

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

  取值范围：

  - ENABLE：打开行迁移开关。
  - DISABLE：关闭行迁移开关。

  默认是打开状态。

- **ordinary_table_name**

  进行迁移的普通表的名称。

  取值范围：已存在的普通表名。

- **{ WITH | WITHOUT } VALIDATION**

  在进行数据迁移时，是否检查普通表中的数据满足指定分区的分区键范围。

  取值范围：

  - WITH：对于普通表中的数据要检查是否满足分区的分区键范围，如果有数据不满足，则报错。
  - WITHOUT：对于普通表中的数据不检查是否满足分区的分区键范围。

  默认是WITH状态。

  由于检查比较耗时，特别是当数据量很大的情况下更甚。所以在保证当前普通表中的数据满足分区的分区键范围时，可以加上WITHOUT来指明不进行检查。

- **VERBOSE**

在VALIDATION是WITH状态时，如果检查出普通表有不满足要交换分区的分区键范围的数据，那么把这些数据插入到正确的分区，如果路由不到任何分区，再报错。

须知

只有在VALIDATION是WITH状态时，才可以指定VERBOSE。

- **partition_new_name**

  分区的新名称。

  取值范围：字符串，要符合**标识符命名规范**。

- **UPDATE GLOBAL INDEX**

  如果使用该参数，则会更新分区表上的所有全局索引，以确保使用全局索引可以查询出正确的数据；如果不使用该参数，则分区表上的所有全局索引将会失效。

示例

请参考CREATE TABLE PARTITION的**示例**。

相关链接

CREATE TABLE PARTITION，**DROP TABLE**

7.14.34 ALTER TABLE SUBPARTITION

功能描述

修改二级分区表分区，包括增删分区、清空分区、切割/合并分区、移动分区表空间、交换分区、重命名分区，以及修改分区属性等。

注意事项

- 添加分区的表空间不能是PG_GLOBAL。
- 添加分区的名称不能与该分区表已有一级分区和二级分区的名称相同。
- 添加分区的分区键值要和分区表的分区键的类型一致。
- 若添加RANGE分区，添加分区键值要大于分区表中最后一个范围分区的上边界。若需要在有MAXVALUE分区的表上新增分区，建议使用SPLIT语法。
- 若添加LIST分区，添加分区键值不能与现有分区键值重复。若需要在有DEFAULT分区的表上新增分区，建议使用SPLIT语法。
- 不支持添加HASH分区。只有一种情况例外，二级分区表的二级分区方式为HASH且一级分区方式不是HASH，此时支持新增一级分区并创建对应的二级分区。
- 如果目标分区表中已有分区数达到了最大值1048575，则不能继续添加分区。
- 当分区表只有一个一级分区或二级分区时，不能删除该分区。
- 不支持删除HASH分区。
- 选择分区使用PARTITION FOR()或SUBPARTITION FOR()，括号里指定值个数应该与定义分区时使用的列个数相同，并且一一对应。

- 切割分区只能对二级分区（叶子节点）进行切割，被切割分区只能是Range、List分区策略，不支持切割hash分区策略。
- 合并分区只能对二级分区（叶子节点）进行合并，且源分区必须属于同一个一级分区。
- 只有分区表的所有者或者被授予了分区表ALTER权限的用户有权限执行ALTER TABLE PARTITION命令，系统管理员默认拥有此权限。
- 删除、切割、清空、交换分区的操作会使Global索引失效，可以申明UPDATE GLOBAL INDEX子句同步更新索引。
- 如果删除、切割、清空、交换分区操作不申明UPDATE GLOBAL INDEX子句，并发的DML业务有可能因为索引不可用而报错。
- 若设置参数enable_gpi_auto_update为on，即使不申明UPDATE GLOBAL INDEX子句，也会自动更新Global索引。

语法格式

修改二级分区表分区包括修改表分区主语法、修改表分区名称的语法和重置分区ID的语法。

- 修改表分区主语法。

```
ALTER TABLE [ IF EXISTS ] { table_name   [*] | ONLY table_name | ONLY ( table_name  )}
  action [, ... ];
```

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

```
  move_clause  |
  exchange_clause  |
  row_clause  |
  merge_clause  |
  modify_clause  |
  add_clause    |
  drop_clause   |
  split_clause  |
  truncate_clause
```

  - move_clause子语法用于移动分区到新的表空间。

```
MOVE SUBPARTITION { subpartion_name | FOR ( subpartition_value [, ...] ) } TABLESPACE
tablespacename
```

  - exchange_clause子语法用于把普通表的数据迁移到指定的分区。

```
EXCHANGE SUBPARTITION { ( subpartition_name ) | FOR ( subpartition_value [, ...] ) }
   WITH TABLE {[ ONLY ] ordinary_table_name | ordinary_table_name * | ONLY
( ordinary_table_name )}
   [ { WITH | WITHOUT } VALIDATION ] [ VERBOSE ] [ UPDATE GLOBAL INDEX ]
```

    进行交换的普通表和分区必须满足如下条件：

    - 普通表和分区的列数目相同，对应列的信息严格一致，包括：列名、列的数据类型、列约束、列的Collation信息、列的存储参数、列的压缩信息等。
    - 普通表和分区的表压缩信息严格一致。
    - 普通表索引和分区Local索引个数相同，且对应索引的信息严格一致。
    - 普通表和分区的表约束个数相同，且对应表约束的信息严格一致。
    - 普通表不可以是临时表，分区表只能是二级分区表。

- 普通表和分区表上不可以有动态数据脱敏，行访问控制约束。

须知

- 完成交换后，普通表和分区的数据被置换，同时普通表和分区的表空间信息被置换。此时，普通表和分区的统计信息变得不可靠，需要对普通表和分区重新执行analyze。
- 由于非分区键不能建立本地唯一索引，只能建立全局唯一索引，所以如果普通表含有唯一索引时，可能会导致不能交换数据。

  如果需要进行数交换数据操作可以通过创建中间表的方式，先将分区数据插入到中间表，truncate分区，普通表数据插入分区表，drop普通表，重命名中间表的方式完成数据交换操作。
- 如果在普通表/分区表上进行了drop column操作，被删除的列依然物理存在，所以需要保证普通表和分区的被删除列也严格对齐才能交换成功。

– row_clause子语法用于设置分区表的行迁移开关。

```
{ ENABLE | DISABLE } ROW MOVEMENT
```

– merge_clause子语法用于把多个分区合并成一个分区。一个命令中合并的源分区上限为300。

```
MERGE SUBPARTITIONS { subpartition_name } [, ...] INTO SUBPARTITION partition_name
   [ TABLESPACE tablespacename ] [ UPDATE GLOBAL INDEX ]
```

须知

对于范围分区，MERGE分区要求源分区的范围连续递增，且MERGE后的分区名可以与最后一个源分区名相同；对于列表分区，则源分区无顺序要求，且MERGE后的分区名可以与任一源分区名相同。如果MERGE后的分区名与源分区名相同，视为同一个分区。

须知

USTORE存储引擎表不支持在事务块/存储过程中执行ALTER TABLE MERGE SUBPARTITIONS的操作。

– modify_clause子语法用于设置分区索引是否可用。语法可以作用在一级分区上。

```
MODIFY PARTITION partition_name { UNUSABLE LOCAL INDEXES | REBUILD UNUSABLE LOCAL INDEXES }
```

也可以作用在二级分区上。

```
MODIFY SUBPARTITION partition_name { UNUSABLE LOCAL INDEXES | REBUILD UNUSABLE LOCAL INDEXES }
```

– add_clause子语法用于为指定的分区表添加一个或多个分区。语法可以作用在一级分区上。

```
ADD {partition_less_than_item | partition_list_item } [ ( subpartition_definition_list ) ]
```

也可以作用在二级分区上。

```
MODIFY PARTITION partition_name ADD subpartition_definition
```

其中，分区项partition_less_than_item为RANGE分区定义语法，具体语法如下。


```
PARTITION partition_name VALUES LESS THAN ( partition_value | MAXVALUE ) [ TABLESPACE tablespacename ]
```

分区项partition_list_item为LIST分区定义语法，具体语法如下。

```
PARTITION partition_name VALUES ( partition_value [, ...] | DEFAULT ) [ TABLESPACE tablespacename ]
```

subpartition_definition_list为1到多个二级分区subpartition_definition对象，subpartition_definition具体语法如下。

```
SUBPARTITION subpartition_name [ VALUES LESS THAN ( partition_value | MAXVALUE ) | VALUES ( partition_value [, ...] | DEFAULT )]  [ TABLESPACE tablespace ]
```

须知

若一级分区为HASH分区，不支持以ADD形式新增一级分区；若二级分区为HASH分区，不支持以MODIFY形式新增二级分区。

- drop_clause子语法用于删除分区表中的指定分区。语法可以作用在一级分区上。

```
DROP PARTITION  { partition_name | FOR (  partition_value )  } [ UPDATE GLOBAL INDEX ]
```

也可以作用在二级分区上。

```
DROP SUBPARTITION  { subpartition_name | FOR (  partition_value, subpartition_value )  } [ UPDATE GLOBAL INDEX ]
```

须知

- 若一级分区为HASH分区，不支持删除一级分区；若二级分区为HASH分区，不支持删除二级分区。
- 不支持删除唯一子分区。

- split_clause子语法用于把一个分区切割成多个分区。

```
SPLIT SUBPARTITION { subpartition_name| FOR ( subpartition_value [, ...] ) } { split_point_clause | no_split_point_clause } [ UPDATE GLOBAL INDEX ]
```

须知

- SPLIT后的分区名可以与源分区名相同，但视为不同的分区。

  - 范围分区指定切割点split_point_clause的语法为：

    ```
    AT ( subpartition_value ) INTO ( SUBPARTITION subpartition_name [ TABLESPACE tablespacename ] , SUBPARTITION subpartition_name [ TABLESPACE tablespacename ] )
    ```

    须知

    切割点的大小要位于正在被切割的分区的分区键范围内，指定切割点的方式只能把一个分区切割成两个新分区。

  - 范围分区不指定切割点no_split_point_clause 的语法如下，其中最后一个分区不能写分区范围定义，即VALUES LESS THAN (subpartition_value)部分，默认继承源分区范围定义的上界值。

    ```
    INTO ( SUBPARTITION subpartition_name VALUES LESS THAN (subpartition_value) [ TABLESPACE tablespacename ][, ...] )
    ```

须知

- 第一个新分区的分区范围定义要大于正在被切割的分区的前一个分区（如果存在的话）的分区范围定义。
- 最后一个新分区不能写分区范围定义，默认继承源分区范围定义的上界值。
- 新分区必须满足分区范围定义递增的约束。

▪ 列表范围分区指定切割点split_point_clause的语法如下：

```
VALUES ( subpartition_value ) INTO ( SUBPARTITION subpartition_name [ TABLESPACE tablespacename ] , SUBPARTITION subpartition_name [ TABLESPACE tablespacename ] )
```

须知

切割点必须是源分区的一个非空真子集，指定切割点的方式只能把一个分区切割成两个新分区。

▪ 列表分区表不指定切割点no_split_point_clause的语法如下，其中最后一个分区不能写分区范围定义，即VALUES (subpartition_value_list)部分，其范围等于源分区去掉其他子分区后的剩余集合。

```
INTO ( SUBPARTITION subpartition_name VALUES (subpartition_value_list) [ TABLESPACE tablespacename ][, ...] )
```

须知

- 最后一个新分区不能写分区范围定义，其范围等于源分区去掉其他子分区后的剩余集合。
- 不指定切割点的方式，每一个新分区都必须是源分区的一个非空真子集，且互不交叉。

– truncate_clause子语法用于清空分区表中的指定分区。语法可以作用在一级分区上。

```
TRUNCATE PARTITION { partition_name | FOR ( partition_value [, ...] ) } [ UPDATE GLOBAL INDEX ]
```

也可以作用在二级分区上。

```
TRUNCATE SUBPARTITION { subpartition_name | FOR ( subpartition_value [, ...] ) } [ UPDATE GLOBAL INDEX ]
```

- 修改表分区名称的语法。可以修改分区表的一级分区。

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name )}
    RENAME PARTITION { partion_name | FOR ( partition_value [, ...] ) } TO partition_new_name;
```

也可以修改分区表的二级分区。

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name )}
    RENAME SUBPARTITION { subpartion_name | FOR ( subpartition_value [, ...] ) } TO subpartition_new_name;
```

- 重置分区ID的语法。

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name )} RESET PARTITION;
```

参数说明

- **table_name**

  分区表名。

  取值范围：已存在的分区表名。

- **subpartition_name**

  二级分区名。

  取值范围：已存在的二级分区名。

- **tablespacename**

  指定分区要移动到哪个表空间。

  取值范围：已存在的表空间名。

- **partition_value**

  一级分区键值。

  通过PARTITION FOR（partition_value [, ...])子句指定的这一组值，可以唯一确定一个一级分区。

  取值范围：需要进行操作的一级分区的分区键的取值范围。

- **subpartition_value**

  一级分区键值和二级分区键值。

  通过SUBPARTITION FOR（subpartition_value [, ...])子句指定的这一组值，可以唯一确定一个二级分区。

  取值范围：对于需要进行操作的二级分区，需要同时有其一级分区分区键和二级分区分区键的取值范围。

- **UNUSABLE LOCAL INDEXES**

  设置该分区上的所有索引不可用。

- **REBUILD UNUSABLE LOCAL INDEXES**

  重建该分区上的所有索引。

- **{ ENABLE | DISABLE } ROW MOVEMET**

  行迁移开关。

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

  取值范围：

  - ENABLE：打开行迁移开关。
  - DISABLE：关闭行迁移开关。

  默认是打开状态。

- **ordinary_table_name**

  进行迁移的普通表的名称。

  取值范围：已存在的普通表名。

- **{ WITH | WITHOUT } VALIDATION**

  在进行数据迁移时，是否检查普通表中的数据满足指定分区的分区键范围。

  取值范围：

  - WITH：对于普通表中的数据要检查是否满足分区的分区键范围，如果有数据不满足，则报错。

– WITHOUT：对于普通表中的数据不检查是否满足分区的分区键范围。

默认是WITH状态。

由于检查比较耗时，特别是当数据量很大的情况下更甚。所以在保证当前普通表中的数据满足分区的分区键范围时，可以加上WITHOUT来指明不进行检查。

- **VERBOSE**

  在VALIDATION是WITH状态时，如果检查出普通表有不满足要交换分区的分区键范围的数据，那么把这些数据插入到正确的分区，如果路由不到任何分区，再报错。

须知

只有在VALIDATION是WITH状态时，才可以指定VERBOSE。

- **partition_new_name**

  分区的新名称。

  取值范围：字符串，要符合**标识符命名规范**。

- **subpartition_new_name**

  二级分区的新名称。

  取值范围：字符串，要符合**标识符命名规范**。

- **UPDATE GLOBAL INDEX**

  如果使用该参数，则会更新分区表上的所有全局索引，以确保使用全局索引可以查询出正确的数据；如果不使用该参数，则分区表上的所有全局索引将会失效。

示例

请参考CREATE TABLE SUBPARTITION的**示例**。

相关链接

CREATE TABLE SUBPARTITION，**DROP TABLE**

7.14.35 ALTER TABLESPACE

功能描述

修改表空间的属性。

注意事项

- 只有表空间的所有者或者被授予了表空间ALTER权限的用户有权限执行ALTER TABLESPACE命令，系统管理员默认拥有此权限。但要修改表空间的所有者，当前用户必须是该表空间的所有者或系统管理员，且该用户是新所有者角色的成员。
- 要修改表空间的所有者A为B，则A必须是B的直接或者间接成员。

说明

如果new_owner与old_owner一致，此处不再校验当前执行操作的用户是否具有修改权限，而直接显示ALTER成功。

语法格式

- 重命名表空间的语法。
```
ALTER TABLESPACE tablespace_name
    RENAME TO new_tablespace_name;
```
- 设置表空间所有者的语法。
```
ALTER TABLESPACE tablespace_name
    OWNER TO new_owner;
```
- 设置表空间属性的语法。
```
ALTER TABLESPACE tablespace_name
    SET ( { tablespace_option = value } [, ... ] );
```
- 重置表空间属性的语法。
```
ALTER TABLESPACE tablespace_name
    RESET ( { tablespace_option } [, ... ] );
```
- 设置表空间限额的语法
```
ALTER TABLESPACE tablespace_name
    RESIZE MAXSIZE { UNLIMITED | 'space_size'};
```

参数说明

- **tablespace_name**

  要修改的表空间。

  取值范围：已存在的表空间名。
- **new_tablespace_name**

  表空间的新名称。

  新名称不能以"PG_"开头。

  取值范围：字符串，符合**标识符命名规范**。
- **new_owner**

  表空间的新所有者。

  取值范围：已存在的用户名。
- **tablespace_option**

  设置或者重置表空间的参数。

  取值范围：
  - seq_page_cost：设置优化器计算一次顺序获取磁盘页面的开销。缺省为1.0。
  - random_page_cost：设置优化器计算一次非顺序获取磁盘页面的开销。缺省为4.0。

    说明

    - random_page_cost是相对于seq_page_cost的取值，等于或者小于seq_page_cost时毫无意义。
    - 默认值为4.0的前提条件是，优化器采用索引来扫描表数据，并且表数据在cache中命中率可以90%左右。
    - 如果表数据空间要比物理内存小，那么减小该值到一个适当水平；相反地，如果表数据在cache中命中率要低于90%，那么适当增大该值。
    - 如果采用了类似于SSD的随机访问代价较小的存储器，可以适当减小该值，以反映真正的随机扫描代价。

  value的取值范围：浮点类型的正数。
- **RESIZE MAXSIZE**

重新设置表空间限额的数值。

取值范围：

- UNLIMITED，该表空间不设置限额。
- 由space_size来确定，其格式参考**CREATE TABLESPACE**。

  说明

  - 若调整后的限额值比当前表空间实际使用的值要小，调整操作可以执行成功，后续用户需要将该表空间的使用值降低到新限额值之下，才能继续往该表空间中写入数据。
  - 修改参数MAXSIZE时也可使用：

```
ALTER TABLESPACE tablespace_name RESIZE MAXSIZE
 { 'UNLIMITED' | 'space_size'};
```

示例

请参考CREATE TABLESPACE的**示例**。

相关链接

CREATE TABLESPACE，**DROP TABLESPACE**

7.14.36 ALTER TRIGGER

功能描述

修改触发器名称。

说明

目前只支持修改名称。

注意事项

触发器所在表的所有者或者被授予了ALTER ANY SEQUENCE权限的用户可以执行ALTER TRIGGER操作，系统管理员默认拥有此权限。

语法格式

```
ALTER TRIGGER trigger_name ON table_name RENAME TO new_name;
```

参数说明

- **trigger_name**

  要修改的触发器名称。

  取值范围：已存在的触发器。
- **table_name**

  要修改的触发器所在的表名称。

  取值范围：已存在的含触发器的表。
- **new_name**

  修改后的新名称。

取值范围：符合**标识符命名规范**的字符串，最大长度不超过63个字符，且不能与所在表上其他触发器同名。

示例

请参见**CREATE TRIGGER**的示例。

相关链接

CREATE TRIGGER，**DROP TRIGGER**，**ALTER TABLE**

7.14.37 ALTER TYPE

功能描述

修改一个类型的定义。

注意事项

类型的所有者或者被授予了类型ALTER权限的用户或者被授予了ALTER ANY TYPE权限的用户可以执行ALTER TYPE命令，系统管理员默认拥有此权限。但要修改类型的所有者或者修改类型的模式，当前用户必须是该类型的所有者或者系统管理员，且该用户是新所有者角色的成员。

语法格式

- 修改类型。
  ```
  ALTER TYPE name action [, ... ];
  ```
  其中action对应的子句如下:
  - 给复合类型增加新的属性。
    ```
    ADD ATTRIBUTE attribute_name data_type [ COLLATE collation ] [ CASCADE | RESTRICT ]
    ```
  - 从复合类型中删除一个属性。
    ```
    DROP ATTRIBUTE [ IF EXISTS ] attribute_name [ CASCADE | RESTRICT ]
    ```
  - 改变一种复合类型中某个属性的类型。
    ```
    ALTER ATTRIBUTE attribute_name [ SET DATA ] TYPE data_type [ COLLATE collation ]
    [ CASCADE | RESTRICT ]
    ```
- 给复合类型增加新的属性。
  ```
  ALTER TYPE name ADD ATTRIBUTE attribute_name data_type [ COLLATE collation ] [ CASCADE |
  RESTRICT ];
  ```
- 从复合类型删除一个属性。
  ```
  ALTER TYPE name DROP ATTRIBUTE [ IF EXISTS ] attribute_name [ CASCADE | RESTRICT ];
  ```
- 改变一种复合类型中某个属性的类型。
  ```
  ALTER TYPE name ALTER ATTRIBUTE attribute_name [ SET DATA ] TYPE data_type [ COLLATE
  collation ] [ CASCADE | RESTRICT ];
  ```
- 改变类型的所有者。
  ```
  ALTER TYPE name OWNER TO { new_owner | CURRENT_USER | SESSION_USER };
  ```
- 改变类型的名称。
  ```
  ALTER TYPE name RENAME TO new_name;
  ```
- 改变一个复合类型中的一个属性的名称。
  ```
  ALTER TYPE name RENAME ATTRIBUTE attribute_name TO new_attribute_name [ CASCADE |
  RESTRICT ];
  ```
- 将类型移至一个新的模式中。


```
ALTER TYPE name SET SCHEMA new_schema;
```

- 为枚举类型增加一个新值。
```
ALTER TYPE name ADD VALUE [ IF NOT EXISTS ] new_enum_value [ { BEFORE | AFTER } neighbor_enum_value ];
```
- 重命名枚举类型的一个标签值。
```
ALTER TYPE name RENAME VALUE existing_enum_value TO new_enum_value;
```

参数说明

- **name**
  一个需要修改的现有的类型的名称(可以有模式修饰) 。
- **new_name**
  该类型的新名称。
- **new_owner**
  新所有者的用户名 。
- **new_schema**
  该类型的新模式 。
- **attribute_name**
  拟增加、更改或删除的属性的名称。
- **new_attribute_name**
  拟改名的属性的新名称。
- **data_type**
  拟新增属性的数据类型或是拟更改的属性的新类型名。
- **new_enum_value**
  枚举类型新增加的标签值，是一个非空的长度不超过63个字节的字符串。
- **neighbor_enum_value**
  一个已有枚举标签值，新值应该被增加在紧接着该枚举值之前或者之后的位置上。
- **existing_enum_value**
  现有的要重命名的枚举值，是一个非空的长度不超过63个字节的字符串
- **CASCADE**
  自动级联更新需更新类型以及相关联的记录和继承它们的子表。
- **RESTRICT**
  如果需联动更新类型是已更新类型的关联记录，则拒绝更新。这是缺省选项。

须知

- ADD ATTRIBUTE、DROP ATTRIBUTE和ALTER ATTRIBUTE选项可以组合成一个列表同时处理。 例如，在一条命令中同时增加几个属性或是更改几个属性的类型是可以实现的。
- 要修改一个类型的模式，必须在新模式上拥有CREATE权限。 要修改所有者，必须是新的所有角色的直接或间接成员， 并且该成员必须在此类型的模式上有CREATE权限。（这些限制强制了修改所有者不会做任何通过删除和重建类型不能做的事情。 不过，系统管理员可以以任何方式修改任意类型的所有权。）要增加一个属性或是修改一个属性的类型，也必须有该类型的USAGE权限。

- **CURRENT_USER**
  当前用户。
- **SESSION_USER**
  当前系统用户。
- **COLLATE collation**
  COLLATE子句为该列（必须是一种可排序数据类型）赋予一个排序规则。 如果没有指定，将使用该列数据类型的默认排序规则。

示例

请参考CREATE TYPE的**示例**。

相关链接

CREATE TYPE，**DROP TYPE**

7.14.38 ALTER USER

功能描述

修改数据库用户的属性。

注意事项

ALTER USER中修改的会话参数只针对指定的用户，且在下一次会话中有效。

语法格式

- 修改用户的权限等信息。
```
ALTER USER user_name [ [ WITH ] option [ ... ] ];
```
其中option子句为。
```
{ CREATEDB | NOCREATEDB }
  | { CREATEROLE | NOCREATEROLE }
  | { INHERIT | NOINHERIT }
  | { AUDITADMIN | NOAUDITADMIN }
  | { SYSADMIN | NOSYSADMIN }
  | {MONADMIN | NOMONADMIN}
  | {OPRADMIN | NOOPRADMIN}
  | {POLADMIN | NOPOLADMIN}
  | { USEFT | NOUSEFT }
  | { LOGIN | NOLOGIN }
  | { REPLICATION | NOREPLICATION }
  | {VCADMIN | NOVCADMIN}
  | {PERSISTENCE | NOPERSISTENCE}
  | CONNECTION LIMIT connlimit
  | [ ENCRYPTED | UNENCRYPTED ] PASSWORD { 'password' [EXPIRED] | DISABLE | EXPIRED }
  | [ ENCRYPTED | UNENCRYPTED ] IDENTIFIED BY { 'password' [ REPLACE 'old_password' |
EXPIRED ] | DISABLE }
  | VALID BEGIN 'timestamp'
  | VALID UNTIL 'timestamp'
  | RESOURCE POOL 'respool'
  | PERM SPACE 'spacelimit'
  | PGUSER
```
- 修改用户名。
```
ALTER USER user_name
  RENAME TO new_name;
```


- 锁定或解锁。
```
ALTER USER user_name
    ACCOUNT { LOCK | UNLOCK };
```
- 修改与用户关联的指定会话参数值。
```
ALTER USER user_name
    SET configuration_parameter { { TO | = } { value | DEFAULT } | FROM CURRENT };
```
- 重置与用户关联的指定会话参数值。
```
ALTER USER user_name
    RESET { configuration_parameter | ALL };
```

参数说明

- **user_name**

  现有用户名。

  取值范围：已存在的用户名，如果用户名中包含大写字母则需要使用双引号括起来。

- **new_password**

  新密码。

  密码规则如下：

  - 不能与当前密码相同。
  - 密码默认不少于8个字符。
  - 不能与用户名及用户名倒序相同。
  - 至少包含大写字母（A-Z），小写字母（a-z），数字（0-9），非字母数字字符（限定为~!@#$%^&*()-_=+\|[{}];:,<.>/?）四类字符中的三类字符。
  - 应当使用单引号将用户密码括起来。

  取值范围：字符串。

- **old_password**

  旧密码。

- **ACCOUNT { LOCK | UNLOCK }**
  - ACCOUNT LOCK：锁定账户，禁止登录数据库。
  - ACCOUNT UNLOCK：解锁账户，允许登录数据库。

- **PGUSER**

  当前版本不允许修改用户的PGUSER属性。

其他参数请参见**CREATE ROLE**和**ALTER ROLE**的参数说明。

示例

```
--创建用户jim，登录密码为********。
gaussdb=# CREATE USER jim PASSWORD '********';

--修改用户jim的登录密码。
gaussdb=# ALTER USER jim IDENTIFIED BY '**********' REPLACE '********';

--将enable_seqscan的值设置为on， 设置成功后，在下一会话中生效。
gaussdb=# ALTER USER jim SET enable_seqscan TO on;

--重置jim的enable_seqscan参数。
gaussdb=# ALTER USER jim RESET enable_seqscan;

--锁定jim账户。
gaussdb=# ALTER USER jim ACCOUNT LOCK;
```



```
--解锁jim账户。
gaussdb=# ALTER USER jim ACCOUNT UNLOCK;

--修改用户名。
gaussdb=# ALTER USER jim RENAME TO lisa;

--删除用户。
gaussdb=# DROP USER lisa CASCADE;
```

相关链接

CREATE ROLE，**CREATE USER**，**DROP USER**

7.14.39 ALTER USER MAPPING

功能描述

ALTER USER MAPPING语句用于更改一个用户到一个外部服务器的映射定义，外部服务器的所有者可以为任何用户更改该服务器的用户映射。此外，如果服务器上的USAGE权限已授权用户，则用户可以更改其自己的用户名的用户映射。

注意事项

- 当在OPTIONS中出现password选项时，需要保证GaussDB每个节点的$GAUSSHOME/bin目录下存在usermapping.key.cipher和usermapping.key.rand文件，如果不存在这两个文件，请使用gs_guc工具生成并使用gs_ssh工具发布到每个节点的$GAUSSHOME/bin目录下。具体操作请参考**OPTIONS**中的说明。
- OPTIONS中的敏感字段（如password）在使用多层引号时，语义和不带引号的场景是不同的，因此不会被识别为敏感字段进行脱敏。

语法格式

```
ALTER USER MAPPING FOR { user_name | USER | CURRENT_USER | PUBLIC }
    SERVER server_name
    OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ... ] )
```

在OPTIONS选项里，ADD、SET和DROP指定要执行的操作，未指定时默认为ADD操作。option和value为对应操作的参数及参数值。

参数说明

- **user_name**

  该映射的用户名。

  CURRENT_USER和USER匹配当前用户的名称。PUBLIC被用来匹配系统中所有当前以及未来的用户名。

- **server_name**

  该用户映射的服务器名。

- **OPTIONS**

  为该用户映射更改选项。新选项会覆盖任何之前指定的选项。ADD、 SET和DROP指定要被执行的动作。如果没有显式地指定操作，将假定为ADD。选项名称必须为唯一，该服务器的外部数据包装器也会验证选项。

说明

- 用户的密码会加密后保存到系统表PG_USER_MAPPING中，加密时需要使用usermapping.key.cipher和usermapping.key.rand作为加密密码文件和加密因子。首次使用前需要通过如下命令创建这两个文件，并将这两个文件放入各节点的$GAUSSHOME/bin目录，且确保具有读权限。gs_ssh工具可以协助您快速将文件放入各节点对应目录下。
  ```
  gs_ssh -c "gs_guc generate -o usermapping -S default -D $GAUSSHOME/bin"
  ```
- 其中-S参数指定default时会随机生成密码，用户也可为-S参数指定密码，此密码用于保证生成密码文件的安全性和唯一性，用户无需保存或记忆。其他参数详见工具参考中gs_guc工具说明。

示例

请参考CREATE USER MAPPING的8.15.104-示例。

相关链接

CREATE USER MAPPING，DROP USER MAPPING

7.14.40 ALTER VIEW

功能描述

ALTER VIEW更改视图的各种辅助属性。（如果用户是更改视图的查询定义，要使用CREATE OR REPLACE VIEW。）

注意事项

只有视图的所有者或者被授予了视图ALTER权限的用户才可以执行ALTER VIEW命令，系统管理员默认拥有该权限。针对所要修改属性的不同，对其还有以下权限约束：

- 修改视图的模式，当前用户必须是视图的所有者或者系统管理员，且要有新模式的CREATE权限。
- 修改视图的所有者，当前用户必须是视图的所有者或者系统管理员，且该用户必须是新所有者角色的成员，并且此角色必须有视图所在模式的CREATE权限。
- 禁止修改视图中列的类型。

语法格式

- 设置视图列的默认值。
  ```
  ALTER VIEW [ IF EXISTS ] view_name
      ALTER [ COLUMN ] column_name SET DEFAULT expression;
  ```
- 取消列视图列的默认值。
  ```
  ALTER VIEW [ IF EXISTS ] view_name
      ALTER [ COLUMN ] column_name DROP DEFAULT;
  ```
- 修改视图的所有者。
  ```
  ALTER VIEW [ IF EXISTS ] view_name
      OWNER TO new_owner;
  ```
- 重命名视图。
  ```
  ALTER VIEW [ IF EXISTS ] view_name
      RENAME TO new_name;
  ```
- 设置视图的所属模式。
  ```
  ALTER VIEW [ IF EXISTS ] view_name
      SET SCHEMA new_schema;
  ```


- 设置视图的选项。
```
ALTER VIEW [ IF EXISTS ] view_name
    SET ( { view_option_name [ = view_option_value ] } [, ... ] );
```
- 重置视图的选项。
```
ALTER VIEW [ IF EXISTS ] view_name
    RESET ( view_option_name [, ... ] );
```

参数说明

- **IF EXISTS**

  使用这个选项，如果视图不存在时不会产生错误，仅有会有一个提示信息。
- **view_name**

  视图名称，可以用模式修饰。

  取值范围：字符串，符合**标识符命名规范**。
- **column_name**

  可选的名称列表，视图的字段名。如果没有给出，字段名取自查询中的字段名。

  取值范围：字符串，符合**标识符命名规范**。
- **SET/DROP DEFAULT**

  设置或删除一个列的缺省值，该参数暂无实际意义。
- **new_owner**

  视图新所有者的用户名称。
- **new_name**

  视图的新名称。
- **new_schema**

  视图的新模式。
- **view_option_name [= view_option_value]**

  该子句为视图指定一个可选的参数。

  目前view_option_name支持的参数仅有security_barrier，当VIEW试图提供行级安全时，应使用该参数。

  取值范围：Boolean类型，TRUE、FALSE。

示例

```
--创建SCHEMA。
gaussdb=# CREATE SCHEMA tpcds;

--创建表tpcds.customer。
gaussdb=# CREATE TABLE tpcds.customer
(
c_customer_sk         INTEGER           NOT NULL,
c_customer_id         CHARACTER(16)     NOT NULL
);

--向表中插入多条记录。
gaussdb=# INSERT INTO tpcds.customer VALUES (1, 'AAAAAAAABAAAAAAA'),(100,
'AAAAAAAACAAAAAAA'),(150, 'AAAAAAAADAAAAAAA');

--创建一个由c_customer_sk小于150的内容组成的视图。
gaussdb=# CREATE VIEW tpcds.customer_details_view_v1 AS
    SELECT * FROM tpcds.customer
    WHERE c_customer_sk < 150;
```



```
--修改视图名称。
gaussdb=# ALTER VIEW tpcds.customer_details_view_v1 RENAME TO customer_details_view_v2;

--修改视图所属schema。
gaussdb=# ALTER VIEW tpcds.customer_details_view_v2 SET schema public;

--删除视图。
gaussdb=# DROP VIEW public.customer_details_view_v2;

--删除表tpcds.customer。
gaussdb=# DROP TABLE tpcds.customer;

--删除SCHEMA。
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

相关链接

CREATE VIEW，**DROP VIEW**

7.14.41 ANALYZE | ANALYSE

功能描述

- 用于收集与数据库中普通表内容相关的统计信息，统计结果存储在系统表PG_STATISTIC、PG_STATISTIC_EXT下，执行ANALYZE命令后，可在上述系统表或系统视图PG_STATS、PG_EXT_STATS内查询收集到的统计信息。执行计划生成器会使用这些统计数据，以确定最有效的执行计划。
- 如果没有指定参数，ANALYZE会分析当前数据库中的每个表和分区表。同时也可以通过指定table_name、column_name和partition_name参数把分析限定在特定的表、列或分区表中。
- ANALYZE|ANALYSE VERIFY用于检测数据库中普通表的数据文件是否损坏。

注意事项

- ANALYZE非临时表不能在一个匿名块、事务块、函数或存储过程内被执行。支持存储过程中ANALYZE临时表，不支持统计信息回滚操作。
- ANALYZE VERIFY 场景不触发远程读，因此远程读参数不生效。对于关键系统表出现错误被系统检测出页面损坏时，将直接报错不再继续检测。
- 如果没有指定参数，ANALYZE处理当前数据库里用户拥有相应权限的每个表。如果参数中指定了表，ANALYZE只处理指定的表。
- 要对一个表进行ANALYZE操作，通常用户必须是表的所有者或者被授予了指定表VACUUM权限的用户，默认系统管理员有该权限。数据库的所有者允许对数据库中除了共享目录以外的所有表进行ANALYZE操作（该限制意味着只有系统管理员才能真正对一个数据库进行ANALYZE操作）。ANALYZE命令会跳过那些用户没有权限的表。
- ANALYZE不收集无法做比较或等值运算的列，例如cursor类型。
- 如果拟分析的表成了一个空表，ANALYZE不会记录该表的统计信息，而原来已有统计信息则会保留。

语法格式

- 收集表的统计信息。

```
{ ANALYZE | ANALYSE } [ VERBOSE ]
    [ table_name [ ( column_name [, ...] ) ] ];
```


- 收集分区表的分区统计信息。该语法在功能上尚不支持。
```
{ ANALYZE | ANALYSE } [ VERBOSE ]
    table_name [ ( column_name [, ...] ) ] PARTITION ( patrition_name ) ;
```

  说明

  普通分区表目前支持针对某个分区的统计信息的语法，但功能上不支持针对某个分区的统计信息收集。

- 手动收集多列统计信息。
```
{ANALYZE | ANALYSE} [ VERBOSE ]
    table_name (( column_1_name, column_2_name [, ...] ));
```

  说明

  - 如果关闭GUC参数enable_functional_dependency，每组多列统计信息最多支持32列；如果开启GUC参数enable_functional_dependency，每组多列统计信息最多支持4列。
  - 不支持收集多列统计信息的表：系统表，全局临时表。

- 自动收集多列统计信息。

  打开auto_statistic_ext_columns参数后执行ANALYZE，自动根据该表的索引前缀创建多列统计信息，多列统计信息的列数不超过设置的auto_statistic_ext_columns值。

  例：表t存在索引(a,b,c,d)，设置auto_statistic_ext_columns参数为4，则analyze t将创建关于(a,b)、(a,b,c)、(a,b,c,d)的多列统计信息。
```
{ ANALYZE | ANALYSE } [ VERBOSE ] table_name;
```

- 检测当前库的数据文件。
```
{ANALYZE | ANALYSE} VERIFY {FAST|COMPLETE};
```

  说明

  - Fast模式校验时，需要对校验的表有并发的DML操作，会导致校验过程中有误报的问题，因为当前Fast模式是直接从磁盘上读取，有其他线程并发修改文件时，会导致获取的数据不准确，建议离线操作。
  - 支持对全库进行操作，由于涉及的表较多，建议以重定向保存结果。
```
gsql -d database -p port -f sqlfile> sqllog.txt  2>&1
```
  - 只有对外可见的表对外提示NOTICE，内部表的检测会包含在它所依赖的外部表，不对外显示和呈现。
  - 此命令的处理可容错ERROR级别的处理。
  - 对于全库操作时，当关键系统表出现损坏则直接报错，不再继续执行。

- 检测表和索引的数据文件
```
{ANALYZE | ANALYSE} VERIFY {FAST|COMPLETE} { table_name|index_name } [CASCADE];
```

  说明

  - 支持对普通表的操作和索引表的操作，但不支持对索引表index使用CASCADE操作。原因是由于CASCADE模式用于处理主表的所有索引表，当单独对索引表进行检测时，无需使用CASCADE模式。
  - 对于主表的检测会同步检测主表的内部表，例如toast表等。
  - 当提示索引表损坏时，建议使用reindex命令进行重建索引操作。

- 检测表分区的数据文件
```
{ANALYZE | ANALYSE} VERIFY {FAST|COMPLETE} table_name PARTITION (patrition_name) [CASCADE];
```

  说明

  支持对表的单独分区进行检测操作，但不支持对索引表index使用CASCADE操作。

参数说明

- **VERBOSE**

  启用显示进度信息。

  说明

  如果指定了VERBOSE，ANALYZE发出进度信息，表明目前正在处理的表。各种有关表的统计信息也会打印出来。

- **table_name**

  需要分析的特定表的表名（可能会带模式名），如果省略，将对数据库中的所有表（非外部表）进行分析。

  对于ANALYZE收集统计信息，目前仅支持行存表。

  取值范围：已有的表名。

- **column_name，column_1_name，column_2_name**

  需要分析特定列的列名，默认为所有列。

  取值范围：已有的列名。

- **partition_name**

  如果table为分区表，在关键字PARTITION后面指定分区名partition_name表示分析该分区表的统计信息。目前语法上支持分区表做ANALYZE，但功能实现上暂不支持对指定分区统计信息的分析。

  取值范围：表的某一个分区名。

- **index_name**

  需要分析的特定索引表的表名（可能会带模式名）。

  取值范围：已有的表名。

- **FAST|COMPLETE**

  FAST模式下主要对于表的CRC和page header进行校验，如果校验失败则会告警；而COMPLETE模式下，则主要对表的指针、tuple进行解析校验。

- **CASCADE**

  CASCADE模式下会对当前表的所有索引进行检测处理。

示例

```
--创建表。
gaussdb=# CREATE TABLE customer_info
(
WR_RETURNED_DATE_SK       INTEGER                  ,
WR_RETURNED_TIME_SK       INTEGER                  ,
WR_ITEM_SK                INTEGER            NOT NULL,
WR_REFUNDED_CUSTOMER_SK   INTEGER
)
;
--创建分区表
gaussdb=# CREATE TABLE customer_par
(
WR_RETURNED_DATE_SK       INTEGER                  ,
WR_RETURNED_TIME_SK       INTEGER                  ,
WR_ITEM_SK                INTEGER            NOT NULL,
WR_REFUNDED_CUSTOMER_SK   INTEGER
)
PARTITION BY RANGE(WR_RETURNED_DATE_SK)
(
PARTITION P1 VALUES LESS THAN(2452275),
PARTITION P2 VALUES LESS THAN(2452640),
```



```
PARTITION P3 VALUES LESS THAN(2453000),
PARTITION P4 VALUES LESS THAN(MAXVALUE)
)
ENABLE ROW MOVEMENT;
--使用ANALYZE语句更新统计信息。
gaussdb=# ANALYZE customer_info;
gaussdb=# ANALYZE customer_par;
--使用ANALYZE VERBOSE语句更新统计信息，并输出表的相关信息。
gaussdb=# ANALYZE VERBOSE customer_info;
INFO:  ANALYZE INFO : estimate total rows of "customer_info": scanned 0 pages of total 0 pages with 1 retry times, containing 0 live rows and 0 dead rows,  estimated 0 total rows(datanode pid=38661)
INFO:  ANALYZE INFO : "customer_info": scanned 0 of 0 pages, containing 0 live rows and 0 dead rows; 0 rows in sample, 0 estimated total rows(datanode pid=38661)
ANALYZE
```

说明

若环境若有故障，需查看数据库主节点的log。

```
--删除表。
gaussdb=# DROP TABLE customer_info;
gaussdb=# DROP TABLE customer_par;
```

7.14.42 BEGIN

功能描述

BEGIN可以用于开始一个匿名块，也可以用于开始一个事务。本节描述用BEGIN开始匿名块的语法，以BEGIN开始事务的语法见**START TRANSACTION**。

匿名块是能够动态地创建和执行过程代码的结构，而不需要以持久化的方式将代码作为数据库对象储存在数据库中。

注意事项

无。

语法格式

- 开启匿名块。

```
[DECLARE [declare_statements]]
BEGIN
execution_statements
END;
/
```

- 开启事务。

```
BEGIN [ WORK | TRANSACTION ]
 [
  {
    ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE | REPEATABLE READ }
    | { READ WRITE | READ ONLY }
   } [, ...]
 ];
```

参数说明

- **declare_statements**

  声明变量，包括变量名和变量类型，如“sales_cnt int”。

- **execution_statements**

  匿名块中要执行的语句。

取值范围：DML操作(数据操纵操作：select、insert、delete、update)或系统表中已注册的函数名称。

示例

```
--使用匿名块输出字符串。
gaussdb=# BEGIN
dbe_output.print_line('Hello');
END;
/
```

相关链接

START TRANSACTION

7.14.43 CALL

功能描述

使用CALL命令可以调用已定义的函数和存储过程。

注意事项

函数或存储过程的所有者、被授予了函数或存储过程EXECUTE权限的用户或被授予EXECUTE ANY FUNCTION权限的用户有权调用函数或存储过程，系统管理员默认拥有此权限。

语法格式

```
CALL [schema.|package.] {func_name| procedure_name} ( param_expr );
```

参数说明

- **schema**

  函数或存储过程所在的模式名称。

- **package**

  函数或存储过程所在的package名称。

- **func_name**

  所调用函数或存储过程的名称。

  取值范围：已存在的函数名称。

  说明

  支持使用DATABASE LINK方式对远端函数或存储过程进行操作，使用方式详情请见DATABASE LINK。

- **param_expr**

  参数列表可以用符号 “:=”或者“=>”将参数名和参数值隔开，这种方法的好处是参数可以以任意顺序排列。若参数列表中仅出现参数值，则参数值的排列顺序必须和函数或存储过程定义时的相同。

  取值范围：已存在的函数参数名称或存储过程参数名称。

说明

参数可以包含入参（参数名和类型之间指定“IN”关键字）和出参（参数名和类型之间指定“OUT”关键字），使用CALL命令调用函数或存储过程时，对于非重载的函数，参数列表必须包含出参，出参可以传入一个变量或者任一常量，详见**示例**。对于重载的package函数，参数列表里可以忽略出参，忽略出参时可能会导致函数找不到。包含出参时，出参只能是常量。

示例

```
--创建一个函数func_add_sql，计算两个整数的和，并返回结果。
gaussdb=# CREATE FUNCTION func_add_sql(num1 integer, num2 integer) RETURN integer
AS
BEGIN
RETURN num1 + num2;
END;
/

--按参数值传递。
gaussdb=# CALL func_add_sql(1, 3);

--使用命名标记法传参。
gaussdb=# CALL func_add_sql(num1 => 1,num2 => 3);
gaussdb=# CALL func_add_sql(num2 := 2, num1 := 3);

--删除函数。
gaussdb=# DROP FUNCTION func_add_sql;

--创建带出参的函数。
gaussdb=# CREATE FUNCTION func_increment_sql(num1 IN integer, num2 IN integer, res OUT integer)
RETURN integer
AS
BEGIN
res := num1 + num2;
END;
/

--出参传入常量。
gaussdb=# CALL func_increment_sql(1,2,1);

--删除函数。
gaussdb=# DROP FUNCTION func_increment_sql;
```

相关链接

CREATE FUNCTION，**CREATE PROCEDURE**

7.14.44 CHECKPOINT

功能描述

检查点（CHECKPOINT）是一个事务日志中的点，所有数据文件都在该点被更新以反映日志中的信息，所有数据文件都将被刷新到磁盘。

设置事务日志检查点。预写式日志（WAL）缺省时在事务日志中每隔一段时间放置一个检查点。可以使用gs_guc命令设置相关运行时参数（checkpoint_segments，checkpoint_timeout和incremental_checkpoint_timeout）来调整这个原子化检查点的间隔。

注意事项

- 只有系统管理员和运维管理员可以调用CHECKPOINT。
- CHECKPOINT强制立即进行检查，而不是等到下一次调度时的检查点。

语法格式

```
CHECKPOINT;
```

参数说明

无。

示例

```
--设置检查点。
gaussdb=# CHECKPOINT;
```

7.14.45 CLEAN CONNECTION

功能描述

用来清理数据库连接。允许在节点上清理指定数据库的指定用户的相关连接。

注意事项

- GaussDB下不支持指定节点，仅支持TO ALL。
- 该功能仅在force模式下，可以清理正在使用的正常连接。

语法格式

```
CLEAN CONNECTION
    TO { COORDINATOR ( nodename [, ... ] ) | NODE ( nodename [, ... ] )| ALL [ CHECK ] [ FORCE ] }
    [ FOR DATABASE dbname ]
    [ TO USER username ];
```

参数说明

- **CHECK**

  仅在节点列表为TO ALL时可以指定。如果指定该参数，会在清理连接之前检查数据库是否被其他会话连接访问。此参数主要用于DROP DATABASE之前的连接访问检查，如果发现有其他会话连接，则将报错并停止删除数据库。

- **FORCE**

  仅在节点列表为TO ALL时可以指定，如果指定该参数，所有和指定dbname和username相关的线程都会收到SIGTERM信号，然后被强制关闭。

- **COORDINATOR (nodename [, ...]) | NODE (nodename [, ...]) | ALL**

  仅支持TO ALL，必须指定该参数，节点上的指定连接会被全部删除。

- **dbname**

  删除指定数据库上的连接。如果不指定，则删除所有数据库的连接。

  取值范围：已存在数据库名。

- **username**

  删除指定用户上的连接。如果不指定，则删除所有用户的连接。

取值范围：已存在的用户。

示例

```
--创建数据库test_clean_connection。
gaussdb=# CREATE DATABASE test_clean_connection;

--创建jack用户。
gaussdb=# CREATE USER jack PASSWORD '********';

--删除用户jack在数据库template1上的所有连接。
gaussdb=# CLEAN CONNECTION TO ALL FOR DATABASE template1 TO USER jack;

--删除用户jack的所有连接。
gaussdb=# CLEAN CONNECTION TO ALL TO USER jack;

--删除在数据库test_clean_connection上的所有连接。
gaussdb=# CLEAN CONNECTION TO ALL FORCE FOR DATABASE test_clean_connection;

--删除用户jack。
gaussdb=# DROP USER jack;

--删除数据库test_clean_connection。
gaussdb=# DROP DATABASE test_clean_connection;
```

7.14.46 CLOSE

功能描述

CLOSE释放和一个游标关联的所有资源。

注意事项

- 不允许对一个已关闭的游标再做任何操作。
- 一个不再使用的游标应该尽早关闭。
- 当创建游标的事务用COMMIT或ROLLBACK终止之后，每个不可保持的已打开游标都隐含关闭。
- 当创建游标的事务通过ROLLBACK退出之后，每个可以保持的游标都将隐含关闭。
- 当创建游标的事务成功提交，可保持的游标将保持打开，直到执行一个明确的CLOSE或者客户端断开。
- GaussDB没有明确打开游标的OPEN语句，因为一个游标在使用CURSOR命令定义的时候就打开了。可以通过查询系统视图pg_cursors看到所有可用的游标。

语法格式

```
CLOSE { cursor_name | ALL } ;
```

参数说明

- **cursor_name**

  一个待关闭的游标名称。

- **ALL**

  关闭所有已打开的游标。

示例

请参考FETCH的**示例**。

相关链接

FETCH，**MOVE**

7.14.47 CLUSTER

功能描述

- 根据一个索引对表进行聚簇排序。
- CLUSTER指定GaussDB通过索引名指定的索引聚簇由表名指定的表。 表名上必须已经定义该索引。
- 当对一个表聚集后，该表将基于索引信息进行物理存储。聚集是一次性操作：当表被更新之后， 更改的内容不会被聚集。也就是说，系统不会试图按照索引顺序对新的存储内容及更新记录进行重新聚集。
- 在对一个表聚簇之后，GaussDB会记录该表在哪个索引上建立了聚簇。CLUSTER table_name将在该表之前记录过的聚簇索引上重新聚簇。用户也可以用ALTER TABLE table_name CLUSTER on index_name来设置指定表用于后续聚簇操作的索引，或使用ALTER TABLE table_name SET WITHOUT CLUSTER来清除指定表之前设置的聚簇索引。
- 不含参数的CLUSTER命令会将当前用户所拥有的数据库中的先前做过聚簇的所有表重新处理，或者系统管理员调用的这些表。
- 在对一个表进行聚簇的时候，会在其上请求一个ACCESS EXCLUSIVE锁。这样就避免了在CLUSTER完成之前对此表执行其它的操作(包括读写)。

注意事项

- 只有行存B-tree索引支持CLUSTER操作。
- 如果用户只是随机访问表中的行，那么表中数据的实际存储顺序是无关紧要的。但是，如果对某些特定数据的访问次数较多，而且有一个索引将这些数据分组，那么使用CLUSTER索引对性能会有所提升。
- 如果一个请求从表中查找的索引是一个范围，或者是一个索引值对应多行，CLUSTER也会有助于应用，因为如果索引标识出了第一匹配行所在的存储页，所有其它行也可能也已经在同一个存储页里了，这样便节省了磁盘访问的时间，加速了查询。
- 在聚簇过程中，系统会先创建一个按照索引顺序建立的表的临时备份，同时也建立表上的每个索引的临时备份。因此，聚簇过程中需要保证磁盘上有足够的剩余空间，至少是表大小与全部索引大小之和。
- 因为CLUSTER记录着哪些索引曾被用于聚簇，所以用户可以在第一次手动指定索引，对指定表进行聚簇，然后设置一个周期化执行的维护脚本，只需执行不带参数的CLUSTER命令，就可以实现对想要周期性聚簇的表进行自动更新。
- 因为优化器记录着有关表的排序的统计，在表上执行聚簇操作后，需运行ANALYZE操作以确保优化器具备最新的排序信息，否则，优化器可能会选择非最优的查询规划。
- CLUSTER不允许在事务中执行。

- 如果没有打开xc_maintenance_mode参数，那么CLUSTER操作将跳过所有系统表。

语法格式

- 对一个表进行聚簇排序。
  ```
  CLUSTER [ VERBOSE ] table_name [ USING index_name ];
  ```
- 对一个分区进行聚簇排序。
  ```
  CLUSTER [ VERBOSE ] table_name PARTITION ( partition_name ) [ USING index_name ];
  ```
- 对已做过聚簇的表重新进行聚簇。
  ```
  CLUSTER [ VERBOSE ];
  ```

参数说明

- **VERBOSE**

  启用显示进度信息。
- **table_name**

  表名称。

  取值范围：已存在的表名称。
- **index_name**

  索引名称。

  取值范围：已存在的索引名称。
- **partition_name**

  分区名称。

  取值范围：已存在的分区名称。

示例

```
--创建SCHEMA。
gaussdb=# CREATE SCHEMA tpcds;

--创建一个分区表。
gaussdb=# CREATE TABLE tpcds.inventory_p1
(
   INV_DATE_SK             INTEGER              NOT NULL,
   INV_ITEM_SK             INTEGER              NOT NULL,
   INV_WAREHOUSE_SK        INTEGER              NOT NULL,
   INV_QUANTITY_ON_HAND    INTEGER
)
PARTITION BY RANGE(INV_DATE_SK)
(
      PARTITION P1 VALUES LESS THAN(2451179),
      PARTITION P2 VALUES LESS THAN(2451544),
      PARTITION P3 VALUES LESS THAN(2451910),
      PARTITION P4 VALUES LESS THAN(2452275),
      PARTITION P5 VALUES LESS THAN(2452640),
      PARTITION P6 VALUES LESS THAN(2453005),
      PARTITION P7 VALUES LESS THAN(MAXVALUE)
);

--创建索引ds_inventory_p1_index1。
gaussdb=# CREATE INDEX ds_inventory_p1_index1 ON tpcds.inventory_p1 (INV_ITEM_SK) LOCAL;

--对表tpcds.inventory_p1进行聚集。
gaussdb=# CLUSTER tpcds.inventory_p1 USING ds_inventory_p1_index1;

--对分区p3进行聚集。
gaussdb=# CLUSTER tpcds.inventory_p1 PARTITION (p3) USING ds_inventory_p1_index1;
```



```
--对数据库中可以进行聚集的表进聚集。
gaussdb=# CLUSTER;

--删除索引。
gaussdb=# DROP INDEX tpcds.ds_inventory_p1_index1;

--删除分区表。
gaussdb=# DROP TABLE tpcds.inventory_p1;

--删除SCHEMA。
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

优化建议

- cluster
  - 建议在新近聚簇的表上运行ANALYZE。否则，优化器可能会选择很差劲的查询规划。
  - 不允许在事务中执行CLUSTER。

7.14.48 COMMENT

功能描述

定义或修改一个对象的注释。

注意事项

- 每个对象只存储一条注释，因此要修改一个注释，对同一个对象发出一条新的COMMENT命令即可。要删除注释，在文本字符串的位置写上NULL即可。当删除对象时，注释自动被删除掉。
- 目前注释浏览没有安全机制：任何连接到某数据库上的用户都可以看到所有该数据库对象的注释。共享对象（比如数据库、角色、表空间）的注释是全局存储的，连接到任何数据库的任何用户都可以看到它们。因此，不要在注释里存放与安全有关的敏感信息。
- 对大多数对象，只有对象的所有者或者被授予了对象COMMENT权限的用户可以设置注释，系统管理员默认拥有该权限。
- 角色没有所有者，所以COMMENT ON ROLE命令仅可以由系统管理员对系统管理员角色执行，有CREATEROLE权限的角色也可以为非系统管理员角色设置注释。系统管理员可以对所有对象进行注释。

语法格式

```
COMMENT ON
{
  AGGREGATE agg_name (agg_type [, ...] ) |
  CAST (source_type AS target_type) |
  COLLATION object_name |
  COLUMN { table_name.column_name | view_name.column_name } |
  CONSTRAINT constraint_name ON table_name |
  CONVERSION object_name |
  DATABASE object_name |
  DOMAIN object_name |
  EXTENSION object_name |
  FOREIGN DATA WRAPPER object_name |

  FUNCTION function_name ( [ {[ argname ] [ argmode ] argtype} [, ...] ] ) |
  INDEX object_name |
```



```
  LARGE OBJECT large_object_oid |
  OPERATOR operator_name (left_type, right_type) |
  OPERATOR CLASS object_name USING index_method |
  OPERATOR FAMILY object_name USING index_method |
  [ PROCEDURAL ] LANGUAGE object_name |
  ROLE object_name |
  RULE rule_name ON table_name |
  SCHEMA object_name |
  SERVER object_name |
  TABLE object_name |
  TABLESPACE object_name |
  TEXT SEARCH CONFIGURATION object_name |
  TEXT SEARCH DICTIONARY object_name |
  TEXT SEARCH PARSER object_name |
  TEXT SEARCH TEMPLATE object_name |
  TYPE object_name |
  VIEW object_name |
  TRIGGER trigger_name ON table_name
}
   IS 'text';
```

参数说明

- **agg_name**
  聚集函数的名称。
- **agg_type**
  聚集函数参数的类型。
- **source_type**
  类型转换的源数据类型。
- **target_type**
  类型转换的目标数据类型。
- **object_name**
  对象名。
- **table_name.column_name**
  view_name.column_name
  列名称。前缀可加表名称或者视图名称。
- **constraint_name**
  表约束的名称。
- **table_name**
  表的名称。
- **function_name**
  函数名称。
- **argname,argmode,argtype**
  函数参数的名称、模式、类型。
- **large_object_oid**
  大对象的OID。
- **operator_name**
  操作符名称。
- **left_type,right_type**

操作参数的数据类型（可以用模式修饰）。当前置或者后置操作符不存在时，可以增加NONE选项。

- **trigger_name**

  触发器名称。

- **text**

  注释。

示例

```
--创建SCHEMA。
gaussdb=# CREATE SCHEMA tpcds;

--创建表tpcds.customer。
gaussdb=# CREATE TABLE tpcds.customer
(
c_customer_sk         INTEGER        NOT NULL,
c_customer_id         CHAR(16)       NOT NULL
);

--向表中插入多条记录。
gaussdb=# INSERT INTO tpcds.customer VALUES (50, 'AAAAAAAABAAAAAAA'),(100,
'AAAAAAAACAAAAAAA'),(150, 'AAAAAAAADAAAAAAA');

--创建表tpcds.customer_demographics_t2。
gaussdb=# CREATE TABLE tpcds.customer_demographics_t2
(
    CD_DEMO_SK                INTEGER               NOT NULL,
    CD_GENDER                 CHAR(1)                     ,
    CD_MARITAL_STATUS         CHAR(1)                     ,
    CD_EDUCATION_STATUS       CHAR(20)                    ,
    CD_PURCHASE_ESTIMATE      INTEGER                     ,
    CD_CREDIT_RATING          CHAR(10)                    ,
    CD_DEP_COUNT              INTEGER                     ,
    CD_DEP_EMPLOYED_COUNT     INTEGER                     ,
    CD_DEP_COLLEGE_COUNT      INTEGER
)
;

--为tpcds.customer_demographics_t2.cd_demo_sk列加注释。
gaussdb=# COMMENT ON COLUMN tpcds.customer_demographics_t2.cd_demo_sk IS 'Primary key of
customer demographics table.';

--创建一个由c_customer_sk小于150的内容组成的视图。
gaussdb=# CREATE VIEW tpcds.customer_details_view_v2 AS
    SELECT *
    FROM tpcds.customer
    WHERE c_customer_sk < 150;

--为tpcds.customer_details_view_v2视图加注释。
gaussdb=# COMMENT ON VIEW tpcds.customer_details_view_v2 IS 'View of customer detail';

--删除view。
gaussdb=# DROP VIEW tpcds.customer_details_view_v2;

--删除tpcds.customer_demographics_t2。
gaussdb=# DROP TABLE tpcds.customer_demographics_t2;

--删除表tpcds.customer。
gaussdb=# DROP TABLE tpcds.customer;

--删除SCHEMA。
gaussdb=# DROP SCHEMA tpcds CASCADE;
```


7.14.49 COMMIT | END

功能描述

通过COMMIT或者END可完成提交事务的功能，即提交事务的所有操作。

注意事项

执行COMMIT这个命令的时候，命令执行者必须是该事务的创建者或系统管理员，且创建和提交操作可以不在同一个会话中。

语法格式

```
{ COMMIT | END } [ WORK | TRANSACTION ] ;
```

参数说明

- **COMMIT | END**
  提交当前事务，让所有当前事务的更改为其他事务可见。
- **WORK | TRANSACTION**
  可选关键字，除了增加可读性没有其他任何作用。

示例

```
--创建SCHEMA。
gaussdb=# CREATE SCHEMA tpcds;

--创建表。
gaussdb=# CREATE TABLE tpcds.customer_demographics_t2
(
   CD_DEMO_SK               INTEGER               NOT NULL,
   CD_GENDER                CHAR(1)                       ,
   CD_MARITAL_STATUS        CHAR(1)                       ,
   CD_EDUCATION_STATUS      CHAR(20)                      ,
   CD_PURCHASE_ESTIMATE     INTEGER                       ,
   CD_CREDIT_RATING         CHAR(10)                      ,
   CD_DEP_COUNT             INTEGER                       ,
   CD_DEP_EMPLOYED_COUNT    INTEGER                       ,
   CD_DEP_COLLEGE_COUNT     INTEGER
)
;

--开启事务。
gaussdb=# START TRANSACTION;

--插入数据。
gaussdb=# INSERT INTO tpcds.customer_demographics_t2 VALUES(1,'M', 'U', 'DOCTOR DEGREE', 1200,
'GOOD', 1, 0, 0);
gaussdb=# INSERT INTO tpcds.customer_demographics_t2 VALUES(2,'F', 'U', 'MASTER DEGREE', 300, 'BAD',
1, 0, 0);

--提交事务，让所有更改永久化。
gaussdb=# COMMIT;

--查询数据。
gaussdb=# SELECT * FROM tpcds.customer_demographics_t2;

--删除表tpcds.customer_demographics_t2。
gaussdb=# DROP TABLE tpcds.customer_demographics_t2;

--删除SCHEMA。
gaussdb=# DROP SCHEMA tpcds CASCADE;
```


相关链接

ROLLBACK

7.14.50 COMMIT PREPARED

功能描述

提交一个早先为两阶段提交准备好的事务。

注意事项

- 该功能仅在维护模式（GUC参数xc_maintenance_mode为on时）下可用。该模式谨慎打开，一般供维护人员排查问题使用，一般用户不应使用该模式。
- 命令执行者必须是该事务的创建者或系统管理员，且创建和提交操作可以不在同一个会话中。
- 事务功能由数据库自动维护，不应显式使用事务功能。

语法格式

```
COMMIT PREPARED transaction_id [WITH commit_sequence_number];
```

参数说明

- **transaction_id**
  待提交事务的标识符。它不能和任何当前预备事务已经使用了的标识符同名。
- **commit_sequence_number**
  待提交事务的序列号。它是一个64位递增无符号数。

示例

```
--开始。
gaussdb=# begin;

--准备标识符为的trans_test的事务。
gaussdb=# PREPARE TRANSACTION 'trans_test';

--创建表。
gaussdb=# CREATE TABLE item1(id int);

--提交标识符为的trans_test的事务。
gaussdb=# COMMIT PREPARED 'trans_test';

--删除表。
gaussdb=# DROP TABLE item1;
```

相关链接

PREPARE TRANSACTION，ROLLBACK PREPARED。

7.14.51 COPY

功能描述

通过COPY命令实现在表和文件之间拷贝数据。

COPY FROM从一个文件拷贝数据到一个表，COPY TO把一个表的数据拷贝到一个文件。

注意事项

- 当参数enable_copy_server_files关闭时，只允许初始用户执行COPY FROM FILENAME或COPY TO FILENAME命令，当参数enable_copy_server_files打开时，允许具有SYSADMIN权限的用户或继承了内置角色gs_role_copy_files权限的用户执行，但默认禁止对数据库配置文件，密钥文件，证书文件和审计日志执行COPY FROM FILENAME或COPY TO FILENAME，以防止用户越权查看或修改敏感文件。同时enable_copy_server_files打开时，管理员可以通过guc参数safe_data_path设置普通用户可以导入导出的路径必须为设置路径的子路径，未设置此guc参数时候（默认情况），不对普通用户使用的路径进行拦截。该参数会对copy使用路径中的相对路径进行报错处理。
- COPY只能用于表，不能用于视图。
- COPY TO需要读取的表的select权限，COPY FROM需要插入的表的INSERT权限。
- 如果声明了一个字段列表，COPY将只在文件和表之间拷贝已声明字段的数据。如果表中有任何不在字段列表里的字段，COPY FROM将为那些字段插入缺省值。
- 如果声明了数据源文件，服务器必须可以访问该文件；如果指定了STDIN，数据将在客户前端和服务器之间流动，输入时，表的列与列之间使用TAB键分隔，在新的一行中以反斜杠和句点（\.）表示输入结束。
- 如果数据文件的任意行包含比预期多或者少的字段，COPY FROM将抛出一个错误。
- 数据的结束可以用一个只包含反斜杠和句点（\.）的行表示。如果从文件中读取数据，数据结束的标记是不必要的；如果在客户端应用之间拷贝数据，必须要有结束标记。
- COPY FROM中\N为空字符串，如果要输入实际数据值\N ，使用\\N。
- COPY FROM 支持通过列表达式对数据做预处理，但是列表达式中不支持子查询这类能力。
- COPY FROM在遇到数据格式错误时会回滚事务，但没有足够的错误信息，不方便用户从大量的原始数据中定位错误数据。
- COPY FROM/TO适合低并发，本地小数据量导入导出。
- 目标表存在trigger，支持COPY操作。
- COPY命令中，生成列不能出现在指定列的列表中。使用COPY… TO导出数据时，如果没有指定列的列表，则该表的所有列除了生成列都会被导出。COPY… FROM导入数据时，生成列会自动更新，并像普通列一样保存。
- COPY是服务端命令，执行环境和数据库服务端进程保持一致；\COPY是客户端元命令，执行环境和客户端gsql保持一致。需要注意的是，当在沙箱环境中使用数据库和gsql时，COPY命令和\COPY命令都使用沙箱内的路径；当在沙箱环境中使用数据库，在沙箱外使用gsql时，COPY命令使用沙箱内的路径，\COPY命令则使用沙箱外的路径。
- 在COPY TO导出的过程中，如果表内字段数据存在“\0”字符，则字段数据在导出时会发生截断，字段中只有'\0'之前的数据会被导出。
- COPY FROM导入过程中，不需要转码场景下，单行数据（包含tuple的元数据，以下均包含）小于1GB-1B；转码场景下单行数据小于256MB-1B，对以下转码场景进行了特殊处理：UTF-8 -> GB18030/GB18030_2022的限制为小于512MB-1B，UTF-8 -> GBK的限制为小于1GB-1B。当单行数据过大而

max_process_memory设置过小时会报错内存不足，需要调整max_process_memory的大小后进行重试。

语法格式

- 从一个文件拷贝数据到一个表。

```
COPY table_name [ ( column_name [, ...] ) ]
    FROM { 'filename' | STDIN }
    [ [ USING ] DELIMITERS 'delimiters' ]
    [ WITHOUT ESCAPING ]
    [ LOG ERRORS ]
    [ LOG ERRORS DATA ]
    [ REJECT LIMIT 'limit' ]
    [ [ WITH ] ( option [, ...] ) ]
    | copy_option
    | [ TRANSFORM ( { column_name [ data_type ] [ AS transform_expr ] } [, ...] ) ]
    | [ FIXED FORMATTER ( { column_name( offset, length ) } [, ...] ) [ ( option [, ...] ) | copy_option
[ ...] ] ];
```

说明

上述语法中fixed formatter与copy_option语法兼容、与option语法不兼容；copy_option与option语法不兼容；transfrom与copy_option、fixed formatter语法兼容。

- 把一个表的数据拷贝到一个文件。

```
COPY table_name [ ( column_name [, ...] ) ]
    TO { 'filename' | STDOUT }
    [ [ USING ] DELIMITERS 'delimiters' ]
    [ WITHOUT ESCAPING ]
    [ [ WITH ] ( option [, ...] ) ]
    | copy_option
    | [ FIXED FORMATTER ( { column_name( offset, length ) } [, ...] ) [ ( option [, ...] ) | copy_option
[ ...] ] ];

COPY query {(SELECT) | (VALUES)}
    TO { 'filename' | STDOUT }
    [ WITHOUT ESCAPING ]
    [ [ WITH ] ( option [, ...] ) ]
    | copy_option
    | [ FIXED FORMATTER ( { column_name( offset, length ) } [, ...] ) [ ( option [, ...] ) | copy_option
[ ...] ] ];
```

说明

1. COPY TO语法形式约束如下：

   (query)与[USING] DELIMITERS不兼容，即若COPY TO的数据来自于一个query的查询结果，那么COPY TO语法不能再指定[USING] DELIMITERS语法子句。

2. 对于FIXED FORMATTER语法后面跟随的copy_option是以空格进行分隔的。
3. copy_option是指COPY原生的参数形式，而option是兼容外表导入的参数形式。

其中可选参数option子句语法为：

```
FORMAT 'format_name'
| FORMAT binary
| DELIMITER 'delimiter_character'
| NULL 'null_string'
| HEADER [ boolean ]
| USEEOF [ boolean ]
| FILEHEADER 'header_file_string'
| FREEZE [ boolean ]
| QUOTE 'quote_character'
| ESCAPE 'escape_character'
| EOL 'newline_character'
| NOESCAPING [ boolean ]
| FORCE_QUOTE { ( column_name [, ...] ) | * }
| FORCE_NOT_NULL ( column_name [, ...] )
| ENCODING 'encoding_name'
```



```
| IGNORE_EXTRA_DATA [ boolean ]
| FILL_MISSING_FIELDS [ boolean ]
| COMPATIBLE_ILLEGAL_CHARS [ boolean ]
| DATE_FORMAT 'date_format_string'
| TIME_FORMAT 'time_format_string'
| TIMESTAMP_FORMAT 'timestamp_format_string'
| SMALLDATETIME_FORMAT 'smalldatetime_format_string'
```

其中可选参数copy_option子句语法为：

```
| NULL 'null_string'
| HEADER
| USEEOF
| FILEHEADER 'header_file_string'
| FREEZE
| FORCE_NOT_NULL column_name [, ...]
| FORCE_QUOTE { column_name [, ...] | * }
| BINARY
| CSV
| QUOTE [ AS ] 'quote_character'
| ESCAPE [ AS ] 'escape_character'
| EOL 'newline_character'
| ENCODING 'encoding_name'
| IGNORE_EXTRA_DATA
| FILL_MISSING_FIELDS [ { 'one' | 'multi' } ]
| COMPATIBLE_ILLEGAL_CHARS
| DATE_FORMAT 'date_format_string'
| TIME_FORMAT 'time_format_string'
| TIMESTAMP_FORMAT 'timestamp_format_string'
| SMALLDATETIME_FORMAT 'smalldatetime_format_string'
| SKIP int_number
| WHEN { ( start - end ) | column_name } { = | != } 'string'
| SEQUENCE ( { column_name ( integer [, incr] ) [, ...] } )
| FILLER ( { column_name [, ...] } )
| CONSTANT ( { column_name 'constant_string' [, ...] } )
```

参数说明

- **query**

  其结果将被拷贝。

  取值范围：仅支持一个SELECT或VALUES命令，命令结尾不需要分号。

- **table_name**

  表的名称（可以有模式修饰）。

  取值范围：已存在的表名。

- **column_name**

  可选的待拷贝字段列表。

  取值范围：如果没有声明字段列表，将使用所有字段。

- **STDIN**

  声明输入是来自标准输入。

- **STDOUT**

  声明输出打印到标准输出。

- **FIXED**

  打开字段固定长度模式。在字段固定长度模式下，不能声明DELIMITER，NULL，CSV选项。指定FIXED类型后，不能再通过option或copy_option指定BINARY、CSV、TEXT等类型。

说明

定长格式定义如下：

1. 每条记录的每个字段长度相同。
2. 长度不足的字段以空格填充，数字类型字段左对齐，字符字段右对齐。
3. 字段和字段之间没有分隔符。

- **[USING] DELIMITERS 'delimiters'**

  在文件中分隔各个字段的字符串，分隔符最大长度不超过10个字节。

  取值范围：文本模式不允许包含\.abcdefghijklmnopqrstuvwxyz0123456789中的任何一个字符，csv格式无此限制。

  缺省值：在文本模式下，缺省是水平制表符，在CSV模式下是一个逗号。

  说明

  出于历史原因，DELIMITER和DELIMITERS都可以指定分隔符，但是DELIMITERS后面可以直接跟括号语法，DELIMITER不可以直接跟括号，会语法报错。

- **WITHOUT ESCAPING**

  在TEXT格式中，不对'\'和后面的字符进行转义。

  取值范围：仅支持TEXT格式。

- **LOG ERRORS**

  若指定，则开启对于COPY FROM语句中数据类型错误的容错机制。

  取值范围：仅支持导入（即COPY FROM）时指定。

  说明

  此容错选项的使用限制如下：

  - 此容错机制仅捕捉COPY FROM过程中数据库主节点上数据解析过程中相关的数据类型错误（DATA_EXCEPTION）。
  - COPY已有的容错选项（如IGNORE_EXTRA_DATA）开启时，对应类型的错误会按照已有的方式处理而不会报出异常，因此错误表也不会有相应数据。

- **LOG ERRORS DATA**

  LOG ERRORS DATA和LOG ERRORS的区别:

  a. LOG ERRORS DATA会填充容错表的rawrecord字段。

  b. 只有super权限的用户才能使用LOG ERRORS DATA参数选项。

  须知

  - 使用“LOG ERRORS DATA”时，若错误内容过于复杂可能存在写入容错表失败的风险，导致任务失败。
  - 对于以某种编码无法读起来的错误，对应ERRCODE_CHARACTER_NOT_IN_REPERTOIRE和ERRCODE_UNTRANSLATABLE_CHARACTER两种错误码，不记录rawrecord字段。
  - 不支持binary格式的文件。

- **REJECT LIMIT 'limit'**

  与LOG ERROR选项共同使用，对COPY FROM的容错机制设置数值上限，一旦此COPY FROM语句错误数据超过选项指定条数，则会按照原有机制报错。

取值范围：正整数（1-INTMAX），'unlimited'（无最大值限制）

缺省值：若未指定LOG ERRORS，则会报错；若指定LOG ERRORS，则默认为0。

说明

如上述LOG ERRORS中描述的容错机制，REJECT LIMIT的计数也是按照执行COPY FROM的数据库主节点上遇到的解析错误数量计算，而不是数据库节点的错误数量。

- **FORMATTER**

  在固定长度模式中，定义每一个字段在数据文件中的位置。按照column(offset,length)格式定义每一列在数据文件中的位置。

  取值范围：

  - offset取值不能小于0，以字节为单位。
  - length取值不能小于0，以字节为单位。

  所有列的总长度和不能大于1GB。

  文件中没有出现的列默认以空值代替。

- **OPTION { option_name ' value ' }**

  用于指定兼容外表的各类参数。

  - FORMAT

    数据源文件的格式。

    取值范围：CSV、TEXT、FIXED、BINARY。

    - CSV格式的文件，可以有效处理数据列中的换行符，但对一些特殊字符处理有欠缺。
    - TEXT格式的文件，可以有效处理一些特殊字符，但无法正确处理数据列中的换行符。
    - FIXED格式的文件，适用于每条数据的数据列都比较固定的数据，长度不足的列会添加空格补齐，过长的列则会自动截断。
    - BINARY形式的选项会使得所有的数据被存储/读作二进制格式而不是文本。 这比TEXT和CSV格式的要快一些，但是一个BINARY格式文件可移植性比较差。

    缺省值：TEXT

  - DELIMITER

    指定数据文件行数据的字段分隔符。

    说明

    - 分隔符不能是\r和\n。
    - 分隔符不能和null参数相同，CSV格式数据的分隔符不能和quote参数相同。
    - TEXT格式数据的分隔符不能包含： 小写字母、数字和特殊字符.\。
    - 数据文件中单行数据长度需<1GB，如果分隔符较长且数据列较多的情况下，会影响导出有效数据的长度。
    - 分隔符推荐使用多字符和不可见字符。多字符例如'$^&'；不可见字符例如0x07，0x08，0x1b等。

    取值范围：支持多字符分隔符，但分隔符不能超过10个字节。

    缺省值：

- TEXT格式的默认分隔符是水平制表符（tab）。
- CSV格式的默认分隔符为“,”。
- FIXED格式没有分隔符。

– NULL

用来指定数据文件中空值的表示。

取值范围：

- null值不能是\r和\n，最大为100个字符。
- null值不能和分隔符、quote参数相同。

缺省值：

- CSV格式下默认值是一个没有引号的空字符串。
- 在TEXT格式下默认值是\N。

– HEADER

指定导出数据文件是否包含标题行，标题行一般用来描述表中每个字段的信息。header只能用于CSV，FIXED格式的文件中。

在导入数据时，如果header选项为on，则数据文件中第一行会被识别为标题行，会忽略此行。如果header为off，而数据文件中第一行会被识别为数据。

在导出数据时，如果header选项为on，则需要指定fileheader。如果header为off，则导出数据文件不包含标题行。

取值范围：true/on，false/off。

缺省值：false

– USEEOF

不对导入数据中的”\.”做报错处理。

取值范围：true/on，false/off。

缺省值：false

– QUOTE

CSV格式文件下的引号字符。

缺省值：双引号 '"'

说明

- quote参数不能和分隔符、null参数相同。
- quote参数只能是单字节的字符。
- 推荐不可见字符作为quote，例如0x07，0x08，0x1b等。

– ESCAPE

CSV格式下，用来指定逃逸字符，逃逸字符只能指定为单字节字符。

缺省值：双引号 '"'。当与quote值相同时，会被替换为'\0'。

– EOL 'newline_character'

指定导入导出数据文件换行符样式。

取值范围：支持多字符换行符，但换行符不能超过10个字节。常见的换行符，如\r、\n、\r\n（设成0x0D、0x0A、0x0D0A效果是相同的），其他字符或字符串，如$、#。

说明

- EOL参数只能用于TEXT格式的导入导出，不支持CSV格式和FIXED格式导入。为了兼容原有EOL参数，仍然支持导出CSV格式和FIXED格式时指定EOL参数为0x0D或0x0D0A。
- EOL参数不能和分隔符、null参数相同。
- EOL参数不能包含：.abcdefghijklmnopqrstuvwxyz0123456789。

– FORCE_QUOTE { (column_name [, ...]) | * }

在CSV COPY TO模式下，强制在每个声明的字段周围对所有非NULL值都使用引号包围。NULL输出不会被引号包围。

取值范围：已存在的字段。

– FORCE_NOT_NULL (column_name [, ...])

在CSV COPY FROM模式下，指定的字段输入不能为空。

取值范围：已存在的字段。

– ENCODING

指定数据文件的编码格式名称，缺省为当前客户端编码格式。

须知

COPY FROM时，ENCODING指定的字符集，应该和文件的编码格式保持一致，否则会报错或者导入数据乱码。

– IGNORE_EXTRA_DATA

若数据源文件比外表定义列数多，是否会忽略对多出的列。该参数只在数据导入过程中使用。

取值范围：true/on、false/off。

- 参数为true/on，若数据源文件比外表定义列数多，则忽略行尾多出来的列。
- 参数为false/off，若数据源文件比外表定义列数多，会显示如下错误信息。

```
extra data after last expected column
```

缺省值：false。

须知

如果行尾换行符丢失，使两行变成一行时，设置此参数为true将导致后一行数据被忽略掉。

– COMPATIBLE_ILLEGAL_CHARS

导入非法字符容错参数。此语法仅对COPY FROM导入有效。

取值范围：true/on，false/off。

- 参数为true/on，则导入时遇到非法字符进行容错处理，非法字符转换后入库，不报错，不中断导入。

- 参数为false/off，导入时遇到非法字符进行报错，中断导入。

缺省值：false/off

说明

导入非法字符容错规则如下：

（1）对于'\0'，容错后转换为空格；

（2）对于其他非法字符，容错后转换为问号；

（3）若compatible_illegal_chars为true/on标识导入时对于非法字符进行容错处理，则若NULL、DELIMITER、QUOTE、ESCAPE设置为空格或问号则会通过如"illegal chars conversion may confuse COPY escape 0x20"等报错信息提示用户修改可能引起混淆的参数以避免导入错误。

- FILL_MISSING_FIELDS

  当数据加载时，若数据源文件中一行的最后一个字段缺失的处理方式。

  取值范围：true/on，false/off。

  缺省值：false/off

- DATE_FORMAT

  导入对于DATE类型指定格式。此参数不支持BINARY格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对COPY FROM导入有效。

  取值范围：合法DATE格式。可参考**时间和日期处理函数和操作符**。

  说明

  对于DATE类型内建为TIMESTAMP类型的数据库，在导入的时候，若需指定格式，可以参考下面的timestamp_format参数。

- TIME_FORMAT

  导入对于TIME类型指定格式。此参数不支持BINARY格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对COPY FROM导入有效。

  取值范围：合法TIME格式，不支持时区。可参考**时间和日期处理函数和操作符**。

- TIMESTAMP_FORMAT

  导入对于TIMESTAMP类型指定格式。此参数不支持BINARY格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对COPY FROM导入有效。

  取值范围：合法TIMESTAMP格式，不支持时区。可参考**时间和日期处理函数和操作符**。

- SMALLDATETIME_FORMAT

  导入对于SMALLDATETIME类型指定格式。此参数不支持BINARY格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对COPY FROM导入有效。

  取值范围：合法SMALLDATETIME格式。可参考**时间和日期处理函数和操作符**。

- **COPY_OPTION { option_name ' value ' }**

  用于指定COPY原生的各类参数。

  - NULL null_string

    用来指定数据文件中空值的表示。

须知

在使用COPY FROM的时候，任何匹配这个字符串的字符串将被存储为NULL值，所以应该确保指定的字符串和COPY TO相同。

取值范围：

- null值不能是\r和\n，最大为100个字符。
- null值不能和分隔符、quote参数相同。

缺省值：

- 在TEXT格式下默认值是\N。
- CSV格式下默认值是一个没有引号的空字符串。

- HEADER

  指定导出数据文件是否包含标题行，标题行一般用来描述表中每个字段的信息。header只能用于CSV，FIXED格式的文件中。

  在导入数据时，如果header选项为on，则数据文件中第一行会被识别为标题行，会忽略此行。如果header为off，而数据文件中第一行会被识别为数据。

  在导出数据时，如果header选项为on，则需要指定fileheader。如果header为off，则导出数据文件不包含标题行。

- USEEOF

  不对导入数据中的”\.”做报错处理。

- FILEHEADER

  导出数据时用于定义标题行的文件，一般用来描述每一列的数据信息。

须知

- 仅在header为on或true的情况下有效。
- fileheader指定的是绝对路径。
- 该文件只能包含一行标题信息，并以换行符结尾，多余的行将被丢弃（标题信息不能包含换行符）。
- 该文件包括换行符在内长度不超过1M。

- FREEZE

  将COPY加载的数据行设置为已经被frozen，就像这些数据行执行过VACUUM FREEZE。

  这是一个初始数据加载的性能选项。仅当以下三个条件同时满足时，数据行会被frozen：

  - 在同一事务中create或truncate这张表之后执行COPY。
  - 当前事务中没有打开的游标。
  - 当前事务中没有原有的快照。

说明

COPY完成后，所有其他会话将会立刻看到这些数据。但是这违反了MVCC可见性的一般原则，用户应当了解这样会导致潜在的风险。

- FORCE NOT NULL column_name [, ...]

  在CSV COPY FROM模式下，指定的字段不为空。若输入为空，则将视为长度为0的字符串。

  取值范围：已存在的字段。

- FORCE QUOTE { column_name [, ...] | * }

  在CSV COPY TO模式下，强制在每个声明的字段周围对所有非NULL值都使用引号包围。NULL输出不会被引号包围。

  取值范围：已存在的字段。

- BINARY

  使用二进制格式存储和读取，而不是以文本的方式。在二进制模式下，不能声明DELIMITER，NULL，CSV选项。指定BINARY类型后，不能再通过option或copy_option指定CSV、FIXED、TEXT等类型。

- CSV

  打开逗号分隔变量（CSV）模式。指定CSV类型后，不能再通过option或copy_option指定BINARY、FIXED、TEXT等类型。

- QUOTE [AS] 'quote_character'

  CSV格式文件下的引号字符。

  缺省值：双引号 '"'。

  说明

  - quote参数不能和分隔符、null参数相同。
  - quote参数只能是单字节的字符。
  - 推荐不可见字符作为quote，例如0x07，0x08，0x1b等。

- ESCAPE [AS] 'escape_character'

  CSV格式下，用来指定逃逸字符，逃逸字符只能指定为单字节字符。

  默认值为双引号 '"'。当与quote值相同时，会被替换为'\0'。

- EOL 'newline_character'

  指定导入导出数据文件换行符样式。

  取值范围：支持多字符换行符，但换行符不能超过10个字节。常见的换行符，如\r、\n、\r\n（设成0x0D、0x0A、0x0D0A效果是相同的），其他字符或字符串，如$、#。

  说明

  - EOL参数只能用于TEXT格式的导入导出，不支持CSV格式和FIXED格式。为了兼容原有EOL参数，仍然支持导出CSV格式和FIXED格式时指定EOL参数为0x0D或0x0D0A。
  - EOL参数不能和分隔符、null参数相同。
  - EOL参数不能包含：.abcdefghijklmnopqrstuvwxyz0123456789。

- ENCODING 'encoding_name'

  指定文件编码格式名称。

  取值范围：有效的编码格式。

缺省值：当前编码格式。

须知

COPY FROM时，ENCODING指定的字符集，应该和文件的编码格式保持一致，否则会报错或者导入数据乱码。

- IGNORE_EXTRA_DATA

  指定当数据源文件比外表定义列数多时，忽略行尾多出来的列。该参数只在数据导入过程中使用。

  若不使用该参数，在数据源文件比外表定义列数多，会显示如下错误信息。

  ```
  extra data after last expected column
  ```

- COMPATIBLE_ILLEGAL_CHARS

  指定导入时对非法字符进行容错处理，非法字符转换后入库。不报错，不中断导入。此参数不支持BINARY格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对COPY FROM导入有效。

  若不使用该参数，导入时遇到非法字符进行报错，中断导入。

  说明

  导入非法字符容错规则如下：

  （1）对于'\0'，容错后转换为空格；

  （2）对于其他非法字符，容错后转换为问号；

  （3）若compatible_illegal_chars为true/on标识，导入时对于非法字符进行容错处理，则若NULL、DELIMITER、QUOTE、ESCAPE设置为空格或问号则会通过如"illegal chars conversion may confuse COPY escape 0x20"等报错信息提示用户修改可能引起混淆的参数以避免导入错误。

- FILL_MISSING_FIELDS [{ 'one' | 'multi' }]

  当数据加载时，若数据源文件中一行的最后部分字段缺失的处理方式。不指定one/multi或者指定one则最后一个字段缺失按默认方式处理，指定multi则最后多个字段缺失都按默认方式处理。

  取值范围：true/on，false/off。

  缺省值：false/off。

须知

目前COPY指定此Option实际不会生效，即不会有相应的容错处理效果（不生效）。需要额外注意的是，打开此选项会导致解析器在数据库主节点数据解析阶段（即COPY错误表容错的涵盖范围）忽略此数据问题，而到数据库节点重新报错，从而使得COPY错误表（打开LOG ERRORS REJECT LIMIT）在此选项打开的情况下无法成功捕获这类少列的数据异常。因此请不要指定此选项。

- DATE_FORMAT 'date_format_string'

  导入对于DATE类型指定格式。此参数不支持BINARY格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对COPY FROM导入有效。

  取值范围：合法DATE格式。可参考**时间和日期处理函数和操作符**

> **说明**
>
> 对于DATE类型内建为TIMESTAMP类型的数据库，在导入的时候，若需指定格式，可以参考下面的timestamp_format参数。

- TIME_FORMAT 'time_format_string'

  导入对于TIME类型指定格式。此参数不支持BINARY格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对COPY FROM导入有效。

  取值范围：合法TIME格式，不支持时区。可参考**时间和日期处理函数和操作符**。

- TIMESTAMP_FORMAT 'timestamp_format_string'

  导入对于TIMESTAMP类型指定格式。此参数不支持BINARY格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对COPY FROM导入有效。

  取值范围：合法TIMESTAMP格式，不支持时区。可参考**时间和日期处理函数和操作符**。

- SMALLDATETIME_FORMAT 'smalldatetime_format_string'

  导入对于SMALLDATETIME类型指定格式。此参数不支持BINARY格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对COPY FROM导入有效。

  取值范围：合法SMALLDATETIME格式。可参考**时间和日期处理函数和操作符**。

- TRANSFORM ({ column_name [data_type] [AS transform_expr] } [, ...])

  指定表中各个列的转换表达式；其中data_type指定该列在表达式参数中的数据类型；transform_expr为目标表达式，返回与表中目标列数据类型一致的结果值，表达式可参考**表达式**。

  - SKIP int_number

    指定数据导入时，跳过数据文件的前 int_number行。

  - WHEN { (start - end) | column_name } { = | != } 'string'

    数据导入时，检查导入的每一行数据，只有符合WHEN条件的数据行才导入表中。

  - SEQUENCE ({ column_name (integer [, incr]) [, ...] })

    数据导入时，SEQUENCE修饰的列，不从数据文件读取数据，通过指定的integer，按照incr递增数值；不指定incr则默认从1开始递增。

  - FILLER ({ column_name [, ...] })

    数据导入时，FILLER修饰的列，从数据文件读取数据后丢弃。

    > **说明**
    >
    > 使用FILLER需要指定待拷贝字段列表，数据处理时根据filler列在字段列表中的位置进行处理。

  - CONSTANT ({ column_name 'constant_string' [, ...] })

    数据导入时，CONSTANT修饰的列，不从数据文件读取数据，使用constant_string对该列进行赋值。

  COPY FROM能够识别的特殊反斜杠序列如下所示：

  - **\b**：反斜杠 （ ASCII 8 ）

- **\f**：换页（ASCII 12）
- **\n**：换行符 （ASCII 10）
- **\r**：回车符 （ASCII 13）
- **\t**：水平制表符 （ASCII 9）
- **\v**：垂直制表符 （ASCII 11）
- **\digits**：反斜杠后面跟着1到3个八进制数，表示ASCII值为该数的字符。
- **\xdigits**：反斜杠x后面跟着1或2个十六进制位声明指定数值编码的字符。

权限控制示例

```
gaussdbs=> copy t1 from '/home/xy/t1.csv';
ERROR:  COPY to or from a file is prohibited for security concerns
HINT:  Anyone can COPY to stdout or from stdin. gsql's \copy command also works for anyone.
gaussdb=> grant gs_role_copy_files to xxx;
```

此错误为非初始用户没有使用copy的权限示例，解决方式为打开enable_copy_server_files参数，则管理员可以使用copy功能，普通用户需要在此基础上加入gs_role_copy_files群组。

示例

```
--创建SCHEMA。
gaussdb=# CREATE SCHEMA tpcds;

--创建tpcds.ship_mode表。
gaussdb=# CREATE TABLE tpcds.ship_mode
(
    SM_SHIP_MODE_SK           INTEGER               NOT NULL,
    SM_SHIP_MODE_ID           CHAR(16)              NOT NULL,
    SM_TYPE                   CHAR(30)                      ,
    SM_CODE                   CHAR(10)                      ,
    SM_CARRIER                CHAR(20)                      ,
    SM_CONTRACT               CHAR(20)
)
;

--向tpcds.ship_mode表插入一条数据。
gaussdb=# INSERT INTO tpcds.ship_mode VALUES (1,'a','b','c','d','e');

--将tpcds.ship_mode中的数据复制到/home/omm/ds_ship_mode.dat文件中。
gaussdb=# COPY tpcds.ship_mode TO '/home/omm/ds_ship_mode.dat';

--将tpcds.ship_mode 输出到STDOUT。
gaussdb=# COPY tpcds.ship_mode TO STDOUT;

--将tpcds.ship_mode 的数据输出到STDOUT，使用参数如下：分隔符为','(delimiter ',')，编码格式为UTF8(encoding 'utf8')。
gaussdb=# COPY tpcds.ship_mode TO STDOUT WITH (delimiter ',', encoding 'utf8');

--将tpcds.ship_mode 的数据输出到STDOUT，使用参数如下：导入格式为CSV（format 'CSV'），引号包围SM_SHIP_MODE_SK字段的导出内容(force_quote(SM_SHIP_MODE_SK))。
gaussdb=# COPY tpcds.ship_mode TO STDOUT WITH (format 'CSV', force_quote(SM_SHIP_MODE_SK));

--创建tpcds.ship_mode_t1表。
gaussdb=# CREATE TABLE tpcds.ship_mode_t1
(
    SM_SHIP_MODE_SK           INTEGER               NOT NULL,
    SM_SHIP_MODE_ID           CHAR(16)              NOT NULL,
    SM_TYPE                   CHAR(30)                      ,
    SM_CODE                   CHAR(10)                      ,
    SM_CARRIER                CHAR(20)                      ,
    SM_CONTRACT               CHAR(20)
);
```



```
--从STDIN复制数据到表tpcds.ship_mode_t1。
gaussdb=# COPY tpcds.ship_mode_t1 FROM STDIN;

--输入一行数据示例
gaussdb=# COPY tpcds.ship_mode_t1 FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1	a	a	a	a	a
>> \.
 备注：数据与数据之间输入tab

--从/home/omm/ds_ship_mode.dat文件复制数据到表tpcds.ship_mode_t1。
gaussdb=# COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat';

--从/home/omm/ds_ship_mode.dat文件复制数据到表tpcds.ship_mode_t1，应用TRANSFORM表达式转换，取SM_TYPE列左边10个字符插入到表中。
gaussdb=# COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat' TRANSFORM (SM_TYPE AS LEFT(SM_TYPE, 10));

--从/home/omm/ds_ship_mode.dat文件复制数据到表tpcds.ship_mode_t1，使用参数如下：导入格式为TEXT（format 'text'），分隔符为'\t'（delimiter E'\t'），忽略多余列（ignore_extra_data 'true'），不指定转义（noescaping 'true'）。
gaussdb=# COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat' WITH(format 'text', delimiter E'\t', ignore_extra_data 'true', noescaping 'true');

--从/home/omm/ds_ship_mode_fixed.dat文件复制数据到表tpcds.ship_mode_t1，使用参数如下：导入格式为FIXED（FIXED），指定定长格式（FORMATTER(SM_SHIP_MODE_SK(0, 2), SM_SHIP_MODE_ID(2,16), SM_TYPE(18,30), SM_CODE(50,10), SM_CARRIER(61,20), SM_CONTRACT(82,20)）），忽略多余列（ignore_extra_data），有数据头（header）。
gaussdb=# COPY tpcds.ship_mode TO '/home/omm/ds_ship_mode_fixed.dat' FIXED FORMATTER(SM_SHIP_MODE_SK(0, 2), SM_SHIP_MODE_ID(2,16), SM_TYPE(18,30), SM_CODE(50,10), SM_CARRIER(61,20), SM_CONTRACT(82,20)) header;
gaussdb=# COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode_fixed.dat' FIXED FORMATTER(SM_SHIP_MODE_SK(0, 2), SM_SHIP_MODE_ID(2,16), SM_TYPE(18,30), SM_CODE(50,10), SM_CARRIER(61,20), SM_CONTRACT(82,20)) header ignore_extra_data;

--删除表和SCHEMA。
gaussdb=# DROP TABLE tpcds.ship_mode;
gaussdb=# DROP TABLE tpcds.ship_mode_t1;
gaussdb=# DROP SCHEMA tpcds;
```

7.14.52 CREATE AGGREGATE

功能描述

定义一个新的聚合函数。

语法格式

```
CREATE AGGREGATE name ( input_data_type [ , ... ] ) (
    SFUNC = sfunc,
    STYPE = state_data_type
    [ , FINALFUNC = ffunc ]
    [ , INITCOND = initial_condition ]
    [ , SORTOP = sort_operator ]
    [ , CFUNC = collection_func ]
)

or the old syntax

CREATE AGGREGATE name (
    BASETYPE = base_type,
    SFUNC = sfunc,
    STYPE = state_data_type
    [ , FINALFUNC = ffunc ]
    [ , INITCOND = initial_condition ]
```



```
    [ , SORTOP = sort_operator ]
    [ , CFUNC = collection_func ]
)
```

参数说明

- **name**

  要创建的聚合函数名(可以有模式修饰) 。

- **input_data_type**

  该聚合函数要处理的输入数据类型。要创建一个零参数聚合函数，可以使用*代替输入数据类型列表。（ count(*)就是这种聚合函数的一个实例。 ）

- **base_type**

  在以前的CREATE AGGREGATE语法中，输入数据类型是通过basetype参数指定的，而不是写在聚合的名称之后。 需要注意的是这种以前语法仅允许一个输入参数。 要创建一个零参数聚合函数，可以将basetype指定为"ANY"(而不是*)。

- **sfunc**

  将在每一个输入行上调用的状态转换函数的名称。 对于有N个参数的聚合函数，sfunc必须有 +1 个参数，其中的第一个参数类型为state_data_type，其余的匹配已声明的输入数据类型。 函数必须返回一个state_data_type类型的值。 这个函数接受当前状态值和当前输入数据，并返回下个状态值。

- **state_data_type**

  聚合的状态值的数据类型。

- **ffunc**

  在转换完所有输入行后调用的最终处理函数，它计算聚合的结果。 此函数必须接受一个类型为state_data_type的参数。 聚合的输出数据 类型被定义为此函数的返回类型。 如果没有声明ffunc则使用聚合结果的状态值作为聚合的结果，且输出类型为state_data_type。

- **initial_condition**

  状态值的初始设置(值)。 它必须是一个state_data_type类型可以接受的文本常量值。 如果没有声明，状态值初始为 NULL 。

- **sort_operator**

  用于MIN或MAX类型聚合的排序操作符。 这个只是一个操作符名 (可以有模式修饰)。这个操作符假设接受和聚合一样的输入数据类型。

- **collection_func**

  目前该参数在集中式下不生效。

示例

```
--创建自定义函数。
gaussdb=# CREATE OR REPLACE FUNCTION int_add(int,int)
 returns int as $BODY$
declare
begin
 return $1 + $2;
end;
$BODY$ language plpgsql;

--创建聚集函数。
gaussdb=# CREATE AGGREGATE sum_add(int)
(
    sfunc = int_add,
```



```
    stype = int,

    initcond = '0'

);

--创建测试表和添加数据。
gaussdb=# CREATE TABLE test_sum(a int,b int,c int);
gaussdb=# INSERT INTO test_sum VALUES(1,2),(2,3),(3,4),(4,5);

--执行聚集函数。
gaussdb=# SELECT sum_add(a) FROM test_sum;
  sum_add
-----------
  10

--删除聚集函数。
gaussdb=# DROP AGGREGATE sum_add(int);

--删除自定义函数。
gaussdb=# DROP FUNCTION int_add(int,int);

--删除测试表。
gaussdb=# DROP TABLE test_sum;
```

7.14.53 CREATE AUDIT POLICY

功能描述

创建统一审计策略。

注意事项

只有poladmin，sysadmin或初始用户能进行此操作。

需要开启安全策略开关，即设置GUC参数enable_security_policy=on，脱敏策略才可以生效。

须知

在使用DATABASE LINK功能的场景下，客户端发起的DATABASE LINK请求，实际的发送方是服务端，发送端ip地址等相关的属性将是服务端的值。详情见**DATABASE LINK**。

语法格式

```
CREATE AUDIT POLICY [ IF NOT EXISTS ] policy_name { privilege_audit_clause | access_audit_clause } [, ... ]
[ filter_group_clause ] [ ENABLE | DISABLE ];
```

- privilege_audit_clause：
  ```
  PRIVILEGES { DDL | ALL } [ ON LABEL ( resource_label_name [, ... ] ) ]
  ```
- access_audit_clause：
  ```
  ACCESS { DML | ALL } [ ON LABEL ( resource_label_name [, ... ] ) ]
  ```
- filter_group_clause：
  ```
  FILTER ON { filter_type ( filter_value [, ... ] ) } [, ... ]
  ```

参数说明

- **policy_name**

  审计策略名称，需要唯一，不可重复；

  取值范围：字符串，要符合**标识符命名规范**。

- **resource_label_name**

  资源标签名称。

- **DDL**

  指的是针对数据库执行如下操作时进行审计，目前支持：CREATE、ALTER、DROP、ANALYZE、COMMENT、GRANT、REVOKE、SET、SHOW。

  取值为ANALYZE时，ANALYZE和VACUUM操作都会被审计。

- **DML**

  指的是针对数据库执行如下操作时进行审计，目前支持：SELECT、COPY、DEALLOCATE、DELETE、EXECUTE、INSERT、PREPARE、REINDEX、TRUNCATE、UPDATE。

- **ALL**

  指的是上述DDL或DML中支持的所有对数据库的操作。当形式为{ DDL | ALL }时，ALL指所有DDL操作；当形式为{ DML | ALL }时，ALL指所有DML操作。

- **filter_type**

  描述策略过滤的条件类型，包括APP、ROLES、IP。

- **filter_value**

  指具体过滤信息内容。

- **ENABLE|DISABLE**

  可以打开或关闭统一审计策略。若不指定ENABLE|DISABLE，语句默认为ENABLE。

示例

```
--创建dev_audit和bob_audit用户。
gaussdb=# CREATE USER dev_audit PASSWORD '********';
gaussdb=# CREATE USER bob_audit PASSWORD '********';

--创建一个表tb_for_audit。
gaussdb=# CREATE TABLE tb_for_audit(col1 text, col2 text, col3 text);

--创建资源标签。
gaussdb=# CREATE RESOURCE LABEL adt_lb0 ADD TABLE(tb_for_audit);

--对数据库执行create操作创建审计策略。
gaussdb=# CREATE AUDIT POLICY adt1 PRIVILEGES CREATE;

--对数据库执行select操作创建审计策略。
gaussdb=# CREATE AUDIT POLICY adt2 ACCESS SELECT;

--仅审计记录用户dev_audit和bob_audit在执行针对adt_lb0资源进行的create操作数据库创建审计策略。
gaussdb=# CREATE AUDIT POLICY adt3 PRIVILEGES CREATE ON LABEL(adt_lb0) FILTER ON
ROLES(dev_audit, bob_audit);

--仅审计记录用户dev_audit和bob_audit,客户端工具为gsql，IP地址为'10.20.30.40', '127.0.0.0/24'，在执行针对adt_lb0资源进行的select、insert、delete操作数据库创建审计策略。
gaussdb=# CREATE AUDIT POLICY adt4 ACCESS SELECT ON LABEL(adt_lb0), INSERT ON LABEL(adt_lb0),
DELETE FILTER ON ROLES(dev_audit, bob_audit), APP(gsql), IP('10.20.30.40', '127.0.0.0/24');

--删除审计策略。
```



```
gaussdb=# DROP AUDIT POLICY adt1, adt2, adt3, adt4;

--删除资源标签。
gaussdb=# DROP RESOURCE LABEL adt_lb0;

--删除表tb_for_audit。
gaussdb=# DROP TABLE tb_for_audit;

--删除dev_audit和bob_audit用户。
gaussdb=# DROP USER dev_audit, bob_audit;
```

相关链接

ALTER AUDIT POLICY，DROP AUDIT POLICY。

7.14.54 CREATE CAST

功能描述

定义一个用户自定义的转换。

语法格式

- 定义通过函数转换的CAST：

```
CREATE CAST (source_type AS target_type)
    WITH FUNCTION function_name (argument_type [, ...])
    [ AS ASSIGNMENT | AS IMPLICIT ];
```

- 定义不通过函数转换的CAST:

```
CREATE CAST (source_type AS target_type)
    WITHOUT FUNCTION
    [ AS ASSIGNMENT | AS IMPLICIT ];
```

- 定义通过I/O转换的CAST：

```
CREATE CAST (source_type AS target_type)
    WITH INOUT
    [ AS ASSIGNMENT | AS IMPLICIT ];
```

参数说明

- **source_type**

  转换的源数据类型。

- **target_type**

  转换的目标数据类型。

- **function_name(argument_type [, ...])**

  用于执行转换的函数。 这个函数名可以是用模式名修饰的。 如果它没有用模式名修饰， 那么该函数将从模式搜索路径中找出来。 函数的结果数据类型必须匹配转换的目标类型。 它的参数在下面讨论。

- **WITHOUT FUNCTION**

  表明源类型是对目标类型是二进制可强制转换的，所以没有函数需要执行此转换。

- **WITH INOUT**

  表明转换是I/O转换，通过调用源数据类型的输出函数来执行，并将结果传给目标数据类型的输入函数。

- **AS ASSIGNMENT**

表示转换可以在赋值模式下隐含调用。

- **AS IMPLICIT**

  表示转换可以在任何环境里隐含调用。

  转换实现函数可以有一到三个参数。第一个参数的类型必须与转换的源类型相同的，或可以从转换的源类型二进制可强制转换的。第二个参数，如果存在，必须是integer类型；它接收这些与目标类型相关联的类型修饰符，或者若什么都没有则是-1。第三个参数，如果存在，必须是boolean类型；若转换是一个显式类型转换则会收到true，否则是false。

  一个转换函数的返回类型必须是与转换的目标类型相同或者对转换的目标类型二进制可强制转换。

  通常，一个转换必须有不同的源和目标数据类型。然而，若有多于一个参数的转换实现函数，则允许声明一个有相同的源和目标类型的转换。这用于表示系统目录中的特定类型的长度强制函数。命名的函数用于强制一个该类型的值为第二个参数给出的类型修饰符值。

  如果一个类型转换的源类型和目标类型不同，并且接收多于一个参数，它就表示从一种类型转换成另外一种类型只用一个步骤，并且同时实施长度转换。如果没有这样的项可用，那么转换成一个使用了类型修饰词的类型将涉及两个步骤，一个是在数据类型之间转换，另外一个是施加修饰词指定的转换。

  对域类型的转换目前没有作用。转换一般是针对域相关的所属数据类型。

  说明

  cast转换是以调用它的用户的权限来执行，高权限用户在调用其他用户创建的转换时，需要检查转换函数的执行内容，以免转换的创建者借用执行者的权限执行了越权的操作。

示例

为了从类型bigint到类型int4创建一个指派映射要通过使用函数int4(bigint):

```
gaussdb=# CREATE CAST (bigint AS int4) WITH FUNCTION int4(bigint) AS ASSIGNMENT;
```

（这个转换在系统中已经预先定义了。）

兼容性

CREATE CAST指令符合SQL标准，除了SQL没有为二进制可强制转换类型或者实现函数的额外参数来实现功能。

相关链接

DROP CAST

7.14.55 CREATE CLIENT MASTER KEY

功能描述

密态等值查询特性使用多级加密模型，主密钥加密列密钥，列密钥加密数据。本语法用于创建主密钥对象。

注意事项

- 本语法属于全密态数据库特有语法。

- 连接数据库时，在数据库驱动侧，需开启密态等值查询特性连接参数，才可执行本语法。
- 主密钥由外部密钥管理者提供，本语法仅处理密钥来源、密钥ID等信息，已支持的外部密钥管理者包括：

  a. 华为云密钥管理服务huawei_kms。
- 在使用本语法前，请参考《特性指南》中“设置密态等值查询”章节，在数据库驱动侧，为外部密钥管理者设置环境变量。

语法格式

```
CREATE CLIENT MASTER KEY client_master_key_name WITH （ KEY_STORE = key_store_name, KEY_PATH = "key_path_value", ALGORITHM = algorithm_type ）
```

参数说明

- **client_master_key_name**

  密钥对象名。在同一命名空间下，需满足命名唯一性约束。

  取值范围：字符串，需符合**标识符命名规范**。
- **KEY_STORE**

  外部密钥管理者。取值见**表7-143**。
- **KEY_PATH**

  由外部密钥管理者管理某个的密钥，不同密钥管理者格式不同。取值为字符串，详见**表7-143**。字符串由单引号或双引号包含，如果字符串长度超过64，则只能使用单引号包含。
- **ALGORITHM**

  密钥用于何种加密算法。取值见**表7-143**。

表 7-143 针对不同密钥管理者的参数值

KEY_STORE	KEY_PATH	ALGORITHM
huawei_kms	格式：‘{KmsApiUrl}/{密钥ID}' 参考：'https://kms.{项目}.myhuaweicloud.com/v1.0/{项目ID}/kms/{密钥ID}' 示例：'https://kms.cn-north-4.myhuaweicloud.com/v1.0/00000000000000000000000000000000/kms/00000000-0000-0000-0000-000000000000'	AES_256

相关链接

DROP CLIENT MASTER KEY

7.14.56 CREATE COLUMN ENCRYPTION KEY

功能描述

创建一个列加密密钥，该密钥可用于加密表中的指定列。

注意事项

本语法属于全密态数据库特有语法。

当使用gsql连接数据库服务器时，需使用‘-C’参数，打开全密态数据库的开关，才能使用本语法。

由该语法创建CEK对象可用于列级加密。在定义表中列字段时，可指定一个CEK对象，用于加密该列。

语法格式

```
CREATE COLUMN ENCRYPTION KEY column_encryption_key_name WITH VALUES(CLIENT_MASTER_KEY = client_master_key_name, ALGORITHM = algorithm_type, ENCRYPTED_VALUE = encrypted_value);
```

参数说明

- **column_encryption_key_name**

  密钥对象名。在同一命名空间下，需满足命名唯一性约束。

  取值范围：字符串，要符合**标识符命名规范**。

- **CLIENT_MASTER_KEY**

  客户端主密钥，用于加密本CEK的CMK，取值为：CMK对象名，该CMK对象由CREATE CLIENT MASTER KEY语法创建。

- **ALGORITHM**

  指定该CEK将用于何种加密算法，取值范围为：AEAD_AES_256_CBC_HMAC_SHA256、AEAD_AES_128_CBC_HMAC_SHA256、AEAD_AES_256_CTR_HMAC_SHA256、AES_256_GCM和SM4_SM3。

  其中不同加密算法的数据膨胀率AEAD_AES_256_CTR_HMAC_SHA256 < AES_256_GCM < AEAD_AES_256_CBC_HMAC_SHA256 = AEAD_AES_128_CBC_HMAC_SHA256 = SM4_SM3，推荐使用AEAD_AES_256_CTR_HMAC_SHA256 和 AES_256_GCM加密算法。

- **ENCRYPTED_VALUE（可选项）**

  该值为用户指定的密钥口令，密钥口令长度范围为28 ~ 256个字符，28个字符派生出来的密钥安全强度满足AES128，若用户需要用AES256，密钥口令的长度需要39个字符，如果不指定，则会自动生成256字符的密钥。

须知

- 国密算法约束：由于SM2、SM3、SM4等算法属于中国国家密码标准算法，为规避法律风险，需配套使用。如果创建CMK时指定SM4算法来加密CEK，则创建CEK时必须指定SM4_SM3算法来加密数据。
- ENCRYPTED_VALUE字段约束：如果使用由Huawei KMS生成的CMK来对CEK进行加密，在CREATE COLUMN ENCRYPTION KEY的语法中，如果使用ENCRYPTED_VALUE字段传入密钥，则传入的密钥的长度应为16字节的整数倍。

示例（在使用 gsql 连接数据库服务器的场景下）

在使用本语法前，需开通密钥服务并配置访问密钥服务的参数，详细示例请参考《特性指南》中”设置密态等值查询”章节。

相关链接

ALTER COLUMN ENCRYPTION KEY，**DROP COLUMN ENCRYPTION KEY**

7.14.57 CREATE CONVERSION

功能描述

定义一种两个字符集编码之间的新转换。该功能为内部使用功能，不建议用户使用。

注意事项

- 参数DEFAULT将在客户端和服务器之间默认执行源编码到目标编码之间的转换。要支持这个用法，需要定义双向转换，即从A到B和从B到A之间的转换。
- 创建转换需拥有函数的EXECUTE权限及目标模式的CREATE权限。
- 源编码和目标编码都不可以使用SQL_ASCII，因为在涉及SQL_ASCII “encoding”的情况下，服务器的行为是硬连接的。
- 使用DROP CONVERSION可以移除用户定义的转换。

语法格式

```
CREATE [ DEFAULT ] CONVERSION name
    FOR 'source_encoding' TO 'dest_encoding' FROM function_name
```

参数说明

- **DEFAULT**

  DEFAULT子句表示这个转换是从源编码到目标编码的默认转换。在一个模式中对于每一个编码对，只应该有一个默认转换。

- **name**

  转换的名称，可以被模式限定。如果没有被模式限定，该转换被定义在当前模式中。在一个模式中，转换名称必须唯一。

- **source_encoding**

  源编码名称。

- **dest_encoding**

  目标编码名称。

- **function_name**

  被用来执行转换的函数。函数名可以被模式限定。如果没有，将在路径中查找该函数。

  该函数必须具有以下格式：

```
conv_proc(
    integer,  -- 原编码ID
    integer,  -- 目标编码ID
    cstring,  -- 源字符串（空值终止的C字符串）
    internal, -- 目标（用一个空值终止的C字符串填充）
    integer,  -- 源字符串长度
) RETURNS void;
```

注意

目前仅支持系统内部创建，用户无法创建。

7.14.58 CREATE DATABASE

功能描述

创建一个新的数据库。缺省情况下新数据库将通过复制标准系统数据库template0来创建，且仅支持使用template0来创建。

注意事项

- 只有拥有CREATEDB权限的用户才可以创建新数据库，系统管理员默认拥有此权限。
- 不能在事务块中执行创建数据库语句。
- 在创建数据库过程中，出现类似“Permission denied”的错误提示，可能是由于文件系统上数据目录的权限不足。出现类似“No space left on device”的错误提示，可能是由于磁盘满引起的。

语法格式

```
CREATE DATABASE database_name
   [ [ WITH ] { [ OWNER [=] user_name ] |
            [ TEMPLATE [=] template ] |
            [ ENCODING [=] 'encoding' ] |
            [ LC_COLLATE [=] 'lc_collate' ] |
            [ LC_CTYPE [=] 'lc_ctype' ] |
            [ DBCOMPATIBILITY [=] 'compatibility_type' ] |
            [ TABLESPACE [=] tablespace_name ] |
            [ CONNECTION LIMIT [=] connlimit ] |
            [ DBTIMEZONE [=] 'time_zone']}[...] ];
```

参数说明

- **database_name**

  数据库名称。

  取值范围：字符串，要符合**标识符命名规范**。

- **OWNER [=] user_name**

  数据库所有者。缺省时，新数据库的所有者是当前用户。

  取值范围：已存在的用户名。

- **TEMPLATE [=] template**

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

  取值范围：仅template0。

- **ENCODING [=] 'encoding'**

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

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

  常用取值：GBK、UTF8、Latin1、GB18030等，具体支持的字符集如下。

  表 7-144 GaussDB 字符集

名称	描述	语言	是否服务器端?	ICU（International Components for Unicode）?	字节/字符	别名
BIG5	Big Five	繁体中文	否	否	1-2	WIN950, Windows950
EUC_CN	扩展UNIX编码-中国	简体中文	是	是	1-3	-
EUC_JP	扩展UNIX编码-日本	日文	是	是	1-3	-
EUC_JIS_2004	扩展UNIX编码-日本, JIS X 0213	日文	是	否	1-3	-
EUC_KR	扩展UNIX编码-韩国	韩文	是	是	1-3	-
EUC_TW	扩展UNIX编码-中国台湾	繁体中文	是	是	1-3	-

名称	描述	语言	是否服务器端?	ICU（International Components for Unicode）?	字节/字符	别名
GB18030	国家标准	中文	是	否	1-4	-
GB18030_2022	国家标准	中文	是	否	1-4	-
GBK	扩展国家标准	简体中文	是	否	1-2	WIN936, Windows936
ISO_8859_5	ISO 8859-5, ECMA 113	拉丁语/西里尔语	是	是	1	-
ISO_8859_6	ISO 8859-6, ECMA 114	拉丁语/阿拉伯语	是	是	1	-
ISO_8859_7	ISO 8859-7, ECMA 118	拉丁语/希腊语	是	是	1	-
ISO_8859_8	ISO 8859-8, ECMA 121	拉丁语/希伯来语	是	是	1	-
JOHAB	JOHAB	韩语	否	否	1-3	-
KOI8R	KOI8-R	西里尔语（俄语）	是	是	1	KOI8
KOI8U	KOI8-U	西里尔语（乌克兰语）	是	是	1	-
LATIN1	ISO 8859-1, ECMA 94	西欧	是	是	1	ISO88591
LATIN2	ISO 8859-2, ECMA 94	中欧	是	是	1	ISO88592

名称	描述	语言	是否服务器端?	ICU（International Components for Unicode）?	字节/字符	别名
LATIN3	ISO 8859-3, ECMA 94	南欧	是	是	1	ISO88593
LATIN4	ISO 8859-4, ECMA 94	北欧	是	是	1	ISO88594
LATIN5	ISO 8859-9, ECMA 128	土耳其语	是	是	1	ISO88599
LATIN6	ISO 8859-10, ECMA 144	日耳曼语	是	是	1	ISO885910
LATIN7	ISO 8859-13	波罗的海	是	是	1	ISO885913
LATIN8	ISO 8859-14	凯尔特语	是	是	1	ISO885914
LATIN9	ISO 8859-15	带欧罗巴和口音的LATIN1	是	是	1	ISO885915
LATIN10	ISO 8859-16, ASRO SR 14111	罗马尼亚语	是	否	1	ISO885916
MULE_INTERNAL	Mule内部编码	多语种编辑器	是	否	1-4	-
SJIS	Shift JIS	日语	否	否	1-2	Mskanji, ShiftJIS, WIN932, Windows932

名称	描述	语言	是否服务器端?	ICU（International Components for Unicode）?	字节/字符	别名
SHIFT_JIS_2004	Shift JIS, JIS X 0213	日语	否	否	1-2	-
SQL_ASCII	未指定（见文本）	*任意*	是	否	1	-
UHC	统一韩语编码	韩语	否	否	1-2	WIN949, Windows949
UTF8	Unicode, 8-bit	*所有*	是	是	1-4	Unicode
WIN866	Windows CP866	西里尔语	是	是	1	ALT
WIN874	Windows CP874	泰语	是	否	1	-
WIN1250	Windows CP1250	中欧	是	是	1	-
WIN1251	Windows CP1251	西里尔语	是	是	1	WIN
WIN1252	Windows CP1252	西欧	是	是	1	-
WIN1253	Windows CP1253	希腊语	是	是	1	-
WIN1254	Windows CP1254	土耳其语	是	是	1	-
WIN1255	Windows CP1255	希伯来语	是	是	1	-
WIN1256	Windows CP1256	阿拉伯语	是	是	1	-

名称	描述	语言	是否服务器端?	ICU（International Components for Unicode）?	字节/字符	别名
WIN1257	Windows CP1257	波罗的海	是	是	1	-
WIN1258	Windows CP1258	越南语	是	是	1	ABC, TCVN, TCVN5712, VSCII

⚠ 注意

- 需要注意并非所有的客户端API都支持上面列出的字符集。
- SQL_ASCII设置与其他设置表现存在差异。如果服务器字符集是SQL_ASCII，服务器把字节值0-127根据ASCII标准解释，而字节值128-255则当作无法解析的字符。如果设置为SQL_ASCII，就不会有编码转换。因此，这个设置基本不是用来声明所使用的指定编码，因为这个声明会忽略编码。在大多数情况下，如果使用了任何非ASCII数据，那么请不要使用SQL_ASCII进行设置，因为数据库将无法转换或者校验非ASCII字符。

须知

- 指定新的数据库字符集编码必须与所选择的本地环境中（LC_COLLATE和LC_CTYPE）的设置兼容。
- 当指定的字符编码集为GBK时，部分中文生僻字无法直接作为对象名。这是因为GBK第二个字节的编码范围在0x40-0x7E之间时，字节编码与ASCII字符@A-Z[\]^_`a-z{|}重叠。其中@[\]^_'{|}是数据库中的操作符，直接作为对象名时，会语法报错。例如“侅”字，GBK16进制编码为0x8240，第二个字节为0x40，与ASCII“@”符号编码相同，因此无法直接作为对象名使用。如果确实要使用，可以在创建和访问对象时，通过增加双引号来规避这个问题。
- 若客户端编码为A，服务器端编码为B，则需要满足数据库中存在编码格式A与B的转换。数据库能够支持的所有的编码格式转换详见系统表**PG_CONVERSION**（若无法转换，则建议客户端编码与服务器端编码保持一致，客户端编码可通过GUC参数**client_encoding**修改）。
- 若要指定数据库字符集编码为GB18030_2022，且客户端编码也要设置为GB18030时，必须确保客户端操作系统支持的GB18030字符集为2022版本，否则由于GB18030字符集自身的各版本间存在不完全兼容，可能导致数据的不一致性。同时，涉及到历史数据切换为GB18030_2022数据库时应当遵循切库流程，进行数据迁移操作。

- **LC_COLLATE [=] 'lc_collate'**

  指定新数据库使用的字符集。例如，通过lc_collate = 'zh_CN.gbk'设定该参数。

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

  取值范围：操作系统支持的字符集。

- **LC_CTYPE [=] 'lc_ctype'**

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

  取值范围：操作系统支持的字符分类。

  说明

  - 对于lc_collate和lc_ctype参数的取值范围，取决于本地环境支持的字符集。例如：在Linux操作系统上，可通过locale -a命令获取操作系统支持的字符集列表，在应用lc_collate和lc_ctype参数时可从中选择用户需要的字符集和字符分类。
  - 当指定的字符编码集为GB18030_2022时，其LC_COLLATE和LC_CTYPE的取值范围与GB18030保持一致。

- **DBCOMPATIBILITY [=] 'compatibility_type'**

  指定兼容的数据库的类型，默认兼容O。

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

  说明

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

- **TABLESPACE [=] tablespace_name**

  指定数据库对应的表空间。

  取值范围：已存在表空间名。

- **CONNECTION LIMIT [=] connlimit**

  数据库可以接受的并发连接数。

  须知

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

  取值范围：[-1, 2^31-1]的整数。默认值为-1，表示没有限制。

有关字符编码的一些限制：

- 若区域设置为C（或POSIX），则允许所有的编码类型，但是对于其他的区域设置，字符编码必须和区域设置相同。

- 若字符编码方式是SQL_ASCII，并且修改者为管理员用户时，则字符编码可以和区域设置不相同。
- 编码和区域设置必须匹配模板数据库，除了将template0当作模板。 因为其他数据库可能会包含不匹配指定编码的数据，或者可能包含排序顺序受LC_COLLATE和LC_CTYPE影响的索引。复制这些数据会导致在新数据库中的索引失效。template0是不包含任何会受到影响的数据或者索引。
- **DBTIMEZONE [=] 'time_zone'**

  指定新数据库的时区。例如，通过DBTIMEZONE = '+00:00'设定该参数。该参数会影响新数据库的时区。默认设置为PRC时区。

  取值范围：操作系统支持的时区名称和缩写或者-15：59到+15：00时间戳范围。

示例

```
--创建jim和tom用户。
gaussdb=# CREATE USER jim PASSWORD '********';
gaussdb=# CREATE USER tom PASSWORD '********';

--创建一个GBK编码的数据库music（本地环境的编码格式必须也为GBK）。
gaussdb=# CREATE DATABASE music ENCODING 'GBK' template = template0;

--创建数据库music2，并指定所有者为jim。
gaussdb=# CREATE DATABASE music2 OWNER jim;

--用模板template0创建数据库music3，并指定所有者为jim。
gaussdb=# CREATE DATABASE music3 OWNER jim TEMPLATE template0;

--设置music数据库的连接数为10。
gaussdb=# ALTER DATABASE music CONNECTION LIMIT= 10;

--将music名称改为music4。
gaussdb=# ALTER DATABASE music RENAME TO music4;

--将数据库music2的所属者改为tom。
gaussdb=# ALTER DATABASE music2 OWNER TO tom;

--设置music3的表空间为PG_DEFAULT。
gaussdb=# ALTER DATABASE music3 SET TABLESPACE PG_DEFAULT;

--关闭在数据库music3上缺省的索引扫描。
gaussdb=# ALTER DATABASE music3 SET enable_indexscan TO off;

--重置enable_indexscan参数。
gaussdb=# ALTER DATABASE music3 RESET enable_indexscan;

--删除数据库。
gaussdb=# DROP DATABASE music2;
gaussdb=# DROP DATABASE music3;
gaussdb=# DROP DATABASE music4;

--删除jim和tom用户。
gaussdb=# DROP USER jim;
gaussdb=# DROP USER tom;

--创建兼容TD格式的数据库。
gaussdb=# CREATE DATABASE td_compatible_db DBCOMPATIBILITY 'C';

--创建兼容A格式的数据库。
gaussdb=# CREATE DATABASE ora_compatible_db DBCOMPATIBILITY 'A';

--删除兼容TD、A格式的数据库。
gaussdb=# DROP DATABASE td_compatible_db;
gaussdb=# DROP DATABASE ora_compatible_db;
```


相关链接

ALTER DATABASE，DROP DATABASE

优化建议

- **create database**

  事务中不支持创建database。

- **ENCODING LC_COLLATE LC_CTYPE**

  当新建数据库Encoding与模板数据库（SQL_ASCII）不匹配（为'GBK' /'UTF8'/'LATIN1'/'GB18030'/'GB18030_2022'）时，必须指定template [=] template0。

7.14.59 CREATE DATABASE LINK

功能描述

创建DATABASE LINK对象。DATABASE LINK详细说明请见DATABASE LINK。

注意事项

- 禁止使用DATABASE LINK连接初始用户。
- 禁止初始用户创建、修改和删除DATABASE LINK对象。
- 升级未提交情况下无法创建使用DATABASE LINK。
- 当使用CURRENT_USER或CONNECT TO连接串省略时，使用当前数据库初始用户名和空密码连接，使用时会连接失败。
- 连接GaussDB时使用localhost、127.0.0.1或本机ip及对应端口可连接当前实例数据库。

语法格式

```
CREATE [ PUBLIC ] DATABASE LINK dblink
  [ CONNECT TO { CURRENT_USER | user IDENTIFIED BY password } ] [ USING ( option 'value' [...]) ];
```

参数说明

- **PUBLIC**

  指定公共以创建对所有用户可见的公共数据库链接。如果省略此子句，则数据库链接是私有的，仅对当前用户可用。

- **dblink**

  要创建的DATABASE LINK的名字。

- **user**

  创建的DATABASE LINK连接远端使用的用户名。

- **password**

  创建的DATABASE LINK连接远端使用的用户对应的密码。

- **CURRENT_USER**

  使用当前数据库初始用户名和空密码连接。

- **USING (option 'value' [, ...])**

USING可选择指定要连接的数据库的ip地址、端口号、远端的database name等参数 ，支持的options包括：

- host：指定连接的地址，不支持ipv6地址。支持以 ‘,’分割的字符串来指定多个IP地址，当前不支持密态数据库和ssl设置和证书认证，不指定默认为空。
- port：指定连接的端口号，不指定默认为5432。
- dbname：指定连接的数据库名称，不指定默认为连接远端使用的用户名。
- fetch_size：从远端每次获取数据量大小，fetch_size取值为0到2147483647，默认为100。

须知

- USING后的括号可以只选择上述关键字中的一部分去写。
- USING关键字也可以不写，同时之后的括号也不要再写。
- DATABASE LINK创建的时候不会去验证是否能连接成功，如果缺乏相关的关键字，可能会在使用时报错。

示例

```
gaussdb=# create public database link test_link1 connect to 'user1' identified by '********' using (port '54399',dbname 'db01');
```

7.14.60 CREATE DIRECTORY

功能描述

使用CREATE DIRECTORY语句创建一个目录对象，该目录对象定义了服务器文件系统上目录的别名，用于存放用户使用的数据文件。

注意事项

- 当enable_access_server_directory=off时，只允许初始用户创建directory对象；当enable_access_server_directory=on时，具有SYSADMIN权限的用户和继承了内置角色gs_role_directory_create权限的用户可以创建directory对象。
- 创建用户默认拥有此路径的READ和WRITE操作权限。
- 目录的默认owner为创建directory的用户。
- 以下路径禁止创建：
  - 路径含特殊字符。
  - 路径是相对路径。
- 创建目录时会进行以下合法性校验：
  - 创建时会检查添加路径是否为操作系统实际存在路径，如不存在会提示用户使用风险。
  - 创建时会校验数据库初始化（omm）用户对于添加路径的权限(即操作系统目录权限，读/写/执行 - R/W/X)，如果权限不全，会提示用户使用风险。
- 在数据库环境下用户指定的路径需要用户保证各节点上路径的一致性，否则在不同节点上执行会产生找不到路径的问题。

语法格式

```
CREATE [OR REPLACE] DIRECTORY directory_name
AS 'path_name';
```

参数说明

- **directory_name**
  目录名称。
  取值范围：字符串，要符合**标识符命名规范**。
- **path_name**
  操作系统的路径。
  取值范围： 有效的操作系统路径。

示例

```
--创建目录。
gaussdb=# CREATE OR REPLACE DIRECTORY  dir  as '/tmp/';
```

相关链接

ALTER DIRECTORY，**DROP DIRECTORY**

7.14.61 CREATE EVENT

功能描述

创建一个新的定时任务。

注意事项

- 定时任务相关操作只有sql_compatibility = 'B'时支持。
- 用户操作（创建/修改/删除）定时任务时，非sysadmin用户需要被sysadmin用户赋予操作定时任务的权限。定时任务操作权限与高级包DBE_SCHEDULER中创建定时任务赋权操作一致。
- 定时任务时间间隔interval表达式目前兼容了浮点数语法，例如interval 0.5 minute，但是计算时会将浮点数取整，所以不建议interval时间间隔使用浮点数形式。
- 同一database下不支持同名定时任务。
- 定时任务中待执行语句范围是除安全相关操作以外任意SQL语句，但对于某些有约束的语句会执行失败。例如：不支持通过复合语句创建database。
- 定时任务待执行语句不支持的安全相关操作范围主要包括：
  - 使用加密函数。
  - 创建、设置用户、group。
  - 连接数据库。
  - 函数加密等。
- 定时任务指定definer选项在以下场景下会指定失败：
  - 操作定时任务的用户不具有sysadmin权限。
  - 当前用户与被指定definer不一致时：

- 指定definer为初始用户。
- 指定definer为运维管理员、监控管理员。
- 开启三权分立，enableSeparationOfDuty=on。

语法格式

```
CREATE
    [DEFINER = user]
EVENT
    [IF NOT EXISTS]
    event_name
    ON SCHEDULE schedule
    [ON COMPLETION [NOT] PRESERVE]
    [ENABLE | DISABLE | DISABLE ON SLAVE]
    [COMMENT 'string']
    DO event_body;
```

- schedule：

  ```
  {
      AT timestamp [+ INTERVAL interval] ...
    | EVERY interval
      [STARTS timestamp [+ INTERVAL interval] ...]
      [ENDS timestamp [+ INTERVAL interval] ...]
  }
  ```

- interval：

  ```
  quantity
          {YEAR | MONTH | DAY | HOUR | MINUTE | SECOND |
           YEAR TO MONTH | DAY TO HOUR | DAY TO MINUTE |
           DAY TO SECOND | HOUR TO MINUTE | HOUR TO SECOND |
           MINUTE TO SECOND}
  ```

参数说明

- **DEFINER**

  定时任务待执行语句在执行时使用的权限。默认情况下使用当前创建定时任务者的权限，当definer被指定时，使用被指定用户user的用户权限。

  definer参数只有具有sysadmin权限的用户有权指定。

- **ON SCHEDULE**

  定时任务执行时刻。定时任务可以通过schedule设置为执行一次，也可以设置为执行多次：

  - AT timestamp [+ INTERVAL interval] 表示设置定时任务只在timestamp [+ INTERVAL interval] 时间点执行一次。
  - EVERY interval 表示设置定时任务在每隔interval时间后重复执行。
    - STARTS timestamp [+ INTERVAL interval] 用户可以给可重复执行的定时任务指定起始时间，即定时任务从timestamp [+ INTERVAL interval] 时刻开始执行。当此参数为空时默认从当前时刻开始执行。
    - ENDS timestamp [+ INTERVAL interval] 用户可以给可重复执行的定时任务指定结束时间，即定时任务从timestamp [+ INTERVAL interval]时刻停止执行。当此参数为空时默认为3999-12-31 16:00:00。

- **INTERVAL**

  时间间隔，interval由quantity数字和时间单位组成，例如1 YEAR。

- **ON COMPLETION [NOT] PRESERVE**

默认情况下，一旦事务处于完成状态，系统表中就会立刻删除该定时任务。用户可以通过设置ON COMPLETION PRESERVE来覆盖默认行为。

- **ENABLE | DISABLE | DISABLE ON SLAVE**

  创建定时任务后，定时任务默认处于ENABLE状态，即到规定时间立即执行待执行语句。用户可以使用DISABLE关键字，改变定时任务的活动状态。DISABLE ON SLAVE表现与DISABLE一致。

- **COMMENT**

  用户可以给定时任务添加注释，注释内容在GS_JOB_ATTRIBUTE表中查看。

- **DO**

  定时任务待执行语句。

示例

```
--创建表。
gaussdb=#  CREATE TABLE t_ev(num int);

--创建一个执行一次的定时任务。
gaussdb=#  CREATE EVENT IF NOT EXISTS event_e1 ON SCHEDULE AT sysdate + interval 5 second + interval 33 minute DISABLE DO insert into t_ev values(0);

--创建一个每隔一分钟执行一次的定时任务。
gaussdb=#  CREATE EVENT IF NOT EXISTS event_e2 ON SCHEDULE EVERY 1 minute DO insert into t_ev values(1);

--查看定时任务。
gaussdb=# SHOW EVENTS;

--删除定时任务。
gaussdb=# DROP EVENT event_e1;
gaussdb=# DROP EVENT event_e2;
```

须知

- 定时任务创建完成后如果执行失败，失败原因可以通过SHOW EVENTS或在PG_JOB表中查看。
- 当定时任务的待执行语句中进行涉及用户密码相关操作时（创建弱口令等），系统表及中会记录密码的明文。因此不建议用户在定时任务的待执行语句中进行涉及用户密码的相关操作。

7.14.62 CREATE FOREIGN DATA WRAPPER

功能描述

创建一个新的外部数据封装器。创建外部数据封装器的用户成为其所有者。

注意事项

- 外部数据封装器的名字必须在数据库中唯一。
- 只有初始用户和系统管理员用户可以创建外部数据封装器。

语法格式

```
CREATE FOREIGN DATA WRAPPER name
[ HANDLER handler_function | NO HANDLER ]
```



```
[ VALIDATOR validator_function | NO VALIDATOR ]
[ OPTIONS ( option 'value' [, ... ] ) ];
```

参数说明

- **name**

  要创建的外部数据封装器的名字

- **HADNLER handler_function**

  handler_function是先前已经注册了的函数的名字，用来为外部表检索执行函数。处理器函数必须没有参数，并且它的返回类型必须为fdw_handler。

  不用处理器函数创建外部数据封装器是可能的，但是使用这种封装器的外部表只能被声明，不能被访问。

- **VALIDATOR validator_function**

  validator_function 是先前已经注册了的函数的名字用来检查提供给外部数据封装器的通用选项， 还有使用该外部数据封装器的外部服务器、用户映射和外部表的选项。 如果没有验证器函数或声明了NO VALIDATOR， 那么在创建时将不检查选项。（外部数据封装器可能在运行时忽略或拒绝无效的选项说明， 取决于实现。）验证器函数必须接受两个参数：一个类型为text[]， 将包含存储在系统目录中的选项的数组，一个类型为oid， 是包含这些选项的系统目录的OID。忽略返回类型；该函数应该使用 ereport(ERROR)函数报告无效选项。

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

  这个子句为新的外部数据封装器声明选项。允许的选项名和值是特定于每个外部数据封装器的， 并且是经过外部数据封装器的验证器函数验证了的。选项名必须是唯一的。

示例

```
--创建一个无用的外部数据封装器dummy。
gaussdb=# CREATE FOREIGN DATA WRAPPER dummy;

--创建一个带有处理器函数file_fdw_handler 的外部数据封装器file。
gaussdb=# CREATE FOREIGN DATA WRAPPER file HANDLER file_fdw_handler;

--创建一个带有一些选项的外部数据封装器mywrapper。
gaussdb=# CREATE FOREIGN DATA WRAPPER mywrapper OPTIONS (debug 'true');
```

相关链接

ALTER FOREIGN DATA WRAPPER，**DROP FOREIGN DATA WRAPPER**

7.14.63 CREATE FUNCTION

功能描述

创建一个函数。

注意事项

- 如果创建函数时参数或返回值带有精度，不进行精度检测。
- 创建函数时，函数定义中对表对象的操作建议都显式指定模式，否则可能会导致函数执行异常。
- 创建存储过程时，仅对CREATE的存储过程或PACKAGE本身加写锁，仅对执行过程中编译、执行会对函数和函数依赖的PACKAGE均加读锁。

- 在创建函数时，函数内部通过SET语句设置current_schema和search_path无效。执行完函数后的search_path和current_schema与执行函数前的search_path和current_schema保持一致。
- 如果函数参数中带有出参，想要出参生效，必须打开guc参数 set behavior_compat_options = 'proc_outparam_override'； SELECT、CALL调用函数时，必须要在出参位置提供实参进行调用，否则函数调用失败。
- 兼容PostgreSQL风格的函数或者带有PACKAGE属性的函数支持重载。在指定REPLACE的时候，如果参数个数、类型、返回值有变化，不会替换原有函数，而是会建立新的函数。
- 不能创建仅形参名字不同（函数名和参数列表类型都一样）的重载函数。
- 不能创建与存储过程拥有相同名称和参数列表的函数。
- 不支持形参仅在自定义ref cursor类型和sys_refcursor类型不同的重载。
- 不支持仅返回的数据类型不同的函数重载。
- 不支持仅默认值不同的函数重载。
- 重载的函数在调用时变量需要明确具体的类型。
- 在函数内部使用未声明的变量，函数被调用时会报错。
- SELECT调用可以指定不同参数来进行同名函数调用。由于语法不支持调用不带有PACKAGE属性的同名函数。
- 在创建function时，不能在avg函数外面嵌套其他agg函数，或者其他系统函数。
- 新创建的函数默认会给PUBLIC授予执行权限（详见**GRANT**）。用户默认继承PUBLIC角色权限，因此其他用户也会有函数的执行权限并可以查看函数的定义，另外执行函数时还需要具备函数所在schema的USAGE权限。用户在创建函数时可以选择收回PUBLIC默认执行权限，然后根据需要将执行权限授予其他用户，为了避免出现新函数能被所有人访问的时间窗口，应在一个事务中创建函数并且设置函数执行权限。开启数据库对象隔离属性后，普通用户只能查看有权限执行的函数定义。
- 在函数内部调用其它无参数的函数时，可以省略括号，直接使用函数名进行调用。
- 在函数内部调用其他有出参的函数，如果在赋值表达式中调用时，需要打开guc参数 set behavior_compat_options = 'proc_outparam_override' ，并提前定义与出参类型相同的变量，然后将变量作为出参调用带有出参的其他函数，出参才能生效。否则，被调函数的出参会被忽略。
- 兼容Oracle风格的函数支持参数注释的查看与导出、导入。
- 兼容Oracle风格的函数支持介于IS/AS与plsql_body之间的注释的查看与导出、导入。
- 被授予CREATE ANY FUNCTION权限的用户，可以在用户模式下创建/替换函数。
- 函数默认为SECURITY INVOKER权限，如果想将默认行为改为SECURITY DEFINER权限，需要设置guc参数behavior_compat_options='plsql_security_definer'。
- 不打开参数set behavior_compat_options = 'proc_outparam_override'时，被匿名块或存储过程直接调用的函数的OUT、IN OUT出参不能使用复合类型，并且RETURN值会被当做OUT出参的第一个值导致调用失败，想正确使用OUT、IN OUT出参，需打开参数set behavior_compat_options = 'proc_outparam_override'，见**示例**。
- 对于PL/SQL函数，打开参数behavior_compat_options='proc_outparam_override'后，out/inout的行为会改

变，函数中如果return和out/inout，可以同时返回，参数打开前只会返回return，见**示例**。

- 对于PL/SQL函数，打开参数behavior_compat_options='proc_outparam_override'后，有以下限制：
  a. 如果同一schema和package中已存在带有out/inout参数函数，不能再次创建带有out/inout参数的同名函数。
  b. 无论使用select还是call调用存储过程，都必须加上out参数。
  c. 部分场景不支持函数参与表达式（与参数打开前相比），如存储过程中左赋值，call function等，见**示例**。
  d. 不支持调用无return的函数，perform function调用。
  e. 存储过程中调用函数，不支持out/inout参数传入常量，可以传入变量，见**示例**。
  f. 打开GUC参数proc_outparam_override后，函数返回值为setof类型时，out出参不会生效。
- 函数创建时依赖未定义对象，如参数behavior_compat_options='plpgsql_dependency'打开，创建可执行，通过WARNING提示；如参数未打开，函数创建不可执行。
- behavior_compat_options='plpgsql_dependency'打开时，函数体中，调用函数A，函数A出入参包含函数B时，函数B不建立依赖。例如functionA(functionB())。gs_dependencies表仅建立和functionA的依赖。
- 如O风格函数已被视图直接依赖，且参数behavior_compat_options='plpgsql_dependency'打开，再次创建函数后视图可正常访问；如参数未打开，视图访问失败。
- 创建函数时，不支持使用函数自身作为入参默认值。
- 带OUT模式参数的函数不能在SQL语句中被调用。
- 带OUT模式参数的函数不能被SELECT INTO语法调用。
- 带OUT模式参数的函数不支持嵌套调用。
  比如：
  ```
  b := func(a,func(c,1));
  ```
  建议改为：
  ```
  tmp := func(c,1); b := func(a,tmp);
  ```
- 在创建函数时，不会检查函数内返回值的类型。
- 如果将定义者权限的函数创建到其他用户Schema下，则会以其他用户的权限执行该函数，有越权风险，请谨慎使用。
- 在运维管理员的Schema下，只允许初始用户和Schema的属主自己创建对象，不允许其他用户在运维管理员的Schema下创建对象或修改对象的Schema为运维管理员的Schema。
- 在表达式中使用out参数作为出参时，如下情况不会生效，例如：使用execute immediate sqlv using func语法执行函数、使用select func into语法执行函数、使用insert、update等DML语句执行、使用select where a=func()；带out出参的函数，作为入参时，fun（func（out b），a），out出参b未生效等。
- 调用带out出参的存储过程，设置GUC参数set behavior_compat_options = 'proc_outparam_transfer_length'后可以传递参数长度。规格限制如下：

a. 支持的基本类型包括：CHAR(n)、CHARACTER(n)、NCHAR(n)、VARCHAR(n)、VARYING(n)、VARCHAR2(n)、NVARCHAR2(n)类型的参数。
b. out出参不生效的情况下（比如perform）不需要传递长度。
c. 不支持精度传递的基本类型包括：NUMERIC、DECIMAL、NUMBER、FLOAT、DEC、INTEGER、TIME、TIMESTAMP、INTERVAL、TIME WITH TIME ZONE、TIMESTAMP WITH TIME ZONE、TIME WITHOUT TIME ZONE、TIMESTAMP WITHOUT TIME ZONE。
d. GUC参数set behavior_compat_options是否设置为proc_outparam_override时都支持传递参数长度。
e. 如果需要传递集合类型的元素长度和被集合类型嵌套的数组类型的元素长度，则需要在GUC参数behavior_compat_options里开启tableof_elem_constraints选项。

语法格式

- 兼容PostgreSQL风格的创建自定义函数语法。

```
CREATE [ OR REPLACE ] FUNCTION function_name
   [ ( [ { argname [ argmode ] argtype [ { DEFAULT | := | = } expression ]} [, ...] ] ) ]
   [ RETURNS rettype [ DETERMINISTIC ] | RETURNS TABLE ( { column_name column_type }
[, ...] )]
   LANGUAGE lang_name
   [
     {IMMUTABLE | STABLE | VOLATILE }
      | {SHIPPABLE | NOT SHIPPABLE}
      | WINDOW
      | [ NOT ] LEAKPROOF
      | {CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
      | {[ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER | AUTHID DEFINER |
AUTHID CURRENT_USER}
      | {fenced | not fenced}
      | {PACKAGE}
      | COST execution_cost
      | ROWS result_rows
      | SET configuration_parameter { {TO | =} value | FROM CURRENT }}
    ][...]
   {
      AS 'definition'
   }
```

- 兼容A模式数据库风格的创建自定义函数的语法。

```
CREATE [ OR REPLACE ] FUNCTION function_name
   ( [ { argname [ argmode ] argtype [ { DEFAULT | := | = } expression ] } [, ...] ] )
   RETURN rettype [ DETERMINISTIC ]
   [
      {IMMUTABLE | STABLE | VOLATILE }
      | {SHIPPABLE | NOT SHIPPABLE}
      | {PACKAGE}
      | {FENCED | NOT FENCED}
      | [ NOT ] LEAKPROOF
      | {CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
      | {[ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER |
AUTHID DEFINER | AUTHID CURRENT_USER
}
      | COST execution_cost
      | ROWS result_rows
      | SET configuration_parameter { {TO | =} value | FROM CURRENT
      | LANGUAGE lang_name
   ][...]

   {
     IS | AS
} plsql_body
/
```


参数说明

- **function_name**

  要创建的函数名称（可以用模式修饰）。

  取值范围：字符串，要符合**标识符命名规范**。且最多为63个字符。若超过63个字符，数据库会截断并保留前63个字符当做函数名称。

- **argname**

  函数参数的名称。

  取值范围：字符串，要符合**标识符命名规范**。且最多为63个字符。若超过63个字符，数据库会截断并保留前63个字符当做函数参数名称。

- **argmode**

  函数参数的模式。

  取值范围：IN，OUT，INOUT或VARIADIC。缺省值是IN。只有OUT模式的参数后面能跟VARIADIC。并且OUT和INOUT模式的参数不能用在RETURNS TABLE的函数定义中。

  说明

  VARIADIC用于声明数组类型的参数。

- **argtype**

  函数参数的类型。可以使用%TYPE或%ROWTYPE间接引用变量或表的类型，详细可参考存储过程章节**定义变量**。

- **expression**

  参数的默认表达式。

  说明

  - 在参数a_format_version值为10c和a_format_dev_version值为s2的情况下，函数参数为INOUT模式时不支持默认表达式。
  - 推荐使用方式：将所有默认值参数定义在所有非默认值参数后。
  - 调用带有默认参数的函数时，入参从左往右排入函数，如果有非默认参数的入参缺失则报错。
  - 打开proc_uncheck_default_param参数，调用带有默认参数的函数时，入参从左往右排入函数，如果有非默认参数的入参缺失，则会用错位的默认值填充该参数。
  - 在参数a_format_version值为10c、a_format_dev_version值为s1和关闭proc_outparam_override，函数参数同时包括out出参和default时，默认值不可缺省。

- **rettype**

  函数返回值的数据类型。与argtype相同，同样可以使用%TYPE或%ROWTYPE间接引用类型。

  如果存在OUT或INOUT参数，可以省略RETURNS子句。如果没有省略，则该子句必须和输出参数所表示的结果类型一致。如果有多个输出参数，则该子句为RECORD类型，否则该子句与单个输出参数的类型相同。

  SETOF修饰词表示该函数将返回一个集合，而不是单独一项。

  说明

  PACKAGE外FUNCTION argtype和rettype中%TYPE不支持引用PACKAGE变量的类型。

- **column_name**

  字段名称。

- **column_type**

  字段类型。

- **definition**

  一个定义函数的字符串常量，含义取决于语言。它可以是一个内部函数名称、一个指向某个目标文件的路径、一个SQL查询、一个过程语言文本。

- **DETERMINISTIC**

  SQL语法兼容接口，未实现功能，不推荐使用。

- **LANGUAGE lang_name**

  用以实现函数的语言的名称。可以是SQL，internal，或者是用户定义的过程语言名称。为了保证向下兼容，该名称可以用单引号（包围）。若采用单引号，则引号内必须为大写。

  由于兼容性问题，A数据库风格的语法无论指定任何语言，最终创建的语言都为plpgsql。

  说明

  - internal函数在定义时，如果AS指定为内部系统函数，则新创建函数的参数类型，参数个数，与返回值类型需要与内部系统函数保持一致，且需要有执行此内部系统函数的权限。
  - internal函数只支持拥有sysadmin权限的用户创建。

- **WINDOW**

  表示该函数是窗口函数。替换函数定义时不能改变WINDOW属性。

  须知

  自定义窗口函数只支持LANGUAGE是internal，并且引用的内部函数必须是窗口函数。

- **IMMUTABLE**

  表示该函数在给出同样的参数值时总是返回同样的结果。

- **STABLE**

  表示该函数不能修改数据库，对相同参数值，在同一次表扫描里，该函数的返回值不变，但是返回值可能在不同SQL语句之间变化。

- **VOLATILE**

  表示该函数值可以在一次表扫描内改变，因此不会做任何优化。

- **SHIPPABLE|NOT SHIPPABLE**

  表示该函数是否可以下推执行。预留接口，不推荐使用。

- **PACKAGE**

  表示该函数是否支持重载。PostgreSQL风格的函数本身就支持重载，此参数主要是针对其它风格的函数。

  - 不允许package函数和非package函数重载或者替换。
  - package函数不支持VARIADIC类型的参数。
  - 不允许修改函数的package属性。

- **LEAKPROOF**

  指出该函数的参数只包括返回值。LEAKPROOF只能由系统管理员设置。

- **CALLED ON NULL INPUT**

  表明该函数的某些参数是NULL的时候可以按照正常的方式调用。该参数可以省略。

- **RETURNS NULL ON NULL INPUT**

  STRICT

  STRICT用于指定如果函数的某个参数是NULL，此函数总是返回NULL。如果声明了这个参数，当有NULL值参数时该函数不会被执行；而只是自动返回一个NULL结果。

  RETURNS NULL ON NULL INPUT和STRICT的功能相同。

- **EXTERNAL**

  目的是和SQL兼容，是可选的，这个特性适合于所有函数，而不仅是外部函数。

- **SECURITY INVOKER**

  AUTHID CURRENT_USER

  表明该函数将带着调用它的用户的权限执行。该参数可以省略。

  SECURITY INVOKER和AUTHID CURRENT_USER的功能相同。

- **SECURITY DEFINER**

  AUTHID DEFINER

  声明该函数将以创建它的用户的权限执行。

  AUTHID DEFINER和SECURITY DEFINER的功能相同。

- **COST execution_cost**

  用来估计函数的执行成本。

  execution_cost以cpu_operator_cost为单位。

  取值范围：正数

- **ROWS result_rows**

  估计函数返回的行数。用于函数返回的是一个集合。

  取值范围：正数，默认值是1000行。

- **configuration_parameter**
  - **value**

    把指定的数据库会话参数值设置为给定的值。如果value是DEFAULT或者RESET，则在新的会话中使用系统的缺省设置。OFF关闭设置。

    取值范围：字符串

    - DEFAULT
    - OFF
    - RESET

    指定默认值。
  - **FROM CURRENT**

    取当前会话中的值设置为configuration_parameter的值。

- **plsql_body**

  PL/SQL存储过程体。

须知

当在函数体中进行创建用户、修改密码或加解密等涉及密码或密钥相关操作时，系统表及日志中会记录密码或密钥的明文信息。为防止敏感信息泄露，不建议用户在函数体中进行涉及密码或密钥等敏感信息的相关操作。

示例

```
--定义函数为SQL查询。
gaussdb=# CREATE FUNCTION func_add_sql(integer, integer) RETURNS integer
    AS 'select $1 + $2;'
    LANGUAGE SQL
    IMMUTABLE
    RETURNS NULL ON NULL INPUT;

--利用参数名用 plpgsql 自增一个整数。
gaussdb=# CREATE OR REPLACE FUNCTION func_increment_plsql(i integer) RETURNS integer AS $$
        BEGIN
                RETURN i + 1;
        END;
$$ LANGUAGE plpgsql;

--返回RECORD类型。
gaussdb=# CREATE OR REPLACE FUNCTION func_increment_sql(i int, out result_1 bigint, out result_2 bigint)
returns SETOF RECORD
as $$
begin
    result_1 = i + 1;
    result_2 = i * 10;
return next;
end;
$$language plpgsql;

--返回一个包含多个输出参数的记录。
gaussdb=# CREATE FUNCTION func_dup_sql(in int, out f1 int, out f2 text)
    AS $$ SELECT $1, CAST($1 AS text) || ' is text' $$
    LANGUAGE SQL;

gaussdb=# SELECT * FROM func_dup_sql(42);

--计算两个整数的和，并返回结果。如果输入为null，则返回null。
gaussdb=# CREATE FUNCTION func_add_sql2(num1 integer, num2 integer) RETURN integer
AS
BEGIN PAC
RETURN num1 + num2;
END;
/
--修改函数func_add_sql2的执行规则为IMMUTABLE，即参数不变时返回相同结果。
gaussdb=# ALTER FUNCTION func_add_sql2(INTEGER, INTEGER) IMMUTABLE;

--将函数func_add_sql2的名称修改为add_two_number。
gaussdb=# ALTER FUNCTION func_add_sql2(INTEGER, INTEGER) RENAME TO add_two_number;

--将函数add_two_number的属者改为omm。
gaussdb=# ALTER FUNCTION omm(INTEGER, INTEGER) OWNER TO omm;

--删除函数。
gaussdb=# DROP FUNCTION add_two_number;
gaussdb=# DROP FUNCTION func_increment_sql;
gaussdb=# DROP FUNCTION func_dup_sql;
gaussdb=# DROP FUNCTION func_increment_plsql;
gaussdb=# DROP FUNCTION func_add_sql;

--设置参数。
gaussdb=# SET behavior_compat_options='proc_outparam_override';
--创建函数。
```

```
gaussdb=# CREATE OR REPLACE FUNCTION func1(in a integer, out b integer)
RETURNS int
AS $$
DECLARE
    c int;
    BEGIN
        c := 1;
        b := a + c;
        return c;
    END; $$
LANGUAGE 'plpgsql' NOT FENCED;
--同时返回return和出参。
gaussdb=# DECLARE
    result integer;
    a integer := 2;
    b integer := NULL;
BEGIN
    result := func1(a => a, b => b);
    raise info 'b is: %', b;
    raise info 'result is: %', result;
END;
/
INFO:  b is: 3
INFO:  result is: 1
ANONYMOUS BLOCK EXECUTE
--不支持左赋值表达式。
gaussdb=# DECLARE
    result integer;
    a integer := 2;
    b integer := NULL;
BEGIN
    result := func1(a => a, b => b) + 1;
    raise info 'b is: %', b;
    raise info 'result is: %', result;
END;
/
ERROR:  when invoking function func1, maybe input something superfluous.
CONTEXT:  compilation of PL/SQL function "inline_code_block" near line 3
--存储过程中部分场景不支持出参赋值(目前仅va := f1(x,y);以及return f1(x,y);方式支持出参赋值）。
gaussdb=# CREATE OR REPLACE PROCEDURE p1 AS
x int := 10;
y int;
BEGIN
if func1(a => x, b => y) = 11 then  --出参y无法赋值
raise info 'case1';
raise info 'y is:%',y;
ELSE
raise info 'case2';
raise info 'y is:%',y;
END IF;
END;
/
gaussdb=# CALL p1();
INFO:  case2
INFO:  y is:<NULL>
 p1
----

(1 row)
--考虑到前向兼容，此场景未报错，在设置参数 plsql_compile_check_options ='plsql_expression_check'后，此用例执行会报错。
gaussdb=# SET plsql_compile_check_options ='plsql_expression_check';
gaussdb=# CALL p1();
ERROR:  function func1 with out parameters cannot be called in another expression.
CONTEXT:  SQL statement "SELECT func1(a => x, b => y)= 11"
PL/SQL function p1() line 5 at IF
--存储过程中不支持out/inout传入常量。
gaussdb=# DECLARE
    result integer;
```



```
    a integer := 2;
    b integer := NULL;
BEGIN
    result := func1(a => a, b => 10);
    raise info 'b is: %', b;
    raise info 'result is: %', result;
END;
/
ERROR:  when invoking function func1, no destination for argments "b"
CONTEXT:  compilation of PL/SQL function "inline_code_block" near line 3
--存储过程中支持out/inout传入变量。
gaussdb=# DECLARE
    result integer;
    a integer := 2;
    b integer := NULL;
BEGIN
    result := func1(a,b);
    raise info 'b is: %', b;
    raise info 'result is: %', result;
END;
/
INFO:  b is: 3
INFO:  result is: 1
ANONYMOUS BLOCK EXECUTE

--不打开参数set behavior_compat_options = 'proc_outparam_override'时，被匿名块或存储过程直接调用的函数的OUT、IN OUT出参不能使用复合类型，并且RETURN值会被当做OUT出参的第一个值导致调用失败。
gaussdb=# CREATE TYPE rec AS(c1 int, c2 int);
gaussdb=# CREATE OR REPLACE FUNCTION func(a in out rec, b in out int) RETURN int
AS
BEGIN
  a.c1:=100;
  a.c2:=200;
  b:=300;
  return 1;
END;
/
gaussdb=# DECLARE
  r rec;
  b int;
BEGIN
  func(r,b);  --不支持。
END;
/
ERROR:  cannot assign non-composite value to a row variable
CONTEXT:  PL/SQL function inline_code_block line 4 at SQL statement

gaussdb=# CREATE OR REPLACE PACKAGE pkg_type AS
    type table_of_index_int is table of integer index by integer; --创建integer类型。
    type table_of_index_int01 is table of table_of_index_int index by integer; --创建嵌套integer类型。
    type table_of_index_var is table of integer index by varchar(5); --创建varchar类型。
    type table_of_index_var01 is table of table_of_index_var index by varchar(5); --创建嵌套varchar类型。
END pkg_type;
/

gaussdb=# CREATE OR REPLACE FUNCTION func_001(a in out pkg_type.table_of_index_int, b in out
pkg_type.table_of_index_var) --#add in & inout #defult value
RETURN pkg_type.table_of_index_int
AS
    table_of_index_int_val pkg_type.table_of_index_int;
    table_of_index_var_val pkg_type.table_of_index_var;
BEGIN
    for i in 1..2 loop
        table_of_index_int_val(i) := i;
        a(i) := i;
        table_of_index_var_val(i) := i;
        b(i) := i;
    end loop;
    raise info '%',table_of_index_int_val;
```



```
    raise info '%',table_of_index_var_val;
    raise info '%',a;
    raise info '%',b;
    return table_of_index_int_val;
END;
/
gaussdb=# DECLARE
    table_of_index_int_val pkg_type.table_of_index_int;
    table_of_index_var_val pkg_type.table_of_index_var;
BEGIN
    func_001(table_of_index_int_val,table_of_index_var_val);
END;
/

INFO:  {indexbyType:int,1=>1,2=>2}
CONTEXT:  PL/SQL function inline_code_block line 5 at SQL statement
INFO:  {indexbyType:varchar,"1"=>1,"2"=>2}
CONTEXT:  PL/SQL function inline_code_block line 5 at SQL statement
INFO:  {indexbyType:int,1=>1,2=>2}
CONTEXT:  PL/SQL function inline_code_block line 5 at SQL statement
INFO:  {indexbyType:varchar,"1"=>1,"2"=>2}
CONTEXT:  PL/SQL function inline_code_block line 5 at SQL statement
ERROR:  expression is of wrong type
CONTEXT:  PL/SQL function inline_code_block line 5 at SQL statement

gaussdb=# CREATE OR REPLACE FUNCTION func_001(a in out date, b in out date) --#add in & inout
#defult value
RETURN integer
AS
BEGIN
    raise info '%', a;
    raise info '%', b;
RETURN 1;
END;
/
gaussdb=# DECLARE
    date1 date := '2022-02-02';
    date2 date := '2022-02-02';
BEGIN
    func_001(date1, date2);
END;
/
INFO:  2022-02-02 00:00:00
CONTEXT:  PL/SQL function inline_code_block line 5 at SQL statement
INFO:  2022-02-02 00:00:00
CONTEXT:  PL/SQL function inline_code_block line 5 at SQL statement
ERROR:  invalid input syntax for type timestamp: "1"
CONTEXT:  PL/SQL function inline_code_block line 5 at SQL statement

gaussdb=# CREATE OR REPLACE FUNCTION func_001(a in out INT, b in out date) --#add in & inout #defult
value
RETURN int
AS
BEGIN
    raise info '%', a;
    raise info '%', b;
RETURN a;
END;
/
gaussdb=# DECLARE
    date1 int := 1;
    date2 date := '2022-02-02';
BEGIN
    func_001(date1, date2);
END;
/
INFO:  1
CONTEXT:  PL/SQL function inline_code_block line 5 at SQL statement
INFO:  2022-02-02 00:00:00
```



```
CONTEXT:  PL/SQL function inline_code_block line 5 at SQL statement
ANONYMOUS BLOCK EXECUTE
```

相关链接

ALTER FUNCTION，**DROP FUNCTION**

7.14.64 CREATE GROUP

功能描述

创建一个新用户组。

注意事项

CREATE GROUP是CREATE ROLE的别名，非SQL标准语法，不推荐使用，建议用户直接使用CREATE ROLE替代。

语法格式

```
CREATE GROUP group_name [ [ WITH ] option [ ... ] ] [ ENCRYPTED | UNENCRYPTED ] { PASSWORD |
IDENTIFIED BY } { 'password' [ EXPIRED ] | DISABLE };
```

其中可选项option子句语法为：

```
{SYSADMIN | NOSYSADMIN}
    | {MONADMIN | NOMONADMIN}
    | {OPRADMIN | NOOPRADMIN}
    | {POLADMIN | NOPOLADMIN}
    | {AUDITADMIN | NOAUDITADMIN}
    | {CREATEDB | NOCREATEDB}
    | {USEFT | NOUSEFT}
    | {CREATEROLE | NOCREATEROLE}
    | {INHERIT | NOINHERIT}
    | {LOGIN | NOLOGIN}
    | {REPLICATION | NOREPLICATION}
    | {VCADMIN | NOVCADMIN}
    | {PERSISTENCE | NOPERSISTENCE}
    | CONNECTION LIMIT connlimit
    | VALID BEGIN 'timestamp'
    | VALID UNTIL 'timestamp'
    | RESOURCE POOL 'respool'
    | PERM SPACE 'spacelimit'
    | TEMP SPACE 'tmpspacelimit'
    | SPILL SPACE 'spillspacelimit'
    | IN ROLE role_name [, ...]
    | IN GROUP role_name [, ...]
    | ROLE role_name [, ...]
    | ADMIN rol e_name [, ...]
    | USER role_name [, ...]
    | SYSID uid
    | DEFAULT TABLESPACE tablespace_name
    | PROFILE DEFAULT
    | PROFILE profile_name
    | PGUSER
```

参数说明

请参考CREATE ROLE的**参数说明**。

示例

```
--创建组my_group。
gaussdb=# CREATE GROUP my_group PASSWORD '********';
```



```
--删除组。
gaussdb=# DROP GROUP my_group;
```

相关链接

ALTER GROUP，**DROP GROUP**，**CREATE ROLE**

7.14.65 CREATE INCREMENTAL MATERIALIZED VIEW

功能描述

CREATE INCREMENTAL MATERIALIZED VIEW会创建一个增量物化视图，后续可以使用REFRESH MATERIALIZED VIEW（全量刷新）和REFRESH INCREMENTAL MATERIALIZED VIEW（增量刷新）刷新物化视图的数据。

CREATE INCREMENTAL MATERIALIZED VIEW类似于CREATE TABLE AS，不过它会记住被用来初始化该视图的查询， 因此它可以在后续中进行数据刷新。一个物化视图有很多和表相同的属性，但是不支持临时物化视图。

注意事项

- 增量物化视图不可以在DATABASE LINK表、临时表或全局临时表上创建。
- 增量物化视图仅支持简单过滤查询和基表UNION ALL查询。
- 创建增量物化视图不支持WITH子句、GROUP BY子句、ORDER BY子句、LIMIT子句、WINDOW子句、DISTINCT算子、AGG算子以及不支持除UNION ALL外的子查询。
- 创建增量物化视图后，基表中的绝大多数DDL操作不再支持。
- 不支持对增量物化视图进行IUD操作。
- 增量物化视图创建后，当基表数据发生变化时，需要使用刷新（REFRESH）命令保持物化视图与基表同步。
- Ustore引擎不支持物化视图的创建和使用。

语法格式

```
CREATE INCREMENTAL MATERIALIZED VIEW mv_name
   [ (column_name [, ...] ) ]
   [ TABLESPACE tablespace_name ]
   AS query;
```

参数说明

- **mv_name**

  要创建的物化视图的名称（可以被模式限定）。

  取值范围：字符串，要符合**标识符命名规范**。

- **column_name**

  新物化视图中的一个列名。物化视图支持指定列，指定列需要和后面的查询语句结果的列数量保持一致；如果没有提供列名，会从查询的输出列名中获取列名。

  取值范围：字符串，要符合**标识符命名规范**。

- **TABLESPACE tablespace_name**

  指定新建物化视图所属表空间。如果没有声明，将使用默认表空间。

- **AS query**
  一个SELECT或者TABLE命令。这个查询将在一个安全受限的操作中运行。

示例

```
--创建一个普通表。
gaussdb=# CREATE TABLE my_table (c1 int, c2 int) WITH(STORAGE_TYPE=ASTORE);

--创建增量物化视图。
gaussdb=# CREATE INCREMENTAL MATERIALIZED VIEW my_imv AS SELECT * FROM my_table;

--基表写入数据。
gaussdb=# INSERT INTO my_table VALUES(1,1),(2,2);

--对增量物化视图my_imv进行增量刷新。
gaussdb=# REFRESH INCREMENTAL MATERIALIZED VIEW my_imv;

--删除增量物化视图。
gaussdb=# DROP MATERIALIZED VIEW my_imv;

--删除普通表my_table。
gaussdb=# DROP TABLE my_table;
```

相关链接

ALTER MATERIALIZED VIEW， **CREATE MATERIALIZED VIEW**，**CREATE TABLE**，**DROP MATERIALIZED VIEW**，**REFRESH INCREMENTAL MATERIALIZED VIEW** ，**REFRESH MATERIALIZED VIEW**

7.14.66 CREATE INDEX

功能描述

在指定的表上创建索引。

索引可以用来提高数据库查询性能，但是不恰当的使用将导致数据库性能下降。建议仅在匹配如下某条原则时创建索引：

- 经常执行查询的字段。
- 在连接条件上创建索引，对于存在多字段连接的查询，建议在这些字段上建立组合索引。例如，select * from t1 join t2 on t1.a=t2.a and t1.b=t2.b，可以在t1表上的a，b字段上建立组合索引。
- where子句的过滤条件字段上（尤其是范围条件）。
- 在经常出现在order by、group by和distinct后的字段。

在分区表上创建索引与在普通表上创建索引的语法不太一样，使用时请注意，如当索引带GLOBAL/LOCAL关键字或者创建索引为GLOBAL索引时不支持创建部分索引。需要注意分区表上创建索引会根据如下规则进行判断：如果创建索引时申明了GLOBAL/LOCAL关键字，则创建对应类型的索引；否则如果创建索引指定分区名，则创建LOCAL索引；否则如果是unique索引，包含非分区键时创建GLOBAL索引，包含全部分区键则创建LOCAL索引；否则默认创建GLOBAL索引。

注意事项

- 索引自身也占用存储空间、消耗计算资源，创建过多的索引将对数据库性能造成负面影响（尤其影响数据导入的性能，建议在数据导入后再建索引）。因此，仅在必要时创建索引。

- 索引定义里的所有函数和操作符都必须是immutable类型的，即它们的结果必须只能依赖于它们的输入参数，而不受任何外部的影响（如另外一个表的内容或者当前时间）。这个限制可以确保该索引的行为是定义良好的。要在一个索引上或WHERE中使用用户定义函数，请把它标记为immutable类型函数。
- 分区表索引分为LOCAL索引与GLOBAL索引，LOCAL索引与某个具体分区绑定，而GLOBAL索引则对应整个分区表。
- 被授予CREATE ANY INDEX权限的用户，可以在public模式和用户模式下创建索引。
- 如果表达式索引中调用的是用户自定义函数，按照函数创建者权限执行表达式索引函数。
- 不支持XML类型数据作为普通索引、UNIQUE索引、GLOBAL索引、LOCAL索引、部分索引。
- 在线创建索引只支持B-tree索引和UB-tree索引，只支持普通索引、GLOBAL索引、LOCAL索引。在线并行创建索引只支持Astore的普通索引、GLOBAL索引、LOCAL索引，Ustore索引不支持在线并行创建。
- 使用CREATE INDEX创建索引可能会改变表的访问方式，从而导致查询执行计划改变。
- 在创建组合索引时，需根据查询条件和组合索引最左匹配原则创建。

  说明

  - 组合索引最左匹配原则：如果查询条件包含了组合索引的一列或者多列，那么组合索引的最左边开始的连续列需要与查询条件匹配上。
  - 当查询为where a = ?, b = ?,c = ?, d = ? 或者 where a = ?, b = ?,c = ? 等时，都是最佳的索引匹配；当查询为where b = ?,c = ?, d = ? 或者 where c = ?, d = ? 等时，在代价计算后可能也会走索引idx_test_abcd，但是这种情况走索引时会扫描索引的全部页面，导致SQL性能并不理想。类似情况建议根据最左匹配原则创建适合此查询条件的组合索引。

    ```
    --创建表test。
    gaussdb=# CREATE TABLE test(a int, b int, c int, d int, e int, f text);
    创建组合索引。
    gaussdb=# CREATE INDEX idx_test_abcd ON test(a,b,c,d);
    ```

语法格式

- 在表上创建索引。

  ```
  CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ [schema_name.] index_name ] ON table_name
  [ USING method ]
      ({ { column_name [ ( length ) ] | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ]
  [ NULLS { FIRST | LAST } ] }[, ...] )
      [ INCLUDE ( column_name [, ...] ) ]
      [ WITH ( {storage_parameter = value} [, ... ] ) ]
      [ TABLESPACE tablespace_name ]
      [ WHERE predicate ];
  ```

- 在分区表上创建索引。

  ```
  CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ [schema_name.] index_name ] ON table_name
  [ USING method ]
      ( { { column_name [ ( length ) ] | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ]
  [ NULLS LAST ] } [, ...] )
      [ LOCAL [ ( { PARTITION index_partition_name | SUBPARTITION index_subpartition_name
  [ TABLESPACE index_partition_tablespace ] } [, ...] ) ] | GLOBAL ]
      [ INCLUDE ( column_name [, ...] ) ]
      [ WITH ( { storage_parameter = value } [, ...] ) ]
      [ TABLESPACE tablespace_name ];
  ```


参数说明

- **UNIQUE**

  创建唯一性索引，每次添加数据时检测表中是否有重复值。如果插入或更新的值会引起重复的记录时，将导致一个错误。

  目前只有B-tree及UBtree索引支持唯一索引。

- **CONCURRENTLY**

  以不阻塞DML的方式创建索引（加ShareUpdateExclusiveLock锁）。创建索引时，一般会阻塞其他语句对该索引所依赖表的访问。指定此关键字，可以实现创建过程中不阻塞DML。

  - 此选项只能指定一个索引的名称。
  - 普通CREATE INDEX命令可以在事务内执行，但是CREATE INDEX CONCURRENTLY不可以在事务内执行。
  - 对于临时表，支持使用CONCURRENTLY关键字创建索引，但是实际创建过程中，采用的是阻塞式的创建方式，因为没有其他会话会并发访问临时表，并且阻塞式创建成本更低。

  说明

  - 创建索引时指定此关键字，Astore需要执行先后两次对该表的全表扫描来完成build，第一次扫描的时候创建索引，不阻塞读写操作；第二次扫描的时候合并更新第一次扫描到目前为止发生的变更；Ustore只需全表扫描一次来完成索引创建。
  - Astore由于需要执行两次对表的扫描和build，而且必须等待现有的所有可能对该表执行修改的事务结束。这意味着该索引的创建比正常耗时更长，同时因此带来的CPU和I/O消耗对其他业务也会造成影响；Ustore虽只需全表扫描一次来完成索引创建，但上述消耗同样存在。
  - 如果在索引构建时发生失败，那会留下一个“不可用”的索引。该索引会被查询忽略，但仍存在消耗开销。这种情况推荐的恢复方法是删除该索引并尝试再次CONCURRENTLY创建索引，或者通过CLUSTER/TRUNCATE/VACUUM FULL/REINDEX重建索引。值得注意的是，ALTER TABLE若涉及表和索引重建将自动清理残留的”不可用”索引。
  - 由于在第二次扫描之后，索引构建必须等待任何持有早于第二次扫描拿的快照的事务终止，而且建索引时加的ShareUpdateExclusiveLock锁（4级）会和大于等于4级的锁冲突，在创建这类索引时，容易引发卡住（hang）或者死锁问题。例如：
    - 两个会话对同一个表创建CONCURRENTLY索引，会引起死锁问题；
    - 两个会话，一个对表创建CONCURRENTLY索引，一个drop table，会引起死锁问题；
    - 三个会话，会话1先对表a加锁，不提交，会话2接着对表b创建CONCURRENTLY索引，会话3接着对表a执行写入操作，在会话1事务未提交之前，会话2会一直被阻塞；
    - 将事务隔离级别设置成可重复读（默认为读已提交），起两个会话，会话1起事务对表a执行写入操作，不提交，会话2对表b创建CONCURRENTLY索引，在会话1事务未提交之前，会话2会一直被阻塞。
  - 在IO、CPU不受限的情况下，在线创建索引对业务性能的劣化一般可以控制在10%以内，但在特殊场景下劣化可能会超过此数值。这是因为在线创建索引本身是一种消耗IO、CPU资源较多的长事务，需要比离线创建索引消耗更多的资源。在线创建索引事务持续时间越长，对业务性能的影响越大。在线创建索引时间与基表数据量、并发DML产生的数据量正相关，在IO、CPU不受限的情况下，在线创建索引时间大约是离线创建索引的2~6倍，但当并发事务量较大（>10000tps）或存在资源争抢的情况时，可能会超过此数值。在Astore模式下，可以使用并行创建索引来缩短创建索引时间；在线并行创建索引性能随着并行工作线程数量增加而提升到一定值后稳定，相比串行创建索引性能一般可提升30%左右。建议在业务低谷期进行在线创建索引，以避免对业务造成较大影响。虽然在线创建索引在一定程度上提供了业务不中断的能力，但仍然需要谨慎实施。

- **schema_name**

  模式的名称。

  取值范围：已存在模式名。

- **index_name**

  要创建的索引名，索引的模式与表相同。

  取值范围：字符串，要符合**标识符命名规范**。

- **table_name**

  需要为其创建索引的表的名称，可以用模式修饰。

  取值范围：已存在的表名。

- **USING method**

  指定创建索引的方法。

  取值范围：

  - btree：B-tree索引使用一种类似于B+树的结构来存储数据的键值，通过这种结构能够快速的查找索引。B-tree适合支持比较查询以及查询范围。建索引时在表为Ustore时会自动变换为UB-tree。
  - ubtree：仅供Ustore表使用的多版本B-tree索引，索引页面上包含事务信息，能并自主回收页面。UB-tree索引默认开启insertpt功能。

  行存表（Astore存储引擎）支持的索引类型：btree（行存表缺省值）、。行存表（Ustore存储引擎）支持的索引类型：ubtree。

- **column_name**

  表中需要创建索引的列的名称（字段名）。

  如果索引方式支持多字段索引，可以声明多个字段。全局索引最多可以声明31个字段，其他索引最多可以声明32个字段。

- **column_name (length)**

  创建一个基于该表一个字段的前缀键索引，column_name为前缀键的字段名，length为前缀长度。

  前缀键将取指定字段数据的前缀作为索引键值，可以减少索引占用的存储空间。含有前缀键字段的部分过滤条件和连接条件可以使用索引。

  说明

  - 前缀键支持的索引方法：btree、ubtree。
  - 前缀键的字段的数据类型必须是二进制类型或字符类型（不包括特殊字符类型）。
  - 前缀长度必须是不超过2676的正整数，并且不能超过字段的最大长度。对于二进制类型，前缀长度以字节数为单位。对于非二进制字符类型，前缀长度以字符数为单位。键值的实际长度受内部页面限制，若字段中含有多字节字符、或者一个索引上有多个键，索引行长度可能会超限，导致报错，设定较长的前缀长度时请考虑此情况。
  - CREATE INDEX语法中，不支持以下关键字作为前缀键的字段名称：COALESCE、EXTRACT、GREATEST、LEAST、LNNVL、NULLIF、NVL、NVL2、OVERLAY、POSITION、REGEXP_LIKE、SUBSTRING、TIMESTAMPDIFF、TREAT、TRIM、XMLCONCAT、XMLELEMENT、XMLEXISTS、XMLFOREST、XMLPARSE、XMLPI、XMLROOT、XMLSERIALIZE。
  - 前缀键属于一种特殊的表达式键，部分未说明的约束和限制，与表达式键一致，请参考表达式索引的说明。

- **expression**

  创建一个基于该表的一个或多个字段的表达式索引，通常必须写在圆括弧中。如果表达式有函数调用的形式，圆括弧可以省略。

表达式索引可用于获取对基本数据的某种变形的快速访问。比如，一个在upper(col)上的函数索引将允许WHERE upper(col) = 'JIM'子句使用索引。

在创建表达式索引时，如果表达式中包含IS NULL子句，则这种索引是无效的。此时，建议用户尝试创建一个部分索引。

- **COLLATE collation**

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

- **opclass**

  操作符类的名称。对于索引的每一列可以指定一个操作符类，操作符类标识了索引那一列的使用的操作符。例如一个B-tree索引在一个四字节整数上可以使用int4_ops；这个操作符类包括四字节整数的比较函数。实际上对于列上的数据类型默认的操作符类是足够用的。操作符类主要用于一些有多种排序的数据。例如，用户想按照绝对值或者实数部分排序一个复数，可以通过定义两个操作符类，然后在建立索引时选择合适的类。当包含字符串类型（varchar、varchar2、text等）的索引的COLLATE的值不是C或者POSIX，但要求索引能够支持前缀匹配时，则需要指定varchar_pattern_ops选项。

- **ASC**

  指定按升序排序（默认）。

- **DESC**

  指定按降序排序。

- **NULLS FIRST**

  指定空值在排序中排在非空值之前，当指定DESC排序时，本选项为默认的。

- **NULLS LAST**

  指定空值在排序中排在非空值之后，未指定DESC排序时，本选项为默认的。

- **LOCAL**

  指定创建的分区索引为LOCAL索引。

- **GLOBAL**

  指定创建的分区索引为GLOBAL索引，当不指定LOCAL、GLOBAL关键字时，默认创建GLOBAL索引。

- **INCLUDE (column_name [, ...]**)

  可选的 INCLUDE 子句指定将一些非键列（non-key columns）包含在索引中。非键列不能用于作为索引扫描的加速搜索条件，同时在检查索引的唯一性约束时会忽略它们。

  仅索引扫描 (Index Only Scan) 可以直接返回非键列中的内容，而不必去访问索引所对应的堆表。

  将非键列添加为 INCLUDE 列需要保守一些，尤其是对于宽列。如果索引元组超过索引类型允许的最大大小，数据将插入失败。需要注意的是，任何情况下为索引添加非键列都会增加索引的空间占用，从而可能减慢搜索速度。

  目前只有ubtree索引访问方式支持该特性。非键列会被保存在与堆元组对应的索引叶子元组中，不会包含在索引上层页面的元组中。

- **WITH ({storage_parameter = value} [, ...])**

  指定索引方法的存储参数。

  取值范围：

Psort之外的索引都支持FILLFACTOR参数。只有UBTREE索引支持INDEXSPLIT参数。只有非分区表的BTREE索引支持DEDUPLICATION参数。

- FILLFACTOR

  一个索引的填充因子（fillfactor）是一个介于10和100之间的百分数。对于大并发插入且键值范围比较密集的场景，插入时同一个索引页面竞争比较大时，请选择较小的填充因子。

  取值范围：10~100

- INDEXSPLIT

  UBTREE索引选择采取哪种分裂策略。其中DEFAULT策略指的是与BTREE相同的分裂策略。INSERTPT策略能在某些场景下显著降低索引空间占用。

  取值范围：INSERTPT, DEFAULT

  默认值：INSERTPT

- ACTIVE_PAGES

  表示索引的页面数量，可能比实际的物理文件页面少，可以用于优化器调优。目前只对ustore的分区表local索引生效，且会被vacuum、analyze更新（包括auto vacuum）。不建议用户手动设置该参数。

- DEDUPLICATION

  索引参数，设置索引是否对键值重复的元组进行去重压缩。在重复键值的索引较多时，开启参数可以有效降低索引占用空间。对主键索引和唯一索引不生效。非唯一索引且索引键值重复度很低或者唯一的场景，开启参数会使索引插入性能小幅度劣化。暂不支持分区表的local/global索引。

  取值范围：布尔值，默认取GUC参数中enable_default_index_deduplication的值（默认为off）。

- **TABLESPACE tablespace_name**

  指定索引的表空间，如果没有声明则使用默认的表空间。

  取值范围：已存在的表空间名。

- **WHERE predicate**

  创建一个部分索引。部分索引是一个只包含表的一部分记录的索引，通常是该表中比其他部分数据更有用的部分。例如，有一个表，表里包含已记账和未记账的订单，未记账的订单只占表的一小部分而且这部分是最常用的部分，此时就可以通过只在未记账部分创建一个索引来改善性能。另外一个可能的用途是使用带有UNIQUE的WHERE强制一个表的某个子集的唯一性。

  取值范围：predicate表达式只能引用表的字段，它可以使用所有字段，而不仅是被索引的字段。目前，子查询和聚集表达式不能出现在WHERE子句里。不建议使用int等数值类型作为predicate，因为int等数值类型可以隐式转换为bool值（非0值隐式转换为true，0转换为false），可能导致非预期的结果。

  对于分区表索引，当创建索引带GLOBAL/LOCAL关键字，或者最终创建的索引类型为GLOBAL索引时，不支持带WHERE子句创建索引。

- **PARTITION index_partition_name**

  索引分区的名称。

  取值范围：字符串，要符合**标识符命名规范**。

- **SUBPARTITION index_subpartition_name**

  索引二级分区的名称。

  取值范围：字符串，要符合**标识符命名规范**。

- **TABLESPACE index_partition_tablespace**
  索引分区的表空间。
  取值范围：如果没有声明，将使用分区表索引的表空间index_tablespace。

示例

```
--创建表tpcds.ship_mode_t1。
gaussdb=# CREATE SCHEMA tpcds;
gaussdb=# CREATE TABLE tpcds.ship_mode_t1
(
    SM_SHIP_MODE_SK           INTEGER               NOT NULL,
    SM_SHIP_MODE_ID           CHAR(16)              NOT NULL,
    SM_TYPE                   CHAR(30)                      ,
    SM_CODE                   CHAR(10)                      ,
    SM_CARRIER                CHAR(20)                      ,
    SM_CONTRACT               CHAR(20)
)
;

--在表tpcds.ship_mode_t1上的SM_SHIP_MODE_SK字段上创建普通的唯一索引。
gaussdb=# CREATE UNIQUE INDEX ds_ship_mode_t1_index1 ON tpcds.ship_mode_t1(SM_SHIP_MODE_SK);

--在表tpcds.ship_mode_t1上的SM_SHIP_MODE_SK字段上创建指定B-tree索引。
gaussdb=# CREATE INDEX ds_ship_mode_t1_index4 ON tpcds.ship_mode_t1 USING
btree(SM_SHIP_MODE_SK);

--在表tpcds.ship_mode_t1上SM_CODE字段上创建表达式索引。
gaussdb=# CREATE INDEX ds_ship_mode_t1_index2 ON tpcds.ship_mode_t1(SUBSTR(SM_CODE,1 ,4));

--在表tpcds.ship_mode_t1上的SM_SHIP_MODE_SK字段上创建SM_SHIP_MODE_SK大于10的部分索引。
gaussdb=# CREATE UNIQUE INDEX ds_ship_mode_t1_index3 ON tpcds.ship_mode_t1(SM_SHIP_MODE_SK)
WHERE SM_SHIP_MODE_SK>10;

--在表tpcds.ship_mode_t1上SM_SHIP_MODE_SK字段上在线创建索引。
gaussdb=# CREATE INDEX CONCURRENTLY ds_ship_mode_t1_index6 ON
tpcds.ship_mode_t1(SM_SHIP_MODE_SK);

--在表tpcds.ship_mode_t1上SM_TYPE字段上创建前缀键索引。
gaussdb=# CREATE INDEX ds_ship_mode_t1_prefix_index ON tpcds.ship_mode_t1(SM_TYPE(4));

--重命名一个现有的索引。
gaussdb=# ALTER INDEX tpcds.ds_ship_mode_t1_index1 RENAME TO ds_ship_mode_t1_index5;

--设置索引不可用。
gaussdb=# ALTER INDEX tpcds.ds_ship_mode_t1_index2 UNUSABLE;

--重建索引。
gaussdb=# ALTER INDEX tpcds.ds_ship_mode_t1_index2 REBUILD;

--删除一个现有的索引。
gaussdb=# DROP INDEX tpcds.ds_ship_mode_t1_index2;

--删除表。
gaussdb=# DROP TABLE tpcds.ship_mode_t1;

--创建表空间。
gaussdb=# CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace1/tablespace_1';
gaussdb=# CREATE TABLESPACE example2 RELATIVE LOCATION 'tablespace2/tablespace_2';
gaussdb=# CREATE TABLESPACE example3 RELATIVE LOCATION 'tablespace3/tablespace_3';
gaussdb=# CREATE TABLESPACE example4 RELATIVE LOCATION 'tablespace4/tablespace_4';
--创建表tpcds.customer_address_p1。
gaussdb=# CREATE TABLE tpcds.customer_address_p1
(
    CA_ADDRESS_SK             INTEGER               NOT NULL,
    CA_ADDRESS_ID             CHAR(16)              NOT NULL,
    CA_STREET_NUMBER          CHAR(10)                      ,
    CA_STREET_NAME            VARCHAR(60)                   ,
    CA_STREET_TYPE            CHAR(15)                      ,
```



```
    CA_SUITE_NUMBER          CHAR(10)                  ,
    CA_CITY                  VARCHAR(60)               ,
    CA_COUNTY                VARCHAR(30)               ,
    CA_STATE                 CHAR(2)                   ,
    CA_ZIP                   CHAR(10)                  ,
    CA_COUNTRY               VARCHAR(20)               ,
    CA_GMT_OFFSET            DECIMAL(5,2)              ,
    CA_LOCATION_TYPE         CHAR(20)
)
TABLESPACE example1
PARTITION BY RANGE(CA_ADDRESS_SK)
(
  PARTITION p1 VALUES LESS THAN (3000),
  PARTITION p2 VALUES LESS THAN (5000) TABLESPACE example1,
  PARTITION p3 VALUES LESS THAN (MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;
--创建分区表索引ds_customer_address_p1_index1，不指定索引分区的名称。
gaussdb=# CREATE INDEX ds_customer_address_p1_index1 ON
tpcds.customer_address_p1(CA_ADDRESS_SK) LOCAL;
--创建分区表索引ds_customer_address_p1_index2，并指定索引分区的名称。
gaussdb=# CREATE INDEX ds_customer_address_p1_index2 ON
tpcds.customer_address_p1(CA_ADDRESS_SK) LOCAL
(
   PARTITION CA_ADDRESS_SK_index1,
   PARTITION CA_ADDRESS_SK_index2 TABLESPACE example3,
   PARTITION CA_ADDRESS_SK_index3 TABLESPACE example4
)
TABLESPACE example2;

--创建GLOBAL分区索引。
gaussdb=# CREATE INDEX ds_customer_address_p1_index3 ON
tpcds.customer_address_p1(CA_ADDRESS_ID) GLOBAL;

--不指定关键字，默认创建GLOBAL分区索引。
gaussdb=# CREATE INDEX ds_customer_address_p1_index4 ON
tpcds.customer_address_p1(CA_ADDRESS_ID);

--在线创建分区表索引ds_customer_address_p1_index5，不指定索引分区的名称。
gaussdb=# CREATE INDEX CONCURRENTLY ds_customer_address_p1_index5 ON
tpcds.customer_address_p1(CA_ADDRESS_SK) LOCAL;

--在线创建GLOBAL分区索引ds_customer_address_p1_index6。
gaussdb=# CREATE INDEX CONCURRENTLY ds_customer_address_p1_index6 ON
tpcds.customer_address_p1(CA_ADDRESS_ID) GLOBAL;

--修改分区表索引CA_ADDRESS_SK_index2的表空间为example1。
gaussdb=# ALTER INDEX tpcds.ds_customer_address_p1_index2 MOVE PARTITION CA_ADDRESS_SK_index2
TABLESPACE example1;

--修改分区表索引CA_ADDRESS_SK_index3的表空间为example2。
gaussdb=# ALTER INDEX tpcds.ds_customer_address_p1_index2 MOVE PARTITION CA_ADDRESS_SK_index3
TABLESPACE example2;

--重命名分区表索引。
gaussdb=# ALTER INDEX tpcds.ds_customer_address_p1_index2 RENAME PARTITION
CA_ADDRESS_SK_index1 TO CA_ADDRESS_SK_index4;

--删除索引和分区表。
gaussdb=# DROP INDEX tpcds.ds_customer_address_p1_index1;
gaussdb=# DROP INDEX tpcds.ds_customer_address_p1_index2;
gaussdb=# DROP TABLE tpcds.customer_address_p1;
--删除表空间。
gaussdb=# DROP TABLESPACE example1;
gaussdb=# DROP TABLESPACE example2;
gaussdb=# DROP TABLESPACE example3;
gaussdb=# DROP TABLESPACE example4;
```


相关链接

ALTER INDEX，**DROP INDEX**

优化建议

- create index

  建议仅在匹配如下条件之一时创建索引：

  - 经常执行查询的字段。
  - 在连接条件上创建索引，对于存在多字段连接的查询，建议在这些字段上建立组合索引。例如，select * from t1 join t2 on t1.a=t2.a and t1.b=t2.b，可以在t1表上的a，b字段上建立组合索引。
  - where子句的过滤条件字段上（尤其是范围条件）。
  - 在经常出现在order by、group by和distinct后的字段。

  约束限制：

  - 普通表的索引支持最大列数为32列；分区表的GLOBAL索引支持最大列数为31列。
  - 单个索引大小不能超过索引页面大小（8k），其中B-tree、UBtree索引不能超过页面大小的三分之一。
  - 分区表上不支持创建部分索引。
  - 分区表创建GLOBAL索引时，存在以下约束条件：
    - 不支持表达式索引、部分索引
    - 仅支持B-tree索引
  - 在相同属性列上，分区LOCAL索引与GLOBAL索引不能共存。
  - 如果alter语句不带有UPDATE GLOBAL INDEX，那么原有的GLOBAL索引将失效，查询时将使用其他索引进行查询；如果alter语句带有UPDATE GLOBAL INDEX，原有的GLOBAL索引仍然有效，并且索引功能正确。

7.14.67 CREATE LANGUAGE

本版本暂不支持使用该语法。

7.14.68 CREATE MASKING POLICY

功能描述

创建脱敏策略。

注意事项

只有poladmin，sysadmin或初始用户能执行此操作。

需要开启安全策略开关，即设置GUC参数enable_security_policy=on，脱敏策略才可以生效。

须知

在使用DATABASE LINK功能的场景下，客户端发起的DATABASE LINK请求，实际的发送方是服务端，发送端ip地址等相关的属性将是服务端的值。详情见**DATABASE LINK**。

语法格式

```
CREATE MASKING POLICY policy_name masking_clause[, ...] [ policy_filter_clause ] [ENABLE | DISABLE];
```

- masking_clause：
  ```
  masking_function ON LABEL(label_name[, ...])
  ```
- masking_function：

  maskall不是预置函数，硬编码在代码中，不支持\df展示。

  预置时脱敏方式如下：
  ```
  { maskall | randommasking | creditcardmasking | basicemailmasking | fullemailmasking |
  shufflemasking | alldigitsmasking | regexpmasking }
  ```
- policy_filter_clause:
  ```
  FILTER ON { FILTER_TYPE ( filter_value [, ...] ) } [, ...]
  ```
- FILTER_TYPE:
  ```
  IP | APP | ROLES
  ```

参数说明

- **policy_name**

  审计策略名称，需要唯一，不可重复。

  取值范围：字符串，要符合**标识符命名规范**。
- **label_name**

  资源标签名称。
- **masking_clause**

  指出使用何种脱敏函数对被label_name标签标记的数据库资源进行脱敏，支持用schema.function的方式指定脱敏函数。
- **policy_filter**

  指出该脱敏策略对何种身份的用户生效，若为空表示对所有用户生效。
- **FILTER_TYPE**

  描述策略过滤的条件类型，包括IP | APP | ROLES。
- **filter_value**

  指具体过滤信息内容，例如具体的IP，具体的APP名称，具体的用户名。
- **ENABLE|DISABLE**

  可以打开或关闭脱敏策略。若不指定ENABLE|DISABLE，语句默认为ENABLE。

示例

```
--创建dev_mask和bob_mask用户。
gaussdb=# CREATE USER dev_mask PASSWORD '********';
gaussdb=# CREATE USER bob_mask PASSWORD '********';

--创建一个表tb_for_masking。
gaussdb=# CREATE TABLE tb_for_masking(idx int, col1 text, col2 text, col3 text, col4 text, col5 text, col6
```



```
text, col7 text,col8 text);
gaussdb=# INSERT INTO tb_for_masking VALUES(1, '9876543210', 'usr321usr', 'abc@huawei.com',
'abc@huawei.com', '1234-4567-7890-0123', 'abcdef 123456 ui 323 jsfd321 j3k2l3', '4880-9898-4545-2525',
'this is a llt case');
gaussdb=# INSERT INTO tb_for_masking VALUES(2, '0123456789', 'lltc123llt', 'abc@gmail.com',
'abc@gmail.com', '9876-5432-1012-3456', '1234 abcd ef 56 gh78ijk90lm', '4856-7654-1234-9865','this,is.a!
LLT?case');
--创建资源标签标记敏感列。
gaussdb=# CREATE RESOURCE LABEL mask_lb1 ADD COLUMN(tb_for_masking.col1);
gaussdb=# CREATE RESOURCE LABEL mask_lb2 ADD COLUMN(tb_for_masking.col2);
gaussdb=# CREATE RESOURCE LABEL mask_lb3 ADD COLUMN(tb_for_masking.col3);
gaussdb=# CREATE RESOURCE LABEL mask_lb4 ADD COLUMN(tb_for_masking.col4);
gaussdb=# CREATE RESOURCE LABEL mask_lb5 ADD COLUMN(tb_for_masking.col5);
gaussdb=# CREATE RESOURCE LABEL mask_lb6 ADD COLUMN(tb_for_masking.col6);
gaussdb=# CREATE RESOURCE LABEL mask_lb7 ADD COLUMN(tb_for_masking.col7);
gaussdb=# CREATE RESOURCE LABEL mask_lb8 ADD COLUMN(tb_for_masking.col8);

--创建脱敏策略。
gaussdb=# CREATE MASKING POLICY maskpol1 maskall ON LABEL(mask_lb1);
gaussdb=# CREATE MASKING POLICY maskpol2 alldigitsmasking ON LABEL(mask_lb2);
gaussdb=# CREATE MASKING POLICY maskpol3 basicemailmasking ON LABEL(mask_lb3);
gaussdb=# CREATE MASKING POLICY maskpol4 fullemailmasking ON LABEL(mask_lb4);
gaussdb=# CREATE MASKING POLICY maskpol5 creditcardmasking ON LABEL(mask_lb5);
gaussdb=# CREATE MASKING POLICY maskpol6 shufflemasking ON LABEL(mask_lb6);
gaussdb=# CREATE MASKING POLICY maskpol7 regexpmasking('[\d+]','*',2, 9) ON LABEL(mask_lb7);

--创建仅对用户dev_mask和bob_mask,客户端工具为gsql，IP地址为'10.20.30.40', '127.0.0.0/24'场景下生效的脱
敏策略。
gaussdb=# CREATE MASKING POLICY maskpol8 randommasking ON LABEL(mask_lb8) FILTER ON
ROLES(dev_mask, bob_mask), APP(gsql), IP('10.20.30.40', '127.0.0.0/24');
--查看脱敏策略生效
gaussdb=# SELECT * FROM tb_for_masking;
 idx |    col1    |    col2    |      col3      |      col4      |        col5         |               col6                |
col7         |        col8
-----+------------+------------+----------------+----------------+---------------------
+--------------------------------------+---------------------+--------------------
   1 | xxxxxxxxxx | usr000usr  | xxx@huawei.com | xxx@xxxxxx.com | xxxx-xxxx-xxxx-0123 |   s
2iju1bcjk243df333d61l 22 53ef3a | 48**-****-*545-2525 | this is a llt case
   2 | xxxxxxxxxx | lltc000llt | xxx@gmail.com  | xxx@xxxxx.com  | xxxx-xxxx-xxxx-3456 | j 71fem0l286dbia543
g k9 ch          | 48**-****-*234-9865 | this,is.a!LLT?case
(2 rows)
--使用gsql工具,IP地址为'10.20.30.40',用户dev_mask查看tb_for_masking。
gaussdb=# GRANT ALL PRIVILEGES TO dev_mask;
gaussdb=# GRANT ALL PRIVILEGES TO bob_mask;
gaussdb=# SET role dev_mask PASSWORD 'xxxxxxxxxx';
--使用maskpol8脱敏，结果随机，每次不同。
gaussdb=# SELECT col8 FROM tb_for_masking;
        col8
--------------------
 9f1425b3835cc30d99
 9585b4ea8ea8ddcc5b
(2 rows)
gaussdb=# SET role bob_mask PASSWORD 'xxxxxxxxxx';
gaussdb=# SELECT col8 FROM tb_for_masking;
        col8
--------------------
 f29ef3a0769a1f417c
 806aa46409482d838f
(2 rows)

--删除脱敏策略。
gaussdb=# DROP MASKING POLICY maskpol1, maskpol2, maskpol3, maskpol4, maskpol5, maskpol6,
maskpol7, maskpol8;

--删除资源标签。
gaussdb=# DROP RESOURCE LABEL mask_lb1, mask_lb2, mask_lb3, mask_lb4, mask_lb5, mask_lb6,
mask_lb7, mask_lb8;

--删除表tb_for_masking。
```



```
gaussdb=# DROP TABLE tb_for_masking;

--删除用户dev_mask和bob_mask。
gaussdb=# DROP USER dev_mask, bob_mask;
```

相关链接

ALTER MASKING POLICY，DROP MASKING POLICY。

7.14.69 CREATE MATERIALIZED VIEW

CREATE MATERIALIZED VIEW会创建一个全量物化视图，并且后续可以使用REFRESH MATERIALIZED VIEW（全量刷新）刷新物化视图的数据。

CREATE MATERIALIZED VIEW类似于CREATE TABLE AS，不过它会记住被用来初始化该视图的查询， 因此它可以在后续中进行数据刷新。一个物化视图有很多和表相同的属性，但是不支持临时物化视图。

注意事项

- 全量物化视图不可以在临时表或全局临时表上创建。
- 全量物化视图不支持nodegroup。
- 创建全量物化视图后，基表中的绝大多数DDL操作不再支持。
- 不支持对全量物化视图进行IUD操作。
- 全量物化视图创建后，当基表数据发生变化时，需要使用刷新（REFRESH）命令保持物化视图与基表同步。
- Ustore引擎不支持物化视图的创建和使用。

语法格式

```
CREATE MATERIALIZED VIEW mv_name
   [ (column_name [, ...] ) ]
   [ WITH ( {storage_parameter = value} [, ... ] ) ]
     [ TABLESPACE tablespace_name ]
   AS query
   [ WITH DATA ];
```

参数说明

- **mv_name**

  要创建的物化视图的名称（可以被模式限定）。

  取值范围：字符串，要符合标识符命名规范。

- **column_name**

  新物化视图中的一个列名。物化视图支持指定列，指定列需要和后面的查询语句结果的列数量保持一致；如果没有提供列名，会从查询的输出列名中获取列名。

  取值范围：字符串，要符合标识符命名规范。

- **WITH (storage_parameter [= value] [, ...])**

  这个子句为表或索引指定一个可选的存储参数。详见CREATE TABLE。

- **TABLESPACE tablespace_name**

  指定新建物化视图所属表空间。如果没有声明，将使用默认表空间。

- **AS query**

一个SELECT、TABLE 或者VALUES命令。这个查询将在一个安全受限的操作中运行。

示例

```
--创建一个普通表。
gaussdb=# CREATE TABLE my_table (c1 int, c2 int);
WITH(STORAGE_TYPE=ASTORE);

--创建全量物化视图。
gaussdb=# CREATE MATERIALIZED VIEW my_mv AS SELECT * FROM my_table;

--基表写入数据。
gaussdb=# INSERT INTO my_table VALUES(1,1),(2,2);

--对全量物化视图my_mv进行全量刷新。
gaussdb=# REFRESH MATERIALIZED VIEW my_mv;

--删除全量物化视图。
gaussdb=# DROP MATERIALIZED VIEW my_mv;

--删除普通表my_table。
gaussdb=# DROP TABLE my_table;
```

相关链接

ALTER MATERIALIZED VIEW， **CREATE INCREMENTAL MATERIALIZED VIEW**，**CREATE TABLE**，**DROP MATERIALIZED VIEW**，**REFRESH INCREMENTAL MATERIALIZED VIEW**，**REFRESH MATERIALIZED VIEW**

7.14.70 CREATE MODEL

功能描述

训练机器学习模型并保存模型。

注意事项

- 模型名称具有唯一性约束，注意命名格式。
- AI训练时长波动较大，在部分情况下训练运行时间较长，设置的GUC参数statement_timeout时长过短会导致训练中断。建议statement_timeout设置为0，不对语句执行时长进行限制。

语法格式

```
CREATE  MODEL model_name USING architecture_name
FEATURES { {attribute_list} }
[TARGET attribute_name, [,attribute_name]],
FROM ([schema.]table_name | subquery)
WITH (hyper_parameter_name [= {hp_value | DEFAULT}]) [, ...]]
```

参数说明

- **model_name**

  对训练模型进行命名，模型名称具有唯一性约束。

  取值范围：字符串，需要符合**标识符命名规范**。

- **architecture_name**

  训练模型的算法类型。

取值范围：字符型，当前支持：logistic_regression、linear_regression、svm_classification、kmeans。

- **attribute_list**

  枚举训练模型的输入列名。

  取值范围：字符型，需要符合数据属性名的命名规范。

- **attribute_name**

  在监督学习任务中训练模型的目标列名(可进行简单的表达式处理)。

  取值范围：字符型，需要符合数据属性名的命名规范。

- **subquery**

  数据源。

  取值范围：字符串，符合数据库SQL语法。

- **hyper_parameter_name**

  机器学习模型的超参名称。

  取值范围：字符串，针对不同算法超参类型范围不同，取值范围详情请参考《特性指南》的“DB4AI: 数据库驱动AI > 原生DB4AI引擎”章节中“算子支持的超参”表的内容。

- **hp_value**

  超参数值。

  取值范围：字符串，针对不同算法范围不同，取值范围详情请参考《特性指南》的“DB4AI: 数据库驱动AI > 原生DB4AI引擎”章节中“超参的默认值以及取值范围”表的内容。

示例

```
--创建数据表。
gaussdb=# CREATE TABLE houses (
id INTEGER,
tax INTEGER,
bedroom INTEGER,
bath DOUBLE PRECISION,
price INTEGER,
size INTEGER,
lot INTEGER,
mark text
);

--插入训练数据。
gaussdb=# INSERT INTO houses(id, tax, bedroom, bath, price, size, lot, mark) VALUES
(1,590,2,1,50000,770,22100,'a+'),
(2,1050,3,2,85000,1410,12000,'a+'),
(3,20,2,1,22500,1060,3500,'a-'),
(4,870,2,2,90000,1300,17500,'a+'),
(5,1320,3,2,133000,1500,30000,'a+'),
(6,1350,2,1,90500,850,25700,'a-'),
(7,2790,3,2.5,260000,2130,25000,'a+'),
(8,680,2,1,142500,1170,22000,'a-'),
(9,1840,3,2,160000,1500,19000,'a+'),
(10,3680,4,2,240000,2790,20000,'a-'),
(11,1660,3,1,87000,1030,17500,'a+'),
(12,1620,3,2,118500,1250,20000,'a-'),
(13,3100,3,2,140000,1760,38000,'a+'),
(14,2090,2,3,148000,1550,14000,'a-'),
(15,650,3,1.5,65000,1450,12000,'a-');

--训练模型。
gaussdb=# CREATE MODEL price_model USING logistic_regression
```



```
  FEATURES size, lot
  TARGET mark
  FROM HOUSES
  WITH learning_rate=0.88, max_iterations=default;

--删除模型。
gaussdb=# DROP MODEL price_model;

--删除表。
gaussdb=# DROP TABLE houses;
```

相关链接

DROP MODEL，**PREDICT BY**

7.14.71 CREATE OPERATOR

功能描述

定义一个新操作符。

注意事项

CREATE OPERATOR定义一个新的 name操作符。 定义该操作符的用户将成为其所有者。如果给出了一个模式名， 那么该操作符将在指定的模式中创建。否则它会在当前模式中创建。

操作符 name 是一个由下列字符组成的字符串：

+ - * / < > = ~ ! @ # % ^ & | ` ?

选择名字的时候有几个限制：

- --和/*不能在操作符名的任何地方出现， 因为它们会被认为是一个注释的开始。
- 一个多字符的操作符不能以+或-结尾， 除非该名字还包含至少下面字符之一：
  ~ ! @ # % ^ & | ` ?
- => 作为一个操作符名的使用已经废弃了。

操作符!=在输入时映射成<>， 因此这两个名称总是等价的。

至少需要定义一个LEFTARG和RIGHTARG。对于双目操作符来说， 两者都需要定义。对右目操作符来说，只需要定义LEFTARG， 而对于左目操作符来说，只需要定义RIGHTARG。

同样，function_name 过程必须已经用CREATE FUNCTION定义过， 而且必须定义为接受正确数量的指定类型参数(一个或是两个)。

其他子句声明可选的操作符优化子句，含义在**第 35.13 节**里定义。

要想能够创建一个操作符，你必须在参数类型和返回类型上有USAGE权限， 还要在底层函数上有EXECUTE权限。如果指定了交换或者负操作符， 你必须拥有这些操作符。

语法格式

```
CREATE OPERATOR name (
    PROCEDURE = function_name
    [, LEFTARG = left_type ] [, RIGHTARG = right_type ]
    [, COMMUTATOR = com_op ] [, NEGATOR = neg_op ]
```



```
    [, RESTRICT = res_proc ] [, JOIN = join_proc ]
    [, HASHES ] [, MERGES ]
)
```

参数说明

- **name**

  要定义的操作符。可用的字符见上文。其名字可以用模式修饰， 比如CREATE OPERATOR myschema.+ (...)。如果没有模式， 则在当前模式中创建操作符。同一个模式中的两个操作符可以同名， 只要同名的操作符在不同的数据类型间进行操作。这是一个重载过程。

- **function_name**

  用于实现该操作符的函数。

- **left_type**

  操作符左边的参数数据类型，如果存在的话。如果是左目操作符，这个参数可以省略。

- **right_type**

  操作符右边的参数数据类型，如果存在的话。如果是右目操作符，这个参数可以省略。

- **com_op**

  该操作符对应的交换操作符。

- **neg_op**

  该操作符对应的负操作符。

- **res_proc**

  此操作符约束选择性评估函数。

- **join_proc**

  此操作符连接选择性评估函数。

- **HASHES**

  表明此操作符支持 Hash 连接。

- **MERGES**

  表明此操作符可以支持一个融合连接。

  使用OPERATOR()语法在com_op 或者其它可选参数里给出一个模式修饰的操作符名，比如：

  ```
  COMMUTATOR = OPERATOR(myschema.===) ,
  ```

示例

下面命令定义一个新操作符：面积相等，用于box数据类型。

```
gaussdb=# CREATE OPERATOR === (
    LEFTARG = box,
    RIGHTARG = box,
    PROCEDURE = area_equal_procedure,
    COMMUTATOR = ===,
    NEGATOR = !==,
    RESTRICT = area_restriction_procedure,
    JOIN = area_join_procedure,
    HASHES, MERGES
);
```


7.14.72 CREATE OPERATOR CLASS

功能描述

定义一个新的操作符类。该功能为内部使用功能，不建议用户使用。

注意事项

- CREATE OPERATOR CLASS定义一个新的操作符类。一个操作符类定义一种特定的数据类型如何与一种索引一起使用。操作符类声明特定的操作符可以为这种数据类型以及索引方法提供特定的角色或者“策略”。当索引列选择定义的操作符类时，操作符类还声明索引方法使用支持的程序。所有操作符类使用的函数和操作符都必须在创建操作符类之前定义。
- 如果指定了模式，那么操作符类就在指定的模式中创建。否则就在当前模式中创建。在同一个模式中的两个操作符类可以有同样的名字，但它们必须用于不同的索引方法。
- 定义操作符类的用户将成为其所有者。目前，创建用户必须是初始用户。
- CREATE OPERATOR CLASS既不检查这个类定义是否包含所有索引方法需要的操作符以及函数，也不检查这些操作符和函数是否形成一个自包含的集合。
- 相关的操作符类可以集合成操作符族。添加一个新的操作符类到一个已经存在的操作符族可以在CREATE OPERATOR CLASS中指定FAMILY选项。没有这个选项时，新建的类会放置到与它同名的族中（如果不存在则创建它）。

语法格式

```
CREATE OPERATOR CLASS
name [ DEFAULT ] FOR TYPE data_type
USING index_method [
FAMILY family_name ] AS
{  OPERATOR strategy_number operator_name [ (
op_type, op_type ) ] [ FOR SEARCH | FOR ORDER BY sort_family_name ]
| FUNCTION
support_number [ ( op_type [ , op_type ] ) ] function_name ( argument_type [,
...] )
| STORAGE storage_type
} [, ... ]
```

参数说明

- **name**

  将要创建的操作符类的名字（可以用模式修饰）。

- **default**

  如果存在，表示该操作符类将成为它的数据类型的缺省操作符类。对于某个数据类型和访问方式而言，最多有一个操作符类是缺省的。

- **data_type**

  操作符类处理的字段的数据类型。

- **index_method**

  操作符类处理的索引方法的名字。

- **family_name**

  操作符类添加到的现有操作符族的名字。如果没有指定，则使用与该操作符类相同名字的操作符族（如果不存在则创建它）。

- **strategy_number**

  与运算符类关联的索引方法的策略编号。

- **operator_name**

  和该操作符类关联的操作符的名字（可以用模式修饰）。

- **op_type**

  在OPERATOR子句中，表示该操作符的操作数的数据类型，或NONE表示左一元运算符或右一元运算符。在与运算符类的数据类型相同的正常情况下，可以省略操作数数据类型。

  在FUNCTION子句中，如果函数的操作数数据类型和函数的输入数据类型（对于B-tree比较函数和哈希函数）或类的数据类型不同，那么就在该子句中写上这个函数要支持的操作数类型。这些缺省是正确的，因此op_type不需要在FUNCTION子句中指定，除了B-tree排序支持函数支持交叉数据类型比较的情况。

- **sort_family_name**

  描述与排序操作符相关的排序顺序的现有B-tree操作符族的名字。

  缺省时是FOR SEARCH。

- **support_number**

  与运算符类关联的函数的索引方法的编号。

- **function_name**

  运算符类的索引方法的函数名称。

- **argument_type**

  函数参数的数据类型。

- **storage_type**

  实际存储在索引里的数据类型。通常它和字段数据类型相同，但是一些索引方法允许它是不同的。除非索引方法允许使用不同的类型，否则必须省略STORAGE子句。

示例

```
--定义一个函数。
gaussdb=# CREATE FUNCTION func_add_sql(num1 integer, num2 integer) RETURN integer AS BEGIN
RETURN num1 + num2;
END;
/
--新建一个操作符类，将上述函数作为其关联的函数。
gaussdb=# CREATE OPERATOR CLASS oc1 DEFAULT FOR TYPE _int4 USING btree AS FUNCTION 1
func_add_sql (integer, integer);

。
```

7.14.73 CREATE PACKAGE

功能描述

创建一个新的PACKAGE。

注意事项

- 在package specification中声明过的函数或者存储过程，必须在package body中找到定义。

- 在实例化中，无法调用带有commit/rollback的存储过程。
- 创建存储过程时，仅对CREATE的存储过程或PACKAGE本身加写锁，仅对执行过程中编译、执行会对函数和函数依赖的PACKAGE均加读锁。
- 不能在Trigger中调用package函数。
- 不能在外部SQL中直接使用package当中的变量。
- 不允许在package外部调用package的私有变量和存储过程。
- 不支持其它存储过程不支持的用法，例如，在function中不允许调用commit/rollback，则package的function中同样无法调用commit/rollback。
- 不支持schema与package同名。
- 只支持A风格的存储过程和函数定义。
- 不支持package内有同名变量，包括包内同名参数。
- package的全局变量为session级，不同session之间package的变量不共享。
- package中调用自治事务的函数，不允许使用package中的cursor变量，以及递归的使用package中cursor变量的函数。
- package中不支持声明ref cursor变量。
- package默认为SECURITY INVOKER权限，如果想将默认行为改为SECURITY DEFINER权限，需要设置guc参数 behavior_compat_options='plsql_security_definer'。
- 被授予CREATE ANY PACKAGE权限的用户，可以在public模式和用户模式下创建PACKAGE。
- 如果需要创建带有特殊字符的package名，特殊字符中不能含有空格，并且最好设置GUC参数behavior_compat_options="skip_insert_gs_source",否则可能引起报错。
- package创建时依赖未定义对象，如参数 behavior_compat_options='plpgsql_dependency'打开，创建可执行，通过WARNING提示；如参数未打开，package创建不可执行。
- 如package中A风格函数已被视图直接依赖，且参数 behavior_compat_options='plpgsql_dependency'打开，再次创建包体后视图可正常访问；如参数未打开，视图访问失败。
- 创建package函数时，其参数默认值不支持含有变量。
- 包头在PG_OBJECT的类型为'S'（SPECIFICATION简称），包体在PG_OBJECT中的类型为'B'（BODY简称），当创建的PACKAGE对象失效时，可以通过PG_OBJECT中的VALID字段来查找PACKAGE失效对象的OID，并且使用“ALTER PACKAGE PKG_NAME COMPILE”来重新编译PACKAGE，使其有效。
- 在创建package内的函数时，如果函数名字为schema.func或package.func形式，则只会取func的名字，前面的Schema声明或者Package声明无效，如果需要默认禁用这种行为，可以设置guc参数 behavior_compat_options='forbid_package_function_with_prefix'

语法格式

- CREATE PACKAGE SPECIFICATION语法格式。

```
CREATE [ OR REPLACE ] PACKAGE [ schema ] package_name
   [ invoker_rights_clause ] { IS | AS } item_list_1 END package_name;

invoker_rights_clause可以被声明为AUTHID DEFINER或者AUTHID CURRENT_USER，分别为定义者权限和调用者权限。
item_list_1可以为声明的变量或者存储过程以及函数。
```


PACKAGE SPECIFICATION（包头）声明了包内的公有变量、函数、异常等，可以被外部函数或者存储过程调用。在PACKAGE SPECIFICATION中只能声明存储过程，函数，不能定义存储过程或者函数。

- CREATE PACKAGE BODY语法格式。

```
CREATE [ OR REPLACE ] PACKAGE BODY [ schema ] package_name
   { IS | AS } declare_section [ initialize_section ] END package_name;
```

PACKAGE BODY(包体内)定义了包的私有变量，函数等。如果变量或者函数没有在PACKAGE SPECIFICATION中声明过，那么这个变量或者函数则为私有变量或者函数。

PACKAGE BODY也可以声明实例化部分，用来初始化package，详见示例。

示例

- CREATE PACKAGE SPECIFICATION示例：

```
gaussdb=# CREATE OR REPLACE PACKAGE emp_bonus IS
var1 int:=1;--公有变量。
var2 int:=2;
PROCEDURE testpro1(var3 int);--公有存储过程，可以被外部调用。
END emp_bonus;
/
```

- CREATE PACKAGE BODY示例：

```
gaussdb=# drop table if exists test1;
gaussdb=# create or replace package body emp_bonus is
var3 int:=3;
var4 int:=4;
procedure testpro1(var3 int)
is
begin
create table if not exists test1(col1 int);
insert into test1 values(var1);
insert into test1 values(var4);
end;
begin  --实例化开始。
var4:=9;
testpro1(var4);
end emp_bonus;
/
```

- ALTER PACKAGE OWNER示例：

```

--将PACKAGE emp_bonus的所属者改为omm。
gaussdb=# ALTER PACKAGE emp_bonus OWNER TO omm;
```

- 调用PACKAGE示例：

```
--使用call调用package存储过程。
gaussdb=# call emp_bonus.testpro1(1);

--使用select调用package存储过程。
gaussdb=# select emp_bonus.testpro1(1);
--匿名块里调用package存储过程。
gaussdb=# begin
emp_bonus.testpro1(1);
end;
/
```

7.14.74 CREATE PROCEDURE

功能描述

创建一个新的存储过程。

注意事项

- 如果创建存储过程时参数或返回值带有精度，不进行精度检测。
- 创建存储过程时，存储过程定义中对表对象的操作建议都显示指定模式，否则可能会导致存储过程执行异常。
- 创建存储过程时，仅对CREATE的存储过程或PACKAGE本身加写锁，仅对执行过程中编译、执行会对函数和函数依赖的PACKAGE均加读锁。
- 在创建存储过程时，存储过程内部通过SET语句设置current_schema和search_path无效。执行完函数search_path和current_schema与执行函数前的search_path和current_schema保持一致。
- SELECT、CALL调用函数时，必须要在出参位置提供实参进行调用，实参不会发生作用。
- 存储过程指定package属性时支持重载。
- 不能创建仅形参名字不同（存储过程名和参数列表类型都一样）的重载存储过程。
- 重载的存储过程在调用时变量需要明确具体的类型。
- 不能创建与函数拥有相同名称和参数列表的存储过程。
- 不支持仅默认值不同的存储过程重载。
- 存储过程仅IN、OUT、INOUT这三种类型不同的参数，打开GUC参数behavior_compat_options（behavior_compat_options='proc_outparam_override'）后，不允许重载。关闭该参数后，可以重载。
- 在存储过程内部使用未声明的变量，存储过程被调用时会报错。
- 在创建procedure时，不能在avg函数外面嵌套其他agg函数，或者其他系统函数。
- 在存储过程内部调用其它无参数的存储过程时，可以省略括号，直接使用存储过程名进行调用。
- 在存储过程内部调用其他有出参的函数，如果在赋值表达式中调用时，需要打开guc参数 set behavior_compat_options = 'proc_outparam_override' ，并提前定义与出参类型相同的变量，然后将变量作为出参调用带有出参的其他函数，出参才能生效。否则，被调函数的出参会被忽略。
- 在表达式中使用out参数作为出参时，如下情况不会生效，例如：使用execute immediate sqlv using func语法执行函数、使用select func into语法执行函数、使用insert、update等DML语句执行、使用select where a=func()；带out出参的函数，作为入参时，fun（func（out b），a），out出参b未生效等。
- 存储过程支持参数注释的查看与导出、导入。
- 存储过程支持介于IS/AS与plsql_body之间的注释的查看与导出、导入。
- 存储过程默认为SECURITY INVOKER权限，如果想将默认行为改为SECURITY DEFINER权限，需要设置guc参数behavior_compat_options='plsql_security_definer'。
- 被授予CREATE ANY FUNCTION权限的用户，可以在用户模式下创建/替换存储过程。
- out/inout参数必须传入变量，不能够传入常量。
- 集中式环境下，想要调用in参数相同，out参数不同的存储过程，需要设置guc参数behavior_compat_options='proc_outparam_override',并且打开参数后，无论

使用select还是call调用存储过程，都必须加上out参数。打开参数后，不支持使用perform调用存储过程或函数。

- 存储过程创建时依赖未定义对象，如参数behavior_compat_options='plpgsql_dependency'打开，创建可执行，通过WARNING提示；如参数未打开，存储过程创建不可执行。
- 当打开三权分立时，对于定义者权限的存储过程，只能由本用户自己重建。
- 如果将定义者权限的存储过程创建到其他用户Schema下，则会以其他用户的权限执行该存储过程，有越权风险，请谨慎使用。
- 调用带out出参的存储过程，设置GUC参数set behavior_compat_options = 'proc_outparam_transfer_length'后可以传递参数长度。规格限制如下：
  a. 支持的基本类型包括：CHAR(n)、CHARACTER(n)、NCHAR(n)、VARCHAR(n)、VARYING(n)、VARCHAR2(n)、NVARCHAR2(n)类型的参数。
  b. out出参不生效的情况下（比如perform）不需要传递长度。
  c. 不支持精度传递的基本类型包括：NUMERIC、DECIMAL、NUMBER、FLOAT、DEC、INTEGER、TIME、TIMESTAMP、INTERVAL、TIME WITH TIME ZONE、TIMESTAMP WITH TIME ZONE、TIME WITHOUT TIME ZONE、TIMESTAMP WITHOUT TIME ZONE。
  d. GUC参数set behavior_compat_options是否设置为proc_outparam_override时都支持传递参数长度。
  e. 要传递集合类型的元素长度和被集合类型嵌套的数组类型的元素长度需要在GUC参数behavior_compat_options里同时开启tableof_elem_constraints选项。

语法格式

```
CREATE [ OR REPLACE ] PROCEDURE procedure_name
   [ ( {[ argname ] [ argmode ] argtype [ { DEFAULT | := | = } expression ]}[,...]) ]
   [
     { IMMUTABLE | STABLE | VOLATILE }
     | { SHIPPABLE | NOT SHIPPABLE }
     | {PACKAGE}
     | [ NOT ] LEAKPROOF
     | { CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
     | {[ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER | AUTHID DEFINER | AUTHID
CURRENT_USER}
     | COST execution_cost
     | SET configuration_parameter { [ TO | = ] value | FROM CURRENT }
   ][ ... ]
 { IS | AS }
plsql_body
/
```

参数说明

- **OR REPLACE**
  当存在同名的存储过程时，替换原来的定义。
- **procedure_name**
  创建的存储过程名称，可以带有模式名。
  取值范围：字符串，要符合**标识符命名规范**。
- **argmode**
  参数的模式。

须知

VARIADIC用于声明数组类型的参数。

取值范围： IN，OUT，INOUT或VARIADIC。缺省值是IN。只有OUT模式的参数能跟在VARIADIC参数之后。

- **argname**

  参数的名称。

  取值范围：字符串，要符合**标识符命名规范**。

- **argtype**

  参数的数据类型。可以使用%TYPE或%ROWTYPE间接引用变量或表的类型，详细可参考存储过程章节**定义变量**。

  取值范围：可用的数据类型。

  说明

  PACKAGE外PROCEDURE argtype中%TYPE不支持引用PACKAGE变量的类型。

- **expression**

  参数的默认表达式。

  说明

  - 在参数a_format_version值为10c和a_format_dev_version值为s2的情况下，参数为INOUT模式时不支持默认表达式。
  - 推荐使用方式：将所有默认值参数定义在所有非默认值参数后。
  - 调用带有默认参数的函数时，入参从左往右排入函数，如果有非默认参数的入参缺失则报错。
  - 打开proc_uncheck_default_param参数，调用带有默认参数的函数时，入参从左往右排入函数，如果有非默认参数的入参缺失，则会用错位的默认值填充该参数。
  - 在参数a_format_version值为10c、a_format_dev_version值为s1和关闭proc_outparam_override，函数参数同时包括out出参和default时，默认值不可缺省。

- **configuration_parameter**

  - **value**

    把指定的配置参数设置为给定的值。如果value是DEFAULT，则在新的会话中使用系统的缺省设置。OFF关闭设置。

    取值范围：字符串

    - DEFAULT
    - OFF
    - 指定默认值。

  - **from current**

    取当前会话中的值设置为configuration_parameter的值。

- **IMMUTABLE、STABLE等**

  行为约束可选项。各参数的功能与CREATE FUNCTION类似，详细说明见**CREATE FUNCTION**

- **plsql_body**

PL/SQL存储过程体。

须知

当在存储过程体中进行创建用户、修改密码或加解密等涉及密码或密钥相关操作时，系统表及日志中会记录密码或密钥的明文信息。为防止敏感信息泄露，不建议用户在存储过程体中进行涉及密码或密钥等敏感信息的相关操作。

说明

argname和argmode的顺序没有严格要求，推荐按照argname、argmode、argtype的顺序使用。

相关链接

DROP PROCEDURE

优化建议

- analyse | analyze
  - 不支持在事务或匿名块中执行analyze 。
  - 不支持在函数或存储过程中执行analyze操作。

7.14.75 CREATE RESOURCE LABEL

功能描述

创建资源标签。

注意事项

只有poladmin，sysadmin或初始用户能正常执行此操作。

语法格式

```
CREATE RESOURCE LABEL [IF NOT EXISTS] label_name ADD label_item_list[, ...];
```

- label_item_list：
  ```
  resource_type(resource_path[, ...])
  ```
- resource_type：
  ```
  { TABLE | COLUMN | SCHEMA | VIEW | FUNCTION }
  ```

参数说明

- **IF NOT EXISTS**

  如果已经存在相同名称的资源标签，不会抛出错误，而是发出一个通知，告知此资源标签已存在。
- **label_name**

  资源标签名称，创建时要求不能与已有标签重名。

  取值范围：字符串，要符合**标识符命名规范**。
- **resource_type**

指的是要标记的数据库资源的类型。

取值范围：表（TABLE）、列（COLUMN）、模式（SCHEMA）、视图（VIEW）、函数（FUNCTION）。

- **resource_path**

  指的是描述具体的数据库资源的路径。

示例

```
--创建一个表tb_for_label。
gaussdb=# CREATE TABLE tb_for_label(col1 text, col2 text, col3 text);

--创建一个模式schema_for_label。
gaussdb=# CREATE SCHEMA schema_for_label;

--创建一个视图view_for_label。
gaussdb=# CREATE VIEW view_for_label AS SELECT 1;

--创建一个函数func_for_label。
gaussdb=# CREATE FUNCTION func_for_label RETURNS TEXT AS $$ SELECT col1 FROM tb_for_label; $$
LANGUAGE SQL;

--基于表创建资源标签。
gaussdb=# CREATE RESOURCE LABEL IF NOT EXISTS table_label add TABLE(public.tb_for_label);

--基于列创建资源标签。
gaussdb=# CREATE RESOURCE LABEL IF NOT EXISTS column_label add COLUMN(public.tb_for_label.col1);

--基于模式创建资源标签。
gaussdb=# CREATE RESOURCE LABEL IF NOT EXISTS schema_label add SCHEMA(schema_for_label);

--基于视图创建资源标签。
gaussdb=# CREATE RESOURCE LABEL IF NOT EXISTS view_label add VIEW(view_for_label);

--基于函数创建资源标签。
gaussdb=# CREATE RESOURCE LABEL IF NOT EXISTS func_label add FUNCTION(func_for_label);

--删除资源标签。
gaussdb=# DROP RESOURCE LABEL func_label, view_label, schema_label, column_label, table_label;

--删除函数func_for_label。
gaussdb=# DROP FUNCTION func_for_label;

--删除视图view_for_label。
gaussdb=# DROP VIEW view_for_label;

--删除模式schema_for_label。
gaussdb=# DROP SCHEMA schema_for_label;

--删除表tb_for_label。
gaussdb=# DROP TABLE tb_for_label;
```

相关链接

ALTER RESOURCE LABEL，**DROP RESOURCE LABEL**。

7.14.76 CREATE RESOURCE POOL

功能描述

创建一个资源池，并指定此资源池相关联的控制组。

注意事项

只要用户对当前数据库有CREATE权限，就可以创建资源池。

语法格式

```
CREATE RESOURCE POOL pool_name
    [WITH (CONTROL_GROUP="group_name",{MEM_PERCENT=pct| ACTIVE_STATEMENTS=stmt | MAX_DOP
= dop | MEMORY_LIMIT='memory_size' | io_limits=io_limits | io_priority='io_priority' |
nodegroup="nodegroupname" | is_foreign=boolean }[, ... ])];
```

参数说明

- **pool_name**

  资源池名称。

  资源池名称不能和当前数据库里已有的资源池重名。

  取值范围：字符串，要符合**标识符命名规范**。

- **group_name**

  控制组名称。

  说明

  - 设置控制组名称时，语法可以使用双引号，也可以使用单引号。
  - group_name对大小写敏感。
  - 若数据库管理员指定自定义Class组下的Workload控制组，如control_group的字符串为："class1:workload1"；代表此资源池指定到class1控制组下的workload1控制组。也可同时指定Workload控制组的层次，如control_group的字符串为："class1:workload1:1"。
  - 若数据库用户指定Timeshare控制组代表的字符串，即"Rush"、"High"、"Medium"或"Low"其中一种，如control_group的字符串为"High"；代表资源池指定到DefaultClass控制组下的"High" Timeshare控制组。
  - 创建资源池的时候必须要指定所关联的控制组，且创建的资源池不允许被关联到默认控制组（DefaultClass控制组下的"Medium" Timeshare控制组）。

  取值范围：字符串，要符合说明中的规则，其指定已创建的控制组。

- **stmt**

  资源池语句执行的最大并发数量。

  取值范围：数值型，-1~2147483647。

- **dop**

  资源池最大并发度，语句执行时能够创建的最多线程数量。

  取值范围：数值型，1~2147483647

- **memory_size**

  资源池最大使用内存。

  取值范围：字符串，内容范围1KB~2047GB

- **mem_percent**

  资源池可用内存占全部内存或者组用户内存使用的比例。

  在多租户场景下，组用户和业务用户的mem_percent范围1-100，默认为20。

  在普通场景下，普通用户的mem_percent范围为0-100，默认值为0。

> **说明**
>
> mem_percent和memory_limit同时指定时，只有mem_percent起作用。

- **io_limits**

  资源池每秒可触发IO次数上限。

  以万次为单位计数。

- **io_priority**

  IO利用率高达90%时，重消耗IO作业进行IO资源管控时关联的优先级等级。

  包括三档可选：Low、Medium和High。不控制时可设置为None。默认为None。

  > **说明**
  >
  > io_limits和io_priority的设置都仅对复杂作业有效。包括批量导入（INSERT INTO SELECT，COPY FROM，CREATE TABLE AS等），单DN数据量大约超过500MB的复杂查询和VACUUM FULL等操作。

- **nodegroup**

  在逻辑数据库实例模式下，指定逻辑数据库实例名称。必须是存在的逻辑数据库实例。

  如果逻辑数据库实例名称包含大写字符、特殊符号或以数字开头，SQL语句中对逻辑数据库实例名称需要加双引号。

- **is_foreign**

  在逻辑数据库实例模式下，指定当前资源池用于控制没有关联本逻辑数据库实例的普通用户的资源。这里的逻辑数据库实例是由资源池nodegroup字段指定的。

  > **说明**
  >
  > - nodegroup必须是存在的逻辑数据库实例，不能是elastic_group和安装的nodegroup（group_version1）。
  > - 如果指定了is_foreign为true，则资源池不能再关联用户，即不允许通过CREATE USER ... RESOURCE POOL语句来将该资源池配置给用户。该资源池自动检查用户是否关联到资源池指定的逻辑数据库实例，如果用户没有关联到该逻辑数据库实例，则这些用户在逻辑数据库实例所包含的数据库节点上运行将受到该资源池的资源控制。

- **max_workers**

  只用于扩容的接口，表示扩容数据重分布时，表内插入并发度。

- **max_connections**

  最大连接数，用来限制资源池可使用的最大连接数。

  > **说明**
  >
  > 所有资源池的最大连接数加起来不能超过整个gaussdb进程设置的guc参数max_connections指定的最大连接数。

示例

本示例假定用户已预先成功创建控制组（创建控制组请联系管理员处理）。

```
--创建一个资源池，其控制组为"class1"组下属的"Medium" Timeshare Workload控制组。
gaussdb=# CREATE RESOURCE POOL pool1  WITH (CONTROL_GROUP="class1:Medium");

--创建一个资源池，其控制组指定为"DefaultClass"组下属的"High" Timeshare Workload控制组。
gaussdb=# CREATE RESOURCE POOL pool2 WITH (CONTROL_GROUP="High");

--创建一个资源池，其控制组指定为"class1"组下属的"Low" Timeshare Workload控制组。
```



```
gaussdb=# CREATE RESOURCE POOL pool3 WITH (CONTROL_GROUP="class1:Low");

--创建一个资源池，其控制组指定为"class1"组下属的"wg1" Workload控制组。
gaussdb=# CREATE RESOURCE POOL pool4 WITH (CONTROL_GROUP="class1:wg1");

--创建一个资源池，其控制组指定为"class1"组下属的"wg2" Workload控制组。
gaussdb=# CREATE RESOURCE POOL pool5 WITH (CONTROL_GROUP="class1:wg2:3");

--删除资源池。
gaussdb=# DROP RESOURCE POOL pool1;
gaussdb=# DROP RESOURCE POOL pool2;
gaussdb=# DROP RESOURCE POOL pool3;
gaussdb=# DROP RESOURCE POOL pool4;
gaussdb=# DROP RESOURCE POOL pool5;
```

相关链接

ALTER RESOURCE POOL，**DROP RESOURCE POOL**

7.14.77 CREATE ROLE

功能描述

创建角色。

角色是拥有数据库对象和权限的实体。在不同的环境中角色可以认为是一个用户，一个组或者兼顾两者。

注意事项

- 在数据库中添加一个新角色，角色无登录权限。
- 创建角色的用户必须具备CREATE ROLE的权限或者是系统管理员。

语法格式

```
CREATE ROLE role_name [ [ WITH ] option [ ... ] ] [ ENCRYPTED | UNENCRYPTED ] { PASSWORD |
IDENTIFIED BY } { 'password' [EXPIRED] | DISABLE };
```

其中角色信息设置子句option语法为：

```
{SYSADMIN | NOSYSADMIN}
   | {MONADMIN | NOMONADMIN}
   | {OPRADMIN | NOOPRADMIN}
   | {POLADMIN | NOPOLADMIN}
   | {AUDITADMIN | NOAUDITADMIN}
   | {CREATEDB | NOCREATEDB}
   | {USEFT | NOUSEFT}
   | {CREATEROLE | NOCREATEROLE}
   | {INHERIT | NOINHERIT}
   | {LOGIN | NOLOGIN}
   | {REPLICATION | NOREPLICATION}
   | {VCADMIN | NOVCADMIN}
   | {PERSISTENCE | NOPERSISTENCE}
   | CONNECTION LIMIT connlimit
   | VALID BEGIN 'timestamp'
   | VALID UNTIL 'timestamp'
   | RESOURCE POOL 'respool'
   | USER GROUP 'groupuser'
   | PERM SPACE 'spacelimit'
   | TEMP SPACE 'tmpspacelimit'
   | SPILL SPACE 'spillspacelimit'
   | NODE GROUP logic_cluster_name
   | IN ROLE role_name [, ...]
   | IN GROUP role_name [, ...]
```



```
| ROLE role_name [, ...]
| ADMIN rol e_name [, ...]
| USER role_name [, ...]
| SYSID uid
| DEFAULT TABLESPACE tablespace_name
| PROFILE DEFAULT
| PROFILE profile_name
| PGUSER
```

参数说明

- **role_name**

  角色名称。

  取值范围：字符串，要符合**标识符命名规范**，且最多为63个字符。若超过63个字符，数据库会截断并保留前63个字符当做角色名称。在创建角色时，数据库的时候会给出提示信息。

  说明

  标识符需要为字母、下划线、数字（0-9）或美元符号（$），且必须以字母（a-z）或下划线（_）开头。

- **password**

  登录密码。

  密码规则如下：

  - 密码默认不少于8个字符。
  - 不能与用户名及用户名倒序相同。
  - 至少包含大写字母（A-Z），小写字母（a-z），数字（0-9），非字母数字字符（限定为~!@#$%^&*()-_=+\|[{}];:,<.>/?）四类字符中的三类字符。
  - 密码也可以是符合格式要求的密文字符串，这种情况主要用于用户数据导入场景，不推荐用户直接使用。如果直接使用密文密码，用户需要知道密文密码对应的明文，并且保证明文密码复杂度，数据库不会校验密文密码复杂度，直接使用密文密码的安全性由用户保证。
  - 创建角色时，应当使用单引号将用户密码括起来。

  取值范围：不为空的字符串。

- **EXPIRED**

  在创建用户时可指定EXPIRED参数，即创建密码失效用户，该用户不允许执行简单查询和扩展查询。只有在修改自身密码后才可正常执行语句。

- DISABLE

  默认情况下，用户可以更改自己的密码，除非密码被禁用。要禁用用户的密码，请指定DISABLE。禁用某个用户的密码后，将从系统中删除该密码，此类用户只能通过外部认证来连接数据库，例如：kerberos认证。只有管理员才能启用或禁用密码。普通用户不能禁用初始用户的密码。要启用密码，请运行ALTER USER并指定密码。

- **ENCRYPTED | UNENCRYPTED**

  控制密码存储在系统表里的口令是否加密。按照产品安全要求，密码必须加密存储，所以，UNENCRYPTED在GaussDB中禁止使用。因为系统无法对指定的加密口令字符串进行解密，所以如果目前的口令字符串已经用SHA256加密的格式，则会继续照此存放，而不管是否声明了ENCRYPTED或UNENCRYPTED。这样就允许在dump/restore的时候重新加载加密的口令。

- **SYSADMIN | NOSYSADMIN**

决定一个新角色是否为“系统管理员”，具有SYSADMIN属性的角色拥有系统最高权限。

缺省为NOSYSADMIN。

三权分立关闭时，具有SYSADMIN属性的用户有权限创建具有SYSADMIN、REPLICATION、CREATEROLE、AUDITADMIN、MONADMIN、POLADMIN、CREATEDB属性的用户和普通用户。

三权分立打开时，具有SYSADMIN属性的用户无权创建用户。

- **MONADMIN | NOMONADMIN**

  定义角色是否是监控管理员。

  缺省为NOMONADMIN。

- **OPRADMIN | NOOPRADMIN**

  定义角色是否是运维管理员。

  缺省为NOOPRADMIN。

- **POLADMIN | NOPOLADMIN**

  定义角色是否是安全策略管理员。

  缺省为NOPOLADMIN。

- **AUDITADMIN | NOAUDITADMIN**

  定义角色是否有审计管理属性。

  缺省为NOAUDITADMIN。

- **CREATEDB | NOCREATEDB**

  决定一个新角色是否能创建数据库。

  新角色没有创建数据库的权限。

  缺省为NOCREATEDB。

- **USEFT | NOUSEFT**

  该参数为保留参数，暂未启用。

- **CREATEROLE | NOCREATEROLE**

  决定一个角色是否可以创建新角色（也就是执行CREATE ROLE和CREATE USER）。 一个拥有CREATEROLE权限的角色也可以修改和删除其他角色。

  缺省为NOCREATEROLE。

  三权分立关闭时，具有CREATEROLE属性的用户有权限创建具有CREATEROLE、AUDITADMIN、MONADMIN、POLADMIN、CREATEDB属性的用户和普通用户。

  三权分立打开时，具有CREATEROLE属性的用户有权限创建具有CREATEROLE、MONADMIN、POLADMIN、CREATEDB属性的用户和普通用户。

- **INHERIT | NOINHERIT**

  这些子句决定一个角色是否“继承”它所在组的角色的权限。不推荐使用。

- **LOGIN | NOLOGIN**

  具有LOGIN属性的角色才可以登录数据库。一个拥有LOGIN属性的角色可以认为是一个用户。

  缺省为NOLOGIN。

- **REPLICATION | NOREPLICATION**

  定义角色是否允许流复制或设置系统为备份模式。REPLICATION属性是特定的角色，仅用于复制。

缺省为NOREPLICATION。

- **VCADMIN | NOVCADMIN**

  该版本没有实际意义。

- **PERSISTENCE | NOPERSISTENCE**

  定义永久用户。仅允许初始用户创建、修改和删除具有PERSISTENCE属性的永久用户。

- **CONNECTION LIMIT**

  声明该角色可以使用的并发连接数量。

  须知

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

  取值范围：[-1, 2^31-1]的整数。缺省值为-1，表示没有限制。

- **VALID BEGIN**

  设置角色生效的时间戳。如果省略了该子句，角色无有效开始时间限制，timestamp为生效时间，格式为'YYYY-MM-DD HH:mm:ss'。

- **VALID UNTIL**

  设置角色失效的时间戳。如果省略了该子句，角色无有效结束时间限制，timestamp为失效时间，格式为'YYYY-MM-DD HH:mm:ss'。

- **RESOURCE POOL**

  设置角色使用的resource pool名称，该名称属于系统表：pg_resource_pool。

- **USER GROUP**

  创建一个user的子用户。当前版本暂不支持。

- **PERM SPACE**

  设置用户使用空间的大小。

- **TEMP SPACE**

  设置用户临时表存储空间限额。

- **SPILL SPACE**

  设置用户算子落盘空间限额。

- **IN ROLE**

  新角色立即拥有IN ROLE子句中列出的一个或多个现有角色拥有的权限。不推荐使用。

- **IN GROUP**

  IN GROUP是IN ROLE过时的拼法。不推荐使用。

- **ROLE**

  ROLE子句列出一个或多个现有的角色，它们将自动添加为这个新角色的成员，拥有新角色所有的权限。

- **ADMIN**

ADMIN子句类似ROLE子句，不同的是ADMIN后的角色可以把新角色的权限赋给其他角色。

- **USER**

  USER子句是ROLE子句过时的拼法。

- **SYSID**

  SYSID子句将被忽略，无实际意义。

- **DEFAULT TABLESPACE**

  DEFAULT TABLESPACE子句将被忽略，无实际意义。

- **PROFILE**

  PROFILE子句将被忽略，无实际意义。

- **PGUSER**

  当前版本该属性没有实际意义，仅为了语法的前向兼容而保留。

示例

```
--创建一个角色，名为manager，密码为********。
gaussdb=# CREATE ROLE manager IDENTIFIED BY '********';

--创建一个角色，从2015年1月1日开始生效，到2026年1月1日失效。
gaussdb=# CREATE ROLE miriam WITH LOGIN PASSWORD '********' VALID BEGIN '2015-01-01' VALID UNTIL
'2026-01-01';

--修改角色manager的密码为**********。
gaussdb=# ALTER ROLE manager IDENTIFIED BY '**********' REPLACE '********';

--修改角色manager为系统管理员。
gaussdb=# ALTER ROLE manager SYSADMIN;

--删除角色manager。
gaussdb=# DROP ROLE manager;

--删除角色miriam。
gaussdb=# DROP GROUP miriam;
```

相关链接

SET ROLE，**ALTER ROLE**，**DROP ROLE**，**GRANT**

7.14.78 CREATE ROW LEVEL SECURITY POLICY

功能描述

对表创建行访问控制策略。

当对表创建了行访问控制策略，只有打开该表的行访问控制开关(ALTER TABLE ... ENABLE ROW LEVEL SECURITY)，策略才能生效。否则不生效。

当前行访问控制影响数据表的读取操作(SELECT、UPDATE、DELETE)，暂不影响数据表的写入操作(INSERT、MERGE INTO)。表所有者或系统管理员可以在USING子句中创建表达式，在客户端执行数据表读取操作时，数据库后台在查询重写阶段会将满足条件的表达式拼接并应用到执行计划中。针对数据表的每一条元组，当USING表达式返回TRUE时，元组对当前用户可见，当USING表达式返回FALSE或NULL时，元组对当前用户不可见。

行访问控制策略名称是针对表的，同一个数据表上不能有同名的行访问控制策略；对不同的数据表，可以有同名的行访问控制策略。

行访问控制策略可以应用到指定的操作(SELECT、UPDATE、DELETE、ALL)，ALL表示会影响SELECT、UPDATE、DELETE三种操作；定义行访问控制策略时，若未指定受影响的相关操作，默认为ALL。

行访问控制策略可以应用到指定的用户(角色)，也可应用到全部用户(PUBLIC)；定义行访问控制策略时，若未指定受影响的用户，默认为PUBLIC。

注意事项

- 支持对行存表、行存分区表、unlogged表、hash表定义行访问控制策略。
- 不支持外表、本地临时表定义行访问控制策略。
- 不支持对视图定义行访问控制策略。
- 同一张表上可以创建多个行访问控制策略，一张表最多创建100个行访问控制策略。
- 系统管理员不受行访问控制影响，可以查看表的全量数据。
- 通过SQL语句、视图、函数、存储过程查询包含行访问控制策略的表，都会受影响。
- 不支持对添加了行级访问控制策略的表字段进行修改数据类型操作。

语法格式

```
CREATE [ ROW LEVEL SECURITY ] POLICY policy_name ON table_name
    [ AS { PERMISSIVE | RESTRICTIVE } ]
    [ FOR { ALL | SELECT | UPDATE | DELETE } ]
    [ TO { role_name | PUBLIC | CURRENT_USER | SESSION_USER } [, ...] ]
    USING ( using_expression )
```

参数说明

- **policy_name**

  行访问控制策略名称，同一个数据表上行访问控制策略名称不能相同。

- **table_name**

  行访问控制策略的表名。

- **PERMISSIVE | RESTRICTIVE**

  PERMISSIVE指定行访问控制策略为宽容性策略，宽容性策略的条件用OR表达式拼接。

  RESTRICTIVE指定行访问控制策略为限制性策略，限制性策略的条件用AND表达式拼接。拼接方式如下：

  ```
  (using_expression_permissive_1 OR using_expression_permissive_2 ...) AND
  (using_expression_restrictive_1 AND using_expression_restrictive_2 ...)
  ```

  缺省值为PERMISSIVE。

- **command**

  当前行访问控制影响的SQL操作，可指定操作包括：ALL、SELECT、UPDATE、DELETE。当未指定时，ALL为默认值，涵盖SELECT、UPDATE、DELETE操作。

  当command为SELECT时，SELECT类操作受行访问控制的影响，只能查看到满足条件(using_expression返回值为TRUE)的元组数据，受影响的操作包括SELECT，SELECT FOR UPDATE/SHARE，UPDATE ... RETURNING，DELETE ... RETURNING。

当command为UPDATE时，UPDATE类操作受行访问控制的影响，只能更新满足条件(using_expression返回值为TRUE)的元组数据，受影响的操作包括UPDATE, UPDATE ... RETURNING, SELECT ... FOR UPDATE/SHARE。

当command为DELETE时，DELETE类操作受行访问控制的影响，只能删除满足条件(using_expression返回值为TRUE)的元组数据，受影响的操作包括DELETE, DELETE ... RETURNING。

行访问控制策略与适配的SQL语法关系如表7-145所示：

表 7-145 ROW LEVEL SECURITY 策略与适配 SQL 语法关系

Command	SELECT/ALL policy	UPDATE/ALL policy	DELETE/ALL policy
SELECT	Existing row	No	No
SELECT FOR UPDATE/SHARE	Existing row	Existing row	No
UPDATE	No	Existing row	No
UPDATE RETURNING	Existing row	Existing row	No
DELETE	No	No	Existing row
DELETE RETURNING	Existing row	No	Existing row

- **role_name**

  行访问控制影响的数据库用户。

  CURRENT_USER表示当前执行环境的用户名；SESSION_USER则表示会话用户名；当未指定时，PUBLIC为默认值，PUBLIC表示影响所有数据库用户，可以指定多个受影响的数据库用户。

  须知

  系统管理员不受行访问控制特性影响。

- **using_expression**

  行访问控制的表达式（返回boolean值）。

  条件表达式中不能包含AGG函数和窗口（WINDOW）函数。在查询重写阶段，如果数据表的行访问控制开关打开，满足条件的表达式会添加到计划树中。针对数据表的每条元组，会进行表达式计算，只有表达式返回值为TRUE时，行数据对用户才可见（SELECT、UPDATE、DELETE）；当表达式返回FALSE时，该元组对当前用户不可见，用户无法通过SELECT语句查看此元组，无法通过UPDATE语句更新此元组，无法通过DELETE语句删除此元组。

示例

```
--创建用户alice。
gaussdb=# CREATE USER alice PASSWORD '********';
```

```
--创建用户bob。
gaussdb=# CREATE USER bob PASSWORD '********';

--创建数据表all_data。
gaussdb=# CREATE TABLE public.all_data(id int, role varchar(100), data varchar(100));

--向数据表插入数据。
gaussdb=# INSERT INTO all_data VALUES(1, 'alice', 'alice data');
gaussdb=# INSERT INTO all_data VALUES(2, 'bob', 'bob data');
gaussdb=# INSERT INTO all_data VALUES(3, 'peter', 'peter data');

--将表all_data的读取权限赋予alice和bob用户。
gaussdb=# GRANT SELECT ON all_data TO alice, bob;

--打开行访问控制策略开关。
gaussdb=# ALTER TABLE all_data ENABLE ROW LEVEL SECURITY;

--创建行访问控制策略，当前用户只能查看用户自身的数据。
gaussdb=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role = CURRENT_USER);

--查看表all_data相关信息。
gaussdb=# \d+ all_data
                            Table "public.all_data"
 Column |          Type          | Modifiers | Storage  | Stats target | Description
--------+------------------------+-----------+----------+--------------+-------------
 id     | integer                |           | plain    |              |
 role   | character varying(100) |           | extended |              |
 data   | character varying(100) |           | extended |              |
Row Level Security Policies:
    POLICY "all_data_rls" FOR ALL
      TO public
      USING (((role)::name = "current_user"()))
Has OIDs: no
Options: orientation=row, compression=no, enable_rowsecurity=true

--当前用户执行SELECT操作。
gaussdb=# SELECT * FROM all_data;
 id | role  |    data
----+-------+------------
  1 | alice | alice data
  2 | bob   | bob data
  3 | peter | peter data
(3 rows)

gaussdb=# EXPLAIN(COSTS OFF) SELECT * FROM all_data;
      QUERY PLAN
----------------------
 Seq Scan on all_data
(1 row)

--切换至alice用户执行SELECT操作。
gaussdb=# SELECT * FROM all_data;
 id | role  |    data
----+-------+------------
  1 | alice | alice data
(1 row)

gaussdb=# EXPLAIN(COSTS OFF) SELECT * FROM all_data;
 QUERY PLAN
------------------------------------------------------------------
 Seq Scan on all_data
   Filter: ((role)::name = 'alice'::name)
 Notice: This query is influenced by row level security feature
(3 rows)

--删除行访问控制策略。
gaussdb=# DROP ROW LEVEL SECURITY POLICY all_data_rls ON all_data;

--删除数据表all_data。
```



```
gaussdb=# DROP TABLE public.all_data;

--删除用户alice, bob。
gaussdb=# DROP USER alice, bob;
```

相关链接

DROP ROW LEVEL SECURITY POLICY，ALTER ROW LEVEL SECURITY POLICY

7.14.79 CREATE RULE

功能描述

定义一个新的重写规则。

注意事项

- 为了在表上定义或修改规则，你必须是该表的拥有者。
- 如果在同一个表定义了多个相同类型的规则，则按规则的名称字母顺序触发它们。
- 在视图上用于INSERT、UPDATE、DELETE的规则中可以添加RETURNING子句基于视图的字段返回。如果规则被INSERT RETURNING、UPDATE RETURNING、DELETE RETURNING命令触发，这些子句将用来计算输出结果。如果规则被不带RETURNING的命令触发，那么规则的RETURNING子句将被忽略。目前仅允许无条件的INSTEAD规则包含RETURNING子句，而且在同一个事件内的所有规则中最多只能有一个RETURNING子句。这样就确保只有一个RETURNING子句可以用于计算结果。如果在任何有效规则中都不存在RETURNING子句，该视图上的RETURNING查询将被拒绝。
- 目前，ON SELECT规则必须是无条件的INSTEAD规则并且必须有一个由单独一条SELECT查询组成的动作。因此，一条ON SELECT规则实际上把表变成了一个视图，它的可见内容是由该规则的SELECT命令返回，而不是直接存在该表中的内容（如果有）。

语法格式

```
CREATE [ OR REPLACE ] RULE name AS ON event
    TO table_name [ WHERE condition ]
    DO [ ALSO | INSTEAD ] { NOTHING | command | ( command ; command ... ) }
```

其中event包含以下几种：

```
SELECT
INSERT
DELETE
UPDATE
```

参数说明

- name

  创建的规则名。它必须在同一个表上的所有规则名字中唯一。

  取值范围：符合标识符命名规范的字符串，且最大长度不超过63个字符。
- event

  SELECT、INSERT、UPDATE、DELETE事件之一。
- table_name

规则作用的表或者视图的名字(可以有模式修饰)。

- condition

  返回boolean的SQL条件表达式，决定是否实际执行规则。表达式除了引用NEW和OLD之外不能引用任何表，并且不能有聚合函数。不建议使用int等数值类型作为condition，因为int等数值类型可以隐式转换为bool值（非0值隐式转换为true，0转换为false），可能导致非预期的结果。

- INSTEAD

  INSTEAD指示使用该命令替换初始事件。

- ALSO

  ALSO指示该命令应该在初始事件执行之后执行。如果既没有声明ALSO也没有声明INSTEAD， 那么ALSO为缺省值。

- command

  组成规则动作的命令。有效的命令是SELECT、 INSERT、UPDATE、 DELETE语句之一。

示例

```
CREATE RULE "_RETURN" AS
   ON SELECT TO t1
   DO INSTEAD
      SELECT * FROM t2;
```

须知

- ON SELECT 后指定的规则名必须为"_RETURN"
- 目前，ON SELECT规则必须是INSTEAD SELECT，而且TO所指定的表会被转为视图，这个前提是该表为空且不带有触发器、索引、子表等限制，也即必须为一张初始的空表。因此，一般不建议采用这种写法，而是直接创建视图。

7.14.80 CREATE SCHEMA

功能描述

创建模式。

访问命名对象时可以使用模式名作为前缀进行访问，如果无模式名前缀，则访问当前模式下的命名对象。创建命名对象时也可用模式名作为前缀修饰。

另外，CREATE SCHEMA可以包括在新模式中创建对象的子命令，这些子命令和那些在创建完模式后发出的命令没有任何区别。如果使用了AUTHORIZATION子句，则所有创建的对象都将被该用户所拥有。

注意事项

- 只要用户对当前数据库有CREATE权限，就可以创建模式。
- 系统管理员在普通用户同名schema下创建的对象，所有者为schema的同名用户（非系统管理员）。

语法格式

- 根据指定的名称创建模式。
```
CREATE SCHEMA schema_name
    [ AUTHORIZATION user_name ]  [ schema_element [ ... ] ];
```
- 根据用户名创建模式。
```
CREATE SCHEMA AUTHORIZATION user_name [ schema_element [ ... ] ];
```
- 创建模式并指定默认字符集和字符序。
```
CREATE SCHEMA schema_name
    [ [DEFAULT] CHARACTER SET | CHARSET [ = ] default_charset ] [ [DEFAULT] COLLATE [ = ] default_collation ];
```

参数说明

- **schema_name**

  模式名称。

  须知

  - 模式名不能和当前数据库里其他的模式重名。
  - 模式名不能和当前数据库的初始用户重名。
  - 模式的名称不可以“pg_”开头。
  - 模式的名称不可以“gs_role_”开头。

  取值范围：字符串，要符合**标识符命名规范**。

- **AUTHORIZATION user_name**

  指定模式的所有者。当不指定schema_name时，把user_name当作模式名，此时user_name只能是角色名。

  取值范围：已存在的用户名/角色名。

- **schema_element**

  在模式里创建对象的SQL语句。目前仅支持CREATE TABLE、CREATE VIEW、CREATE INDEX、CREATE PARTITION、CREATE SEQUENCE、CREATE TRIGGER、GRANT子句。

  子命令所创建的对象都被AUTHORIZATION子句指定的用户所拥有。

- **default_charset**

  仅在sql_compatibility='B'时支持该语法。指定模式的默认字符集，单独指定时会将模式的默认字符序设置为指定的字符集的默认字符序。

- **default_collation**

  仅在sql_compatibility='B'时支持该语法。指定模式的默认字符序，单独指定时会将模式的默认字符集设置为指定的字符序对应的字符集。

  支持字符序参见**表1 B模式（即sql_compatibility = 'B'）下支持的字符集和字符序介绍**。

  说明

  如果当前搜索路径上的模式中存在同名对象时，需要明确指定引用对象所在的模式。可以通过命令SHOW SEARCH_PATH来查看当前搜索路径上的模式。

示例

```
--创建一个角色role1。
gaussdb=# CREATE ROLE role1 IDENTIFIED BY '********';

-- 为用户role1创建一个同名schema，子命令创建的表films和winners的拥有者为role1。
gaussdb=# CREATE SCHEMA AUTHORIZATION role1
    CREATE TABLE films (title text, release date, awards text[])
    CREATE VIEW winners AS
    SELECT title, release FROM films WHERE awards IS NOT NULL;

-- 创建一个schema ds，指定schema的默认字符集为utf8mb4，默认字符序为utf8mb4_bin。
gaussdb=# CREATE SCHEMA ds CHARACTER SET utf8mb4 COLLATE utf8mb4_bin;

--删除schema。
gaussdb=# DROP SCHEMA role1 CASCADE;
--删除用户。
gaussdb=# DROP USER role1 CASCADE;
```

相关链接

ALTER SCHEMA，**DROP SCHEMA**

7.14.81 CREATE SEQUENCE

功能描述

CREATE SEQUENCE用于向当前数据库里增加一个新的序列。序列的Owner为创建此序列的用户。

注意事项

- Sequence是一个存放等差数列的特殊表。这个表没有实际意义，通常用于为行或者表生成唯一的标识符。
- 如果给出一个模式名，则该序列就在给定的模式中创建，否则会在当前模式中创建。序列名必须和同一个模式中的其他序列、表、索引、视图或外表的名称不同。
- 创建序列后，在表中使用序列的nextval()函数和generate_series(1,N)函数对表插入数据，请保证nextval的可调用次数大于等于N+1次，否则会因为generate_series()函数会调用N+1次而导致报错。
- Sequence默认最大值为2^63-1，如果使用了Large标识则最大值可以支持到2^127-1。
- 被授予CREATE ANY SEQUENCE权限的用户，可以在public模式和用户模式下创建序列。

语法格式

```
CREATE [ LARGE | TEMPORARY | TEMP ] SEQUENCE name [ INCREMENT [ BY ] increment ]
    [ MINVALUE minvalue | NO MINVALUE | NOMINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE |
NOMAXVALUE]
    [ START [ WITH ] start ] [ CACHE cache ] [ [ NO ] CYCLE | NOCYCLE ]
    [ OWNED BY { table_name.column_name | NONE } ];
```

参数说明

- **TEMPORARY | TEMP**
  临时序列关键字。

说明

- 创建临时序列需要数据库启用PG兼容。
- 临时序列的生命周期是会话级的，临时序列对象是会话隔离的。序列在会话退出后会自动删除清理。
- 临时序列存在于一个特殊的模式中，每个会话有且仅有一个临时模式，若提前创建了临时模式，则在创建临时序列时可给出模式名，若未提前创建，则在创建临时序列时不能给出模式名，并且每个会话只能访问自己的临时模式中的对象，无法访问其他会话的临时模式中的对象。如果访问的临时模式不属于本会话，会报告错误。
- 当临时序列存在时，已有的同名永久序列（在这个会话中）会变得不可见，不过可以用模式限定的名称来引用同名永久序列。

- **name**

  将要创建的序列名称。

  取值范围: 仅可以使用小写字母（a~z）、 大写字母（A~Z），数字和特殊字符"#"，"_"，"$"的组合。

- **increment**

  指定序列的步长。一个正数将生成一个递增的序列，一个负数将生成一个递减的序列。

  缺省值为1。

  说明

  在B兼容模式下，步长为浮点数时会自动转为整型。其他模式下，该参数不支持输入浮点数。

- **MINVALUE minvalue | NO MINVALUE| NOMINVALUE**

  执行序列的最小值。如果没有声明minvalue或者声明了NO MINVALUE，则递增序列的缺省值为1，递减序列的缺省值为$-2^{63}-1$。NOMINVALUE等价于NO MINVALUE

- **MAXVALUE maxvalue | NO MAXVALUE| NOMAXVALUE**

  执行序列的最大值。如果没有声明maxvalue或者声明了NO MAXVALUE，则递增序列的缺省值为$2^{63}-1$，递减序列的缺省值为-1。NOMAXVALUE等价于NO MAXVALUE

- **start**

  指定序列的起始值。缺省值：对于递增序列为minvalue，递减序列为maxvalue。

- **cache**

  为了快速访问，而在内存中预先存储序列号的个数。

  缺省值为1，表示一次只能生成一个值，也就是没有缓存。

  说明

  不建议同时定义cache和maxvalue或minvalue。因为定义cache后不能保证序列的连续性，可能会产生空洞，造成序列号段浪费。

- **CYCLE**

  用于使序列达到maxvalue或者minvalue后可循环并继续下去。

  如果声明了NO CYCLE，则在序列达到其最大值后任何对nextval的调用都会返回一个错误。

  NOCYCLE的作用等价于NO CYCLE。

  缺省值为NO CYCLE。

若定义序列为CYCLE，则不能保证序列的唯一性。

- **OWNED BY**

  将序列和一个表的指定字段进行关联。这样，在删除那个字段或其所在表的时候会自动删除已关联的序列。关联的表和序列的所有者必须是同一个用户，并且在同一个模式中。需要注意的是，通过指定OWNED BY，仅仅是建立了表的对应列和sequence之间关联关系，并不会在插入数据时在该列上产生自增序列。

  缺省值为OWNED BY NONE，表示不存在这样的关联。

须知

通过OWNED BY创建的Sequence不建议用于其他表，如果希望多个表共享Sequence，该Sequence不应该从属于特定表。

示例

创建一个名为test的临时序列：

```
gaussdb=# CREATE TEMPORARY SEQUENCE test;
```

创建一个名为serial的递增序列，从101开始：

```
gaussdb=# CREATE SEQUENCE serial
 START 101
 CACHE 20;
```

从序列中选出下一个数字：

```
gaussdb=# SELECT nextval('serial');
 nextval
 ---------
     101
```

从序列中选出下一个数字：

```
gaussdb=# SELECT nextval('serial');
 nextval
 ---------
     102
```

创建与表关联的序列：

```
gaussdb=# CREATE TABLE customer_address
(
    ca_address_sk             integer               not null,
    ca_address_id             char(16)              not null,
    ca_street_number          char(10)                      ,
    ca_street_name            varchar(60)                   ,
    ca_street_type            char(15)                      ,
    ca_suite_number           char(10)                      ,
    ca_city                   varchar(60)                   ,
    ca_county                 varchar(30)                   ,
    ca_state                  char(2)                       ,
    ca_zip                    char(10)                      ,
    ca_country                varchar(20)                   ,
    ca_gmt_offset             decimal(5,2)                  ,
    ca_location_type          char(20)
);

gaussdb=# CREATE SEQUENCE serial1
 START 101
 CACHE 20
OWNED BY customer_address.ca_address_sk;
```

```
--删除表和序列
gaussdb=# DROP TABLE customer_address;
gaussdb=# DROP SEQUENCE serial cascade;
gaussdb=# DROP SEQUENCE serial1 cascade;
```

相关链接

DROP SEQUENCE，ALTER SEQUENCE

7.14.82 CREATE SERVER

功能描述

定义一个新的外部服务器。

注意事项

OPTIONS中的敏感字段（如password, secret_access_key）在使用多层引号时，语义和不带引号的场景是不同的，因此不会被识别为敏感字段进行脱敏。

语法格式

```
CREATE SERVER server_name
    FOREIGN DATA WRAPPER fdw_name
    OPTIONS ( { option_name ' value ' } [, ...] ) ;
```

参数说明

- **server_name**

  server的名称。

  取值范围：长度必须小于等于63。

- **fdw_name**

  指定外部数据封装器的名称。

  取值范围：dist_fdw，log_fdw，file_fdw。其中log_fdw和file_fdw仅作语法兼容，可以创建外表，无实际使用意义，不做额外使用说明。

- **OPTIONS ({ option_name ' value ' } [, ...])**

  这个子句为服务器指定选项。这些选项通常定义该服务器的连接细节，但是实际的名称和值取决于该服务器的外部数据包装器。

  - 用于指定外部服务器的各类参数，详细的参数说明如下所示。

    - encrypt

      是否对数据进行加密，该参数仅支持type为OBS时设置。默认值为on。

      取值范围：

      - on表示对数据进行加密，使用HTTPS协议通信。
      - off表示不对数据进行加密，使用HTTP协议通信。

    - access_key

      OBS访问协议对应的AK值（OBS云服务界面由用户获取）。该参数仅支持type为OBS时设置。

    - secret_access_key

OBS访问协议对应的SK值（OBS云服务界面由用户获取）。该参数仅支持type为OBS时设置。

除了libpq支持的连接参数外，还额外提供以下参数：

- **fdw_startup_cost**

  执行一个外表扫描时的启动耗时估算。这个值通常包含建立连接、远端对请求的分析和生成计划的耗时。默认值为100。

- **fdw_typle_cost**

  在远端服务器上对每一个元组进行扫描时的额外消耗。这个值通常表示数据在server间传输的额外消耗。默认值为0.01。

示例

```
--创建server。
gaussdb=# create server my_server foreign data wrapper log_fdw;
CREATE SERVER

--删除my_server。
gaussdb=# DROP SERVER my_server;
DROP SERVER
```

相关链接

ALTER SERVER，**DROP SERVER**

7.14.83 CREATE SYNONYM

功能描述

创建一个同义词对象。同义词是数据库对象的别名，用于记录与其他数据库对象名间的映射关系，用户可以使用同义词访问关联的数据库对象。

注意事项

- 定义同义词的用户成为其所有者。
- 若指定模式名称，则同义词在指定模式中创建。否则，在当前模式创建。
- 支持通过同义词访问的数据库对象包括：表、视图、类型、包、函数和存储过程。
- 使用同义词时，用户需要具有对关联对象的相应权限。
- 支持使用同义词的DML语句包括：SELECT、INSERT、UPDATE、DELETE、EXPLAIN、CALL。
- 不建议对临时表创建同义词。如果需要创建的话，需要指定同义词的目标临时表的模式名，否则无法正常使用该同义词，并且在当前会话结束前执行DROP SYNONYM命令。
- 删除原对象后，与之关联同义词不会被级联删除，继续访问该同义词会报错，对于表提示已失效，对于函数、存储过程、包的等会提示对象不存在。
- 被授予了CREATE ANY SYNONYM权限的用户能够在用户模式下创建同义词。
- 不支持针对包含加密列的密态表及基于密态表的视图、函数、存储过程创建同义词。

- 同义词关联的对象可以是package，也可以是package下的函数。可以通过关联package的方式访问package下的函数；不支持同义词关联package下的函数后，通过同义词直接访问package下的函数。
- 同义词的SCHEMA是用户所在SCHEMA时，该同义词OWNER为SCHEMA的OWNER，其他场景同义词OWNER默认为同义词的创建者。
- 设置SEARCH_PATH，未指定同义词SCHEMA情况下，存储过程和函数会优先按照名称检索PG_PROC表，在没有同名函数时，检索同义词，最后按照SEARCH_PATH检索；其他对象优先检索SEARCH_PATH，同SCHEMA下，本名的对象优先于同义词被访问。
- 不支持通过DDL语句CREATE、DROP、ALTER操作同义词的方式访问同义词所关联的对象。

语法格式

```
CREATE [ OR REPLACE ] SYNONYM synonym_name
    FOR object_name;
```

参数说明

- **OR REPLACE**

  可选。如果同义词已存在，则重新定义。

- **synonym_name**

  创建的同义词名字，可以带模式名。

  取值范围：字符串，要符合**标识符命名规范**。

- **object_name**

  关联的对象名字，可以带模式名。

  取值范围：字符串，要符合**标识符命名规范**。

  说明

  - object_name可以是不存在的对象名称。
  - object_name可以是使用DATABASE LINK方式访问远程对象。DATABASE LINK详细使用方式请见**DATABASE LINK**。

注意

避免对包含敏感信息的函数（如加解密类函数gs_encrypt、gs_decrypt等）创建别名并且使用别名调用，防止敏感信息泄露。

示例

```
--创建模式ot。
gaussdb=# CREATE SCHEMA ot;

--创建表ot.t1及其同义词t1。
gaussdb=# CREATE TABLE ot.t1(id int, name varchar2(10));
gaussdb=# CREATE OR REPLACE SYNONYM t1 FOR ot.t1;

--使用同义词t1。
gaussdb=# SELECT * FROM t1;
gaussdb=# INSERT INTO t1 VALUES (1, 'ada'), (2, 'bob');
gaussdb=# UPDATE t1 SET t1.name = 'cici' WHERE t1.id = 2;
```



```
--创建同义词v1及其关联视图ot.v_t1。
gaussdb=# CREATE SYNONYM v1 FOR ot.v_t1;
gaussdb=# CREATE VIEW ot.v_t1 AS SELECT * FROM ot.t1;

--使用同义词v1。
gaussdb=# SELECT * FROM v1;

--创建重载函数ot.add及其同义词add。
gaussdb=# CREATE OR REPLACE FUNCTION ot.add(a integer, b integer) RETURNS integer AS
$$
SELECT $1 + $2
$$
LANGUAGE sql;

gaussdb=# CREATE OR REPLACE FUNCTION ot.add(a decimal(5,2), b decimal(5,2)) RETURNS decimal(5,2)
AS
$$
SELECT $1 + $2
$$
LANGUAGE sql;

gaussdb=# CREATE OR REPLACE SYNONYM add FOR ot.add;

--使用同义词add。
gaussdb=# SELECT add(1,2);
gaussdb=# SELECT add(1.2,2.3);

--创建存储过程ot.register及其同义词register。
gaussdb=# CREATE PROCEDURE ot.register(n_id integer, n_name varchar2(10))
SECURITY INVOKER
AS
BEGIN
    INSERT INTO ot.t1 VALUES(n_id, n_name);
END;
/

gaussdb=# CREATE OR REPLACE SYNONYM register FOR ot.register;

--使用同义词register，调用存储过程。
gaussdb=# CALL register(3,'mia');

--删除同义词。
gaussdb=# DROP SYNONYM t1;
gaussdb=# DROP SYNONYM IF EXISTS v1;
gaussdb=# DROP SYNONYM IF EXISTS add;
gaussdb=# DROP SYNONYM register;
gaussdb=# DROP SCHEMA ot CASCADE;
```

相关链接

ALTER SYNONYM，**DROP SYNONYM**

7.14.84 CREATE TABLE

功能描述

在当前数据库中创建一个新的空白表，该表由命令执行者所有。

注意事项

- 如果在建表过程中数据库系统发生故障，系统恢复后可能无法自动清除之前已创建的、大小为0的磁盘文件。此种情况出现概率小，不影响数据库系统的正常运行。

- 使用JDBC时，支持通过PrepareStatement对DEFAULT值进行参数化设置。
- 被授予CREATE ANY TABLE权限的用户，可以在public模式和用户模式下创建表。如果想要创建包含serial类型列的表，还需要授予CREATE ANY SEQUENCE创建序列的权限。
- XML类型不能作为主键、外键。
- 表约束个数不能超过32767个。

语法格式

创建表。

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE [ IF NOT EXISTS ] table_name
    ({ column_name data_type [ CHARACTER SET | CHARSET charset ] [ compress_mode ] [ COLLATE
collation ] [ column_constraint [ ... ] ]
        | table_constraint
        | LIKE source_table [ like_option [...] ] }
        [, ... ])
    [ AUTO_INCREMENT [ = ] value ]
    [ [DEFAULT] CHARACTER SET | CHARSET [ = ] default_charset ] [ [DEFAULT] COLLATE [ = ]
default_collation ]
    [ WITH ( {storage_parameter = value} [, ... ] ) ]
    [ ON COMMIT { PRESERVE ROWS | DELETE ROWS } ]
    [ COMPRESS | NOCOMPRESS ]
    [ TABLESPACE tablespace_name ];
```

- 其中列约束column_constraint为：

  ```
  [ CONSTRAINT constraint_name ]
  { NOT NULL |
    NULL |
    CHECK ( expression ) |
    DEFAULT default_expr |
    GENERATED ALWAYS AS ( generation_expr ) [STORED] |
    AUTO_INCREMENT |
    UNIQUE [KEY] index_parameters |
    ENCRYPTED WITH ( COLUMN_ENCRYPTION_KEY = column_encryption_key, ENCRYPTION_TYPE =
  encryption_type_value ) |
    PRIMARY KEY index_parameters |
    REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
      [ ON DELETE action ] [ ON UPDATE action ] }
  [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
  ```

- 其中列的压缩可选项compress_mode为：

  ```
  { DELTA | PREFIX | DICTIONARY | NUMSTR | NOCOMPRESS }
  ```

- 其中表约束table_constraint为：

  ```
  [ CONSTRAINT [ constraint_name ] ]
  { CHECK ( expression ) |
    UNIQUE [ index_name ][ USING method ] ( { { column_name | ( expression ) } [ ASC | DESC ] }
  [, ... ] ) index_parameters |
    PRIMARY KEY [ USING method ] ( { column_name [ ASC | DESC ] } [, ... ] ) index_parameters |
    FOREIGN KEY [ index_name ] ( column_name [, ... ] ) REFERENCES reftable [ (refcolumn [, ... ] ) ]
      [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE action ] [ ON UPDATE action ]
   }
  [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
  ```

- 其中like选项like_option为：

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

- 其中索引参数index_parameters为：

  ```
  [ WITH ( {storage_parameter = value} [, ... ] ) ]
  [ USING INDEX TABLESPACE tablespace_name ]
  ```

参数说明

- **UNLOGGED**

如果指定此关键字，则创建的表为非日志表。在非日志表中写入的数据不会被写入到预写日志中，这样就会比普通表快很多。但是非日志表在冲突、执行操作系统重启、数据库重启、主备切换、切断电源操作或异常关机后会被自动截断，会造成数据丢失的风险。非日志表中的内容也不会被复制到备服务器中。在非日志表中创建的索引也不会被自动记录。

使用场景：非日志表不能保证数据的安全性，用户应该在确保数据已经做好备份的前提下使用，例如系统升级时进行数据的备份。

故障处理：当异常关机等操作导致非日志表上的索引发生数据丢失时，用户应该对发生错误的索引进行重建。

- **GLOBAL | LOCAL**

  创建临时表时可以在TEMP或TEMPORARY前指定GLOBAL或LOCAL关键字。如果指定GLOBAL关键字，GaussDB会创建全局临时表，否则GaussDB会创建本地临时表。

- **TEMPORARY | TEMP**

  如果指定TEMP或TEMPORARY关键字，则创建的表为临时表。临时表分为全局临时表和本地临时表两种类型。创建临时表时如果指定GLOBAL关键字则为全局临时表，否则为本地临时表。

  全局临时表的元数据对所有会话可见，会话结束后元数据继续存在。会话与会话之间的用户数据、索引和统计信息相互隔离，每个会话只能看到和更改自己提交的数据。全局临时表有两种模式：一种是基于会话级别的(ON COMMIT PRESERVE ROWS), 当会话结束时自动清空用户数据；一种是基于事务级别的(ON COMMIT DELETE ROWS), 当执行commit或rollback时自动清空用户数据。建表时如果没有指定ON COMMIT选项，则缺省为会话级别。与本地临时表不同，全局临时表建表时可以指定非pg_temp_开头的schema。

  本地临时表只在当前会话可见，本会话结束后会自动删除。因此，在除当前会话连接的数据库节点故障时，仍然可以在当前会话上创建和使用临时表。由于临时表只在当前会话创建，对于涉及对临时表操作的DDL语句，会产生DDL失败的报错。因此，建议DDL语句中不要对临时表进行操作。TEMP和TEMPORARY等价。

须知

- 本地临时表通过每个会话独立的以pg_temp开头的schema来保证只对当前会话可见，因此，不建议用户在日常操作中手动删除以pg_temp、pg_toast_temp开头的schema。
- 如果建表时不指定TEMPORARY/TEMP关键字，而指定表的schema为当前会话的pg_temp_开头的schema，则此表会被创建为临时表。
- ALTER/DROP全局临时表和索引，如果其它会话正在使用它，禁止操作（ALTER INDEX index_name REBUILD除外）。
- 全局临时表的DDL只会影响当前会话的用户数据和索引。例如truncate、reindex、analyze只对当前会话有效。
- 全局临时表功能可以通过设置GUC参数max_active_global_temporary_table控制是否启用。如果max_active_global_temporary_table=0，关闭全局临时表功能。
- 临时表只对当前会话可见，因此不支持与\parallel on并行执行一起使用。
- 临时表不支持主备切换。
- 全局临时表不响应自动清理，在长连接场景使用时尽量使用on commit delete rows的全局临时表，或定期手动执行vacuum，否则可能导致clog日志不回收。
- 全局临时表不支持以下场景：
  - 不支持创建全局临时sequence，各个会话的全局临时表使用共享的sequence，只能保证唯一性，不保证连续性。
  - 不支持创建全局临时视图。
  - 不支持创建分区表。
  - 不支持创建Hash bucket表。
  - 不支持扩展统计信息。
  - 不支持ON COMMIT DROP属性。

- **IF NOT EXISTS**

  如果已经存在相同名称的表，不会报出错误，而会发出通知，告知通知此表已存在。

- **table_name**

  要创建的表名。

须知

物化视图的一些处理逻辑会通过表名的前缀来识别是不是物化视图日志表和物化视图关联表，因此，用户不要创建表名以mlog_或matviewmap_为前缀的表，否则会影响此表的一些功能。

- **column_name**

  新表中要创建的字段名。

- **constraint_name**

  建表时指定的约束名称。

须知

在B模式数据库下（即sql_compatibility = 'B'）constraint_name为可选项，在其他模式数据库下，必须加上constraint_name。

- **index_name**

  索引名。

  须知

  - index_name仅在B模式数据库下（即sql_compatibility = 'B'）支持，其他模式数据库下不支持。
  - 对于外键约束，constraint_name和index_name同时指定时，索引名为constraint_name。
  - 对于唯一键约束，constraint_name和index_name同时指定时，索引名以index_name。

- **USING method**

  指定创建索引的方法。

  取值范围参考**参数说明**中的USING method。

  须知

  - USING method仅在B模式数据库下（即sql_compatibility = 'B'）支持，其他模式数据库下不支持。
  - 在B模式下，未指定USING method时，对于ASTORE的存储方式，默认索引方法为btree；对于USTORE的存储方式，默认索引方法为ubtree。

- **ASC | DESC**

  ASC表示指定按升序排序（默认）。DESC指定按降序排序。

  须知

  ASC|DESC只在B模式数据库下（即sql_compatibility = 'B'）支持，其他模式数据库不支持。

- **expression**

  创建一个基于该表的一个或多个字段的表达式索引约束，必须写在圆括弧中。

  须知

  UNIQUE约束中的表达式索引只在B模式数据库下支持（即sql_compatibility = 'B'），其他模式数据库不支持。

- **data_type**

字段的数据类型。

- **compress_mode**

  表字段的压缩选项。该选项指定表字段优先使用的压缩算法。行存表不支持压缩。

  取值范围：DELTA、PREFIX、DICTIONARY、NUMSTR、NOCOMPRESS

- **CHARACTER SET | CHARSET charset**

  只在B模式数据库下（即sql_compatibility = 'B'）支持该语法，其他模式数据库不支持。指定表字段的字符集，单独指定时会将字段的字符序设置为指定的字符集的默认字符序。

- **COLLATE collation**

  COLLATE子句指定列的排序规则（字符序）（该列必须是可排列的数据类型）。如果没有指定，则使用默认的排序规则。排序规则可以使用“select * from pg_collation;”命令从pg_collation系统表中查询，默认的排序规则为查询结果中以default开始的行。对于B模式数据库下（即sql_compatibility = 'B'）还支持utf8mb4_bin、utf8mb4_general_ci、utf8mb4_unicode_ci、binary字符序。

  说明

  - 仅字符类型支持指定字符集，指定为binary字符集或字符序实际是将字符类型转化为对应的二进制类型，若类型映射不存在则报错。当前仅有TEXT类型转化为BLOB的映射。
  - 除binary字符集和字符序外，当前仅支持指定与数据库编码相同的字符集。
  - 未显式指定字段字符集或字符序时，若指定了表的默认字符集或字符序，字段字符集和字符序将从表上继承。若表的默认字符集或字符序不存在，当b_format_behavior_compat_options = 'default_collation'时，字段的字符集和字符序将继承当前数据库的字符集及其对应的默认字符序。

  表 7-146 B 模式（即 sql_compatibility = 'B'）下支持的字符集和字符序介绍

字符序名称	对应的字符集	描述
utf8mb4_general_ci	utf8mb4（即utf8）	使用通用排序规则，不区分大小写。
utf8mb4_unicode_ci	utf8mb4（即utf8）	使用通用排序规则，不区分大小写。
utf8mb4_bin	utf8mb4（即utf8）	使用二进制排序规则，区分大小写。
binary	binary	使用二进制排序规则。

- **LIKE source_table [like_option ...]**

  LIKE子句声明一个表，新表自动从这个表中继承所有字段名及其数据类型和非空约束。

  新表与源表之间在创建动作完毕之后是完全无关的。在源表做的任何修改都不会传播到新表中，并且也不可能在扫描源表的时候包含新表的数据。

  被复制的列和约束并不使用相同的名称进行融合。如果明确的指定了相同的名称或者在另外一个LIKE子句中，将会报错。

  - 源表上的字段缺省表达式只有在指定INCLUDING DEFAULTS时，才会复制到新表中。缺省是不包含缺省表达式的，即新表中的所有字段的缺省值都是NULL。

- 如果指定了INCLUDING GENERATED，则源表列的生成表达式会复制到新表中。默认不复制生成表达式。
- 源表上的CHECK约束仅在指定INCLUDING CONSTRAINTS时，会复制到新表中，而其他类型的约束永远不会复制到新表中。非空约束总是复制到新表中。此规则同时适用于表约束和列约束。
- 如果指定了INCLUDING INDEXES，则源表上的索引也将在新表上创建，默认不建立索引。
- 如果指定了INCLUDING STORAGE，则源表列的STORAGE设置会复制到新表中，默认情况下不包含STORAGE设置。
- 如果指定了INCLUDING COMMENTS，则源表列、约束和索引的注释会复制到新表中。默认情况下，不复制源表的注释。
- 如果指定了INCLUDING PARTITION，则源表的分区定义会复制到新表中，同时新表将不能再使用PARTITION BY子句。默认情况下，不拷贝源表的分区定义。如果源表上带有索引，可以使用INCLUDING PARTITION INCLUDING INDEXES语法实现。如果对分区表只使用INCLUDING INDEXES，目标表定义将是普通表，但是索引是分区索引，最后结果会报错，因为普通表不支持分区索引。
- 如果指定了INCLUDING RELOPTIONS，则源表的存储参数（即源表的WITH子句）会复制到新表中。默认情况下，不复制源表的存储参数。
- INCLUDING ALL包含了INCLUDING DEFAULTS、INCLUDING GENERATED、INCLUDING CONSTRAINTS、INCLUDING INDEXES、INCLUDING STORAGE、INCLUDING COMMENTS、INCLUDING PARTITION和INCLUDING RELOPTIONS的内容。

须知

- 如果源表包含serial、bigserial、smallserial、largeserial类型，或者源表字段的默认值是Sequence，且Sequence属于源表（通过CREATE SEQUENCE ... OWNED BY创建），这些Sequence不会关联到新表中，新表中会重新创建属于自己的sequence。这和之前版本的处理逻辑不同。如果用户希望源表和新表共享Sequence，需要首先创建一个共享的Sequence（避免使用OWNED BY），并配置为源表字段默认值，这样创建的新表会和源表共享该Sequence。
- 不建议将其他表私有的Sequence配置为源表字段的默认值，尤其是其他表只分布在特定的NodeGroup上，这可能导致CREATE TABLE ... LIKE执行失败。另外，如果源表配置其他表私有的Sequence，当该表删除时Sequence也会连带删除，这样源表的Sequence将不可用。如果用户希望多个表共享Sequence，建议创建共享的Sequence。
- 对于分区表EXCLUDING，需要配合INCLUDING ALL使用，如INCLUDING ALL EXCLUDING DEFAULTS，除源分区表的DEFAULTS，其它全包含。

- **AUTO_INCREMENT [=] value**

  这个子句为自动增长列指定一个初始值，value必须为正整数，不得超过2^{127}-1。

须知

该子句仅在参数sql_compatibility='B'时有效。

- **WITH ({ storage_parameter = value } [, ...])**

  这个子句为表或索引指定一个可选的存储参数。用于表的WITH子句还可以包含OIDS=FALSE表示不分配OID。

  说明

  使用任意精度类型Numeric定义列时，建议指定精度p以及刻度s。在不指定精度和刻度时，会按输入的显示出来。

  参数的详细描述如下所示。

  – FILLFACTOR

    一个表的填充因子（fillfactor）是一个介于10和100之间的百分数。在Ustore存储引擎下，默认值为92，在Astore存储引擎下默认值为100（完全填充）。如果指定了较小的填充因子，INSERT操作仅按照填充因子指定的百分率填充表页。每个页上的剩余空间将用于在该页上更新行，这就使得UPDATE有机会在同一页上放置同一条记录的新版本，这比把新版本放置在其他页上更有效。对于一个从不更新的表将填充因子设为100是最佳选择，但是对于频繁更新的表，选择较小的填充因子则更加合适。

    取值范围：10~100

  – ORIENTATION

    指定表数据的存储方式，该参数设置成功后就不再支持修改。

    取值范围：

    - ROW，表示表的数据将以行式存储。

      行存储适合于OLTP业务，适用于点查询或者增删操作较多的场景。

    默认值：

    若指定表空间为普通表空间，默认值为ROW。

  – STORAGE_TYPE

    指定存储引擎类型，该参数设置成功后就不再支持修改。

    取值范围：

    - USTORE，表示表支持Inplace-Update存储引擎。特别需要注意，使用UStore表，必须要开启track_counts和track_activities参数，否则会引起空间膨胀。
    - ASTORE，表示表支持Append-Only存储引擎。

    默认值：

    不指定表时，默认是Inplace-Update存储。

  – INIT_TD

    创建UStore表时，指定初始化的TD个数，该参数可以通过alter table进行修改。特别需要注意，该参数会影响数据页面存放的单个元组的最大大小，具体换算方法为MAX_TUPLE_SIZE = BLCKSZ - INIT_TD * TD_SIZE，例如用户将INIT_TD数量从4修改为8，单个元组最大大小会减小4 * INIT_TD大小。

    取值范围：2~128，默认值为4。

  – COMPRESSION

    指定表数据的压缩级别，它决定了表数据的压缩比以及压缩时间。一般来讲，压缩级别越高，压缩比也越大，压缩时间也越长；反之亦然。实际压缩比取决于加载的表数据的分布特征。行存表不支持压缩。

    取值范围：

行存表默认值为NO。

- COMPRESSLEVEL

  指定表数据同一压缩级别下的不同压缩水平，它决定了同一压缩级别下表数据的压缩比以及压缩时间。对同一压缩级别进行了更加详细的划分，为用户选择压缩比和压缩时间提供了更多的空间。总体来讲，此值越大，表示同一压缩级别下压缩比越大，压缩时间越长；反之亦然。

  取值范围：0~3，默认值为0。

- segment

  预留参数，暂不支持。

- parallel_workers

  表示创建索引时起的bgworker线程数量，例如2就表示将会起2个bgworker线程并发创建索引。

  取值范围：[0,32]，int类型，0表示关闭并行建索引。

  默认值：不设置该参数，表示未开启并行建索引功能。

- hasuids

  参数开启：更新表元组时，为元组分配表级唯一标识id。

  取值范围：on/off。

  默认值：off。

- collate

  在B模式数据库下（即sql_compatibility = 'B'）用于记录表的默认字符序，一般只用于内部存储和导入导出，不推荐用户指定或修改。

  取值范围：B模式数据库中独立支持的字符序的oid。

  默认值：0。

- **WITHOUT OIDS**

  等价于WITH（OIDS=FALSE）的语法。

- **ON COMMIT { PRESERVE ROWS | DELETE ROWS }**

  ON COMMIT选项决定在事务中执行创建临时表操作，当事务提交时，此临时表的后续操作。当前支持PRESERVE ROWS和DELETE ROWS选项。

  - PRESERVE ROWS（缺省值）：提交时不对临时表做任何操作，临时表及其表数据保持不变。
  - DELETE ROWS：提交时删除临时表中数据。

- **COMPRESS | NOCOMPRESS**

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

  缺省值：NOCOMPRESS，即不对元组数据进行压缩。

- **TABLESPACE tablespace_name**

  创建新表时指定此关键字，表示新表将要在指定表空间内创建。如果没有声明，将使用默认表空间。

- **CONSTRAINT constraint_name**

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

定义约束有两种方法：

- 列约束：作为一个列定义的一部分，仅影响该列。
- 表约束：不和某个列绑在一起，可以作用于多个列。

- **NOT NULL**

  字段值不允许为NULL。

- **NULL**

  字段值允许为NULL ，这是缺省值。

  这个子句只是为和非标准SQL数据库兼容。不建议使用。

- **CHECK (expression)**

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

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

  说明

  expression表达式中，如果存在“<>NULL”或“！=NULL”，这种写法是无效的，需要写成“IS NOT NULL”。

- **DEFAULT default_expr**

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

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

- **GENERATED ALWAYS AS (generation_expr) [STORED]**

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

  说明

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

- **AUTO_INCREMENT**

  该关键字将字段指定为自动增长列。

  若在插入时不指定此列的值（或指定此列的值为0、NULL、DEFAULT），此列的值将由自增计数器自动增长得到。

若插入或更新此列为一个大于当前自增计数器的值，执行成功后，自增计数器将刷新为此值。

自增初始值由“AUTO_INCREMENT [=] value”子句设置，若不设置，默认为1。

说明

- 仅在参数sql_compatibility='B'时可以指定自动增长列。
- 自动增长列数据类型只能为整数类型、4字节或8字节浮点类型、布尔类型。
- 每个表只能有一个自动增长列。
- 自动增长列必须是主键约束或唯一约束的第一个字段。
- 自动增长列不能指定DEFAULT缺省值。
- CHECK约束的表达式中不能含有自动增长列，生成列的表达式中不能含有自动增长列。
- 可以指定自动增长列允许NULL，若不指定，默认自动增长列含有NOT NULL约束。
- 含有自动增长列的表创建时，会创建一个依赖于此列的序列作为自增计数器，不允许通过序列相关功能修改或删除此序列，可以查看序列的值。请勿使其他的序列依赖或关联于此自动增长列。
- 本地临时表中的自动增长列不会创建序列。
- 自增计数器自增和刷新操作不会回滚。

- **[DEFAULT] CHARACTER SET | CHARSET [=] default_charset**

  仅在sql_compatibility='B'时支持该语法。指定表的默认字符集，单独指定时会将表的默认字符序设置为指定的字符集的默认字符序。

- **[DEFAULT] COLLATE [=] default_collation**

  仅在sql_compatibility='B'时支持该语法。指定表的默认字符序，单独指定时会将表的默认字符集设置为指定的字符序对应的字符集。字符序参见**表1 B模式（即sql_compatibility = 'B'）下支持的字符集和字符序介绍**。

  说明

  未显式指定表的字符集或字符序时，若指定了模式的默认字符集或字符序，表字符集和字符序将从模式上继承。若模式的默认字符集或字符序不存在，当b_format_behavior_compat_options = 'default_collation'时，表的字符集和字符序将继承当前数据库的字符集及其对应的默认字符序。

- **UNIQUE [KEY] index_parameters**

  UNIQUE (column_name [, ...]) index_parameters

  UNIQUE约束表示表里的一个字段或多个字段的组合必须在全表范围内唯一。

  对于唯一约束，NULL被认为是互不相等的。

  UNIQUE KEY只能在sql_compatibility='B'时使用，与UNIQUE语义相同。

- **PRIMARY KEY index_parameters**

  PRIMARY KEY (column_name [, ...]) index_parameters

  主键约束声明表中的一个或者多个字段只能包含唯一的非NULL值。

  一个表只能声明一个主键。

- **REFERENCES reftable [(refcolumn)] [MATCH matchtype] [ON DELETE action] [ON UPDATE action] (column constraint)**

  FOREIGN KEY (column_name [, ...]) REFERENCES reftable [(refcolumn [, ...])] [MATCH matchtype] [ON DELETE action] [ON UPDATE action] (table constraint)

外键约束要求新表中一列或多列构成的组应该只包含、匹配被参考表中被参考字段值。若省略refcolumn，则将使用reftable的主键。被参考列应该是被参考表中的唯一字段或主键。外键约束不能被定义在临时表和永久表之间。

参考字段与被参考字段之间存在三种类型匹配，分别是：

- MATCH FULL：不允许一个多字段外键的字段为NULL，除非全部外键字段都是NULL。
- MATCH SIMPLE（缺省）：允许任意外键字段为NULL。
- MATCH PARTIAL：目前暂不支持。

另外，当被参考表中的数据发生改变时，某些操作也会在新表对应字段的数据上执行。ON DELETE子句声明当被参考表中的被参考行被删除时要执行的操作。ON UPDATE子句声明当被参考表中的被参考字段数据更新时要执行的操作。对于ON DELETE子句、ON UPDATE子句的可能动作：

- NO ACTION（缺省）：删除或更新时，创建一个表明违反外键约束的错误。若约束可推迟，且若仍存在任何引用行，那这个错误将会在检查约束的时候产生。
- RESTRICT：删除或更新时，创建一个表明违反外键约束的错误。与NO ACTION相同，只是动作不可推迟。
- CASCADE：删除新表中任何引用了被删除行的行，或更新新表中引用行的字段值为被参考字段的新值。
- SET NULL：设置引用字段为NULL。
- SET DEFAULT：设置引用字段为它们的缺省值。

- **DEFERRABLE | NOT DEFERRABLE**

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

- **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

  如果约束是可推迟的，则这个子句声明检查约束的缺省时间。

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

  约束检查的时间可以用SET CONSTRAINTS命令修改。

- **USING INDEX TABLESPACE tablespace_name**

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

- **ENCRYPTION_TYPE = encryption_type_value**

  为ENCRYPTED WITH约束中的加密类型，encryption_type_value的值为[DETERMINISTIC | RANDOMIZED]

示例

```
--创建简单的表。
gaussdb=# CREATE TABLE tpcds.warehouse_t1
(
    W_WAREHOUSE_SK          INTEGER               NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
```



```
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
);

gaussdb=# CREATE TABLE tpcds.warehouse_t2
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60),
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
);
--创建表，并指定W_STATE字段的缺省值为GA。
gaussdb=# CREATE TABLE tpcds.warehouse_t3
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)           DEFAULT 'GA',
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
);

--创建表，并在事务结束时检查W_WAREHOUSE_NAME字段是否有重复。
gaussdb=# CREATE TABLE tpcds.warehouse_t4
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)   UNIQUE DEFERRABLE,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
);
--创建一个带有70%填充因子的表。
gaussdb=# CREATE TABLE tpcds.warehouse_t5
```



```
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2),
    UNIQUE(W_WAREHOUSE_NAME) WITH(fillfactor=70)
);

--或者用下面的语法。
gaussdb=# CREATE TABLE tpcds.warehouse_t6
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)           UNIQUE,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
) WITH(fillfactor=70);

--创建表，并指定该表数据不写入预写日志。
gaussdb=# CREATE UNLOGGED TABLE tpcds.warehouse_t7
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
);

--创建表临时表。
gaussdb=# CREATE TEMPORARY TABLE warehouse_t24
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
```



```
    W_STATE                 CHAR(2)                 ,
    W_ZIP                   CHAR(10)                ,
    W_COUNTRY               VARCHAR(20)             ,
    W_GMT_OFFSET            DECIMAL(5,2)
);

--创建本地临时表，并指定提交事务时删除该临时表数据。
gaussdb=# CREATE TEMPORARY TABLE warehouse_t25
(
    W_WAREHOUSE_SK          INTEGER               NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)             ,
    W_WAREHOUSE_SQ_FT       INTEGER                 ,
    W_STREET_NUMBER         CHAR(10)                ,
    W_STREET_NAME           VARCHAR(60)             ,
    W_STREET_TYPE           CHAR(15)                ,
    W_SUITE_NUMBER          CHAR(10)                ,
    W_CITY                  VARCHAR(60)             ,
    W_COUNTY                VARCHAR(30)             ,
    W_STATE                 CHAR(2)                 ,
    W_ZIP                   CHAR(10)                ,
    W_COUNTRY               VARCHAR(20)             ,
    W_GMT_OFFSET            DECIMAL(5,2)
) ON COMMIT DELETE ROWS;

--创建全局临时表，并指定会话结束时删除该临时表数据，当前Ustore存储引擎不支持全局临时表。
gaussdb=# CREATE GLOBAL TEMPORARY TABLE gtt1
(
    ID                      INTEGER             NOT NULL,
    NAME                    CHAR(16)            NOT NULL,
    ADDRESS                 VARCHAR(50)             ,
    POSTCODE                CHAR(6)
) ON COMMIT PRESERVE ROWS;

--创建表时，不希望因为表已存在而报错。
gaussdb=# CREATE TABLE IF NOT EXISTS tpcds.warehouse_t8
(
    W_WAREHOUSE_SK          INTEGER               NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)             ,
    W_WAREHOUSE_SQ_FT       INTEGER                 ,
    W_STREET_NUMBER         CHAR(10)                ,
    W_STREET_NAME           VARCHAR(60)             ,
    W_STREET_TYPE           CHAR(15)                ,
    W_SUITE_NUMBER          CHAR(10)                ,
    W_CITY                  VARCHAR(60)             ,
    W_COUNTY                VARCHAR(30)             ,
    W_STATE                 CHAR(2)                 ,
    W_ZIP                   CHAR(10)                ,
    W_COUNTRY               VARCHAR(20)             ,
    W_GMT_OFFSET            DECIMAL(5,2)
);

--创建普通表空间。
gaussdb=# CREATE TABLESPACE DS_TABLESPACE1 RELATIVE LOCATION 'tablespace/tablespace_1';
--创建表时，指定表空间。
gaussdb=# CREATE TABLE tpcds.warehouse_t9
(
    W_WAREHOUSE_SK          INTEGER               NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)             ,
    W_WAREHOUSE_SQ_FT       INTEGER                 ,
    W_STREET_NUMBER         CHAR(10)                ,
    W_STREET_NAME           VARCHAR(60)             ,
    W_STREET_TYPE           CHAR(15)                ,
    W_SUITE_NUMBER          CHAR(10)                ,
    W_CITY                  VARCHAR(60)             ,
    W_COUNTY                VARCHAR(30)             ,
    W_STATE                 CHAR(2)                 ,
```



```
    W_ZIP                 CHAR(10)                    ,
    W_COUNTRY             VARCHAR(20)                 ,
    W_GMT_OFFSET          DECIMAL(5,2)
) TABLESPACE DS_TABLESPACE1;

--创建表时，单独指定W_WAREHOUSE_NAME的索引表空间。
gaussdb=# CREATE TABLE tpcds.warehouse_t10
(
    W_WAREHOUSE_SK        INTEGER               NOT NULL,
    W_WAREHOUSE_ID        CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME      VARCHAR(20)           UNIQUE USING INDEX TABLESPACE
DS_TABLESPACE1,
    W_WAREHOUSE_SQ_FT     INTEGER                     ,
    W_STREET_NUMBER       CHAR(10)                    ,
    W_STREET_NAME         VARCHAR(60)                 ,
    W_STREET_TYPE         CHAR(15)                    ,
    W_SUITE_NUMBER        CHAR(10)                    ,
    W_CITY                VARCHAR(60)                 ,
    W_COUNTY              VARCHAR(30)                 ,
    W_STATE               CHAR(2)                     ,
    W_ZIP                 CHAR(10)                    ,
    W_COUNTRY             VARCHAR(20)                 ,
    W_GMT_OFFSET          DECIMAL(5,2)
);
--创建一个有主键约束的表。
gaussdb=# CREATE TABLE tpcds.warehouse_t11
(
    W_WAREHOUSE_SK        INTEGER           PRIMARY KEY,
    W_WAREHOUSE_ID        CHAR(16)          NOT NULL,
    W_WAREHOUSE_NAME      VARCHAR(20)               ,
    W_WAREHOUSE_SQ_FT     INTEGER                   ,
    W_STREET_NUMBER       CHAR(10)                  ,
    W_STREET_NAME         VARCHAR(60)               ,
    W_STREET_TYPE         CHAR(15)                  ,
    W_SUITE_NUMBER        CHAR(10)                  ,
    W_CITY                VARCHAR(60)               ,
    W_COUNTY              VARCHAR(30)               ,
    W_STATE               CHAR(2)                   ,
    W_ZIP                 CHAR(10)                  ,
    W_COUNTRY             VARCHAR(20)               ,
    W_GMT_OFFSET          DECIMAL(5,2)
);

--或使用下面的语法，效果完全一样。
gaussdb=# CREATE TABLE tpcds.warehouse_t12
(
    W_WAREHOUSE_SK        INTEGER               NOT NULL,
    W_WAREHOUSE_ID        CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME      VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT     INTEGER                     ,
    W_STREET_NUMBER       CHAR(10)                    ,
    W_STREET_NAME         VARCHAR(60)                 ,
    W_STREET_TYPE         CHAR(15)                    ,
    W_SUITE_NUMBER        CHAR(10)                    ,
    W_CITY                VARCHAR(60)                 ,
    W_COUNTY              VARCHAR(30)                 ,
    W_STATE               CHAR(2)                     ,
    W_ZIP                 CHAR(10)                    ,
    W_COUNTRY             VARCHAR(20)                 ,
    W_GMT_OFFSET          DECIMAL(5,2),
    PRIMARY KEY(W_WAREHOUSE_SK)
);

--或使用下面的语法，指定约束的名称。
gaussdb=# CREATE TABLE tpcds.warehouse_t13
(
    W_WAREHOUSE_SK        INTEGER               NOT NULL,
    W_WAREHOUSE_ID        CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME      VARCHAR(20)                 ,
```

```
    W_WAREHOUSE_SQ_FT         INTEGER                   ,
    W_STREET_NUMBER           CHAR(10)                  ,
    W_STREET_NAME             VARCHAR(60)               ,
    W_STREET_TYPE             CHAR(15)                  ,
    W_SUITE_NUMBER            CHAR(10)                  ,
    W_CITY                    VARCHAR(60)               ,
    W_COUNTY                  VARCHAR(30)               ,
    W_STATE                   CHAR(2)                   ,
    W_ZIP                     CHAR(10)                  ,
    W_COUNTRY                 VARCHAR(20)               ,
    W_GMT_OFFSET              DECIMAL(5,2),
    CONSTRAINT W_CSTR_KEY1 PRIMARY KEY(W_WAREHOUSE_SK)
);

--创建一个有复合主键约束的表。
gaussdb=# CREATE TABLE tpcds.warehouse_t14
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                   ,
    W_STREET_NUMBER           CHAR(10)                  ,
    W_STREET_NAME             VARCHAR(60)               ,
    W_STREET_TYPE             CHAR(15)                  ,
    W_SUITE_NUMBER            CHAR(10)                  ,
    W_CITY                    VARCHAR(60)               ,
    W_COUNTY                  VARCHAR(30)               ,
    W_STATE                   CHAR(2)                   ,
    W_ZIP                     CHAR(10)                  ,
    W_COUNTRY                 VARCHAR(20)               ,
    W_GMT_OFFSET              DECIMAL(5,2),
    CONSTRAINT W_CSTR_KEY2 PRIMARY KEY(W_WAREHOUSE_SK, W_WAREHOUSE_ID)
);

--定义一个检查列约束。
gaussdb=# CREATE TABLE tpcds.warehouse_t19
(
    W_WAREHOUSE_SK            INTEGER               PRIMARY KEY CHECK (W_WAREHOUSE_SK > 0),
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)           CHECK (W_WAREHOUSE_NAME IS NOT NULL),
    W_WAREHOUSE_SQ_FT         INTEGER                   ,
    W_STREET_NUMBER           CHAR(10)                  ,
    W_STREET_NAME             VARCHAR(60)               ,
    W_STREET_TYPE             CHAR(15)                  ,
    W_SUITE_NUMBER            CHAR(10)                  ,
    W_CITY                    VARCHAR(60)               ,
    W_COUNTY                  VARCHAR(30)               ,
    W_STATE                   CHAR(2)                   ,
    W_ZIP                     CHAR(10)                  ,
    W_COUNTRY                 VARCHAR(20)               ,
    W_GMT_OFFSET              DECIMAL(5,2)
);

gaussdb=# CREATE TABLE tpcds.warehouse_t20
(
    W_WAREHOUSE_SK            INTEGER               PRIMARY KEY,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)           CHECK (W_WAREHOUSE_NAME IS NOT NULL),
    W_WAREHOUSE_SQ_FT         INTEGER                   ,
    W_STREET_NUMBER           CHAR(10)                  ,
    W_STREET_NAME             VARCHAR(60)               ,
    W_STREET_TYPE             CHAR(15)                  ,
    W_SUITE_NUMBER            CHAR(10)                  ,
    W_CITY                    VARCHAR(60)               ,
    W_COUNTY                  VARCHAR(30)               ,
    W_STATE                   CHAR(2)                   ,
    W_ZIP                     CHAR(10)                  ,
    W_COUNTRY                 VARCHAR(20)               ,
    W_GMT_OFFSET              DECIMAL(5,2),
```



```
    CONSTRAINT W_CONSTR_KEY2 CHECK(W_WAREHOUSE_SK > 0 AND W_WAREHOUSE_NAME IS NOT
NULL)
);

--创建一个有外键约束的表。
gaussdb=# CREATE TABLE tpcds.city_t23
(
    W_CITY            VARCHAR(60)               PRIMARY KEY,
    W_ADDRESS         TEXT
);
gaussdb=# CREATE TABLE tpcds.warehouse_t23
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)           REFERENCES tpcds.city_t23(W_CITY),
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
);

--或使用下面的语法，效果完全一样。
gaussdb=# CREATE TABLE tpcds.warehouse_t23
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)                  ,
    FOREIGN KEY(W_CITY) REFERENCES tpcds.city_t23(W_CITY)
);

--或使用下面的语法，指定约束的名称。
gaussdb=# CREATE TABLE tpcds.warehouse_t23
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)                  ,
    CONSTRAINT W_FORE_KEY1 FOREIGN KEY(W_CITY) REFERENCES tpcds.city_t23(W_CITY)
);

--向tpcds.warehouse_t19表中增加一个varchar列。
```



```
gaussdb=# ALTER TABLE tpcds.warehouse_t19 ADD W_GOODS_CATEGORY varchar(30);

--给tpcds.warehouse_t19表增加一个检查约束。
gaussdb=# ALTER TABLE tpcds.warehouse_t19 ADD CONSTRAINT W_CONSTR_KEY4 CHECK (W_STATE IS
NOT NULL);

--在一个操作中改变两个现存字段的类型。
gaussdb=# ALTER TABLE tpcds.warehouse_t19
    ALTER COLUMN W_GOODS_CATEGORY TYPE varchar(80),
    ALTER COLUMN W_STREET_NAME TYPE varchar(100);

--此语句与上面语句等效。
gaussdb=# ALTER TABLE tpcds.warehouse_t19 MODIFY (W_GOODS_CATEGORY varchar(30),
W_STREET_NAME varchar(60));

--给一个已存在字段添加非空约束。
gaussdb=# ALTER TABLE tpcds.warehouse_t19 ALTER COLUMN W_GOODS_CATEGORY SET NOT NULL;

--移除已存在字段的非空约束。
gaussdb=# ALTER TABLE tpcds.warehouse_t19 ALTER COLUMN W_GOODS_CATEGORY DROP NOT NULL;
--将表移动到另一个表空间。
gaussdb=# ALTER TABLE tpcds.warehouse_t19 SET TABLESPACE PG_DEFAULT;
--创建模式joe。
gaussdb=# CREATE SCHEMA joe;

--将表移动到另一个模式中。
gaussdb=# ALTER TABLE tpcds.warehouse_t19 SET SCHEMA joe;

--重命名已存在的表。
gaussdb=# ALTER TABLE joe.warehouse_t19 RENAME TO warehouse_t23;

--从warehouse_t23表中删除一个字段。
gaussdb=# ALTER TABLE joe.warehouse_t23 DROP COLUMN W_STREET_NAME;

--删除表空间、模式joe和模式表warehouse。
gaussdb=# DROP TABLE tpcds.warehouse_t1;
gaussdb=# DROP TABLE tpcds.warehouse_t2;
gaussdb=# DROP TABLE tpcds.warehouse_t3;
gaussdb=# DROP TABLE tpcds.warehouse_t4;
gaussdb=# DROP TABLE tpcds.warehouse_t5;
gaussdb=# DROP TABLE tpcds.warehouse_t6;
gaussdb=# DROP TABLE tpcds.warehouse_t7;
gaussdb=# DROP TABLE tpcds.warehouse_t8;
gaussdb=# DROP TABLE tpcds.warehouse_t9;
gaussdb=# DROP TABLE tpcds.warehouse_t10;
gaussdb=# DROP TABLE tpcds.warehouse_t11;
gaussdb=# DROP TABLE tpcds.warehouse_t12;
gaussdb=# DROP TABLE tpcds.warehouse_t13;
gaussdb=# DROP TABLE tpcds.warehouse_t14;
gaussdb=# DROP TABLE tpcds.warehouse_t18;
gaussdb=# DROP TABLE tpcds.warehouse_t20;
gaussdb=# DROP TABLE tpcds.warehouse_t21;
gaussdb=# DROP TABLE tpcds.warehouse_t22;
gaussdb=# DROP TABLE joe.warehouse_t23;
gaussdb=# DROP TABLE tpcds.warehouse_t24;
gaussdb=# DROP TABLE tpcds.warehouse_t25;
gaussdb=# DROP TABLESPACE DS_TABLESPACE1;
gaussdb=# DROP SCHEMA IF EXISTS joe CASCADE;

-- 创建t1表，设置t1的默认字符集为utf8mb4，默认字符序为utf8mb4_bin，设置c1字段的字符集为utf8mb4，字
符序为utf8mb4_unicode_ci
gaussdb=# CREATE TABLE T1(C1 VARCHAR(20) CHARSET utf8mb4 COLLATE utf8mb4_unicode_ci)
CHARSET = utf8mb4 COLLATE = utf8mb4_bin;
```

相关链接

ALTER TABLE，**DROP TABLE**，**CREATE TABLESPACE**

优化建议

- UNLOGGED
  - UNLOGGED表和表上的索引因为数据写入时不通过WAL日志机制，写入速度远高于普通表。因此，可以用于缓冲存储复杂查询的中间结果集，增强复杂查询的性能。
  - UNLOGGED表无主备机制，在系统故障或异常断点等情况下，会有数据丢失风险，因此，不可用来存储基础数据。
- TEMPORARY | TEMP
  - 临时表只在当前会话可见，会话结束后会自动删除。
- LIKE
  - 新表自动从这个表中继承所有字段名及其数据类型和非空约束，新表与源表之间在创建动作完毕之后是完全无关的。
- LIKE INCLUDING DEFAULTS
  - 源表上的字段缺省表达式只有在指定INCLUDING DEFAULTS时，才会复制到新表中。缺省是不包含缺省表达式的，即新表中的所有字段的缺省值都是NULL。
- LIKE INCLUDING CONSTRAINTS
  - 源表上的CHECK约束仅在指定INCLUDING CONSTRAINTS时，会复制到新表中，而其他类型的约束永远不会复制到新表中。非空约束总是复制到新表中。此规则同时适用于表约束和列约束。
- LIKE INCLUDING INDEXES
  - 如果指定了INCLUDING INDEXES，则源表上的索引也将在新表上创建，默认不建立索引。
- LIKE INCLUDING STORAGE
  - 如果指定了INCLUDING STORAGE，则复制列的STORAGE设置会复制到新表中，默认情况下不包含STORAGE设置。
- LIKE INCLUDING COMMENTS
  - 如果指定了INCLUDING COMMENTS，则源表列、约束和索引的注释会复制到新表中。默认情况下，不复制源表的注释。
- LIKE INCLUDING PARTITION
  - 如果指定了INCLUDING PARTITION，则源表的分区定义会复制到新表中，同时新表将不能再使用PARTITION BY子句。默认情况下，不拷贝源表的分区定义。

  须知

  列表/哈希分区表暂不支持LIKE INCLUDING PARTITION。

- LIKE INCLUDING RELOPTIONS
  - 如果指定了INCLUDING RELOPTIONS，则源表的存储参数（即源表的WITH子句）会复制到新表中。默认情况下，不复制源表的存储参数。
- LIKE INCLUDING ALL
  - INCLUDING ALL包含了INCLUDING DEFAULTS、INCLUDING CONSTRAINTS、INCLUDING INDEXES、INCLUDING STORAGE、

INCLUDING COMMENTS、INCLUDING PARTITION、INCLUDING RELOPTIONS的内容。

- ORIENTATION ROW
  - 创建行存表，行存储适合于OLTP业务，此类型的表上交互事务比较多，一次交互会涉及表中的多个列，用行存查询效率较高。

7.14.85 CREATE TABLE AS

功能描述

根据查询结果创建表。

CREATE TABLE AS创建一个表并且用来自SELECT命令的结果填充该表。该表的字段和SELECT输出字段的名称及数据类型相关。不过用户可以通过明确地给出一个字段名称列表来覆盖SELECT输出字段的名称。

CREATE TABLE AS对源表进行一次查询，然后将数据写入新表中，而查询视图结果会根据源表的变化而有所改变。相比之下，每次做查询的时候，视图都重新计算定义它的SELECT语句。

注意事项

- 分区表不能采用此方式进行创建。
- 如果在建表过程中数据库系统发生故障，系统恢复后可能无法自动清除之前已创建的、大小非0的磁盘文件。此种情况出现概率小，不影响数据库系统的正常运行。

语法格式

```
CREATE [ [ GLOBAL | LOCAL ] [ TEMPORARY | TEMP ] | UNLOGGED ] TABLE table_name
   [ (column_name [, ...] ) ]
   [ WITH ( {storage_parameter = value} [, ... ] ) ]
   [ ON COMMIT { PRESERVE ROWS | DELETE ROWS } ]
   [ COMPRESS | NOCOMPRESS ]
   [ TABLESPACE tablespace_name ]
   AS query
   [ WITH [ NO ] DATA ];
```

参数说明

- **UNLOGGED**

  指定表为非日志表。在非日志表中写入的数据不会被写入到预写日志中，这样就会比普通表快很多。但是，这也是不安全的，非日志表在冲突或异常关机后会被自动删截。非日志表中的内容也不会被复制到备用服务器中。在该类表中创建的索引也不会被自动记录。

  - 使用场景：非日志表不能保证数据的安全性，用户应该在确保数据已经做好备份的前提下使用，例如系统升级时进行数据的备份。
  - 故障处理：当异常关机等操作导致非日志表上的索引发生数据丢失时，用户应该对发生错误的索引进行重建。

- **GLOBAL | LOCAL**

  创建临时表时可以在TEMP或TEMPORARY前指定GLOBAL或LOCAL关键字。如果指定GLOBAL关键字，GaussDB会创建全局临时表，否则GaussDB会创建本地临时表。

- **TEMPORARY | TEMP**

  如果指定TEMP或TEMPORARY关键字，则创建的表为临时表。临时表分为全局临时表和本地临时表两种类型。创建临时表时如果指定GLOBAL关键字则为全局临时表，否则为本地临时表。

  全局临时表的元数据对所有会话可见，会话结束后元数据继续存在。会话与会话之间的用户数据、索引和统计信息相互隔离，每个会话只能看到和更改自己提交的数据。全局临时表有两种模式：一种是基于会话级别的（ON COMMIT PRESERVE ROWS），当会话结束时自动清空用户数据；一种是基于事务级别的（ON COMMIT DELETE ROWS），当执行COMMIT或ROLLBACK时自动清空用户数据。建表时如果没有指定ON COMMIT选项，则缺省为会话级别。与本地临时表不同，全局临时表建表时可以指定非pg_temp_开头的SCHEMA。

  本地临时表只在当前会话可见，本会话结束后会自动删除。因此，在除当前会话连接的数据库节点故障时，仍然可以在当前会话上创建和使用临时表。由于临时表只在当前会话创建，对于涉及对临时表操作的DDL语句，会产生DDL失败的报错。因此，建议DDL语句中不要对临时表进行操作。TEMP和TEMPORARY等价。

  须知

  - 本地临时表通过每个会话独立的以pg_temp开头的SCHEMA来保证只对当前会话可见，因此，不建议用户在日常操作中手动删除以pg_temp，pg_toast_temp开头的SCHEMA。
  - 如果建表时不指定TEMPORARY/TEMP关键字，而指定表的SCHEMA为当前会话的pg_temp_开头的SCHEMA，则此表会被创建为临时表。
  - ALTER/DROP全局临时表和索引，如果其它会话正在使用它，禁止操作。
  - 全局临时表的DDL只会影响当前会话的用户数据和索引。例如TRUNCATE、REINDEX、ANALYZE只对当前会话有效。

- **table_name**

  要创建的表名。

  取值范围：字符串，要符合**标识符命名规范**。

- **column_name**

  新表中要创建的字段名。

  取值范围：字符串，要符合**标识符命名规范**。

- **WITH (storage_parameter [= value] [, ...])**

  这个子句为表或索引指定一个可选的存储参数。参数的详细说明如下所示。

  - FILLFACTOR

    一个表的填充因子（fillfactor）是一个介于10和100之间的百分数。在Ustore存储引擎下，默认值为92，在Astore存储引擎下默认值为100（完全填充）。如果指定了较小的填充因子，INSERT操作仅按照填充因子指定的百分率填充表页。每个页上的剩余空间将用于在该页上更新行，这就使得UPDATE有机会在同一页上放置同一条记录的新版本，这比把新版本放置在其他页上更有效。对于一个从不更新的表将填充因子设为100是最佳选择，但是对于频繁更新的表，选择较小的填充因子则更加合适。

    取值范围：10~100

  - ORIENTATION

    取值范围：

ROW（缺省值）：表的数据将以行式存储。

- COMPRESSION

  指定表数据的压缩级别，它决定了表数据的压缩比以及压缩时间。一般来讲，压缩级别越高，压缩比也越大，压缩时间也越长；反之亦然。实际压缩比取决于加载的表数据的分布特征。

  取值范围：

  行存表不支持压缩。

- **ON COMMIT { PRESERVE ROWS | DELETE ROWS }**

  ON COMMIT选项决定在事务中执行创建临时表操作，当事务提交时，此临时表的后续操作。当前仅支持PRESERVE ROWS和DELETE ROWS选项。

  - PRESERVE ROWS（缺省值）：提交时不对临时表执行任何操作，临时表及其表数据保持不变。
  - DELETE ROWS：提交时删除临时表中数据。

- **COMPRESS / NOCOMPRESS**

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

  缺省值：NOCOMPRESS，即不对元组数据进行压缩。

- **TABLESPACE tablespace_name**

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

- **AS query**

  一个SELECT VALUES命令或者一个运行预备好的SELECT或VALUES查询的EXECUTE命令。

- **[WITH [NO] DATA]**

  创建表时，是否也插入查询到的数据。默认是要数据，选择“NO”参数时，则不要数据。

示例

```
--创建一个SCHEMA。
gaussdb=# CREATE SCHEMA tpcds;

--创建一个表tpcds.store_returns表。
gaussdb=# CREATE TABLE tpcds.store_returns
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    sr_item_sk                VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER
);

--向表中插入一条记录。
gaussdb=# INSERT INTO tpcds.store_returns(W_WAREHOUSE_SK, W_WAREHOUSE_ID, sr_item_sk,
W_WAREHOUSE_SQ_FT) VALUES (1,
'AAAAAAAABAAAAAAA', '4800', '20');

--创建一个表tpcds.store_returns_t1并插入tpcds.store_returns表中sr_item_sk字段中大于16的数值。
gaussdb=# CREATE TABLE tpcds.store_returns_t1 AS SELECT * FROM tpcds.store_returns WHERE sr_item_sk
> '4795';

--使用tpcds.store_returns拷贝一个新表tpcds.store_returns_t2。
```



```
gaussdb=# CREATE TABLE tpcds.store_returns_t2 AS table tpcds.store_returns;

--删除表。
gaussdb=# DROP TABLE tpcds.store_returns_t1 ;
gaussdb=# DROP TABLE tpcds.store_returns_t2 ;
gaussdb=# DROP TABLE tpcds.store_returns;

--删除SCHEMA。
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

相关链接

CREATE TABLE，**SELECT**

7.14.86 CREATE TABLE PARTITION

功能描述

创建分区表。分区表是把逻辑上的一张表根据某种方案分成几张物理块进行存储，这张逻辑上的表称之为分区表，物理块称之为分区。分区表是一张逻辑表，不存储数据，数据实际是存储在分区上的。

常见的分区方案有范围分区（Range Partitioning）、间隔分区（Interval Partitioning）、哈希分区（Hash Partitioning）、列表分区（List Partitioning）、数值分区（Value Partition）等。目前行存表支持范围分区、间隔分区、哈希分区、列表分区。

范围分区是根据表的一列或者多列，将要插入表的记录分为若干个范围，这些范围在不同的分区里没有重叠。为每个范围创建一个分区，用来存储相应的数据。

范围分区的分区策略是指记录插入分区的方式。目前范围分区仅支持范围分区策略。

范围分区策略：根据分区键值将记录映射到已创建的某个分区上，如果可以映射到已创建的某一分区上，则把记录插入到对应的分区上，否则给出报错和提示信息。这是最常用的分区策略。

间隔分区是一种特殊的范围分区，相比范围分区，新增间隔值定义，当插入记录找不到匹配的分区时，可以根据间隔值自动创建分区。

间隔分区只支持基于表的一列分区，并且该列只支持TIMESTAMP[(p)] [WITHOUT TIME ZONE]、TIMESTAMP[(p)] [WITH TIME ZONE]、DATE数据类型。

间隔分区策略：根据分区键值将记录映射到已创建的某个分区上，如果可以映射到已创建的某一分区上，则把记录插入到对应的分区上，否则根据分区键值和表定义信息自动创建一个分区，然后将记录插入新分区中，新创建的分区数据范围等于间隔值。

哈希分区是根据表的一列，为每个分区指定模数和余数，将要插入表的记录划分到对应的分区中，每个分区所持有的行都需要满足条件：分区键的值除以为其指定的模数将产生为其指定的余数。

哈希分区策略：根据分区键值将记录映射到已创建的某个分区上，如果可以映射到已创建的某一分区上，则把记录插入到对应的分区上，否则返回报错和提示信息。

列表分区是根据表的一列，将要插入表的记录通过每一个分区中出现的键值划分到对应的分区中，这些键值在不同的分区里没有重叠。为每组键值创建一个分区，用来存储相应的数据。

列表分区策略：根据分区键值将记录映射到已创建的某个分区上，如果可以映射到已创建的某一分区上，则把记录插入到对应的分区上，否则给出报错和提示信息。

分区可以提供若干好处：

- 某些类型的查询性能可以得到极大提升。特别是表中访问率较高的行位于一个单独分区或少数几个分区上的情况下。分区可以减少数据的搜索空间，提高数据访问效率。
- 当查询或更新一个分区的大部分记录时，连续扫描那个分区而不是访问整个表可以获得巨大的性能提升。
- 如果需要大量加载或者删除的记录位于单独的分区上，则可以通过直接读取或删除那个分区以获得巨大的性能提升，同时还可以避免由于大量DELETE导致的VACUUM超载（哈希分区不支持删除分区）。

注意事项

- 唯一约束和主键约束的约束键包含所有分区键将为约束创建LOCAL索引，否则创建GLOBAL索引。
- 目前哈希分区仅支持单列构建分区键，暂不支持多列构建分区键。
- 只需要有间隔分区表的INSERT权限，往该表INSERT数据时就可以自动创建分区。
- 对于分区表PARTITION FOR (values)语法，values只能是常量。
- 对于分区表PARTITION FOR (values)语法，values在需要数据类型转换时，建议使用强制类型转换，以防隐式类型转换结果与预期不符。
- 分区数最大值为1048575个，一般情况下业务不可能创建这么多分区，这样会导致内存不足。应参照参数local_syscache_threshold的值合理创建分区，分区表使用内存大致为（分区数 * 3 / 1024）MB。理论上分区占用内存不允许大于local_syscache_threshold的值，同时还需要预留部分空间以供其他功能使用。
- 当分区数太多导致内存不足时，会间接导致性能急剧下降。
- 指定分区语句目前不能走全局索引扫描。
- 不支持XML类型数据作为分区键、二级分区键。

语法格式

```
CREATE TABLE [ IF NOT EXISTS ] partition_table_name
( [
   { column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]
   | table_constraint
   | LIKE source_table [ like_option [...] ] }[, ... ]
] )
   [ AUTO_INCREMENT [ = ] value ]
   [ WITH ( {storage_parameter = value} [, ... ] ) ]
   [ COMPRESS | NOCOMPRESS ]
   [ TABLESPACE tablespace_name ]
    PARTITION BY {
       {RANGE [COLUMNS] (partition_key) [ INTERVAL ('interval_expr') [ STORE IN (tablespace_name
[, ... ] ) ] ] [ PARTITIONS integer ] ( partition_less_than_item [, ... ] )} |
       {RANGE [COLUMNS] (partition_key) [ INTERVAL ('interval_expr') [ STORE IN (tablespace_name
[, ... ] ) ] ] [ PARTITIONS integer ] ( partition_start_end_item [, ... ] )} |
       {LIST [COLUMNS] (partition_key) [ PARTITIONS integer ] ( PARTITION partition_name VALUES [IN]
(list_values) [TABLESPACE [=] tablespace_name][, ... ] ) } |
       {{ HASH | KEY } (partition_key) [ PARTITIONS integer ] ( PARTITION partition_name [TABLESPACE [=]
tablespace_name][, ... ] ) }
   } [ { ENABLE | DISABLE } ROW MOVEMENT ];
```

- 列约束column_constraint：

  ```
  [ CONSTRAINT constraint_name ]
  { NOT NULL |
    NULL |
    CHECK ( expression ) |
    DEFAULT default_e xpr |
  ```



```
  GENERATED ALWAYS AS ( generation_expr ) [STORED] |
  AUTO_INCREMENT |
  UNIQUE [KEY] index_parameters |
  PRIMARY KEY index_parameters |
  REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
      [ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

- 表约束table_constraint：

```
[ CONSTRAINT [ constraint_name ] ]
{ CHECK ( expression ) |
  UNIQUE [ index_name ][ USING method ] ( { column_name [ ASC | DESC ] } [, ... ] )
index_parameters |
  PRIMARY KEY [ USING method ] ( { column_name [ ASC | DESC ] } [, ... ] ) index_parameters |
  FOREIGN KEY [ index_name ] ( column_name [, ... ] ) REFERENCES reftable [ ( refcolumn [, ... ] ) ]
     [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE action ] [ ON UPDATE
action ] }
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

- like选项like_option：

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

- 索引存储参数index_parameters：

```
[ WITH ( {storage_parameter = value} [, ... ] ) ]
[ USING INDEX TABLESPACE tablespace_name ]
```

- partition_less_than_item：

```
PARTITION partition_name VALUES LESS THAN {( { partition_value | MAXVALUE } [,...] ) |
MAXVALUE } [TABLESPACE [=] tablespace_name]
```

- partition_start_end_item：

```
PARTITION partition_name {
      {START(partition_value) END (partition_value) EVERY (interval_value)} |
      {START(partition_value) END ({partition_value | MAXVALUE})} |
      {START(partition_value)} |
      {END({partition_value | MAXVALUE})}
} [TABLESPACE [=] tablespace_name]
```

参数说明

- **IF NOT EXISTS**

  如果已经存在相同名称的表，不抛出错误，而是发出一个notice，告知表已存在。

- **partition_table_name**

  分区表的名称。

  取值范围：字符串，要符合**标识符命名规范**。

- **column_name**

  新表中要创建的字段名。

  取值范围：字符串，要符合**标识符命名规范**。

- **data_type**

  字段的数据类型。

- **COLLATE collation**

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

- **CONSTRAINT constraint_name**

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

定义约束有两种方法：

- 列约束：作为一个列定义的一部分，仅影响该列。
- 表约束：不和某个列绑在一起，可以作用于多个列。在B模式数据库下（即sql_compatibility = 'B'）constraint_name为可选项，在其他模式数据库下，必须加上constraint_name。

- **index_name**

  索引名。

  须知

  - index_name仅在B模式数据库下（即sql_compatibility = 'B'）支持，其他模式数据库下不支持。
  - 对于外键约束，constraint_name和index_name同时指定时，索引名为constraint_name。
  - 对于唯一键约束，constraint_name和index_name同时指定时，索引名以index_name。

- **USING method**

  指定创建索引的方法。

  取值范围参考**参数说明**中的USING method。

  须知

  - USING method仅在B模式数据库下（即sql_compatibility = 'B'）支持，其他模式数据库下不支持。
  - 在B模式下，未指定USING method时，对于ASTORE的存储方式，默认索引方法为btree；对于USTORE的存储方式，默认索引方法为ubtree。

- **ASC | DESC**

  ASC表示指定按升序排序（默认）。DESC指定按降序排序。

  须知

  ASC|DESC只在B模式数据库下（即sql_compatibility = 'B'）支持，其他模式数据库不支持。

- **LIKE source_table [like_option ...]**

  LIKE子句声明一个表，新表自动从这个表里面继承所有字段名及其数据类型和非空约束。

  新表与原表之间在创建动作完毕之后是完全无关的。在源表做的任何修改都不会传播到新表中，并且也不可能在扫描源表的时候包含新表的数据。

  - 字段缺省表达式只有在声明了INCLUDING DEFAULTS之后才会包含进来。缺省是不包含缺省表达式的，即新表中所有字段的缺省值都是NULL。
  - 如果指定了INCLUDING GENERATED，则原表列的生成表达式会复制到新表中。默认不复制生成表达式。

  - 非空约束将总是复制到新表中，CHECK约束则仅在指定了INCLUDING CONSTRAINTS的时候才复制，而其他类型的约束则永远也不会被复制。此规则同时适用于表约束和列约束。
  - 被复制的列和约束并不使用相同的名称进行融合。如果明确的指定了相同的名称或者在另外一个LIKE子句中，将会报错。
  - 如果指定了INCLUDING INDEXES，则原表上的索引也将在新表上创建，默认不建立索引。
  - 如果指定了INCLUDING STORAGE，则原表列的STORAGE设置也将被拷贝，默认情况下不包含STORAGE设置。
  - 如果指定了INCLUDING COMMENTS，则原表列、约束和索引的注释也会被拷贝过来。默认情况下，不拷贝原表的注释。
  - 如果指定了INCLUDING RELOPTIONS，则原表的存储参数（即源表的WITH子句）也将拷贝至新表。默认情况下，不拷贝原表的存储参数。
  - INCLUDING ALL包含了INCLUDING DEFAULTS、INCLUDING CONSTRAINTS、INCLUDING INDEXES、INCLUDING STORAGE、INCLUDING COMMENTS、INCLUDING PARTITION和INCLUDING RELOPTIONS的内容。

- **AUTO_INCREMENT [=] value**

  这个子句为自动增长列指定一个初始值，value必须为正整数，不得超过2^{127}-1。

  须知

  该子句仅在参数sql_compatibility='B'时有效。

- **WITH (storage_parameter [= value] [, ...])**

  这个子句为表或索引指定一个可选的存储参数。参数的详细描述如下所示：

  - FILLFACTOR

    一个表的填充因子（fillfactor）是一个介于10和100之间的百分数。在Ustore存储引擎下，默认值为92，在Astore存储引擎下默认值为100（完全填充）。如果指定了较小的填充因子，INSERT操作仅按照填充因子指定的百分率填充表页。每个页上的剩余空间将用于在该页上更新行，这就使得UPDATE有机会在同一页上放置同一条记录的新版本，这比把新版本放置在其他页上更有效。对于一个从不更新的表将填充因子设为100是最佳选择，但是对于频繁更新的表，选择较小的填充因子则更加合适。

    取值范围：10~100

  - ORIENTATION

    决定了表的数据的存储方式。

    取值范围：

    - ROW（缺省值）：表的数据将以行式存储。

      须知

      orientation不支持修改。

  - STORAGE_TYPE

指定存储引擎类型，该参数设置成功后就不再支持修改。

取值范围：

- USTORE，表示表支持Inplace-Update存储引擎。特别需要注意，使用USTORE表，必须要开启track_counts和track_activities参数，否则会引起空间膨胀。
- ASTORE，表示表支持Append-Only存储引擎。

默认值：

不指定表时，默认是Inplace-Update存储。

– COMPRESSION

- 行存表不支持压缩。

– segment

预留参数，暂不支持。

- **COMPRESS / NOCOMPRESS**

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

  缺省值为NOCOMPRESS，即不对元组数据进行压缩。

- **TABLESPACE tablespace_name**

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

- **PARTITION BY RANGE [COLUMNS] (partition_key)**

  创建范围分区。partition_key为分区键的名称。

  COLUMNS关键字只能在sql_compatibility='B'时使用，“PARTITION BY RANGE COLUMNS” 语义同 “PARTITION BY RANGE”。

  （1）对于从句是VALUES LESS THAN的语法格式：

  须知

  对于从句是VALUE LESS THAN的语法格式，范围分区策略的分区键最多支持16列。

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

  （2）对于从句是START END的语法格式：

  须知

  对于从句是START END的语法格式，范围分区策略的分区键仅支持1列。

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

（3）对于指定了INTERVAL子句的语法格式：

须知

对于指定了INTERVAL子句的语法格式，范围分区策略的分区键仅支持1列。

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

- **PARTITION partition_name VALUES LESS THAN {({ partition_value | MAXVALUE } [,...]) | MAXVALUE }**

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

  须知

  - 每个分区都需要指定一个上边界。
  - 分区上边界的类型应当和分区键的类型一致。
  - 分区列表是按照分区上边界升序排列的，值较小的分区位于值较大的分区之前。
  - 不在括号内的MAXVALUE只能在sql_compatibility='B'时使用，并且只能有一个分区键。

- **PARTITION partition_name {START (partition_value) END (partition_value) EVERY (interval_value)} | {START (partition_value) END (partition_value|MAXVALUE)} | {START(partition_value)} | {END (partition_value | MAXVALUE)}**

  指定各分区的信息，各参数意义如下：

  - partition_name：范围分区的名称或名称前缀，除以下情形外（假定其中的partition_name是p1），均为分区的名称。
    - 若该定义是START+END+EVERY从句，则语义上定义的分区的名称依次为p1_1, p1_2, ...。例如对于定义“PARTITION p1 START(1) END(4) EVERY(1)”，则生成的分区是：[1, 2), [2, 3) 和 [3, 4)，名称依次为p1_1, p1_2和p1_3，即此处的p1是名称前缀。
    - 若该定义是第一个分区定义，且该定义有START值，则范围（MINVALUE, START）将自动作为第一个实际分区，其名称为p1_0，然后该定义语义描述的分区名称依次为p1_1, p1_2, ...。例如对于完整定义“PARTITION p1 START(1), PARTITION p2 START(2)”，则生成的分区是：(MINVALUE, 1), [1, 2) 和 [2, MAXVALUE)，其名称依次为p1_0, p1_1和p2，即此处p1是名称前缀，p2是分区名称。这里MINVALUE表示最小值。
  - partition_value：范围分区的端点值（起始或终点），取值依赖于partition_key的类型，不可是MAXVALUE。

- interval_value：对[START，END) 表示的范围进行切分，interval_value是指定切分后每个分区的宽度，不可是MAXVALUE；如果（END-START）值不能整除以EVERY值，则仅最后一个分区的宽度小于EVERY值。
- MAXVALUE：表示最大值，它通常用于设置最后一个范围分区的上边界。

须知

1. 在创建分区表若第一个分区定义含START值，则范围（MINVALUE，START）将自动作为实际的第一个分区。
2. START END语法需要遵循以下限制：
   - 每个partition_start_end_item中的START值（如果有的话，下同）必须小于其END值；
   - 相邻的两个partition_start_end_item，第一个的END值必须等于第二个的START值；
   - 每个partition_start_end_item中的EVERY值必须是正向递增的，且必须小于（END-START）值；
   - 每个分区包含起始值，不包含终点值，即形如：[起始值，终点值)，起始值是MINVALUE时则不包含；
   - 一个partition_start_end_item创建的每个分区所属的TABLESPACE一样；
   - partition_name作为分区名称前缀时，其长度不要超过57字节，超过时自动截断；
   - 在创建、修改分区表时请注意分区表的分区总数不可超过最大限制（1048575）；
3. 在创建分区表时START END与LESS THAN语法不可混合使用。
4. 即使创建分区表时使用START END语法，备份（gs_dump）出的SQL语句也是VALUES LESS THAN语法格式。

- **INTERVAL ('interval_expr') [STORE IN (tablespace_name [, ...])]**

  间隔分区定义信息。
  - interval_expr：自动创建分区的间隔，例如：1 day、1 month。
  - STORE IN (tablespace_name [, ...])：指定存放自动创建分区的表空间列表，如果有指定，则自动创建的分区从表空间列表中循环选择使用，否则使用分区表默认的表空间。
- **PARTITION BY LIST [COLUMNS] (partition_key)**

  创建列表分区。partition_key为分区键的名称。

  COLUMNS关键字只能在sql_compatibility='B'时使用，“PARTITION BY LIST COLUMNS” 语义同 “PARTITION BY LIST”。
  - 对于partition_key，列表分区策略的分区键最多支持16列。
  - 对于从句是VALUES [IN] (list_values)的语法格式，list_values中包含了对应分区存在的键值，每个分区的键值数量不超过64个。
  - 从句"VALUES IN"只能在sql_compatibility='B'时使用，语义同"VALUES"。

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

- **PARTITION BY HASH(partition_key)**

  创建哈希分区。partition_key为分区键的名称。

  对于partition_key，哈希分区策略的分区键仅支持1列。

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

- **PARTITION BY KEY(partition_key)**

  只能在sql_compatibility='B'时使用，语义同“PARTITION BY HASH(partition_key)”。

- **PARTITIONS integer**

  指定分区个数。

  integer为分区数，必须为大于0的整数，且不得大于1048575。

  - 当在RANGE和LIST分区后指定此子句时，必须显式定义每个分区，且定义分区的数量必须与integer值相等。只能在sql_compatibility='B'时在RANGE和LIST分区后指定此子句。
  - 当在HASH和KEY分区后指定此子句时，若不列出各个分区定义，将自动生成integer个分区，自动生成的分区名为“p+数字”，数字依次为0到integer-1，分区的表空间默认为此表的表空间；也可以显式列出每个分区定义，此时定义分区的数量必须与integer值相等。若既不列出分区定义，也不指定分区数量，将创建唯一一个分区。

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

  行迁移开关。

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

  取值范围：

  - ENABLE（缺省值）：行迁移开关打开。
  - DISABLE：行迁移开关关闭。

  在打开行迁移开关情况下，并发UPDATE、DELETE操作可能会报错，原因如下：

  UPDATE和DELETE操作对于旧数据都是标记为已删除。在打开行迁移开关情况下，如果更新分区键时，导致了跨分区更新，内核会把旧分区中旧数据标记为已删除，在新分区中新增加一条数据，无法通过旧数据找到新数据。

  在UPDATE和UPDATE并发、DELETE和DELETE并发、UPDATE和DELETE并发三个并发场景下，如果并发操作同一行数据时，数据跨分区和非跨分区结果有不同的行为。

  a. 对于数据非跨分区结果，第一个操作执行完后，第二个操作不会报错。

     - 如果第一个操作是UPDATE，第二个操作能成功找到最新的数据，之后对新数据操作。
     - 如果第一个操作是DELETE，第二个操作看到当前数据已经被删除而且找不到最新数据，就终止操作。

  b. 对于数据跨分区结果，第一个操作执行完后，第二个操作会报错。

     - 如果第一个操作是UPDATE，由于新数据在新分区中，第二个操作不能成功找到最新的数据，就无法操作，之后会报错。

    - 如果第一个操作是DELETE，第二个操作看到当前数据已经被删除而且找不到最新数据，但无法判断删除旧数据的操作是UPDATE还是DELETE。如果是UPDATE，报错处理。如果是DELETE，终止操作。为了保持数据的正确性，只能报错处理。

  如果是UPDATE和UPDATE并发，UPDATE和DELETE并发场景，需要串行执行才能解决问题，如果是DELETE和DELETE并发，关闭行迁移开关可以解决问题。

- **NOT NULL**

  字段值不允许为NULL。ENABLE用于语法兼容，可省略。

- **NULL**

  字段值允许NULL ，这是缺省。

  这个子句只是为和非标准SQL数据库兼容。不建议使用。

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

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

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

  用NO INHERIT标记的约束将不会传递到子表中去。

  ENABLE用于语法兼容，可省略。

- **DEFAULT default_expr**

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

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

- **GENERATED ALWAYS AS (generation_expr) [STORED]**

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

  说明

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

- **AUTO_INCREMENT**

  指定列为自动增长列。

  详见：**•AUTO_INCREMENT**。

- **UNIQUE [KEY] index_parameters**

  UNIQUE (column_name [, ...]) index_parameters

  UNIQUE约束表示表里的一个字段或多个字段的组合必须在全表范围内唯一。

  对于唯一约束，NULL被认为是互不相等的。

  UNIQUE KEY只能在sql_compatibility='B'时使用，与UNIQUE语义相同。

- **PRIMARY KEY index_parameters**

  PRIMARY KEY (column_name [, ...]) index_parameters

  主键约束声明表中的一个或者多个字段只能包含唯一的非NULL值。

  一个表只能声明一个主键。

- **DEFERRABLE | NOT DEFERRABLE**

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

- **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

  如果约束是可推迟的，则这个子句声明检查约束的缺省时间。

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

  约束检查的时间可以用SET CONSTRAINTS命令修改。

- **USING INDEX TABLESPACE tablespace_name**

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

示例

- 示例1：创建范围分区表tpcds.web_returns_p1，含有8个分区，分区键为integer类型。 分区的范围分别为：wr_returned_date_sk< 2450815，2450815<= wr_returned_date_sk< 2451179，2451179<=wr_returned_date_sk< 2451544，2451544 <= wr_returned_date_sk< 2451910，2451910 <= wr_returned_date_sk< 2452275，2452275 <= wr_returned_date_sk< 2452640，2452640 <= wr_returned_date_sk< 2453005，wr_returned_date_sk>=2453005。

```
--创建SCHEMA。
gaussdb=# CREATE SCHEMA tpcds;

--创建表tpcds.web_returns。
gaussdb=# CREATE TABLE tpcds.web_returns
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
```



```
    W_COUNTRY                 VARCHAR(20)                 ,
    W_GMT_OFFSET              DECIMAL(5,2)
);
--创建分区表tpcds.web_returns_p1。
gaussdb=# CREATE TABLE tpcds.web_returns_p1
(
    WR_RETURNED_DATE_SK       INTEGER                     ,
    WR_RETURNED_TIME_SK       INTEGER                     ,
    WR_ITEM_SK                INTEGER               NOT NULL,
    WR_REFUNDED_CUSTOMER_SK   INTEGER                     ,
    WR_REFUNDED_CDEMO_SK      INTEGER                     ,
    WR_REFUNDED_HDEMO_SK      INTEGER                     ,
    WR_REFUNDED_ADDR_SK       INTEGER                     ,
    WR_RETURNING_CUSTOMER_SK  INTEGER                     ,
    WR_RETURNING_CDEMO_SK     INTEGER                     ,
    WR_RETURNING_HDEMO_SK     INTEGER                     ,
    WR_RETURNING_ADDR_SK      INTEGER                     ,
    WR_WEB_PAGE_SK            INTEGER                     ,
    WR_REASON_SK              INTEGER                     ,
    WR_ORDER_NUMBER           BIGINT                NOT NULL,
    WR_RETURN_QUANTITY        INTEGER                     ,
    WR_RETURN_AMT             DECIMAL(7,2)                ,
    WR_RETURN_TAX             DECIMAL(7,2)                ,
    WR_RETURN_AMT_INC_TAX     DECIMAL(7,2)                ,
    WR_FEE                    DECIMAL(7,2)                ,
    WR_RETURN_SHIP_COST       DECIMAL(7,2)                ,
    WR_REFUNDED_CASH          DECIMAL(7,2)                ,
    WR_REVERSED_CHARGE        DECIMAL(7,2)                ,
    WR_ACCOUNT_CREDIT         DECIMAL(7,2)                ,
    WR_NET_LOSS               DECIMAL(7,2)
)
PARTITION BY RANGE(WR_RETURNED_DATE_SK)
(
        PARTITION P1 VALUES LESS THAN(2450815),
        PARTITION P2 VALUES LESS THAN(2451179),
        PARTITION P3 VALUES LESS THAN(2451544),
        PARTITION P4 VALUES LESS THAN(2451910),
        PARTITION P5 VALUES LESS THAN(2452275),
        PARTITION P6 VALUES LESS THAN(2452640),
        PARTITION P7 VALUES LESS THAN(2453005),
        PARTITION P8 VALUES LESS THAN(MAXVALUE)
);

--从示例数据表导入数据。
gaussdb=# INSERT INTO tpcds.web_returns_p1 SELECT * FROM tpcds.web_returns;

--删除分区P8。
gaussdb=# ALTER TABLE tpcds.web_returns_p1 DROP PARTITION P8;

--增加分区WR_RETURNED_DATE_SK介于2453005和2453105之间。
gaussdb=# ALTER TABLE tpcds.web_returns_p1 ADD PARTITION P8 VALUES LESS THAN (2453105);

--增加分区WR_RETURNED_DATE_SK介于2453105和MAXVALUE之间。
gaussdb=# ALTER TABLE tpcds.web_returns_p1 ADD PARTITION P9 VALUES LESS THAN (MAXVALUE);

--删除分区P8。
gaussdb=# gaussdb=#

--分区P7重命名为P10。
gaussdb=# ALTER TABLE tpcds.web_returns_p1 RENAME PARTITION P7 TO P10;

--分区P6重命名为P11。
gaussdb=# ALTER TABLE tpcds.web_returns_p1 RENAME PARTITION FOR (2452639) TO P11;

--查询分区P10的行数。
gaussdb=# SELECT count(*) FROM tpcds.web_returns_p1 PARTITION (P10);
 count
--------
 0
```



```
(1 row)

--查询分区P1的行数。
gaussdb=# SELECT COUNT(*) FROM tpcds.web_returns_p1 PARTITION FOR (2450815);
 count
--------
 0
(1 row)

--删除表tpcds.web_returns_p1。
gaussdb=# DROP TABLE tpcds.web_returns_p1;

--删除表tpcds.web_returns。
gaussdb=# DROP TABLE tpcds.web_returns;

--删除SCHEMA。
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

- 示例2：创建范围分区表tpcds.web_returns_p2，含有8个分区，分区键类型为integer类型，其中第8个分区上边界为MAXVALUE。

  八个分区的范围分别为： wr_returned_date_sk< 2450815，2450815<= wr_returned_date_sk< 2451179，2451179<=wr_returned_date_sk< 2451544，2451544 <= wr_returned_date_sk< 2451910，2451910 <= wr_returned_date_sk< 2452275，2452275 <= wr_returned_date_sk< 2452640，2452640 <= wr_returned_date_sk< 2453005，wr_returned_date_sk>=2453005。

  分区表tpcds.web_returns_p2的表空间为example1；分区P1到P7没有声明表空间，使用采用分区表tpcds.web_returns_p2的表空间example1；指定分区P8的表空间为example2。

  假定数据库节点的数据目录/pg_location/mount1/path1，数据库节点的数据目录/pg_location/mount2/path2，数据库节点的数据目录/pg_location/mount3/path3，数据库节点的数据目录/pg_location/mount4/path4是dwsadmin用户拥有读写权限的空目录。

```
gaussdb=# CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace1/tablespace_1';
gaussdb=# CREATE TABLESPACE example2 RELATIVE LOCATION 'tablespace2/tablespace_2';
gaussdb=# CREATE TABLESPACE example3 RELATIVE LOCATION 'tablespace3/tablespace_3';
gaussdb=# CREATE TABLESPACE example4 RELATIVE LOCATION 'tablespace4/tablespace_4';

--创建SCHEMA。
gaussdb=# CREATE SCHEMA tpcds;

gaussdb=# CREATE TABLE tpcds.web_returns_p2
(
    WR_RETURNED_DATE_SK       INTEGER                   ,
    WR_RETURNED_TIME_SK       INTEGER                   ,
    WR_ITEM_SK                INTEGER             NOT NULL,
    WR_REFUNDED_CUSTOMER_SK   INTEGER                     ,
    WR_REFUNDED_CDEMO_SK      INTEGER                     ,
    WR_REFUNDED_HDEMO_SK      INTEGER                     ,
    WR_REFUNDED_ADDR_SK       INTEGER                     ,
    WR_RETURNING_CUSTOMER_SK  INTEGER                     ,
    WR_RETURNING_CDEMO_SK     INTEGER                     ,
    WR_RETURNING_HDEMO_SK     INTEGER                     ,
    WR_RETURNING_ADDR_SK      INTEGER                     ,
    WR_WEB_PAGE_SK            INTEGER                   ,
    WR_REASON_SK              INTEGER                   ,
    WR_ORDER_NUMBER           BIGINT              NOT NULL,
    WR_RETURN_QUANTITY        INTEGER                   ,
    WR_RETURN_AMT             DECIMAL(7,2)              ,
    WR_RETURN_TAX             DECIMAL(7,2)              ,
    WR_RETURN_AMT_INC_TAX     DECIMAL(7,2)              ,
    WR_FEE                    DECIMAL(7,2)              ,
    WR_RETURN_SHIP_COST       DECIMAL(7,2)              ,
    WR_REFUNDED_CASH          DECIMAL(7,2)              ,
```



```
    WR_REVERSED_CHARGE        DECIMAL(7,2)                ,
    WR_ACCOUNT_CREDIT         DECIMAL(7,2)                ,
    WR_NET_LOSS               DECIMAL(7,2)
)
TABLESPACE example1
PARTITION BY RANGE(WR_RETURNED_DATE_SK)
(
        PARTITION P1 VALUES LESS THAN(2450815),
        PARTITION P2 VALUES LESS THAN(2451179),
        PARTITION P3 VALUES LESS THAN(2451544),
        PARTITION P4 VALUES LESS THAN(2451910),
        PARTITION P5 VALUES LESS THAN(2452275),
        PARTITION P6 VALUES LESS THAN(2452640),
        PARTITION P7 VALUES LESS THAN(2453005),
        PARTITION P8 VALUES LESS THAN(MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;

--以like方式创建一个分区表。
gaussdb=# CREATE TABLE tpcds.web_returns_p3 (LIKE tpcds.web_returns_p2 INCLUDING PARTITION);

--修改分区P1的表空间为example2。
gaussdb=# ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P1 TABLESPACE example2;

--修改分区P2的表空间为example3。
gaussdb=# ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P2 TABLESPACE example3;

--以2453010为分割点切分P8。
gaussdb=# ALTER TABLE tpcds.web_returns_p2 SPLIT PARTITION P8 AT (2453010) INTO
(
        PARTITION P9,
        PARTITION P10
);

--将P6，P7合并为一个分区。
gaussdb=# ALTER TABLE tpcds.web_returns_p2 MERGE PARTITIONS P6, P7 INTO PARTITION P8;

--修改分区表迁移属性。
gaussdb=# ALTER TABLE tpcds.web_returns_p2 DISABLE ROW MOVEMENT;
--删除表和表空间。
gaussdb=# DROP TABLE tpcds.web_returns_p1;
gaussdb=# DROP TABLE tpcds.web_returns_p2;
gaussdb=# DROP TABLE tpcds.web_returns_p3;
gaussdb=# DROP TABLESPACE example1;
gaussdb=# DROP TABLESPACE example2;
gaussdb=# DROP TABLESPACE example3;
gaussdb=# DROP TABLESPACE example4;

--删除SCHEMA。
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

- 示例3：START END语法创建、修改Range分区表。

  假定/home/omm/startend_tbs1，/home/omm/startend_tbs2，/home/omm/startend_tbs3，/home/omm/startend_tbs4是omm用户拥有读写权限的空目录。

```
-- 创建表空间
gaussdb=# CREATE TABLESPACE startend_tbs1 LOCATION '/home/omm/startend_tbs1';
gaussdb=# CREATE TABLESPACE startend_tbs2 LOCATION '/home/omm/startend_tbs2';
gaussdb=# CREATE TABLESPACE startend_tbs3 LOCATION '/home/omm/startend_tbs3';
gaussdb=# CREATE TABLESPACE startend_tbs4 LOCATION '/home/omm/startend_tbs4';

-- 创建临时schema
gaussdb=# CREATE SCHEMA tpcds;
gaussdb=# SET CURRENT_SCHEMA TO tpcds;

-- 创建分区表，分区键是integer类型
gaussdb=# CREATE TABLE tpcds.startend_pt (c1 INT, c2 INT)
TABLESPACE startend_tbs1
```



```
PARTITION BY RANGE (c2) (
    PARTITION p1 START(1) END(1000) EVERY(200) TABLESPACE startend_tbs2,
    PARTITION p2 END(2000),
    PARTITION p3 START(2000) END(2500) TABLESPACE startend_tbs3,
    PARTITION p4 START(2500),
    PARTITION p5 START(3000) END(5000) EVERY(1000) TABLESPACE startend_tbs4
)
ENABLE ROW MOVEMENT;

-- 查看分区表信息
gaussdb=# SELECT relname, boundaries, spcname FROM pg_partition p JOIN pg_tablespace t ON
p.reltablespace=t.oid and p.parentid='tpcds.startend_pt'::regclass ORDER BY 1;
   relname   | boundaries |    spcname
-------------+------------+---------------
 p1_0        | {1}        | startend_tbs2
 p1_1        | {201}      | startend_tbs2
 p1_2        | {401}      | startend_tbs2
 p1_3        | {601}      | startend_tbs2
 p1_4        | {801}      | startend_tbs2
 p1_5        | {1000}     | startend_tbs2
 p2          | {2000}     | startend_tbs1
 p3          | {2500}     | startend_tbs3
 p4          | {3000}     | startend_tbs1
 p5_1        | {4000}     | startend_tbs4
 p5_2        | {5000}     | startend_tbs4
 startend_pt |            | startend_tbs1
(12 rows)

-- 导入数据，查看分区数据量
gaussdb=# INSERT INTO tpcds.startend_pt VALUES (GENERATE_SERIES(0, 4999),
GENERATE_SERIES(0, 4999));
gaussdb=# SELECT COUNT(*) FROM tpcds.startend_pt PARTITION FOR (0);
 count
-------
     1
(1 row)

gaussdb=# SELECT COUNT(*) FROM tpcds.startend_pt PARTITION (p3);
 count
-------
   500
(1 row)

-- 增加分区: [5000, 5300), [5300, 5600), [5600, 5900), [5900, 6000)
gaussdb=# ALTER TABLE tpcds.startend_pt ADD PARTITION p6 START(5000) END(6000) EVERY(300)
TABLESPACE startend_tbs4;

-- 增加MAXVALUE分区: p7
gaussdb=# ALTER TABLE tpcds.startend_pt ADD PARTITION p7 END(MAXVALUE);

-- 重命名分区p7为p8
gaussdb=# ALTER TABLE tpcds.startend_pt RENAME PARTITION p7 TO p8;

-- 删除分区p8
gaussdb=# ALTER TABLE tpcds.startend_pt DROP PARTITION p8;

-- 重命名5950所在的分区为：p71
gaussdb=# ALTER TABLE tpcds.startend_pt RENAME PARTITION FOR(5950) TO p71;

-- 分裂4500所在的分区[4000, 5000)
gaussdb=# ALTER TABLE tpcds.startend_pt SPLIT PARTITION FOR(4500) INTO(PARTITION q1
START(4000) END(5000) EVERY(250) TABLESPACE startend_tbs3);

-- 修改分区p2的表空间为startend_tbs4
gaussdb=# ALTER TABLE tpcds.startend_pt MOVE PARTITION p2 TABLESPACE startend_tbs4;

-- 查看分区情形
gaussdb=# SELECT relname, boundaries, spcname FROM pg_partition p JOIN pg_tablespace t ON
p.reltablespace=t.oid and p.parentid='tpcds.startend_pt'::regclass ORDER BY 1;
```



```
  relname   | boundaries |    spcname
------------+------------+---------------
 p1_0       | {1}        | startend_tbs2
 p1_1       | {201}      | startend_tbs2
 p1_2       | {401}      | startend_tbs2
 p1_3       | {601}      | startend_tbs2
 p1_4       | {801}      | startend_tbs2
 p1_5       | {1000}     | startend_tbs2
 p2         | {2000}     | startend_tbs4
 p3         | {2500}     | startend_tbs3
 p4         | {3000}     | startend_tbs1
 p5_1       | {4000}     | startend_tbs4
 p6_1       | {5300}     | startend_tbs4
 p6_2       | {5600}     | startend_tbs4
 p6_3       | {5900}     | startend_tbs4
 p71        | {6000}     | startend_tbs4
 q1_1       | {4250}     | startend_tbs3
 q1_2       | {4500}     | startend_tbs3
 q1_3       | {4750}     | startend_tbs3
 q1_4       | {5000}     | startend_tbs3
 startend_pt |            | startend_tbs1
(19 rows)

-- 删除表和表空间
gaussdb=# DROP SCHEMA tpcds CASCADE;
gaussdb=# DROP TABLESPACE startend_tbs1;
gaussdb=# DROP TABLESPACE startend_tbs2;
gaussdb=# DROP TABLESPACE startend_tbs3;
gaussdb=# DROP TABLESPACE startend_tbs4;
```

- 示例4：创建间隔分区表sales，初始包含2个分区，分区键为DATE类型。 分区的范围分别为：time_id < '2019-02-01 00:00:00'，'2019-02-01 00:00:00' <= time_id < '2019-02-02 00:00:00' 。

```
--创建表sales
gaussdb=# CREATE TABLE sales
(prod_id NUMBER(6),
 cust_id NUMBER,
 time_id DATE,
 channel_id CHAR(1),
 promo_id NUMBER(6),
 quantity_sold NUMBER(3),
 amount_sold NUMBER(10,2)
)
PARTITION BY RANGE (time_id)
INTERVAL('1 day')
( PARTITION p1 VALUES LESS THAN ('2019-02-01 00:00:00'),
  PARTITION p2 VALUES LESS THAN ('2019-02-02 00:00:00')
);

-- 数据插入分区p1
gaussdb=# INSERT INTO sales VALUES(1, 12, '2019-01-10 00:00:00', 'a', 1, 1, 1);

-- 数据插入分区p2
gaussdb=# INSERT INTO sales VALUES(1, 12, '2019-02-01 00:00:00', 'a', 1, 1, 1);

-- 查看分区信息
gaussdb=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition t1, pg_class t2 WHERE
t1.parentid = t2.oid AND t2.relname = 'sales' AND t1.parttype = 'p';
 relname | partstrategy |        boundaries
---------+--------------+-------------------------
 p1      | r            | {"2019-02-01 00:00:00"}
 p2      | r            | {"2019-02-02 00:00:00"}
(2 rows)

-- 插入数据没有匹配的分区，新创建一个分区，并将数据插入该分区
-- 新分区的范围为 '2019-02-05 00:00:00' <= time_id < '2019-02-06 00:00:00'
gaussdb=# INSERT INTO sales VALUES(1, 12, '2019-02-05 00:00:00', 'a', 1, 1, 1);

-- 插入数据没有匹配的分区，新创建一个分区，并将数据插入该分区
```



```
-- 新分区的范围为 '2019-02-03 00:00:00' <= time_id < '2019-02-04 00:00:00'
gaussdb=# INSERT INTO sales VALUES(1, 12, '2019-02-03 00:00:00', 'a', 1, 1, 1);

-- 查看分区信息
gaussdb=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition t1, pg_class t2 WHERE
t1.parentid = t2.oid AND t2.relname = 'sales' AND t1.parttype = 'p';
 relname | partstrategy |       boundaries
---------+--------------+------------------------
 sys_p1  | i            | {"2019-02-06 00:00:00"}
 sys_p2  | i            | {"2019-02-04 00:00:00"}
 p1      | r            | {"2019-02-01 00:00:00"}
 p2      | r            | {"2019-02-02 00:00:00"}
(4 rows)
```

- 示例5：创建LIST分区表test_list，初始包含4个分区，分区键为INT类型。4个分区的范围分别为：2000，3000，4000，5000。

```
--创建表test_list
gaussdb=# create table test_list (col1 int, col2 int)
partition by list(col1)
(
partition p1 values (2000),
partition p2 values (3000),
partition p3 values (4000),
partition p4 values (5000)
);

-- 数据插入
gaussdb=# INSERT INTO test_list VALUES(2000, 2000);
INSERT 0 1
gaussdb=# INSERT INTO test_list VALUES(3000, 3000);
INSERT 0 1

-- 查看分区信息
gaussdb=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition t1, pg_class t2 WHERE
t1.parentid = t2.oid AND t2.relname = 'test_list' AND t1.parttype = 'p';
 relname | partstrategy | boundaries
---------+--------------+------------
 p1      | l            | {2000}
 p2      | l            | {3000}
 p3      | l            | {4000}
 p4      | l            | {5000}
(4 rows)

-- 插入数据没有匹配到分区，报错处理
gaussdb=# INSERT INTO test_list VALUES(6000, 6000);
ERROR:  inserted partition key does not map to any table partition

-- 添加分区
gaussdb=# alter table test_list add partition p5 values (6000);
ALTER TABLE
gaussdb=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition t1, pg_class t2 WHERE
t1.parentid = t2.oid AND t2.relname = 'test_list' AND t1.parttype = 'p';
 relname | partstrategy | boundaries
---------+--------------+------------
 p5      | l            | {6000}
 p4      | l            | {5000}
 p1      | l            | {2000}
 p2      | l            | {3000}
 p3      | l            | {4000}
(5 rows)
gaussdb=# INSERT INTO test_list VALUES(6000, 6000);
INSERT 0 1

-- 分区表和普通表交换数据
gaussdb=# create table t1 (col1 int, col2 int);
CREATE TABLE
gaussdb=# select * from test_list partition (p1);
 col1 | col2
------+------
 2000 | 2000
```



```
(1 row)
gaussdb=# alter table test_list exchange partition (p1) with table t1;
ALTER TABLE
gaussdb=# select * from test_list partition (p1);
 col1 | col2
------+------
(0 rows)
gaussdb=# select * from t1;
 col1 | col2
------+------
 2000 | 2000
(1 row)

-- truncate分区
gaussdb=# select * from test_list partition (p2);
 col1 | col2
------+------
 3000 | 3000
(1 row)
gaussdb=# alter table test_list truncate partition p2;
ALTER TABLE
gaussdb=# select * from test_list partition (p2);
 col1 | col2
------+------
(0 rows)

-- 删除分区
gaussdb=# alter table test_list drop partition p5;
ALTER TABLE
gaussdb=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition t1, pg_class t2 WHERE
t1.parentid = t2.oid AND t2.relname = 'test_list' AND t1.parttype = 'p';
 relname | partstrategy | boundaries
---------+--------------+------------
 p4      | l            | {5000}
 p1      | l            | {2000}
 p2      | l            | {3000}
 p3      | l            | {4000}
(4 rows)

gaussdb=# INSERT INTO test_list VALUES(6000, 6000);
ERROR:  inserted partition key does not map to any table partition

-- 合并分区
gaussdb=# alter table test_list merge partitions p1,p2 into partition p2;
ALTER TABLE
gaussdb=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition t1, pg_class t2 WHERE
t1.parentid = t2.oid AND t2.relname = 'test_list' AND t1.parttype = 'p';
 relname | partstrategy | boundaries
---------+--------------+------------
 p2      | l            | {2000,3000}
 p4      | l            | {5000}
 p3      | l            | {4000}
(3 rows)

-- 分割分区
gaussdb=# alter table test_list split partition p2 values(2000) into (partition p1, partition p2);
ALTER TABLE
gaussdb=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition t1, pg_class t2 WHERE
t1.parentid = t2.oid AND t2.relname = 'test_list' AND t1.parttype = 'p';
 relname | partstrategy | boundaries
---------+--------------+------------
 p2      | l            | {3000}
 p1      | l            | {2000}
 p4      | l            | {5000}
 p3      | l            | {4000}
(4 rows)

-- 删除分区表
gaussdb=# drop table test_list;
```


- 示例6：创建HASH分区表test_hash，初始包含2个分区，分区键为INT类型。

```
--创建表test_hash
gaussdb=# create table test_hash (col1 int, col2 int)
partition by hash(col1)
(
partition p1,
partition p2
);

-- 数据插入
gaussdb=# INSERT INTO test_hash VALUES(1, 1);
INSERT 0 1
gaussdb=# INSERT INTO test_hash VALUES(2, 2);
INSERT 0 1
gaussdb=# INSERT INTO test_hash VALUES(3, 3);
INSERT 0 1
gaussdb=# INSERT INTO test_hash VALUES(4, 4);
INSERT 0 1

-- 查看分区信息
gaussdb=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition t1, pg_class t2 WHERE
t1.parentid = t2.oid AND t2.relname = 'test_hash' AND t1.parttype = 'p';
 relname | partstrategy | boundaries
---------+--------------+------------
 p1      | h            | {0}
 p2      | h            | {1}
(2 rows)

-- 查看数据
gaussdb=# select * from test_hash partition (p1);
 col1 | col2
------+------
    3 |    3
    4 |    4
(2 rows)

gaussdb=# select * from test_hash partition (p2);
 col1 | col2
------+------
    1 |    1
    2 |    2
(2 rows)

-- 分区表和普通表交换数据
gaussdb=# create table t1 (col1 int, col2 int);
CREATE TABLE
gaussdb=# alter table test_hash exchange partition (p1) with table t1;
ALTER TABLE
gaussdb=# select * from test_hash partition (p1);
 col1 | col2
------+------
(0 rows)
gaussdb=# select * from t1;
 col1 | col2
------+------
    3 |    3
    4 |    4
(2 rows)

-- truncate分区
gaussdb=# alter table test_hash truncate partition p2;
ALTER TABLE
gaussdb=#  select * from test_hash partition (p2);
 col1 | col2
------+------
(0 rows)

-- 删除分区表
gaussdb=# drop table test_hash;
```


- 示例7：创建LIST分区表t_multi_keys_list，初始包含5个分区，两个分区键分别为INT类型和VARCHAR类型。

```
--创建表t_multi_keys_list
gaussdb=# CREATE TABLE t_multi_keys_list (a int, b varchar(4), c int)
PARTITION BY LIST (a,b)
(
    PARTITION p1 VALUES ( (0,NULL) ),
    PARTITION p2 VALUES ( (0,'1'), (0,'2'), (0,'3'), (1,'1'), (1,'2') ),
    PARTITION p3 VALUES ( (NULL,'0'), (2,'1') ),
    PARTITION p4 VALUES ( (3,'2'), (NULL,NULL) ),
    PARTITION pd VALUES ( DEFAULT )
);
```

相关链接

ALTER TABLE PARTITION，**DROP TABLE**

7.14.87 CREATE TABLESPACE

功能描述

在数据库中创建一个新的表空间。

注意事项

- 系统管理员或者继承了内置角色gs_role_tablespace权限的用户可以创建表空间。
- 不允许在一个事务块内部执行CREATE TABLESPACE。
- 执行CREATE TABLESPACE失败，如果内部创建目录（文件）操作成功了就会产生残留的目录（文件），重新创建时需要用户手动清理表空间指定的目录下残留的内容。如果在创建过程中涉及到数据目录下的表空间软连接残留，需要先将软连接的残留文件删除，再重新执行OM相关操作。
- CREATE TABLESPACE不支持两阶段事务，如果部分节点执行失败，不支持回滚。
- 在公有云场景下一般不建议用户使用自定义的表空间。原因：用户自定义表空间通常配合主存（即默认表空间所在的存储设备，如磁盘）以外的其它存储介质使用，以隔离不同业务可以使用的IO资源，而在公有云场景下，存储设备都是采用标准化的配置，无其它可用的存储介质，自定义表空间使用不当不利于系统长稳运行以及影响整体性能，因此建议使用默认表空间即可。

语法格式

```
CREATE TABLESPACE tablespace_name
    [ OWNER user_name ] [ RELATIVE ] LOCATION 'directory' [ MAXSIZE 'space_size' ]
    [with_option_clause];
```

其中普通表空间的with_option_clause为：

```
WITH ( {filesystem= { ' general ' | " general " | general } | address = { ' ip:port [, ... ] ' | " ip:port [, ... ] " } |
cfgpath = { ' path ' | " path " } | storepath = { ' rootpath ' | " rootpath " } | random_page_cost = { ' value ' | "
value " | value } | seq_page_cost = { ' value ' | " value " | value }}[, ... ])
```

参数说明

- **tablespace_name**

  要创建的表空间名称。

  表空间名称不能和数据库中的其他表空间重名，且名称不能以"pg"开头，这样的名称留给系统表空间使用。

取值范围：字符串，要符合**标识符命名规范**。

- **OWNER user_name**

  指定该表空间的所有者。缺省时，新表空间的所有者是当前用户。

  只有系统管理员可以创建表空间，但是可以通过OWNER子句把表空间的所有权赋给其他非系统管理员。

  取值范围：字符串，已存在的用户。

- **RELATIVE**

  若指定该参数，表示使用相对路径，LOCATION目录是相对于各个CN/DN数据目录下的。

  目录层次：数据目录/pg_location/相对路径。相对路径最多指定两层。

  若没有指定该参数，表示使用绝对表空间路径，LOCATION目录需要使用绝对路径。

- **LOCATION directory**

  用于表空间的目录。当创建绝对表空间路径时，对于目录有如下要求：

  - GaussDB系统用户必须对该目录拥有读写权限，并且目录为空。如果该目录不存在，将由系统自动创建。
  - 目录必须是绝对路径，目录中不得含有特殊字符（如$，> 等）。
  - 目录不允许指定在数据库数据目录下。
  - 目录需为本地路径。

  取值范围：字符串，有效的目录。

- **MAXSIZE 'space_size'**

  指定表空间在单个数据库节点上的最大值。

  取值范围：字符串格式为正整数+单位，单位当前支持K/M/G/T/P。解析后的数值以K为单位，且范围不能够超过64比特表示的有符号整数，即1KB~9007199254740991KB。

- **random_page_cost**

  指定随机读取page的开销。

  取值范围：0~1.79769e+308。

  默认值：使用GUC参数random_page_cost的值。

- **seq_page_cost**

  指定顺序读取page的开销。

  取值范围：0~1.79769e+308。

  默认值：使用GUC参数seq_page_cost的值。

示例

```
--创建表空间。
gaussdb=# CREATE TABLESPACE ds_location1 RELATIVE LOCATION 'tablespace/tablespace_1';

--创建用户joe。
gaussdb=# CREATE ROLE joe IDENTIFIED BY '********';

--创建用户jay。
gaussdb=# CREATE ROLE jay IDENTIFIED BY '********';

--创建表空间，且所有者指定为用户joe。
gaussdb=# CREATE TABLESPACE ds_location2 OWNER joe RELATIVE LOCATION 'tablespace/tablespace_1';
```



```
--把表空间ds_location1重命名为ds_location3。
gaussdb=# ALTER TABLESPACE ds_location1 RENAME TO ds_location3;

--改变表空间ds_location2的所有者。
gaussdb=# ALTER TABLESPACE ds_location2 OWNER TO jay;

--删除表空间。
gaussdb=# DROP TABLESPACE ds_location2;
gaussdb=# DROP TABLESPACE ds_location3;

--删除用户。
gaussdb=# DROP ROLE joe;
gaussdb=# DROP ROLE jay;
```

相关链接

CREATE DATABASE，**CREATE TABLE**，**CREATE INDEX**，**DROP TABLESPACE**，**ALTER TABLESPACE**

优化建议

- create tablespace

  不建议在事务内部创建表空间。

7.14.88 CREATE TABLE SUBPARTITION

功能描述

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

二级分区表的分区方案是由两个一级分区的分区方案组合而来的，一级分区的分区方案详见章节CREATE TABLE PARTITION。

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

注意事项

- 二级分区表有两个分区键，每个分区键只能支持1列。
- 唯一约束和主键约束的约束键包含所有分区键将为约束创建LOCAL索引，否则创建GLOBAL索引。如果指定创建local唯一索引，必须包含所有分区键。
- 创建二级分区表时，如果在其一级分区下不显示指定二级分区，会自动创建一个同范围的二级分区。
- 二级分区表的二级分区（叶子节点）个数不能超过1048575个，一级分区无限制，但一级分区下面至少有一个二级分区。
- 二级分区表的总分区数（包括一级分区和二级分区）最大值为1048575个，一般情况下业务不可能创建这么多分区，这样会导致内存不足。应参照参数local_syscache_threshold的值合理创建分区，二级分区表使用内存大致为（总分区数 * 3 / 1024）MB。理论上分区占用内存不允许大于local_syscache_threshold的值，同时还需要预留部分空间以供其他功能使用。

- 当分区数太多导致内存不足时，会间接导致性能急剧下降。
- 二级分区表不支持hashbucket。
- 不支持cluster。
- 指定分区查询时，如SELECT * FROM tablename PARTITION/SUBPARTITION(partitionname)，关键字PARTITION和SUBPARTITION注意不要写错。如果写错，查询不会报错，这时查询会变为对表起别名进行查询。
- 不支持密态数据库、行级访问控制。
- 对于二级分区表PARTITION/SUBPARTITION FOR (values)语法，values只能是常量。
- 对于二级分区表PARTITION/SUBPARTITION FOR (values)语法，values在需要数据类型转换时，建议使用强制类型转换，以防隐式类型转换结果与预期不符。
- 指定分区语句目前不能走全局索引扫描。

语法格式

```
CREATE TABLE [ IF NOT EXISTS ] subpartition_table_name
(
{ column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]
| table_constraint
| LIKE source_table [ like_option [...] ] }[, ... ]
)
[ AUTO_INCREMENT [ = ] value ]
[ WITH ( {storage_parameter = value} [, ... ] ) ]
[ COMPRESS | NOCOMPRESS ][ COMPRESS | NOCOMPRESS ]
[ TABLESPACE tablespace_name ]
PARTITION BY {RANGE [ COLUMNS ] | LIST [ COLUMNS ] | HASH | KEY} (partition_key) [ PARTITIONS
integer ] SUBPARTITION BY {RANGE | LIST | HASH | KEY} (subpartition_key) [ SUBPARTITIONS integer ]
(
  PARTITION partition_name1 [ VALUES LESS THAN {(val1) | MAXVALUE} | VALUES [IN] (val1[, …]) ]
[ TABLESPACE [=] tablespace ]
  [(
      { SUBPARTITION subpartition_name1 [ VALUES LESS THAN (val1_1) | VALUES (val1_1[, …])]
[ TABLESPACE [=] tablespace ] } [, ...]
  )][, ...]
)[ { ENABLE | DISABLE } ROW MOVEMENT ];
```

- 列约束column_constraint：

  ```
  [ CONSTRAINT constraint_name ]
  { NOT NULL |
    NULL |
    CHECK ( expression ) |
    DEFAULT default_e xpr |
    GENERATED ALWAYS AS ( generation_expr ) [STORED] |
    AUTO_INCREMENT |
    UNIQUE [KEY] index_parameters |
    PRIMARY KEY index_parameters |
    REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
        [ ON DELETE action ] [ ON UPDATE action ] }
  [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
  ```

- 表约束table_constraint：

  ```
  [ CONSTRAINT [ constraint_name ] ]
  { CHECK ( expression ) |
    UNIQUE  [ index_name ][ USING method ] ( { column_name [ ASC | DESC ] } [, ... ] )
  index_parameters |
    PRIMARY KEY  [ USING method ] ( { column_name [ ASC | DESC ] } [, ... ] ) index_parameters |
    FOREIGN KEY [ index_name ] ( column_name [, ... ] ) REFERENCES reftable [ ( refcolumn [, ... ] ) ]
       [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE action ] [ ON UPDATE
  action ] }
  [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
  ```

- like选项like_option：


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

- 索引存储参数index_parameters：

```
[ WITH ( {storage_parameter = value} [, ... ] ) ]
[ USING INDEX TABLESPACE tablespace_name ]
```

参数说明

- **IF NOT EXISTS**

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

- **subpartition_table_name**

  二级分区表的名称。

  取值范围：字符串，要符合**标识符命名规范**。

- **column_name**

  新表中要创建的字段名。

  取值范围：字符串，要符合**标识符命名规范**。

- **data_type**

  字段的数据类型。

- **COLLATE collation**

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

- **CONSTRAINT constraint_name**

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

  定义约束有两种方法：

  - 列约束：作为列定义的一部分，仅影响该列。
  - 表约束：作用于多个列。

  须知

  在B模式数据库下（即sql_compatibility = 'B'）constraint_name为可选项，在其他模式数据库下，必须加上constraint_name。

- **index_name**

  索引名。

须知

- index_name仅在B模式数据库下（即sql_compatibility = 'B'）支持，其他模式数据库下不支持。
- 对于外键约束，constraint_name和index_name同时指定时，索引名为constraint_name。
- 对于唯一键约束，constraint_name和index_name同时指定时，索引名以index_name。

- **USING method**

  指定创建索引的方法。

  取值范围参考**参数说明**中的USING method。

须知

- USING method仅在B模式数据库下（即sql_compatibility = 'B'）支持，其他模式数据库下不支持。
- 在B模式下，未指定USING method时，对于ASTORE的存储方式，默认索引方法为btree；对于USTORE的存储方式，默认索引方法为ubtree。

- **ASC | DESC**

  ASC表示指定按升序排序（默认）。DESC指定按降序排序。

须知

ASC|DESC只在B模式数据库下（即sql_compatibility = 'B'）支持，其他模式数据库不支持。

- **LIKE source_table [like_option ...]**

  LIKE子句声明一个表，新表自动从这个表里面继承所有字段名及其数据类型和非空约束。新表与原表之间在创建动作完毕之后是完全无关的。在原表做的任何修改都不会传播到新表中，并且也不可能在扫描原表的时候包含新表的数据。

  - 字段缺省表达式只有在声明了INCLUDING DEFAULTS之后才会包含进来。缺省是不包含缺省表达式的，即新表中所有字段的缺省值都是NULL。
  - 如果指定了INCLUDING GENERATED，则原表列的生成表达式会复制到新表中。默认不复制生成表达式。
  - 非空约束将总是复制到新表中，CHECK约束则仅在指定了INCLUDING CONSTRAINTS的时候才复制，而其他类型的约束则永远也不会被复制。此规则同时适用于表约束和列约束。
  - 被复制的列和约束并不使用相同的名称进行融合。如果明确的指定了相同的名称或者在另外一个LIKE子句中，将会报错。
  - 如果指定了INCLUDING INDEXES，则原表上的索引也将在新表上创建，默认不建立索引。
  - 如果指定了INCLUDING STORAGE，则原表列的STORAGE设置也将被拷贝，默认情况下不包含STORAGE设置。

  - 如果指定了INCLUDING COMMENTS，则原表列、约束和索引的注释也会被拷贝过来。默认情况下，不拷贝原表的注释。
  - 如果指定了INCLUDING RELOPTIONS，则原表的存储参数（即原表的WITH子句）也将拷贝至新表。默认情况下，不拷贝原表的存储参数。
  - INCLUDING ALL包含了INCLUDING DEFAULTS、INCLUDING CONSTRAINTS、INCLUDING INDEXES、INCLUDING STORAGE、INCLUDING COMMENTS、INCLUDING PARTITION和INCLUDING RELOPTIONS的内容。

- **AUTO_INCREMENT [=] value**

  这个子句为自动增长列指定一个初始值，value必须为正整数，不得超过2^{127}-1。

  须知

  该子句仅在参数sql_compatibility='B'时有效。

- **WITH (storage_parameter [= value] [, ...])**

  这个子句为表或索引指定一个可选的存储参数。参数的详细描述如下所示：

  - FILLFACTOR

    一个表的填充因子（fillfactor）是一个介于10和100之间的百分数。在Ustore存储引擎下，默认值为92，在Astore存储引擎下默认值为100（完全填充）。如果指定了较小的填充因子，INSERT操作仅按照填充因子指定的百分率填充表页。每个页上的剩余空间将用于在该页上更新行，这就使得UPDATE有机会在同一页上放置同一条记录的新版本，这比把新版本放置在其他页上更有效。对于一个从不更新的表将填充因子设为100是最佳选择，但是对于频繁更新的表，选择较小的填充因子则更加合适。

    取值范围：10~100

  - ORIENTATION

    决定了表的数据的存储方式。

    取值范围：

    - ROW（缺省值）：表的数据将以行式存储。

      须知

      orientation不支持修改。

  - STORAGE_TYPE

    指定存储引擎类型，该参数设置成功后就不再支持修改。

    取值范围：

    - USTORE，表示表支持Inplace-Update存储引擎。特别需要注意，使用USTORE表，必须要开启track_counts和track_activities参数，否则会引起空间膨胀。
    - ASTORE，表示表支持Append-Only存储引擎。

    默认值：

    不指定表时，默认是Inplace-Update存储。

  - COMPRESSION
    - 行存表不支持压缩。
  - segment
    预留参数，暂不支持。

- **COMPRESS / NOCOMPRESS**
  创建一个新表时，需要在创建表语句中指定关键字COMPRESS，这样，当对该表进行批量插入时就会触发压缩特性。该特性会在页范围内扫描所有元组数据，生成字典、压缩元组数据并进行存储。指定关键字NOCOMPRESS则不对表进行压缩。行存表不支持压缩。
  缺省值为NOCOMPRESS，即不对元组数据进行压缩。
- **TABLESPACE tablespace_name**
  指定新表将要在tablespace_name表空间内创建。如果没有声明，将使用默认表空间。
- **PARTITION BY {RANGE [COLUMNS] | LIST [COLUMNS] | HASH | KEY} (partition_key)**
  - 对于partition_key，分区策略的分区键仅支持1列。
  - 分区键支持的数据类型和一级分区表约束保持一致。
  - COLUMNS关键字只能在sql_compatibility='B'时使用，只能加在RANGE或LIST之后，“RANGE COLUMNS” 语义同 “RANGE”，“LIST COLUMNS” 语义同 “LIST”。
  - KEY关键字只能在sql_compatibility='B'时使用，KEY与HASH同义。
- **SUBPARTITION BY {RANGE | LIST | HASH | KEY} (subpartition_key)**
  - 对于subpartition_key，分区策略的分区键仅支持1列。
  - 分区键支持的数据类型和一级分区表约束保持一致。
  - KEY关键字只能在sql_compatibility='B'时使用，KEY与HASH同义。
- **PARTITIONS integer**
  指定分区个数。
  integer为分区数，必须为大于0的整数，且不得大于1048575。
  - 当在RANGE和LIST分区后指定此子句时，必须显式定义每个分区，且定义分区的数量必须与integer值相等。只能在sql_compatibility='B'时在RANGE和LIST分区后指定此子句。
  - 当在HASH和KEY分区后指定此子句时，若不列出各个分区定义，将自动生成integer个分区，自动生成的分区名为“p+数字”，数字依次为0到integer-1，分区的表空间默认为此表的表空间；也可以显式列出每个分区定义，此时定义分区的数量必须与integer值相等。若既不列出分区定义，也不指定分区数量，将创建唯一一个分区。
- **SUBPARTITIONS integer**
  指定二级分区数量。
  integer为二级分区个数，必须为大于0的整数，且不得大于1048575。
  - 只能在HASH和KEY二级分区后指定此子句。
    - 若不列出各个二级分区定义，将在每个一级分区内自动生成integer个二级分区，自动生成的二级分区名为“一级分区名+sp+数字”，数字依次为0到integer-1，分区的表空间默认为此表的表空间。

  - 也可以列出每个二级分区定义，此时二级分区的数量必须与integer值相等。
  - 若既不列出每个二级分区定义，也不指定二级分区数量，将创建唯一一个二级分区。

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

  行迁移开关。

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

  取值范围：

  - ENABLE（缺省值）：行迁移开关打开。
  - DISABLE：行迁移开关关闭。

  在打开行迁移开关情况下，并发update、delete操作可能会报错，原因如下：

  update和delete操作对于旧数据都是标记为已删除。在打开行迁移开关情况下，如果更新分区键时，导致了跨分区更新，内核会把旧分区中旧数据标记为已删除，在新分区中新增加一条数据，无法通过旧数据找到新数据。

  在以下三个并发场景下，UPDATE和UPDATE并发、DELETE和DELETE并发和UPDATE和DELETE并发，如果并发操作同一行数据时，数据跨分区和非跨分区结果有不同的行为。

  a. 对于数据非跨分区结果，第一个操作执行完后，第二个操作不会报错。
     - 如果第一个操作是UPDATE，第二个操作能成功找到最新的数据，之后对新数据操作。
     - 如果第一个操作是DELETE，第二个操作看到当前数据已经被删除而且找不到最新数据，就终止操作。

  b. 对于数据跨分区结果，第一个操作执行完后，第二个操作会报错。
     - 如果第一个操作是UPDATE，由于新数据在新分区中，第二个操作不能成功找到最新的数据，就无法操作，之后会报错。
     - 如果第一个操作是DELETE，第二个操作看到当前数据已经被删除而且找不到最新数据，但无法判断删除旧数据的操作是UPDATE还是DELETE。如果是UPDATE，报错处理。如果是DELETE，终止操作。为了保持数据的正确性，只能报错处理。

  如果是UPDATE和UPDATE并发，UPDATE和DELETE并发场景，需要串行执行才能解决问题，如果是DELETE和DELETE并发，关闭行迁移开关可以解决问题。

- **NOT NULL**

  字段值不允许为NULL。ENABLE用于语法兼容，可省略。

- **NULL**

  字段值允许NULL ，这是缺省。

  这个子句只是为和非标准SQL数据库兼容。不建议使用。

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

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

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

用NO INHERIT标记的约束将不会传递到子表中去。

ENABLE用于语法兼容，可省略。

- **DEFAULT default_expr**

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

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

- **GENERATED ALWAYS AS (generation_expr) [STORED]**

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

  说明

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

- **AUTO_INCREMENT**

  指定列为自动增长列。

  详见：**•AUTO_INCREMENT**。

- **UNIQUE [KEY] index_parameters**

  UNIQUE (column_name [, ...]) index_parameters

  UNIQUE约束表示表里的一个字段或多个字段的组合必须在全表范围内唯一。

  对于唯一约束，NULL被认为是互不相等的。

  UNIQUE KEY只能在sql_compatibility='B'时使用，与UNIQUE语义相同。

- **PRIMARY KEY index_parameters**

  PRIMARY KEY (column_name [, ...]) index_parameters

  主键约束声明表中的一个或者多个字段只能包含唯一的非NULL值。

  一个表只能声明一个主键。

- **DEFERRABLE | NOT DEFERRABLE**

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

- **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

  如果约束是可推迟的，则这个子句声明检查约束的缺省时间。

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

  约束检查的时间可以用SET CONSTRAINTS命令修改。

- **USING INDEX TABLESPACE tablespace_name**

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

示例

- 示例1：创建各种组合类型的二级分区表

```
CREATE TABLE list_list
(
   month_code VARCHAR2 ( 30 ) NOT NULL ,
   dept_code  VARCHAR2 ( 30 ) NOT NULL ,
   user_no    VARCHAR2 ( 30 ) NOT NULL ,
   sales_amt  int
)
PARTITION BY LIST (month_code) SUBPARTITION BY LIST (dept_code)
(
 PARTITION p_201901 VALUES ( '201902' )
 (
   SUBPARTITION p_201901_a VALUES ( '1' ),
   SUBPARTITION p_201901_b VALUES ( '2' )
 ),
 PARTITION p_201902 VALUES ( '201903' )
 (
   SUBPARTITION p_201902_a VALUES ( '1' ),
   SUBPARTITION p_201902_b VALUES ( '2' )
 )
);
insert into list_list values('201902', '1', '1', 1);
insert into list_list values('201902', '2', '1', 1);
insert into list_list values('201902', '1', '1', 1);
insert into list_list values('201903', '2', '1', 1);
insert into list_list values('201903', '1', '1', 1);
insert into list_list values('201903', '2', '1', 1);
select * from list_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 1         | 1       |         1
 201902     | 2         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(6 rows)

drop table list_list;
CREATE TABLE list_hash
(
   month_code VARCHAR2 ( 30 ) NOT NULL ,
   dept_code  VARCHAR2 ( 30 ) NOT NULL ,
   user_no    VARCHAR2 ( 30 ) NOT NULL ,
   sales_amt  int
)
PARTITION BY LIST (month_code) SUBPARTITION BY HASH (dept_code)
(
 PARTITION p_201901 VALUES ( '201902' )
 (
   SUBPARTITION p_201901_a,
   SUBPARTITION p_201901_b
 ),
 PARTITION p_201902 VALUES ( '201903' )
```



```
 (
   SUBPARTITION p_201902_a,
   SUBPARTITION p_201902_b
 )
);
insert into list_hash values('201902', '1', '1', 1);
insert into list_hash values('201902', '2', '1', 1);
insert into list_hash values('201902', '3', '1', 1);
insert into list_hash values('201903', '4', '1', 1);
insert into list_hash values('201903', '5', '1', 1);
insert into list_hash values('201903', '6', '1', 1);
select * from list_hash;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 4         | 1       |         1
 201903     | 5         | 1       |         1
 201903     | 6         | 1       |         1
 201902     | 2         | 1       |         1
 201902     | 3         | 1       |         1
 201902     | 1         | 1       |         1
(6 rows)

drop table list_hash;
CREATE TABLE list_range
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY LIST (month_code) SUBPARTITION BY RANGE (dept_code)
(
  PARTITION p_201901 VALUES ( '201902' )
  (
    SUBPARTITION p_201901_a values less than ('4'),
    SUBPARTITION p_201901_b values less than ('6')
  ),
  PARTITION p_201902 VALUES ( '201903' )
  (
    SUBPARTITION p_201902_a values less than ('3'),
    SUBPARTITION p_201902_b values less than ('6')
  )
);
insert into list_range values('201902', '1', '1', 1);
insert into list_range values('201902', '2', '1', 1);
insert into list_range values('201902', '3', '1', 1);
insert into list_range values('201903', '4', '1', 1);
insert into list_range values('201903', '5', '1', 1);
insert into list_range values('201903', '6', '1', 1);
ERROR:  inserted partition key does not map to any table partition
select * from list_range;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 4         | 1       |         1
 201903     | 5         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 2         | 1       |         1
 201902     | 3         | 1       |         1
(5 rows)

drop table list_range;
CREATE TABLE range_list
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY RANGE (month_code) SUBPARTITION BY LIST (dept_code)
```

```
(
 PARTITION p_201901 VALUES LESS THAN( '201903' )
 (
  SUBPARTITION p_201901_a values ('1'),
  SUBPARTITION p_201901_b values ('2')
 ),
 PARTITION p_201902 VALUES LESS THAN( '201904' )
 (
  SUBPARTITION p_201902_a values ('1'),
  SUBPARTITION p_201902_b values ('2')
 )
);
insert into range_list values('201902', '1', '1', 1);
insert into range_list values('201902', '2', '1', 1);
insert into range_list values('201902', '1', '1', 1);
insert into range_list values('201903', '2', '1', 1);
insert into range_list values('201903', '1', '1', 1);
insert into range_list values('201903', '2', '1', 1);
select * from range_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 2         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 1         | 1       |         1
(6 rows)

drop table range_list;
CREATE TABLE range_hash
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY RANGE (month_code) SUBPARTITION BY HASH (dept_code)
(
 PARTITION p_201901 VALUES LESS THAN( '201903' )
 (
  SUBPARTITION p_201901_a,
  SUBPARTITION p_201901_b
 ),
 PARTITION p_201902 VALUES LESS THAN( '201904' )
 (
  SUBPARTITION p_201902_a,
  SUBPARTITION p_201902_b
 )
);
insert into range_hash values('201902', '1', '1', 1);
insert into range_hash values('201902', '2', '1', 1);
insert into range_hash values('201902', '1', '1', 1);
insert into range_hash values('201903', '2', '1', 1);
insert into range_hash values('201903', '1', '1', 1);
insert into range_hash values('201903', '2', '1', 1);
select * from range_hash;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 2         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 1         | 1       |         1
(6 rows)

drop table range_hash;
CREATE TABLE range_range
```



```
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY RANGE (month_code) SUBPARTITION BY RANGE (dept_code)
(
  PARTITION p_201901 VALUES LESS THAN( '201903' )
  (
    SUBPARTITION p_201901_a VALUES LESS THAN( '2' ),
    SUBPARTITION p_201901_b VALUES LESS THAN( '3' )
  ),
  PARTITION p_201902 VALUES LESS THAN( '201904' )
  (
    SUBPARTITION p_201902_a VALUES LESS THAN( '2' ),
    SUBPARTITION p_201902_b VALUES LESS THAN( '3' )
  )
);
insert into range_range values('201902', '1', '1', 1);
insert into range_range values('201902', '2', '1', 1);
insert into range_range values('201902', '1', '1', 1);
insert into range_range values('201903', '2', '1', 1);
insert into range_range values('201903', '1', '1', 1);
insert into range_range values('201903', '2', '1', 1);
select * from range_range;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 2         | 1       |         1
 201903     | 1         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
(6 rows)

drop table range_range;
CREATE TABLE hash_list
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY hash (month_code) SUBPARTITION BY LIST (dept_code)
(
  PARTITION p_201901
  (
    SUBPARTITION p_201901_a VALUES ( '1' ),
    SUBPARTITION p_201901_b VALUES ( '2' )
  ),
  PARTITION p_201902
  (
    SUBPARTITION p_201902_a VALUES ( '1' ),
    SUBPARTITION p_201902_b VALUES ( '2' )
  )
);
insert into hash_list values('201901', '1', '1', 1);
insert into hash_list values('201901', '2', '1', 1);
insert into hash_list values('201901', '1', '1', 1);
insert into hash_list values('201903', '2', '1', 1);
insert into hash_list values('201903', '1', '1', 1);
insert into hash_list values('201903', '2', '1', 1);
select * from hash_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 1         | 1       |         1
```



```
 201901     | 2         | 1       |         1
 201901     | 1         | 1       |         1
 201901     | 1         | 1       |         1
(6 rows)

drop table hash_list;
CREATE TABLE hash_hash
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY hash (month_code) SUBPARTITION BY hash (dept_code)
(
  PARTITION p_201901
  (
    SUBPARTITION p_201901_a,
    SUBPARTITION p_201901_b
  ),
  PARTITION p_201902
  (
    SUBPARTITION p_201902_a,
    SUBPARTITION p_201902_b
  )
);
insert into hash_hash values('201901', '1', '1', 1);
insert into hash_hash values('201901', '2', '1', 1);
insert into hash_hash values('201901', '1', '1', 1);
insert into hash_hash values('201903', '2', '1', 1);
insert into hash_hash values('201903', '1', '1', 1);
insert into hash_hash values('201903', '2', '1', 1);
select * from hash_hash;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 1         | 1       |         1
 201901     | 2         | 1       |         1
 201901     | 1         | 1       |         1
 201901     | 1         | 1       |         1
(6 rows)

drop table hash_hash;
CREATE TABLE hash_range
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY hash (month_code) SUBPARTITION BY range (dept_code)
(
  PARTITION p_201901
  (
    SUBPARTITION p_201901_a VALUES LESS THAN ( '2' ),
    SUBPARTITION p_201901_b VALUES LESS THAN ( '3' )
  ),
  PARTITION p_201902
  (
    SUBPARTITION p_201902_a VALUES LESS THAN ( '2' ),
    SUBPARTITION p_201902_b VALUES LESS THAN ( '3' )
  )
);
insert into hash_range values('201901', '1', '1', 1);
insert into hash_range values('201901', '2', '1', 1);
insert into hash_range values('201901', '1', '1', 1);
insert into hash_range values('201903', '2', '1', 1);
insert into hash_range values('201903', '1', '1', 1);
```



```
insert into hash_range values('201903', '2', '1', 1);
select * from hash_range;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 1         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
 201901     | 1         | 1       |         1
 201901     | 1         | 1       |         1
 201901     | 2         | 1       |         1
(6 rows)
```

- 示例2：对二级分区表进行DML指定分区操作

```
CREATE TABLE range_list
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY RANGE (month_code) SUBPARTITION BY LIST (dept_code)
(
  PARTITION p_201901 VALUES LESS THAN( '201903' )
  (
    SUBPARTITION p_201901_a values ('1'),
    SUBPARTITION p_201901_b values ('2')
  ),
  PARTITION p_201902 VALUES LESS THAN( '201910' )
  (
    SUBPARTITION p_201902_a values ('1'),
    SUBPARTITION p_201902_b values ('2')
  )
);
--指定一级分区插入数据
insert into range_list partition (p_201901) values('201902', '1', '1', 1);
--实际分区和指定分区不一致，报错
insert into range_list partition (p_201902) values('201902', '1', '1', 1);
ERROR:  inserted partition key does not map to the table partition
DETAIL:  N/A.
--指定二级分区插入数据
insert into range_list subpartition (p_201901_a) values('201902', '1', '1', 1);
--实际分区和指定分区不一致，报错
insert into range_list subpartition (p_201901_b) values('201902', '1', '1', 1);
ERROR:  inserted subpartition key does not map to the table subpartition
DETAIL:  N/A.
insert into range_list partition for ('201902') values('201902', '1', '1', 1);
insert into range_list subpartition for ('201902','1') values('201902', '1', '1', 1);

--指定分区查询数据
select * from range_list partition (p_201901);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(4 rows)

select * from range_list subpartition (p_201901_a);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(4 rows)

select * from range_list partition for ('201902');
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
```



```
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(4 rows)

select * from range_list subpartition for ('201902','1');
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(4 rows)

--指定分区更新数据
update range_list partition (p_201901) set user_no = '2';
select * from range_list;
select *from range_list; month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 2       |         1
 201902     | 1         | 2       |         1
 201902     | 1         | 2       |         1
 201902     | 1         | 2       |         1
(4 rows)
update range_list subpartition (p_201901_a) set user_no = '3';
select * from range_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 3       |         1
 201902     | 1         | 3       |         1
 201902     | 1         | 3       |         1
 201902     | 1         | 3       |         1
(4 rows)
update range_list partition for ('201902') set user_no = '4';
select * from range_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 4       |         1
 201902     | 1         | 4       |         1
 201902     | 1         | 4       |         1
 201902     | 1         | 4       |         1
(4 rows)
update range_list subpartition for ('201902','2') set user_no = '5';
gaussdb=# select *from range_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 4       |         1
 201902     | 1         | 4       |         1
 201902     | 1         | 4       |         1
 201902     | 1         | 4       |         1
(4 rows)
select * from range_list;

--指定分区删除数据
delete from range_list partition (p_201901);
DELETE 4
delete from range_list partition for ('201903');
DELETE 0
delete from range_list subpartition (p_201901_a);
DELETE 0
delete from range_list subpartition for ('201903','2');
DELETE 0
--参数sql_compatibility='B'时，可指定多分区删除数据
delete from range_list as t partition (p_201901_a, p_201901);
DELETE 0

--指定分区insert数据
insert into range_list partition (p_201901)  values('201902', '1', '1', 1)  ON DUPLICATE KEY UPDATE
```

```
sales_amt = 5;
insert into range_list subpartition (p_201901_a)  values('201902', '1', '1', 1)  ON DUPLICATE KEY
UPDATE sales_amt = 10;
insert into range_list partition for ('201902')  values('201902', '1', '1', 1)  ON DUPLICATE KEY UPDATE
sales_amt = 30;
insert into range_list subpartition for ('201902','1')  values('201902', '1', '1', 1)  ON DUPLICATE KEY
UPDATE sales_amt = 40;
select * from range_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(4 rows)

--指定分区merge into数据
CREATE TABLE newrange_list
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY RANGE (month_code) SUBPARTITION BY LIST (dept_code)
(
  PARTITION p_201901 VALUES LESS THAN( '201903' )
  (
    SUBPARTITION p_201901_a values ('1'),
    SUBPARTITION p_201901_b values ('2')
  ),
  PARTITION p_201902 VALUES LESS THAN( '201910' )
  (
    SUBPARTITION p_201902_a values ('1'),
    SUBPARTITION p_201902_b values ('2')
  )
);
insert into newrange_list values('201902', '1', '1', 1);
insert into newrange_list values('201903', '1', '1', 2);

MERGE INTO range_list partition (p_201901) p
USING newrange_list partition (p_201901) np
ON p.month_code= np.month_code
WHEN MATCHED THEN
  UPDATE SET dept_code = np.dept_code, user_no = np.user_no, sales_amt = np.sales_amt
WHEN NOT MATCHED THEN
  INSERT VALUES (np.month_code, np.dept_code, np.user_no, np.sales_amt);

select * from range_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(4 rows)

MERGE INTO range_list partition for ('201901') p
USING newrange_list partition for ('201901') np
ON p.month_code= np.month_code
WHEN MATCHED THEN
  UPDATE SET dept_code = np.dept_code, user_no = np.user_no, sales_amt = np.sales_amt
WHEN NOT MATCHED THEN
  INSERT VALUES (np.month_code, np.dept_code, np.user_no, np.sales_amt);

select * from range_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
```



```
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(4 rows)

MERGE INTO range_list subpartition (p_201901_a) p
USING newrange_list subpartition (p_201901_a) np
ON p.month_code= np.month_code
WHEN MATCHED THEN
  UPDATE SET dept_code = np.dept_code, user_no = np.user_no, sales_amt = np.sales_amt
WHEN NOT MATCHED THEN
  INSERT VALUES (np.month_code, np.dept_code, np.user_no, np.sales_amt);

select * from range_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(4 rows)

MERGE INTO range_list subpartition for ('201901', '1') p
USING newrange_list subpartition for ('201901', '1') np
ON p.month_code= np.month_code
WHEN MATCHED THEN
  UPDATE SET dept_code = np.dept_code, user_no = np.user_no, sales_amt = np.sales_amt
WHEN NOT MATCHED THEN
  INSERT VALUES (np.month_code, np.dept_code, np.user_no, np.sales_amt);

select * from range_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(4 rows)
```

- 示例3对二级分区表进行truncate操作

```
CREATE TABLE list_list
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY LIST (month_code) SUBPARTITION BY LIST (dept_code)
(
  PARTITION p_201901 VALUES ( '201902' )
  (
    SUBPARTITION p_201901_a VALUES ( '1' ),
    SUBPARTITION p_201901_b VALUES ( default )
  ),
  PARTITION p_201902 VALUES ( '201903' )
  (
    SUBPARTITION p_201902_a VALUES ( '1' ),
    SUBPARTITION p_201902_b VALUES ( '2' )
  )
);
insert into list_list values('201902', '1', '1', 1);
insert into list_list values('201902', '2', '1', 1);
insert into list_list values('201902', '1', '1', 1);
insert into list_list values('201903', '2', '1', 1);
insert into list_list values('201903', '1', '1', 1);
insert into list_list values('201903', '2', '1', 1);
select * from list_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 2         | 1       |         1
```



```
 201903     | 2         | 1       |         1
 201903     | 1         | 1       |         1
 201902     | 2         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(6 rows)

select * from list_list partition (p_201901);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 2         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(3 rows)

alter table list_list truncate partition p_201901;
select * from list_list partition (p_201901);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
(0 rows)

select * from list_list partition (p_201902);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 1         | 1       |         1
(3 rows)

alter table list_list truncate partition p_201902;
select * from list_list partition (p_201902);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
(0 rows)

select * from list_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
(0 rows)

insert into list_list values('201902', '1', '1', 1);
insert into list_list values('201902', '2', '1', 1);
insert into list_list values('201902', '1', '1', 1);
insert into list_list values('201903', '2', '1', 1);
insert into list_list values('201903', '1', '1', 1);
insert into list_list values('201903', '2', '1', 1);
select * from list_list subpartition (p_201901_a);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(2 rows)

alter table list_list truncate subpartition p_201901_a;
select * from list_list subpartition (p_201901_a);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
(0 rows)

select * from list_list subpartition (p_201901_b);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 2         | 1       |         1
(1 row)

alter table list_list truncate subpartition p_201901_b;
select * from list_list subpartition (p_201901_b);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
```

```
(0 rows)

select * from list_list subpartition (p_201902_a);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 1         | 1       |         1
(1 row)

alter table list_list truncate subpartition p_201902_a;
select * from list_list subpartition (p_201902_a);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
(0 rows)

select * from list_list subpartition (p_201902_b);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
(2 rows)

alter table list_list truncate subpartition p_201902_b;
select * from list_list subpartition (p_201902_b);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
(0 rows)

select * from list_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
(0 rows)

drop table list_list;
```

- 示例4：对二级分区表进行split操作

```
CREATE TABLE list_list
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY LIST (month_code) SUBPARTITION BY LIST (dept_code)
(
  PARTITION p_201901 VALUES ( '201902' )
  (
    SUBPARTITION p_201901_a VALUES ( '1' ),
    SUBPARTITION p_201901_b VALUES ( default )
  ),
  PARTITION p_201902 VALUES ( '201903' )
  (
    SUBPARTITION p_201902_a VALUES ( '1' ),
    SUBPARTITION p_201902_b VALUES ( default )
  )
);
insert into list_list values('201902', '1', '1', 1);
insert into list_list values('201902', '2', '1', 1);
insert into list_list values('201902', '1', '1', 1);
insert into list_list values('201903', '2', '1', 1);
insert into list_list values('201903', '1', '1', 1);
insert into list_list values('201903', '2', '1', 1);
select * from list_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 1         | 1       |         1
 201902     | 2         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
```



```
(6 rows)

select * from list_list subpartition (p_201901_a);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(2 rows)

select * from list_list subpartition (p_201901_b);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 2         | 1       |         1
(1 row)

alter table list_list split subpartition p_201901_b values (2) into
(
 subpartition p_201901_b,
 subpartition p_201901_c
);
select * from list_list subpartition (p_201901_a);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(2 rows)

select * from list_list subpartition (p_201901_b);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 2         | 1       |         1
(1 row)

select * from list_list subpartition (p_201901_c);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
(0 rows)

select * from list_list partition (p_201901);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 2         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(3 rows)

select * from list_list subpartition (p_201902_a);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 1         | 1       |         1
(1 row)

select * from list_list subpartition (p_201902_b);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
(2 rows)

alter table list_list split subpartition p_201902_b values (3) into
(
 subpartition p_201902_b,
 subpartition p_201902_c
);
select * from list_list subpartition (p_201902_a);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 1         | 1       |         1
(1 row)
```



```
select * from list_list subpartition (p_201902_b);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
(0 rows)

select * from list_list subpartition (p_201902_c);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
(2 rows)

drop table list_list;
```

7.14.89 CREATE TRIGGER

功能描述

创建一个触发器。 触发器将与指定的表或视图关联，并在特定条件下执行指定的函数。

注意事项

- 当前仅支持在普通行存表上创建触发器，不支持在临时表、unlogged表等类型表上创建触发器。
- 如果为同一事件定义了多个相同类型的触发器，则按触发器的名称字母顺序触发它们。
- 触发器常用于多表间数据关联同步场景，对SQL执行性能影响较大，不建议在大数据量同步及对性能要求高的场景中使用。
- 执行创建触发器操作的用户需要拥有指定表的TRIGGER权限或被授予了CREATE ANY TRIGGER权限。
- BEFORE触发的行级触发器函数可以返回一个NULL值，表示忽略对该行的操作，随后的触发器将不再执行，并且不会对该行产生INSERT/UPDATE/DELETE动作。AFTER触发器函数返回值无影响。
- DELETE的BEFORE触发器的情况下，触发器函数返回值NEW等于NULL；INSERT的BEFORE触发器的情况下，触发器函数返回值OLD等于NULL；UPDATE的BEFORE触发器的情况下，触发器函数返回值只有显示为NULL才是NULL值。
- 对于event为INSERT/UPDATE的触发器函数，正常返回值是NEW。如果返回一个非NULL的行，将修改那个插入或者更新的行。对于event为DELETE的触发器函数，正常返回值是OLD。
- INSTEAD OF触发器只能作用于视图，其触发器函数同样可以返回NULL值，表示随后的触发器将不再执行。

语法格式

```
CREATE [ CONSTRAINT ] TRIGGER name { BEFORE | AFTER | INSTEAD OF } { event [ OR ... ] }
   ON table_name
   [ FROM referenced_table_name ]
   { NOT DEFERRABLE | [ DEFERRABLE ] { INITIALLY IMMEDIATE | INITIALLY DEFERRED } }
   [ FOR [ EACH ] { ROW | STATEMENT } ]
   [ WHEN ( condition ) ]
   EXECUTE PROCEDURE function_name ( arguments );
```

其中event包含以下几种：

```
   INSERT
   UPDATE [ OF column_name [, ... ] ]
```



```
DELETE
TRUNCATE
```

参数说明

- **CONSTRAINT**

  可选项，指定此参数将创建约束触发器，即触发器作为约束来使用。除了可以使用SET CONSTRAINTS调整触发器触发的时间之外，这与常规触发器相同。 约束触发器必须是AFTER ROW触发器。

- **name**

  触发器名称，该名称不能限定模式，因为触发器自动继承其所在表的模式，且同一个表的触发器不能重名。 对于约束触发器，使用**SET CONSTRAINTS**修改触发器行为时也使用此名称。

  取值范围：符合**标识符命名规范**的字符串，且最大长度不超过63个字符。

- **BEFORE**

  触发器函数是在触发事件发生前执行。

- **AFTER**

  触发器函数是在触发事件发生后执行，约束触发器只能指定为AFTER。

- **INSTEAD OF**

  触发器函数直接替代触发事件。

- **event**

  启动触发器的事件，取值范围包括：INSERT、UPDATE、DELETE或TRUNCATE，也可以通过OR同时指定多个触发事件。

  对于UPDATE事件类型，可以使用下面语法指定列：

  ```
  UPDATE OF column_name1 [, column_name2 ... ]
  ```

  表示当这些列作为UPDATE语句的目标列时，才会启动触发器，但是INSTEAD OF UPDATE类型不支持指定列信息。如果UPDATE OF指定的列包含生成列，当生成列依赖的列是UPDATE语句的目标列时，也会启动触发器。

- **table_name**

  需要创建触发器的表名称。

  取值范围：数据库中已经存在的表名称。

- **referenced_table_name**

  约束引用的另一个表的名称。 只能为约束触发器指定，常见于外键约束。

  取值范围：数据库中已经存在的表名称。

- **DEFERRABLE | NOT DEFERRABLE**

  约束触发器的启动时机，仅作用于约束触发器。这两个关键字设置该约束是否可推迟。

  详细介绍请参见**CREATE TABLE**。

- **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

  如果约束是可推迟的，则这个子句声明检查约束的缺省时间，仅作用于约束触发器。

  详细介绍请参见**CREATE TABLE**。

- **FOR EACH ROW | FOR EACH STATEMENT**

  触发器的触发频率。

- FOR EACH ROW是指该触发器是受触发事件影响的每一行触发一次。
- FOR EACH STATEMENT是指该触发器是每个SQL语句只触发一次。

未指定时默认值为FOR EACH STATEMENT。约束触发器只能指定为FOR EACH ROW。

- **condition**

  决定是否实际执行触发器函数的条件表达式。当指定WHEN时，只有在条件返回true时才会调用该函数。

  在FOR EACH ROW触发器中，WHEN条件可以通过分别写入OLD.column_name或NEW.column_name来引用旧行或新行值的列。 INSERT触发器不能引用OLD和DELETE触发器不能引用NEW。

  INSTEAD OF触发器不支持WHEN条件。

  WHEN表达式不能包含子查询。

  对于约束触发器，WHEN条件的评估不会延迟，而是在执行更新操作后立即发生。 如果条件返回值不为true，则触发器不会排队等待延迟执行。

- **function_name**

  用户定义的函数，必须声明为不带参数并返回类型为触发器，在触发器触发时执行。

- **arguments**

  执行触发器时要提供给函数的可选的以逗号分隔的参数列表。参数是文字字符串常量，简单的名称和数字常量也可以写在这里，但它们都将被转换为字符串。 请检查触发器函数的实现语言的描述，以了解如何在函数内访问这些参数。

  说明

  关于触发器种类：

  - INSTEAD OF的触发器必须标记为FOR EACH ROW，并且只能在视图上定义。
  - BEFORE和AFTER触发器作用在视图上时，只能标记为FOR EACH STATEMENT。
  - TRUNCATE类型触发器仅限FOR EACH STATEMENT。

表 7-147 表和视图上支持的触发器种类：

触发时机	触发事件	行级	语句级
BEFORE	INSERT/UPDATE/DELETE	表	表和视图
	TRUNCATE	不支持	表
AFTER	INSERT/UPDATE/DELETE	表	表和视图
	TRUNCATE	不支持	表
INSTEAD OF	INSERT/UPDATE/DELETE	视图	不支持
	TRUNCATE	不支持	不支持

表 7-148 plpgsql 类型触发器函数特殊变量：

变量名	变量含义
NEW	INSERT及UPDATE操作涉及tuple信息中的新值，对DELETE为空。
OLD	UPDATE及DELETE操作涉及tuple信息中的旧值，对INSERT为空。
TG_NAME	触发器名称。
TG_WHEN	触发器触发时机（BEFORE/AFTER/INSTEAD OF）。
TG_LEVEL	触发频率（ROW/STATEMENT）。
TG_OP	触发操作（INSERT/UPDATE/DELETE/TRUNCATE）。
TG_RELID	触发器所在表OID。
TG_RELNAME	触发器所在表名（已废弃，现用TG_TABLE_NAME替代）。
TG_TABLE_NAME	触发器所在表名。
TG_TABLE_SCHEMA	触发器所在表的SCHEMA信息。
TG_NARGS	触发器函数参数个数。
TG_ARGV[]	触发器函数参数列表。

示例

```
--创建源表及触发表
gaussdb=# CREATE TABLE test_trigger_src_tbl(id1 INT, id2 INT, id3 INT);
gaussdb=# CREATE TABLE test_trigger_des_tbl(id1 INT, id2 INT, id3 INT);

--创建触发器函数
gaussdb=# CREATE OR REPLACE FUNCTION tri_insert_func() RETURNS TRIGGER AS
        $$
        DECLARE
        BEGIN
                INSERT INTO test_trigger_des_tbl VALUES(NEW.id1, NEW.id2, NEW.id3);
                RETURN NEW;
        END
        $$ LANGUAGE plpgsql;

gaussdb=# CREATE OR REPLACE FUNCTION tri_update_func() RETURNS TRIGGER AS
        $$
        DECLARE
        BEGIN
                UPDATE test_trigger_des_tbl SET id3 = NEW.id3 WHERE id1=OLD.id1;
                RETURN OLD;
        END
        $$ LANGUAGE plpgsql;

gaussdb=# CREATE OR REPLACE FUNCTION TRI_DELETE_FUNC() RETURNS TRIGGER AS
        $$
        DECLARE
        BEGIN
```

```
        DELETE FROM test_trigger_des_tbl WHERE id1=OLD.id1;
        RETURN OLD;
    END
    $$ LANGUAGE plpgsql;

--创建INSERT触发器
gaussdb=# CREATE TRIGGER insert_trigger
    BEFORE INSERT ON test_trigger_src_tbl
    FOR EACH ROW
    EXECUTE PROCEDURE tri_insert_func();

--创建UPDATE触发器
gaussdb=# CREATE TRIGGER update_trigger
    AFTER UPDATE ON test_trigger_src_tbl
    FOR EACH ROW
    EXECUTE PROCEDURE tri_update_func();

--创建DELETE触发器
gaussdb=# CREATE TRIGGER delete_trigger
    BEFORE DELETE ON test_trigger_src_tbl
    FOR EACH ROW
    EXECUTE PROCEDURE tri_delete_func();

--执行INSERT触发事件并检查触发结果
gaussdb=# INSERT INTO test_trigger_src_tbl VALUES(100,200,300);
gaussdb=# SELECT * FROM test_trigger_src_tbl;
gaussdb=# SELECT * FROM test_trigger_des_tbl;  //查看触发操作是否生效。

--执行UPDATE触发事件并检查触发结果
gaussdb=# UPDATE test_trigger_src_tbl SET id3=400 WHERE id1=100;
gaussdb=# SELECT * FROM test_trigger_src_tbl;
gaussdb=# SELECT * FROM test_trigger_des_tbl;  //查看触发操作是否生效

--执行DELETE触发事件并检查触发结果
gaussdb=# DELETE FROM test_trigger_src_tbl WHERE id1=100;
gaussdb=# SELECT * FROM test_trigger_src_tbl;
gaussdb=# SELECT * FROM test_trigger_des_tbl;  //查看触发操作是否生效

--修改触发器
gaussdb=# ALTER TRIGGER delete_trigger ON test_trigger_src_tbl RENAME TO delete_trigger_renamed;

--禁用insert_trigger触发器
gaussdb=# ALTER TABLE test_trigger_src_tbl DISABLE TRIGGER insert_trigger;

--禁用当前表上所有触发器
gaussdb=# ALTER TABLE test_trigger_src_tbl DISABLE TRIGGER ALL;

--删除触发器
gaussdb=# DROP TRIGGER insert_trigger ON test_trigger_src_tbl;
gaussdb=# DROP TRIGGER update_trigger ON test_trigger_src_tbl;
gaussdb=# DROP TRIGGER delete_trigger_renamed ON test_trigger_src_tbl;
```

相关链接

ALTER TRIGGER，**DROP TRIGGER**，**ALTER TABLE**

7.14.90 CREATE TYPE

功能描述

在当前数据库中定义一种新的数据类型。定义数据类型的用户将成为该数据类型的拥有者。类型只适用于行存表。

有五种形式的CREATE TYPE，分别为：复合类型、基本类型、shell类型、枚举类型和集合类型。

- 复合类型

  复合类型由一个属性名和数据类型的列表指定。如果属性的数据类型是可排序的，也可以指定该属性的排序规则。复合类型本质上和表的行类型相同，但是如果只想定义一种类型，使用CREATE TYPE避免了创建一个实际的表。单独的复合类型也是很有用的，例如可以作为函数的参数或者返回类型。

  为了能够创建复合类型，必须拥有在其所有属性类型上的USAGE特权。

- 基本类型

  用户可以自定义一种新的基本类型（标量类型）。通常来说这些函数必须是底层语言所编写。

- shell类型

  shell类型是一种用于后面要定义的类型的占位符，通过发出一个不带除类型名之外其他参数的CREATE TYPE命令可以创建这种类型。在创建基本类型时，需要shell类型作为一种向前引用。

- 枚举类型

  由若干个标签构成的列表，每一个标签值都是一个非空字符串，且字符串长度必须不超过63个字节。

- 集合类型

  类似数组，但是没有长度限制，主要在存储过程中使用。

- 被授予CREATE ANY TYPE权限的用户，可以在public模式和用户模式下创建类型。

注意事项

- 如果给定一个模式名，那么该类型将被创建在指定的模式中。否则它会被创建在当前模式中。类型名称必须与同一个模式中任何现有的类型或者域相区别（因为表具有相关的数据类型，类型名称也必须与同一个模式中任何现有表的名称不同）。
- 用户使用关联函数的方式创建非系统自带类型时，需要了解该类型定义及该类型所关联的函数。如果使用不当，可能会因为所关联的函数而产生权限被利用的风险。

语法格式

```
CREATE TYPE name AS
    ( [ attribute_name data_type [ COLLATE collation ] [, ... ] ] );

CREATE TYPE name (
    INPUT = input_function,
    OUTPUT = output_function
    [ , RECEIVE = receive_function ]
    [ , SEND = send_function ]
    [ , TYPMOD_IN =
type_modifier_input_function ]
    [ , TYPMOD_OUT =
type_modifier_output_function ]
    [ , ANALYZE = analyze_function ]
    [ , INTERNALLENGTH = { internallength |
VARIABLE } ]
    [ , PASSEDBYVALUE ]
    [ , ALIGNMENT = alignment ]
  [ , STORAGE = storage ]
    [ , LIKE = like_type ]
    [ , CATEGORY = category ]
    [ , PREFERRED = preferred ]
    [ , DEFAULT = default ]
```



```
    [ , ELEMENT = element ]
    [ , DELIMITER = delimiter ]
    [ , COLLATABLE = collatable ]
);

CREATE TYPE name;

CREATE TYPE name AS ENUM
    ( [ 'label' [, ... ] ] );

CREATE TYPE name AS TABLE OF data_type;
```

参数说明

复合类型

- **name**

  要创建的类型的名称（可以被模式限定）。

- **attribute_name**

  复合类型的一个属性（列）的名称。

- **data_type**

  要成为复合类型的一个列的现有数据类型的名称。可以使用%ROWTYPE间接引用表的类型，或者使用%TYPE间接引用表或复合类型中某一列的类型。

- **collation**

  要关联到复合类型的一列的现有排序规则的名称。排序规则可以使用“SELECT * FROM pg_collation”命令从pg_collation系统表中查询，默认的排序规则为查询结果中以default开始的行。

基本类型

自定义基本类型时，参数可以以任意顺序出现，input_function和output_function为必选参数，其它为可选参数。

- **input_function**

  将数据从类型的外部文本形式转换为内部形式的函数名。

  输入函数可以被声明为有一个cstring类型的参数，或者有三个类型分别为cstring、 oid、integer的参数。

  - cstring参数是以C字符串存在的输入文本。
  - oid参数是该类型自身的OID（对于数组类型则是其元素类型的OID）。
  - integer参数是目标列的typmod（如果知道，不知道则将传递 -1）。

  输入函数必须返回一个该数据类型本身的值。通常，一个输入函数应该被声明为STRICT。 如果不是这样，在读到一个NULL输入值时，调用输入函数时第一个参数会是NULL。在这种情况下，该函数必须仍然返回NULL，除非调用函数发生了错误（这种情况主要是想支持域输入函数，域输入函数可能需要拒绝NULL输入）。

说明

- 输入和输出函数能被声明为具有新类型的结果或参数是因为：必须在创建新类型之前创建这两个函数。而新类型应该首先被定义为一种shell type，它是一种占位符类型，除了名称和拥有者之外它没有其他属性。这可以通过不带额外参数的命令CREATE TYPE name做到。然后用C写的I/O函数可以被定义为引用这种shell type。最后，用带有完整定义的CREATE TYPE把该shell type替换为一个完全的、合法的类型定义，之后新类型就可以正常使用了。
- 输入和输出函数若为internal类型且指定为内部系统函数，则其输入函数和输出函数的参数类型需保持一致，且新类型的INTERNALLENGTH和PASSEDBYVALUE需要与输入函数和输出函数的参数类型保持一致。

- **output_function**

  将数据从类型的内部形式转换为外部文本形式的函数名。

  输出函数必须被声明为有一个新数据类型的参数。输出函数必须返回类型cstring。对于NULL值不会调用输出函数。

- **receive_function**

  可选参数。将数据从类型的外部二进制形式转换成内部形式的函数名。

  如果没有该函数，该类型不能参与到二进制输入中。二进制表达转换成内部形式代价更低，然而却更容易移植（例如，标准的整数数据类型使用网络字节序作为外部二进制表达，而内部表达是机器本地的字节序）。receive_function应该执行足够的检查以确保该值是有效的。

  接收函数可以被声明为有一个internal类型的参数，或者有三个类型分别为internal、oid、integer的参数。

  - internal参数是一个指向StringInfo缓冲区的指针，其中保存着接收到的字节串。
  - oid和integer参数和文本输入函数的相同。

  接收函数必须返回一个该数据类型本身的值。通常，一个接收函数应该被声明为STRICT。如果不是这样，在读到一个NULL输入值时调用接收函数时第一个参数会是NULL。在这种情况下，该函数必须仍然返回NULL，除非接收函数发生了错误（这种情况主要是想支持域接收函数，域接收函数可能需要拒绝NULL输入）。

- **send_function**

  可选参数。将数据从类型的内部形式转换为外部二进制形式的函数名。

  如果没有该函数，该类型将不能参与到二进制输出中。发送函数必须被声明为有一个新数据类型的参数。发送函数必须返回类型bytea。对于NULL值不会调用发送函数。

- **type_modifier_input_function**

  可选参数。将类型的修饰符数组转换为内部形式的函数名。

- **type_modifier_output_function**

  可选参数。将类型的修饰符的内部形式转换为外部文本形式的函数名。

说明

如果该类型支持修饰符（附加在类型声明上的可选约束，例如，char(5)或numeric(30,2)），则需要可选的type_modifier_input_function以及type_modifier_output_function。GaussDB允许用户定义的类型有一个或者多个简单常量或者标识符作为修饰符。不过，为了存储在系统目录中，该信息必须能被打包到一个非负整数值中。所声明的修饰符会被以cstring数组的形式传递给type_modifier_input_function。 type_modifier_input_function必须检查该值的合法性（如果值错误就抛出一个错误），如果值正确，要返回一个非负integer值，该值将被存储在“typmod”列中。如果类型没有 type_modifier_input_function则类型修饰符将被拒绝。type_modifier_output_function把内部的整数typmod值转换回正确的形式用于用户显示。type_modifier_output_function必须返回一个cstring值，该值就是追加到类型名称后的字符串。例如，numeric的函数可能会返回(30,2)。如果默认的显示格式就是只把存储的typmod整数值放在圆括号内，则允许省略type_modifier_output_function。

- **analyze_function**

  可选参数。为该数据类型执行统计分析的函数名的可选参数。

  默认情况下，如果该类型有一个默认的B-tree操作符类，ANALYZE将尝试用类型的“equals”和“less-than”操作符来收集统计信息。这种行为对于非标量类型并不合适，因此可以通过指定一个自定义分析函数来覆盖这种行为。分析函数必须被声明为有一个类型为internal的参数，并且返回一个boolean结果。

- **internallength**

  可选参数。一个数字常量，用于指定新类型的内部表达的字节长度。默认为变长。

  虽然只有I/O函数和其他为该类型创建的函数才知道新类型的内部表达的细节， 但是内部表达的一些属性必须被向GaussDB声明。其中最重要的是internallength。基本数据类型可以是定长的（这种情况下internallength是一个正整数）或者是变长的（把internallength设置为VARIABLE，在内部通过把typlen设置为-1表示）。所有变长类型的内部表达都必须以一个4字节整数开始，internallength定义了总长度。

- **PASSEDBYVALUE**

  可选参数。表示这种数据类型的值需要被传值而不是传引用。传值的类型必须是定长的，并且它们的内部表达不能超过Datum类型（某些机器上是4字节，其他机器上是8字节）的尺寸。

- **alignment**

  可选参数。该参数指定数据类型的存储对齐需求。如果被指定，必须是char、int2、int4或者double。默认是int4。

  允许的值等同于以1、2、4或8字节边界对齐。要注意变长类型的alignment参数必须至少为4，因为它们需要包含一个int4作为它们的第一个组成部分。

- **storage**

  可选参数。该数据类型的存储策略。

  如果被指定，必须是plain、external、extended或者main。 默认是plain。

  - plain指定该类型的数据将总是被存储在线内并且不会被压缩。（对定长类型只允许plain）
  - extended 指定系统将首先尝试压缩一个长的数据值，并且将在数据仍然太长的情况下把值移出主表行。
  - external允许值被移出主表， 但是系统将不会尝试对它进行压缩。
  - main允许压缩，但是不鼓励把值移出主表（如果没有其他办法让行的大小变得合适，具有这种存储策略的数据项仍将被移出主表，但比起extended以及external项来，这种存储策略的数据项会被优先考虑保留在主表中）。

除plain之外所有的storage值都暗示该数据类型的函数能处理被TOAST过的值。指定的值仅仅是决定一种可TOAST数据类型的列的默认TOAST存储策略，用户可以使用ALTER TABLE SET STORAGE为列选取其他策略。

- **like_type**

  可选参数。与新类型具有相同表达的现有数据类型的名称。会从这个类型中复制internallength、 passedbyvalue、 alignment以及storage的值（ 除非在这个CREATE TYPE命令的其他地方用显式说明覆盖）。

  当新类型的底层实现是以一种现有的类型为参考时，用这种方式指定表达特别有用。

- **category**

  可选参数。这种类型的分类码（一个ASCII 字符）。 默认是“用户定义类型”的'U'。为了创建自定义分类， 也可以选择其他 ASCII字符。

- **preferred**

  可选参数。如果这种类型是其类型分类中的优先类型则为TRUE，否则为FALSE。默认为假。在一个现有类型分类中创建一种新的优先类型要非常谨慎， 因为这可能会导致很大的改变。

  说明

  category和preferred参数可以被用来帮助控制在混淆的情况下应用哪一种隐式造型。每一种数据类型都属于一个用单个ASCII 字符命名的分类，并且每一种类型可以是其所属分类中的“首选”。当有助于解决重载函数或操作符时，解析器将优先造型到首选类型（但是只能从同类的其他类型造型）。对于没有隐式转换到或来自任意其他类型的类型，让这些设置保持默认即可。不过，对于有隐式转换的相关类型的组，把它们都标记为属于同一个类别并且选择一种或两种“最常用”的类型作为该类别的首选通常是很有用的。在把一种用户定义的类型增加到一个现有的内建类别（例如，数字或者字符串类型）中时，category参数特别有用。不过，也可以创建新的全部是用户定义类型的类别。对这样的类别，可选择除大写字母之外的任何ASCII 字符。

- **default**

  可选参数。数据类型的默认值。如果被省略，默认值是空。

  如果用户希望该数据类型的列被默认为某种非空值，可以指定一个默认值。默认值可以用DEFAULT关键词指定（这样一个默认值可以被附加到一个特定列的显式DEFAULT子句覆盖）。

- **element**

  可选参数。被创建的类型是一个数组，element指定了数组元素的类型。例如，要定义一个4字节整数的数组（int4）， 应指定ELEMENT = int4。

- **delimiter**

  可选参数。指定这种类型组成的数组中分隔值的定界符。

  可以把delimiter设置为一个特定字符，默认的定界符是逗号（,）。注意定界符是与数组元素类型相关的，而不是数组类型本身相关。

- **collatable**

  可选参数。如果这个类型的操作可以使用排序规则信息，则为TRUE。默认为FALSE。

  如果collatable为TRUE，这种类型的列定义和表达式可能通过使用COLLATE子句携带有排序规则信息。在该类型上操作的函数的实现负责真正利用这些信息，仅把类型标记为可排序的并不会让它们自动地去使用这类信息。

- **label**

  可选参数。与枚举类型的一个值相关的文本标签，其值为长度不超过63个字符的非空字符串。

说明

在创建用户定义类型的时候， GaussDB会自动创建一个与之关联的数组类型，其名称由该元素类型的名称前缀一个下划线组成。

示例

```
--创建一种复合类型，建表并插入数据以及查询。
gaussdb=# CREATE TYPE compfoo AS (f1 int, f2 text);
gaussdb=# CREATE TABLE t1_compfoo(a int, b compfoo);
gaussdb=# CREATE TABLE t2_compfoo(a int, b compfoo);
gaussdb=# INSERT INTO t1_compfoo values(1,(1,'demo'));
gaussdb=# INSERT INTO t2_compfoo SELECT * FROM t1_compfoo;
gaussdb=# SELECT (b).f1 FROM t1_compfoo;
gaussdb=# SELECT * FROM t1_compfoo t1 JOIN t2_compfoo t2 ON (t1.b).f1=(t1.b).f1;

--重命名数据类型。
gaussdb=# ALTER TYPE compfoo RENAME TO compfoo1;

--要改变一个用户定义类型compfoo1的所有者为usr1。
gaussdb=# CREATE USER usr1 PASSWORD '********';
gaussdb=# ALTER TYPE compfoo1 OWNER TO usr1;

--把用户定义类型compfoo1的模式改变为usr1。
gaussdb=# ALTER TYPE compfoo1 SET SCHEMA usr1;

--给一个数据类型增加一个新的属性。
gaussdb=# ALTER TYPE usr1.compfoo1 ADD ATTRIBUTE f3 int;

--删除compfoo1类型。
gaussdb=# DROP TYPE usr1.compfoo1 cascade;

--删除相关表和用户。
gaussdb=# DROP TABLE t1_compfoo;
gaussdb=# DROP TABLE t2_compfoo;
gaussdb=# DROP SCHEMA usr1;
gaussdb=# DROP USER usr1;

--创建一个枚举类型。
gaussdb=# CREATE TYPE bugstatus AS ENUM ('create', 'modify', 'closed');

--添加一个标签值。
gaussdb=# ALTER TYPE bugstatus ADD VALUE IF NOT EXISTS 'regress' BEFORE 'closed';

--重命名一个标签值。
gaussdb=# ALTER TYPE bugstatus RENAME VALUE 'create' TO 'new';

--创建一个集合类型
gaussdb=# CREATE TYPE compfoo_table AS TABLE OF compfoo;
```

相关链接

ALTER TYPE，**DROP TYPE**

7.14.91 CREATE USER

功能描述

创建一个用户。

注意事项

- 通过CREATE USER创建的用户，默认具有LOGIN权限。

- 通过CREATE USER创建用户的同时，系统会在执行该命令的数据库中，为该用户创建一个同名的SCHEMA。
- 系统管理员在普通用户同名SCHEMA下创建的对象，所有者为SCHEMA的同名用户（非系统管理员）。

语法格式

```
CREATE USER user_name [ [ WITH ] option [ ... ] ] [ ENCRYPTED | UNENCRYPTED ] { PASSWORD | IDENTIFIED BY } { 'password' [EXPIRED] | DISABLE };
```

其中option子句用于设置权限及属性等信息。

```
{SYSADMIN | NOSYSADMIN}
    | {MONADMIN | NOMONADMIN}
    | {OPRADMIN | NOOPRADMIN}
    | {POLADMIN | NOPOLADMIN}
    | {AUDITADMIN | NOAUDITADMIN}
    | {CREATEDB | NOCREATEDB}
    | {USEFT | NOUSEFT}
    | {CREATEROLE | NOCREATEROLE}
    | {INHERIT | NOINHERIT}
    | {LOGIN | NOLOGIN}
    | {REPLICATION | NOREPLICATION}
    | {VCADMIN | NOVCADMIN}
    | {PERSISTENCE | NOPERSISTENCE}
    | CONNECTION LIMIT connlimit
    | VALID BEGIN 'timestamp'
    | VALID UNTIL 'timestamp'
    | RESOURCE POOL 'respool'
    | USER GROUP 'groupuser'
    | PERM SPACE 'spacelimit'
    | TEMP SPACE 'tmpspacelimit'
    | SPILL SPACE 'spillspacelimit'
    | NODE GROUP logic_cluster_name
    | IN ROLE role_name [, ...]
    | IN GROUP role_name [, ...]
    | ROLE role_name [, ...]
    | ADMIN role_name [, ...]
    | USER role_name [, ...]
    | SYSID uid
    | DEFAULT TABLESPACE tablespace_name
    | PROFILE DEFAULT
    | PROFILE profile_name
    | PGUSER
```

参数说明

- **user_name**

  用户名称。

  取值范围：字符串，要符合**标识符命名规范**。且最大长度不超过63个字符。

- **password**

  登录密码。

  密码规则如下：

  - 密码默认不少于8个字符。
  - 不能与用户名及用户名倒序相同。
  - 至少包含大写字母（A~Z）、小写字母（a~z）、数字（0~9）和非字母数字字符（限定为~!@#$%^&*()-_=+\|[{}];:,<.>/?）四类字符中的三类字符。
  - 密码也可以是符合格式要求的密文字符串，这种情况主要用于用户数据导入场景，不推荐用户直接使用。如果直接使用密文密码，用户需要知道密文密

码对应的明文，并且保证明文密码复杂度，数据库不会校验密文密码复杂度，直接使用密文密码的安全性由用户保证。

- 创建用户时，应当使用单引号将用户密码括起来。

取值范围：字符串。

CREATE USER的其他参数值请参考**CREATE ROLE**。

示例

```
--创建用户jim，登录密码为********。
gaussdb=# CREATE USER jim PASSWORD '********';

--下面语句与上面的等价。
gaussdb=# CREATE USER kim IDENTIFIED BY '********';

--如果创建有“创建数据库”权限的用户，则需要加CREATEDB关键字。
gaussdb=# CREATE USER dim CREATEDB PASSWORD '********';

--修改用户jim的登录密码（使用ALTER USER ... IDENTIFIED BY ... REPLACE ...语法修改用户密码。其中，REPLACE前为新密码，REPLACE后为原密码）。
gaussdb=# ALTER USER jim IDENTIFIED BY '**********' REPLACE '********';

--为用户jim追加CREATEROLE权限。
gaussdb=# ALTER USER jim CREATEROLE;

--将enable_seqscan的值设置为on， 设置成功后，在下一会话中生效。
gaussdb=# ALTER USER jim SET enable_seqscan TO on;

--重置jim的enable_seqscan参数。
gaussdb=# ALTER USER jim RESET enable_seqscan;

--锁定jim账户。
gaussdb=# ALTER USER jim ACCOUNT LOCK;

--删除用户。
gaussdb=# DROP USER kim CASCADE;
gaussdb=# DROP USER jim CASCADE;
gaussdb=# DROP USER dim CASCADE;
```

相关链接

ALTER USER，**CREATE ROLE**，**DROP USER**

7.14.92 CREATE USER MAPPING

功能描述

定义一个用户到一个外部服务器的新映射。

注意事项

当在OPTIONS中出现password选项时，需要保证GaussDB每个节点的$GAUSSHOME/bin目录下存在usermapping.key.cipher和usermapping.key.rand文件，如果不存在这两个文件，请使用gs_guc工具生成并使用gs_ssh工具发布到GaussDB每个节点的$GAUSSHOME/bin目录下。

OPTIONS中的敏感字段（如password）在使用多层引号时，语义和不带引号的场景是不同的，因此不会被识别为敏感字段进行脱敏。

语法格式

```
CREATE USER MAPPING FOR { user_name | USER | CURRENT_USER | PUBLIC }
    SERVER server_name
    [ OPTIONS ( option 'value' [ , ... ] ) ]
```

参数说明

- **user_name**

  要映射到外部服务器的一个现有用户的名称。

  CURRENT_USER和USER匹配当前用户的名称。 当PUBLIC被指定时，一个公共映射会被创建，当没有特定用户的映射可用时将会使用它。

- **server_name**

  将为其创建用户映射的现有服务器的名称。

- **OPTIONS ({ option_name ' value ' } [, ...])**

  这个子句指定用户映射的选项。这些选项通常定义该映射实际的用户名和用户密码。选项名必须唯一。允许的选项名和值与该服务器的外部数据包装器有关。

  说明

  - 用户的密码会加密后保存到系统表PG_USER_MAPPING中，加密时需要使用usermapping.key.cipher和usermapping.key.rand作为加密密码文件和加密因子。首次使用前需要通过如下命令创建这两个文件，并将这两个文件放入各节点目录$GAUSSHOME/bin，且确保具有读权限。gs_ssh工具可以协助您快速将文件放入各节点对应目录下。
    ```
    gs_ssh -c "gs_guc generate -o usermapping -S default -D $GAUSSHOME/bin"
    ```
  - 其中-S参数指定default时会随机生成密码，用户也可为-S参数指定密码，此密码用于保证生成密码文件的安全性和唯一性，用户无需保存或记忆。其他参数详见工具参考中gs_guc工具说明。

示例

```
-- 创建角色。
gaussdb=# CREATE ROLE bob PASSWORD '********';

-- 创建外部服务器
gaussdb=# CREATE SERVER my_server FOREIGN DATA WRAPPER log_fdw;

-- 创建USER MAPPING。
gaussdb=# CREATE USER MAPPING FOR bob SERVER my_server OPTIONS (user 'bob', password '********');

-- 修改USER MAPPING。
gaussdb=# ALTER USER MAPPING FOR bob SERVER my_server OPTIONS (SET password '********');

-- 删除USER MAPPING。
gaussdb=# DROP USER MAPPING FOR bob SERVER my_server;

-- 删除外部服务器。
gaussdb=# DROP SERVER my_server;

-- 删除角色。
gaussdb=# DROP ROLE bob;
```

相关链接

ALTER USER MAPPING，DROP USER MAPPING

7.14.93 CREATE VIEW

功能描述

创建一个视图。视图与基本表不同，是一个虚拟的表。数据库中仅存放视图的定义，而不存放视图对应的数据，这些数据仍存放在原来的基本表中。若基本表中的数据发生变化，从视图中查询出的数据也随之改变。从这个意义上讲，视图就像一个窗口，透过它可以看到数据库中用户感兴趣的数据及变化。

注意事项

被授予CREATE ANY TABLE权限的用户，可以在public模式和用户模式下创建视图。

语法格式

```
CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] VIEW view_name [ ( column_name [, ...] ) ]
    [ WITH ( {view_option_name [= view_option_value]} [, ... ] ) ]
    AS query;
```

说明

创建视图时使用WITH(security_barrier)可以创建一个相对安全的视图，避免攻击者利用低成本函数的RAISE语句打印出隐藏的基表数据。

当视图创建后，不允许使用REPLACE修改本视图当中的列名，也不允许删除列。

参数说明

- **OR REPLACE**

  如果视图已存在，则重新定义。

- **TEMP | TEMPORARY**

  创建临时视图。

- **view_name**

  要创建的视图名称。可以用模式修饰。

  取值范围：字符串，符合**标识符命名规范**。

- **column_name**

  可选的名称列表，用作视图的字段名。如果没有给出，字段名取自查询中的字段名。

  取值范围：字符串，符合**标识符命名规范**。

- **view_option_name [= view_option_value]**

  该子句为视图指定一个可选的参数。

  目前view_option_name支持的参数仅有security_barrier，当VIEW试图提供行级安全时，应使用该参数。

  取值范围：Boolean类型，TRUE、FALSE

- **query**

  为视图提供行和列的SELECT或VALUES语句。

须知

若query包含指定分区表分区的子句，创建视图会将所指定分区的OID硬编码到系统表中。如果使用导致指定分区的OID发生变更的分区DDL语法，如DROP/SPLIT/MERGE该分区，则会导致视图不可用。需要重新创建视图。

示例

```
--创建字段spcname为pg_default组成的视图。
gaussdb=# CREATE VIEW myView AS
    SELECT * FROM pg_tablespace WHERE spcname = 'pg_default';

--查看视图。
gaussdb=# SELECT * FROM myView ;

--删除视图myView。
gaussdb=# DROP VIEW myView;
```

相关链接

ALTER VIEW，**DROP VIEW**

7.14.94 CREATE WEAK PASSWORD DICTIONARY

功能描述

向gs_global_config表中插入一个或者多个弱口令。

注意事项

- 只有初始用户、系统管理员和安全管理员拥有权限执行本语法。
- 弱口令字典中的口令存放在gs_global_config系统表中。
- 弱口令字典默认为空，用户通过本语法可以新增一条或多条弱口令。
- 当用户尝试通过本语法插入gs_global_config表中已存在的弱口令时，会只在表中保留一条该弱口令。

语法格式

```
CREATE WEAK PASSWORD DICTIONARY
    [WITH VALUES] ( {'weak_password'} [, ...] );
```

参数说明

weak_password

弱口令。

范围：字符串。

示例

```
--向gs_global_config系统表中插入单个弱口令。
gaussdb=# CREATE WEAK PASSWORD DICTIONARY WITH VALUES ('password1');

--向gs_global_config系统表中插入多个弱口令。
gaussdb=# CREATE WEAK PASSWORD DICTIONARY WITH VALUES ('password2'),('password3');
```



```
--清空gs_global_config系统表中所有弱口令。
gaussdb=# DROP WEAK PASSWORD DICTIONARY;

--查看现有弱口令。
gaussdb=# SELECT * FROM gs_global_config WHERE NAME LIKE 'weak_password';
```

相关链接

DROP WEAK PASSWORD DICTIONARY

7.14.95 CURSOR

功能描述

CURSOR命令用于创建一个游标，从一个查询里面检索出指定的几行数据。

为了处理SQL语句，存储过程进程分配一段内存区域来保存上下文联系。游标是指向上下文区域的句柄或指针。借助游标，存储过程可以控制上下文区域的变化。

注意事项

- 游标命令只能在事务块里使用。
- 通常游标和SELECT一样返回文本格式。因为数据在系统内部是用二进制格式存储的，系统必须对数据做一定转换以生成文本格式。一旦数据是以文本形式返回，客户端应用需要把它们转换成二进制进行操作。使用FETCH语句，游标可以返回文本或二进制格式。
- 应该小心使用二进制游标。文本格式一般都比对应的二进制格式占用的存储空间大。二进制游标返回内部二进制形态的数据，可能更易于操作。如果想以文本方式显示数据，则以文本方式检索会为用户节约很多客户端的工作。比如，如果查询从某个整数列返回1，在缺省的游标里将获得一个字符串1，但在二进制游标里将得到一个4字节的包含该数值内部形式的数值（大端顺序）。

语法格式

```
CURSOR cursor_name
    [ BINARY ]  [ NO SCROLL ]  [ { WITH | WITHOUT } HOLD ]
    FOR query;
```

参数说明

- **cursor_name**

  将要创建的游标名。

  取值范围：遵循数据库对象命名规范。

- **BINARY**

  指明游标以二进制而不是文本格式返回数据。

- **NO SCROLL**

  声明游标检索数据行的方式。

  - NO SCROLL：声明该游标不能用于以倒序的方式检索数据行。
  - 未声明：根据执行计划的不同，自动判断该游标是否可以用于以倒序的方式检索数据行。

- **WITH HOLD | WITHOUT HOLD**

声明当创建游标的事务结束后，游标是否能继续使用。

- WITH HOLD：声明该游标在创建它的事务结束后仍可继续使用。
- WITHOUT HOLD：声明该游标在创建它的事务之外不能再继续使用，此游标将在事务结束时被自动关闭。
- 如果不指定WITH HOLD或WITHOUT HOLD，默认行为是WITHOUT HOLD。
- 跨节点事务不支持WITH HOLD（例如在多Coordinator部署集群中所创建的含有DDL的事务属于跨节点事务）。

- **query**

  使用SELECT或VALUES子句指定游标返回的行。

  取值范围：SELECT或VALUES子句。

示例

请参考FETCH的示例。

相关链接

FETCH

7.14.96 DEALLOCATE

功能描述

DEALLOCATE用于删除预备语句。

注意事项

- 如果用户没有明确删除一个预备语句，那么它将在会话结束的时候被删除。
- PREPARE关键字总被忽略。

语法格式

```
DEALLOCATE [ PREPARE ] { name | ALL };
```

参数说明

- **name**

  将要删除的预备语句。

- **ALL**

  删除所有预备语句。

示例

无。

7.14.97 DECLARE

功能描述

DECLARE命令既可以定义一个游标，用于在一个大的查询里面检索少数几行数据，也可以作为一个匿名块的开始。

本节主要描述定义为游标的用法，开启匿名块的用法见**BEGIN**。

为了处理SQL语句，存储过程进程分配一段内存区域来保存上下文联系。游标是指向上下文区域的句柄或指针。借助游标，存储过程可以控制上下文区域的变化。

通常游标和SELECT一样返回文本格式。因为数据在系统内部是用二进制格式存储的，系统必须对数据做一定转换以生成文本格式。一旦数据是以文本形式返回，客户端应用需要把它们转换成二进制进行操作。使用FETCH语句，游标可以返回文本或二进制格式。

注意事项

- 游标命令只能在事务块里使用。
- 应该小心使用二进制游标。文本格式一般都比对应的二进制格式占用的存储空间大。二进制游标返回内部二进制形态的数据，可能更易于操作。如果想以文本方式显示数据，则以文本方式检索会为用户节约很多客户端的工作。比如，如果查询从某个整数列返回1，在缺省的游标里将获得一个字符串1，但在二进制游标里将得到一个4字节的包含该数值内部形式的数值（大端顺序）。

语法格式

- 定义游标

```
DECLARE cursor_name [ BINARY ] [ NO SCROLL ]
    CURSOR [ { WITH | WITHOUT } HOLD ] FOR query ;
```

- 开启匿名块

```
[DECLARE [declare_statements]]
BEGIN
execution_statements
END;
/
```

参数说明

- **cursor_name**

  将要创建的游标名。

  取值范围：遵循数据库对象命名规范。

- **BINARY**

  指明游标以二进制而不是文本格式返回数据。

- **NO SCROLL**

  声明游标检索数据行的方式。

  - NO SCROLL：声明该游标不能用于以倒序的方式检索数据行。
  - 未声明：根据执行计划的不同，自动判断该游标是否可以用于以倒序的方式检索数据行。

- **WITH HOLD**

  WITHOUT HOLD

声明当创建游标的事务结束后，游标是否能继续使用。

- WITH HOLD：声明该游标在创建它的事务结束后仍可继续使用。
- WITHOUT HOLD：声明该游标在创建它的事务之外不能再继续使用，此游标将在事务结束时被自动关闭。
- 如果不指定WITH HOLD或WITHOUT HOLD，默认行为是WITHOUT HOLD。

须知

- 声明为WITH HOLD的游标，在事务结束时，会缓存游标所有数据，若游标数据量较大，此过程耗时可能较长。

- **query**

  使用SELECT或VALUES子句指定游标返回的行。

  取值范围：SELECT或VALUES子句。

- **declare_statements**

  声明变量，包括变量名和变量类型，如“sales_cnt int”。

- **execution_statements**

  匿名块中要执行的语句。

  取值范围：已存在的函数名称。

示例

定义游标示例请参考FETCH的**示例**。

相关链接

BEGIN，**FETCH**

7.14.98 DELETE

功能描述

DELETE从指定的表里删除满足WHERE子句的行。如果WHERE子句不存在，将删除表中所有行，结果只保留表结构。

注意事项

- 表的所有者、被授予表DELETE权限的用户或被授予DELETE ANY TABLE权限的用户有权删除表中数据，当三权分立开关关闭时，系统管理员默认拥有此权限。同时也必须有USING子句引用的表以及condition上读取表的SELECT权限。
- 对于多表删除语法，暂时不支持对视图和含有RULE的表进行多表删除。
- 对于子查询是STREAM计划的DELETE语句，不支持删除的行数据同时进行UPDATE更新操作。

语法格式

单表删除：


```
[ WITH [ RECURSIVE ] with_query [, ...] ]
DELETE [/*+ plan_hint */] [FROM] [ ONLY ] table_name [ * ] [ [ [partition_clause]  [ [ AS ] alias ] ] |
[ [ [ AS ] alias ] [partitions_clause] ] ]
    [ USING using_list ]
    [ WHERE condition | WHERE CURRENT OF cursor_name ]
    [ ORDER BY {expression [ ASC | DESC | USING operator ] }]
    [ LIMIT { count } ]
    [ RETURNING { * | { output_expr [ [ AS ] output_name ] } [, ...] } ];
```

多表删除：

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
DELETE [/*+ plan_hint */] [FROM]
    {[ ONLY ] table_name [ * ] [ [ [partition_clause]  [ [ AS ] alias ] ] | [ [ [ AS ] alias ]
[partitions_clause] ] ]} [, ...]
    [ USING using_list ]
    [ WHERE condition ];
```

或：

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
DELETE [/*+ plan_hint */]
    {[ ONLY ] table_name [ * ] [ [ [partition_clause]  [ [ AS ] alias ] ] | [ [ [ AS ] alias ]
[partitions_clause] ] ]} [, ...]
    [ FROM using_list ]
    [ WHERE condition ];
```

其中with_query的详细格式为：

```
with_query_name [ ( column_name [, ...] ) ] AS [ [ NOT ] MATERIALIZED ]
( {select | values | insert | update | delete} )
```

参数说明

- **WITH [RECURSIVE] with_query [, ...]**

  用于声明一个或多个可以在主查询中通过名称引用的子查询，相当于临时表。

  如果声明了RECURSIVE，那么允许SELECT子查询通过名称引用它自己。

  - with_query_name指定子查询生成的结果集名称，在查询中可使用该名称访问子查询的结果集。
  - column_name指定子查询结果集中显示的列名。
  - 每个子查询可以是SELECT，VALUES，INSERT，UPDATE或DELETE语句。
  - 用户可以使用MATERIALIZED / NOT MATERIALIZED对CTE进行修饰。
    - 如果声明为MATERIALIZED，WITH查询将被物化，生成一个子查询结果集的拷贝，在引用处直接查询该拷贝，因此WITH子查询无法和主干SELECT语句进行联合优化（如谓词下推、等价类传递等），对于此类场景可以使用NOT MATERIALIZED进行修饰，如果WITH查询语义上可以作为子查询内联执行，则可以进行上述优化。
    - 如果用户没有显示声明物化属性则遵守以下规则：如果CTE只在所属主干语句中被引用一次，且语义上支持内联执行，则会被改写为子查询内联执行，否则以CTE Scan的方式物化执行。

- **plan_hint子句**

  以/*+ */的形式在DELETE关键字后，用于对DELETE对应的语句块生成的计划进行hint调优，详细用法请参见章节**使用Plan Hint进行调优**。每条语句中只有第一个/*+ plan_hint */注释块会作为hint生效，里面可以写多条hint。

- **ONLY**

  如果指定ONLY则只有该表被删除；如果没有声明，则该表和它的所有子表将都被删除。

- **table_name**
  目标表的名称（可以有模式修饰）。
  取值范围：已存在的表名。

  说明

  支持使用DATABASE LINK方式对远端表进行操作，使用方式详情请见**DATABASE LINK**。

- **partition_clause**
  指定分区删除操作。
  PARTITION { (partition_name) | FOR (partition_value [, ...]) } |
  SUBPARTITION { (subpartition_name) | FOR (subpartition_value [, ...]) }
  关键字详见**SELECT**章节介绍。
  示例详见**CREATE TABLE SUBPARTITION**。
- **partitions_clause**
  指定多个分区删除操作。
  PARTITION { ({ partition_name | subpartition_name } [, ...]) }
  此语法仅在参数sql_compatibility='B'时生效。
  关键字详见**SELECT**章节介绍。
  示例详见**CREATE TABLE SUBPARTITION**。
- **alias**
  目标表的别名。
  取值范围：字符串，符合**标识符命名规范**。
- **using_list**
  using子句。

  须知

  当参数sql_compatibility='B'或删除多张目标表时，using_list指定关联表的集合时可以同时出现目标表，并且可以定义表的别名并在目标表中使用。其他情况下则目标表不可重复出现在using_list中。

- **condition**
  一个返回Boolean值的表达式，用于判断哪些行需要被删除。不建议使用int等数值类型作为condition，因为int等数值类型可以隐式转换为bool值（非0值隐式转换为true，0转换为false），可能导致非预期的结果。
- **WHERE CURRENT OF cursor_name**
  当cursor指向表的某一行时，可以使用此语法删除cursor当前指向的行。使用限制及约束请参考**UPDATE**章节对此语法介绍。
- **ORDER BY子句**
  关键字详见**SELECT**章节介绍。
- **LIMIT子句**
  关键字详见**SELECT**章节介绍。
- **output_expr**

DELETE命令删除行之后计算输出结果的表达式。该表达式可以使用表的任意字段。可以使用*返回被删除行的所有字段。

- **output_name**

  一个字段的输出名称。

  取值范围：字符串，符合**标识符命名规范**。

示例

```
--创建一个SCHEMA。
gaussdb=# CREATE SCHEMA tpcds;

--创建表tpcds.customer_address。
gaussdb=# CREATE TABLE tpcds.customer_address
(
ca_address_sk         INTEGER           NOT NULL,
ca_address_id         CHARACTER(16)     NOT NULL,
ca_street_number      INTEGER                   ,
ca_street_name        CHARACTER (20)
);

--向表中插入多条记录。
gaussdb=# INSERT INTO tpcds.customer_address VALUES (1, 'AAAAAAAABAAAAAAA', '18', 'Jackson'),
(10000, 'AAAAAAAACAAAAAAA', '362', 'Washington 6th'),(15000, 'AAAAAAAADAAAAAAA', '585', 'Dogwood
Washington');

--创建表tpcds.customer_address_bak。
gaussdb=# CREATE TABLE tpcds.customer_address_bak AS TABLE tpcds.customer_address;

--删除tpcds.customer_address_bak中ca_address_sk大于14888的职员。
gaussdb=# DELETE FROM tpcds.customer_address_bak WHERE ca_address_sk > 14888;

--同时删除tpcds.customer_address和tpcds.customer_address_bak中ca_address_sk小于50的职员。
gaussdb=# DELETE FROM  tpcds.customer_address a,tpcds.customer_address_bak b where a.ca_address_sk
= b.ca_address_sk and a.ca_address_sk < 50;

--删除tpcds.customer_address_bak中所有数据。
gaussdb=# DELETE FROM tpcds.customer_address_bak;

--删除tpcds.customer_address_bak表。
gaussdb=# DROP TABLE tpcds.customer_address_bak;

--删除tpcds.customer_address表。
gaussdb=# DROP TABLE tpcds.customer_address;

--删除SCHEMA。
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

优化建议

- delete

  如果要删除表中的所有记录，建议使用truncate语法。

7.14.99 DO

功能描述

执行匿名代码块。

代码块被看作是没有参数的一段函数体，返回值类型是void。它的解析和执行是同一时刻发生的。

注意事项

- 程序语言在使用之前，必须通过命令CREATE LANGUAGE安装到当前的数据库中。 plpgsql是默认的安装语言，其它语言安装时必须指定。
- 如果语言是不受信任的，用户必须有使用程序语言的USAGE权限，或者是系统管理员权限。

语法格式

```
DO [ LANGUAGE lang_name ] code;
```

参数说明

- **lang_name**

  用来解析代码的程序语言的名称，如果缺省，默认的语言是plpgsql。
- **code**

  可以被执行的程序语言代码，必须指定为字符串。

示例

```
--创建用户webuser。
gaussdb=# CREATE USER webuser PASSWORD '********';

--授予用户webuser对模式tpcds下视图的所有操作权限。
gaussdb=# DO $$DECLARE r record;
BEGIN
    FOR r IN SELECT c.relname table_name,n.nspname table_schema FROM pg_class c,pg_namespace n
            WHERE c.relnamespace = n.oid AND n.nspname = 'tpcds' AND relkind IN ('r','v')
    LOOP
        EXECUTE 'GRANT ALL ON ' || quote_ident(r.table_schema) || '.' || quote_ident(r.table_name) || ' TO
webuser';
    END LOOP;
END$$;


--删除用户webuser。
gaussdb=# DROP USER webuser CASCADE;
```

7.14.100 DROP AGGREGATE

功能描述

删除一个聚合函数。

注意事项

DROP AGGREGATE删除一个现存的聚合函数，执行这条命令的用户必须是该聚合函数的所有者。

语法格式

```
DROP AGGREGATE [ IF EXISTS ] name ( argtype [ , ... ] ) [ CASCADE | RESTRICT ]
```

参数说明

- **IF EXISTS**

  如果指定的聚合不存在，那么发出一个 notice 而不是抛出一个错误。

- **name**
  现存的聚合函数名(可以有模式修饰)
- **argtype**
  聚合函数操作的输入数据类型，要引用一个零参数聚合函数，请用*代替输入数据类型列表。
- **CASCADE**
  级联删除依赖于这个聚合函数的对象。
- **RESTRICT**
  如果有任何依赖对象，则拒绝删除这个聚合函数。这是缺省处理。

示例

将integer类型的聚合函数myavg删除：

```
DROP AGGREGATE myavg(integer);
```

兼容性

SQL 标准里没有DROP AGGREGATE语句。

7.14.101 DROP AUDIT POLICY

功能描述

删除一个审计策略。

注意事项

只有poladmin，sysadmin或初始用户才能进行此操作。

语法格式

```
DROP AUDIT POLICY [IF EXISTS] policy_name;
```

参数说明

- policy_name
  审计策略名称，需要唯一，不可重复。
  取值范围：字符串，要符合**标识符命名规范**。

示例

请参考CREATE AUDIT POLICY的**示例**。

相关链接

ALTER AUDIT POLICY，**CREATE AUDIT POLICY**。

7.14.102 DROP CAST

功能描述

DROP CAST用于删除类型转换。

注意事项

要能删除一个类型转换，必须拥有源或者目的数据类型的权限。这是和创建一个类型转换相同的权限。

语法格式

```
DROP CAST [ IF EXISTS ] (source_type AS target_type) [ CASCADE | RESTRICT ]
```

参数说明

- **IF EXISTS**
  如果指定的转换不存在，那么发出一个 notice 而不是抛出一个错误。
- **source_type**
  类型转换里的源数据类型。
- **target_type**
  类型转换里的目标数据类型。
- **CASCADE | RESTRICT**
  这些键字没有任何效果，因为在类型转换上没有依赖关系。

示例

删除从text到int的转换：

```
DROP CAST (text AS int);
```

兼容性

DROP CAST遵循 SQL 标准。

7.14.103 DROP CLIENT MASTER KEY

功能描述

删除客户端加密主密钥(CMK)。

注意事项

- 只有客户端加密主密钥所有者或者被授予了DROP权限的用户有权限执行命令，系统管理员默认拥有此权限。
- 该命令只能删除数据库中系统表记录的密钥对象元数据信息，无法删除由客户端密钥工具或在线密钥服务中管理的密钥实体。

语法格式

```
DROP CLIENT MASTER KEY [ IF EXISTS ] client_master_key_name [, ...] [ CASCADE | RESTRICT ];
```


参数说明

- **IF EXISTS**

  如果指定的客户端加密主密钥不存在，则发出一个notice而不是抛出一个错误。

- **client_master_key_name**

  要删除的客户端加密主密钥名称。

  取值范围：字符串，已存在的客户端加密主密钥对象的名称。

- **CASCADE | RESTRICT**

  - 表示允许/不允许级联删除依赖于客户端加密主密钥的对象。

示例

```
--删除客户端加密主密钥对象。
gaussdb=> DROP CLIENT MASTER KEY ImgCMK CASCADE;
NOTICE:  drop cascades to column setting: imgcek
DROP CLIENT MASTER KEY
```

7.14.104 DROP COLUMN ENCRYPTION KEY

功能描述

删除一个列加密密钥(CEK)。

注意事项

只有列加密密钥所有者或者被授予了DROP权限的用户有权限执行该命令，系统管理员默认拥有此权限。

语法格式

```
DROP COLUMN ENCRYPTION KEY [ IF EXISTS ] client_column_key_name [, ...] [ CASCADE | RESTRICT ];
```

参数说明

- **IF EXISTS**

  如果指定的列加密密钥不存在，则发出一个notice而不是抛出一个错误。

- **client_column_key_name**

  要删除的列加密密钥名称。

  取值范围：字符串，已存在的列加密密钥名称。

- **CASCADE | RESTRICT**

  对于密态特性来说，级联删除加密列属于危险操作，实际上都无法删除依赖于列加密密钥的加密列。

示例

```
--删除列加密密钥。
gaussdb=# DROP COLUMN ENCRYPTION KEY ImgCEK CASCADE;
ERROR:  cannot drop column setting: imgcek cascadely because encrypted column depend on it.
HINT:  we have to drop encrypted column: name, ... before drop column setting: imgcek cascadely.
```


7.14.105 DROP DATABASE

功能描述

删除一个数据库。

注意事项

- 只有数据库所有者或者被授予了数据库DROP权限的用户有权限执行DROP DATABASE命令，系统管理员默认拥有此权限。
- 不能对系统默认安装的三个数据库（POSTGRES、TEMPLATE0和TEMPLATE1）执行删除操作，系统做了保护。如果想查看当前服务中有哪几个数据库，可以用gsql的\l命令查看。
- 如果有用户正在与要删除的数据库连接，则删除操作失败。
- 不能在事务块中执行DROP DATABASE命令。
- 如果执行DROP DATABASE失败，事务回滚，需要再次执行一次DROP DATABASE IF EXISTS。

须知

DROP DATABASE一旦执行将无法撤销，请谨慎使用。

语法格式

```
DROP DATABASE [ IF EXISTS ] database_name ;
```

参数说明

- **IF EXISTS**
  如果指定的数据库不存在，则发出一个notice而不是抛出一个错误。
- **database_name**
  要删除的数据库名称。
  取值范围：字符串，已存在的数据库名称。

示例

请参见CREATE DATABASE的**示例**。

相关链接

CREATE DATABASE

优化建议

- DROP DATABASE
  不支持在事务中删除DATABASE。

7.14.106 DROP DATABASE LINK

功能描述

删除DATABASE LINK对象。

语法格式

```
DROP [ PUBLIC ] DATABASE LINK dblink ;
```

参数说明

- dblink
  连接名称。
- PUBLIC
  连接类型，不加public默认private。

7.14.107 DROP DIRECTORY

功能描述

删除指定的DIRECTORY对象。

注意事项

- 当enable_access_server_directory=off时，只允许初始用户删除directory对象。
- 当enable_access_server_directory=on时，具有SYSADMIN权限的用户、directory对象的属主、被授予了该directory的DROP权限的用户或者继承了内置角色gs_role_directory_drop权限的用户可以删除directory对象。

语法格式

```
DROP DIRECTORY [ IF EXISTS ] directory_name;
```

参数说明

- **directory_name**
  目录名称。
  取值范围：已经存在的目录名。

示例

```
--创建目录。
gaussdb=# CREATE OR REPLACE DIRECTORY  dir  as '/tmp/';

--删除目录。
gaussdb=# DROP DIRECTORY dir;
```

相关链接

CREATE DIRECTORY，**ALTER DIRECTORY**

7.14.108 DROP EVENT

功能描述

删除一个定时任务。

注意事项

定时任务相关操作只有sql_compatibility = 'B'时支持。

语法格式

```
DROP EVENT [IF EXISTS] event_name
```

参数说明

- IF EXISTS
  如果定时任务不存在，会输出一个NOTICE。

示例

```
gaussdb=# DROP EVENT event_e1;
```

7.14.109 DROP FOREIGN DATA WRAPPER

功能描述

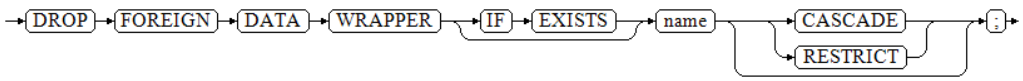
删除指定的外部数据封装。

注意事项

只有在support_extended_features=on时才能够成功执行drop语句。

语法格式

```
DROP FOREIGN DATA WRAPPER [ IF EXISTS ] name [ CASCADE | RESTRICT ];
```



参数说明

- **IF EXISTS**
  当使用IF EXISTS,如果外部数据封装器不存在时，不会抛出错误，而是产生一个通知。
- **name**
  已存在的外部数据封装器的名称。
- **CASCADE**
  自动删除依赖外部数据封装器的对象（如服务器）。
- **RESTRICT**

如果有依赖于外部数据封装器的对象，则不允许删除外部数据封装器。这是缺省行为。

示例

```
--删除外部数据封装器dbi
gaussdb=# DROP FOREIGN DATA WRAPPER dbi;
```

相关链接

ALTER FOREIGN DATA WRAPPER，**CREATE FOREIGN DATA WRAPPER**

7.14.110 DROP FUNCTION

功能描述

删除一个已存在的函数。

注意事项

- 如果函数中涉及对临时表相关操作，则无法使用DROP FUNCTION删除函数。
- 只有函数的所有者或者被授予了函数DROP权限的用户才能执行DROP FUNCTION命令，系统管理员默认拥有该权限。

语法格式

```
DROP FUNCTION [ IF EXISTS ] function_name
[ ( [ {[ argname ] [ argmode ] argtype} [, ...] ] ) [ CASCADE | RESTRICT ] ];
```

参数说明

- **IF EXISTS**

  IF EXISTS表示，如果函数存在则执行删除操作，函数不存在也不会报错，只是发出一个notice。
- **function_name**

  要删除的函数名称。

  取值范围：已存在的函数名。
- **argmode**

  函数参数的模式。
- **argname**

  函数参数的名称。
- **argtype**

  函数参数的类型
- **CASCADE | RESTRICT**
  - CASCADE：级联删除依赖于函数的对象 。
  - RESTRICT：如果有任何依赖对象存在，则拒绝删除该函数（缺省行为）。

示例

请参见的**示例**。

相关链接

ALTER FUNCTION，CREATE FUNCTION

7.14.111 DROP GLOBAL CONFIGURATION

功能描述

删除系统表gs_global_config中的参数值。

注意事项

- 仅支持数据库初始用户运行此命令。
- 参数名称不能为weak_password、undostoragetype。

语法格式

```
DROP GLOBAL CONFIGURATION name [, ...];
```

参数说明

name

参数名称必须是gs_global_config中已经存在的，删除不存在的参数将报错。

相关链接

ALTER GLOBAL CONFIGURATION

7.14.112 DROP GROUP

功能描述

删除用户组。DROP GROUP是DROP ROLE的别名。

注意事项

DROP GROUP是GaussDB管理工具封装的接口，用来实现GaussDB管理。该接口不建议用户直接使用，以免对GaussDB状态造成影响。

语法格式

```
DROP GROUP [ IF EXISTS ] group_name [, ...];
```

参数说明

- **IF EXISTS**
  如果指定的角色不存在，则发出一个notice而不是抛出一个错误。
- **group_name**
  要删除的角色名称。
  取值范围：已存在的角色。

相关链接

CREATE GROUP，ALTER GROUP，DROP ROLE

7.14.113 DROP INDEX

功能描述

删除索引。

注意事项

索引的所有者、索引所在模式的所有者、拥有索引所在表的INDEX权限的用户或者被授予了DROP ANY INDEX权限的用户有权限执行DROP INDEX命令，系统管理员默认拥有此权限。

对于全局临时表，当某个会话已经初始化了全局临时表对象（包括创建全局临时表和第一次向全局临时表内插入数据）时，其他会话不能够执行该表上索引的删除操作。

语法格式

```
DROP INDEX [ CONCURRENTLY ] [ IF EXISTS ]
    index_name [, ...] [ CASCADE | RESTRICT ];
```

参数说明

- **CONCURRENTLY**

  以不加锁的方式删除索引。删除索引时，一般会阻塞其他语句对该索引所依赖表的访问。加此关键字，可在删除过程中避免阻塞。

  此选项只能指定一个索引的名称， 并且不支持CASCADE选项。

  普通DROP INDEX命令可以在事务内执行，但是DROP INDEX CONCURRENTLY不可以在事务内执行。

- **IF EXISTS**

  如果指定的索引不存在，则发出一个notice而不是抛出一个错误。

- **index_name**

  要删除的索引名。

  取值范围：已存在的索引。

- **CASCADE | RESTRICT**
  - CASCADE：表示允许级联删除依赖于该索引的对象。
  - RESTRICT：表示有依赖于此索引的对象存在时，该索引无法被删除。此选项为缺省值。

示例

请参见CREATE INDEX的示例。

相关链接

ALTER INDEX，CREATE INDEX

7.14.114 DROP LANGUAGE

本版本暂不支持使用该语法。

7.14.115 DROP MASKING POLICY

功能描述

删除脱敏策略。

注意事项

只有POLADMIN，SYSADMIN或初始用户才能执行此操作。

语法格式

```
DROP MASKING POLICY [ IF EXISTS ] policy_name;
```

参数说明

- **policy_name**

  审计策略名称，不可重复。

  取值范围：字符串，要符合**标识符命名规范**。已存在的策略名称。

示例

```
--删除一个脱敏策略。
gaussdb=# DROP MASKING POLICY IF EXISTS maskpol1;

--删除一组脱敏策略。
gaussdb=# DROP MASKING POLICY IF EXISTS maskpol1, maskpol2, maskpol3;
```

相关链接

ALTER MASKING POLICY，**CREATE MASKING POLICY**。

7.14.116 DROP MATERIALIZED VIEW

功能描述

删除数据库中已有的物化视图。

注意事项

物化视图的所有者、物化视图所在模式的所有者、被授予了物化视图DROP权限的用户或拥有DROP ANY TABLE权限的用户才有权限执行DROP MATERIALIZED VIEW命令，系统管理员默认拥有此权限。

语法格式

```
DROP MATERIALIZED VIEW [ IF EXISTS ] mv_name [, ...] [ CASCADE | RESTRICT ];
```


参数说明

- **IF EXISTS**

  如果指定的物化视图不存在，则发出一个notice而不是抛出一个错误。

- **mv_name**

  要删除的物化视图名称。

- **CASCADE | RESTRICT**
  - CASCADE：级联删除依赖此物化视图的对象。
  - RESTRICT：如果有依赖对象存在，则拒绝删除此物化视图。此选项为缺省值。

示例

```
--创建表。
gaussdb=# CREATE TABLE my_table (c1 int, c2 int)
WITH(STORAGE_TYPE=ASTORE);

--创建名为my_mv的物化视图。
gaussdb=# CREATE MATERIALIZED VIEW my_mv AS SELECT * FROM my_table;

--删除名为my_mv的物化视图。
gaussdb=# DROP MATERIALIZED VIEW my_mv;

--删除表。
gaussdb=# DROP TABLE my_table;
```

相关链接

ALTER MATERIALIZED VIEW，**CREATE INCREMENTAL MATERIALIZED VIEW**，**CREATE MATERIALIZED VIEW**，**CREATE TABLE**，**REFRESH INCREMENTAL MATERIALIZED VIEW**，**REFRESH MATERIALIZED VIEW**

7.14.117 DROP MODEL

功能描述

删除一个已训练完成保存的模型对象。

注意事项

所删除模型可在系统表gs_model_warehouse中查看到。

语法格式

```
DROP MODEL model_name;
```

参数说明

model_name

模型名称

取值范围：字符串，需要符合**标识符命名规范**。

相关链接

CREATE MODEL，PREDICT BY

7.14.118 DROP OPERATOR

删除一个操作符的定义。

语法格式

```
DROP OPERATOR [ IF EXISTS ]  name ( { left_type | NONE } , { right_type | NONE } );
```

参数说明

- **name**
  一个现有操作符的名字。
- **left_type**
  操作符的左操作数的数据类型；如果没有左操作数，那么写NONE。
- **right_type**
  操作符的右操作数的数据类型；如果没有右操作数，那么写NONE。

示例

删除一个用于text的用户定义操作符a @@ b：

```
DROP OPERATOR @@ (text, text);
```

7.14.119 DROP OWNED

功能描述

删除一个数据库角色所拥有的数据库对象的权限。

注意事项

- 所有该角色在当前数据库里和共享对象（数据库，表空间）上的所有对象上的权限都将被撤销。
- DROP OWNED常常被用来为移除一个或者多个角色做准备。因为DROP OWNED只影响当前数据库中的对象，通常需要在包含将被移除角色所拥有的对象的每一个数据库中都执行这个命令。
- 使用CASCADE选项可能导致这个命令递归去删除由其他用户所拥有的对象。
- 角色所拥有的数据库、表空间将不会被移除。
- 角色所拥有的私有DATABASE LINK连接需要添加CASCADE才可删除。

语法格式

```
DROP OWNED BY name [, ...] [ CASCADE | RESTRICT ];
```

参数说明

- **name**
  角色名。

- **CASCADE | RESTRICT**
  - CASCADE：级联删除所有依赖于被删除对象的对象。
  - RESTRICT（缺省值）：拒绝删除那些有任何依赖对象存在的对象。

相关链接

REASSIGN OWNED , **DROP ROLE**

7.14.120 DROP PACKAGE

功能描述

删除已存在的PACKAGE或者PACKAGE BODY。

注意事项

删除PACKAGE BODY后，PACKAGE内的存储过程及函数会同时失效。

语法格式

```
DROP PACKAGE [ IF EXISTS ] package_name;
DROP PACKAGE BODY [ IF EXISTS ] package_name;
```

参数说明

- **IF EXISTS**
  如果指定的存储过程不存在，会提示一个notice而不是产生一个错误。
- **package_name**
  要删除的package名称。
  取值范围：已存在的package名。

示例

```
--创建PACKAGE。
gaussdb=# CREATE OR REPLACE PACKAGE PCK1
IS
a int;
END pck1;
/
CREATE PACKAGE

--删除PACKAGE。
gaussdb=# DROP PACKAGE PCK1;
DROP PACKAGE
```

相关链接

ALTER PACKAGE，**CREATE PACKAGE**

7.14.121 DROP PROCEDURE

功能描述

删除已存在的存储过程。

语法格式

```
DROP PROCEDURE [ IF EXISTS ] procedure_name ;
```

参数说明

- **IF EXISTS**
  如果指定的存储过程不存在，发出一个notice而不是抛出一个错误。
- **procedure_name**
  要删除的存储过程名称。
  取值范围：已存在的存储过程名。

相关链接

CREATE PROCEDURE

7.14.122 DROP RESOURCE LABEL

功能描述

删除资源标签。

注意事项

只有POLADMIN，SYSADMIN或初始用户才能执行此操作。

语法格式

```
DROP RESOURCE LABEL [IF EXISTS] label_name[, ...];
```

参数说明

- **label_name**
  资源标签名称。
  取值范围：字符串，要符合**标识符命名规范**。

示例

```
--删除一个资源标签。
gaussdb=# DROP RESOURCE LABEL IF EXISTS res_label1;

--删除一组资源标签。
gaussdb=# DROP RESOURCE LABEL IF EXISTS res_label1, res_label2, res_label3;
```

相关链接

ALTER RESOURCE LABEL，**CREATE RESOURCE LABEL**。

7.14.123 DROP RESOURCE POOL

功能描述

删除一个资源池。

说明

如果某个角色已关联到该资源池，无法删除。

注意事项

只要用户对当前数据库有DROP权限，就可以删除资源池。

语法格式

```
DROP RESOURCE POOL [ IF EXISTS ] pool_name;
```

参数说明

- **IF EXISTS**
  如果指定的资源池不存在，发出一个notice而不是抛出一个错误。
- **pool_name**
  已创建过的资源池名称。
  取值范围：字符串，要符合**标识符命名规范**。

示例

请参见CREATE RESOURCE POOL的**示例**。

相关链接

ALTER RESOURCE POOL，**CREATE RESOURCE POOL**

7.14.124 DROP ROLE

功能描述

删除指定的角色。

语法格式

```
DROP ROLE [ IF EXISTS ] role_name [, ...];
```

参数说明

- **IF EXISTS**
  如果指定的角色不存在，则发出一个notice而不是抛出一个错误。
- **role_name**
  要删除的角色名称。
  取值范围：已存在的角色。

示例

请参见CREATE ROLE的**示例**。

相关链接

CREATE ROLE，ALTER ROLE，SET ROLE

7.14.125 DROP ROW LEVEL SECURITY POLICY

功能描述

删除表上某个行访问控制策略。

注意事项

仅表的所有者或者管理员用户才能删除表的行访问控制策略。

语法格式

```
DROP [ ROW LEVEL SECURITY ] POLICY [ IF EXISTS ] policy_name ON table_name [ CASCADE | RESTRICT ]
```

参数说明

- **IF EXISTS**
  如果指定的行访问控制策略不存在，发出一个notice而不是抛出一个错误。
- **policy_name**
  要删除的行访问控制策略的名称。
- **table_name**
  行访问控制策略所在的数据表名。
- **CASCADE | RESTRICT**
  仅适配此语法，无对象依赖于该行访问控制策略，CASCADE和RESTRICT效果相同。

示例

```
--创建数据表all_data。
gaussdb=# CREATE TABLE all_data(id int, role varchar(100), data varchar(100));

--创建行访问控制策略。
gaussdb=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role = CURRENT_USER);

--删除行访问控制策略。
gaussdb=# DROP ROW LEVEL SECURITY POLICY all_data_rls ON all_data;

--删除数据表all_data。
gaussdb=# DROP TABLE all_data;
```

相关链接

ALTER ROW LEVEL SECURITY POLICY，CREATE ROW LEVEL SECURITY POLICY

7.14.126 DROP RULE

功能描述

删除一个重写规则。

语法格式

```
DROP RULE [ IF EXISTS ] name ON table_name [ CASCADE | RESTRICT ]
```

参数说明

- **IF EXISTS**

  如果该规则不存在，会抛出一个NOTICE。

- **name**

  要删除的现存规则名称。

- **table_name**

  该规则应用的表名。

- **CASCADE**

  自动级联删除依赖于此规则的对象。

- **RESTRICT**

  如果有任何依赖对象，则拒绝删除此规则。该值为缺省值。

示例

```
--创建表def_test、视图def_view_test用于创建RULE
gaussdb=# CREATE TABLE def_test (
c1 int4 DEFAULT 5,
c2 text DEFAULT 'initial_default'
);
gaussdb=# CREATE VIEW def_view_test AS SELECT * FROM def_test;
--创建RULE def_view_test_ins
gaussdb=# CREATE RULE def_view_test_ins AS
gaussdb=# ON INSERT TO def_view_test
gaussdb=#DO INSTEAD INSERT INTO def_test SELECT new.*;
--删除RULE def_view_test_ins
gaussdb=# DROP RULE def_view_test_ins ON def_view_test;
--删除表def_test、视图def_view_test
gaussdb=# DROP VIEW def_view_test;
gaussdb=# DROP TABLE def_test;
```

7.14.127 DROP SCHEMA

功能描述

从数据库中删除模式。

注意事项

- 只有SCHEMA的所有者或者被授予了模式DROP权限的用户可以限执行DROP SCHEMA命令，系统管理员默认拥有此权限。
- 只有初始用户和运维管理员可以对运维管理员执行DROP SCHEMA，其他用户无法对运维管理员执行DROP SCHEMA。
- allow_system_table_mods关闭时，禁止删除DBE_PLDEVELOPER。

语法格式

```
DROP SCHEMA [ IF EXISTS ] schema_name [, ...] [ CASCADE | RESTRICT ];
```


参数说明

- **IF EXISTS**
  如果指定的模式不存在，发出一个notice而不是抛出一个错误。
- **schema_name**
  模式的名称。
  取值范围：已存在模式名。
- **CASCADE | RESTRICT**
  - CASCADE：自动删除包含在模式中的对象。
  - RESTRICT：如果模式包含任何对象，则删除失败（缺省行为）。

须知

不要随意删除pg_temp或pg_toast_temp开头的模式，这些模式是系统内部使用的，如果删除，可能导致无法预知的结果。

说明

无法删除当前模式。如果要删除当前模式，须切换到其他模式下。

示例

请参见CREATE SCHEMA的**示例**。

相关链接

ALTER SCHEMA，**CREATE SCHEMA**。

7.14.128 DROP SEQUENCE

功能描述

从当前数据库里删除序列。

注意事项

- 序列的所有者、序列所在模式的所有者、被授予了序列DROP权限的用户或者被授予了DROP ANY SEQUENCE权限的用户才能删除。系统管理员默认拥有该权限。
- 如果SEQUENCE被创建时使用了LARGE标识，DROP时也需要使用LARGE标识。

语法格式

```
DROP [ LARGE ] SEQUENCE [ IF EXISTS ] {[schema.]sequence_name} [ , ... ] [ CASCADE | RESTRICT ];
```

参数说明

- **IF EXISTS**
  如果指定的序列不存在，则发出一个notice而不是抛出一个错误。
- **sequence_name**

序列名称。

- **CASCADE**
  级联删除依赖序列的对象。
- **RESTRICT**
  如果存在任何依赖的对象，则拒绝删除序列。此项是缺省值。

示例

```
--创建一个名为serial的递增序列，从101开始。
gaussdb=# CREATE SEQUENCE serial START 101;

--删除序列。
gaussdb=# DROP SEQUENCE serial;
```

相关链接

ALTER SEQUENCE， **DROP SEQUENCE**

7.14.129 DROP SERVER

功能描述

删除现有的一个数据服务器。

注意事项

只有SERVER的所有者或者被授予了SERVER的DROP权限的用户才可以删除，系统管理员默认拥有该权限。

语法格式

```
DROP SERVER [ IF EXISTS ] server_name [ CASCADE | RESTRICT ] ;
```

参数描述

- **IF EXISTS**
  如果指定的数据服务器不存在，则发出一个notice而不是抛出一个错误。
- **server_name**
  服务器名称。
- **CASCADE | RESTRICT**
  - CASCADE：级联删除依赖于SERVER的对象 。
  - RESTRICT（缺省值）：如果存在依赖对象，则拒绝删除该SERVER。

示例

```
--创建server。
gaussdb=# CREATE SERVER my_server FOREIGN DATA WRAPPER log_fdw;
CREATE SERVER

--删除my_server。
gaussdb=# DROP SERVER my_server;
DROP SERVER
```


相关链接

ALTER SERVER，**CREATE SERVER**

7.14.130 DROP SYNONYM

功能描述

删除指定的SYNONYM对象。

注意事项

SYNONYM的所有者或者被授予了DROP ANY SEQUENCE权限的用户有权限执行DROP SYNONYM命令，系统管理员默认拥有此权限。

语法格式

```
DROP SYNONYM [ IF EXISTS ] synonym_name [ CASCADE | RESTRICT ];
```

参数描述

- **IF EXISTS**
  如果指定的同义词不存在，则发出一个notice而不是抛出一个错误。
- **synonym_name**
  同义词名字，可以带模式名。
- **CASCADE | RESTRICT**
  - CASCADE：级联删除依赖同义词的对象（比如视图）。
  - RESTRICT：如果有依赖对象存在，则拒绝删除同义词。此选项为缺省值。

示例

请参考CREATE SYNONYM的**示例**。

相关链接

ALTER SYNONYM，**CREATE SYNONYM**

7.14.131 DROP TABLE

功能描述

删除指定的表。

注意事项

- DROP TABLE删除表后，依赖该表的索引会被删除，而使用到该表的函数和存储过程将无法执行。删除分区表，会同时删除分区表中的所有分区。
- 表的所有者、表所在模式的所有者、被授予了表的DROP权限的用户或被授予DROP ANY TABLE权限的用户，有权删除指定表，系统管理员默认拥有该权限。
- DROP TABLE时，如果被指删除的表作为外键表引用了另一张表，会级联删除被引用表上的触发器，此时需要对被引用表加八级锁，可能造成业务的阻塞。

语法格式

```
DROP TABLE [ IF EXISTS ]
    { [schema.]table_name } [, ...] [ CASCADE | RESTRICT ] [ PURGE ];
```

参数说明

- **IF EXISTS**

  如果指定的表不存在，则发出一个notice而不是抛出一个error。

- **schema**

  模式名称。

- **table_name**

  表名称。

- **CASCADE | RESTRICT**
  - CASCADE：表示允许级联删除依赖于该表的对象（比如视图）。
  - RESTRICT：表示有依赖于该表的对象存在时，该索引无法被删除。此选项为缺省值。

- **PURGE**

  该参数表示即使开启回收站功能，使用DROP TABLE删除表时，也会直接物理删除表，而不是将其放入回收站中。

示例

请参考CREATE TABLE的**示例**。

相关链接

ALTER TABLE，**CREATE TABLE**

7.14.132 DROP TABLESPACE

功能描述

删除一个表空间。

注意事项

- 只有表空间所有者或者被授予了表空间DROP权限的用户有权限执行DROP TABLESPACE命令，系统管理员默认拥有此权限。
- 在删除一个表空间之前，表空间里面不能有任何数据库对象，否则会报错。
- DROP TABLESPACE不支持回滚，因此，不能出现在事务块内部。
- 执行DROP TABLESPACE操作时，如果有另外的会话执行\db查询操作，可能会由于TABLESPACE事务的原因导致查询失败，请重新执行\db查询操作。
- 如果执行DROP TABLESPACE失败，需要再次执行一次DROP TABLESPACE IF EXISTS。

语法格式

```
DROP TABLESPACE [ IF EXISTS ] tablespace_name;
```


参数说明

- **IF EXISTS**
  如果指定的表空间不存在，则发出一个notice而不是抛出一个错误。
- **tablespace_name**
  表空间的名称。
  取值范围：已存在的表空间的名称。

示例

请参见CREATE TABLESPACE的**示例**。

相关链接

ALTER TABLESPACE， **CREATE TABLESPACE**

优化建议

- DROP TABLESPACE
  不支持在事务中删除TABLESPACE。

7.14.133 DROP TRIGGER

功能描述

删除触发器。

注意事项

触发器的所有者或者被授予了DROP ANY TRIGGER权限的用户可以执行DROP TRIGGER操作，系统管理员默认拥有此权限。

语法格式

```
DROP TRIGGER [ IF EXISTS ] trigger_name ON table_name [ CASCADE | RESTRICT ];
```

参数说明

- **IF EXISTS**
  如果指定的触发器不存在，则发出一个notice而不是抛出一个错误。
- **trigger_name**
  要删除的触发器名称。
  取值范围：已存在的触发器。
- **table_name**
  要删除的触发器所在的表名称。
  取值范围：已存在的含触发器的表。
- **CASCADE | RESTRICT**
  - CASCADE：级联删除依赖此触发器的对象。
  - RESTRICT：如果有依赖对象存在，则拒绝删除此触发器。此选项为缺省值。

示例

请参见**CREATE TRIGGER**的**示例**。

相关链接

CREATE TRIGGER，**ALTER TRIGGER**，**ALTER TABLE**

7.14.134 DROP TYPE

功能描述

删除一个用户定义的数据类型。

注意事项

类型的所有者、被授予了类型DROP权限的用户或者被授予了DROP ANY TYPE权限的用户有权限执行DROP TYPE命令，系统管理员默认拥有此权限。

语法格式

```
DROP TYPE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

参数说明

- **IF EXISTS**
  如果指定的类型不存在，那么发出一个notice而不是抛出一个错误。
- **name**
  要删除的类型名(可以有模式修饰)。
- **CASCADE**
  级联删除依赖该类型的对象(比如字段、函数、操作符等)
  RESTRICT
  如果有依赖对象，则拒绝删除该类型（缺省行为）。

示例

请参考CREATE TYPE的**示例**。

相关链接

CREATE TYPE，**ALTER TYPE**

7.14.135 DROP USER

功能描述

删除用户，同时会删除同名的SCHEMA。

注意事项

- 须使用CASCADE级联删除依赖用户的对象（除数据库外）。当删除用户的级联对象时，如果级联对象处于锁定状态，则此级联对象无法被删除，直到对象被解锁或锁定级联对象的进程结束。
- 在数据库中删除用户时，如果依赖用户的对象在其他数据库中或者依赖用户的对象是其他数据库，请用户先手动删除其他数据库中的依赖对象或直接删除依赖数据库，再删除用户。即DROP USER不支持跨数据库进行级联删除。
- 在删除用户时，需要先删除该用户拥有的所有对象并且收回该用户在其他对象上的权限，或者通过指定CASCADE级联删除该用户拥有的对象和被授予的权限。

语法格式

```
DROP USER [ IF EXISTS ] user_name [, ...] [ CASCADE | RESTRICT ];
```

参数说明

- **IF EXISTS**

  如果指定的用户不存在，发出一个notice而不是抛出一个错误。

- **user_name**

  待删除的用户名。

  取值范围：已存在的用户名。

- **CASCADE | RESTRICT**
  - CASCADE：级联删除依赖用户的对象，并收回授予该用户的权限。
  - RESTRICT：如果用户还有任何依赖的对象或被授予了其他对象的权限，则拒绝删除该用户（缺省行为）。

    说明

    在GaussDB中，存在一个配置参数enable_kill_query，此参数在配置文件postgresql.conf中。此参数影响级联删除用户对象的行为：

    - 当参数enable_kill_query为on，且使用CASCADE模式删除用户时，会自动kill锁定用户级联对象的进程，并删除用户。
    - 当参数enable_kill_query为off，且使用CASCADE模式删除用户时，会等待锁定级联对象的进程结束之后再删除用户。

示例

请参考CREATE USER的**示例**。

相关链接

ALTER USER，**CREATE USER**

7.14.136 DROP USER MAPPING

功能描述

移除一个用于外部服务器的用户映射。

语法格式

```
DROP USER MAPPING [ IF EXISTS ] FOR { user_name | USER | CURRENT_USER | PUBLIC } SERVER
server_name;
```

参数描述

- **IF EXISTS**

  如果该用户映射不存在则不要抛出一个错误，而是发出一个提示。

- **user_name**

  该映射的用户名。

  CURRENT_USER和USER匹配当前用户的名称。PUBLIC被用来匹配系统中所有现存和未来的用户名。

- **server_name**

  用户映射的服务器名。

相关链接

ALTER USER MAPPING，**CREATE USER MAPPING**

7.14.137 DROP VIEW

功能描述

DROP VIEW语句用于删除数据库中的视图。

注意事项

视图的所有者、视图所在模式的所有者、被授予了视图DROP权限的用户或拥有DROP ANY TABLE权限的用户，有权限执行DROP VIEW的命令，系统管理员默认拥有此权限。

语法格式

```
DROP VIEW [ IF EXISTS ] view_name [, ...] [ CASCADE | RESTRICT ];
```

参数说明

- **IF EXISTS**

  如果指定的视图不存在，则发出一个notice而不是抛出一个错误。

- **view_name**

  要删除的视图名称。

  取值范围：已存在的视图。

- **CASCADE | RESTRICT**
  - CASCADE：级联删除依赖此视图的对象（比如其他视图）。
  - RESTRICT：如果有依赖对象存在，则拒绝删除此视图。此选项为缺省值。

示例

请参见CREATE VIEW的**示例**。

相关链接

ALTER VIEW，**CREATE VIEW**

7.14.138 DROP WEAK PASSWORD DICTIONARY

功能描述

清空gs_global_config中的所有弱口令。

注意事项

只有初始用户、系统管理员和安全管理员拥有权限执行本语法。

语法格式

```
DROP WEAK PASSWORD DICTIONARY;
```

示例

参见CREATE WEAK PASSWORD DICTIONARY的**示例**。

相关链接

CREATE WEAK PASSWORD DICTIONARY

7.14.139 EXECUTE

功能描述

执行一个前面准备好的预备语句。因为一个预备语句只在会话的生命期里存在，所以预备语句必须是在当前会话的前些时候用PREPARE语句创建的。

注意事项

如果创建预备语句时，PREPARE语句声明了一些参数，那么传递给EXECUTE语句的必须是一个兼容的参数集，否则会出现错误。

语法格式

```
EXECUTE name [ ( parameter [, ...] ) ];
```

参数说明

- **name**

  要执行的预备语句的名称。

- **parameter**

  给预备语句的一个参数的具体数值。它必须是一个和生成与创建这个预备语句时指定参数的数据类型相兼容的值的表达式，不支持ROWNUM作为参数。

示例

```
--创建SCHEMA。
gaussdb=# CREATE SCHEMA tpcds;

--创建表reason。
gaussdb=# CREATE TABLE tpcds.reason (
    CD_DEMO_SK          INTEGER          NOT NULL,
    CD_GENDER           character(16)            ,
    CD_MARITAL_STATUS   character(100)
)
;

--插入数据。
gaussdb=# INSERT INTO tpcds.reason VALUES(51, 'AAAAAAAADDAAAAAA', 'reason 51');

--创建表reason_t1。
gaussdb=# CREATE TABLE tpcds.reason_t1 AS TABLE tpcds.reason;

--为一个INSERT语句创建一个预备语句然后执行它。
gaussdb=# PREPARE insert_reason(integer,character(16),character(100)) AS INSERT INTO tpcds.reason_t1
VALUES($1,$2,$3);

gaussdb=# EXECUTE insert_reason(52, 'AAAAAAAADDAAAAAA', 'reason 52');

--删除表reason和reason_t1。
gaussdb=# DROP TABLE tpcds.reason;
gaussdb=# DROP TABLE tpcds.reason_t1;

--删除SCHEMA。
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

7.14.140 EXPDP DATABASE

功能描述

导出DATABASE的全部物理文件。

语法格式

```
EXPDP DATABASE db_name LOCATION = 'directory';
```

参数说明

- **db_name**
  导出的库名。
- **directory**
  导出文件的存储目录。

示例

```
expdp database test location = '/data1/expdp/database';
```

7.14.141 EXPDP TABLE

功能描述

导出与表相关的索引、sequence、分区、toast、toast index等全部文件。

语法格式

```
EXPDP TABLE table_name LOCATION = 'directory';
```

参数说明

- **table_name**
  导出的表名。
- **directory**
  导出文件的存储目录。

示例

```
expdp table test_t location = '/data1/expdp/table0';
```

7.14.142 EXPLAIN

功能描述

显示SQL语句的执行计划。

执行计划将显示SQL语句所引用的表会采用什么样的扫描方式，如：简单的顺序扫描、索引扫描等。如果引用了多个表，执行计划还会显示用到的JOIN算法。

执行计划的最关键的部分是语句的预计执行开销，是指计划生成器估算执行该语句将花费多长的时间。

若指定了ANALYZE选项，则该语句会被执行，然后根据实际的运行结果显示统计数据，包括每个计划节点内时间总开销（毫秒为单位）和实际返回的总行数。这对于判断计划生成器的估计值是否接近实际非常有用。

注意事项

在指定ANALYZE选项时，语句会被执行。如果用户想使用EXPLAIN分析INSERT，UPDATE，DELETE，CREATE TABLE AS或EXECUTE语句，而不想改动数据（执行这些语句会影响数据），请使用如下方法。

```
START TRANSACTION;
EXPLAIN ANALYZE ...;
ROLLBACK;
```

语法格式

- 显示SQL语句的执行计划，支持多种选项，对选项顺序无要求。

  ```
  EXPLAIN [ ( option [, ...] ) ] statement;
  ```

  其中选项option子句的语法为。

  ```
  ANALYZE [ boolean ] |
    ANALYSE [ boolean ] |
    VERBOSE [ boolean ] |
    COSTS [ boolean ] |
    CPU [ boolean ] |
    DETAIL [ boolean ] |
    BUFFERS [ boolean ] |
    TIMING [ boolean ] |
    PLAN [ boolean ] |
    BLOCKNAME [ boolean ] |
    FORMAT { TEXT | XML | JSON | YAML }
    OPTEVAL [ boolean ]
  ```


- 显示SQL语句的执行计划，且要按顺序给出选项。
  ```
  EXPLAIN { [ ANALYZE | ANALYSE ] [ VERBOSE ] | PERFORMANCE } statement;
  ```

参数说明

- **statement**

  指定要分析的SQL语句。

- **ANALYZE boolean | ANALYSE boolean**

  显示实际运行时间和其他统计数据。当两个参数同时使用时，在option中排在后面的一个生效。

  取值范围：

  - TRUE（缺省值）：显示实际运行时间和其他统计数据。
  - FALSE：不显示。

- **VERBOSE boolean**

  显示有关计划的额外信息。

  取值范围：

  - TRUE（缺省值）：显示额外信息。
  - FALSE：不显示。

- **COSTS boolean**

  包括每个规划节点的估计总成本，以及估计的行数和每行的宽度。

  取值范围：

  - TRUE（缺省值）：显示估计总成本和宽度。
  - FALSE：不显示。

- **CPU boolean**

  打印CPU的使用情况的信息。

  取值范围：

  - TRUE（缺省值）：显示CPU的使用情况。
  - FALSE：不显示。

- **DETAIL boolean**

  打印数据库节点上的信息。

  取值范围：

  - TRUE（缺省值）：打印数据库节点的信息。
  - FALSE：不打印。

- **BUFFERS boolean**

  包括缓冲区的使用情况的信息。需要结合ANALYZE或ANALYSE选项一起使用。

  取值范围：

  - TRUE：显示缓冲区的使用情况。
  - FALSE（缺省值）：不显示。

- **TIMING boolean**

  包括实际的启动时间和花费在输出节点上的时间信息。需要结合ANALYZE或ANALYSE选项一起使用。

  取值范围：

  - TRUE（缺省值）：显示启动时间和花费在输出节点上的时间信息。
  - FALSE：不显示。

- **PLAN boolean**

  是否将执行计划存储在PLAN_TABLE中。当该选项开启时，会将执行计划存储在PLAN_TABLE中，不打印到当前屏幕，因此该选项为on时，不能与其他选项同时使用。

  取值范围：

  - ON（缺省值）：将执行计划存储在plan_table中，不打印到当前屏幕。执行成功返回EXPLAIN SUCCESS。
  - OFF：不存储执行计划，将执行计划打印到当前屏幕。

- **BLOCKNAME boolean**

  是否显示计划的每个操作所处于的查询块。当该选项开启时，会将每个操作所处于的查询块的名字输出在Query Block列上，方便用户获取查询块名字，并使用Hint修改执行计划：

  - TRUE（缺省值）：显示计划时，将每个操作所处于的查询块的名字输出在新增列Query Block列上。该选项需要在pretty 模式下使用。见**指定Hint所处于的查询块Queryblock**。
  - FALSE：不对计划显示产生影响。

- **FORMAT**

  指定输出格式。

  取值范围：TEXT，XML，JSON和YAML。

  默认值：TEXT。

- **PERFORMANCE**

  使用此选项时，即打印执行中的所有相关信息。下述为部分信息描述：

  - ex c/r：代表平均每行使用cpu周期数，等于（ex cyc）/（ex row）。
  - ex row：执行行数。
  - ex cyc：代表使用的cpu周期数。
  - inc cyc：代表包含子节点使用的总cpu周期数。
  - shared hit：代表算子的share buffer命中情况。
  - loops：算子循环执行次数。
  - total_calls：生成元素总数。
  - remote query poll time stream gather：算子用于侦听各DN数据到达CN的网络poll时间。
  - deserialize time：反序列化所需时间。
  - estimated time：估计时间。

- **OPTEVAL boolean**

  是否显示SCAN算子（当前仅支持seqscan、indexscan、indexonlyscan、bitmapheapscan）的代价淘汰明细，当开启此开关的时候，会在执行计划中显示一个名字为Cost Evaluation Info (identified by plan id)的计划块，该选项仅仅可以和costs、verbose、format三个选项共存。此计划块中的具体参数明细，请参考**示例2**。

  取值范围：

  - TRUE：显示SCAN算子的代价淘汰明细。

- FALSE（缺省值）：不显示。

示例 1

```
--创建SCHEMA。
gaussdb=# CREATE SCHEMA tpcds;

--创建表tpcds.customer_address。
gaussdb=# CREATE TABLE tpcds.customer_address
(
ca_address_sk         INTEGER           NOT NULL,
ca_address_id         CHARACTER(16)     NOT NULL
);

--向表中插入多条记录。
gaussdb=# INSERT INTO tpcds.customer_address VALUES (5000, 'AAAAAAAABAAAAAAA'),(10000,
'AAAAAAAACAAAAAAA');

--创建一个表tpcds.customer_address_p1。
gaussdb=# CREATE TABLE tpcds.customer_address_p1 AS TABLE tpcds.customer_address;

--修改explain_perf_mode为normal。
gaussdb=# SET explain_perf_mode=normal;

--显示表简单查询的执行计划。
gaussdb=# EXPLAIN SELECT * FROM tpcds.customer_address_p1;
QUERY PLAN
--------------------------------------------------
Data Node Scan  (cost=0.00..0.00 rows=0 width=0)
Node/s: All dbnodes
(2 rows)

--以JSON格式输出的执行计划（explain_perf_mode为normal时）。
gaussdb=# EXPLAIN(FORMAT JSON) SELECT * FROM tpcds.customer_address_p1;
              QUERY PLAN
--------------------------------------
 [                              +
   {                            +
     "Plan": {                    +
       "Node Type": "Data Node Scan",+
       "Startup Cost": 0.00,        +
       "Total Cost": 0.00,          +
       "Plan Rows": 0,              +
       "Plan Width": 0,             +
       "Node/s": "All dbnodes"     +
     }                            +
   }                            +
 ]
(1 row)

--如果有一个索引，当使用一个带索引WHERE条件的查询，可能会显示一个不同的计划。
gaussdb=# EXPLAIN SELECT * FROM tpcds.customer_address_p1 WHERE ca_address_sk=10000;
QUERY PLAN
--------------------------------------------------
Data Node Scan  (cost=0.00..0.00 rows=0 width=0)
Node/s: dn_6005_6006
(2 rows)

--以YAML格式输出的执行计划（explain_perf_mode为normal时）。
gaussdb=# EXPLAIN(FORMAT YAML) SELECT * FROM tpcds.customer_address_p1 WHERE
ca_address_sk=10000;
           QUERY PLAN
---------------------------------
 - Plan:                      +
     Node Type: "Data Node Scan"+
     Startup Cost: 0.00        +
     Total Cost: 0.00          +
     Plan Rows: 0              +
     Plan Width: 0             +
```



```
    Node/s: "dn_6005_6006"
(1 row)

--禁止开销估计的执行计划。
gaussdb=# EXPLAIN(COSTS FALSE)SELECT * FROM tpcds.customer_address_p1 WHERE
ca_address_sk=10000;
      QUERY PLAN
-----------------------
 Data Node Scan
   Node/s: dn_6005_6006
(2 rows)

--带有聚集函数查询的执行计划。
gaussdb=# EXPLAIN SELECT SUM(ca_address_sk) FROM tpcds.customer_address_p1 WHERE
ca_address_sk<10000;
                                QUERY PLAN
---------------------------------------------------------------------------------
 Aggregate  (cost=18.19..14.32 rows=1 width=4)
   ->  Streaming (type: GATHER)  (cost=18.19..14.32 rows=3 width=4)
         Node/s: All dbnodes
         ->  Aggregate  (cost=14.19..14.20 rows=3 width=4)
               ->  Seq Scan on customer_address_p1  (cost=0.00..14.18 rows=10 width=4)
                     Filter: (ca_address_sk < 10000)
(6 rows)


--创建一个二级分区表。
gaussdb=# CREATE TABLE range_list
gaussdb-# (
gaussdb(#     month_code VARCHAR2 ( 30 ) NOT NULL ,
gaussdb(#     dept_code  VARCHAR2 ( 30 ) NOT NULL ,
gaussdb(#     user_no    VARCHAR2 ( 30 ) NOT NULL ,
gaussdb(#     sales_amt  int
gaussdb(# )
gaussdb-# PARTITION BY RANGE (month_code) SUBPARTITION BY LIST (dept_code)
gaussdb-# (
gaussdb(#   PARTITION p_201901 VALUES LESS THAN( '201903' )
gaussdb(#   (
gaussdb(#     SUBPARTITION p_201901_a values ('1'),
gaussdb(#     SUBPARTITION p_201901_b values ('2')
gaussdb(#   ),
gaussdb(#   PARTITION p_201902 VALUES LESS THAN( '201910' )
gaussdb(#   (
gaussdb(#     SUBPARTITION p_201902_a values ('1'),
gaussdb(#     SUBPARTITION p_201902_b values ('2')
gaussdb(#   )
gaussdb(# );
CREATE TABLE

--执行带有二级分区表的查询语句。
--Iterations 和 Sub Iterations分别标识遍历了几个一级分区和二级分区。
--Selected Partitions标识哪些一级分区被实际扫描，Selected Subpartitions: (p:s)标识第p个一级分区下s个二级
分区被实际扫描，如果一级分区下所有二级分区都被扫描则s显示为ALL。
gaussdb=# EXPLAIN SELECT * FROM range_list WHERE dept_code = '1';
                              QUERY PLAN
---------------------------------------------------------------------------------
 Partition Iterator  (cost=0.00..13.81 rows=2 width=238)
   Iterations: 2, Sub Iterations: 2
   ->  Partitioned Seq Scan on range_list  (cost=0.00..13.81 rows=2 width=238)
         Filter: ((dept_code)::text = '1'::text)
         Selected Partitions:  1..2
         Selected Subpartitions:  1:1, 2:1
(6 rows)

--删除表tpcds.customer_address_p1。
gaussdb=# DROP TABLE tpcds.customer_address_p1;

--删除表tpcds.customer_address。
gaussdb=# DROP TABLE tpcds.customer_address;
```



```
--删除SCHEMA。
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

示例 2

```
--创建tb_a，tb_b。
gaussdb=# CREATE TABLE tb_a(c1 int);
gaussdb=# INSERT INTO tb_a VALUES(1),(2),(3);
gaussdb=# CREATE TABLE tb_b AS SELECT * FROM tb_a;
gaussdb=# EXPLAIN (OPTEVAL on )SELECT * FROM tb_a a, tb_b b WHERE a.c1=b.c1 AND a.c1=1;
 id |                  operation                  | E-rows | E-width |     E-costs
----+---------------------------------------------+--------+---------+-----------------
  1 | ->  Nested Loop (2,3)                       |   2401 |      30 | 0.000..312.321
  2 |    ->  Index Scan using tb_a_idx_c1 on tb_a a |     49 |      15 | 0.000..141.090
  3 |    ->  Materialize                          |     49 |      15 | 0.000..141.342
  4 |       ->  Index Scan using tb_b_idx_c1 on tb_b b |     49 |      15 | 0.000..141.097
(4 rows)

 Predicate Information (identified by plan id)
-----------------------------------------------
   2 --Index Scan using tb_a_idx_c1 on tb_a a
         Index Cond: (c1 = 1)
   4 --Index Scan using tb_b_idx_c1 on tb_b b
         Index Cond: (c1 = 1)
(4 rows)

                                                Cost Evaluation Info (identified by plan id)
-----------------------------------------------------------------------------------------------------------------------------------------------
------------------------------
   2 --Index Scan using tb_a_idx_c1 on tb_a a(id=#2#, rpages=1.00, ipages=1.00, selec=0.00488000, ml=0,
iscost=1, lossy=0, uidx=0)
       -> Seq Scan on tb_a a(id=#1#)
            Filter: (c1 = 1)
            Cost Info: (cost=0.00..180.00 tuples=10000.00, rpages=55.00, ipages=-1.00, selec=-1.00000000,
ml=0, iscost=0, lossy=0, uidx=0)
       -> Bitmap Heap Scan on tb_a a(id=#3#)
            Recheck Cond: (c1 = 1)
            Cost Info: (cost=10000000004.63..1000000006104.52 tuples=49.00, rpages=34.00, ipages=0.00,
selec=0.00488000, ml=0, iscost=0, lossy=0, uidx=0)
            -> Bitmap Index Scan using tb_a_idx_c1(id=#2#)
                 Index Cond: (c1 = 1)
                 Cost Info: (cost=0.00..4.62 tuples=49.00, rpages=1.00, ipages=1.00, selec=0.00488000, ml=0,
iscost=1, lossy=0, uidx=0)
   4 --Index Scan using tb_b_idx_c1 on tb_b b(id=#5#, rpages=1.00, ipages=1.00, selec=0.00491949, ml=1,
iscost=1, lossy=0, uidx=0)
       -> Seq Scan on tb_b b(id=#4#)
            Filter: (c1 = 1)
            Cost Info: (cost=0.00..180.00 tuples=10000.00, rpages=55.00, ipages=-1.00, selec=-1.00000000,
ml=0, iscost=0, lossy=0, uidx=0)
       -> Bitmap Heap Scan on tb_b b(id=#6#)
            Recheck Cond: (c1 = 1)
            Cost Info: (cost=10000000004.63..1000000006104.52 tuples=49.00, rpages=34.00, ipages=0.00,
selec=0.00491949, ml=0, iscost=0, lossy=0, uidx=0)
            -> Bitmap Index Scan using tb_b_idx_c1(id=#5#)
                 Index Cond: (c1 = 1)
                 Cost Info: (cost=0.00..4.62 tuples=49.00, rpages=1.00, ipages=1.00, selec=0.00491949, ml=1,
iscost=1, lossy=0, uidx=0)
(20 rows)
--删除表tb_a，tb_b。
gaussdb=# DROP TABLE tb_a;
gaussdb=# DROP TABLE tb_b;
```


说明

针对Cost Evaluation Info (identified by plan id)计划块：

1. “2 --”与“4 --”所在行表示的是当前胜选算子，它们各自下面的缩进计划块表示的是当前胜选算子直接淘汰的算子。如算子2淘汰的算子有Seq Scan算子和Bitmap Heap Scan算子。
2. 如上所示，对上述看到的关键参数进行说明：
    1. id表示当前算子从指定id的path转换而来，该值主要用于在debug2日志中方便定位某一条路径。
    2. rpage表示在代价模型计算过程中使用的基表的页面数。
    3. ipage表示在代价模型计算过程中使用的索引的页面数。
    4. tuples表示在代价模型计算过程中使用的tuple的数量。
    5. selec表示在代价模型计算过程中使用的索引选择率，-1表示当前算子的索引选择率无效。
    6. ml表示在代价模型计算过程中，当前触发的ML模型事件，1表示effective_cache_size指定的缓存足够，2表示effective_cache_size指定的缓存不足，3表示effective_cache_size指定的缓存严重不足。
    7. iscost表示在模型代价计算过程中是否发生过忽略启动代价的事件。
    8. lossy表示在代价模型计算过程中是否触发bitmap heap scan的lossy机制。
    9. uidx表示在代价模型计算过程中是否触发唯一索引优先规则。

相关链接

ANALYZE | ANALYSE

7.14.143 EXPLAIN PLAN

功能描述

通过EXPLAIN PLAN命令可以将查询执行的计划信息存储于PLAN_TABLE表中。与EXPLAIN命令不同的是，EXPLAIN PLAN仅将计划信息进行存储，而不会打印到屏幕。

语法格式

```
EXPLAIN PLAN
[ SET STATEMENT_ID = name ]
FOR statement ;
```

参数说明

- **EXPLAIN PLAN**

  其中的PLAN选项表示需要将计划信息存储于PLAN_TABLE中，存储成功将返回“EXPLAIN SUCCESS”。

- **STATEMENT_ID**

  用户可通过STATEMENT_ID对查询设置标签，输入的标签信息也将存储于PLAN_TABLE中。

- **name**

  查询标签。

  取值范围：字符串

说明

用户在执行EXPLAIN PLAN时，如果没有设置STATEMENT_ID，则默认为空值。同时，用户可输入的STATEMENT_ID最大长度为30个字节，超过长度将会产生报错。

注意事项

- EXPLAIN PLAN不支持在数据库节点上执行。
- 对于执行错误的SQL无法进行计划信息的收集。
- PLAN_TABLE中的数据是session级生命周期并且session隔离和用户隔离，用户只能看到当前session、当前用户的数据。

示例 1

使用EXPLAIN PLAN收集SQL语句的执行计划，通常包括以下步骤：

步骤1 执行EXPLAIN PLAN。

说明

执行EXPLAIN PLAN 后会将计划信息自动存储于PLAN_TABLE中，不支持对PLAN_TABLE进行INSERT、UPDATE、ANALYZE等操作。

PLAN_TABLE详细介绍见**PLAN_TABLE**。

```
--创建表foo1,foo2。
gaussdb=# CREATE TABLE foo1(f1 int, f2 text, f3 text[]);
gaussdb=# CREATE TABLE foo2(f1 int, f2 text, f3 text[]);

--执行EXPLAIN PLAN。
gaussdb=# EXPLAIN PLAN SET STATEMENT_ID = 'TPCH-Q4' FOR SELECT f1, count(*) FROM foo1 WHERE f1
> 1 AND f1 < 3 AND EXISTS (SELECT * FROM foo2) GROUP BY f1;
```

步骤2 查询PLAN_TABLE。

```
gaussdb=# SELECT * FROM plan_table;
```

步骤3 清理PLAN_TABLE表中的数据。

```
gaussdb=# DELETE FROM plan_table WHERE STATEMENT_ID = 'TPCH-Q4';
gaussdb=# DROP TABLE foo1;
gaussdb=# DROP TABLE foo2;
```

----结束

7.14.144 FETCH

功能描述

FETCH通过已创建的游标来检索数据。

每个游标都有一个供FETCH使用的关联位置。游标的关联位置可以在查询结果的第一行之前，或者在结果中的任意行，或者在结果的最后一行之后：

- 游标刚创建完之后，关联位置在第一行之前。
- 在抓取了一些移动行之后，关联位置在检索到的最后一行上。
- 如果FETCH抓取完了所有可用行，它就停在最后一行后面，或者在反向抓取的情况下是停在第一行前面。
- FETCH ALL或FETCH BACKWARD ALL总是把游标的关联位置放在最后一行或者在第一行前面。

注意事项

- 如果游标定义了NO SCROLL，则不允许使用例如FETCH BACKWARD之类的反向抓取。
- NEXT，PRIOR，FIRST，LAST，ABSOLUTE，RELATIVE形式在恰当地移动游标之后抓取一条记录。如果后面没有数据行，就返回一个空的结果，此时游标就会停在查询结果的最后一行之后（向后查询时）或者第一行之前（向前查询时）。
- FORWARD和BACKWARD形式在向前或者向后移动的过程中抓取指定的行数，然后把游标定位在最后返回的行上；或者，如果count大于可用的行数，则在所有行之后（向后查询时）或者之前（向前查询时）。
- RELATIVE 0，FORWARD 0，BACKWARD 0都要求在不移动游标的前提下抓取当前行，也就是重新抓取最近刚抓取过的行。除非游标定位在第一行之前或者最后一行之后，否则这个动作都应该成功。而当游标定位在第一行之前或者最后一行之后，不返回任何行。

语法格式

```
FETCH [ direction { FROM | IN } ] cursor_name;
```

其中direction子句为可选参数。

```
NEXT
  | PRIOR
  | FIRST
  | LAST
  | ABSOLUTE count
  | RELATIVE count
  | count
  | ALL
  | FORWARD
  | FORWARD count
  | FORWARD ALL
  | BACKWARD
  | BACKWARD count
  | BACKWARD ALL
```

参数说明

- **direction**

  定义抓取数据的方向。

  取值范围：

  - NEXT（缺省值）

    从当前关联位置开始，抓取下一行。
  - PRIOR

    从当前关联位置开始，抓取上一行。
  - FIRST

    抓取查询的第一行（和ABSOLUTE 1相同）。
  - LAST

    抓取查询的最后一行（和ABSOLUTE -1相同）。
  - ABSOLUTE count

    抓取查询中第count行。

    ABSOLUTE抓取不会比用相对位移移动到需要的数据行更快，因为下层的实现必须遍历所有中间的行。

count取值范围：有符号的整数

- count为正数，就从查询结果的第一行开始，抓取第count行。
- count为负数，就从查询结果末尾抓取第abs(count)行。
- count为0时，定位在第一行之前。

– RELATIVE count
从当前关联位置开始，抓取随后或前面的第count行。
取值范围：有符号的整数

- count为正数就抓取当前关联位置之后的第count行。
- count为负数就抓取当前关联位置之前的第abs(count)行。
- 如果当前行没有数据的话，RELATIVE 0返回空。

– count
抓取随后的count行（和FORWARD count一样）。
– ALL
从当前关联位置开始，抓取所有剩余的行（和FORWARD ALL一样）。
– FORWARD
抓取下一行（和NEXT一样）。
– FORWARD count
从当前关联位置开始，抓取随后或前面的count行。
– FORWARD ALL
从当前关联位置开始，抓取所有剩余行。
– BACKWARD
从当前关联位置开始，抓取前面一行(和PRIOR一样) 。
– BACKWARD count
从当前关联位置开始，抓取前面的count行（向后扫描）。
取值范围：有符号的整数

- count为正数就抓取当前关联位置之前的count行。
- count为负数就抓取当前关联位置之后的abs（count）行。
- 如果有数据的话，BACKWARD 0重新抓取当前行。

– BACKWARD ALL
从当前关联位置开始，抓取所有前面的行（向后扫描） 。

- **{ FROM | IN } cursor_name**
使用关键字FROM或IN指定游标名称。
取值范围：已创建的游标的名称。

示例

```
--创建一个schema。
gaussdb=# CREATE SCHEMA tpcds;
```



```
--创建表tpcds.customer_address。
gaussdb=# CREATE TABLE tpcds.customer_address
(
ca_address_sk         INTEGER           NOT NULL,
ca_address_id         CHARACTER(16)     NOT NULL,
ca_street_number      INTEGER                   ,
ca_street_name        CHARACTER (20)
);

--向表中插入多条记录。
gaussdb=# INSERT INTO tpcds.customer_address VALUES (1, 'AAAAAAAABAAAAAAA', '18', 'Jackson'),(2,
'AAAAAAAACAAAAAAA', '362', 'Washington 6th'),(3, 'AAAAAAAADAAAAAAA', '585', 'Dogwood Washington');

--SELECT语句，用一个游标读取一个表。开始一个事务。
gaussdb=# START TRANSACTION;

--建立一个名为cursor1的游标。
gaussdb=# CURSOR cursor1 FOR SELECT * FROM tpcds.customer_address ORDER BY 1;

--抓取头3行到游标cursor1里。
gaussdb=# FETCH FORWARD 3 FROM cursor1;
 ca_address_sk |  ca_address_id   | ca_street_number |   ca_street_name
---------------+------------------+------------------+--------------------
             1 | AAAAAAAABAAAAAAA | 18               | Jackson
             2 | AAAAAAAACAAAAAAA | 362              | Washington 6th
             3 | AAAAAAAADAAAAAAA | 585              | Dogwood Washington
(3 rows)

--关闭游标并提交事务。
gaussdb=# CLOSE cursor1;

--结束一个事务。
gaussdb=# END;

--VALUES子句，用一个游标读取VALUES子句中的内容。开始一个事务。
gaussdb=# START TRANSACTION;

--建立一个名为cursor2的游标。
gaussdb=# CURSOR cursor2 FOR VALUES(1,2),(0,3) ORDER BY 1;

--抓取头2行到游标cursor2里。
gaussdb=# FETCH FORWARD 2 FROM cursor2;
column1 | column2
---------+---------
0 |       3
1 |       2
(2 rows)

--关闭游标并提交事务。
gaussdb=# CLOSE cursor2;

--结束一个事务。
gaussdb=# END;

--WITH HOLD游标的使用，开启事务。
gaussdb=# START TRANSACTION;

--创建一个with hold游标。
gaussdb=# DECLARE cursor1 CURSOR WITH HOLD FOR SELECT * FROM tpcds.customer_address ORDER BY
1;

--抓取头2行到游标cursor1里。
gaussdb=# FETCH FORWARD 2 FROM cursor1;
 ca_address_sk |  ca_address_id   | ca_street_number |   ca_street_name
---------------+------------------+------------------+--------------------
             1 | AAAAAAAABAAAAAAA | 18               | Jackson
             2 | AAAAAAAACAAAAAAA | 362              | Washington 6th
(2 rows)
```



```
--结束事务。
gaussdb=# END;

--抓取下一行到游标cursor1里。
gaussdb=# FETCH FORWARD 1 FROM cursor1;
 ca_address_sk |  ca_address_id   | ca_street_number |   ca_street_name
---------------+------------------+------------------+--------------------
             3 | AAAAAAAADAAAAAAA | 585              | Dogwood Washington
(1 row)

--关闭游标。
gaussdb=# CLOSE cursor1;

--删除表tpcds.customer_address。
gaussdb=# DROP TABLE tpcds.customer_address;

--删除SCHEMA。
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

相关链接

CLOSE，**MOVE**

7.14.145 GRANT

功能描述

对角色和用户进行授权操作。

使用GRANT命令进行用户授权包括以下场景：

- **将系统权限授权给角色或用户**

  系统权限又称为用户属性，包括SYSADMIN、CREATEDB、CREATEROLE、AUDITADMIN、MONADMIN、OPRADMIN、POLADMIN、INHERIT、REPLICATION、VCADMIN和LOGIN等。

  系统权限一般通过CREATE/ALTER ROLE语法来指定。其中，SYSADMIN权限可以通过GRANT/REVOKE ALL PRIVILEGE授予或撤销。但系统权限无法通过ROLE和USER的权限被继承，也无法授予PUBLIC。

- **将数据库对象授权给角色或用户**

  将数据库对象（表、视图、指定字段、数据库、函数、模式、表空间等）的相关权限授予特定角色或用户；

  GRANT命令将数据库对象的特定权限授予一个或多个角色。这些权限会追加到已有的权限上。

  关键字PUBLIC表示该权限要赋予所有角色，包括以后创建的用户。PUBLIC可以看做是一个隐含定义好的组，它总是包括所有角色。任何角色或用户都将拥有通过GRANT直接赋予的权限和所属的权限，再加上PUBLIC的权限。

  如果声明了WITH GRANT OPTION，则被授权的用户也可以将此权限赋予他人，否则就不能授权给他人。这个选项不能赋予PUBLIC，这是GaussDB特有的属性。

  GaussDB会将某些类型的对象上的权限授予PUBLIC。默认情况下，对表、表字段、序列、外部数据源、外部服务器、模式或表空间对象的权限不会授予PUBLIC，而以下这些对象的权限会授予PUBLIC：数据库的CONNECT权限和CREATE TEMP TABLE权限、函数的EXECUTE特权、语言和数据类型（包括域）的USAGE特权。对象拥有者可以撤销默认授予PUBLIC的权限并专门授予权限给其他用户。为了更安全，建议在同一个事务中创建对象并设置权限，这样其他用户就没有时间窗口使用该对象。另外可参考安全加固指南的权限控制章节，对PUBLIC

用户组的权限进行限制。这些初始的默认权限可以使用ALTER DEFAULT PRIVILEGES命令修改。

对象的所有者缺省具有该对象上的所有权限，出于安全考虑所有者可以舍弃部分权限，但ALTER、DROP、COMMENT、INDEX、VACUUM以及对象的可再授予权限属于所有者固有的权限，隐式拥有。

- **将角色或用户的权限授权给其他角色或用户**

  将一个角色或用户的权限授予一个或多个其他角色或用户。在这种情况下，每个角色或用户都可视为拥有一个或多个数据库权限的集合。

  当声明了WITH ADMIN OPTION，被授权的用户可以将该权限再次授予其他角色或用户，以及撤销所有由该角色或用户继承到的权限。当授权的角色或用户发生变更或被撤销时，所有继承该角色或用户权限的用户拥有的权限都会随之发生变更。

  数据库系统管理员可以给任何角色或用户授予/撤销任何权限。拥有CREATEROLE权限的角色可以赋予或者撤销任何非系统管理员角色的权限。

- **将ANY权限授予给角色或用户**

  将ANY权限授予特定的角色和用户，ANY权限的取值范围参见语法格式。当声明了WITH ADMIN OPTION，被授权的用户可以将该ANY权限再次授予其他角色/用户，或从其他角色/用户处回收该ANY权限。ANY权限可以通过角色被继承，但不能赋予PUBLIC。初始用户和三权分立关闭时的系统管理员用户可以给任何角色/用户授予或撤销ANY权限。

  目前支持以下ANY权限：CREATE ANY TABLE、ALTER ANY TABLE、DROP ANY TABLE、SELECT ANY TABLE、INSERT ANY TABLE、UPDATE ANY TABLE、DELETE ANY TABLE、CREATE ANY SEQUENCE、CREATE ANY INDEX、CREATE ANY FUNCTION、EXECUTE ANY FUNCTION、 CREATE ANY PACKAGE、EXECUTE ANY PACKAGE、CREATE ANY TYPE、ALTER ANY TYPE、DROP ANY TYPE、ALTER ANY SEQUENCE、DROP ANY SEQUENCE、SELECT ANY SEQUENCE、ALTER ANY INDEX、DROP ANY INDEX、CREATE ANY SYNONYM、DROP ANY SYNONYM、CREATE ANY TRIGGER、ALTER ANY TRIGGER、DROP ANY TRIGGER。详细的ANY权限范围描述参考**表7-149**

注意事项

- 不允许将ANY权限授予PUBLIC，也不允许从PUBLIC回收ANY权限。
- ANY权限属于数据库内的权限，只对授予该权限的数据库内的对象有效，例如SELECT ANY TABLE只允许用户查看当前数据库内的所有用户表数据，对其他数据库内的用户表无查看权限。
- ANY权限与原有的权限相互无影响。
- 如果用户被授予CREATE ANY TABLE权限，在同名schema下创建表的属主是该schema的属主，用户对表进行其他操作时，需要授予相应的操作权限。与此类似的还有CREATE ANY FUNCTION、CREATE ANY PACKAGE、CREATE ANY TYPE、CREATE ANY SEQUENCE和CREATE ANY INDEX，在同名模式下创建的对象的属主是同名模式的属主；而对于CREATE ANY TRIGGER和CREATE ANY SYNONYM，在同名模式下创建的对象的属主为创建者。
- 需要谨慎授予用户CREATE ANY FUNCTION或CREATE ANY PACKAGE的权限，以免其他用户利用DEFINER类型的函数或PACKAGE进行权限提升。
- 通过GRANT授予用于使用表的权限时，如果用户使用不当，可能通过ALTER语法在表的默认值、约束增加表达式，通过创建索引在INDEX上增加表达式等操作，导致权限被利用的风险。

- 通过GRANT授予用户使用TRIGGER的权限时，如果用户使用不当，可能通过WHEN条件创建表达式，当触发器被触发时，存在权限被利用的风险。
- 给用户赋权时，需要特别注意定义者函数/PACKAGE，定义者函数/PACKAGE会使用函数/PACKAGE的owner权限执行，若赋权不当（包括GRANT ROLE TO ROLE），则存在权限被利用风险。

语法格式

- 将表或视图的访问权限赋予指定的用户或角色。

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER | ALTER | DROP
| COMMENT | INDEX | VACUUM } [, ...]
    | ALL [ PRIVILEGES ] }
  ON { [ TABLE ] table_name [, ...]
    | ALL TABLES IN SCHEMA schema_name [, ...] }
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

- 将表中字段的访问权限赋予指定的用户或角色。

```
GRANT { {{ SELECT | INSERT | UPDATE | REFERENCES | COMMENT } ( column_name [, ...] )} [, ...]
    | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
  ON [ TABLE ] table_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

- 将序列的访问权限赋予指定的用户或角色，LARGE字段属性可选，赋权语句不区分序列是否为LARGE。

```
GRANT { { SELECT | UPDATE | USAGE | ALTER | DROP | COMMENT } [, ...]
    | ALL [ PRIVILEGES ] }
  ON { [ [ LARGE ] SEQUENCE ] sequence_name [, ...]
    | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

- 将数据库的访问权限赋予指定的用户或角色。

```
GRANT { { CREATE | CONNECT | TEMPORARY | TEMP | ALTER | DROP | COMMENT } [, ...]
    | ALL [ PRIVILEGES ] }
  ON DATABASE database_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

- 将域的访问权限赋予指定的用户或角色。

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
  ON DOMAIN domain_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

说明

本版本暂时不支持赋予域的访问权限。

- 将客户端加密主密钥CMK的访问权限赋予指定的用户或角色。

```
GRANT { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
  ON CLIENT_MASTER_KEY client_master_key [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

- 将列加密密钥CEK的访问权限赋予指定的用户或角色。

```
GRANT { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
  ON COLUMN_ENCRYPTION_KEY column_encryption_key [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

- 将外部数据源的访问权限赋予给指定的用户或角色。

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
  ON FOREIGN DATA WRAPPER fdw_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```


- 将外部服务器的访问权限赋予给指定的用户或角色。

```
GRANT { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON FOREIGN SERVER server_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

- 将函数的访问权限赋予给指定的用户或角色。

```
GRANT { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON { FUNCTION {function_name ( [ {[ argmode ] [ arg_name ] arg_type} [, ...] ] )} [, ...]
      | ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

- 将存储过程的访问权限赋予给指定的用户或角色。

```
GRANT { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON PROCEDURE {proc_name ( [ {[ argmode ] [ arg_name ] arg_type} [, ...] ] )} [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

- 将过程语言的访问权限赋予给指定的用户或角色。

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON LANGUAGE lang_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

- 将大对象的访问权限赋予指定的用户或角色。

```
GRANT { { SELECT | UPDATE } [, ...] | ALL [ PRIVILEGES ] }
    ON LARGE OBJECT loid [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

说明

本版本暂时不支持大对象。

- 将模式的访问权限赋予指定的用户或角色。

```
GRANT { { CREATE | USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON SCHEMA schema_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

说明

将模式中的表或者视图对象授权给其他用户时，需要将表或视图所属的模式的USAGE权限同时授予该用户，若没有该权限，则只能看到这些对象的名称，并不能实际进行对象访问。 同名模式下创建表的权限无法通过此语法赋予，可以通过将角色的权限赋予其他用户或角色的语法，赋予同名模式下创建表的权限。

- 将表空间的访问权限赋予指定的用户或角色。

```
GRANT { { CREATE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON TABLESPACE tablespace_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

- 将类型的访问权限赋予指定的用户或角色。

```
GRANT { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON TYPE type_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

说明

本版本暂时不支持赋予类型的访问权限。

- 将directory对象的权限赋予指定的角色。

```
GRANT { { READ | WRITE | ALTER | DROP } [, ...] | ALL [PRIVILEGES] }
  ON DIRECTORY directory_name [, ...]
  TO { [GROUP] role_name | PUBLIC } [, ...]
  [WITH GRANT OPTION];
```


- 将package对象的权限赋予指定的角色。

```
GRANT { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [PRIVILEGES] }
  ON { PACKAGE package_name [, ...]
    | ALL PACKAGES IN SCHEMA schema_name [, ...] }
  TO { [GROUP] role_name | PUBLIC } [, ...]
  [WITH GRANT OPTION];
```

- 将角色的权限赋予其他用户或角色的语法。

```
GRANT role_name [, ...]
  TO role_name [, ...]
  [ WITH ADMIN OPTION ];
```

- 将sysadmin权限赋予指定的角色。

```
GRANT ALL { PRIVILEGES | PRIVILEGE }
  TO role_name;
```

- 将ANY权限赋予其他用户或角色的语法。

```
GRANT { CREATE ANY TABLE | ALTER ANY TABLE | DROP ANY TABLE | SELECT ANY TABLE | INSERT
ANY TABLE | UPDATE ANY TABLE |
  DELETE ANY TABLE | CREATE ANY SEQUENCE | CREATE ANY INDEX | CREATE ANY FUNCTION |
EXECUTE ANY FUNCTION |
  CREATE ANY PACKAGE | EXECUTE ANY PACKAGE | CREATE ANY TYPE | ALTER ANY TYPE | DROP ANY
TYPE | ALTER ANY SEQUENCE | DROP ANY SEQUENCE |
  SELECT ANY SEQUENCE | ALTER ANY INDEX | DROP ANY INDEX | CREATE ANY SYNONYM | DROP
ANY SYNONYM | CREATE ANY TRIGGER | ALTER ANY TRIGGER | DROP ANY TRIGGER
 } [, ...]
  TO [ GROUP ] role_name [, ...]
  [ WITH ADMIN OPTION ];
```

- 将DATABASE LINK对象权限赋予指定用户。

```
GRANT { CREATE | ALTER | DROP } [PUBLIC] DATABASE LINK TO role_name;
```

说明

- PUBLIC：指定公共以创建对所有用户可见的公共数据库链接。如果省略此子句，则数据库链接是私有的，只作为兼容接口使用，无实际意义。远程数据库上可访问的数据取决于数据库链接在连接到远程数据库时使用的标识。
- 当赋予用户创建DATABASE LINK权限时，相当于许可用户使用服务端DATABASE的IP对远端进行访问。若不希望有此效果，应不要使用GRANT对用户赋权。
- 现在除DATABASE LINK的直接赋权语句外，还可以通过权限继承和赋予管理员用户获取到DATABASE LINK的相关权限。
- DATABASE LINK详细说明请见**DATABASE LINK**。

参数说明

GRANT的权限分类如下所示。

- **SELECT**

  允许对指定的表、视图、序列执行SELECT命令，UPDATE或DELETE时也需要对应字段上的SELECT权限。

- **INSERT**

  允许对指定的表执行INSERT命令。

- **UPDATE**

  允许对声明的表中任意字段执行UPDATE命令，UPDATE命令也需要SELECT权限来查询出哪些行需要更新。SELECT… FOR UPDATE和SELECT… FOR SHARE除了需要SELECT权限外，还需要UPDATE权限。

- **DELETE**

  允许执行DELETE命令删除指定表中的数据，DELETE命令也需要SELECT权限来查询出哪些行需要删除。

- **TRUNCATE**

  允许执行TRUNCATE语句删除指定表中的所有记录。

- **REFERENCES**

  创建一个外键约束，必须拥有参考表和被参考表的REFERENCES权限。

- **TRIGGER**

  允许在指定的表上创建触发器。

- **CREATE**
  - 对于数据库，允许在数据库里创建新的模式。
  - 对于模式，允许在模式中创建新的对象。如果要重命名一个对象，用户除了必须是该对象的所有者外，还必须拥有该对象所在模式的CREATE权限。
  - 对于表空间，允许在表空间中创建表，允许在创建数据库和模式的时候把该表空间指定为缺省表空间。

- **CONNECT**

  允许用户连接到指定的数据库。

- **EXECUTE**

  允许使用指定的函数，以及利用这些函数实现的操作符。

- **USAGE**
  - 对于过程语言，允许用户在创建函数的时候指定过程语言。
  - 对于模式，USAGE允许访问包含在指定模式中的对象，若没有该权限，则只能看到这些对象的名称。
  - 对于序列，USAGE允许使用nextval函数。

- **ALTER**

  允许用户修改指定对象的属性，但不包括修改对象的所有者和修改对象所在的模式。

- **DROP**

  允许用户删除指定的对象。

- **COMMENT**

  允许用户定义或修改指定对象的注释。

- **INDEX**

  允许用户在指定表上创建索引，并管理指定表上的索引，还允许用户对指定表执行REINDEX和CLUSTER操作。

- **VACUUM**

  允许用户对指定的表执行ANALYZE和VACUUM操作。

- **ALL PRIVILEGES**

  一次性给指定用户/角色赋予所有可赋予的权限。只有系统管理员有权执行GRANT ALL PRIVILEGES。

GRANT的参数说明如下所示。

- **role_name**

  已存在用户名称。

- **table_name**

  已存在表名称。

- **column_name**
  已存在字段名称。
- **schema_name**
  已存在模式名称。
- **database_name**
  已存在数据库名称。
- **function_name**
  已存在函数名称。
- **procedure_name**
  已存在存储过程名称。
- **sequence_name**
  已存在序列名称。
- **domain_name**
  已存在域类型名称。
- **fdw_name**
  已存在外部数据包名称。
- **lang_name**
  已存在语言名称。
- **type_name**
  已存在类型名称。
- **argmode**
  参数模式。
  取值范围：字符串，要符合**标识符命名规范**。
- **arg_name**
  参数名称。
  取值范围：字符串，要符合**标识符命名规范**。
- **arg_type**
  参数类型。
  取值范围：字符串，要符合**标识符命名规范**。
- **loid**
  包含本页的大对象的标识符。
  取值范围：字符串，要符合**标识符命名规范**。
- **tablespace_name**
  表空间名称。
- **client_master_key**
  客户端加密主密钥的名称。
  取值范围：字符串，要符合**标识符命名规范**。
- **column_encryption_key**
  列加密密钥的名称。
  取值范围：字符串，要符合**标识符命名规范**。

- **directory_name**

  目录名称。

  取值范围：字符串，要符合**标识符命名规范**。

- **WITH GRANT OPTION**

  如果声明了WITH GRANT OPTION，则被授权的用户也可以将此权限赋予他人，否则就不能授权给他人。这个选项不能赋予PUBLIC。

非对象所有者给其他用户授予对象权限时，命令按照以下规则执行：

- 如果用户没有该对象上指定的权限，命令立即失败。
- 如果用户有该对象上的部分权限，则GRANT命令只授予他有授权选项的权限。
- 如果用户没有可用的授权选项，GRANT ALL PRIVILEGES形式将发出一个警告信息，其他命令形式将发出在命令中提到的且没有授权选项的相关警告信息。

说明

数据库系统管理员可以访问所有对象，而不会受对象的权限设置影响。这个特点类似Unix系统的root的权限。和root一样，除了必要的情况外，建议不要总是以系统管理员身份进行操作。

- **WITH ADMIN OPTION**

  对于角色，当声明了WITH ADMIN OPTION，被授权的用户可以将该角色再授予其他角色/用户，或从其他角色/用户回收该角色。

  对于ANY权限，当声明了WITH ADMIN OPTION，被授权的用户可以将该ANY权限再授予其他角色/用户，或从其他角色/用户回收该ANY权限。

表 7-149 ANY 权限列表

系统权限名称	描述
CREATE ANY TABLE	用户能够在public模式和用户模式下创建表或视图。如果想要创建serial类型列的表，还需要授予创建序列的权限。
ALTER ANY TABLE	用户拥有对public模式和用户模式下表或视图的ALTER权限。如果想要修改表的唯一索引为表增加主键约束或唯一约束，还需要授予该表的索引权限。
DROP ANY TABLE	用户拥有对public模式和用户模式下表或视图的DROP权限。
SELECT ANY TABLE	用户拥有对public模式和用户模式下表或视图的SELECT权限，仍然受行级访问控制限制。
UPDATE ANY TABLE	用户拥有对public模式和用户模式下表或视图的UPDATE权限，仍然受行级访问控制限制。
INSERT ANY TABLE	用户拥有对public模式和用户模式下表或视图的INSERT权限。
DELETE ANY TABLE	用户拥有对public模式和用户模式下表或视图的DELETE权限，仍然受行级访问控制限制。
CREATE ANY FUNCTION	用户能够在用户模式下创建函数或存储过程。

系统权限名称	描述
EXECUTE ANY FUNCTION	用户拥有在public模式和用户模式下函数或存储过程的EXECUTE权限。
CREATE ANY PACKAGE	用户能够在public模式和用户模式下创建PACKAGE。
EXECUTE ANY PACKAGE	用户拥有在public模式和用户模式下PACKAGE的EXECUTE权限。
CREATE ANY TYPE	用户能够在public模式和用户模式下创建类型。
CREATE ANY SEQUENCE	用户能够在public模式和用户模式下创建序列。
CREATE ANY INDEX	用户能够在public模式和用户模式下创建索引。如果在某表空间创建分区表索引，需要授予用户该表空间的创建权限。
ALTER ANY TYPE	用户拥有对public模式和用户模式下类型的ALTER权限，但不包括修改类型的所有者或者修改类型的模式。
DROP ANY TYPE	用户拥有对public模式和用户模式下类型的DROP权限。
ALTER ANY SEQUENCE	用户拥有对public模式和用户模式下序列的ALTER权限，但不包括修改序列的所有者。
DROP ANY SEQUENCE	用户拥有对public模式和用户模式下序列的DROP权限。
SELECT ANY SEQUENCE	用户拥有对public模式和用户模式下序列的SELECT、USAGE和UPDATE权限。
ALTER ANY INDEX	用户拥有对public模式和用户模式下索引的ALTER权限。如果要重命名索引，还需要索引所在模式下创建对象的权限。如果涉及表空间的操作，还需要对应表空间的相应操作权限。如果设置索引不可用（UNUSABLE），还需要DROP ANY INDEX权限。
DROP ANY INDEX	用户拥有对public模式和用户模式下索引的DROP权限。
CREATE ANY TRIGGER	用户能够在public模式和用户模式下创建触发器。
ALTER ANY TRIGGER	用户拥有对public模式和用户模式下触发器的ALTER权限。
DROP ANY TRIGGER	用户拥有对public模式和用户模式下触发器的DROP权限。
CREATE ANY SYNONYM	用户能够在用户模式下创建同义词。
DROP ANY SYNONYM	用户拥有对public模式和用户模式下同义词的DROP权限。

说明

用户被授予任何一种ANY权限后，用户对public模式和用户模式具有USAGE权限，对**表13-1**中除public之外的系统模式没有USAGE权限。

示例

示例：将系统权限授权给用户或者角色。

创建名为joe的用户，并将sysadmin权限授权给他。

```
gaussdb=# CREATE USER joe PASSWORD '********';
gaussdb=# GRANT ALL PRIVILEGES TO joe;
```

授权成功后，用户joe会拥有sysadmin的所有权限。

示例：将对象权限授权给用户或者角色。

1. 撤销joe用户的sysadmin权限，然后将模式tpcds的使用权限和表tpcds.reason的所有权限授权给用户joe。

   ```
   gaussdb=# CREATE SCHEMA tpcds;
   CREATE SCHEMA
   gaussdb=# CREATE TABLE tpcds.reason
   (
   r_reason_sk         INTEGER      NOT NULL,
   r_reason_id         CHAR(16)     NOT NULL,
   r_reason_desc       VARCHAR(20)
   );
   CREATE TABLE
   gaussdb=# REVOKE ALL PRIVILEGES FROM joe;
   gaussdb=# GRANT USAGE ON SCHEMA tpcds TO joe;
   gaussdb=# GRANT ALL PRIVILEGES ON tpcds.reason TO joe;
   ```

   授权成功后，joe用户就拥有了tpcds.reason表的所有权限，包括增删改查等权限。

2. 将tpcds.reason表中r_reason_sk、r_reason_id、r_reason_desc列的查询权限，r_reason_desc的更新权限授权给joe。

   ```
   gaussdb=# GRANT select (r_reason_sk,r_reason_id,r_reason_desc),update (r_reason_desc) ON tpcds.reason TO joe;
   ```

   授权成功后，用户joe对tpcds.reason表中r_reason_sk，r_reason_id、r_reason_desc的查询权限会立即生效。如果joe用户需要拥有将这些权限授权给其他用户的权限，可以通过以下语法对joe用户进行授权。

   ```
   gaussdb=# GRANT select (r_reason_sk, r_reason_id) ON tpcds.reason TO joe WITH GRANT OPTION;
   ```

   将数据库的连接权限授权给用户joe，并给予其在GaussDB中创建schema的权限，而且允许joe将此权限授权给其他用户。

   ```
   gaussdb=# CREATE DATABASE testdb;
   gaussdb=# GRANT create,connect on database testdb TO joe WITH GRANT OPTION;
   ```

   创建角色tpcds_manager，将模式tpcds的访问权限授权给角色tpcds_manager，并授予该角色在tpcds下创建对象的权限，不允许该角色中的用户将权限授权给其他人。

   ```
   gaussdb=# CREATE ROLE tpcds_manager PASSWORD '********';
   gaussdb=# GRANT USAGE,CREATE ON SCHEMA tpcds TO tpcds_manager;
   ```

   将表空间tpcds_tbspc的所有权限授权给用户joe，但用户joe无法将权限继续授予其他用户。

   ```
   gaussdb=# CREATE TABLESPACE tpcds_tbspc RELATIVE LOCATION 'tablespace/tablespace_1';
   gaussdb=# GRANT ALL ON TABLESPACE tpcds_tbspc TO joe;
   ```


示例：将用户或者角色的权限授权给其他用户或角色。

1. 创建角色manager，将joe的权限授权给manager，并允许该角色将权限授权给其他人。
```
gaussdb=# CREATE ROLE manager PASSWORD '********';
gaussdb=# GRANT joe TO manager WITH ADMIN OPTION;
```
2. 创建用户senior_manager，将用户manager的权限授权给该用户。
```
gaussdb=# CREATE ROLE senior_manager PASSWORD '*********';
gaussdb=# GRANT manager TO senior_manager;
```
3. 撤销权限，并清理用户。
```
gaussdb=# REVOKE joe FROM manager;
gaussdb=# REVOKE manager FROM senior_manager;
gaussdb=# DROP USER manager;
```

示例：将CMK或者CEK的权限授权给其他用户或角色。

1. 连接密态数据库
```
前置条件：请参考《特性指南》中”设置密态等值查询”章节，配置参数并通过CREATE CLIENT MASTER KEY语法创建名为MyCMK1的主密钥。
gaussdb=# CREATE COLUMN ENCRYPTION KEY MyCEK1 WITH VALUES (CLIENT_MASTER_KEY = MyCMK1, ALGORITHM = AEAD_AES_256_CBC_HMAC_SHA256);
CREATE COLUMN ENCRYPTION KEY
```
2. 创建角色newuser，将密钥的权限授权给newuser。
```
gaussdb=# CREATE USER newuser PASSWORD '********';
CREATE ROLE
gaussdb=# GRANT ALL ON SCHEMA public TO newuser;
GRANT
gaussdb=# GRANT USAGE ON COLUMN_ENCRYPTION_KEY MyCEK1 to newuser;
GRANT
gaussdb=# GRANT USAGE ON CLIENT_MASTER_KEY MyCMK1 to newuser;
GRANT
```
3. 设置该用户连接数据库,使用该CEK创建加密表。
```
gaussdb=# SET SESSION AUTHORIZATION newuser PASSWORD '********';
gaussdb=>  CREATE TABLE acltest1 (x int, x2 varchar(50) ENCRYPTED WITH (COLUMN_ENCRYPTION_KEY = MyCEK1, ENCRYPTION_TYPE = DETERMINISTIC));
CREATE TABLE
gaussdb=> SELECT has_cek_privilege('newuser', 'MyCEK1', 'USAGE');
 has_cek_privilege
-------------------
 t
(1 row)
```
4. 撤销权限，并清理用户。
```
gaussdb=# REVOKE USAGE ON COLUMN_ENCRYPTION_KEY MyCEK1 FROM newuser;
gaussdb=# REVOKE USAGE ON CLIENT_MASTER_KEY MyCMK1 FROM newuser;
gaussdb=# DROP TABLE newuser.acltest1;
gaussdb=# DROP COLUMN ENCRYPTION KEY MyCEK1;
gaussdb=# DROP CLIENT MASTER KEY MyCMK1;
gaussdb=# DROP SCHEMA IF EXISTS newuser CASCADE;
gaussdb=# REVOKE ALL ON SCHEMA public FROM newuser;
gaussdb=# DROP ROLE IF EXISTS newuser;
```

示例：撤销上述授予的权限，并清理角色和用户。

```
gaussdb=# REVOKE ALL PRIVILEGES ON tpcds.reason FROM joe;
gaussdb=# REVOKE ALL PRIVILEGES ON SCHEMA tpcds FROM joe;
gaussdb=# REVOKE ALL ON TABLESPACE tpcds_tbspc FROM joe;
gaussdb=# DROP TABLESPACE tpcds_tbspc;
gaussdb=# REVOKE USAGE,CREATE ON SCHEMA tpcds FROM tpcds_manager;
gaussdb=# DROP ROLE tpcds_manager;
gaussdb=# DROP ROLE senior_manager;
gaussdb=# DROP USER joe CASCADE;
gaussdb=# DROP TABLE tpcds.reason;
gaussdb=# DROP SCHEMA tpcds CASCADE;
```


相关链接

REVOKE，**ALTER DEFAULT PRIVILEGES**

7.14.146 IMPDP DATABASE CREATE

功能描述

导入DATABASE的准备阶段。

语法格式

```
IMPDP DATABASE [db_name] CREATE SOURCE = 'directory' OWNER = user [LOCAL];
```

参数说明

- db_name：导入后的新库名，如不指定则导入后保持原库名。
- directory：导入的database的数据源目录。
- user：导入后库的属主。
- LOCAL：指定该字段表示导入到原集群，如不指定表示导入到新集群。

示例

```
impdp database test create source = '/data1/impdp/database' owner=admin;
```

7.14.147 IMPDP RECOVER

功能描述

导入DATABASE的执行阶段。

语法格式

```
IMPDP DATABASE RECOVER SOURCE = 'directory' OWNER = user [LOCAL];
```

参数说明

- directory：导入的database的数据源目录。
- user：导入后库的属主。
- LOCAL：指定该字段表示导入到原集群，如不指定表示导入到新集群。

示例

```
impdp database recover source = '/data1/impdp/database' owner=admin;
```

7.14.148 IMPDP TABLE

功能描述

导入表的执行阶段。

语法格式

```
IMPDP TABLE [AS table_name] SOURCE = 'directory' OWNER = user;
```


参数说明

- table_name：导入后的新表名，如不指定则导入后保持原表名。
- directory：导入的表的数据源目录。
- user：导入后表的属主。

示例

```
impdp table source = '/data1/impdp/table0' owner=admin;
```

7.14.149 IMPDP TABLE PREPARE

功能描述

导入表的准备阶段。

语法格式

```
IMPDP TABLE PREPARE SOURCE = 'directory' OWNER = user;
```

参数说明

- directory：导入的表的数据源目录。
- user：导入后表的属主。

示例

```
impdp table prepare source = '/data1/impdp/table0' owner=admin;
```

7.14.150 INSERT

功能描述

向表中添加一行或多行数据。

注意事项

- 只有拥有表INSERT权限的用户，才可以向表中插入数据。用户被授予INSERT ANY TABLE权限，相当于用户对除系统模式之外的任何模式具有USAGE权限，并且拥有这些模式下表的INSERT权限。
- 如果使用RETURNING子句，用户必须要有该表的SELECT权限。
- 如果使用ON DUPLICATE KEY UPDATE，用户必须要有该表的INSERT、UPDATE权限，UPDATE子句中列的SELECT权限。
- 如果使用query子句插入来自查询里的数据行，用户还需要拥有在查询里使用的表的SELECT权限。
- 生成列不能被直接写入。在INSERT命令中不能为生成列指定值，但是可以指定关键字DEFAULT。
- 当连接到TD兼容的数据库时，td_compatible_truncation参数设置为on时，将启用超长字符串自动截断功能，在后续的INSERT语句中（不包含外表的场景下），对目标表中CHAR和VARCHAR类型的列上插入超长字符串时，系统会自动按照目标表中相应列定义的最大长度对超长字符串进行截断。

> **说明**
>
> 如果向字符集为字节类型编码（SQL_ASCII，LATIN1等）的数据库中插入多字节字符数据（如汉字等），且字符数据跨越截断位置，这种情况下，按照字节长度自动截断，自动截断后会在尾部产生非预期结果。如果用户有对于截断结果正确性的要求，建议用户采用UTF8等能够按照字符截断的输入字符集作为数据库的编码集。

语法格式

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
INSERT [/*+ plan_hint */] INTO table_name [partition_clause] [ AS alias ] [ ( column_name [, ...] ) ]
    { DEFAULT VALUES
    | VALUES {( { expression | DEFAULT } [, ...] ) }[, ...]
    | query }
    [ ON DUPLICATE KEY UPDATE { NOTHING | { column_name = { expression | DEFAULT } } [, ...] [ WHERE condition ] }]
    [ RETURNING {* | {output_expression [ [ AS ] output_name ] }[, ...]} ];
```

参数说明

- **WITH [RECURSIVE] with_query [, ...]**

  用于声明一个或多个可以在主查询中通过名称引用的子查询，相当于临时表。

  如果声明了RECURSIVE，那么允许SELECT子查询通过名称引用它自己。

  其中with_query的详细格式为：

  ```
  with_query_name [ ( column_name [, ...] ) ] AS [ [ NOT ] MATERIALIZED ]
  ( {SELECT | VALUES | INSERT | UPDATE | DELETE} )
  ```

  – with_query_name指定子查询生成的结果集名称，在查询中可使用该名称访问子查询的结果集。

  – column_name指定子查询结果集中显示的列名。

  – 每个子查询可以是SELECT，VALUES，INSERT，UPDATE或DELETE语句。

  – 用户可以使用MATERIALIZED / NOT MATERIALIZED对CTE进行修饰。

  - 如果声明为MATERIALIZED，WITH查询将被物化，生成一个子查询结果集的拷贝，在引用处直接查询该拷贝，因此WITH子查询无法和主干SELECT语句进行联合优化（如谓词下推、等价类传递等），对于此类场景可以使用NOT MATERIALIZED进行修饰，如果WITH查询语义上可以作为子查询内联执行，则可以进行上述优化。
  - 如果用户没有显示声明物化属性则遵守以下规则：如果CTE只在所属主干语句中被引用一次，且语义上支持内联执行，则会被改写为子查询内联执行，否则以CTE Scan的方式物化执行。

  > **说明**
  >
  > - INSERT ON DUPLICATE KEY UPDATE不支持WITH及WITH RECURSIVE子句。
  > - INSERT语句的输出，只会显示最外层query block插入的tuple数量。比如：
  >
  >   ```
  >   with cte as (insert into t1 values(1) retunring * ) insert into t1 select * from cte;
  >   ```
  >
  >   只会显示插入一条tuple，但实际上插入了两条。

- **plan_hint子句**

  以/*+ */的形式在INSERT关键字后，用于对INSERT对应的语句块生成的计划进行hint调优，详细用法请参见章节**使用Plan Hint进行调优**。每条语句中只有第一个/*+ plan_hint */注释块会作为hint生效，里面可以写多条hint。

- **table_name**

  要插入数据的目标表名。

取值范围：已存在的表名。

说明

支持使用DATABASE LINK方式对远端表进行操作，使用方式详情请见**DATABASE LINK**。

- **partition_clause**

  指定分区插入操作

  ```
  PARTITION { ( partition_name ) | FOR ( partition_value [, ...] ) } |
  SUBPARTITION { ( subpartition_name ) | FOR ( subpartition_value [, ...] ) }
  ```

  关键字详见**SELECT**一节介绍

  如果value子句的值和指定分区不一致，会抛出异常。

  示例详见**CREATE TABLE SUBPARTITION**

- **column_name**

  目标表中的字段名：

  - 字段名可以使用子字段名或者数组下标修饰。
  - 没有在字段列表中出现的每个字段，将由系统默认值，或者声明时的默认值填充，若都没有则用NULL填充。例如，向一个复合类型中的某些字段插入数据的话，其他字段将是NULL。
  - 目标字段（column_name）可以按顺序排列。如果没有列出任何字段，则默认全部字段，且顺序为表声明时的顺序。
  - 如果value子句和query中只提供了N个字段，则目标字段为前N个字段。
  - value子句和query提供的值在表中从左到右关联到对应列。

  取值范围：已存在的字段名。

- **expression**

  赋予对应column的一个有效表达式或值：

  - 如果是INSERT ON DUPLICATE KEY UPDATE语句下，expression可以为VALUES(column_name)或EXCLUDED.column_name用来表示引用冲突行对应的column_name字段的值。需注意，其中VALUES(column_name)不支持嵌套在表达式中（例如VALUES(column_name)+1），但EXCLUDED不受此限制。
  - 向表中字段插入单引号 " ' "时需要使用单引号自身进行转义。
  - 如果插入行的表达式不是正确的数据类型，系统试图进行类型转换，若转换不成功，则插入数据失败，系统返回错误信息。

- **DEFAULT**

  对应字段名的缺省值。如果没有缺省值，则为NULL。

- **query**

  一个查询语句（SELECT语句），将查询结果作为插入的数据。

- **RETURNING**

  返回实际插入的行，RETURNING列表的语法与SELECT的输出列表一致。注意：INSERT ON DUPLICATE KEY UPDATE不支持RETURNING子句。

- **output_expression**

  INSERT命令在每一行都被插入之后用于计算输出结果的表达式。

  取值范围：该表达式可以使用table的任意字段。可以使用*返回被插入行的所有字段。

- **output_name**

  字段的输出名称。

  取值范围：字符串，符合**标识符命名规范**。

- **ON DUPLICATE KEY UPDATE**

  对于带有唯一约束（UNIQUE INDEX或PRIMARY KEY）的表，如果插入数据违反唯一约束，则对冲突行执行UPDATE子句完成更新，对于不带唯一约束的表，则仅执行插入。UPDATE时，若指定NOTHING则忽略此条插入，可通过"EXCLUDE." 或者 "VALUES()" 来选择源数据相应的列。

  - 支持触发器，触发器执行顺序由实际执行流程决定：
    - 执行insert： 触发 before insert、 after insert触发器。
    - 执行update：触发before insert、before update、after update触发器。
    - 执行update nothing： 触发before insert触发器。
  - 不支持延迟生效（DEFERRABLE）的唯一约束或主键。
  - 如果表中存在多个唯一约束，如果所插入数据违反多个唯一约束，对于检测到冲突的第一行进行更新，其他冲突行不更新（检查顺序与索引维护具有强相关性，一般先创建的索引先进行冲突检查）。
  - 如果插入多行，这些行均与表中同一行数据存在唯一约束冲突，则按照顺序，第一条执行插入或更新，之后依次执行更新。
  - 主键、唯一索引列不允许UPDATE。
  - 不支持外表。
  - expression不支持使用子查询表达式。

示例

```
--创建SCHEMA。
gaussdb=# CREATE SCHEMA tpcds;

--创建表tpcds.reason。
gaussdb=# CREATE TABLE tpcds.reason
(
  r_reason_sk    integer,
  r_reason_id    character(16),
  r_reason_desc  character(100)
);

--向表中插入一条记录。
gaussdb=# INSERT INTO tpcds.reason(r_reason_sk, r_reason_id, r_reason_desc) VALUES (0,
'AAAAAAAAAAAAAAAA', 'reason0');

--创建表tpcds.reason_t2。
gaussdb=# CREATE TABLE tpcds.reason_t2
(
  r_reason_sk    integer,
  r_reason_id    character(16),
  r_reason_desc  character(100)
);

--向表中插入一条记录。
gaussdb=# INSERT INTO tpcds.reason_t2(r_reason_sk, r_reason_id, r_reason_desc) VALUES (1,
'AAAAAAAABAAAAAAA', 'reason1');

--向表中插入一条记录，和上一条语法等效。
gaussdb=# INSERT INTO tpcds.reason_t2 VALUES (2, 'AAAAAAAABAAAAAAA', 'reason2');
```



```
--向表中插入多条记录。
gaussdb=# INSERT INTO tpcds.reason_t2 VALUES (3, 'AAAAAAAACAAAAAAA','reason3'),(4,
'AAAAAAAADAAAAAAA', 'reason4'),(5, 'AAAAAAAAEAAAAAAA','reason5');

--向表中插入tpcds.reason中r_reason_sk小于5的记录。
gaussdb=# INSERT INTO tpcds.reason_t2 SELECT * FROM tpcds.reason WHERE r_reason_sk <5;

--对表创建唯一索引。
gaussdb=# CREATE UNIQUE INDEX reason_t2_u_index ON tpcds.reason_t2(r_reason_sk);

--向表中插入多条记录，如果冲突则更新冲突数据行中r_reason_id字段为'BBBBBBBBCAAAAAAA'。
gaussdb=# INSERT INTO tpcds.reason_t2 VALUES (5, 'BBBBBBBBCAAAAAAA','reason5'),(6,
'AAAAAAAADAAAAAAA', 'reason6') ON DUPLICATE KEY UPDATE r_reason_id = 'BBBBBBBBCAAAAAAA';

--删除表tpcds.reason_t2。
gaussdb=# DROP TABLE tpcds.reason_t2;

--删除表tpcds.reason。
gaussdb=# DROP TABLE tpcds.reason;

--删除SCHEMA。
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

优化建议

- VALUES

通过INSERT语句批量插入数据时，建议将多条记录合并入一条语句中执行插入，以提高数据加载性能。例如，INSERT INTO sections VALUES (30, 'Administration', 31, 1900),(40, 'Development', 35, 2000), (50, 'Development' , 60 , 2001);

7.14.151 LOCK

功能描述

LOCK TABLE获取表级锁。

GaussDB在为一个引用了表的命令自动请求锁时，尽可能选择最小限制的锁模式。如果用户需要一种更为严格的锁模式，可以使用LOCK命令。例如，一个应用是在Read Committed隔离级别上运行事务，并且它需要保证表中的数据在事务的运行过程中不被修改。为实现这个目的，则可以在查询之前对表使用SHARE锁模式进行锁定。这样将防止数据不被并发修改，从而保证后续的查询可以读到已提交的持久化的数据。因为SHARE锁模式与任何写操作需要的ROW EXCLUSIVE模式冲突，并且LOCK TABLE name IN SHARE MODE语句将等到所有当前持有ROW EXCLUSIVE模式锁的事务提交或回滚后才能执行。因此，一旦获得该锁，就不会存在未提交的写操作，并且其他操作也只能等到该锁释放之后才能开始。

注意事项

- LOCK TABLE只能在一个事务块的内部有用，因为锁在事务结束时就会被释放。出现在任意事务块外面的LOCK TABLE都会报错。
- 如果没有声明锁模式，缺省为最严格的模式ACCESS EXCLUSIVE。
- LOCK TABLE ... IN ACCESS SHARE MODE需要在目标表上有SELECT权限。所有其他形式的LOCK需要UPDATE或DELETE权限。
- 没有UNLOCK TABLE命令，锁总是在事务结束时释放。
- LOCK TABLE只处理表级的锁，因此那些带“ROW”字样的锁模式都是有歧义的。这些模式名称通常可理解为用户试图在一个被锁定的表中获取行级的锁。同

样，ROW EXCLUSIVE模式也是一个可共享的表级锁。注意，只要是涉及到LOCK TABLE，所有锁模式都有相同的语意，区别仅在于规则中锁与锁之间是否冲突，规则请参见**表7-150**。

- 如果没有打开xc_maintenance_mode参数，那么对系统表申请ACCESS EXCLUSIVE级别锁将报错。

语法格式

```
LOCK [ TABLE ] {[ ONLY ] name [, ...]| {name [ * ]} [, ...]}
    [ IN {ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE | SHARE | SHARE ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE} MODE ]
    [ NOWAIT ];
```

参数说明

表 7-150 冲突的锁模式

请求的锁模式/当前锁模式	ACCESS SHARE	ROW SHARE	ROW EXCLUSIVE	SHARE UPDATE EXCLUSIVE	SHARE	SHARE ROW EXCLUSIVE	EXCLUSIVE	ACCESS EXCLUSIVE
ACCESS SHARE	-	-	-	-	-	-	-	X
ROW SHARE	-	-	-	-	-	-	X	X
ROW EXCLUSIVE	-	-	-	-	X	X	X	X
SHARE UPDATE EXCLUSIVE	-	-	-	X	X	X	X	X
SHARE	-	-	X	X	-	X	X	X
SHARE ROW EXCLUSIVE	-	-	X	X	X	X	X	X
EXCLUSIVE	-	X	X	X	X	X	X	X
ACCESS EXCLUSIVE	X	X	X	X	X	X	X	X

LOCK的参数说明如下所示：

- **name**

  要锁定的表的名称，可以有模式修饰。

  LOCK TABLE命令中声明的表的顺序就是上锁的顺序。

  取值范围：已存在的表名。

  说明

  支持使用DATABASE LINK方式对远端表进行操作，使用方式详情请见**DATABASE LINK**。

- **ONLY**

  如果指定ONLY，只有该表被锁定。如果没有声明，该表和他的所有子表将都被锁定。

- **ACCESS SHARE**

  只与ACCESS EXCLUSIVE冲突。

  SELECT命令在被引用的表上请求一个这种锁。通常， 任何只读取表而不修改它的命令都请求这种锁模式。

- **ROW SHARE**

  与EXCLUSIVE和ACCESS EXCLUSIVE锁模式冲突。

  SELECT FOR UPDATE和SELECT FOR SHARE命令会自动在目标表上请求ROW SHARE锁（且所有被引用但不是FOR SHARE/FOR UPDATE的其他表上，还会自动加上ACCESS SHARE锁）。

- **ROW EXCLUSIVE**

  与ROW SHARE锁相同，ROW EXCLUSIVE允许并发读取表，但是禁止修改表中数据。UPDATE，DELETE，INSERT命令会自动在目标表上请求这个锁（且所有被引用的其他表上还会自动加上的ACCESS SHARE锁）。通常情况下，所有会修改表数据的命令都会请求表的ROW EXCLUSIVE锁。

- **SHARE UPDATE EXCLUSIVE**

  这个模式保护一个表的模式不被并发修改，以及禁止在目标表上执行垃圾回收命令（VACUUM ）。

  VACUUM（不带FULL选项），ANALYZE，CREATE INDEX CONCURRENTLY命令会自动请求这样的锁。

- **SHARE**

  SHARE锁允许并发的查询，但是禁止对表进行修改。

  CREATE INDEX（不带CONCURRENTLY选项）语句会自动请求这种锁。

- **SHARE ROW EXCLUSIVE**

  SHARE ROW EXCLUSIVE锁禁止对表进行任何的并发修改，而且是独占锁，因此一个会话中只能获取一次。

  任何SQL语句都不会自动请求这个锁模式。

- **EXCLUSIVE**

  EXCLUSIVE锁允许对目标表进行并发查询，但是禁止任何其他操作。

  这个模式只允许并发加ACCESS SHARE锁，也就是说，只有对表的读动作可以和持有这个锁模式的事务并发执行。

  任何SQL语句都不会在用户表上自动请求这个锁模式。然而在某些操作的时候，会在某些系统表上请求它。

- **ACCESS EXCLUSIVE**

  这个模式保证其所有者（事务）是可以访问该表的唯一事务。

  ALTER TABLE，DROP TABLE，TRUNCATE，REINDEX命令会自动请求这种锁。

  在LOCK TABLE命令没有明确声明需要的锁模式时，它是缺省锁模式。

- **NOWAIT**

  声明LOCK TABLE不去等待任何冲突的锁释放，如果无法立即获取该锁，该命令退出并且发出一个错误信息。

  在不指定NOWAIT的情况下获取表级锁时，如果有其他互斥锁存在的话，则等待其他锁的释放。

示例

```
--创建SCHEMA。
gaussdb=# CREATE SCHEMA tpcds;

--创建表tpcds.reason。
gaussdb=# CREATE TABLE tpcds.reason
(
r_reason_sk         INTEGER      NOT NULL,
r_reason_id         CHAR(16)     NOT NULL,
r_reason_desc       INTEGER
);

--向表中插入多条记录。
gaussdb=# INSERT INTO tpcds.reason VALUES (1, 'AAAAAAAABAAAAAAA', '18'),(5, 'AAAAAAAACAAAAAAA',
'362'),(7, 'AAAAAAAADAAAAAAA', '585');

--在执行删除操作时对一个有主键的表进行 SHARE ROW EXCLUSIVE 锁。
gaussdb=# CREATE TABLE tpcds.reason_t1 AS TABLE tpcds.reason;

gaussdb=# START TRANSACTION;

gaussdb=# LOCK TABLE tpcds.reason_t1 IN SHARE ROW EXCLUSIVE MODE;

gaussdb=# DELETE FROM tpcds.reason_t1 WHERE r_reason_desc IN(SELECT r_reason_desc FROM
tpcds.reason_t1 WHERE r_reason_sk < 6 );

gaussdb=# DELETE FROM tpcds.reason_t1 WHERE r_reason_sk = 7;

gaussdb=# COMMIT;

--删除表tpcds.reason_t1。
gaussdb=# DROP TABLE tpcds.reason_t1;

--删除表。
gaussdb=# DROP TABLE tpcds.reason;

--删除SCHEMA。
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

7.14.152 MERGE INTO

功能描述

通过MERGE INTO语句，将目标表和源表中数据针对关联条件进行匹配，若关联条件匹配时对目标表进行UPDATE，无法匹配时对目标表执行INSERT。此语法可以很方便地用来合并执行UPDATE和INSERT，避免多次执行。

注意事项

进行MERGE INTO操作的用户需要同时拥有目标表的UPDATE和INSERT权限，以及源表的SELECT权限。

语法格式

```
MERGE [/*+ plan_hint */] INTO table_name [ partition_clause ] [ [ AS ] alias ]
USING { { table_name | view_name } | subquery } [ [ AS ] alias ]
ON ( condition )
[
  WHEN MATCHED THEN
  UPDATE SET { column_name = { expression | subquery | DEFAULT } |
         ( column_name [, ...] ) = ( { expression | subquery | DEFAULT } [, ...] ) } [, ...]
  [ WHERE condition ]
]
[
  WHEN NOT MATCHED THEN
  INSERT { DEFAULT VALUES |
  [ ( column_name [, ...] ) ] VALUES ( { expression | subquery | DEFAULT } [, ...] ) [, ...] [ WHERE condition ] }
];
NOTICE: 'subquery' in the UPDATE and INSERT clauses are only available in CENTRALIZED mode!
```

参数说明

- **plan_hint子句**

  以/*+ */的形式在MERGE关键字后，用于对MERGE对应的语句块生成的计划进行hint调优，详细用法请参见章节**使用Plan Hint进行调优**。每条语句中只有第一个/*+ plan_hint */注释块会作为hint生效，里面可以写多条hint。

- **INTO子句**

  指定正在更新或插入的目标表。

- **table_name**

  目标表的表名。

- **partition_clause**

  指定分区MERGE操作：

  ```
  PARTITION { ( partition_name ) | FOR ( partition_value [, ...] ) } |
  SUBPARTITION { ( subpartition_name ) | FOR ( subpartition_value [, ...] ) }
  ```

  关键字详见**SELECT**一节介绍。

  如果value子句的值和指定分区不一致，会抛出异常。

  示例详见**CREATE TABLE SUBPARTITION**。

- **alias**

  目标表的别名。

  取值范围：字符串，符合**标识符命名规范**。

- **USING子句**

  指定源表，源表可以为表、视图或子查询。

- **ON子句**

  关联条件，用于指定目标表和源表的关联条件。不支持更新关联条件中的字段。

- **WHEN MATCHED子句**

  当源表和目标表中数据针对关联条件可以匹配上时，选择WHEN MATCHED子句进行UPDATE操作。

  不支持更新系统表、系统列。

- **WHEN NOT MATCHED子句**

  当源表和目标表中数据针对关联条件无法匹配时，选择WHEN NOT MATCHED子句进行INSERT操作。

  不支持INSERT子句中包含多个VALUES。

  WHEN MATCHED和WHEN NOT MATCHED子句顺序可以交换，可以缺省其中一个，但不能同时缺省，不支持同时指定两个WHEN MATCHED或WHEN NOT MATCHED子句。

- **DEFAULT**

  用对应字段的缺省值填充该字段。

  如果没有缺省值，则为NULL。

- **WHERE condition**

  UPDATE子句和INSERT子句的条件，只有在条件满足时才进行更新操作，可缺省。不支持WHERE条件中引用系统列。不建议使用int等数值类型作为condition，因为int等数值类型可以隐式转换为bool值（非0值隐式转换为true，0转换为false），可能导致非预期的结果。

示例

```
-- 创建目标表products和源表newproducts，并插入数据
gaussdb=# CREATE TABLE products
(
product_id INTEGER,
product_name VARCHAR2(60),
category VARCHAR2(60)
);

gaussdb=# INSERT INTO products VALUES (1501, 'vivitar 35mm', 'electrncs');
gaussdb=# INSERT INTO products VALUES (1502, 'olympus is50', 'electrncs');
gaussdb=# INSERT INTO products VALUES (1600, 'play gym', 'toys');
gaussdb=# INSERT INTO products VALUES (1601, 'lamaze', 'toys');
gaussdb=# INSERT INTO products VALUES (1666, 'harry potter', 'dvd');

gaussdb=# CREATE TABLE newproducts
(
product_id INTEGER,
product_name VARCHAR2(60),
category VARCHAR2(60)
);

gaussdb=# INSERT INTO newproducts VALUES (1502, 'olympus camera', 'electrncs');
gaussdb=# INSERT INTO newproducts VALUES (1601, 'lamaze', 'toys');
gaussdb=# INSERT INTO newproducts VALUES (1666, 'harry potter', 'toys');
gaussdb=# INSERT INTO newproducts VALUES (1700, 'wait interface', 'books');

-- 进行MERGE INTO操作
gaussdb=# MERGE INTO products p
USING newproducts np
ON (p.product_id = np.product_id)
WHEN MATCHED THEN
  UPDATE SET p.product_name = np.product_name, p.category = np.category WHERE p.product_name !=
'play gym'
WHEN NOT MATCHED THEN
  INSERT VALUES (np.product_id, np.product_name, np.category) WHERE np.category = 'books';
MERGE 4

-- 查询更新后的结果
gaussdb=# SELECT * FROM products ORDER BY product_id;
 product_id |  product_name  | category
------------+----------------+-----------
       1501 | vivitar 35mm   | electrncs
       1502 | olympus camera | electrncs
```



```
     1600 | play gym       | toys
     1601 | lamaze         | toys
     1666 | harry potter   | toys
     1700 | wait interface | books
(6 rows)

-- 删除表
gaussdb=# DROP TABLE products;
gaussdb=# DROP TABLE newproducts;
```

7.14.153 MOVE

功能描述

MOVE在不检索数据的情况下重新定位一个游标。MOVE的作用类似于**FETCH**命令，但只是重定位游标而不返回行。

语法格式

```
MOVE [ direction [ FROM | IN ] ] cursor_name;
```

其中direction子句为可选参数。

```
NEXT
  | PRIOR
  | FIRST
  | LAST
  | ABSOLUTE count
  | RELATIVE count
  | count
  | ALL
  | FORWARD
  | FORWARD count
  | FORWARD ALL
  | BACKWARD
  | BACKWARD count
  | BACKWARD ALL
```

参数说明

MOVE命令的参数与FETCH的相同，详细请参见FETCH的**参数说明**。

说明

成功完成时，MOVE命令将返回一个“MOVE count”的标签，count是一个使用相同参数的FETCH命令会返回的行数（可能为零）。

示例

```
--创建SCHEMA。
gaussdb=# CREATE SCHEMA tpcds;

--创建表tpcds.reason。
gaussdb=# CREATE TABLE tpcds.reason
(
r_reason_sk         INTEGER      NOT NULL,
r_reason_id         CHAR(16)     NOT NULL,
r_reason_desc       VARCHAR(40)
);

--向表中插入多条记录。
gaussdb=# INSERT INTO tpcds.reason VALUES (1, 'AAAAAAAABAAAAAAA', 'Xxxxxxxxx'),(2,
'AAAAAAAACAAAAAAA', ' Xxxxxxxxx'),(3, 'AAAAAAAADAAAAAAA', ' Xxxxxxxxx'),(4, 'AAAAAAAAEAAAAAAA',
'Not the product that was ordered'),(5, 'AAAAAAAAFAAAAAAA', 'Parts missing'),(6, 'AAAAAAAAGAAAAAAA',
```



```
'Does not work with a product that I have'),(7, 'AAAAAAAAHAAAAAAA', 'Gift exchange');

--开始一个事务。
gaussdb=# START TRANSACTION;

--定义一个名为cursor1的游标。
gaussdb=# CURSOR cursor1 FOR SELECT * FROM tpcds.reason;

--忽略游标cursor1的前3行。
gaussdb=# MOVE FORWARD 3 FROM cursor1;

--抓取游标cursor1的前4行。
gaussdb=# FETCH 4 FROM cursor1;
 r_reason_sk |   r_reason_id    |                                r_reason_desc
-------------+------------------
+-------------------------------------------------------------------------------------------------------------
           4 | AAAAAAAAEAAAAAAA | Not the product that was
ordered
           5 | AAAAAAAAFAAAAAAA | Parts missing
           6 | AAAAAAAAGAAAAAAA | Does not work with a product that I
have
           7 | AAAAAAAAHAAAAAAA | Gift
exchange
(4 rows)

--关闭游标。
gaussdb=# CLOSE cursor1;

--结束一个事务。
gaussdb=# END;

--删除表。
gaussdb=# DROP TABLE tpcds.reason;

--删除SCHEMA。
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

相关链接

CLOSE，**FETCH**

7.14.154 PREDICT BY

功能描述

利用完成训练的模型进行推测任务。

注意事项

调用的模型名称在系统表gs_model_warehouse中可查看到。

语法格式

```
PREDICT BY model_name (FEATURES attribute [, attribute]...);
```

参数说明

- **model_name**
  用于推测任务的模型名称。
  取值范围：字符串，需要符合**标识符命名规范**。
- **attribute**

推测任务的输入特征列名。

取值范围：字符串，需要符合**标识符命名规范**。

示例

```
--创建数据表
gaussdb=# CREATE TABLE houses (
id INTEGER,
tax INTEGER,
bedroom INTEGER,
bath DOUBLE PRECISION,
price INTEGER,
size INTEGER,
lot INTEGER,
mark text
);

--插入训练数据
gaussdb=# INSERT INTO houses(id, tax, bedroom, bath, price, size, lot, mark) VALUES
(1,590,2,1,50000,770,22100,'a+'),
(2,1050,3,2,85000,1410,12000,'a+'),
(3,20,2,1,22500,1060,3500,'a-'),
(4,870,2,2,90000,1300,17500,'a+'),
(5,1320,3,2,133000,1500,30000,'a+'),
(6,1350,2,1,90500,850,25700,'a-'),
(7,2790,3,2.5,260000,2130,25000,'a+'),
(8,680,2,1,142500,1170,22000,'a-'),
(9,1840,3,2,160000,1500,19000,'a+'),
(10,3680,4,2,240000,2790,20000,'a-'),
(11,1660,3,1,87000,1030,17500,'a+'),
(12,1620,3,2,118500,1250,20000,'a-'),
(13,3100,3,2,140000,1760,38000,'a+'),
(14,2090,2,3,148000,1550,14000,'a-'),
(15,650,3,1.5,65000,1450,12000,'a-');

--训练模型
gaussdb=# CREATE MODEL price_model USING logistic_regression
 FEATURES size, lot
 TARGET mark
 FROM HOUSES
 WITH learning_rate=0.88, max_iterations=default;

——预测
gaussdb=# SELECT id, PREDICT BY price_model (FEATURES size,lot) FROM houses;

--删除模型
gaussdb=# DROP MODEL price_model;

--删除表
gaussdb=# DROP TABLE houses;
```

相关链接

CREATE MODEL，**DROP MODEL**

7.14.155 PREPARE

功能描述

创建一个预备语句。

预备语句是服务端的对象，可以用于优化性能。在执行PREPARE语句的时候，指定的查询被解析、分析、重写。当随后发出EXECUTE语句的时候，预备语句被规划和执行。这种设计避免了重复解析、分析工作。PREPARE语句创建后在整个数据库会话期

间一直存在，一旦创建成功，即便是在事务块中创建，事务回滚，PREPARE也不会删除。只能通过显式调用**DEALLOCATE**进行删除，会话结束时，PREPARE也会自动删除。

语法格式

```
PREPARE name [ ( data_type [, ...] ) ] AS statement;
```

参数说明

- **name**
  指定预备语句的名称。它必须在该会话中是唯一的。
- **data_type**
  参数的数据类型。
- **statement**
  SELECT INSERT、UPDATE、DELETE、MERGE INTO或VALUES语句之一。

示例

请参见EXECUTE的**示例**。

相关链接

DEALLOCATE

7.14.156 PREPARE TRANSACTION

功能描述

为当前事务做两阶段提交的准备。

在命令之后，事务就不再和当前会话关联了；它的状态完全保存在磁盘上，它被提交成功的可能性非常高，即使是在请求提交之前数据库发生了崩溃也如此。

一旦准备好了，一个事务就可以在稍后用**COMMIT PREPARED**或 **ROLLBACK PREPARED**命令分别进行提交或者回滚。这些命令可以从任何会话中发出，而不光是最初执行事务的那个会话。

从发出命令的会话的角度来看，PREPARE TRANSACTION不同于ROLLBACK：在执行它之后，就不再有活跃的当前事务了，并且预备事务的效果无法见到 (在事务提交的时候其效果会再次可见)。

如果PREPARE TRANSACTION因为某些原因失败，那么它就会变成一个ROLLBACK，当前事务被取消。

注意事项

- 事务功能由数据库自动维护，不应显式使用事务功能。
- 在运行PREPARE TRANSACTION命令时，必须在postgresql.conf配置文件中增大max_prepared_transactions的数值。建议至少将其设置为等于max_connections，这样每个会话都可以有一个等待中的预备事务。

语法格式

```
PREPARE TRANSACTION transaction_id;
```

参数说明

- **transaction_id**

  待提交事务的标识符，用于后面在COMMIT PREPARED或ROLLBACK PREPARED的时候标识这个事务。它不能和任何当前预备事务已经使用了的标识符同名。

  取值范围：标识符必须以字符串文本的方式书写，并且必须小于200字节长。

示例

```
--开始。
gaussdb=# BEGIN;
BEGIN

--准备标识符为的trans_test的事务。
gaussdb=# PREPARE TRANSACTION 'trans_test';
PREPARE TRANSACTION
```

相关链接

COMMIT PREPARED，**ROLLBACK PREPARED**

7.14.157 PURGE

功能描述

使用PURGE语句可以实现如下功能：

- 从回收站中清理表或索引，并释放对象相关的全部空间。
- 清理回收站。
- 清理回收站中指定表空间的对象。

注意事项

- 清除（PURGE）操作支持：表（PURGE TABLE）、索引（PURGE INDEX）、回收站（PURGE RECYCLEBIN）。
- 执行PURGE操作的权限要求如下：
  - PURGE TABLE：用户必须是表的所有者，且用户必须拥有表所在模式的USAGE权限，系统管理员默认拥有此权限。
  - PURGE INDEX：用户必须是索引的所有者，用户必须拥有索引所在模式的USAGE权限，系统管理员默认拥有此权限。
  - PURGE RECYCLEBIN：普通用户只能清理回收站中当前用户拥有的对象，且用户必须拥有对象所在模式的USAGE权限，系统管理员默认可以清理回收站所有对象。

前提条件

- 开启enable_recyclebin参数，启用回收站，参数使用请联系管理员处理。
- recyclebin_retention_time参数用于设置回收站对象保留时间，超过该时间的回收站对象将被自动清理，参数使用请联系管理员处理。

语法格式

```
PURGE { TABLE [schema_name.]table_name
      | INDEX index_name
      | RECYCLEBIN
    }
```

参数说明

- **schema_name**
  模式名。
- **TABLE [schema_name.] table_name**
  清空回收站中指定的表。
- **INDEX *index_name***
  清空回收站中指定的索引。
- **RECYCLEBIN**
  清空回收站中的所有对象。

示例

```
-- 创建角色tpcds。
gaussdb=#  CREATE ROLE tpcds IDENTIFIED BY '*********';

-- 创建表空间reason_table_space
gaussdb=#  CREATE TABLESPACE REASON_TABLE_SPACE1 owner tpcds RELATIVE location 'tablespace/
tsp_reason1';

-- 创建SCHEMA。
gaussdb=# CREATE SCHEMA tpcds;

-- 在表空间创建表tpcds.reason_t1
gaussdb=#  CREATE TABLE tpcds.reason_t1
 (
  r_reason_sk    integer,
  r_reason_id    character(16),
  r_reason_desc  character(100)
  ) tablespace reason_table_space1;
-- 在表空间创建表tpcds.reason_t2
gaussdb=#  CREATE TABLE tpcds.reason_t2
 (
  r_reason_sk    integer,
  r_reason_id    character(16),
  r_reason_desc  character(100)
  ) tablespace reason_table_space1;
-- 在表空间创建表tpcds.reason_t3
gaussdb=#  CREATE TABLE tpcds.reason_t3
 (
  r_reason_sk    integer,
  r_reason_id    character(16),
  r_reason_desc  character(100)
  ) tablespace reason_table_space1;
-- 对表tpcds.reason_t1创建索引
gaussdb=#  CREATE INDEX index_t1 on tpcds.reason_t1(r_reason_id);
gaussdb=#  DROP TABLE tpcds.reason_t1;
gaussdb=#  DROP TABLE tpcds.reason_t2;
gaussdb=#  DROP TABLE tpcds.reason_t3;
--查看回收站
gaussdb=#  SELECT rcyname,rcyoriginname,rcytablespace FROM GS_RECYCLEBIN;
        rcyname         | rcyoriginname | rcytablespace
------------------------+---------------+---------------
 BIN$16409$2CEE988==$0 | reason_t1     |         16408
 BIN$16412$2CF2188==$0 | reason_t2     |         16408
 BIN$16415$2CF2EC8==$0 | reason_t3     |         16408
```



```
 BIN$16418$2CF3EC8==$0 | index_t1     |            0
(4 rows)
--PURGE清除表
gaussdb=#  PURGE TABLE tpcds.reason_t3;
gaussdb=#  SELECT rcyname,rcyoriginname,rcytablespace FROM GS_RECYCLEBIN;
        rcyname        | rcyoriginname | rcytablespace
-----------------------+---------------+---------------
 BIN$16409$2CEE988==$0 | reason_t1     |         16408
 BIN$16412$2CF2188==$0 | reason_t2     |         16408
 BIN$16418$2CF3EC8==$0 | index_t1      |             0
(3 rows)
--PURGE清除索引
gaussdb=#  PURGE INDEX tpcds.index_t1;
gaussdb=#  SELECT rcyname,rcyoriginname,rcytablespace FROM GS_RECYCLEBIN;
        rcyname        | rcyoriginname | rcytablespace
-----------------------+---------------+---------------
 BIN$16409$2CEE988==$0 | reason_t1     |         16408
 BIN$16412$2CF2188==$0 | reason_t2     |         16408
(2 rows)
--PURGE清除回收站所有对象
gaussdb=#  PURGE recyclebin;
gaussdb=#  SELECT rcyname,rcyoriginname,rcytablespace FROM GS_RECYCLEBIN;
        rcyname        | rcyoriginname | rcytablespace
-----------------------+---------------+---------------
(0 rows)

-- 删除SCHEMA。
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

7.14.158 REASSIGN OWNED

功能描述

修改数据库对象的所有者。

REASSIGN OWNED用于将旧角色的数据库对象所有者改为新角色。

注意事项

- REASSIGN OWNED常用于在删除角色之前的准备工作。
- 执行REASSIGN OWNED需要有旧角色和目标角色上的权限。

语法格式

```
REASSIGN OWNED BY old_role [, ...] TO new_role;
```

参数说明

- **old_role**

  旧属主的角色名。

- **new_role**

  将要成为这些对象属主的新角色的名称。注意：只有初始用户才可以通过REASSIGN OWNED语法将属主修改为初始化用户。

示例

无。

7.14.159 REFRESH INCREMENTAL MATERIALIZED VIEW

功能描述

REFRESH INCREMENTAL MATERIALIZED VIEW会以增量刷新的方式对物化视图进行刷新。

注意事项

- 增量刷新仅支持增量物化视图。
- 刷新物化视图需要当前用户拥有基表的SELECT权限。

语法格式

```
REFRESH INCREMENTAL MATERIALIZED VIEW mv_name;
```

参数说明

- **mv_name**

  要刷新的物化视图的名称。

示例

```
--创建一个普通表。
gaussdb=# CREATE TABLE my_table (c1 int, c2 int)
WITH(STORAGE_TYPE=ASTORE);

--创建增量物化视图。
gaussdb=# CREATE INCREMENTAL MATERIALIZED VIEW my_imv AS SELECT * FROM my_table;

--基表写入数据。
gaussdb=# INSERT INTO my_table VALUES(1,1),(2,2);

--对增量物化视图my_imv进行增量刷新。
gaussdb=# REFRESH INCREMENTAL MATERIALIZED VIEW my_imv;

--删除增量物化视图。
gaussdb=# DROP MATERIALIZED VIEW my_imv;

--删除表my_table。
gaussdb=# DROP TABLE my_table;
```

相关链接

ALTER MATERIALIZED VIEW，**CREATE INCREMENTAL MATERIALIZED VIEW**，**CREATE MATERIALIZED VIEW**，**CREATE TABLE**， **DROP MATERIALIZED VIEW**，**REFRESH MATERIALIZED VIEW**

7.14.160 REFRESH MATERIALIZED VIEW

功能描述

REFRESH MATERIALIZED VIEW会以全量刷新的方式对物化视图进行刷新。

注意事项

- 全量刷新既可以对全量物化视图执行，也可以对增量物化视图执行。

- 刷新物化视图需要当前用户拥有基表的SELECT权限。

语法格式

```
REFRESH MATERIALIZED VIEW mv_name;
```

参数说明

- **mv_name**
  要刷新的物化视图的名称。

示例

```
--创建一个普通表。
gaussdb=# CREATE TABLE my_table (c1 int, c2 int)
WITH(STORAGE_TYPE=ASTORE);

--创建全量物化视图。
gaussdb=# CREATE MATERIALIZED VIEW my_mv AS SELECT * FROM my_table;

--创建增量物化视图。
gaussdb=# CREATE INCREMENTAL MATERIALIZED VIEW my_imv AS SELECT * FROM my_table;

--基表写入数据。
gaussdb=# INSERT INTO my_table VALUES(1,1),(2,2);

--对全量物化视图my_mv进行全量刷新。
gaussdb=# REFRESH MATERIALIZED VIEW my_mv;

--对增量物化视图my_imv进行全量刷新。
gaussdb=# REFRESH MATERIALIZED VIEW my_imv;

--删除增量物化视图。
gaussdb=# DROP MATERIALIZED VIEW my_imv;

--删除全量物化视图。
gaussdb=# DROP MATERIALIZED VIEW my_mv;

--删除表my_table。
gaussdb=# DROP TABLE my_table;
```

相关链接

ALTER MATERIALIZED VIEW，**CREATE INCREMENTAL MATERIALIZED VIEW**，**CREATE MATERIALIZED VIEW**，**CREATE TABLE**， **DROP MATERIALIZED VIEW**，**REFRESH INCREMENTAL MATERIALIZED VIEW**

7.14.161 REINDEX

功能描述

为表中的数据重建索引。

在以下几种情况下需要使用REINDEX重建索引：

- 索引崩溃，并且不再包含有效的数据。
- 索引变得“臃肿”，包含大量的空页或接近空页。
- 为索引更改了存储参数（例如填充因子），并且希望这个更改完全生效。
- 使用CONCURRENTLY选项创建索引失败，留下了一个“非法”索引。

注意事项

- REINDEX DATABASE和SYSTEM这种形式的重建索引不能在事务块中执行。目前不支持对物化视图进行REINDEX操作。
- REINDEX CONCURRENTLY在线重建索引导致表上索引顺序改变时，可能会导致查询计划改变。

语法格式

- 重建普通索引。
```
REINDEX { INDEX | TABLE | DATABASE | SYSTEM } [CONCURRENTLY] name [ FORCE ];
```
- 重建索引分区。
```
REINDEX  { INDEX|  TABLE} name
   PARTITION partition_name [ FORCE  ];
```

参数说明

- **INDEX**

  重新建立指定的索引。

- **TABLE**

  重新建立指定表的所有索引，如果表有从属的“TOAST”表，则这个表也会重建索引。如果表上有索引已经被alter unusable失效，则这个索引无法被重新创建。当指定CONCURRENTLY选项时，暂不支持重建从属“TOAST”表上的索引。

- **DATABASE**

  重建当前数据库里的所有索引。当指定CONCURRENTLY选项时，暂不支持重建数据库中表的从属“TOAST”表上的索引。

- **SYSTEM**

  在当前数据库上重建所有系统表上的索引。不会处理在用户表上的索引。

- **CONCURRENTLY**

  以不阻塞DML的方式重建索引（加ShareUpdateExclusiveLock锁）。重建索引时，一般会阻塞其他语句对该索引所依赖表的访问。指定此关键字，可以实现重建过程中不阻塞DML。不支持在线重建系统表上的索引。不支持REINDEX INTERNAL TABLE CONCURRENTLY和REINDEX SYSTEM CONCURRENTLY。当执行REINDEX DATABASE CONCURRENTLY时，在线重建当前数据库中用户表上的所有索引（不会处理系统表上的索引）。REINDEX CONCURRENTLY不可以在事务内执行。在线重建索引只支持B-tree索引和UB-tree索引，只支持普通索引、GLOBAL索引、LOCAL索引。在线并行重建索引只支持Astore的普通索引、GLOBAL索引、LOCAL索引，Ustore索引不支持在线并行重建。如果在线重建索引失败，可能会留下非法的新索引，在系统无法自动清理失败新索引的情况下（比如数据库宕机），需要尽快手动清除（使用DROP INDEX语句）非法新索引，以防占用更多资源。一般来说，非法的新索引的后缀名为_ccnew。REINDEX INDEX CONCURRENTLY对表加4级会话锁，且其前几个阶段与CREATE INDEX CONCURRENTLY相似，因此也可能产生卡住或死锁的问题，具体场景与CREATE INDEX CONCURRENTLY相似（比如两个会话同时对同一个索引或表进行REINDEX CONCURRENTLY操作，会引发死锁问题），详见**CONCURRENTLY**章节。

- **name**

  需要重建索引的索引、表、数据库的名称。表和索引可以有模式修饰。

说明

REINDEX DATABASE和SYSTEM只能重建当前数据库的索引，所以name必须和当前数据库名称相同。

- **FORCE**

  废弃选项，仅为保持前向兼容，故继续保留。

- **partition_name**

  需要重建索引的分区的名称或者索引分区的名称。

  取值范围：

  - 如果前面是REINDEX INDEX，则这里应该指定索引分区的名称；
  - 如果前面是REINDEX TABLE，则这里应该指定分区的名称；

须知

REINDEX DATABASE和SYSTEM这种形式的重建索引不能在事务块中执行。

示例

```
--创建SCHEMA。
gaussdb=# CREATE SCHEMA tpcds;

--创建表tpcds. customer。
gaussdb=# CREATE TABLE tpcds.customer
(
c_customer_sk         INTEGER        NOT NULL,
c_customer_id         CHAR(16)       NOT NULL
);

--向表中插入多条记录。
gaussdb=# INSERT INTO tpcds.customer VALUES (1, 'AAAAAAAABAAAAAAA'),(5, 'AAAAAAAACAAAAAAA'),
(10, 'AAAAAAAADAAAAAAA');

--创建一个行存表tpcds.customer_t1，并在tpcds.customer_t1表上的c_customer_sk字段创建索引。
gaussdb=# CREATE TABLE tpcds.customer_t1
(
    c_customer_sk             integer               not null,
    c_customer_id             char(16)               not null,
    c_current_cdemo_sk        integer                       ,
    c_current_hdemo_sk        integer                       ,
    c_current_addr_sk         integer                       ,
    c_first_shipto_date_sk    integer                       ,
    c_first_sales_date_sk     integer                       ,
    c_salutation              char(10)                      ,
    c_first_name              char(20)                      ,
    c_last_name               char(30)                      ,
    c_preferred_cust_flag     char(1)                       ,
    c_birth_day               integer                       ,
    c_birth_month             integer                       ,
    c_birth_year              integer                       ,
    c_birth_country           varchar(20)                   ,
    c_login                   char(13)                      ,
    c_email_address           char(50)                      ,
    c_last_review_date        char(10)
)
WITH (orientation = row);

gaussdb=# CREATE INDEX tpcds_customer_index1 ON tpcds.customer_t1 (c_customer_sk);

gaussdb=# INSERT INTO tpcds.customer_t1 SELECT * FROM tpcds.customer WHERE c_customer_sk < 10;
```

```
--重建一个单独索引。
gaussdb=# REINDEX INDEX tpcds.tpcds_customer_index1;

--在线重建一个单独索引。
gaussdb=# REINDEX INDEX CONCURRENTLY tpcds.tpcds_customer_index1;

--重建表tpcds.customer_t1上的所有索引。
gaussdb=# REINDEX TABLE tpcds.customer_t1;

--在线重建表tpcds.customer_t1上的所有索引。
gaussdb=# REINDEX TABLE CONCURRENTLY tpcds.customer_t1;

--删除tpcds.customer_t1表。
gaussdb=# DROP TABLE tpcds.customer_t1;

--删除表。
gaussdb=# DROP TABLE tpcds.customer;

--删除SCHEMA。
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

优化建议

- DATABASE
  不能在事务中reindex database。
- SYSTEM
  不能在事务中reindex系统表。

7.14.162 RELEASE SAVEPOINT

功能描述

RELEASE SAVEPOINT删除一个当前事务先前定义的保存点。

把一个保存点删除就令其无法作为回滚点使用，除此之外它没有其它用户可见的行为。它并不能撤销在保存点建立起来之后执行的命令的影响。要撤销那些命令可以使用ROLLBACK TO SAVEPOINT 。当不再需要的时候删除一个保存点可以令系统在事务结束之前提前回收一些资源。

RELEASE SAVEPOINT也删除所有在指定的保存点建立之后的所有保存点。

注意事项

- 不能RELEASE一个没有定义的保存点，语法上会报错。
- 如果事务在回滚状态，则不能释放保存点。
- 如果多个保存点拥有同样的名称，只有最近定义的那个才被释放。

语法格式

```
RELEASE [ SAVEPOINT ] savepoint_name;
```

参数说明

savepoint_name

要删除的保存点的名称

示例

```
--创建SCHEMA。
gaussdb=# CREATE SCHEMA tpcds;

--创建一个新表。
gaussdb=# CREATE TABLE tpcds.table1(a int);

--开启事务。
gaussdb=# START TRANSACTION;

--插入数据。
gaussdb=# INSERT INTO tpcds.table1 VALUES (3);

--建立保存点。
gaussdb=# SAVEPOINT my_savepoint;

--插入数据。
gaussdb=# INSERT INTO tpcds.table1 VALUES (4);

--删除保存点。
gaussdb=# RELEASE SAVEPOINT my_savepoint;

--提交事务。
gaussdb=# COMMIT;

--查询表的内容，会同时看到3和4。
gaussdb=# SELECT * FROM tpcds.table1;

--删除表。
gaussdb=# DROP TABLE tpcds.table1;

--删除SCHEMA。
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

相关链接

SAVEPOINT，**ROLLBACK TO SAVEPOINT**

7.14.163 RESET

功能描述

RESET将指定的运行时参数恢复为缺省值。这些参数的缺省值是指postgresql.conf配置文件中所描述的参数缺省值。

RESET命令与如下命令的作用相同：

```
SET configuration_parameter TO DEFAULT;
```

注意事项

RESET的事务性行为和SET相同，它的影响将会被事务回滚撤销。

语法格式

```
RESET {configuration_parameter | CURRENT_SCHEMA | TIME ZONE | TRANSACTION ISOLATION LEVEL |
SESSION AUTHORIZATION | ALL };
```

参数说明

- **configuration_parameter**
  运行时参数的名称。

取值范围：可以使用SHOW ALL命令查看运行时参数。

- **CURRENT_SCHEMA**
  当前模式
- **TIME ZONE**
  时区。
- **TRANSACTION ISOLATION LEVEL**
  事务的隔离级别。
- **SESSION AUTHORIZATION**
  当前会话的用户标识符。
- **ALL**
  所有运行时参数。

示例

```
--把timezone设为缺省值。
gaussdb=# RESET timezone;

--把所有参数设置为缺省值。
gaussdb=# RESET ALL;
```

相关链接

SET，**SHOW**

7.14.164 REVOKE

功能描述

REVOKE用于撤销一个或多个角色的权限。

注意事项

非对象所有者试图在对象上REVOKE权限，命令按照以下规则执行：

- 如果授权用户没有该对象上的权限，则命令立即失败。
- 如果授权用户有部分权限，则只撤销那些有授权选项的权限。
- 如果授权用户没有授权选项，REVOKE ALL PRIVILEGES形式将发出一个错误信息，而对于其他形式的命令而言，如果是命令中指定名称的权限没有相应的授权选项，该命令将发出一个警告。

语法格式

- 回收指定表或视图上权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | ALTER | DROP | COMMENT |
INDEX | VACUUM }[, ...]
    | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...]
        | ALL TABLES IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收表上指定字段权限。


```
REVOKE [ GRANT OPTION FOR ]
    { {{ SELECT | INSERT | UPDATE | REFERENCES | COMMENT } ( column_name [, ...] )}[, ...]
    | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
    ON [ TABLE ] table_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收指定序列上权限，LARGE字段属性可选，回收语句不区分序列是否为LARGE。

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | UPDATE | ALTER | DROP | COMMENT }[, ...]
    | ALL [ PRIVILEGES ] }
    ON { [ [ LARGE ] SEQUENCE ] sequence_name [, ...]
        | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收指定数据库上权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { CREATE | CONNECT | TEMPORARY | TEMP | ALTER | DROP | COMMENT } [, ...]
    | ALL [ PRIVILEGES ] }
    ON DATABASE database_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收指定域上权限。

```
REVOKE [ GRANT OPTION FOR ]
    { USAGE | ALL [ PRIVILEGES ] }
    ON DOMAIN domain_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收指定客户端加密主密钥上的权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | DROP } [, ...] | ALL [PRIVILEGES] }
    ON CLIENT_MASTER_KEYS client_master_keys_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收指定列加密密钥上的权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | DROP } [, ...] | ALL [PRIVILEGES]}
    ON COLUMN_ENCRYPTION_KEYS column_encryption_keys_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收指定目录上权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { READ | WRITE | ALTER | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON DIRECTORY directory_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收指定外部数据源上权限。

```
REVOKE [ GRANT OPTION FOR ]
   { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN DATA WRAPPER fdw_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收指定外部服务器上权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON FOREIGN SERVER server_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收指定函数上权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON { FUNCTION {function_name ( [ {[ argmode ] [ arg_name ] arg_type} [, ...] ] )} [, ...]
```



```
    | ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ];
```

- 回收指定存储过程上权限。

```
REVOKE [ GRANT OPTION FOR ]
  { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
  ON { PROCEDURE {proc_name ( [ {[ argmode ] [ arg_name ] arg_type} [, ...] ] )} [, ...]
    | ALL PROCEDURE IN SCHEMA schema_name [, ...] }
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ];
```

- 回收指定过程语言上权限。

```
REVOKE [ GRANT OPTION FOR ]
  { USAGE | ALL [ PRIVILEGES ] }
  ON LANGUAGE lang_name [, ...]
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ];
```

- 回收指定大对象上权限。

```
REVOKE [ GRANT OPTION FOR ]
  { { SELECT | UPDATE } [, ...] | ALL [ PRIVILEGES ] }
  ON LARGE OBJECT loid [, ...]
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ];
```

- 回收指定模式上权限。

```
REVOKE [ GRANT OPTION FOR ]
  { { CREATE | USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
  ON SCHEMA schema_name [, ...]
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ];
```

- 回收指定表空间上权限。

```
REVOKE [ GRANT OPTION FOR ]
  { { CREATE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
  ON TABLESPACE tablespace_name [, ...]
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ];
```

- 回收指定类型上权限。

```
REVOKE [ GRANT OPTION FOR ]
  { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
  ON TYPE type_name [, ...]
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ];
```

- 回收package对象的权限。

```
REVOKE [ GRANT OPTION FOR ]
  { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [PRIVILEGES] }
  ON PACKAGE package_name [, ...]
  FROM {[GROUP] role_name | PUBLIC} [, ...]
  [ CASCADE | RESTRICT ];
```

- 按角色回收角色上的权限。

```
REVOKE [ ADMIN OPTION FOR ]
  role_name [, ...] FROM role_name [, ...]
  [ CASCADE | RESTRICT ];
```

- 回收角色上的sysadmin权限。

```
REVOKE ALL { PRIVILEGES | PRIVILEGE } FROM role_name;
```

- 回收ANY权限。

```
REVOKE [ ADMIN OPTION FOR ]
 { CREATE ANY TABLE | ALTER ANY TABLE | DROP ANY TABLE | SELECT ANY TABLE | INSERT ANY
TABLE | UPDATE ANY TABLE |
 DELETE ANY TABLE | CREATE ANY SEQUENCE | CREATE ANY INDEX | CREATE ANY FUNCTION |
EXECUTE ANY FUNCTION |
 CREATE ANY PACKAGE | EXECUTE ANY PACKAGE | CREATE ANY TYPE | ALTER ANY TYPE | DROP ANY
TYPE | ALTER ANY SEQUENCE | DROP ANY SEQUENCE |
 SELECT ANY SEQUENCE | ALTER ANY INDEX | DROP ANY INDEX | CREATE ANY SYNONYM | DROP
ANY SYNONYM | CREATE ANY TRIGGER | ALTER ANY TRIGGER |  DROP ANY TRIGGER
```



```
} [, ...]
  FROM [ GROUP ] role_name [, ...];
```

- 回收DATABASE LINK对象权限。

```
REVOKE { CREATE | ALTER | DROP } [PUBLIC] DATABASE LINK FROM role_name;
```

说明

DATABASE LINK详细说明请见**DATABASE LINK**。

参数说明

关键字PUBLIC表示一个隐式定义的拥有所有角色的组。

权限类别和参数说明，请参见GRANT的**参数说明**。

任何特定角色拥有的特权包括直接授予该角色的特权、从该角色作为其成员的角色中得到的权限以及授予给PUBLIC的权限。因此，从PUBLIC收回SELECT特权并不一定会意味着所有角色都会失去在该对象上的SELECT特权，那些直接被授予的或者通过另一个角色被授予的角色仍然会拥有它。类似地，从一个用户收回SELECT后，如果PUBLIC仍有SELECT权限，该用户还是可以使用SELECT。

指定GRANT OPTION FOR时，只撤销对该权限授权的权力，而不撤销该权限本身。

如用户A拥有某个表的UPDATE权限，及WITH GRANT OPTION选项，同时A把这个权限赋予了用户B，则用户B持有的权限称为依赖性权限。当用户A持有的权限或者授权选项被撤销时，必须声明CASCADE，将所有依赖性权限都撤销。

一个用户只能撤销由它自己直接赋予的权限。例如，如果用户A被指定授权（WITH ADMIN OPTION）选项，且把一个权限赋予了用户B，然后用户B又赋予了用户C，则用户A不能直接将C的权限撤销。但是，用户A可以撤销用户B的授权选项，并且使用CASCADE。这样，用户C的权限就会自动被撤销。另外一个例子：如果A和B都赋予了C同样的权限，则A可以撤销他自己的授权选项，但是不能撤销B的，因此C仍然拥有该权限。

如果执行REVOKE的角色持有的权限是通过多层成员关系获得的，则具体是哪个包含的角色执行的该命令是不确定的。在这种场合下，最好的方法是使用SET ROLE成为特定角色，然后执行REVOKE，否则可能导致删除了不想删除的权限，或者是任何权限都没有删除。

示例

请参考GRANT的**示例**。

相关链接

GRANT

7.14.165 ROLLBACK

功能描述

回滚当前事务并取消当前事务中的所有更新。

在事务运行的过程中发生了某种故障，事务不能继续执行，系统将事务中对数据库的所有已完成的操作全部撤销，数据库状态回到事务开始时。

注意事项

如果不在一个事务内部发出ROLLBACK不会有问题，但是将抛出一个NOTICE信息。

语法格式

```
ROLLBACK [ WORK | TRANSACTION ];
```

参数说明

WORK | TRANSACTION

可选关键字。除了增加可读性，没有任何其他作用。

示例

```
--开启一个事务
gaussdb=# START TRANSACTION;

--取消所有更改
gaussdb=# ROLLBACK;
```

相关链接

COMMIT | END

7.14.166 ROLLBACK PREPARED

功能描述

取消一个先前为两阶段提交准备好的事务。

注意事项

- 该功能仅在维护模式（GUC参数xc_maintenance_mode为on时）下可用。该模式谨慎打开，一般供维护人员排查问题使用，一般用户不应使用该模式。
- 要想回滚一个预备事务，必须是最初发起事务的用户，或者是系统管理员。
- 事务功能由数据库自动维护，不应显式使用事务功能。

语法格式

```
ROLLBACK PREPARED transaction_id ;
```

参数说明

transaction_id

待提交事务的标识符。它不能和任何当前预备事务已经使用了的标识符同名。

示例

```
--开始。
gaussdb=# BEGIN;
BEGIN

--准备标识符为的trans_test的事务。
gaussdb=# PREPARE TRANSACTION 'trans_test';
```



```
PREPARE TRANSACTION

--取消标识符为的trans_test的事务。
gaussdb=# ROLLBACK PREPARED 'trans_test';
ROLLBACK PREPARED
```

相关链接

COMMIT PREPARED，**PREPARE TRANSACTION**。

7.14.167 ROLLBACK TO SAVEPOINT

功能描述

ROLLBACK TO SAVEPOINT用于回滚到一个保存点，隐含地删除所有在该保存点之后建立的保存点。

回滚所有指定保存点建立之后执行的命令。保存点仍然有效，并且需要时可以再次回滚到该点。

注意事项

- 不能回滚到一个未定义的保存点，语法上会报错。
- 在保存点方面，游标有一些非事务性的行为。任何在保存点里打开的游标都会在回滚掉这个保存点之后关闭。如果一个前面打开了的游标在保存点里面，并且游标被一个FETCH命令影响，而这个保存点稍后回滚了，那么这个游标的位置仍然在FETCH让它指向的位置（也就是FETCH不会被回滚）。关闭一个游标的行为也不会被回滚给撤销掉。如果一个游标的操作导致事务回滚，那么这个游标就会置于不可执行状态，所以，尽管一个事务可以用ROLLBACK TO SAVEPOINT重新恢复，但是游标不能再使用了。
- 使用ROLLBACK TO SAVEPOINT回滚到一个保存点。使用RELEASE SAVEPOINT删除一个保存点，但是保留该保存点建立后执行的命令的效果。

语法格式

```
ROLLBACK [ WORK | TRANSACTION ] TO [ SAVEPOINT ] savepoint_name;
```

参数说明

savepoint_name

回滚截至的保存点

示例

```
--撤销 my_savepoint 建立之后执行的命令的影响。
gaussdb=# START TRANSACTION;
gaussdb=# SAVEPOINT my_savepoint;
gaussdb=# ROLLBACK TO SAVEPOINT my_savepoint;
--游标位置不受保存点回滚的影响。
gaussdb=# DECLARE foo CURSOR FOR SELECT 1 UNION SELECT 2;
gaussdb=# SAVEPOINT foo;
gaussdb=# FETCH 1 FROM foo;
 ?column?
----------
        1
gaussdb=# ROLLBACK TO SAVEPOINT foo;
gaussdb=# FETCH 1 FROM foo;
```



```
 ?column?
----------
        2
gaussdb=# RELEASE SAVEPOINT my_savepoint;
gaussdb=# COMMIT;
```

相关链接

SAVEPOINT，**RELEASE SAVEPOINT**

7.14.168 SAVEPOINT

功能描述

SAVEPOINT用于在当前事务里建立一个新的保存点。

保存点是事务中的一个特殊记号，它允许将那些在它建立后执行的命令全部回滚，把事务的状态恢复到保存点所在的时刻。

注意事项

- 使用ROLLBACK TO SAVEPOINT回滚到一个保存点。使用RELEASE SAVEPOINT删除一个保存点，但是保留该保存点建立后执行的命令的效果。
- 保存点只能在一个事务块里面建立。在一个事务里面可以定义多个保存点。
- 由于节点故障或者通信故障引起的节点线程或进程退出导致的报错，以及由于COPY FROM操作中源数据与目标表的表结构不一致导致的报错，均不能正常回滚到保存点之前，而是整个事务回滚。
- SQL标准要求，使用SAVEPOINT建立一个同名保存点时，需要自动删除前面那个同名保存点。在GaussDB数据库里，将保留旧的保存点，但是在回滚或者释放的时候，只使用最近的那个。释放了新的保存点将导致旧的再次成为ROLLBACK TO SAVEPOINT和RELEASE SAVEPOINT可以访问的保存点。除此之外，SAVEPOINT是完全符合SQL标准的。

语法格式

```
SAVEPOINT savepoint_name;
```

参数说明

savepoint_name

新建保存点的名称。

须知

使用SAVEPOINT时，建议及时RELEASE SAVEPOINT，避免子事务的嵌套个数过大，建议嵌套个数不要超过10000，嵌套数过多时，可能引发当前事务性能裂化。

示例

```
--创建一个新表。
gaussdb=# CREATE TABLE table1(a int);
```



```
--开启事务。
gaussdb=# START TRANSACTION;

--插入数据。
gaussdb=# INSERT INTO table1 VALUES (1);

--建立保存点。
gaussdb=# SAVEPOINT my_savepoint;

--插入数据。
gaussdb=# INSERT INTO table1 VALUES (2);

--回滚保存点。
gaussdb=# ROLLBACK TO SAVEPOINT my_savepoint;

--插入数据。
gaussdb=# INSERT INTO table1 VALUES (3);

--提交事务。
gaussdb=# COMMIT;

--查询表的内容，会同时看到1和3,不能看到2，因为2被回滚。
gaussdb=# SELECT * FROM table1;

--删除表。
gaussdb=# DROP TABLE table1;

--创建一个新表。
gaussdb=# CREATE TABLE table2(a int);

--开启事务。
gaussdb=# START TRANSACTION;

--插入数据。
gaussdb=# INSERT INTO table2 VALUES (3);

--建立保存点。
gaussdb=# SAVEPOINT my_savepoint;

--插入数据。
gaussdb=# INSERT INTO table2 VALUES (4);

--回滚保存点。
gaussdb=# RELEASE SAVEPOINT my_savepoint;

--提交事务。
gaussdb=# COMMIT;

--查询表的内容，会同时看到3和4。
gaussdb=# SELECT * FROM table2;

--删除表。
gaussdb=# DROP TABLE table2;
```

相关链接

RELEASE SAVEPOINT，**ROLLBACK TO SAVEPOINT**

7.14.169 SELECT

功能描述

SELECT用于从表或视图中取出数据。

SELECT语句就像叠加在数据库表上的过滤器，利用SQL关键字从数据表中过滤出用户需要的数据。

注意事项

- 表的所有者、拥有表SELECT权限的用户或拥有SELECT ANY TABLE权限的用户，有权读取表或视图中数据，系统管理员默认拥有此权限。
- 必须对每个在SELECT命令中使用的字段有SELECT权限。
- 使用FOR UPDATE、FOR NO KEY UPDATE、FOR SHARE或FOR KEY SHARE要求用户除了SELECT权限外还需有UPDATE权限。

语法格式

- 查询数据

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT [/*+ plan_hint */] [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
{ * | {expression [ [ AS ] output_name ]} [, ...] }
[ into_option ]
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ [ START WITH condition ] CONNECT BY [NOCYCLE] condition [ ORDER SIBLINGS BY expression ] ]
[ GROUP BY grouping_element [, ...] ]
[ HAVING condition [, ...] ]
[ WINDOW {window_name AS ( window_definition )} [, ...] ]
[ { UNION | INTERSECT | EXCEPT | MINUS } [ ALL | DISTINCT ] select ]
[ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS { FIRST |
LAST } ]} [, ...] ]
[ LIMIT { [offset,] count | ALL } ]
[ OFFSET start [ ROW | ROWS ] ]
[ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
[ into_option ]
[ {FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE } [ OF table_name [, ...] ] [ NOWAIT | WAIT n ]}
[...] ]
[into_option];
```

说明

condition和expression中可以使用targetlist中表达式的别名。

- 只能同一层引用。
- 只能引用targetlist中的别名。
- 只能是后面的表达式引用前面的表达式。
- 不能包含volatile函数。
- 不能包含Window function函数。
- 不支持在join on条件中引用别名。
- targetlist中有多个要应用的别名则报错。

须知

缓存SELECT语句计划的场景下，WHERE IN候选子集不易过大，建议条件个数不要超过100，防止引发动态内存过高问题：

- WHERE IN 候选子集过大时，生成计划的内存占用会增大。
- 当拼接SQL构造的WHERE IN 子集不同，缓存计划的SQL模板无法复用。会生成大量不同的计划，且计划无法共享 ，占用大量内存。

- 其中子查询with_query为：

```
with_query_name [ ( column_name [, ...] ) ]
    AS [ [ NOT ] MATERIALIZED ] ( {select | values | insert | update | delete} )
```


- 其中into子句为：

```
into_option: {
        INTO var_name [, var_name] ...
    | INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
    | INTO DUMPFILE 'file_name'
}
export_options: {
[FIELDS
 [TERMINATED BY 'string']
 [[OPTIONALLY] ENCLOSED BY 'char']
 [ESCAPED BY 'char' ]
   ][LINES
 [STARTING BY 'string']
 [TERMINATED BY 'string']
   ]
}
```

- 其中指定查询源from_item为：

```
{[ ONLY ] table_name [ * ] [ partition_clause ] [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
[ TABLESAMPLE sampling_method ( argument [, ...] ) [ REPEATABLE ( seed ) ] ]
[ TIMECAPSULE {TIMESTAMP | CSN} expression ]
|( select ) [ AS ] alias [ ( column_alias [, ...] ) ]
|with_query_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
|function_name ( [ argument [, ...] ] ) [ AS ] alias [ ( column_alias [, ...] | column_definition [, ...] ) ]
|function_name ( [ argument [, ...] ] ) AS ( column_definition [, ...] )
|from_item unpivot_clause
|from_item pivot_clause
|from_item [ NATURAL ] join_type from_item [ ON join_condition | USING ( join_column [, ...] ) ]}
```

- 其中group子句为：

```
( )
| expression
| ( expression [, ...] )
| ROLLUP ( { expression | ( expression [, ...] ) } [, ...] )
| CUBE ( { expression | ( expression [, ...] ) } [, ...] )
| GROUPING SETS ( grouping_element [, ...] )
```

- 其中指定分区partition_clause为：

```
PARTITION { ( partition_name ) | FOR (  partition_value [, ...] ) } |
SUBPARTITION { ( subpartition_name ) | FOR (  subpartition_value [, ...] )}
```

  说明

  指定分区只适合分区表。

- 其中设置排序方式nlssort_expression_clause为：

```
NLSSORT ( column_name, ' NLS_SORT = { SCHINESE_PINYIN_M | generic_m_ci } ' )
```

- 简化版查询语法，功能相当于SELECT * FROM table_name。

```
TABLE { ONLY {(table_name)| table_name} | table_name [ * ]};
```

参数说明

- **WITH [RECURSIVE] with_query [, ...]**

  用于声明一个或多个可以在主查询中通过名称引用的子查询，相当于临时表。这种子查询语句结构称为CTE（Common Table Expression）结构，应用这种结构时，执行计划中将存在CTE SCAN的内容。

  如果声明了RECURSIVE，那么允许SELECT子查询通过名称引用它自己。

  其中with_query的详细格式为：with_query_name [(column_name [, ...])] AS [[NOT] MATERIALIZED] ({select | values | insert | update | delete})

  – with_query_name指定子查询生成的结果集名称，在查询中可使用该名称访问子查询的结果集。

  - column_name指定子查询结果集中显示的列名。
  - 每个子查询可以是SELECT，VALUES，INSERT，UPDATE或DELETE语句。
  - RECURSIVE只能出现在WITH后面，多个CTE的情况下，只需要在第一个CTE处声明RECURSIVE。
  - 用户可以使用MATERIALIZED / NOT MATERIALIZED对CTE进行修饰。
    - 如果声明为MATERIALIZED，WITH查询将被物化，生成一个子查询结果集的拷贝，在引用处直接查询该拷贝，因此WITH子查询无法和主干SELECT语句进行联合优化（如谓词下推、等价类传递等）。当使用NOT MATERIALIZED进行修饰时，如果WITH查询语义上可以作为子查询内联执行，则可以进行上述优化。
    - 如果用户没有显式声明物化属性则遵守以下规则：如果CTE只在所属SELECT主干中被引用一次，且语义上支持内联执行，则会被改写为子查询内联执行，否则以CTE Scan的方式物化执行。

- **plan_hint子句**

  以/*+ */的形式在SELECT关键字后，用于对SELECT对应的语句块生成的计划进行hint调优，详细用法请参见章节**使用Plan Hint进行调优**。每条语句中只有第一个/*+ plan_hint */注释块会作为hint生效，里面可以写多条hint。

- **ALL**

  声明返回所有符合条件的行，是默认行为，可以省略该关键字。

- **DISTINCT [ON (expression [, ...])]**

  从SELECT的结果集中删除所有重复的行，使结果集中的每行都是唯一的。

  ON（expression [, ...]）只保留那些在给出的表达式上运算出相同结果的行集合中的第一行。

  须知

  DISTINCT ON表达式是使用与ORDER BY相同的规则进行解释的。除非使用了ORDER BY来保证需要的行首先出现，否则，"第一行" 是不可预测的。

- **SELECT列表**

  指定查询表中列名，可以是部分列或者是全部（使用通配符*表示）。

  通过使用子句AS output_name可以为输出字段取个别名，这个别名通常用于输出字段的显示。支持关键字name、value和type作为列别名。

  列名可以用下面几种形式表达：
  - 手动输入列名，多个列之间用英文逗号（,）分隔。
  - 可以是FROM子句里面计算出来的字段。

- **INTO子句**

  将select出的结果输出到指定用户自定义变量或文件。
  - var_name

    用户自定义的变量名。详见**SET章节**中的var_name。
  - OUTFILE
    - CHARACTER SET 指定编码格式。

- FIELDS 指定每个字段的属性：
  TERMINATED 指定间隔符。
  [OPTIONALLY] ENCLOSED 指定引号符，指定OPTIONALLY时只对字符串数据类型起作用。
  ESCAPED 指定转义符。
- LINES 指定行属性：
  STARTING 指定行开头。
  TERMINATED 指定行结尾。

– DUMPFILE
  导出无间隔符，无换行符的单行数据到文件。
– file_name
  指定文件的绝对路径。

```
into_option三处位置：
--创建自定义变量
gaussdb=# CREATE DATABASE user_var dbcompatibility 'b';
gaussdb=# \c user_var
user_var=# SET b_format_behavior_compat_options = enable_set_variables;
user_var=# SET @my_var := 1;
--创建表t
user_var=# CREATE TABLE t (a int, b text);
user_var=# INSERT INTO t values(1,'a');
--在from子句之前。
user_var=# SELECT a INTO @my_var FROM t;
--在锁定子句之前。
user_var=# SELECT a FROM t INTO @my_var FOR UPDATE;
--在select语句结尾。
user_var=# SELECT a FROM t FOR UPDATE INTO @my_var;

导出到文件：
user_var=# SELECT * FROM t;
 a | b
---+---
 1 | a
(1 row)
--导出数据到outfile文件。
user_var=# SELECT * FROM t INTO OUTFILE '/tmp/t.txt' FIELDS TERMINATED BY '~' ENCLOSED BY 't' ESCAPED BY '^' LINES STARTING BY '$' TERMINATED BY '&\n';
--文件内容：$t1t~tat&，其中LINES STARTING BY($),FIELDS TERMINATED BY(~),ENCLOSED BY(t),LINES TERMINATED BY(&\n)。
--导出数据到dumpfile文件。
user_var=# SELECT * FROM t INTO DUMPFILE '/tmp/t.txt';
--文件内容：1a
--切换到初始数据库,请使用真实的初始数据库名称替换init_db。
user_var=# \c init_db;
--删除数据库
gaussdb=# DROP DATABASE user_var;
```

- **FROM子句**
  为SELECT声明一个或者多个源表。
  FROM子句涉及的元素如下所示。
  – table_name
    表名或视图名，名称前可加上模式名，如：schema_name.table_name。

    说明

    支持使用DATABASE LINK方式对远端表、同义词进行操作，使用方式详情请见**DATABASE LINK**。

- alias

  给表或复杂的表引用起一个临时的表别名，以便被其余的查询引用。

  别名用于缩写或者在自连接中消除歧义。如果提供了别名，它就会完全隐藏表的实际名称。

  须知

  当为JOIN产生的表joined_table指定别名时，如果joined_table被()包裹，即(joined_table)，非保留关键字UNPIVOT和PIVOT不允许作为别名使用。

- TABLESAMPLE sampling_method (argument [, ...]) [REPEATABLE (seed)]

  table_name之后的TABLESAMPLE子句表示应该用指定的sampling_method来检索表中行的子集。

  可选的REPEATABLE子句指定一个用于产生采样方法中随机数的*种子*数。种子值可以是任何非空常量值。如果查询时表没有被更改，指定相同种子和argument值的两个查询将会选择该表相同的采样。但是不同的种子值通常将会产生不同的采样。如果没有给出REPEATABLE，则会基于一个系统产生的种子为每一个查询选择一个新的随机采样。

- TIMECAPSULE { TIMESTAMP | CSN } expression

  查询指定CSN点或者指定时间点表的内容。

  目前不支持闪回查询的表：系统表、DFS表、全局临时表、本地临时表、UNLOGGED表、视图、序列表、hashbucket表、共享表、继承表。

  - TIMECAPSULE TIMESTAMP

    关键字，闪回查询的标识，根据date日期，闪回查找指定时间点的结果集。date日期必须是一个过去有效的时间戳。

  - expression

    常量、函数或SQL表达式。

  - TIMECAPSULE CSN

    关键字，闪回查询的标识，根据表的CSN闪回查询指定CSN点的结果集。其中CSN可从gs_txn_snapshot记录的snpcsn号查得。

    说明

    - 闪回查询不能跨越影响表结构或物理存储的语句，否则会报错。即闪回点和当前点之间，如果执行过修改表结构或影响物理存储的语句（TRUNCATE、DDL、DCL、VACUUM FULL），则闪回失败。执行过DDL的表进行闪回操作报错：“ERROR：The table definition of %s has been changed.”。涉及namespace、表名改变等操作的DDL执行闪回操作报错： "ERROR: recycle object %s desired does not exist;”。
    - 闪回查询不支持索引查询，闪回查询仅支持seqScan进行全表扫描。
    - 闪回点过旧时，因闪回版本被回收等导致无法获取旧版本会导致闪回失败，报错：Restore point too old。
    - 通过时间方式指定闪回点，闪回数据和实际时间点最多偏差为3秒。
    - 对表执行TRUNCATE之后，再进行闪回查询或者闪回表操作。通过时间点进行的闪回操作会报错：Snapshot too old。通过CSN进行的闪回操作会找不到数据，或者报错：Snapshot too old。

- column_alias

  列别名。

- PARTITION

  查询分区表的某个分区的数据。

- partition_name

  分区名。

- partition_value

  指定的分区键值。在创建分区表时，如果指定了多个分区键，可以通过PARTITION FOR子句指定的这一组分区键的值，唯一确定一个分区。

- SUBPARTITION

  查询分区表的某个二级分区的数据。

- subpartition_name

  二级分区名。

- subpartition_value

  指定的一级分区和二级分区键值。可以通过SUBPARTITION FOR子句指定的两个分区键的值，唯一确定一个二级分区。

- subquery

  FROM子句中可以出现子查询，创建一个临时表保存子查询的输出。

- with_query_name

  WITH子句同样可以作为FROM子句的源，可以通过WITH查询的名称对其进行引用。

- function_name

  函数名称。函数调用也可以出现在FROM子句中。

- join_type

  有5种类型，如下所示。

  - [INNER] JOIN

    一个JOIN子句组合两个FROM项。可使用圆括弧以决定嵌套的顺序。如果没有圆括弧，JOIN从左向右嵌套。

  - LEFT [OUTER] JOIN

    返回笛卡尔积中所有符合连接条件的行，再加上左表中通过连接条件没有匹配到右表行的那些行。这样，左边的行将扩展为生成表的全长，方法是在那些右表对应的字段位置填上NULL。请注意，只在计算匹配的时候，才使用JOIN子句的条件，外层的条件是在计算完毕之后施加的。

  - RIGHT [OUTER] JOIN

    返回所有内连接的结果行，加上每个不匹配的右边行（左边用NULL扩展）。

    这只是一个符号上的方便，因为总是可以把它转换成一个LEFT OUTER JOIN，只要把左边和右边的输入互换位置即可。

  - FULL [OUTER] JOIN

    返回所有内连接的结果行，加上每个不匹配的左边行（右边用NULL扩展），再加上每个不匹配的右边行（左边用NULL扩展）。

- CROSS JOIN

  CROSS JOIN等效于INNER JOIN ON（TRUE），即没有被条件删除的行。这种连接类型只是符号上的方便，因为它们与简单的FROM和WHERE的效果相同。

  说明

  必须为INNER和OUTER连接类型声明一个连接条件，即NATURAL ON，join_condition，USING (join_column [， ...]) 之一。但是它们不能出现在CROSS JOIN中。

其中CROSS JOIN和INNER JOIN生成一个简单的笛卡尔积，和在FROM的顶层列出两个项的结果相同。

– ON join_condition

  连接条件，用于限定连接中的哪些行是匹配的。如：ON left_table.a = right_table.a。不建议使用int等数值类型作为join_condition，因为int等数值类型可以隐式转换为bool值（非0值隐式转换为true，0转换为false），可能导致非预期的结果。

– USING(join_column[，...])

  ON left_table.a = right_table.a AND left_table.b = right_table.b ... 的简写。要求对应的列必须同名。

– NATURAL

  NATURAL是具有相同名称的两个表的所有列的USING列表的简写。

– from item

  用于连接的查询源对象的名称。

– unpivot_clause

  unpivot_clause可将列转置为行，其对应语法格式为：

```
UNPIVOT [ {INCLUDE | EXCLUDE} NULLS ]
  (
     unpivot_col_clause
     unpivot_for_clause
     unpivot_in_clause
  )
```

  - {INCLUDE | EXCLUDE} NULLS

    该子句用于控制转置后的结果是否包含存在NULL值的行，INCLUDE NULLS将使得结果包含存在NULL值的行，而EXCLUDE NULLS将从结果集中过滤掉这些行数据。如果忽略该子句，unpivot操作默认会从结果集中剔除存在NULL值的行。

  - unpivot_col_clause为：

```
unpivot_col_element
```

    unpivot_col_element指定了输出的列名，这些列会保存待转置列的列值。

  - unpivot_col_element为：

```
{ column | ( column [, column]... ) }
```

    unpivot_col_element有两种形式：column；(column [, column]...)。

  - unpivot_for_clause为：

```
FOR { unpivot_col_element }
```

    unpivot_for_clause中的unpivot_col_element用于指定输出的列名，这些列会保存待转置列的列名或别名。

- unpivot_in_clause为：
  ```
  IN ( unpivot_in_element [,unpivot_in_element...] )
  ```
  unpivot_in_clause指定了待转置列，这些列的列名和列值将保存在之前指定的输出列中。

  unpivot_in_element为：
  ```
  { unpivot_col_element }[  AS { unpivot_alias_element } ]
  ```
  unpivot_col_element为指定的待转置列，若采用(column [, column]...)形式指定待转置列，(column [, column]...)中所有的column列名将通过下划线 "_" 进行拼接，并保存在输出列中。如IN ((col1, col2)) 将会生成列名 "col1_col2"，并保存在unpivot_for_clause指定的输出列中。此外，AS关键字可为待转置列指定别名，一旦指定别名，输出列中将保存别名而不再保存待转置列的列名。

- unpivot_alias_element为：
  ```
  { alias | ( alias [, alias]... )}
  ```
  与unpivot_col_element类似，unpivot_alias_element也有两种形式。其中，alias为指定的别名。

须知

目前unpivot_clause存在如下约束：

- 仅支持在A兼容模式下使用。
- unpivot_clause子句内不支持与hint配合使用。
- 对于unpivot_col_clause，其unpivot_col_element指定的输出列数目需与unpivot_in_clause中unpivot_col_element的列数目相同。
- 对于unpivot_for_clause，其unpivot_col_element指定的输出列数目需与unpivot_in_clause中unpivot_alias_element的别名数目相同。
- 对于unpivot_in_clause，别名必须为常量，或者可以转换为常量的表达式。
- 对于unpivot_in_clause，常量表达式支持的函数只能是不可变（IMMUTABLE）函数。
- 对于unpivot_in_clause的所有unpivot_col_element而言，如果这些unpivot_col_element相同位置的column类型存在差异，则unpivot会尝试进行类型转换，以将这些转置列的列值转换为公共类型。类似地，对于所有unpivot_alias_element而言，如果这些unpivot_alias_element相同位置的alias类型存在差异，unpivot也会进行类似的类型转换。

  例如，假定存在"IN (col1, col2)"形式的unpivot_in_clause，其中col1为int类型，而col2为float类型，则unpivot在计算过程中会尝试将col1的列值转为公共类型float。

– pivot_clause

  pivot_clause可将行转置为列，其对应语法格式为：
  ```
  PIVOT [ XML ]
  ( aggregate_function ( expr ) [[AS] alias ]
      [, aggregate_function ( expr ) [[AS] alias ] ]...
      pivot_for_clause
      pivot_in_clause
  )
  ```

- aggregate_function（expr）[[AS] alias]

  aggregate_function针对给定的表达式进行聚合计算，计算结果将保存在pivot_in_clause指定的输出列中。[AS] alias（AS关键字可省略）可为aggregate_function指定别名，别名将以 "_别名" 格式附加在pivot_in_clause指定的输出列名后。

- pivot_for_clause为：

  ```
  FOR { column
    | ( column [, column]... )
    }
  ```

  pivot_for_clause指定了待转置行，column表示待转置行的某一列。

- pivot_in_clause为：

  ```
  IN ( { { { expr
         | ( expr [, expr]... )
         } [ [AS] alias]
      }...
    }
  )
  ```

  pivot_in_clause指定了输出结果的列名，列名可由一个expr或多个expr构成，例如，（expr1, expr2）。当列名由多个expr构成时，这些expr将按顺序通过下划线 "_" 进行连接，即（expr1, expr2）对应的输出列名为"expr1_expr2"。除了生成输出列名外，这些expr还决定着聚合函数触发时机，当待转置行的行值与这些expr的值相同时，pivot将进行聚合函数aggregate_function的计算，并将计算结果保存在列名由这些expr构成的输出列中。假定expr1为1，expr2为2，对于行"1 2"，pivot将进行aggregate_function的计算，对于行"1 1"，则不会触发计算。

须知

目前pivot_clause存在如下约束：

- 仅支持在A兼容模式下使用。
- pivot_clause子句内不支持与hint配合使用。
- 当指定多于一个aggregate_function时，最多允许一个aggregate_function没有别名，其余aggregate_function均需指定别名。
- XML只支持语法不支持功能。
- pivot_in_clause中的expr可以是常量，或者是可以转换为常量的表达式。若不是一元表达式，则需为expr指定别名。
- 对于pivot_in_clause中的expr，常量表达式支持的函数只能是不可变（IMMUTABLE）函数。
- 对于pivot_in_clause中的expr，当通过as为其指定别名时，非保留关键字可作为别名使用，否则不能。
- 当输出列的列名长度超过63时，后续的字符将不会被打印。

- **WHERE子句**

  WHERE子句构成一个行选择表达式，用来缩小SELECT查询的范围。condition是返回值为布尔型的任意表达式，任何不满足该条件的行都不会被检索。不建议使用int等数值类型作为condition，因为int等数值类型可以隐式转换为bool值（非0值隐式转换为true，0转换为false），可能导致非预期的结果。

WHERE子句中可以通过指定"(+)"操作符的方法将表的连接关系转换为外连接。但是不建议用户使用这种用法，因为这并不是SQL的标准语法，在做平台迁移的时候可能面临语法兼容性的问题。同时，使用"(+)"有很多限制：

a. "(+)"只能出现在where子句中。
b. 如果from子句中已经有指定表连接关系，那么不能再在where子句中使用"(+)"。
c. "(+)"只能作用在表或者视图的列上，不能作用在表达式上。
d. 如果表A和表B有多个连接条件，那么必须在所有的连接条件中指定"(+)"，否则"(+)"将不会生效，表连接会转化成内连接，并且不给出任何提示信息。
e. "(+)"作用的连接条件中的表不能跨查询或者子查询。如果"(+)"作用的表，不在当前查询或者子查询的from子句中，则会报错。如果"(+)"作用的对端的表不存在，则不报错，同时连接关系会转化为内连接。
f. "(+)"作用的表达式不能直接通过"OR"连接。
g. 如果"(+)"作用的列是和一个常量的比较关系， 那么这个表达式会成为JOIN条件的一部分。
h. 同一个表不能对应多个外表。
i. "(+)"只能出现"比较表达式"，"NOT表达式"，“ANY表达式”，“ALL表达式”，“IN表达式”，“NULLIF表达式”，“IS DISTINCT FROM表达式”，“IS OF”表达式。"(+)"不能出现在其他类型表达式中，并且这些表达式中不允许出现通过“AND”和“OR”连接的表达式。
j. "(+)"只能转化为左外连接或者右外连接，不能转化为全连接，即不能在一个表达式的两个表上同时指定"(+)"

须知

- 对于WHERE子句的LIKE操作符，当LIKE中要查询特殊字符“%”、“_”、“\”的时候需要使用反斜杠“\”来进行转义。
- 对于层次查询，在多表连接查询中，会对WHERE表达式做以下处理：

  将WHERE表达式按照析取、合取动作进行分解，查看每一个子表达式是否涉及当前层查询的多个表（潜在的连接条件），如果此子表达式不是子链接且仅涉及当前层查询的多个表，则将其下推至层次查询的非递归（START WITH）和递归（CONNECT BY）子句中，作为JOIN条件最先执行；否则，将其置于递归（CONNECT BY）子句结束之后，作为过滤条件执行。

  在内部实现时，是先把原WHERE表达式的语法树中的不满足下推的子表达式删除后得到最终下推的表达式，再把原WHERE表达式的parsetree中的满足下推的子表达式删除后得到最终不下推的表达式。

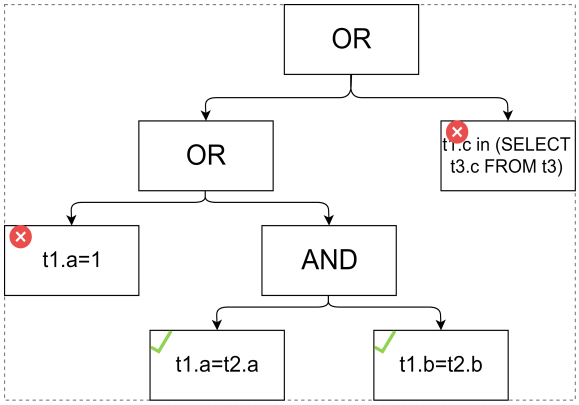
  综上所述，有以下几点需要额外强调：

  - 拆分WHERE表达式时，析取也会被拆分，因此OR连接的两个表达式也会被拆分。
  - 子链接一定不被下推。
  - 如果存在非当前层查询传入的列，则一定不被下推。

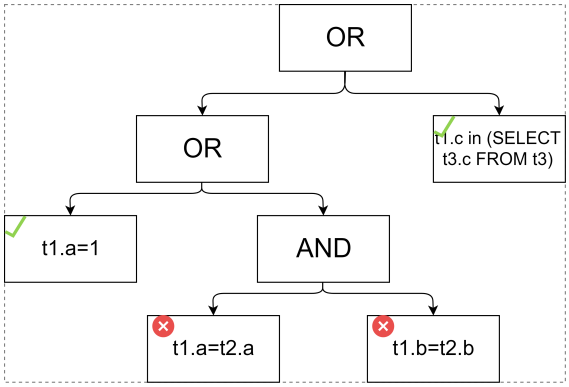
  那么，对于下面例子：

```
SELECT * FROM t1,t2
WHERE t1.a=1 or t1.a=t2.a and t1.b=t2.b or t1.c in (SELECT t3.c FROM t3)
START WITH t1.c=1
CONNECT BY PRIOR t2.c=t1.c;
```

  将WHERE表达式按照析取、合取拆开后，被下推的条件如下图所示：



  其等价于 "t1.a=t2.a AND t1.b=t2.b"，不被下推的条件如下图所示：



  其等价于"t1.a=1 OR t1.c in (SELECT t3.c FROM t3)"。

此外，对于另外一个例子：

```
SELECT * FROM t3 WHERE EXISTS(
SELECT * FROM t1,t2
WHERE t1.a+t2.a=t3.a
START WITH t1.c=1
CONNECT BY PRIOR t2.c=t1.c;
)
```

其中where条件存在外层查询传入的t3.a，因此该where条件不会被下推。

- **START WITH子句**

  START WITH子句通常与CONNECT BY子句同时出现，数据进行层次递归遍历查询，START WITH代表递归的初始条件。若省略该子句，单独使用CONNECT BY子句，则表示以表中的所有行作为初始集合。该功能详见**•CONNECT BY子句**。

- **CONNECT BY子句**

  CONNECT BY代表递归连接条件，和START WITH子句一起使用，实现数据遍历递归的功能。如：

```
gaussdb=#  CREATE TABLE test(name varchar, id int, fatherid int);
gaussdb=#  INSERT INTO test VALUES('A', 1, 0), ('B', 2, 1),('C',3,1),('D',4,1),('E',5,2);
gaussdb=#  SELECT * FROM test START WITH id = 1 CONNECT PRIOR id = fatherid ORDER SIBLINGS BY id DESC;
 name | id | fatherid
------+----+----------
 A    | 1 |        0
 D    | 4 |        1
 C    | 3 |        1
 B    | 2 |        1
 E    | 5 |        2
(5 rows)
```

  CONNECT BY条件中可以对列指定PRIOR关键字代表以这列为递归键进行递归。若在递归连接条件前加NOCYCLE，则表示遇到循环记录时停止递归（注意：含START WITH .. CONNECT BY子句的SELECT语句不支持使用FOR SHARE/UPDATE锁）。

  START WITH语句的执行流程是：

  a. 由START WITH 区域的条件选择初始的数据集。上述例子里，先把 ('A', 1, 0) 选择出来了。然后把初始的数据集设为工作集。
  b. 只要工作集不为空，会用工作集的数据作为输入，查询下一轮的数据，过滤条件由connect by区域指定。其中，PRIOR关键字表示当前记录，如上文例子中prior id = fatherid表示当前记录的id是下一条记录的fatherid。
  c. 把2中筛选出来的数据集，设为工作集，返回第二步重复操作。

  同时，数据库为每一条选出来的数据添加下述的伪列，方便用户了解数据在递归或者树状结构中的位置。

  - LEVEL：节点的层级，根节点层级为1。
  - CONNECT_BY_ISLEAF：是否为叶子节点。

须知

connect_by_isleaf：当某个节点不存在任何子节点时，其即为叶子节点，connect_by_isleaf会被置为1；否则会被置为0。

此处举例说明，假设存在表T1，其中数据如图7-12所示：

图 7-12 数据结构

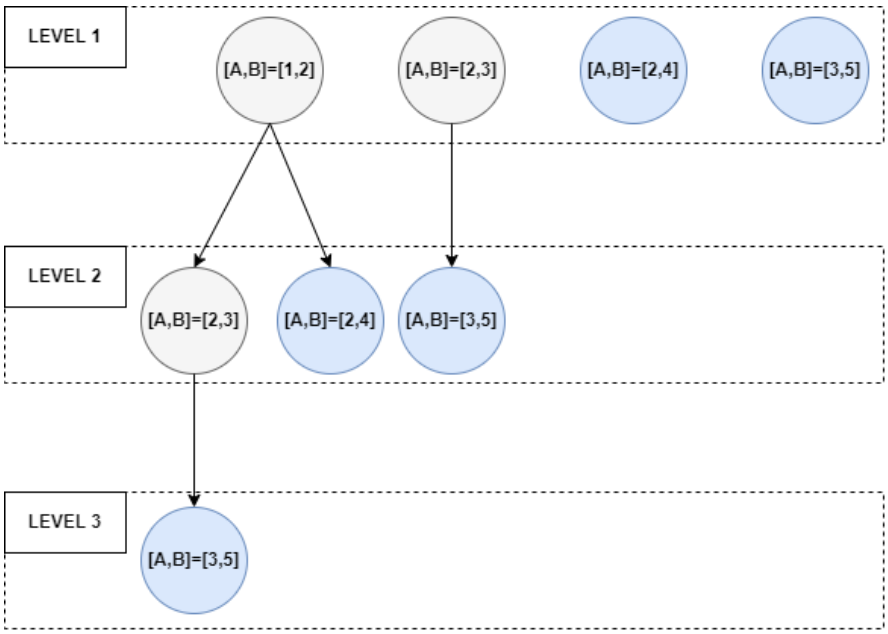
T1	
A	B
1	2
2	3
2	4
3	5

执行语句：

```
SELECT * FROM T1 CONNECT BY PRIOR B=A AND LEVEL<=3;
```

其结果的树型结构如图7-13所示，其中蓝色的节点为叶子节点。

图 7-13 执行逻辑结构图



除了伪列之外，还提供下述的查询函数（详见**层次递归查询函数**）

- sys_connect_by_path(col, separator)：返回从根节点到当前行的连接路径。参数col为路径中显示的列的名称，separator为连接符。
- connect_by_root(col)：显示该节点最顶级的节点，col为输出列的名称。

如果数据集中存在循环，数据库会提供循环检测。默认行为检查到循环会直接报错，不返回任何数据。同时，提供NOCYCLE关键字，查询可以正常执行，只是碰到第一条重复的数据时，会直接退出，而不是报错。

此外，在层次查询过程中，严格按照深度优先搜索的顺序进行。如果在START WITH或CONNECT BY中使用rownum作为过滤条件，对于每条尝试被返回的记录，rownum会先加1，之后按照rownum相关条件判断；对于不满足的记录，会被丢弃且rownum会减1。

须知

- PRIOR 关键字只能出现在CONNECT BY语句中，不能出现在START WITH语句中。
- 除targetlist外，“prior(单列)”会被解析为“prior 单列”，不感知用户自定义的名为prior的函数。
- 只能对表中的列指定PRIOR，不支持对表达式、伪列及类型转换指定PRIOR关键字，如 PRIOR (a + 1) 不被允许。
- CONNECT BY语句中，PRIOR 修饰的列不可以和 level/rownum 等伪列在同一个条件里；但是可以在不同条件里。如 (PRIOR a = level) 不允许， (PRIOR a = b) and (level = 1) 允许。不同条件指的是CONNECT BY语句最上层的 and 连接起来的条件。比如（PRIOR a = 1 or level = 1）算作一个条件，也不被允许。
- START WITH/CONNECT BY语句中禁止将伪列用于子链接，即类似于 "rownum = (子查询)" 或 "rownum in (子查询)"。
- CONNECT BY子句中不建议使用ROWNUM，如需使用请充分测试，避免结果和预期不一致。
- 在with as定义的cte上调用START WITH/CONNECT BY时，如果cte有多个，需要保证每一个cte的定义不依赖于其他cte。
- 如果数据中不存在环路，但是报错runs into cycle，需要考虑增大max_recursive_times。
- connect_by_isleaf、connect_by_iscycle和level类型均为int。
- connect_by_isleaf、connect_by_iscycle和level在层次查询下，一定会被解析为伪列。
- START WITH、CONNECT BY不支持调用投影列中定义的别名。
- START WITH调优建议：
  - 根据CONNECT BY中的条件，建立对应的索引，来提高START WITH语句的性能。
  - 根据 explain performance或者WDR报告中的计划识别瓶颈点，如果发现Recursive Union的递归部分的算子（内层计划）为 HASH JOIN，但是Hash 表是针对临时表 tmp_result构建或者计划中显示hash表发生物化（batch大于1），可能是 work_mem 过小导致无法对外层数据表建立Hash表。可以通过调大work_mem参数来提高性能。

    说明：GaussDB会对小数据量的表有优化，把表的结果缓存在hash表中来提高性能，此时不需要索引。但是如果数据量超过work_mem的限制，该优化会失效，此时可采用建立索引的方式尝试优化。

须知

不建议在创建视图定义时使用递归查询，否则所得视图定义有误或出现报错。这是因为递归查询语句底层会被改写成Recursive Union语句，而视图显示的定义是通过反解析Recursive Union的查询树所得，因此，创建带有递归查询的视图之后，查看视图定义是Recursive Union语句，而不是原的START WITH语句，甚至语句不一定合法。虽然不影响当前数据库视图的使用功能，但是使用gs_dump/gs_restore等通过查看视图定义来迁移视图的工具迁移数据时，会导致restore失败，需要手动重新创建视图。

警告

- ORDER SIBLINGS BY子句

  START WITH语句输出时，不同层的数据会依次返回。但是在每一层内部，是没有任何顺序保证的，这是因为每一轮查询的过程中，数据库会自动选择最优的执行路径。上文的例子中，保证A会被先输出，但是B、C、D之间的顺序不固定。如果用户对最终输出顺序有需求，可以用ORDER SIBLINGS BY子句，用法和ORDER BY子句一样，用于在递归过程中每层内部的排序。

须知

ORDER SIBLINGS BY后的表达式仅支持对普通列、列名偏移量、以及对列名的非聚集、窗口函数调用的方式进行排序，不支持对列名调用START WITH相关系统函数和使用START WITH相关伪列等方式。

- **GROUP BY子句**

  将查询结果按某一列或多列的值分组，值相等的为一组。

  - CUBE ({ expression | (expression [, ...]) } [, ...])

    CUBE是自动对group by子句中列出的字段进行分组汇总，结果集将包含维度列中各值的所有可能组合，以及与这些维度值组合相匹配的基础行中的聚合值。它会为每个分组返回一行汇总信息，用户可以使用CUBE来产生交叉表值。如，在CUBE子句中给出三个表达式（n = 3），运算结果为$2^n = 2^3 = 8$组。 以n个表达式的值分组的行称为常规行，其余的行称为超级聚集行。

  - GROUPING SETS (grouping_element [, ...])

    GROUPING SETS子句是GROUP BY子句的进一步扩展，它可以使用户指定多个GROUP BY选项。这样做可以通过裁剪用户不需要的数据组来提高效率。当用户指定了所需的数据组时，数据库不需要执行完整CUBE或ROLLUP生成的聚合集合。

须知

- 如果SELECT列表的表达式中引用了那些没有分组的字段，则会报错，除非使用了聚集函数，因为对于未分组的字段，可能返回多个数值。
- 如果SELECT列表的表达式中引用了常量，则无需在GROUP BY子句中对该常量进行分组，否则会报错。
- 当在GROUP BY子句中使用主键时，SELECT语句中的所有非聚合列都可以出现在结果集中，因为主键的唯一性确保了每个分组中的非聚合列值的确定性。

- **HAVING子句**

  与GROUP BY子句配合用来选择特殊的组。HAVING子句将组的一些属性与一个常数值比较，只有满足HAVING子句中的逻辑表达式的组才会被提取出来。

- **WINDOW子句**

  一般形式为WINDOW window_name AS (window_definition) [， ...]，window_name是可以被随后的窗口定义所引用的名称，window_definition可以是以下的形式：

```
[ existing_window_name ]
[ PARTITION BY expression [, ...] ]
[ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS { FIRST | LAST } ] [, ...] ]
[ frame_clause ]
```

frame_clause为窗函数定义一个窗口框架window frame，窗函数（并非所有）依赖于框架，window frame是当前查询行的一组相关行。frame_clause可以是以下的形式：

```
[ RANGE | ROWS ] frame_start
[ RANGE | ROWS ] BETWEEN frame_start AND frame_end
```

frame_start和frame_end可以是：

```
UNBOUNDED PRECEDING
value PRECEDING
CURRENT ROW
value FOLLOWING
UNBOUNDED FOLLOWING
```

说明

frame_start和frame_end当前不支持value PRECEDING和value FOLLOWING。

- **UNION子句**

  UNION计算多个SELECT语句返回行集合的并集。

  UNION子句有如下约束条件：

  - 除非声明了ALL子句，否则缺省的UNION结果不包含重复的行。
  - 同一个SELECT语句中的多个UNION操作符是从左向右计算的，除非用圆括弧进行了标识。
  - FOR UPDATE，FOR NO KEY UPDATE，FOR SHARE和FOR KEY SHARE不能在UNION的结果或输入中声明。
  - 通过GUC参数enable_union_all_order控制UNION ALL子查询是否保序：
    - 开启方式：set enable_union_all_order to on。
    - 关闭方式：set enable_union_all_order to off。
    - 查询方式：show enable_union_all_order。

    在开启enable_union_all_order参数时，在UNION声明了ALL并且主查询未排序的情况下，支持子查询保序。其他情况均不支持子查询保序。

  一般表达式：

  select_statement UNION [ALL] select_statement

  - select_statement可以是任何没有ORDER BY、LIMIT、FOR UPDATE，FOR NO KEY UPDATE，FOR SHARE或FOR KEY SHARE子句的SELECT语句。
  - 如果用圆括弧包围，ORDER BY和LIMIT可以附着在子表达式里。

须知

拼接UNION ALL子句时，推荐子句个数<100，超出此值时需保证实例内存足够，以避免内存不足。

- **INTERSECT子句**

  INTERSECT计算多个SELECT语句返回行集合的交集，不含重复的记录。

  INTERSECT子句有如下约束条件：

- 同一个SELECT语句中的多个INTERSECT操作符是从左向右计算的，除非用圆括弧进行了标识。
- 当对多个SELECT语句的执行结果进行UNION和INTERSECT操作的时候，会优先处理INTERSECT。

一般形式：

select_statement INTERSECT select_statement

select_statement可以是任何没有FOR UPDATE，FOR NO KEY UPDATE，FOR SHARE或FOR KEY SHARE子句的SELECT语句。

- **EXCEPT子句**

  EXCEPT子句有如下的通用形式：

  select_statement EXCEPT [ALL] select_statement

  select_statement是任何没有FOR UPDATE，FOR NO KEY UPDATE，FOR SHARE或FOR KEY SHARE子句的SELECT表达式。

  EXCEPT操作符计算存在于左边SELECT语句的输出而不存在于右边SELECT语句输出的行。

  EXCEPT的结果不包含任何重复的行，除非声明了ALL选项。使用ALL时，一个在左边表中有m个重复而在右边表中有n个重复的行将在结果中出现max(m-n,0) 次。

  除非用圆括弧指明顺序，否则同一个SELECT语句中的多个EXCEPT操作符是从左向右计算的。EXCEPT和UNION的绑定级别相同。

  目前，不能给EXCEPT的结果或者任何EXCEPT的输入声明FOR UPDATE，FOR NO KEY UPDATE，FOR SHARE和FOR KEY SHARE子句。

- **MINUS子句**

  与EXCEPT子句具有相同的功能和用法。

- **ORDER BY子句**

  对SELECT语句检索得到的数据进行升序或降序排序。对于ORDER BY表达式中包含多列的情况：

  - 首先根据最左边的列进行排序，如果这一列的值相同，则根据下一个表达式进行比较，以此类推。
  - 如果对于所有声明的表达式都相同，则按随机顺序返回。
  - 在与DISTINCT关键字一起使用的情况下，ORDER BY中排序的列必须包括在SELECT语句所检索的结果集的列中。
  - 在与GROUP BY子句一起使用的情况下，ORDER BY中排序的列必须包括在SELECT语句所检索的结果集的列中。

  须知

  如果要支持中文拼音排序，需要在初始化数据库时指定编码格式为UTF-8、GB18030、GB18030_2022或GBK。命令如下:

  ```
  initdb –E UTF8 –D ../data –locale=zh_CN.UTF-8、initdb -E GB18030 -D ../data -locale=zh_CN.GB18030、initdb -E GB18030_2022 -D ../data -locale=zh_CN.GB18030或initdb –E GBK –D ../data –locale=zh_CN.GBK。
  ```

- **LIMIT子句**

  LIMIT子句由两个独立的子句组成：

  LIMIT { count | ALL }限制返回行数，count为指定行数，LIMIT ALL的效果和省略LIMIT子句一样。

OFFSET start count声明返回的最大行数，而start声明开始返回行之前忽略的行数。如果两个都指定了，会在开始计算count个返回行之前先跳过start行。

LIMIT子句不支持ROWNUM作为count或者offset。

- **OFFSET子句**

  SQL：2008开始提出一种不同的语法：

  OFFSET start { ROW | ROWS }

  start声明开始返回行之前忽略的行数。

- **FETCH { FIRST | NEXT } [count] { ROW | ROWS } ONLY**

  如果不指定count，默认值为1，FETCH子句限定返回查询结果从第一行开始的总行数。

- **锁定子句**

  FOR UPDATE子句将对SELECT检索出来的行进行加锁。这样避免它们在当前事务结束前被其他事务修改或者删除，即其他企图UPDATE、 DELETE、 SELECT FOR UPDATE、SELECT FOR NO KEY UPDATE, SELECT FOR SHARE 或 SELECT FOR KEY SHARE这些行的事务将被阻塞，直到当前事务结束。任何在一行上的DELETE命令也会获得FOR UPDATE锁模式，在主键列上修改值的UPDATE也会获得该锁模式。反过来，SELECT FOR UPDATE将等待已经在相同行上运行以上这些命令的并发事务，并且接着锁定并且返回被更新的行（或者没有行，因为行可能已被删除）。

  FOR NO KEY UPDATE行为与FOR UPDATE类似，不过获得的锁较弱，这种锁将不会阻塞尝试在相同行上获得锁的SELECT FOR KEY SHARE命令。任何不获取FOR UPDATE锁的UPDATE也会获得这种锁模式。

  FOR SHARE的行为类似，只是它在每个检索出来的行上要求一个共享锁，而不是一个排他锁。一个共享锁阻塞其它事务执行UPDATE、DELETE、SELECT FOR UPDATE或者SELECT FOR NO KEY UPDATE，不阻塞SELECT FOR SHARE或者SELECT FOR KEY SHARE。

  FOR KEY SHARE行为与FOR SHARE类似，不过锁较弱，SELECT FOR UPDATE会被阻塞，但是SELECT FOR NO KEY UPDATE不会被阻塞。一个键共享锁会阻塞其他事务执行修改键值的DELETE或者UPDATE，但不会阻塞其他UPDATE，也不会阻止SELECT FOR NO KEY UPDATE、SELECT FOR SHARE或者SELECT FOR KEY SHARE。

  为了避免操作等待其他事务提交，可使用NOWAIT选项，如果被选择的行不能立即被锁住，将会立即报错，而不是等待；WAIT n选项，如果被选择的行不能立即被锁住，等待n秒（其中，n为int类型，取值范围：0 <= n <= 2147483），n秒内获取锁则正常执行，否则报错。

  如果在锁定子句中明确指定了表名称，则只有这些指定的表被锁定，其他在SELECT中使用的表将不会被锁定。否则，将锁定该命令中所有使用的表。

  如果锁定子句应用于一个视图或者子查询，它同样将锁定所有该视图或子查询中使用到的表。

  多个锁定子句可以用于为不同的表指定不同的锁定模式。

  如果一个表中同时出现（或隐含同时出现）在多个子句中，则按照最强的锁处理。类似的，如果影响一个表的任意子句中出现了NOWAIT，该表将按照NOWAIT处理。

须知

- 对ustore表的查询只支持FOR SHARE/FOR UPDATE，不支持FOR KEY SHARE/FOR NO KEY UPDATE。
- 对于子查询是stream计划的FOR UPDATE/SHARE语句，不支持加锁的同一行被并发更新。

- **NLS_SORT**

  指定某字段按照特殊方式排序。目前仅支持中文拼音格式排序和不区分大小写排序。如果要支持此排序方式，在创建数据库时需要指定编码格式为“UTF8”、”GB18030”、”GB18030_2022”或“GBK”；如果指定为其他编码，如SQL_ASCII，则可能报错或者排序无效。

  取值范围：

  – SCHINESE_PINYIN_M，按照中文拼音排序。
  – generic_m_ci，不区分大小写排序（可选，仅支持纯英文不区分大小写排序）。

- **PARTITION子句**

  查询某个分区表中相应分区的数据。

示例

```
--创建SCHEMA。
gaussdb=# CREATE SCHEMA tpcds;

--创建表tpcds.reason。
gaussdb=# CREATE TABLE tpcds.reason
(
  r_reason_sk       integer,
  r_reason_id       character(16),
  r_reason_desc     character(100)
);

--向表中插入多条记录。
gaussdb=# INSERT INTO tpcds.reason values(3,'AAAAAAAABAAAAAAA','reason 1'),
(10,'AAAAAAAABAAAAAAA','reason 2'),(4,'AAAAAAAABAAAAAAA','reason 3'),
(10,'AAAAAAAABAAAAAAA','reason 4'),(10,'AAAAAAAABAAAAAAA','reason 5'),
(20,'AAAAAAAACAAAAAAA','N%reason 6'),(30,'AAAAAAAACAAAAAAA','W%reason 7');

--先通过子查询得到一张临时表temp_t，然后查询表temp_t中的所有数据。
gaussdb=# WITH temp_t(name,isdba) AS (SELECT usename,usesuper FROM pg_user) SELECT * FROM
temp_t;

--查询tpcds.reason表的所有r_reason_sk记录，且去除重复。
gaussdb=# SELECT DISTINCT(r_reason_sk) FROM tpcds.reason;

--LIMIT子句示例：获取表中一条记录。
gaussdb=# SELECT * FROM tpcds.reason LIMIT 1;

--查询所有记录，且按字母升序排列。
gaussdb=# SELECT r_reason_desc FROM tpcds.reason ORDER BY r_reason_desc;

--通过表别名，从pg_user和pg_user_status这两张表中获取数据。
gaussdb=# SELECT a.usename,b.locktime FROM pg_user a,pg_user_status b WHERE a.usesysid=b.roloid;

--FULL JOIN子句示例：将pg_user和pg_user_status这两张表的数据进行全连接显示，即数据的合集。
gaussdb=# SELECT a.usename,b.locktime,a.usesuper FROM pg_user a FULL JOIN pg_user_status b on
a.usesysid=b.roloid;

--GROUP BY子句示例：根据查询条件过滤，并对结果进行分组。
gaussdb=# SELECT r_reason_id, AVG(r_reason_sk) FROM tpcds.reason GROUP BY r_reason_id HAVING
```



```
AVG(r_reason_sk) > 25;

--GROUP BY CUBE子句示例：根据查询条件过滤，并对结果进行分组汇总。
gaussdb=# SELECT r_reason_id,AVG(r_reason_sk) FROM tpcds.reason GROUP BY
CUBE(r_reason_id,r_reason_sk);

--GROUP BY GROUPING SETS子句示例:根据查询条件过滤，并对结果进行分组汇总。
gaussdb=# SELECT r_reason_id,AVG(r_reason_sk) FROM tpcds.reason GROUP BY GROUPING
SETS((r_reason_id,r_reason_sk),r_reason_sk);

--UNION子句示例：将表tpcds.reason里r_reason_desc字段中的内容以W开头和以N开头的进行合并。
gaussdb=# SELECT r_reason_sk, tpcds.reason.r_reason_desc
    FROM tpcds.reason
    WHERE tpcds.reason.r_reason_desc LIKE 'W%'
UNION
SELECT r_reason_sk, tpcds.reason.r_reason_desc
    FROM tpcds.reason
    WHERE tpcds.reason.r_reason_desc LIKE 'N%';

--NLS_SORT子句示例：中文拼音排序。
gaussdb=# SELECT * FROM tpcds.reason ORDER BY NLSSORT( r_reason_desc, 'NLS_SORT =
SCHINESE_PINYIN_M');


--不区分大小写排序（可选，仅支持纯英文不区分大小写排序）：
gaussdb=# SELECT * FROM tpcds.reason ORDER BY NLSSORT( r_reason_desc, 'NLS_SORT = generic_m_ci');

--创建分区表tpcds.reason_p。
gaussdb=# CREATE TABLE tpcds.reason_p
(
  r_reason_sk integer,
  r_reason_id character(16),
  r_reason_desc character(100)
)
PARTITION BY RANGE (r_reason_sk)
(
  partition P_05_BEFORE values less than (05),
  partition P_15 values less than (15),
  partition P_25 values less than (25),
  partition P_35 values less than (35),
  partition P_45_AFTER values less than (MAXVALUE)
)
;

--插入数据。
gaussdb=# INSERT INTO tpcds.reason_p values(3,'AAAAAAAABAAAAAAA','reason 1'),
(10,'AAAAAAAABAAAAAAA','reason 2'),(4,'AAAAAAAABAAAAAAA','reason 3'),
(10,'AAAAAAAABAAAAAAA','reason 4'),(10,'AAAAAAAABAAAAAAA','reason 5'),
(20,'AAAAAAAACAAAAAAA','reason 6'),(30,'AAAAAAAACAAAAAAA','reason 7');

--PARTITION子句示例：从tpcds.reason_p的表分区P_05_BEFORE中获取数据。
gaussdb=#  SELECT * FROM tpcds.reason_p PARTITION (P_05_BEFORE);
 r_reason_sk |   r_reason_id    |   r_reason_desc
-------------+------------------+-----------------------------------
           4 | AAAAAAAABAAAAAAA | reason 3
           3 | AAAAAAAABAAAAAAA | reason 1
(2 rows)

--GROUP BY子句示例：按r_reason_id分组统计tpcds.reason_p表中的记录数。
gaussdb=# SELECT COUNT(*),r_reason_id FROM tpcds.reason_p GROUP BY r_reason_id;
 count |   r_reason_id
-------+------------------
     2 | AAAAAAAACAAAAAAA
     5 | AAAAAAAABAAAAAAA
(2 rows)

--GROUP BY CUBE子句示例：根据查询条件过滤，并对查询结果分组汇总。
gaussdb=# SELECT * FROM tpcds.reason GROUP BY  CUBE (r_reason_id,r_reason_sk,r_reason_desc);
```



```
--GROUP BY GROUPING SETS子句示例：根据查询条件过滤，并对查询结果分组汇总。
gaussdb=# SELECT * FROM tpcds.reason GROUP BY  GROUPING SETS
((r_reason_id,r_reason_sk),r_reason_desc);

--HAVING子句示例：按r_reason_id分组统计tpcds.reason_p表中的记录，并只显示r_reason_id个数大于2的信息。
gaussdb=# SELECT COUNT(*) c,r_reason_id FROM tpcds.reason_p GROUP BY r_reason_id HAVING c>2;
 c |   r_reason_id
---+------------------
 5 | AAAAAAAABAAAAAAA
(1 row)

--IN子句示例：按r_reason_id分组统计tpcds.reason_p表中的r_reason_id个数，并只显示r_reason_id值为AAAAAAAABAAAAAAA或AAAAAAAADAAAAAAA的个数。
gaussdb=# SELECT COUNT(*),r_reason_id FROM tpcds.reason_p GROUP BY r_reason_id HAVING r_reason_id IN('AAAAAAAABAAAAAAA','AAAAAAAADAAAAAAA');
count |   r_reason_id
-------+------------------
     5 | AAAAAAAABAAAAAAA
(1 row)

--INTERSECT子句示例：查询r_reason_id等于AAAAAAAABAAAAAAA，并且r_reason_sk小于5的信息。
gaussdb=# SELECT * FROM tpcds.reason_p WHERE r_reason_id='AAAAAAAABAAAAAAA' INTERSECT SELECT * FROM tpcds.reason_p WHERE r_reason_sk<5;
 r_reason_sk |   r_reason_id    |      r_reason_desc
-------------+------------------+----------------------------------
           4 | AAAAAAAABAAAAAAA | reason 3
           3 | AAAAAAAABAAAAAAA | reason 1
(2 rows)

--EXCEPT子句示例：查询r_reason_id等于AAAAAAAABAAAAAAA，并且去除r_reason_sk小于4的信息。
gaussdb=# SELECT * FROM tpcds.reason_p WHERE r_reason_id='AAAAAAAABAAAAAAA' EXCEPT SELECT * FROM tpcds.reason_p WHERE r_reason_sk<4;
r_reason_sk |   r_reason_id    |      r_reason_desc
-------------+------------------+----------------------------------
          10 | AAAAAAAABAAAAAAA | reason 2
          10 | AAAAAAAABAAAAAAA | reason 5
          10 | AAAAAAAABAAAAAAA | reason 4
           4 | AAAAAAAABAAAAAAA | reason 3
(4 rows)

--创建表store_returns、customer
gaussdb=#  CREATE TABLE tpcds.store_returns (sr_item_sk int, sr_customer_id varchar(50),sr_customer_sk int);
gaussdb=#  CREATE TABLE tpcds.customer (c_item_sk int, c_customer_id varchar(50),c_customer_sk int);

--通过在where子句中指定"(+)"来实现左连接。
gaussdb=# SELECT t1.sr_item_sk ,t2.c_customer_id FROM tpcds.store_returns t1, tpcds.customer t2 WHERE t1.sr_customer_sk  = t2.c_customer_sk(+)
ORDER BY 1 DESC LIMIT 1;
 sr_item_sk | c_customer_id
------------+---------------
      18000 |
(1 row)

--通过在where子句中指定"(+)"来实现右连接。
gaussdb=# SELECT t1.sr_item_sk ,t2.c_customer_id FROM tpcds.store_returns t1, tpcds.customer t2 WHERE t1.sr_customer_sk(+)  = t2.c_customer_sk
ORDER BY 1 DESC LIMIT 1;
 sr_item_sk |  c_customer_id
------------+-----------------
            | AAAAAAAAJNGEBAAA
(1 row)

--通过在where子句中指定"(+)"来实现左连接，并且增加连接条件。
gaussdb=# SELECT t1.sr_item_sk ,t2.c_customer_id FROM tpcds.store_returns t1, tpcds.customer t2 WHERE t1.sr_customer_sk  = t2.c_customer_sk(+) and t2.c_customer_sk(+) < 1 ORDER BY  1  LIMIT 1;
 sr_item_sk | c_customer_id
------------+---------------
```



```
        1 |
(1 row)

--不支持在where子句中指定"(+)"的同时使用内层嵌套AND/OR的表达式。
gaussdb=# SELECT t1.sr_item_sk ,t2.c_customer_id FROM tpcds.store_returns t1, tpcds.customer t2 WHERE
NOT (t1.sr_customer_sk  = t2.c_customer_sk(+) and t2.c_customer_sk(+) < 1);
ERROR:  Operator "(+)" can not be used in nesting expression.
LINE 1: ...tomer_id from store_returns t1, customer t2 where not(t1.sr_...
                                                             ^
--where子句在不支持表达式宏指定"(+)"会报错。
gaussdb=# SELECT t1.sr_item_sk ,t2.c_customer_id FROM tpcds.store_returns t1, tpcds.customer t2 WHERE
(t1.sr_customer_sk  = t2.c_customer_sk(+))::bool;
ERROR:  Operator "(+)" can only be used in common expression.

--where子句在表达式的两边都指定"(+)"会报错。
gaussdb=# SELECT t1.sr_item_sk ,t2.c_customer_id FROM tpcds.store_returns t1, tpcds.customer t2 WHERE
t1.sr_customer_sk(+)  = t2.c_customer_sk(+);
ERROR:  Operator "(+)" can't be specified on more than one relation in one join condition
HINT:  "t1", "t2"...are specified Operator "(+)" in one condition.

--删除表。
gaussdb=# DROP TABLE tpcds.reason_p;

--闪回查询示例，使用闪回功能需要设置undo_retention_time参数
--创建表tpcds.time_table
gaussdb=#  CREATE TABLE tpcds.time_table(idx integer, snaptime timestamp, snapcsn bigint, timeDesc
character(100));
--向表tpcds.time_table中插入记录
gaussdb=#  INSERT INTO tpcds.time_table select 1, now(),int8in(xidout(next_csn)), 'time1' from
gs_get_next_xid_csn();
gaussdb=#  INSERT INTO tpcds.time_table select 2, now(),int8in(xidout(next_csn)), 'time2' from
gs_get_next_xid_csn();
gaussdb=#  INSERT INTO tpcds.time_table select 3, now(),int8in(xidout(next_csn)), 'time3' from
gs_get_next_xid_csn();
gaussdb=#  INSERT INTO tpcds.time_table select 4, now(),int8in(xidout(next_csn)), 'time4' from
gs_get_next_xid_csn();
gaussdb=#  SELECT * FROM tpcds.time_table;

 idx |          snaptime          | snapcsn |                                                timedesc
-----+----------------------------+---------
+-------------------------------------------------------------------------------------------------------------
   1 | 2021-04-25 17:50:05.360326 |  107322 | time1
   2 | 2021-04-25 17:50:10.886848 |  107324 | time2
   3 | 2021-04-25 17:50:16.12921  |  107327 | time3
   4 | 2021-04-25 17:50:22.311176 |  107330 | time4
(4 rows)
gaussdb=#  DELETE tpcds.time_table;
DELETE 4
--2021-04-25 17:50:22.311176应该使用tpcds.time_table中第四条snaptime字段值
gaussdb=#  SELECT * FROM tpcds.time_table TIMECAPSULE TIMESTAMP to_timestamp('2021-04-25
17:50:22.311176','YYYY-MM-DD HH24:MI:SS.FF');
 idx |          snaptime          | snapcsn |                                                timedesc
-----+----------------------------+---------
+-------------------------------------------------------------------------------------------------------------
   1 | 2021-04-25 17:50:05.360326 |  107322 | time1
   2 | 2021-04-25 17:50:10.886848 |  107324 | time2
   3 | 2021-04-25 17:50:16.12921  |  107327 | time3
(3 rows)
--107330 csn应该使用tpcds.time_table中第四条snapcsn字段值
gaussdb=#  SELECT * FROM tpcds.time_table TIMECAPSULE CSN 107330;
 idx |          snaptime          | snapcsn |                                                timedesc
-----+----------------------------+---------
+-------------------------------------------------------------------------------------------------------------
   1 | 2021-04-25 17:50:05.360326 |  107322 | time1
   2 | 2021-04-25 17:50:10.886848 |  107324 | time2
   3 | 2021-04-25 17:50:16.12921  |  107327 | time3
(3 rows)

--WITH RECURSIVE查询示例：计算从1到100的累加值。
```



```
gaussdb=#  WITH RECURSIVE t1(a) AS (
 SLEECT 100
),
t(n) AS (
    VALUES (1)
  UNION ALL
    SELECT n+1 FROM t WHERE n < (SELECT max(a) FROM t1)
)
SELECT sum(n) FROM t;
 sum
------
 5050
(1 row)

gaussdb=# DROP TABLE t;
--UNPIVOT子句示例：将表p1的math列和phy列转置为（class，score）行
gaussdb=# CREATE TABLE p1(id int, math int, phy int);
gaussdb=# INSERT INTO p1 values(1,20,30);
gaussdb=# INSERT INTO p1 values(2,30,40);
gaussdb=# INSERT INTO p1 values(3,40,50);
gaussdb=# SELECT * FROM p1;
 id | math | phy
----+------+-----
  1 |   20 |  30
  2 |   30 |  40
  3 |   40 |  50
(3 rows)

gaussdb=# SELECT * FROM p1 UNPIVOT(score FOR class IN(math, phy));
 id | class | score
----+-------+-------
  1 | MATH  |    20
  1 | PHY   |    30
  2 | MATH  |    30
  2 | PHY   |    40
  3 | MATH  |    40
  3 | PHY   |    50
(6 rows)

--PIVOT子句示例：将表p2的（class，score）行转置为'MATH'列和 'PHY'列
gaussdb=# CREATE TABLE p2(id int, class varchar(10), score int);
gaussdb=# INSERT INTO p2 SELECT * FROM p1 UNPIVOT(score FOR class IN(math, phy));
gaussdb=# SELECT * FROM p2;
 id | class | score
----+-------+-------
  1 | MATH  |    20
  1 | PHY   |    30
  2 | MATH  |    30
  2 | PHY   |    40
  3 | MATH  |    40
  3 | PHY   |    50
(6 rows)

gaussdb=# SELECT * FROM p2 PIVOT(max(score) FOR class IN ('MATH', 'PHY'));
 id | 'MATH' | 'PHY'
----+--------+-------
  1 |     20 |    30
  3 |     40 |    50
  2 |     30 |    40
(3 rows)

gaussdb=# DROP TABLE p1;
gaussdb=# DROP TABLE p2;
--删除表。
gaussdb=# DROP TABLE tpcds.reason;

--删除SCHEMA。
gaussdb=# DROP SCHEMA tpcds CASCADE;
```


7.14.170 SELECT INTO

功能描述

SELECT INTO用于根据查询结果创建一个新表，并且将查询到的数据插入到新表中。

数据并不返回给客户端，这一点和普通的SELECT不同。新表的字段具有和SELECT的输出字段相同的名称和数据类型。

注意事项

CREATE TABLE AS的作用和SELECT INTO类似，且提供了SELECT INTO所提供功能的超集。建议使用CREATE TABLE AS语法替代SELECT INTO，因为SELECT INTO不能在存储过程中使用。

语法格式

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    { * | {expression [ [ AS ] output_name ]} [, ...] }
    INTO [ [ GLOBAL | LOCAL ] [ TEMPORARY | TEMP ] | UNLOGGED ] [ TABLE ] new_table
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY expression [, ...] ]
    [ HAVING condition [, ...] ]
    [ WINDOW {window_name AS ( window_definition )} [, ...] ]
    [ { UNION | INTERSECT | EXCEPT | MINUS } [ ALL | DISTINCT ] select ]
    [ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS { FIRST |
LAST } ]} [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start [ ROW | ROWS ] ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
    [ {FOR { UPDATE | SHARE } [ OF table_name [, ...] ] [ NOWAIT |WAIT N]} [...] ];
```

参数说明

- **new_table**

  new_table指定新建表的名称。

- **UNLOGGED**

  指定表为非日志表。在非日志表中写入的数据不会被写入到预写日志中，这样就会比普通表快很多。但是，它也是不安全的，非日志表在冲突或异常关机后会被自动删截。非日志表中的内容也不会被复制到备用服务器中。在该类表中创建的索引也不会被自动记录。

  - 使用场景：非日志表不能保证数据的安全性，用户应该在确保数据已经做好备份的前提下使用，例如系统升级时进行数据的备份。
  - 故障处理：当异常关机等操作导致非日志表上的索引发生数据丢失时，用户应该对发生错误的索引进行重建。

- **GLOBAL | LOCAL**

  创建临时表时可以在TEMP或TEMPORARY前指定GLOBAL或LOCAL关键字。如果指定GLOBAL关键字，GaussDB会创建全局临时表，否则GaussDB会创建本地临时表。

- **TEMPORARY | TEMP**

  如果指定TEMP或TEMPORARY关键字，则创建的表为临时表。临时表分为全局临时表和本地临时表两种类型。创建临时表时如果指定GLOBAL关键字则为全局临时表，否则为本地临时表。

全局临时表的元数据对所有会话可见，会话结束后元数据继续存在。会话与会话之间的用户数据、索引和统计信息相互隔离，每个会话只能看到和更改自己提交的数据。全局临时表有两种模式：一种是基于会话级别的(ON COMMIT PRESERVE ROWS)，当会话结束时自动清空用户数据；一种是基于事务级别的(ON COMMIT DELETE ROWS)，当执行commit或rollback时自动清空用户数据。建表时如果没有指定ON COMMIT选项，则缺省为会话级别。与本地临时表不同，全局临时表建表时可以指定非pg_temp_开头的schema。

本地临时表只在当前会话可见，本会话结束后会自动删除。因此，在除当前会话连接的数据库节点故障时，仍然可以在当前会话上创建和使用临时表。由于临时表只在当前会话创建，对于涉及对临时表操作的DDL语句，会产生DDL失败的报错。因此，建议DDL语句中不要对临时表进行操作。TEMP和TEMPORARY等价。

须知

- 本地临时表通过每个会话独立的以pg_temp开头的schema来保证只对当前会话可见，因此，不建议用户在日常操作中手动删除以pg_temp、pg_toast_temp开头的schema。
- 如果建表时不指定TEMPORARY/TEMP关键字，而指定表的schema为当前会话的pg_temp_开头的schema，则此表会被创建为临时表。
- 如果其它会话正在使用全局临时表或索引，则无法对其执行ALTER/DROP操作。
- 全局临时表的DDL只会影响当前会话的用户数据和索引。例如truncate、reindex、analyze只对当前会话有效。

说明

SELECT INTO的其它参数可参考SELECT的**参数说明**。

示例

```
--创建SCHEMA。
gaussdb=# CREATE SCHEMA tpcds;

--创建表tpcds.reason。
gaussdb=# CREATE TABLE tpcds.reason
(
  r_reason_sk       integer,
  r_reason_id       character(16),
  r_reason_desc     character(100)
);

--向表中插入多条记录。
gaussdb=# INSERT INTO tpcds.reason values(1,'AAAAAAAABAAAAAAA','reason 1'),
(2,'AAAAAAAABAAAAAAA','reason 2'),(3,'AAAAAAAABAAAAAAA','reason 3'),
(4,'AAAAAAAABAAAAAAA','reason 4'),(4,'AAAAAAAABAAAAAAA','reason 5'),
(4,'AAAAAAAACAAAAAAA','reason 6'),(5,'AAAAAAAACAAAAAAA','reason 7');

--将tpcds.reason表中r_reason_sk小于5的值加入到新建表中。
gaussdb=# SELECT * INTO tpcds.reason_t1 FROM tpcds.reason WHERE r_reason_sk < 5;
INSERT 0 6

--删除tpcds.reason_t1表。
gaussdb=# DROP TABLE tpcds.reason_t1;

--删除表。
gaussdb=# DROP TABLE tpcds.reason;
```



```
--删除SCHEMA。
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

相关链接

SELECT

优化建议

- **DATABASE**

  不建议在事务中reindex database。

- **SYSTEM**

  不建议在事务中reindex系统表。

7.14.171 SET

功能描述

用于修改运行时配置参数。

注意事项

大多数运行时参数都可以用SET在运行时设置，但有些则在服务运行过程中或会话开始之后不能修改。

语法格式

- 设置所处的时区。

```
SET [ SESSION | LOCAL ] TIME ZONE { timezone | LOCAL | DEFAULT };
```

- 设置所属的模式。

```
SET [ SESSION | LOCAL ]
    {CURRENT_SCHEMA { TO | = } { schema | DEFAULT }
    | SCHEMA 'schema'};
```

- 设置客户端编码集。

```
SET [ SESSION | LOCAL ] NAMES encoding_name;
```

- 设置XML的解析方式。

```
SET [ SESSION | LOCAL ] XML OPTION { DOCUMENT | CONTENT };
```

- 设置其他运行时参数。

```
SET [ LOCAL | SESSION ]
    {config_parameter { { TO | = } { value | DEFAULT }
                      | FROM CURRENT }};
```

- 在兼容B模式（sql_compatibility = 'B'）下设置参数。

```
SET [ SESSION | @@SESSION. | @@]
      {config_parameter = { expr | DEFAULT }};
```

- 设置自定义用户变量。

```
SET @var_name := expr [, @var_name := expr] ...
SET @var_name = expr [, @var_name = expr] ...
```

参数说明

- **SESSION**

  声明的参数只对当前会话起作用。如果SESSION和LOCAL都没出现，则SESSION为缺省值。

如果在事务中执行了此命令，命令的产生影响将在事务回滚之后消失。如果该事务已提交，影响将持续到会话的结束，除非被另外一个SET命令重置参数。

- **LOCAL**

  声明的参数只在当前事务中有效。在COMMIT或ROLLBACK之后，会话级别的设置将再次生效。

  不论事务是否提交，此命令的影响只持续到当前事务结束。一个特例是：在一个事务里面，既有SET命令，又有SET LOCAL命令，且SET LOCAL在SET后面，则在事务结束之前，SET LOCAL命令会起作用，但事务提交之后，则是SET命令会生效。

- **TIME ZONE timezone**

  用于指定当前会话的本地时区。

  取值范围：有效的本地时区。该选项对应的运行时参数名称为TimeZone，DEFAULT缺省值为PRC。

- **CURRENT_SCHEMA schema**

  CURRENT_SCHEMA用于指定当前的模式。

  取值范围：已存在模式名称。如果模式名不存在，会导致CURRENT_SCHEMA值为空。

- **SCHEMA schema**

  同CURRENT_SCHEMA。此处的schema是个字符串。

  例如：set schema 'public';

- **NAMES encoding_name**

  用于设置客户端的字符编码。等价于set client_encoding to encoding_name。

  取值范围：有效的字符编码。该选项对应的运行时参数名称为client_encoding，默认编码为UTF8。

- **XML OPTION option**

  用于设置XML的解析方式。

  取值范围：CONTENT（缺省）、DOCUMENT

- **config_parameter**

  可设置的运行时参数的名称。可用的运行时参数可以使用SHOW ALL命令查看。

  说明

  部分通过SHOW ALL查看的参数不能通过SET设置。如max_datanodes。

- **value**

  config_parameter的新值。可以声明为字符串常量、标识符、数字，或者逗号分隔的列表。DEFAULT用于把这些参数设置为它们的缺省值。

- **SESSION | @@SESSION. | @@**

  声明的参数生效方式为superuser、user，可通过pg_settings系统视图的context字段确定，如果没有出现GLOBAL /SESSION，则SESSION为缺省值。支持config_parameter赋值为表达式。

说明

1. SET SESSION 只有在兼容B模式下（sql_compatibility = 'B'）支持，并且GUC参数b_format_behavior_compat_options设置值包含 enable_set_variables的场景下才支持（set b_format_behavior_compat_options = 'enable_set_variables')。
2. 使用@@config_parameter进行操作符运算时，尽量使用空格隔开。比如set @@config_parameter1=@@config_parameter1*2; 命令中，会将=@@当做操作符，可将其修改为set @@config_parameter1= @@config_parameter1 * 2 。

- **var_name**

  自定义变量名。变量名只能由数字、字母、下划线（_），点（.）、$组成，如果使用单引号、双引号等引用时，则可以使用其他字符，如'var_name'，“var_name”，`var_name`。

  说明

  - set自定义用户变量的只有在B模式下（即sql_compatibility = 'B'）支持，并且GUC参数b_format_behavior_compat_options设置值包含enable_set_variables的场景下才支持（即b_format_behavior_compat_options = 'enable_set_variables'）。
  - 自定义变量只会存储整型、浮点型、字符串、位串和NULL。对于BOOLEAN、INT1、INT2、INT4、INT8类型会转为INT8类型；FLOAT4、FLOAT8、NUMERIC会转化为FLOAT8进行存储（需要注意浮点型可能会有精度丢失）；BIT类型以BIT存储，VARBIT类型以VARBIT存储；NULL值以NULL存储；其他类型若可转化为字符串，则转为TEXT存储。
  - 使用@var_name进行操作符运算时，尽量使用空格隔开。比如set @v1=@v2+1;命令中，会将=@当做操作符，可将其修改为set @v1= @v2+1。
  - 当sql_compatibility = 'B' && b_format_behavior_compat_options = 'enable_set_variables'时，对于数据库原始的@ expr，请参考**数字操作符**，@需要与expr有空格，否则会将其解析成用户变量。
  - 未初始化的变量值为NULL。
  - prepare语句中用户自定义变量存储的字符串只支持select/insert/update/delete/merge语法。
  - 对于连续赋值的场景，只支持@var_name1 := @var_name2 := ⋯ := expr和@var_name1 = @var_name2 := ⋯ := expr，等号只有放在首位才表示赋值，其他位置表示比较操作符。

- **expr**

  表达式，支持可直接或间接转为整型、浮点型、字符串、位串和NULL的表达式。

  说明

  字符串表达式中避免包含口令等敏感信息的函数，如加解密类函数gs_encrypt，gs_decrypt等，防止敏感信息泄露。

示例

```
--设置模式搜索路径。
gaussdb=# SET search_path TO tpcds, public;

--把日期时间风格设置为传统的 POSTGRES 风格(日在月前)。
gaussdb=# SET datestyle TO postgres,dmy;

--SET自定义用户变量的功能
gaussdb=# create database user_var dbcompatibility 'b';
gaussdb=# \c user_var
user_var=# SET b_format_behavior_compat_options = enable_set_variables;
user_var=# SET @v1 := 1, @v2 := 1.1, @v3 := true, @v4 := 'dasda', @v5 := x'41';
--查询自定义用户变量
user_var=# select @v1, @v2, @v3, @v4, @v5, @v6, @v7;
```



```
--PREPARE语法使用自定义用户变量
user_var=# SET @sql = 'select 1';
user_var=# PREPARE stmt as @sql;
user_var=# EXECUTE stmt;
```

```
--设置B兼容性参数
gaussdb=# create database test_set dbcompatibility 'B';
gaussdb=# \c test_set
test_set=# set b_format_behavior_compat_options = 'enable_set_variables';
```

相关链接

RESET，**SHOW**

7.14.172 SET CONSTRAINTS

功能描述

SET CONSTRAINTS设置当前事务检查行为的约束条件。

IMMEDIATE约束是在每条语句后面进行检查。DEFERRED约束一直到事务提交时才检查。每个约束都有自己的模式。

从创建约束条件开始，一个约束总是设定为DEFERRABLE INITIALLY DEFERRED、DEFERRABLE INITIALLY IMMEDIATE和NOT DEFERRABLE三个特性之一。第三种总是IMMEDIATE，并且不会受SET CONSTRAINTS影响。前两种以指定的方式启动每个事务，但是其行为可以在事务里用SET CONSTRAINTS改变。

带着一个约束名列表的SET CONSTRAINTS改变这些约束的模式（都必须是可推迟的）。如果有多个约束匹配某个名称，则所有都会被影响。SET CONSTRAINTS ALL改变所有可推迟约束的模式。

当SET CONSTRAINTS把一个约束从DEFERRED改成IMMEDIATE的时候，任何将在事务结束准备进行的数据修改都将在SET CONSTRAINTS的时候执行检查。如果违反了任何约束，SET CONSTRAINTS都会失败（并且不会修改约束模式）。因此，SET CONSTRAINTS可以用于强制在事务中某一点进行约束检查。检查约束总是不可推迟的。

注意事项

SET CONSTRAINTS只在当前事务里设置约束的行为。因此，如果用户在事务块之外（START TRANSACTION/COMMIT对）执行这个命令，它将没有任何作用。

语法格式

```
SET CONSTRAINTS { ALL | { name } [, ...] } { DEFERRED | IMMEDIATE };
```

参数说明

- **name**
  约束名。
  取值范围：已存在的约束名。可以在系统表pg_constraint中查到。
- **ALL**
  所有约束。
- **DEFERRED**

约束一直到事务提交时才检查。

- **IMMEDIATE**
  约束在每条语句后进行检查。

示例

```
--设置所有约束在事务提交时检查。
gaussdb=# SET CONSTRAINTS ALL DEFERRED;
```

7.14.173 SET ROLE

功能描述

设置当前会话的当前用户标识符。

注意事项

- 当前会话的用户必须是指定的rolename角色的成员，但系统管理员可以选择任何角色。
- 使用这条命令，它可能会增加一个用户的权限，也可能会限制一个用户的权限。如果会话用户的角色有INHERITS属性，则它自动拥有它能SET ROLE变成的角色的所有权限；在这种情况下，SET ROLE实际上是删除了所有直接赋予会话用户的权限，以及它的所属角色的权限，只剩下指定角色的权限。另一方面，如果会话用户的角色有NOINHERITS属性，SET ROLE删除直接赋予会话用户的权限，而获取指定角色的权限。

语法格式

- 设置当前会话的当前用户标识符。
  ```
  SET [ SESSION | LOCAL ] ROLE role_name PASSWORD 'password';
  ```
- 重置当前用户标识为当前会话用户标识符。
  ```
  RESET ROLE;
  ```

参数说明

- **SESSION**
  声明这个命令只对当前会话起作用，此参数为缺省值。
- **LOCAL**
  声明该命令只在当前事务中有效。
- **role_name**
  角色名。
  取值范围：字符串，要符合**标识符命名规范**。
- **password**
  角色的密码。要求符合密码的命名规则。

  说明

  使用密文密码限制如下：

  - 管理员用户不能使用密文密码切换到其他管理员用户，只能向权限更低用户切换。

  使用密文密码通常用于gs_dump、gs_dumpall导出场景，其他场景不建议直接使用密文密码。

- **RESET ROLE**

  用于重置当前用户标识。

示例

```
--创建角色paul。
gaussdb=# CREATE ROLE paul IDENTIFIED BY '********';

--设置当前用户为paul。
gaussdb=# SET ROLE paul PASSWORD '********';

--查看当前会话用户，当前用户。
gaussdb=> SELECT SESSION_USER, CURRENT_USER;

--重置当前用户。
gaussdb=> RESET ROLE;

--删除用户。
gaussdb=# DROP USER paul;
```

7.14.174 SET SESSION AUTHORIZATION

功能描述

把当前会话里的会话用户标识和当前用户标识都设置为指定的用户。

注意事项

只有在初始会话用户有系统管理员权限的时候，会话用户标识符才能改变。否则，只有在指定了被认证的用户名的情况下，系统才接受该命令。

语法格式

- 为当前会话设置会话用户标识符和当前用户标识符。

  ```
  SET [ SESSION | LOCAL ] SESSION AUTHORIZATION role_name PASSWORD 'password';
  ```

- 重置会话和当前用户标识符为初始认证的用户名。

  ```
  {SET [ SESSION | LOCAL ] SESSION AUTHORIZATION DEFAULT
      | RESET SESSION AUTHORIZATION};
  ```

参数说明

- **SESSION**

  声明这个命令只对当前会话起作用。

- **LOCAL**

  声明该命令只在当前事务中有效。

- **role_name**

  用户名。

  取值范围：字符串，要符合**标识符命名规范**。

- **password**

  角色的密码。要求符合密码的命名规则。

说明

使用密文密码限制如下：

- 管理员用户不能使用密文密码切换到其他管理员用户，只能向权限更低用户切换。

使用密文密码通常用于gs_dump、gs_dumpall导出场景，其他场景不建议直接使用密文密码。

- **DEFAULT**

  重置会话和当前用户标识符为初始认证的用户名。

示例

```
--创建角色paul。
gaussdb=# CREATE ROLE paul IDENTIFIED BY '********';

--设置当前用户为paul。
gaussdb=# SET SESSION AUTHORIZATION paul password '********';

--查看当前会话用户，当前用户。
gaussdb=> SELECT SESSION_USER, CURRENT_USER;

--重置当前用户。
gaussdb=> RESET SESSION AUTHORIZATION;

--删除用户。
gaussdb=# DROP USER paul;
```

相关参考

SET ROLE

7.14.175 SET TRANSACTION

功能描述

为事务设置特性。事务特性包括事务隔离级别、事务访问模式(读/写或者只读)。可以设置当前事务的特性（LOCAL)，也可以设置会话的默认事务特性（SESSION），也可以设置当前数据库的全局会话的事务特性（GLOBAL）。

注意事项

设置当前事务特性需要在事务中执行（即执行SET TRANSACTION之前需要执行START TRANSACTION或者BEGIN），否则设置不生效。设置当前数据库全局会话（GLOBAL）的事务特性在重新连接后生效。

语法格式

设置事务的隔离级别、读写模式。

```
{ SET [ LOCAL | SESSION | GLOBAL ] TRANSACTION|SET SESSION CHARACTERISTICS AS TRANSACTION }
  { ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE | REPEATABLE READ }
  | { READ WRITE | READ ONLY } };
```

参数说明

- **LOCAL**

  声明该命令只在当前事务中有效。

- **SESSION**

  声明这个命令只对当前会话起作用。

  SET SESSION TRANSACTION语句需要在B兼容模式下，设置GUC参数b_format_behavior_compat_options为set_session_transaction后生效。其他情形使用SET SESSION CHARACTERISTICS语句。

- **GLOBAL**

  声明这个命令对当前数据库的全局会话生效。

  作用范围：在集中式的B兼容模式下生效。对后续连接的会话生效。

- **ISOLATION LEVEL**

  指定事务隔离级别，该参数决定当一个事务中存在其他并发运行事务时能够看到什么数据。

  说明

  - 在事务中第一个数据修改语句（SELECT，INSERT，DELETE，UPDATE，FETCH，COPY）执行之后，当前事务的隔离级别就不能再次设置。
  - 事务块内SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION LEVEL对当前事务不生效，需要COMMIT之后才生效。

  取值范围：

  - READ COMMITTED：读已提交隔离级别，只能读到已经提交的数据，而不会读到未提交的数据。这是缺省值。
  - REPEATABLE READ：可重复读隔离级别，仅仅能看到事务开始之前提交的数据，不能看到未提交的数据，以及在事务执行期间由其它并发事务提交的修改。
  - SERIALIZABLE：GaussDB目前功能上不支持此隔离级别，等价于REPEATABLE READ。

- **READ WRITE | READ ONLY**

  指定事务访问模式（读/写或者只读）。

示例

```
--开启一个事务，设置事务的隔离级别为READ COMMITTED，访问模式为READ ONLY。
gaussdb=# START TRANSACTION;
gaussdb=# SET LOCAL TRANSACTION ISOLATION LEVEL READ COMMITTED READ ONLY;
gaussdb=# COMMIT;
--设置当前会话的事务隔离级别、读写模式。
--在sql_compatibility = 'B'场景下,b_format_behavior_compat_options设置为set_session_transaction。
gaussdb=# SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
gaussdb=# SET SESSION TRANSACTION READ ONLY;
--设置当前数据库全局会话的事务隔离级别、读写模式(sql_compatibility = 'B'场景下)。
gaussdb=# SET GLOBAL TRANSACTION ISOLATION LEVEL READ COMMITTED;
gaussdb=# SET GLOBAL TRANSACTION READ ONLY;
```

7.14.176 SHOW

功能描述

SHOW将显示当前运行时参数的数值。

语法格式

```
SHOW
 {
```



```
  [VARIABLES LIKE] configuration_parameter |
  CURRENT_SCHEMA |
  TIME ZONE |
  TRANSACTION ISOLATION LEVEL |
  SESSION AUTHORIZATION |
  ALL
};
```

参数说明

显示变量的参数请参见RESET的**参数说明**。

示例

```
--显示 timezone 参数值。
gaussdb=# SHOW timezone;

--显示所有参数。
gaussdb=# SHOW ALL;

--显示参数名中包含”var”的所有参数
gaussdb=# SHOW VARIABLES LIKE var;
```

相关链接

SET，**RESET**

7.14.177 SHOW EVENTS

功能描述

显示指定SCHEMA下所有定时任务的基本信息。

注意事项

定时任务相关操作只有sql_compatibility = 'B'时支持。

语法格式

```
SHOW EVENTS
  [{FROM | IN} schema_name]
  [LIKE 'pattern' | WHERE condition]
```

参数说明

- {FROM | IN}

  指定要查询的schema，默认情况下为当前schema。

- LIKE

  LIKE可以模式匹配定时任务名称，不指定则打印当前schema下所有定时任务。

- WHERE

  WHERE子句构成一个行选择表达式，用来缩小SHOW EVENTS查询的范围。condition是返回值为布尔型的任意表达式，任何不满足该条件的行都不会被检索。

示例

```
--查看event_a schema下的通过模式匹配'_e'查询出的所有定时任务的信息
gaussdb=#  SHOW EVENTS IN event_a LIKE '_e';
```

7.14.178 SHUTDOWN

功能描述

SHUTDOWN将关闭当前连接的数据库节点。

注意事项

仅拥有管理员权限的用户可以运行此命令。

语法格式

```
SHUTDOWN [FAST | IMMEDIATE];
```

参数说明

- **""**

  不指定关闭模式，默认为fast。

- **fast**

  不等待客户端中断连接，将所有活跃事务回滚并且强制断开客户端，然后关闭数据库节点。

- **immediate**

  强行关闭，在下次重新启动的时候将导致故障恢复。

示例

```
--关闭当前数据库节点。
gaussdb=# SHUTDOWN;

--使用fast模式关闭当前数据库节点。
gaussdb=# SHUTDOWN FAST;
```

7.14.179 SNAPSHOT

功能描述

针对多用户情况下，对数据进行统一的版本控制。

注意事项

- 本特性GUC参数db4ai_snapshot_mode，快照存储模型分为MSS和CSS两种；GUC参数db4ai_snapshot_version_delimiter，用于设定版本分隔符，仅接受设定单字节参数值，默认为“@”;GUC参数db4ai_snapshot_version_separator，用于设定子版本分隔符，仅接受设定单字节参数值，默认为“.”。
- 当快照选用增量存储方式时，各个快照中具有依赖关系。删除快照需要按照依赖顺序进行删除。
- snapshot特性用于团队不同成员间维护数据，涉及管理员和普通用户之间的数据转写。所以在三权分立(enableSeparationOfDuty=ON)等状态下，数据库不支持snapshot功能特性。

- 当需要稳定可用的快照用于AI训练等任务时，用户需要将快照发布。

语法格式

1. 创建快照

   可以采用“CREATE SNAPSHOT … AS”以及“CREATE SNAPSHOT … FROM”语句创建数据表快照。

   - CREATE SNAPSHOT AS

     ```
     CREATE SNAPSHOT qualified_name [@ [version]]
       [COMMENT IS comment_item]
       AS query;
     ```

   - CREATE SNAPSHOT FROM

     ```
     CREATE SNAPSHOT qualified_name [@ [version]]
       FROM @ version
       [COMMENT IS comment_item]
           USING (
           { INSERT [INTO SNAPSHOT] insert_item
             | UPDATE [SNAPSHOT] [AS alias] SET set_item [FROM from_item] [WHERE where_item]
             | DELETE [FROM SNAPSHOT] [AS alias] [USING using_item] [WHERE where_item]
             | ALTER [SNAPSHOT] { ADD and_item | DROP drop_item } [, ...]
         } [; ...]
       );
     ```

2. 删除快照。

   PURGE SNAPSHOT

   ```
   PURGE SNAPSHOT qualified_name @ version;
   ```

3. 快照采样。

   SAMPLE SNAPSHOT

   ```
   SAMPLE SNAPSHOT qualified_name @ version
     [STRATIFY BY attr_list]
     { AS alias AT RATIO num [COMMENT IS comment_item] } [, ...]
   ```

4. 快照发布。

   PUBLISH SNAPSHOT

   ```
   PUBLISH SNAPSHOT qualified_name @ version;
   ```

5. 快照存档。

   ARCHIVE SNAPSHOT

   ```
   ARCHIVE SNAPSHOT qualified_name @ version;
   ```

6. 查询快照。

   ```
   SELECT * FROM DB4AISHOT (qualified_name @ version );
   ```

参数说明

- **qualified_name**

  创建snapshot的名称。

  取值范围：字符串，需要符合**标识符命名规范**。

- **version**

  (可省略)snapshot的版本号，当省略设置。系统会自动顺延编号。

  取值范围：字符串，数字编号配合分隔符。

- **comment_item**

  指定添加的评论内容。

  取值范围：字符串，需要符合**标识符命名规范**。

- **insert_item**

需要插入的对象名字。
取值范围：字符串，需要符合**标识符命名规范**。

- **alias**
  对当前对象取的别名。
  取值范围：字符串，需要符合**标识符命名规范**。
- **set_item**
  当前操作的对象名。
  取值范围：字符串，需要符合**标识符命名规范**。
- **from_item**
  用于连接的查询源对象的名称。
  取值范围：字符串，需要符合**标识符命名规范**。
- **where_item**
  返回值为布尔型的任意表达式。
  取值范围：字符串，需要符合**标识符命名规范**。
- **using_item**
  用于连接的对象名称。
  取值范围：字符串，需要符合**标识符命名规范**。
- **and_item**
  需要并列处理的对象名称。
  取值范围：字符串，需要符合**标识符命名规范**。
- **drop_item**
  需要drop的对象名称。
  取值范围：字符串，需要符合**标识符命名规范**。
- **attr_list**
  目标对象的list集合。
  取值范围：字符串，需要符合**标识符命名规范**。
- **num**
  指定的比例值。
  取值范围：数字。

示例

```
——创建数据表
gaussdb=# CREATE TABLE t1 (id int, name varchar);

——插入数据
gaussdb=# INSERT INTO t1 VALUES (1, 'zhangsan');
gaussdb=# INSERT INTO t1 VALUES (2, 'lisi');
gaussdb=# INSERT INTO t1 VALUES (3, 'wangwu');
gaussdb=# INSERT INTO t1 VALUES (4, 'lisa');
gaussdb=# INSERT INTO t1 VALUES (5, 'jack');

——创建快照
gaussdb=# CREATE SNAPSHOT s1@1.0 comment is 'first version' AS SELECT * FROM t1;

——迭代创建快照
gaussdb=# CREATE SNAPSHOT s1@2.0 FROM @1.0 comment is 'inherits from @1.0' USING (INSERT
VALUES(6, 'john'), (7, 'tim'); DELETE WHERE id = 1);
```



```
——查看快照内容
gaussdb=#SELECT * FROM DB4AISHOT(s1@1.0);

——快照采样
gaussdb=# SAMPLE SNAPSHOT s1@2.0 stratify by name as nick at ratio .5;

——删除快照
gaussdb=# PURGE SNAPSHOT s1@2.0;
gaussdb=# PURGE SNAPSHOT s1nick@2.0;
gaussdb=# PURGE SNAPSHOT s1@1.0;

——删除表格
gaussdb=# DROP TABLE t1;
```

7.14.180 START TRANSACTION

功能描述

通过START TRANSACTION启动事务。如果声明了隔离级别、读写模式，那么新事务就使用这些特性，类似执行了**SET TRANSACTION**。

语法格式

格式一：START TRANSACTION格式

```
START TRANSACTION
 [
  {
    ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE | REPEATABLE READ }
    | { READ WRITE | READ ONLY }
  } [...]
 ];
```

格式二：BEGIN格式

```
BEGIN [ WORK | TRANSACTION ]
 [
  {
    ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE | REPEATABLE READ }
    | { READ WRITE | READ ONLY }
  } [...]
 ];
```

参数说明

- **WORK | TRANSACTION**

  BEGIN格式中的可选关键字，没有实际作用。

- **ISOLATION LEVEL**

  指定事务隔离级别，它决定当一个事务中存在其他并发运行事务时它能够看到什么数据。

  说明

  在事务中第一个数据修改语句（SELECT, INSERT，DELETE，UPDATE，FETCH，COPY）执行之后，事务隔离级别就不能再次设置。

  取值范围：

  - READ COMMITTED：读已提交隔离级别，只能读到已经提交的数据，而不会读到未提交的数据。这是缺省值。

  - REPEATABLE READ： 可重复读隔离级别，仅仅看到事务开始之前提交的数据，它不能看到未提交的数据，以及在事务执行期间由其它并发事务提交的修改。
  - SERIALIZABLE：GaussDB目前功能上不支持此隔离级别，等价于REPEATABLE READ。
- **READ WRITE | READ ONLY**

  指定事务访问模式（读/写或者只读）。

示例

```
--创建SCHEMA。
gaussdb=# CREATE SCHEMA tpcds;

--创建表tpcds.reason。
gaussdb=# CREATE TABLE tpcds.reason (c1 int, c2 int);

--以默认方式启动事务。
gaussdb=# START TRANSACTION;
gaussdb=# SELECT * FROM tpcds.reason;
gaussdb=# END;

--以默认方式启动事务。
gaussdb=# BEGIN;
gaussdb=# SELECT * FROM tpcds.reason;
gaussdb=# END;

--以隔离级别为READ COMMITTED，读/写方式启动事务。
gaussdb=# START TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
gaussdb=# SELECT * FROM tpcds.reason;
gaussdb=# COMMIT;

--删除表tpcds.reason。
gaussdb=# DROP TABLE tpcds.reason;

--删除SCHEMA。
gaussdb=# DROP SCHEMA tpcds;
```

相关链接

COMMIT | END，**ROLLBACK**，**SET TRANSACTION**

7.14.181 TIMECAPSULE TABLE

功能描述

在人为操作或应用程序错误时，使用TIMECAPSULE TABLE语句恢复可将表恢复到一个早期状态。

表可以闪回到过去的时间点，这依赖于系统中保存的旧版本数据。此外GaussDB数据库不能恢复到通过DDL操作改变了表结构的早期状态。

注意事项

- TIMECAPSULE TABLE语句的用法主要分为两大类：闪回旧版本数据和从回收站中闪回。
  - TO TIMECAPSULE和TO CSN能够将表闪回到过去的某个版本。
  - 回收站记录了DROP和TRUNCATE的对象数据。TO BEFORE DROP和TO BEFORE TRUNCATE就是从回收站中闪回。

- 不支持闪回表的对象类型：系统表、DFS表、全局临时表、本地临时表、UNLOGGED表、序列表、hashbucket表、密态表。
- 不支持含有自定义类型表的闪回。
- 开启闪回后，回收站里的表可以进行表级备份，无法进行表级恢复。
- 闪回点和当前点之间，执行过修改表结构或影响物理存储的语句（DDL、DCL、VACUUM FULL），闪回失败。
- 执行闪回删除需要用户具有如下权限：用户必须具有垃圾对象所在SCHEMA的CREATE和USAGE权限，并且用户必须是SCHEMA的所有者或者是垃圾对象的所有者。

  执行闪回TRUNCATE需要用户具有如下权限：用户必须具有垃圾对象所在SCHEMA的CREATE和USAGE权限，并且用户必须是SCHEMA的所有者或者是垃圾对象的所有者，另外用户必须具有垃圾对象的TRUNCATE权限。
- 不适用闪回DROP/TRUNCATE功能的场景或表：
  - 回收站关闭场景：enable_recyclebin = off；
  - 系统处于维护态（xc_maintenance_mode = on）或升级场景；
  - 多对象删除场景：DROP/TRUNCATE TABLE命令同时指定多个对象；
  - 系统表、DFS表、全局临时表、本地临时表、UNLOGGED表、序列表、hashbucket表。
  - 如果表依赖的对象为外部对象（如表列为复合类型、自定义类型等），则采用物理删除，不将表放入回收站。
- DROP闪回约束：

  可以指定原始用户指定的表的名称，或对象删除时数据库分配的系统生成名称。
  - 回收站中系统生成的对象名称是唯一的。因此，如果指定系统生成名称，那么数据库检索指定的对象。使用“select * from gs_recyclebin;”语句查看回收站中的内容。
  - 如果指定了用户指定的名称，且回收站中包含多个该名称的对象，那么数据库检索回收站中最近移动的对象。如果需要检索更早版本的表，请按以下步骤执行：
    - 指定需要检索的表的系统生成名称。
    - 执行TIMECAPSULE TABLE ... TO BEFORE DROP语句，直到找到要检索的表。
  - 恢复DROP表时，只恢复基表名，其他子对象名均保持回收站对象名。用户可根据需要，执行DDL命令手工调整子对象名。
  - 如果表存在缺省值引用序列和自定义函数，那么闪回DROP表成功但不会恢复缺省值。
  - 如果表存在视图引用，DROP表时需要级联删除视图，那么闪回DROP表成功但不会恢复视图。
  - 回收站对象不支持DML、DCL、DDL等写操作，不支持DQL查询操作（后续支持）。
  - recyclebin_retention_time配置参数用于设置回收站对象保留时间，超过该时间的回收站对象将被自动清理。
  - 不支持DROP多表的恢复。
  - 不支持级联删除账号/schema场景的恢复。

  - 删除账号/Schema时，若回收站中存在该Schema/账号的对象，普通删除会失败，需要级联删除。
- TRUNCATE闪回约束：
  - TRUNCATE闪回后，统计信息无变化，仍显示为0，可以在业务低峰期（以降低性能影响）的时候手动analyze来修正统计信息。
  - RENAME TO仅支持DROP闪回操作为检索表指定新名称，不支持TRUNCATE闪回。
  - TRUNCATE闪回不能跨越影响表结构或物理存储的语句，否则会报错。即闪回点和当前点之间，如果执行过修改表结构或影响物理存储的语句（DDL、DCL、VACUUM FULL、增加/删除/切割/合成等分区操作），则闪回失败。执行过DDL的表进行闪回操作报错：“ERROR：The table definition of %s has been changed.”。涉及namespace、表名改变等操作的DDL执行闪回操作报错：“ERROR: recycle object %s desired does not exist.”。

前提条件

- 开启enable_recyclebin参数，启用回收站，参数使用请联系管理员处理。
- recyclebin_retention_time参数用于设置回收站对象保留时间，超过该时间的回收站对象将被自动清理，参数使用请联系管理员处理。

语法格式

```
TIMECAPSULE TABLE [schema.]table_name TO { CSN expr | TIMESTAMP expr | BEFORE { DROP [RENAME TO table_name] | TRUNCATE } }
```

参数说明

- **schema**

  指定模式包含的表。如果缺省，则为当前模式。
- **table_name**

  指定表名。
- **TO CSN**

  指定要返回表的时间点对应的事务提交序列号（CSN）。expr必须计算一个数字，代表有效的CSN。
- **TO TIMESTAMP**

  指定要返回表的时间点对应的时间戳。expr必须计算一个过去有效的时间戳（使用TO_TIMESTAMP函数将字符串转换为时间类型）。表将被闪回到指定时间戳大约3秒内的时间点。

  说明：闪回点过旧时，因旧版本被回收导致无法获取旧版本，会导致闪回失败并报错：Restore point too old。
- **TO BEFORE DROP**

  使用这个子句检索回收站中已删除的表及其子对象。
- **RENAME TO**

  为从回收站中检索的表指定一个新名称。
- **TO BEFORE TRUNCATE**

  闪回到TRUNCATE之前。

示例

```
-- 创建SCHEMA
gaussdb=# CREATE SCHEMA tpcds;

-- 删除表tpcds.reason_t2
DROP TABLE IF EXISTS tpcds.reason_t2;
-- 创建表tpcds.reason_t2
gaussdb=# CREATE TABLE tpcds.reason_t2
 (
  r_reason_sk     integer,
  r_reason_id     character(16),
  r_reason_desc   character(100)
  )with(storage_type = ustore);
--向表tpcds.reason_t2中插入记录
gaussdb=#  INSERT INTO tpcds.reason_t2 VALUES (1, 'AA', 'reason1'),(2, 'AB', 'reason2'),(3, 'AC', 'reason3');
INSERT 0 3
--清空tpcds.reason_t2表中的数据
gaussdb=#  TRUNCATE TABLE tpcds.reason_t2;
--查询tpcds.reason_t2表中的数据
gaussdb=#  select * from tpcds.reason_t2;
 r_reason_sk | r_reason_id | r_reason_desc
-------------+-------------+---------------
(0 rows)
--执行闪回TRUNCATE
gaussdb=#  TIMECAPSULE TABLE tpcds.reason_t2 to BEFORE TRUNCATE;
gaussdb=#  select * from tpcds.reason_t2;
 r_reason_sk |   r_reason_id    |                                                r_reason_desc
-------------+------------------
+-----------------------------------------------------------------------------------------------------------
           1 | AA               | reason1
           2 | AB               | reason2
           3 | AC               | reason3
(3 rows)
--删除表tpcds.reason_t2
gaussdb=#  DROP TABLE tpcds.reason_t2;
--执行闪回DROP
gaussdb=#  TIMECAPSULE TABLE tpcds.reason_t2 to BEFORE DROP;
TimeCapsule Table
--执行闪回DROP，为检索表指定新名称
gaussdb=#  TIMECAPSULE TABLE tpcds.reason_t2 TO BEFORE DROP rename to reason_t3;
TimeCapsule Table
-- 清空回收站，删除SCHEMA
gaussdb=# PURGE RECYCLEBIN;
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

7.14.182 TRUNCATE

功能描述

清理表数据，TRUNCATE快速地从表中删除所有行。

它和在目标表上进行无条件的DELETE有同样的效果，但由于TRUNCATE不做表扫描，因而快得多。在大表上操作效果更明显。

注意事项

- TRUNCATE TABLE在功能上与不带WHERE子句DELETE语句相同：二者均删除表中的全部行。
- TRUNCATE TABLE比DELETE速度快且使用系统和事务日志资源少：
  - DELETE语句每次删除一行，并在事务日志中为所删除每行记录一项。
  - TRUNCATE TABLE通过释放存储表数据所用数据页来删除数据，并且只在事务日志中记录页的释放。

- TRUNCATE，DELETE，DROP三者的差异如下：
  - TRUNCATE TABLE，删除内容，释放空间，但不删除定义。
  - DELETE TABLE，删除内容，不删除定义，不释放空间。
  - DROP TABLE，删除内容和定义，释放空间。

语法格式

- 清理表数据。

```
TRUNCATE [ TABLE ] [ ONLY ] {table_name [ * ]} [, ... ]
    [ CONTINUE IDENTITY ] [ CASCADE | RESTRICT] [ PURGE ]};
```

- 清理表分区的数据。

```
ALTER TABLE [ IF EXISTS  ] { [ ONLY  ] table_name
                        | table_name *
                        | ONLY ( table_name )  }
    TRUNCATE PARTITION { partition_name
                    | FOR (  partition_value  [, ...] )  } [ UPDATE GLOBAL INDEX ];
```

参数说明

- **ONLY**

  如果声明ONLY，只有指定的表会被清空。如果没有声明ONLY，这个表以及其所有子表（若有）会被清空。

- **table_name**

  目标表的名称（可以有模式修饰）。

  取值范围：已存在的表名。

- **CONTINUE IDENTITY**

  不改变序列的值。这是缺省值。

- **CASCADE | RESTRICT**
  - CASCADE：级联清空所有由于CASCADE而被添加到组中的表。
  - RESTRICT（缺省值）：如果其他表在该表上有外键引用则拒绝清空。

- **PURGE**

  默认将表数据放入回收站中，PURGE直接清理。

- **partition_name**

  目标分区表的分区名。

  取值范围：已存在的分区名。

- **partition_value**

  指定的分区键值。

  通过PARTITION FOR子句指定的这一组值，可以唯一确定一个分区。

  取值范围：需要进行删除数据分区的分区键的取值范围。

  须知

  使用PARTITION FOR子句时，partition_value所在的整个分区会被清空。

- **UPDATE GLOBAL INDEX**

如果使用该参数，则会更新分区表上的所有全局索引，以确保使用全局索引可以查询出正确的数据；如果不使用该参数，则分区表上的所有全局索引将会失效。

示例

```
--创建SCHEMA。
gaussdb=# CREATE SCHEMA tpcds;

--创建表tpcds.reason。
gaussdb=# CREATE TABLE tpcds.reason
(
  r_reason_sk       integer,
  r_reason_id       character(16),
  r_reason_desc     character(100)
);

--向表中插入多条记录。
gaussdb=# INSERT INTO tpcds.reason values(1,'AAAAAAAABAAAAAAA','reason 1'),
(5,'AAAAAAAABAAAAAAA','reason 2'),(15,'AAAAAAAABAAAAAAA','reason 3'),
(25,'AAAAAAAABAAAAAAA','reason 4'),(35,'AAAAAAAABAAAAAAA','reason 5'),
(45,'AAAAAAAACAAAAAAA','reason 6'),(55,'AAAAAAAACAAAAAAA','reason 7');

--创建表。
gaussdb=# CREATE TABLE tpcds.reason_t1 AS TABLE tpcds.reason;

--清空表tpcds.reason_t1。
gaussdb=# TRUNCATE TABLE tpcds.reason_t1;

--删除表。
gaussdb=# DROP TABLE tpcds.reason_t1;
--创建分区表。
gaussdb=# CREATE TABLE tpcds.reason_p
(
  r_reason_sk integer,
  r_reason_id character(16),
  r_reason_desc character(100)
)PARTITION BY RANGE (r_reason_sk)
(
  partition p_05_before values less than (05),
  partition p_15 values less than (15),
  partition p_25 values less than (25),
  partition p_35 values less than (35),
  partition p_45_after values less than (MAXVALUE)
);

--插入数据。
gaussdb=# INSERT INTO tpcds.reason_p SELECT * FROM tpcds.reason;

--清空分区p_05_before。
gaussdb=# ALTER TABLE tpcds.reason_p TRUNCATE PARTITION p_05_before;

--清空分区p_15。
gaussdb=# ALTER TABLE tpcds.reason_p TRUNCATE PARTITION for (13);

--清空分区表。
gaussdb=# TRUNCATE TABLE tpcds.reason_p;

--删除表。
gaussdb=# DROP TABLE tpcds.reason_p;

--删除表。
gaussdb=# DROP TABLE tpcds.reason;

--删除SCHEMA。
gaussdb=# DROP SCHEMA tpcds CASCADE;
```


7.14.183 UPDATE

功能描述

更新表中的数据，UPDATE修改满足条件的所有行中指定的字段值，WHERE子句声明条件，SET子句指定的字段会被修改，没有出现的字段则保持它们的原值。

注意事项

- 表的所有者、拥有表UPDATE权限的用户或拥有UPDATE ANY TABLE权限的用户，有权更新表中的数据，系统管理员默认拥有此权限。
- 对expression或condition条件里涉及到的任何表要有SELECT权限。
- 生成列不能被直接写入。在UPDATE命令中不能为生成列指定值，但是可以指定关键字DEFAULT。
- 对于多表更新语法，暂时不支持对视图和含有RULE的表进行多表更新。
- 对于子查询是stream计划的UPDATE语句，不支持并发更新同一行。
- 不支持用户通过update系统表的方式将数据库编码更改为GB18030_2022字符集或将GB18030_2022字符集数据库更改为其他字符编码，进行更改数据库字符编码的操作会导致存量数据和部分操作异常。若需更改数据库的字符集编码，应当遵循切库流程，进行相关的数据迁移操作。

语法格式

```
单表更新：
[ WITH [ RECURSIVE ] with_query [, ...] ]
UPDATE [/*+ plan_hint */] [ ONLY ] table_name [ partition_clause ] [ * ] [ [ AS ] alias ]
SET {column_name = { expression | DEFAULT }
    |( column_name [, ...] ) = {( { expression | DEFAULT } [, ...] ) |sub_query }}[, ...]
    [ FROM from_list] [ WHERE condition | WHERE CURRENT OF cursor_name ]
    [ ORDER BY {expression [ [ ASC | DESC | USING operator ]
    [ LIMIT { count } ]
    [ RETURNING {*
                | {output_expression [ [ AS ] output_name ]} [, ...] }];

多表更新：
[ WITH [ RECURSIVE ] with_query [, ...] ]
UPDATE [/*+ plan_hint */] table_list
SET {column_name = { expression | DEFAULT }
    |( column_name [, ...] ) = {( { expression | DEFAULT } [, ...] ) |sub_query }}[, ...]
    [ FROM from_list] [ WHERE condition ];

where sub_query can be:
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
{ * | {expression [ [ AS ] output_name ]} [, ...] }
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY grouping_element [, ...] ]
[ HAVING condition [, ...] ]
[ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS { FIRST |
LAST } ]} [, ...] ]
[ LIMIT { [offset,] count | ALL } ]
```

- 其中子查询with_query为：

  ```
  with_query_name [ ( column_name [, ...] ) ]
      AS [ [ NOT ] MATERIALIZED ] ( {select | values | insert | update | delete} )
  ```


参数说明

- **WITH [RECURSIVE] with_query [, ...]**

  用于声明一个或多个可以在主查询中通过名称引用的子查询，相当于临时表。这种子查询语句结构称为CTE（Common Table Expression）结构，应用这种结构时，执行计划中将存在CTE SCAN的内容。

  如果声明了RECURSIVE，那么允许SELECT子查询通过名称引用它自己。

  其中with_query的详细格式为：

  ```
  with_query_name [ ( column_name [, ...] ) ] AS [ [ NOT ] MATERIALIZED ] ( {select | values | insert | update | delete} )
  ```

  - with_query_name指定子查询生成的结果集名称，在查询中可使用该名称访问子查询的结果集。
  - column_name指定子查询结果集中显示的列名。
  - 每个子查询可以是SELECT、VALUES、INSERT、UPDATE或DELETE语句。
  - 用户可以使用MATERIALIZED / NOT MATERIALIZED对CTE进行修饰。
    - 如果声明为MATERIALIZED，WITH查询将被物化，生成一个子查询结果集的拷贝，在引用处直接查询该拷贝，因此WITH子查询无法和主干SELECT语句进行联合优化（如谓词下推、等价类传递等），对于此类场景可以使用NOT MATERIALIZED进行修饰，如果WITH查询语义上可以作为子查询内联执行，则可以进行上述优化。
    - 如果用户没有显示声明物化属性则遵守以下规则：如果CTE只在所属SELECT主干中被引用一次，且语义上支持内联执行，则会被改写为子查询内联执行，否则以CTE Scan的方式物化执行。

- **plan_hint子句**

  以/*+ */的形式在UPDATE关键字后，用于对UPDATE对应的语句块生成的计划进行hint调优，详细用法请参见章节**使用Plan Hint进行调优**。每条语句中只有第一个/*+ plan_hint */注释块会作为hint生效，里面可以写多条hint。

- **table_name**

  要更新的表名，可以使用模式修饰。

  取值范围：已存在的表名称。

  说明

  支持使用DATABASE LINK方式对远端表进行操作，使用方式详情请见**DATABASE LINK**。

- **partition_clause**

  指定分区更新操作

  PARTITION { (partition_name) | FOR (partition_value [, ...]) } |

  SUBPARTITION { (subpartition_name) | FOR (subpartition_value [, ...]) }

  关键字详见**SELECT**一节介绍

  示例详见**CREATE TABLE SUBPARTITION**

- **alias**

  目标表的别名。

  取值范围：字符串，符合**标识符命名规范**。

- **table_list**

  一个表的表达式列表，与from_list类似，但可以同时声明目标表和关联表，仅在多表更新语法中使用。

- **column_name**

  要修改的字段名。

  支持使用目标表的别名加字段名来引用这个字段。例如：

  UPDATE foo AS f SET f.col_name = 'namecol';

  取值范围：已存在的字段名。

- **expression**

  赋给字段的值或表达式。

- **DEFAULT**

  用对应字段的缺省值填充该字段。

  如果没有缺省值，则为NULL。

- **sub_query**

  子查询。

  使用同一数据库里其他表的信息来更新一个表可以使用子查询的方法。其中SELECT子句具体介绍请参考**SELECT**。

  在UPDATE单列时，支持使用ORDER BY子句与LIMIT子句；而在UPDATE多列时，则不支持使用ORDER BY子句与LIMIT子句。

  说明

  对于UPDATE t1 SET (c1,c2) = (SELECT c1, c2 FROM t2 ...)形式的UPDATE语句，在执行计划中，对于每一个字段，会生成一个子计划。当更新字段数较多时，子计划数量较多，对性能影响较大。

- **from_list**

  一个表的表达式列表，允许在WHERE条件里使用其他表的字段。与在一个SELECT语句的FROM子句里声明表列表类似。

  须知

  目标表不能出现在from_list里，除非在使用一个自连接（此时它必须以from_list的别名出现）。

- **condition**

  一个返回Boolean类型结果的表达式。只有这个表达式返回true的行才会被更新。不建议使用int等数值类型作为condition，因为int等数值类型可以隐式转换为Boolean值（非0值隐式转换为true，0转换为false），可能导致非预期的结果。

- **WHERE CURRENT OF cursor_name**

  当cursor指向表的某一行时，可以使用此语法更新cursor当前指向的行。

  cursor_name：指定游标的名称。

须知

- B兼容模式的数据库不支持使用此语法。
- 此语法仅支持普通表，不支持分区表。
- 仅支持在存储过程中使用。
- 不支持与其他WHERE条件组合使用。
- 不支持多表更新。
- 不支持与WITH、USING、ORDER BY、FROM组合使用。
- CURSOR对应的SELECT语句必须声明为FOR UPDATE。
- CURSOR对应的SELECT语句仅支持单表，不支持LIMIT/OFFSET，不支持带有子查询、子链接。
- 存储过程中声明为FOR UPDATE的CURSOR，在COMMIT/ROLLBACK后，将无法再次使用。
- 若CURSOR指向的行已经不存在，在A兼容性模式下将报错指定的行不存在（仅UPDATE时报错，DELETE不报错），其他兼容模式下不报错。

- ORDER BY子句
  关键字详见**SELECT**章节介绍。
- LIMIT子句
  关键字详见**SELECT**章节介绍。
- **output_expression**
  在所有需要更新的行都被更新之后，UPDATE命令用于计算返回值的表达式。
  取值范围：使用任何TABLE以及FROM中列出的表的字段。*表示返回所有字段。
- **output_name**
  字段的返回名称。

示例

```
--创建表student1。
gaussdb=# CREATE TABLE student1
gaussdb=#(
gaussdb=#   stuno     int,
gaussdb=#   classno   int
gaussdb=#);

--插入数据。
gaussdb=# INSERT INTO student1 VALUES(1,1);
gaussdb=# INSERT INTO student1 VALUES(2,2);
gaussdb=# INSERT INTO student1 VALUES(3,3);

--查看数据。
gaussdb=# SELECT * FROM student1;

--直接更新所有记录的值。
gaussdb=# UPDATE student1 SET classno = classno*2;

--查看数据。
gaussdb=# SELECT * FROM student1;

--删除表。
gaussdb=# DROP TABLE student1;

--WHERE CURRENT OF cursor_name用例
```

```
gaussdb=# CREATE TABLE t1(c1 int, c2 varchar2) WITH (storage_type = 'ustore', fillfactor=100); -- 创建表
gaussdb=# INSERT INTO t1 VALUES(generate_series(1,1000),'abcd'); -- 插入数据

gaussdb=# DECLARE
gaussdb-# CURSOR cur1 IS SELECT * FROM t1 WHERE c1 = 1 FOR UPDATE;
gaussdb-# va t1%rowtype;
gaussdb-# BEGIN
gaussdb$# OPEN cur1;
gaussdb$# FETCH cur1 INTO va;
gaussdb$# UPDATE t1 SET c2 = c2 || c2  WHERE CURRENT OF cur1; -- 使用WHERE CURRENT OF
cursor_name更新数据
gaussdb$# CLOSE cur1;
gaussdb$# COMMIT;
gaussdb$# END;
gaussdb$# /

gaussdb=# SELECT * FROM t1 WHERE c1 = 1; -- 查询数据

gaussdb=# DROP TABLE t1; --删除表
```

7.14.184 VACUUM

功能描述

VACUUM回收表或B-Tree索引中已经删除的行所占据的存储空间。在一般的数据库操作里，那些已经DELETE的行并没有从它们所属的表中物理删除；在完成VACUUM之前它们仍然存在。因此有必要周期地运行VACUUM，特别是在经常更新的表上。

注意事项

- 如果没有参数，VACUUM处理当前数据库里用户拥有相应权限的每个表。如果参数指定了一个表，VACUUM只处理指定的那个表。
- 要对一个表进行VACUUM操作，通常用户必须是表的所有者或者被授予了指定表VACUUM权限的用户，默认系统管理员有该权限。数据库的所有者允许对数据库中除了共享目录以外的所有表进行VACUUM操作（该限制意味着只有系统管理员才能真正对一个数据库进行VACUUM操作）。VACUUM命令会跳过那些用户没有权限的表进行垃圾回收操作。
- VACUUM不能在事务块内执行。
- 建议生产数据库经常清理（至少每晚一次），以保证不断地删除失效的行。尤其是在增删了大量记录之后，对受影响的表执行VACUUM ANALYZE命令是一个很好的习惯。这样将更新系统目录为最近的更改，并且允许查询优化器在规划用户查询时有更优选择。
- 不建议日常使用FULL选项，但是可以在特殊情况下使用。例如在用户删除了一个表的大部分行之后，希望从物理上缩小该表以减少磁盘空间占用。VACUUM FULL通常要比单纯的VACUUM收缩更多的表尺寸。FULL选项并不清理索引，所以推荐周期性的运行**REINDEX**命令。如果执行此命令后所占用物理空间无变化（未减少），请确认是否有其他活跃事务（删除数据事务开始之前开始的事务，并在VACUUM FULL执行前未结束）存在，如果有等其他活跃事务退出进行重试。
- VACUUM会导致I/O流量的大幅增加，这可能会影响其他活动会话的性能。因此，有时候会建议使用基于开销的VACUUM延迟特性。
- 如果指定了VERBOSE选项，VACUUM将打印处理过程中的信息，以表明当前正在处理的表。各种有关当前表的统计信息也会打印出来。
- 当含有带括号的选项列表时，选项可以以任何顺序写入。如果没有括号，则选项必须按语法显示的顺序给出。

- VACUUM和VACUUM FULL时，会根据参数vacuum_defer_cleanup_age延迟清理行存表记录，即不会立即清理刚刚删除的元组。
- VACUUM ANALYZE先执行一个VACUUM操作，然后给每个选定的表执行一个ANALYZE。对于日常维护脚本而言，这是一个很方便的组合。
- 简单的VACUUM（不带FULL选项）只是简单地回收空间并且令其可以再次使用。这种形式的命令可以和对表的普通读写并发操作，因为没有请求排他锁。
  VACUUM FULL执行更广泛的处理，包括跨块移动行，以便把表压缩到最少的磁盘块数目里。这种形式要慢许多并且在处理的时候需要在表上施加一个排他锁。
- 同时执行多个VACUUM FULL可能出现死锁。
- 如果没有打开xc_maintenance_mode参数，那么VACUUM FULL操作将跳过所有系统表。
- 执行DELETE后立即执行VACUUM FULL命令不会回收空间。执行DELETE后再执行1000个非SELECT事务，或者等待1s后再执行1个事务，之后再执行VACUUM FULL命令空间才会回收。
- VACUUM FULL期间会对表加排他锁，不建议在业务高峰期运行VACUUM FULL，否则会导致等锁时间过长或者死锁。

语法格式

- 回收空间并更新统计信息，对关键字顺序无要求。

```
VACUUM [ ( { FULL | FREEZE | VERBOSE | {ANALYZE | ANALYSE }} [,...] ) ]
    [ table_name [ (column_name [, ...] ) ] [ PARTITION ( partition_name ) | SUBPARTITION
( subpartition_name ) ] ];
```

- 仅回收空间，不更新统计信息。

```
VACUUM [ FULL [COMPACT] ] [ FREEZE ] [ VERBOSE ] [ table_name
[ PARTITION ( partition_name ) | SUBPARTITION ( subpartition_name ) ] ];
```

- 回收空间并更新统计信息，且对关键字顺序有要求。

```
VACUUM [ FULL ] [ FREEZE ] [ VERBOSE ] { ANALYZE | ANALYSE } [ VERBOSE ]
    [ table_name [ (column_name [, ...] ) ] ] [ PARTITION ( partition_name ) ];
```

参数说明

- **FULL**

  选择“FULL”清理，这样可以恢复更多的空间，但是需要耗时更多，并且在表上施加了排他锁。

  说明

  使用FULL参数会导致统计信息丢失，如果需要收集统计信息，请在VACUUM FULL语句中加上analyze关键字。

- **FREEZE**

  指定FREEZE相当于执行VACUUM时将vacuum_freeze_min_age参数设为0。

- **VERBOSE**

  为每个表打印一份详细的清理工作报告。

- **ANALYZE | ANALYSE**

  更新用于优化器的统计信息，以决定执行查询的最有效方法。

  说明

  ustore分区表在autovacuum=analyze的时候也会触发vacuum。

- **table_name**

要清理的表的名称（可以有模式修饰）。

取值范围：要清理的表的名称。缺省时为当前数据库中的所有表。

- **column_name**

  要分析的具体的字段名称，需要配合analyze选项使用。

  取值范围：要分析的具体的字段名称。缺省时为所有字段。

  说明

  由于vacuum analyze语句的机制是依次执行vacuum和analyze，因此当column_name错误时存在vacuum执行成功但analyze执行失败的情况，对于分区表，则会出现对某个分区成功vacuum之后analyze失败的情况。

- **PARTITION**

  COMPACT和PARTITION参数不能同时使用。

- **partition_name**

  要清理的表的一级分区名称。缺省时为所有一级分区。

- **subpartition_name**

  要清理的表的二级分区名称。缺省时为所有二级分区。

示例

```
--创建SCHEMA。
gaussdb=# CREATE SCHEMA tpcds;

--创建表tpcds.reason。
gaussdb=# CREATE TABLE tpcds.reason
(
  r_reason_sk       integer,
  r_reason_id       character(16),
  r_reason_desc     character(100)
);

--向表中插入多条记录。
gaussdb=# INSERT INTO tpcds.reason values(1,'AAAAAAAABAAAAAAA','reason 1'),
(2,'AAAAAAAABAAAAAAA','reason 2');

--在表tpcds.reason上创建索引。
gaussdb=# CREATE UNIQUE INDEX ds_reason_index1 ON tpcds.reason(r_reason_sk);

--对带索引的表tpcds.reason执行VACUUM操作。
gaussdb=# VACUUM (VERBOSE, ANALYZE) tpcds.reason;

--删除索引。
gaussdb=# DROP INDEX ds_reason_index1 CASCADE;
gaussdb=# DROP TABLE tpcds.reason;
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

优化建议

VACUUM

- VACUUM不能在事务块内执行。
- 建议生产数据库经常清理（至少每晚一次），以保证不断地删除失效的行。尤其是在增删了大量记录后，对相关表执行VACUUM ANALYZE命令。
- 不建议日常使用FULL选项，但是可以在特殊情况下使用。例如，一个例子就是在用户删除了一个表的大部分行之后，希望从物理上缩小该表以减少磁盘空间占用。

7.14.185 VALUES

功能描述

根据给定的值表达式计算一个或一组行的值。它通常用于在一个较大的命令内生成一个“常数表”。

注意事项

- 应当避免使用VALUES返回数量非常大的结果行，否则可能会出现内存耗尽或者性能低下的情况。出现在INSERT中的VALUES是一个特殊情况，因为目标字段类型可以从INSERT的目标表获知，并不需要通过扫描VALUES列表来推测，所以在此情况下可以处理非常大的结果行。
- 如果指定了多行，那么每一行都必须拥有相同的元素个数。

语法格式

```
VALUES {( expression [, ...] )} [, ...]
    [ ORDER BY { sort_expression [ ASC | DESC | USING operator ] } [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start [ ROW | ROWS ] ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ];
```

参数说明

- **expression**

  用于计算或插入结果表指定地点的常量或者表达式。

  在一个出现在INSERT顶层的VALUES列表中，expression可以被DEFAULT替换以表示插入目的字段的缺省值。除此以外，当VALUES出现在其他场合的时候是不能使用DEFAULT的。

- **sort_expression**

  一个表示如何排序结果行的表达式或者整数常量。

- **ASC**

  指定按照升序排列。

- **DESC**

  指定按照降序排列。

- **operator**

  一个排序操作符。

- **count**

  返回的最大行数。

- **OFFSET start [ROW | ROWS]**

  声明返回的最大行数，而start声明开始返回行之前忽略的行数。

- **FETCH { FIRST | NEXT } [count] { ROW | ROWS } ONLY**

  FETCH子句限定返回查询结果从第一行开始的总行数，count的缺省值为1。

示例

请参见INSERT的**示例**。

7.14.186 ALTER EXTENSION

须知

扩展功能为内部使用功能，不建议用户使用。

功能描述

修改插件扩展。

注意事项

ALTER EXTENSION 修改一个已安装的扩展的定义，要使用该接口，需要设置**support_extended_features**为true。这里有几种方式：

- UPDATE

  这种方式更新这个扩展到一个新的版本。这个扩展必须满足一个适用的更新脚本(或者一系列脚本) 这样就能修改当前安装版本到一个要求的版本。

- SET SCHEMA

  这种方式移动扩展对象到另一个模式。这个扩展必须relocatable才能使命令成功。

- ADD member_object

  这种方式添加一个已存在对象到扩展。这主要在扩展更新脚本上有用。这个对象接着会被视为扩展的成员; 显而易见,该对象只能通过取消扩展来取消 。

- DROP member_object

  这个方式从扩展上移除一个成员对象。主要在扩展更新脚本上有用 .这个对象没有被取消, 只是从扩展里分开了。

  您必须拥有扩展来使用 ALTER EXTENSION。 这个 ADD/DROP 方式要求 添加/删除对象的所有权。

语法格式

- 修改扩展的版本

```
ALTER EXTENSION name UPDATE [ TO new_version ];
```

- 修改扩展的模式

```
ALTER EXTENSION name SET SCHEMA new_schema;
```

- 添加或删除扩展的成员对象

```
ALTER EXTENSION name { ADD | DROP } member_object;
```

其中成员对象member_object写法为：

```
AGGREGATE agg_name (agg_type [, ...] ) |
CAST (source_type AS target_type) |
COLLATION object_name |
CONVERSION object_name |
DOMAIN object_name |
EVENT TRIGGER object_name |
FOREIGN DATA WRAPPER object_name |

FUNCTION function_name ( [ [ argname ] [ argmode ] argtype [, ...] ] ) |
MATERIALIZED VIEW object_name |
```



```
OPERATOR operator_name (left_type, right_type) |
OPERATOR CLASS object_name USING index_method |
OPERATOR FAMILY object_name USING index_method |
[ PROCEDURAL ] LANGUAGE object_name |
SCHEMA object_name |
SEQUENCE object_name |
SERVER object_name |
TABLE object_name |
TEXT SEARCH CONFIGURATION object_name |
TEXT SEARCH DICTIONARY object_name |
TEXT SEARCH PARSER object_name |
TEXT SEARCH TEMPLATE object_name |
TYPE object_name |
VIEW object_name
```

参数说明

- **name**

  已安装扩展的名称。

- **new_version**

  扩展的新版本。可以通过被标识符和字面字符重写。如果不指定的扩展的新版本，ALTER EXTENSION UPDATE会更新到扩展的控制文件中显示的默认版本。

- **new_schema**

  扩展的新模式。

- **object_name**

  agg_name

  function_name

  operator_name

  从扩展里被添加或移除的对象的名称。包含表、聚合 、域、外链表、函数、操作符、操作符类、操作符族、序列、文本搜索对象、类型和能被模式合格的视图的名称。

- **agg_type**

  在聚合函数操作上的一个输入数据类型，去引用一个零参数聚合函数，写 * 代替这些输入数据类型列表。

- **source_type**

  强制转换的源数据类型的名称。

- **target_type**

  强制转换的目标数据类型的名称。

- **argmode**

  这个函数参数的模型：IN、OUT、INOUT或者 VARIADIC。如果省略的话，默认值为IN。ALTER EXTENSION 不关心OUT参数 ，因为确认函数的一致性只需要输入参数，因此列出 IN、INOUT和 VARIADIC参数就足够了。

- **argname**

  函数参数的名称。ALTER EXTENSION不关心参数名称，确认函数的一致性只需要参数数据类型。

- **argtype**

  函数参数的数据类型（可以有模式修饰）。

- **left_type**

  right_type

操作符参数的数据类型（可以有模式修饰），为前缀或后缀运算符的丢失参数写NONE 。

示例

更新 plpgsql 扩展到版本 2.0：

```
ALTER EXTENSION plpgsql UPDATE TO '2.0';
```

更新 plpgsql 扩展的模式为utils：

```
ALTER EXTENSION plpgsql SET SCHEMA utils;
```

添加一个已存在的函数给 plpgsql 扩展：

```
ALTER EXTENSION plpgsql ADD FUNCTION populate_record(anyelement, plpgsql);
```

7.14.187 CREATE EXTENSION

须知

扩展功能为内部使用功能，不建议用户使用。

功能描述

安装一个扩展。

注意事项

- 在使用CREATE EXTENSION载入扩展到数据库中之前，必须先安装好该扩展的支持文件。
- CREATE EXTENSION命令安装一个新的扩展到一个数据库中，必须保证没有同名的扩展已经被安装。
- 安装一个扩展意味着执行一个扩展的脚本文件，这个脚本会创建一个新的SQL实体，例如函数、数据类型、操作符和索引支持的方法。
- 安装扩展需要有和创建他的组件对象相同的权限。对于大多数扩展这意味着需要超户或者数据库所有者的权限，对于后续的权限检查和该扩展脚本所创建的实体，运行CREATE EXTENSION命令的角色将变为扩展的所有者。
- CREATE EXTENSION时如果数据库中存在与EXTENSION内同名的PACKAGE、同义词、操作符、目录、函数、存储过程、视图、表这些数据库对象，将会导致CREATE EXTENSION失败。
- 数据库禁止直接创建扩展，因为扩展可能会引起非预期的错误，且在升级后面临不兼容的问题。如果需要创建扩展，需要设置**enable_extension**为true才能够创建。
- CREATE EXTENSION时，如果enable_object_special_character值为off，且扩展的脚本文件中使用"@extschema@"，那么扩展的支持文件中schema参数的值不能包含["$'\]中任意特殊字符。

语法格式

```
CREATE EXTENSION [ IF NOT EXISTS ] extension_name
[ WITH ] [ SCHEMA schema_name ]
```

```
[ VERSION version ]
[ FROM old_version ]
```

参数说明

- **IF NOT EXISTS**

  如果系统已经存在一个同名的扩展，不会报错。这种情况下会给出一个提示。请注意该参数不保证系统存在的扩展和现在脚本创建的扩展相同。

- **extension_name**

  将被安装扩展的名字，数据库将使用文件SHAREDIR/extension/extension_name.control中的详细信息创建扩展。

- **schema_name**

  扩展的实例被安装在该模式下，扩展的内容可以被重新安装。指定的模式必须已经存在，如果没有指定，扩展的控制文件也不指定一个模式，这样将使用默认模式。

  ⚠ 注意

  扩展不认为它在任何模式里面：扩展在一个数据库范围内的名字是不受限制的，但是一个扩展的实例是属于一个模式的。

- **version**

  安装扩展的版本，可以写为一个标识符或者字符串.默认的版本在扩展的控制文件中指定。

- **old_version**

  如果需要升级安装old style模块中没有的内容时，必须指定FROM old_version。这个选项使CREATE EXTENSION运行一个安装脚本，将新的内容安装到扩展中，而不是创建一个新的实体。注意：SCHEMA指定了包括这些已存在实体的模式。

示例

在当前数据库安装扩展。例如安装testxxx：

```
CREATE EXTENSION testxxx;
```

7.14.188 DROP EXTENSION

须知

扩展功能为内部使用功能，不建议用户使用。

功能描述

删除一个扩展。

注意事项

- DROP EXTENSION 命令从数据库中删除一个扩展。 在删除扩展的过程中，构成扩展的组件也会一起删除。

- 必须是扩展的拥有者才能够使用DROP EXTENSION命令。

语法格式

```
DROP EXTENSION [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

参数说明

- **IF EXISTS**

  当使用IF EXISTS参数，如果扩展不存在时，不会抛出错误，而是产生一个通知。

- **name**

  已经安装的扩展模块的名称。

- **CASCADE**

  自动删除依赖于该扩展的对象。

- **RESTRICT**

  如果有依赖于扩展的对象，则不允许删除此扩展（除非其所有的成员对象和其他扩展对象在一条 DROP命令一起删除）。 这是缺省行为。

示例

从当前数据库中删除扩展plpgsql

```
DROP EXTENSION plpgsql;
```

在当前数据库中，如果有使用plpgsql的对象的，这条命令就会失败，比如 任一表中的字段使用plpgsql类型。这时增加CASCADE选项会强制删除扩展和 依赖于扩展的对象。

7.15 附录

7.15.1 扩展函数

下表列举了GaussDB中支持的扩展函数，不作为商用特性交付，仅供参考。

分类	函数名称	描述
访问权限查询函数	has_sequence_privilege(user, sequence, privilege)	指定用户是否有访问序列的权限
	has_sequence_privilege(sequence, privilege)	当前用户是否有访问序列的权限
触发器函数	pg_get_triggerdef(oid)	为触发器获取CREATE [CONSTRAINT] TRIGGER命令
	pg_get_triggerdef(oid, boolean)	为触发器获取CREATE [CONSTRAINT] TRIGGER命令

7.15.2 扩展语法

GaussDB提供的扩展语法如下。

表 7-151 扩展 SQL 语法

类别	语法关键字	描述
创建表CREATE TABLE	column_constraint: **REFERENCES reftable [(refcolumn)] [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE][ON DELETE action] [ON UPDATE action]**	支持用REFERENCES reftable[(refcolumn)] [MATCH FULL |MATCH PARTIAL | MATCH SIMPLE] [ON DELETE action] [ON UPDATE action] 为表创建外键约束。
加载模块	**CREATE EXTENSION**	把一个新的模块加载进当前数据库中。该特性为内部使用，不建议用户使用。
	DROP EXTENSION	删除已加载的模块。该特性为内部使用，不建议用户使用。
聚集函数	**CREATE AGGREGATE**	定义一个新的聚集函数。
	ALTER AGGREGATE	修改一个聚集函数的定义。
	DROP AGGREGATE	删除一个现存的聚集函数。

7.15.3 美元引用的字符串常量

如果在字符串序列中包含有'(单引号)，那么应当将'(单引号)加倍为''(两个单引号)否则sql语句很可能无法执行。

如果字符串中包含很多单引号或者反斜杠，那么理解字符串的内容可能就会变得很苦涩，并且容易出错，因为单引号都要加倍。

为了让这种场合下的查询更具可读性，允许另外一种称作"美元符界定"的字符串常量书写办法。一个通过美元符界定声明的字符串常量由一个美元符号($)、零个或多个字符组成的"记号"、另一个美元符号、组成字符串常量的任意字符序列、一个美元符号、与前面相同的记号、一个美元符号组成的。

```
gaussdb=# SELECT $$it's an example$$;
   ?column?
-----------------
 it's an example
(1 row)
```


7.15.4 DATABASE LINK

功能描述

在本地数据库利用DATABASE LINK与远程数据库建立连接，并通过DATABASE LINK对远程数据库进行访问。

DATABASE LINK可以分为public或private，private DATABASE LINK仅能被创建者访问，而当DATABASE LINK为public时则所有用户都能访问。

所有已创建的DATABASE LINK信息都存在本地数据库的系统视图gs_db_links中。

注意事项

- DATABASE LINK特性只在A兼容版本下可以使用。
- DATABASE LINK连接的远端数据库仅支持503.1及之后版本。
- 用户需要保证本地和远端数据库的兼容性参数DBCOMPATIBILITY和GUC参数behavior_compat_options、a_format_dev_version、a_format_version取值一致。
- DATABASE LINK连接开启session时会设置如下GUC参数。

  ```
  set search_path=pg_catalog, '$user', 'public';
  set datestyle=ISO;
  set intervalstyle=postgres;
  set extra_float_digits=3;
  ```

  其他参数为远端设置的参数，远端参数与本地参数不同时，可能会出现数据显示格式不一致等情况，使用时应尽量保证远端与本地参数相同。
- 使用前置准备：使用gs_guc在pg_hba.conf文件中添加白名单允许客户端连接。

  ```
  示例：gs_guc reload -I all -N all -Z datanode -h "host all all 192.168.11.11/32 sha256"
  ```

  详细配置参数信息参考gs_guc客户端认证策略设置。
- 创建DATABASE LINK权限需要使用GRANT语法赋予，新建用户默认无权限，系统管理员拥有权限。详见GRANT相关说明。
- 使用DATABASE LINK对远端表操作时，会在本地创建与远端对应的Schema，若本地不存在该表的元数据信息，会将元数据信息写入本地系统表中，此时会使用7级锁保证写入的一致性，持续到事务结束放锁，删除DATABASE LINK时会将相应的元数据信息删除。
- 使用DATABASE LINK时在本地创建的表仅用于存储远端表的元数据信息，无法通过\d或pg_get_tabledef函数查询到表结构。
- 如果业务中有长事务首次使用dblink操作远端对象，会持续持锁直到事务结束，其他首次使用dblink的事务会被阻塞。可通过一条快速执行的语句先对要使用的远端对象做查询操作使其元数据落盘来规避这种情况，如 "select * from t1@dblink where 1=2;"。另外，远端表结构发生变化时本地要更新存储的元数据信息，也会有类似情况。
- 如果本地与远端字符集不同，可能会出现无法转换的报错，报错信息为远端返回报错。当本地数据库字符编码为gb18030_2022时，发送到远端会被转换为gb18030。因此，若本地数据库的字符集为GB18030_2022时，远程数据库字符集只能是GB18030或GB18030_2022。
- 如果在创建DATABASE LINK对象后重新生成密钥文件，使用DATABASE LINK时将会报错。因为创建DATABASE LINK时使用的密钥文件与后续解密时使用的密钥文件不一致。
- 在本地创建与远端对应的SCHEMA时会使用“USERNAME（私有DATABASE LINK才有）#远端SCHEMA@DBLINK名”作为SCHEMA名，名称长度上限为63。

须知

当赋予用户创建DATABASE LINK权限时，相当于许可用户使用服务端DATABASE的IP对远端进行访问。若不希望有此效果，应不要使用GRANT对用户赋权。

语法格式

- 创建DATABASE LINK。
```
CREATE [ PUBLIC ] DATABASE LINK dblink
 [ CONNECT TO { CURRENT_USER | user IDENTIFIED BY password } ] [ USING ( option 'value' [...]) ];
```
- 修改DATABASE LINK信息。
```
ALTER [ PUBLIC ] DATABASE LINK dblink
 { CONNECT TO user IDENTIFIED BY password };
```
- 删除指定DATABASE LINK。
```
DROP [ PUBLIC ] DATABASE LINK dblink ;
```

说明

- **PUBLIC**：指定公共以创建对所有用户可见的公共数据库链接。如果省略此子句，则数据库链接是私有的，仅对当前用户可用。远程数据库上可访问的数据取决于数据库链接在连接到远程数据库时使用的标识。
- 如果指定CONNECT TO user IDENTIFIED BY password，则数据库链接将使用指定的用户和密码连接。
- 如果指定CONNECT TO CURRENT_USER，则数据库连接将以本地初始用户的身份连接到远程数据库。
- 如果忽略了上述两个子句，则数据库连接将以本地初始用户的身份连接到远程数据库。
- **dblink**：要创建的DATABASE LINK的名字。
- **user**：创建的DATABASE LINK使用的用户名。
- **password**：用户名对应的密码。
- **USING (option 'value' [, ...])**

  USING 可选择指定要连接的数据库的IP地址、端口号、远端的database name等参数，支持的options包括：
  - host：指定连接的地址，不支持ipv6地址。支持以‘,’分割的字符串来指定多个IP地址，当前不支持密态数据库和ssl设置和证书认证，不指定默认为空。
  - port：指定连接的端口号，不指定默认为5432。
  - dbname：指定连接的数据库名称，不指定默认为连接远端使用的用户名。
  - fetch_size：从远端每次获取数据量大小，fetch_size取值为0到2147483647，默认为100。

  须知:
  - USING后的括号可以只选择上述关键字中的一部分去写。
  - USING关键字也可以不写，同时之后的括号也不要再写。
  - DATABASE LINK创建的时候不会去验证是否能连接成功，如果缺乏相关的关键字，可能会在使用时报错。
- 通过DATABASE LINK进行select操作。

说明

使用建立好的DATABASE LINK对远程数据库对象进行访问的语法和访问本地对象的语法基本一致，区别在于，在被访问的远程对象描述符后加@dblink。SQL语句具体支持情况有一些约束，详见**表7-152**。

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
 SELECT [/*+ plan_hint */] [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
```



```
{ * | {expression [ [ AS ] output_name ]} [, ...] }
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ [ START WITH condition ] CONNECT BY [NOCYCLE] condition [ ORDER SIBLINGS BY expression ] ]
[ GROUP BY grouping_element [, ...] ]
[ HAVING condition [, ...] ]
[ { UNION | INTERSECT | EXCEPT | MINUS } [ ALL | DISTINCT ] select ]
[ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS
{ FIRST | LAST } ]} [, ...] ]
[ LIMIT { [offset,] count | ALL } ]
[ OFFSET start [ ROW | ROWS ] ]
[ {FOR { UPDATE | SHARE} [ OF table_name [, ...] ] } [...] ];
{[ ONLY ] table_name [ * ] [ partition_clause ] @ dblink  [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
 |( select ) [ AS ] alias [ ( column_alias [, ...] ) ]
 |with_query_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
 |[function_name] ( [ argument [, ...] ] ) [ AS ] alias [ ( column_alias [, ...] | column_definition [, ...] ) ]
 |[function_name] ( [ argument [, ...] ] ) AS ( column_definition [, ...] )
 |from_item [ NATURAL ] join_type from_item [ ON join_condition | USING ( join_column [, ...] ) ]};
```

- 通过DATABASE LINK进行INSERT操作。

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
 INSERT [/*+ plan_hint */] INTO table_name [ partition_clause ] @ dblink [ ( column_name [, ...] ) ]
    { DEFAULT VALUES
    | VALUES {( { expression | DEFAULT } [, ...] ) }[, ...]
    | query }
    [ RETURNING { {output_expression [ [ AS ] output_name ] }[, ...]} ];
```

- 通过DATABASE LINK进行UPDATE操作。

```
UPDATE [/*+ plan_hint */] [ ONLY ] table_name [ partition_clause ] @ dblink  [ [ AS ] alias ]
 SET {column_name = { expression | DEFAULT }
   |( column_name [, ...] ) = {( { expression | DEFAULT } [, ...] ) |sub_query }}[, ...]
   [ FROM from_list ] [ WHERE condition ]
   [ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS
{ FIRST | LAST } ]} [, ...] ]
   [ LIMIT { [offset,] count | ALL } ]
   [ RETURNING { {output_expression [ [ AS ] output_name ]} [, ...] }];
 where sub_query can be:
 SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
 { * | {expression [ [ AS ] output_name ]} [, ...] }
 [ FROM from_item [, ...] ]
 [ WHERE condition ]
 [ GEOUP BY grouping_element [, ...] ]
 [ HAVING condition [, ...] ];
```

- 通过DATABASE LINK进行DELETE操作。

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
DELETE [/*+ plan_hint */] FROM [ ONLY ] table_name [ partition_clause ] @ dblink  [ [ AS ] alias ]
   [ USING using_list ]
   [ WHERE condition]
   [ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS
{ FIRST | LAST } ]} [, ...] ]
   [ LIMIT { [offset,] count | ALL } ]
   [ RETURNING { * | { output_expr [ [ AS ] output_name ] } [, ...] } ];
```

- 通过DATABASE LINK进行LOCK TABLE操作。

```
LOCK [ TABLE ] {[ ONLY ] name @ dblink  [, ...]}

   [ IN {ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE | SHARE |
SHARE ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE} MODE ]
   [ NOWAIT ];
```

- 调用远程数据库的存储过程或函数。

```
CALL | SELECT [ schema. ] { func_name@dblink | procedure_name@dblink } ( param_expr );
```


⚠ 注意

- DATABASE LINK调用远程函数或存储过程不支持自定义类型、out出参、聚集函数、窗口函数、以及返回set函数，不支持使用SELECT * FROM func@dblink()形式调用。
- DATABASE LINK调用远程函数或存储过程不指定schema默认调用PUBLIC下的函数。
- DATABASE LINK调用远程函数或存储过程时，param_expr不支持用符号":="或者"=>"将参数名和参数值隔开。

说明

- 上述语法中SQL语句涉及到的DATABASE LINK无关参数含义与原SQL语句中含义相同。
- 指定列名时可以在列名后加上"@dblink"指定为对应DATABASE LINK所指向的表的列。

规格约束

- **事务**

  使用DATABASE LINK的时候本地和远程事务的关系如下：

  a. 本地事务会同步控制远程事务的提交/回滚状态。

  b. 隔离级别的对应关系为：

本地隔离级别	远程隔离级别
Read Uncommitted	Repeatable Read
Read Committed	Repeatable Read
Repeatable Read	Repeatable Read
Serializable	Serializable

  须知

  本地事务提交过程中会向远端发送事务提交请求，如果远端事务提交成功后出现异常情况导致本地的事务提交失败，如连接异常，本地集群实例异常等情况，远端的事务提交无法被撤回，可能出现本地事务与远端事务不一致的情况。

- **本地用户对DATABASE LINK的使用权限**

  a. 如果使用了public关键词，就是公有的DATABASE LINK，可以被所有用户/模式使用。

  b. 如果没有使用public关键词，就是私有的的DATABASE LINK，仅能被当前用户/模式使用（包括sys用户也无法跨schema使用DATABASE LINK）。

- **通过DATABASE LINK访问远程数据库对象的权限**

  对远程数据库对象的访问权限与DATABASE LINK绑定的远程连接用户的权限保持一致。

- **支持SQL范围**
  - DATABASE LINK相关语句支持情况见。**表7-152**
  - DATABASE LINK相关表类型支持情况见**表7-153**。
- **DATABASE LINK函数调用**
  - DATABASE LINK调用远程函数不支持自定义类型、OUT/INOUT参数、PACKAGE内函数、聚集函数、窗口函数、以及返回set函数。
  - PLSQL_BODY内通过DATABASE LINK调用远程数据库的存储过程或函数不支持自定义类型、OUT/INOUT参数、PACKAGE内函数、重载函数、聚集函数、窗口函数、以及返回set函数。

须知

- PLSQL_BODY内调用远程数据库的存储过程或函数时，应使用[CALL | SELECT] [schema.] { func_name@dblink | procedure_name@dblink } (param_expr)语法格式调用。
- PLSQL_BODY内调用远程数据库的无参存储过程或函数时，应使用[CALL | SELECT] [schema.] { func_name@dblink | procedure_name@dblink } ( )语法格式调用。

- **同义词**
  - 不支持将DATABASE LINK名创建为一个同义词的使用方法。
  - 不支持通过DATABASE LINK调用远端数据库中指向一个DATABASE LINK对象的同义词。例如如下场景：
    - 步骤一：在DB1上创建表TABLE1。
    - 步骤二：在DB2上创建连接DB1的DBLINK1，并创建同义词"CREATE SYNONYM T1 FOR TABLE1@DBLINK1"。
    - 步骤三：在DB3上创建连接DB2的DBLINK2，通过DBLINK2调用DB2上的同义词T1，"SELECT * FROM T1@DBLINK2"。
- **表类型约束**
  - HASHBUCKET：不支持通过DATABASE LINK对远端Hash bucket表进行查询或DML操作。
  - SLICE：不支持通过DATABASE LINK对远端slice表进行查询或DML操作。
  - 复制表：不支持通过DATABASE LINK对远端复制表进行查询或DML操作。
  - TEMPORARY：不支持通过DATABASE LINK对远端临时表进行查询或DML操作。
- **视图**
  - 目前支持对DATABASE LINK的远端表创建视图，但是当远端表本身的结构发生变化时，该视图使用时可能会发生异常。例如：
    - 步骤一：在DB1上创建表TABLE1。
    - 步骤二：在DB2上创建连接DB1的DBLINK，并创建视图"CREATE VIEW V1 AS SELECT * FROM TABLE1@DBLINK。

    - 步骤三：在DB1上删除TABLE1的一列，在DB2上查询该视图会产生报错。

- **其他场景：**
  - DATABASE LINK表不支持TRIGGER，包括TRIGGER调用函数内使用DATABASE LINK场景、trigger调用函数为DATABASE LINK函数、在DATABASE LINK上定义TRIGGER情况。
  - 暂不支持UPSERT、MERGE语法。
  - 不支持current cursor语法。
  - 不支持查询表的隐藏字段。
- **dump与备份**
  - 不支持DATABASE LINK相关数据库对象的dump，备机不支持DATABASE LINK调用，也不支持被DATABASE LINK连接。
  - 不支持DATABASE LINK相关数据库对象的集群备份后恢复使用。因为不同集群的密钥文件不同，在使用 DATABASE LINK 时需要使用集群密钥文件进行解密。
- **谓词下推约束**

  仅支持WHERE子句使用的数据类型、操作符和函数是内置的，并且使用的函数是IMMUTABLE类型。
- **聚集函数下推约束**

  仅支持单表且没有GROUP、ORDER BY、HAVING、LIMIT子句的SELECT语句，并且不支持窗口函数。
- **hint下推**

  支持针对DATABASE LINK表对象的hint条件下推，仅限scan方式的hint下推，语法格式如下：

  ```
  [no] tablescan|indexscan|indexonlyscan(table [index])
  ```

  并要求在一个 queryblock 中的表名或表别名不能重复。

表 7-152 支持 SQL 范围

SQL类型	操作对象	支持选项说明	执行上下文
创建 DATABASE LINK	DATABASE LINK	NA	普通事务块
修改 DATABASE LINK	DATABASE LINK	仅支持用户名、密码的修改	普通事务块
删除 DATABASE LINK	DATABASE LINK	NA	普通事务块

SQL类型	操作对象	支持选项说明	执行上下文
SELECT语句	普通表、普通视图、全量物化视图	• WHERE子句 • DATABASE LINK表和内部表JOIN • DATABASE LINK表和DATABASE LINK表JOIN • 聚集函数 • LIMIT子句 • ORDER BY子句 • GROUP BY子句、HAVING子句 • UNION子句 • WITH子句 • START WITH子句和CONNECT BY子句 • FOR UPDATE子句 • Rownum使用	普通事务块、存储过程、函数、高级包、逻辑视图
INSERT语句	普通表	• 多VALUE插入	普通事务块、存储过程、函数、高级包
UPDATE语句	普通表	• LIMIT子句 • ORDER BY子句 • WHERE子句	普通事务块、存储过程、函数、高级包
DELETE语句	普通表	• LIMIT子句 • ORDER BY子句 • WHERE子句	普通事务块、存储过程、函数、高级包
LOCK TABLE语句	普通表	• LOCKMODE子句 • NOWAIT子句	普通事务块

表 7-153 表类型支持情况

维度	GaussDB表类型		DATABASE LINK支持情况
TEMP选项	临时表		不支持
	全局临时表		支持
UNLOGGED选项	非日志表		支持
存储特性	行存	Astore	支持
		Ustore	支持

维度	GaussDB表类型	DATABASE LINK支持情况
	分区表	支持
	二级分区表	支持
视图	DATABASE LINK访问远程视图	支持dql，不支持dml
	本地视图通过 DATABASE LINK 关联远程表	支持dql，不支持dml

示例

```
--DDL语句。
CREATE USER user2 WITH PASSWORD '********'; -- 创建普通用户
GRANT CREATE PUBLIC DATABASE LINK TO user2; --赋予用户创建DATABASE LINK对象的权限
GRANT DROP PUBLIC DATABASE LINK TO user2;  --赋予用户删除DATABASE LINK对象的权限
GRANT ALTER PUBLIC DATABASE LINK TO user2;   --赋予用户修改DATABASE LINK对象的权限
REVOKE CREATE PUBLIC DATABASE LINK FROM user2;  --回收用户创建DATABASE LINK对象的权限
REVOKE DROP PUBLIC DATABASE LINK FROM user2;    --回收用户删除DATABASE LINK对象的权限
REVOKE ALTER PUBLIC DATABASE LINK FROM user2;   --回收用户修改DATABASE LINK对象的权限
CREATE PUBLIC DATABASE LINK dblink CONNECT TO 'user1' IDENTIFIED BY '*********' USING (HOST
'192.168.11.11', PORT '5432', DBNAME 'db1'); --创建DATABASE LINK对象
ALTER PUBLIC DATABASE LINK dblink CONNECT TO 'user1' IDENTIFIED BY '********'; -- 修改DATABASE LINK信
息
DROP PUBLIC DATABASE LINK dblink; -- 删除DATABASE LINK对象

-- DATABASE LINK具体操作语句
-- 前置操作
CREATE USER user1 WITH SYSADMIN PASSWORD '********';
CREATE USER user2 WITH SYSADMIN PASSWORD '********';
CREATE DATABASE db1; --远端数据库
CREATE DATABASE db2; --测试DATABASE LINK数据库
\c db1 user1
-- 创建普通表
db1=> CREATE TABLE remote_tb(f1 int, f2 text, f3 text[]);
db1=> INSERT INTO remote_tb VALUES (0,'a','{"a0","b0","c0"}');
db1=> INSERT INTO remote_tb VALUES (1,'bb','{"a1","b1","c1"}');
db1=> INSERT INTO remote_tb VALUES (2,'cc','{"a2","b2","c2"}');
-- 创建function
CREATE OR REPLACE FUNCTION f(a in int, b in int)
return int AS
    tmp int :=  a + b;
    begin
        return tmp;
    end;
/
CREATE FUNCTION
-- 创建同义词
db1=> CREATE SYNONYM remote_sy FOR remote_tb;

\c db2 user2
db2=> CREATE TABLE local_tb(f1 int, f2 text, f3 text[]);
db2=> INSERT INTO local_tb VALUES (2,'c','{"a2","b2","c2"}');
db2=> CREATE PUBLIC DATABASE LINK dblink CONNECT TO 'user1' IDENTIFIED BY '********' USING (HOST
'192.168.11.11', PORT '5432', DBNAME 'db1'); -- host和port需要根据实际情况填写
db2=> SELECT * FROM remote_tb@dblink; --查询远端表
 f1 | f2 |     f3
----+----+------------
  1 | bb | {a1,b1,c1}
  2 | cc | {a2,b2,c2}
  0 | a  | {a0,b0,c0}
(3 rows)
db2=> INSERT INTO remote_tb@dblink VALUES (4,'d','{"a1","b2","c3"}'); --向远端表插入数据
```

```
INSERT 0 1
db2=> UPDATE remote_tb@dblink SET f2 = 'aa' WHERE f1 = 0; --更新远端表
UPDATE 1
db2=> DELETE remote_tb@dblink WHERE f1 = 1; --删除远端表数据
DELETE 1
db2=> SELECT * FROM remote_tb@dblink JOIN local_tb ON local_tb.f1 = remote_tb.f1@dblink; --本地表join
远程表
 f1 | f2 |     f3     | f1 | f2 |     f3
----+----+------------+----+----+------------
  2 | cc | {a2,b2,c2} |  2 | c  | {a2,b2,c2}
(1 row)

db2=> SELECT count(*) FROM remote_tb@dblink;
 count
-------
     3
(1 row)
db2=>
db2=> SELECT f@dblink(1,2); --访问远端函数
 f
---
 3
(1 row)
CREATE OR REPLACE FUNCTION call_f(a in int, b in int) -- plsql_body内访问远端函数
RETURN int AS
    tmp int;
    begin
        tmp := f@dblink(a, b);
        RETURN tmp;
    end;
/
CREATE FUNCTION
db2=> SELECT call_f(1, 2);
 call_f
--------
      3
(1 row)
db2=> CREATE SYNONYM local_sy FOR remote_tb@dblink; --创建DATABASE LINK对象的同义词
CREATE SYNONYM
db2=> SELECT * FROM local_sy;
 f1 | f2 |     f3
----+----+------------
  1 | bb  | {a1,b1,c1}
  2 | cc | {a2,b2,c2}
  0 | a  | {a0,b0,c0}
(3 rows)
db2=> SELECT * FROM remote_sy@dblink; --访问远端数据库的同义词
 f1 | f2 |     f3
----+----+------------
  1 | bb | {a1,b1,c1}

  2 | cc | {a2,b2,c2}
  0 | a  | {a0,b0,c0}
(3 rows)
db2=> EXPLAIN VERBOSE SELECT /*+ tablescan(remote_sy) */ * FROM remote_sy@dblink; --DATABASE LINK
支持部分hint下推
                                QUERY PLAN
---------------------------------------------------------------------------------
 Foreign Scan on public.remote_tb remote_sy  (cost=100.00..100.03 rows=1 width=68)
   Output: f1, f2, f3
   Remote SQL: SELECT /*+  tablescan(remote_sy) */ f1, f2, f3 FROM public.remote_tb
(3 rows)

db2=> SELECT * FROM gs_database_link; --查看DATABASE LINK系统表
db2=> START TRANSACTION;
START TRANSACTION
db2=> SELECT * FROM remote_sy@dblink;
 f1 | f2 |     f3
----+----+------------
```



```
 1 | bb | {a1,b1,c1}
 2 | cc | {a2,b2,c2}
 0 | a  | {a0,b0,c0}
(3 rows)

db2=> SELECT intransaction FROM gs_db_links; --查看DATABASE LINK系统视图
 intransaction
---------------
 t
(1 row)
db2=> END;
COMMIT
db2=> ALTER PUBLIC DATABASE LINK dblink CONNECT TO 'user1' IDENTIFIED BY '********'; --修改DATABASE LINK信息
db2=> DROP PUBLIC DATABASE LINK dblink; --删除DATABASE LINK对象
```

相关链接

CREATE DATABASE LINK、**GS_DATABASE_LINK**、**GS_DB_LINKS**

8 最佳实践

8.1 表设计最佳实践

8.1.1 使用分区表

分区表是把逻辑上的一张表根据某种方案分成几张物理块进行存储。这张逻辑上的表称之为分区表，物理块称之为分区。分区表是一张逻辑表，不存储数据，数据实际是存储在分区上的。分区表和普通表相比具有以下优点：

1. 改善查询性能：对分区对象的查询可以仅搜索自己关心的分区，提高检索效率。
2. 增强可用性：如果分区表的某个分区出现故障，表在其他分区的数据仍然可用。
3. 方便维护：如果分区表的某个分区出现故障，需要修复数据，只修复该分区即可。

   GaussDB数据库支持的分区表为一级分区表和二级分区表，其中一级分区表包括范围分区表、间隔分区表、列表分区表、哈希分区表四种，二级分区表包括范围分区、列表分区、哈希分区两两组合的九种。

   - 范围分区表：将数据基于范围映射到每一个分区，这个范围是由创建分区表时指定的分区键决定的。这种分区方式是最为常用的，并且分区键经常采用日期，例如将销售数据按照月份进行分区。
   - 间隔分区表：是一种特殊的范围分区表，相比范围分区表，新增间隔值定义，当插入记录找不到匹配的分区时，可以根据间隔值自动创建分区。
   - 列表分区表：将数据中包含的键值分别存储在不同的分区中，依次将数据映射到每一个分区，分区中包含的键值由创建分区表时指定。
   - 哈希分区表：将数据根据内部哈希算法依次映射到每一个分区中，包含的分区个数由创建分区表时指定。
   - 二级分区表：由范围分区、列表分区、哈希分区任意组合得到的分区表，其一级分区和二级分区均可以使用前面三种定义方式。

8.1.2 选择数据类型

高效数据类型，主要包括以下三方面：

1. **尽量使用执行效率比较高的数据类型**

一般来说整型数据运算(包括=、＞、＜、≧、≦、≠等常规的比较运算，以及group by)的效率比字符串、浮点数要高。

2. **尽量使用短字段的数据类型**

   长度较短的数据类型不仅可以减小数据文件的大小，提升I/O性能；同时也可以减小相关计算时的内存消耗，提升计算性能。比如对于整型数据，如果可以用smallint就尽量不用int，如果可以用int就尽量不用bigint。

3. **使用一致的数据类型**

   表关联列尽量使用相同的数据类型。如果表关联列数据类型不同，数据库必须动态地转化为相同的数据类型进行比较，这种转换会带来一定的性能开销。

8.2 SQL 查询最佳实践

根据数据库的SQL执行机制以及大量的实践总结发现：通过一定的规则调整SQL语句，在保证结果正确的基础上，能够提高SQL执行效率。

- **使用union all代替union**

  union在合并两个集合时会执行去重操作，而union all则直接将两个结果集合并、不执行去重。执行去重会消耗大量的时间，因此，在一些实际应用场景中，如果通过业务逻辑已确认两个集合不存在重叠，可用union all替代union以便提升性能。

- **join列增加非空过滤条件**

  若join列上的NULL值较多，则可以加上is not null过滤条件，以实现数据的提前过滤，提高join效率。

- **not in转not exists**

  not in语句需要使用nestloop anti join来实现，而not exists则可以通过hash anti join来实现。在join列不存在null值的情况下，not exists和not in等价。因此在确保没有null值时，可以通过将not in转换为not exists，通过生成hash join来提升查询效率。

  建表语句如下：

  ```
  DROP SCHEMA IF EXISTS no_in_to_no_exists_test CASCADE;
  CREATE SCHEMA no_in_to_no_exists_test;
  SET CURRENT_SCHEMA=no_in_to_no_exists_test;
  CREATE TABLE t1(c1 int, c2 int, c3 int);
  CREATE TABLE t2(d1 int, d2 int NOT NULL, d3 int);
  ```

  使用NOT IN实现查询语句如下：

  ```
  SELECT * FROM t1 WHERE  c1 NOT IN (SELECT d2 FROM t2);
  ```

  其计划如下所示：

  ```
  gaussdb=# EXPLAIN SELECT * FROM t1 WHERE  c1 NOT IN (SELECT d2 FROM t2);
                              QUERY PLAN
  ------------------------------------------------------------------------
   Nested Loop Anti Join  (cost=0.00..29749.02 rows=968 width=12)
     Join Filter: ((t1.c1 = t2.d2) OR (t1.c1 IS NULL))
     ->  Seq Scan on t1  (cost=0.00..29.45 rows=1945 width=12)
     ->  Materialize  (cost=0.00..39.17 rows=1945 width=4)
           ->  Seq Scan on t2  (cost=0.00..29.45 rows=1945 width=4)
  (5 rows)
  ```

  因为t2.d2字段中没有NULL值（t2.d2字段在表定义中为NOT NULL），所以查询可以等价修改如下：

  ```
  SELECT * FROM t1 WHERE  NOT EXISTS (SELECT * FROM t2 WHERE t1.c1=t2.d2);
  ```

  其生成的计划如下：


```
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE  NOT EXISTS (SELECT * FROM t2 WHERE t1.c1=t2.d2);
                              QUERY PLAN
------------------------------------------------------------------------
 Hash Anti Join  (cost=53.76..99.14 rows=972 width=12)
   Hash Cond: (t1.c1 = t2.d2)
   ->  Seq Scan on t1  (cost=0.00..29.45 rows=1945 width=12)
   ->  Hash  (cost=29.45..29.45 rows=1945 width=4)
         ->  Seq Scan on t2  (cost=0.00..29.45 rows=1945 width=4)
(5 rows)
```

- **选择hashagg。**

  查询语句中如果存在GROUP BY条件则生成的计划（Plan）中可能存在排序操作，即计划中包含GroupAgg+Sort算子，导致性能较差。可以通过设置GUC参数work_mem增大可用内存，生成带有HashAgg的计划（Plan）避免排序操作从而提升性能。work_mem设置请联系管理员。

- **尝试将函数替换为case语句。**

  数据库函数调用性能较低，如果出现过多的函数调用导致性能下降很多，可以根据情况把可下推函数的函数改成CASE表达式。

- **避免对索引使用函数或表达式运算。**

  对索引使用函数或表达式运算会停止使用索引转而执行全表扫描。

- **尽量避免在where子句中使用!=或<>操作符、null值判断、or连接、参数隐式转换。**

- **对复杂SQL语句进行拆分。**

  对于过于复杂并且不易通过以上方法调整性能的SQL可以考虑拆分的方法，把SQL中某一部分拆分成独立的SQL并把执行结果存入临时表，拆分常见的场景包括但不限于：

  - 作业中多个SQL有同样的子查询，并且子查询数据量较大。
  - Plan cost计算不准，导致子查询hash bucket太小，比如实际数据1000W行，hash bucket只有1000。
  - 函数（如substr、to_number）导致大数据量子查询选择度计算不准。
  - 多DN环境下对大表做broadcast的子查询。

其他更多调优点，请参考**典型SQL调优点**。

9 用户自定义函数

数据库实例启动时，除了启动gaussdb主进程之外，还会启动UDF master进程。当需要执行fenced模式的UDF时，UDF master进程会fork出UDF worker进程，UDF worker进程执行fenced模式的UDF。

9.1 PL/SQL 语言函数

PL/SQL是一种可载入的过程语言。

用PL/SQL创建的函数可以被用在任何可以使用内建函数的地方。例如，可以创建复杂条件的计算函数并且后面用它们来定义操作符或把它们用于索引表达式。

SQL被大多数数据库用作查询语言。它是可移植的并且容易学习。但是每一个SQL语句必须由数据库服务器单独执行。

客户端应用对于每一个查询都要执行发送查询到数据库服务器、等待查询被接收、接收并处理结果、进行相关计算、然后发送更多查询给服务器的过程。如果客户端和数据库服务器不在同一台机器上，该过程还会引起进程间通信问题并且将带来网络负担。

通过PL/SQL，可以将一整块计算和一系列查询分组在数据库服务器内部，这样就有了一种过程语言的能力并且使SQL更易用，同时能节省客户端/服务器通信开销。

- 客户端和服务器之间的额外往返通信被消除。
- 客户端不需要的中间结果不必被整理或者在服务器和客户端之间传送。
- 多轮的查询解析可以被避免。

PL/SQL可以使用SQL中所有的数据类型、操作符和函数。应用PL/SQL创建函数的语法为**CREATE FUNCTION**。

PL/SQL是一种可载入的过程语言，其应用方法与**存储过程**相似，但**存储过程**无返回值，PL/SQL语言函数有返回值。

XML类型数据支持作为自定义函数的入参，出参，自定义变量，返回值。

10 存储过程

10.1 存储过程

商业规则和业务逻辑可以通过程序存储在GaussDB中，这个程序就是存储过程。

存储过程是SQL和PL/SQL的组合。存储过程使执行商业规则的代码可以从应用程序中移动到数据库。从而，代码存储一次能够被多个程序使用。

存储过程的创建及调用办法请参考**CREATE PROCEDURE**。

PL/SQL语言函数节所提到的PL/SQL语言创建的函数与存储过程的应用方法相同。下面各节中，除非特别声明，否则内容通用于存储过程和PL/SQL语言函数。

10.2 数据类型

数据类型是一组值的集合以及定义在这个值集上的一组操作。GaussDB数据库是由表的集合组成的，而各表中的列定义了该表，每一列都属于一种数据类型，GaussDB根据数据类型有相应函数对其内容进行操作，例如GaussDB可对数值型数据进行加、减、乘、除等操作。

XML类型数据支持作为存储过程的入参、出参、自定义变量、返回值。支持自治事务的存储过程。

10.3 数据类型转换

数据库中有些数据类型间允许进行隐式类型转换（例如赋值、函数调用的参数等）、有些数据类型间不允许进行隐式数据类型转换（例如int和复合类型），可尝试使用GaussDB提供的类型转换函数，例如CAST进行数据类型强转。

GaussDB数据库常见的隐式类型转换，请参见**表10-1**。

须知

GaussDB支持的DATE的效限范围是：公元前4713年到公元294276年。

表 10-1 隐式类型转换表

原始数据类型	目标数据类型	备注
CHAR	VARCHAR2	-
CHAR	NUMBER	原数据必须由数字组成。
CHAR	DATE	原数据不能超出合法日期范围。
CHAR	RAW	-
CHAR	CLOB	-
VARCHAR2	CHAR	-
VARCHAR2	NUMBER	原数据必须由数字组成。
VARCHAR2	DATE	原数据不能超出合法日期范围。
VARCHAR2	CLOB	-
NUMBER	CHAR	-
NUMBER	VARCHAR2	-
DATE	CHAR	-
DATE	VARCHAR2	-
RAW	CHAR	-
RAW	VARCHAR2	-
CLOB	CHAR	-
CLOB	VARCHAR2	-
CLOB	NUMBER	原数据必须由数字组成。
INT4	CHAR	-
INT4	BOOLEAN	-
BOOLEAN	INT4	-

10.4 数组、集合和 record

10.4.1 数组

10.4.1.1 数组类型的使用

在使用数组之前，需要自定义一个数组类型。

在存储过程中紧跟AS关键字后面定义数组类型。定义方法如下。

```
TYPE array_type IS VARRAY(size) OF data_type;
```

其中：

- array_type：要定义的数组类型名。
- VARRAY：表示要定义的数组类型。
- size：取值为正整数，表示可以容纳的成员的最大数量。
- data_type：要创建的数组中成员的类型。

说明

- 在GaussDB中，数组会自动增长，访问越界会返回一个NULL，不会报错，打开varray_compat参数后，会增加对下标的检查，访问越界会进行报错
- 在存储过程中定义的数组类型，其作用域仅在该存储过程中。
- size信息会记录到pg_type系统表中，打开varray_compat参数后，对数组的操作都会进行长度以及下标的检查，未开参数时则不使用size信息。
- data_type也可以为存储过程中定义的record类型（匿名块不支持）、集合类型，但不可以为存储过程中定义的数组类型、游标类型。
- data_type为集合类型时，不支持使用多维数组。
- 不支持NOT NULL语法。
- array类型的构造器仅支持在A兼容模式下使用。
- array类型的构造器不支持作为函数或存储过程参数的默认值。
- 当数组是集合类型的元素并且数组的data_type为varchar、numeric等可以定义长度和精度的类型时，要校验该数组的元素长度或者将元素转换成对应的精度，需要开启tableof_elem_constraints参数（设置behavior_compat_options参数值为tableof_elem_constraints）。
- 开启varray_compat参数后（设置behavior_compat_options参数值为varray_compat），在匿名块中定义的数组类型，匿名块执行ROLLBACK或发生EXCEPTION后，数组类型将无法继续使用。
- 设置enable_recordtype_check_strict参数值为on后，成员是record类型，且record类型有列具有not null属性或default属性，在存储过程或PACKAGE编译时会报错。

GaussDB支持使用圆括号来访问数组元素，且还支持一些特有的函数，如extend、count、first、last、prior、exists、trim、next、delete来访问数组的内容。

说明

- 存储过程中如果有DML语句（SELECT、UPDATE、INSERT、DELETE），DML语句推荐使用中括号来访问数组元素，使用小括号默认识别为数组访问，若数组不存在，则识别为函数表达式。
- 如果clob类型大于1GB，则存储过程中的table of类型、record类型、clob作为出入参、游标、raise info等功能不支持。

示例

```
--演示在存储过程中对数组进行操作。
gaussdb=# CREATE OR REPLACE PROCEDURE array_proc AS
DECLARE
    TYPE ARRAY_INTEGER IS VARRAY(1024) OF INTEGER;--定义数组类型
    ARRINT ARRAY_INTEGER := ARRAY_INTEGER();  --声明数组类型的变量
BEGIN
    ARRINT.extend(10);
    FOR I IN 1..10 LOOP
        ARRINT(I) := I;
    END LOOP;
```



```
    DBE_OUTPUT.PRINT_LINE(ARRINT.COUNT);
    DBE_OUTPUT.PRINT_LINE(ARRINT(1));
    DBE_OUTPUT.PRINT_LINE(ARRINT(10));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.FIRST));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.LAST));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.NEXT(ARRINT.FIRST)));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.PRIOR(ARRINT.LAST)));
    ARRINT.TRIM();

    IF ARRINT.EXISTS(10) THEN
        DBE_OUTPUT.PRINT_LINE('Exist 10th element');
    ELSE
        DBE_OUTPUT.PRINT_LINE('Not exist 10th element');
    END IF;
    DBE_OUTPUT.PRINT_LINE(ARRINT.COUNT);
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.FIRST));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.LAST));
    ARRINT.DELETE();
END;
/

--调用该存储过程显示结果。
gaussdb=# CALL array_proc();
10
1
10
1
10
2
9
Not exist 10th element
9
1
9
 array_proc
------------


(1 row)

--删除存储过程。
gaussdb=# DROP PROCEDURE array_proc;
```


10.4.1.2 数组支持的函数

说明

- 以下几点说明描述了数组类型函数在开启GUC参数varray_compat前后的差异行为：
  - count、extend、trim、delete、first、last、next和prior函数应用在未初始化的数组即数组为NULL时，开启参数会报Reference to uninitialized collection的错误，未开启则不报错。
  - 对于extend(count)函数和extend(count, idx)函数：count加上数组现有元素个数超过数组定义大小时，开启参数后会报Subscript outside of limit的错误，未开启则不报错。
  - 对于extend(count, idx)函数：下标idx不合法时开启参数后会报Subscript outside of limit或者Subscript beyond count的错误。
  - 对于trim(n)函数：当n大于数组元素个数时，开启参数后会报Subscript beyond count的错误，未开启则不报错。
  - 对于delete(idx)函数：下标idx不合法时开启参数后会报Subscript outside of limit或者Subscript beyond count的错误，未开启参数时返回原数组。
- 以下示例为A兼容模式下的示例，函数定义说明里[]中的内容代表可选项，如count[()]表示可以写成count或count()。
- 如在执行以下示例时遇到报错Cannot change the guc status while in the same session时，需要切换session后重新执行。
- 在内层表达式中，不支持通过嵌套的方式调用数组类型函数。

- extend[([count])]

  参数：count为int4类型。

  返回值：无返回值。

  功能描述：在varray变量末尾拓展1个或count个元素，元素默认扩展为NULL。

  示例1：extend，扩展1个元素，默认扩展为NULL。

```
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer();
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.extend;
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$# end;
gaussdb$# /
0
1
ANONYMOUS BLOCK EXECUTE
```

  示例2：extend(count)，扩展count个元素，默认扩展为NULL。

```
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer();
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.extend(3);
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$# end;
gaussdb$# /
0
3
ANONYMOUS BLOCK EXECUTE
```

  示例3：开启varray_compat参数时，extend后超过数组定义大小会报错。


```
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer();
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.extend(3);
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.extend(8); -- error
gaussdb$# end;
gaussdb$# /
0
3
ERROR:  Subscript outside of limit
CONTEXT:  PL/SQL function inline_code_block line 7 at assignment
```

示例4：开启varray_compat参数时，数组未初始化会报错。

```
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer;
gaussdb-# begin
gaussdb$#     arrint.extend(3);
gaussdb$# end;
gaussdb$# /
ERROR:  Reference to uninitialized collection
CONTEXT:  PL/pgSQL function inline_code_block line 4 at assignment
```

- extend(count, idx)

  参数：idx和count为int4类型。

  返回值：无返回值。

  功能描述：在varray变量末尾拓展count个元素，并且扩展元素的值等于给定idx下标元素的值。该功能需要设置a_format_version值为10c和设置a_format_dev_version值为s1后才能使用。

  示例1：extend(count, idx)，扩展第idx位置的元素。

```
gaussdb=# set a_format_version='10c';
SET
gaussdb=# set a_format_dev_version='s1';
SET
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer(1,2);
gaussdb-# begin
gaussdb$#     arrint.extend(2, 1);
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     for i in 1..arrint.count loop
gaussdb$#         dbe_output.print_line(arrint(i));
gaussdb$#     end loop;
gaussdb$# end;
gaussdb$# /
4
1
2
1
1
ANONYMOUS BLOCK EXECUTE
```

  示例2：extend(count, idx)，开启了varray_compat参数，且idx越界会报错。

```
gaussdb=# set a_format_version='10c';
SET
gaussdb=# set a_format_dev_version='s1';
SET
```



```
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer();
gaussdb-# begin
gaussdb$#     arrint.extend(10, 1); -- error index.
gaussdb$# end;
gaussdb$# /
ERROR:  Subscript beyond count
CONTEXT:  PL/SQL function inline_code_block line 4 at assignment
```

示例3：extend(count, idx)，开启了varray_compat参数，数组未初始化时会报错。

```
gaussdb=# set a_format_version='10c';
SET
gaussdb=# set a_format_dev_version='s1';
SET
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer;
gaussdb-# begin
gaussdb$#     arrint.extend(10, 1);
gaussdb$# end;
gaussdb$# /
ERROR:  Reference to uninitialized collection
CONTEXT:  PL/pgSQL function inline_code_block line 4 at assignment
```

- count[()]

  参数：无。

  返回值：int4类型。

  功能描述：返回数组中的元素个数。

  示例1：查看已初始化的数组元素个数。

```
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer;
gaussdb-#     len int;
gaussdb-# begin
gaussdb$#     arrint := array_integer(1, 2, 3);
gaussdb$#     len := arrint.count();
gaussdb$#     dbe_output.print_line(len);
gaussdb$# end;
gaussdb$# /
3
ANONYMOUS BLOCK EXECUTE
```

  示例2：开启varray_compat参数，且数组未初始化时会报错。

```
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer;
gaussdb-#     len int;
gaussdb-# begin
gaussdb$#     len := arrint.count();
gaussdb$#     dbe_output.print_line(len);
gaussdb$# end;
gaussdb$# /
ERROR:  Reference to uninitialized collection
CONTEXT:  PL/SQL function inline_code_block line 5 at assignment
```

- trim[(n)]

参数：n为int4类型。

返回值：无返回值。

功能描述：无参数时，删除数组末尾一个元素空间，给定参数n时，删除数组末尾指定数量元素空间。

示例1：删除数组中的n个元素，未开启参数varray_compat。

```
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer(1,2,3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.trim(1);
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$# end;
gaussdb$# /
3
2
ANONYMOUS BLOCK EXECUTE
-- n大于数组元素个数，清空数组元素
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer(1,2,3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.trim(4);
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$# end;
gaussdb$# /
3
0
ANONYMOUS BLOCK EXECUTE
```

示例2：删除数组中的n个元素，且n大于数组元素个数，未开启参数varray_compat。

```
-- n大于数组元素个数，清空数组元素
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer(1,2,3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.trim(4);
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$# end;
gaussdb$# /
3
0
ANONYMOUS BLOCK EXECUTE
```

示例3：删除数组中的n个元素，且n大于数组元素个数，同时打开varray_compat参数。

```
-- n大于数组元素个数时会报错
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer(1,2,3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count); -- count > 0.
gaussdb$#     arrint.trim(4);
gaussdb$# end;
gaussdb$# /
3
ERROR:  Subscript beyond count
```



```
CONTEXT:  PL/SQL function inline_code_block line 5 at assignment
-- 数组未初始化报错
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer;
gaussdb-# begin
gaussdb$#     arrint.trim(1);
gaussdb$# end;
gaussdb$# /
ERROR:  Reference to uninitialized collection
CONTEXT:  PL/SQL function inline_code_block line 4 at assignment
```

示例4：数组未初始化，同时打开varray_compat参数。

```
-- 数组未初始化报错
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer;
gaussdb-# begin
gaussdb$#     arrint.trim(1);
gaussdb$# end;
gaussdb$# /
ERROR:  Reference to uninitialized collection
CONTEXT:  PL/SQL function inline_code_block line 4 at assignment
```

- delete[()]

  参数：无参数。

  返回值：无返回值。

  功能描述：清空数组中的元素。

  示例1：清空数组中的元素，未开启参数varray_compat。

```
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer(1,2,3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.delete();
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$# end;
gaussdb$# /
3
0
ANONYMOUS BLOCK EXECUTE
-- 数组未初始化
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer;
gaussdb-# begin
gaussdb$#     arrint.delete();
gaussdb$# end;
gaussdb$# /
ANONYMOUS BLOCK EXECUTE
```

  示例2：清空数组中的元素，同时开启参数varray_compat。

```
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer(1,2,3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.delete();
gaussdb$#     dbe_output.print_line(arrint.count);
```



```
gaussdb$# end;
gaussdb$# /
3
0
ANONYMOUS BLOCK EXECUTE
-- 数组未初始化报错
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer;
gaussdb-# begin
gaussdb$#     arrint.delete();
gaussdb$# end;
gaussdb$# /
ERROR:  Reference to uninitialized collection
CONTEXT:  PL/SQL function inline_code_block line 4 at assignment
```

- delete(idx)

  参数：int4类型。

  返回值：无返回值。

  功能描述：给定下标idx在数组范围内，则删除数组中给定下标idx的元素。

  示例1：给定idx下标在数组范围内。

```
gaussdb=# set a_format_version='';
SET
gaussdb=# set a_format_dev_version='';
SET
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer(1, 2, 3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.delete(2);
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     for i in 1..arrint.count loop
gaussdb$#         dbe_output.print_line(arrint(i));
gaussdb$#     end loop;
gaussdb$# end;
gaussdb$# /
3
2
1
3
ANONYMOUS BLOCK EXECUTE
```

  示例2：给定idx下标在数组范围外，未开启参数varray_compat。

```
-- 未开参数varray_compat
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer(1, 2, 3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.delete(4);
gaussdb$#     raise info '%', arrint;
gaussdb$# end;
gaussdb$# /
3
INFO:  {1,2,3}
ANONYMOUS BLOCK EXECUTE
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer:= array_integer(1, 2, 3);
gaussdb-# begin
```



```
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.delete(11);
gaussdb$#     raise info '%', arrint;
gaussdb$# end;
gaussdb$# /
3
INFO:  {1,2,3}
ANONYMOUS BLOCK EXECUTE

-- 开启参数varray_compat后报错
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer(1, 2, 3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.delete(4);
gaussdb$#     raise info '%', arrint;
gaussdb$# end;
gaussdb$# /
3
ERROR:  Subscript beyond count
CONTEXT:  PL/SQL function inline_code_block line 5 at assignment

gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer:= array_integer(1, 2, 3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.delete(11);
gaussdb$#     raise info '%', arrint;
gaussdb$# end;
gaussdb$# /
3
ERROR:  Subscript outside of limit
CONTEXT:  PL/SQL function inline_code_block line 5 at assignment
```

示例3：给定idx下标在数组范围外，开启参数varray_compat。

```
-- 开启参数varray_compat后报错
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer(1, 2, 3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.delete(4);
gaussdb$#     raise info '%', arrint;
gaussdb$# end;
gaussdb$# /
3
ERROR:  Subscript beyond count
CONTEXT:  PL/SQL function inline_code_block line 5 at assignment

gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer:= array_integer(1, 2, 3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.delete(11);
gaussdb$#     raise info '%', arrint;
gaussdb$# end;
gaussdb$# /
3
```



```
ERROR:  Subscript outside of limit
CONTEXT:  PL/SQL function inline_code_block line 5 at assignment
```

示例4：开启varray_compat参数后，数组未初始化时报错。

```
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer;
gaussdb-# begin
gaussdb$#     arrint.delete(2);
gaussdb$# end;
gaussdb$# /
ERROR:  Reference to uninitialized collection
CONTEXT:  PL/pgSQL function inline_code_block line 4 at assignment
```

- first[()]

  参数：无。

  返回值：int4类型。

  功能描述：返回数组中第一个元素的下标，若没有第一个元素，则返回NULL。

  示例：

```
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr := varr('asd','zxc');
gaussdb-# begin
gaussdb$#     raise info 'first is %',v.first();
gaussdb$# end;
gaussdb$# /
INFO:  first is 1
ANONYMOUS BLOCK EXECUTE
-- 数组未初始化返回NULL
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr;
gaussdb-# begin
gaussdb$#     raise info 'first is %',v.first;
gaussdb$# end;
gaussdb$# /
INFO:  first is <NULL>
ANONYMOUS BLOCK EXECUTE
-- 开启varray_copmat参数后，数组未初始化报错
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr;
gaussdb-# begin
gaussdb$#     raise info 'first is %',v.first;
gaussdb$# end;
gaussdb$# /
ERROR:  Reference to uninitialized collection
CONTEXT:  PL/pgSQL function inline_code_block line 4 at RAISE
```

- last[()]

  参数：无。

  返回值：int4类型。

  功能描述：返回数组中最后一个元素的下标。若没有最后一个元素，则返回NULL。

  示例1：

```
gaussdb=# set behavior_compat_options = '';
SET
```



```
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr := varr('asd','zxc');
gaussdb-# begin
gaussdb$#     raise info 'last is %',v.last();
gaussdb$# end;
gaussdb$# /
INFO:  last is 2
ANONYMOUS BLOCK EXECUTE
-- 数组未初始化返回NULL
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr;
gaussdb-# begin
gaussdb$#     raise info 'last is %',v.last;
gaussdb$# end;
gaussdb$# /
INFO:  last is <NULL>
ANONYMOUS BLOCK EXECUTE
-- 开启varray_copmat参数后，数组未初始化报错
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr;
gaussdb-# begin
gaussdb$#     raise info 'first is %',v.last;
gaussdb$# end;
gaussdb$# /
ERROR:  Reference to uninitialized collection
CONTEXT:  PL/pgSQL function inline_code_block line 4 at RAISE
```

- prior(idx)

  参数：int4类型。

  返回值：int4类型。

  功能描述：返回数组中指定下标的前一个元素下标，若无法找到对应元素下标则返回NULL。

  示例：

```
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr := varr('asd','zxc');
gaussdb-# begin
gaussdb$#     raise info 'prior is %',v.prior(2);
gaussdb$# end;
gaussdb$# /
INFO:  prior is 1
ANONYMOUS BLOCK EXECUTE

-- 下标越界，大于数组范围
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr := varr('asd','zxc','qwe');
gaussdb-# begin
gaussdb$#     raise info 'prior is %',v.prior(10);
gaussdb$# end;
gaussdb$# /
INFO:  prior is 3
ANONYMOUS BLOCK EXECUTE

-- 开启varray_copmat参数后，数组未初始化报错
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
```



```
gaussdb-#     v varr;
gaussdb-# begin
gaussdb$#     raise info 'prior is %',v.prior(2);
gaussdb$# end;
gaussdb$# /
ERROR:  Reference to uninitialized collection
CONTEXT:  PL/pgSQL function inline_code_block line 4 at RAISE
```

- next(idx)

  参数：int4类型。

  返回值：int4类型。

  功能描述：返回数组中指定下标的后一个元素下标，若无法找到对应元素下标则返回NULL。

  示例：

```
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr := varr('asd','zxc');
gaussdb-# begin
gaussdb$#     raise info 'next is %',v.next(1);
gaussdb$# end;
gaussdb$# /
INFO:  next is 2
ANONYMOUS BLOCK EXECUTE

-- 下标越界，大于数组范围
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr := varr('asd','zxc','qwe');
gaussdb-# begin
gaussdb$#     raise info 'next  is %',v.next(10);
gaussdb$# end;
gaussdb$# /
INFO:  next  is <NULL>
ANONYMOUS BLOCK EXECUTE

-- 开启varray_copmat参数后，数组未初始化报错
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr;
gaussdb-# begin
gaussdb$#     raise info 'next  is %',v.next(1);
gaussdb$# end;
gaussdb$# /
ERROR:  Reference to uninitialized collection
CONTEXT:  PL/pgSQL function inline_code_block line 4 at RAISE
```

- exists(idx)

  参数：int4类型。

  返回值：true或false，Boolean类型。

  功能描述：查找指定位置是否存在有效元素。

  示例：

```
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr := varr('asd','zxc');
gaussdb-#     flag bool;
gaussdb-# begin
gaussdb$#     flag := v.exists(1);
gaussdb$#     raise info '%',flag;
gaussdb$#     flag := v.exists(3);
```



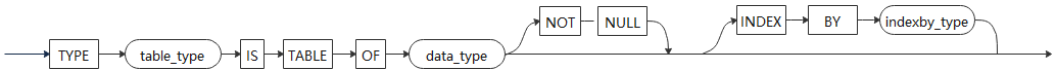
```
gaussdb$#     raise info '%',flag;
gaussdb$#     flag := v.exists(7);
gaussdb$#     raise info '%',flag;
gaussdb$# end;
gaussdb$# /
INFO:  t
INFO:  f
INFO:  f
ANONYMOUS BLOCK EXECUTE
-- 数组未初始化返回false
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr;
gaussdb-#     flag bool;
gaussdb-# begin
gaussdb$#     flag := v.exists(1);
gaussdb$#     raise info '%',flag;
gaussdb$#     flag := v.exists(3);
gaussdb$#     raise info '%',flag;
gaussdb$#     flag := v.exists(7);
gaussdb$#     raise info '%',flag;
gaussdb$# end;
gaussdb$# /
INFO:  f
INFO:  f
INFO:  f
ANONYMOUS BLOCK EXECUTE
```

10.4.2 集合

10.4.2.1 集合类型的使用

在使用集合之前，需要自定义一个集合类型。

在存储过程中紧跟AS关键字后面定义集合类型。定义方法如下。

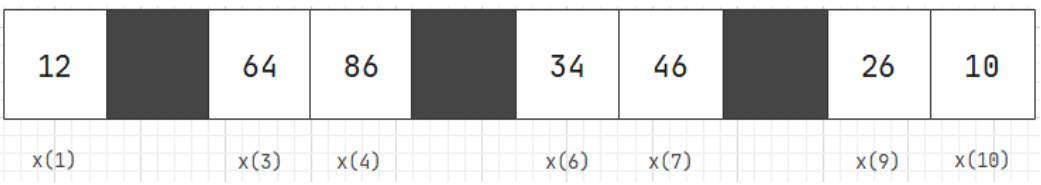


其中：

- table_type：要定义的集合类型名。
- TABLE：表示要定义的集合类型。
- data_type：要创建的集合中成员的类型。
- indexby_type：创建集合索引的类型。

无索引的集合类型

以变长数组的方式存储指定数据类型的成员，用户可以通过extend函数扩展存储空间，通过trim函数释放存储空间。存储空间为10，成员类型为int的集合变量x，存储方式如下图所示：



其中成员 x(2)，x(5)，x(8)三个成员是无效的，但是存储空间仍然保留，后续可以继续赋值，而不需要重新分配空间。

定义集合类型后，使用table_type作为类型名声明变量：

```
var_name table_type [:= table_type([v1[,...]])];
```

可在变量声明时或者声明后使用类型构造器对变量进行初始化。如未初始化，变量var_name的值为NULL。

变量声明和初始化后，可通过下标访问集合成员，或者对成员进行赋值。下标的范围为 [1, upper]，upper 的值为当前空间的大小。如访问被删除的成员，会返回no data found的错误信息。

说明

- 非兼容A模式下（参数sql_compatibility值不为A），不支持创建集合类型。
- 在GaussDB中，无索引的集合不会自动增长，访问下标越界时会报错。
- 支持在schema、匿名块、存储过程、自定义函数、package中定义的集合类型，其作用域各不相同。
- NOT NULL只支持语法不支持功能。
- 当data_type为varchar、numeric等可以定义长度和精度的类型时，要校验集合的元素长度或者将元素转换成对应的精度，需要开启tableof_elem_constraints参数。
- data_type为数组类型时，数组类型的元素长度校验或精度转换也受参数tableof_elem_constraints是否开启影响。
- 通过数组类型转换成的集合类型的值不支持对元素长度校验或精度转换。
- data_type可以为基础数据类型、或存储过程内定义的record类型、集合类型、数组类型，不支持ref cursor类型。
- 不同的集合类型的变量不能相互赋值。即使成员类型相同，但集合类型名称不同，也是不同的集合类型。如TYPE t1 IS TABLE OF int;和TYPE t2 IS TABLE OF int;定义的两个集合类型，t1和t2是不同的集合类型，以其定义的变量不支持相互赋值（作为成员类型时该约束不保证生效，赋值逻辑受父类型影响）。
- 只支持集合的等值(=)与非等值(<>或!=)比较，不支持其他关系运算和算数运算操作。
- 集合类型与NULL比较时，请使用 IS [NOT] NULL，使用 = 操作符与NULL比较的结果不准确。
- 支持集合类型变量作为函数的参数和返回值，此时要求参数或者返回值的类型是在schema或者package中定义的集合类型。
- 无索引的集合作为函数入参时，可以传入对应子元素类型相同的数组类型作为入参，不支持多维数组，且要求数组下标从1开始（过时的方法，不建议使用该功能。可执行“set behavior_compat_options = 'disable_rewrite_nesttable';”禁用）。
- 不支持对XML类型数据操作。
- 集合类型以及嵌套集合的类型不支持作为表中的一列来创建表。
- 集合类型的构造器不支持浮点数以及表达式作为下标。
- 在匿名块中定义的集合类型，匿名块执行ROLLBACK或发生EXCEPTION后，集合类型将无法继续使用。
- 开启enable_recordtype_check_strict参数后，成员是record类型，且record类型有列具有not null属性或default属性，在存储过程或PACKAGE编译时会报错。

GaussDB支持使用圆括号来访问集合元素，且还支持一些特有的函数，如extend，count，first，last，prior，next，delete来访问集合的内容。

集合函数支持multiset union/intersect/except all/distinct函数。

带索引的集合类型

该集合类型将下标和对应成员值以键值对的方式存储在HASH表中，对该类型变量的所有操作实际就是对HASH表的操作。用户无需自行扩展或释放存储空间，仅需通过赋值或delete方式进行存储和删除成员。集合相关操作、说明如下：

1. 类型定义

   索引集合类型定义需同时指定成员类型data_type和下标类型indexby_type，其中下标类型仅支持integer和varchar。

2. 变量声明和初始化

   索引集合类型声明后存在3种初始化场景：未初始化、初始化为空、初始化指定下标和成员值。其中未初始化和初始化为空场景对变量的效果一致。未初始化或初始化为空后变量不为NULL，后续都可以对变量直接进行赋值。初始化指定下标和成员值场景会将指定的下标和成员值以键值对的形式保存到变量中。

3. 变量赋值

   索引集合类型变量赋值分为两种：成员赋值和整体赋值。成员赋值可通过指定下标方式对某个成员赋值，若该成员不存在则直接赋值，若存在则刷新该成员值。整体赋值则会将被赋值变量中原有成员都清空后重新保存新的成员值。整体赋值场景不能给变量赋NULL值，否则报错。

4. 变量取值

   通过指定下标方式可获取变量中对应下标的成员值，若通过下标找不到该成员则会返回no data found的错误信息。

说明

- 非兼容A模式下（参数sql_compatibility值不为A），不支持创建带索引集合类型。
- 支持在匿名块、存储过程、自定义函数、package中定义带索引集合类型，其作用域各不相同。不支持在schema中定义带索引集合类型。
- NOT NULL只支持语法不支持功能。
- 当data_type为varchar、numeric等可以定义长度和精度的类型时，要校验集合的元素长度或者将元素转换成对应的精度，需要开启tableof_elem_constraints参数。
- data_type为数组类型时，数组类型的元素长度校验或精度转换也受参数tableof_elem_constraints是否开启影响。
- 通过数组类型转换成的集合类型的值不支持对元素长度校验或精度转换。
- data_type可以为基础数据类型，或存储过程内定义的record类型，集合类型，数组类型，不支持ref cursor类型。
- indexby_type仅支持integer和varchar。
- indexby_type为varchar时，开启参数tableof_elem_constraints后在对带索引集合类型赋值时会校验index值的长度，校验行为不受char_coerce_compat参数是否开启影响，index长度大于定义长度则报错；不开启参数tableof_elem_constraints则不会对索引值进行长度校验。
- 未初始化的带索引集合类型变量非NULL。
- 带索引集合类型变量不能赋NULL值，否则报错。
- 带索引集合类型变量作为入参不能赋NULL值或''。
- 不同的带索引集合类型的变量不能相互赋值。即使成员类型和下标类型相同，但集合类型名称不同，也是不同的集合类型。如 TYPE t1 IS TABLE OF int index by int; 和 TYPE t2 IS TABLE OF int index by int; 定义的两个集合类型，t1和t2是不同的集合类型，以其定义的变量不支持相互赋值（作为成员类型时该约束不保证生效，赋值逻辑受父类型影响）。
- 带索引集合类型不支持关系运算和算数运算操作。
- select ... bulk collect into 方式赋值带索引集合类型变量时，只支持下标为integer类型的集合类型，下标为varchar类型集合不支持。
- 支持带索引集合类型变量作为函数的参数和返回值，此时要求参数或者返回值的类型是在package中定义的集合类型。
- 带索引的集合作为函数入参时，可以传入对应子元素类型相同的数组类型作为入参，不支持多维数组，不支持索引类型为varchar（过时的方法，不建议使用该功能。可执行“set behavior_compat_options = 'disable_rewrite_nesttable';”禁用）。
- 类型构造器目前仅支持集合类型，其参数个数的上限与用户自定义函数参数个数上限相同。对于带索引的集合类型，构造器在使用时索引的值仅支持为常量。
- 不支持对XML类型数据操作。
- 集合类型以及嵌套集合的类型不支持作为表中的一列来创建表。
- 集合类型的构造器不支持浮点数以及表达式作为下标。
- 在匿名块中定义的集合类型，匿名块执行ROLLBACK或发生EXCEPTION后，集合类型将无法继续使用。

示例

示例1：无索引的集合类型。

```
--演示在存储过程中对集合进行操作。
gaussdb=# CREATE OR REPLACE PROCEDURE table_proc AS
DECLARE
    TYPE TABLE_INTEGER IS TABLE OF INTEGER;--定义集合类型
    TABLEINT TABLE_INTEGER := TABLE_INTEGER();  --声明集合类型的变量
BEGIN
    TABLEINT.extend(10);
    FOR I IN 1..10 LOOP
        TABLEINT(I) := I;
```



```
    END LOOP;
    DBE_OUTPUT.PRINT_LINE(TABLEINT.COUNT);
    DBE_OUTPUT.PRINT_LINE(TABLEINT(1));
    DBE_OUTPUT.PRINT_LINE(TABLEINT(10));
END;
/
CREATE PROCEDURE

--调用该存储过程。
gaussdb=# CALL table_proc();

10
1
10
 table_proc
------------

(1 row)

--删除存储过程。
gaussdb=# DROP PROCEDURE table_proc;
DROP PROCEDURE

--演示在存储过程中对嵌套集合进行操作。
gaussdb=# CREATE OR REPLACE PROCEDURE nest_table_proc AS
DECLARE
    TYPE TABLE_INTEGER IS TABLE OF INTEGER;--定义集合类型
    TYPE NEST_TABLE_INTEGER IS TABLE OF TABLE_INTEGER;--定义集合类型
    NEST_TABLE_VAR NEST_TABLE_INTEGER := NEST_TABLE_INTEGER(); --声明嵌套集合类型的变量
BEGIN
    NEST_TABLE_VAR.extend(10);
    FOR I IN 1..10 LOOP
        NEST_TABLE_VAR(I) := TABLE_INTEGER();
        NEST_TABLE_VAR(I).extend(10);
        NEST_TABLE_VAR(I)(I) := I;
    END LOOP;
    DBE_OUTPUT.PRINT_LINE(NEST_TABLE_VAR.COUNT);
    DBE_OUTPUT.PRINT_LINE(NEST_TABLE_VAR(1)(1));
    DBE_OUTPUT.PRINT_LINE(NEST_TABLE_VAR(10)(10));
END;
/
CREATE PROCEDURE

--调用该存储过程。
gaussdb=# CALL nest_table_proc();
10
1
10
 nest_table_proc
-----------------

(1 row)
--删除存储过程。
gaussdb=# DROP PROCEDURE nest_table_proc;
DROP PROCEDURE
```

示例2：带索引的集合类型。

```
--演示在存储过程中对带索引集合进行操作。
gaussdb=# CREATE OR REPLACE PROCEDURE index_table_proc AS
DECLARE
    TYPE TABLE_INTEGER IS TABLE OF INTEGER INDEX BY INTEGER; --定义集合类型
    TYPE TABLE_VARCHAR IS TABLE OF INTEGER INDEX BY VARCHAR; --定义集合类型
    TABLEINT_01 TABLE_INTEGER;                              --声明集合类型变量,未初始化
    TABLEINT_02 TABLE_INTEGER := TABLE_INTEGER();           --声明集合类型变量,初始化为空
    TABLEINT_03 TABLE_INTEGER := TABLE_INTEGER(2=>3,3=>4);   --声明集合类型变量,初始化指定值
    RES INTEGER;
BEGIN
    FOR I IN 1..10 LOOP
        TABLEINT_01(I) := I;     --成员赋值
```



```
        TABLEINT_02(I) := I + 1; --成员赋值
    END LOOP;
    TABLEINT_01 := TABLEINT_02;  --整体赋值
    RES := TABLEINT_03(2);       --取值
    DBE_OUTPUT.PRINT_LINE(RES);
    DBE_OUTPUT.PRINT_LINE(TABLEINT_01(1));
    DBE_OUTPUT.PRINT_LINE(TABLEINT_01(10));
END;
/
CREATE PROCEDURE
--调用该存储过程。
gaussdb=# CALL index_table_proc();
3
2
11
 index_table_proc
------------------

(1 row)
--删除存储过程。
gaussdb=# DROP PROCEDURE index_table_proc;
DROP PROCEDURE

--演示在存储过程中对嵌套集合进行操作。
gaussdb=# CREATE OR REPLACE PROCEDURE nest_table_proc AS
DECLARE
    TYPE TABLE_INTEGER IS TABLE OF INTEGER INDEX BY INTEGER;          --定义集合类型
    TYPE NEST_TABLE_INTEGER IS TABLE OF TABLE_INTEGER INDEX BY INTEGER;--定义集合类型
    NEST_TABLE_VAR NEST_TABLE_INTEGER;                                --声明嵌套集合类型的变量
BEGIN
    FOR I IN 1..10 LOOP
        NEST_TABLE_VAR(I)(I) := I;
    END LOOP;
    DBE_OUTPUT.PRINT_LINE(NEST_TABLE_VAR.COUNT);
    DBE_OUTPUT.PRINT_LINE(NEST_TABLE_VAR(1)(1));
    DBE_OUTPUT.PRINT_LINE(NEST_TABLE_VAR(10)(10));
END;
/

CREATE PROCEDURE

--调用该存储过程。
gaussdb=# CALL nest_table_proc();

10
1
10
 nest_table_proc
-----------------

(1 row)

--删除存储过程。
gaussdb=# DROP PROCEDURE nest_table_proc;
DROP PROCEDURE
```

10.4.2.2 集合支持的函数

集合操作符

- =

  参数：nesttable类型

  返回值：true或false，boolean类型。

  功能描述：判断两个集合是否相等。

  示例：


```
gaussdb=# DECLARE
 type nest is table of int;
 a nest := nest(1,2);
 b nest := nest(1,2);
 flag bool;
 BEGIN
    flag := a = b;
    raise info '%', flag;
END;
 /
INFO:  t
ANONYMOUS BLOCK EXECUTE
```

- <>

  参数：nesttable类型

  返回值：true或false，boolean类型。

  功能描述：判断两个集合是否不相等。

  示例：

```
gaussdb=# DECLARE
  type nest is table of int;
  a nest := nest(1,2);
  b nest := nest(1,2);
  flag bool;
 BEGIN
   flag := a <> b;
   raise info '%', flag;
END;
/
INFO:  f
ANONYMOUS BLOCK EXECUTE
```

集合 MULTISET 函数

- MULTISET UNION [ALL | DISTINCT]

  参数：nesttable类型。

  返回值：nesttable类型。

  功能描述：两个集合变量的并集，ALL表示不去除重复元素，DISTINCT表示去除重复元素，ALL为默认值。

  示例1，求两个集合变量的并集，且不去除重复元素，即MULTISET UNION ALL：

```
gaussdb=# DECLARE
   type nest is table of int;
   a nest := nest(1,2);
   b nest := nest(2,3);
 BEGIN
   a := a MULTISET UNION ALL b;
   raise info '%', a;
 END;
 /
INFO:  {1,2,2,3}
ANONYMOUS BLOCK EXECUTE
```

  示例2，求两个集合变量的并集，且去除重复元素，即MULTISET UNION DISTINCT：

```
gaussdb=# DECLARE
    type nest is table of int;
    a nest := nest(1,2);
    b nest := nest(2,3);
 BEGIN
    a := a MULTISET UNION DISTINCT b;
    raise info '%', a;
 END;
 /
```



```
INFO:  {1,2,3}
ANONYMOUS BLOCK EXECUTE
```

- MULTISET EXCEPT [ALL | DISTINCT]

  参数：nesttable类型。

  返回值：nesttable类型。

  功能描述：两个集合变量的差集。如A MULTISET EXCEPT B：ALL表示去除A中A具有的元素且B也具有的元素并返回，其中去除动作计算元素个数。例如，对于特定元素，如果A中出现了m次，B中出现了n次，当 m > n时，则去除A中m - n个该元素，当m <= n时，则去除A中m个该元素；DISTINCT表示去除A中A具有的元素且B也具有的元素并返回，其中去除动作不计算元素个数。对于特定元素，如果在A中出现且又在B中出现了，则去除A中所有该元素。ALL为默认值。

  示例1，求两个集合变量的差集，去除A中A具有的元素且B也具有的元素并返回，其中去除动作计算元素个数。即MULTISET EXCEPT ALL：

```
gaussdb=# DECLARE
  type nest is table of int;
  a nest := nest(1,2,2);
  b nest := nest(2,3);
 BEGIN
  a := a MULTISET EXCEPT ALL b;
  raise info '%', a;
 END;
 /
INFO:  {1,2}
ANONYMOUS BLOCK EXECUTE
```

  示例2，求两个集合变量的差集，去除A中A具有的元素且B也具有的元素并返回，其中去除动作不计算元素个数。即MULTISET EXCEPT DISTINCT：

```
gaussdb=# DECLARE
  type nest is table of int;
  a nest := nest(1,2,2);
  b nest := nest(2,3);
 BEGIN
  a := a MULTISET EXCEPT DISTINCT b;
  raise info '%', a;
 END;
 /
INFO:  {1}
ANONYMOUS BLOCK EXECUTE
```

- MULTISET INTERSECT [ALL | DISTINCT]

  参数：nesttable类型。

  返回值：nesttable类型。

  功能描述：两个集合变量的交集。如 A MULTISET INTERSECT B：ALL表示取A与B所有重复的元素；DISTINCT表示取A与B中重复元素，且去除其中重复元素。ALL为默认值。

  示例1，求两个集合变量的交集，不去除重复元素，即MULTISET INTERSECT ALL：

```
gaussdb=# DECLARE
  type nest is table of int;
  a nest := nest(1,2,2);
  b nest := nest(2,2,3);
 BEGIN
  a := a MULTISET INTERSECT ALL b;
  raise info '%', a;
 END;
 /
INFO:  {2,2}
ANONYMOUS BLOCK EXECUTE
```


示例2，求两个集合变量的交集，去除重复元素，即MULTISET INTERSECT DISTINCT：

```
gaussdb=# DECLARE
  type nest is table of int;
  a nest := nest(1,2,2);
  b nest := nest(2,2,3);
 BEGIN
  a := a MULTISET INTERSECT DISTINCT b;
  raise info '%', a;
 END;
 /
INFO:  {2}
ANONYMOUS BLOCK EXECUTE
```

集合类型函数

说明

- 以下函数定义说明里[]中的内容代表可选项，如：count[()]表示可以写成count或count()。
- 在内层表达式中，不支持通过嵌套的方式调用集合类型函数。

- exists(idx)

  参数：idx为int4类型或varchar类型。

  返回值：true或false，boolean类型。

  功能描述：查找指定位置是否存在有效元素。

  示例：

```
gaussdb=# DECLARE
  type nest is table of varchar2;
  a nest := nest('happy','?');
  flag bool;
 BEGIN
  flag := a.exists(1);
  raise info '%', flag;
  flag := a.exists(10);
  raise info '%', flag;
 END;
 /
INFO:  t
INFO:  f
ANONYMOUS BLOCK EXECUTE

gaussdb=# DECLARE
  type nest is table of varchar2 index by varchar2;
  a nest;
  flag bool;
 BEGIN
  a('1') := 'Be';
  a('2') := 'happy';
  a('3') := '.';
  flag := a.exists('1');
  raise info '%', flag;
  flag := a.exists('ddd');
  raise info '%', flag;
 END;
 /
INFO:  t
INFO:  f
ANONYMOUS BLOCK EXECUTE
```

- extend[(count[, idx])]

  参数：idx和count为int4类型。

  返回值：无返回值。

功能描述：extend函数仅支持nesttable类型变量使用。在nesttable变量末尾拓展1个或count个元素。存在idx下标元素时，复制count个idx下标元素到变量末尾。

示例1，nesttable类型变量扩展1个元素且所扩展的元素值为NULL：

```
gaussdb=# DECLARE
  type nest is table of int;
  a nest := nest(1);
 BEGIN
  raise info '%', a;
  a.extend;
  raise info '%', a;
  a.extend;
  raise info '%', a;
 END;
 /
INFO:  {1}
INFO:  {1,NULL}
INFO:  {1,NULL,NULL}
ANONYMOUS BLOCK EXECUTE
```

示例2，nesttable类型变量扩展n个元素且所扩展的元素值为NULL：

```
gaussdb=# DECLARE
  type nest is table of int;
  a nest := nest(1);
 BEGIN
  raise info '%', a;
  a.extend(2);
  raise info '%', a;
 END;
 /
INFO:  {1}
INFO:  {1,NULL,NULL}
ANONYMOUS BLOCK EXECUTE
```

示例3，nesttable类型变量扩展n个元素且所扩展的元素值为指定下标元素的值：

```
gaussdb=# DECLARE
  type nest is table of int;
  a nest := nest(9);
 BEGIN
  raise info '%', a;
  a.extend(2,1);
  raise info '%', a;
 END;
 /
INFO:  {9}
INFO:  {9,9,9}
ANONYMOUS BLOCK EXECUTE
```

- delete[(idx1[, idx2])]

  参数：idx1和idx2为int4类型或varchar2类型。

  返回值：无返回值。

  功能描述：无参数时，无索引集合nesttable类型变量删除集合类型的所有元素和空间，后续使用需要重新扩展空间，带索引集合indexbytable类型变量删除所有元素内容；一个参数删除指定位置元素（不删除空间）；两个参数则删除下标区间内的所有元素（不删除空间）。

  示例1，无索引集合nesttable类型变量删除集合类型的所有元素和空间：

```
gaussdb=# DECLARE
  type nest is table of int;
  a nest := nest(1,2,3,4,5);
 BEGIN
  raise info '%', a;
  a.delete();
  raise info '%', a;
 END;
 /
```



```
INFO:  {1,2,3,4,5}
INFO:  {}
ANONYMOUS BLOCK EXECUTE
```

示例2，无索引集合nesttable类型变量删除指定位置元素：

```
gaussdb=# DECLARE
  type nest is table of int;
  a nest := nest(1,2,3,4,5);
 BEGIN
  raise info '%', a;
  a.delete(3);
  raise info '%', a;
  a(3) := 3;
  raise info '%', a;
 END;
 /
INFO:  {1,2,3,4,5}
INFO:  {1,2,4,5}
INFO:  {1,2,3,4,5}
ANONYMOUS BLOCK EXECUTE
```

示例3，无索引集合nesttable类型变量删除指定区间元素：

```
gaussdb=# DECLARE
  type nest is table of int;
  a nest := nest(1,2,3,4,5);
 BEGIN
  raise info '%', a;
  a.delete(2,4);
  raise info '%', a(1);
  raise info '%', a(5);
  raise info '%', a;
 END;
 /
INFO:  {1,2,3,4,5}
INFO:  1
INFO:  5
INFO:  {1,5}
ANONYMOUS BLOCK EXECUTE
```

示例4，带索引集合indexbytable类型变量删除集合类型的所有元素和空间：

```
gaussdb=# DECLARE
  type t1 is table of int index by varchar;
  v t1 := t1('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4);
 BEGIN
  v.delete();
  raise info '%', v.count();
 END;
 /
INFO:  0
ANONYMOUS BLOCK EXECUTE
```

示例5，带索引集合indexbytable类型变量删除指定位置元素：

```
gaussdb=# DECLARE
  type t1 is table of int index by varchar;
  v t1 := t1('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4);
 BEGIN
  raise info '%', v('a');
  v.delete('a');
  raise info '%', v('a');
 END;
 /
INFO:  1
ERROR:  no data found
CONTEXT:  PL/pgSQL function inline_code_block line 6 at RAISE
```

示例6，带索引集合indexbytable类型变量删除指定区间元素：

```
gaussdb=# DECLARE
  type t1 is table of int index by varchar;
  v t1 := t1('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4);
```



```
BEGIN
  raise info '%', v('b');
  v.delete('a', 'c');
  raise info '%', v('b');
END;
/
INFO:  2
ERROR:  no data found
CONTEXT:  PL/pgSQL function inline_code_block line 6 at RAISE
```

- trim[(n)]

  参数：n为int4类型。

  返回值：无返回值。

  功能描述：trim函数仅支持nesttable类型变量使用。无参数时，删除末尾一个元素空间，输入参数合法时，删除末尾指定数量元素空间。

  示例：

```
gaussdb=# DECLARE
  type nest is table of int;
  aa nest := nest(11,22,33,44,55);
BEGIN
  raise info 'aa:%' ,aa;
  aa.trim;  -- 无参数
  raise info 'aa:%' ,aa;
  aa.trim();  -- 无参数
  raise info 'aa:%' ,aa;
  aa.trim(2);  -- 参数合法
  raise info 'aa:%' ,aa;
  aa.trim(2);  -- 集合元素空间不足2, 参数不合法报错
END;
/
INFO:  aa:{11,22,33,44,55}
INFO:  aa:{11,22,33,44}
INFO:  aa:{11,22,33}
INFO:  aa:{11}
ERROR:  Subscript beyond count
CONTEXT:  PL/pgSQL function inline_code_block line 11 at assignment
```

- count[()]

  参数：无。

  返回值：int类型。

  功能描述：返回集合中存在有效元素的个数。

  示例1，无索引集合类型变量使用count函数：

```
gaussdb=# DECLARE
 type nest is table of int;
 aa nest:=nest(11,22,33,44,55);
 BEGIN
 raise info 'count:%' ,aa.count;
 aa.delete(3);  -- 删除一个元素，下标为3的元素无效
 raise info 'count:%' ,aa.count();
 END;
 /
INFO:  count:5
INFO:  count:4
ANONYMOUS BLOCK EXECUTE
```

  示例2，带索引集合类型变量使用count函数：

```
gaussdb=# DECLARE
 type t1 is table of int index by int;
 aa t1;
 BEGIN
 aa(1) := 111;
 aa(2) := 222;
 aa(3) := 333;
```



```
 raise info 'count:%' ,aa.count();
 END;
 /
INFO:  count:3
ANONYMOUS BLOCK EXECUTE

gaussdb=# DECLARE
 type t1 is table of int index by varchar;
 aa t1;
 BEGIN
 aa('aaa') := 111;
 aa('bbb') := 222;
 aa('ccc') := 333;
 raise info 'count:%' ,aa.count;
 END;
 /
INFO:  count:3
ANONYMOUS BLOCK EXECUTE
```

- first[()]

  参数：无。

  返回值：int类型或varchar类型。

  功能描述：返回集合中第一个有效元素的下标。

  示例1，无索引集合类型变量使用first函数：

```
gaussdb=# DECLARE
 type nest is table of int;
 aa nest:=nest(11,22,33,44,55);
 BEGIN
 raise info 'first:%' ,aa.first();
 aa.delete(1);
 raise info 'first:%' ,aa.first;  -- 下标1的元素无效，第一个有效元素下标为2
 END;
 /
INFO:  first:1
INFO:  first:2
ANONYMOUS BLOCK EXECUTE
```

  示例2，带索引集合类型变量使用first函数：

```
gaussdb=# DECLARE
 type t1 is table of int index by int;
 aa t1;
 BEGIN
 aa(3) := 111;
 aa(2) := 222;
 aa(1) := 333;
 raise info 'first:%' ,aa.first;
 END;
 /
INFO:  first:1
ANONYMOUS BLOCK EXECUTE

gaussdb=# DECLARE
 type t1 is table of int index by varchar;
 aa t1;
 BEGIN
 aa('aaa') := 111;
 aa('bbb') := 222;
 aa('ccc') := 333;
 raise info 'first:%' ,aa.first;
 END;
 /
INFO:  first:aaa
ANONYMOUS BLOCK EXECUTE
```

- last[()]

  参数：无。

返回值：int类型或varchar类型。

功能描述：返回集合中最后一个有效元素的下标。

示例1，无索引集合类型变量使用last函数：

```
gaussdb=# DECLARE
 type nest is table of int;
 aa nest:=nest(11,22,33,44,55);
 BEGIN
 raise info 'last:%' ,aa.last;
 aa.delete(5);
 raise info 'last:%' ,aa.last();  -- 下标5的元素无效，最后一个有效元素下标为4
 END;
 /
INFO:  last:5
INFO:  last:4
ANONYMOUS BLOCK EXECUTE
```

示例2，带索引集合类型变量使用last函数：

```
gaussdb=# DECLARE
 type t1 is table of int index by varchar;
 aa t1;
 BEGIN
 aa(3) := 111;
 aa(2) := 222;
 aa(1) := 333;
 raise info 'last:%' ,aa.last();
 END;
 /
INFO:  last:3
ANONYMOUS BLOCK EXECUTE

gaussdb=# DECLARE
 type t1 is table of int index by varchar;
 aa t1;
 BEGIN
 aa('aaa') := 111;
 aa('bbb') := 222;
 aa('ccc') := 333;
 raise info 'last:%' ,aa.last;
 END;
 /
INFO:  last:ccc
ANONYMOUS BLOCK EXECUTE
```

- prior(idx)

  参数：idx为int类型或varchar类型。

  返回值：int类型或varchar类型。

  功能描述：返回集合中给定下标的前一个有效元素下标。

  示例1，无索引集合类型变量使用prior函数：

  ```
  gaussdb=# DECLARE
   type nest is table of int;
   aa nest:=nest(11,22,33,44,55);
   BEGIN
   raise info 'prior:%' ,aa.prior(3);
   END;
   /
  INFO:  prior:2
  ANONYMOUS BLOCK EXECUTE
  ```

  示例2，带索引集合类型变量使用prior函数：

  ```
  gaussdb=# DECLARE
   type t1 is table of int index by varchar;
   aa t1;
   BEGIN
   aa('ccc') := 111;
  ```



```
 aa('bbb') := 222;
 aa('aaa') := 333;
 raise info 'prior:%' ,aa.prior('bbb');
 raise info 'prior:%' ,aa.prior('ddd');
 END;
 /
INFO:  prior:aaa
INFO:  prior:ccc
ANONYMOUS BLOCK EXECUTE
```

- next(idx)

  参数：idx为int类型或varchar类型。

  返回值：int类型或varchar类型。

  功能描述：返回集合中给定下标的后一个有效元素下标。

  示例1，无索引集合类型变量使用next函数：

```
gaussdb=# DECLARE
 type nest is table of int;
 aa nest:=nest(11,22,33,44,55);
 BEGIN
 raise info 'next:%' ,aa.next(3);
 END;
 /
INFO:  next:4
ANONYMOUS BLOCK EXECUTE
```

  示例2，带索引集合类型变量使用next函数：

```
gaussdb=# DECLARE
 type t1 is table of int index by int;
 aa t1;
 BEGIN
 aa(3) := 111;
 aa(2) := 222;
 aa(1) := 333;
 raise info 'next:%' ,aa.next(2);
 raise info 'next:%' ,aa.next(-999);
 END;
 /
INFO:  next:3
INFO:  next:1
ANONYMOUS BLOCK EXECUTE
```

- limit

  参数：无。

  返回值：NULL。

  功能描述：仅用于nesttable类型变量，且返回null值。

  示例：

```
gaussdb=# DECLARE
 type nest is table of int;
 aa nest:=nest(11,22,33,44,55);
 BEGIN
 raise info 'limit:%' ,aa.limit;
 END;
 /
INFO:  limit:<NULL>
ANONYMOUS BLOCK EXECUTE
```

集合相关函数

- unnest_table(anynesttable)或unnest(anynesttable)

  参数：任意无索引集合类型。

  描述：返回给定nesttable中所有有效元素的结果集，会返回多行数据。

返回类型：setof anyelement。

示例：

```
gaussdb=# create or replace procedure f1()
as
    type t1 is table of int;
    v2 t1 := t1(null, 2, 3, 4, null);
    tmp int;
    cursor c1 is select * from unnest_table(v2);
begin
open c1;
for i in 1 .. v2.count loop
    fetch c1 into tmp;
    if tmp is null then
        dbe_output.print_line(i || ': is null');
    else
        dbe_output.print_line(i || ': ' || tmp);
    end if;
end loop;
close c1;
end;
/

gaussdb=# call f1();
1: is null
2: 2
3: 3
4: 4
5: is null
 f1
----

(1 row)

-- nesttable嵌套record类型示例
gaussdb=# create or replace procedure p1() is
  type rec is record(c1 int, c2 int);
  type t1 is table of rec;

  v t1 := t1(rec(1, 1), rec(2, null), rec(null, null), null);
  v2 t1 := t1();
  cursor cur is select * from unnest(v);
begin
  v2.extend(v.count);
  open cur;
  for i in 1 .. v.count loop
    fetch cur into v2(i);
    raise info '%', v2(i);
  end loop;
  close cur;
end;
/

gaussdb=# call p1();
INFO:  (1,1)
INFO:  (2,)
INFO:  (,)
INFO:  (,)
 p1
----

(1 row)

gaussdb=# drop procedure if exists p1();
DROP PROCEDURE
```


说明

当集合的元素类型为record类型且有元素为NULL时，该元素不会被返回，而是会返回一个非NULL的record值，该record的所有列为NULL，具体可参考示例。

- unnest_table(anyindexbytable)或unnest(anyindexbytable)

  参数：任意带索引集合类型。

  描述：返回给定indexbytable中所有元素根据索引排序后的结果集，会返回多行数据。

  返回类型：setof anyelement。

  约束：只支持index by int类型，不支持index by varchar类型。

  示例：

```
gaussdb=# create or replace procedure p1()
as
    type t1 is table of int index by int;
    v2 t1 := t1(1=>1, -10=>(-10), 6=>6, 4=>null);
    tmp int;
    cursor c1 is select * from unnest_table(v2);
begin
open c1;
for i in 1 .. v2.count loop
    fetch c1 into tmp;
    if tmp is null then
        dbe_output.print_line(i || ': is null');
    else
        dbe_output.print_line(i || ': ' || tmp);
    end if;
end loop;
close c1;
end;
/

gaussdb=# call p1();
1: -10
2: 1
3: is null
4: 6
 p1
----


(1 row)

-- index by table嵌套record类型示例
gaussdb=# create or replace procedure p1() is
  type rec is record(c1 int, c2 int);
  type t1 is table of rec index by int;

  v t1 := t1(1 => rec(1, 1), 2 => rec(2, null), 3 => rec(null, null), 4 => null);
  v2 t1 := t1();
  cursor cur is select * from unnest(v);
begin
  open cur;
  for i in 1 .. v.count loop
    fetch cur into v2(i);
    raise info '%', v2(i);
  end loop;
  close cur;
END;
/

gaussdb=# call p1();
INFO:  (1,1)
INFO:  (2,)
INFO:  (,)
INFO:  (,)
```



```
 p1
----

(1 row)

gaussdb=# drop procedure if exists p1();
DROP PROCEDURE
```

说明

当集合的元素类型为record类型且有元素为NULL时，该元素不会被返回，而是会返回一个非NULL的record值，该record的所有列为NULL，具体可参考示例。

10.4.3 record

record 类型的变量

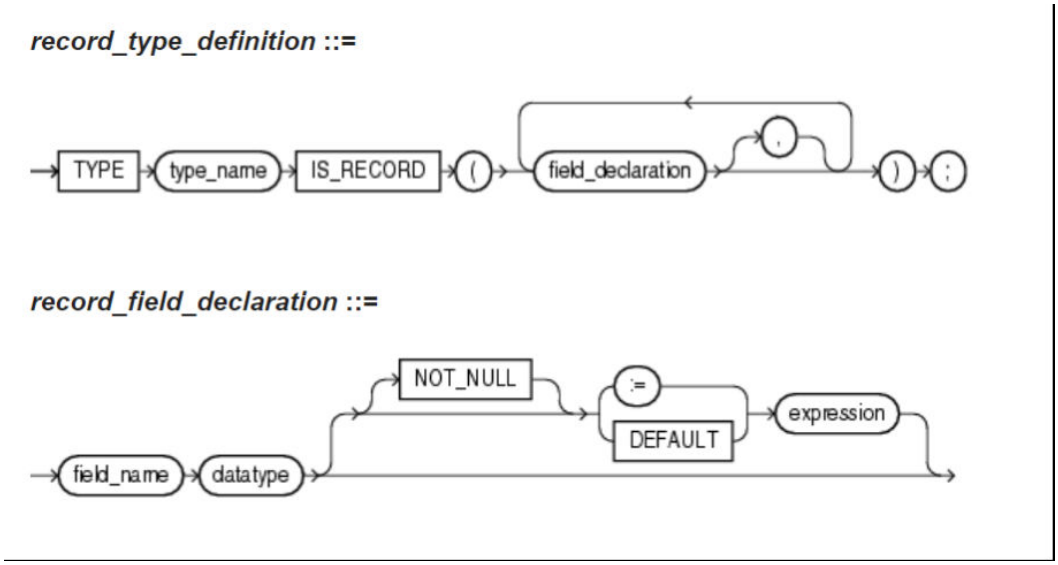
创建一个record变量的方式：

定义一个record类型 ，然后使用该类型来声明一个变量。

语法

record类型的语法参见图10-1。

图 10-1 record 类型的语法



对以上语法格式的解释如下：

- type_name：声明的类型名称。
- field_name：record类型中的成员名称。
- datatype：record类型中成员的类型。
- expression：设置默认值的表达式。

说明

在GaussDB中：

- record类型变量的赋值支持：
  - 在函数或存储过程的声明阶段，声明一个record类型，并且可以在该类型中定义成员变量。
  - 一个record变量到另一个record变量的赋值。
  - SELECT INTO和FETCH向一个record类型的变量中赋值。
  - 将一个NULL值赋值给一个record变量。
- 不支持INSERT和UPDATE语句使用record变量进行插入数据和更新数据。
- 不支持record类型的构造器作为函数或存储过程参数的默认值。
- 通过package_name.record_type声明record变量时，record类型的默认值功能会失效，建议不要在record类型有默认值时使用package_name.record_type声明record变量。
- 如果成员有复合类型，在声明阶段不支持指定默认值，该行为同声明阶段的变量一样。
- 如果成员有record类型，则内层record类型的默认值不会传递至外层record类型中。
- 设置enable_recordtype_check_strict = on后，成员是record类型，且record类型有列具有not null属性或default属性，在存储过程或PACKAGE编译时会报错。
- 设置enable_recordtype_check_strict = on后，当PACKAGE里有record类型，且record类型有列具有not null属性或default属性会在创建或编译时报错。
- 设置enable_recordtype_check_strict = on后，存储过程里的record类型not null功能生效。
- datatype可以为存储过程中定义record类型、数组类型和集合类型（匿名块不支持）。
- 打开guc参数 set behavior_compat_options = 'proc_outparam_override'; 后：
  - 带有out出参且返回record的函数，支持在外部SQL使用SELECT、CALL调用，在存储过程内支持使用PERFORM、表达式调用。
  - 在函数直接返回未定义的record类型时，至少需要带有一个out参数；如果返回的是已经定义的record类型，则可以不带out参数。详见示例。
  - 带out出参返回复合类型或record的函数，需要保持函数的预期返回类型与实际返回类型一致。

示例

```
--创建一个表，并插入一些数据：
gaussdb=# CREATE TABLE emp_rec(
gaussdb(#     empno numeric(4,0) not null,
gaussdb(#     ename varchar(10)
gaussdb(# );
CREATE TABLE
gaussdb=# INSERT INTO emp_rec VALUES(111, 'aaa'), (222, 'bbb'), (333, 'ccc');
INSERT 0 3
-- 表定义如下：
gaussdb=# \d emp_rec
          Table "public.emp_rec"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 empno  | numeric(4,0)          | not null
 ename  | character varying(10) |

--演示在函数中对record进行操作。
gaussdb=# CREATE OR REPLACE FUNCTION regress_record(p_w VARCHAR2) RETURNS VARCHAR2 AS $$
gaussdb$# DECLARE
gaussdb$#     --声明一个record类型.
gaussdb$#     type rec_type is record (name  varchar2(100), epno int);
gaussdb$#     employer rec_type;
gaussdb$#     --使用%type声明record类型
gaussdb$#     type rec_type1 is record (name  emp_rec.ename%type, epno int :=10);
gaussdb$#     employer1 rec_type1;
gaussdb$#     --声明带有默认值的record类型
```



```
gaussdb$#     type rec_type2 is record (
gaussdb$#         name varchar2 not null := 'SCOTT',
gaussdb$#         epno int not null :=10
gaussdb$#     );
gaussdb$#     employer2 rec_type2;
gaussdb$#     CURSOR C1 IS  select ename,empno from emp_rec order by 1 limit 1;
gaussdb$# BEGIN
gaussdb$#     --对一个record类型的变量的成员赋值。
gaussdb$#     employer.name := 'WARD';
gaussdb$#     employer.epno = 18;
gaussdb$#     raise info 'employer name: % , epno:%', employer.name, employer.epno;
gaussdb$#
gaussdb$#     --将一个record类型的变量赋值给另一个变量。
gaussdb$#     employer1 := employer;
gaussdb$#     raise info 'employer1 name: % , epno: %',employer1.name, employer1.epno;
gaussdb$#
gaussdb$#     --将一个record类型变量赋值为NULL。
gaussdb$#     employer1 := NULL;
gaussdb$#     raise info 'employer1 name: % , epno: %',employer1.name, employer1.epno;
gaussdb$#
gaussdb$#     --获取record变量的默认值。
gaussdb$#     raise info 'employer2 name: % ,epno: %', employer2.name, employer2.epno;
gaussdb$#
gaussdb$#     --在for循环中使用record变量
gaussdb$#     for employer in select ename,empno from emp_rec order by 1 limit 1 loop
gaussdb$#         raise info 'employer name: % , epno: %', employer.name, employer.epno;
gaussdb$#     END loop;
gaussdb$#
gaussdb$#     --在select into 中使用record变量。
gaussdb$#     select ename,empno  into employer2 from emp_rec order by 1 limit 1;
gaussdb$#     raise info 'employer name: % , epno: %', employer2.name, employer2.epno;
gaussdb$#
gaussdb$#     --在cursor中使用record变量。
gaussdb$#     OPEN C1;
gaussdb$#     FETCH C1 INTO employer2;
gaussdb$#     raise info 'employer name: % , epno: %', employer2.name, employer2.epno;
gaussdb$#     CLOSE C1;
gaussdb$#     RETURN employer.name;
gaussdb$# END; $$
gaussdb-# LANGUAGE plpgsql;
CREATE FUNCTION

--调用该函数。
gaussdb=# CALL regress_record('abc');
INFO:  employer name: WARD , epno:18
INFO:  employer1 name: WARD , epno: 18
INFO:  employer1 name: <NULL> , epno: <NULL>
INFO:  employer2 name: SCOTT ,epno: 10
INFO:  employer name: aaa , epno: 111
INFO:  employer name: aaa , epno: 111
INFO:  employer name: aaa , epno: 111
 regress_record
----------------
 aaa
(1 row)
--删除函数。
gaussdb=# DROP FUNCTION regress_record;
DROP FUNCTION
--删除表
gaussdb=# DROP TABLE emp_rec;
DROP TABLE

-- 打开兼容性参数proc_outparam_override时，返回已定义的record类型，函数可以不需out参数
gaussdb=# CREATE TYPE rec_type IS (c1 int, c2 int);
CREATE TYPE
gaussdb=# SET behavior_compat_options = 'proc_outparam_override';
SET
gaussdb=# CREATE OR REPLACE FUNCTION func(a in int) RETURN rec_type IS
gaussdb$#     r rec_type;
```



```
gaussdb$# BEGIN
gaussdb$#     r.c1:=1;
gaussdb$#     r.c2:=1;
gaussdb$#     return r;
gaussdb$# END;
gaussdb$# /
CREATE FUNCTION
gaussdb=# CALL func(0);
 c1 | c2
----+----
  1 |  1
(1 row)
gaussdb=# DROP FUNCTION func;
DROP FUNCTION
gaussdb=# drop type rec_type;
DROP TYPE
-- 打开兼容性参数proc_outparam_override时，函数直接返回未定义的record类型时，至少需要带有一个out参数
gaussdb=# SET behavior_compat_options = 'proc_outparam_override';
SET
gaussdb=# CREATE OR REPLACE FUNCTION func(a out int) RETURN record IS
gaussdb$#     type rc is record(c1 int, c2 int);
gaussdb$#     r rc;
gaussdb$# BEGIN
gaussdb$#     r.c1 := 1;
gaussdb$#     r.c2 := 1;
gaussdb$#     a := 999;
gaussdb$#     return r;
gaussdb$# END;
gaussdb$# /
CREATE FUNCTION
gaussdb=# CALL func(1);
 func  |  a
-------+-----
 (1,1) | 999
(1 row)
gaussdb=# DROP FUNCTION func;
DROP FUNCTION
```

10.5 声明语法

10.5.1 基本结构

结构

PL/SQL块中可以包含子块，子块可以位于PL/SQL中任何部分。PL/SQL块的结构如下：

- 声明部分：声明PL/SQL用到的变量、类型、游标、局部的存储过程和函数。
  ```
  DECLARE
  ```

  说明

  不涉及变量声明时声明部分可以没有。

  - 对匿名块来说，没有变量声明部分时，可以省去DECLARE关键字。
  - 对存储过程来说，没有DECLARE， AS相当于DECLARE。即便没有变量声明的部分，关键字AS也必须保留。
- 执行部分：过程及SQL语句，程序的主要部分。必选。
  ```
  BEGIN
  ```
- 执行异常部分：错误处理。可选。
  ```
  EXCEPTION
  ```


- 结束。必选。
```
END;
/
```

须知

禁止在PL/SQL块中使用连续的Tab，连续的Tab可能会造成在使用gsql工具带“-r”参数执行PL/SQL块时出现异常。

分类

PL/SQL块可以分为以下几类：

- 匿名块：动态构造，只能执行一次。语法请参考图10-2。
- 子程序：存储在数据库中的存储过程、函数、操作符和高级包等。当在数据库上建立好后，可以在其他程序中调用它们。

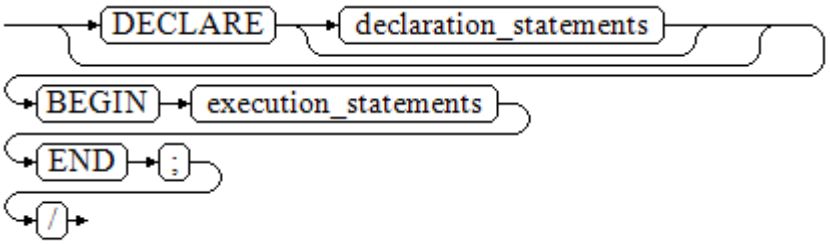
10.5.2 匿名块

匿名块（Anonymous Block）一般用于不频繁执行的脚本或不重复进行的活动。它们在一个会话中执行，并不被存储。

语法

匿名块的语法参见图10-2。

图 10-2 anonymous_block::=



对以上语法图的解释如下：

- 匿名块程序实施部分，以BEGIN语句开始，以END语句停顿，以一个分号结束。输入“/”按回车执行它。

须知

最后的结束符“/”必须独占一行，不能直接跟在END后面。

- 声明部分包括变量定义、类型、游标定义等。
- 最简单的匿名块不执行任何命令。但一定要在任意实施块里至少有一个语句，甚至是一个NULL语句。

示例

下面列举了基本的匿名块程序：

```
--空语句块
gaussdb=# BEGIN
    NULL;
END;
/
ANONYMOUS BLOCK EXECUTE
--将信息打印到控制台：
gaussdb=# BEGIN
    dbe_output.print_line('hello world!');
END;
/
hello world!
ANONYMOUS BLOCK EXECUTE
--将变量内容打印到控制台：
gaussdb=# DECLARE
    my_var VARCHAR2(30);
BEGIN
    my_var :='world';
    dbe_output.print_line('hello'||my_var);
END;
/
helloworld
ANONYMOUS BLOCK EXECUTE
```

10.5.3 子程序

存储在数据库中的存储过程、函数、操作符和高级包等。当在数据库上建立好后，可以在其他程序中调用它们。

10.6 基本语句

在编写PL/SQL过程中，会定义一些变量，给变量赋值，调用其他存储过程等。介绍PL/SQL中的基本语句，包括定义变量、赋值语句、调用语句以及返回语句。

说明

尽量不要在存储过程中调用包含密码的SQL语句，因为存储在数据库中的存储过程文本可能被其他有权限的用户看到导致密码信息被泄漏。如果存储过程中包含其他敏感信息也需要配置存储过程的访问权限，保证敏感信息不会泄漏。

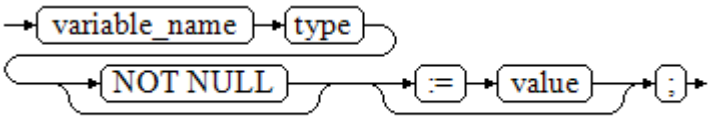
10.6.1 定义变量

介绍PL/SQL中变量的声明，以及该变量在代码中的作用域。

变量声明

变量声明语法请参见图10-3。

图 10-3 declare_variable::=



对以上语法格式的解释如下：

- variable_name：变量名。
- type：变量类型。
- value：该变量的初始值（如果不给定初始值，则初始为NULL）。value也可以是表达式。

示例

```
gaussdb=# DECLARE
    emp_id  INTEGER := 7788;  --定义变量并赋值
BEGIN
    emp_id := 5*7784;  --变量赋值
END;
/
ANONYMOUS BLOCK EXECUTE
```

变量类型除了支持基本类型，还可以是使用%TYPE和%ROWTYPE去声明一些与其他表字段或表结构本身相关的变量。

%TYPE属性

%TYPE主要用于声明某个与其他变量类型（例如，表中某列的类型）相同的变量。若想定义一个my_name变量，它的变量类型与employee的firstname类型相同，可以通过如下定义：

```
my_name employee.firstname%TYPE
--示例
DROP TABLE IF EXISTS employee;
NOTICE:  table "employee" does not exist, skipping
DROP TABLE
CREATE TABLE employee(firstname varchar,secondname varchar);
CREATE TABLE
DECLARE
    my_name employee.firstname%TYPE;
BEGIN
    my_name = 'abc';
    DBE_OUTPUT.PRINT_LINE(my_name);
END;
/
abc
ANONYMOUS BLOCK EXECUTE
```

这样定义可以带来两个好处，首先，不用预先知道employee 表的firstname类型具体是什么。其次，即使之后firstname类型有了变化，也不需要再次修改my_name的类型。

```
TYPE employee_record is record (id INTEGER, firstname VARCHAR2(20));
my_employee employee_record;
my_id my_employee.id%TYPE;
my_id_copy my_id%TYPE;
--示例
DECLARE
    TYPE employee_record is record (id INTEGER, firstname VARCHAR2(20));
    my_employee employee_record;
    my_id my_employee.id%TYPE;
    my_id_copy my_id%TYPE;
BEGIN
    my_employee.id := 1;
    my_employee.firstname := 'ab2';
    my_id = 2;
    my_id_copy = 3;
    DBE_OUTPUT.PRINT_LINE(my_employee.id);
    DBE_OUTPUT.PRINT_LINE(my_employee.firstname);
    DBE_OUTPUT.PRINT_LINE(my_id);
    DBE_OUTPUT.PRINT_LINE(my_id_copy);
```



```
END;
/
1
ab2
2
3
ANONYMOUS BLOCK EXECUTE
```

%ROWTYPE属性

%ROWTYPE属性主要用于对一组数据的类型声明，用于存储表中的一行数据或从游标匹配的结果。假如，需要一组数据，该组数据的字段名称与字段类型都与employee表相同。可以通过如下定义：

```
my_employee employee%ROWTYPE
--示例
DROP TABLE IF EXISTS employee;
DROP TABLE
CREATE TABLE employee(firstname varchar,secondname varchar);
DECLARE
    my_employee employee%ROWTYPE;
BEGIN
    my_employee.firstname := 'ab1';
    my_employee.secondname := 'ab2';
    DBE_OUTPUT.PRINT_LINE(my_employee.firstname);
    DBE_OUTPUT.PRINT_LINE(my_employee.secondname);
END;
/
ab1
ab2
ANONYMOUS BLOCK EXECUTE
```

同样可以使用在cursor上面，该组数据的字段名称与字段类型都与employee表相同（对于PACKAGE中的cursor，可以省略%ROWTYPE）。%TYPE也可以引用cursor中某一列的类型，可以通过如下定义：

```
CURSOR cur IS SELECT * FROM employee;
my_employee cur%ROWTYPE
my_name cur.firstname%TYPE
my_employee2 cur -- 对于PACKAGE中定义的cursor，可以省略%ROWTYPE字段
--示例
SET behavior_compat_options = 'allow_procedure_compile_check';
SET
DROP TABLE IF EXISTS employee;
DROP TABLE
CREATE TABLE employee(firstname varchar,secondname varchar);
CREATE TABLE
CREATE OR REPLACE procedure proc1()
IS
    CURSOR cur IS SELECT * FROM employee;
    my_employee cur%ROWTYPE;
    my_name cur.firstname%TYPE;
BEGIN
    my_employee.firstname := 'ab1';
    my_employee.secondname := 'ab2';
    my_name := 'ab3';
    DBE_OUTPUT.PRINT_LINE(my_employee.firstname);
    DBE_OUTPUT.PRINT_LINE(my_employee.secondname);
    DBE_OUTPUT.PRINT_LINE(my_name);
END;
/
CREATE PROCEDURE
```


须知

- %TYPE不支持引用复合类型或RECORD类型变量的类型、RECORD类型的某列类型、跨PACKAGE复合类型变量的某列类型、跨PACKAGE cursor变量的某列类型等。
- %ROWTYPE不支持引用复合类型或RECORD类型变量的类型、跨PACKAGE cursor的类型。
- 若使用%TYPE操作从record变量.column中取得类型，且该record变量.column的类型由%TYPE操作定义，则不会保留原始类型的约束（带约束的数据类型如NUMBER(3)、VARCHAR2(10)等）。

变量作用域

变量的作用域表示变量在代码块中的可访问性和可用性。只有在它的作用域内，变量才有效。

- 变量必须在declare部分声明，即必须建立BEGIN-END块。块结构也强制变量必须先声明后使用，即变量在过程内有不同作用域、不同的生存期。
- 同一变量可以在不同的作用域内定义多次，内层的定义会覆盖外层的定义。
- 在外部块定义的变量，可以在嵌套块中使用。但外部块不能访问嵌套块中的变量。

示例

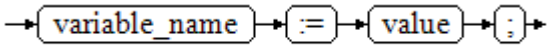
```
gaussdb=# DECLARE
    emp_id  INTEGER :=7788;  --定义变量并赋值
    outer_var  INTEGER :=6688;  --定义变量并赋值
BEGIN
    DECLARE
        emp_id INTEGER :=7799;  --定义变量并赋值
        inner_var  INTEGER :=6688;  --定义变量并赋值
    BEGIN
        dbe_output.print_line('inner emp_id ='||emp_id);  --显示值为7799
        dbe_output.print_line('outer_var ='||outer_var);  --引用外部块的变量
    END;
    dbe_output.print_line('outer emp_id ='||emp_id);  --显示值为7788
END;
/
```

10.6.2 赋值语句

变量语法

给变量赋值的语法请参见图10-4。

图 10-4 assignment_value::=



对以上语法格式的解释如下：

- variable_name：变量名。
- value：可以是值或表达式。值value的类型需要和变量variable_name的类型兼容才能正确赋值。

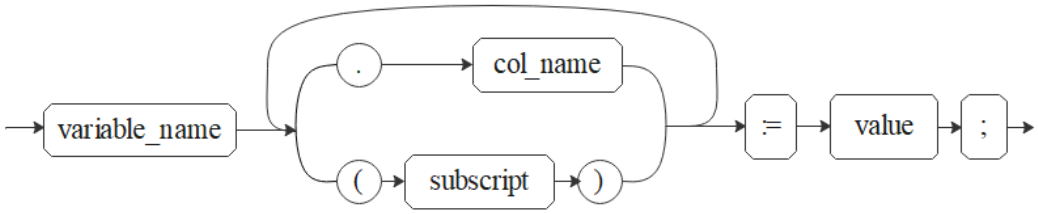
示例：

```
gaussdb=# DECLARE
    emp_id  INTEGER := 7788;--赋值
BEGIN
    emp_id := 5;--赋值
    DBE_OUTPUT.PRINT_LINE(emp_id);
    emp_id := 5*7784;
    DBE_OUTPUT.PRINT_LINE(emp_id);
END;
/
--结果如下：
5
38920
ANONYMOUS BLOCK EXECUTE
```

嵌套赋值

给变量嵌套赋值的语法请参见图10-5。

图 10-5 nested_assignment_value::=



对以上语法格式的解释如下：图10-5

- variable_name：变量名。
- col_name：列名。
- subscript：下标，针对数组变量使用，可以是值或表达式，类型必须为int。
- value：可以是值或表达式。值value的类型需要和变量variable_name的类型兼容才能正确赋值。

示例：

```
gaussdb=# CREATE TYPE o1 AS (a int, b int);
CREATE TYPE
gaussdb=# DECLARE
    TYPE r1 is VARRAY(10) of o1;
    emp_id  r1;
BEGIN
    emp_id(1).a := 5;--赋值
    emp_id(1).b := 5*7784;
END;
/
ANONYMOUS BLOCK EXECUTE
```

须知

INTO方式赋值仅支持对第一层列赋值，且不支持二维及以上数组。

INTO/BULK COLLECT INTO

- 将存储过程内语句返回的值存储到变量内，BULK COLLECT INTO允许将部分或全部返回值暂存到数组内部。
- 支持返回空结果集。

语法格式

```
SELECT select_expressions INTO [STRICT] target FROM ...
SELECT INTO [STRICT] target expression [FROM ..]
```

说明

- 通过基础 SQL 命令加INTO子句可以将单行或多列的结果赋值给一个变量（记录、行类型、标量变量列表）。
- target参数可以是一个记录变量、一个行变量或一个有逗号分隔的简单变量和记录/行域列表。
- STRICT选项，在开启参数set behavior_compat_options = 'select_into_return_null'的前提下（默认未开启），若指定该选项则该查询必须刚好返回一行不为空的结果集，否则会报错，报错信息可能是NO_DATA_FOUND（没有行）、TOO_MANY_ROWS（多于一行）或QUERY_RETURNED_NO_ROWS (没有数据返回)。若不指定该选项则没有该限定，且支持返回空结果集。
- BULK COLLECT INTO只支持在A兼容性数据库下使用。

示例：

```
gaussdb=# DROP TABLE IF EXISTS customers;
NOTICE:  table "customers" does not exist, skipping
DROP TABLE
gaussdb=# CREATE TABLE customers(id int,name varchar);
CREATE TABLE
gaussdb=# INSERT INTO customers VALUES(1,'ab');
gaussdb=# DECLARE
    my_id integer;
BEGIN
    select id into my_id from customers limit 1; -- 赋值
END;
/
ANONYMOUS BLOCK EXECUTE

gaussdb=# DECLARE
    type id_list is varray(6) of customers.id%type;
    id_arr id_list;
BEGIN
    select id bulk collect into id_arr from customers order by id DESC limit 20; -- 批量赋值
END;
/
ANONYMOUS BLOCK EXECUTE

gaussdb=# CREATE TABLE test(a integer);
CREATE TABLE
gaussdb=# insert into test values(1);
INSERT 0 1
gaussdb=# CREATE OR REPLACE FUNCTION check_test() RETURNS integer
    language plpgsql
    AS $function$
    DECLARE
     b integer;
    BEGIN
     SELECT INTO b a FROM test WHERE a=1; -- 返回空结果集
           RETURN b;
    END;
    $function$;
gaussdb=# SELECT check_test();
 check_test
------------
```



```
      1
(1 row)
gaussdb=# DROP TABLE customers;
DROP TABLE
gaussdb=# DROP TABLE test;
DROP TABLE
```

须知

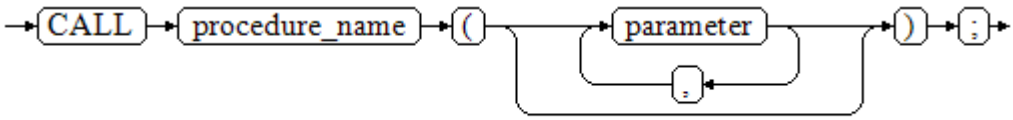
- BULK COLLECT INTO 只支持批量赋值给数组或集合。集合类型合理使用LIMIT字段避免操作过量数据导致性能下降。
- INTO/BULK COLLECT INTO 只支持4层以下Record类型直接嵌套。
- 返回空结果集需要数据库初始化使用PG兼容参数，配置GUC参数**set behavior_compat_options = 'select_into_return_null'**为开启。配置GUC参数**set behavior_compat_options =** ''则关闭。
- 对于数组变量，小括号"()"将优先识别为下标，因此对于带括号的表达式，不支持写在数组变量后面。如对于select (1+3) into va(5)，不支持写为select into va(5) (1+3)或select into va[5] (1+3)。
- INSERT INTO、UPDATE INTO、DELETE INTO、EXECUTION INTO不支持返回空结果集。
- 给多个变量赋值时，由于后面的变量存在语法错误，所以均不赋值。

10.6.3 调用语句

语法

调用一个语句的语法请参见图10-6。

图 10-6 call_clause::=



对以上语法格式的解释如下：

- procedure_name：存储过程名。
- parameter：存储过程的参数，可以没有或者有多个参数。

示例

```
gaussdb=# CREATE TABLE staffs (   section_id INTEGER,   salary INTEGER );
CREATE TABLE
gaussdb=# INSERT INTO staffs VALUES (30, 10);
INSERT 0 1
gaussdb=# INSERT INTO staffs VALUES (30, 20);
INSERT 0 1

--创建存储过程proc_staffs
gaussdb=# CREATE OR REPLACE PROCEDURE proc_staffs
(
```



```
section     NUMBER(6),
salary_sum out NUMBER(8,2),
staffs_count out INTEGER
)
IS
BEGIN
SELECT sum(salary), count(*) INTO salary_sum, staffs_count FROM staffs where section_id = section;
END;
/
CREATE PROCEDURE

--调用存储过程proc_return.
gaussdb=# CALL proc_staffs(2,8,6);
 salary_sum | staffs_count
------------+--------------
            |            0
(1 row)


--清除存储过程
gaussdb=# DROP PROCEDURE proc_staffs;
DROP PROCEDURE
DECLARE
DROP TABLE
```

10.7 动态语句

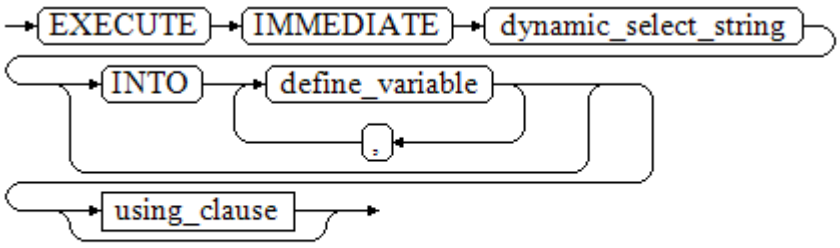
10.7.1 执行动态查询语句

介绍执行动态查询语句。GaussDB提供两种方式：使用EXECUTE IMMEDIATE、OPEN FOR实现动态查询。前者通过动态执行SELECT语句，后者结合了游标的使用。当需要将查询的结果保存在一个数据集用于提取时，可使用OPEN FOR实现动态查询。

EXECUTE IMMEDIATE

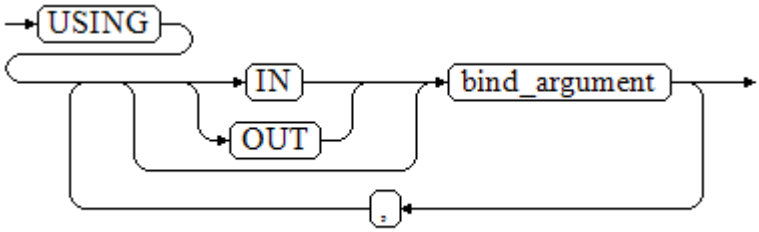
语法图请参见图10-7。

图 10-7 EXECUTE IMMEDIATE dynamic_select_clause::=



using_clause子句的语法图参见图10-8。

图 10-8 using_clause::=



对以上语法格式的解释如下：

- define_variable：用于指定存放单行查询结果的变量。
- USING IN bind_argument：用于指定存放传递给动态SQL值的变量，即在dynamic_select_string中存在占位符时使用。
- USING OUT bind_argument：用于指定存放动态SQL返回值的变量。

须知

- 查询语句中，into和out不能同时存在；
- 占位符命名以“:”开始，后面可跟数字、字符或字符串（不能使用带引号的数字、字符或字符串），与USING子句的bind_argument一一对应；
- bind_argument只能是值、变量或表达式，不能是表名、列名、数据类型等数据库对象，即不支持使用bind_argument为动态SQL语句传递模式对象。如果存储过程需要通过声明参数传递数据库对象来构造动态SQL语句（常见于执行DDL语句时），建议采用连接运算符“||”拼接dynamic_select_clause；
- 动态PL/SQL块允许出现重复的占位符，即相同占位符只能与USING子句的一个bind_argument按位置对应。当设置guc参数behavior_compat_options值为dynamic_sql_compat时，会按照占位符的顺序依次匹配USING子句bind_argument，重复的占位符不会再识别为同一个占位符。

示例：

```
gaussdb=# DROP SCHEMA IF EXISTS hr CASCADE;
gaussdb=# CREATE SCHEMA hr;
gaussdb=# SET CURRENT_SCHEMA = hr;
gaussdb=# CREATE TABLE staffs
(
  staff_id NUMBER,
  first_name VARCHAR2,
  salary NUMBER
);
gaussdb=# INSERT INTO staffs VALUES (200, 'mike', 5800);
gaussdb=# INSERT INTO staffs VALUES (201, 'lily', 3000);
gaussdb=# INSERT INTO staffs VALUES (202, 'john', 4400);

--从动态语句检索值（INTO 子句）：
gaussdb=# DECLARE
   staff_count  VARCHAR2(20);
BEGIN
   EXECUTE IMMEDIATE 'select count(*) from hr.staffs'
      INTO staff_count;
   dbe_output.print_line(staff_count);
END;
/
```

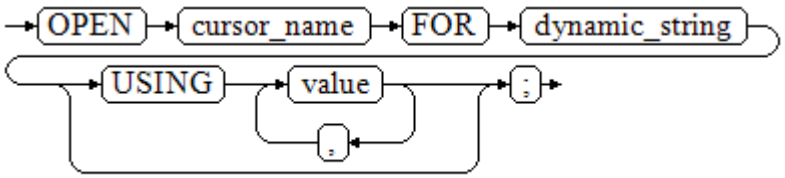
```
--传递并检索值（INTO子句用在USING子句前）：
gaussdb=# CREATE OR REPLACE PROCEDURE dynamic_proc
AS
   staff_id     NUMBER(6) := 200;
   first_name   VARCHAR2(20);
   salary       NUMBER(8,2);
BEGIN
   EXECUTE IMMEDIATE 'select first_name, salary from hr.staffs where staff_id = :1'
      INTO first_name, salary
      USING IN staff_id;
   dbe_output.print_line(first_name || ' ' || salary);
END;
/

--调用存储过程
gaussdb=# CALL dynamic_proc();

--删除存储过程
gaussdb=# DROP PROCEDURE dynamic_proc;
```

OPEN FOR

动态查询语句还可以使用OPEN FOR打开动态游标来执行。

语法参见**图10-9**。

图 10-9 open_for::=



参数说明：

- cursor_name：要打开的游标名。
- dynamic_string：动态查询语句。
- USING value：在dynamic_string中存在占位符时使用。

游标的使用请参考**游标**。

示例

```
gaussdb=# CREATE SCHEMA hr;
gaussdb=# SET CURRENT_SCHEMA = hr;
gaussdb=# CREATE TABLE staffs
(
  section_id NUMBER,
  first_name VARCHAR2,
  phone_number VARCHAR2,
  salary NUMBER
  );
gaussdb=# INSERT INTO staffs VALUES (30, 'mike', '13567829252', 5800);
gaussdb=# INSERT INTO staffs VALUES (40, 'john', '17896354637', 4000);

gaussdb=# DECLARE
    name          VARCHAR2(20);
    phone_number  VARCHAR2(20);
    salary        NUMBER(8,2);
    sqlstr        VARCHAR2(1024);
```



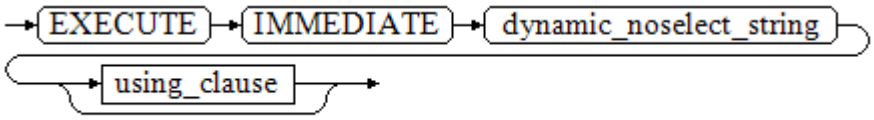
```
    TYPE app_ref_cur_type IS REF CURSOR;  --定义游标类型
    my_cur app_ref_cur_type;  --定义游标变量

BEGIN
    sqlstr := 'select first_name,phone_number,salary from hr.staffs
        where section_id = :1';
    OPEN my_cur FOR sqlstr USING '30';  --打开游标, using是可选的
    FETCH my_cur INTO name, phone_number, salary; --获取数据
    WHILE my_cur%FOUND LOOP
        dbe_output.print_line(name||'#'||phone_number||'#'||salary);
        FETCH my_cur INTO name, phone_number, salary;
    END LOOP;
    CLOSE my_cur;   --关闭游标
END;
/
```

10.7.2 执行动态非查询语句

语法

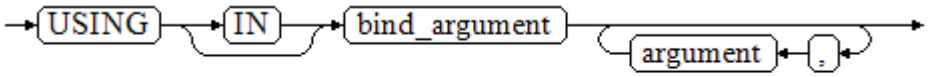
语法请参见图10-10。

图 10-10 noselect::=



using_clause子句的语法参见图10-11。

图 10-11 using_clause::=



对以上语法格式的解释如下：

USING IN bind_argument用于指定存放传递给动态SQL值的变量，在dynamic_noselect_string中存在占位符时使用，即动态SQL语句执行时，bind_argument将替换相对应的占位符。要注意的是，bind_argument只能是值、变量或表达式，不能是表名、列名、数据类型等数据库对象。如果存储过程需要通过声明参数传递数据库对象来构造动态SQL语句（常见于执行DDL语句时），建议采用连接运算符“||”拼接dynamic_select_clause。另外，动态语句允许出现重复的占位符，相同占位符只能与唯一一个bind_argument按位置一一对应。当设置guc参数**behavior_compat_options**值为dynamic_sql_compat时，会按照占位符的顺序依次匹配USING子句bind_argument，重复的占位符不会再识别为同一个占位符（占位符名不能使用带引号的数字、字符或字符串）。

示例

```
--创建表
gaussdb=# CREATE TABLE sections_t1
(
```



```
  section       NUMBER(4) ,
  section_name  VARCHAR2(30),
  manager_id    NUMBER(6),
  place_id      NUMBER(4)
);
CREATE TABLE

--声明变量
gaussdb=# DECLARE
  section       NUMBER(4) := 280;
  section_name  VARCHAR2(30) := 'Info support';
  manager_id    NUMBER(6) := 103;
  place_id      NUMBER(4) := 1400;
  new_colname   VARCHAR2(10) := 'sec_name';
BEGIN
--执行查询
   EXECUTE IMMEDIATE 'insert into sections_t1 values(:1, :2, :3, :4)'
      USING section, section_name, manager_id,place_id;
--执行查询（重复占位符）
   EXECUTE IMMEDIATE 'insert into sections_t1 values(:1, :2, :3, :1)'
      USING section, section_name, manager_id;
--执行ALTER语句（建议采用“||”拼接数据库对象构造DDL语句）
   EXECUTE IMMEDIATE 'alter table sections_t1 rename section_name to ' || new_colname;
END;
/
ANONYMOUS BLOCK EXECUTE
--查询数据
gaussdb=# SELECT * FROM sections_t1;
 section |   sec_name   | manager_id | place_id
---------+--------------+------------+----------
     280 | Info support |        103 |     1400
     280 | Info support |        103 |      280
(2 rows)

--删除表
gaussdb=# DROP TABLE sections_t1;
DROP TABLE
```

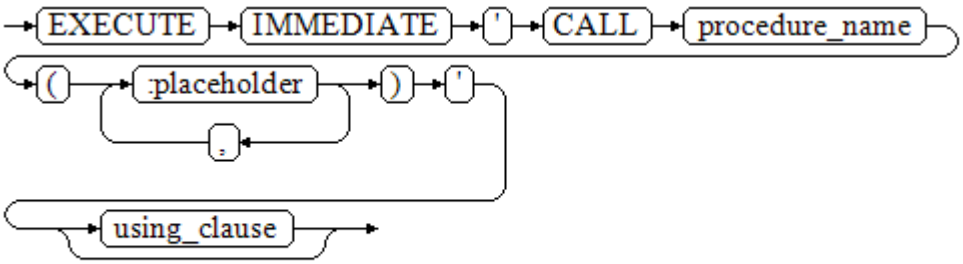
10.7.3 动态调用存储过程

动态调用存储过程必须使用匿名的语句块将存储过程或语句块包在里面，使用EXECUTE IMMEDIATE…USING语句后面带IN、OUT来输入、输出参数。

语法

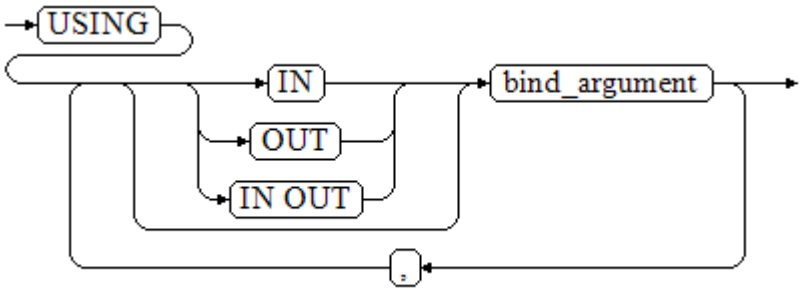
语法请参见图10-12。

图 10-12 call_procedure::=



using_clause子句的语法参见图10-13。

图 10-13 using_clause::=



对以上语法格式的解释如下：

- CALL procedure_name: 调用存储过程。
- [:placeholder1，:placeholder2，…]: 存储过程参数占位符列表，占位符名不能使用带引号的数字、字符或字符串。占位符个数与参数个数相同。
- USING [IN|OUT|IN OUT] bind_argument: 用于指定存放传递给存储过程参数值的变量。bind_argument前的修饰符与对应参数的修饰符一致。
- 不支持调用带有占位符的重载函数或者存储过程。

示例：

```
--创建存储过程proc_add。
gaussdb=# CREATE OR REPLACE PROCEDURE proc_add
(
    param1    in   INTEGER,
    param2    out  INTEGER,
    param3    in   INTEGER
)
AS
BEGIN
   param2:= param1 + param3;
END;
/

gaussdb=# DECLARE
    input1 INTEGER:=1;
    input2 INTEGER:=2;
    statement  VARCHAR2(200);
    param2     INTEGER;
BEGIN
   --声明调用语句
    statement := 'call proc_add(:col_1, :col_2, :col_3)';
    --执行语句
    EXECUTE IMMEDIATE statement
        USING IN input1, OUT param2, IN input2;
    dbe_output.print_line('result is: '||to_char(param2));
END;
/

--删除存储过程
gaussdb=# DROP PROCEDURE proc_add;
```

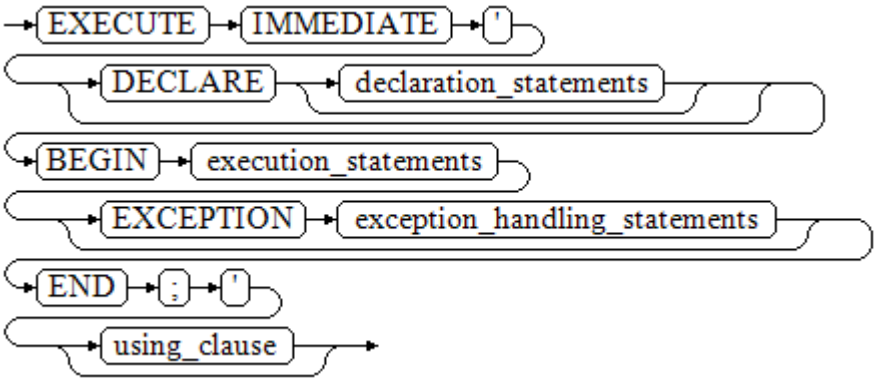
10.7.4 动态调用匿名块

动态调用匿名块是指在动态语句中执行匿名块，使用EXECUTE IMMEDIATE…USING语句后面带IN、OUT来输入、输出参数。

语法

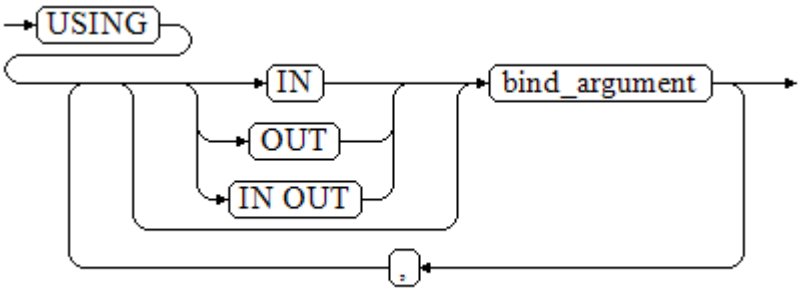
语法请参见图10-14。

图 10-14 call_anonymous_block::=



using_clause子句的语法参见图10-15。

图 10-15 using_clause::=



对以上语法格式的解释如下：

- 匿名块程序实施部分，以BEGIN语句开始，以END语句停顿，以一个分号结束。
- USING [IN|OUT|IN OUT] bind_argument，用于指定存放传递给存储过程参数值的变量。bind_argument前的修饰符与对应参数的修饰符一致。
- 匿名块中间的输入输出参数使用占位符来指明，要求占位符个数与参数个数相同（占位符名不能使用带引号的数字、字符或字符串），并且占位符所对应参数的顺序和USING中参数的顺序一致。
- 目前GaussDB在动态语句调用匿名块时，EXCEPTION语句中暂不支持使用占位符进行输入输出参数的传递。
- 不支持调用带有占位符的重载函数。
- 不支持绑定参数时使用PERFORM关键字调用存储过程。
- 不支持同一条语句同时使用匿名块内声明的变量和绑定参数。
- 不支持匿名块中SELECT INTO语句调用含有出参的FUNCTION/PROCEDURE时，绑定出参。

- 仅支持匿名块中调用SQL语句以及带有out/inout参数的存储过程绑定参数，其余绑定参数场景皆不支持。例如：匿名块中使用表达式以及cursor等、匿名块中嵌套调用动态语句。
- 打开dynamic_sql_check参数时，占位符个数与参数个数一致时使用同名占位符作为匿名块参数会报错，需修改为不同名参数。详情见**示例**。

示例

```
gaussdb=# DROP SCHEMA IF EXISTS hr CASCADE;
gaussdb=# CREATE SCHEMA hr;
gaussdb=# SET CURRENT_SCHEMA = hr;
gaussdb=# CREATE TABLE staffs
(
  staff_id NUMBER,
  first_name VARCHAR2,
  salary NUMBER
);
gaussdb=# INSERT INTO staffs VALUES (200, 'mike', 5800);
gaussdb=# INSERT INTO staffs VALUES (201, 'lily', 3000);
gaussdb=# INSERT INTO staffs VALUES (202, 'john', 4400);

--创建存储过程dynamic_proc。
gaussdb=# CREATE OR REPLACE PROCEDURE dynamic_proc
AS
   staff_id     NUMBER(6) := 200;
   first_name   VARCHAR2(20);
   salary       NUMBER(8,2);
BEGIN
    --执行匿名块 。
    EXECUTE IMMEDIATE 'begin select first_name, salary into :first_name, :salary from hr.staffs where
staff_id= :dno; end;'
       USING OUT first_name, OUT salary, IN staff_id;
   dbe_output.print_line(first_name|| ' ' || salary);
END;
/

--调用存储过程。
gaussdb=# CALL dynamic_proc();
mike 5800.00
 dynamic_proc
--------------

(1 row)
--删除存储过程。
gaussdb=# DROP PROCEDURE dynamic_proc;

--开启dynamic_sql_check时报错示例
gaussdb=# SET behavior_compat_options = 'dynamic_sql_check';
SET
gaussdb=# CREATE OR REPLACE PROCEDURE test_proc_exception001(a out integer, b inout integer, c
integer)
as
BEGIN
 a := 1;
 begin b := 1/0; end;
EXCEPTION
 WHEN others THEN
  b := 2;
END;
/
CREATE PROCEDURE
gaussdb=#
DECLARE
 a integer := 1;
 c integer;
BEGIN
  execute immediate 'begin test_proc_exception001(:1,:2,:1); end;' using in out a, out c, a;
```



```
END;
/
ERROR:  argnum not match in Dynamic SQL, using args num : 3 , actual sql args num : 2
CONTEXT:  PL/pgSQL function inline_code_block line 4 at EXECUTE statement
--修改同名占位符
gaussdb=#
DECLARE
 a integer := 1;
 c integer;
BEGIN
  execute immediate 'begin test_proc_exception001(:1,:2,:3); end;' using in out a, out c, a;
END;
/
ANONYMOUS BLOCK EXECUTE
gaussdb=# DROP PROCEDURE test_proc_exception001;
DROP PROCEDURE
gaussdb=# reset behavior_compat_options;
```

10.8 控制语句

10.8.1 返回语句

GaussDB提供两种方式返回数据：RETURN或RETURN NEXT及RETURN QUERY。其中，RETURN NEXT和RETURN QUERY只适用于函数，不适用存储过程。

10.8.1.1 RETURN

语法

返回语句的语法请参见图10-16。

图 10-16 return_clause::=

对以上语法的解释如下：

用于将控制从存储过程或函数返回给调用者。

示例

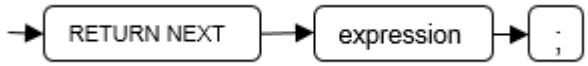
请参见调用语句的示例。

10.8.1.2 RETURN NEXT 及 RETURN QUERY

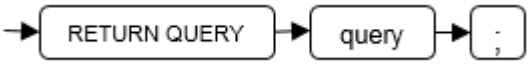
语法

创建函数时需要指定返回值SETOF datatype。

return_next_clause::=



return_query_clause::=



对以上语法的解释如下：

当需要函数返回一个集合时，使用RETURN NEXT或者RETURN QUERY向结果集追加结果，然后继续执行函数的下一条语句。随着后续的RETURN NEXT或RETURN QUERY命令的执行，结果集中会有多个结果。函数执行完成后会一起返回所有结果。

RETURN NEXT可用于标量和复合数据类型。

RETURN QUERY有一种变体RETURN QUERY EXECUTE，后面还可以增加动态查询，通过USING向查询插入参数。

示例

```
gaussdb=# DROP TABLE t1;
gaussdb=# CREATE TABLE t1(a int);
gaussdb=# INSERT INTO t1 VALUES(1),(10);

--RETURN NEXT
gaussdb=# CREATE OR REPLACE FUNCTION fun_for_return_next() RETURNS SETOF t1 AS $$
DECLARE
   r t1%ROWTYPE;
BEGIN
   FOR r IN select * from t1
   LOOP
      RETURN NEXT r;
   END LOOP;
   RETURN;
END;
$$ LANGUAGE plpgsql;
gaussdb=# call fun_for_return_next();
 a
---
 1
 10
(2 rows)

-- RETURN QUERY
gaussdb=# CREATE OR REPLACE FUNCTION fun_for_return_query() RETURNS SETOF t1 AS $$
DECLARE
   r t1%ROWTYPE;
BEGIN
   RETURN QUERY select * from t1;
END;
$$
language plpgsql;
gaussdb=# call fun_for_return_query();
 a
---
 1
 10
(2 rows)
```

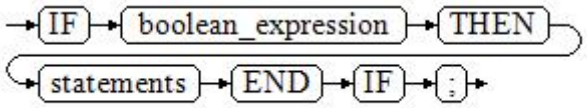
10.8.2 条件语句

条件语句的主要作用判断参数或者语句是否满足已给定的条件，根据判定结果执行相应的操作。

GaussDB有五种形式的IF：

- IF_THEN

  图 10-17 IF_THEN::=

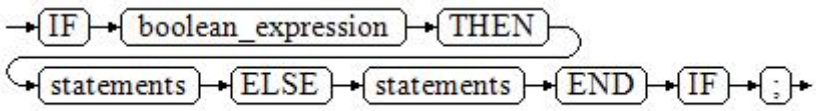
  IF_THEN语句是IF的最简单形式。如果条件为真，statements将被执行。否则，将忽略它们的结果使该IF_THEN语句执行结束。

  示例

```
gaussdb=#
DECLARE
    v_user_id integer default 1;
BEGIN
IF v_user_id <> 0 THEN
    raise info 'v_user_id is NOT 0';

END IF;
END;
/
INFO:  v_user_id is NOT 0
ANONYMOUS BLOCK EXECUTE
```

- IF_THEN_ELSE

  图 10-18 IF_THEN_ELSE::=

  IF_THEN_ELSE语句增加了ELSE的分支，可以声明在条件为假的时候执行的语句。

  示例

```
gaussdb=#
DECLARE
    v_user_id integer default 1;
BEGIN
    IF v_user_id <> 0 THEN
        raise info 'v_user_id is NOT 0';
    ELSE
        raise info 'v_user_id is 0';
    END IF;
END;
/
INFO:  v_user_id is NOT 0
ANONYMOUS BLOCK EXECUTE
```

- IF_THEN_ELSE IF

  IF语句可以嵌套，嵌套方式如下：

```
gaussdb=#
DECLARE
    v_user_id integer default 1;
BEGIN
    IF v_user_id = 0 THEN
        raise info 'v_user_id is  0';
    ELSE
```

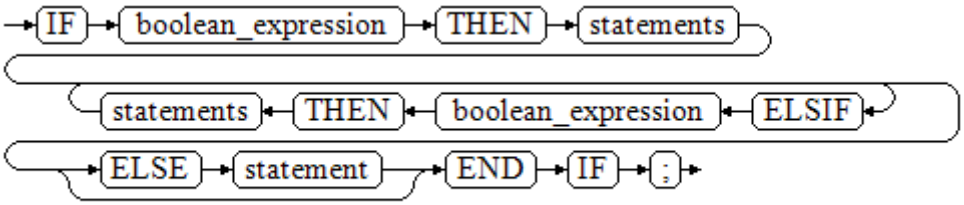


```
        IF v_user_id > 0 THEN
            raise info 'v_user_id > 0';
        END IF;
    END IF;
END;
/
INFO:  v_user_id > 0
ANONYMOUS BLOCK EXECUTE
```

这种形式实际上就是在一个IF语句的ELSE部分嵌套了另一个IF语句。因此需要一个END IF语句给每个嵌套的IF，另外还需要一个END IF语句结束父IF-ELSE。如果有多个选项，可使用下面的形式。

- IF_THEN_ELSIF_ELSE

  图 10-19 IF_THEN_ELSIF_ELSE::=

  示例

```
gaussdb=#
DECLARE
    v_user_id integer default NULL;
BEGIN
    IF v_user_id = 0 THEN
        raise info 'v_user_id is  0';
    ELSIF v_user_id > 0 THEN
        raise info 'v_user_id > 0';
    ELSIF v_user_id < 0 THEN
        raise info 'v_user_id < 0';
    ELSE
        raise info 'v_user_id is NULL';
    END IF;
END;
/
INFO:  v_user_id is NULL
ANONYMOUS BLOCK EXECUTE
```

- IF_THEN_ELSEIF_ELSE

  ELSEIF是ELSIF的别名。

  综合示例

```
gaussdb=# CREATE OR REPLACE PROCEDURE proc_control_structure(i in integer)
AS
    BEGIN
        IF i > 0 THEN
            raise info 'i:% is greater than 0. ',i;
        ELSIF i < 0 THEN
            raise info 'i:% is smaller than 0. ',i;
        ELSE
            raise info 'i:% is equal to 0. ',i;
        END IF;
        RETURN;
    END;
/
CREATE PROCEDURE

gaussdb=# CALL proc_control_structure(3);
INFO:  i:3 is greater than 0.
```



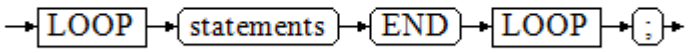
```
 proc_control_structure
------------------------

(1 row)

--删除存储过程
gaussdb=# DROP PROCEDURE proc_control_structure;
DROP PROCEDURE
```

10.8.3 循环语句

简单 LOOP 语句

语法图

图 10-20 loop::=



示例

```
gaussdb=# CREATE OR REPLACE PROCEDURE proc_loop(i in integer, count out integer)
AS
    BEGIN
        count:=0;
        LOOP
        IF count > i THEN
            raise info 'count is %. ', count;
            EXIT;
        ELSE
            count:=count+1;
        END IF;
        END LOOP;
    END;
/
CREATE PROCEDURE
gaussdb=# CALL proc_loop(10,5);
INFO:  count is 11.
 count
-------
    11
(1 row)
```

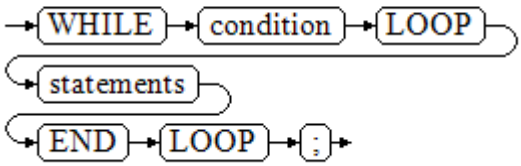
须知

该循环必须要结合EXIT使用，否则将陷入死循环。

WHILE_LOOP 语句

语法图

图 10-21 while_loop::=



只要条件表达式为真，WHILE语句就会不停地在一系列语句上进行循环，在每次进入循环体的时候进行条件判断。

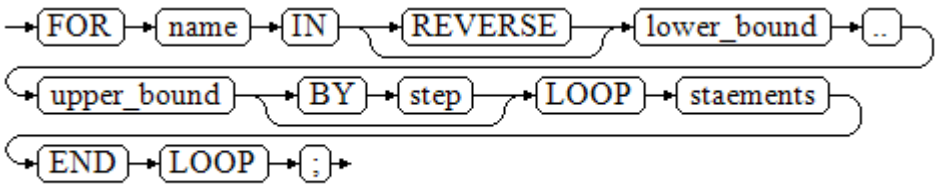
示例

```
gaussdb=# CREATE TABLE integertable(c1 integer);
CREATE TABLE
gaussdb=# CREATE OR REPLACE PROCEDURE proc_while_loop(maxval in integer)
AS
    DECLARE
    i int :=1;
    BEGIN
        WHILE i < maxval LOOP
            INSERT INTO integertable VALUES(i);
            i:=i+1;
        END LOOP;
    END;
/
CREATE PROCEDURE

--调用函数
gaussdb=# CALL proc_while_loop(10);
 proc_while_loop
-----------------

(1 row)

--删除存储过程和表
gaussdb=# DROP PROCEDURE proc_while_loop;
DROP PROCEDURE
gaussdb=# DROP TABLE integertable;
DROP TABLE
```

FOR_LOOP（integer 变量）语句

语法图

图 10-22 for_loop::=



说明

- 变量name会自动定义为integer类型并且只在此循环里存在。变量name介于lower_bound和upper_bound之间。
- 当使用REVERSE关键字时，lower_bound必须大于等于upper_bound，否则循环体不会被执行。

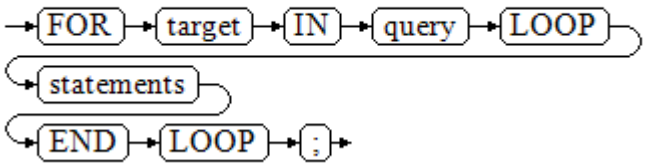
示例

```
--从0到5进行循环
gaussdb=# CREATE OR REPLACE PROCEDURE proc_for_loop()
AS
    BEGIN
    FOR I IN 0..5 LOOP
        DBE_OUTPUT.PRINT_LINE('It is '||to_char(I) || ' time;') ;
    END LOOP;
END;
/
CREATE PROCEDURE

--调用存储过程
gaussdb=# CALL proc_for_loop();
It is 0 time;
It is 1 time;
It is 2 time;
It is 3 time;
It is 4 time;
It is 5 time;
 proc_for_loop
---------------

(1 row)

--删除存储过程
gaussdb=# DROP PROCEDURE proc_for_loop;
DROP PROCEDURE
```

FOR_LOOP 查询语句

语法图

图 10-23 for_loop_query::=



说明

- 变量target会自动定义，类型和query的查询结果的类型一致，并且只在此循环中有效。target的取值就是query的查询结果。
- query可以使用EXECUTE增加动态查询，通过USING向查询插入参数。详见**示例**中的动态查询相关内容。

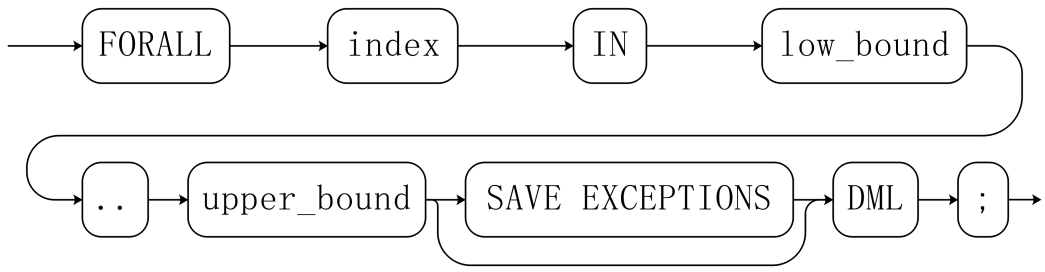
示例


```
--循环输出查询结果。
gaussdb=# CREATE OR REPLACE PROCEDURE proc_for_loop_query()
AS
    record VARCHAR2(50);
BEGIN
    FOR record IN SELECT spcname FROM pg_tablespace LOOP
    dbe_output.print_line(record);
    END LOOP;
END;
/
CREATE PROCEDURE

--调用存储过程
gaussdb=# CALL proc_for_loop_query();
pg_default
pg_global
 proc_for_loop_query
---------------------

(1 row)

--删除存储过程
gaussdb=# DROP PROCEDURE proc_for_loop_query;
DROP PROCEDURE
--动态查询
gaussdb=# CREATE TABLE t1(id int);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(1);
INSERT 0 1
gaussdb=#
DECLARE
a int := 1;
item t1%rowtype;
BEGIN
    FOR item IN EXECUTE 'SELECT * FROM t1 WHERE id = :1' USING a LOOP
        RAISE INFO '%', item;
    END LOOP;
END;
/
INFO:  (1)
ANONYMOUS BLOCK EXECUTE
gaussdb=# DROP TABLE t1;
DROP TABLE
```

FORALL 批量查询语句

语法图

图 10-24 forall::=



说明

- 变量index会自动定义为integer类型并且只在此循环里存在。index的取值介于low_bound和upper_bound之间。
- 如果声明了SAVE EXCEPTIONS，则会将循环体DML执行过程中每次遇到的异常保存在SQL&BULK_EXCEPTIONS中，并在执行结束后统一抛出一个异常，循环过程中没有异常的执行的结果在当前子事务内不会回滚。

示例

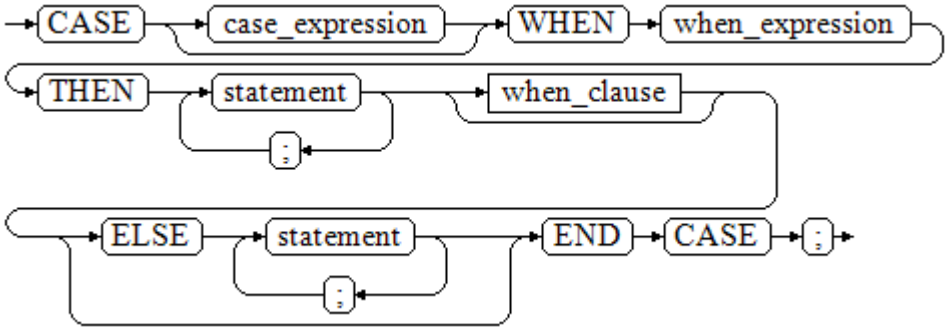
```
gaussdb=# CREATE TABLE hdfs_t1 (
  title NUMBER(6),
  did VARCHAR2(20),
  data_period VARCHAR2(25),
  kind VARCHAR2(25),
  interval VARCHAR2(20),
  time DATE,
  isModified VARCHAR2(10)
);
CREATE TABLE

gaussdb=# INSERT INTO hdfs_t1 VALUES( 8, 'Donald', 'OConnell', 'DOCONNEL', '650.507.9833',
to_date('21-06-1999', 'dd-mm-yyyy'), 'SH_CLERK' );
INSERT 0 1

gaussdb=# CREATE OR REPLACE PROCEDURE proc_forall()
AS
BEGIN
    FORALL i IN 100..120
        update hdfs_t1 set title = title + 100*i;
END;
/
CREATE PROCEDURE

--调用存储过程
gaussdb=# CALL proc_forall();
 proc_forall
-------------

(1 row)

--查询存储过程调用结果
gaussdb=# SELECT * FROM hdfs_t1;
 title  |  did   | data_period |   kind   |   interval   |        time         | ismodified
--------+--------+-------------+----------+--------------+---------------------+------------
 231008 | Donald | OConnell    | DOCONNEL | 650.507.9833 | 1999-06-21 00:00:00 | SH_CLERK
(1 row)

--删除存储过程和表
gaussdb=# DROP PROCEDURE proc_forall;
DROP PROCEDURE
gaussdb=# DROP TABLE hdfs_t1;
DROP TABLE
```

10.8.4 分支语句

语法

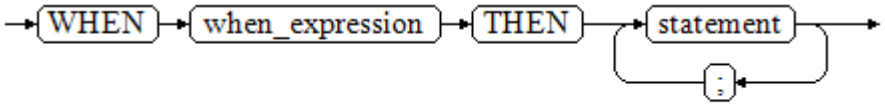
分支语句的语法请参见**图10-25**。

图 10-25 case_when::=



when_clause子句的语法图参见图10-26。

图 10-26 when_clause::=



参数说明：

- case_expression：变量或表达式。
- when_expression：常量或者条件表达式。
- statement：执行语句。

示例

```
gaussdb=# CREATE OR REPLACE PROCEDURE proc_case_branch(pi_result in integer, pi_return out integer)
AS
   BEGIN
      CASE pi_result
         WHEN 1 THEN
            pi_return := 111;
         WHEN 2 THEN
            pi_return := 222;
         WHEN 3 THEN
            pi_return := 333;
         WHEN 6 THEN
            pi_return := 444;
         WHEN 7 THEN
            pi_return := 555;
         WHEN 8 THEN
            pi_return := 666;
         WHEN 9 THEN
            pi_return := 777;
         WHEN 10 THEN
            pi_return := 888;
         ELSE
            pi_return := 999;
      END CASE;
      raise info 'pi_return : %',pi_return ;
END;
/
CREATE PROCEDURE

gaussdb=# CALL proc_case_branch(3,0);
```

```
INFO:  pi_return : 333
 pi_return
-----------
       333
(1 row)

--删除存储过程
gaussdb=# DROP PROCEDURE proc_case_branch;
DROP PROCEDURE
```

10.8.5 空语句

在PL/SQL程序中，可以用NULL语句来说明“不用做任何事情”，相当于一个占位符，可以使某些语句变得有意义，提高程序的可读性。

语法

空语句的用法如下：

```
DECLARE
   ...
BEGIN
   ...
   IF v_num IS NULL THEN
      NULL; -- 不需要处理任何数据。
   END IF;
END;
/
```

示例

```
gaussdb=# DECLARE
   v_num integer default NULL;
BEGIN
   IF v_num IS NOT NULL THEN
      raise info 'v_num is NULL';
   ELSE
      NULL; -- 不需要处理任何数据。
   END IF;
END;
/
ANONYMOUS BLOCK EXECUTE
```

10.8.6 错误捕获语句

缺省时，当PL/SQL函数执行过程中发生错误时退出函数执行，并且周围的事务也会回滚。可以用一个带有EXCEPTION子句的BEGIN块捕获错误并且从中恢复。其语法是正常的BEGIN块语法的一个扩展：

```
[<<label>>]
[DECLARE
   declarations]
BEGIN
   statements
EXCEPTION
   WHEN condition [OR condition ...] THEN
      handler_statements
   [WHEN condition [OR condition ...] THEN
      handler_statements
   ...]
END;
```

如果没有发生错误，这种形式的包围块只是简单地执行所有语句，然后转到END之后的下一个语句。但是如果在执行的语句内部发生了一个错误，则这个语句将会回滚，然后转到EXCEPTION列表。寻找匹配错误的第一个条件。若找到匹配，则执行对应的

handler_statements，然后转到END之后的下一个语句。如果没有找到匹配，则会向事务的外层报告错误，和没有EXCEPTION子句一样。错误码可以捕获同一类的其他错误码。

也就是说该错误可以被一个包围块用EXCEPTION捕获，如果没有包围块，则进行退出函数处理。

condition的名称可以是SQL标准错误码编号说明的任意值。特殊的条件名OTHERS匹配除了QUERY_CANCELED之外的所有错误类型。

如果在选中的handler_statements里发生了新错误，则不能被这个EXCEPTION子句捕获，而是向事务的外层报告错误。一个外层的EXCEPTION子句可以捕获它。

如果一个错误被EXCEPTION捕获，PL/SQL函数的局部变量保持错误发生时的原值，但是所有该块中想写入数据库中的状态都回滚。

示例：

```
gaussdb=# CREATE TABLE mytab(id INT,firstname VARCHAR(20),lastname VARCHAR(20));
CREATE TABLE

gaussdb=# INSERT INTO mytab(firstname, lastname) VALUES('Tom', 'Jones');
INSERT 0 1

gaussdb=# CREATE FUNCTION fun_exp() RETURNS INT
AS $$
DECLARE
    x INT :=0;
    y INT;
BEGIN
    UPDATE mytab SET firstname = 'Joe' WHERE lastname = 'Jones';
    x := x + 1;
    y := x / 0;
EXCEPTION
    WHEN division_by_zero THEN
        RAISE NOTICE 'caught division_by_zero';
        RETURN x;
END;$$
LANGUAGE plpgsql;
CREATE FUNCTION

gaussdb=# call fun_exp();
NOTICE:  caught division_by_zero
 fun_exp
---------
       1
(1 row)

gaussdb=# select * from mytab;
 id | firstname | lastname
----+-----------+----------
    | Tom       | Jones
(1 row)

gaussdb=# DROP FUNCTION fun_exp();
DROP FUNCTION

gaussdb=# DROP TABLE mytab;
DROP TABLE
```

当控制到达给y赋值的地方时，会有一个division_by_zero错误失败。这个错误将被EXCEPTION子句捕获。而在RETURN语句里返回的数值将是x的增量值。

说明

进入和退出一个包含EXCEPTION子句的块要比不包含的块开销大的多。因此，不必要的时候不要使用EXCEPTION。

在下列场景中，无法捕获处理异常，整个存储过程回滚：节点故障、网络故障引起的存储过程参与节点线程退出以及COPY FROM操作中源数据与目标表的表结构不一致造成的异常。

示例： UPDATE/INSERT异常

这个例子根据使用异常处理器执行恰当的UPDATE或INSERT 。

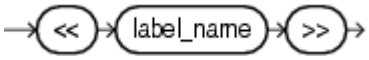
```
gaussdb=# CREATE TABLE db (a INT, b TEXT);
CREATE TABLE

gaussdb=# CREATE FUNCTION merge_db(key INT, data TEXT) RETURNS VOID AS
$$
BEGIN
    LOOP

--第一次尝试更新key
        UPDATE db SET b = data WHERE a = key;
        IF found THEN
            RETURN;
        END IF;
--不存在，所以尝试插入key，如果其他人同时插入相同的key，可能得到唯一key失败。
        BEGIN
            INSERT INTO db(a,b) VALUES (key, data);
            RETURN;
        EXCEPTION WHEN unique_violation THEN
        --什么也不做，并且循环尝试再次更新。
        END;
    END LOOP;
END;
$$
LANGUAGE plpgsql;
CREATE FUNCTION

gaussdb=# SELECT merge_db(1, 'david');
 merge_db
----------

(1 row)

gaussdb=# SELECT merge_db(1, 'dennis');
 merge_db
----------

(1 row)

--删除FUNCTION和TABLE
gaussdb=# DROP FUNCTION merge_db;
DROP FUNCTION

gaussdb=# DROP TABLE db;
DROP TABLE
```

10.8.7 GOTO 语句

GOTO语句可以实现从GOTO位置到目标语句的无条件跳转。GOTO语句会改变原本的执行逻辑，因此应该慎重使用，或者也可以使用EXCEPTION处理特殊场景。当执行GOTO语句时，目标Label必须是唯一的。

语法

label declaration ::=



goto statement ::=

示例

```
gaussdb=# CREATE OR REPLACE PROCEDURE GOTO_test()
AS
DECLARE
    v1  int;
BEGIN
    v1  := 0;
        LOOP
        EXIT WHEN v1 > 100;
                v1 := v1 + 2;
                if v1 > 25 THEN
                        GOTO pos1;
                END IF;
        END LOOP;
<<pos1>>
v1 := v1 + 10;
raise info 'v1 is %. ', v1;
END;
/
CREATE PROCEDURE

gaussdb=# call GOTO_test();
INFO:  v1 is 36.
 goto_test
-----------

(1 row)
```

限制场景

GOTO使用有以下限制场景

- 不支持有多个相同的GOTO labels目标场景，无论是否在同一个block中。

  ```
  BEGIN
    GOTO pos1;
    <<pos1>>
    SELECT * FROM ...
    <<pos1>>
    UPDATE t1 SET ...
  END;
  ```

- 不支持GOTO跳转到IF语句，CASE语句，LOOP语句中。

  ```
  BEGIN
     GOTO pos1;
     IF valid THEN
       <<pos1>>
       SELECT * FROM ...
     END IF;
  END;
  ```

- 不支持GOTO语句从一个IF子句跳转到另一个IF子句，或从一个CASE语句的WHEN子句跳转到另一个WHEN子句。

  ```
  BEGIN
     IF valid THEN
       GOTO pos1;
       SELECT * FROM ...
     ELSE
       <<pos1>>
  ```



```
    UPDATE t1 SET ...
  END IF;
END;
```

- 不支持从外部块跳转到内部的BEGIN-END块。

```
BEGIN
  GOTO pos1;
  BEGIN
    <<pos1>>
    UPDATE t1 SET ...
  END;
END;
```

- 不支持从异常处理部分跳转到当前的BEGIN-END块。但可以跳转到上层BEGIN-END块。

```
BEGIN
  <<pos1>>
  UPDATE t1 SET ...
  EXCEPTION
    WHEN condition THEN
      GOTO pos1;
END;
```

- 如果从GOTO到一个不包含执行语句的位置，需要添加NULL语句。

```
DECLARE
  done  BOOLEAN;
BEGIN
  FOR i IN 1..50 LOOP
    IF done THEN
      GOTO end_loop;
    END IF;
    <<end_loop>>  -- not allowed unless an executable statement follows
    NULL; -- add NULL statement to avoid error
  END LOOP;  -- raises an error without the previous NULL
END;
/
```

10.9 事务管理

存储过程本身就处于一个事务中，开始调用最外围存储过程时会自动开启一个事务，在调用结束时自动提交或者发生异常时回滚。除了系统自动的事务控制外，也可以使用COMMIT/ROLLBACK来控制存储过程中的事务。在存储过程中调用COMMIT/ROLLBACK命令，将提交/回滚当前事务并自动开启一个新的事务，后续的所有操作都会在此新事务中运行。

保存点SAVEPOINT是事务中的一个特殊记号，它允许将那些在它建立后执行的命令全部回滚，把事务的状态恢复到保存点所在的时刻。存储过程中允许使用保存点来进行事务管理，当前支持保存点的创建、回滚和释放操作。存储过程中使用回滚保存点只是回退当前事务的修改，而不会改变存储过程的执行流程，也不会回退存储过程中的局部变量值等。

语法格式

```
定义保存点
    SAVEPOINT savepoint_name;
回滚保存点
    ROLLBACK TO [SAVEPOINT] savepoint_name;
释放保存点
    RELEASE [SAVEPOINT] savepoint_name;
```

使用场景

支持调用的上下文环境：

- 支持在PL/SQL的存储过程内使用COMMIT/ROLLBACK/SAVEPOINT。
- 支持含有EXCEPTION的存储过程使用COMMIT/ROLLBACK/SAVEPOINT。
- 支持在存储过程的EXCEPTION语句内使用COMMIT/ROLLBACK/SAVEPOINT。
- 支持在事务块里调用含有COMMIT/ROLLBACK/SAVEPOINT的存储过程，即通过/BEGIN/START/END等开启控制的外部事务。
- 支持在子事务中调用含有SAVEPOINT的存储过程，即存储过程中使用外部定义的SAVEPOINT，回退事务状态到存储过程外定义的SAVEPOINT位置。
- 支持存储过程外部对存储过程内定义的SAVEPOINT可见，即存储过程外可以将事务修改回滚到存储过程中定义SAVEPOINT的位置。
- 支持多数PL/SQL的上下文和语句内调用COMMIT/ROLLBACK/SAVEPOINT，包括常用的IF/FOR/CURSOR LOOP/WHILE。
- 支持存储过程返回值与简单表达式计算中调用含有COMMIT/ROLLBACK/SAVEPOINT的存储过程或者函数。

支持提交/回滚的内容：

- 支持DDL在COMMIT/ROLLBACK后的提交/回滚。
- 支持DML的COMMIT/ROLLBACK后的提交。
- 支持存储过程内GUC参数的回滚提交。

使用限制

不支持调用的上下文环境：

- 不支持除PL/SQL的其他存储过程中调用COMMIT/ROLLBACK/SAVEPOINT，例如PLJAVA、PLPYTHON等。
- 不支持函数中调用COMMIT/ROLLBACK/SAVEPOINT，包括函数调用含有COMMIT/ROLLBACK/SAVEPOINT的存储过程。
- 不支持事务块中调用了SAVEPOINT后，调用含有COMMIT/ROLLBACK的存储过程。
- 不支持TRIGGER中调用含有COMMIT/ROLLBACK/SAVEPOINT语句的存储过程。
- 不支持EXECUTE语句中调用COMMIT/ROLLBACK/SAVEPOINT语句。
- 不支持在CURSOR语句中打开一个含有COMMIT/ROLLBACK/SAVEPOINT的存储过程。
- 不支持带有IMMUTABLE以及SHIPPABLE的存储过程调用COMMIT/ROLLBACK/SAVEPOINT，或调用带有COMMIT/ROLLBACK/SAVEPOINT语句的存储过程。
- 不支持SQL中调用含有COMMIT/ROLLBACK/SAVEPOINT语句的存储过程，除了SELECT PROC以及CALL PROC。
- 存储过程头带有GUC参数设置的不允许调用COMMIT/ROLLBACK/SAVEPOINT语句。
- 不支持CURSOR/EXECUTE语句，以及各类表达式内调用COMMIT/ROLLBACK/SAVEPOINT。
- 不支持存储过程中释放存储过程外部定义的保存点。
- 自治事务和存储过程事务是两个独立的事务，不能互相使用对方事务中定义的保存点。

不支持提交回滚的内容：

- 不支持存储过程内声明变量以及传入变量的提交/回滚。
- 不支持存储过程内必须重启生效的GUC参数的提交/回滚。

示例

- 示例1：支持在PL/SQL的存储过程内使用COMMIT/ROLLBACK。

```
gaussdb=# CREATE TABLE EXAMPLE1(COL1 INT);
CREATE TABLE

gaussdb=# CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE()
AS
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1(COL1) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
END;
/
CREATE PROCEDURE
```

- 示例2：

  支持含有EXCEPTION的存储过程使用COMMIT/ROLLBACK。

  支持在存储过程的EXCEPTION语句内使用COMMIT/ROLLBACK。

  支持DDL在COMMIT/ROLLBACK后的提交/回滚。

```
gaussdb=# CREATE OR REPLACE PROCEDURE TEST_COMMIT_INSERT_EXCEPTION_ROLLBACK()
AS
BEGIN
 DROP TABLE IF EXISTS TEST_COMMIT;
 CREATE TABLE TEST_COMMIT(A INT, B INT);
 INSERT INTO TEST_COMMIT SELECT 1, 1;
 COMMIT;
        CREATE TABLE TEST_ROLLBACK(A INT, B INT);
 RAISE EXCEPTION 'RAISE EXCEPTION AFTER COMMIT';
EXCEPTION
    WHEN OTHERS THEN
 INSERT INTO TEST_COMMIT SELECT 2, 2;
 ROLLBACK;
END;
/
CREATE PROCEDURE
```

- 示例3：支持在事务块里调用含有COMMIT/ROLLBACK的存储过程，即通过/BEGIN/START/END等开启控制的外部事务。

```
gaussdb=# BEGIN;
    CALL TEST_COMMIT_INSERT_EXCEPTION_ROLLBACK();
END;

NOTICE:  table "test_commit" does not exist, skipping
CONTEXT:  SQL statement "DROP TABLE IF EXISTS TEST_COMMIT"
PL/pgSQL function test_commit_insert_exception_rollback() line 3 at SQL statement
 test_commit_insert_exception_rollback
---------------------------------------

(1 row)

COMMIT
```

- 示例4：支持多数PL/SQL的上下文和语句内调用COMMIT/ROLLBACK，包括常用的IF/FOR/CURSOR LOOP/WHILE。

```
gaussdb=# CREATE OR REPLACE PROCEDURE TEST_COMMIT2()
IS
```



```
BEGIN
    DROP TABLE IF EXISTS TEST_COMMIT;
    CREATE TABLE TEST_COMMIT(A INT);
    FOR I IN REVERSE 3..0 LOOP
 INSERT INTO TEST_COMMIT SELECT I;
 COMMIT;
    END LOOP;
    FOR I IN REVERSE 2..4 LOOP
 UPDATE TEST_COMMIT SET A=I;
 COMMIT;
    END LOOP;
EXCEPTION
WHEN OTHERS THEN
 INSERT INTO TEST_COMMIT SELECT 4;
    COMMIT;
END;
/
CREATE PROCEDURE
```

- 示例5：支持存储过程返回值与简单表达式计算。

```
gaussdb=# CREATE OR REPLACE PROCEDURE exec_func3(RET_NUM OUT INT)
AS
BEGIN
    RET_NUM := 1+1;
COMMIT;
END;
/
CREATE OR REPLACE PROCEDURE exec_func4(ADD_NUM IN INT)
AS
SUM_NUM INT;
BEGIN
SUM_NUM := ADD_NUM + exec_func3();
COMMIT;
END;
/
CREATE PROCEDURE
```

- 示例6：支持存储过程内GUC参数的回滚提交。

```
gaussdb=# SHOW explain_perf_mode;
 explain_perf_mode
-------------------
 normal
(1 row)
gaussdb=# SHOW enable_force_vector_engine;
 enable_force_vector_engine
----------------------------
 off
(1 row)

gaussdb=# CREATE OR REPLACE PROCEDURE GUC_ROLLBACK()
AS
BEGIN
    SET enable_force_vector_engine = on;
    COMMIT;
    SET explain_perf_mode TO pretty;
    ROLLBACK;
END;
/
CREATE PROCEDURE

gaussdb=# call GUC_ROLLBACK();
 guc_rollback
--------------

(1 row)
gaussdb=# SHOW explain_perf_mode;
 explain_perf_mode
-------------------
 normal
(1 row)
```



```
gaussdb=# SHOW enable_force_vector_engine;
 enable_force_vector_engine
----------------------------
 on
(1 row)
gaussdb=# SET enable_force_vector_engine = off;
SET
```

- 示例7：函数（Function）中不允许调用commit/rollback语句，同时不允许函数调用含有commit/rollback的存储过程。

```
gaussdb=# CREATE OR REPLACE FUNCTION FUNCTION_EXAMPLE1() RETURN INT
AS
EXP INT;
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1(col1) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
    SELECT COUNT(*) FROM EXAMPLE1 INTO EXP;
    RETURN EXP;
END;
/
CREATE FUNCTION
```

- 示例8：函数（Fucntion）中不允许调用带有commit/rollback语句的存储过程。

```
gaussdb=# CREATE OR REPLACE FUNCTION FUNCTION_EXAMPLE2() RETURN INT
AS
EXP INT;
BEGIN
    --transaction_example为存储过程，带有commit/rollback语句
    CALL transaction_example();
    SELECT COUNT(*) FROM EXAMPLE1 INTO EXP;
    RETURN EXP;
END;
/
CREATE FUNCTION
```

- 示例9：不允许Trigger的存储过程包含commit/rollback语句，或调用带有commit/rollback语句的存储过程。

```
gaussdb=# CREATE OR REPLACE FUNCTION FUNCTION_TRI_EXAMPLE2() RETURN TRIGGER
AS
EXP INT;
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1(col1) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
    SELECT COUNT(*) FROM EXAMPLE1 INTO EXP;
END;
/
CREATE FUNCTION

gaussdb=# CREATE TRIGGER TRIGGER_EXAMPLE AFTER DELETE ON EXAMPLE1
FOR EACH ROW EXECUTE PROCEDURE FUNCTION_TRI_EXAMPLE2();
CREATE TRIGGER

gaussdb=# DELETE FROM EXAMPLE1;
DELETE 0
```

- 示例10：不支持带有IMMUTABLE以及SHIPPABLE的存储过程调用commit/rollback，或调用带有commit/rollback语句的存储过程。


```
gaussdb=# CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE1()
IMMUTABLE
AS
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1 (col1) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
END;
/
CREATE PROCEDURE
```

- 示例11：不支持存储过程中任何变量的提交，包括存储过程内声明的变量或者传入的参数。

```
gaussdb=# CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE2(EXP_OUT OUT INT)
AS
EXP INT;
BEGIN
    EXP_OUT := 0;
    COMMIT;
    DBE_OUTPUT.PRINT_LINE('EXP IS:'||EXP);
    EXP_OUT := 1;
    ROLLBACK;
    DBE_OUTPUT.PRINT_LINE('EXP IS:'||EXP);
END;
/
CREATE PROCEDURE
```

- 示例12：不支持出现在SQL中的调用（除了Select Procedure）。

```
gaussdb=# CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE3()
AS
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1 (col1) VALUES (i);
        IF i % 2 = 0 THEN
            EXECUTE IMMEDIATE 'COMMIT';
        ELSE
            EXECUTE IMMEDIATE 'ROLLBACK';
        END IF;
    END LOOP;
END;
/
CREATE PROCEDURE
```

- 示例13：存储过程头带有GUC参数设置的不允许调用commit/rollback语句。

```
gaussdb=# CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE4()
SET ARRAY_NULLS TO "ON"
AS
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1 (col1) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
END;
/
CREATE PROCEDURE
```

- 示例14：游标open的对象不允许为带有commit/rollback语句的存储过程。

```
gaussdb=# CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE5(INTIN IN INT, INTOUT OUT
INT)
AS
BEGIN
```



```
INTOUT := INTIN + 1;
COMMIT;
END;
/
CREATE PROCEDURE

gaussdb=# CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE6()
AS
CURSOR CURSOR1(EXPIN INT)
IS SELECT TRANSACTION_EXAMPLE5(EXPIN);
INTEXP INT;
BEGIN
    FOR i IN 0..20 LOOP
        OPEN CURSOR1(i);
        FETCH CURSOR1 INTO INTEXP;
        INSERT INTO EXAMPLE1(COL1) VALUES (INTEXP);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
        CLOSE CURSOR1;
    END LOOP;
END;
/
CREATE PROCEDURE
```

- 示例15：不支持CURSOR/EXECUTE语句，以及各类表达式内调用COMMIT/ROLLBACK。

```
gaussdb=# CREATE OR REPLACE PROCEDURE exec_func1()
AS
BEGIN
    CREATE TABLE TEST_exec(A INT);
COMMIT;
END;
/
gaussdb=# CREATE OR REPLACE PROCEDURE exec_func2()
AS
BEGIN
EXECUTE exec_func1();
COMMIT;
END;
/
CREATE PROCEDURE
```

- 示例16：存储过程使用保存点回退事务部分修改。

```
gaussdb=# CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE1()
AS
BEGIN
    INSERT INTO EXAMPLE1 VALUES(1);
    SAVEPOINT s1;
    INSERT INTO EXAMPLE1 VALUES(2);
    ROLLBACK TO s1;  -- 回退插入记录2
    INSERT INTO EXAMPLE1 VALUES(3);
END;
/
CREATE PROCEDURE
```

- 示例17：存储过程中使用保存点回退到存储过程外部定义的保存点。

```
gaussdb=# CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE2()
AS
BEGIN
    INSERT INTO EXAMPLE1 VALUES(2);
    ROLLBACK TO s1;  -- 回退插入记录2
    INSERT INTO EXAMPLE1 VALUES(3);
END;
/
CREATE PROCEDURE

gaussdb=# BEGIN;
```



```
BEGIN
gaussdb=# INSERT INTO EXAMPLE1 VALUES(1);
INSERT 0 1
gaussdb=# SAVEPOINT s1;
SAVEPOINT
gaussdb=# CALL STP_SAVEPOINT_EXAMPLE2();
 stp_savepoint_example2
------------------------

(1 row)

gaussdb=# SELECT * FROM EXAMPLE1;
 col1
------
    1
    3
(2 rows)
gaussdb=# COMMIT;
COMMIT
```

- 示例18：不支持存储过程中释放存储过程外部定义的保存点。

```
gaussdb=# CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE3()
AS
BEGIN
    INSERT INTO EXAMPLE1 VALUES(2);
    RELEASE SAVEPOINT s1;  -- 释放存储过程外部定义的保存点
    INSERT INTO EXAMPLE1 VALUES(3);
END;
/
CREATE PROCEDURE
gaussdb=# BEGIN;
BEGIN
gaussdb=# INSERT INTO EXAMPLE1 VALUES(1);
INSERT 0 1
gaussdb=# SAVEPOINT s1;
SAVEPOINT
gaussdb=# CALL STP_SAVEPOINT_EXAMPLE3();
ERROR:  cannot release outer savepoint
CONTEXT:  PL/pgSQL function stp_savepoint_example3() line 4 at RELEASE SAVEPOINT
gaussdb=# COMMIT;
ROLLBACK
```

- 示例19：存储过程外部SQL/其它存储过程回退到存储过程中定义的保存点。

```
gaussdb=# CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE4()
AS
BEGIN
    INSERT INTO EXAMPLE1 VALUES(1);
    SAVEPOINT s1;
    INSERT INTO EXAMPLE1 VALUES(2);
END;
/
CREATE PROCEDURE
gaussdb=# BEGIN;
BEGIN
gaussdb=# INSERT INTO EXAMPLE1 VALUES(3);
INSERT 0 1
gaussdb=# CALL STP_SAVEPOINT_EXAMPLE4();
 stp_savepoint_example4
------------------------

(1 row)
gaussdb=# ROLLBACK TO SAVEPOINT s1; --回退存储过程中插入记录2
ROLLBACK
gaussdb=# SELECT * FROM EXAMPLE1;
 col1
------
    1
    3
    3
    1
```



```
(4 rows)
gaussdb=# COMMIT;
COMMIT
gaussdb=# DROP TABLE EXAMPLE1;
DROP TABLE
```

10.10 其他语句

10.10.1 锁操作

GaussDB提供了多种锁模式用于控制对表中数据的并发访问。这些模式可以用在MVCC（多版本并发控制）无法给出期望行为的场合。同样，大多数GaussDB命令自动施加恰当的锁，以保证被引用的表在命令的执行过程中不会以一种不兼容的方式被删除或者修改。比如，在存在其他并发操作的时候，ALTER TABLE是不能在同一个表上执行的。

完整的锁操作请参见**LOCK**。

10.10.2 游标操作

GaussDB中游标（cursor）是系统为用户开设的一个数据缓冲区，存放着SQL语句的执行结果。每个游标区都有一个名称。用户可以用SQL语句逐一从游标中获取记录，并赋给主变量，交由主语言进一步处理。

游标的操作主要有游标的定义、打开、获取和关闭。

完整的游标操作示例可参考**显式游标**。

10.11 游标

10.11.1 游标概述

为了处理SQL语句，存储过程进程分配一段内存区域来保存上下文联系。游标是指向上下文区域的句柄或指针。借助游标，存储过程可以控制上下文区域的变化。

须知

- 当游标作为存储过程的返回值时，如果使用JDBC调用该存储过程，返回的游标将不可用。
- 存储过程内commit/rollback时，显式游标为保证在commit/rollback后仍可用，会缓存游标所有数据，若游标数据量较大，此过程耗时可能较长。
- 在存储过程内修改表数据后，开启和该表相关的游标，并在rollback后继续fetch游标内数据，会报错。

游标的使用分为显式游标和隐式游标。对于不同的SQL语句，游标的使用情况不同，详细信息请参见**表10-2**。

表 10-2 游标使用情况

SQL语句	游标
非查询语句	隐式的
结果是单行的查询语句	隐式的或显式的
结果是多行的查询语句	显式的

10.11.2 显式游标

显式游标主要用于对查询语句的处理，尤其是在查询结果为多条记录的情况下。

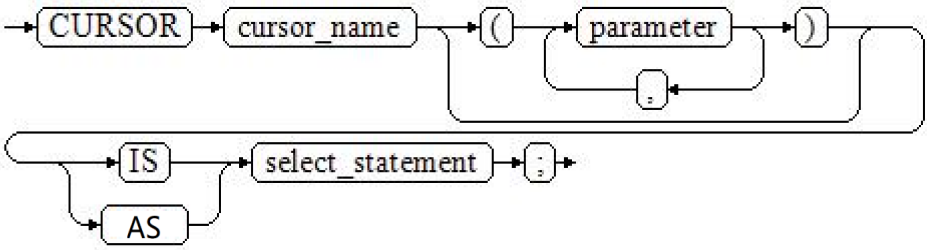
处理步骤

显式游标处理需六个PL/SQL步骤：

步骤1 **定义静态游标：**就是定义一个游标名，以及与其相对应的SELECT语句。

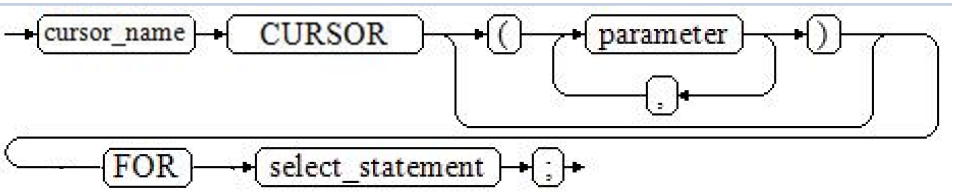
定义静态游标的语法图，请参见图10-27。

图 10-27 static_cursor_define::=



或

图 10-28 static_cursor_define::=



参数说明：

- cursor_name：定义的游标名。
- parameter：游标参数，只能为输入参数，其格式为：
  ```
  parameter_name datatype
  ```
- select_statement：查询语句。

说明

- 根据执行计划的不同，系统会自动判断该游标是否可以用于以倒序的方式检索数据行。
- 语法上支持parameter为输出参数，但其行为与输入参数保持一致。

定义动态游标：指ref游标，可以通过一组静态的SQL语句动态的打开游标。首先定义ref游标类型，然后定义该游标类型的游标变量，在打开游标时通过OPEN FOR动态绑定SELECT语句。

定义动态游标的语法图，请参见图10-29和图10-30。

图 10-29 cursor_typename::=

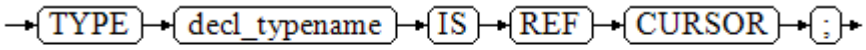
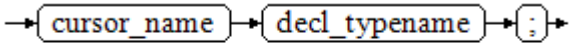


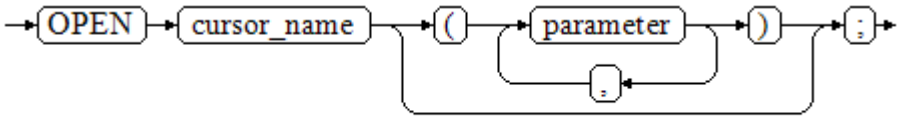
图 10-30 dynamic_cursor_define::=



步骤2 **打开静态游标：**就是执行游标所对应的SELECT语句，将其查询结果放入工作区，并且指针指向工作区的首部，标识游标结果集合。如果游标查询语句中带有FOR UPDATE选项，OPEN语句还将锁定数据库表中游标结果集合对应的数据行。

打开静态游标的语法图，请参见图10-31。

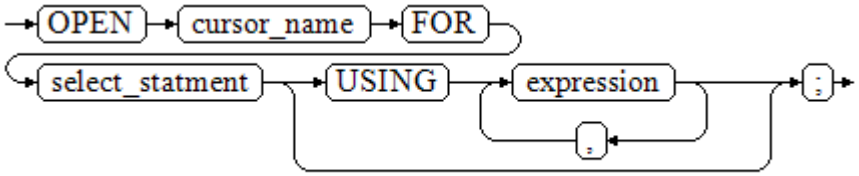
图 10-31 open_static_cursor::=



打开动态游标：可以通过OPEN FOR语句打开动态游标，动态绑定SQL语句。

打开动态游标的语法图，请参见图10-32。

图 10-32 open_dynamic_cursor::=

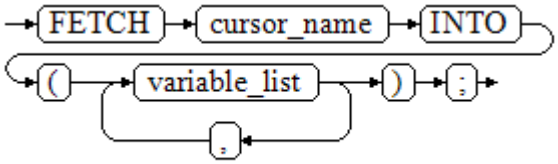


PL/SQL程序不能用OPEN语句重复打开一个游标。

步骤3 提取游标数据：检索结果集合中的数据行，放入指定的输出变量中。

提取游标数据的语法图，请参见图10-33。

图 10-33 fetch_cursor::=



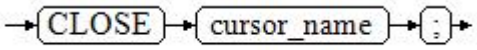
步骤4 对该记录进行处理。

步骤5 继续处理，直到活动集合中没有记录。

步骤6 关闭游标：当提取和处理完游标结果集合数据后，应及时关闭游标，以释放该游标所占用的系统资源，并使该游标的工作区变成无效，不能再使用FETCH语句获取其中数据。关闭后的游标可以使用OPEN语句重新打开。

关闭游标的语法图，请参见**图10-34**。

图 10-34 close_cursor::=



----结束

属性

游标的属性用于控制程序流程或者了解程序的状态。当运行DML语句时，PL/SQL打开一个内建游标并处理结果，游标是维护查询结果的内存中的一个区域，游标在运行DML语句时打开，完成后关闭。显式游标的属性为：

- %FOUND布尔型属性：当最近一次读记录时成功返回，则值为TRUE。
- %NOTFOUND布尔型属性：当最近一次读记录时失败返回，则值为TRUE。
- %ISOPEN布尔型属性：当游标已打开时返回TRUE。
- %ROWCOUNT数值型属性：返回已从游标中读取的记录数。

示例

前置DDL、DML，本节后续示例依赖此用例。

```
gaussdb=# DROP SCHEMA IF EXISTS hr cascade;
gaussdb=# CREATE schema hr;
gaussdb=# SET current_schema = hr;
gaussdb=# DROP TABLE IF EXISTS sections;
gaussdb=# DROP TABLE IF EXISTS staffs;
gaussdb=# DROP TABLE IF EXISTS department;
--创建部门表。
gaussdb=# CREATE TABLE sections(
    section_name varchar(100),
    place_id     int,
    section_id   int
);
gaussdb=# INSERT INTO sections VALUES ('hr',1,1);

--创建员工表。
gaussdb=# CREATE TABLE staffs(
    staff_id number(6),
    salary number(8,2),
```



```
    section_id int,
    first_name varchar(20)
);
gaussdb=# INSERT INTO staffs VALUES (1,100,1,'Tom');

--创建部门表。
gaussdb=# CREATE TABLE department(
    section_id int
);
--游标参数的传递方法。
gaussdb=# CREATE OR REPLACE PROCEDURE cursor_proc1()
AS
DECLARE
    DEPT_NAME VARCHAR(100);
    DEPT_LOC NUMBER(4);
    --定义游标
    CURSOR C1 IS
        SELECT section_name, place_id FROM hr.sections WHERE section_id <= 50;
    CURSOR C2(sect_id INTEGER) IS
        SELECT section_name, place_id FROM hr.sections WHERE section_id <= sect_id;
    TYPE CURSOR_TYPE IS REF CURSOR;
    C3 CURSOR_TYPE;
    SQL_STR VARCHAR(100);
BEGIN
    OPEN C1;--打开游标
    LOOP
        --通过游标取值
        FETCH C1 INTO DEPT_NAME, DEPT_LOC;
        EXIT WHEN C1%NOTFOUND;
        DBE_OUTPUT.PRINT_LINE(DEPT_NAME||'---'||DEPT_LOC);
    END LOOP;
    CLOSE C1;--关闭游标

    OPEN C2(10);
    LOOP
        FETCH C2 INTO DEPT_NAME, DEPT_LOC;
        EXIT WHEN C2%NOTFOUND;
        DBE_OUTPUT.PRINT_LINE(DEPT_NAME||'---'||DEPT_LOC);
    END LOOP;
    CLOSE C2;

    SQL_STR := 'SELECT section_name, place_id FROM hr.sections WHERE section_id <= :DEPT_NO;';
    OPEN C3 FOR SQL_STR USING 50;
    LOOP
        FETCH C3 INTO DEPT_NAME, DEPT_LOC;
        EXIT WHEN C3%NOTFOUND;
        DBE_OUTPUT.PRINT_LINE(DEPT_NAME||'---'||DEPT_LOC);
    END LOOP;
    CLOSE C3;
END;
/
CREATE PROCEDURE

gaussdb=# CALL cursor_proc1();
hr---1
hr---1
hr---1
 cursor_proc1
--------------

(1 row)

gaussdb=# DROP PROCEDURE cursor_proc1;
DROP PROCEDURE
--给工资低于3000的员工增加工资500。
gaussdb=# CREATE TABLE hr.staffs_t1 AS TABLE hr.staffs;
INSERT 0 1

gaussdb=# CREATE OR REPLACE PROCEDURE cursor_proc2()
```



```
AS
DECLARE
  V_EMPNO  NUMBER(6);
  V_SAL    NUMBER(8,2);
  CURSOR C IS SELECT staff_id, salary FROM hr.staffs_t1;
BEGIN
  OPEN C;
  LOOP
    FETCH C INTO V_EMPNO, V_SAL;
    EXIT WHEN C%NOTFOUND;
    IF V_SAL<=3000 THEN
         UPDATE hr.staffs_t1 SET salary =salary + 500 WHERE staff_id = V_EMPNO;
    END IF;
  END LOOP;
  CLOSE C;
END;
/
CREATE PROCEDURE

gaussdb=# CALL cursor_proc2();
 cursor_proc2
--------------

(1 row)

--删除存储过程
gaussdb=# DROP PROCEDURE cursor_proc2;
DROP PROCEDURE
gaussdb=# DROP TABLE hr.staffs_t1;
DROP TABLE
--SYS_REFCURSOR类型作为函数参数
gaussdb=# CREATE OR REPLACE PROCEDURE proc_sys_ref(O OUT SYS_REFCURSOR)
IS
C1 SYS_REFCURSOR;
BEGIN
OPEN C1 FOR SELECT section_ID FROM HR.sections ORDER BY section_ID;
O := C1;
END;
/
CREATE PROCEDURE

gaussdb=# DECLARE
C1 SYS_REFCURSOR;
TEMP NUMBER(4);
BEGIN
proc_sys_ref(C1);
LOOP
  FETCH C1 INTO TEMP;
  DBE_OUTPUT.PRINT_LINE(C1%ROWCOUNT);
  EXIT WHEN C1%NOTFOUND;
END LOOP;
END;
/
1
1
ANONYMOUS BLOCK EXECUTE

--删除存储过程
gaussdb=# DROP PROCEDURE proc_sys_ref;
DROP PROCEDURE
```

10.11.3 隐式游标

对于非查询语句，如修改、删除操作，则由系统自动地为这些操作设置游标并创建其工作区，这些由系统隐含创建的游标称为隐式游标，隐式游标的名称为SQL，这是由系统定义的。

简介

对于隐式游标的操作，如定义、打开、取值及关闭操作，都由系统自动地完成，无需用户进行处理。用户只能通过隐式游标的相关属性，来完成相应的操作。在隐式游标的工作区中，所存放的数据是最新处理的一条SQL语句所包含的数据，与用户自定义的显式游标无关。

格式调用为： SQL%

说明

- INSERT，UPDATE，DELETE，SELECT语句中不必明确定义游标。
- 兼容A模式下，GUC参数behavior_compat_options为compat_cursor时，隐式游标跨存储过程有效。
- 隐式游标属性不受commit\rollback操作影响。打开guc参数set behavior_compat_options='compat_cursor'时，隐式游标属性受savepoint\commit \rollback操作影响，将重置相关属性为默认值。

属性

隐式游标属性为：

- SQL%FOUND布尔型属性：当最近一次读记录时成功返回，则值为TRUE。
- SQL%NOTFOUND布尔型属性：当最近一次读记录时失败返回，则值为TRUE。
- SQL%ROWCOUNT数值型属性：返回已从游标中读取的记录数。
- SQL%ISOPEN布尔型属性：取值总是FALSE。SQL语句执行完毕立即关闭隐式游标。

示例

```
--删除员工表hr.staffs中某部门的所有员工，如果该部门中已没有员工，则在部门表hr.sections中删除该部门。
gaussdb=# CREATE OR REPLACE PROCEDURE proc_cursor3()
AS
   DECLARE
   V_DEPTNO NUMBER(4) := 100;
   BEGIN
      DELETE FROM hr.staffs WHERE section_ID = V_DEPTNO;
      --根据游标状态做进一步处理
      IF SQL%NOTFOUND THEN
      DELETE FROM hr.department WHERE section_ID = V_DEPTNO;
      END IF;
   END;
/
CREATE PROCEDURE
gaussdb=# CALL proc_cursor3();
 proc_cursor3
--------------

(1 row)

--删除存储过程和临时表
gaussdb=# DROP PROCEDURE proc_cursor3;
DROP PROCEDURE
```

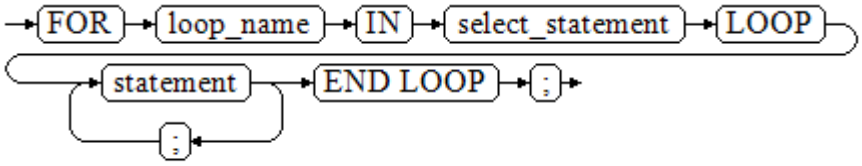
10.11.4 游标循环

游标在WHILE语句、LOOP语句中的使用称为游标循环，一般这种循环都需要使用OPEN、FETCH和CLOSE语句。下面要介绍的一种循环不需要这些操作，可以简化游标循环的操作，这种循环方式适用于静态游标的循环，不用执行静态游标的四个步骤。

语法

FOR AS循环的语法请参见图10-35。

图 10-35 FOR_AS_loop::=



注意事项

- 不能在该循环语句中对查询的表进行更新操作。
- 变量loop_name会自动定义且只在此循环中有效，类型和select_statement的查询结果类型一致。loop_name的取值就是select_statement的查询结果。
- 变量loop_name在编译过程中不会解析具体的类型，如果有需要解析具体类型的场景（比如loop_name作为重载函数或者存储过程的出入参）会编译报错。如需解析变量的具体类型，可以设置参数behavior_compat_options值为allow_procedure_compile_check。
- 游标的属性中%FOUND、%NOTFOUND、%ROWCOUNT在GaussDB数据库中都是访问同一个内部变量，事务和匿名块不支持多个游标同时访问。

示例

```
gaussdb=# BEGIN
FOR ROW_TRANS IN
        SELECT first_name FROM hr.staffs
    LOOP
        DBE_OUTPUT.PRINT_LINE (ROW_TRANS.first_name );
    END LOOP;
END;
/
Tom
ANONYMOUS BLOCK EXECUTE
--创建表
gaussdb=# CREATE TABLE integerTable1( A INTEGER);
gaussdb=# CREATE TABLE integerTable2( B INTEGER);
gaussdb=# INSERT INTO integerTable2 VALUES(2);

--多游标共享游标属性的标量
gaussdb=# DECLARE
   CURSOR C1 IS SELECT A FROM integerTable1;--声明游标
   CURSOR C2 IS SELECT B FROM integerTable2;
   PI_A INTEGER;
   PI_B INTEGER;
BEGIN
   OPEN C1;--打开游标
   OPEN C2;
   FETCH C1 INTO PI_A; ----  C1%FOUND 和 C2%FOUND 值为 FALSE
   FETCH C2 INTO PI_B; ----  C1%FOUND 和 C2%FOUND 的值都为 TRUE
   --判断游标状态
   IF C1%FOUND THEN
      IF C2%FOUND THEN
        DBE_OUTPUT.PRINT_LINE('Dual cursor share parameter.');
      END IF;
   END IF;
   CLOSE C1;--关闭游标
```



```
  CLOSE C2;
END;
/
ANONYMOUS BLOCK EXECUTE
--删除临时表
gaussdb=# DROP TABLE integerTable1;
DROP TABLE
gaussdb=# DROP TABLE integerTable2;
DROP TABLE
```

10.12 高级包

高级包现有两套接口，第一套为基础接口，第二套是为了提高易用性做了二次封装的接口，推荐使用第二套接口。

10.12.1 基础接口

10.12.1.1 PKG_SERVICE

PKG_SERVICE支持的所有接口请参见**表10-3**。

表 10-3 PKG_SERVICE

接口名称	描述
PKG_SERVICE.SQL_IS_CONTEXT_ACTIVE	确认该CONTEXT是否已注册。
PKG_SERVICE.SQL_CLEAN_ALL_CONTEXTS	取消所有注册的CONTEXT。
PKG_SERVICE.SQL_REGISTER_CONTEXT	注册一个CONTEXT。
PKG_SERVICE.SQL_UNREGISTER_CONTEXT	取消注册该CONTEXT。
PKG_SERVICE.SQL_SET_SQL	向CONTEXT设置一条SQL语句，目前只支持SELECT。
PKG_SERVICE.SQL_RUN	在一个CONTEXT上执行设置的SQL语句。
PKG_SERVICE.SQL_NEXT_ROW	读取该CONTEXT中的下一行数据。
PKG_SERVICE.SQL_GET_VALUE	读取该CONTEXT中动态定义的列值
PKG_SERVICE.SQL_SET_RESULT_TYPE	根据类型OID动态定义该CONTEXT的一个列。
PKG_SERVICE.JOB_CANCEL	通过任务ID来删除定时任务。
PKG_SERVICE.JOB_FINISH	禁用或者启用定时任务。
PKG_SERVICE.JOB_SUBMIT	提交一个定时任务。作业号由系统自动生成或由用户指定。

接口名称	描述
PKG_SERVICE.JOB_UPDATE	修改定时任务的属性，包括任务内容、下次执行时间、执行间隔。
PKG_SERVICE.SUBMIT_ON_NODES	提交一个任务到所有节点，作业号由系统自动生成。
PKG_SERVICE.ISUBMIT_ON_NODES	提交一个任务到所有节点，作业号由用户指定。
PKG_SERVICE.SQL_GET_ARRAY_RESULT	获取该CONTEXT中返回的数组值。
PKG_SERVICE.SQL_GET_VARIABLE_RESULT	获取该CONTEXT中返回的列值。

- PKG_SERVICE.SQL_IS_CONTEXT_ACTIVE

  该函数用来确认一个CONTEXT是否已注册。该函数传入想查找的CONTEXT ID，如果该CONTEXT存在返回TRUE，反之返回FALSE。

  PKG_SERVICE.SQL_IS_CONTEXT_ACTIVE函数原型为：

  ```
  PKG_SERVICE.SQL_IS_CONTEXT_ACTIVE(
   context_id      IN INTEGER
  )
  RETURN BOOLEAN;
  ```

  表 10-4 PKG_SERVICE.SQL_IS_CONTEXT_ACTIVE 接口说明

参数名称	描述
context_id	想查找的CONTEXT ID号。

- PKG_SERVICE.SQL_CLEAN_ALL_CONTEXTS

  该函数用来取消所有CONTEXT

  PKG_SERVICE.SQL_CLEAN_ALL_CONTEXTS函数原型为：

  ```
  PKG_SERVICE.SQL_CLEAN_ALL_CONTEXTS(
  )
  RETURN VOID;
  ```

- PKG_SERVICE.SQL_REGISTER_CONTEXT

  该函数用来打开一个CONTEXT，是后续对该CONTEXT进行各项操作的前提。该函数不传入任何参数，内部自动递增生成CONTEXT ID，并作为返回值返回给integer定义的变量。

  PKG_SERVICE.SQL_REGISTER_CONTEXT函数原型为：

  ```
  DBE_SQL.REGISTER_CONTEXT(
  )
  RETURN INTEGER;
  ```

- PKG_SERVICE.SQL_UNREGISTER_CONTEXT

  该函数用来关闭一个CONTEXT，是该CONTEXT中各项操作的结束。如果在存储过程结束时没有调用该函数，则该CONTEXT占用的内存仍然会保存，因此关闭

CONTEXT非常重要。由于异常情况的发生会中途退出存储过程，导致CONTEXT未能关闭，因此建议存储过程中有异常处理，将该接口包含在内。

PKG_SERVICE.SQL_UNREGISTER_CONTEXT函数原型为：

```
PKG_SERVICE.SQL_UNREGISTER_CONTEXT(
 context_id      IN INTEGER
)
RETURN INTEGER;
```

表 10-5 PKG_SERVICE.SQL_UNREGISTER_CONTEXT 接口说明

参数名称	描述
context_id	打算关闭的CONTEXT ID号。

- PKG_SERVICE.SQL_SET_SQL

  该函数用来解析给定游标的查询语句，被传入的查询语句会立即执行。目前仅支持SELECT查询语句的解析，且语句参数仅可通过text类型传递，长度不大于1G。

  PKG_SERVICE.SQL_SET_SQL函数的原型为：

  ```
  PKG_SERVICE.SQL_SET_SQL(
  context_id      IN INTEGER,
  query_string    IN TEXT,
  language_flag   IN INTEGER
  )
  RETURN BOOLEAN;
  ```

  表 10-6 PKG_SERVICE.SQL_SET_SQL 接口说明

参数名称	描述
context_id	执行查询语句解析的CONTEXT ID。
query_string	执行的查询语句。
language_flag	版本语言号，目前只支持1。

- PKG_SERVICE.SQL_RUN

  该函数用来执行一个给定的CONTEXT。该函数接收一个CONTEXT ID，运行后获得的数据用于后续操作。目前仅支持SELECT查询语句的执行。

  PKG_SERVICE.SQL_RUN函数的原型为：

  ```
  PKG_SERVICE.SQL_RUN(
  context_id      IN INTEGER,
  )
  RETURN INTEGER;
  ```

  表 10-7 PKG_SERVICE.SQL_RUN 接口说明

参数名称	描述
context_id	执行查询语句解析的CONTEXT ID。

- PKG_SERVICE.SQL_NEXT_ROW

  该函数返回执行SQL实际返回的数据行数，每一次运行该接口都会获取到新的行数的集合，直到数据读取完毕获取不到新行为止。

PKG_SERVICE.SQL_NEXT_ROW函数的原型为：

```
PKG_SERVICE.SQL_NEXT_ROW(
context_id      IN INTEGER,
)
RETURN INTEGER;
```

表 10-8 PKG_SERVICE.SQL_NEXT_ROW 接口说明

参数名称	描述
context_id	执行的CONTEXT ID。

- PKG_SERVICE.SQL_GET_VALUE

  该函数用来返回给定CONTEXT中给定位置的CONTEXT元素值，该接口访问的是PKG_SERVICE.SQL_NEXT_ROW获取的数据。

  PKG_SERVICE.SQL_GET_VALUE函数的原型为：

  ```
  PKG_SERVICE.SQL_GET_VALUE(
  context_id          IN     INTEGER,
  pos                 IN     INTEGER,
  col_type            IN     ANYELEMENT
  )
  RETURN ANYELEMENT;
  ```

  表 10-9 PKG_SERVICE.SQL_GET_VALUE 接口说明

参数名称	描述
context_id	执行的CONTEXT ID。
pos	动态定义列在查询中的位置。
col_type	任意类型变量，定义列的返回值类型。

- PKG_SERVICE.SQL_SET_RESULT_TYPE

  该函数用来定义从给定CONTEXT返回的列，该接口只能应用于SELECT定义的CONTEXT中。定义的列通过查询列表的相对位置来标识，
  PKG_SERVICE.SQL_SET_RESULT_TYPE函数的原型为：

  ```
  PKG_SERVICE.SQL_SET_RESULT_TYPE(
  context_id      IN INTEGER,
  pos             IN INTEGER,
  coltype_oid     IN ANYELEMENT,
  maxsize         IN INTEGER
  )
  RETURN INTEGER;
  ```

  表 10-10 PKG_SERVICE.SQL_SET_RESULT_TYPE 接口说明

参数名称	描述
context_id	执行的CONTEXT ID。
pos	动态定义列在查询中的位置。
coltype_oid	任意类型的变量，可根据变量类型得到对应类型OID。

参数名称	描述
maxsize	定义的列的长度。

- PKG_SERVICE.JOB_CANCEL
  存储过程CANCEL删除指定的定时任务。
  PKG_SERVICE.JOB_CANCEL函数原型为：

```
PKG_SERVICE.JOB_CANCEL(
id  IN  INTEGER);
```

表 10-11 PKG_SERVICE.JOB_CANCEL 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	integer	IN	否	指定的作业号。

- PKG_SERVICE.JOB_FINISH
  存储过程FINISH禁用或者启用定时任务。
  PKG_SERVICE.JOB_FINISH函数原型为：

```
PKG_SERVICE.JOB_FINISH(
id          IN   INTEGER,
broken      IN   BOOLEAN,
next_time   IN   TIMESTAMP  DEFAULT  sysdate);
```

表 10-12 PKG_SERVICE.JOB_FINISH 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	integer	IN	否	指定的作业号。
broken	boolean	IN	否	状态标志位，true代表禁用，false代表启用。根据true或false值更新当前job；如果为空值，则不改变原有job的状态。
next_time	timestamp	IN	是	下次运行时间，默认为当前系统时间。如果参数broken状态为true，则更新该参数为'4000-1-1'；如果参数broken状态为false，且如果参数next_time不为空值，则更新指定job的next_time值，如果next_time为空值，则不更新next_time值。该参数可以省略，为默认值。

- PKG_SERVICE.JOB_SUBMIT
  存储过程JOB_SUBMIT提交一个系统提供的定时任务。

PKG_SERVICE.JOB_SUBMIT函数原型为：

```
PKG_SERVICE.JOB_SUBMIT(
id            IN   BIGINT,
content       IN   TEXT,
next_date     IN   TIMESTAMP DEFAULT sysdate,
interval_time IN   TEXT  DEFAULT 'null',
job           OUT  INTEGER);
```

说明

当创建一个定时任务（JOB）时，系统默认将当前数据库和用户名与当前创建的定时任务绑定起来。该接口函数可以通过call或select调用，如果通过select调用，可以不填写出参。如果在存储过程中，则需要通过perform调用该接口函数。如果提交的sql语句任务使用到非public的schema，应该指定表或者函数的schema，或者在sql语句前添加set current_schema = xxx;语句。

表 10-13 PKG_SERVICE.JOB_SUBMIT 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	bigint	IN	否	作业号。如果传入id为NULL，则内部会生成作业ID。
content	text	IN	否	要执行的SQL语句。支持一个或多个‘DML’，‘匿名块’，‘调用存储过程的语句’或3种混合的场景。
next_time	timestamp	IN	否	下次作业运行时间。默认值为当前系统时间（sysdate）。如果是过去时间，在提交作业时表示立即执行。
interval_time	text	IN	是	用来计算下次作业运行时间的时间表达式，可以是interval表达式，也可以是sysdate加上一个numeric值（例如：sysdate+1.0/24）。如果为空值或字符串"null"表示只执行一次，执行后JOB状态STATUS变成'd' 不再执行。
job	integer	OUT	否	作业号。范围为1～32767。当使用select调用pkg_service.job_submit时，该参数可以省略。

示例：

```
CREATE TABLE test_table(a int);
CREATE TABLE

CREATE OR REPLACE PROCEDURE test_job(a in int) IS
BEGIN
INSERT INTO test_table VALUES(a);
COMMIT;
END;
/
CREATE PROCEDURE
SELECT PKG_SERVICE.JOB_SUBMIT(NULL, 'call pro_xxx();', to_date('20180101','yyyymmdd'),'sysdate
+1');
 job_submit
```



```
------------
     28269
(1 row)
SELECT PKG_SERVICE.JOB_SUBMIT(NULL, 'call pro_xxx();', to_date('20180101','yyyymmdd'),'sysdate
+1.0/24');
 job_submit
------------
       1506
(1 row)

CALL PKG_SERVICE.JOB_SUBMIT(NULL, 'INSERT INTO T_JOB  VALUES(1);  call pro_1(); call pro_2();',
add_months(to_date('201701','yyyymm'),1), 'date_trunc(''day'',SYSDATE) + 1 +(8*60+30.0)/
(24*60)' ,:jobid);
  job
-------
 14131
(1 row)

SELECT PKG_SERVICE.JOB_SUBMIT (101, 'insert_msg_statistic1;', sysdate, 'sysdate+3.0/24');
 job_submit
------------
       101
(1 row)
```

- PKG_SERVICE.JOB_UPDATE

  存储过程UPDATE修改定时任务的属性，包括任务内容、下次执行时间、执行间隔。

  PKG_SERVICE.JOB_UPDATE函数原型为：

```
PKG_SERVICE.JOB_UPDATE(
id             IN   BIGINT,
next_time      IN   TIMESTAMP,
interval_time  IN   TEXT,
content        IN   TEXT);
```

表 10-14 PKG_SERVICE.JOB_UPDATE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	integer	IN	否	指定的作业号。
next_time	timestamp	IN	是	下次运行时间。如果该参数为空值，则不更新指定job的next_time值，否则更新指定job的next_time值。
interval_time	text	IN	是	用来计算下次作业运行时间的时间表达式。如果该参数为空值，则不更新指定job的interval_time值；如果该参数不为空值，会校验interval_time是否为有效的时间类型或interval类型，则更新指定job的interval_time值。如果为字符串"null"表示只执行一次，执行后JOB状态STATUS变成'd' 不再执行。
content	text	IN	是	执行的存储过程名或者sql语句块。如果该参数为空值，则不更新指定job的content值，否则更新指定job的content值。

示例：

```
CALL PKG_SERVICE.JOB_UPDATE(101, sysdate, 'sysdate + 1.0/1440', 'call userproc();');
 job_update
------------


(1 row)
CALL PKG_SERVICE.JOB_UPDATE(101, sysdate, 'sysdate + 1.0/1440', 'insert into tbl_a values(sysdate);');
 job_update
------------


(1 row)
```

- PKG_SERVICE.SUBMIT_ON_NODES

  存储过程SUBMIT_ON_NODES创建一个节点上的定时任务，仅sysadmin/monitor admin有此权限。

  PKG_SERVICE.SUBMIT_ON_NODES函数原型为：

  ```
  PKG_SERVICE.SUBMIT_ON_NODES(
  node_name     IN   NAME,
  database      IN   NAME,
  what          IN   TEXT,
  next_date     IN   TIMESTAMP WITHOUT TIME ZONE,
  job_interval IN   TEXT,
  job           OUT  INTEGER);
  ```

  表 10-15 PKG_SERVICE.SUBMIT_ON_NODES 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
node_name	text	IN	否	指定作业的执行节点，当前仅支持值为'ALL_NODE'（在所有节点执行）。
database	text	IN	否	数据库实例作业所使用的database，节点类型为'ALL_NODE'时仅支持值为'postgres'。
what	text	IN	否	要执行的SQL语句。支持一个或多个‘DML’，‘匿名块’，‘调用存储过程的语句’或3种混合的场景。
nextdate	timestamp	IN	否	下次作业运行时间。默认值为当前系统时间（sysdate）。如果是过去时间，在提交作业时表示立即执行。
job_interval	text	IN	否	用来计算下次作业运行时间的时间表达式，可以是interval表达式，也可以是sysdate加上一个numeric值（例如：sysdate+1.0/24）。如果为空值或字符串"null"表示只执行一次，执行后JOB状态STATUS变成'd'不再执行。
job	integer	OUT	否	作业号。范围为1～32767。当使用select调用dbms.submit_on_nodes时，该参数可以省略。

示例：


```
SELECT pkg_service.submit_on_nodes('ALL_NODE', 'postgres', 'select
capture_view_to_json(''dbe_perf.statement'', 0);', sysdate, 'interval ''60 second''');
 submit_on_nodes
-----------------
           25376
(1 row)
```

- PKG_SERVICE.ISUBMIT_ON_NODES

  ISUBMIT_ON_NODES与SUBMIT_ON_NODES语法功能相同，但其第一个参数是入参，即指定的作业号，SUBMIT最后一个参数是出参，表示系统自动生成的作业号。仅sysadmin/monitor admin有此权限。

- PKG_SERVICE.SQL_GET_ARRAY_RESULT

  该函数用来返回绑定的数组类型的OUT参数的值，可以用来获取存储过程中的OUT参数。

  PKG_SERVICE.SQL_GET_ARRAY_RESULT函数原型为：

```
PKG_SERVICE.SQL_GET_ARRAY_RESULT(
    context_id in int,
    pos in VARCHAR2,
    column_value inout anyarray,
    result_type in anyelement
);
```

表 10-16 PKG_SERVICE.SQL_GET_ARRAY_RESULT 接口说明

参数名称	描述
context_id	想查找的CONTEXT ID号。
pos	绑定的参数名。
column_value	返回值。
result_type	返回值类型。

- PKG_SERVICE.SQL_GET_VARIABLE_RESULT

  该函数用来返回绑定的非数组类型的OUT参数的值，可以用来获取存储过程中的OUT参数。

  PKG_SERVICE.SQL_GET_VARIABLE_RESULT函数原型为：

```
PKG_SERVICE.SQL_GET_VARIABLE_RESULT(
    context_id in int,
    pos in VARCHAR2,
    result_type in anyelement
)
RETURNS anyelement;
```

表 10-17 PKG_SERVICE.SQL_GET_VARIABLE_RESULT 接口说明

参数名称	描述
context_id	想查找的CONTEXT ID号。
pos	绑定的参数名。
result_type	返回值类型。

10.12.1.2 PKG_UTIL

PKG_UTIL支持的所有接口请参见**表10-18**：

表 10-18 PKG_UTIL

接口名称	描述
PKG_UTIL.LOB_READ	读取lob对象的一部分。
PKG_UTIL.LOB_WRITE	将源对象按照指定格式写入到目标对象。
PKG_UTIL.LOB_APPEND	将lob源对象追加到目标lob对象。
PKG_UTIL.LOB_COMPARE	根据指定长度比较两个lob对象。
PKG_UTIL.LOB_MATCH	返回一个字符串在LOB中第N次出现的位置。
PKG_UTIL.LOB_RESET	将lob的指定位置重置为指定字符。
PKG_UTIL.LOB_GET_LENGTH	该函数获取并返回指定的LOB类型对象的长度。
PKG_UTIL.LOB_READ_HUGE	根据指定的长度及起始位置偏移读取LOB内容的一部分,并返回读取到的LOB和长度。
PKG_UTIL.LOB_WRITEAPPEND_HUGE	该函数将源blob/clob对象读取指定长度内容，并追加到目标blob/clob对象, 并返回目标对象。
PKG_UTIL.LOB_APPEND_HUGE	该函数将源blob/clob对象追加到目标blob/clob对象, 并返回目标对象。
PKG_UTIL.READ_BFILE_TO_BLOB	该函数将源BFILE文件读取成目标BLOB对象, 并返回目标对象。
PKG_UTIL.LOB_COPY_HUGE	该函数将源blob/clob对象，从指定偏移读取指定长度内容，写入到目标blob/clob对象的指定偏移位置, 并返回目标对象。
PKG_UTIL.BLOB_RESET	该函数将BLOB中一段数据set为value值，返回处理后的BLOB以及实际处理的长度。
PKG_UTIL.CLOB_RESET	该函数将一段数据set为空格，返回处理后的CLOB以及实际处理的长度。
PKG_UTIL.LOADBLOBFROMFILE	将源BFILE对象，从指定偏移读取指定长度内容，写入到目标BLOB对象的指定偏移位置, 并返回目标对象，读取位置，写入位置。

接口名称	描述
PKG_UTIL.LOADCLOBFROMFILE	将源BFILE对象，从指定偏移读取指定长度内容，写入到目标CLOB对象的指定偏移位置, 并返回目标对象，读取位置，写入位置。
PKG_UTIL.LOB_CONVERTTOBLOB_HUGE	将src_clob从指定偏移位置读取指定长度内容转成BLOB，并写入dest_lob的指定位置，amount为要转换的长度。
PKG_UTIL.LOB_CONVERTTOCLOB_HUGE	将src_clob从指定偏移位置读取指定长度内容转成CLOB，并写入dest_lob的指定位置，amount为要转换的长度。
PKG_UTIL.BFILE_GET_LENGTH	该函数获取并返回指定的BFILE文件的长度。
PKG_UTIL.BFILE_OPEN	该函数打开BFILE文件，返回文件描述符。
PKG_UTIL.BFILE_CLOSE	该函数关闭打开的BFILE文件。
PKG_UTIL.LOB_WRITE_HUGE	将源对象从起始位置读取len长度内容，写入目标LOB对象的指定偏移位置，覆盖该位置原本内容, 并返回目标LOB对象。
PKG_UTIL.IO_PRINT	将字符串打印输出。
PKG_UTIL.RAW_GET_LENGTH	获取raw的长度。
PKG_UTIL.RAW_CAST_FROM_VARCHAR2	将varchar2转化为raw。
PKG_UTIL.RAW_CAST_FROM_BINARY_INTEGER	将binary integer转化为raw。
PKG_UTIL.RAW_CAST_TO_BINARY_INTEGER	将raw转化为binary integer。
PKG_UTIL.RANDOM_SET_SEED	设置随机种子。
PKG_UTIL.RANDOM_GET_VALUE	返回随机值。
PKG_UTIL.FILE_SET_DIRNAME	设置当前操作的目录。
PKG_UTIL.FILE_OPEN	根据指定文件名和设置的目录打开一个文件。
PKG_UTIL.FILE_SET_MAX_LINE_SIZE	设置写入文件一行的最大长度。
PKG_UTIL.FILE_IS_CLOSE	检测一个文件句柄是否关闭。
PKG_UTIL.FILE_READ	从一个打开的文件句柄中读取指定长度的数据。
PKG_UTIL.FILE_READLINE	从一个打开的文件句柄中读取一行数据。

接口名称	描述
PKG_UTIL.FILE_WRITE	将BUFFER中的数据写入到文件中。
PKG_UTIL.FILE_WRITELINE	将BUFFER写入文件，并追加换行符。
PKG_UTIL.FILE_NEWLINE	新起一行。
PKG_UTIL.FILE_READ_RAW	从一个打开的文件句柄中读取指定长度的二进制数据。
PKG_UTIL.FILE_WRITE_RAW	将二进制数据写入到文件中。
PKG_UTIL.FILE_FLUSH	将一个文件句柄中的数据写入到物理文件中。
PKG_UTIL.FILE_CLOSE	关闭一个打开的文件句柄。
PKG_UTIL.FILE_REMOVE	删除一个物理文件，操作需要有对应权限。
PKG_UTIL.FILE_RENAME	对于磁盘上的文件进行重命名，类似Unix的mv。
PKG_UTIL.FILE_SIZE	返回文件大小。
PKG_UTIL.FILE_BLOCK_SIZE	返回文件含有的块数量。
PKG_UTIL.FILE_EXISTS	判断文件是否存在。
PKG_UTIL.FILE_GETPOS	返回文件的偏移量，单位字节。
PKG_UTIL.FILE_SEEK	设置文件位置为指定偏移。
PKG_UTIL.FILE_CLOSE_ALL	关闭一个会话中打开的所有文件句柄。
PKG_UTIL.EXCEPTION_REPORT_ERROR	抛出一个异常。
PKG_UTIL.APP_READ_CLIENT_INFO	读取client_info信息。
PKG_UTIL.APP_SET_CLIENT_INFO	设置client_info信息。
PKG_UTIL.LOB_CONVERTTOBLOB	clob类型转换成blob类型。
PKG_UTIL.LOB_CONVERTTOCLOB	blob类型转换成clob类型。
PKG_UTIL.LOB_RAWTOTEXT	raw类型转成text类型。
PKG_UTIL.LOB_TEXTTORAW	text类型转成raw类型。
PKG_UTIL.MATCH_EDIT_DISTANCE_SIMILARITY	计算两个字符串的差距。
PKG_UTIL.RAW_CAST_TO_VARCHAR2	raw类型转成varchar2类型。
PKG_UTIL.SESSION_CLEAR_CONTEXT	清空session_context中的属性值。
PKG_UTIL.SESSION_SEARCH_CONTEXT	查找一个属性值。

接口名称	描述
PKG_UTIL.SESSION_SET_CONTEXT	设置一个属性值。
PKG_UTIL.UTILITY_FORMAT_CALL_STACK	查看存储过程的调用堆栈。
PKG_UTIL.UTILITY_FORMAT_ERROR_BACKTRACE	查看存储过程的错误堆栈。
PKG_UTIL.UTILITY_FORMAT_ERROR_STACK	查看存储过程的报错信息。
PKG_UTIL.UTILITY_GET_TIME	查看系统unix时间戳。
PKG_UTIL.UTILITY_COMPILE_SCHEMA	重编译指定schema包 函数 和存储过程。当编译到的PL/SQL对象遇到报错时，将直接返回，不再继续编译。该包已废弃。推荐使用pkg_util.gs_compile_schema。
PKG_UTIL.GS_COMPILE_SCHEMA	重编译指定schema包、函数和存储过程。当编译遇到PL/SQL对象报错时，异常将会被捕获，继续编译其它对象，直到全部对象编译成功或者到达尝试次数后停止。 通过jdbc执行该高级包，sqlstate会打印00000错误码，00000错误码代表成功完成，具体请参见《错误码参考》中的“SQL标准错误码说明”。

- PKG_UTIL.LOB_GET_LENGTH

  该函数LOB_GET_LENGTH获取输入数据的长度。

  PKG_UTIL.LOB_GET_LENGTH函数原型为：

  ```
  PKG_UTIL.LOB_GET_LENGTH(
  lob       IN   CLOB
  )
  RETURN INTEGER;

  PKG_UTIL.LOB_GET_LENGTH(
  lob       IN   BLOB
  )
  RETURN INTEGER;
  ```


表 10-19 PKG_UTIL.LOB_GET_LENGTH 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
lob	CLOB/BLOB	IN	否	待获取长度的对象。

- PKG_UTIL.LOB_READ

  该函数LOB_READ读取一个对象，并返回指定部分。

  PKG_UTIL.LOB_READ函数原型为：

```
PKG_UTIL.LOB_READ(
lob       IN   ANYELEMENT,
len       IN   INT,
start     IN   INT,
mode      IN   INT
)
RETURN ANYELEMENT;
```

表 10-20 PKG_UTIL.LOB_READ 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
lob	CLOB/BLOB	IN	否	clob或者blob类型数据。
len	INT	IN	否	返回结果长度。
start	INT	IN	否	相较于第一个字符的偏移量。
mode	INT	IN	否	判断读取操作的类型， 0： read; 1 : trim; 2 : substr。

- PKG_UTIL.LOB_WRITE

  该函数LOB_WRITE将源对象按照指定的参数写入目标对象, 并返回目标对象。

  PKG_UTIL.LOB_WRITE函数原型为：

```
PKG_UTIL.LOB_WRITE(
dest_lob    INOUT   BLOB,
src_lob     IN      RAW
len         IN      INT,
start_pos   IN      BIGINT
)
RETURN BLOB;
PKG_UTIL.LOB_WRITE(
dest_lob    INOUT   CLOB,
src_lob     IN      VARCHAR2
len         IN      INT,
start_pos       IN      BIGINT
)
RETURN CLOB;
```


表 10-21 PKG_UTIL.LOB_WRITE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
dest_lob	CLOB / BLOB	INOUT	否	写入的目标对象。
src_lob	CLOB / BLOB	IN	否	被写入的源对象。
len	INT	IN	否	源对象的写入长度。
start_pos	BIGINT	IN	否	目标对象的写入起始位置。

- PKG_UTIL.LOB_APPEND

  该函数LOB_APPEND将源blob/clob对象追加到目标blob/clob对象, 并返回目标对象。

  PKG_UTIL.LOB_APPEND函数原型为：

```
PKG_UTIL.LOB_APPEND(
dest_lob    INOUT   BLOB,
src_lob     IN      BLOB,
len         IN      INT DEFAULT NULL
)
RETURN BLOB;

PKG_UTIL.LOB_APPEND(
dest_lob    INOUT   CLOB,
src_lob     IN      CLOB,
len         IN      INT DEFAULT NULL
)
RETURN CLOB;
```

表 10-22 PKG_UTIL.LOB_APPEND 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
dest_lob	CLOB/ BLOB	INOUT	否	写入的目标blob/clob对象。
src_lob	CLOB/ BLOB	IN	否	被写入的源blob/clob对象。
len	INT	IN	是	src中读取并append到dest上的长度，默认null，将src全部append上去。

- PKG_UTIL.LOB_COMPARE

  该函数LOB_COMPARE按照指定的起始位置、个数比较对象是否相同，lob1大则返回1，lob2大返回-1，相等返回0。

  PKG_UTIL.LOB_COMPARE函数原型为：

```
PKG_UTIL.LOB_COMPARE(
lob1        IN   ANYELEMENT,
lob2        IN   ANYELEMENT,
len         IN   INT DEFAULT 1073741771,
start_pos1      IN   INT DEFAULT 1,
start_pos2      IN   INT DEFAULT 1
)
RETURN INTEGER;
```

  表 10-23 PKG_UTIL.LOB_COMPARE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
lob1	CLOB/BLOB	IN	否	待比较的字符串。
lob2	CLOB/BLOB	IN	否	待比较的字符串。
len	INT	IN	否	比较的长度。
start_pos1	INT	IN	否	lob1起始偏移量。
start_pos2	INT	IN	否	lob2起始偏移量。

- PKG_UTIL.LOB_MATCH

  该函数LOB_MATCH返回pattern出现在lob对象中第match_nth次的位置。

  PKG_UTIL.LOB_MATCH函数原型为：

```
PKG_UTIL.LOB_MATCH(
lob         IN   ANYELEMENT,
pattern     IN   ANYELEMENT,
start       IN   INT,
match_nth   IN   INT DEFAULT 1
)
RETURN INTEGER;
```

  表 10-24 PKG_UTIL.LOB_MATCH 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
lob	CLOB/BLOB	IN	否	待比较的字符串。
pattern	CLOB/BLOB	IN	否	待匹配的pattern。

参数	类型	入参/出参	是否可以为空	描述
start	INT	IN	否	lob的起始比较位置。
match_nth	INT	IN	否	第几次匹配到。

- PKG_UTIL.LOB_RESET

  该函数LOB_RESET清除一段数据为字符value。

  PKG_UTIL.LOB_RESET函数原型为：

```
PKG_UTIL.LOB_RESET(
lob          INOUT   BLOB,
len          INOUT   INT,
start        IN   INT DEFAULT 1 ,
value         IN   INT DEFAULT 0
)
RETURN RECORD;
```

  表 10-25 PKG_UTIL.LOB_RESET 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
lob	BLOB	IN	否	待重置的字符串。
len	INT	IN	否	重置的长度。
start	INT	IN	否	重置的起始位置。
value	INT	IN	是	设置的字符。默认值‘0’。

- PKG_UTIL.LOB_GET_LENGTH

  该函数获取并返回指定的LOB类型对象的长度。

  PKG_UTIL.LOB_GET_LENGTH函数原型为：

```
PKG_UTIL.LOB_GET_LENGTH(
   lob IN BLOB)
RETURN BIGINT;

PKG_UTIL.LOB_GET_LENGTH(
   lob IN CLOB)
RETURN BIGINT;
```


表 10-26 PKG_UTIL.LOB_GET_LENGTH 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
lob	BLOB/CLOB	IN	否	指定的LOB类型对象。

- PKG_UTIL.LOB_READ_HUGE

  根据指定的长度及起始位置偏移读取LOB内容的一部分,并返回读取到的LOB和长度。

  PKG_UTIL.LOB_READ_HUGE函数原型为：

```
PKG_UTIL.LOB_READ_HUGE(
   lob       IN CLOB,
   len       IN BIGINT,
   start_pos IN BIGINT,
   mode      IN INTEGER)
RETURN RECORD;

PKG_UTIL.LOB_READ_HUGE(
   lob       IN BLOB,
   len       IN BIGINT,
   start_pos IN BIGINT,
   mode      IN INTEGER)
RETURN RECORD;

PKG_UTIL.LOB_READ_HUGE(
   fd        IN INTEGER,
   len       IN BIGINT,
   start_pos IN BIGINT,
   mode      IN INTEGER)
RETURN RECORD;
```

表 10-27 PKG_UTIL.LOB_READ_HUGE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
lob/fd	BLOB/CLOB/INTEGER	IN	否	指定的LOB类型对象/BFILE文件的文件描述符。
len	BIGINT	IN	否	读取长度。
start_pos	BIGINT	IN	否	读取起始偏移位置。

参数	类型	入参/出参	是否可以为空	描述
mode	INTEGER	IN	否	read模式（0代表read、1代表trim、2代表substr）。

- PKG_UTIL.LOB_WRITEAPPEND_HUGE

  该函数LOB_WRITEAPPEND_HUGE将源blob/clob对象读取指定长度内容，并追加到目标blob/clob对象, 并返回目标对象。

  PKG_UTIL.LOB_WRITEAPPEND_HUGE函数原型为：

```
PKG_UTIL.LOB_WRITEAPPEND_HUGE(
   dest_lob INOUT CLOB,
   len      IN    INTEGER,
   src_lob  IN    VARCHAR2
)RETURN CLOB;

PKG_UTIL.LOB_WRITEAPPEND_HUGE(
   dest_lob INOUT BLOB,
   len      IN    INTEGER,
   src_lob  IN    RAW
)RETURN BLOB;
```

  表 10-28 PKG_UTIL.LOB_WRITEAPPEND_HUGE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
dest_lob	BLOB/CLOB	INOUT	否	写入的目标BLOB/CLOB对象。
len	INTEGER	IN	是	写入源对象的长度，为NULL则默认写入源对象全部。
src_lob	VARCHAR2/RAW	IN	否	被写入的源BLOB/CLOB对象。

- PKG_UTIL.LOB_APPEND_HUGE

  该函数LOB_APPEND_HUGE将源blob/clob对象追加到目标blob/clob对象, 并返回目标对象。

  PKG_UTIL.LOB_APPEND_HUGE函数原型为：

```
PKG_UTIL.LOB_APPEND_HUGE(
   dest_lob INOUT BLOB,
   src_lob  IN    BLOB)
RETURN BLOB;
PKG_UTIL.LOB_APPEND_HUGE(
   dest_lob INOUT CLOB,
```



```
  src_lob  IN    CLOB)
RETURN CLOB;
```

表 10-29 PKG_UTIL.LOB_APPEND_HUGE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
dest_lob	BLOB/CLOB	INOUT	否	写入的目标BLOB/CLOB对象。
src_lob	BLOB/CLOB	IN	否	被写入的源BLOB/CLOB对象。

- PKG_UTIL.READ_BFILE_TO_BLOB

  该函数READ_BFILE_TO_BLOB将源BFILE文件读取成目标BLOB对象, 并返回目标对象。

  PKG_UTIL.READ_BFILE_TO_BLOB函数原型为：

```
PKG_UTIL.READ_BFILE_TO_BLOB(
   fd IN INTEGER
)RETURN BLOB;
```

表 10-30 PKG_UTIL.READ_BFILE_TO_BLOB 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
fd	INTEGER	IN	否	读取的源BFILE文件。

- PKG_UTIL.LOB_COPY_HUGE

  该函数LOB_COPY_HUGE将源blob/clob对象，从指定偏移读取指定长度内容，写入到目标blob/clob对象的指定偏移位置, 并返回目标对象。

  PKG_UTIL.LOB_COPY_HUGE函数原型为：

```
PKG_UTIL.LOB_COPY_HUGE(
   lob_obj    INOUT BLOB,
   source_obj  IN    BLOB,
   amount      IN    BIGINT,
   dest_offset IN    BIGINT DEFAULT 1,
   src_offset  IN    BIGINT DEFAULT 1
)RETURN BLOB;

PKG_UTIL.LOB_COPY_HUGE(
   lob_obj    INOUT CLOB,
   source_obj  IN    CLOB,
   amount      IN    BIGINT,
   dest_offset IN    BIGINT DEFAULT 1,
   src_offset  IN    BIGINT DEFAULT 1
)RETURN CLOB;
```


表 10-31 PKG_UTIL.LOB_COPY_HUGE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
lob_obj	BLOB/CLOB	INOUT	否	目标BLOB/CLOB对象。
source_obj	BLOB/CLOB	IN	否	源BLOB/CLOB对象。
amount	BIGINT	IN	否	拷贝的长度（BLOB以字节为单位，CLOB以字符为单位）。
dest_offset	BIGINT	IN	否	目标LOB的载入偏移位置。
src_offset	BIGINT	IN	否	源LOB的读取偏移位置。

- PKG_UTIL.BLOB_RESET

  该函数将BLOB中一段数据set为value值，返回处理后的BLOB以及实际处理的长度。

  PKG_UTIL.BLOB_RESET函数原型为：

```
PKG_UTIL.BLOB_RESET(
   lob       INOUT BLOB,
   len       INOUT BIGINT,
   start_pos IN    BIGINT DEFAULT 1,
   value     IN    INTEGER DEFAULT 0
)RETURN RECORD;
```

表 10-32 PKG_UTIL.BLOB_RESET 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
lob	BLOB	INOUT	否	待重置的LOB。
len	INTEGER	INOUT	否	重置的长度，单位字节。
start	INTEGER	IN	否	重置的起始位置。

参数	类型	入参/出参	是否可以为空	描述
value	INTEGER	IN	是	设置的字符。默认值‘0’。

- PKG_UTIL.CLOB_RESET

  该函数将一段数据set为空格。

  PKG_UTIL.CLOB_RESET函数原型为：

```
PKG_UTIL.CLOB_RESET(
   lob       INOUT CLOB,
   len       INOUT BIGINT,
   start_pos IN    BIGINT DEFAULT 1
)RETURN RECORD;
```

  表 10-33 PKG_UTIL.CLOB_RESET 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
lob	CLOB	INOUT	否	待重置的LOB。
len	INTEGER	INOUT	否	重置的长度，单位字符。
start	INTEGER	IN	否	重置的起始位置，默认为1。

- PKG_UTIL.LOADBLOBFROMFILE

  该函数LOADBLOBFROMFILE将源BFILE对象，从指定偏移读取指定长度内容，写入到目标BLOB对象的指定偏移位置, 并返回目标对象，读取位置，写入位置。

  PKG_UTIL.LOADBLOBFROMFILE函数原型为：

```
PKG_UTIL.LOADBLOBFROMFILE(
   dest_lob     INOUT  BLOB,
   fd           IN     INTEGER,
   amount       IN     BIGINT,
   dest_offset  INOUT  BIGINT,
   file_offset  INOUT  BIGINT
)RETURN RECORD;
```


表 10-34 PKG_UTIL.LOADBLOBFROMFILE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
dest_lob	BLOB	INOUT	否	IN参数为目标BLOB对象，OUT参数为写入后的目标BLOB对象。
fd	INTEGER	IN	否	源BFILE对象的文件描述符。
amount	BIGINT	IN	否	拷贝的长度（BLOB以字节为单位，CLOB以字符为单位）。
dest_offset	BIGINT	INOUT	否	IN参数为目标LOB的写入偏移位置，OUT参数为实际写入位置。
src_offset	BIGINT	INOUT	否	IN参数为源BFILE的读取偏移位置，OUT参数为实际读取位置。

- PKG_UTIL.LOADCLOBFROMFILE

  该函数LOADCLOBFROMFILE将源BFILE对象，从指定偏移读取指定长度内容，写入到目标CLOB对象的指定偏移位置, 并返回目标对象，读取位置，写入位置。

  PKG_UTIL.LOADCLOBFROMFILE函数原型为：

```
PKG_UTIL.LOADCLOBFROMFILE(
    dest_lob      INOUT  CLOB,
    fd            IN     INTEGER,
    amount        IN     BIGINT,
    dest_offset   INOUT  BIGINT,
    file_offset   INOUT  BIGINT
)RETURN RECORD;
```

表 10-35 PKG_UTIL.LOADCLOBFROMFILE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
dest_lob	CLOB	INOUT	否	IN参数为目标CLOB对象，OUT参数为写入后的目标CLOB对象。
fd	INTEGER	IN	否	源BFILE对象的文件描述符。
amount	BIGINT	IN	否	拷贝的长度（CLOB以字符为单位）。
dest_offset	BIGINT	INOUT	否	IN参数为目标LOB的写入偏移位置，OUT参数为实际写入位置。
src_offset	BIGINT	INOUT	否	IN参数为源BFILE的读取偏移位置，OUT参数为实际读取位置。

- PKG_UTIL.LOB_CONVERTTOBLOB_HUGE

  将src_clob从指定偏移位置读取指定长度内容转成BLOB，并写入dest_lob的指定位置，amount为要转换的长度。

  PKG_UTIL.LOB_CONVERTTOBLOB_HUGE函数原型为：

```
PKG_UTIL.LOB_CONVERTTOBLOB_HUGE(
   dest_lob    INOUT  BLOB,
   src_clob    IN     CLOB,
   amount      IN     BIGINT,
   dest_offset INOUT  BIGINT,
   src_offset  INOUT  BIGINT)
)RETURN RECORD;
```

  表 10-36 PKG_UTIL.LOB_CONVERTTOBLOB_HUGE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
dest_lob	BLOB	INOUT	否	目标lob。
src_clob	CLOB	IN	否	要转换的clob。
amount	BIGINT	IN	否	转换的长度，字符为单位。
dest_offset	BIGINT	INOUT	否	IN参数为目标lob的写入起始位置，OUT参数为实际的写入位置。
src_offset	BIGINT	INOUT	否	IN参数为源clob的读取起始位置，OUT参数为实际的读取起始位置。

- PKG_UTIL.LOB_CONVERTTOCLOB_HUGE

  将src_clob从指定偏移位置读取指定长度内容转成CLOB，并写入dest_lob的指定位置，amount为要转换的长度。

  PKG_UTIL.LOB_CONVERTTOCLOB_HUGE函数原型为：

```
PKG_UTIL.LOB_CONVERTTOCLOB_HUGE(
   dest_lob    INOUT  CLOB,
   src_blob    IN     BLOB,
   amount      IN     BIGINT,
   dest_offset INOUT  BIGINT,
   src_offset  INOUT  BIGINT)
)RETURN RECORD;
```


表 10-37 PKG_UTIL.LOB_CONVERTTOCLOB_HUGE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
dest_lob	CLOB	INOUT	否	目标lob。
src_blob	BLOB	IN	否	要转换的blob。
amount	BIGINT	IN	否	转换的长度，字节为单位。
dest_offset	BIGINT	INOUT	否	IN参数为目标lob的写入起始位置，OUT参数为实际的写入位置。
src_offset	BIGINT	INOUT	否	IN参数为源clob的读取起始位置，OUT参数为实际的读取起始位置。

- PKG_UTIL.BFILE_GET_LENGTH

  该函数获取并返回指定的BFILE文件的长度。

  PKG_UTIL.BFILE_GET_LENGTH函数原型为：

```
PKG_UTIL.BFILE_GET_LENGTH(
    fd INTEGER
)RETURN BIGINT;
```

表 10-38 PKG_UTIL.LOB_GET_LENGTH 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
fd	INTEGER	IN	否	指定的BFILE文件的文件描述符。

- PKG_UTIL.BFILE_OPEN

  该函数打开BFILE文件，返回文件描述符。

  PKG_UTIL.BFILE_OPEN函数原型为：


```
PKG_UTIL.BFILE_OPEN(
   file_name TEXT,
   open_mode TEXT)
RETURN INTEGER;
```

表 10-39 PKG_UTIL.BFILE_OPEN 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file_name	TEXT	IN	否	指定的BFILE文件的文件name。
open_mode	TEXT	IN	否	打开方式（只支持‘r’，read功能）。

- PKG_UTIL.BFILE_CLOSE

  该函数关闭打开的BFILE文件。

  PKG_UTIL.BFILE_CLOSE函数原型为：

```
PKG_UTIL.BFILE_CLOSE(
   fd INTEGER)
RETURN BOOL;
```

表 10-40 PKG_UTIL.BFILE_CLOSE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
fd	INTEGER	IN	否	指定的BFILE文件的文件描述符。

- PKG_UTIL.LOB_WRITE_HUGE

  将源对象从起始位置读取len长度内容，写入目标LOB对象的指定偏移位置，覆盖该位置原本内容, 并返回目标LOB对象。

  PKG_UTIL.LOB_WRITE_HUGE函数原型为：

```
PKG_UTIL.LOB_WRITE_HUGE(
   dest_lob  INOUT BLOB,
   len       IN INTEGER,
   start_pos IN BIGINT,
   src_lob   IN RAW
)RETURN BLOB;

PKG_UTIL.LOB_WRITE_HUGE(
   dest_lob  INOUT CLOB,
   len       IN INTEGER,
   start_pos IN BIGINT,
   src_lob   IN VARCHAR2
)RETURN CLOB;
```


表 10-41 PKG_UTIL.LOB_WRITE_HUGE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
dest_lob	BLOB/CLOB	INOUT	否	IN参数为待写入的目标LOB，OUT参数为写入内容后的LOB
len	INTEGER	IN	否	待写入的长度（BLOB以字节为单位，CLOB以字符为单位）
start_pos	BIGINT	IN	否	在dest_lob中写入的偏移位置
src_lob	RAW/VARCHAR2	IN	否	源LOB对象

- PKG_UTIL.IO_PRINT

  该函数IO_PRINT将一段字符串打印输出。

  PKG_UTIL.IO_PRINT函数原型为：

```
PKG_UTIL.IO_PRINT(
format       IN   TEXT,
is_one_line  IN   BOOLEAN
)
RETURN VOID;
```

表 10-42 PKG_UTIL.IO_PRINT 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
format	TEXT	IN	否	待打印输出的字符串。
is_one_line	BOOLEAN	IN	否	是否输出为一行。

- PKG_UTIL.RAW_GET_LENGTH

  该函数RAW_GET_LENGTH获取raw的长度。

  PKG_UTIL.RAW_GET_LENGTH函数原型为：

```
PKG_UTIL.RAW_GET_LENGTH(
value       IN   RAW
)
RETURN INTEGER;
```


表 10-43 PKG_UTIL.RAW_GET_LENGTH 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
value	RAW	IN	否	待获取长度的对象。

- PKG_UTIL.RAW_CAST_FROM_VARCHAR2

  该函数RAW_CAST_FROM_VARCHAR2将varchar2转化为raw。

  PKG_UTIL.RAW_CAST_FROM_VARCHAR2函数原型为：

```
PKG_UTIL.RAW_CAST_FROM_VARCHAR2(
str      IN   VARCHAR2
)
RETURN RAW;
```

表 10-44 PKG_UTIL.RAW_CAST_FROM_VARCHAR2 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
str	VARCHAR2	IN	否	需要转化的源数据。

- PKG_UTIL.RAW_CAST_FROM_BINARY_INTEGER

  该函数RAW_CAST_FROM_BINARY_INTEGER将BIGINT转化为RAW。

  PKG_UTIL.RAW_CAST_FROM_BINARY_INTEGER函数原型为：

```
PKG_UTIL.RAW_CAST_FROM_BINARY_INTEGER(
value      IN   BIGINT,
endianess   IN   INTEGER
)
RETURN RAW;
```

表 10-45 PKG_UTIL.RAW_CAST_FROM_BINARY_INTEGER 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
value	BIGINT	IN	否	需要转化的源数据。
endianess	INTEGER	IN	否	表示字典序的整型值，现支持1或2或3（1代表BIG_ENDIAN，2代表LITTLE_ENDIAN，3代表MACHINE_ENDIAN）。

- PKG_UTIL.RAW_CAST_TO_BINARY_INTEGER

  该函数RAW_CAST_TO_BINARY_INTEGER将RAW转化为BINARY_INTEGER。

PKG_UTIL.RAW_CAST_TO_BINARY_INTEGER函数原型为：

```
PKG_UTIL.RAW_CAST_TO_BINARY_INTEGER(
value       IN   RAW,
endianess   IN   INTEGER
)
RETURN INTEGER;
```

表 10-46 PKG_UTIL.RAW_CAST_TO_BINARY_INTEGER 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
value	RAW	IN	否	需要转化的源数据。
endianess	INTEGER	IN	否	表示字典序的整型值，现支持1或2或3（1代表BIG_ENDIAN，2代表LITTLE_ENDIAN，3代表MACHINE_ENDIAN）。

- PKG_UTIL.RANDOM_SET_SEED

  该函数RANDOM_SET_SEED设置随机数种子。

  PKG_UTIL.RANDOM_SET_SEED函数原型为：

  ```
  PKG_UTIL.RANDOM_SET_SEED(
  seed        IN   INT
  )
  RETURN INTEGER;
  ```

  表 10-47 PKG_UTIL.RANDOM_SET_SEED 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
seed	INT	IN	否	随机数种子。

- PKG_UTIL.RANDOM_GET_VALUE

  该函数RANDOM_GET_VALUE返回0~1区间的一个随机数，其有效数字为15位。

  PKG_UTIL.RANDOM_GET_VALUE函数原型为：

  ```
  PKG_UTIL.RANDOM_GET_VALUE(
  )
  RETURN NUMEIRC;
  ```

- PKG_UTIL.FILE_SET_DIRNAME

  设置当前操作的目录，基本上所有涉及单个目录的操作，都需要调用此方法先设置操作的目录。

  PKG_UTIL.FILE_SET_DIRNAME函数原型为：

  ```
  PKG_UTIL.FILE_SET_DIRNAME(
  dir  IN  TEXT
  )
  RETURN BOOL;
  ```


表 10-48 PKG_UTIL.FILE_SET_DIRNAME 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
dirname	TEXT	IN	否	文件的目录位置，这个字符串是一个目录对象名。 **说明** 文件目录的位置，需要添加到系统表**PG_DIRECTORY**中，如果传入的路径和**PG_DIRECTORY**中的路径不匹配，会报路径不存在的错误，下面的涉及location作为参数的函数也是同样的情况。

- PKG_UTIL.FILE_OPEN

  该函数用来打开一个文件，最多可以同时打开50个文件。并且该函数返回INTEGER类型的一个句柄。

  PKG_UTIL.FILE_OPEN函数原型为：

  ```
  PKG_UTIL.FILE_OPEN(
  file_name    IN  TEXT,
  open_mode    IN  INTEGER)
  ```

  表 10-49 PKG_UTIL.FILE_OPEN 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file_name	TEXT	IN	否	文件名，包含扩展（文件类型），不包括路径名。如果文件名中包含路径，在OPEN中会被忽略，在Unix系统中，文件名不能以/.结尾。
open_mode	INTEGER	IN	否	指定文件的打开模式，包含r：read text，w: write text和a: append text。 **说明** 对于写操作，会检测文件类型，如果写入elf文件，将会报错并退出。

- PKG_UTIL.FILE_SET_MAX_LINE_SIZE

  设置写入文件一行的最大长度。

  PKG_UTIL.FILE_SET_MAX_LINE_SIZE函数原型为：

  ```
  PKG_UTIL.FILE_SET_MAX_LINE_SIZE(
  max_line_size IN INTEGER)
  RETURN BOOL
  ```

表 10-50 PKG_UTIL.FILE_SET_MAX_LINE_SIZE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
max_line_size	INTEGER	IN	否	每行最大字符数，包含换行符（最小值是1，最大值是32767）。如果没有指定，会指定一个默认值1024。

- PKG_UTIL.FILE_IS_CLOSE

  检测一个文件句柄是否关闭。

  PKG_UTIL.FILE_IS_CLOSE函数原型为：

  ```
  PKG_UTIL.FILE_IS_CLOSE(
  file IN INTEGER
  )
  RETURN BOOL
  ```

  表 10-51 PKG_UTIL.FILE_IS_CLOSE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	一个打开的文件句柄。

- PKG_UTIL.FILE_READ

  根据指定的长度从一个打开的文件句柄中读取出数据。

  PKG_UTIL.FILE_READ函数原型为：

  ```
  PKG_UTIL.FILE_READ(
  file     IN   INTEGER,
  buffer   OUT  TEXT,
  len      IN   BIGINT DEFAULT 1024)
  ```


表 10-52 PKG_UTIL.FILE_READ 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	通过调用OPEN打开的文件句柄，文件必须以读的模式打开，否则会抛出INVALID_OPERATION的异常。
buffer	TEXT	IN	否	用于接收数据的BUFFER。
len	BIGINT	IN	是	从文件中读取的字节数。

- PKG_UTIL.FILE_READLINE

  根据指定的长度从一个打开的文件句柄中读取出一行数据。

  PKG_UTIL.FILE_READLINE函数原型为：

```
PKG_UTIL.FILE_READLINE(
file    IN  INTEGER,
buffer  OUT TEXT,
len     IN  INTEGER DEFAULT NULL)
```

表 10-53 PKG_UTIL.FILE_READLINE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	通过调用OPEN打开的文件句柄，文件必须以读的模式打开，否则会抛出INVALID_OPERATION的异常。
buffer	TEXT	IN	否	用于接收数据的BUFFER。
len	INTEGER	IN	是	从文件中读取的字节数，默认是NULL。如果是默认NULL，会使用max_line_size来指定大小。

- PKG_UTIL.FILE_WRITE

将BUFFER中指定的数据写入到文件中。

PKG_UTIL.FILE_WRITE函数原型为：

```
PKG_UTIL.FILE_WRITE(
file IN INTEGER,
buffer IN TEXT
)
RETURN BOOL;
```

表 10-54 PKG_UTIL.FILE_WRITE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	一个打开的文件句柄。
buffer	TEXT	IN	否	要写入文件的文本数据，BUFFER的最大值是32767个字节。如果没有指定值，默认是1024个字节，没有刷新到文件之前，一系列的PUT操作的BUFFER总和不能超过32767个字节。 **说明** 对于写操作，会检测文件类型，如果写入elf文件，将会报错并退出。

- PKG_UTIL.FILE_NEWLINE

  向一个打开的文件中写入一个行终结符。行终结符和平台相关。

  PKG_UTIL.FILE_NEWLINE函数原型为：

  ```
  PKG_UTIL.FILE_NEWLINE(
  file IN INTEGER
  )
  RETURN BOOL;
  ```

  表 10-55 PKG_UTIL.FILE_NEWLINE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	一个打开的文件句柄。

- PKG_UTIL.FILE_WRITELINE

向一个打开的文件中写入一行。

PKG_UTIL.FILE_WRITELINE函数原型为：

```
PKG_UTIL.FILE_WRITELINE(
file IN INTEGER,
buffer IN TEXT
)
RETURN BOOL;
```

表 10-56 PKG_UTIL.FILE_WRITELINE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	一个打开的文件句柄。
buffer	TEXT	IN	否	要写入的内容。

- PKG_UTIL.FILE_READ_RAW

  从一个打开的文件句柄中读取指定长度的二进制数据，返回读取的二进制数据，返回类型为raw。

  PKG_UTIL.FILE_READ_RAW函数原型为：

  ```
  PKG_UTIL.FILE_READ_RAW(
  file      IN INTEGER,
  length    IN INTEGER DEFAULT NULL
  )
  RETURN RAW;
  ```

  表 10-57 PKG_UTIL.FILE_READ_RAW 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	一个打开的文件句柄。
length	INTEGER	IN	是	要读取的长度，默认为NULL。默认情况下读取文件中所有数据，最大为1G。

- PKG_UTIL.FILE_WRITE_RAW

  向一个打开的文件中写入传入二进制对象RAW。插入成功返回true。

  PKG_UTIL.FILE_WRITE_RAW函数原型为：

```
PKG_UTIL.FILE_WRITE_RAW(
file IN INTEGER,
r    IN RAW
)
RETURN BOOL;
```

  表 10-58 PKG_UTIL.FILE_NEWLINE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	一个打开的文件句柄。
r	RAW	IN	是	准备传入文件的数据 **说明** 对于写操作，会检测文件类型，如果写入elf文件，将会报错并退出。

- PKG_UTIL.FILE_FLUSH

  一个文件句柄中的数据要写入到物理文件中，缓冲区中的数据必须要有一个行终结符。当文件必须在打开时读取，刷新非常有用。例如，调试信息可以刷新到文件中，以便立即读取。

  PKG_UTIL.FILE_FLUSH函数原型为：

```
PKG_UTIL.FILE_FLUSH（
file IN INTEGER
）
RETURN VOID;
```

  表 10-59 PKG_UTIL.FILE_FLUSH 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	一个打开的文件句柄。

- PKG_UTIL.FILE_CLOSE

  关闭一个打开的文件句柄。

  PKG_UTIL.FILE_CLOSE函数原型为：

  ```
  PKG_UTIL.FILE_CLOSE（
  file IN INTEGER
  ）
  RETURN BOOL;
  ```

  表 10-60 PKG_UTIL.FILE_CLOSE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	一个打开的文件句柄。

- PKG_UTIL.FILE_REMOVE

  删除一个磁盘文件，操作的时候需要有充分的权限。

  PKG_UTIL.FILE_REMOVE函数原型为：

  ```
  PKG_UTIL.FILE_REMOVE(
  file_name IN TEXT
  )
  RETURN VOID;
  ```

  表 10-61 PKG_UTIL.FILE_REMOVE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
filen_ame	TEXT	IN	否	要删除的文件名

- PKG_UTIL.FILE_RENAME

  对于磁盘上的文件进行重命名，类似Unix的mv。

  PKG_UTIL.FILE_RENAME函数原型为：

  ```
  PKG_UTIL.FILE_RENAME(
  src_dir IN TEXT,
  src_file_name IN TEXT,
  dest_dir IN TEXT,
  dest_file_name IN TEXT,
  overwrite BOOLEAN DEFAULT FALSE)
  ```


表 10-62 PKG_UTIL.FILE_RENAME 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
src_dir	TEXT	IN	是	源文件目录（大小写敏感）。 **说明** • 文件目录的位置，需要添加到系统表**PG_DIRECTORY**中，如果传入的路径和**PG_DIRECTORY**中的路径不匹配，会报路径不存在的错误。 • 在打开guc参数safe_data_path时，用户只能通过高级包操作safe_data_path指定文件路径下的文件。
src_file_name	TEXT	IN	是	源文件名。
dest_dir	TEXT	IN	是	目标文件目录（大小写敏感）。 **说明** • 文件目录的位置，需要添加到系统表**PG_DIRECTORY**中，如果传入的路径和**PG_DIRECTORY**中的路径不匹配，会报路径不存在的错误。 • 在打开guc参数safe_data_path时，用户只能通过高级包操作safe_data_path指定文件路径下的文件。
dest_file_name	TEXT	IN	是	目标文件名。
overwrite	BOOLEAN	IN	否	默认是false，如果目的目录下存在一个同名的文件，不会进行重写。

- PKG_UTIL.FILE_SIZE
  返回指定的文件大小。
  PKG_UTIL.FILE_SIZE函数原型为：

```
BIGINT PKG_UTIL.FILE_SIZE(
file_name IN TEXT
)RETURN BIGINT;
```

表 10-63 PKG_UTIL.FILE_SIZE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file_name	TEXT	IN	否	文件名

- PKG_UTIL.FILE_BLOCK_SIZE

  返回指定的文件含有的块数量。

  PKG_UTIL.FILE_BLOCK_SIZE函数原型为：

  ```
  BIGINT PKG_UTIL.FILE_BLOCK_SIZE(
  file_name IN TEXT
  )RETURN BIGINT
  ```

  表 10-64 PKG_UTIL.FILE_BLOCK_SIZE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file_name	TEXT	IN	否	文件名

- PKG_UTIL.FILE_EXISTS

  判断指定的文件是否存在。

  PKG_UTIL.FILE_EXISTS函数原型为：

  ```
  PKG_UTIL.FILE_EXISTS(
  file_name IN TEXT
  )
  RETURN BOOL;
  ```

  表 10-65 PKG_UTIL.FILE_EXISTS 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file_name	TEXT	IN	否	文件名

- PKG_UTIL.FILE_GETPOS
  返回文件的偏移量，单位字节。
  PKG_UTIL.FILE_GETPOS函数原型为：

```
PKG_UTIL.FILE_GETPOS(
file IN INTEGER
)
RETURN BIGINT;
```

表 10-66 PKG_UTIL.FILE_GETPOS 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	一个打开的文件句柄。

- PKG_UTIL.FILE_SEEK
  根据用户指定的字节数向前或者向后调整文件指针的位置。
  PKG_UTIL.FILE_SEEK函数原型为：

```
void PKG_UTIL.FILE_SEEK(
file IN INTEGER,
start IN BIGINT
)
RETURN VOID;
```

表 10-67 PKG_UTIL.FILE_SEEK 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	一个打开的文件句柄。
start	BIGINT	IN	否	文件偏移，字节。

- PKG_UTIL.FILE_CLOSE_ALL
  关闭一个会话中打开的所有的文件句柄。

PKG_UTIL.FILE_CLOSE_ALL函数原型为：

```
PKG_UTIL.FILE_CLOSE_ALL(
)
RETURN VOID;
```

表 10-68 PKG_UTIL.FILE_CLOSE_ALL 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
无	无	无	无	无

- PKG_UTIL.EXCEPTION_REPORT_ERROR
  返回一个异常。
  PKG_UTIL.EXCEPTION_REPORT_ERROR函数原型为：

```
PKG_UTIL.EXCEPTION_REPORT_ERROR(
code INTEGER，
log TEXT,
flag BOOLEAN DEFAULT FALSE
)
RETURN INTEGER;
```

表 10-69 PKG_UTIL.EXCEPTION_REPORT_ERROR 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
code	INTEGER	IN	否	返回异常所打印的错误码。
log	TEXT	IN	否	返回异常所打印的日志提示信息。
flag	BOOLEAN	IN	是	保留字段，默认为false。

- PKG_UTIL.APP_READ_CLIENT_INFO
  读取client_info信息。
  PKG_UTIL.APP_READ_CLIENT_INFO函数原型为：


```
PKG_UTIL.APP_READ_CLIENT_INFO(
OUT buffer TEXT
)RETURN TEXT;
```

表 10-70 PKG_UTIL.APP_READ_CLIENT_INFO 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
buffer	TEXT	IN	否	返回的client_info信息。

- PKG_UTIL.APP_SET_CLIENT_INFO
  设置client_info信息。
  PKG_UTIL.APP_SET_CLIENT_INFO函数原型为：

```
PKG_UTIL.APP_SET_CLIENT_INFO(
str TEXT
)
```

表 10-71 PKG_UTIL.APP_SET_CLIENT_INFO 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
str	TEXT	IN	否	要设置的client_info信息。

- PKG_UTIL.LOB_CONVERTTOBLOB
  将clob转成blob，amount为要转换的长度。
  PKG_UTIL.LOB_CONVERTTOBLOB函数原型为：

```
PKG_UTIL.LOB_CONVERTTOBLOB(
dest_lob BLOB,
src_clob CLOB,
amount INTEGER,
dest_offset INTEGER,
src_offset INTEGER
)RETURN RAW;
```


表 10-72 PKG_UTIL.LOB_CONVERTTOBLOB 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
dest_lob	BLOB	IN	否	目标lob。
src_clob	CLOB	IN	否	要转换的clob。
amount	INTEGER	IN	否	转换的长度。
dest_offset	INTEGER	IN	否	目标lob的起始位置。
src_offset	INTEGER	IN	否	源clob的起始位置。

- PKG_UTIL.LOB_CONVERTTOCLOB
  将blob转成clob，amount为要转换的长度。
  PKG_UTIL.LOB_CONVERTTOCLOB函数原型为：

```
PKG_UTIL.LOB_CONVERTTOCLOB(
dest_lob CLOB,
src_blob BLOB,
amount INTEGER,
dest_offset INTEGER,
src_offset INTEGER
)RETURN TEXT;
```


表 10-73 PKG_UTIL.LOB_CONVERTTOCLOB 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
dest_lob	CLOB	IN	否	目标lob。
src_blob	BLOB	IN	否	要转换的blob。
amount	INTEGER	IN	否	转换的长度。
dest_offset	INTEGER	IN	否	目标lob的起始位置。
src_offset	INTEGER	IN	否	源clob的起始位置。

- PKG_UTIL.LOB_RAWTOTEXT

  将RAW转成TEXT。

  PKG_UTIL.LOB_RAWTOTEXT函数原型为：

```
PKG_UTIL.LOB_RAWTOTEXT(
src_lob IN BLOB
)
RETURN TEXT
```

表 10-74 PKG_UTIL.LOB_RAWTOTEXT 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
src_lob	BLOB	IN	否	要转换的lob数据。

- PKG_UTIL.LOB_TEXTTORAW

  将text转成raw。

  PKG_UTIL.LOB_TEXTTORAW函数原型为：

  ```
  PKG_UTIL.lob_texttoraw(
  src_lob CLOB
  )
  RETURN RAW;
  ```

  表 10-75 PKG_UTIL.LOB_TEXTTORAW 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
src_lob	CLOB	IN	否	要转换的lob数据。

- PKG_UTIL.MATCH_EDIT_DISTANCE_SIMILARITY

  计算两个字符串的差别。

  PKG_UTIL.MATCH_EDIT_DISTANCE_SIMILARITY函数原型为：

  ```
  PKG_UTIL.MATCH_EDIT_DISTANCE_SIMILARITY(
  str1 TEXT,
  str2 TEXT
  )
  RETURN INTEGER;
  ```

  表 10-76 PKG_UTIL.MATCH_EDIT_DISTANCE_SIMILARITY 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
str1	TEXT	IN	否	第一个字符串。
str2	TEXT	IN	否	第二个字符串。

- PKG_UTIL.RAW_CAST_TO_VARCHAR2

  raw类型转成varchar2。

  PKG_UTIL.RAW_CAST_TO_VARCHAR2函数原型为：

  ```
  PKG_UTIL.RAW_CAST_TO_VARCHAR2(
  str RAW
  )
  RETURN VARCHAR2;
  ```


表 10-77 PKG_UTIL.RAW_CAST_TO_VARCHAR2 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
str	RAW	IN	否	十六进制字符串

- PKG_UTIL.SESSION_CLEAR_CONTEXT

  清除session_context信息。

  PKG_UTIL.SESSION_CLEAR_CONTEXT函数原型为：

```
PKG_UTIL.SESSION_CLEAR_CONTEXT(
namespace TEXT,
client_identifier TEXT,
attribute TEXT
)
RETURN INTEGER;
```

表 10-78 PKG_UTIL.SESSION_CLEAR_CONTEXT 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
namespace	TEXT	IN	否	属性的命名空间。
client_identifier	TEXT	IN	否	client_identifier，一般与namespace相同即可，当为null时，默认修改所有namespace。
attribute	TEXT	IN	否	要清除的属性值。

- PKG_UTIL.SESSION_SEARCH_CONTEXT

  查找属性值。

  PKG_UTIL.SESSION_SEARCH_CONTEXT函数原型为：

```
PKG_UTIL.SESSION_SEARCH_CONTEXT(
namespace TEXT,
attribute TEXT
)
RETURN INTEGER;
```


表 10-79 PKG_UTIL.SESSION_SEARCH_CONTEXT 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
namespace	TEXT	IN	否	属性的命名空间。
attribute	TEXT	IN	否	要查找的属性值。

- PKG_UTIL.SESSION_SET_CONTEXT
  设置属性值。
  PKG_UTIL.SESSION_SET_CONTEXT函数原型为：

```
PKG_UTIL.SESSION_SET_CONTEXT(
namespace TEXT,
attribute TEXT,
value TEXT
)
RETURN INTEGER;
```

表 10-80 PKG_UTIL.SESSION_SET_CONTEXT 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
namespace	TEXT	IN	否	属性的命名空间
attribute	TEXT	IN	否	要设置的属性
value	TEXT	IN	否	属性对应的值

- PKG_UTIL.UTILITY_GET_TIME
  打印unix时间戳。
  PKG_UTIL.UTILITY_GET_TIME函数原型为：

```
PKG_UTIL.UTILITY_GET_TIME()
RETURN BIGINT;
```

- PKG_UTIL.UTILITY_FORMAT_ERROR_BACKTRACE
  查看存储过程的错误堆栈。
  PKG_UTIL.UTILITY_FORMAT_ERROR_BACKTRACE函数原型为：

```
PKG_UTIL.UTILITY_FORMAT_ERROR_BACKTRACE()
RETURN TEXT;
```


- PKG_UTIL.UTILITY_FORMAT_ERROR_STACK

  查看存储过程的报错信息。

  PKG_UTIL.UTILITY_FORMAT_ERROR_STACK函数原型为：

  ```
  PKG_UTIL.UTILITY_FORMAT_ERROR_STACK()
  RETURN TEXT;
  ```

- PKG_UTIL.UTILITY_COMPILE_SCHEMA

  重编译指定schema中的包函数和存储过程。

  PKG_UTIL.UTILITY_COMPILE_SCHEMA函数原型为：

  ```
  pkg_util.UTILITY_COMPILE_SCHEMA(
  schema IN VARCHAR2,
  compile_all IN BOOLEAN DEFAULT TRUE,
  reuse_settings IN BOOLEAN DEFAULT FALSE
  )
  returns VOID;
  ```

- PKG_UTIL.GS_COMPILE_SCHEMA

  重编译指定schema中的包函数和存储过程。

  PKG_UTIL.GS_COMPILE_SCHEMA存储过程原型为：

  ```
  PKG_UTIL.GS_COMPILE_SCHEMA(
  schema_name IN VARCHAR2 DEFAULT NULL,
  compile_all IN BOOLEAN DEFAULT FALSE,
  retry_times IN INT DEFAULT 10
  )
  ```

  表 10-81 PKG_UTIL.GS_COMPILE_SCHEMA 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
schema_name	VARCHAR2	IN	是	命名空间的名称。
compile_all	BOOLEAN	IN	是	编译所有。 ● false：编译pg_object表中状态为false的包、函数、存储过程。 ● true：编译pg_object表中所有的包、函数、存储过程。
retry_times	INT	IN	是	重试次数。

- PKG_UTIL.UTILITY_FORMAT_CALL_STACK

  查看存储过程调用堆栈。

  PKG_UTIL.UTILITY_FORMAT_CALL_STACK函数原型为：


```
    PKG_UTIL.UTILITY_FORMAT_CALL_STACK()
    RETURN TEXT;
```

示例

```
-- 删除schema
drop schema if exists pkg_var_test cascade;
-- 创建pkg_var_test
create schema pkg_var_test;
-- 设置schema和参数
set current_schema = pkg_var_test;
set behavior_compat_options ='plpgsql_dependency';
-- 创建包
create or replace package test_pkg as
    referenced_var int;
    unreferenced_var int;
end test_pkg;
/
-- 创建函数
create or replace function test_func return int
is
begin
  return 1;
end;
/
-- 创建存储过程
create or replace procedure test_proc
is
  proc_var int;
begin
  proc_var := 1;
end;
/
-- 重编译pkg_var_test下的包 函数和存储过程
call pkg_util.utility_compile_schema('pkg_var_test');
或者
call pkg_util.gs_compile_schema(' ',false,2);
-- 删掉pkg_var_test
drop schema if exists pkg_var_test cascade;
```

10.12.1.3 DBE_XML

DBE_XML支持的所有接口请参见**表10-82**：

表 10-82 DBE_XML 接口参数说明

接口名称	描述
DBE_XML.XML_FREE_PARSER	释放PARSER。
DBE_XML.XML_PARSER_GET_DOC	获取解析的document节点。
DBE_XML.XML_GET_VALIDATION_MODE	获取validate属性。
DBE_XML.XML_SET_VALIDATION_MODE	新建PARSER实例。
DBE_XML.XML_PARSE_BUFFER	解析VARCHAR字符串。

接口名称	描述
DBE_XML.XML_PARSE_CLOB	解析CLOB字符串。
DBE_XML.XML_SET_VALIDATION_MODE	设置validate属性。
DBE_XML.XML_DOM_APPEND_CHILD	将newchild node添加到parent(n)节点最后面,并返回新添加的Node节点。
DBE_XML.XML_DOM_CREATE_ELEMENT	返回创建指定名称的DOMELEMENT对象。
DBE_XML.XML_DOM_CREATE_ELEMENT_NS	返回创建指定名称和命名空间的DOMELEMENT对象。
DBE_XML.XML_DOM_CREATE_TEXT_NODE	创建并返回DOMTEXT对象。
DBE_XML.XML_DOM_FREE_DOCUMENT	将指定的xmldom类型对象释放。
DBE_XML.XML_DOM_FREE_ELEMENT	将指定的xmldom类型对象释放。
DBE_XML.XML_DOM_FREE_NODE	释放DOMNODE节点。
DBE_XML.XML_DOM_FREE_NODELIST	释放DOMNODELIST节点。
DBE_XML.XML_DOM_GET_ATTRIBUTE	获取指定的xmldom类型对象的属性。
DBE_XML.XML_DOM_GET_ATTRIBUTES	将DOMNode节点属性值作为map返回。
DBE_XML.XML_DOM_GET_CHILD_NODES	将节点下的若干子节点转换成节点列表。
DBE_XML.XML_DOM_GET_CHILDREN_BY_TAGNAME	获取指定的xmldom类型对象指定子节点组成的列表。
DBE_XML.XML_DOM_GET_CHILDREN_BY_TAGNAME_NS	获取指定的xmldom类型对象指定命名空间指定子节点组成的列表。
DBE_XML.XML_DOM_GET_DOCUMENT_ELEMENT	返回指定DOCUMENT的首个子节点。
DBE_XML.XML_DOM_GET_FIRST_CHILD	返回node节点的第一个子节点。
DBE_XML.XML_DOM_GET_LAST_CHILD	返回node节点的最后一个子节点。

接口名称	描述
DBE_XML.XML_DOM_GET_LENGTH	根据类型节点中内容返回节点数。
DBE_XML.XML_DOM_GET_LOCALNAME	返回给定对象的本地名称。
DBE_XML.XML_DOM_GET_NAMED_ITEM	检索由名称指定的节点。
DBE_XML.XML_DOM_GET_NAMED_ITEM_NS	检索由名称和命名空间指定的节点。
DBE_XML.XML_DOM_GET_NEXT_SIBLING	返回该节点的下一个节点。
DBE_XML.XML_DOM_GET_NODE_NAME	返回节点的名称。
DBE_XML.XML_DOM_GET_NODE_TYPE	返回节点的类型。
DBE_XML.XML_DOM_GET_NODE_VALUE	返回NODE节点的值。
DBE_XML.XML_DOM_GET_PARENT_NODE	返回给定NODE节点的父节点。
DBE_XML.XML_DOM_GET_TAGNAME	获取指定的xmldom类型对象的标签名。
DBE_XML.XML_DOM_HAS_CHILD_NODES	检查DOMNODE对象是否拥有任一子节点。
DBE_XML.XML_DOM_IMPORT_NODE	该函数将节点复制到另一节点中，并将复制后的节点挂载到指定document中。
DBE_XML.XML_DOM_IS_NULL	判断给定对象是否为NULL。
DBE_XML.XML_DOM_ITEM	根据索引返回list或map中与索引对应的元素。
DBE_XML.XML_DOM_MAKE_ELEMENT	返回转换后的DOMELEMENT对象。
DBE_XML.XML_DOM_MAKENODE	将给定对象强制转换为DOMNODE类型。
DBE_XML.XML_DOM_NEW_DOM_DOCUMENT_EMPTY	返回新的DOMDOCUMENT对象。
XML_DOM_NEW_DOM_DOCUMENT_CLOB	返回从指定的CLOB类型创建的新DOMDOCUMENT实例对象。
DBE_XML.XML_DOM_NEW_DOCUMENT_XMLTYPE	返回从指定的XMLType类型创建的新DOMDOCUMENT实例对象。

接口名称	描述
DBE_XML.XML_DOM_SET_ATTRIBUTE	设置指定的xmldom类型对象的属性。
DBE_XML.XML_DOM_SET_CHARSET	设置DOM设置DOMDOCUMENT的CHATSET字符集。
DBE_XML.XML_DOM_SET_DOCTYPE	设置DOMDOCUMENT的外部DTD。
DBE_XML.XML_DOM_SET_NODE_VALUE	此函数用于向DOMNODE对象中设置节点的值。
DBE_XML.XML_DOM_WRITE_TO_BUFFER_DOC	将给定的DOMDOCUMENT类型对象写入缓冲区。
DBE_XML.XML_DOM_WRITE_TO_BUFFER_NODE	将给定的DOMNODE类型对象写入缓冲区。
DBE_XML.XML_DOM_WRITE_TO_CLOB_DOC	将给定的DOMDOCUMENT类型对象写入Clob。
DBE_XML.XML_DOM_WRITE_TO_CLOB_NODE	将给定的DOMNODE类型对象写入Clob。
DBE_XML.XML_DOM_WRITE_TO_FILE_DOC	使用数据库字符集将XML节点写入指定文件。
DBE_XML.XML_DOM_WRITE_TO_FILE_NODE	使用数据库字符集将XML节点写入指定文件。
DBE_XML.XML_DOM_GET_SESSION_TREE_NUM	显示当前session中所有类型的dom树的数量。
DBE_XML.XML_DOM_GET_DOC_TREES_INFO	显示document类型的dom树的内存占用、节点数量等统计信息。
DBE_XML.XML_DOM_GET_DETAIL_DOC_TREE_INFO	显示特定的document变量的各类型节点数量。

- DBE_XML.XML_FREE_PARSER

  释放给定的PARSER对象。

  DBE_XML.XML_FREE_PARSER的存储过程原型为：

  ```
  DBE_XML.XML_FREE_PARSER(
  id IN RAW(13))
  RETURNS VOID;
  ```

  表 10-83 dbe_xml.xml_free_parser 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	是	指定的parser类型对象。

- DBE_XML.XML_PARSER_GET_DOC

  XML_PARSER_GET_DOC返回PARSER构建的DOM树文档的根节点。

  DBE_XML.XML_PARSER_GET_DOC的函数原型为：

  ```
  DBE_XML.XML_PARSER_GET_DOC(
  id IN RAW(13))
  RETURNS RAW(13);
  ```

  表 10-84 DBE_XML.XML_PARSER_GET_DOC 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	是	指定的parser类型对象。

  说明

  - DBE_XML.XML_PARSER_GET_DOC函数传空，返回null。
  - DBE_XML.XML_PARSER_GET_DOC函数传入的parser还没有解析文档，返回null。

- DBE_XML.XML_GET_VALIDATION_MODE

  获取给定Parser的解析验证模式。如果DTD验证开启返回TRUE，否则返回FALSE。

  DBE_XML.XML_GET_VALIDATION_MODE的函数原型为：

  ```
  DBE_XML.XML_GET_VALIDATION_MODE(
  id RAW(13))
  RETURNS BOOL;
  ```

  表 10-85 DBE_XML.XML_GET_VALIDATION_MODE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	是	指定的parser类型对象。

- DBE_XML.XML_NEW_PARSER

  新建Parser对象，返回一个新的解析器实例。

  DBE_XML.XML_NEW_PARSER的函数原型为：

  ```
  DBE_XML.XML_NEW_PARSER()
  RETURNS RAW(13);
  ```

- DBE_XML.XML_PARSE_BUFFER

  XML_PARSE_BUFFER解析存储在字符串中的XML文档。

  DBE_XML.XML_PARSE_BUFFER的存储过程原型为：

  ```
  DBE_XML.XML_PARSE_BUFFER(
  id RAW(13),
  xmlstr VARCHAR2)
  RETURNS VOID;
  ```


表 10-86 DBE_XML.XML_PARSE_BUFFER 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	是	指定的parser类型对象。
xmlstr	VARCHAR2	IN	否	存储XML文档的字符串。

说明

- XML_PARSE_BUFFER函数能够解析的字符串最大长度为32767，超过最大长度解析报错。
- 与A数据库差异：字符串encoding只支持UTF-8；version字段只支持1.0，1.0-1.9解析警告但正常执行，1.9以上报错。
- 与A数据库DTD校验差异：
  - !ATTLIST to type (CHECK|check|Check) "Ch..."将报错，因默认值"Ch..."不属于括号中枚举值，而A数据库不报错。
  - <!ENTITY baidu "www.baidu.com">..... &Baidu;&writer将报错，因区分字母大小写，Baidu无法与baidu对应，而A数据库不报错。
- 与A数据库命名空间校验差异：解析未声明的命名空间标签正常执行，而A数据库会报错。
- 与A数据库xml预定义实体解析差异：'"会被解析转译为字符’”，而A数据库中预定义实体统一都没有转译为字符。

- DBE_XML.XML_PARSE_CLOB

  XML_PARSE_CLOB解析存储在Clob中的XML文档。

  DBE_XML.XML_PARSE_CLOB的存储过程原型为：

```
DBE_XML.XML_PARSE_CLOB(
id IN RAW(13),
doc IN CLOB)
RETURNS VOID;
```

表 10-87 DBE_XML.XML_PARSE_CLOB 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	是	指定的parser类型对象。
doc	CLOB	IN	否	存储XML文档的字符串。

说明

- XML_PARSE_CLOB不支持解析大于等于2GB的clob。
- 与A数据库差异：字符串encoding只支持UTF-8；version字段只支持1.0，1.0-1.9解析警告但正常执行，1.9以上报错。
- 与A数据库DTD校验差异：
  - !ATTLIST to type (CHECK|check|Check) "Ch..."将报错，因默认值"Ch..."不属于括号中枚举值，而A数据库不报错。
  - <!ENTITY baidu "www.baidu.com">..... &Baidu;&writer将报错，因区分字母大小写，Baidu无法与baidu对应，而A数据库不报错。
- 与A数据库命名空间校验差异：解析未声明的命名空间标签正常执行，而A数据库会报错。
- 与A数据库xml预定义实体解析差异：'"会被解析转译为字符’”，而A数据库中预定义实体统一都没有转译为字符。

- DBE_XML.XML_SET_VALIDATION_MODE

  设置给定Parser的解析验证模式。

  DBE_XML.XML_SET_VALIDATION_MODE的存储过程原型为：

```
DBE_XML.XML_SET_VALIDATION_MODE(
id RAW(13),
validate BOOLEAN)
RETURNS VOID;
```

表 10-88 DBE_XML.XML_SET_VALIDATION_MODE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	是	指定的parser类型对象
validate	BOOLEAN	IN	是	要设置的模式： ● TRUE：开启DTD验证。 ● FALSE：不开启验证。

说明

- XML_SET_VALIDATION_MODE函数validate传入为空，不改变parser的解析验证模式。
- parser初始化默认为开启DTD验证模式。

- DBE_XML.XML_DOM_APPEND_CHILD

  将newchild node添加到parent(n)节点最后面,并返回新添加的Node节点。

  DBE_XML.XML_DOM_APPEND_CHILD的存储过程原型为：

```
DBE_XML.XML_DOM_APPEND_CHILD(
   parentId IN RAW(13),
   childId IN RAW(13)
)
RETURNS RAW(13);
```


表 10-89 DBE_XML.XML_DOM_APPEND_CHILD 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
parentId	RAW(13)	IN	否	指定的xmldom类型对象。
childId	RAW(13)	IN	否	指定的xmldom类型对象。

- DBE_XML.XML_DOM_CREATE_ELEMENT

  返回创建指定名称的DOMELEMENT对象。

  DBE_XML.XML_DOM_CREATE_ELEMENT的函数原型为：

```
DBE_XML.XML_DOM_CREATE_ELEMENT(
    id IN RAW(13),
    tagname IN VARCHAR2
)
RETURNS RAW(13);
```

表 10-90 DBE_XML.XML_DOM_CREATE_ELEMENT 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象。
tagname	VARCHAR2	IN	否	新建的DOMELEMENT名称。

- DBE_XML.XML_DOM_CREATE_ELEMENT_NS

  返回创建指定名称和命名空间的DOMELEMENT对象。

  DBE_XML.XML_DOM_CREATE_ELEMENT_NS的函数原型为：

```
DBE_XML.XML_DOM_CREATE_ELEMENT_NS(
    id IN RAW(13),
    tagname IN VARCHAR2,
    ns IN VARCHAR2
)
RETURNS RAW(13);
```

表 10-91 DBE_XML.XML_DOM_CREATE_ELEMENT_NS 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象。
tagname	VARCHAR2	IN	否	新建的DOMELEMENT名称。
ns	VARCHAR2	IN	否	命名空间。

- DBE_XML.XML_DOM_CREATE_TEXT_NODE

  创建并返回DOMTEXT对象。

  DBE_XML.XML_DOM_CREATE_TEXT_NODE的函数原型为：

```
dbe_xml.xml_dom_create_text_node(
    id IN RAW(13),
    data IN VARCHAR2
)
RETURNS RAW(13);
```

  表 10-92 DBE_XML.XML_DOM_CREATE_TEXT_NODE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象。
data	VARCHAR2	IN	否	新建的DOMTEXT节点内容。

- DBE_XML.XML_DOM_FREE_DOCUMENT

  将指定的xmldom类型对象释放。

  DBE_XML.XML_DOM_FREE_DOCUMENT的存储过程原型为：

```
DBE_XML.XML_DOM_FREE_DOCUMENT(
    id RAW(13)
)
RETURNS VOID;
```

  表 10-93 DBE_XML.XML_DOM_FREE_DOCUMENT 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象。

- DBE_XML.XML_DOM_FREE_ELEMENT

  将指定的xmldom类型对象释放。

  DBE_XML.XML_DOM_FREE_ELEMENT的存储过程原型为：

```
DBE_XML.XML_DOM_FREE_ELEMENT (
    id RAW(13)
)
RETURNS VOID;
```

  表 10-94 dbe_xml.xml_dom_free_element 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象。

- DBE_XML.XML_DOM_FREE_NODE

  释放DOMNODE节点。

  DBE_XML.XML_DOM_FREE_NODE的函数原型为：

```
DBE_XML.XML_DOM_FREE_NODE (
    id RAW(13)
)
RETURNS VOID;
```

  表 10-95 DBE_XML.XML_DOM_FREE_NODE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象

- DBE_XML.XML_DOM_FREE_NODELIST

  释放DOMNODELIST节点。

  DBE_XML.XML_DOM_FREE_NODELIST的存储过程原型为：

```
DBE_XML.XML_DOM_FREE_NODELIST(
    id IN RAW(13)
)
RETURNS VOID;
```

  表 10-96 DBE_XML.XML_DOM_FREE_NODELIST 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象。

- DBE_XML.XML_DOM_GET_ATTRIBUTE

  获取指定的xmldom类型对象的属性。

  DBE_XML.XML_DOM_GET_ATTRIBUTE的存储过程原型为：

```
DBE_XML.XML_DOM_GET_ATTRIBUTE (
    docid   IN  RAW(13),
    name    IN  VARCHAR2
)
RETURNS VARCHAR2;
```

  表 10-97 DBE_XML.XML_DOM_GET_ATTRIBUTE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
docid	RAW(13)	IN	否	指定的xmldom类型对象。
name	VARCHAR2	IN	否	字符串。

- DBE_XML.XML_DOM_GET_ATTRIBUTES
  将DOMNode节点属性值作为map返回。
  DBE_XML.XML_DOM_GET_ATTRIBUTES的函数原型为：

```
DBE_XML.XML_DOM_GET_ATTRIBUTES (
    id RAW(13)
)
RETURNS RAW(13);
```

表 10-98 DBE_XML.XML_DOM_GET_ATTRIBUTES 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象。

- DBE_XML.XML_DOM_GET_CHILD_NODES
  将节点下的若干子节点转换成节点列表。
  DBE_XML.XML_DOM_GET_CHILD_NODES的函数原型为：

```
DBE_XML.XML_DOM_GET_CHILD_NODES(
    id IN RAW(13)
)
RETURNS RAW(13);
```

表 10-99 DBE_XML.XML_DOM_GET_CHILD_NODES 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象。

- DBE_XML.XML_DOM_GET_CHILDREN_BY_TAGNAME
  获取指定的xmldom类型对象指定子节点组成的列表。
  DBE_XML.XML_DOM_GET_CHILDREN_BY_TAGNAME的存储过程原型为：

```
DBE_XML.XML_DOM_GET_CHILDREN_BY_TAGNAME (
    docid   IN  RAW(13),
    name    IN  VARCHAR2
)
RETURNS RAW(13);
```

表 10-100 DBE_XML.XML_DOM_GET_CHILDREN_BY_TAGNAME 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
docid	RAW(13)	IN	否	指定的xmldom类型对象。
name	VARCHAR2	IN	否	字符串。

- DBE_XML.XML_DOM_GET_CHILDREN_BY_TAGNAME_NS
  获取指定的xmldom类型对象指定命名空间指定子节点组成的列表。
  DBE_XML.XML_DOM_GET_CHILDREN_BY_TAGNAME_NS的存储过程原型为：

```
DBE_XML.XML_DOM_GET_CHILDREN_BY_TAGNAME_NS (
   docid   IN  RAW(13),
   name    IN  VARCHAR2,
   ns      IN  VARCHAR2
)
RETURNS RAW(13);
```

  表 10-101 DBE_XML.XML_DOM_GET_CHILDREN_BY_TAGNAME_NS 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
docid	RAW(13)	IN	否	指定的xmldom类型对象。
name	VARCHAR2	IN	否	字符串。
ns	VARCHAR2	IN	是	字符串。

- DBE_XML.XML_DOM_GET_DOCUMENT_ELEMENT
  返回指定DOCUMENT的首个子节点。
  DBE_XML.XML_DOM_GET_DOCUMENT_ELEMENT的存储过程原型为：

```
DBE_XML.XML_DOM_GET_DOCUMENT_ELEMENT(
   id  RAW(13)
)
RETURNS RAW(13);
```

  表 10-102 DBE_XML.XML_DOM_GET_DOCUMENT_ELEMENT 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象。

- DBE_XML.XML_DOM_GET_FIRST_CHILD
  返回node节点的第一个子节点。
  DBE_XML.XML_DOM_GET_FIRST_CHILD的函数原型为：

```
dbe_xml.xml_dom_get_first_child(
   id IN RAW(13)
)
RETURNS RAW(13);
```


表 10-103 DBE_XML.XML_DOM_GET_FIRST_CHILD 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象。

- DBE_XML.XML_DOM_GET_LAST_CHILD
  返回node节点的最后一个子节点。
  DBE_XML.XML_DOM_GET_LAST_CHILD的函数原型为：

```
DBE_XML.XML_DOM_GET_LAST_CHILD(
    id IN RAW(13)
)
RETURNS RAW(13);
```

表 10-104 DBE_XML.XML_DOM_GET_LAST_CHILD 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象。

- DBE_XML.XML_DOM_GET_LENGTH
  根据类型节点中内容返回节点数。
  DBE_XML.XML_DOM_GET_LENGTH的存储过程原型为：

```
DBE_XML.XML_DOM_GET_LENGTH(
    id RAW(13)
)
RETURNS VOID;
```

表 10-105 DBE_XML.XML_DOM_GET_LENGTH 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	是	指定的xmldom类型对象。

- DBE_XML.XML_DOM_GET_LOCALNAME
  返回给定对象的本地名称。
  DBE_XML.XML_DOM_GET_LOCALNAME的存储过程原型为：

```
DBE_XML.XML_DOM_GET_LOCALNAME (
    id  RAW(13)
)
RETURNS VARCHAR2;
```


表 10-106 DBE_XML.XML_DOM_GET_LOCALNAME 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	是	指定的xmldom类型对象。

- DBE_XML.XML_DOM_GET_NAMED_ITEM

  检索由名称指定的节点。

  DBE_XML.XML_DOM_GET_NAMED_ITEM的函数原型为：

```
DBE_XML.XML_DOM_GET_NAMED_ITEM(
    id IN RAW(13),
    nodeName IN VARCHAR2
)
RETURNS RAW(13);
```

表 10-107 DBE_XML.XML_DOM_GET_NAMED_ITEM 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象。
nodeName	VARCHAR2	IN	否	要检索的元素的名称。

- DBE_XML.XML_DOM_GET_NAMED_ITEM_NS

  检索由名称和命名空间指定的节点。

  DBE_XML.XML_DOM_GET_NAMED_ITEM_NS的函数原型为：

```
DBE_XML.XML_DOM_GET_NAMED_ITEM_NS(
    id  RAW(13),
    nodeName IN VARCHAR2,
    ns IN VARCHAR2
)
RETURNS RAW(13);
```

表 10-108 DBE_XML.XML_DOM_GET_NAMED_ITEM_NS 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象。
nodeName	VARCHAR2	IN	否	要检索的元素的名称。
ns	VARCHAR2	IN	是	命名空间。

- DBE_XML.XML_DOM_GET_NEXT_SIBLING

  返回该节点的下一个节点。

DBE_XML.XML_DOM_GET_NEXT_SIBLING的函数原型为：

```
DBE_XML.XML_DOM_GET_NEXT_SIBLING(
    id IN RAW(13)
)
RETURNS RAW(13);
```

表 10-109 DBE_XML.XML_DOM_GET_NEXT_SIBLING 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象。

- DBE_XML.XML_DOM_GET_NODE_NAME

  返回节点的名称。

  DBE_XML.XML_DOM_GET_NODE_NAME的函数原型为：

  ```
  DBE_XML.XML_DOM_GET_NODE_NAME(
      id IN RAW(13)
  )
  RETURNS VARCHAR2;
  ```

  表 10-110 DBE_XML.XML_DOM_GET_NODE_NAME 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象。

- DBE_XML.XML_DOM_GET_NODE_TYPE

  返回节点的类型。

  DBE_XML.XML_DOM_GET_NODE_TYPE的函数原型为：

  ```
  DBE_XML.XML_DOM_GET_NODE_TYPE(
      id IN RAW(13)
  )
  RETURNS INTEGER;
  ```

  表 10-111 DBE_XML.XML_DOM_GET_NODE_TYPE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象。

- DBE_XML.XML_DOM_GET_NODE_VALUE

  返回NODE节点的值。

  DBE_XML.XML_DOM_GET_NODE_VALUE的存储过程原型为：

  ```
  DBE_XML.XML_DOM_GET_NODE_VALUE(
  id IN RAW(13))
  RETURNS VARCHAR2;
  ```


表 10-112 DBE_XML.XML_DOM_GET_NODE_VALUE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	是	指定的xmldom类型对象。

- DBE_XML.XML_DOM_GET_PARENT_NODE
  返回给定NODE节点的父节点。
  DBE_XML.XML_DOM_GET_PARENT_NODE的存储过程原型为：

```
DBE_XML.XML_DOM_GET_PARENT_NODE(
id IN RAW(13))
RETURNS RAW(13);
```

表 10-113 DBE_XML.XML_DOM_GET_PARENT_NODE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	是	指定的xmldom类型对象。

- DBE_XML.XML_DOM_GET_TAGNAME
  获取指定的xmldom类型对象的标签名。
  DBE_XML.XML_DOM_GET_TAGNAME的存储过程原型为：

```
DBE_XML.XML_DOM_GET_TAGNAME (
    docid RAW(13)
)
RETURNS VARCHAR2;
```

表 10-114 DBE_XML.XML_DOM_GET_TAGNAME 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
docid	RAW(13)	IN	是	指定的xmldom类型对象。

- DBE_XML.XML_DOM_HAS_CHILD_NODES
  检查DOMNODE对象是否拥有任一子节点。
  DBE_XML.XML_DOM_HAS_CHILD_NODES的存储过程原型为：

```
DBE_XML.XML_DOM_HAS_CHILD_NODES(
id IN RAW(13))
RETURNS BOOLEAN
```


表 10-115 DBE_XML.XML_DOM_HAS_CHILD_NODES 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	是	指定的xmldom类型对象。

- DBE_XML.XML_DOM_IMPORT_NODE

  该函数将节点复制到另一节点中，并将复制后的节点挂载到指定document中。若被复制节点的类型不属于xmldom的constants所规定的12种类型，则直接抛出类型不支持异常。

  DBE_XML.XML_DOM_IMPORT_NODE的函数原型为：

```
DBE_XML.XML_DOM_IMPORT_NODE(
    doc_id IN RAW(13),
    node_id IN RAW(13),
    deep IN BOOLEAN
)
RETURNS RAW(13);
```

表 10-116 DBE_XML.XML_DOM_IMPORT_NODE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
doc_id	RAW(13)	IN	否	节点挂载的文档。
node_id	RAW(13)	IN	否	将要导入的节点。
deep	BOOLEAN	IN	否	设置递归导入： ● 如果为TRUE，则导入该节点及其所有子节点。 ● 如果为FALSE，则指导入节点本身。

- DBE_XML.XML_DOM_IS_NULL

  判断给定对象是否为NULL，如果是则返回True，否则返回false。

  DBE_XML.XML_DOM_IS_NULL的函数原型为：

```
DBE_XML.XML_DOM_IS_NULL (
    id  RAW(13)
)
RETURNS boolean;
```

表 10-117 DBE_XML.XML_DOM_IS_NULL 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	是	指定的xmldom类型对象。

- DBE_XML.XML_DOM_ITEM
  根据索引返回list或map中与索引对应的元素。
  DBE_XML.XML_DOM_ITEM的函数原型为：

```
DBE_XML.XML_DOM_ITEM (
    id IN RAW(13),
    index IN INTEGER
)
RETURNS RAW(13);
```

表 10-118 DBE_XML.XML_DOM_ITEM 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象。
index	INTEGER	IN	否	要检索的元素的索引。

- DBE_XML.XML_DOM_MAKE_ELEMENT
  返回转换后的DOMELEMENT对象。
  DBE_XML.XML_DOM_MAKE_ELEMENT的存储过程原型为：

```
DBE_XML.XML_DOM_MAKE_ELEMENT(
id IN RAW(13))
RETURNS RAW(13)
```

表 10-119 DBE_XML.XML_DOM_MAKE_ELEMENT 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象。

- DBE_XML.XML_DOM_MAKENODE
  将给定对象强制转换为DOMNODE类型。
  DBE_XML.XML_DOM_MAKENODE的存储过程原型为：

```
DBE_XML.XML_DOM_MAKENODE(
  id    RAW(13)
)
RETURNS DOMNODE;
```

表 10-120 DBE_XML.XML_DOM_MAKENODE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	是	指定的xmldom类型对象。

- DBE_XML.XML_DOM_NEW_DOM_DOCUMENT_EMPTY
  返回新的DOMDOCUMENT对象。
  DBE_XML.XML_DOM_NEW_DOM_DOCUMENT_EMPTY的函数原型为：

```
DBE_XML.XML_DOM_NEW_DOM_DOCUMENT_EMPTY()
RETURNS RAW(13);
```

- XML_DOM_NEW_DOM_DOCUMENT_CLOB
  返回从指定的CLOB类型创建的新DOMDOCUMENT实例对象。
  XML_DOM_NEW_DOM_DOCUMENT_CLOB的函数原型为：

```
XML_DOM_NEW_DOM_DOCUMENT_CLOB(
   content IN  CLOB
)
RETURNS RAW(13);
```

表 10-121 XML_DOM_NEW_DOM_DOCUMENT_CLOB 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
content	CLOB	IN	否	指定的CLOB类型。

- DBE_XML.XML_DOM_NEW_DOCUMENT_XMLTYPE
  返回从指定的XMLType类型创建的新DOMDOCUMENT实例对象。
  DBE_XML.XML_DOM_NEW_DOCUMENT_XMLTYPE的函数原型为：

```
DBE_XML.XML_DOM_NEW_DOCUMENT_XMLTYPE(
   content IN  CLOB
)
RETURNS RAW(13);
```

表 10-122 DBE_XML.XML_DOM_NEW_DOCUMENT_XMLTYPE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
content	CLOB	IN	否	指定的CLOB类型。

- DBE_XML.XML_DOM_SET_ATTRIBUTE
  设置指定的xmldom类型对象的属性。
  DBE_XML.XML_DOM_SET_ATTRIBUTE的存储过程原型为：

```
DBE_XML.XML_DOM_SET_ATTRIBUTE(
   docid   IN  RAW(13),
   name    IN  VARCHAR2,
   value   IN  VARCHAR2
)
RETURNS VOID;
```


表 10-123 DBE_XML.XML_DOM_SET_ATTRIBUTE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象。
name	VARCHAR2	IN	否	字符串
value	VARCHAR2	IN	否	字符串

- DBE_XML.XML_DOM_SET_CHARSET

  设置DOM设置DOMDOCUMENT的CHATSET字符集。

  DBE_XML.XML_DOM_SET_CHARSET的函数原型为：

```
DBE_XML.XML_DOM_SET_CHARSET(
   id      IN RAW(13),
   charset IN VARCHAR2
)
RETURNS VOID;
```

表 10-124 DBE_XML.XML_DOM_SET_CHARSET 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象。
charset	VARCHAR2	IN	否	字符集。

- DBE_XML.XML_DOM_SET_DOCTYPE

  设置DOMDOCUMENT的外部DTD。

  DBE_XML.XML_DOM_SET_DOCTYPE的函数原型为：

```
DBE_XML.XML_DOM_SET_DOCTYPE(
   id          IN  RAW(13),
   dtd_name    IN  VARCHAR2,
   system_id   IN  VARCHAR2,
   public_id   IN  VARCHAR2
)
RETURNS void;
```

表 10-125 DBE_XML.XML_DOM_SET_DOCTYPE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象。
dtd_name	VARCHAR2	IN	否	需要初始化doctype的名称。

参数	类型	入参/出参	是否可以为空	描述
system_id	VARCHAR2	IN	否	需要初始化doctype的system ID。
public_id	VARCHAR2	IN	否	需要初始化doctype的public ID。

- DBE_XML.XML_DOM_SET_NODE_VALUE

  此函数用于向DOMNODE对象中设置节点的值。

  DBE_XML.XML_DOM_SET_NODE_VALUE的存储过程原型为：

  ```
  DBE_XML.XML_DOM_SET_NODE_VALUE(
  id IN RAW(13),
  node_value IN VARCHAR2)
  RETURNS VOID;
  ```

  表 10-126 DBE_XML.XML_DOM_SET_NODE_VALUE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象。
node_value	VARCHAR2	IN	否	向DOMNODE对象中设置的字符串。

- DBE_XML.XML_DOM_WRITE_TO_BUFFER_DOC

  将给定的DOMDOCUMENT类型对象写入缓冲区。

  DBE_XML.XML_DOM_WRITE_TO_BUFFER_DOC的存储过程原型为：

  ```
  DBE_XML.XML_DOM_WRITE_TO_BUFFER_DOC(
  id IN RAW(13))
  RETURNS VARCHAR2;
  ```

  表 10-127 DBE_XML.XML_DOM_WRITE_TO_BUFFER_DOC 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	是	指定的xmldom类型对象。

- DBE_XML.XML_DOM_WRITE_TO_BUFFER_NODE

  将给定的DOMNODE类型对象写入缓冲区。

  DBE_XML.XML_DOM_WRITE_TO_BUFFER_NODE的存储过程原型为：

  ```
  DBE_XML.XML_DOM_WRITE_TO_BUFFER_NODE(
  id IN RAW(13))
  RETURNS VARCHAR2;
  ```

表 10-128 DBE_XML.XML_DOM_WRITE_TO_BUFFER_NODE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	是	指定的xmldom类型对象。

- DBE_XML.XML_DOM_WRITE_TO_CLOB_DOC
  将给定的DOMDOCUMENT类型对象写入Clob。
  DBE_XML.XML_DOM_WRITE_TO_CLOB_DOC的存储过程原型为：

```
DBE_XML.XML_DOM_WRITE_TO_CLOB_DOC(
    id IN RAW(13)
)
RETURNS VARCHAR2;
```

表 10-129 DBE_XML.XML_DOM_WRITE_TO_CLOB_DOC 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	是	指定的xmldom类型对象。

- DBE_XML.XML_DOM_WRITE_TO_CLOB_NODE
  将给定的DOMNODE类型对象写入Clob。
  DBE_XML.XML_DOM_WRITE_TO_CLOB_NODE的存储过程原型为：

```
DBE_XML.XML_DOM_WRITE_TO_CLOB_NODE(
    id          IN  RAW(13)
)
RETURNS clob;
```

表 10-130 DBE_XML.XML_DOM_WRITE_TO_CLOB_NODE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	是	指定的xmldom类型对象。

- DBE_XML.XML_DOM_WRITE_TO_FILE_DOC
  使用数据库字符集将XML节点写入指定文件。
  DBE_XML.XML_DOM_WRITE_TO_FILE_DOC的存储过程原型为：

```
DBE_XML.XML_DOM_WRITE_TO_FILE_DOC(
id IN  RAW(13),
file_dir IN  VARCHAR2)
RETURNS VOID;

DBE_XML.XML_DOM_WRITE_TO_FILE_DOC(
id IN  RAW(13),
file_dir IN  VARCHAR2,
charset  IN  VARCHAR2)
RETURNS VOID PACKAGE
```


表 10-131 DBE_XML.XML_DOM_WRITE_TO_FILE_DOC 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	是	指定的xmldom类型对象。
file_dir	VARCHAR2	IN	否	要写入的文件。
charset	VARCHAR2	IN	否	指定字符集。

- DBE_XML.XML_DOM_WRITE_TO_FILE_NODE
  使用数据库字符集将XML节点写入指定文件。
  DBE_XML.XML_DOM_WRITE_TO_FILE_NODE的存储过程原型为：

```
DBE_XML.XML_DOM_WRITE_TO_FILE_NODE(
id IN  RAW(13),
filename IN  VARCHAR2)
RETURNS VOID;
```

表 10-132 DBE_XML.XML_DOM_WRITE_TO_FILE_NODE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象。
filename	VARCHAR2	IN	否	指定文件地址。

- DBE_XML.XML_DOM_GET_SESSION_TREE_NUM
  查询当前session中所有类型的dom树数量。
  DBE_XML.XML_DOM_GET_SESSION_TREE_NUM的函数原型为：

```
DBE_XML.XML_DOM_GET_SESSION_TREE_NUM()
RETURNS INTEGER;
```

- DBE_XML.XML_DOM_GET_DOC_TREES_INFO
  查询当前session中Document类型的dom树信息，如内存占用等。
  DBE_XML.XML_DOM_GET_DOC_TREES_INFO的函数原型为：

```
DBE_XML.XML_DOM_GET_DOC_TREES_INFO()
RETURNS VARCHAR2;
```

- DBE_XML.XML_DOM_GET_DETAIL_DOC_TREE_INFO
  查询传入的document内的各类型子节点的数量。
  DBE_XML.XML_DOM_GET_DETAIL_DOC_TREE_INFO的函数原型为：

```
DBE_XML.XML_DOM_GET_DETAIL_DOC_TREE_INFO(
id IN  RAW(13))
RETURNS VARCHAR2;
```

表 10-133 DBE_XML.XML_DOM_GET_DETAIL_DOC_TREE_INFO 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	RAW(13)	IN	否	指定的xmldom类型对象。

10.12.2 二次封装接口(推荐)

10.12.2.1 DBE_LOB

接口介绍

高级功能包DBE_LOB支持的所有接口参见**表10-134**。

说明

- A数据库中空格的实际字节内容为00，GaussDB中空格对应字节内容为ASCII码值(32)。
- 集中式环境中，clob、blob以及bfile最大支持32TB。
- LOBMAXSIZE最大支持1073741771字节。

表 10-134 DBE_LOB

接口名称	描述
DBE_LOB.GET_LENGTH	获取并返回指定的LOB类型对象的长度（不支持大于2GB）。
DBE_LOB.LOB_GET_LENGTH	获取并返回指定的LOB类型对象/BFILE文件的长度。
DBE_LOB.OPEN	打开一个LOB类型对象返回一个LOB的描述符。
DBE_LOB.READ	根据指定的长度及起始位置偏移读取LOB内容的一部分到BUFFER缓冲区。
DBE_LOB.LOB_READ	根据指定的长度及起始位置偏移读取LOB内容的一部分到BUFFER缓冲区（支持bfile读取）。
DBE_LOB.WRITE	根据指定长度及起始位置偏移将BUFFER中内容写入到LOB中。
DBE_LOB.WRITE_APPEND	根据指定长度将BUFFER中内容写入到LOB的尾部。
DBE_LOB.LOB_WRITE_APPEND	根据指定长度将BUFFER中内容写入到LOB的尾部。
DBE_LOB.COPY	根据指定长度及起始位置偏移将LOB内容写入到另一个LOB中。

接口名称	描述
DBE_LOB.LOB_COPY	根据指定长度及起始位置偏移将LOB内容写入到另一个LOB中。
DBE_LOB.ERASE	根据指定长度及起始位置偏移删除LOB中的内容（不支持大于1GB）。
DBE_LOB.LOB_ERASE	根据指定长度及起始位置偏移删除LOB中的内容。
DBE_LOB.CLOSE	关闭已经打开的LOB描述符。
DBE_LOB.MATCH	返回一个字符串在LOB中第N次出现的位置。
DBE_LOB.COMPARE	比较两个LOB或者两个LOB的某一部分（支持bfile比较）。
DBE_LOB.SUBSTR	用于读取一个LOB的子串，返回读取到的子串。
DBE_LOB.LOB_SUBSTR	用于读取一个LOB或者BFILE的子串，返回读取到的子串。
DBE_LOB.STRIP	用于截断指定长度的LOB，执行完会将LOB的长度设置为参数指定的长度。
DBE_LOB.LOB_STRIP	用于截断指定长度的LOB，执行完会将LOB的长度设置为参数指定的长度。
DBE_LOB.CREATE_TEMPORARY	创建一个临时的BLOB或者CLOB对象。
DBE_LOB.APPEND	将源LOB的内容拼接到目的LOB中。
DBE_LOB.LOB_APPEND	将源LOB的内容拼接到目的LOB中。
DBE_LOB.FREETEMPORARY	删除一个临时的BLOB或者CLOB对象。
DBE_LOB.FILEOPEN	打开一个数据库BFILE文件，并返回文件描述符。
DBE_LOB.FILECLOSE	关闭由FILEOPEN打开的BFILE文件。
DBE_LOB.BFILEOPEN	打开一个数据库BFILE文件。
DBE_LOB.BFILECLOSE	关闭一个由BFILEOPEN打开的BFILE文件。
DBE_LOB.LOADFROMFILE	读取指定位置和长度的数据库BFILE文件到指定位置的BLOB对象中。
DBE_LOB.LOADFROMBFILE	读取指定位置和长度的数据库BFILE文件到指定位置的LOB中。
DBE_LOB.LOADBLOBFROMFILE	读取指定位置和长度的数据库外部文件到指定位置的BLOB中（不支持大于1GB）。
DBE_LOB.LOADBLOBFROMBFIL...	读取指定位置和长度的数据库BFILE文件到指定位置的BLOB中。

接口名称	描述
DBE_LOB.LOADCLOBFROMFILE	读取指定位置和长度的数据库外部文件到指定位置的CLOB中（不支持大于1GB）。
DBE_LOB.LOADCLOBFROMBFIL...	读取指定位置和长度的数据库BFILE文件到指定位置的CLOB中。
DBE_LOB.CONVERTTOBLOB	将CLOB类型文件转换为BLOB类型文件（不支持大于1GB）。
DBE_LOB.CONVERTTOCLOB	将BLOB类型文件转换为CLOB类型文件（不支持大于1GB）。
DBE_LOB.LOB_CONVERTTOBLO...	将CLOB类型文件转换为BLOB类型文件。
DBE_LOB.LOB_CONVERTTOCLO...	将BLOB类型文件转换为CLOB类型文件。
DBE_LOB.GETCHUNKSIZE	获取数据库中CHUNK结构中用于存储LOB数据的最大SIZE。
DBE_LOB.LOB_WRITE	将源对象从起始位置读取指定长度内容，写入目标LOB对象的指定偏移位置，覆盖该位置原本内容, 并返回目标LOB对象。
DBE_LOB.BFILENAME	根据目录和文件名构造返回DBE_LOB.BFILE对象。

- DBE_LOB.GET_LENGTH

  函数GET_LENGTH获取并返回指定的LOB类型对象的长度，最大支持2GB。

  DBE_LOB.GET_LENGTH函数原型为：

  ```
  DBE_LOB.GET_LENGTH (
      blob_obj IN BLOB)
  RETURN INTEGER;

  DBE_LOB.GET_LENGTH (
      clob_obj IN CLOB)
  RETURN INTEGER;
  ```

  表 10-135 DBE_LOB.GET_LENGTH 接口参数说明

参数	描述
blob_obj/ clob_obj	待获取长度的BLOB/CLOB类型对象。

- DBE_LOB.LOB_GET_LENGTH

  函数LOB_GET_LENGTH获取并返回指定的LOB类型对象/BFILE文件的长度，最大支持32TB。

  DBE_LOB.LOB_GET_LENGTH函数原型为：

  ```
  DBE_LOB.LOB_GET_LENGTH (
      blob_obj IN BLOB)
  RETURN BIGINT;
  ```

```
DBE_LOB.LOB_GET_LENGTH (
   clob_obj IN CLOB)
RETURN BIGINT;

DBE_LOB.LOB_GET_LENGTH (
   bfile IN DBE_LOB.BFILE)
RETURN BIGINT;
```

表 10-136 DBE_LOB.LOB_GET_LENGTH 接口参数说明

参数	描述
blob_obj/ clob_obj/bfile	待获取长度的BLOB/CLOB/BFILE类型对象。

- DBE_LOB.OPEN

  存储过程打开一个LOB，并返回一个LOB描述符，该过程无实际意义，仅用于兼容。

  DBE_LOB.OPEN函数原型为：

```
DBE_LOB.OPEN (
   lob INOUT BLOB);

DBE_LOB.OPEN (
   lob INOUT CLOB);

DBE_LOB.OPEN (
   bfile     INOUT DBE_LOB.BFILE,
   open_mode IN    TEXT DEFAULT 'null');
```

表 10-137 DBE_LOB.OPEN 接口参数说明

参数	描述
lob/bfile	被打开的BLOB或者CLOB对象或者bfile文件。
open_mode	操作模式，现支持[R,W,A,RB,WB,AB]。

- DBE_LOB.READ

  存储过程READ根据指定长度及起始位置偏移读取LOB内容的一部分到out_put缓冲区。

  DBE_LOB.READ函数原型为：

```
DBE_LOB.READ (
   blob_obj IN  BLOB,
   amount   IN  INTEGER,
   off_set  IN  INTEGER,
   out_put  OUT RAW);

DBE_LOB.READ (
   clob_obj IN  CLOB,
   amount   IN  INTEGER,
   off_set  IN  INTEGER,
   out_put  OUT VARCHAR2);
```


表 10-138 DBE_LOB.READ 接口参数说明

参数	说明
blob_obj/ clob_obj	待读入的BLOB/CLOB类型对象。
amount	读入长度。 **说明** 如果读入长度小于1，或大于32767，则报错。
off_set	指定从参数lob的哪个位置开始读取的偏移（即相对lob内容起始位置的字节数）。如果偏移量小于1或者大于lob长度，则报错。初始位置为1。
out_put	读取参数lob内容后存放的目标缓冲区。

- DBE_LOB.LOB_READ

  存储过程LOB_READ根据指定长度及起始位置偏移读取LOB/BFILE内容的一部分到out_put缓冲区。

  DBE_LOB.LOB_READ函数原型为：

```
DBE_LOB.LOB_READ(
   blob_obj IN    BLOB,
   amount   INOUT BIGINT,
   off_set  IN    BIGINT,
   out_put  OUT   RAW);

DBE_LOB.LOB_READ(
   clob_obj IN    CLOB,
   amount   INOUT BIGINT,
   off_set  IN    BIGINT,
   out_put  OUT   VARCHAR2);

DBE_LOB.LOB_READ(
   bfile   IN    DBE_LOB.BFILE,
   amount  INOUT BIGINT,
   off_set IN    BIGINT,
   out_put OUT   RAW);
```

表 10-139 DBE_LOB.LOB_READ 接口参数说明

参数	说明
blob_obj/ clob_obj/bfile	待读入的BLOB/CLOB/BFILE类型对象（支持大于1GB）。
amount	IN参数为读入长度，OUT参数为实际读取的长度。 **说明** 如果读入长度为小于1，或大于32767，则报错。
off_set	指定从参数lob的哪个位置开始读取的偏移（即相对lob内容起始位置的字节数）。如果偏移量小于1或者大于lob长度，则报错。初始位置为1。
out_put	读取参数lob内容后存放的目标缓冲区。

- DBE_LOB.WRITE

存储过程WRITE根据指定长度及起始位置将source中内容写入到LOB对象中。

DBE_LOB.WRITE函数原型为：

```
DBE_LOB.WRITE (
   blob_obj INOUT BLOB,
   amount   IN    INTEGER,
   off_set  IN    INTEGER,
   source   IN    RAW);

DBE_LOB.WRITE (
   clob_obj INOUT CLOB,
   amount   IN    INTEGER,
   off_set  IN    INTEGER,
   source   IN    VARCHAR2);
```

表 10-140 DBE_LOB.WRITE 接口参数说明

参数	说明
blob_obj/ clob_obj	待写入的BLOB/CLOB类型对象。
amount	写入长度，最大支持32767字符。 **说明** 如果写入长度小于1或写入长度大于待写入的内容长度，则报错。
off_set	指定从blob_obj/clob_obj的哪个位置开始写入的偏移（即相对LOB内容起始位置的字节数）。 **说明** 如果偏移量小于1或者大于LOBMAXSIZE时，则报错。初始位置是1，最大值为LOB类型最大长度。
source	待写入的内容。

- DBE_LOB.WRITE_APPEND

  存储过程WRITE_APPEND根据指定长度将source_obj中内容写入到LOB的尾部。

  DBE_LOB.WRITE_APPEND函数原型为：

```
DBE_LOB.WRITE_APPEND (
   blob_obj   INOUT BLOB,
   amount     IN    INTEGER,
   source_obj IN    RAW);

DBE_LOB.WRITE_APPEND (
   clob_obj   INOUT CLOB,
   amount     IN    INTEGER,
   source_obj IN    VARCHAR2);
```

表 10-141 DBE_LOB.WRITE_APPEND 接口参数说明

参数	说明
blob_obj/ clob_obj	待写入的指定BLOB/CLOB类型对象。
amount	写入长度，最大支持32767字符。 **说明** 如果写入长度小于1或写入长度大于待写入的内容长度，则报错。
source_obj	待写入的内容。

- DBE_LOB.LOB_WRITE_APPEND

  存储过程LOB_WRITE_APPEND根据指定长度将source_obj中内容写入到LOB的尾部。

  DBE_LOB.LOB_WRITE_APPEND函数原型为：

```
DBE_LOB.LOB_WRITE_APPEND(
   blob_obj    INOUT BLOB,
   amount      IN    INTEGER,
   source_obj  IN    RAW);

DBE_LOB.LOB_WRITE_APPEND (
   clob_obj    INOUT CLOB,
   amount      IN    INTEGER,
   source_obj  IN    VARCHAR2);
```

  表 10-142 DBE_LOB.LOB_WRITE_APPEND 接口参数说明

参数	说明
blob_obj/ clob_obj	待写入的指定BLOB/CLOB类型对象。
amount	写入长度，最大支持32767字符。 **说明** 如果写入长度小于1或写入长度大于待写入的内容长度，则报错。
source_obj	待写入的内容。

- DBE_LOB.COPY

  存储过程COPY根据指定长度及起始位置偏移将LOB内容拷贝到另一个LOB中。

  DBE_LOB.COPY函数原型为：

```
DBE_LOB.COPY (
   dest_lob   INOUT BLOB,
   src_lob    IN    BLOB,
   len        IN    INTEGER,
   dest_start IN    INTEGER DEFAULT 1,
   src_start  IN    INTEGER DEFAULT 1);
```

  表 10-143 DBE_LOB.COPY 接口参数说明

参数	说明
dest_lob	待拷入的LOB类型对象。
src_lob	待拷出的LOB类型对象。
len	拷贝长度。
dest_start	指定从dest_lob内容的哪个位置开始拷入的偏移（即相对LOB内容起始位置的字节数）。
src_start	指定从src_lob内容的哪个位置开始拷出的偏移（即相对LOB内容起始位置的字节数）。

- DBE_LOB.LOB_COPY

  存储过程COPY根据指定长度及起始位置偏移将LOB内容拷贝到另一个LOB中。

DBE_LOB.LOB_COPY函数原型为：

```
DBE_LOB.LOB_COPY(
   blob_obj    INOUT BLOB,
   source_obj  IN    BLOB,
   amount      IN    BIGINT,
   dest_offset IN    BIGINT DEFAULT 1,
   src_offset  IN    BIGINT DEFAULT 1);

DBE_LOB.LOB_COPY(
   clob_obj    INOUT CLOB,
   source_obj  IN    CLOB,
   amount      IN    BIGINT,
   dest_offset IN    BIGINT DEFAULT 1,
   src_offset  IN    BIGINT DEFAULT 1);
```

表 10-144 DBE_LOB.LOB_COPY 接口参数说明

参数	说明
blob_obj/ clob_obj	待拷入的LOB类型对象。
source_obj	待拷出的LOB类型对象。
amount	拷贝长度。 **说明** 如果拷入长度小于1或拷入长度大于LOBMAXSIZE，则报错。
dest_offset	指定从blob_obj/clob_obj内容的哪个位置开始拷入的偏移（即相对LOB内容起始位置的字节数/字符数，BLOB对象以字节为单位，CLOB对象以字符为单位）。 **说明** 如果偏移量小于1或者大于LOBMAXSIZE，则报错。
src_offset	指定从source_obj内容的哪个位置开始拷出的偏移（即相对LOB内容起始位置的字节数/字符数，BLOB对象以字节为单位，CLOB对象以字符为单位）。 **说明** 如果偏移量小于1则报错。

- DBE_LOB.ERASE

  存储过程ERASE根据指定长度及起始位置偏移删除blob_obj中的内容（不支持大于1GB），blob_obj中删除部分的字节填充为0。

  DBE_LOB.ERASE函数原型为：

  ```
  DBE_LOB.ERASE (
     blob_obj INOUT BLOB,
     amount   INOUT INTEGER,
     off_set  IN    INTEGER DEFAULT 1);
  ```

  表 10-145 DBE_LOB.ERASE 接口参数说明

参数	说明
blob_obj	IN参数为待删除内容的LOB类型对象，OUT参数为删除指定部分后的LOB类型对象，传空报错。

参数	说明
amount	IN参数为待删除的长度（BLOB对象以字节为单位），OUT参数为实际删除的长度。 **说明** 如果删除长度小于1或传空，则报错。
off_set	指定从LOB内容的哪个位置开始删除的偏移（即相对BLOB内容起始位置的字节数，不支持大于1GB）。 **说明** 如果偏移量小于1或偏移量传空，则报错。

- DBE_LOB.LOB_ERASE

  存储过程LOB_ERASE根据指定长度及起始位置偏移删除LOB中的内容，blob中删除部分的字节填充为0，clob中删除部分的字符填充为空格，支持LOB大于1GB，最大支持32TB。

  DBE_LOB.LOB_ERASE函数原型为：

```
DBE_LOB.LOB_ERASE (
   blob_obj INOUT BLOB,
   amount   INOUT BIGINT,
   off_set  IN    BIGINT DEFAULT 1);

DBE_LOB.LOB_ERASE (
   clob_obj INOUT CLOB,
   amount   INOUT BIGINT,
   off_set  IN    BIGINT DEFAULT 1);
```

  表 10-146 DBE_LOB.LOB_ERASE 接口参数说明

参数	说明
blob_obj/ clob_obj	IN参数为待删除内容的LOB类型对象，OUT参数为删除指定部分后的LOB类型对象，传空报错。
amount	IN参数为待删除的长度（BLOB对象以字节为单位，CLOB对象以字符为单位），OUT参数为实际删除的长度。 **说明** 如果删除长度小于1或传空，则报错。
off_set	指定从LOB内容的哪个位置开始删除的偏移（即相对BLOB内容起始位置的字节数/相对CLOB内容起始位置的字符数）。 **说明** 如果偏移量小于1或偏移量传空，则报错。

- DBE_LOB.CLOSE

  存储过程CLOSE关闭已经打开的LOB描述符。

  DBE_LOB.CLOSE函数原型为：

```
DBE_LOB.CLOSE(
   lob IN BLOB);

DBE_LOB.CLOSE (
   lob IN CLOB);
```



```
DBE_LOB.CLOSE (
   file IN INTEGER);
```

表 10-147 DBE_LOB.CLOSE 接口参数说明

参数	说明
lob/file	待关闭的BLOB/CLOB/文件类型对象。

- DBE_LOB.MATCH

  该函数返回字符串在LOB或者BFILE文件中第N次出现的位置，如果输入的是一些无效值会返回NULL值。支持LOB或者BFILE文件大于1GB，最大支持32TB。

  DBE_LOB.MATCH函数原型为：

```
DBE_LOB.MATCH(
   blob_obj    IN BLOB,
   blob_obj2   IN RAW,
   beg_index   IN BIGINT DEFAULT 1,
   occur_index IN BIGINT DEFAULT 1)
RETURN BIGINT;

DBE_LOB.MATCH(
   clob_obj    IN CLOB,
   clob_obj2   IN VARCHAR2,
   beg_index   IN BIGINT DEFAULT 1,
   occur_index IN BIGINT DEFAULT 1)
RETURN BIGINT;

DBE_LOB.MATCH(
   bfile       IN DBE_LOB.BFILE,
   blob_obj2     IN RAW,
   beg_index   IN BIGINT DEFAULT 1,
   occur_index IN BIGINT DEFAULT 1)
RETURN BIGINT;
```

表 10-148 DBE_LOB.MATCH 接口参数说明

参数	说明
blob_obj/ clob_obj/ bfile	要查找的BLOB/CLOB描述符，或者BFILE文件（必须先通过DBE_LOB.BFILEOPEN打开），传空返回null。
blob_obj 2/ clob_obj2	要匹配的模式，对于BLOB/BFILE是由一组RAW类型的数据组成，对于CLOB是由一组VARCHAR2类型的数据组成，传空返回null。
beg_inde x	对于BLOB/BFILE是以字节为单位的绝对偏移量，对于CLOB是以字符为单位的偏移量，模式匹配的起始位置是1。 **说明** 有效范围为1~LOBMAXSIZE，超过返回null。
occur_ind ex	模式匹配的次数，最小值为1。 **说明** 若大于模式串在lob中最大能匹配上的次数，则返回0，若不在范围1~LOBMAXSIZE，则返回null。

- DBE_LOB.COMPARE

  这个函数比较部分或者全部LOB或BFILE。

  - 如果比较的结果相等返回0，否则返回非零的值。
  - 如果第一个LOB比第二个小，返回-1；如果第一个LOB比第二个大，返回1。
  - 如果len，start1，start2这几个参数有无效参数返回NULL，有效的偏移量范围是1~LOBMAXSIZE。
  - 如果start_pos1，start_pos2同时超过LOB/BFILE长度，则返回0。

  DBE_LOB.COMPARE函数原型为：

```
DBE_LOB.COMPARE (
   lob1       IN BLOB,
   lob2       IN BLOB,
   len        IN BIGINT DEFAULT 1073741312,
   start_pos1 IN BIGINT DEFAULT 1,
   start_pos2 IN BIGINT DEFAULT 1)
RETURN INTEGER;

DBE_LOB.COMPARE (
   lob1       IN CLOB,
   lob2       IN CLOB,
   len        IN BIGINT DEFAULT 1073741312,
   start_pos1 IN BIGINT DEFAULT 1,
   start_pos2 IN BIGINT DEFAULT 1)
RETURN INTEGER;
DBE_LOB.COMPARE (
   file1      IN DBE_LOB.BFILE,
   file2      IN DBE_LOB.BFILE,
   len        IN BIGINT DEFAULT 1073741312,
   start_pos1 IN BIGINT DEFAULT 1,
   start_pos2 IN BIGINT DEFAULT 1)
RETURN INTEGER;
```

  表 10-149 DBE_LOB.COMPARE 接口参数说明

参数	说明
lob1/file1	第一个要比较的BLOB/CLOB/BFILE类型对象（必须先通过DBE_LOB.BFILEOPEN打开）。
lob2/file2	第二个要比较的BLOB/CLOB/BFILE类型对象（必须先通过DBE_LOB.BFILEOPEN打开）。
len	要比较的字符数或者字节数，默认值为1073741312。
start_pos1	第一个LOB描述符的偏移量，初始位置是1，最大值为lob类型最大长度。
start_pos2	第二个LOB描述符的偏移量，初始位置是1，最大值为lob类型最大长度。

- DBE_LOB.SUBSTR

  该函数用于读取一个LOB的子串，返回读取的子串。

  DBE_LOB.SUBSTR函数原型为：

```
DBE_LOB.SUBSTR(
   lob_loc IN BLOB,
   amount  IN INTEGER DEFAULT 32767,
   off_set IN INTEGER  DEFAULT 1)
RETURN RAW;
```



```
DBE_LOB.SUBSTR(
   lob_loc IN CLOB,
   amount  IN INTEGER DEFAULT 32767,
   off_set IN INTEGER  DEFAULT 1)
RETURN VARCHAR2;
```

表 10-150 DBE_LOB.SUBSTR 接口参数说明

参数	说明
lob_loc	将要读取子串的LOB描述符，对于BLOB的返回值是读取的RAW类型，对于CLOB类型的返回值是VARCHAR2类型。
amount	要读取的字节数或者字符数量。 **说明** 范围为1~32767，超出返回NULL。
off_set	从开始位置偏移的字符数或者字节数量。 **说明** 范围为1~LOBMAXSIZE，超出返回NULL。

- DBE_LOB.LOB_SUBSTR

  该函数用于读取一个LOB或者BFILE的子串，返回读取的子串，支持LOB或者BFILE大于1GB，最大支持32TB。

  DBE_LOB.LOB_SUBSTR函数原型为：

```
DBE_LOB.LOB_SUBSTR(
   lob_loc IN BLOB,
   amount  IN INTEGER DEFAULT 32767,
   off_set IN BIGINT  DEFAULT 1)
RETURN RAW;

DBE_LOB.LOB_SUBSTR(
   lob_loc IN CLOB,
   amount  IN INTEGER DEFAULT 32767,
   off_set IN BIGINT  DEFAULT 1)
RETURN VARCHAR2;

DBE_LOB.LOB_SUBSTR(
   bfile   IN DBE_LOB.BFILE,
   amount  IN INTEGER DEFAULT 32767,
   off_set IN BIGINT  DEFAULT 1)
RETURN RAW;
```

表 10-151 DBE_LOB.LOB_SUBSTR 接口参数说明

参数	说明
lob_loc/ bfile	将要读取子串的LOB描述符或BFILE文件（必须先通过DBE_LOB.BFILEOPEN打开），对于BLOB/BFILE类型的返回值是读取的RAW类型，对于CLOB类型的返回值是VARCHAR2类型。
amount	要读取的字节数或者字符数量。 **说明** 范围为1~32767，超出返回null。
off_set	从开始位置偏移的字符数或者字节数量。 **说明** 范围为1~LOBMAXSIZE，超出返回null。

- DBE_LOB.STRIP

  这个存储过程用于截断指定长度的LOB，执行完这个存储过程会将LOB的长度设置为newlen参数指定的长度。

  DBE_LOB.STRIP函数原型为：

```
DBE_LOB.STRIP(
   lob_loc INOUT BLOB,
   newlen  IN INTEGER);

DBE_LOB.STRIP(
   lob_loc INOUT CLOB,
   newlen  IN INTEGER);
```

  表 10-152 DBE_LOB.STRIP 接口参数说明

参数	说明
lob_loc	IN参数为待读入的指定LOB类型对象，OUT参数为截断后的对象，传空报错。
newlen	截断后LOB的新长度，对于BLOB是字节数，对于CLOB是字符数。

- DBE_LOB.LOB_STRIP

  这个存储过程用于截断指定长度的LOB，执行完这个存储过程会将LOB的长度设置为newlen参数指定的长度。支持LOB大于1GB，最大支持32TB。

  DBE_LOB.LOB_STRIP函数原型为：

```
DBE_LOB.LOB_STRIP(
   lob_loc INOUT BLOB,
   newlen  IN    BIGINT);

DBE_LOB.LOB_STRIP(
   lob_loc INOUT CLOB,
   newlen  IN    BIGINT);
```

  表 10-153 DBE_LOB.LOB_STRIP 接口参数说明

参数	说明
lob_loc	IN参数为待读入的指定LOB类型对象，OUT参数为截断后的对象，传空报错。
newlen	截断后LOB的新长度，对于BLOB是字节数，对于CLOB是字符数。 **说明** 小于1返回null，大于lob的长度，报错。

- DBE_LOB.CREATE_TEMPORARY

  这个存储过程创建一个临时的BLOB或者CLOB，这个存储过程仅用于语法上的兼容，并无实际意义。

  DBE_LOB.CREATE_TEMPORARY函数原型为：

```
DBE_LOB.CREATE_TEMPORARY (
   lob_loc INOUT BLOB,
   cache   IN    BOOLEAN,
   dur     IN    INTEGER DEFAULT 10);

DBE_LOB.CREATE_TEMPORARY (
```



```
  lob_loc INOUT CLOB,
  cache   IN    BOOLEAN,
  dur     IN    INTEGER DEFAULT 10);
```

表 10-154 DBE_LOB.CREATE_TEMPORARY 接口参数说明

参数	说明
lob_loc	LOB描述符。
cache	仅用于语法上的兼容。
dur	仅用于语法上的兼容。

- DBE_LOB.APPEND

  存储过程APPEND将source_obj拼接在目标lob之后。

  DBE_LOB.APPEND函数原型为：

```
DBE_LOB.APPEND (
   blob_obj INOUT BLOB,
   source_obj  IN    BLOB);

DBE_LOB.APPEND (
   clob_obj INOUT CLOB,
   source_obj  IN    CLOB);
```

  表 10-155 DBE_LOB.APPEND 接口参数说明

参数	说明
blob_obj/ clob_obj	要写入的BLOB/CLOB对象。
source_obj	读取的BLOB/CLOB对象。

- DBE_LOB.LOB_APPEND

  存储过程LOB_APPEND将source_obj拼接在目标lob之后。

  DBE_LOB.LOB_APPEND函数原型为：

```
DBE_LOB.LOB_APPEND(
   blob_obj   INOUT BLOB,
   source_obj IN    BLOB);

DBE_LOB.LOB_APPEND(
   clob_obj   INOUT CLOB,
   source_obj IN    CLOB);
```

  表 10-156 DBE_LOB.LOB_APPEND 接口参数说明

参数	说明
blob_obj/ clob_obj	要写入的BLOB/CLOB对象。
source_obj	读取的BLOB/CLOB对象。

- DBE_LOB.FREETEMPORARY

存储过程用于释放由CREATE_TEMPORARY创建的LOB文件。

DBE_LOB.FREETEMPORARY函数原型为：

```
DBE_LOB.FREETEMPORARY (
   blob INOUT BLOB);

DBE_LOB.FREETEMPORARY (
   clob INOUT CLOB);
```

表 10-157 DBE_LOB.FREETEMPORARY 接口参数说明

参数	说明
blob/clob	要释放的BLOB/CLOB对象。

- DBE_LOB.FILEOPEN

  这个函数用于打开数据库外部BFILE类型文件，并返回这个文件对用的文件描述符（fd），一个会话最多支持打开10个BFILE文件。BFILE类型定义为：

```
DBE_LOB.BFILE (
   directory TEXT,
   filename  TEXT,
   fd        INTEGER);
```

  DBE_LOB.FILEOPEN函数原型为：

```
DBE_LOB.FILEOPEN (
   bfile     IN DBE_LOB.BFILE,
   open_mode IN TEXT)
RETURN INTEGER;
```

  表 10-158 DBE_LOB.FILEOPEN 接口参数说明

参数	说明
bfile	要打开的数据库外部文件（BFILE类型包含了文件路径和文件名、文件描述符（fd））。 **说明** file变量中包含文件目录的位置directory，文件名filename。 • 文件目录的位置，需要添加到系统表20.2.57-PG_DIRECTORY中，如果传入的路径和20.2.57-PG_DIRECTORY中的路径不匹配，会报路径不存在的错误。 • 在打开guc参数safe_data_path时，用户只能通过高级包读写safe_data_path指定文件路径下的文件。 • 文件名，包含扩展（文件类型），不包括路径名。如果文件名中包含路径，在OPEN中会被忽略，在Unix系统中，文件名不能以/.结尾。
open_mode	文件打开模式，只支持read模式（r），其他模式报错。

- DBE_LOB.FILECLOSE

  这个函数用于关闭数据外部BFILE类型文件。

  DBE_LOB.FILECLOSE函数原型为：

```
DBE_LOB.FILECLOSE (
   file IN INTEGER);
```


表 10-159 DBE_LOB.FILECLOSE 接口参数说明

参数	说明
file	要关闭的数据库外部文件（由FILEOPEN返回的文件描述符）。

- DBE_LOB.BFILEOPEN

  这个存储过程用于打开数据库外部BFILE类型文件，一个会话最多支持打开10个BFILE文件。

  DBE_LOB.BFILEOPEN原型为：

```
DBE_LOB.BFILEOPEN (
   bfile     INOUT DBE_LOB.BFILE,
   open_mode IN    TEXT DEFAULT 'R');
```

表 10-160 DBE_LOB.BFILEOPEN 接口参数说明

参数	说明
bfile	INOUT参数为打开的数据库BFILE文件。 **说明** bfile变量中包含文件目录的位置directory，文件名filename。 • 文件目录的位置，需要添加到系统表**20.2.57-PG_DIRECTORY**中，如果传入的路径和**20.2.57-PG_DIRECTORY**中的路径不匹配，会报路径不存在的错误。 • 在打开guc参数safe_data_path时，用户只能通过高级包读写safe_data_path指定文件路径下的文件。 • 文件名，包含扩展（文件类型），不包括路径名。如果文件名中包含路径，在OPEN中会被忽略，在Unix系统中，文件名不能以/.结尾。
open_mode	文件打开模式，只支持read模式（r），其他模式报错。

示例

```
--获取bfile文件的子串
DECLARE
bfile dbe_lob.bfile;
BEGIN
bfile = DBE_LOB.BFILENAME(dir_name, file_name); --获取对应bfile文件对象
DBE_LOB.bfileopen(bfile, 'r'); --打开bfile文件
RAISE NOTICE 'res:%', DBE_LOB.lob_substr(bfile, 10, 1); --获取子串，并打印
DBE_LOB.bfileclose(bfile);--关闭bfile文件
END;
/
```

- DBE_LOB.BFILECLOSE

  这个存储过程用于关闭数据库外部BFILE类型文件。

  DBE_LOB.BFILECLOSE原型为：

```
DBE_LOB.BFILECLOSE (
   bfile INOUT DBE_LOB.BFILE);
```


表 10-161 DBE_LOB.BFILECLOSE 接口参数说明

参数	说明
bfile	INOUT参数为关闭的数据库BFILE文件。

- DBE_LOB.LOADFROMFILE

  这个函数用于将BFILE类型外部文件读取到BLOB文件中，并以RAW类型返回该对象。

  DBE_LOB.LOADFROMFILE函数原型为：

```
DBE_LOB.LOADFROMFILE (
   dest_lob    IN BLOB,
   src_file    IN INTEGER,
   amount      IN INTEGER,
   dest_offset IN INTEGER,
   src_offset  IN INTEGER)
RETURN RAW;
```

表 10-162 DBE_LOB.LOADFROMFILE 接口参数说明

参数	说明
dest_lob	目标BLOB对象，BFILE文件将读取到这个文件中的指定偏移位置。
src_bfile	需要读取的源BFILE文件。
amount	读取BFILE文件内容的长度。 **说明** 长度小于1或者大于32767则报错。
dest_offset	BLOB对象的偏移长度。 **说明** 偏移量小于1或者大于LOBMAXSIZE则报错。
src_offset	BFILE文件的偏移长度。 **说明** • 偏移量小于1或者大于LOBMAXSIZE则报错。 • amount + src_offset 大于 src_bfile的长度 + 1 则报错。

- DBE_LOB.LOADFROMBFILE

  这个存储过程用于将BFILE类型外部文件读取到LOB对象中。

  DBE_LOB.LOADFROMBFILE函数原型为：

```
DBE_LOB.LOADFROMBFILE (
   dest_lob    INOUT BLOB,
   src_file    IN    DBE_LOB.BFILE,
   amount      IN    BIGINT,
   dest_offset IN    BIGINT DEFAULT 1,
   src_offset  IN    BIGINT DEFAULT 1)
RETURN BLOB;

DBE_LOB.LOADFROMBFILE (
   dest_lob    INOUT CLOB,
   src_file    IN    DBE_LOB.BFILE,
   amount      IN    BIGINT,
   dest_offset IN    BIGINT DEFAULT 1,
```

```
  src_offset  IN    BIGINT DEFAULT 1)
RETURN CLOB;
```

表 10-163 DBE_LOB.LOADFROMBFILE 接口参数说明

参数	说明
dest_lob	IN参数为目标LOB对象，BFILE文件（必须先通过DBE_LOB.BFILEOPEN打开）将读取到这个对象中，OUT参数为完成读取后返回的LOB对象，大小支持超过1GB，最大支持32TB。
src_file	需要读取的源BFILE文件，BFILE文件的大小支持超过1GB，最大支持32TB。
amount	读取BFILE文件内容和写入lob的长度。 **说明** 长度小于1或者大于LOBMAXSIZE则报错。
dest_offset	LOB对象的偏移长度 **说明** 偏移量小于1或者大于LOBMAXSIZE则报错。
src_offset	BFILE文件的偏移长度 **说明** • 偏移量小于1或者大于LOBMAXSIZE则报错。 • amount + src_offset 大于 src_bfile的长度 + 1 则报错。

- DBE_LOB.LOADBLOBFROMFILE

  这个函数用于将BFILE类型外部文件读取到BLOB文件中，并以RAW类型返回该对象。

  DBE_LOB.LOADBLOBFROMFILE函数原型为：

```
DBE_LOB.LOADBLOBFROMFILE (
   dest_lob    IN BLOB,
   src_file    IN INTEGER,
   amount      IN INTEGER,
   dest_offset IN INTEGER,
   src_offset  IN INTEGER)
RETURN RAW;
```

表 10-164 DBE_LOB.LOADBLOBFROMFILE 接口参数说明

参数	说明
dest_lob	目标BLOB对象，BFILE文件将读取到这个对象中。
src_file	需要读取的源BFILE文件。
amount	BLOB对象的长度，超过这个阈值的文件将不会保存到BLOB中。 **说明** 长度小于1或者大于32767则报错。

参数	说明
dest_offset	BLOB对象的偏移长度。 **说明** 偏移量小于1或者大于LOBMAXSIZE则报错。
src_offset	BFILE文件的偏移长度。 **说明** ● 偏移量小于1或者大于LOBMAXSIZE则报错。 ● amount + src_offset 大于 src_bfile的长度 + 1 则报错。

- DBE_LOB.LOADBLOBFROMBFILE

  这个存储过程用于将BFILE类型外部文件读取到BLOB对象中。

  DBE_LOB.LOADBLOBFROMBFILE函数原型为：

```
DBE_LOB.LOADBLOBFROMFILE (
    dest_lob     INOUT BLOB,
    src_file     IN     DEB_LOB.BFILE,
    amount       IN     BIGINT,
    dest_offset INOUT BIGINT,
    src_offset   INOUT BIGINT)
```

表 10-165 DBE_LOB.LOADBLOBFROMBFILE 接口参数说明

参数	说明
dest_lob	IN参数为目标BLOB对象，BFILE文件（必须先通过DBE_LOB.BFILEOPEN打开）将读取到这个对象中，OUT参数为完成读取后返回的BLOB对象。大小支持超过1GB，最大支持32TB。
src_file	需要读取的源BFILE文件，BFILE文件的大小支持超过1GB，最大支持32TB。
amount	BLOB对象的长度，超过这个阈值的文件将不会保存到BLOB中。 **说明** 长度小于1或者大于LOBMAXSIZE则报错。
dest_offset	BLOB对象的偏移长度。 **说明** 偏移量小于1或者大于LOBMAXSIZE则报错。
src_offset	BFILE文件的偏移长度。 **说明** ● 偏移量小于1或者大于LOBMAXSIZE则报错。 ● amount + src_offset 大于 src_bfile的长度 + 1 则报错。

- DBE_LOB.LOADCLOBFROMFILE

  这个函数用于将BFILE类型外部文件读取到CLOB文件中，并以RAW类型返回该对象。

  DBE_LOB.LOADCLOBFROMFILE函数原型为：


```
DBE_LOB.LOADCLOBFROMFILE (
   dest_lob     IN CLOB,
   src_file     IN INTEGER,
   amount       IN INTEGER,
   dest_offset IN INTEGER,
   src_offset   IN INTEGER)
RETURN RAW;
```

表 10-166 DBE_LOB.LOADCLOBFROMFILE 接口参数说明

参数	说明
dest_lob	目标CLOB对象，BFILE文件将读取到这个文件中。
src_file	需要读取的源BFILE文件。
amount	CLOB对象的长度。 **说明** 长度小于1或者大于32767则报错。
dest_offset	CLOB对象的偏移长度。 **说明** 偏移量小于1或者大于LOBMAXSIZE则报错。
src_offset	BFILE文件的偏移长度。 **说明** ● 偏移量小于1或者大于LOBMAXSIZE则报错。 ● amount + src_offset 大于 src_bfile的长度 + 1 则报错。

- DBE_LOB.LOADCLOBFROMBFILE

  这个存储过程用于将BFILE类型外部文件读取到CLOB对象中。

  DBE_LOB.LOADCLOBFROMBFILE函数原型为：

```
DBE_LOB.LOADCLOBFROMBFILE (
   dest_lob     INOUT CLOB,
   src_file     IN     DEB_LOB.BFILE,
   amount       IN     BIGINT,
   dest_offset INOUT BIGINT,
   src_offset   INOUT BIGINT)
```

表 10-167 DBE_LOB.LOADCLOBFROMBFILE 接口参数说明

参数	说明
dest_lob	IN参数为目标CLOB对象，BFILE文件（必须先通过DBE_LOB.BFILEOPEN打开）将读取到这个对象中，OUT参数为完成读取后返回的CLOB对象。大小支持超过1GB，最大支持32TB。
src_file	需要读取的源BFILE文件。BFILE文件的大小支持超过1GB，最大支持32TB。
amount	CLOB对象的长度，超过这个阈值的文件将不会保存到CLOB中。 **说明** 长度小于1或者大于LOBMAXSIZE则报错。

参数	说明
dest_offset	CLOB对象的偏移长度。 **说明** 偏移量小于1或者大于LOBMAXSIZE则报错。
src_offset	BFILE文件的偏移长度。 **说明** ● 偏移量小于1或者大于LOBMAXSIZE则报错。 ● amount + src_offset 大于 src_bfile的长度 + 1 则报错。

- DBE_LOB.CONVERTTOBLOB

  这个函数将clob对象转换成blob对象，不支持大于1GB的场景。

  DBE_LOB.CONVERTTOBLOB函数原型为：

```
DBE_LOB.CONVERTTOBLOB(
   dest_blob   IN BLOB,
   src_clob    IN CLOB,
   amount      IN INTEGER DEFAULT 32767,
   dest_offset IN INTEGER DEFAULT 1,
   src_offset  IN INTEGER DEFAULT 1)
RETURN RAW;
```

  表 10-168 DBE_LOB.CONVERTTOBLOB 接口参数说明

参数	说明
dest_blob	目标blob对象，clob转换后的文件。
src_clob	需要读取的源clob对象。
amount	clob对象的长度，超过这个阈值的文件将不会保存到blob中。长度小于1或者大于LOBMAXSIZE则报错。
dest_offset	blob对象的偏移长度，dest_offset=1将从文件起始位置开始载入，以此类推。
src_offset	clob对象的偏移长度，src_offset=1将从文件起始位置开始读取，以此类推。

- DBE_LOB.LOB_CONVERTTOBLOB

  这个函数将clob对象转换成blob对象，支持LOB大于1GB。

  DBE_LOB.LOB_CONVERTTOBLOB函数原型为：

```
DBE_LOB.LOB_CONVERTTOBLOB(
   dest_blob   INOUT BLOB,
   src_clob    IN    CLOB,
   amount      IN    BIGINT,
   dest_offset INOUT BIGINT,
   src_offset  INOUT BIGINT)
```

  表 10-169 DBE_LOB.LOB_CONVERTTOBLOB 接口参数说明

参数	说明
dest_blob	目标blob对象，clob转换后的文件。

参数	说明
src_clob	需要读取的源clob对象。
amount	clob对象的长度，超过这个阈值的文件将不会保存到blob中。长度小于1或者大于LOBMAXSIZE则报错。
dest_offset	blob对象的偏移长度，dest_offset=1将从文件起始位置开始载入，以此类推。偏移量小于1或者大于LOBMAXSIZE则报错。
src_offset	clob对象的偏移长度，src_offset=1将从文件起始位置开始读取，以此类推。偏移量小于1或者大于LOBMAXSIZE则报错。

- DBE_LOB.CONVERTTOCLOB

  这个函数将blob对象转换成clob对象，不支持大于1GB的场景。

  DBE_LOB.CONVERTTOCLOB函数原型为：

```
DBE_LOB.CONVERTTOCLOB(
   dest_clob   IN CLOB,
   src_blob    IN BLOB,
   amount      IN INTEGER DEFAULT 32767,
   dest_offset IN INTEGER DEFAULT 1,
   src_offset  IN INTEGER DEFAULT 1)
RETURN text;
```

  表 10-170 DBE_LOB.CONVERTTOCLOB 接口参数说明

参数	说明
dest_clob	目标clob对象，blob转换后的文件。
src_blob	需要读取的源blob对象。
amount	blob对象的长度，超过这个阈值的文件将不会保存到clob中。
dest_offset	clob对象的偏移长度，dest_offset=1将从文件起始位置开始载入，以此类推。
src_offset	blob对象的偏移长度，src_offset=1将从文件起始位置开始读取，以此类推。

- DBE_LOB.LOB_CONVERTTOCLOB

  这个函数将blob对象转换成clob对象，支持LOB大于1G。

  DBE_LOB.LOB_CONVERTTOCLOB函数原型为：

```
DBE_LOB.LOB_CONVERTTOCLOB(
   dest_clob    INOUT CLOB,
   src_blob    IN    BLOB,
   amount      IN    BIGINT,
   dest_offset INOUT BIGINT,
   src_offset  INOUT BIGINT)
```


表 10-171 DBE_LOB.LOB_CONVERTTOCLOB 接口参数说明

参数	说明
dest_clob	目标clob对象，blob转换后的文件。
src_blob	需要读取的源blob对象。
amount	blob对象的长度，超过这个阈值的文件将不会保存到clob中。
dest_offset	clob对象的偏移长度，dest_offset=1将从文件起始位置开始载入，以此类推。偏移量小于1或者大于LOBMAXSIZE则报错。
src_offset	blob对象的偏移长度，src_offset=1将从文件起始位置开始读取，以此类推。偏移量小于1或者大于LOBMAXSIZE则报错。

- DBE_LOB.GETCHUNKSIZE

  在数据库存储LOB数据时，内部使用toast存储，本函数返回TOAST_MAX_CHUNK_SIZE。

  DBE_LOB.GETCHUNKSIZE的函数原型为：

```
DBE_LOB.GETCHUNKSIZE(
    lob_loc IN CLOB
)RETURN INTEGER

DBE_LOB.GETCHUNKSIZE(
    lob_loc IN BLOB
)RETURN INTEGER
```

表 10-172 DBE_LOB.GETCHUNKSIZE 接口参数说明

参数	说明
lob_loc	目标CLOB/BLOB对象。

- DBE_LOB.LOB_WRITE

  将源对象从起始位置读取指定长度内容，写入目标LOB对象的指定偏移位置，覆盖该位置原本内容, 并返回目标LOB对象。

  DBE_LOB.LOB_WRITE函数原型为：

```
DBE_LOB.LOB_WRITE(
    clob_obj INOUT CLOB,
    amount   IN    INTEGER,
    off_set  IN    BIGINT,
    source   IN    VARCHAR2
)
RETURN CLOB;

DBE_LOB.LOB_WRITE(
    blob_obj INOUT BLOB,
    amount   IN    INTEGER,
    off_set  IN    BIGINT,
    source   IN    RAW
)
RETURN BLOB;
```


表 10-173 DBE_LOB.LOB_WRITE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
blob_obj / clob_obj	BLOB/ CLOB	INOUT	否	IN参数为待写入的目标LOB，OUT参数为写入内容后的LOB。
amount	INTEGER	IN	否	待写入的长度（BLOB以字节为单位，CLOB以字符为单位）。
off_set	BIGINT	IN	否	在blob_obj/clob_obj中写入的偏移位置。
source	RAW/ VARCHAR2	IN	否	源对象。

- DBE_LOB.BFILENAME

  这个函数用于根据目录和文件名构造BFILE类型对象。

  DBE_LOB.BFILENAME原型为：

```
DBE_LOB.BFILENAME(
    directory IN TEXT,
    filename  IN TEXT)
RETURN DBE_LOB.BFILE;
```

表 10-174 DBE_LOB.BFILENAME 接口参数说明

参数	说明
directory	文件目录 **说明** 文件目录的位置，需要添加到系统表**20.2.57-PG_DIRECTORY**中，如果传入的路径和**20.2.57-PG_DIRECTORY**中的路径不匹配，会报路径不存在的错误。 • 在打开guc参数safe_data_path时，用户只能通过高级包读写safe_data_path指定文件路径下的文件。 • 文件名，包含扩展（文件类型），不包括路径名。如果文件名中包含路径，在OPEN中会被忽略，在Unix系统中，文件名不能以/.结尾。
filename	文件名

示例

```
--获取字符串的长度
SELECT DBE_LOB.GET_LENGTH('12345678');
 get_length
```

```
------------
         8
(1 row)

-- DBE_LOB.READ接口示例
DECLARE
myraw  RAW(100);
amount INTEGER :=2;
buffer INTEGER :=1;
begin
DBE_LOB.READ('123456789012345',amount,buffer,myraw);
dbe_output.print_line(myraw);
end;
/
0123
ANONYMOUS BLOCK EXECUTE

CREATE TABLE blob_Table (t1 blob);
CREATE TABLE blob_Table_bak (t2 blob);
INSERT INTO blob_Table VALUES('abcdef');
INSERT INTO blob_Table_bak VALUES('22222');

-- DBE_LOB多接口示例
DECLARE
str varchar2(100) := 'abcdef';
source raw(100);
dest blob;
copyto blob;
amount int;
PSV_SQL varchar2(100);
PSV_SQL1 varchar2(100);
a int :=1;
len int;
BEGIN
source := dbe_raw.cast_from_varchar2_to_raw(str);
amount := dbe_raw.get_length(source);

PSV_SQL :='select * from blob_Table for update';
PSV_SQL1 := 'select * from blob_Table_bak for update';

EXECUTE IMMEDIATE PSV_SQL into dest;
EXECUTE IMMEDIATE PSV_SQL1 into copyto;

DBE_LOB.WRITE(dest, amount, 1, source);
DBE_LOB.WRITE_APPEND(dest, amount, source);

DBE_LOB.ERASE(dest, a, 1);
DBE_OUTPUT.PRINT_LINE(a);
DBE_LOB.COPY(copyto, dest, amount, 10, 1);
perform DBE_LOB.CLOSE(dest);
RETURN;
END;
/
1
ANONYMOUS BLOCK EXECUTE

--删除表
DROP TABLE blob_Table;
DROP TABLE
DROP TABLE blob_Table_bak;
DROP TABLE
```

10.12.2.2 DBE_RANDOM

接口介绍

高级功能包DBE_RANDOM支持的所有接口请参见**表 DBE_RANDOM接口参数说明**。

表 10-175 DBE_RANDOM 接口参数说明

接口名称	描述
DBE_RANDOM.SET_SEED	设置一个随机数的种子。
DBE_RANDOM.GET_VALUE	生成一个大小介于指定的low及high之间的随机数。

- DBE_RANDOM.SET_SEED

  存储过程SEED用于设置一个随机数的种子。DBE_RANDOM.SET_SEED函数原型为：

  ```
  DBE_RANDOM.SET_SEED (seed  IN  INTEGER);
  ```

  表 10-176 DBE_RANDOM.SET_SEED 接口参数说明

参数	描述
seed	用于产生一个随机数的种子。

- DBE_RANDOM.GET_VALUE

  函数GET_VALUE生成一个大小介于指定的low及high之间的随机数。DBE_RANDOM.GET_VALUE函数原型为：

  ```
  DBE_RANDOM.GET_VALUE(
  min  IN  NUMBER default 0,
  max  IN  NUMBER default 1)
  RETURN NUMBER;
  ```

  表 10-177 DBE_RANDOM.GET_VALUE 接口参数说明

参数	描述
min	指定随机数大小的下边界，生成的随机数大于或等于min。
max	指定随机数大小的上边界，生成的随机数小于max。

  说明

  - 实际上，只要求这里的参数类型是NUMERIC即可，对于左右边界的大小并没有要求。
  - DBE_RANDOM实现的是伪随机，所以若使用的初值（种子）不变，那么伪随机数的数序也不变，使用时需要注意。
  - 生成的随机数有效数字为15位。

示例

```
--产生0到1之间的随机数：
SELECT DBE_RANDOM.GET_VALUE(0,1);
    get_value
------------------
 .917468812743886
(1 row)
```

```
--对于指定范围内的整数，要加入参数min和max，并从结果中截取较小的数（最大值不能被作为可能的值）。所以对于0到99之间的整数，使用下面的代码：
SELECT TRUNC(DBE_RANDOM.GET_VALUE(0,100));
 trunc
-------
    26
(1 row)
```

10.12.2.3 DBE_OUTPUT

说明

当使用DBE_OUTPUT.PUT_LINE打印DBE_FILE.READ_LINE_NCHAR接口得到的结果时，需要确保UTF-8字符集编码能够转换成当前数据库字符集编码，满足上述条件后可以正常输出结果。DBE_OUTPUT.PRINT_LINE不支持该功能。

接口介绍

高级功能包DBE_OUTPUT支持的所有接口请参见**表 DBE_OUTPUT**。

表 10-178 DBE_OUTPUT

接口名称	描述
DBE_OUTPUT.PRINT_LINE	输出指定的文本，并添加换行符。
DBE_OUTPUT.PRINT	输出指定的文本，不添加换行符。
DBE_OUTPUT.SET_BUFFER_SIZE	设置输出缓冲区的大小，如果不指定则缓冲区最大能容纳20000字节，如果指定小于等于2000字节，则缓冲区允许容纳2000字节。
DBE_OUTPUT.DISABLE	禁用对PUT、PUT_LINE、NEW_LINE、GET_LINE和GET_LINES的调用，并清空输出缓冲区。
DBE_OUTPUT.ENABLE	开启缓冲区，允许对PUT、PUT_LINE、NEW_LINE、GET_LINE和GET_LINES的调用，设置缓冲区大小。
DBE_OUTPUT.GET_LINE	从缓冲区中以换行符作为分界获取一行数据，获取的数据将不会输出到客户端。
DBE_OUTPUT.GET_LINES	以VARCHAR数组的形式获取缓冲区的指定行数的字符串，被取出的内容将会在缓冲区中清除，不会输出到客户端。
DBE_OUTPUT.NEW_LINE	放置一行在缓冲区末尾，放置行尾标记，空出新的一行。
DBE_OUTPUT.PUT	将输入字符串放入到缓冲区，末尾不加换行符，在匿名块执行结束时会将以换行符结尾的行输出显示。

接口名称	描述
DBE_OUTPUT.PUT_LINE	将输入字符串放入到缓冲区，并末尾添加换行符，在匿名块执行结束时会将以换行符结尾的行输出显示。

- DBE_OUTPUT.PRINT_LINE

  存储过程PRINT_LINE输出指定的文本，并添加换行符。
  DBE_OUTPUT.PRINT_LINE函数原型为：

  ```
  DBE_OUTPUT.PRINT_LINE (
  format IN VARCHAR2);
  ```

  表 10-179 DBE_OUTPUT.PRINT_LINE 接口参数说明

参数	描述
format	输出的文本。

- DBE_OUTPUT.PRINT

  存储过程PRINT输出指定的文本，不添加换行符。DBE_OUTPUT.PRINT函数原型为：

  ```
  DBE_OUTPUT.PRINT (
  format IN VARCHAR2);
  ```

  表 10-180 DBE_OUTPUT.PRINT 接口参数说明

参数	描述
format	输出的文本。

- DBE_OUTPUT.SET_BUFFER_SIZE

  存储过程SET_BUFFER_SIZE设置输出缓冲区的大小，如果不指定的话缓冲区最大只能容纳20000字节。DBE_OUTPUT.SET_BUFFER_SIZE函数原型为：

  ```
  DBE_OUTPUT.SET_BUFFER_SIZE (
  size IN INTEGER default 20000);
  ```

  表 10-181 DBE_OUTPUT.SET_BUFFER_SIZE 接口参数说明

参数	描述
size	设置输出缓冲区的大小。

- DBE_OUTPUT.DISABLE

  存储过程DISABLE禁用PUT、PUT_LINE、NEW_LINE、GET_LINE、GET_LINES调用，并清空输出缓冲区。DBE_OUTPUT.DISABLE函数原型为：

  ```
  DBE_OUTPUT.DISABLE;
  ```

- DBE_OUTPUT.ENABLE

存储过程ENABLE开启缓冲区，允许对 PUT、PUT_LINE、NEW_LINE、GET_LINE 和 GET_LINES 的调用，并设置缓冲区大小，如果未指定缓冲区大小则默认容纳20000字节。DBE_OUTPUT.ENABLE函数原型为：

```
DBE_OUTPUT.ENABLE (
    buffer_size IN INTEGER DEFAULT 20000
);
```

表 10-182 DBE_OUTPUT.ENABLE 接口参数说明

参数	描述
buffer_size	缓冲区大小的上限，以字节为单位。将 buffer_size 设置为 NULL 表示使用默认值（20000字节）。

- DBE_OUTPUT.GET_LINE

  存储过程GET_LINE从缓冲区中以换行符作为分界获取一行数据，获取的数据将不会输出到客户端。DBE_OUTPUT.GET_LINE函数原型为：

  ```
  DBE_OUTPUT.GET_LINE (
      line   OUT VARCHAR2,
      status OUT INTEGER
  );
  ```

  表 10-183 DBE_OUTPUT.GET_LINE 接口参数说明

参数	描述
line	表示获取到的字符串。
status	表示调用状态是否正常，如果获取到字符串则为0，否则为1。

- DBE_OUTPUT.GET_LINES

  存储过程GET_LINES以VARCHAR数组的形式获取缓冲区的指定行数的字符串，被取出的内容将会在缓冲区中清除，不会输出到客户端。DBE_OUTPUT.GET_LINES函数原型为：

  ```
  DBE_OUTPUT.GET_LINES (
      lines    OUT   VARCHAR[],
      numlines IN OUT INTEGER
  );
  ```

  表 10-184 DBE_OUTPUT.GET_LINES 接口参数说明

参数	描述
lines	输出从缓冲区读取的多行字符串数组。
numlines	输入值为需要从缓冲区检索的行数。输出值为实际检索到的行数，如果输出值小于输入值，则表明缓冲区中存在的行数小于输入的行数。

- DBE_OUTPUT.NEW_LINE

  存储过程NEW_LINE放置一行在缓冲区末尾，放置行尾标记，空出新的一行。DBE_OUTPUT.NEW_LINE函数原型为：

  ```
  DBE_OUTPUT.NEW_LINE;
  ```


- DBE_OUTPUT.PUT

  存储过程PUT将输入字符串放入到缓冲区，末尾不加换行符，在匿名块执行结束时会将以换行符结尾的行输出显示。DBE_OUTPUT.PUT函数原型为：

  ```
  DBE_OUTPUT.PUT (
     item IN VARCHAR2);
  ```

  表 10-185 DBE_OUTPUT.PUT 接口参数说明

参数	描述
item	待放在缓存区的字符或字符串。

- DBE_OUTPUT.PUT_LINE

  存储过程PUT_LINE将输入字符串放入到缓冲区，并且末尾添加换行符，在匿名块执行结束时会将以换行符结尾的行输出显示。DBE_OUTPUT.PUT_LINE函数原型为：

  ```
  DBE_OUTPUT.PUT_LINE (
     item IN VARCHAR2);
  ```

  表 10-186 DBE_OUTPUT.PUT_LINE 接口参数说明

参数	描述
item	待放在缓存区的字符或字符串。

示例

```
BEGIN
    DBE_OUTPUT.SET_BUFFER_SIZE(50);
    DBE_OUTPUT.PRINT('hello, ');
    DBE_OUTPUT.PRINT_LINE('database!');--输出hello, database!
END;
/
-- 预期结果为：
hello, database!
ANONYMOUS BLOCK EXECUTE

-- 测试disable禁用put、put_line、new_line、get_line、get_lines调用,测试put_line不输出
BEGIN
 dbe_output.disable();
        dbe_output.put_line('1');
END;
/
-- 预期结果为：
ANONYMOUS BLOCK EXECUTE

-- 测试enable启用put、put_line、new_line、get_line、get_lines调用，测试put_line输出1
BEGIN
 dbe_output.enable();
        dbe_output.put_line('1');
END;
/
-- 预期结果为：
ANONYMOUS BLOCK EXECUTE

-- 测试put，输入字符串a放入到缓冲区，末尾不加换行符，a不输出
BEGIN
 dbe_output.enable();
 dbe_output.put('a');
```



```
END;
/
-- 预期结果为：
ANONYMOUS BLOCK EXECUTE

-- 测试new_line,添加换行，输出a
BEGIN
 dbe_output.enable();
 dbe_output.put('a');
        dbe_output.new_line;
END;
/
-- 预期结果为：
a
ANONYMOUS BLOCK EXECUTE

-- 测试get_line获取缓冲区数据保存到变量，使用put_line输出
DECLARE
 line VARCHAR(32672);
 status INTEGER := 0;
BEGIN
 dbe_output.put_line('hello');
 dbe_output.get_line(line, status);
 dbe_output.put_line('-----------');
 dbe_output.put_line(line);
 dbe_output.put_line(status);
END;
/
-- 预期结果为：
-----------
hello
0
ANONYMOUS BLOCK EXECUTE

-- 测试get_line获取缓冲区多行内容，使用put_line输出
DECLARE
    lines dbms_output.chararr;
    numlines integer;
BEGIN
    dbe_output.put_line('output line 1');
    dbe_output.put_line('output line 2');
    dbe_output.put_line('output line 3');
    numlines := 100;
    dbe_output.get_lines(lines, numlines);
    dbe_output.put_line('num: ' || numlines);
    dbe_output.put_line('get line 1: ' || lines(1));
    dbe_output.put_line('get line 2: ' || lines(2));
    dbe_output.put_line('get line 3: ' || lines(3));
END;
/
-- 预期结果为：
num: 3
get line 1: output line 1
get line 2: output line 2
get line 3: output line 3
ANONYMOUS BLOCK EXECUTE
```

10.12.2.4 DBE_RAW

DBE_RAW包提供了用于操作RAW类型的接口。

须知

RAW类型的外部表现形式是十六进制，内部存储形式是二进制。例如一个RAW类型的数据11001011的表现形式为‘CB’，即在实际的类型转换中输入的是‘CB’。

接口介绍

高级功能包DBE_RAW支持的所有接口请参见**表 DBE_RAW**。

表 10-187 DBE_RAW

接口名称	描述
DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW	将INTEGER类型值转换为RAW类型。
DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER	将RAW类型值转换为INTEGER类型。
DBE_RAW.GET_LENGTH	返回RAW类型值的长度。
DBE_RAW.CAST_FROM_VARCHAR2_TO_RAW	将VARCHAR2类型值转换为RAW类型。
DBE_RAW.CAST_TO_VARCHAR2	将RAW类型值转换为VARCHAR2类型。
DBE_RAW.BIT_OR	返回RAW类型值按位或计算后的值。
DBE_RAW.SUBSTR	返回RAW类型值取子串后的值。
DBE_RAW.BIT_AND	返回RAW类型按位与计算后的值。
DBE_RAW.BIT_COMPLEMENT	返回RAW类型按位取反计算后的值。
DBE_RAW.BIT_XOR	返回RAW类型按位异或计算后的值。
DBE_RAW.CAST_FROM_BINARY_DOUBLE_TO_RAW	将BINARY_DOUBLE类型值转换为RAW类型。
DBE_RAW.CAST_FROM_RAW_TO_BINARY_DOUBLE	将RAW类型值转换为BINARY_DOUBLE类型。
DBE_RAW.CAST_FROM_RAW_TO_BINARY_FLOAT	将RAW类型值转换为FLOAT4类型。
DBE_RAW.CAST_FROM_BINARY_FLOAT_TO_RAW	将FLOAT4类型值转换为RAW类型。
DBE_RAW.CAST_FROM_RAW_TO_NUMBER	将RAW类型值转换为NUMERIC类型。
DBE_RAW.CAST_FROM_NUMBER_TO_RAW	将NUMERIC类型值转换为RAW类型。
DBE_RAW.CAST_FROM_RAW_TO_NVARCHAR2	将RAW类型值转换为NVARCHAR2类型。
DBE_RAW.COMPARE	返回两个RAW类型值首个不相同的位置。
DBE_RAW.CONCAT	将最多12个RAW类型值连接到一个新的RAW类型值中，并返回。

接口名称	描述
DBE_RAW.CONVERT	将一个RAW类型值从源编码方式from_charset转换为目标编码方式to_charset。
DBE_RAW.COPIES	将一个RAW类型值复制n次，串联在一起，并返回连接后的结果。
DBE_RAW.OVERLAY	用一个RAW类型数据对另一个RAW类型数据进行覆盖，可以指定起始覆盖位置和覆盖长度。
DBE_RAW.REVERSE	将RAW类型数据按字节进行翻转。
DBE_RAW.TRANSLATE	转换或舍弃RAW类型值的指定字节。
DBE_RAW.TRANSLITERATE	转换RAW类型值的指定字节。
DBE_RAW.XRANGE	从一个字节到另一个字节的所有字节拼接后返回。

- DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW

  函数CAST_FROM_BINARY_INTEGER_TO_RAW将INTEGER类型值转换为RAW类型。

  DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW函数原型为：

  ```
  DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW(
      value     IN BIGINT,
      endianess IN INTEGER DEFAULT 1)
  RETURN RAW;
  ```

  表 10-188 DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW 接口参数说明

参数	描述
value	待转成RAW类型的INTEGER类型值。 **说明** DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW预期处理INTEGER类型值，为了兼容性考虑，该参数被定义为BIGINT类型，BIGINT类型范围比INTEGER类型范围更大，不影响接口的使用。
endianess	字节序，取值范围：整型值1、2或3（1表示BIG_ENDIAN，2表示LITTLE_ENDIAN，3表示MACHINE_ENDIAN，默认值为BIG_ENDIAN，如果机器默认为BIG_ENDIAN，则函数执行结果MACHINE_ENDIAN和BIG_ENDIAN一致，如果机器默认为LITTLE_ENDIAN，则函数执行结果MACHINE_ENDIAN和LITTLE_ENDIAN一致）。

- DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER

  函数CAST_FROM_RAW_TO_BINARY_INTEGER将RAW类型值转换为INTEGER类型。

  DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER函数原型为：

  ```
  DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER(
      value     IN RAW,
  ```

```
    endianess IN INTEGER DEFAULT 1)
RETURN INTEGER;
```

表 10-189 DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER 接口参数说明

参数	描述
value	待转成INTEGER类型的RAW类型值。
endianess	字节序，取值范围：整型值1、2或3（1表示BIG_ENDIAN，2表示LITTLE_ENDIAN，3表示MACHINE_ENDIAN，默认值为BIG_ENDIAN，如果机器默认为BIG_ENDIAN，则函数执行结果MACHINE_ENDIAN和BIG_ENDIAN一致，如果机器默认为LITTLE_ENDIAN，则函数执行结果MACHINE_ENDIAN和LITTLE_ENDIAN一致）。

- DBE_RAW.GET_LENGTH

  函数GET_LENGTH返回RAW类型值的长度。

  DBE_RAW.GET_LENGTH函数原型为：

  ```
  DBE_RAW.GET_LENGTH(
     value IN RAW)
  RETURN INTEGER;
  ```

  表 10-190 DBE_RAW.GET_LENGTH 接口参数说明

参数	描述
value	RAW类型值。

- DBE_RAW.CAST_FROM_VARCHAR2_TO_RAW

  函数CAST_FROM_VARCHAR2_TO_RAW将VARCHAR2类型值转换为RAW类型。

  DBE_RAW.CAST_FROM_VARCHAR2_TO_RAW函数原型为：

  ```
  DBE_RAW.CAST_FROM_VARCHAR2_TO_RAW(
     str IN VARCHAR2)
  RETURN RAW;
  ```

  表 10-191 DBE_RAW.CAST_FROM_VARCHAR2_TO_RAW 接口参数说明

参数	描述
str	待转成RAW类型的VARCHAR2类型值。

- DBE_RAW.CAST_TO_VARCHAR2

  函数CAST_TO_VARCHAR2将RAW类型值转换为VARCHAR2类型。

  DBE_RAW.CAST_TO_VARCHAR2函数原型为：

  ```
  DBE_RAW.CAST_TO_VARCHAR2(
     str IN RAW)
  RETURN VARCHAR2;
  ```


表 10-192 DBE_RAW.CAST_TO_VARCHAR2 接口参数说明

参数	描述
str	待转成VARCHAR2类型的RAW类型值。

- DBE_RAW.BIT_OR

  函数BIT_OR返回两个RAW类型值按位或计算后的值。

  DBE_RAW.BIT_OR函数原型为：

```
DBE_RAW.BIT_OR(
    str1 IN TEXT,
    str2 IN TEXT)
RETURN TEXT;
```

表 10-193 DBE_RAW.BIT_OR 接口参数说明

参数	描述
str1	按位或的第一个字符串。 **说明** 由于历史原因，该参数被定义为TEXT类型，但是DBE_RAW.SUBSTR接口预期并处理RAW类型值，这里定义为TEXT类型不影响对RAW类型值的处理。
str2	按位或的第二个字符串。 **说明** 由于历史原因，该参数被定义为TEXT类型，但是DBE_RAW.SUBSTR接口预期并处理RAW类型值，这里定义为TEXT类型不影响对RAW类型值的处理。

- DBE_RAW.SUBSTR

  函数SUBSTR将RAW类型值按起始位off_set和长度amount截取。

  DBE_RAW.SUBSTR函数原型为：

```
DBE_RAW.SUBSTR(
    IN lob_loc BLOB,
    IN off_set integer default 1,
    IN amount integer default 32767)
RETURN RAW;
```

表 10-194 DBE_RAW.SUBSTR 接口参数说明

参数	描述
lob_loc	源RAW类型值。 **说明** 由于历史原因，该参数被定义为BLOB类型，但是DBE_RAW.SUBSTR接口预期并处理RAW类型值，这里定义为BLOB类型不影响对RAW类型值的处理。
off_set	子串的起始位置，默认值1。
amount	子串的长度，默认值32767。

- DBE_RAW.BIT_AND

  函数BIT_AND求两个RAW类型按位与的结果。

  DBE_RAW.BIT_AND函数原型为：

```
DBE_RAW.BIT_AND(
   r1 IN RAW,
   r2 IN RAW)
RETURN RAW;
```

  表 10-195 DBE_RAW.BIT_AND 接口参数说明

参数	描述
r1	与r2进行按位与的RAW类型值，最大长度不超过32767。
r2	与r1进行按位与的RAW类型值，最大长度不超过32767。

- DBE_RAW.BIT_COMPLEMENT

  函数BIT_COMPLEMENT求一个RAW类型按位取反的结果。

  DBE_RAW.BIT_COMPLEMENT函数原型为：

```
DBE_RAW.BIT_COMPLEMENT(
   r IN RAW)
RETURN RAW;
```

  表 10-196 DBE_RAW.BIT_COMPLEMENT 接口参数说明

参数	描述
r	进行按位取反的RAW类型值，最大长度不超过32767。

- DBE_RAW.BIT_XOR

  函数BIT_XOR求两个RAW类型按位异或的结果。

  DBE_RAW.BIT_XOR函数原型为：

```
DBE_RAW.BIT_XOR(
   r1 IN RAW,
   r2 IN RAW)
RETURN RAW;
```

  表 10-197 DBE_RAW.BIT_XOR 接口参数说明

参数	描述
r1	与r2进行按位异或的RAW类型值，最大长度不超过32767。
r2	与r1进行按位异或的RAW类型值，最大长度不超过32767。

- DBE_RAW.CAST_FROM_BINARY_DOUBLE_TO_RAW

  函数CAST_FROM_BINARY_DOUBLE_TO_RAW将BINARY_DOUBLE类型转换为RAW类型。

  DBE_RAW.CAST_FROM_BINARY_DOUBLE_TO_RAW函数原型为：

```
DBE_RAW.CAST_FROM_BINARY_DOUBLE_TO_RAW (
   n         IN BINARY_DOUBLE,
```



```
  endianess IN INTEGER DEFAULT 1)
RETURN RAW;
```

表 10-198 DBE_RAW.CAST_FROM_BINARY_DOUBLE_TO_RAW 接口参数说明

参数	描述
n	需要进行转换的BINARY_DOUBLE类型值。
endianess	字节序，取值范围：整型值1、2或3（1表示BIG_ENDIAN，2表示LITTLE_ENDIAN，3表示MACHINE_ENDIAN，默认值为BIG_ENDIAN，如果机器默认为BIG_ENDIAN，则函数执行结果MACHINE_ENDIAN和BIG_ENDIAN一致，如果机器默认为LITTLE_ENDIAN，则函数执行结果MACHINE_ENDIAN和LITTLE_ENDIAN一致）。

- DBE_RAW.CAST_FROM_RAW_TO_BINARY_DOUBLE

  函数CAST_FROM_RAW_TO_BINARY_DOUBLE将RAW类型转换为BINARY_DOUBLE类型。

  DBE_RAW.CAST_FROM_RAW_TO_BINARY_DOUBLE函数原型为：

  ```
  DBE_RAW.CAST_FROM_RAW_TO_BINARY_DOUBLE(
    r         IN RAW,
    endianess IN INTEGER DEFAULT 1)
  RETURN BINARY_DOUBLE;
  ```

  表 10-199 DBE_RAW.CAST_FROM_RAW_TO_BINARY_DOUBLE 接口参数说明

参数	描述
r	需要进行转换的RAW类型值，最短长度不低于8，最大长度不超过32767。
endianess	字节序，取值范围：整型值1、2或3（1表示BIG_ENDIAN，2表示LITTLE_ENDIAN，3表示MACHINE_ENDIAN，默认值为BIG_ENDIAN，如果机器默认为BIG_ENDIAN，则函数执行结果MACHINE_ENDIAN和BIG_ENDIAN一致，如果机器默认为LITTLE_ENDIAN，则函数执行结果MACHINE_ENDIAN和LITTLE_ENDIAN一致）。

- DBE_RAW.CAST_FROM_RAW_TO_BINARY_FLOAT

  函数CAST_FROM_RAW_TO_BINARY_FLOAT将RAW类型转换为FLOAT4类型。

  DBE_RAW.CAST_FROM_RAW_TO_BINARY_FLOAT函数原型为：

  ```
  DBE_RAW.CAST_FROM_RAW_TO_BINARY_FLOAT(
    r         IN RAW,
    endianess IN INTEGER DEFAULT 1)
  RETURN FLOAT4;
  ```

  表 10-200 DBE_RAW.CAST_FROM_RAW_TO_BINARY_FLOAT 接口参数说明

参数	描述
r	需要进行转换的RAW类型值，最短长度不低于4，最大长度不超过32767。

参数	描述
endianess	字节序，取值范围：整型值1、2或3（1表示BIG_ENDIAN，2表示LITTLE_ENDIAN，3表示MACHINE_ENDIAN，默认值为BIG_ENDIAN，如果机器默认为BIG_ENDIAN，则函数执行结果MACHINE_ENDIAN和BIG_ENDIAN一致，如果机器默认为LITTLE_ENDIAN，则函数执行结果MACHINE_ENDIAN和LITTLE_ENDIAN一致）。

- DBE_RAW.CAST_FROM_BINARY_FLOAT_TO_RAW

  函数CAST_FROM_BINARY_FLOAT_TO_RAW将FLOAT4类型转换为RAW类型。

  DBE_RAW.CAST_FROM_BINARY_FLOAT_TO_RAW函数原型为：

  ```
  DBE_RAW.CAST_FROM_BINARY_FLOAT_TO_RAW(
     n         IN FLOAT4,
     endianess IN INTEGER DEFAULT 1)
  RETURN RAW;
  ```

  表 10-201 DBE_RAW.CAST_FROM_BINARY_FLOAT_TO_RAW 接口参数说明

参数	描述
n	需要进行转换的FLOAT4类型值。
endianess	字节序，取值范围：整型值1、2或3（1表示BIG_ENDIAN，2表示LITTLE_ENDIAN，3表示MACHINE_ENDIAN，默认值为BIG_ENDIAN，如果机器默认为BIG_ENDIAN，则函数执行结果MACHINE_ENDIAN和BIG_ENDIAN一致，如果机器默认为LITTLE_ENDIAN，则函数执行结果MACHINE_ENDIAN和LITTLE_ENDIAN一致）。

- DBE_RAW.CAST_FROM_RAW_TO_NUMBER

  函数CAST_FROM_RAW_TO_NUMBER将RAW类型转换为NUMERIC类型。

  说明

  由于A数据库与GaussDB的NUMBER数据类型底层实现不同，而RAW类型为数据的底层实现的二进制流的十六进制表示，故该函数在A数据库与GaussDB的表现不同。用户不能将A数据库中的一个NUMBER类型数据对应的RAW类型在GaussDB中得到相同的NUMBER类型数据，在GaussDB中的表现可参考示例。

  DBE_RAW.CAST_FROM_RAW_TO_NUMBER函数原型为：

  ```
  DBE_RAW.CAST_FROM_RAW_TO_NUMBER(
     r IN RAW)
  RETURN NUMERIC;
  ```

  表 10-202 DBE_RAW.CAST_FROM_RAW_TO_NUMBER 接口参数说明

参数	描述
r	需要进行转换的RAW类型，最短长度不低于6，最大长度不超过32767。

- DBE_RAW.CAST_FROM_NUMBER_TO_RAW

函数CAST_FROM_NUMBER_TO_RAW将NUMERIC类型转换为RAW类型。

说明

由于A数据库与GaussDB的NUMBER数据类型底层实现不同，而RAW类型为数据的底层实现的二进制流的十六进制表示，故该函数在A数据库与GaussDB的表现不同。用户不能将A数据库的NUMBER类型数据的RAW类型表示在GaussDB数据库中还原成A数据库原NUMBER类型数值，该函数在GaussDB中的表现可参考示例。

DBE_RAW.CAST_FROM_NUMBER_TO_RAW函数原型为：

```
DBE_RAW.CAST_FROM_NUMBER_TO_RAW(
    n IN NUMERIC)
RETURN RAW;
```

表 10-203 DBE_RAW.CAST_FROM_NUMBER_TO_RAW 接口参数说明

参数	描述
n	需要进行转换的NUMERIC类型。

- DBE_RAW.CAST_FROM_RAW_TO_NVARCHAR2

  函数CAST_FROM_RAW_TO_NVARCHAR2将RAW类型转换为NVARCHAR2类型。

  DBE_RAW.CAST_FROM_RAW_TO_NVARCHAR2函数原型为：

  ```
  DBE_RAW.CAST_FROM_RAW_TO_NVARCHAR2(
      r IN RAW)
  RETURN NVARCHAR2;
  ```

  表 10-204 DBE_RAW.CAST_FROM_RAW_TO_NVARCHAR2 接口参数说明

参数	描述
r	需要进行转换的RAW类型，最大长度不超过32767。

- DBE_RAW.COMPARE

  函数COMPARE返回两个RAW类型值首个不相同的位置。

  DBE_RAW.COMPARE函数原型为：

  ```
  DBE_RAW.COMPARE(
      r1  IN RAW,
      r2  IN RAW,
      pad IN RAW DEFAULT NULL)
  RETURN INTEGER;
  ```

  表 10-205 DBE_RAW.COMPARE 接口参数说明

参数	描述
r1	要比较的第一个数据，可以为NULL或长度为0，最大长度不超过32767。
r2	要比较的第二个数据，可以为NULL或长度为0，最大长度不超过32767。

参数	描述
pad	取pad的首字节，用于扩展r1或r2中较短的一个，最大长度不超过32767。当该参数为NULL、长度为0或缺省时，扩展值为0x00。

- DBE_RAW.CONCAT

  函数CONCAT将最多12个RAW类型值连接到一个新的RAW类型值中，并返回。如果连接后的长度超过32767，则会报错。

  DBE_RAW.CONCAT函数原型为：

```
DBE_RAW.CONCAT(
   r1  IN RAW DEFAULT NULL,
   r2  IN RAW DEFAULT NULL,
   r3  IN RAW DEFAULT NULL,
   r4  IN RAW DEFAULT NULL,
   r5  IN RAW DEFAULT NULL,
   r6  IN RAW DEFAULT NULL,
   r7  IN RAW DEFAULT NULL,
   r8  IN RAW DEFAULT NULL,
   r9  IN RAW DEFAULT NULL,
   r10 IN RAW DEFAULT NULL,
   r11 IN RAW DEFAULT NULL,
   r12 IN RAW DEFAULT NULL)
RETURN RAW;
```

  表 10-206 DBE_RAW.CONCAT 接口参数说明

参数	描述
r1...r12	需要进行连接的RAW类型值。

- DBE_RAW.CONVERT

  函数CONVERT将一个RAW类型值从源编码方式from_charset转换为目标编码方式to_charset。

  如果源编码格式到目标编码格式的转化规则不存在，则参数r不进行任何转换直接返回，如GBK和LATIN1之间的转换规则是不存在的，具体转换规则可以通过查看系统表pg_conversion获得。

  DBE_RAW.CONVERT函数原型为：

```
DBE_RAW.CONVERT(
   r            IN RAW,
   to_charset   IN VARCHAR2,
   from_charset IN VARCHAR2)
RETURN RAW;
```

  表 10-207 DBE_RAW.CONVERT 接口参数说明

参数	描述
r	需要进行转换的RAW类型，最大长度不超过32767。
to_charset	目标编码字符集名称。
from_charset	源编码字符集名称，在该编码下，r必须是合法的。

- DBE_RAW.COPIES

  函数COPIES将一个RAW类型值复制n次，串联在一起，并返回连接后的结果。如果复制后的长度超过32767，则会报错。

  DBE_RAW.COPIES函数原型为：

```
DBE_RAW.COPIES(
    r IN RAW,
    n IN NUMERIC)
RETURN RAW;
```

  表 10-208 DBE_RAW.COPIES 接口参数说明

参数	描述
r	需要进行复制的RAW类型值。
n	复制的次数（必须为正数）。

- DBE_RAW.OVERLAY

  函数OVERLAY用一个RAW类型数据对另一个RAW类型数据进行覆盖，可以指定起始覆盖位置和覆盖长度。

  DBE_RAW.OVERLAY函数原型为：

```
DBE_RAW.OVERLAY(
    overlay_str IN RAW,
    target      IN RAW,
    pos         IN BINARY_INTEGER DEFAULT 1,
    len         IN BINARY_INTEGER DEFAULT NULL,
    pad         IN RAW            DEFAULT NULL)
RETURN RAW;
```

  表 10-209 DBE_RAW.OVERLAY 接口参数说明

参数	描述
overlay_str	用于覆盖的字节，不能为NULL。
target	被覆盖的源字节串，RAW类型，最大长度不超过32767字节，不能为NULL。
pos	从第几个字节开始覆盖，第一个字节位置为1，限制pos>=1且len+pos<=32767，缺省值为1。
len	要覆盖的长度，限制len>=0且len+pos<=32767，缺省值为覆盖字节（overlay_str）的长度。
pad	填充字节，只有第一个字节有效，缺省值为NULL，为NULL时默认为0x00。

- DBE_RAW.REVERSE

  函数REVERSE将RAW类型数据按字节进行翻转。

  DBE_RAW.REVERSE函数原型为：

```
DBE_RAW.REVERSE(
    r IN RAW
)
RETURN RAW;
```


表 10-210 DBE_RAW.REVERSE 接口参数说明

参数	描述
r	需要进行翻转的RAW类型值，最大长度不超过32767字节，如果为NULL则返回NULL。

- DBE_RAW.TRANSLATE

  函数TRANSLATE将转换或舍弃RAW类型数据的指定字节。

  DBE_RAW.TRANSLATE函数原型为：

```
DBE_RAW.TRANSLATE(
   r        IN RAW,
   from_set IN RAW,
   to_set   IN RAW)
RETURN RAW;
```

表 10-211 DBE_RAW.TRANSLATE 接口参数说明

参数	描述
r	要转换的源字节串，RAW类型，最大长度不超过32767字节，不能为NULL。
from_set	源字节串中要转换的字节码，RAW类型，不能为NULL。源字节串中存在于from_set中的字节都将被替换成to_set中对应位置的字节，假如from_set中有多个相同的字节，只会替换成第一个该字节对应的字节。比如：r[x]=from_set[n]，那么r[x]将被替换成to_set[n]；如果from_set[n]对应的to_set[n]不存在（也就是to_set字节数不超过n）时，r[x]将被舍弃。
to_set	对应from_set字节转换为的字节码，RAW类型，不能为NULL。

- DBE_RAW.TRANSLITERATE

  函数TRANSLITERATE将RAW类型转换为NUMERIC类型。

  DBE_RAW.TRANSLITERATE函数原型为：

```
DBE_RAW.TRANSLITERATE(
   r        IN RAW,
   from_set IN RAW DEFAULT NULL,
   to_set   IN RAW DEFAULT NULL,
   pad      IN RAW DEFAULT NULL)
RETURN RAW;
```

表 10-212 DBE_RAW.TRANSLITERATE 接口参数说明

参数	描述
r	要转换的源字节串，RAW类型，最大长度不超过32767字节，不能为NULL。
to_set	对应from_set字节转换为的字节码，RAW类型，缺省值为NULL。如果为NULL，则r中的所有存在于from_set中的字节都将被转换成pad。

参数	描述
from_set	源字节串r中要转换的字节码，RAW类型，缺省值为NULL。如果from_set为NULL，则源字节串r的所有字节将被转换成由pad填充的等长数据。否则源字节串r中存在于from_set中的字节都将被替换成to_set中对应位置的字节，比如r[x]=from_set[n]，那么r[x]将被转换成to_set[n]，如果to_set[n]不存在，r[x]将被转换成pad。
pad	填充字节，只有第一个字节有效，缺省值为NULL，为NULL时默认为0x00。

- DBE_RAW.XRANGE

  函数XRANGE返回包含以指定字节码开始和结束的连续一字节编码的RAW值。

  DBE_RAW.XRANGE函数原型为：

```
DBE_RAW.XRANGE(
   start_byte IN RAW,
   end_byte   IN RAW)
RETURN RAW;
```

表 10-213 DBE_RAW.XRANGE 接口参数说明

参数	描述
start_byte	开始字节，只有第一个字节有效，如果为NULL则默认为0x00。
end_byte	结束字节，只有第一个字节有效，如果为NULL则默认为0xFF。如果end_byte<start_byte，则会从end_byte拼接到0XFF，再从0x00拼接到start_byte。

权限介绍

DBE_RAW模式属于系统目录，DBE_RAW模式除了系统管理员用户在初始化数据库时拥有创建权限，模式及模式下的所有接口，所有用户仅具有使用权限。

示例

```
DECLARE
v_raw RAW;
v_int INTEGER;
v_length INTEGER;
v_str VARCHAR2;
v_double BINARY_DOUBLE;
v_float FLOAT4;
v_numeric NUMERIC;
v_nvarchar2 NVARCHAR2;
BEGIN
-- INTEGER类型值转RAW类型
SELECT DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW(170,1) INTO v_raw; -- 000000AA
SELECT DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW(170,2) INTO v_raw; -- AA000000
SELECT DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW(170,3) INTO v_raw; -- AA000000
-- RAW类型值转INTEGER类型
SELECT
DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER(DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW(170,
1),1) INTO v_int; -- 170
SELECT
```



```
DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER(DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW(170,
2),2) INTO v_int; -- 170
SELECT
DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER(DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW(170,
3),3) INTO v_int; -- 170
-- 求RAW类型值长度
SELECT DBE_RAW.GET_LENGTH(DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW(170,1)) INTO v_length;
-- 4
-- VARCHAR2类型值转RAW类型
SELECT DBE_RAW.CAST_FROM_VARCHAR2_TO_RAW('AA') INTO v_raw; -- 4141
-- RAW类型值转VARCHAR2类型
SELECT DBE_RAW.CAST_TO_VARCHAR2('4141') INTO v_str; -- AA
-- RAW类型按位或
SELECT DBE_RAW.BIT_OR('0000', '1111') INTO v_raw; -- 1111
-- RAW类型取字串
SELECT DBE_RAW.SUBSTR('ABCD', 1, 2) INTO v_raw; -- ABCD
-- RAW类型按位与
SELECT DBE_RAW.BIT_AND('AFF', 'FF0B') INTO v_raw; -- 0A0B
-- RAW类型按位取反
SELECT DBE_RAW.BIT_COMPLEMENT('0AFF') INTO v_raw; -- F500
-- RAW类型按位异或
SELECT DBE_RAW.BIT_XOR('AFF', 'FF0B') INTO v_raw; -- F5F4
-- BINARY_DOUBLE类型值转RAW类型
SELECT DBE_RAW.CAST_FROM_BINARY_DOUBLE_TO_RAW(1.0001,1) INTO v_raw; -- 3FF00068DB8BAC71
-- RAW类型值转BINARY_DOUBLE类型
SELECT DBE_RAW.CAST_FROM_RAW_TO_BINARY_DOUBLE('3FF00068DB8BAC7',1) INTO v_double; -- 1.0001
-- RAW类型转FLOAT4类型
SELECT DBE_RAW.CAST_FROM_RAW_TO_BINARY_FLOAT('40200000',1) INTO v_float; -- 2.5
-- FLOAT4类型转RAW类型
SELECT DBE_RAW.CAST_FROM_BINARY_FLOAT_TO_RAW('2.5',1) INTO v_raw; -- 40200000
-- RAW类型转NUMERIC类型
SELECT DBE_RAW.CAST_FROM_RAW_TO_NUMBER('808002008813') INTO v_numeric; -- 2.5
-- NUMERIC类型转RAW类型
SELECT DBE_RAW.CAST_FROM_NUMBER_TO_RAW('2.5') INTO v_raw; -- 808002008813
-- RAW类型转NVARCHAR2类型
SELECT DBE_RAW.CAST_FROM_RAW_TO_NVARCHAR2('12345678') INTO v_nvarchar2; -- \x124Vx
-- RAW类型COMPARE
SELECT DBE_RAW.COMPARE('ABCD','AB') INTO v_numeric; -- 2
-- RAW类型CONCAT
SELECT DBE_RAW.CONCAT('ABCD','AB') INTO v_raw; -- ABCDAB
-- RAW类型CONVERT
SELECT DBE_RAW.CONVERT('E695B0', 'GBK','UTF8') INTO v_raw; -- CAFD
-- RAW类型COPIES
SELECT DBE_RAW.COPIES('ABCD',2) INTO v_raw; -- ABCDABCD
-- RAW类型指定位置和长度进行覆盖
SELECT DBE_RAW.OVERLAY('abcef', '12345678123456', 2, 5, '9966') INTO v_raw; -- 120ABCEF999956
-- RAW类型按字节翻转
SELECT DBE_RAW.REVERSE('12345678') INTO v_raw; -- 78563412
-- RAW类型字节转换（无填充码）
SELECT DBE_RAW.TRANSLATE('1122112233', '1133','55') INTO v_raw; -- 55225522
-- RAW类型字节转换（有填充码）
SELECT DBE_RAW.TRANSLITERATE('1122112233', '55','1133','FFEE') INTO v_raw; -- 55225522FF
-- RAW类型两个字节间的所有字节
SELECT DBE_RAW.XRANGE('00','03') INTO v_raw; -- 00010203
END;
/
ANONYMOUS BLOCK EXECUTE
```

10.12.2.5 DBE_TASK

接口介绍

高级功能包DBE_TASK支持的所有接口请参见**表 DBE_TASK**。

表 10-214 DBE_TASK

接口名称	描述
DBE_TASK.SUBMIT	提交一个定时任务。作业号由系统自动生成。
DBE_TASK.JOB_SUBMIT	同DBE_TASK.SUBMIT。但提供语法兼容参数。
DBE_TASK.ID_SUBMIT	提交一个定时任务。作业号由用户指定。
DBE_TASK.CANCEL	通过作业号来删除定时任务。
DBE_TASK.RUN	运行定时任务。
DBE_TASK.FINISH	禁用或者启用定时任务。
DBE_TASK.UPDATE	修改定时任务的属性，包括任务内容、下次执行时间、执行间隔。
DBE_TASK.CHANGE	同DBE_TASK.UPDATE。但提供语法兼容参数。
DBE_TASK.CONTENT	修改定时任务的内容属性。
DBE_TASK.NEXT_TIME	修改定时任务的下次执行时间属性。
DBE_TASK.INTERVAL	修改定时任务的执行间隔属性。

- DBE_TASK.SUBMIT

  存储过程SUBMIT提交一个系统提供的定时任务。

  DBE_TASK.SUBMIT函数原型为：

  ```
  DBE_TASK.SUBMIT(
  what          IN   TEXT,
  next_time     IN   TIMESTAMP DEFAULT sysdate,
  interval_time IN   TEXT  DEFAULT 'null',
  id            OUT  INTEGER
  )RETURN INTEGER;
  ```

  说明

  当创建一个定时任务（DBE_TASK）时，系统默认将当前数据库和用户名与当前创建的定时任务（DBE_TASK）绑定起来。该接口函数可以通过call或select调用，如果通过select调用，可以不填写出参，如果通过call调用，需要填写出参。如果在存储过程中则需要用通过perform调用该接口函数。如果提交的sql语句任务使用到非public的schema，应该指定表或者函数的schema，或者在sql语句前添加set current_schema = xxx;语句。

表 10-215 DBE_TASK.SUBMIT 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
what	text	IN	否	要执行的SQL语句。支持一个或多个‘DDL’(不支持DB相关操作)，‘DML’，‘匿名块’，‘调用存储过程的语句’或4种混合的场景。
next_time	timestamp	IN	否	下次作业运行时间。默认值为当前系统时间（sysdate）。如果是过去时间，在提交作业时表示立即执行。
interval_time	text	IN	是	用来计算下次作业运行时间的时间表达式，可以是interval表达式，也可以是sysdate加上一个numeric值（例如：sysdate+1.0/24）。如果为空值或字符串"null"表示只执行一次，执行后JOB状态STATUS变成'd'不再执行。
id	integer	OUT	否	作业号。范围为1～32767。当使用select调用时，该参数不能添加，当使用call调用时，该参数必须添加。

须知

当在TASK的参数what中创建用户时，日志会记录密码的明文。因此不建议在TASK任务中创建用户。

示例：

```
gaussdb=# SELECT DBE_TASK.SUBMIT('call pro_xxx();', to_date('20180101','yyyymmdd'),'sysdate+1');
 submit
--------
  31031
(1 row)

gaussdb=# SELECT DBE_TASK.SUBMIT('call pro_xxx();', to_date('20180101','yyyymmdd'),'sysdate
+1.0/24');
 submit
--------
    512
(1 row)

gaussdb=# DECLARE
gaussdb-#     jobid int;
gaussdb-# BEGIN
gaussdb$#     PERFORM DBE_TASK.SUBMIT('call pro_xxx();', sysdate, 'interval ''5 minute''', jobid);
gaussdb$# END;
gaussdb$# /
ANONYMOUS BLOCK EXECUTE
```

- DBE_TASK.JOB_SUBMIT

  存储过程SUBMIT提交一个系统提供的定时任务。并提供了额外的兼容性参数。

  DBE_TASK.JOB_SUBMIT函数原型为：

```
DBE_TASK.JOB_SUBMIT(
job           OUT  INTEGER,
what           IN   TEXT,
next_date      IN   TIMESTAMP  DEFAULT sysdate,
job_interval  IN   TEXT       DEFAULT 'null',
no_parse       IN   BOOLEAN    DEFAULT false,
instance       IN   INTEGER    DEFAULT 0,
force          IN   BOOLEAN    DEFAULT false
)RETURN INTEGER;
```

表 10-216 DBE_TASK.JOB_SUBMIT 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
job	integer	OUT	否	作业号。范围为1～32767。当使用select调用dbe.job_submit时，该参数可以省略。
what	text	IN	否	要执行的SQL语句。支持一个或多个‘DDL’(不支持DB相关操作)，‘DML’，‘匿名块’，‘调用存储过程的语句’或4种混合的场景。
next_date	timestamp	IN	是	下次作业运行时间。默认值为当前系统时间（sysdate）。如果是过去时间，在提交作业时表示立即执行。
job_interval	text	IN	是	用来计算下次作业运行时间的时间表达式，可以是interval表达式，也可以是sysdate加上一个numeric值（例如：sysdate+1.0/24）。如果为空值或字符串"null"表示只执行一次，执行后JOB状态STATUS变成'd'不再执行。
no_parse	boolean	IN	是	默认值false，仅用于语法上的兼容。
instance	integer	IN	是	默认值0，仅用于语法上的兼容。
force	boolean	IN	是	默认值false，仅用于语法上的兼容。

示例：

```
gaussdb=# DECLARE
gaussdb-#     id integer;
gaussdb-# BEGIN
gaussdb$#     id = DBE_TASK.JOB_SUBMIT(
gaussdb$#         what => 'insert into t1 values (1, 2)',
gaussdb$#         job_interval => 'sysdate + 1' --daily
gaussdb$#     );
gaussdb$# END;
gaussdb$# /
ANONYMOUS BLOCK EXECUTE
```

- DBE_TASK.ID_SUBMIT

ID_SUBMIT与SUBMIT语法功能相同，但其第一个参数是入参，即指定的作业号，SUBMIT最后一个参数是出参，表示系统自动生成的作业号。

```
DBE_TASK.ID_SUBMIT(
id            IN   BIGINT,
what          IN   TEXT,
next_time     IN   TIMESTAMP DEFAULT sysdate,
interval_time IN   TEXT  DEFAULT 'null');
```

示例：

```
gaussdb=# CALL dbe_task.id_submit(101, 'insert_msg_statistic1;', sysdate, 'sysdate+3.0/24');
 id_submit
-----------


(1 row)
```

- DBE_TASK.CANCEL

  存储过程CANCEL删除指定的定时任务。

  DBE_TASK.CANCEL函数原型为：

  ```
  CANCEL(id  IN  INTEGER);
  ```

  表 10-217 DBE_TASK.CANCEL 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	integer	IN	否	指定的作业号。

  示例：

  ```
  gaussdb=# CALL dbe_task.cancel(101);
   cancel
  --------


  (1 row)
  ```

- DBE_TASK.RUN

  存储过程RUN运行定时任务。

  DBE_TASK.RUN函数原型为：

  ```
  DBE_TASK.RUN(
  job          IN   BIGINT,
  force        IN   BOOLEAN DEFAULT FALSE);
  ```

  表 10-218 DBE_TASK.RUN 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
job	bigint	IN	否	指定的作业号。
force	boolean	IN	是	仅用于语法上的兼容。

示例：

```
gaussdb=# BEGIN
gaussdb$#     DBE_TASK.ID_SUBMIT(12345, 'insert_msg_statistic1;', sysdate, 'sysdate+3.0/24');
gaussdb$#     DBE_TASK.RUN(12345);
gaussdb$# END;
gaussdb$# /
ANONYMOUS BLOCK EXECUTE
```

- DBE_TASK.FINISH

  存储过程FINISH禁用或者启用定时任务。

  DBE_TASK.FINISH函数原型为：

```
DBE_TASK.FINISH(
id          IN   INTEGER,
broken      IN   BOOLEAN,
next_time   IN   TIMESTAMP  DEFAULT  sysdate);
```

表 10-219 DBE_TASK.FINISH 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	integer	IN	否	指定的作业号。
broken	boolean	IN	否	状态标志位，true代表禁用，false代表启用。具体true或false值更新当前job；如果为空值，则不改变原有job的状态。
next_time	timestamp	IN	是	下次运行时间，默认为当前系统时间。如果参数broken状态为true，则更新该参数为'4000-1-1'；如果参数broken状态为false，且如果参数next_time不为空值，则更新指定job的next_time值，如果next_time为空值，则不更新next_time值。该参数可以省略，为默认值。

示例：

```
gaussdb=# CALL dbe_task.id_submit(101, 'insert_msg_statistic1;', sysdate, 'sysdate+3.0/24');
 id_submit
-----------

(1 row)

gaussdb=# CALL dbe_task.finish(101, true);
 finish
--------

(1 row)

gaussdb=# CALL dbe_task.finish(101, false, sysdate);
 finish
--------

(1 row)
```


- DBE_TASK.UPDATE

  存储过程UPDATE修改定时任务的属性，包括任务内容、下次执行时间、执行间隔。

  DBE_TASK.UPDATE函数原型为：

```
DBE_TASK.UPDATE(
id             IN     INTEGER,
content        IN     TEXT,
next_time      IN     TIMESTAMP,
interval_time  IN     TEXT);
```

  表 10-220 DBE_TASK.UPDATE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	integer	IN	否	指定的作业号。
content	text	IN	是	执行的存储过程名或者sql语句块。如果该参数为空值，则不更新指定job的content值，否则更新指定job的content值。
next_time	timestamp	IN	是	下次运行时间。如果该参数为空值，则不更新指定job的next_time值，否则更新指定job的next_time值。
interval_time	text	IN	是	用来计算下次作业运行时间的时间表达式。如果该参数为空值，则不更新指定job的interval_time值；如果该参数不为空值，会校验interval_time是否为有效的时间类型或interval类型，则更新指定job的interval_time值。如果为字符串"null"表示只执行一次，执行后JOB状态STATUS变成'd' 不再执行。

  示例：

```
gaussdb=# CALL dbe_task.update(101, 'call userproc();', sysdate, 'sysdate + 1.0/1440');
 update
--------


(1 row)

gaussdb=# CALL dbe_task.update(101, 'insert into tbl_a values(sysdate);', sysdate, 'sysdate +
1.0/1440');
 update
--------


(1 row)
```

- DBE_TASK.CHANGE

  存储过程CHANGE修改定时任务的属性，包括任务内容、下次执行时间、执行间隔。

  DBE_TASK.CHANGE函数原型为：

```
DBE_TASK.CHANGE(
job            IN     INTEGER,
```



```
what          IN    TEXT       DEFAULT NULL,
next_date     IN    TIMESTAMP  DEFAULT NULL,
job_interval  IN    TEXT       DEFAULT NULL,
instance      IN    INTEGER    DEFAULT NULL,
force         IN    BOOLEAN    DEFAULT false);
```

表 10-221 DBE_TASK.CHANGE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
job	integer	IN	否	指定的作业号。
what	text	IN	是	执行的存储过程名或者sql语句块。如果该参数为空值，则不更新指定job的what值，否则更新指定job的what值。
next_date	timestamp	IN	是	下次运行时间。如果该参数为空值，则不更新指定job的next_date值，否则更新指定job的next_date值。
job_interval	text	IN	是	用来计算下次作业运行时间的时间表达式。如果该参数为空值，则不更新指定job的job_interval值；如果该参数不为空值，会校验job_interval是否为有效的时间类型或interval类型，则更新指定job的job_interval值。如果为字符串"null"表示只执行一次，执行后JOB状态STATUS变成'd' 不再执行。
instance	integer	IN	是	仅用于语法上的兼容。
force	boolean	IN	否	仅用于语法上的兼容。

示例：

```
gaussdb=# BEGIN
gaussdb$#     DBE_TASK.CHANGE(
gaussdb$#         job => 101,
gaussdb$#         what => 'insert into t2 values (2);'
gaussdb$#     );
gaussdb$# END;
gaussdb$# /
ANONYMOUS BLOCK EXECUTE
```

- DBE_TASK.CONTENT

  存储过程CONTENT修改定时任务的任务内容属性。

  DBE_TASK.CONTENT函数原型为：

```
DBE_TASK.CONTENT(
id          IN     INTEGER,
content     IN TEXT);
```

表 10-222 DBE_TASK.CONTENT 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	integer	IN	否	指定的作业号。
content	text	IN	否	执行的存储过程调用或者sql语句块或者程序块。

说明

- 当content参数是一个或多个可以执行成功的sql语句/程序块/调用存储过程时，该接口函数才能被执行成功，否则会执行失败。
- 若content参数为一个简单的insert、update等语句，需要在表前加模式名。

示例：

```
gaussdb=# CALL dbe_task.content(101, 'call userproc();');
 content
---------

(1 row)

gaussdb=# CALL dbe_task.content(101, 'insert into tbl_a values(sysdate);');
 content
---------

(1 row)
```

- DBE_TASK.NEXT_TIME

  存储过程NEXT_TIME修改定时任务的下次执行时间属性。

  DBE_TASK.NEXT_TIME函数原型为：

  ```
  DBE_TASK.NEXT_TIME(
  id         IN    BIGINT,
  next_time  IN    TEXT);
  ```

表 10-223 DBE_TASK.NEXT_TIME 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	bigint	IN	否	指定的作业号。
next_time	text	IN	否	下次运行时间。

说明

如果输入的next_time的值小于当前日期值，该job会立即执行一次。

示例：

```
gaussdb=# CALL dbe_task.next_time(101, sysdate);
 next_time
-----------
```



```
(1 row)
```

- DBE_TASK.INTERVAL

  存储过程INTERVAL修改定时任务的执行间隔属性。

  DBE_TASK.INTERVAL函数原型为：

```
DBE_TASK.INTERVAL(
id             IN      INTEGER,
interval_time      IN    TEXT);
```

表 10-224 DBE_TASK.INTERVAL 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	integer	IN	否	指定的作业号。
interval_time	text	IN	是	用来计算下次作业运行时间的时间表达式。如果为空值或字符串"null"表示只执行一次，执行后JOB状态STATUS变成'd' 不再执行。作业间隔需要为有效的时间类型或interval类型。

示例：

```
gaussdb=# CALL dbe_task.interval(101, 'sysdate + 1.0/1440');
 interval
----------

(1 row)

gaussdb=# CALL dbe_task.cancel(101);
 cancel
--------

(1 row)
```

说明

对于指定job正在运行状态（即job_status为'r'）时，不允许通过cancel、update、next_time、content、interval等接口删除或修改job的参数信息。

约束说明

1. job只能通过dbe_task高级包提供的接口进行创建、更新、删除操作，因为高级包的接口中会考虑所有数据库主节点间job信息的同步和pg_job与pg_job_proc表主键的关联操作，如果通过DML语句对pg_job表进行增删改，会导致job信息在数据库主节点间不一致和系统表无法关联变更的混乱问题，会严重影响job内部的管理。
2. 由于用户创建的每个任务和数据库主节点绑定，当任务运行过程中，该数据库主节点故障，则该任务的状态无法实时刷新，仍为’r’状态，需要等数据库主节点启动正常后才能刷新为’s’状态。如果在任务未执行时数据库主节点故障，则该数据库主节点上的任务都得不到正常的调度和执行，需要人为干预让该数据库主节点恢复正常，或进行节点删除/替换、job才能正常的调度和执行。

3. job在定时执行过程中，需要在当前job所属的数据库主节点上实时更新该job的运行状态、最近执行开始时间、最近执行结束时间、下次开始时间、失败次数（如果job执行失败）等相关参数信息到pg_job系统表中，并同步到其他数据库主节点，保证job信息的一致性。如果其他数据库主节点存在节点故障，那么job所属数据库主节点会同步超时重发的处理，导致job执行时间变长，但数据库主节点间同步超时失败后，原数据库主节点上pg_job表中job的相关信息仍然能正常更新，且job能正常执行成功。当故障数据库主节点恢复正常后，可能出现该数据库主节点上pg_job表中当前job的执行时间、运行状态等参数与原数据库主节点上不一致的情况，需要原数据库主节点上再次执行该job后才能保证job信息的同步。
4. 对于并发同时有多个job到达执行时间的场景，由于会为每个job创建一个线程来执行job，由于系统内部启动每个线程的时间会有延迟，因此会导致同时并发执行的job的开始时间有延迟，每个job的延迟时间在0.1ms左右。

10.12.2.6 DBE_SCHEDULER

接口介绍

高级功能包DBE_SCHEDULER支持通过调度（schedule）和程序（program）更加灵活的创建定时任务。支持的所有接口请见**表10-225**。

表 10-225 DBE_SCHEDULER

接口名称	描述
DBE_SCHEDULER.CREATE_JOB	创建定时任务。
DBE_SCHEDULER.DROP_JOB	删除定时任务。
DROP_SINGLE_JOB	删除单个定时任务。
DBE_SCHEDULER.SET_ATTRIBUTE	设置对象属性。
DBE_SCHEDULER.RUN_JOB	运行定时任务。
DBE_SCHEDULER.RUN_BACKEND_JOB	后台运行定时任务。
DBE_SCHEDULER.RUN_FOREGROUND_JOB	前台运行定时任务。
DBE_SCHEDULER.STOP_JOB	停止定时任务。
DBE_SCHEDULER.STOP_SINGLE_JOB	停止单个定时任务。
DBE_SCHEDULER.GENERATE_JOB_NAME	生成定时任务名。
DBE_SCHEDULER.CREATE_PROGRAM	创建程序。
DBE_SCHEDULER.DEFINE_PROGRAM_ARGUMENT	定义程序参数。

接口名称	描述
DBE_SCHEDULER.DROP_PROGRAM	删除程序。
DBE_SCHEDULER.DROP_SINGLE_PROGRAM	删除单个程序。
DBE_SCHEDULER.SET_JOB_ARGUMENT_VALUE	设置定时任务参数值。
DBE_SCHEDULER.CREATE_SCHEDULE	创建调度。
DBE_SCHEDULER.DROP_SCHEDULE	删除调度。
DBE_SCHEDULER.DROP_SINGLE_SCHEDULE	删除单个调度。
DBE_SCHEDULER.CREATE_JOB_CLASS	创建定时任务类。
DBE_SCHEDULER.DROP_JOB_CLASS	删除定时任务类。
DBE_SCHEDULER.DROP_SINGLE_JOB_CLASS	删除单个定时任务类。
DBE_SCHEDULER.GRANT_USER_AUTHORIZATION	赋予用户特殊权限。
DBE_SCHEDULER.REVOKE_USER_AUTHORIZATION	撤销用户特殊权限。
DBE_SCHEDULER.CREATE_CREDENTIAL	创建证书。
DBE_SCHEDULER.DROP_CREDENTIAL	销毁证书。
DBE_SCHEDULER.ENABLE	启用对象。
DBE_SCHEDULER.ENABLE_SINGLE	启动单个对象。
DBE_SCHEDULER.DISABLE	停用对象。
DBE_SCHEDULER.DISABLE_SINGLE	停用单个对象。
DBE_SCHEDULER.EVAL_CALENDAR_STRING	分析Calendar格式字符串。
DBE_SCHEDULER.EVALUATE_C...	分析Calendar格式字符串。

- DBE_SCHEDULER.CREATE_JOB
  创建一个定时任务。
  DBE_SCHEDULER.CREATE_JOB函数原型可以分为4种：

```
-- 内联调度和程序的定时任务
DBE_SCHEDULER.CREATE_JOB(
job_name TEXT,
job_type TEXT,
job_action TEXT,
number_of_arguments INTEGER             DEFAULT 0,
start_date TIMESTAMP WITH TIME ZONE     DEFAULT NULL,
repeat_interval TEXT                    DEFAULT NULL,
end_date TIMESTAMP WITH TIME ZONE       DEFAULT NULL,
job_class TEXT            DEFAULT 'DEFAULT_JOB_CLASS',
enabled BOOLEAN                         DEFAULT FALSE,
auto_drop BOOLEAN                       DEFAULT TRUE,
comments TEXT                           DEFAULT NULL,
credential_name TEXT                    DEFAULT NULL,
destination_name TEXT                   DEFAULT NULL
)

-- 引用创建好的调度和程序的定时任务
DBE_SCHEDULER.CREATE_JOB(
job_name TEXT,
program_name TEXT,
schedule_name TEXT,
job_class TEXT            DEFAULT 'DEFAULT_JOB_CLASS',
enabled BOOLEAN                         DEFAULT FALSE,
auto_drop BOOLEAN                       DEFAULT TRUE,
comments TEXT                           DEFAULT NULL,
job_style TEXT                         DEFAULT 'REGULAR',
credential_name TEXT                    DEFAULT NULL,
destination_name TEXT                   DEFAULT NULL
)

-- 引用创建好的程序，内联调度的定时任务
DBE_SCHEDULER.CREATE_JOB(
job_name text,
program_name TEXT,
start_date TIMESTAMP WITH TIME ZONE     DEFAULT NULL,
repeat_interval TEXT                    DEFAULT NULL,
end_date TIMESTAMP WITH TIME ZONE       DEFAULT NULL,
job_class TEXT            DEFAULT 'DEFAULT_JOB_CLASS',
enabled BOOLEAN                         DEFAULT FALSE,
auto_drop BOOLEAN                       DEFAULT TRUE,
comments TEXT                           DEFAULT NULL,
job_style TEXT                         DEFAULT 'REGULAR',
credential_name TEXT                    DEFAULT NULL,
destination_name TEXT                   DEFAULT NULL
)

-- 引用创建好的调度，内联程序的定时任务
DBE_SCHEDULER.CREATE_JOB(
job_name TEXT,
schedule_name TEXT,
job_type TEXT,
job_action TEXT,
number_of_arguments INTEGER             DEFAULT 0,
job_class TEXT             DEFAULT 'DEFAULT_JOB_CLASS',
enabled BOOLEAN                         DEFAULT FALSE,
auto_drop BOOLEAN                       DEFAULT TRUE,
comments TEXT                           DEFAULT NULL,
credential_name TEXT                    DEFAULT NULL,
destination_name TEXT                   DEFAULT NULL
)
```


说明

利用DBE_SCHEDULER创建的定时任务不会与DBE_TASK中的定时任务相冲突。

DBE_SCHEDULER创建的定时任务会生成对应的job_id，但是在使用过程中这个id并没有实际意义。

表 10-226 DBE_SCHEDULER.CREATE_JOB 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
job_name	text	IN	否	定时任务名称。
job_type	text	IN	否	定时任务内联程序类型，可用类型为： ● 'PLSQL_BLOCK'：匿名存储过程快。 ● 'STORED_PROCEDURE'：保存的存储过程。 ● 'EXTERNAL_SCRIPT'：外部脚本。
job_action	text	IN	否	定时任务内联程序执行内容。
number_of_arguments	integer	IN	否	定时任务内联程序参数个数。
program_name	text	IN	否	定时任务引用程序名称。
start_date	timestamp with time zone	IN	是	定时任务内联调度起始时间。
repeat_interval	text	IN	是	定时任务内联调度任务周期。
end_date	timestamp with time zone	IN	是	定时任务内联调度失效时间。
schedule_name	text	IN	否	定时任务引用调度名称。
job_class	text	IN	否	定时任务类名。
enabled	boolean	IN	否	定时任务启用状态。
auto_drop	boolean	IN	否	定时任务自动删除。

参数	类型	入参/出参	是否可以为空	描述
comments	text	IN	是	备注。
job_style	text	IN	否	定时任务行为模式，仅支持‘REGULAR'。
credential_name	text	IN	是	定时任务证书名。
destination_name	text	IN	是	定时任务目标名。

示例：

```
gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'SELECT
pg_sleep(1);', 3, false, 'test');
 create_program
----------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule1', NULL, 'sysdate', NULL, 'test');
 create_schedule
-----------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.create_job(job_name=>'job1', program_name=>'program1',
schedule_name=>'schedule1');
 create_job
------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_schedule('schedule1');
 drop_schedule
---------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
 drop_program
--------------

(1 row)
```

须知

创建'EXTERNAL_SCRIPT'类型的定时任务需要管理员赋予相关的权限和证书，且需要数据库启动用户对该外部脚本有读取权限才可以正常生效。

- DBE_SCHEDULER.DROP_JOB

  删除定时任务。

  DBE_SCHEDULER.DROP_JOB函数原型为：

```
DBE_SCHEDULER.drop_job(
job_name text,
force boolean                    default false,
defer boolean                    default false,
commit_semantics text      default 'STOP_ON_FIRST_ERROR'
)
```

  说明

  DBE_SCHEDULER.DROP_JOB可以指定一个、多个任务，或指定任务类进行定时任务删除。

  表 10-227 DBE_SCHEDULER.DROP_JOB 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
job_name	text	IN	否	定时任务或定时任务类名称，可以指定一个或多个，当指定多个定时任务时需要利用逗号隔开
force	boolean	IN	否	删除定时任务行为标记位 true：尝试先停止当前定时任务，再进行删除 false：如果定时任务正在运行会删除失败
defer	boolean	IN	否	删除定时任务行为标记位 true：允许定时任务完成后再进行删除
commit_semantics	text	IN	否	提交规则： 'STOP_ON_FIRST_ERROR'：在第一个报错之前的删除操作会提交 'TRANSACTIONAL'：事务级提交，报错前的删除操作会回滚 'ABSORB_ERRORS'：尝试越过报错，将成功的删除操作提交

  示例：

```
gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'SELECT
pg_sleep(1);', 3, false, 'test');
 create_program
----------------


(1 row)

gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule1', NULL, 'sysdate', NULL, 'test');
 create_schedule
-----------------
```



```
(1 row)

gaussdb=# CALL DBE_SCHEDULER.create_job(job_name=>'job1', program_name=>'program1',
schedule_name=>'schedule1');
 create_job
------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_schedule('schedule1');
 drop_schedule
---------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
 drop_program
--------------

(1 row)
```

须知

commit_semantic中的'TRANSACTIONAL'选项必须在force为false的情况下才会生效。

- DBE_SCHEDULER.DROP_SINGLE_JOB

  删除一个定时任务。

  DBE_SCHEDULER.DROP_SINGLE_JOB函数原型为：

  ```
  DBE_SCHEDULER.drop_single_job(
  job_name text,
  force boolean                   default false,
  defer boolean                   default false
  )
  ```

- DBE_SCHEDULER.SET_ATTRIBUTE

  修改定时任务属性。

  DBE_SCHEDULER.SET_ATTRIBUTE函数4种原型为：

  ```
  DBE_SCHEDULER.set_attribute(
  name              text,
  attribute         text,
  value             boolean
  )

  DBE_SCHEDULER.set_attribute(
  name              text,
  attribute         text,
  value             text
  )

  DBE_SCHEDULER.set_attribute(
  name              text,
  attribute         text,
  value             timestamp
  )
  ```



```
DBE_SCHEDULER.set_attribute(
name                 text,
attribute            text,
value                timestamp with time zone
)

DBE_SCHEDULER.set_attribute(
name text,
attribute text,
value text,
value2 text                      default NULL
)
```

说明

name在这里可以指定任何DBE_SCHEDULER内部的对象。

表 10-228 DBE_SCHEDULER.SET_ATTRIBUTE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
name	text	IN	否	对象名。
attribute	text	IN	否	属性名。
value	boolean/date/timestamp/timestamp with time zone/text	IN	否	属性值，可选属性如下： • 定时任务相关：job_type, job_action, number_of_arguments, start_date, repeat_interval, end_date, job_class, enabled, auto_drop, comments, credential_name, destination_name, program_name, schedule_name, job_style。 • 程序相关：program_action, program_type, number_of_arguments, comments。 • 调度相关：start_date, repeat_interval, end_date, comments。
value2	text	IN	是	额外属性值。保留参数位，目前尚不支持拥有额外属性值的目标属性。

示例：

```
gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'SELECT
pg_sleep(1);', 3, false, 'test');
 create_program
----------------


(1 row)

gaussdb=# CALL DBE_SCHEDULER.set_attribute('program1', 'number_of_arguments', 0);
 set_attribute
---------------
```



```
(1 row)

gaussdb=# CALL DBE_SCHEDULER.set_attribute('program1', 'program_type', 'STORED_PROCEDURE');
 set_attribute
---------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
 drop_program
--------------

(1 row)
```

须知

不要使用DBE_SCHEDULER.SET_ATTRIBUTE来将参数置空。

对象名不能通过DBE_SCHEDULER.SET_ATTRIBUTE来更改。

- DBE_SCHEDULER.RUN_JOB

  运行定时任务。

  DBE_SCHEDULER.RUN_JOB函数原型为：

  ```
  DBE_SCHEDULER.run_job(
  job_name text,
  use_current_session boolean             default true
  )
  ```

  说明

  DBE_SCHEDULER.RUN_JOB主要用于立即运行定时作业，独立于定时任务本身的调度，甚至可以同时运行。

  表 10-229 DBE_SCHEDULER.RUN_JOB 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
job_name	text	IN	否	定时任务名称，可以指定一个或多个，当指定多个定时任务时需要利用逗号隔开
use_current_session	boolean	IN	否	运行定时任务标志位： true：使用当前会话运行，主要用于查看定时任务是否可以正常运行 false：后台拉起定时任务，运行结果会打印到日志中

  示例：

  ```
  gaussdb=# SELECT dbe_scheduler.create_job('job1','PLSQL_BLOCK','BEGIN INSERT INTO test1
  VALUES(12); end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
   create_job
  ------------
  ```



```
(1 row)

gaussdb=# CALL DBE_SCHEDULER.run_job('job1', false);
 run_job
---------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)
```

须知

use_current_session不能用于打印执行结果。

- DBE_SCHEDULER.RUN_BACKEND_JOB

  后台运行定时任务。

  DBE_SCHEDULER.RUN_BACKEND_JOB函数原型为：

  ```
  DBE_SCHEDULER.run_backend_job(
  job_name text
  )
  ```

  示例：

  ```
  gaussdb=# SELECT dbe_scheduler.create_job('job1','PLSQL_BLOCK','BEGIN INSERT INTO test1
  VALUES(12); end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
   create_job
  ------------

  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.run_backend_job('job1');
   run_backend_job
  -----------------

  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
   drop_job
  ----------

  (1 row)
  ```

- DBE_SCHEDULER.RUN_FOREGROUND_JOB

  当前会话运行定时任务。

  仅支持运行external 类型任务。

  返回值：text。

  DBE_SCHEDULER.RUN_FOREGROUND_JOB函数原型为：

  ```
  DBE_SCHEDULER.run_foreground_job(
  job_name text
  )return text
  ```

  示例：

  ```
  gaussdb=# CREATE USER test1 IDENTIFIED BY '*********';
  NOTICE:  The encrypted password contains MD5 ciphertext, which is not secure.
  CREATE ROLE
  gaussdb=# SELECT DBE_SCHEDULER.create_credential('cre_1', 'test1', '*********');
   create_credential
  ```



```
--------------------

(1 row)

gaussdb=# SELECT DBE_SCHEDULER.create_job(job_name=>'job1', job_type=>'EXTERNAL_SCRIPT',
job_action=>'/usr/bin/pwd', enabled=>true, auto_drop=>false, credential_name => 'cre_1');
 create_job
------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.run_foreground_job('job1');
  run_foreground_job
--------------------------------
 Host key verification failed.\r+

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_credential('cre_1', false);
 drop_credential
-----------------

(1 row)

gaussdb=# DROP USER test1;
DROP ROLE
```

- DBE_SCHEDULER.STOP_JOB

  终止定时任务。

  DBE_SCHEDULER.STOP_JOB函数原型为：

  ```
  DBE_SCHEDULER.stop_job(
  job_name text,
  force boolean                    default false,
  commit_semantics text                default 'STOP_ON_FIRST_ERROR'
  )
  ```

  表 10-230 DBE_SCHEDULER.STOP_JOB 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
job_name	text	IN	否	定时任务或定时任务类名称，可以指定一个或多个，当指定多个定时任务时需要利用逗号隔开。
force	boolean	IN	否	删除定时任务行为标记位： ● true：调度器会发送终止信号立即结束任务线程。 ● false：调度器会尝试利用打断信号终止定时任务线程。

参数	类型	入参/出参	是否可以为空	描述
commit_semantics	text	IN	否	提交规则： ● 'STOP_ON_FIRST_ERROR'：在第一个报错之前的打断操作会提交。 ● 'ABSORB_ERRORS'：尝试越过报错，将成功的打断操作提交。

示例：

```
gaussdb=# SELECT dbe_scheduler.create_job('job1','PLSQL_BLOCK','BEGIN INSERT INTO test1
VALUES(12); end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
 create_job
------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.stop_job('job1', true, 'STOP_ON_FIRST_ERROR');
 stop_job
----------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)
```

- DBE_SCHEDULER.STOP_SINGLE_JOB

  终止单个定时任务。

  DBE_SCHEDULER.STOP_SINGLE_JOB函数原型为：

  ```
  DBE_SCHEDULER.stop_single_job(
  job_name text,
  force boolean                    default false
  )
  ```

  示例：

  ```
  gaussdb=# SELECT dbe_scheduler.create_job('job1','PLSQL_BLOCK','BEGIN INSERT INTO test1
  VALUES(12); end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
   create_job
  ------------

  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.stop_single_job('job1', true);
   stop_single_job
  -----------------

  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
   drop_job
  ----------

  (1 row)
  ```

- DBE_SCHEDULER.GENERATE_JOB_NAME

  生成定时任务名称。

DBE_SCHEDULER.GENERATE_JOB_NAME函数原型为：

```
DBE_SCHEDULER.generate_job_name(
prefix text                         default 'JOB$_'
)return text
```

须知

首次执行DBE_SCHEDULER.GENERATE_JOB_NAME会在public下创建一个临时序列用于保存当前名称的序号。由于普通用户没有在public下create权限，因此如果普通用户为当前db下第一次调用该函数，会失败，需要授权该普通用户在public下的create权限，或者使用有该权限的用户调用该接口以创建临时序列。

示例：

```
gaussdb=# CALL DBE_SCHEDULER.generate_job_name();
 generate_job_name
-------------------
 JOB$_1
(1 row)

gaussdb=# CALL DBE_SCHEDULER.generate_job_name();
 generate_job_name
-------------------
 JOB$_2
(1 row)

gaussdb=# CALL DBE_SCHEDULER.generate_job_name('job');
 generate_job_name
-------------------
 job3
(1 row)

gaussdb=# CALL DBE_SCHEDULER.generate_job_name('job');
 generate_job_name
-------------------
 job4
(1 row)
```

- DBE_SCHEDULER.CREATE_PROGRAM

  创建程序。

  DBE_SCHEDULER.CREATE_PROGRAM函数原型为：

  ```
  DBE_SCHEDULER.create_program(
  program_name text,
  program_type text,
  program_action text,
  number_of_arguments integer           default 0,
  enabled boolean                     default false,
  comments text                       default NULL
  )
  ```

  示例：

  ```
  gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'SELECT
  pg_sleep(1);', 3, false, 'test');
   create_program
  ----------------

  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
   drop_program
  --------------

  (1 row)
  ```


- DBE_SCHEDULER.DEFINE_PROGRAM_ARGUMENT

  定义程序参数。

  修正带默认值default_value字段的接口默认转换成小写行为，对字符大小写做出区分。

  DBE_SCHEDULER.DEFINE_PROGRAM_ARGUMENT函数原型为：

  ```
  DBE_SCHEDULER.define_program_argument(
  program_name text,
  argument_position integer,
  argument_name text                  default NULL,
  argument_type text,
  out_argument boolean                 default false
  )

  -- 带有默认值 --
  DBE_SCHEDULER.define_program_argument(
  program_name text,
  argument_position integer,
  argument_name text                  default NULL,
  argument_type text,
  default_value text,
  out_argument boolean                 default false
  )
  ```

  示例：

  ```
  gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'SELECT
  pg_sleep(1);', 2, false, 'test');
  create_program
  ----------------

  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.define_program_argument('program1', 1, 'pa1', 'type1', false);
   define_program_argument
  -------------------------

  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.define_program_argument('program1', 1, 'pa1', 'type1', 'value1',
  false);
   define_program_argument
  -------------------------

  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
   drop_program
  --------------

  (1 row)
  ```

- DBE_SCHEDULER.DROP_PROGRAM

  删除程序。

  DBE_SCHEDULER.DROP_PROGRAM函数原型为：

  ```
  DBE_SCHEDULER.drop_program(
  program_name text,
  force boolean                    default false
  )
  ```

  示例：

  ```
  gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'SELECT
  pg_sleep(1);', 2, false, 'test');
  create_program
  ----------------
  ```



```
(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
 drop_program
--------------

(1 row)
```

- DBE_SCHEDULER.DROP_SINGLE_PROGRAM

  删除单个程序。

  DBE_SCHEDULER.DROP_SINGLE_PROGRAM函数原型为：

  ```
  DBE_SCHEDULER.drop_single_program(
  program_name text,
  force boolean                    default false
  )
  ```

  示例：

  ```
  gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'SELECT
  pg_sleep(1);', 2, false, 'test');
  create_program
  ----------------

  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.drop_single_program('program1', false);
   drop_single_program
  ---------------------

  (1 row)
  ```

- DBE_SCHEDULER.SET_JOB_ARGUMENT_VALUE

  设置定时任务参数值。argument_value字段支持赋空入参。

  DBE_SCHEDULER.SET_JOB_ARGUMENT_VALUE函数原型为：

  ```
  DBE_SCHEDULER.set_job_argument_value(
  job_name text,
  argument_position integer,
  argument_value text
  )

  DBE_SCHEDULER.set_job_argument_value(
  job_name text,
  argument_name text,
  argument_value text
  )
  ```

  示例：

  ```
  gaussdb=# CALL dbe_scheduler.create_job('job1','EXTERNAL_SCRIPT','BEGIN INSERT INTO test1
  VALUES(12); end;',2,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
   create_job
  ------------

  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.set_job_argument_value('job1', 1, 'value1');
   set_job_argument_value
  ------------------------

  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
   drop_job
  ----------

  (1 row)
  ```

- DBE_SCHEDULER.CREATE_SCHEDULE

  创建调度。

DBE_SCHEDULER.CREATE_SCHEDULE函数原型为：

```
DBE_SCHEDULER.create_schedule(
schedule_name text,
start_date timestamp with time zone     default NULL,
repeat_interval text,
end_date timestamp with time zone       default NULL,
comments text                           default NULL
)
```

示例：

```
gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule1', sysdate, 'sysdate + 3 / (24 * 60 * 60)',
null, 'test1');
 create_schedule
-----------------


(1 row)

gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule2', sysdate, 'FREQ=DAILY; BYHOUR=6;',
null, 'test1');
 create_schedule
-----------------


(1 row)

gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule3', sysdate, 'FREQ=DAILY; BYHOUR=6;');
 create_schedule
-----------------


(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_schedule('schedule1');
 drop_single_schedule
----------------------


(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_schedule('schedule2', false);
 drop_single_schedule
----------------------


(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_schedule('schedule3', true);
 drop_single_schedule
----------------------


(1 row)
```

- DBE_SCHEDULER.DROP_SCHEDULE

  删除调度。

  DBE_SCHEDULER.DROP_SCHEDULE函数原型为：

  ```
  DBE_SCHEDULER.drop_schedule(
  schedule_name text,
  force boolean                       default false
  )
  ```

  示例：

  ```
  gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule1', sysdate, 'sysdate + 3 / (24 * 60 * 60)',
  null, 'test1');
   create_schedule
  -----------------


  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule2', sysdate, 'FREQ=DAILY; BYHOUR=6;',
  null, 'test1');
   create_schedule
  -----------------
  ```



```
(1 row)

gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule3', sysdate, 'FREQ=DAILY; BYHOUR=6;');
 create_schedule
-----------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_schedule('schedule1');
 drop_single_schedule
----------------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_schedule('schedule2', false);
 drop_single_schedule
----------------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_schedule('schedule3', true);
 drop_single_schedule
----------------------

(1 row)
```

- DBE_SCHEDULER.DROP_SINGLE_SCHEDULE

  删除单个调度。

  DBE_SCHEDULER.DROP_SINGLE_SCHEDULE函数原型为：

  ```
  DBE_SCHEDULER.drop_single_schedule(
  schedule_name text,
  force boolean                 default false
  )
  ```

  示例：

  ```
  gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule1', sysdate, 'sysdate + 3 / (24 * 60 * 60)',
  null, 'test1');
   create_schedule
  -----------------

  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule2', sysdate, 'FREQ=DAILY; BYHOUR=6;',
  null, 'test1');
   create_schedule
  -----------------

  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule3', sysdate, 'FREQ=DAILY; BYHOUR=6;');
   create_schedule
  -----------------

  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.drop_single_schedule('schedule1');
   drop_single_schedule
  ----------------------

  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.drop_single_schedule('schedule2', false);
   drop_single_schedule
  ----------------------

  (1 row)
  ```



```
gaussdb=# CALL DBE_SCHEDULER.drop_single_schedule('schedule3', true);
 drop_single_schedule
----------------------


(1 row)
```

- DBE_SCHEDULER.CREATE_JOB_CLASS

  创建定时任务类。

  DBE_SCHEDULER.CREATE_JOB_CLASS函数原型为：

  ```
  DBE_SCHEDULER.create_job_class(
  job_class_name text,
  resource_consumer_group text            default NULL,
  service text                          default NULL,
  logging_level integer                   default 0,
  log_history integer                   default NULL,
  comments text                          default NULL
  )
  ```

  示例：

  ```
  gaussdb=# CALL DBE_SCHEDULER.create_job_class(job_class_name => 'jc1', resource_consumer_group
  => '123');
   create_job_class
  ------------------


  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.drop_job_class('jc1', false);
   drop_job_class
  ----------------


  (1 row)
  ```

- DBE_SCHEDULER.DROP_JOB_CLASS

  删除定时任务类。

  DBE_SCHEDULER.DROP_JOB_CLASS函数原型为：

  ```
  DBE_SCHEDULER.drop_job_class(
  job_class_name text,
  force boolean                        default false
  )
  ```

  示例：

  ```
  gaussdb=# CALL DBE_SCHEDULER.create_job_class(job_class_name => 'jc1', resource_consumer_group
  => '123');
   create_job_class
  ------------------


  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.drop_job_class('jc1', false);
   drop_job_class
  ----------------


  (1 row)
  ```

- DBE_SCHEDULER.DROP_SINGLE_JOB_CLASS

  删除单个定时任务类。

  DBE_SCHEDULER.DROP_SINGLE_JOB_CLASS函数原型为：

  ```
  DBE_SCHEDULER.drop_single_job_class(
  job_class_name text,
  force boolean                        default false
  )
  ```

  示例：

  ```
  gaussdb=# CALL DBE_SCHEDULER.create_job_class(job_class_name => 'jc1', resource_consumer_group
  => '123');
  ```



```
 create_job_class
------------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_single_job_class('jc1', false);
 drop_job_class
----------------

(1 row)
```

- DBE_SCHEDULER.GRANT_USER_AUTHORIZATION

  为数据库用户提供定时任务权限。调用该函数的用户需要具有SYSADMIN权限。

  DBE_SCHEDULER.GRANT_USER_AUTHORIZATION函数原型为：

  ```
  DBE_SCHEDULER.grant_user_authorization(
  username              text,
  privilege             text
  )
  ```

  示例：

  ```
  gaussdb=# CREATE USER user1 PASSWORD '1*s*****';
  NOTICE:  The encrypted password contains MD5 ciphertext, which is not secure.
  CREATE ROLE
  gaussdb=# CALL DBE_SCHEDULER.grant_user_authorization('user1', 'CREATE JOB');
   grant_user_authorization
  --------------------------

  (1 row)
  gaussdb=# DROP USER user1;
  DROP ROLE
  ```

- DBE_SCHEDULER.REVOKE_USER_AUTHORIZATION

  撤销数据库用户的定时任务权限。调用该函数的用户需要具有SYSADMIN权限。

  DBE_SCHEDULER.REVOKE_USER_AUTHORIZATION函数原型为：

  ```
  DBE_SCHEDULER.revoke_user_authorization(
  username              text,
  privilege             text
  )
  ```

  示例：

  ```
  gaussdb=# CREATE USER user1 PASSWORD '1*s*****';
  NOTICE:  The encrypted password contains MD5 ciphertext, which is not secure.
  CREATE ROLE
  gaussdb=# CALL DBE_SCHEDULER.grant_user_authorization('user1', 'CREATE JOB');
   grant_user_authorization
  --------------------------

  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.revoke_user_authorization('user1', 'CREATE JOB');
   revoke_user_authorization
  ---------------------------

  (1 row)
  gaussdb=# DROP USER user1;
  DROP ROLE
  ```

- DBE_SCHEDULER.CREATE_CREDENTIAL

  创建授权证书。调用该函数的用户需要具有SYSADMIN权限。

  DBE_SCHEDULER.CREATE_CREDENTIAL函数原型为：

  ```
  DBE_SCHEDULER.create_credential(
  credential_name         text,
  username              text,
  password              text          default NULL,
  database_role          text          default NULL,
  ```



```
windows_domain          text            default NULL,
comments                text            default NULL
)
```

示例：

```
gaussdb=# CALL DBE_SCHEDULER.create_credential('cre_1', 'user1', '');
 create_credential
-------------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_credential('cre_1', false);
 drop_credential
-----------------

(1 row)
```

须知

DBE_SCHEDULER.CREATE_CREDENTIAL的password字段请传NULL或者'******',该参数仅做兼容性，不代表实际含义。禁止使用安装用户对应的os用户名创建证书。

- DBE_SCHEDULER.DROP_CREDENTIAL

  销毁授权证书。调用该函数的用户需要具有SYSADMIN权限。

  DBE_SCHEDULER.DROP_CREDENTIAL函数原型为：

  ```
  DBE_SCHEDULER.drop_credential(
  credential_name  text,
  force            boolean default false
  )
  ```

  示例：

  ```
  gaussdb=# CALL DBE_SCHEDULER.create_credential('cre_1', 'user1', '');
   create_credential
  -------------------

  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.drop_credential('cre_1', false);
   drop_credential
  -----------------

  (1 row)
  ```

- DBE_SCHEDULER.ENABLE

  启用对象。

  DBE_SCHEDULER.ENABLE函数原型为：

  ```
  DBE_SCHEDULER.enable(
  name text,
  commit_semantics text                 default 'STOP_ON_FIRST_ERROR'
  )
  ```

  示例：

  ```
  gaussdb=# CALL dbe_scheduler.create_job('job1','PLSQL_BLOCK','BEGIN INSERT INTO test1
  VALUES(12); end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
   create_job
  ------------

  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'stored_procedure', 'INSERT INTO
  tb_job_test(key) VALUES(null);', 0, false, '');
   create_program
  ```



```
----------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.enable('job1');
 enable
--------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.enable('program1', 'STOP_ON_FIRST_ERROR');
 enable
--------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
 drop_program
--------------

(1 row)
```

- DBE_SCHEDULER.ENABLE_SINGLE

  启用单个对象。

  DBE_SCHEDULER.ENABLE_SINGLE函数原型为：

  ```
  DBE_SCHEDULER.enable_single(
  name text
  )
  ```

  示例：

  ```
  gaussdb=# CALL dbe_scheduler.create_job('job1','PLSQL_BLOCK','BEGIN INSERT INTO test1
  VALUES(12); end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
   create_job
  ------------

  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.enable_single('job1');
   enable_single
  ---------------

  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
   drop_job
  ----------

  (1 row)
  ```

- DBE_SCHEDULER.DISABLE

  禁用多个对象，name为逗号分隔的字符串，每个逗号分隔的字符串为一个对象。仅启用操作同步时同步操作。

  DBE_SCHEDULER.DISABLE函数原型为：

  ```
  DBE_SCHEDULER.disable(
  name text,
  force boolean                   default false,
  commit_semantics text               default 'STOP_ON_FIRST_ERROR'
  )
  ```

  示例：


```
gaussdb=# CALL dbe_scheduler.create_job('job1','PLSQL_BLOCK','BEGIN INSERT INTO test1
VALUES(12); end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
 create_job
------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'stored_procedure', 'INSERT INTO
tb_job_test(key) VALUES(null);', 0, false, '');
 create_program
----------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.disable('job1');
 disable
---------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.disable('program1', false, 'STOP_ON_FIRST_ERROR');
 disable
---------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
 drop_program
--------------

(1 row)
```

- DBE_SCHEDULER.DISABLE_SINGLE

  禁用单个对象。

  DBE_SCHEDULER.DISABLE_SINGLE函数原型为：

  ```
  DBE_SCHEDULER.disable_single(
  name text,
  force boolean                    default false
  )
  ```

  示例：

  ```
  gaussdb=# CALL dbe_scheduler.create_job('job1','PLSQL_BLOCK','BEGIN INSERT INTO test1
  VALUES(12); end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
   create_job
  ------------

  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.disable_single('job1', false);
   disable_single
  ----------------

  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
   drop_job
  ----------

  (1 row)
  ```

- DBE_SCHEDULER.EVAL_CALENDAR_STRING

  分析调度任务周期。

返回值类型：timestamp with time zone

DBE_SCHEDULER.EVAL_CALENDAR_STRING函数原型为：

```
DBE_SCHEDULER.evaluate_calendar_string(
IN calendar_string text,
IN start_date timestamp with time zone,
IN return_date_after timestamp with time zone
)return timestamp with time zone
```

示例：

```
gaussdb=# CALL DBE_SCHEDULER.eval_calendar_string('FREQ=DAILY; BYHOUR=6;', sysdate, sysdate);
  eval_calendar_string
------------------------
 2023-09-16 06:05:55+08
(1 row)
```

- DBE_SCHEDULER.EVALUATE_CALENDAR_STRING

  分析调度任务周期。

  DBE_SCHEDULER.EVALUATE_CALENDAR_STRING函数原型为：

```
DBE_SCHEDULER.evaluate_calendar_string(
IN calendar_string text,
IN start_date timestamp with time zone,
IN return_date_after timestamp with time zone,
OUT next_run_date timestamp with time zone
)return timestamp with time zone
```

  示例：

```
gaussdb=# CREATE OR REPLACE PROCEDURE pr1(calendar_str text) as
 DECLARE
    start_date        timestamp with time zone;
    return_date_after timestamp with time zone;
    next_run_date     timestamp with time zone;
 BEGIN
    start_date := '2003-2-1 10:30:00.111111+8'::timestamp with time zone;
    return_date_after := start_date;
    DBE_SCHEDULER.evaluate_calendar_string(
        calendar_str,
        start_date, return_date_after, next_run_date);
    DBE_OUTPUT.PRINT_LINE('next_run_date: ' || next_run_date);
    return_date_after := next_run_date;
 END;
 /
CREATE PROCEDURE
gaussdb=# CALL pr1('FREQ=hourly;INTERVAL=2;BYHOUR=6,10;BYMINUTE=0;BYSECOND=0');
next_run_date: 2003-02-02 06:00:00+08
 pr1
-----


(1 row)
```

10.12.2.7 DBE_SQL

数据类型介绍

- DBE_SQL.DESC_REC

  该类型是复合类型，用来存储SQL_DESCRIBE_COLUMNS接口中的描述信息。

  DBE_SQL.DESC_REC函数的原型为：

```
CREATE TYPE DBE_SQL.DESC_REC AS (
    col_type            int,
    col_max_len         int,
    col_name            VARCHAR2(32),
    col_name_len        int,
    col_schema_name     VARCHAR2(32),
    col_schema_name_len int,
```



```
    col_precision       int,
    col_scale           int,
    col_charsetid       int,
    col_charsetform     int,
    col_null_ok         BOOLEAN
);
```

- DBE_SQL.DESC_TAB
  该类型是DESC_REC的TABLE类型，通过TABLE OF语法实现。
  DBE_SQL.DESC_TAB函数的原型为：
  ```
  CREATE TYPE DBE_SQL.DESC_TAB AS TABLE OF DBE_SQL.DESC_REC INDEX BY INTEGER;
  ```
- DBE_SQL.DATE_TABLE
  该类型是DATE的TABLE类型，通过TABLE OF语法实现。
  DBE_SQL.DATE_TABLE函数的原型为：
  ```
  CREATE TYPE DBE_SQL.DATE_TABLE AS TABLE OF DATE INDEX BY INTEGER;
  ```
- DBE_SQL.NUMBER_TABLE
  该类型是NUMBER的TABLE类型，通过TABLE OF语法实现。
  DBE_SQL.NUMBER_TABLE函数的原型为：
  ```
  CREATE TYPE DBE_SQL.NUMBER_TABLE AS TABLE OF NUMBER INDEX BY INTEGER;
  ```
- DBE_SQL.VARCHAR2_TABLE
  该类型是VARCHAR2的TABLE类型，通过TABLE OF语法实现。
  DBE_SQL.VARCHAR2_TABLE函数的原型为：
  ```
  CREATE TYPE DBE_SQL.VARCHAR2_TABLE AS TABLE OF VARCHAR2(32767) INDEX BY INTEGER;
  ```
- DBE_SQL.BLOB_TABLE
  该类型是BLOB的TABLE类型，通过TABLE OF语法实现。
  DBE_SQL.BLOB_TABLE函数的原型为：
  ```
  CREATE TYPE DBE_SQL.BLOB_TABLE AS TABLE OF BLOB INDEX BY INTEGER;
  ```

接口介绍

高级功能包DBE_SQL支持的接口请参见**表10-231**。

表 10-231 DBE_SQL

接口名称	描述
DBE_SQL.REGISTER_CONTEXT	打开一个游标。
DBE_SQL.SQL_UNREGISTER_CONTEXT	关闭一个已打开的游标。
DBE_SQL.SQL_SET_SQL	向游标传递一组SQL语句或匿名块。
DBE_SQL.SQL_RUN	执行给定游标中的SQL语句或匿名块。
DBE_SQL.NEXT_ROW	读取游标一行数据。
DBE_SQL.SET_RESULT_TYPE	动态定义一个列。

接口名称	描述
DBE_SQL.SET_RESULT_TYPE_CHAR	动态定义一个char类型的列。
DBE_SQL.SET_RESULT_TYPE_INT	动态定义一个int类型的列。
DBE_SQL.SET_RESULT_TYPE_LONG	动态定义一个long类型的列。
DBE_SQL.SET_RESULT_TYPE_RAW	动态定义一个raw类型的列。
DBE_SQL.SET_RESULT_TYPE_BYTEA	动态定义一个bytea类型的列。
DBE_SQL.SET_RESULT_TYPE_TEXT	动态定义一个text类型的列。
DBE_SQL.SET_RESULT_TYPE_UNKNOWN	动态定义一个未知列（类型不识别时入此接口）。
DBE_SQL.GET_RESULT	读取一个已动态定义的列值。
DBE_SQL.GET_RESULT_CHAR	读取一个已动态定义的列值（指定char类型）。
DBE_SQL.GET_RESULT_INT	读取一个已动态定义的列值（指定int类型）。
DBE_SQL.GET_RESULT_LONG	读取一个已动态定义的列值（指定long类型）。
DBE_SQL.GET_RESULT_RAW	读取一个已动态定义的列值（指定raw类型）。
DBE_SQL.GET_RESULT_BYTEA	读取一个已动态定义的列值（指定bytea类型）。
DBE_SQL.GET_RESULT_TEXT	读取一个已动态定义的列值（指定text类型）。
DBE_SQL.GET_RESULT_UNKNOWN	读取一个已动态定义的列值（类型不识别时入此接口）。
DBE_SQL.DBE_SQL_GET_RESULT_CHAR	读取一个已动态定义的列值（指定char类型）。
DBE_SQL.DBE_SQL_GET_RESULT_LONG	读取一个已动态定义的列值（指定long类型）。
DBE_SQL.DBE_SQL_GET_RESULT_RAW	读取一个已动态定义的列值（指定raw类型）。
DBE_SQL.IS_ACTIVE	检查游标是否已打开。
DBE_SQL.LAST_ROW_COUNT	返回获取行数的累积计数。
DBE_SQL.RUN_AND_NEXT	在游标上执行一组动态定义操作后，读取游标数据。

接口名称	描述
DBE_SQL.SQL_BIND_VARIABLE	根据语句中的变量，绑定一个值到该变量。
DBE_SQL.SQL_BIND_ARRAY	根据语句中的变量，绑定一组值到该变量。
DBE_SQL.SET_RESULT_TYPE_INTS	动态定义一个int数组类型的列。
DBE_SQL.SET_RESULT_TYPE_TEXTS	动态定义一个text数组类型的列。
DBE_SQL.SET_RESULT_TYPE_RAWS	动态定义一个raw数组类型的列。
DBE_SQL.SET_RESULT_TYPE_BYTEAS	动态定义一个bytea数组类型的列。
DBE_SQL.SET_RESULT_TYPE_CHARS	动态定义一个char数组类型的列。
DBE_SQL.SET_RESULTS_TYPE	动态定义一个数组类型的列。
DBE_SQL.GET_RESULTS_INT	读取一个已动态定义的列值（指定int数组类型）。
DBE_SQL.GET_RESULTS_TEXT	读取一个已动态定义的列值（指定text数组类型）。
DBE_SQL.GET_RESULTS_RAW	读取一个已动态定义的列值（指定raw数组类型）。
DBE_SQL.GET_RESULTS_BYTEA	读取一个已动态定义的列值（指定bytea数组类型）。
DBE_SQL.GET_RESULTS_CHAR	读取一个已动态定义的列值（指定char数组类型）。
DBE_SQL.GET_RESULTS	读取一个已动态定义的列值。
DBE_SQL.SQL_DESCRIBE_COLUMNS	描述游标读取的列信息。
DBE_SQL.DESCRIBE_COLUMNS	描述游标读取的列信息。
DBE_SQL.BIND_VARIABLE	绑定参数接口
DBE_SQL.SQL_SET_RESULTS_TYPE_C	动态定义一个数组类型的列。
DBE_SQL.SQL_GET_VALUES_C	读取一个已动态定义的列值。
DBE_SQL.GET_VARIABLE_RESULT	读取一个SQL语句执行后的返回值。
DBE_SQL.GET_VARIABLE_RESULT_CHAR	读取一个SQL语句执行后的返回值（指定char类型）。
DBE_SQL.GET_VARIABLE_RESULT_RAW	读取一个SQL语句执行后的返回值（指定raw类型）。

接口名称	描述
DBE_SQL.GET_VARIABLE_RESULT_TEXT	读取一个SQL语句执行后的返回值（指定text类型）。
DBE_SQL.GET_VARIABLE_RESULT_INT	读取一个SQL语句执行后的返回值（指定int类型）。
DBE_SQL.GET_VARIABLE_RESULT_TEXT	读取一个SQL语句执行后的返回值（指定text数组类型）。
DBE_SQL.GET_ARRAY_RESULT_RAW	读取一个SQL语句执行后的返回值（指定raw数组类型）。
DBE_SQL.GET_ARRAY_RESULT_CHAR	读取一个SQL语句执行后的返回值（指定char数组类型）。
DBE_SQL.GET_ARRAY_RESULT_INT	读取一个SQL语句执行后的返回值（指定int数组类型）。
DBE_SQL.SQL_SET_TABLEOF_RESULTS_TYPE_C	动态定义一个tableof类型的列。
DBE_SQL.SQL_GET_TABLEOF_VALUES_C	读取一个已动态定义的tableof类型的列值。

说明

- 建议使用dbe_sql.set_result_type及dbe_sql.get_result定义参数列。
- 当结果集大于work_mem设定值时会触发结果集临时下盘，但最大阈值不超过512MB。

- DBE_SQL.REGISTER_CONTEXT

  该函数用来打开一个游标，是后续dbe_sql各项操作的前提。该函数不传入任何参数，内部自动递增生成游标ID，并作为返回值返回给integer定义的变量。

  注意

  DBE_SQL打开的游标是会话级的变量，不支持跨会话调用打开的游标（如自治事务），如果调用跨会话的游标行为不可预知。

  DBE_SQL.REGISTER_CONTEXT函数原型为：

  ```
  DBE_SQL.REGISTER_CONTEXT(
  )
  RETURN INTEGER;
  ```

- DBE_SQL.SQL_UNREGISTER_CONTEXT

  该函数用来关闭一个游标，是dbe_sql各项操作的结束。如果在存储过程结束时没有调用该函数，则该游标占用的内存仍然会保存，因此关闭游标非常重要。由于异常情况的发生会中途退出存储过程，导致游标未能关闭，因此建议，如果存储过程中有异常处理，应将该接口包含在内。

  DBE_SQL.SQL_UNREGISTER_CONTEXT函数原型为：

  ```
  DBE_SQL.SQL_UNREGISTER_CONTEXT(
      context_id     IN INTEGER
  ```

```
)
RETURN INTEGER;
```

表 10-232 DBE_SQL.SQL_UNREGISTER_CONTEXT 接口说明

参数名称	描述
context_id	打算关闭的游标ID号。

- DBE_SQL.SQL_SET_SQL

  该函数用来解析给定游标的SQL语句或匿名块。目前语句参数仅可通过text类型传递，长度不大于1G。

  DBE_SQL.SQL_SET_SQL函数的原型为：

```
DBE_SQL.SQL_SET_SQL(
    context_id     IN INTEGER,
    query_string   IN TEXT,
    language_flag IN INTEGER
)
RETURN BOOLEAN;
```

表 10-233 DBE_SQL.SQL_SET_SQL 接口说明

参数名称	描述
context_id	执行查询语句解析的游标ID。
query_string	执行解析的查询语句。
language_flag	版本语言号，指定不同版本的行为1为非兼容版本，2为兼容A版本。

- DBE_SQL.SQL_RUN

  该函数用来执行一个给定的游标。该函数接收一个游标ID，执行给定游标中的SQL语句或匿名块。

  DBE_SQL.SQL_RUN函数的原型为：

```
DBE_SQL.SQL_RUN(
    context_id     IN INTEGER,
)
RETURN INTEGER;
```

表 10-234 DBE_SQL.SQL_RUN 接口说明

参数名称	描述
context_id	执行查询语句解析的游标ID。

- DBE_SQL.NEXT_ROW

  该函数返回符合查询条件的数据行数，每一次运行该接口都会获取到新的行数的集合，直到数据读取完毕获取不到新行为止。

  DBE_SQL.NEXT_ROW函数的原型为：

```
DBE_SQL.NEXT_ROW(
    context_id     IN INTEGER,
)
RETURN INTEGER;
```


表 10-235 DBE_SQL.NEXT_ROW 接口说明

参数名称	描述
context_id	执行的游标ID。

- DBE_SQL.SET_RESULT_TYPE

  该函数用来定义从给定游标返回的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。

  DBE_SQL.SET_RESULT_TYPE函数的原型为：

```
DBE_SQL.SET_RESULT_TYPE(
    context_id IN INTEGER,
    pos        IN INTEGER,
    column_ref IN ANYELEMENT,
    maxsize    IN INTEGER default 1024
)
RETURN INTEGER;
```

表 10-236 DBE_SQL.SET_RESULT_TYPE 接口说明

参数名称	描述
context_id	执行的游标ID。
pos	查询列在返回结果中的相对位置，起始为1。
column_ref	任意类型的变量，可根据变量类型选择适当的接口动态定义列。
maxsize	定义的列返回类型长度。

- DBE_SQL.SET_RESULT_TYPE_CHAR

  该函数用来定义从给定游标返回的CHAR类型的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。

  DBE_SQL.SET_RESULT_TYPE_CHAR函数的原型为：

```
DBE_SQL.SET_RESULT_TYPE_CHAR(
    context_id  IN INTEGER,
    pos         IN INTEGER,
    column_ref  IN TEXT,
    column_size IN INTEGER
)
RETURN INTEGER;
```

表 10-237 DBE_SQL.SET_RESULT_TYPE_CHAR 接口说明

参数名称	描述
context_id	执行的游标ID。
pos	动态定义列在查询中的位置。
column_ref	需要定义的某类型的参数变量。

参数名称	描述
column_size	动态定义列长度。

- DBE_SQL.SET_RESULT_TYPE_INT

  该函数用来定义从给定游标返回的INT类型的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。

  DBE_SQL.SET_RESULT_TYPE_INT函数的原型为：

```
DBE_SQL.SET_RESULT_TYPE_INT(
    context_id IN INTEGER,
    pos        IN INTEGER
)
RETURN INTEGER;
```

  表 10-238 DBE_SQL.SET_RESULT_TYPE_INT 接口说明

参数名称	描述
context_id	执行的游标ID。
pos	动态定义列在查询中的位置。

- DBE_SQL.SET_RESULT_TYPE_LONG

  该函数用来定义从给定游标返回的长列类型（非数据类型long）的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。长列的大小限制为1G。

  DBE_SQL.SET_RESULT_TYPE_LONG函数的原型为：

```
DBE_SQL.SET_RESULT_TYPE_LONG(
    context_id IN INTEGER,
    pos        IN INTEGER
)
RETURN INTEGER;
```

  表 10-239 DBE_SQL.SET_RESULT_TYPE_LONG 接口说明

参数名称	描述
context_id	执行的游标ID。
pos	动态定义列在查询中的位置。

- DBE_SQL.SET_RESULT_TYPE_RAW

  该函数用来定义从给定游标返回的RAW类型的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。

  DBE_SQL.SET_RESULT_TYPE_RAW函数的原型为：

```
DBE_SQL.SET_RESULT_TYPE_RAW(
    context_id  IN INTEGER,
    pos         IN INTEGER,
    column_ref  IN RAW,
    column_size IN INTEGER
)
RETURN INTEGER;
```


表 10-240 DBE_SQL.SET_RESULT_TYPE_RAW 接口说明

参数名称	描述
context_id	执行的游标ID。
pos	动态定义列在查询中的位置。
column_ref	RAW类型的参数变量。
column_size	列的长度。

- DBE_SQL.SET_RESULT_TYPE_BYTEA

  该函数用来定义从给定游标返回的BYTEA类型的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。

  DBE_SQL.SET_RESULT_TYPE_BYTEA函数的原型为：

```
DBE_SQL.SET_RESULT_TYPE_BYTEA(
    context_id  IN INTEGER,
    pos         IN INTEGER,
    column_ref  IN BYTEA,
    column_size IN INTEGER
)
RETURN INTEGER;
```

表 10-241 DBE_SQL.SET_RESULT_TYPE_BYTEA 接口说明

参数名称	描述
context_id	执行的游标ID。
pos	动态定义列在查询中的位置。
column_ref	BYTEA类型的参数变量。
column_size	列的长度。

- DBE_SQL.SET_RESULT_TYPE_TEXT

  该函数用来定义从给定游标返回的TEXT类型的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。

  DBE_SQL.SET_RESULT_TYPE_TEXT函数的原型为：

```
DBE_SQL.DEFINE_COLUMN_CHAR(
    context_id IN INTEGER,
    pos        IN INTEGER,
    maxsize    IN INTEGER
)
RETURN INTEGER;
```

表 10-242 DBE_SQL.SET_RESULT_TYPE_TEXT 接口说明

参数名称	描述
context_id	执行的游标ID
pos	动态定义列在查询中的位置

参数名称	描述
maxsize	定义的TEXT类型的最大长度

- DBE_SQL.SET_RESULT_TYPE_UNKNOWN

  该函数用来处理从给定游标返回的未知数据类型的列，该接口仅用于类型不识别时的报错退出。

  DBE_SQL.SET_RESULT_TYPE_UNKNOWN函数的原型为：

```
DBE_SQL.SET_RESULT_TYPE_UNKNOWN(
    context_id IN INTEGER,
    pos        IN INTEGER,
    col_type   IN TEXT
)
RETURN INTEGER;
```

  表 10-243 DBE_SQL.SET_RESULT_TYPE_UNKNOWN 接口说明

参数名称	描述
context_id	执行的游标ID。
posn	动态定义列在查询中的位置。
col_type	动态定义的参数。

- DBE_SQL.GET_RESULT

  该函数用来返回给定游标给定位置的游标元素值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  DBE_SQL.GET_RESULT函数的原型为：

```
DBE_SQL.GET_RESULT(
    context_id   IN    INTEGER,
    pos          IN    INTEGER,
    column_value INOUT ANYELEMENT
)
RETURN ANYELEMENT;
```

  表 10-244 DBE_SQL.GET_RESULT 接口说明

参数名称	描述
context_id	执行的游标ID。
pos	查询列在返回结果中的相对位置，起始为1。
column_value	指定列的查询结果返回值。

- DBE_SQL.GET_RESULT_CHAR

  该存储过程用来返回给定游标给定位置的游标CHAR类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  DBE_SQL.GET_RESULT_CHAR存储过程的原型为：

```
DBE_SQL.GET_RESULT_CHAR(
    context_id    IN    INTEGER,
    pos           IN    INTEGER,
```



```
  tr            INOUT CHARACTER,
  err           INOUT NUMERIC,
  actual_length INOUT INTEGER
);
```

表 10-245 DBE_SQL.GET_RESULT_CHAR 接口说明

参数名称	描述
context_id	执行的游标ID。
pos	查询列在返回结果中的相对位置，起始为1。
tr	返回值
err	错误号。传出参数，须传入变量做参数。目前未实现，固定传出-1。
actual_length	返回值的实际长度。

DBE_SQL.GET_RESULT_CHAR存储过程的重载为：

```
DBE_SQL.GET_RESULT_CHAR(
  context_id IN    INTEGER,
  pos        IN    INTEGER,
  tr         INOUT CHARACTER
);
```

表 10-246 DBE_SQL.GET_RESULT_CHAR 接口说明

参数名称	描述
context_id	执行的游标ID。
pos	动态定义列在查询中的位置。
tr	返回值。

- DBE_SQL.GET_RESULT_INT

  该函数用来返回给定游标给定位置的游标INT类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。DBE_SQL.GET_RESULT_INT函数的原型为：

  ```
  DBE_SQL.GET_RESULT_INT(
    context_id IN INTEGER,
    pos        IN INTEGER
  )
  RETURN INTEGER;
  ```

  表 10-247 DBE_SQL.GET_RESULT_INT 接口说明

参数名称	描述
context_id	执行的游标ID。
pos	动态定义列在查询中的位置。

- DBE_SQL.GET_RESULT_LONG

该函数用来返回给定游标给定位置的游标长列（非long/bigint整型）类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

DBE_SQL.GET_RESULT_LONG函数的原型为：

```
DBE_SQL.GET_RESULT_LONG(
    context_id IN INTEGER,
    pos        IN    INTEGER,
    lgth       IN    INTEGER,
    off_set    IN    INTEGER,
    vl         INOUT TEXT,
    vl_length  INOUT INTEGER
)
RETURN RECORD;
```

表 10-248 DBE_SQL.GET_RESULT_LONG 接口说明

参数名称	描述
context_id	执行的游标ID。
pos	动态定义列在查询中的位置。
lgth	返回值的长度。
off_set	返回值的起始位置。
vl	返回值。
vl_length	实际返回值的长度。

- DBE_SQL.GET_RESULT_RAW

  该存储过程用来返回给定游标给定位置的游标RAW类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  DBE_SQL.GET_RESULT_RAW存储过程的原型为：

```
DBE_SQL.GET_RESULT_RAW(
    context_id    IN    INTEGER,
    pos           IN    INTEGER,
    tr            INOUT RAW,
    err           INOUT NUMERIC,
    actual_length INOUT INTEGER
);
```

表 10-249 DBE_SQL.GET_RESULT_RAW 接口说明

参数名称	描述
context_id	执行的游标ID。
pos	动态定义列在查询中的位置。
tr	返回的列值。
err	错误号。传出参数，须传入变量做参数。目前未实现，固定传出-1。
actual_length	返回值的实际长度，不能长于此值，否则截断。

DBE_SQL.GET_RESULT_RAW存储过程的重载为：


```
DBE_SQL.GET_RESULT_RAW(
   context_id IN    INTEGER,
   pos        IN    INTEGER,
   tr         INOUT RAW
);
```

表 10-250 DBE_SQL.GET_RESULT_RAW 接口说明

参数名称	描述
context_id	执行的游标ID。
pos	动态定义列在查询中的位置。
tr	返回的列值。

- DBE_SQL.GET_RESULT_BYTEA

  该存储过程用来返回给定游标给定位置的游标BYTEA类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  DBE_SQL.GET_RESULT_BYTEA存储过程的原型为：

```
DBE_SQL.GET_RESULT_BYTEA(
   context_id IN INTEGER,
   pos        IN INTEGER
)
RETURN BYTEA;
```

表 10-251 DBE_SQL.GET_RESULT_BYTEA 接口说明

参数名称	描述
context_id	执行的游标ID。
pos	动态定义列在查询中的位置。

- DBE_SQL.GET_RESULT_TEXT

  该函数用来返回给定游标给定位置的游标TEXT类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  DBE_SQL.GET_RESULT_TEXT函数的原型为：

```
DBE_SQL.GET_RESULT_TEXT(
   context_id IN  INTEGER,
   pos        IN  INTEGER
)
RETURN TEXT;
```

表 10-252 DBE_SQL.GET_RESULT_TEXT 接口说明

参数名称	描述
context_id	执行的游标ID。
pos	动态定义列在查询中的位置。

- DBE_SQL.GET_RESULT_UNKNOWN

  该函数用来返回给定游标给定位置的游标未知类型的值，该接口为类型不支持时的报错处理接口。

DBE_SQL.GET_RESULT_UNKNOWN函数的原型为：

```
DBE_SQL.GET_RESULT_UNKNOWN(
    context_id IN INTEGER,
    pos        IN NTEGER,
    col_type   IN TEXT
)
RETURN TEXT;
```

表 10-253 DBE_SQL.GET_RESULT_UNKNOWN 接口说明

参数名称	描述
context_id	执行的游标ID。
pos	动态定义列在查询中的位置。
col_type	返回的参数类型。

- DBE_SQL.DBE_SQL_GET_RESULT_CHAR

  该函数用来返回给定游标给定位置的游标CHAR类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。和DBE_SQL.GET_RESULT_CHAR的区别是，不设置返回值长度，返回整个字符串。

  DBE_SQL.DBE_SQL_GET_RESULT_CHAR函数的原型为：

  ```
  DBE_SQL.DBE_SQL_GET_RESULT_CHAR(
      context_id IN INTEGER,
      pos        IN INTEGER
  )
  RETURN CHARACTER;
  ```

  表 10-254 DBE_SQL.DBE_SQL_GET_RESULT_CHAR 接口说明

参数名称	描述
context_id	执行的游标ID。
pos	动态定义列在查询中的位置。

- DBE_SQL.DBE_SQL_GET_RESULT_LONG

  该函数用来返回给定游标给定位置的游标长列（非long/bigint整型）类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  和DBE_SQL.GET_RESULT_LONG的区别是，不设置返回值长度，返回整个BIGINT值。

  DBE_SQL.DBE_SQL_GET_RESULT_LONG函数的原型为：

  ```
  DBE_SQL.DBE_SQL_GET_RESULT_LONG(
      context_id IN INTEGER,
      pos        IN INTEGER
  )
  RETURN BIGINT;
  ```

  表 10-255 DBE_SQL.DBE_SQL_GET_RESULT_LONG 接口说明

参数名称	描述
context_id	执行的游标ID。

参数名称	描述
pos	动态定义列在查询中的位置。

- DBE_SQL.DBE_SQL_GET_RESULT_RAW

  该函数用来返回给定游标给定位置的游标RAW类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  和函数DBE_SQL.GET_RESULT_RAW的区别是，不设置返回值长度，返回整个字符串。

  DBE_SQL.DBE_SQL_GET_RESULT_RAW函数的原型为：

```
DBE_SQL.GET_RESULT_RAW(
   context_id IN    INTEGER,
   pos        IN    INTEGER,
   tr         INOUT RAW
)
RETURN RAW;
```

  表 10-256 DBE_SQL.GET_RESULT_RAW 接口说明

参数名称	描述
context_id	执行的游标ID。
pos	动态定义列在查询中的位置。

- DBE_SQL.IS_ACTIVE

  该函数用来返回游标的当前状态：打开、解析、执行、定义。取值是为TRUE，关闭后为FALSE，未知时报错，其余默认为关闭。

  DBE_SQL.IS_ACTIVE函数的原型为：

```
DBE_SQL.IS_ACTIVE(
   context_id    IN   INTEGER
)
RETURN BOOLEAN;
```

  表 10-257 DBE_SQL.IS_ACTIVE 接口说明

参数名称	描述
context_id	被查询的游标ID。

- DBE_SQL.LAST_ROW_COUNT

  该函数用来返回最近一次NEXT_ROW执行后，获取的数据行数的累积计数。

  DBE_SQL.LAST_ROW_COUNT函数的原型为：

```
DBE_SQL.LAST_ROW_COUNT(
)
RETURN INTEGER;
```

- DBE_SQL.RUN_AND_NEXT

  该函数的功能等同于在调用SQL_RUN后接着调用NEXT_ROW。

  DBE_SQL.RUN_AND_NEXT函数的原型为：

```
DBE_SQL.RUN_AND_NEXT(
   context_id IN INTEGER
```



```
)
RETURNS INTEGER;
```

表 10-258 DBE_SQL.RUN_AND_NEXT 接口说明

参数名称	描述
context_id	执行查询语句解析的游标ID。

- DBE_SQL.SQL_BIND_VARIABLE

  该函数用来绑定一个参数到SQL语句，当执行SQL语句时，会根据该绑定的值来执行。

  DBE_SQL.SQL_BIND_VARIABLE函数的原型为：

```
DBE_SQL.SQL_BIND_VARIABLE(
    context_id     IN int,
    query_string   IN text,
    value          IN anyelement,
    out_value_size IN int default null
)
RETURNS void;
```

表 10-259 DBE_SQL.SQL_BIND_VARIABLE 接口说明

参数名称	描述
context_id	被查询的游标ID。
query_string	绑定的变量名。
value	绑定的值。
out_value_size	返回值的大小，默认值为NULL。在兼容A模式下，仅当value参数值类型为VARCHAR或CHAR时，参数的行为与A数据库一致。

- DBE_SQL.SQL_BIND_ARRAY

  该函数用来绑定一组参数到SQL语句，当执行SQL语句时，会根据该绑定的数组来执行。

  DBE_SQL.SQL_BIND_ARRAY函数的原型为：

```
DBE_SQL.SQL_BIND_ARRAY(
    context_id   IN int,
    query_string IN text,
    value        IN anyarray
)
RETURNS void;
DBE_SQL.SQL_BIND_ARRAY(
    context_id   IN int,
    query_string IN text,
    value        IN anyarray,
    lower_index  IN int,
    higher_index IN int
)
RETURNS void;

DBE_SQL.SQL_BIND_ARRAY(
    context_id   IN int,
    query_string IN text,
```



```
    value        IN anyindexbytable
)
RETURNS void;
DBE_SQL.SQL_BIND_ARRAY(
    context_id   IN int,
    query_string IN text,
    value        IN anyindexbytable,
    lower_index  IN int,
    higher_index IN int
)
RETURNS void;
```

表 10-260 DBE_SQL.SQL_BIND_ARRAY 接口说明

参数名称	描述
context_id	被查询的游标ID。
query_string	绑定的变量名。
value	绑定的数组。
lower_index	绑定数组的最小下标。
higher_index	绑定数组的最大下标。

说明

DBE_SQL.SQL_BIND_ARRAY不支持用户自定义的table类型，请使用**数据类型介绍**中提供的table类型。

- DBE_SQL.SET_RESULT_TYPE_INTS

  该函数用来定义从给定游标返回的INT数组类型的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。

  DBE_SQL.SET_RESULT_TYPE_INTS函数的原型为：

```
DBE_SQL.SET_RESULT_TYPE_INTS(
    context_id IN int,
    pos        IN int,
    column_ref IN anyarray,
    cnt        IN int,
    lower_bnd  IN int
)
RETURNS integer;
```

表 10-261 DBE_SQL.SET_RESULT_TYPE_INTS 接口说明

参数名称	描述
context_id	被查询的游标ID。
pos	动态定义列在查询中的位置。
column_ref	标记返回的数组类型。
cnt	标记一次获取多少个值。
lower_bnd	标记返回数组时的开始下标。

- DBE_SQL.SET_RESULT_TYPE_TEXTS

  该函数用来定义从给定游标返回的TEXT数组类型的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。

  DBE_SQL.SET_RESULT_TYPE_TEXTS函数的原型为：

```
DBE_SQL.SET_RESULT_TYPE_TEXTS(
    context_id IN int,
    pos        IN int,
    column_ref IN anyarray,
    cnt        IN int,
    lower_bnd  IN int,
    maxsize    IN int
)
RETURNS integer;
```

  表 10-262 DBE_SQL.SET_RESULT_TYPE_TEXTS 接口说明

参数名称	描述
context_id	被查询的游标ID。
pos	动态定义列在查询中的位置。
column_ref	标记返回的数组类型。
cnt	标记一次获取多少个值。
lower_bnd	标记返回数组时的开始下标。
maxsize	定义的TEXT类型的最大长度。

- DBE_SQL.SET_RESULT_TYPE_RAWS

  该函数用来定义从给定游标返回的RAW数组类型的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。

  DBE_SQL.SET_RESULT_TYPE_RAWS函数的原型为：

```
DBE_SQL.set_result_type_raws(
    context_id  IN int,
    pos         IN int,
    column_ref  IN anyarray,
    cnt         IN int,
    lower_bnd   IN int,
    column_size IN int
)
RETURNS integer;
```

  表 10-263 DBE_SQL.SET_RESULT_TYPE_RAWS 接口说明

参数名称	描述
context_id	被查询的游标ID。
pos	动态定义列在查询中的位置。
column_ref	标记返回的数组类型。

参数名称	描述
cnt	标记一次获取多少个值。
lower_bnd	标记返回数组时的开始下标。
column_size	列的长度。

- DBE_SQL.SET_RESULT_TYPE_BYTEAS

  该函数用来定义从给定游标返回的BYTEA数组类型的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。

  DBE_SQL.SET_RESULT_TYPE_BYTEAS函数的原型为：

```
DBE_SQL.set_result_type_byteas(
    context_id  IN int,
    pos         IN int,
    column_ref  IN anyarray,
    cnt         IN int,
    lower_bnd   IN int,
    column_size IN int
)
RETURNS integer;
```

  表 10-264 DBE_SQL.SET_RESULT_TYPE_BYTEAS 接口说明

参数名称	描述
context_id	被查询的游标ID。
pos	动态定义列在查询中的位置。
column_ref	标记返回的数组类型。
cnt	标记一次获取多少个值。
lower_bnd	标记返回数组时的开始下标。
column_size	列的长度。

- DBE_SQL.SET_RESULT_TYPE_CHARS

  该函数用来定义从给定游标返回的CHAR数组类型的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。

  DBE_SQL.SET_RESULT_TYPE_CHARS函数的原型为：

```
DBE_SQL.SET_RESULT_TYPE_CHARS(
    context_id  IN int,
    pos         IN int,
    column_ref  IN anyarray,
    cnt         IN int,
    lower_bnd   IN int,
    column_size IN int
)
RETURNS integer;
```


表 10-265 DBE_SQL.SET_RESULT_TYPE_CHARS 接口说明

参数名称	描述
context_id	被查询的游标ID。
pos	动态定义列在查询中的位置。
column_ref	标记返回的数组类型。
cnt	标记一次获取多少个值。
lower_bnd	标记返回数组时的开始下标。
column_size	列的长度。

- DBE_SQL.SET_RESULTS_TYPE

  该函数用来定义从给定游标返回的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。

  DBE_SQL.SET_RESULTS_TYPE函数的原型为：

```
DBE_SQL.SET_RESULTS_TYPE(
    context_id IN int,
    pos        IN int,
    column_ref IN anyarray,
    cnt        IN int,
    lower_bnd  IN int,
    maxsize    IN int DEFAULT 1024
) returns void;

DBE_SQL.SET_RESULTS_TYPE(
    context_id IN int,
    pos        IN int,
    column_ref IN dbe_sql.number_table,
    cnt        IN int,
    lower_bnd  IN int,
    maxsize    IN int DEFAULT 1024
);

DBE_SQL.SET_RESULTS_TYPE(
    context_id IN int,
    pos        IN int,
    column_ref IN dbe_sql.varchar2_table,
    cnt        IN int,
    lower_bnd  IN int,
    maxsize    IN int DEFAULT 32767
);

DBE_SQL.SET_RESULTS_TYPE(
    context_id IN int,
    pos        IN int,
    column_ref IN dbe_sql.date_table,
    cnt        IN int,
    lower_bnd  IN int,
    maxsize    IN int DEFAULT 1024
);

DBE_SQL.SET_RESULTS_TYPE(
    context_id IN int,
    pos        IN int,
    column_ref IN dbe_sql.blob_table,
    cnt        IN int,
    lower_bnd  IN int,
```



```
    maxsize    IN int DEFAULT 32767
);
```

表 10-266 DBE_SQL.SET_RESULTS_TYPE 接口说明

参数名称	描述
context_id	被查询的游标ID。
pos	动态定义列在查询中的位置。
column_ref	标记返回的数组类型。
cnt	标记一次获取多少个值。
lower_bnd	标记返回数组时的开始下标。
maxsize	定义的类型的最大长度。

说明

DBE_SQL.SET_RESULTS_TYPE不支持用户自定义的table类型，请使用数据类型介绍中提供的table类型。

- DBE_SQL.GET_RESULTS_INT

  该函数用来返回给定游标给定位置的游标INT数组类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  DBE_SQL.GET_RESULTS_INT函数的原型为：

```
DBE_SQL.GET_RESULTS_INT(
    context_id   IN    int,
    pos          IN    int,
    column_value INOUT anyarray
);
```

表 10-267 DBE_SQL.GET_RESULTS_INT 接口说明

参数名称	描述
context_id	被查询的游标ID。
pos	动态定义列在查询中的位置。
column_value	返回值。

- DBE_SQL.GET_RESULTS_TEXT

  该函数用来返回给定游标给定位置的游标TEXT数组类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  DBE_SQL.GET_RESULTS_TEXT函数的原型为：

```
DBE_SQL.GET_RESULTS_TEXT(
    context_id   IN    int,
    pos          IN    int,
    column_value INOUT anyarray
);
```


表 10-268 DBE_SQL.GET_RESULTS_TEXT 接口说明

参数名称	描述
context_id	被查询的游标ID。
pos	动态定义列在查询中的位置。
column_value	返回值。

- DBE_SQL.GET_RESULTS_RAW

  该函数用来返回给定游标给定位置的游标RAW数组类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  DBE_SQL.GET_RESULTS_RAW函数的原型为：

```
DBE_SQL.GET_RESULTS_RAW(
    context_id   IN    int,
    pos          IN    int,
    column_value INOUT anyarray
);
```

表 10-269 DBE_SQL.GET_RESULTS_RAW 接口说明

参数名称	描述
context_id	被查询的游标ID。
pos	动态定义列在查询中的位置。
column_value	返回值。

- DBE_SQL.GET_RESULTS_BYTEA

  该函数用来返回给定游标给定位置的游标BYTEA数组类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  DBE_SQL.GET_RESULTS_BYTEA函数的原型为：

```
DBE_SQL.GET_RESULTS_BYTEA(
    context_id   IN    int,
    pos          IN    int,
    column_value INOUT anyarray
);
```

表 10-270 DBE_SQL.GET_RESULTS_BYTEA 接口说明

参数名称	描述
context_id	被查询的游标ID。
pos	动态定义列在查询中的位置。
column_value	返回值。

- DBE_SQL.GET_RESULTS_CHAR

  该函数用来返回给定游标给定位置的游标CHAR数组类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  DBE_SQL.GET_RESULTS_CHAR函数的原型为：


```
DBE_SQL.GET_RESULTS_CHAR(
   context_id   IN    int,
   pos          IN    int,
   column_value INOUT anyarray
);
```

表 10-271 DBE_SQL.GET_RESULTS_CHAR 接口说明

参数名称	描述
context_id	被查询的游标ID。
pos	动态定义列在查询中的位置。
column_value	返回值。

- DBE_SQL.GET_RESULTS

  该函数用来返回给定游标给定位置的游标数组类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  说明

  由于DBE_SQL.GET_RESULTS底层机制通过数组实现，当用不同的数组获取同一列的返回值时，会由于内部索引的不连续向数组中填充NULL值来确保数组本身索引的连续性，这会导致返回结果数组的长度和Oracle的不一致。

  DBE_SQL.GET_RESULTS函数的原型为：

```
DBE_SQL.GET_RESULTS(
   context_id   IN    int,
   pos          IN    int,
   column_value INOUT anyarray
);

DBE_SQL.GET_RESULTS(
   context_id   IN    int,
   pos          IN    int,
   column_value INOUT dbe_sql.varchar2_table
);

DBE_SQL.GET_RESULTS(
   context_id   IN    int,
   pos          IN    int,
   column_value INOUT dbe_sql.number_table
);

DBE_SQL.GET_RESULTS(
   context_id   IN    int,
   pos          IN    int,
   column_value INOUT dbe_sql.date_table
);

DBE_SQL.GET_RESULTS(
   context_id   IN    int,
   pos          IN    int,
   column_value INOUT dbe_sql.blob_table
);
```

表 10-272 DBE_SQL.GET_RESULTS 接口说明

参数名称	描述
context_id	被查询的游标ID。

参数名称	描述
pos	动态定义列在查询中的位置。
column_value	返回值。

说明

DBE_SQL.GET_RESULTS不支持用户自定义的table类型，请使用**数据类型介绍**中提供的table类型。

- DBE_SQL.SQL_DESCRIBE_COLUMNS

  该函数用来描述列信息，该接口只能应用于SELECT定义的游标中。

  DBE_SQL.SQL_DESCRIBE_COLUMNS函数的原型为：

```
DBE_SQL.SQL_DESCRIBE_COLUMNS(
    context_id IN    int,
    col_cnt    INOUT int,
    desc_t     INOUT dbe_sql.desc_tab
)RETURNS record ;
```

表 10-273 DBE_SQL.SQL_DESCRIBE_COLUMNS 接口说明

参数名称	描述
context_id	被查询的游标ID。
col_cnt	返回的列的数量。
desc_t	返回的列的描述信息。

- DBE_SQL.DESCRIBE_COLUMNS

  该函数用来描述列信息，该接口为兼容接口，只能应用于SELECT定义的游标中。

  DBE_SQL.DESCRIBE_COLUMNS函数的原型为：

```
DBE_SQL.DESCRIBE_COLUMNS(
    context_id IN  int,
    col_cnt    OUT int,
    desc_t     OUT dbe_sql.desc_tab
)
```

表 10-274 DBE_SQL.DESCRIBE_COLUMNS 接口说明

参数名称	描述
context_id	被查询的游标ID。
col_cnt	返回的列的数量。
desc_t	返回的列的描述信息。

- DBE_SQL.BIND_VARIABLE

  该函数是绑定参数接口，建议使用DBE_SQL.SQL_BIND_VARIABLE。

- DBE_SQL.SQL_SET_RESULTS_TYPE_C
  该函数是动态定义一个数组类型的列，不建议用户使用。
  DBE_SQL.SQL_SET_RESULTS_TYPE_C函数的原型为：

```
DBE_SQL.sql_set_results_type_c(
    context_id IN int,
    pos        IN int,
    column_ref IN anyarray,
    cnt        IN int,
    lower_bnd  IN int,
    col_type   IN anyelement,
    maxsize    IN int
)return integer;
```

表 10-275 DBE_SQL.SQL_SET_RESULTS_TYPE_C 接口说明

参数名称	描述
context_id	被查询的游标ID。
pos	动态定义列在查询中的位置。
column_ref	标记返回的数组类型。
cnt	标记一次获取多少个值。
lower_bnd	标记返回数组时的开始下标。
col_type	标记返回的数组类型对应的变量类型。
maxsize	定义的类型的最大长度。

- DBE_SQL.SQL_GET_VALUES_C
  该函数是读取一个已动态定义的列值，不建议用户使用。
  DBE_SQL.SQL_GET_VALUES_C函数的原型为：

```
DBE_SQL.sql_get_values_c(
    context_id   IN    int,
    pos          IN    int,
    results_type INOUT anyarray,
    result_type  IN    anyelement
)return anyarray;
```

表 10-276 DBE_SQL.SQL_GET_VALUES_C 接口说明

参数名称	描述
context_id	被查询的游标ID。
pos	参数位置信息。
results_type	获取的结果。
result_type	获取的结果类型。

- DBE_SQL.GET_VARIABLE_RESULT
  该函数用来返回绑定的OUT参数的值，可以用来获取存储过程中的OUT参数。
  DBE_SQL.GET_VARIABLE_RESULT函数的原型为：


```
DBE_SQL.get_variable_result(
   context_id   IN    int,
   pos          IN    VARCHAR2,
   column_value INOUT anyelement
);
```

表 10-277 DBE_SQL.GET_VARIABLE_RESULT 接口说明

参数名称	描述
context_id	被查询的游标ID。
pos	绑定的参数名。
column_value	返回值。

- DBE_SQL.GET_VARIABLE_RESULT_CHAR

  该函数用来返回绑定的CHAR类型的OUT参数的值，可以用来获取存储过程中的OUT参数。

  DBE_SQL.GET_VARIABLE_RESULT_CHAR函数的原型为：

```
DBE_SQL.get_variable_result_char(
   context_id IN int,
   pos        IN VARCHAR2
)
RETURNS char
```

表 10-278 DBE_SQL.GET_VARIABLE_RESULT_CHAR 接口说明

参数名称	描述
context_id	被查询的游标ID。
pos	绑定的参数名。

- DBE_SQL.GET_VARIABLE_RESULT_RAW

  该函数用来返回绑定的RAW类型的OUT参数的值，可以用来获取存储过程中的OUT参数。

  DBE_SQL.GET_VARIABLE_RESULT_RAW函数的原型为：

```
CREATE OR REPLACE FUNCTION DBE_SQL.get_variable_result_raw(
   context_id IN    int,
   pos        IN    VARCHAR2,
   value      INOUT anyelement
)
RETURNS anyelement
```

表 10-279 DBE_SQL.GET_VARIABLE_RESULT_RAW 接口说明

参数名称	描述
context_id	被查询的游标ID。
pos	绑定的参数名。
value	返回值。

- DBE_SQL.GET_VARIABLE_RESULT_TEXT

  该函数用来返回绑定的TEXT类型的OUT参数的值，可以用来获取存储过程中的OUT参数。

  DBE_SQL.GET_VARIABLE_RESULT_TEXT函数的原型为：

```
CREATE OR REPLACE FUNCTION DBE_SQL.get_variable_result_text(
    context_id IN int,
    pos        IN VARCHAR2
)
RETURNS text
```

  表 10-280 DBE_SQL.GET_VARIABLE_RESULT_TEXT 接口说明

参数名称	描述
context_id	被查询的游标ID。
pos	绑定的参数名。

- DBE_SQL.GET_VARIABLE_RESULT_INT

  该函数用来返回绑定的INT类型的OUT参数的值，可以用来获取存储过程中的OUT参数。

  DBE_SQL.GET_VARIABLE_RESULT_INT函数的原型为：

```
DBE_SQL.get_variable_result_int(
    context_id IN     int,
    pos        IN     VARCHAR2,
    value      INOUT anyelement
)
RETURNS anyelement
```

  表 10-281 DBE_SQL.GET_VARIABLE_RESULT_INT 接口说明

参数名称	描述
context_id	被查询的游标ID。
pos	绑定的参数名。
value	返回值。

- DBE_SQL.GET_ARRAY_RESULT_TEXT

  该函数用来返回绑定的TEXT数组类型的OUT参数的值，可以用来获取存储过程中的OUT参数。

  DBE_SQL.GET_ARRAY_RESULT_TEXT函数的原型为：

```
DBE_SQL.get_array_result_text(
    context_id   IN     int,
    pos          IN     VARCHAR2,
    column_value INOUT anyarray
)
```

  表 10-282 DBE_SQL.GET_ARRAY_RESULT_TEXT 接口说明

参数名称	描述
context_id	被查询的游标ID。

参数名称	描述
pos	绑定的参数名。
column_value	返回值。

- DBE_SQL.GET_ARRAY_RESULT_RAW

  该函数用来返回绑定的RAW数组类型的OUT参数的值，可以用来获取存储过程中的OUT参数。

  DBE_SQL.GET_ARRAY_RESULT_RAW函数的原型为：

```
DBE_SQL.get_array_result_raw(
    context_id   IN    int,
    pos          IN    VARCHAR2,
    column_value INOUT anyarray
)
```

  表 10-283 DBE_SQL.GET_ARRAY_RESULT_RAW 接口说明

参数名称	描述
context_id	被查询的游标ID。
pos	绑定的参数名。
column_value	返回值。

- DBE_SQL.GET_ARRAY_RESULT_CHAR

  该函数用来返回绑定的CHAR数组类型的OUT参数的值，可以用来获取存储过程中的OUT参数。

  DBE_SQL.GET_ARRAY_RESULT_CHAR函数的原型为：

```
DBE_SQL.get_array_result_char(
    context_id   IN    int,
    pos          IN    VARCHAR2,
    column_value INOUT anyarray
)
```

  表 10-284 DBE_SQL.GET_ARRAY_RESULT_CHAR 接口说明

参数名称	描述
context_id	被查询的游标ID。
pos	绑定的参数名。
column_value	返回值。

- DBE_SQL.GET_ARRAY_RESULT_INT

  该函数用来返回绑定的INT数组类型的OUT参数的值，可以用来获取存储过程中的OUT参数。

  DBE_SQL.GET_ARRAY_RESULT_INT函数的原型为：

```
DBE_SQL.get_array_result_int(
    context_id   IN    int,
    pos          IN    VARCHAR2,
```



```
    column_value INOUT anyarray
)
```

表 10-285 DBE_SQL.GET_ARRAY_RESULT_INT 接口说明

参数名称	描述
context_id	被查询的游标ID。
pos	绑定的参数名。
column_value	返回值。

- DBE_SQL.SQL_SET_TABLEOF_RESULTS_TYPE_C

  该函数是动态定义一个tableof类型的列，不建议用户使用。

  DBE_SQL.SQL_SET_TABLEOF_RESULTS_TYPE_C函数的原型为：

```
DBE_SQL.SQL_SET_TABLEOF_RESULTS_TYPE_C(
    context_id IN int,
    pos        IN int,
    column_ref IN anyindexbytable,
    cnt        IN int,
    lower_bnd  IN int,
    col_type   IN anyelement,
    maxsize    IN int
)return integer;
```

表 10-286 DBE_SQL.SQL_SET_TABLEOF_RESULTS_TYPE_C 接口说明

参数名称	描述
context_id	被查询的游标ID。
pos	动态定义列在查询中的位置。
column_ref	标记返回的数组类型。
cnt	标记一次获取多少个值。
lower_bnd	标记返回数组时的开始下标。
col_type	标记返回的数组类型对应的变量类型。
maxsize	定义的类型的最大长度。

- DBE_SQL.SQL_GET_TABLEOF_VALUES_C

  该函数是读取一个已动态定义的tableof类型的列值，不建议用户使用。

  DBE_SQL.SQL_GET_TABLEOF_VALUES_C函数的原型为：

```
DBE_SQL.SQL_GET_TABLEOF_VALUES_C(
    context_id   IN    int,
    pos          IN    int,
    results_type INOUT anyindexbytable,
    result_type  IN    anyelement
)return anyindexbytable;
```

表 10-287 DBE_SQL.SQL_GET_TABLEOF_VALUES_C 接口说明

参数名称	描述
context_id	被查询的游标ID。
pos	参数位置信息。
results_type	获取的结果。
result_type	获取的结果类型。

示例

```
-- 示例1
-- 创建表并插入数据
CREATE TABLE test_desc_cols(
    id NUMBER,
    name VARCHAR2(50)
);
INSERT INTO test_desc_cols(id, name) VALUES (1, 'xiaoming');
INSERT INTO test_desc_cols(id, name) VALUES (2, 'xiaohong');
INSERT INTO test_desc_cols(id, name) VALUES (3, 'xiaolan');

DECLARE
context_id  INTEGER;
col_cnt     INTEGER;
v_id int;
v_name varchar2;
execute_ret  INTEGER;
BEGIN
-- 打开游标
context_id := DBE_SQL.REGISTER_CONTEXT();
-- 编译游标
DBE_SQL.SQL_SET_SQL(context_id, 'SELECT * FROM test_desc_cols', 2);
-- 设置列返回值的类型
DBE_SQL.SET_RESULT_TYPE(context_id, 1, v_id);
DBE_SQL.SET_RESULT_TYPE(context_id, 2, v_name);
execute_ret := DBE_SQL.SQL_RUN(context_id);

loop
exit when (DBE_SQL.NEXT_ROW(context_id) <= 0);
--获取值
DBE_SQL.GET_RESULT(context_id, 1, v_id);
DBE_SQL.GET_RESULT(context_id, 2, v_name);
--输出结果
dbe_output.print_line('id:'|| v_id || ' name:' || v_name);
end loop;

DBE_SQL.SQL_UNREGISTER_CONTEXT(context_id);
END;
/

CREATE OR REPLACE PROCEDURE test_square(n NUMBER, square OUT NUMBER) IS
BEGIN
  square := n * n;
END;
/

DECLARE
cur NUMBER;
query varchar(2000);
ret integer;
n NUMBER;
square Integer;
BEGIN
```



```
  n := 2;
  cur := DBE_SQL.REGISTER_CONTEXT();
  query := 'BEGIN test_square(:n_bnd, :square_bnd); END;';
  DBE_SQL.SQL_SET_SQL(cur, query, 2);
  DBE_SQL.SQL_BIND_VARIABLE(cur, 'n_bnd', n);
  DBE_SQL.SQL_BIND_VARIABLE(cur, 'square_bnd', square);
  ret := DBE_SQL.SQL_RUN(cur);
  DBE_SQL.GET_VARIABLE_RESULT(cur, 'square_bnd', square);
  DBE_OUTPUT.PRINT_LINE('square = ' || square);
  DBE_SQL.SQL_UNREGISTER_CONTEXT(cur);
END;
/

-- 示例2
-- DESCRIBE_COLUMNS、RUN_AND_NEXT和LAST_ROW_COUNT接口示例
-- 创建列描述信息打印存储过程
CREATE OR REPLACE PROCEDURE print_rec(
    rec in DBE_SQL.DESC_REC
)package AS
BEGIN
    raise INFO 'col_type            =    %', rec.col_type;
    raise INFO 'col_name            =    %', rec.col_name;
    raise INFO 'col_name_len        =    %', rec.col_name_len;
END;
/

-- 进行功能验证
DECLARE
context_id  INTEGER;
col_cnt     INTEGER;
rec_tab     DBE_SQL.DESC_TAB;
excute_ret  INTEGER;
nextrow_ret INTEGER;
last_row_count INTEGER;
BEGIN
-- 打开游标
context_id := DBE_SQL.REGISTER_CONTEXT();
-- 编译游标
DBE_SQL.SQL_SET_SQL(context_id, 'SELECT * FROM test_desc_cols', 2);
-- 进行列描述和信息打印
DBE_SQL.DESCRIBE_COLUMNS(context_id, col_cnt, rec_tab);
FOR var IN 1..col_cnt LOOP
    print_rec(rec_tab(var));
END LOOP;
-- 执行并获取一行数据
excute_ret := DBE_SQL.RUN_AND_NEXT(context_id);
-- 获取一行数据
nextrow_ret := DBE_SQL.NEXT_ROW(context_id);
-- 得到目前已经获取的数据行数
last_row_count := DBE_SQL.LAST_ROW_COUNT;
DBE_OUTPUT.PRINT_LINE('last_row_count =  '|| last_row_count);
DBE_SQL.SQL_UNREGISTER_CONTEXT(context_id);
END;
/
```

10.12.2.8 DBE_FILE

DBE_FILE包为存储过程提供了读、写操作系统文本文件的能力。

注意事项

- DBE_FILE要求以DBE_FILE.FOPEN打开的文件是以数据库字符集编码，如果打开的文件未按预期的字符集编码，在使用DBE_FILE.READ_LINE读取文件时，会发生编码校验错误；DBE_FILE要求以DBE_FILE.FOPEN_NCHAR打开的文件是以UTF-8字符集编码，如果打开的文件未按预期的字符集编码，在使用DBE_FILE.READ_LINE_NCHAR读取文件时，会发生编码校验错误。

- 当使用DBE_OUTPUT.PUT_LINE打印DBE_FILE.READ_LINE_NCHAR接口得到的结果时，需要确保UTF-8字符集编码能够转换成当前数据库字符集编码，满足上述条件后可以正常输出结果。DBE_OUTPUT.PRINT_LINE不支持该功能。
- DBE_FILE要求客户端字符集编码与数据库字符集编码保持一致。
- 当数据库字符集编码为ASCII编码、客户端字符集编码为支持中文的编码，且在客户端调用DBE_FILE.WRITE_NCHAR或DBE_FILE.WRITE_LINE_NCHAR接口写入中文相关内容时，若输入的内容为UTF-8编码格式，无法保证写入的内容按UTF-8格式编码。这可能会导致后续使用DBE_FILE.READ_LINE_NCHAR时报错。

数据类型介绍

- DBE_FILE.FILE_TYPE

  DBE_FILE.FILE_TYPE类型定义了DBE_FILE包中文件的表示方式，DBE_FILE.FILE_TYPE中的字段是DBE_FILE包的私有字段，请不要直接修改DBE_FILE.FILE_TYPE类型对象中字段的值。

```
CREATE TYPE DBE_FILE.FILE_TYPE AS(
    id INTEGER,
    datatype INTEGER,
    byte_mode BOOLEAN
);
```

表 10-288 DBE_FILE.FILE_TYPE 字段说明

参数	描述
id	文件句柄。
datatype	表明文件是CHAR文件还是NCHAR文件或者二进制文件，目前支持CHAR文件和NCHAR文件。CHAR文件返回1，NCHAR文件返回2。
byte_mode	表明文件是以二进制模式打开（TRUE）还是以文本模式打开（FALSE）。

接口介绍

高级功能包DBE_FILE支持的所有接口请参见**表10-289**。

表 10-289 DBE_FILE

接口名称	描述
DBE_FILE.OPEN/ DBE_FILE.F...	根据指定的目录和文件名打开一个文件，返回对应的文件句柄或者封装了文件句柄的DBE_FILE.FILE_TYPE类型对象。
DBE_FILE.IS_CLOSE	检测一个文件句柄是否关闭。
DBE_FILE.IS_OPEN	检测一个文件句柄是否打开。
DBE_FILE.READ_LINE	从一个打开的文件句柄中读取一行指定长度的数据。
DBE_FILE.WRITE	将数据写入到一个打开的文件的缓冲区中。

接口名称	描述
DBE_FILE.NEW_LINE	将一个或者多个行终结符写入到一个打开的文件的缓冲区中。
DBE_FILE.WRITE_LINE	将数据写入到一个打开的文件的缓冲区中，并自动追加一个行终结符。
DBE_FILE.FORMAT_WRITE	将数据按指定格式写入到一个打开的文件的缓冲区中。
DBE_FILE.GET_RAW	从一个打开的文件中读取指定字节数的RAW类型数据。
DBE_FILE.PUT_RAW	将RAW类型数据写入到一个打开的文件的缓冲区中。
DBE_FILE.FLUSH	将缓存区中的数据写入到物理文件中。
DBE_FILE.CLOSE	关闭一个打开的文件句柄。
DBE_FILE.CLOSE_ALL	关闭一个会话中打开的所有的文件句柄。
DBE_FILE.REMOVE	根据指定的目录和文件名删除一个磁盘文件，操作的时候需要有充分的权限。
DBE_FILE.RENAME	重命名一个磁盘文件，类似Unix的mv指令。
DBE_FILE.COPY	复制一个连续区域的内容到一个新创建的文件中，如果忽略了start_line和end_line会复制整个文件。
DBE_FILE.GET_ATTR	读取并返回一个磁盘文件的属性。
DBE_FILE.SEEK	根据用户指定的字节数向前或者向后调整文件指针的位置。
DBE_FILE.GET_POS	以字节为单位返回文件当前的偏移量。
DBE_FILE.FOPEN_NCHAR	以NCHAR模式根据指定的目录和文件名打开一个文件。
DBE_FILE.WRITE_NCHAR	将NVARCHAR2类型的数据写入到一个打开的NCHAR模式文件缓冲区中。
DBE_FILE.WRITE_LINE_NCHAR	将NVARCHAR2类型的数据写入到一个打开的NCHAR模式文件缓冲区中，并自动追加一个行终结符。
DBE_FILE.FORMAT_WRITE_NCHAR	将NVARCHAR2类型的数据按指定格式写入到一个打开的NCHAR模式文件缓冲区中。
DBE_FILE.READ_LINE_NCHAR	从一个打开的NCHAR模式文件中读取一行指定长度的数据。

- DBE_FILE.OPEN/DBE_FILE.FOPEN

  函数DBE_FILE.OPEN用来打开一个文件，可以指定文件每行的最大字节数，一个会话内最多可以同时打开50个文件。该函数返回一个INTEGER类型的文件句柄。
  函数DBE_FILE.FOPEN功能和DBE_FILE.OPEN类似，返回一个DBE_FILE.FILE_TYPE类型的对象。

DBE_FILE.OPEN和DBE_FILE.FOPEN函数原型为：

```
DBE_FILE.OPEN(
   dir           IN TEXT,
   file_name     IN TEXT,
   open_mode     IN TEXT,
   max_line_size IN INTEGER DEFAULT 1024)
RETURN INTEGER;

DBE_FILE.FOPEN(
   dir           IN TEXT,
   file_name     IN TEXT,
   open_mode     IN TEXT,
   max_line_size IN INTEGER DEFAULT 1024)
RETURN DBE_FILE.FILE_TYPE;
```

表 10-290 DBE_FILE.OPEN/DBE_FILE.FOPEN 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
dir	TEXT	IN	否	文件的目录位置，这个字符串是一个目录对象名。 **说明** • 文件目录的位置，需要添加到系统表PG_DIRECTORY中，如果传入的路径和PG_DIRECTORY中的路径不匹配，会报路径不存在的错误。 • 在打开guc参数safe_data_path时，用户只能通过高级包读写safe_data_path指定文件路径下的文件。
file_name	TEXT	IN	否	文件名，包含扩展（文件类型），不包括路径名。如果文件名中包含路径，在OPEN中会被忽略，在Unix系统中，文件名不能以/.结尾。
open_mode	TEXT	IN	否	指定文件的打开模式，包含r：read text，w: write text, a: append text, rb: read byte, wb: write byte和ab: append byte。 **说明** 对于写操作，会检测写入文件类型，如果为elf类型文件，会报错退出。
max_line_size	INTEGER	IN	是	每行最大字符数，包含换行符（最小值是1，最大值是32767）。如果没有指定，会指定一个默认值1024。

- DBE_FILE.IS_CLOSE

  函数DBE_FILE.IS_CLOSE用于检测一个文件句柄是否已经关闭，返回布尔值，异常情况是INVALID_FILEHANDLE。

  DBE_FILE.IS_CLOSE函数原型为：

  ```
  DBE_FILE.IS_CLOSE(
     file IN INTEGER)
  ```



```
RETURN BOOLEAN;

DBE_FILE.IS_CLOSE(
    file IN DBE_FILE.FILE_TYPE)
RETURN BOOLEAN;
```

表 10-291 DBE_FILE.IS_CLOSE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER或DBE_FILE.FILE_TYPE	IN	是	待检测的文件句柄或DBE_FILE.FILE_TYPE类型的对象，为空时DBE_FILE.IS_CLOSE接口返回空。

- DBE_FILE.IS_OPEN

  函数DBE_FILE.IS_OPEN用于检测一个文件是否已经打开，返回一个布尔值，异常情况是INVALID_FILEHANDLE。

  DBE_FILE.IS_OPEN函数原型为：

```
DBE_FILE.IS_OPEN(
    file IN INTEGER)
RETURN BOOLEAN;

DBE_FILE.IS_OPEN(
    file IN DBE_FILE.FILE_TYPE)
RETURN BOOLEAN;
```


表 10-292 DBE_FILE.IS_OPEN 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER或DBE_FILE.FILE_TYPE	IN	是	待检测的文件句柄或DBE_FILE.FILE_TYPE类型的对象，为空时DBE_FILE.IS_OPEN接口返回FALSE。

- DBE_FILE.READ_LINE

  存储过程DBE_FILE.READ_LINE从一个打开的文件读取数据，并把读取的结果存放到BUFFER中。读取的时候会读取到行尾，但不包含行终结符，或者读取到文件末尾，或者读取到len参数指定的大小。读取的长度不能超过OPEN的时候指定的max_line_size。

  DBE_FILE.READ_LINE存储过程原型为：

```
DBE_FILE.READ_LINE(
   file   IN  INTEGER,
   buffer OUT TEXT,
   len    IN  INTEGER DEFAULT NULL);

DBE_FILE.READ_LINE(
   file   IN  DBE_FILE.FILE_TYPE,
   buffer OUT TEXT,
   len    IN  INTEGER DEFAULT NULL);
```

表 10-293 DBE_FILE.READ_LINE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER或DBE_FILE.FILE_TYPE	IN	否	通过OPEN打开的文件句柄或者FOPEN打开的DBE_FILE.FILE_TYPE类型的对象，文件必须以读模式打开，否则会抛出INVALID_OPERATION的异常。

参数	类型	入参/出参	是否可以为空	描述
buffer	TEXT	OUT	否	接收数据的BUFFER。
len	INTEGE	IN	是	从文件中读取的字节数，默认值为NULL。如果是NULL，会使用max_line_size来指定大小。

- DBE_FILE.WRITE

  函数DBE_FILE.WRITE用于向文件对应的缓冲区中写入BUFFER中的数据，文件必须以写模式打开，这个操作不会写入行终结符。

  DBE_FILE.WRITE函数原型为：

```
DBE_FILE.WRITE(
    file   IN INTEGER,
    buffer IN TEXT)
RETURN BOOLEAN;

DBE_FILE.WRITE(
    file   IN DBE_FILE.FILE_TYPE,
    buffer IN TEXT)
RETURN VOID;
```

表 10-294 DBE_FILE.WRITE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER或DBE_FILE.FILE_TYPE	IN	否	通过OPEN打开的文件句柄或者FOPEN打开的DBE_FILE.FILE_TYPE类型的对象，要写入的文件必须以写模式打开，这个操作不会写入行终结符。

参数	类型	入参/出参	是否可以为空	描述
buffer	TEXT	IN	是	写入文件的文本数据。每行的累计写入长度不能大于或等于OPEN或FOPEN时指定或默认的max_line_size，否则会在刷新到文件时报错，该参数为空时接口会直接返回。 **说明** 对于写操作，会检测写入文件类型，如果为elf类型文件，会报错退出。

- DBE_FILE.NEW_LINE

  函数DBE_FILE.NEW_LINE用于向文件对应的缓冲区中写入一个或者多个行终结符，行终结符和平台相关。

  DBE_FILE.NEW_LINE函数原型为：

```
DBE_FILE.NEW_LINE(
   file      IN INTEGER,
   line_nums IN INTEGER DEFAULT 1)
RETURN BOOLEAN;

DBE_FILE.NEW_LINE(
   file      IN DBE_FILE.FILE_TYPE,
   line_nums IN INTEGER DEFAULT 1)
RETURN VOID;
```

  表 10-295 DBE_FILE.NEW_LINE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER或DBE_FILE.FILE_TYPE	IN	否	通过OPEN打开的文件句柄或者FOPEN打开的DBE_FILE.FILE_TYPE类型的对象。

参数	类型	入参/出参	是否可以为空	描述
line_nums	INTEGER	IN	是	写入到文件中的行终结符的数量，默认值为1，指定为空时不写入行终结符。

- DBE_FILE.WRITE_LINE

  函数DBE_FILE.WRITE_LINE用于向文件对应的缓冲区中写入BUFFER中的数据，文件必须以写模式打开，这个操作会自动追加行终结符。

  DBE_FILE.WRITE_LINE函数原型为：

```
DBE_FILE.WRITE_LINE(
   file   IN INTEGER,
   buffer IN TEXT,
   flush  IN BOOLEAN DEFAULT FALSE)
RETURN BOOLEAN;

DBE_FILE.WRITE_LINE(
   file   IN DBE_FILE.FILE_TYPE,
   buffer IN TEXT,
   flush  IN BOOLEAN DEFAULT FALSE)
RETURN VOID;
```

  表 10-296 DBE_FILE.WRITE_LINE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	通过OPEN打开的文件句柄或者FOPEN打开的DBE_FILE.FILE_TYPE类型的对象。
buffer	TEXT	IN	是	要写入文件的文本数据，每行的长度（包含换行符）不能大于OPEN或FOPEN时指定或默认的max_line_size，否则会在刷新到文件时报错。 **说明** 对于写操作，会检测写入文件类型，如果为elf类型文件，会报错退出。
flush	BOOLEAN	IN	是	在WRITE_LINE后是否要将文件对应缓冲区中的数据刷到磁盘，默认值或者该参数为空时为FALSE。

- DBE_FILE.FORMAT_WRITE

  函数DBE_FILE.FORMAT_WRITE将格式化数据写入到一个打开的文件对应的缓冲区中，是允许格式化的DBE_FILE.WRITE接口。

  DBE_FILE.FORMAT_WRITE函数原型为：

```
DBE_FILE.FORMAT_WRITE(
   file   IN INTEGER,
   format IN TEXT,
   arg1   IN TEXT DEFAULT NULL,
   . . .
   arg6   IN TEXT DEFAULT NULL)
RETURN BOOLEAN;

DBE_FILE.FORMAT_WRITE(
   file   IN DBE_FILE.FILE_TYPE,
   format IN TEXT,
   arg1   IN TEXT DEFAULT NULL,
   . . .
   arg6   IN TEXT DEFAULT NULL)
RETURN VOID;
```

  表 10-297 DBE_FILE.FORMAT_WRITE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER或DBE_FILE.FILE_TYPE	IN	否	通过OPEN打开的文件句柄或者FOPEN打开的DBE_FILE.FILE_TYPE类型的对象。
format	TEXT	IN	是	格式化的字符串，包含文本和格式符\n和%s。若指定为空时，则不写入任何数据。
[arg1. . .arg6]	TEXT	IN	是	1到6个可选的参数串，参数和格式化字符的位置是一一对应的，如果存在格式化字符而没有提供参数或者参数为空，会使用空串来替代%s。

- DBE_FILE.GET_RAW

  存储过程DBE_FILE.GET_RAW从一个打开的文件读取RAW类型数据，并把读取的结果存放到buffer中，从r中返回。

DBE_FILE.GET_RAW存储过程原型为：

```
DBE_FILE.GET_RAW(
   file   IN  INTEGER,
   r      OUT RAW,
   length IN  INTEGER DEFAULT NULL);

DBE_FILE.GET_RAW(
   file   IN  DBE_FILE.FILE_TYPE,
   r      OUT RAW,
   length IN  INTEGER DEFAULT NULL);
```

表 10-298 DBE_FILE.GET_RAW 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	通过OPEN打开的文件句柄或者FOPEN打开的DBE_FILE.FILE_TYPE类型的对象。
r	RAW	OUT	否	接收RAW类型数据的BUFFER。
length	INTEGER	IN	是	从文件中读取的字节数，默认值为NULL，如果是NULL，会使用RAW类型最大长度来指定大小。

- DBE_FILE.PUT_RAW

  函数DBE_FILE.PUT_RAW用于向文件对应的缓冲区中写入RAW类型数据。

  DBE_FILE.PUT_RAW函数原型为：

```
DBE_FILE.PUT_RAW (
   file  IN INTEGER,
   r     IN RAW,
   flush IN BOOLEAN DEFAULT FALSE)
RETURN BOOLEAN;

DBE_FILE.PUT_RAW (
   file  IN DBE_FILE.FILE_TYPE,
   r     IN RAW,
   flush IN BOOLEAN DEFAULT FALSE)
RETURN VOID;
```


表 10-299 DBE_FILE.PUT_RAW 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER或DBE_FILE.FILE_TYPE	IN	否	通过OPEN打开的文件句柄或者FOPEN打开的DBE_FILE.FILE_TYPE类型的对象。
r	RAW	IN	否	写入文件的RAW类型数据。 **说明** 对于写操作，会检测写入文件类型，如果为elf类型文件，会报错退出。
flush	BOOLEAN	IN	是	在PUT_RAW后是否要刷到磁盘，不指定或指定为空时采用FALSE。

- DBE_FILE.FLUSH

  函数DBE_FILE.FLUSH将缓冲区中的数据写入到物理文件中，缓存中的数据必须要有一个行终结符。该函数可以将缓冲区的数据及时写入到对应的物理文件中。

  DBE_FILE.FLUSH函数原型为：

```
DBE_FILE.FLUSH(
   file IN INTEGER)
RETURN VOID;

DBE_FILE.FLUSH(
   file IN DBE_FILE.FILE_TYPE)
RETURN VOID;
```

表 10-300 DBE_FILE.FLUSH 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER 或 DBE_FILE.FILE_TYPE	IN	否	通过OPEN打开的文件句柄或者FOPEN打开的DBE_FILE.FILE_TYPE类型的对象。

- DBE_FILE.CLOSE

  函数DBE_FILE.CLOSE用于关闭一个打开的文件句柄，当调用这个函数的时候，如果还有等待写入的缓存的数据，可能会收到异常信息，正常关闭返回TRUE。

  DBE_FILE.CLOSE函数原型为：

```
DBE_FILE.CLOSE(
    file IN INTEGER)
RETURN BOOLEAN;

DBE_FILE.CLOSE(
    file IN DBE_FILE.FILE_TYPE)
RETURN BOOLEAN;
```

表 10-301 DBE_FILE.CLOSE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER 或 DBE_FILE.FILE_TYPE	IN	否	通过OPEN打开的文件句柄或者FOPEN打开的DBE_FILE.FILE_TYPE类型的对象。

- DBE_FILE.CLOSE_ALL

  函数DBE_FILE.CLOSE_ALL关闭一个会话中打开的所有的文件句柄，可用于紧急的清理操作。

  DBE_FILE.CLOSE_ALL函数原型为：

```
DBE_FILE.CLOSE_ALL()
RETRUN VOID;
```

- DBE_FILE.REMOVE

  函数DBE_FILE.REMOVE删除一个磁盘文件，使用的时候需要有充分的权限。

  DBE_FILE.REMOVE函数原型为：

```
DBE_FILE.REMOVE(
    dir       IN TEXT,
```



```
    file_name IN TEXT)
RETURN VOID;
```

表 10-302 DBE_FILE.REMOVE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
dir	TEXT	IN	否	文件的目录位置，这个字符串是一个目录对象名。 **说明** • 文件目录的位置，需要添加到系统表PG_DIRECTORY中，如果传入的路径和PG_DIRECTORY中的路径不匹配，会报路径不存在的错误。 • 在打开guc参数safe_data_path时，用户只能通过高级包操作safe_data_path指定文件路径下的文件。
file_name	TEXT	IN	否	文件名。

- DBE_FILE.RENAME

  函数DBE_FILE.RENAME重命名一个磁盘文件，类似Unix的mv指令。

  DBE_FILE.RENAME函数原型为：

```
DBE_FILE.RENAME(
    src_dir        IN TEXT,
    src_file_name  IN TEXT,
    dest_dir       IN TEXT,
    dest_file_name IN TEXT,
    overwrite      IN BOOLEAN DEFAULT FALSE)
RETURN VOID;
```


表 10-303 DBE_FILE.RENAME 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
src_dir	TEXT	IN	否	源文件的目录位置（大小写敏感）。 **说明** • 文件目录的位置，需要添加到系统表PG_DIRECTORY中，如果传入的路径和PG_DIRECTORY中的路径不匹配，会报路径不存在的错误。 • 在打开guc参数safe_data_path时，用户只能通过高级包操作safe_data_path指定文件路径下的文件。
src_file_name	TEXT	IN	否	要进行命名的源文件。
dest_dir	TEXT	IN	否	目的目录位置（大小写敏感）。 **说明** • 文件目录的位置，需要添加到系统表PG_DIRECTORY中，如果传入的路径和PG_DIRECTORY中的路径不匹配，会报路径不存在的错误。 • 在打开guc参数safe_data_path时，用户只能通过高级包操作safe_data_path指定文件路径下的文件。
dest_file_name	TEXT	IN	否	新的文件名。
overwrite	BOOLEAN	IN	是	是否重写，参数指定为空或者不指定时表示不重写。在不重写的情况下，如果目的目录下已存在同名文件会报错。

- DBE_FILE.COPY

  函数DBE_FILE.COPY复制一个连续区域的内容到一个新创建的文件中，如果忽略了start_line和end_line会复制整个文件。

  DBE_FILE.COPY函数原型为：

```
DBE_FILE.COPY(
    src_dir        IN TEXT,
    src_file_name  IN TEXT,
    dest_dir       IN TEXT,
    dest_file_name IN TEXT,
    start_line     IN INTEGER DEFAULT 1,
    end_line       IN INTEGER DEFAULT NULL)
RETURN VOID;
```

表 10-304 DBE_FILE.COPY 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
src_dir	TEXT	IN	否	源文件所在的目录。 **说明** • 文件目录的位置，需要添加到系统表PG_DIRECTORY中，如果传入的路径和PG_DIRECTORY中的路径不匹配，会报路径不存在的错误。 • 在打开guc参数safe_data_path时，用户只能通过高级包操作safe_data_path指定文件路径下的文件。
src_file_name	TEXT	IN	否	要复制的源文件名。
dest_dir	TEXT	IN	否	目的文件所在的目录。 **说明** • 文件目录的位置，需要添加到系统表PG_DIRECTORY中，如果传入的路径和PG_DIRECTORY中的路径不匹配，会报路径不存在的错误。 • 在打开guc参数safe_data_path时，用户只能通过高级包操作safe_data_path指定文件路径下的文件。
dest_file_name	TEXT	IN	否	目的文件名。 **说明** 对于写操作，会检测写入文件类型，如果为elf类型文件，会报错退出。
start_line	INTEGER	IN	否	复制开始的行号，默认为1。
end_line	INTEGER	IN	是	复制结束的行号，默认为NULL，如果是NULL，则指定到文件尾。

- DBE_FILE.GET_ATTR

  存储过程DBE_FILE.GET_ATTR读取并返回一个磁盘文件的属性。

  DBE_FILE.GET_ATTR存储过程原型为：

```
DBE_FILE.GET_ATTR(
    location     IN  TEXT,
    filename     IN  TEXT,
    fexists      OUT BOOLEAN,
```



```
  file_length OUT BIGINT,
  block_size  OUT INTEGER);
```

表 10-305 DBE_FILE.GET_ATTR 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
location	TEXT	IN	否	文件所在的目录。 **说明** • 文件目录的位置，需要添加到系统表PG_DIRECTORY中，如果传入的路径和PG_DIRECTORY中的路径不匹配，会报路径不存在的错误。 • 在打开guc参数safe_data_path时，用户只能通过高级包操作safe_data_path指定文件路径下的文件。
filename	TEXT	IN	否	文件名。
fexists	BOOLEAN	OUT	否	文件是否存在。
file_length	BIGINT	OUT	否	文件的字节长度，如果文件不存在返回NULL。
block_size	INTEGER	OUT	否	文件系统的块大小（单位字节），如果文件不存在返回NULL。

- DBE_FILE.SEEK

  函数DBE_FILE.SEEK根据用户指定的字节数向前或者向后调整文件指针的位置。

  DBE_FILE.SEEK函数原型为：

```
DBE_FILE.SEEK(
   file          IN INTEGER,
   absolute_start IN BIGINT DEFAULT NULL,
   relative_start IN BIGINT DEFAULT NULL)
RETURN VOID;

DBE_FILE.SEEK(
   file          IN DBE_FILE.FILE_TYPE,
   absolute_start IN BIGINT DEFAULT NULL,
   relative_start IN BIGINT DEFAULT NULL)
RETURN VOID;
```


表 10-306 DBE_FILE.SEEK 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER或DBE_FILE.FILE_TYPE	IN	否	通过OPEN打开的文件句柄或者FOPEN打开的DBE_FILE.FILE_TYPE类型的对象。
absolute_start	BIGINT	IN	是	文件偏移的绝对位置，默认值为NULL。
relative_start	BIGINT	IN	是	文件偏移的相对位置。如果值是正数，向前偏移；如果是负数，向后偏移；默认值为NULL。如果和absolute_start参数同时指定，以absolute_start参数为准。

- DBE_FILE.GET_POS

  函数DBE_FILE.GET_POS以字节为单位返回文件当前的偏移量。

  DBE_FILE.FGETPOS函数原型为：

```
DBE_FILE.GET_POS(
    file IN INTEGER)
RETURN BIGINT;

DBE_FILE.GET_POS(
    file IN DBE_FILE.FILE_TYPE)
RETURN BIGINT;
```


表 10-307 DBE_FILE.GET_POS 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER或DBE_FILE.FILE_TYPE	IN	否	通过OPEN打开的文件句柄或者FOPEN打开的DBE_FILE.FILE_TYPE类型的对象。

- DBE_FILE.FOPEN_NCHAR

  函数DBE_FILE.FOPEN_NCHAR用来打开一个文件，可以指定最大行的大小，一个会话内最多可以同时打开50个文件。该函数返回一个DBE_FILE.FILE_TYPE类型的文件句柄。该函数以国家字符集模式打开文件以进行输入或输出。

  DBE_FILE.FOPEN_NCHAR函数原型为：

```
DBE_FILE.FOPEN_NCHAR(
    dir           IN TEXT,
    file_name     IN TEXT,
    open_mode     IN TEXT,
    max_line_size IN INTEGER DEFAULT 1024)
RETURN DBE_FILE.FILE_TYPE;
```


表 10-308 DBE_FILE.FOPEN_NCHAR 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
dir	TEXT	IN	否	文件的目录位置，这个字符串是一个目录对象名。 **说明** • 文件目录的位置，需要添加到系统表PG_DIRECTORY中，如果传入的路径和PG_DIRECTORY中的路径不匹配，会报路径不存在的错误。 • 在打开guc参数safe_data_path时，用户只能通过高级包读写safe_data_path指定文件路径下的文件。
file_name	TEXT	IN	否	文件名，包含扩展（文件类型），不包括路径名。如果文件名中包含路径，在FOPEN_NCHAR中会被忽略，在Unix系统中，文件名不能以/.结尾。
open_mode	TEXT	IN	否	指定文件的打开模式，包含r：read text，w: write text, a: append text, rb: read byte, wb: write byte和ab: append byte。 **说明** 对于写操作，会检测写入文件类型，如果为elf类型文件，会报错退出。
max_line_size	INTEGER	IN	是	每行最大字符数，包含换行符（最小值是1，最大值是32767）。如果没有指定，会指定一个默认值1024。

- DBE_FILE.WRITE_NCHAR

  函数DBE_FILE.WRITE_NCHAR用于向文件中写入buffer中的数据，文件必须以国家字符集模式和写模式打开，这个操作不会写入行终结符，返回值永远为TRUE。文本字符串将以UTF8字符集格式写入。

  DBE_FILE.WRITE_NCHAR函数原型为：

```
DBE_FILE.WRITE_NCHAR(
    file   IN DBE_FILE.FILE_TYPE,
    buffer IN NVARCHAR2)
RETURN VOID;
```

表 10-309 DBE_FILE.WRITE_NCHAR 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	DBE_FILE.FILE_TYPE	IN	否	通过FOPEN_NCHAR打开的DBE_FILE.FILE_TYPE类型的对象，要写入的文件必须以写模式打开，这个操作不会写入行终结符。
buffer	VARCHAR2	IN	是	写入文件的文本数据。每行的累计写入长度不能大于或等于FOPEN_NCHAR时指定或默认的max_line_size，否则会在刷新到文件时报错。 **说明** 对于写操作，会检测写入文件类型，如果为elf类型文件，会报错退出。

- DBE_FILE.WRITE_LINE_NCHAR

  函数DBE_FILE.WRITE_LINE_NCHAR用于向文件中写入buffer中的数据，文件必须以国家字符集模式和写模式打开，这个操作会自动追加行终结符，返回值永远为TRUE。文本字符串将以UTF8字符集格式写入。

  DBE_FILE.WRITE_LINE_NCHAR函数原型为：

```
DBE_FILE.WRITE_LINE_NCHAR(
   file   IN DBE_FILE.FILE_TYPE,
   buffer IN NVARCHAR2)
RETURN VOID;
```


表 10-310 DBE_FILE.WRITE_LINE_NCHAR 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	DBE_FILE.FILE_TYPE	IN	否	通过FOPEN_NCHAR打开的DBE_FILE.FILE_TYPE类型的对象。
buffer	VARCHAR2	IN	是	要写入文件的文本数据，每行的长度（包含换行符）不能大于FOPEN_NCHAR时指定或默认的max_line_size，否则会在刷新到文件时报错。 **说明** 对于写操作，会检测写入文件类型，如果为elf类型文件，会报错退出。

- DBE_FILE.FORMAT_WRITE_NCHAR

  函数DBE_FILE.FORMAT_WRITE_NCHAR将格式化数据写入到一个打开的文件，是允许格式化的DBE_FILE.WRITE_NCHAR接口，返回值永远为TRUE。

  DBE_FILE.FORMAT_WRITE_NCHAR函数原型为：

```
DBE_FILE.FORMAT_WRITE_NCHAR(
   file   IN DBE_FILE.FILE_TYPE,
   format IN NVARCHAR2,
   arg1   IN NVARCHAR2 DEFAULT NULL,
   . . .
   arg5   IN NVARCHAR2 DEFAULT NULL)
RETURN VOID;
```


表 10-311 DBE_FILE.FORMAT_WRITE_NCHAR 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	DBE_FILE.FILE_TYPE	IN	否	通过FOPEN_NCHAR打开的DBE_FILE.FILE_TYPE类型的对象。
format	VARCHAR2	IN	是	格式化的字符串，包含文本和格式符\n和%s。
[arg1...arg5]	VARCHAR2	IN	是	从1到5个可选的参数串，参数和格式化字符的位置是一一对应的，如果存在格式化字符而没有提供参数，会使用空串来替代%s。

- DBE_FILE.READ_LINE_NCHAR

  存储过程DBE_FILE.READ_LINE_NCHAR从一个打开的文件读取数据，并把读取的结果存放到buffer中。读取的时候会读取到行尾，但不包含行终结符，或者读取到文件末尾，或者读取到len参数指定的大小。读取的长度不能超过FOPEN_NCHAR的时候指定的max_line_size。

  DBE_FILE.READ_LINE_NCHAR存储过程原型为：

```
DBE_FILE.READ_LINE_NCHAR(
   file   IN  DBE_FILE.FILE_TYPE,
   buffer OUT NVARCHAR2,
   len    IN  INTEGER DEFAULT NULL);
```


表 10-312 DBE_FILE.READ_LINE_NCHAR 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	DBE_FILE.FILE_TYPE	IN	否	通过FOPEN_NCHAR打开的DBE_FILE.FILE_TYPE类型的对象，文件必须以读模式打开，否则会抛出INVALID_OPERATION的异常。
buffer	VARCHAR2	OUT	否	接收数据的buffer。
len	INTEGER	IN	是	从文件中读取的字节数，默认值为NULL。如果是NULL，会使用max_line_size来指定大小。

示例

```
--系统管理员向PG_DIRECTORY系统表中加入目录/temp/：
CREATE OR REPLACE DIRECTORY dir AS '/tmp/';
-- 预期结果为：
CREATE DIRECTORY

--打开一个文件并向文件中写入数据
DECLARE
f DBE_FILE.FILE_TYPE;
buffer VARCHAR2;
BEGIN
-- 写文件
f := DBE_FILE.FOPEN('dir', 'sample.txt', 'w');
DBE_FILE.WRITE(f, 'A');
DBE_FILE.WRITE(f, 'B');
DBE_FILE.WRITE(f, 'C');
DBE_FILE.NEW_LINE(f);
DBE_FILE.WRITE_LINE(f, 'ABC');
DBE_FILE.FORMAT_WRITE(f, '[1 -> %s, 2 -> %s]\n', 'GaussDB', 'DBE_FILE');
DBE_FILE.CLOSE(f);
-- 读文件
f := DBE_FILE.FOPEN('dir', 'sample.txt', 'r');
DBE_FILE.READ_LINE(f, buffer); -- ABC
DBE_FILE.READ_LINE(f, buffer); -- ABC
DBE_FILE.READ_LINE(f, buffer); -- [1 -> GaussDB, 2 -> DBE_FILE]
DBE_OUTPUT.PRINT_LINE(buffer);
DBE_FILE.CLOSE(f);
END;
/
```



```
-- 预期结果为：
[1 -> GaussDB, 2 -> DBE_FILE]
ANONYMOUS BLOCK EXECUTE

-- 对文件句柄执行位置偏移，并获取文件的当前位置
DECLARE
f DBE_FILE.FILE_TYPE;
buffer VARCHAR2;
pos BIGINT;
BEGIN
-- 读文件
f := DBE_FILE.FOPEN('dir', 'sample.txt', 'r');
DBE_FILE.SEEK(f, 8);
DBE_FILE.READ_LINE(f, buffer); -- [1 -> GaussDB, 2 -> DBE_FILE]
DBE_OUTPUT.PRINT_LINE(buffer);
pos := DBE_FILE.GET_POS(f); -- 38
DBE_OUTPUT.PRINT_LINE(pos);
DBE_FILE.CLOSE(f);
END;
/
-- 预期结果为：
[1 -> GaussDB, 2 -> DBE_FILE]
38
ANONYMOUS BLOCK EXECUTE

-- NCHAR读写相关接口用例
DECLARE
f DBE_FILE.FILE_TYPE;
buffer NVARCHAR2;
BEGIN
-- 写文件
f := DBE_FILE.FOPEN_NCHAR('dir', 'sample02.txt', 'w');
DBE_FILE.WRITE_NCHAR(f, 'A');
DBE_FILE.WRITE_NCHAR(f, 'B');
DBE_FILE.WRITE_NCHAR(f, 'C');
DBE_FILE.NEW_LINE(f);
DBE_FILE.WRITE_LINE_NCHAR(f, 'ABC');
DBE_FILE.FORMAT_WRITE_NCHAR(f, '[1 -> %s, 2 -> %s]\n', 'GaussDB', 'DBE_FILE');
DBE_FILE.CLOSE(f);
-- 读文件
f := DBE_FILE.FOPEN_NCHAR('dir', 'sample02.txt', 'r');
DBE_FILE.READ_LINE_NCHAR(f, buffer); -- ABC
DBE_FILE.READ_LINE_NCHAR(f, buffer); -- ABC
DBE_FILE.READ_LINE_NCHAR(f, buffer); -- [1 -> GaussDB, 2 -> DBE_FILE]
DBE_OUTPUT.PRINT_LINE(buffer);
DBE_FILE.CLOSE(f);
END;
/
-- 预期结果为：
[1 -> GaussDB, 2 -> DBE_FILE]
ANONYMOUS BLOCK EXECUTE
```

10.12.2.9 DBE_UTILITY

接口介绍

高级功能包DBE_UTILITY支持的所有接口请参见**表10-313**。

表 10-313 DBE_UTILITY

接口名称	描述
DBE_UTILITY.FORMAT_ERROR_BACKTRACE	输出存储过程异常的调用堆栈。

接口名称	描述
DBE_UTILITY.FORMAT_ERROR_STACK	输出存储过程异常的具体信息。
DBE_UTILITY.FORMAT_CALL_STACK	输出存储过程的调用堆栈。
DBE_UTILITY.GET_TIME	输出当前时间，一般用于做差得到执行时长。
DBE_UTILITY.CANONICALIZE	用于给表名字符串做规范。
DBE_UTILITY.COMMA_TO_TABLE	将用逗号隔开的名字列表的字符串转换为PL/SQL表名列表。
DBE_UTILITY.DB_VERSION	返回数据库的版本号和兼容性版本号。
DBE_UTILITY.EXEC_DDL_STATEMENT	用于执行用户输入的DDL语句。
DBE_UTILITY.EXPAND_SQL_TEXT_PROC	用于展开SQL查询的视图。
DBE_UTILITY.GET_CPU_TIME	返回当前CPU处理时间的测量值。
DBE_UTILITY.GET_ENDIANNESS	用于获取数据库所在平台字节序的大小端信息。
DBE_UTILITY.GET_HASH_VALUE	返回一个给定字符串的hash值。
DBE_UTILITY.GET_SQL_HASH	输出一个给定字符串的hash值，该存储过程在不打开proc_outparam_override时使用。
DBE_UTILITY.IS_BIT_SET	用于检查参数n是否存在于r。
DBE_UTILITY.IS_CLUSTER_DATABASE	用于判断当前数据库是否在数据库集群模式下运行。
DBE_UTILITY.NAME_RESOLVE	解析给定的对象名称，包括同义词翻译和必要的授权检查。
DBE_UTILITY.NAME_TOKENIZE	用于解析a [. b [. c]][@ dblink]形式的名字。
DBE_UTILITY.OLD_CURRENT_SCHEMA	返回当前用户环境下的数据库模式名称。
DBE_UTILITY.OLD_CURRENT_USER	返回当前用户的名称。
DBE_UTILITY.TABLE_TO_COMMA	将PL/SQL中的表名转换为用逗号隔开的名字列表的字符串。
DBE_UTILITY.GET_SQL_HASH_FUNC	功能同DBE_UTILITY.GET_SQL_HASH，该函数在打开proc_outparam_override时使用。
DBE_UTILITY.EXPAND_SQL_TEXT	内部函数，不建议用户使用。
DBE_UTILITY.CANONICALIZE_RET	内部函数，不建议用户使用。

接口名称	描述
DBE_UTILITY.COMMA_TO_TABLE_FUN	内部函数，不建议用户使用。
DBE_UTILITY.COMPILE_SCHEMA	内部函数，不建议用户使用，推荐使用pkg_util.gs_compile_schema。
DBE_UTILITY.NAME_SEPARATE	内部函数，不建议用户使用。
DBE_UTILITY.NAME_TOKENIZE_FUNC	内部函数，不建议用户使用。
DBE_UTILITY.NAME_TOKENIZE_LOWER	内部函数，不建议用户使用。
DBE_UTILITY.NAME_TOKENIZE_LOWER_FUNC	内部函数，不建议用户使用。
DBE_UTILITY.PRIVILEGE_CHECK	内部函数，不建议用户使用。
DBE_UTILITY.SEARCH_CLASS_WITH_NSPOID_ONAME_TYPE	内部函数，不建议用户使用。
DBE_UTILITY.SEARCH_OBJECTS	内部函数，不建议用户使用。
DBE_UTILITY.SEARCH_OBJECTS_SYNONYM_FILL_SECHEMA	内部函数，不建议用户使用。
DBE_UTILITY.SEARCH_PROCEDURE_WITH_NSPOID_ONAME	内部函数，不建议用户使用。
DBE_UTILITY.SEARCH_SYNONM_WITH_NSPOID_ONAME	内部函数，不建议用户使用。
DBE_UTILITY.TABLE_TO_COMMA_FUNC	内部函数，不建议用户使用。
DBE_UTILITY.USER_NAME	内部函数，不建议用户使用。

- DBE_UTILITY.FORMAT_ERROR_BACKTRACE

  存储过程FORMAT_ERROR_BACKTRACE返回在执行过程中出现错误时，出现错误位置的调用堆栈。DBE_UTILITY.FORMAT_ERROR_BACKTRACE函数原型为：

  ```
  DBE_UTILITY.FORMAT_ERROR_BACKTRACE()
  RETURN TEXT;
  ```

- DBE_UTILITY.FORMAT_ERROR_STACK

  存储过程FORMAT_ERROR_STACK返回在执行过程中出现错误时，出现错误位置的具体信息。DBE_UTILITY.FORMAT_ERROR_STACK函数原型为：

  ```
  DBE_UTILITY.FORMAT_ERROR_STACK()
  RETURN TEXT;
  ```

- DBE_UTILITY.FORMAT_CALL_STACK

  存储过程FORMAT_CALL_STACK设置输出函数调用堆栈。
  DBE_UTILITY.FORMAT_CALL_STACK函数原型为：

  ```
  DBE_UTILITY.FORMAT_CALL_STACK()
  RETURN TEXT;
  ```

- DBE_UTILITY.COMPILE_SCHEMA

重编译指定schema下的PL/SQL类型包和函数（系统自带的包和函数除外），DBE_UTILITY.COMPILE_SCHEMA函数原型为：

```
DBE_UTILITY.COMPILE_SCHEMA (
SCHEMA IN VARCHAR2,
COMPILE_ALL IN BOOLEAN DEFAULT TRUE,
REUSE_SETTINGS IN BOOLEAN DEFAULT FALSE
)
RETURNS VOID;
示例参考11.12.1.2中pkg_util.utility_compile_schema函数使用方式， 调用处改为：
call DBE_UTILITY.compile_schema('pkg_var_test');
```

- DBE_UTILITY.GET_TIME

  存储过程GET_TIME设置输出时间，通常用于做差，单独的返回值没有意义。DBE_UTILITY.GET_TIME函数原型为：

  ```
  DBE_UTILITY.GET_TIME()
  RETURN BIGINT;
  ```

- DBE_UTILITY.CANONICALIZE

  存储过程CANONICALIZE用于给表名字符串做规范。该过程处理单个保留字或关键字，并删除单个标识符的空白，以便“table”成为TABLE。DBE_UTILITY.CANONICALIZE函数原型为：

  ```
  DBE_UTILITY.CANONICALIZE(
    name       IN   VARCHAR2,
    canon_name OUT  VARCHAR2,
    canon_len  IN   BINARY_INTEGER DEFAULT 1024
  );
  ```

  表 10-314 DBE_UTILITY.CANONICALIZE 接口说明

参数名称	类型	入参/出参	是否可以为空	描述
name	VARCHAR2	IN	否	待规范的字符串。
canon_name	VARCHAR2	OUT	是	规范好的字符串。
canon_len	BINARY_INTEGER	IN	是	要规范的字符串长度（以字节为单位）。若此参数的值小于待规范的字符串的实际长度（字节），则会以字节为粒度截断字符串。

- DBE_UTILITY.COMMA_TO_TABLE

  存储过程COMMA_TO_TABLE将用逗号隔开的名字列表的字符串转换为PL/SQL表名列表。DBE_UTILITY.COMMA_TO_TABLE函数原型为：

```
DBE_UTILITY.COMMA_TO_TABLE (
   list   IN  VARCHAR2,
   tablen OUT BINARY_INTEGER,
   tab    OUT VARCHAR2[]
);
```

  表 10-315 DBE_UTILITY.COMMA_TO_TABLE 接口说明

参数名称	类型	入参/出参	是否可以为空	描述
list	VARCHAR2	IN	否	逗号隔开的名字列表字符串。
tablen	BINARY_INTEGER	OUT	是	列表名字的个数。
tab	VARCHAR2	OUT	是	转换后的表名称列表。

- DBE_UTILITY.DB_VERSION

  存储过程DB_VERSION返回数据库的版本号和兼容性版本号。
  DBE_UTILITY.DB_VERSION函数原型为：

```
DBE_UTILITY.DB_VERSION (
   version OUT VARCHAR2
);
```


表 10-316 DBE_UTILITY.DB_VERSION 接口说明

参数名称	类型	入参/出参	是否可以为空	描述
version	VARCHAR2	OUT	否	输出参数，表示内部的数据库软件版本信息，是一个字符串。

- DBE_UTILITY.EXEC_DDL_STATEMENT

  存储过程EXEC_DDL_STATEMENT用于执行用户输入的DDL语句。DBE_UTILITY.EXEC_DDL_STATEMENT函数原型为：

```
DBE_UTILITY. EXEC_DDL_STATEMENT (
    parse_string IN TEXT
);
```

表 10-317 DBE_UTILITY.EXEC_DDL_STATEMENT 接口说明

参数名称	类型	入参/出参	是否可以为空	描述
parse_string	TEXT	IN	是	需要执行的DDL语句。

- DBE_UTILITY.EXPAND_SQL_TEXT_PROC

  存储过程EXPAND_SQL_TEXT_PROC用于展开SQL查询的视图，会递归展开视图中的视图对象，一直展开显示到表。DBE_UTILITY.EXPAND_SQL_TEXT_PROC函数原型为：

```
DBE_UTILITY.EXPAND_SQL_TEXT_PROC (
    input_sql_text  IN  CLOB,
    output_sql_text OUT CLOB
);
```


表 10-318 DBE_UTILITY.EXPAND_SQL_TEXT_PROC 接口说明

参数名称	类型	入参/出参	是否可以为空	描述
input_sql_text	CLOB	IN	否	输入的SQL文本。
output_sql_text	CLOB	OUT	否	输出展开视图的SQL文本。

说明

用户输入的input_sql_text参数中，SQL语句中的对象需要增加schema前缀，否则函数会报无法找到对象的错误。若设置set behavior_compat_options参数值为bind_procedure_searchpath，则无需强制指定schema前缀。

- DBE_UTILITY.GET_CPU_TIME

  存储过程GET_CPU_TIME返回当前CPU处理时间的测量值（以百分之一秒为单位）。DBE_UTILITY.GET_CPU_TIME函数原型为：

  ```
  DBE_UTILITY.GET_CPU_TIME()
  RETURN NUMBER;
  ```

- DBE_UTILITY.GET_ENDIANNESS

  存储过程GET_ENDIANNESS用于获取数据库所在平台字节序的大小端信息。DBE_UTILITY.GET_ENDIANNESS函数原型为：

  ```
  DBE_UTILITY.GET_ENDIANNESS
  RETURN INTEGER;
  ```

- DBE_UTILITY.GET_HASH_VALUE

  存储过程GET_HASH_VALUE返回一个给定字符串的hash值。DBE_UTILITY.GET_HASH_VALUE函数原型为：

  ```
  DBE_UTILITY.GET_HASH_VALUE(
     name      IN VARCHAR2(n),
     base      IN INTEGER,
     hash_size IN INTEGER)
  RETURN INTEGER;
  ```


表 10-319 DBE_UTILITY.GET_HASH_VALUE 接口说明

参数名称	类型	入参/出参	是否可以为空	描述
name	VARCHAR2	IN	否	待哈希转换的字符串。
base	INTEGER	IN	否	返回的hash值的起始值。
hash_size	INTEGER	IN	否	哈希映射到的哈希表的大小。

- DBE_UTILITY.GET_SQL_HASH

  存储过程GET_SQL_HASH通过MD5算法输出一个给定字符串的hash值。
  DBE_UTILITY.GET_SQL_HASH函数原型为：

```
DBE_UTILITY.GET_SQL_HASH(
    name       IN  VARCHAR2,
    hash       OUT RAW,
    last4bytes OUT BIGINT
);
```

表 10-320 DBE_UTILITY.GET_SQL_HASH 接口说明

参数名称	类型	入参/出参	是否可以为空	描述
name	VARCHAR2	IN	否	待哈希转换的字符串。
hash	RAW	OUT	否	完整的16进制MD5哈希值。

参数名称	类型	入参/出参	是否可以为空	描述
last4bytes	BIGINT	OUT	否	MD5哈希值的最后四字节，以无符号整数形式展现。

说明

设置set behavior_compat_options为非proc_outparam_override参数后（参数设置请联系管理员处理），请调用DBE_UTILITY.GET_SQL_HASH函数，如调用DBE_UTILITY.GET_SQL_HASH_FUNC则会发生赋值不成功。

- DBE_UTILITY.IS_BIT_SET

  存储过程IS_BIT_SET用于检查参数n是否存在于r。DBE_UTILITY.IS_BIT_SET函数原型为：

```
DBE_UTILITY.IS_BIT_SET (
   r IN RAW,
   n IN INTEGER)
RETURN INTEGER;
```

表 10-321 DBE_UTILITY.IS_BIT_SET 接口说明

参数名称	类型	入参/出参	是否可以为空	描述
r	RAW	IN	否	4字节加上实际的十六进制字符串。
n	INTEGER	IN	否	用于在二进制中判断是否存在该数值。

- DBE_UTILITY.IS_CLUSTER_DATABASE

  存储过程IS_CLUSTER_DATABASE用于判断当前数据库是否在数据库集群模式下运行。DBE_UTILITY.IS_CLUSTER_DATABASE函数原型为：

```
DBE_UTILITY.IS_CLUSTER_DATABASE
RETURN BOOLEAN;
```

- DBE_UTILITY.NAME_RESOLVE

  存储过程NAME_RESOLVE解析给定的对象名称，包括同义词翻译和必要的授权检查。DBE_UTILITY.NAME_RESOLVE函数原型为：

```
DBE_UTILITY.NAME_RESOLVE (
   name          IN   VARCHAR2,
```



```
  context       IN   INTEGER,
  schema         OUT VARCHAR2,
  part1         OUT VARCHAR2,
  part2         OUT VARCHAR2,
  dblink        OUT VARCHAR2,
  part1_type    OUT INTEGER,
  object_number OUT OID
);
```

表 10-322 DBE_UTILITY.NAME_RESOLVE 接口说明

参数名称	类型	入参/出参	是否可以为空	描述
name	VARCHAR2	IN	否	待解析对象名，结构为[[a.]b.]c[@d]。
context	INTEGER	IN	否	返回的hash值的起始值。
schema	VARCHAR2	OUT	否	对象所在的模式。
part1	VARCHAR2	OUT	否	名称的第一部分，该字段的类型由part1_type指定。
part2	VARCHAR2	OUT	是	如果该字段不为空，则为子程序名称。
dblink	VARCHAR2	OUT	是	数据库链接。

参数名称	类型	入参/出参	是否可以为空	描述
part1_type	INTEGER	OUT	否	part1的类型： ● 5：synonym ● 7：procedure(top level) ● 8：function(top level) ● 9：package
object_number	OID	OUT	否	对象标识。object_number在A数据库中类型为Number，表示对象标识，而GaussDB中对象标识类型为OID，并且不支持Number到OID的隐式转换。

- DBE_UTILITY.NAME_TOKENIZE

  存储过程NAME_TOKENIZE用于解析a [. b [. c]][@ dblink]形式的名字，如果名字带有双引号则去掉，否则变为大写字母。DBE_UTILITY.NAME_TOKENIZE函数原型为：

```
DBE_UTILITY.NAME_TOKENIZE (
name    IN  VARCHAR2,
a       OUT VARCHAR2,
b       OUT VARCHAR2,
c       OUT VARCHAR2,
dblink  OUT VARCHAR2,
nextpos OUT INTEGER
);
```

表 10-323 DBE_UTILITY.NAME_TOKENIZE 接口说明

参数名称	类型	入参/出参	是否可以为空	描述
name	VARCHAR2	IN	否	名字，由SQL标识符组成（比如, scott.foo@dblink）。
a	VARCHAR2	OUT	否	名字的第一个token。

参数名称	类型	入参/出参	是否可以为空	描述
b	VARCHAR2	OUT	是	名字的第二个token。
c	VARCHAR2	OUT	是	名字的第三个token。
dblink	VARCHAR2	OUT	是	数据库链接。
nextpos	INTEGER	OUT	否	字符串经过解析后的下一个位置。

- DBE_UTILITY.OLD_CURRENT_SCHEMA

  存储过程OLD_CURRENT_SCHEMA返回当前用户环境下的数据库模式名称。DBE_UTILITY.OLD_CURRENT_SCHEMA函数原型为：

  ```
  DBE_UTILITY.OLD_CURRENT_SCHEMA()
  RETURN VARCHAR;
  ```

- DBE_UTILITY.OLD_CURRENT_USER

  存储过程OLD_CURRENT_USER返回当前用户的名称。DBE_UTILITY.OLD_CURRENT_USER函数原型为：

  ```
  DBE_UTILITY.OLD_CURRENT_USER()
  RETURN TEXT;
  ```

- DBE_UTILITY.TABLE_TO_COMMA

  存储过程TABLE_TO_COMMA将PL/SQL中的表名转换为用逗号隔开的名字列表的字符串。DBE_UTILITY.TABLE_TO_COMMA函数原型为：

  ```
  DBE_UTILITY.TABLE_TO_COMMA (
     tab    IN  VARCHAR2[],
     tablen OUT BINARY_INTEGER,
     list   OUT VARCHAR2
  );
  ```


表 10-324 DBE_UTILITY.TABLE_TO_COMMA 接口说明

参数名称	类型	入参/出参	是否可以为空	描述
tab	VARCHAR2[]	IN	否	包含表名的结构表数组。
tablen	BINARY_INTEGER	OUT	否	结构表中的表个数。
list	VARCHAR2	OUT	否	以逗号分隔表名的字符串。

- DBE_UTILITY.GET_SQL_HASH_FUNC

  函数GET_SQL_HASH_FUNC通过MD5算法输出一个给定字符串的hash值。
  DBE_UTILITY.GET_SQL_HASH_FUNC函数原型为：

```
DBE_UTILITY.GET_SQL_HASH_FUNC(
    name       IN  VARCHAR2,
    hash       OUT RAW,
    last4bytes OUT BIGINT
);
```

表 10-325 DBE_UTILITY.GET_SQL_HASH_FUNC 接口说明

参数名称	类型	入参/出参	是否可以为空	描述
name	VARCHAR2	IN	否	待哈希转换的字符串。

参数名称	类型	入参/出参	是否可以为空	描述
hash	RAW	OUT	否	完整的16进制MD5哈希值。
last4bytes	BIGINT	OUT	否	MD5哈希值的最后四字节，以无符号整数形式展现。

说明

设置set behavior_compat_options = 'proc_outparam_override' 参数后，请调用DBE_UTILITY.GET_SQL_HASH_FUNC函数，如调用DBE_UTILITY.GET_SQL_HASH则会报参数不匹配异常。

示例

```
CREATE OR REPLACE PROCEDURE test_get_time1()
AS
declare
    start_time  bigint;
    end_time  bigint;
BEGIN
    start_time:= dbe_utility.get_time ();
    pg_sleep(1);
    end_time:=dbe_utility.get_time ();
    dbe_output.print_line(end_time - start_time);
END;
/
CREATE PROCEDURE

-- 给表名字符串做规范
declare
    cname varchar2(50);
begin
    dbe_utility.canonicalize('seg1', cname, 50);
    dbe_output.put_line(cname);
end;
/
-- 预期结果为：
SEG1
ANONYMOUS BLOCK EXECUTE

-- 将输入的字符串转换成一个表名的数组
DECLARE
 tab_list VARCHAR2(100) := 't1,t2';
 len BINARY_INTEGER;
 tab varchar2[];
BEGIN
 dbe_output.put_line('table list is ： ' || tab_list);
 dbe_utility.comma_to_table(tab_list, len, tab);
END;
/
-- 预期结果为：
table list is: t1,t2
```



```
ANONYMOUS BLOCK EXECUTE

-- 查看数据库的版本号和兼容性版本号
declare
    v_version varchar2;
begin
    dbe_utility.db_version(v_version);
    v_version:=left(v_version, 8);
    dbe_output.print_line('version:' || v_version);
end;
/
-- 预期结果为：
version:gaussdb
ANONYMOUS BLOCK EXECUTE

-- 查看当前CPU处理时间的测量值
DECLARE
    cputime NUMBER;
BEGIN
    cputime := dbe_utility.get_cpu_time;
    dbe_output.put_line('cpu time：' || cputime);
END;
/
-- 预期结果为（数值并非固定）：
cpu time：9851
ANONYMOUS BLOCK EXECUTE

-- 获取数据库所在平台字节序的大小端信息
BEGIN
    dbe_output.PUT_LINE(dbe_utility.GET_ENDIANNESS);
END;
/
-- 预期结果为：
2
ANONYMOUS BLOCK EXECUTE

-- 获取一个给定字符串的hash值
DECLARE
    result NUMBER(28);
BEGIN
    result := dbe_utility.get_hash_value('hello',10,10);
    dbe_output.put_line(result);
END;
/
-- 预期结果为：
11
ANONYMOUS BLOCK EXECUTE

-- 判断当前数据库是否为集群模式
DECLARE
    is_cluster BOOLEAN;
BEGIN
    is_cluster := dbe_utility.IS_CLUSTER_DATABASE;
    dbe_output.put_line('CLUSTER DATABASE: ' || CASE WHEN is_cluster THEN 'TRUE' ELSE 'FALSE' END);
END;
/
-- 预期结果为：
CLUSTER DATABASE: FALSE
ANONYMOUS BLOCK EXECUTE

-- 获取当前用户环境下的数据库模式名称
DECLARE
    schm varchar2(100);
BEGIN
    schm := dbe_utility.old_current_schema;
    dbe_output.put_line('current schema: ' || schm);
END;
/
-- 预期结果为（结果为当前数据库的模式名，并非固定）：
```



```
current schema: public
ANONYMOUS BLOCK EXECUTE

-- 获取当前用户名称
select dbe_utility.old_current_user from sys_dummy;
-- 预期结果为（结果为当前数据库的用户名，并非固定）：
 old_current_user
------------------
 admin
(1 row)
```

10.12.2.10 DBE_SESSION

接口介绍

高级功能包DBE_SESSION支持的所有接口请参见**表10-326**。DBE_SESSION作用范围是会话级别。

表 10-326 DBE_SESSION

接口名称	描述
DBE_SESSION.SET_CONTEXT	设置指定context下，某一属性(attribute)的值(value)。
DBE_SESSION.CLEAR_CONTEXT	清除指定context下，某一属性(attribute)的值(value)。
DBE_SESSION.SEARCH_CONTEXT	查找指定context下，某一属性(attribute)的值(value)。

- DBE_SESSION.SET_CONTEXT

  向指定namespace(context)下，设置某一属性(attribute)的值(value)。
  DBE_SESSION.SET_CONTEXT函数原型为：

  ```
  DBE_SESSION.SET_CONTEXT(
      namespace text,
      attribute text,
      value text
  )returns void;
  ```

  表 10-327 DBE_SESSION.SET_CONTEXT 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
namespace	TEXT	IN	否	需要设置的context名称，当context不存在时，新建context，最长支持128个字节，超长时将会截断。

参数	类型	入参/出参	是否可以为空	描述
attribute	TEXT	IN	否	属性名称，最长支持1024个字节，超长时将会截断。
value	TEXT	IN	否	要设置的值的名称，最长支持1024个字节，超长时将会截断。

- DBE_SESSION.CLEAR_CONTEXT

  清除指定namespace(context)下，某一属性(attribute)的值(value)。
  DBE_SESSION.CLEAR_CONTEXT函数原型为：

```
DBE_SESSION.CLEAR_CONTEXT (
    namespace text,
    client_identifier text default null,
    attribute text default null
)returns void ;
```

表 10-328 DBE_SESSION.CLEAR_CONTEXT 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
namespace	TEXT	IN	否	用户指定的context，最长支持128个字节，超长时将会截断。
client_identifier	TEXT	IN	是	客户端认证，默认值null，通常情况用户无需手动设置。
attribute	TEXT	IN	是	要清除的属性，默认值null，表示清除指定context的所有的属性，最长支持1024个字节，超长时将会截断。 **注意** 为了保证前向兼容，参数值为字符串'null'时也会清除指定context的所有属性。

- DBE_SESSION.SEARCH_CONTEXT

  查找指定namespace(context)下，某一属性(attribute)的值(value)。
  DBE_SESSION.SEARCH_CONTEXT函数原型为：

```
DBE_SESSION.SEARCH_CONTEXT (
    namespace text,
    attribute text
)returns text;
```


表 10-329 DBE_SESSION.SEARCH_CONTEXT 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
namespace	TEXT	IN	否	用户指定的context，最长支持128个字节，超长时将会截断。
attribute	TEXT	IN	否	要查找的属性，最长支持1024个字节，超长时将会截断。

示例

```
DECLARE
    a text;
BEGIN
    DBE_SESSION.set_context('test', 'gaussdb', 'one');   --设置名为test的context下属性为gaussdb的值为one
    a := DBE_SESSION.search_context('test', 'gaussdb');
    DBE_OUTPUT.PRINT_LINE(a);
    DBE_SESSION.clear_context('test', 'test','gaussdb');
END;
/
-- 预期结果为：
one
ANONYMOUS BLOCK EXECUTE
```

10.12.2.11 DBE_MATCH

接口介绍

高级功能包DBE_MATCH支持的所有接口请参见**表10-330**。

表 10-330 DBE_MATCH

接口名称	描述
DBE_MATCH.EDIT_DISTANCE_SIMILARITY	比较两个字符串的差距(删除、新增、变换的最小步骤)，并归一化到0-100（100表示完全一致，0表示完全不一致）。

- DBE_MATCH.EDIT_DISTANCE_SIMILARITY

  比较两个字符串的差距(删除、新增、变换的最小步骤)，并归一化到0-100（100表示完全一致，0表示完全不一致）,DBE_MATCH.EDIT_DISTANCE_SIMILARITY函数原型为：

  ```
  DBE_MATCH.EDIT_DISTANCE_SIMILARITY(
      str1 IN text,
      str2 IN text
  )returns integer ;
  ```


表 10-331 DBE_MATCH.EDIT_DISTANCE_SIMILARITY 接口参数说明

参数	描述
str1	第一个字符串，如果为null，直接输出0。
str2	第二个字符串，如果为null，直接输出0。

10.12.2.12 DBE_APPLICATION_INFO

接口介绍

高级功能包DBE_APPLICATION_INFO支持的所有接口请参见**表10-332**。
DBE_APPLICATION_INFO作用范围是当前session。

表 10-332 DBE_APPLICATION_INFO

接口名称	描述
DBE_APPLICATION_INFO.SET_CLIENT_INFO	写入客户端信息。
DBE_APPLICATION_INFO.READ_CLIENT_INFO	读取客户端信息。

- DBE_APPLICATION_INFO.SET_CLIENT_INFO
  写入客户端信息。DBE_APPLICATION_INFO.SET_CLIENT_INFO函数原型为：

```
DBE_APPLICATION_INFO.SET_CLIENT_INFO(
    str text
)returns void;
```

表 10-333 DBE_APPLICATION_INFO.SET_CLIENT_INFO 接口参数说明

参数	描述
str	写入的客户端信息。

- DBE_APPLICATION_INFO.READ_CLIENT_INFO
  读取客户端信息DBE_APPLICATION_INFO.READ_CLIENT_INFO函数原型为：

```
DBE_APPLICATION_INFO.READ_CLIENT_INFO(
 OUT client_info text);
```

表 10-334 DBE_APPLICATION_INFO.READ_CLIENT_INFO 接口参数说明

参数	描述
client_info	客户端信息

10.12.2.13 DBE_XMLDOM

接口介绍

高级功能包DBE_XMLDOM用于访问XMLType对象，实现DOM(Document Object Model)，一个用于访问HTML和XML DOCUMENTS API。高级功能包DBE_XMLDOM支持的所有类型请参见 **表10-335**，DBE_XMLDOM支持的所有接口请参见 **表10-336**。

说明

当在数据库的字符集设置为SQL_ASCII的情况下使用DBE_XMLDOM高级包，传入超出ASCII范围的字符时，会产生报错信息。

表 10-335 DBE_XMLDOM 数据类型说明

类型名称	描述
DOMATTR	实现DOM Attribute接口。
DOMDOCUMENT	实现DOM Document接口。
DOMELEMENT	实现DOM Element接口。
DOMNAMEDNODEMAP	实现DOM Named Node Map接口。
DOMNODELIST	实现DOM Node List接口。
DOMNODE	实现DOM Node接口。
DOMTEXT	实现DOM Text接口。

表 10-336 DBE_XMLDOM 接口参数说明

接口名称	描述
DBE_XMLDOM.APPENDCHILD	将newchild node添加到parent(n)节点最后面，并返回新添加的Node节点。
DBE_XMLDOM.CREATEELEMENT	创建指定名称的DOMELEMENT对象。
DBE_XMLDOM.CREATETEXTNODE	创建DOMTEXT节点。
DBE_XMLDOM.FREEDOCUMENT	释放DOMDOCUMENT节点相关资源。
DBE_XMLDOM.FREEELEMENT	释放DOMELEMENT节点相关资源。
DBE_XMLDOM.FREENODE	释放DOMNODE节点相关资源。
DBE_XMLDOM.FREENODELIST	释放DOMNODELIST节点相关资源。
DBE_XMLDOM.GETATTRIBUTE	按名称返回DOMELEMENT属性的值。
DBE_XMLDOM.GETATTRIBUTES	将DOMNODE节点属性值作为map返回。
DBE_XMLDOM.GETCHILDNODES	将节点下的若干子节点转换成节点列表。

接口名称	描述
DBE_XMLDOM.GETCHILDRENBYTAGNAME	按名称返回DOMELEMENT的子节点。
DBE_XMLDOM.GETDOCUMENTELEMENT	返回指定DOCUMENT的首个子节点。
DBE_XMLDOM.GETFIRSTCHILD	返回第一个子节点。
DBE_XMLDOM.GETLASTCHILD	返回最后一个子节点。
DBE_XMLDOM.GETLENGTH	获取给定节点中的节点个数。
DBE_XMLDOM.GETLOCALNAME	检索节点的本地名称。
DBE_XMLDOM.GETNAMEDITEM	检索由名称指定的节点。
DBE_XMLDOM.GETNEXTSIBLING	返回该节点的下一个节点。
DBE_XMLDOM.GETNODENAME	返回节点名称。
DBE_XMLDOM.GETNODETYPE	返回节点类型。
DBE_XMLDOM.GETNODEVALUE	此函数用于获取节点的值，具体取决于其类型。
DBE_XMLDOM.GETPARENTNODE	检索此节点的父节点。
DBE_XMLDOM.GETTAGNAME	返回指定DOMELEMENT的标签名称。
DBE_XMLDOM.HASCHILDNODES	检查DOMNODE对象是否拥有任一子节点。
DBE_XMLDOM.IMPORTNODE	复制节点并为该节点指定所属文档。
DBE_XMLDOM.ISNULL	检测节点是否为空。
DBE_XMLDOM.ITEM	返回映射中与索引参数对应的项。
DBE_XMLDOM.MAKEELEMENT	将DOMNODE对象转换为DOMELEMENT类型。
DBE_XMLDOM.MAKENODE	将节点强制转换为DOMNODE类型。
DBE_XMLDOM.NEWDOMDOCUMENT	返回新的DOMDOCUMENT对象。
DBE_XMLDOM.SETATTRIBUTE	按名称设置DOMELEMENT属性的值。
DBE_XMLDOM.SETCHARSET	设置DOMDOCUMENT的CHATSET字符集。
DBE_XMLDOM.SETDOCTYPE	设置DOMDOCUMENT的外部DTD。
DBE_XMLDOM.SETNODEVALUE	此函数用于向DOMNODE对象中设置节点的值。
DBE_XMLDOM.WRITETOBUFFER	将 XML 节点写入指定缓冲区。

接口名称	描述
DBE_XMLDOM.WRITETOCLOB	将 XML 节点写入指定CLOB。
DBE_XMLDOM.WRITETOFILE	将 XML 节点写入指定文件。
DBE_XMLDOM.GETSESSIONTREENUM	显示当前session中所有类型的dom树的数量。
DBE_XMLDOM.GETDOCTREESINFO	显示document类型的dom树的内存占用、节点数量等统计信息。
DBE_XMLDOM.GETDETAILDOCTREEINFO	显示特定的document变量的各类型节点数量。

- DBE_XMLDOM.APPENDCHILD

  将newchild node添加到parent(n)节点最后面,并返回新添加的Node节点。
  DBE_XMLDOM.APPENDCHILD函数原型为：

```
DBE_XMLDOM.APPENDCHILD(
   n IN DOMNode,
   newchild IN DOMNode)
RETURN DOMNODE;
```

  表 10-337 DBE_XMLDOM.APPENDCHILD 接口参数说明

参数	描述
n	被添加的node。
newchild	添加的新node。

  说明

  1. DOCUMENT类型节点下APPEND ATTR类型节点会报“operation not support”错误，A数据库在此场景下不报错，但实际并没有挂载成功；
  2. ATTR类型节点下APPEND ATTR类型节点会报“operation not support”错误，A数据库在此场景下不报错，但实际并没有挂载成功；
  3. 父节点在添加多个ATTR类型子节点时，不允许KEY值相同的子节点同时存在于同一个父节点下。

  示例：

```
--为指定的DOC树添加DOMNODE节点，并通过DBE_XMLDOM.HASCHILDNODES()验证子节点是否添加成功。
DECLARE
   doc DBE_XMLDOM.DOMDocument;
   doc1 DBE_XMLDOM.DOMDocument;
   root DBE_XMLDOM.DOMElement;
   rootnode DBE_XMLDOM.DOMNode;
   child1 DBE_XMLDOM.DOMElement;
   child2 DBE_XMLDOM.DOMElement;
   attr DBE_XMLDOM.DOMAttr;
   text DBE_XMLDOM.DOMTEXT;
   node DBE_XMLDOM.DOMNode;
   child1_node DBE_XMLDOM.DOMNode;
   attr_node DBE_XMLDOM.DOMNode;
   parent DBE_XMLDOM.DOMNode;
```

```
    buf varchar2(1000);
BEGIN
    doc := DBE_XMLDOM.newDOMDocument();
    root := DBE_XMLDOM.createElement(doc, 'root');
    rootnode := DBE_xmldom.makeNode(root);
    node := DBE_XMLDOM.appendChild(DBE_xmldom.makeNode(doc), rootnode);
    child1 := DBE_XMLDOM.createElement(doc, 'child1');
    child1_node := DBE_XMLDOM.makeNode(child1);
    node := DBE_XMLDOM.appendChild(rootnode, child1_node);
    attr := DBE_XMLDOM.createAttribute(doc, 'abc');
    attr_node := DBE_XMLDOM.makeNode(attr);
    node := DBE_XMLDOM.appendChild(child1_node, attr_node);
    IF DBE_XMLDOM.HASCHILDNODES(child1_node) THEN
        DBE_OUTPUT.print_line('HAS CHILD NODES');
    ELSE
        DBE_OUTPUT.print_line('NOT HAS CHILD NODES ');
    END IF;
    parent := DBE_XMLDOM.GETPARENTNODE(attr_node);
    buf := DBE_XMLDOM.GETNODENAME(parent);
    DBE_OUTPUT.print_line(buf);
END;
/
```

- DBE_XMLDOM.CREATEELEMENT

  返回创建指定名称的DOMELEMENT对象。DBE_XMLDOM.CREATEELEMENT的函数原型为：

```
DBE_XMLDOM.CREATEELEMENT(
  doc        IN      DOMDOCUMENT,
  tagName    IN      VARCHAR2)
 RETURN DOMELEMENT;
```

  返回创建指定名称和命名空间的DOMELEMENT对象。
  DBE_XMLDOM.CREATEELEMENT的函数原型为：

```
DBE_XMLDOM.CREATEELEMENT(
  doc        IN     DOMDOCUMENT,
  tagName    IN     VARCHAR2,
  ns         IN     VARCHAR2)
 RETURN DOMELEMENT;
```

表 10-338 DBE_XMLDOM.CREATEELEMENT 接口参数说明

参数	描述
doc	指定的DOMDOCUMENT对象。
tagName	新建的DOMELEMENT名称。
ns	命名空间。

说明

1. tagName参数传入NULL和空字符串时，都会抛出异常 "NULL or invalid TagName argument specified"。
2. tagName和ns默认的最大长度为32767，超过该长度会抛出异常。

示例：

```
--1. 创建指定名称的DOMELEMENT对象。
DECLARE
  doc dbe_xmldom.domdocument;
  attr DBE_XMLDOM.DOMATTR;
  elem DBE_XMLDOM.DOMELEMENT;
  ans DBE_XMLDOM.DOMATTR;
  buf varchar2(1010);
```



```
BEGIN
    doc := dbe_xmldom.newdomdocument('<?xml version="1.0" encoding="UTF-8"?>
        <computer size="ITX"><cpu>Ryzen 9 3950X</cpu>
        <ram>32GBx2 DDR4 3200MHz</ram>
        <motherboard>ROG X570i</motherboard>
        <gpu>RTX2070 Super</gpu>
        <ssd>1TB NVMe Toshiba + 2TB NVMe WD Black</ssd>
        <hdd>12TB WD Digital</hdd>
        <psu>CORSAIR SF750</psu>
        <case>LIANLI TU150</case>
        </computer>');
    elem := dbe_xmldom.createelement(doc,'elem');
    DBE_XMLDOM.WRITETOBUFFER(dbe_xmldom.makenode(elem), buf);
    DBE_OUTPUT.print_line(buf);
END;
/

--2. 创建指定名称和命名空间的DOMELEMENT对象。
DECLARE
   doc dbe_xmldom.domdocument;
   attr DBE_XMLDOM.DOMATTR;
   elem DBE_XMLDOM.DOMELEMENT;
   ans DBE_XMLDOM.DOMNODE;
   buf varchar2(1010);
   list DBE_XMLDOM.DOMNODELIST;
   node DBE_XMLDOM.DOMNODE;
BEGIN
    doc := dbe_xmldom.newdomdocument('<h:data xmlns:h="http://www.w3.org/TR/html4/">
        <h:da1 len="10">test namespace</h:da1><h:da1>bbbbbbbbbb</h:da1></h:data>');
    elem := dbe_xmldom.createelement(doc,'elem','http://www.w3.org/TR/html5/');
    ans := DBE_XMLDOM.APPENDCHILD(dbe_xmldom.makenode(doc), dbe_xmldom.makenode(elem));
    DBE_XMLDOM.WRITETOBUFFER(doc, buf);
    DBE_OUTPUT.print_line(buf);
END;
/
```

- DBE_XMLDOM.CREATETEXTNODE

  创建并返回DOMTEXT对象。DBE_XMLDOM.CREATETEXTNOD的函数原型为：

```
DBE_XMLDOM.CREATETEXTNODE(
   doc IN DOMDocument,
   data IN VARCHAR2)
RETURN DOMTEXT;
```

表 10-339 DBE_XMLDOM.CREATETEXTNODE 接口参数说明

参数	描述
doc	指定的DOMDOCUMENT。
data	DOMText节点的内容。

说明

1.data可以输入空字符串和NULL值。

2.data默认的最大长度为32767，超过该长度会抛出异常。

示例：

```
--为DOC树添加DOMTEXT节点，并将DOC树打印输出到缓冲区。
DECLARE
   doc DBE_XMLDOM.DOMDOCUMENT;
   doctext DBE_XMLDOM.DOMTEXT;
   node DBE_XMLDOM.DOMNODE;
```



```
  buffer varchar2(1010);
BEGIN
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
     <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)>
     <!ELEMENT to (#PCDATA)>
     <!ELEMENT from (#PCDATA)>
     <!ELEMENT heading (#PCDATA)>
     <!ELEMENT body (#PCDATA)>]>
     <note>
       <to>中文</to>
       <from>Jani</from>
       <heading>Reminder</heading>
       <body>Don''t forget me this weekend!</body>
     </note>');
  doctext := DBE_XMLDOM.CREATETEXTNODE(doc, 'there is nothing');
  node := DBE_XMLDOM.MAKENODE(doctext);
  dbe_xmldom.writetobuffer(node, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
END;
/
```

- DBE_XMLDOM.FREEDOCUMENT

  释放DOMDOCUMENT节点。DBE_XMLDOM.FREEDOCUMENT的函数原型为：

```
DBE_XMLDOM.FREEDOCUMENT(
  doc     IN     DOMDOCUMENT);
```

表 10-340 DBE_XMLDOM.FREEDOCUMENT 接口参数说明

参数	描述
doc	指定的DOMDOCUMENT节点。

示例：

```
--在DOC树中添加DOMNODE节点后，将整个DOC树的资源释放。
DECLARE
  doc dbe_xmldom.domdocument;
  elem dbe_xmldom.domelement;
  doc_node dbe_xmldom.DOMNODE;
  root_elmt dbe_xmldom.DOMELEMENT;
  root_node dbe_xmldom.DOMNODE;
  value  varchar(1000);
BEGIN
  doc := dbe_xmldom.newdomdocument();
  doc_node := dbe_xmldom.MAKENODE(doc);
  root_elmt := dbe_xmldom.CREATEELEMENT(doc,'staff');
  root_node:=dbe_xmldom.APPENDCHILD(doc_node, dbe_xmldom.MAKENODE(root_elmt));
  dbe_xmldom.freedocument(doc);
END;
/
```

- DBE_XMLDOM.FREEELEMENT

  释放DOMELEMENT节点。DBE_XMLDOM.FREEELEMENT的函数原型为：

```
DBE_XMLDOM.FREEELEMENT(
  elem     IN     DOMELEMENT);
```


表 10-341 DBE_XMLDOM.FREEELEMENT 接口参数说明

参数	描述
elem	指定的DOMELEMENT节点。

示例：

```
--从DOC中获取DOMELEMENT节点后对其进行释放，对比其free前后是否为空的情况。
DECLARE
  doc dbe_xmldom.domdocument;
  elem dbe_xmldom.domelement;
  node dbe_xmldom.domnode;
  node1 dbe_xmldom.domnode;
  nodelist DBE_XMLDOM.DOMNODELIST;
  len INTEGER;
  buffer varchar2(1010);
BEGIN
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
     <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)>
     <!ELEMENT to (#PCDATA)>
     <!ELEMENT from (#PCDATA)>
     <!ELEMENT heading (#PCDATA)>
     <!ELEMENT body (#PCDATA)>]>
     <note>
       <to>中文</to>
       <from>Jani</from>
       <heading>Reminder</heading>
       <body>Don''t forget me this weekend!</body>
     </note>');
  elem := dbe_xmldom.GETDOCUMENTELEMENT(doc);
  IF DBE_XMLDOM.ISNULL(elem) THEN
      dbe_output.print_line('IS NULL');
  ELSE
      dbe_output.print_line('NOT NULL');
  END IF;
  dbe_xmldom.FREEELEMENT(elem);
  IF DBE_XMLDOM.ISNULL(elem) THEN
       dbe_output.print_line('IS NULL');
  ELSE
       dbe_output.print_line('NOT NULL');
  END IF;
END;
/
```

- DBE_XMLDOM.FREENODE

  释放DOMNODE节点。DBE_XMLDOM.FREENODE的函数原型为：

```
DBE_XMLDOM.FREENODE(
  n IN DOMNODE);
```

表 10-342 DBE_XMLDOM.FREENODE 接口参数说明

参数	描述
n	指定的DOMNODE节点。

说明

1. GaussDB进行FREENODE操作后，被释放的节点不会出现重新可用的情况；A数据库在FREENODE后存在被释放的节点重新可用并变成其他节点的情况。
2. 其他接口在调用被释放的DOMNOD节点时与A数据库存在差异。

示例：

```
--从DOC树中获取一个DOMNODE节点后对其进行释放，对比其free前后是否为空的情况。
DECLARE
  doc dbe_xmldom.domdocument;
  node dbe_xmldom.domnode;
  node1 dbe_xmldom.domnode;
  nodelist DBE_XMLDOM.DOMNODELIST;
  len INTEGER;
  buffer1 varchar2(1010);
BEGIN
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
     <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)>
     <!ELEMENT to (#PCDATA)>
     <!ELEMENT from (#PCDATA)>
     <!ELEMENT heading (#PCDATA)>
     <!ELEMENT body (#PCDATA)>]>
     <note>
       <to>中文</to>
       <from>Jani</from>
       <heading>Reminder</heading>
       <body>Don''t forget me this weekend!</body>
     </note>');
  node := dbe_xmldom.makenode(doc);
  node := dbe_xmldom.GETFIRSTCHILD(node);
  IF DBE_XMLDOM.ISNULL(node) THEN
        dbe_output.print_line('IS NULL');
  ELSE
        dbe_output.print_line('NOT NULL');
  END IF;
  DBE_XMLDOM.FREENODE(node);
  IF DBE_XMLDOM.ISNULL(node) THEN
        dbe_output.print_line('IS NULL');
  ELSE
        dbe_output.print_line('NOT NULL');
  END IF;
END;
/
```

- DBE_XMLDOM.FREENODELIST

  释放DOMNODELIST节点。DBE_XMLDOM.FREENODE的函数原型为：

```
DBE_XMLDOM.GETLENGTH(
  nl     IN    DOMNODELIST);
```

表 10-343 DBE_XMLDOM.FREENODELIST 接口参数说明

参数	描述
nl	指定的DOMNODELIST节点。

说明

1. FREENODELIST会彻底释放NODELIST。
2. 其他接口在调用被释放的DOMNODELIST节点时与A数据库存在差异。
3. freenodelist不允许空值入参。

示例：

```
--从DOC树中获取一个DOMNODELIST节点后对其进行释放，对比其free前后的长度。
DECLARE
  doc dbe_xmldom.domdocument;
  node dbe_xmldom.domnode;
  node1 dbe_xmldom.domnode;
  nodelist DBE_XMLDOM.DOMNODELIST;
  len INTEGER;
  buffer1 varchar2(1010);
BEGIN
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
    <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)>
    <!ELEMENT to (#PCDATA)>
    <!ELEMENT from (#PCDATA)>
    <!ELEMENT heading (#PCDATA)>
    <!ELEMENT body (#PCDATA)>]>
    <note>
      <to>中文</to>
      <from>Jani</from>
      <heading>Reminder</heading>
      <body>Don''t forget me this weekend!</body>
    </note>');
  node := dbe_xmldom.makenode(doc);
  node := dbe_xmldom.GETFIRSTCHILD(node);
  nodelist := DBE_XMLDOM.GETCHILDNODES(node);
  len := DBE_XMLDOM.GETLENGTH(nodelist);
  RAISE NOTICE 'len :  %', len;
  DBE_XMLDOM.FREENODELIST(nodelist);
  len := DBE_XMLDOM.GETLENGTH(nodelist);
  RAISE NOTICE 'len :  %', len;
END;
/
```

- DBE_XMLDOM.GETATTRIBUTE

  按名称返回DOMELEMENT属性的值。DBE_XMLDOM.GETATTRIBUTE的函数原型为：

  ```
  DBE_XMLDOM.GETATTRIBUTE(
    elem       IN      DOMELEMENT,
    name       IN      VARCHAR2)
  RETURN VARCHAR2;
  ```

  按名称和命名空间URI返回DOMELEMENT属性的值。
  DBE_XMLDOM.GETATTRIBUTE的函数原型为：

  ```
  DBE_XMLDOM.GETATTRIBUTE(
    elem      IN     DOMELEMENT,
    name      IN     VARCHAR2,
    ns        IN     VARCHAR2)
  RETURN VARCHAR2;
  ```

  表 10-344 DBE_XMLDOM.GETATTRIBUTE 接口参数说明

参数	描述
elem	指定的DOMELEMENT节点。
name	属性名称。
ns	命名空间。

说明

1. DBE_XMLDOM.GETATTRIBUTE接口的参数ns不支持传入参数" * "。
2. GaussDB不支持将命名空间前缀作为属性，不允许通过DBE_XMLDOM.GETATTRIBUTE接口查询该前缀的值。

示例：

```
--1. 按名称返回DOMELEMENT属性的值。
DECLARE
  doc dbe_xmldom.domdocument;
  elem dbe_xmldom.domelement;
  docnode DBE_XMLDOM.DOMNode;
  buffer varchar2(1010);
  value  varchar2(1000);
BEGIN
  doc := dbe_xmldom.newDOMDocument();
  elem := DBE_XMLDOM.CREATEELEMENT(doc, 'root');
  DBE_XMLDOM.setattribute(elem, 'len', '50cm');
  docnode := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(doc),
DBE_XMLDOM.makeNode(elem));
  value := DBE_XMLDOM.getattribute(elem, 'len');
  dbe_output.print_line('value: ');
  dbe_output.print_line(value);
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
END;
/

--2. 按名称和命名空间URI返回DOMELEMENT属性的值。
DECLARE
  doc dbe_xmldom.domdocument;
  elem dbe_xmldom.domelement;
  docnode DBE_XMLDOM.DOMNode;
  buffer varchar2(1010);
  value  varchar(1000);

BEGIN
  doc := dbe_xmldom.newDOMDocument();
  elem := DBE_XMLDOM.CREATEELEMENT(doc, 'root');
  DBE_XMLDOM.setattribute(elem, 'len', '50cm', 'www.huawei.com');
  docnode := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(doc),
DBE_XMLDOM.makeNode(elem));
  value := DBE_XMLDOM.getattribute(elem, 'len', 'www.huawei.com');
  dbe_output.print_line('value: ');
  dbe_output.print_line(value);
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
END;
/
```

- DBE_XMLDOM.GETATTRIBUTES

  将DOMNode节点属性值作为map返回。DBE_XMLDOM.GETATTRIBUTES的函数原型为：

```
DBE_XMLDOM.GETATTRIBUTES(
  n IN DOMNode)
RETURN DOMNAMEDNODEMAP;
```

表 10-345 DBE_XMLDOM.GETATTRIBUTES 接口参数说明

参数	描述
n	指定的DOMNODE节点。

示例：

```
--获取DOMNODE节点下的属性值并返回DOMNAMEDNODEMAP类型，输出DOMNAMEDNODEMAP的长度和第一个节点值。
DECLARE
  doc dbe_xmldom.domdocument;
  node dbe_xmldom.domnode;
  node1 dbe_xmldom.domnode;
  len INTEGER;
  map DBE_XMLDOM.DOMNAMEDNODEMAP;
  buffer1 varchar2(1010);
BEGIN
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
     <note a="16" b="176" c="asd">
       <to>中文</to>
       <from>Jani</from>
       <heading>Reminder</heading>
       <body>Don''t forget me this weekend!</body>
     </note>');
  node := dbe_xmldom.makenode(doc);
  node := dbe_xmldom.GETFIRSTCHILD(node);
  map := DBE_XMLDOM.GETATTRIBUTES(node);
  IF DBE_XMLDOM.ISNULL(map) THEN
        dbe_output.print_line('IS NULL');
  ELSE
        dbe_output.print_line('NOT NULL');
  END IF;
   len := DBE_XMLDOM.GETLENGTH(map);
  RAISE NOTICE 'len :  %', len;
   node1 := DBE_XMLDOM.ITEM(map, 0);
  dbe_xmldom.writetobuffer(node1, buffer1);
  dbe_output.print_line('buffer1: ');
  dbe_output.print_line(buffer1);
END;
/
```

- DBE_XMLDOM.GETCHILDNODES

  函数将节点下的若干子节点转换成节点列表。DBE_XMLDOM.GETCHILDNODES的函数原型为：

```
DBE_XMLDOM.GETCHILDNODES(
  n IN DOMNode)
RETURN DOMNodeList;
```

表 10-346 DBE_XMLDOM.GETCHILDNODES 接口参数说明

参数	描述
n	指定的DOMNODE节点。

示例：

```
--获取DOC树的第一个子节点后，将节点下的若干子节点转换成节点列表，输出其长度信息。
DECLARE
   doc dbe_xmldom.domdocument;
   doc_node dbe_xmldom.domnode;
   root_node dbe_xmldom.domnode;
   node_list dbe_xmldom.domnodelist;
   list_len integer;
   node_name varchar2(1000);
   node_type integer;
   buffer varchar2(1010);
BEGIN
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
     <note>
       <to>中文</to>
```

```
        <from>Jani</from>
        <heading>Reminder</heading>
        <body>Don''t forget me this weekend!</body>
      </note>');
  doc_node := DBE_XMLDOM.MAKENODE(doc);
  root_node := DBE_XMLDOM.GETFIRSTCHILD(doc_node);
  node_name := DBE_XMLDOM.GETNODENAME(root_node);
  node_type := DBE_XMLDOM.GETNODETYPE(root_node);
  dbe_output.print_line(node_name);
  dbe_output.print_line(node_type);
  node_list := DBE_XMLDOM.GETCHILDNODES(root_node);
  list_len := DBE_XMLDOM.GETLENGTH(node_list);
  dbe_output.print_line(list_len);
END;
/
```

- DBE_XMLDOM.GETCHILDRENBYTAGNAME

  按名称返回DOMELEMENT的子节点。
  DBE_XMLDOM.GETCHILDRENBYTAGNAME的函数原型为：

```
DBE_XMLDOM.GETCHILDRENBYTAGNAME (
  elem       IN      DOMELEMENT,
  name       IN      VARCHAR2)
RETURN DOMNODELIST;
```

  按名称和命名空间返回DOMELEMENT的子节点。
  DBE_XMLDOM.GETCHILDRENBYTAGNAME的函数原型为：

```
DBE_XMLDOM.GETCHILDRENBYTAGNAME (
  elem      IN     DOMELEMENT,
  name      IN     VARCHAR2,
  ns        IN     VARCHAR2)
 RETURN DOMNODELIST;
```

  表 10-347 DBE_XMLDOM.GETCHILDRENBYTAGNAME 接口参数说明

参数	描述
elem	指定的DOMELEMENT节点。
name	属性名称。
ns	命名空间。

  说明

  DBE_XMLDOM.GETCHILDRENBYTAGNAME接口的参数ns不支持传入参数" * "，如需获取节点下全部属性，可使用DBE_XMLDOM.GETCHILDNODES接口。

  示例：

```
--1. 按名称返回DOMELEMENT的子节点。
DECLARE
  doc dbe_xmldom.domdocument;
  elem dbe_xmldom.domelement;
  docnodelist dbe_xmldom.domnodelist;
  node_elem dbe_xmldom.domelement;
  node dbe_xmldom.domnode;
  buffer varchar2(1010);
  value  varchar2(1000);
BEGIN
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0" encoding="UTF-8"?>
    <students age="16" hight="176">
    <student>
      <name>Jerry</name><age>519</age><sex>man</sex><abc>12345</abc>
```



```
    </student>
    <student>
      <name>Bob</name><age>245</age><sex>woman</sex><abc>54321</abc>
    </student>
    </students>');
  elem := dbe_xmldom.GETDOCUMENTELEMENT(doc);
  docnodelist := dbe_xmldom.GETCHILDRENBYTAGNAME(elem, 'student');
  node := dbe_xmldom.ITEM(docnodelist, 0);
  node_elem := dbe_xmldom.makeelement(node);
  value := DBE_XMLDOM.gettagname(node_elem);
  dbe_output.print_line('value: ');
  dbe_output.print_line(value);
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
END;
/

--2. 按名称和命名空间返回DOMELEMENT的子节点。
DECLARE
  doc dbe_xmldom.domdocument;
  elem dbe_xmldom.domelement;
  node dbe_xmldom.domnode;
  node_elem dbe_xmldom.domelement;
  docnodelist dbe_xmldom.domnodelist;
  buffer varchar2(1010);
  value varchar2(1000);
BEGIN
  doc := dbe_xmldom.newdomdocument('
    <note xmlns:h="www.huawei.com">
    <h:to h:len="50cm">中文</h:to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don''t forget me this weekend!</body>
    </note>');
  elem := dbe_xmldom.GETDOCUMENTELEMENT(doc);
  docnodelist := dbe_xmldom.GETCHILDRENBYTAGNAME(elem, 'to', 'www.huawei.com');
  node := dbe_xmldom.ITEM(docnodelist, 0);
  node_elem := dbe_xmldom.makeelement(node);
  value := DBE_XMLDOM.getattribute(node_elem, 'len');
  dbe_output.print_line('value: ');
  dbe_output.print_line(value);
END;
/
```

- DBE_XMLDOM.GETDOCUMENTELEMENT

  返回指定DOCUMENT的首个子节点。DBE_XMLDOM.GETDOCUMENTELEMENT的函数原型为：

```
DBE_XMLDOM.GETDOCUMENTELEMENT(
  doc      IN      DOMDOCUMENT)
 RETURN DOMELEMENT;
```

表 10-348 DBE_XMLDOM.GETDOCUMENTELEMENT 接口参数说明

参数	描述
doc	指定的DOMDOCUMENT节点。

示例：

```
--获取DOC树中的首个子节点，并输出该节点的名称。
DECLARE
  doc dbe_xmldom.domdocument;
  elem dbe_xmldom.domelement;
```



```
  doc_node dbe_xmldom.DOMNODE;
  root_elmt dbe_xmldom.DOMELEMENT;
  root_node dbe_xmldom.DOMNODE;
  value  varchar(1000);
BEGIN
  doc := dbe_xmldom.newdomdocument();
  doc_node := dbe_xmldom.MAKENODE(doc);
  root_elmt := dbe_xmldom.CREATEELEMENT(doc,'staff');
  root_node:=dbe_xmldom.APPENDCHILD(doc_node, dbe_xmldom.MAKENODE(root_elmt));
  elem := dbe_xmldom.GETDOCUMENTELEMENT(doc);
  value := DBE_XMLDOM.gettagname(elem);
  dbe_output.print_line(value);
END;
/
```

- DBE_XMLDOM.GETFIRSTCHILD

  返回节点的第一个子节点。DBE_XMLDOM.GETFIRSTCHILD的函数原型为：

```
DBE_XMLDOM.GETFIRSTCHILD(
  n IN DOMNODE)
RETURN DOMNODE;
```

表 10-349 DBE_XMLDOM.GETFIRSTCHILD 接口参数说明

参数	描述
n	指定的DOMNODE节点。

示例：

```
--获取DOC转后成DOMNODE类型后的第一个子节点后输出其名称和类型；在获取到的第一个子节点基础上，获取该DOMNODE的第一个子节点并输出其名称。
DECLARE
    doc dbe_xmldom.domdocument;
    doc_node dbe_xmldom.domnode;
     root_node dbe_xmldom.domnode;
    inside_node dbe_xmldom.domnode;
    node_name varchar2(1000);
    node_type integer;
BEGIN
    doc := dbe_xmldom.newdomdocument('<?xml version="1.0" encoding="UTF-8"?>
       <students age="16" hight="176">
       <student1>
          <name>Jerry</name><age>519</age><sex>man</sex><abc>12345</abc>
       </student1>
       <student2>
          <name>Bob</name><age>245</age><sex>woman</sex><abc>54321</abc>
       </student2>
       </students>');
    doc_node := DBE_XMLDOM.MAKENODE(doc);
    root_node := DBE_XMLDOM.GETFIRSTCHILD(doc_node);
    node_name := DBE_XMLDOM.GETNODENAME(root_node);
    node_type := DBE_XMLDOM.GETNODETYPE(root_node);
    dbe_output.print_line(node_name);
    dbe_output.print_line(node_type);
    inside_node := DBE_XMLDOM.GETFIRSTCHILD(root_node);
    node_name := DBE_XMLDOM.GETNODENAME(inside_node);
    dbe_output.print_line(node_name);
END;
/
```

- DBE_XMLDOM.GETLASTCHILD

  返回节点的最后一个子节点。DBE_XMLDOM.GETLASTCHILD的函数原型为：


```
DBE_XMLDOM.GETLASTCHILD(
  n IN DOMNODE)
RETURN DOMNODE;
```

表 10-350 DBE_XMLDOM.GETLASTCHILD 接口参数说明

参数	描述
n	指定的DOMNODE节点。

示例：

```
--获取DOC转后成DOMNODE类型后的最后一个子节点后输出其名称和类型；在获取到的最后一个子节点基础上，获取该DOMNODE的最后一个子节点并输出其名称。
DECLARE
    doc dbe_xmldom.domdocument;
    doc_node dbe_xmldom.domnode;
    root_node dbe_xmldom.domnode;
    inside_node dbe_xmldom.domnode;
    node_name varchar2(1000);
    node_type integer;
BEGIN
    doc := dbe_xmldom.newdomdocument('<?xml version="1.0" encoding="UTF-8"?>
       <students age="16" hight="176">
       <student1>
          <name>Jerry</name><age>519</age><sex>man</sex><abc>12345</abc>
       </student1>
       <student2>
          <name>Bob</name><age>245</age><sex>woman</sex><abc>54321</abc>
       </student2>
       </students>');
    doc_node := DBE_XMLDOM.MAKENODE(doc);
    root_node := DBE_XMLDOM.GETFIRSTCHILD(doc_node);
    node_name := DBE_XMLDOM.GETNODENAME(root_node);
    node_type := DBE_XMLDOM.GETNODETYPE(root_node);
    dbe_output.print_line(node_name);
    dbe_output.print_line(node_type);
    inside_node := DBE_XMLDOM.GETLASTCHILD(root_node);
    node_name := DBE_XMLDOM.GETNODENAME(inside_node);
    dbe_output.print_line(node_name);
END;
/
```

- DBE_XMLDOM.GETLENGTH

  返回DOMNAMEDNODEMAP类型节点中的节点数。DBE_XMLDOM.GETLENGTH的函数原型为：

  ```
  DBE_XMLDOM.GETLENGTH(
    nnm      IN     DOMNAMEDNODEMAP)
   RETURN NUMBER;
  ```

  返回DOMNODELIST类型节点中的节点数。DBE_XMLDOM.GETLENGTH的函数原型为：

  ```
  DBE_XMLDOM.GETLENGTH(
    nl     IN    DOMNODELIST)
   RETURN NUMBER;
  ```


表 10-351 DBE_XMLDOM.GETLENGTH 接口参数说明

参数	描述
nnm	指定的DOMNAMEDNODEMAP类型节点。
nl	指定的DOMNODELIST类型节点。

示例：

```
--1. DOMNAMEDNODEMAP类型作为函数参数。
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  elem DBE_XMLDOM.DOMElement;
  map DBE_XMLDOM.DOMNAMEDNODEMAP;
  node DBE_XMLDOM.DOMNODE;
  buf varchar2(10000);
  len INTEGER;
BEGIN
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
     <bookstore category="web" cover="paperback">
     <book category="cooking">
       <title lang="en">Everyday Italian</title>
       <author>Giada De Laurentiis</author>
       <year>2005</year>
       <price>30.00</price>
     </book>
     </bookstore>');
  elem := DBE_XMLDOM.GETDOCUMENTELEMENT(doc);
  node := DBE_XMLDOM.MAKENODE(elem);
  map := DBE_XMLDOM.GETATTRIBUTES(node);
  len := DBE_XMLDOM.GETLENGTH(map);
  DBE_OUTPUT.print_line(len);
END;
/

--2. Nodelist类型作为函数参数。
DECLARE
  doc dbe_xmldom.domdocument;
  node dbe_xmldom.domnode;
  node1 dbe_xmldom.domnode;
  nodelist DBE_XMLDOM.DOMNODELIST;
  len INTEGER;
  buffer1 varchar2(1010);
BEGIN
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0" encoding="UTF-8"?>
     <students age="16" hight="176">
     <student>
       <name>Jerry</name><age>519</age><sex>man</sex><abc>12345</abc>
     </student>
     <student>
       <name>Jerry</name><age>519</age><sex>man</sex><abc>12345</abc>
     </student>
     </students>');
  node := dbe_xmldom.makenode(doc);
  node := dbe_xmldom.GETFIRSTCHILD(node);
  nodelist := DBE_XMLDOM.GETCHILDNODES(node);
  len := DBE_XMLDOM.GETLENGTH(nodelist);
  RAISE NOTICE 'len :  %', len;
END;
/
```

- DBE_XMLDOM.GETLOCALNAME

函数返回给定的DOMATTR类型节点的本地名称。DBE_XMLDOM.MAKENODE的函数原型为：

```
DBE_XMLDOM.GETLOCALNAME(
  a       IN     DOMATTR)
RETURN VARCHAR2;
```

函数返回给定的DOMELEMENT类型节点的本地名称。
DBE_XMLDOM.MAKENODE的函数原型为

```
DBE_XMLDOM.GETLOCALNAME(
  elem       IN     DOMELEMENT)
RETURN VARCHAR2;
```

存储过程返回给定的DOMNODE类型节点的本地名称。
DBE_XMLDOM.MAKENODE的函数原型为

```
DBE_XMLDOM.GETLOCALNAME(
  n      IN     DOMNODE,
  data   OUT    VARCHAR2);
```

表 10-352 DBE_XMLDOM.GETLOCALNAME 接口参数说明

参数	描述
a	指定的DOMATTR类型节点。
elem	指定的DOMELEMENT类型节点。
n	指定的DOMNODE类型节点。
data	返回的本地名称。

示例：

```
--1. createAttribute函数生成attr节点，获取它的本地名称。
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  root DBE_XMLDOM.DOMElement;
  attr1 DBE_XMLDOM.DOMATTR;
  value VARCHAR2(1000);
BEGIN
  doc := DBE_xmldom.newdomdocument('<?xml version="1.0"?>
    <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)>
    <!ELEMENT to (#PCDATA)>
    <!ELEMENT from (#PCDATA)>
    <!ELEMENT heading (#PCDATA)>
    <!ELEMENT body (#PCDATA)>]>
    <note><to>中文</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don''t forget me this weekend!</body>
    </note>');
  attr1 := DBE_XMLDOM.createAttribute(doc,'len');
  value := DBE_XMLDOM.getlocalname(attr1);
  DBE_output.print_line('value: ');
  DBE_output.print_line(value);
END;
/

--2. createElement函数生成elem节点，获取它的本地名称。
DECLARE
  doc DBE_xmldom.domdocument;
  elem DBE_xmldom.domelement;
  value  varchar2(10000);
BEGIN
  doc := DBE_xmldom.newdomdocument();
  elem := DBE_XMLDOM.createELEMENT(doc, 'root');
```



```
  value := DBE_XMLDOM.getlocalname(elem);
  DBE_output.print_line('value: ');
  DBE_output.print_line(value);
END;
/

--3. Element节点转换成node节点后，取其本地名称。
DECLARE
  doc DBE_xmldom.domdocument;
  elem DBE_xmldom.domelement;
  node DBE_xmldom.domnode;
  value  varchar2(100);
  buf varchar2(100);
BEGIN
  doc := DBE_xmldom.newdomdocument();
  elem := DBE_XMLDOM.createELEMENT(doc, 'root');
  node := DBE_xmldom.makenode(elem);
  DBE_XMLDOM.getlocalname(node, buf);
  DBE_output.print_line('buf: ');
  DBE_output.print_line(buf);
END;
/
```

- DBE_XMLDOM.GETNAMEDITEM

  检索由名称指定的节点。DBE_XMLDOM.GETNAMEDITEM的函数原型为：

```
DBE_XMLDOM.GETNAMEDITEM(
  nnm IN DOMNAMEDNODEMAP,
  name IN VARCHAR2)
RETURN DOMNODE;
```

  检索由名称和命名空间指定的节点。DBE_XMLDOM.GETNAMEDITEM的函数原型为：

```
DBE_XMLDOM.GETNAMEDITEM(
  nnm IN DOMNAMEDNODEMAP,
  name IN VARCHAR2,
  ns IN VARCHAR2)
RETURN DOMNODE;
```

表 10-353 DBE_XMLDOM.GETNAMEDITEM 接口参数说明

参数	描述
nnm	DOMNAMEDNODEMAP。
name	要检索的元素的名称。
ns	命名空间。

说明

1. name和nnm可以输入NULL值，但不可不入参。
2. name和ns默认的最大长度为32767，超出该长度会报错。
3. name和ns可输入int类型，长度可超出127位。

示例：

```
--1. 检索由名称指定的节点。
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  elem DBE_XMLDOM.DOMElement;
  map DBE_XMLDOM.DOMNAMEDNODEMAP;
  node DBE_XMLDOM.DOMNODE;
  node2 DBE_XMLDOM.DOMNODE;
```



```
  buf varchar2(1000);
  buf2 varchar2(1000);
BEGIN
  doc := dbe_xmldom.newdomdocument('<bookstore category="web" cover="paperback">
     <book category="cooking"><title lang="en">Everyday Italian</title>
     <author>Giada De Laurentiis</author><year>2005</year>
     <price>30.00</price></book></bookstore>');
  elem := DBE_XMLDOM.GETDOCUMENTELEMENT(doc);
  node := DBE_XMLDOM.MAKENODE(elem);
  map := DBE_XMLDOM.GETATTRIBUTES(node);
  node2:= DBE_XMLDOM.GETNAMEDITEM(map,'category');
  DBE_XMLDOM.writeToBuffer(node2, buf2);
  dbe_output.print_line(buf2);
END;
/

--2. 检索由名称和命名空间指定的节点
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  root DBE_XMLDOM.DOMElement;
  elem DBE_XMLDOM.DOMElement;
  map DBE_XMLDOM.DOMNAMEDNODEMAP;
  node DBE_XMLDOM.DOMNODE;
  buf varchar2(1000);
  buf2 varchar2(1000);
BEGIN
  doc := dbe_xmldom.newdomdocument('<h:table xmlns:h="http://www.w3.org/TR/html4/">
     <h:tr h:id="10"><h:td >Apples</h:td>
     <h:td>Bananas</h:td></h:tr></h:table>');
  root := DBE_XMLDOM.getDocumentElement(doc);
  node := DBE_XMLDOM.MAKENODE(root);
  node := dbe_xmldom.GETFIRSTCHILD(node);
  map := DBE_XMLDOM.GETATTRIBUTES(node);
  node := DBE_XMLDOM.GETNAMEDITEM(map,'id','http://www.w3.org/TR/html4/');
  DBE_XMLDOM.writeToBuffer(node, buf2);
  dbe_output.print_line(buf2);
END;
/
```

- DBE_XMLDOM.GETNEXTSIBLING

  返回下一个节点。DBE_XMLDOM.GETNEXTSIBLING的函数原型为：

```
DBE_XMLDOM.GETNEXTSIBLING(
  n  IN  DOMNODE)
RETURN DOMNODE;
```

表 10-354 DBE_XMLDOM.GETNEXTSIBLING 接口参数说明

参数	描述
n	指定的DOMNODE节点。

示例：

```
--首先获取DOC转后成DOMNODE类型后的第一个子节点；在获取到的第一个子节点基础上，获取该DOMNODE的第一个子节点；通过DBE_XMLDOM.GETNEXTSIBLING获取该节点的下一个节点，并输出下一个节点的名称。
DECLARE
   doc dbe_xmldom.domdocument;
   doc_node dbe_xmldom.domnode;
   root_node dbe_xmldom.domnode;
   inside_node dbe_xmldom.domnode;
   node_name varchar2(1000);
   node_type integer;
BEGIN
  doc := dbe_xmldom.newdomdocument('<computer size="ITX">
```



```
      <cpu>Ryzen 9 3950X</cpu>
      <ram>32GBx2 DDR4 3200MHz</ram>
      <motherboard>X570i</motherboard>
   </computer>');
   doc_node := DBE_XMLDOM.MAKENODE(doc);
   root_node := DBE_XMLDOM.GETFIRSTCHILD(doc_node);
   node_name := DBE_XMLDOM.GETNODENAME(root_node);
   node_type := DBE_XMLDOM.GETNODETYPE(root_node);
   dbe_output.print_line(node_name);
   dbe_output.print_line(node_type);
   inside_node := DBE_XMLDOM.GETFIRSTCHILD(root_node);
   node_name := DBE_XMLDOM.GETNODENAME(inside_node);
   dbe_output.print_line(node_name);
   inside_node := DBE_XMLDOM.GETNEXTSIBLING(inside_node);
   node_name := DBE_XMLDOM.GETNODENAME(inside_node);
   dbe_output.print_line(node_name);
END;
/
```

- DBE_XMLDOM.GETNODENAME

  返回NODE节点的名称。DBE_XMLDOM.GETNODENAME的函数原型为：

```
DBE_XMLDOM.GETNODENAME(
  n  IN  DOMNODE)
RETURN  VARCHAR2;
```

表 10-355 DBE_XMLDOM.GETNODENAME 接口参数说明

参数	描述
n	指定的DOMNODE节点。

示例：

```
--在DOC树中获取DOMNODE节点，输出该节点的名称。
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  root DBE_XMLDOM.DOMElement;
  root_node DBE_XMLDOM.DOMNode;
  inside_node DBE_XMLDOM.DOMNode;
  buf VARCHAR2(1000);
BEGIN
   doc := dbe_xmldom.newdomdocument('<bookstore category="web" cover="paperback">
      <book category="cooking"><title lang="en">Everyday Italian</title>
      <author>Giada De Laurentiis</author><year>2005</year>
      <price>30.00</price></book></bookstore>');
   root := DBE_XMLDOM.getDocumentElement(doc);
   root_node := DBE_XMLDOM.MAKENODE(root);
   inside_node := DBE_XMLDOM.GETFIRSTCHILD(root_node);
   buf := DBE_XMLDOM.GETNODENAME(inside_node);
   dbe_output.print_line(buf);
END;
/
```

- DBE_XMLDOM.GETNODETYPE

  返回NODE节点的类型。DBE_XMLDOM.GETNODETYPE的函数原型为：

```
DBE_XMLDOM.GETNODETYPE(
  n  IN  DOMNODE)
RETURN  NUMBER;
```


表 10-356 DBE_XMLDOM.GETNODETYPE 接口参数说明

参数	描述
n	指定的DOMNODE节点。

示例：

```
--在DOC树中获取DOMNODE节点，输出该节点的类型值。
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  doc_node DBE_XMLDOM.DOMNode;
  num number;
  buf varchar2(1000);
BEGIN
  doc := dbe_xmldom.newdomdocument('<bookstore category="web" cover="paperback">
     <book category="cooking"><title lang="en">Everyday Italian</title>
     <author>Giada De Laurentiis</author><year>2005</year>
     <price>30.00</price></book></bookstore>');
  doc_node := DBE_XMLDOM.makeNode(doc);
  num := DBE_XMLDOM.GETNODETYPE(doc_node);
  dbe_output.print_line(num);
  buf := DBE_XMLDOM.GETNODENAME(doc_node);
  dbe_output.print_line(buf);
END;
/
```

- DBE_XMLDOM.GETNODEVALUE

  返回NODE节点的值。DBE_XMLDOM.GETNODEVALUE的函数原型为：

```
DBE_XMLDOM.GETNODEVALUE(
  n  IN  DOMNODE)
RETURN  VARCHAR2;
```

表 10-357 DBE_XMLDOM.GETNODEVALUE 接口参数说明

参数	描述
n	指定的DOMNODE对象。

示例：

```
--将DOMTEXT类型节点转换为DOMNODE类型后获取该节点的值并输出。
DECLARE
  buf VARCHAR2(1000);
  doc DBE_XMLDOM.DOMDocument;
  text DBE_XMLDOM.DOMText;
  elem2 DBE_XMLDOM.DOMElement;
  node DBE_XMLDOM.DOMNode;
begin
  doc := DBE_XMLDOM.NEWDOMDOCUMENT();
  text := DBE_XMLDOM.createTextNode(doc, 'aaa');
  DBE_XMLDOM.SETNODEVALUE(DBE_XMLDOM.makeNode(text), 'ccc');
  buf := DBE_XMLDOM.GETNODEVALUE(DBE_XMLDOM.makeNode(text));
  DBE_OUTPUT.print_line(buf);
end;
/
```

- DBE_XMLDOM.GETPARENTNODE

  返回给定NODE节点的父节点。DBE_XMLDOM.GETPARENTNODE的函数原型为：


```
DBE_XMLDOM.GETPARENTNODE(
  n  IN  DOMNODE)
RETURN  DOMNODE;
```

表 10-358 DBE_XMLDOM.GETPARENTNODE 接口参数说明

参数	描述
n	指定的DOMNODE对象。

示例：

```
--向DOC树中添加子节点后，获取该子节点的父节点，输出父节点的名称。
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  doc1 DBE_XMLDOM.DOMDocument;
  root DBE_XMLDOM.DOMElement;
  child1 DBE_XMLDOM.DOMElement;
  child2 DBE_XMLDOM.DOMElement;
  attr DBE_XMLDOM.DOMAttr;
  text DBE_XMLDOM.DOMTEXT;
  node DBE_XMLDOM.DOMNode;
  parent DBE_XMLDOM.DOMNode;
  buf varchar2(1000);
BEGIN
  doc := DBE_XMLDOM.newDOMDocument();
  root := DBE_XMLDOM.createElement(doc, 'root');
  node := DBE_XMLDOM.appendChild(DBE_xmldom.makeNode(doc),DBE_xmldom.makeNode(root));
  child1 := DBE_XMLDOM.createElement(doc, 'child1');
  node := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(root),
DBE_XMLDOM.makeNode(child1));
  child2 := DBE_XMLDOM.createElement(doc, 'child2');
  node := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(child1),
DBE_XMLDOM.makeNode(child2));
  parent := DBE_XMLDOM.GETPARENTNODE(DBE_XMLDOM.makeNode(child2));
  buf := DBE_XMLDOM.GETNODENAME(parent);
  DBE_OUTPUT.print_line(buf);
END;
/
```

- DBE_XMLDOM.GETTAGNAME

  返回指定DOMELEMENT的标签名称。DBE_XMLDOM.GETTAGNAME的函数原型为：

```
DBE_XMLDOM.GETTAGNAME(
  elem  IN  DOMELEMENT)
RETURN  VARCHAR2;
```

表 10-359 DBE_XMLDOM.GETTAGNAME 接口参数说明

参数	描述
elem	指定的DOMELEMENT节点。

示例：

```
--创建DOMELEMENT节点后，输出其标签名称。
DECLARE
  doc dbe_xmldom.domdocument;
  elem dbe_xmldom.domelement;
  buffer varchar2(1010);
```



```
  value  varchar(1000);
BEGIN
  doc := dbe_xmldom.newDOMDocument();
  elem := DBE_XMLDOM.CREATEELEMENT(DBE_XMLDOM.NEWDOMDOCUMENT(), 'root');
  value := DBE_XMLDOM.gettagname(elem);
  dbe_output.print_line('value: ');
  dbe_output.print_line(value);
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
END;
/
```

- DBE_XMLDOM.HASCHILDNODES

  检查DOMNODE对象是否拥有任一子节点。DBE_XMLDOM.HASCHILDNODES的函数原型为：

```
DBE_XMLDOM.HASCHILDNODES(
  n  IN  DOMNODE)
RETURN  BOOLEAN;
```

表 10-360 DBE_XMLDOM.HASCHILDNODES 接口参数说明

参数	描述
n	指定的DOMNODE对象。

示例：

```
--创建节点child1并将其挂载到DOC树中，为child1节点添加子节点后，判断其是否拥有任一子节点。
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  doc1 DBE_XMLDOM.DOMDocument;
  root DBE_XMLDOM.DOMElement;
  child1 DBE_XMLDOM.DOMElement;
  child2 DBE_XMLDOM.DOMElement;
  attr DBE_XMLDOM.DOMAttr;
  text DBE_XMLDOM.DOMTEXT;
  node DBE_XMLDOM.DOMNode;
  buf varchar2(1000);
BEGIN
  doc := DBE_XMLDOM.newDOMDocument();
  root := DBE_XMLDOM.createElement(doc, 'root');
  node := DBE_XMLDOM.appendChild(DBE_xmldom.makeNode(doc),DBE_xmldom.makeNode(root));
  child1 := DBE_XMLDOM.createElement(doc, 'child1');
  node := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(root),
DBE_XMLDOM.makeNode(child1));
  child2 := DBE_XMLDOM.createElement(doc, 'child2');
  node := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(child1),
DBE_XMLDOM.makeNode(child2));
  IF DBE_XMLDOM.HASCHILDNODES(DBE_XMLDOM.makeNode(child1)) THEN
    DBE_OUTPUT.print_line('HAS CHILD NODES');
  ELSE
    DBE_OUTPUT.print_line('NOT HAS CHILD NODES ');
  END IF;
END;
/
```

- DBE_XMLDOM.IMPORTNODE

  该函数将节点复制到另一节点中，并将复制后的节点挂载到指定document中。若被复制节点的类型不属于xmldom的constants所规定的12种类型，则直接抛出类型不支持异常。DBE_XMLDOM.IMPORTNODE的函数原型为：


```
DBE_XMLDOM.IMPORTNODE(
  doc IN DOMDOCUMENT,
  importedNode IN DOMNODE,
  deep IN BOOLEAN)
RETURN DOMNODE;
```

表 10-361 DBE_XMLDOM.IMPORTNODE 接口参数说明

参数	描述
doc	节点挂载的文档。
importedNode	将要导入的节点。
deep	设置递归导入： ● 如果为TRUE，则导入该节点及其所有子节点。 ● 如果为FALSE，则指导入节点本身。

示例：

```
--获取将DOC2树中的节点root2_node，并将其复制并挂载到DOC树中。
DECLARE
   doc dbe_xmldom.domdocument;
   doc2 dbe_xmldom.domdocument;
   doc_node dbe_xmldom.domnode;
   doc2_node dbe_xmldom.domnode;
    root_node dbe_xmldom.domnode;
   root2_node dbe_xmldom.domnode;
   import_node dbe_xmldom.domnode;
   result_node dbe_xmldom.domnode;
   buffer varchar2(1010);
BEGIN
   doc := dbe_xmldom.newdomdocument('<bookstore category="web" cover="paperback">
      <book category="cooking"><title lang="en">Everyday Italian</title>
      <author>Giada De Laurentiis</author><year>2005</year>
      <price>30.00</price></book></bookstore>');
   doc2 := dbe_xmldom.newdomdocument('<case>LIANLI TU150</case>');
   doc_node := DBE_XMLDOM.MAKENODE(doc);
   doc2_node := DBE_XMLDOM.MAKENODE(doc2);
   root_node := DBE_XMLDOM.GETFIRSTCHILD(doc_node);
   root2_node := DBE_XMLDOM.GETFIRSTCHILD(doc2_node);
   DBE_XMLDOM.WRITETOBUFFER(doc, buffer);
   dbe_output.print_line(buffer);
   import_node := DBE_XMLDOM.IMPORTNODE(doc, root2_node, TRUE);
   result_node := DBE_XMLDOM.APPENDCHILD(root_node, import_node);
   DBE_XMLDOM.WRITETOBUFFER(doc, buffer);
   dbe_output.print_line(buffer);
END;
/
```

- DBE_XMLDOM.ISNULL

  检测给定的DOMATTR类型节点是否为NULL。如果是返回TRUE，否则返回FALSE。DBE_XMLDOM.ISNULL的函数原型为：

  ```
  DBE_XMLDOM.ISNULL(
    a      IN     DOMATTR)
  RETURN BOOLEAN;
  ```

  检测给定的DOMDOCUMENT类型节点是否为NULL。如果是返回TRUE，否则返回FALSE。DBE_XMLDOM.ISNULL的函数原型为：


```
DBE_XMLDOM.ISNULL(
  doc      IN     DOMDOCUMENT)
RETURN BOOLEAN;
```

检测给定的DOMELEMENT类型节点是否为NULL。如果是返回TRUE，否则返回FALSE。DBE_XMLDOM.ISNULL的函数原型为：

```
DBE_XMLDOM.ISNULL(
  elem     IN     DOMELEMENT)
RETURN BOOLEAN;
```

检测给定的DOMNAMEDNODEMAP类型节点是否为NULL。如果是返回TRUE，否则返回FALSE。DBE_XMLDOM.ISNULL的函数原型为：

```
DBE_XMLDOM.ISNULL(
  nnm      IN     DOMNAMEDNODEMAP)
RETURN BOOLEAN;
```

检测给定的DOMNODE类型节点是否为NULL。如果是返回TRUE，否则返回FALSE。DBE_XMLDOM.ISNULL的函数原型为：

```
DBE_XMLDOM.ISNULL(
  n      IN     DOMNODE)
RETURN BOOLEAN;
```

检测给定的DOMNODELIST类型节点是否为NULL。如果是返回TRUE，否则返回FALSE。DBE_XMLDOM.ISNULL的函数原型为：

```
DBE_XMLDOM.ISNULL(
  nl       IN     DOMNODELIST)
RETURN BOOLEAN;
```

检测给定的DOMTEXT类型节点是否为NULL。如果是返回TRUE，否则返回FALSE。DBE_XMLDOM.ISNULL的函数原型为：

```
DBE_XMLDOM.ISNULL(
  t      IN     DOMTEXT)
RETURN BOOLEAN;
```

表 10-362 DBE_XMLDOM.ISNULL 接口参数说明

参数	描述
a	指定的DOMATTR类型节点。
doc	指定的DOMDOCUMENT类型节点。
elem	指定的DOMELEMENT类型节点。
nnm	指定的DOMNAMEDNODEMAP类型节点。
n	指定的DOMNODE类型节点。
nl	指定的DOMNODELIST类型节点。
t	指定的DOMTEXT类型节点。

说明

由于DBE_XMLDOM.FREEDOCUMENT的实现差异，DBE_XMLDOM.ISNULL接口在调用释放后的DOMDOCUMENT节点时会发生报错。

示例：

```
--1. 通过createAttribute创建DOMATTR节点，并判断其是否为空。
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  attr DBE_XMLDOM.DOMATTR;
```



```
  buf VARCHAR2(1000);
BEGIN
  doc := DBE_xmldom.newdomdocument('<?xml version="1.0"?>
    <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)>
    <!ATTLIST note color CDATA #REQUIRED>
    <!ELEMENT to (#PCDATA)>
    <!ELEMENT from (#PCDATA)>
    <!ELEMENT heading (#PCDATA)>
    <!ELEMENT body (#PCDATA)>]>
    <note color="red"><to>中文</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don''t forget me this weekend!</body>
    </note>');
  attr := DBE_XMLDOM.CREATEATTRIBUTE (doc, 'length');
  if DBE_XMLDOM.ISNULL(attr) then
    DBE_OUTPUT.print_line('null');
  else
    DBE_OUTPUT.print_line('not null');
  end if;
END;
/

--2. DOMELEMENT仅声明不初始化，并判断其是否为空。
DECLARE
  docelem   DBE_XMLDOM.DOMElement;
BEGIN
  if DBE_XMLDOM.ISNULL(docelem) then
    DBE_OUTPUT.print_line('null');
  else
    DBE_OUTPUT.print_line('not null');
  end if;
END;
/
--3. 通过newDomdocument构建良构的DOMDOCUMENT节点，判断其是否为空。
Declare
 doc dbe_xmldom.domdocument;
BEGIN
 doc := DBE_xmldom.newdomdocument('<?xml version="1.0"?>
    <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)>
    <!ATTLIST note color CDATA #REQUIRED>
    <!ELEMENT to (#PCDATA)>
    <!ELEMENT from (#PCDATA)>
    <!ELEMENT heading (#PCDATA)>
    <!ELEMENT body (#PCDATA)>]>
    <note color="red"><to>中文</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don''t forget me this weekend!</body>
    </note>');
 if DBE_XMLDOM.ISNULL(doc) then
    DBE_OUTPUT.print_line('null');
 else
    DBE_OUTPUT.print_line('not null');
  end if;
END;
/
```

- DBE_XMLDOM.ITEM

  根据索引返回list中与索引对应的元素。DBE_XMLDOM.ITEM的函数原型为：

```
DBE_XMLDOM.ITEM(
  nl IN DOMNODELIST,
  index IN NUMBER)
RETURN DOMNODE;
```

  根据索引返回map中与索引对应的元素。DBE_XMLDOM.ITEM的函数原型为：

```
DBE_XMLDOM.ITEM(
  nnm IN DOMNAMEDNODEMAP,
```



```
  index IN NUMBER)
RETURN DOMNODE;
```

表 10-363 DBE_XMLDOM.ITEM 接口参数说明

参数	描述
nl	DOMNODELIST。
nnm	DOMNAMEDNODEMAP。
index	要检索的元素的索引。

说明

map类型函数item对不合理的输入参数如bool、clob，会默认指向第一个index的值。

示例：

```
--1. 根据索引返回map中与索引对应的元素。
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  elem DBE_XMLDOM.DOMElement;
  map DBE_XMLDOM.DOMNAMEDNODEMAP;
  node DBE_XMLDOM.DOMNODE;
  node2 DBE_XMLDOM.DOMNODE;
  buf varchar2(1000);
BEGIN
  doc := dbe_xmldom.newdomdocument('<bookstore category="web" cover="paperback"><book
category="cooking">
      <title lang="en">Everyday Italian</title><author>Giada De Laurentiis</author>
      <year>2005</year><price>30.00</price></book></bookstore>');
  elem := DBE_XMLDOM.GETDOCUMENTELEMENT(doc);
  node := DBE_XMLDOM.MAKENODE(elem);
  map := DBE_XMLDOM.GETATTRIBUTES(DBE_XMLDOM.getFirstChild(node));
  node2:= DBE_XMLDOM.item(map,0);
  DBE_XMLDOM.writeToBuffer(node2, buf);
  dbe_output.print_line(buf);
  dbe_xmldom.freedocument(doc);
  RAISE NOTICE '%', buf;
END;
/

--2. 根据索引返回list中与索引对应的元素。
DECLARE
  doc dbe_xmldom.domdocument;
  node dbe_xmldom.domnode;
  node1 dbe_xmldom.domnode;
  nodelist DBE_XMLDOM.DOMNODELIST;
  len INTEGER;
  buffer1 varchar2(1010);
BEGIN
  doc := dbe_xmldom.newdomdocument('<bookstore category="web" cover="paperback"><book
category="cooking">
      <title lang="en">Everyday Italian</title><author>Giada De Laurentiis</author>
      <year>2005</year><price>30.00</price></book></bookstore>');
  node := dbe_xmldom.makenode(doc);
  node := dbe_xmldom.GETFIRSTCHILD(node);
  node := dbe_xmldom.GETFIRSTCHILD(node);
  nodelist := DBE_XMLDOM.GETCHILDNODES(node);
  len := DBE_XMLDOM.GETLENGTH(nodelist);
  RAISE NOTICE 'len :  %', len;
  node1 := DBE_XMLDOM.ITEM(nodelist, 0);
  IF DBE_XMLDOM.ISNULL(node1) THEN
      dbe_output.print_line('IS NULL');
  ELSE
```



```
        dbe_output.print_line('NOT NULL');
  END IF;
  dbe_xmldom.writetobuffer(node1, buffer1);
  dbe_output.print_line('buffer1: ');
  dbe_output.print_line(buffer1);
END;
/
```

- DBE_XMLDOM.MAKEELEMENT

  返回转换后的DOMELEMENT对象。DBE_XMLDOM.MAKEELEMENT的函数原型为：

```
DBE_XMLDOM.MAKEELEMENT(
  n IN DOMNODE)
RETURN DOMELEMENT;
```

表 10-364 DBE_XMLDOM.MAKEELEMENT 接口参数说明

参数	描述
n	指定的DOMNODE对象。

示例：

```
--将DOMELEMENT类型转换后的DOMNODE类型节点node强制转换回DOMELEMENT类型。
DECLARE
  buf VARCHAR2(1000);
  doc DBE_XMLDOM.DOMDocument;
  elem DBE_XMLDOM.DOMElement;
  elem2 DBE_XMLDOM.DOMElement;
  node DBE_XMLDOM.DOMNode;
BEGIN
  doc := DBE_XMLDOM.NEWDOMDOCUMENT();
  elem := DBE_XMLDOM.createElement(doc, 'aaa');
  node := DBE_XMLDOM.makeNode(elem);
  elem2 := DBE_XMLDOM.makeElement(node);
  buf := DBE_XMLDOM.GETNODENAME(DBE_XMLDOM.makeNode(elem2));
  DBE_OUTPUT.print_line(buf);
END;
/
```

- DBE_XMLDOM.MAKENODE

  将给定的DOMATTR类型节点强制转换为DOMNODE类型，返回DOMNODE节点。DBE_XMLDOM.MAKENODE的函数原型为：

```
DBE_XMLDOM.MAKENODE(
  a       IN     DOMATTR)
 RETURN DOMNODE;
```

  将给定的DOMDOCUMENT类型节点强制转换为DOMNODE类型，返回DOMNODE节点。DBE_XMLDOM.MAKENODE的函数原型为：

```
DBE_XMLDOM.MAKENODE(
  doc      IN     DOMDOCUMENT)
 RETURN DOMNODE;
```

  将给定的DOMELEMENT类型节点强制转换为DOMNODE类型，返回DOMNODE节点。DBE_XMLDOM.MAKENODE的函数原型为：

```
DBE_XMLDOM.MAKENODE(
  elem       IN     DOMELEMENT)
 RETURN DOMNODE;
```

  将给定的DOMTEXT类型节点强制转换为DOMNODE类型，返回DOMNODE节点。DBE_XMLDOM.MAKENODE的函数原型为：


```
DBE_XMLDOM.MAKENODE(
  t       IN     DOMTEXT)
 RETURN DOMNODE;
```

表 10-365 DBE_XMLDOM.MAKENODE 接口参数说明

参数	描述
a	指定的DOMATTR类型节点。
doc	指定的DOMDOCUMENT类型节点。
elem	指定的DOMELEMENT类型节点。
t	指定的DOMTEXT类型节点。

说明

由于语法限制，DBE_XMLDOM.MAKENODE作为函数返回值时，不能直接通过如下命令实现：

```
return DBE_XMLDOM.MAKENODE(doc);
```

建议写为：

```
tmp_node := DBE_XMLDOM.MAKENODE(doc );
return tmp_node;
```

示例：

```
--1. createattr生成ATTR,将其转换为node。
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  attr DBE_XMLDOM.DOMATTR;
  dom_node DBE_XMLDOM.DOMNode;
  buf VARCHAR2(1000);
BEGIN
  doc := DBE_xmldom.newdomdocument('<?xml version="1.0"?>
      <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)>
      <!ELEMENT to (#PCDATA)>
      <!ELEMENT from (#PCDATA)>
      <!ELEMENT heading (#PCDATA)>
      <!ELEMENT body (#PCDATA)>]>
      <note><to>中文</to>
      <from>Jani</from>
      <heading>Reminder</heading>
      <body>Don''t forget me this weekend!</body>
      </note>');
   attr := DBE_XMLDOM.CREATEATTRIBUTE (doc, 'length');
   dom_node := DBE_XMLDOM.makeNode(attr);
   buf := DBE_XMLDOM.getNodeName(dom_node);
   DBE_OUTPUT.print_line(buf);
END;
/

--2. getdocumentelement函数生成elem节点后进行makenode。
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  root DBE_XMLDOM.DOMElement;
  attr DBE_XMLDOM.DOMATTR;
  node DBE_XMLDOM.DOMNODE;
  buf VARCHAR2(1000);
BEGIN
  doc := DBE_xmldom.newdomdocument('<?xml version="1.0"?>
      <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)>
      <!ATTLIST note color CDATA #REQUIRED>
      <!ELEMENT to (#PCDATA)>
      <!ELEMENT from (#PCDATA)>
```



```
    <!ELEMENT heading (#PCDATA)>
    <!ELEMENT body (#PCDATA)>]>
    <note color="red"><to>中文</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don''t forget me this weekend!</body>
    </note>');
  root := DBE_XMLDOM.getDocumentElement(doc);
  node := DBE_XMLDOM.makenode(root);
  DBE_OUTPUT.print_line(DBE_XMLDOM.GETNODENAME(node));
END;
/

--3. 通过newdomdocument创建DOMDOCUMENT类型参数，非空内容，并作为MAKENODE的输入参数。
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  buf VARCHAR2(1000);
  dom_node DBE_XMLDOM.DOMNODE;
BEGIN
  doc := DBE_xmldom.newdomdocument('<?xml version="1.0"?>
            <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)>
            <!ELEMENT to (#PCDATA)>
            <!ELEMENT from (#PCDATA)>
            <!ELEMENT heading (#PCDATA)>
            <!ELEMENT body (#PCDATA)>]>
            <note><to>中文</to>
            <from>Jani</from>
            <heading>Reminder</heading>
            <body>Don''t forget me this weekend!</body>
            </note>');
  DBE_OUTPUT.print_line('doc.id: ');
  DBE_OUTPUT.print_line(doc.id);
  dom_node := DBE_XMLDOM.makeNode(doc);
  DBE_OUTPUT.print_line('dom_node.id: ');
  DBE_OUTPUT.print_line(dom_node.id);
  buf := DBE_XMLDOM.GETNODENAME(dom_node);
  DBE_OUTPUT.print_line(buf);
END;
/

--4. DOMTEXT声明变量，不初始化, 并作为MAKENODE的输入参数。
DECLARE
  text DBE_XMLDOM.DOMTEXT;
  buf VARCHAR2(1000);
  dom_node DBE_XMLDOM.DOMNODE;
BEGIN
  dom_node := DBE_XMLDOM.makeNode(text);
  buf := DBE_XMLDOM.GETNODENAME(dom_node);
  DBE_OUTPUT.print_line(buf);
END;
/
```

- DBE_XMLDOM.NEWDOMDOCUMENT

  返回新的DOMDOCUMENT对象。DBE_XMLDOM.NEWDOMDOCUMENT的函数原型为：

  ```
  DBE_XMLDOM.NEWDOMDOCUMENT
  RETURN DOMDOCUMENT;
  ```

  返回从指定的XMLType类型创建的新DOMDOCUMENT实例对象。DBE_XMLDOM.NEWDOMDOCUMENT的函数原型为：

  ```
  DBE_XMLDOM.NEWDOMDOCUMENT(
    xmldoc    IN SYS.XMLTYPE)
  RETURN DOMDOCUMENT;
  ```

  返回从指定的CLOB类型创建的新DOMDOCUMENT实例对象。DBE_XMLDOM.NEWDOMDOCUMENT的函数原型为


```
DBE_XMLDOM.NEWDOMDOCUMENT(
  cl       IN    CLOB)
RETURN DOMDOCUMENT;
```

表 10-366 DBE_XMLDOM.NEWDOMDOCUMENT 接口参数说明

参数	描述
xmldoc	指定的XMLType类型。
cl	指定的CLOB类型。

说明

- 入参大小需限制在2GB以内。
- 目前暂不支持外部DTD解析。
- newdomdocument创建的doc，默认UTF-8字符集。
- 从同一个xmltype实例中解析出的每一个doc都是独立的，对doc的修改也不会影响到xmltype。
- 与A数据库差异参见**DBE_XMLPARSER.PARSECLOB**。

示例：

```
--1. 返回新的DOMDOCUMENT对象。
DECLARE
  doc dbe_xmldom.domdocument;
  buffer varchar2(1010);
BEGIN
  doc := dbe_xmldom.newdomdocument();
  dbe_xmldom.setdoctype(doc, 'note', 'sysid', 'pubid');
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
  dbe_xmldom.freedocument(doc);
END;
/

--2. 返回从指定的CLOB类型创建的新DOMDOCUMENT实例对象。
DECLARE
  doc dbe_xmldom.domdocument;
  buffer varchar2(1010);
BEGIN
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
      <note><to>test</to><from>Jani</from><heading>Reminder</heading>
      <body>Don''t forget me this weekend!</body></note>');
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
  dbe_xmldom.freedocument(doc);
END;
/

--3. 返回从指定的XMLType类型创建的新DOMDOCUMENT实例对象。
DECLARE
doc dbe_xmldom.domdocument;
  xt xmltype;
  buffer varchar2(1010);
BEGIN
  xt := xmltype('<h:data xmlns:h="http://www.w3.org/TR/html4/">
    <h:da1 len="10">test namespace</h:da1>
    <h:da1>bbbbbbbbbb</h:da1>
    </h:data>');
  doc := dbe_xmldom.newdomdocument(xt);
```



```
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
  dbe_xmldom.freedocument(doc);
END;
/
```

- DBE_XMLDOM.SETATTRIBUTE

  按名称设置DOMELEMENT属性的值。DBE_XMLDOM.SETATTRIBUTE的函数原型为：

```
DBE_XMLDOM.SETATTRIBUTE(
  elem    IN  DOMELEMENT,
  name    IN  VARCHAR2,
  value   IN  VARCHAR2);
```

  按名称和命名空间URI设置DOMELEMENT属性的值。
  DBE_XMLDOM.SETATTRIBUTE的函数原型为：

```
DBE_XMLDOM.SETATTRIBUTE(
  elem    IN  DOMELEMENT,
  name    IN  VARCHAR2,
  value   IN  VARCHAR2,
  ns      IN  VARCHAR2);
```

表 10-367 DBE_XMLDOM.SETATTRIBUTE 接口参数说明

参数	描述
elem	指定的DOMELEMENT节点。
name	属性名称。
value	属性值。
ns	命名空间。

说明

DBE_XMLDOM.SETATTRIBUTE接口可以添加多个属性，属性名称不可以为null，且同一个DOMELEMENT节点不能出现同名属性。如需添加同名属性，应显示地为每个同名属性设置命名空间，但是应尽量避免此类操作。如果属性存在于某命名空间下，当修改属性时，应显示指定命名空间，否则视为添加同名属性。

示例：

```
--1. 按名称设置DOMELEMENT属性的值。
DECLARE
  doc dbe_xmldom.domdocument;
  elem dbe_xmldom.domelement;
  docnode DBE_XMLDOM.DOMNode;
  buffer varchar2(1010);
  value  varchar(1000);
BEGIN
  doc := dbe_xmldom.newDOMDocument();
  elem := DBE_XMLDOM.CREATEELEMENT(doc, 'root');
  DBE_XMLDOM.setattribute(elem, 'len', '50cm');
  docnode := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(doc),
DBE_XMLDOM.makeNode(elem));
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
END;
/
```



```
--2. 按名称和命名空间URI设置DOMELEMENT属性的值。
DECLARE
  doc dbe_xmldom.domdocument;
  elem dbe_xmldom.domelement;
  docnode DBE_XMLDOM.DOMNode;
  buffer varchar2(1010);
  value  varchar(1000);
begin
  doc := dbe_xmldom.newDOMDocument();
  elem := DBE_XMLDOM.CREATEELEMENT(doc, 'root');
  DBE_XMLDOM.setattribute(elem, 'len', '50cm', 'www.huawei.com');
  docnode := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(doc),
DBE_XMLDOM.makeNode(elem));
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
END;
/

--3. 按名称修改DOMELEMENT属性的值。
DECLARE
  doc dbe_xmldom.domdocument;
  elem dbe_xmldom.domelement;
  docnode DBE_XMLDOM.DOMNode;
  buffer varchar2(1010);
  value  varchar(1000);
BEGIN
  doc := dbe_xmldom.newDOMDocument();
  elem := DBE_XMLDOM.CREATEELEMENT(doc, 'root');
  DBE_XMLDOM.setattribute(elem, 'len', '50cm');
  DBE_XMLDOM.setattribute(elem, 'len', '55cm');
  docnode := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(doc),
DBE_XMLDOM.makeNode(elem));
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
END;
/

--4. 按名称和命名空间URI修改DOMELEMENT属性的值。
DECLARE
  doc dbe_xmldom.domdocument;
  elem dbe_xmldom.domelement;
  docnode DBE_XMLDOM.DOMNode;
  buffer varchar2(1010);
  value  varchar(1000);
begin
  doc := dbe_xmldom.newDOMDocument();
  elem := DBE_XMLDOM.CREATEELEMENT(doc, 'root');
  DBE_XMLDOM.setattribute(elem, 'len', '50cm', 'www.huawei.com');
  DBE_XMLDOM.setattribute(elem, 'len', '55cm', 'www.huawei.com');
  docnode := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(doc),
DBE_XMLDOM.makeNode(elem));
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
END;
/
```

- DBE_XMLDOM.SETCHARSET

  设置DOMDOCUMENT的CHATSET字符集。DBE_XMLDOM.SETCHARSET的函数原型为：

```
DBE_XMLDOM.SETCHARSET(
  doc       IN     DOMDocument,
  charset   IN     VARCHAR2);
```


表 10-368 DBE_XMLDOM.SETCHARSET 接口参数说明

参数	描述
doc	指定的DOMDOCUMENT节点
charset	字符集

说明

- charset限制为60个字节以内。
- 目前支持的字符集有：UTF-8、UTF-16、UCS-4、UCS-2、ISO-8859-1、ISO-8859-2、ISO-8859-3、ISO-8859-4、ISO-8859-5、ISO-8859-6、ISO-8859-7、ISO-8859-8、ISO-8859-9、ISO-2022-JP、Shift_JIS、EUC-JP、ASCII。输入其他字符集会报错或者可能导致输出乱码。

示例：

```
--为DOC树设置UTF-16字符集后，将DOC树输出到缓冲区。
DECLARE
  doc dbe_xmldom.domdocument;
  buffer varchar2(1010);
BEGIN
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
     <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)><!ELEMENT to (#PCDATA)>
     <!ELEMENT from (#PCDATA)><!ELEMENT heading (#PCDATA)><!ELEMENT body (#PCDATA)>]>
     <note><to>test</to><from>Jani</from><heading>Reminder</heading>
     <body>Don''t forget me this weekend!</body></note>');
  dbe_xmldom.setcharset(doc, 'utf-16');
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
  dbe_xmldom.freedocument(doc);
END;
/
```

- DBE_XMLDOM.SETDOCTYPE

  设置DOMDOCUMENT的外部DTD。DBE_XMLDOM.SETDOCTYPE的函数原型为：

```
DBE_XMLDOM.SETDOCTYPE(
 doc     IN   DOMDocument,
 name    IN   VARCHAR2,
 sysid   IN   VARCHAR2,
 pubid   IN   VARCHAR2);
```

表 10-369 DBE_XMLDOM.SETDOCTYPE 接口参数说明

参数	描述
doc	指定的DOMDOCUMENT节点。
name	需要初始化doctype的名称。
sysid	需要初始化doctype的system ID。
pubid	需要初始化doctype的public ID。

说明

name、sysid、pubid的总长度限制在32500个字节以内。

示例：

```
--为DOMDOCUMENT的外部DTD分别设置初始化的system ID、public ID和名称后，分别将每次修改后的DOC树输出到缓冲区。
DECLARE
  doc dbe_xmldom.domdocument;
  buffer varchar2(1010);
begin
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
     <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)><!ELEMENT to (#PCDATA)>
     <!ELEMENT from (#PCDATA)><!ELEMENT heading (#PCDATA)><!ELEMENT body (#PCDATA)>]>
     <note><to>test</to><from>Jani</from><heading>Reminder</heading>
     <body>Don''t forget me this weekend!</body></note>');
  dbe_xmldom.setdoctype(doc, 'note', 'sysid', 'pubid');
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
  dbe_output.print_line('-----------------------------------------------');
  dbe_xmldom.setdoctype(doc, 'n0te', NULL, '');
  dbe_xmldom.setdoctype(doc, 'n0t1e', NULL, '');
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
  dbe_xmldom.freedocument(doc);
END;
/
```

- DBE_XMLDOM.SETNODEVALUE

  此函数用于向DOMNODE对象中设置节点的值。DBE_XMLDOM.SETNODEVALUE的函数原型为：

```
DBE_XMLDOM.SETNODEVALUE(
 n IN DOMNODE,
 nodeValue IN VARCHAR2);
```

表 10-370 DBE_XMLDOM.SETNODEVALUE 接口参数说明

参数	描述
n	指定的DOMNODE对象。
nodeValue	向DOMNODE对象中设置的字符串。

说明

1. nodeValue可以输入空字符串和NULL值，但不会对节点值进行修改。
2. nodeValue暂不支持转义字符'&'，如字符串中包含该转义字符，会清空节点值。
3. nodeValue默认的最大长度受限于VARCHAR2类型，为32767字节，超过该长度会抛出异常。

示例：

```
--对DOMTEXT转换后的DOMNODE节点设置与初始值不同的节点值后，获取并输出该节点的值。
DECLARE
  buf VARCHAR2(1000);
  doc DBE_XMLDOM.DOMDocument;
  text DBE_XMLDOM.DOMText;
  elem2 DBE_XMLDOM.DOMElement;
  node DBE_XMLDOM.DOMNode;
BEGIN
```



```
 doc := DBE_XMLDOM.NEWDOMDOCUMENT();
 text := DBE_XMLDOM.createTextNode(doc, 'aaa');
 DBE_XMLDOM.SETNODEVALUE(DBE_XMLDOM.makeNode(text), 'ccc');
 buf := DBE_XMLDOM.GETNODEVALUE(DBE_XMLDOM.makeNode(text));
 DBE_OUTPUT.print_line(buf);
END;
/
```

- DBE_XMLDOM.WRITETOBUFFER

  使用数据库字符集将 XML 节点写入指定缓冲区。
  DBE_XMLDOM.WRITETOBUFFER的函数原型为：

```
DBE_XMLDOM.WRITETOBUFFER(
  doc       IN      DOMDOCUMENT,
  buffer    INOUT   VARCHAR2);
```

  使用数据库字符集将 XML 文档写入指定缓冲区。
  DBE_XMLDOM.WRITETOBUFFER的函数原型为：

```
DBE_XMLDOM.WRITETOBUFFER(
  n         IN      DOMNODE,
  buffer    INOUT   VARCHAR2);
```

表 10-371 DBE_XMLDOM.WRITETOBUFFER 接口参数说明

参数	描述
doc	指定的DOMDOCUMENT节点。
buffer	写入操作的缓冲区。
n	指定的DOMNODE节点。

说明

- writetobuffer输出buffer限制在1GB以内。
- 该函数会添加缩进等内容，将输出格式化。输出doc将包含XML声明version和encoding。
- 默认以UTF-8字符集输出xml。

示例：

```
--1. 输入DOMNODE类型参数。
DECLARE
   doc dbe_xmldom.domdocument;
   elem DBE_XMLDOM.DOMELEMENT;
   buf varchar2(1000);
BEGIN
    doc := dbe_xmldom.newdomdocument();
    elem := dbe_xmldom.createelement(doc,'elem');
    DBE_XMLDOM.WRITETOBUFFER(dbe_xmldom.makenode(elem), buf);
    DBE_OUTPUT.print_line(buf);
END;
/

--2. 输入DOMDOCUMENT类型参数。
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  buf VARCHAR2(1000);
BEGIN
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
      <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)><!ELEMENT to (#PCDATA)>
      <!ELEMENT from (#PCDATA)><!ELEMENT heading (#PCDATA)><!ELEMENT body (#PCDATA)>]>
      <note><to>test</to><from>Jani</from><heading>Reminder</heading>
```



```
    <body>Don''t forget me this weekend!</body></note>');
  DBE_XMLDOM.WRITETOBUFFER(doc, buf);
  DBE_OUTPUT.print_line('doc: ');
  DBE_OUTPUT.print_line(buf);
  DBE_XMLDOM.FREEDOCUMENT(doc);
END;
/
```

- DBE_XMLDOM.WRITETOCLOB

  使用数据库字符集将 XML 节点写入指定的 CLOB。
  DBE_XMLDOM.WRITETOCLOB的函数原型为：

```
DBE_XMLDOM.WRITETOCLOB(
   doc     IN      DOMDOCUMENT,
   cl      INOUT  CLOB);
```

  使用数据库字符集将 XML 节点写入指定的 CLOB。
  DBE_XMLDOM.WRITETOCLOB的函数原型为：

```
DBE_XMLDOM.WRITETOCLOB(
   n       IN      DOMNODE,
   cl      INOUT  CLOB);
```

表 10-372 DBE_XMLDOM.WRITETOCLOB 接口参数说明

参数	描述
doc	指定的DOMDOCUMENT节点。
cl	要写入的CLOB。
n	指定的DOMNODE节点。

说明

- document入参，writetoclob支持2GB以内。
- 该函数会添加缩进等内容，将输出格式化。输出doc将包含XML声明version和encoding。
- 默认以UTF-8字符集输出xml。

示例：

```
--1. 输入DOMNODE类型参数。
DECLARE
  CL   CLOB;
  N    DBE_XMLDOM.DOMNODE;
BEGIN
  DBE_XMLDOM.WRITETOCLOB(N, CL);
  DBE_OUTPUT.PRINT_LINE(CL);
END;
/

--2. 输入DOMDOCUMENT类型参数。
DECLARE
    doc dbe_xmldom.domdocument;
    mclob clob;
BEGIN
    doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
       <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)><!ELEMENT to (#PCDATA)>
       <!ELEMENT from (#PCDATA)><!ELEMENT heading (#PCDATA)><!ELEMENT body (#PCDATA)>]>
       <note><to>test</to><from>Jani</from><heading>Reminder</heading>
       <body>Don''t forget me this weekend!</body></note>');
    dbe_xmldom.writetoclob(doc, mclob);
    dbe_output.print_line('mclob: ');
```



```
    dbe_output.print_line(mclob);
    dbe_xmldom.freedocument(doc);
END;
/
```

- DBE_XMLDOM.WRITETOFILE

  使用数据库字符集将 XML 节点写入指定文件。DBE_XMLDOM.WRITETOFILE的函数原型为：

  ```
  DBE_XMLDOM.WRITETOCLOB(
    doc      IN       DOMDOCUMENT,
    fileName   IN       VARCHAR2);
  ```

  使用数据库字符集将 XML 节点写入指定文件。DBE_XMLDOM.WRITETOFILE的函数原型为：

  ```
  DBE_XMLDOM.WRITETOCLOB(
    n        IN       DOMNODE,
    fileName   IN       VARCHAR2);
  ```

  使用指定字符集将 XML 文档写入指定文件。DBE_XMLDOM.WRITETOFILE的函数原型为：

  ```
  DBE_XMLDOM.WRITETOCLOB(
    doc      IN       DOMDOCUMENT,
    fileName   IN       VARCHAR2,
    charset    IN    VARCHAR2);
  ```

  使用指定字符集将 XML 文档写入指定文件。DBE_XMLDOM.WRITETOFILE的函数原型为：

  ```
  DBE_XMLDOM.WRITETOCLOB(
    n        IN       DOMNODE,
    fileName   IN       VARCHAR2,
    charset    IN    VARCHAR2);
  ```

  表 10-373 DBE_XMLDOM.WRITETOFILE 接口参数说明

参数	描述
doc	指定的DOMDOCUMENT节点
fileName	要写入的文件。
n	指定的DOMNODE节点
charset	指定字符集

说明

- document入参，filename长度限制在255个字节以内，charset限制在60个字节以内，charset支持字符集请参考**DBE_XMLDOM.SETCHARSET**接口。
- 该函数会添加缩进等内容，将输出格式化。输出doc将包含XML声明version和encoding。
- 传入newdomdocument()无参创建的doc，在不指定charset时不会报错，默认UTF-8字符集。
- filename需要在pg_directory中创建的路径下，filename中的\会被转换成/，只允许存在一个/。文件名格式应为pg_directory_name/file_name.xml，输出文件仅支持xml类型。
- 在打开guc参数**safe_data_path**时，用户只能通过高级包读写safe_data_path指定文件路径下的文件。
- 创建目录前需要保证路径为操作系统实际存在的路径，且用户需要拥有该目录的读和写权限。关于目录创建，请参考**CREATE DIRECTORY**。

示例：

```
--创建目录前需要保证路径为操作系统实际存在的路径，且用户需要拥有该目录的读和写权限。
create directory dir as '/tmp';
--1. 使用数据库字符集将 XML 节点写入指定文件。
DECLARE
  FPATH VARCHAR2(1000);
  DOC   DBE_XMLDOM.DOMDOCUMENT;
BEGIN
  DOC := DBE_XMLDOM.NEWDOMDOCUMENT('<ROOT>
    <A ATTR1="A_VALUE">
        <ACHILD>ACHILD TXT</ACHILD>
    </A>
    <B>B TXT</B>
    <C/>
  </ROOT>');
  FPATH := 'dir/simplexml.xml';
  DBE_XMLDOM.WRITETOFILE(DOC, FPATH);
END;
/

--2. 使用指定字符集将 XML 文档写入指定文件。
DECLARE
  SRC   VARCHAR(1000);
  FPATH VARCHAR2(1000);
  DOC   DBE_XMLDOM.DOMDOCUMENT;
  ELE   DBE_XMLDOM.DOMELEMENT;
BEGIN
  FPATH := 'dir/simplexml.xml';
  SRC := '<ROOT>
    <A ATTR1="A_VALUE">
        <ACHILD>ACHILD TXT</ACHILD>
    </A>
    <B>B TXT</B>
    <C/>
  </ROOT>';
  DOC := DBE_XMLDOM.NEWDOMDOCUMENT(SRC);
  ELE := DBE_XMLDOM.GETDOCUMENTELEMENT(DOC);
  DBE_XMLDOM.WRITETOFILE(DBE_XMLDOM.MAKENODE(ELE), FPATH, 'ASCII');
  DBE_XMLDOM.FREEDOCUMENT(DOC);
END;
/
drop directory dir;
```

- DBE_XMLDOM.GETSESSIONTREENUM

  查询当前session中所有类型的dom树数量。
  DBE_XMLDOM.GETSESSIONTREENUM的函数原型为：

```
DBE_XMLDOM.GETSESSIONTREENUM()
RETURN INTEGER;
```


说明

对于使用过FREEELEMENT和FREENODE的dom树，该函数依然会将其统计在内。

示例：

```
-- 创建三个document，并获取当前session中所有dom树的数量
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  doc2 DBE_XMLDOM.DOMDocument;
  doc3 DBE_XMLDOM.DOMDocument;

  buffer varchar2(1010);
BEGIN
  -- 创建三个document
  doc := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
<root>
   <elem1 attr="attrtest">
      <elem2>Im text</elem2>
      <elem3>Im text too</elem3>
   </elem1>
   <elem4>Text</elem4>
</root>
');
  doc2 := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
<computer size="ITX" price="19999">
   <cpu>Ryzen 9 3950X</cpu>
   <ram>32GBx2 DDR4 3200MHz</ram>
   <motherboard>ROG X570i</motherboard>
   <gpu>RTX2070 Super</gpu>
   <ssd>1TB NVMe Toshiba + 2TB NVMe WD Black</ssd>
   <hdd>12TB WD Digital</hdd>
   <psu>CORSAIR SF750</psu>
   <case>LIANLI TU150</case>
</computer>
');
  doc3 := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
<bookstore>
   <book genre="autobiography" publicationdate="1981" ISBN="1-861003-11-0">
      <title>The Autobiography of Benjamin Franklin</title>
      <author>
         <first-name>Benjamin</first-name>
         <last-name>Franklin</last-name>
      </author>
      <price>8.99</price>
   </book>
   <book genre="novel" publicationdate="1967" ISBN="0-201-63361-2">
      <title>The Confidence Man</title>
      <author>
         <first-name>Herman</first-name>
         <last-name>Melville</last-name>
      </author>
      <price>11.99</price>
   </book>
   <book genre="philosophy" publicationdate="1991" ISBN="1-861001-57-6">
      <title>The Gorgias</title>
      <author>
         <name>Plato</name>
      </author>
      <price>9.99</price>
   </book>
</bookstore>
');

  -- 打印id
  DBE_OUTPUT.PRINT_LINE(doc.id);
  DBE_OUTPUT.PRINT_LINE(doc2.id);
  DBE_OUTPUT.PRINT_LINE(doc3.id);
  -- 调用该函数并打印
  DBE_OUTPUT.PRINT_LINE(DBE_XMLDOM.GETSESSIONTREENUM());
```



```
 -- 释放document
 DBE_XMLDOM.FREEDOCUMENT(doc);
 DBE_XMLDOM.FREEDOCUMENT(doc2);
 DBE_XMLDOM.FREEDOCUMENT(doc3);
END;
/
```

- DBE_XMLDOM.GETDOCTREESINFO

  查询当前session中Document类型的dom树信息，如内存占用等。
  DBE_XMLDOM.GETDOCTREESINFO的函数原型为：

  ```
  DBE_XMLDOM.GETDOCTREESINFO()
  RETURN VARCHAR2;
  ```

  说明

  该函数只统计Document类型的dom树节点。

  示例：

  ```
  -- 创建三个document，并获取当前session中document类型的树的信息
  DECLARE
   doc DBE_XMLDOM.DOMDocument;
   doc2 DBE_XMLDOM.DOMDocument;
   doc3 DBE_XMLDOM.DOMDocument;

   buffer varchar2(1010);
  BEGIN
   -- 创建三个document
   doc := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
  <root>
     <elem1 attr="attrtest">
        <elem2>Im text</elem2>
        <elem3>Im text too</elem3>
     </elem1>
     <elem4>Text</elem4>
  </root>
  ');
   doc2 := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
  <computer size="ITX" price="19999">
     <cpu>Ryzen 9 3950X</cpu>
     <ram>32GBx2 DDR4 3200MHz</ram>
     <motherboard>ROG X570i</motherboard>
     <gpu>RTX2070 Super</gpu>
     <ssd>1TB NVMe Toshiba + 2TB NVMe WD Black</ssd>
     <hdd>12TB WD Digital</hdd>
     <psu>CORSAIR SF750</psu>
     <case>LIANLI TU150</case>
  </computer>
  ');
   doc3 := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
  <bookstore>
     <book genre="autobiography" publicationdate="1981" ISBN="1-861003-11-0">
        <title>The Autobiography of Benjamin Franklin</title>
        <author>
           <first-name>Benjamin</first-name>
           <last-name>Franklin</last-name>
        </author>
        <price>8.99</price>
     </book>
     <book genre="novel" publicationdate="1967" ISBN="0-201-63361-2">
        <title>The Confidence Man</title>
        <author>
           <first-name>Herman</first-name>
           <last-name>Melville</last-name>
        </author>
        <price>11.99</price>
     </book>
     <book genre="philosophy" publicationdate="1991" ISBN="1-861001-57-6">
        <title>The Gorgias</title>
        <author>
  ```



```
        <name>Plato</name>
      </author>
      <price>9.99</price>
   </book>
</bookstore>
');

 -- 打印id
 DBE_OUTPUT.PRINT_LINE(doc.id);
 DBE_OUTPUT.PRINT_LINE(doc2.id);
 DBE_OUTPUT.PRINT_LINE(doc3.id);
 -- 调用该函数并打印
 DBE_OUTPUT.PRINT_LINE(DBE_XMLDOM.GETDOCTREESINFO());
 -- 释放document
 DBE_XMLDOM.FREEDOCUMENT(doc);
 DBE_XMLDOM.FREEDOCUMENT(doc2);
 DBE_XMLDOM.FREEDOCUMENT(doc3);
END;
/
```

- DBE_XMLDOM.GETDETAILDOCTREEINFO

  查询传入的document内的各类型子节点的数量。
  DBE_XMLDOM.GETDETAILDOCTREEINFO的函数原型为：

```
DBE_XMLDOM.GETDETAILDOCTREEINFO(
   doc     IN      DOMDOCUMENT
)
RETURN VARCHAR2;
```

表 10-374 DBE_XMLDOM.GETDETAILDOCTREEINFO 接口参数说明

参数	描述
doc	指定的DOMDOCUMENT节点

说明

该函数只统计Document类型的dom树节点。

示例：

```
-- 创建三个document，并使用该函数分别获取每一个document内的各类型节点数量
DECLARE
 doc DBE_XMLDOM.DOMDocument;
 doc2 DBE_XMLDOM.DOMDocument;
 doc3 DBE_XMLDOM.DOMDocument;

 buffer varchar2(1010);
BEGIN
 -- 创建三个document
 doc := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
<root>
   <elem1 attr="attrtest">
      <elem2>Im text</elem2>
      <elem3>Im text too</elem3>
   </elem1>
   <elem4>Text</elem4>
</root>
');
 doc2 := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
<computer size="ITX" price="19999">
   <cpu>Ryzen 9 3950X</cpu>
   <ram>32GBx2 DDR4 3200MHz</ram>
   <motherboard>ROG X570i</motherboard>
   <gpu>RTX2070 Super</gpu>
   <ssd>1TB NVMe Toshiba + 2TB NVMe WD Black</ssd>
```



```
    <hdd>12TB WD Digital</hdd>
    <psu>CORSAIR SF750</psu>
    <case>LIANLI TU150</case>
</computer>
');
  doc3 := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
<bookstore>
    <book genre="autobiography" publicationdate="1981" ISBN="1-861003-11-0">
        <title>The Autobiography of Benjamin Franklin</title>
        <author>
            <first-name>Benjamin</first-name>
            <last-name>Franklin</last-name>
        </author>
        <price>8.99</price>
    </book>
    <book genre="novel" publicationdate="1967" ISBN="0-201-63361-2">
        <title>The Confidence Man</title>
        <author>
            <first-name>Herman</first-name>
            <last-name>Melville</last-name>
        </author>
        <price>11.99</price>
    </book>
    <book genre="philosophy" publicationdate="1991" ISBN="1-861001-57-6">
        <title>The Gorgias</title>
        <author>
            <name>Plato</name>
        </author>
        <price>9.99</price>
    </book>
</bookstore>
');

  -- 打印id
  DBE_OUTPUT.PRINT_LINE(doc.id);
  DBE_OUTPUT.PRINT_LINE(doc2.id);
  DBE_OUTPUT.PRINT_LINE(doc3.id);
  -- 调用该函数并打印
  buffer := DBE_XMLDOM.GETDETAILDOCTREEINFO(doc);
  DBE_OUTPUT.PRINT_LINE(buffer);
  buffer := DBE_XMLDOM.GETDETAILDOCTREEINFO(doc2);
  DBE_OUTPUT.PRINT_LINE(buffer);
  buffer := DBE_XMLDOM.GETDETAILDOCTREEINFO(doc3);
  DBE_OUTPUT.PRINT_LINE(buffer);
  -- 释放document
  DBE_XMLDOM.FREEDOCUMENT(doc);
  DBE_XMLDOM.FREEDOCUMENT(doc2);
  DBE_XMLDOM.FREEDOCUMENT(doc3);
END;
/
```

10.12.2.14 DBE_XMLPARSER

接口介绍

DBE_XMLPARSER用于将xml字符串反序列化，将存储xml文档的字符串转换为document节点。高级包DBE_XMLPARSER支持的所有接口请参见**表10-375**。

XMLPARSER数据类型可以被用来存储XMLPARSER数据，存储Xmlparser的数量上限为16777215。XMLPARSER数据类型能够根据输入的字符串解析建立domdocument节点，高级包还提供相应的set、get型接口，对解析过程的约束属性进行操作。

说明

DBE_XMLPARSER高级包在字符集设置为SQL_ASCII的数据库内使用的情况下，传入超出ASCII范围的字符，会导致报错。

DBE_XMLPARSER高级包只支持O兼容模式。

表 10-375 DBE_XMLPARSER 接口参数说明

接口名称	描述
DBE_XMLPARSER.FREEPARSER	释放PARSER。
DBE_XMLPARSER.GETDOCUMENT	获取解析的document节点。
DBE_XMLPARSER.GETVALIDATIONMODE	获取validate属性。
DBE_XMLPARSER.NEWPARSER	新建PARSER实例。
DBE_XMLPARSER.PARSEBUFFER	解析VARCHAR字符串。
DBE_XMLPARSER.PARSECLOB	解析CLOB字符串。
DBE_XMLPARSER.SETVALIDATIONMODE	设置validate属性。

- DBE_XMLPARSER.FREEPARSER

  释放给定的PARSER对象。

  DBE_XMLPARSER.FREEPARSER的存储过程原型为：

  ```
  DBE_XMLPARSER.FREEPARSER (
    p     IN     parser);
  ```

  表 10-376 DBE_XMLPARSER.FREEPARSER 接口参数说明

参数	描述
p	指定的parser类型对象。

  示例：

  ```
  -- 新建parser ，随后释放。
  DECLARE
    l_parser dbe_xmlparser.parser;
    BEGIN
    l_parser := dbe_xmlparser.newparser;
    -- 直接释放l_parser实例
    dbe_xmlparser.freeparser(l_parser);
  END;
  /
  ```

  执行结果：执行成功

- DBE_XMLPARSER.GETDOCUMENT

  GETDOCUMENT返回PARSER构建的DOM树文档的根节点。只有在解析文档后，才能调用此函数。

  DBE_XMLPARSER.GETDOCUMENT的函数原型为：


```
DBE_XMLPARSER.GETDOCUMENT (
 p     IN     parser)
 RETURN DOMDocument;
```

表 10-377 DBE_XMLPARSER.GETDOCUMENT 接口参数说明

参数	描述
p	指定的parser类型对象。

说明

- GETDOCUMENT函数无传入参数，报错。
- GETDOCUMENT函数参数parser传入为空，返回null。
- GETDOCUMENT函数传入的parser还没有解析文档，返回null。

示例：

```
-- 新建parser解析字符串，GETDOCUMENT获取文档打印出来。
DECLARE
  l_parser dbe_xmlparser.parser;
  l_doc dbe_xmldom.domdocument;
  buffer varchar2 :=
'<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Donot forget me this weekend!</body>
</note>';
  buffer2 varchar2;
  BEGIN
  l_parser := dbe_xmlparser.newparser;
  dbe_xmlparser.PARSEBUFFER(l_parser, buffer);
  l_doc := dbe_xmlparser.getdocument(l_parser);
  -- l_parser解析字符串，通过GETDOCUMENT获取domdocument节点
  dbe_xmldom.writetobuffer(l_doc, buffer2);
  RAISE NOTICE '%', buffer2;
  --将l_doc中的内容打印出来
  dbe_xmlparser.freeparser(l_parser);
  dbe_xmldom.freedocument(l_doc);
END;
/
```

执行结果：

```
NOTICE:  <?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Donot forget me this weekend!</body>
</note>
```

- DBE_XMLPARSER.GETVALIDATIONMODE

  获取给定Parser的解析验证模式。如果DTD验证开启返回TRUE，否则返回FALSE。

  DBE_XMLPARSER.GETVALIDATIONMODE的函数原型为：

```
DBE_XMLPARSER.GETVALIDATIONMODE (
 p     IN     parser)
 RETURN BOOLEAN;
```


表 10-378 DBE_XMLPARSER.GETVALIDATIONMODE 接口参数说明

参数	描述
p	指定的parser类型对象。

示例：

```
-- 新建parser,通过GETVALIDATIONMODE获取parser解析验证模式是否打开。
DECLARE
  l_parser dbe_xmlparser.parser;
BEGIN
  l_parser := dbe_xmlparser.newparser();
    if (dbe_xmlparser.GETVALIDATIONMODE(l_parser) = true) then
 RAISE NOTICE 'validation';
    else
    RAISE NOTICE 'no validation';
  end if;
  dbe_xmlparser.freeparser(l_parser);
END;
/
```

执行结果：

```
NOTICE:  validation
```

- DBE_XMLPARSER.NEWPARSER

  新建Parser对象，返回一个新的解析器实例。

  DBE_XMLPARSER.NEWPARSER的函数原型为：

  ```
  DBE_XMLPARSER.NEWPARSER
   RETURN Parser;
  ```

  示例：

  ```
  -- 新建parser 解析字符串，随后释放。
  DECLARE
    -- Create a parser.
    l_parser dbe_xmlparser.parser;
    l_doc dbe_xmldom.domdocument;
    buffer varchar2(1000) :=
      '<?xml version="1.0" encoding="UTF-8"?>
      <note>
      <to>Tove</to>
      <from>Jani</from>
      <heading>Reminder</heading>
      <body>Donot forget me this weekend!</body>
      </note>';
    buffer2 varchar2(1000);
    BEGIN
    l_parser := dbe_xmlparser.newparser;
    -- Parse the document and create a new DOM document.
    dbe_xmlparser.PARSEBUFFER(l_parser, buffer);

    dbe_xmlparser.freeparser(l_parser);
  END;
  /
  ```

  执行结果：执行成功

- DBE_XMLPARSER.PARSEBUFFER

  PARSEBFER解析存储在字符串中的XML文档。

  DBE_XMLPARSER.PARSEBUFFER的存储过程原型为：

  ```
  DBE_XMLPARSER.PARSEBUFFER (
    p     IN     parser,
    doc   IN VARCHAR2);
  ```


表 10-379 DBE_XMLPARSER.PARSEBUFFER 接口参数说明

参数	描述
p	指定的parser类型对象。
doc	存储XML文档的字符串。

说明

- PARSEBUFFER函数能够解析的字符串最大长度为32767，超过最大长度解析报错。
- 与A数据库差异：字符串encoding只支持UTF-8；version字段只支持1.0，1.0-1.9解析警告但正常执行，1.9以上报错。
- 与A数据库DTD校验差异：
  - !ATTLIST to type (CHECK|check|Check) "Ch..."将报错，因默认值"Ch..."不属于括号中枚举值，而A数据库不报错。
  - <!ENTITY baidu "www.baidu.com">..... &Baidu;&writer将报错，因区分字母大小写，Baidu无法与baidu对应，而A数据库不报错。
- 与A数据库命名空间校验差异：解析未声明的命名空间标签正常执行，而A数据库会报错。
- 与A数据库xml预定义实体解析差异：'"会被解析转译为字符’”，而A数据库中预定义实体统一都没有转译为字符。

示例：

```
-- 新建parser，PARSEBUFFER解析字符串，获取文档打印出来。
DECLARE
  l_parser dbe_xmlparser.parser;
  l_doc dbe_xmldom.domdocument;
  buffer varchar2 :=
'<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Donot forget me this weekend!</body>
</note>';
  buffer2 varchar2;
  BEGIN
  l_parser := dbe_xmlparser.newparser;
  dbe_xmlparser.PARSEBUFFER(l_parser, buffer);
  l_doc := dbe_xmlparser.getdocument(l_parser);

  dbe_xmldom.writetobuffer(l_doc, buffer2);
  RAISE NOTICE '%', buffer2;

  dbe_xmlparser.freeparser(l_parser);
  dbe_xmldom.freedocument(l_doc);
END;
/
```

执行结果：

```
NOTICE:  <?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Donot forget me this weekend!</body>
</note>
```

- DBE_XMLPARSER.PARSECLOB

  PARSECLOB解析存储在Clob中的XML文档。

DBE_XMLPARSER.PARSECLOB的存储过程原型为：

```
DBE_XMLPARSER.PARSECLOB (
  p     IN     parser,
  doc   IN CLOB);
```

表 10-380 DBE_XMLPARSER.PARSECLOB 接口参数说明

参数	描述
p	指定的parser类型对象
doc	存储XML文档的clob字符串

说明

- PARSECLOB不支持解析大于等于2GB的clob。
- 与A数据库差异：字符串encoding只支持UTF-8；version字段只支持1.0，1.0-1.9解析警告但正常执行，1.9以上报错。
- 与A数据库DTD校验差异：
  - !ATTLIST to type (CHECK|check|Check) "Ch..."将报错，因默认值"Ch..."不属于括号中枚举值，而A数据库不报错。
  - <!ENTITY baidu "www.baidu.com">..... &Baidu;&writer将报错，因区分字母大小写，Baidu无法与baidu对应，而A数据库不报错。
- 与A数据库命名空间校验差异：解析未声明的命名空间标签正常执行，而A数据库会报错。
- 与A数据库xml预定义实体解析差异：'"会被解析转译为字符’”，而A数据库中预定义实体统一都没有转译为字符。

示例：

```
-- 新建parser，parseclob解析字符串，获取文档打印出来。
DECLARE
l_clob clob :=
 '<?xml version="1.0" encoding="UTF-8"?>
  <note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>this weekend!</body>
  </note>';
  -- Create a parser.
  l_parser dbe_xmlparser.parser;
  l_doc dbe_xmldom.domdocument;
  buffer varchar2(1000);
  BEGIN
  l_parser := dbe_xmlparser.newparser;
  -- Parse the document and create a new DOM document.
  dbe_xmlparser.parseclob(l_parser, l_clob);
  l_doc := dbe_xmlparser.getdocument(l_parser);
  dbe_xmldom.writetobuffer(l_doc, buffer);
  RAISE NOTICE '%',buffer;

  dbe_xmlparser.freeparser(l_parser);
  dbe_xmldom.freedocument(l_doc);

  END;
  /
```

执行结果：

```
NOTICE:  <?xml version="1.0" encoding="UTF-8"?>
<note>
```



```
 <to>Tove</to>
 <from>Jani</from>
 <heading>Reminder</heading>
 <body>this weekend!</body>
</note>
```

- DBE_XMLPARSER.SETVALIDATIONMODE

  设置给定Parser的解析验证模式。

  DBE_XMLPARSER.SETVALIDATIONMODE的存储过程原型为：

```
DBE_XMLPARSER.SETVALIDATIONMODE(
 p     IN     parser)
 yes   IN BOOLEAN);
```

表 10-381 DBE_XMLPARSER.SETVALIDATIONMODE 接口参数说明

参数	描述
p	指定的parser类型对象。
yes	要设置的模式： ● TRUE：开启DTD验证。 ● FALSE：不开启验证

说明

- SETVALIDATIONMODE函数yes传入为空，不改变parser的解析验证模式。
- parser初始化默认为开启DTD验证模式。

示例1：

```
-- 新建parser，设置的待解析xml字符串同DTD格式不匹配。
-- setValidationMode设置为false可以正常解析,设置为true后解析报错。
DECLARE
 l_clob clob :=
'<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<table>
<name attr1="WEB" attr2="web2">African Coffee Table</name>
<width>80</width>
<length>120</length>
</table>';
 l_parser dbe_xmlparser.parser;
 l_doc dbe_xmldom.domdocument;
buffer varchar2(1000);
 BEGIN
 l_parser := dbe_xmlparser.newparser;
 -- 设为 false，去解析
   dbe_xmlparser.setValidationMode(l_parser, false);
   dbe_xmlparser.parseclob(l_parser, l_clob);
   l_doc := dbe_xmlparser.getdocument(l_parser);
   dbe_xmldom.writetobuffer(l_doc, buffer);
   RAISE NOTICE '%', buffer;
   dbe_xmlparser.freeparser(l_parser);
   dbe_xmldom.freedocument(l_doc);
 END;
 /
```


执行结果：

```
NOTICE:  <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note [
<!ELEMENT note (to , from , heading , body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<table>
 <name attr1="WEB" attr2="web2">African Coffee Table</name>
 <width>80</width>
 <length>120</length>
 </table>
```

示例2：

```
-- 新建parser，设置的待解析xml字符串同DTD格式不匹配。
-- setValidationMode设置为true后解析报错。
DECLARE
  l_clob clob :=
 '<!DOCTYPE note [
 <!ELEMENT note (to,from,heading,body)>
 <!ELEMENT to (#PCDATA)>
 <!ELEMENT from (#PCDATA)>
 <!ELEMENT heading (#PCDATA)>
 <!ELEMENT body (#PCDATA)>
 ]>
 <table>
 <name attr1="WEB" attr2="web2">African Coffee Table</name>
 <width>80</width>
 <length>120</length>
 </table>';
  l_parser dbe_xmlparser.parser;
  l_doc dbe_xmldom.domdocument;
 buffer varchar2(1000);
  BEGIN
  l_parser := dbe_xmlparser.newparser;
  -- 设为 true，去解析。
  --xml字符串不符合DTD格式，预期将报错
    dbe_xmlparser.setValidationMode(l_parser, true);
    dbe_xmlparser.parseclob(l_parser, l_clob);
    l_doc := dbe_xmlparser.getdocument(l_parser);
    dbe_xmldom.writetobuffer(l_doc, buffer);
    dbe_xmlparser.freeparser(l_parser);
    dbe_xmldom.freedocument(l_doc);
  END;
  /
```

执行结果：

```
xmlparser解析报错
ERROR:  invalid XML document
```

10.13 Retry 管理

Retry是数据库在SQL或存储过程（包含匿名块）执行失败时，在数据库内部进行重新执行的过程，以提高执行成功率和用户体验。数据库内部通过检查发生错误时的错误码及Retry相关配置，决定是否进行重试。

- 失败时回滚之前执行的语句，并重新执行存储过程进行Retry。

  示例：

```
gaussdb=# CREATE TABLE t1(a int);
gaussdb=# CREATE TABLE

gaussdb=# CREATE OR REPLACE PROCEDURE retry_basic ( IN  x INT)
AS
```



```
BEGIN
   INSERT INTO t1 (a) VALUES (x);
   INSERT INTO t1 (a) VALUES (x+1);
END;
/
CREATE PROCEDURE

gaussdb=# CALL retry_basic(1);
 retry_basic
-------------

(1 row)
gaussdb=# DROP TABLE t1;
DROP TABLE
```

10.14 调试

语法

RAISE语法

有以下五种语法格式：

图 10-36 raise_format::=

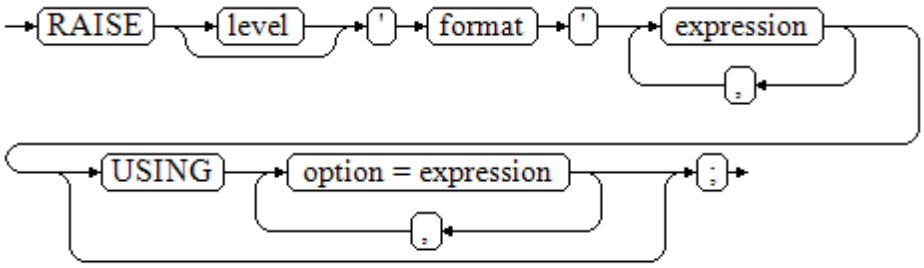


图 10-37 raise_condition::=

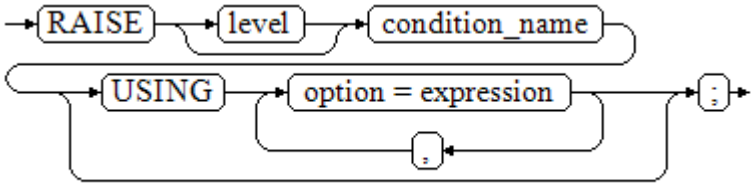


图 10-38 raise_sqlstate::=

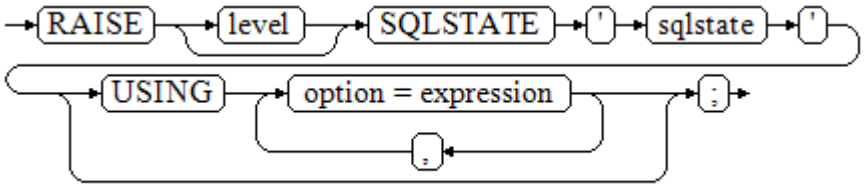


图 10-39 raise_option::=

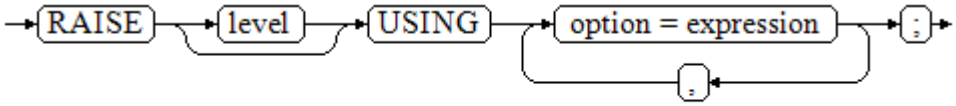
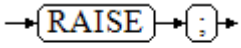


图 10-40 raise::=



参数说明：

- level选项用于指定错误级别，有DEBUG，LOG，INFO，NOTICE，WARNING以及EXCEPTION（默认值）。EXCEPTION抛出一个正常终止当前事务的异常，其他的仅产生不同异常级别的信息。特殊级别的错误信息是否报告到客户端、写到服务器日志由**log_min_messages**和**client_min_messages**这两个GUC配置参数控制。
- format：格式字符串，指定要报告的错误消息文本。格式字符串后可跟表达式，用于向消息文本中插入。在格式字符串中，%由format后面跟着的参数的值替换，%%用于打印出%。例如：

  ```
  --v_job_id 将替换字符串中的 %：
  RAISE NOTICE 'Calling cs_create_job(%)',v_job_id;
  ```

- option = expression：向错误报告中添加另外的信息。关键字option可以是MESSAGE、DETAIL、HINT以及ERRCODE，并且每一个expression可以是任意的字符串。
  - MESSAGE，指定错误消息文本，这个选项不能用于在USING前包含一个格式字符串的RAISE语句中。
  - DETAIL，说明错误的详细信息。
  - HINT，用于打印出提示信息。
  - ERRCODE，向报告中指定错误码（SQLSTATE）。可以使用条件名称或者直接用五位字符的SQLSTATE错误码。
- condition_name：错误码对应的条件名。
- sqlstate：错误码。

如果在RAISE EXCEPTION命令中既没有指定条件名也没有指定SQLSTATE，默认用RAISE EXCEPTION (P0001)。如果没有指定消息文本，默认用条件名或者SQLSTATE作为消息文本。

须知

- 当由SQLSTATE指定了错误码，则不局限于已定义的错误码，可以选择任意包含五个数字或者大写的ASCII字母的错误码，而不是00000。建议避免使用以三个0结尾的错误码，因为这种错误码是类别码，会被整个种类捕获。
- 兼容O模式下，SQLCODE等于SQLSTATE。

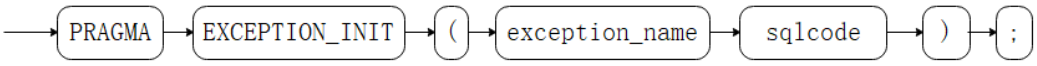
说明

图10-40所示的语法不接任何参数。这种形式仅用于一个BEGIN块中的EXCEPTION语句，它使得错误重新被处理。

EXCEPTION_INIT语法

兼容O模式下，支持使用EXCEPTION_INIT语法自定义错误码SQLCODE。语法格式如下：

图 10-41 exception_init::=



参数说明：

- exception_name为用户申明的异常名，EXCEPTION_INIT语法必须出现在与申明异常相同部分，位于申明异常之后。
- sqlcode为自定义的SQLCODE，必须为负整数，取值范围-2147483647~-1。

须知

使用EXCEPTION_INIT语法自定义错误码SQLCODE时，SQLSTATE与SQLCODE相同，SQLERRM格式为" xxx: non-GaussDB Exception"。比如自定义SQLCODE=-1，则SQLSTATE="-1"，SQLERRM=" 1: non-GaussDB Exception"。

示例

终止事务时，给出错误和提示信息：

```
CREATE OR REPLACE PROCEDURE proc_raise1(user_id in integer)
AS
BEGIN
RAISE EXCEPTION 'Noexistence ID --> %',user_id USING HINT = 'Please check your user ID';
END;
/

call proc_raise1(300011);

--执行结果
ERROR:  Noexistence ID --> 300011
HINT:  Please check your user ID
```

两种设置SQLSTATE的方式：

```
CREATE OR REPLACE PROCEDURE proc_raise2(user_id in integer)
AS
BEGIN
RAISE 'Duplicate user ID: %',user_id USING ERRCODE = 'unique_violation';
END;
/

\set VERBOSITY verbose
call proc_raise2(300011);

--执行结果
ERROR:  Duplicate user ID: 300011
```



```
SQLSTATE: 23505
LOCATION:  exec_stmt_raise, pl_exec.cpp:3482
```

如果主要的参数是条件名或者是SQLSTATE，可以使用：

RAISE division_by_zero;

RAISE SQLSTATE '22012';

例如：

```
CREATE OR REPLACE PROCEDURE division(div in integer, dividend in integer)
AS
DECLARE
res int;
    BEGIN
    IF dividend=0 THEN
        RAISE division_by_zero;
        RETURN;
    ELSE
        res := div/dividend;
        RAISE INFO 'division result: %', res;
        RETURN;
    END IF;
    END;
/
call division(3,0);

--执行结果
ERROR:  division_by_zero
```

或者另一种方式：

```
RAISE unique_violation USING MESSAGE = 'Duplicate user ID: ' || user_id;
```

兼容O模式下，支持使用语法EXCEPTION_INIT自定义错误码SQLCODE:

```
declare
    deadlock_detected exception;
    pragma exception_init(deadlock_detected, -1);
begin
    if 1 > 0 then
        raise deadlock_detected;
    end if;
exception
    when deadlock_detected then
        raise notice 'sqlcode:%,sqlstate:%,sqlerrm:%',sqlcode,sqlstate,sqlerrm;
end;
/
--执行结果
NOTICE:  sqlcode:-1,sqlstate:-1,sqlerrm: 1: non-GaussDB Exception
```

10.15 package

package是一组相关存储过程、函数、变量、常量、游标等PL/SQL程序的组合，具有面向对象的特点，可以对PL/SQL程序设计元素进行封装。package中的函数具有统一性，创建、删除、修改都统一进行。

package包含包头（Package Specification）和Package Body两个部分，其中包头所包含的声明可以被外部函数、匿名块等访问，而在包体中包含的声明不能被外部函数、匿名块等访问，只能被包体内函数和存储过程等访问。

PACKAGE的创建请参见**CREATE PACKAGE**。

须知

- 跨PACKAGE变量不支持作为FOR循环中控制变量使用。
- PACKAGE中定义类型不支持删除、修改等操作，也不支持定义表。
- 不支持以SCHEMA.PACKAGE.CURSOR的形式引用cursor变量。
- 带参数的CURSOR仅支持在当前PACKAGE内打开。
- 不支持package变量作为函数或存储过程参数的默认值。

11 自治事务

自治事务（Autonomous Transaction），在主事务执行过程中新启的独立的事务。自治事务的提交和回滚不会影响主事务已提交的数据，同时自治事务也不受主事务影响。

自治事务在存储过程、函数、匿名块和package中定义，用PRAGMA AUTONOMOUS_TRANSACTION关键字来声明。

11.1 存储过程支持自治事务

自治事务可以在存储过程中定义，标识符为PRAGMA AUTONOMOUS_TRANSACTION，其余语法与创建存储过程语法相同，示例如下。

```
--建表
gaussdb=# create table t2(a int, b int);
CREATE TABLE
gaussdb=# insert into t2 values(1,2);
INSERT 0 1
gaussdb=# select * from t2;
 a | b
---+---
 1 | 2
(1 row)

--创建包含自治事务的存储过程
gaussdb=# CREATE OR REPLACE PROCEDURE autonomous_4(a int, b int)  AS
DECLARE
    num3 int := a;
    num4 int := b;
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    insert into t2 values(num3, num4);
    dbe_output.print_line('just use call.');
END;
/
CREATE PROCEDURE
--创建调用自治事务存储过程的普通存储过程
gaussdb=# CREATE OR REPLACE PROCEDURE autonomous_5(a int, b int)  AS
DECLARE
BEGIN
    dbe_output.print_line('just no use call.');
    insert into t2 values(666, 666);
    autonomous_4(a,b);
    rollback;
END;
/
```



```
CREATE PROCEDURE
--调用普通存储过程
gaussdb=# select autonomous_5(11,22);
just no use call.
just use call.
 autonomous_5
--------------

(1 row)

--查看表结果
gaussdb=# select * from t2 order by a;
 a  | b
----+----
  1 |  2
 11 | 22
(2 rows)
```

上述例子，最后在回滚的事务块中执行包含自治事务的存储过程，直接说明了自治事务的特性，即主事务的回滚，不会影响自治事务已经提交的内容。

11.2 匿名块支持自治事务

自治事务可以在匿名块中定义，标识符为PRAGMA AUTONOMOUS_TRANSACTION，其余语法与创建匿名块语法相同，示例如下。

```
gaussdb=# create table t1(a int ,b text);
CREATE TABLE

gaussdb=# START TRANSACTION;
START TRANSACTION

gaussdb=# DECLARE
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    dbe_output.print_line('just use call.');
    insert into t1 values(1,'you are so cute,will commit!');
END;
/
just use call.
ANONYMOUS BLOCK EXECUTE

gaussdb=# insert into t1 values(1,'you will rollback!');
INSERT 0 1
gaussdb=# rollback;
ROLLBACK

gaussdb=# select * from t1;
 a |              b
---+-------------------------------
 1 | you are so cute,will commit!
(1 row)
```

上述例子，最后在回滚的事务块前执行包含自治事务的匿名块，也能直接说明了自治事务的特性，即主事务的回滚，不会影响自治事务已经提交的内容。

11.3 函数支持自治事务

自治事务可以在函数中定义，标识符为PRAGMA AUTONOMOUS_TRANSACTION，其余语法与函数语法相同，示例如下。

```
gaussdb=# create table t4(a int, b int, c text);

gaussdb=# CREATE OR REPLACE function autonomous_32(a int ,b int ,c text) RETURN int AS
```



```
DECLARE
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    insert into t4 values(a, b, c);
    return 1;
END;
/
CREATE FUNCTION
gaussdb=# CREATE OR REPLACE function autonomous_33(num1 int) RETURN int AS
DECLARE
    num3 int := 220;
    tmp int;
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    num3 := num3/num1;
    return num3;
EXCEPTION
    WHEN division_by_zero THEN
        select autonomous_32(num3, num1, sqlerrm) into tmp;
        return 0;
END;
/
CREATE FUNCTION

gaussdb=# select autonomous_33(0);
 autonomous_33
---------------
             0
(1 row)

gaussdb=# select * from t4;
  a  | b |        c
-----+---+------------------
 220 | 0 | division by zero
(1 row)
```

11.4 Package 支持自治事务

自治事务可以在package中的存储过程或者函数中定义，标识符为PRAGMA AUTONOMOUS_TRANSACTION，其余语法与创建package中存储过程或函数语法相同，示例如下。

```
--建表
gaussdb=# drop table if exists t2;
gaussdb=# create table t2(a int, b int);
CREATE TABLE
gaussdb=# insert into t2 values(1,2);
INSERT 0 1
gaussdb=# select * from t2;
 a | b
---+---
 1 | 2
(1 row)

--创建包含自治事务的package中的存储过程和函数
gaussdb=# CREATE OR REPLACE PACKAGE autonomous_pkg AS
  PROCEDURE autonomous_4(a int, b int);
  FUNCTION autonomous_32(a int ,b int) RETURN int;
END autonomous_pkg;
/
CREATE PACKAGE
gaussdb=# CREATE OR REPLACE PACKAGE body autonomous_pkg AS
PROCEDURE autonomous_4(a int, b int)  AS
DECLARE
 num3 int := a;
 num4 int := b;
 PRAGMA AUTONOMOUS_TRANSACTION;
```



```
BEGIN
    insert into t2 values(num3, num4);
END;
FUNCTION autonomous_32(a int ,b int) RETURN int AS
DECLARE
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    insert into t2 values(a, b);
    return 1;
END;
END autonomous_pkg;
/
CREATE PACKAGE BODY
--创建调用package自治事务存储过程和函数的普通存储过程
gaussdb=# CREATE OR REPLACE PROCEDURE autonomous_5(a int, b int)  AS
DECLARE
va int;
BEGIN
    insert into t2 values(666, 666);
    autonomous_pkg.autonomous_4(a,b);
        va := autonomous_pkg.autonomous_32(a + 1, b + 1);
    rollback;
END;
/
CREATE PROCEDURE
--调用普通存储过程
gaussdb=# select autonomous_5(11,22);
 autonomous_5
--------------

(1 row)

--查看表结果
gaussdb=# select * from t2 order by a;
 a  | b
----+----
  1 |  2
 11 | 22
 12 | 23
(3 rows)
```

上述例子，最后在回滚的事务块中执行包含package自治事务的存储过程和函数，直接说明了自治事务的特性，即主事务的回滚，不会影响自治事务已经提交的内容。

11.5 规格约束

⚠ 注意

- 自治事务执行时，将会在后台启动自治事务session，可以通过max_concurrent_autonomous_transactions设置自治事务执行的最大并行量，该参数取值范围为0~10000，默认值为10。
- 当max_concurrent_autonomous_transactions参数设置为0时，自治事务将无法执行。
- 自治事务新启session后，将使用默认session参数，不共享主session下对象（包括session级别变量，本地临时变量，全局临时表的数据等）。
- 自治事务理论上限为10000，实际上限为动态值，参考GUC参数**max_concurrent_autonomous_transactions**描述。
- 自治事务受通信缓冲区影响，返回给客户端的信息大小受限于通信缓冲区长度，超过通信缓冲区长度时报错。
- 自治事务的锁不受lock timeout影响，锁超时时间为2147483s，自治事务执行超过此时间会报错锁超时。
- 自治事务设置建立连接超时时间5s，建立连接尝试5次。建立连接期间不立即响应信号，每次建立连接前检查信号。高并发、高CPU、高内存，以及线程池扩容场景下可能存在超时报错现象。

- 触发器函数不支持自治事务。

```
gaussdb=# CREATE TABLE test_trigger_des_tbl(id1 INT, id2 INT, id3 INT);
CREATE TABLE
gaussdb=# CREATE OR REPLACE FUNCTION tri_insert_func() RETURNS TRIGGER AS
$$
DECLARE
 PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
INSERT INTO test_trigger_des_tbl VALUES(NEW.id1, NEW.id2, NEW.id3);
RETURN NEW;
END
$$ LANGUAGE plpgsql;
ERROR:  Triggers do not support autonomous transactions
DETAIL:  N/A
CONTEXT:  compilation of PL/pgSQL function "tri_insert_func" near line 4
```

- 自治事务不支持非顶层匿名块调用（仅支持顶层自治事务,包括存储过程、函数、匿名块）。

```
gaussdb=# drop table if exists t1;
gaussdb=# create table t1(a int ,b text);
CREATE TABLE
gaussdb=# DECLARE
 --PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
 DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
 BEGIN
  dbe_output.print_line('just use call.');
  insert into t1 values(1,'can you rollback!');
 END;
 insert into t1 values(2,'I will rollback!');
 rollback;
END;
/
```

```
just use call.
ANONYMOUS BLOCK EXECUTE

gaussdb=# select * from t1;
 a | b
---+---
(0 rows)
```

- 自治事务仅支持PROCEDURE OUT参数传递ref cursor参数，不支持IN、INOUT以及FUNCTION传递ref cursor参数。

```
gaussdb=# create table sections(section_ID int);
CREATE TABLE
gaussdb=# insert into sections values(1);
INSERT 0 1
gaussdb=# insert into sections values(1);
INSERT 0 1
gaussdb=# insert into sections values(1);
INSERT 0 1
gaussdb=# insert into sections values(1);
INSERT 0 1

1. PROCEDURE OUT出参传递ref cursor(支持）
gaussdb=# CREATE OR REPLACE PROCEDURE proc_sys_ref(OUT c1 refcursor)
IS
declare
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
 OPEN c1 FOR SELECT section_ID FROM sections ORDER BY section_ID;

END;
/
CREATE PROCEDURE

gaussdb=# CREATE OR REPLACE PROCEDURE proc_sys_call() AS
DECLARE
  c1 SYS_REFCURSOR;
  TEMP NUMBER(4);
BEGIN
  proc_sys_ref(c1);
  if c1%isopen then
    raise notice '%','ok';
 end if;

  LOOP
    FETCH C1 INTO TEMP;
    raise notice '%',C1%ROWCOUNT;
    EXIT WHEN C1%NOTFOUND;
  END LOOP;
END;
/
CREATE PROCEDURE

2. PROCEDURE IN或INOUT出参传递ref cursor(不支持）
gaussdb=# CREATE OR REPLACE PROCEDURE proc_sys_ref(IN c1 refcursor)
IS
declare
  PRAGMA AUTONOMOUS_TRANSACTION;
  TEMP NUMBER(4);
BEGIN
  if c1%isopen then
    raise notice '%','ok';
  end if;

  LOOP
    FETCH C1 INTO TEMP;
    raise notice '%',C1%ROWCOUNT;
    EXIT WHEN C1%NOTFOUND;
  END LOOP;
END;
/
```



```
CREATE PROCEDURE

gaussdb=# CREATE OR REPLACE PROCEDURE proc_sys_call() AS
DECLARE
  c1 SYS_REFCURSOR;
  TEMP NUMBER(4);
BEGIN
  OPEN c1 FOR SELECT section_ID FROM sections ORDER BY section_ID;
  proc_sys_ref(c1);
END;
/
CREATE PROCEDURE
gaussdb=# CALL proc_sys_call();
ERROR:  Unsupported: ref_cursor parameter is not supported for autonomous transactions.
CONTEXT:  SQL statement "CALL proc_sys_ref(c1)"
PL/pgSQL function proc_sys_call() line 7 at PERFORM

3. FUNCTION RETURN传递ref cursor（不支持）
gaussdb=# CREATE OR REPLACE function proc_sys_ref()
return SYS_REFCURSOR
IS
declare
  PRAGMA AUTONOMOUS_TRANSACTION;
        C1 SYS_REFCURSOR;
BEGIN
 OPEN C1 FOR SELECT section_ID FROM sections ORDER BY section_ID;
 return C1;
END;
/
gaussdb=# DROP PROCEDURE proc_sys_ref;
DROP PROCEDURE

4. FUNCTION OUT出参传递ref cursor（不支持）
gaussdb=# CREATE OR REPLACE function proc_sys_ref(C1 out SYS_REFCURSOR)
return SYS_REFCURSOR
IS
declare
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
 OPEN C1 FOR SELECT section_ID FROM sections ORDER BY section_ID;
 return 1;
END;
/
ERROR:  Autonomous function do not support ref cursor as return types or out, inout arguments.
DETAIL:  N/A
CONTEXT:  compilation of PL/pgSQL function "proc_sys_ref" near line 4
```

- 自治事务函数不支持直接返回record类型和out出参&record类型同时返回。

```
gaussdb=# create table test_in (id int,a date);
CREATE TABLE
gaussdb=# CREATE OR REPLACE FUNCTION autonomous_out()
RETURNS record
LANGUAGE plpgsql AS $$
DECLARE PRAGMA AUTONOMOUS_TRANSACTION;
         r1 record;
BEGIN
    DBE_OUTPUT.PRINT_LINE('this is in autonomous_f_139_7');
    truncate test_in;
    insert into test_in values (1,'1909-01-01');
    select * into r1 from test_in;
RETURN r1;
END;
$$;
CREATE FUNCTION
gaussdb=# select ok.id,ok.a from autonomous_out() as ok(id int,a date);
ERROR:  unrecognized return type for PLSQL function.
gaussdb=# create type rec is (e1 integer, e2 varchar2);
CREATE TYPE
gaussdb=# create or replace function func(ele3 inout varchar2) return rec as
i integer;
```



```
ele1 rec;
PRAGMA AUTONOMOUS_TRANSACTION;
begin
NULL;
return ele1;
end;
/
CREATE FUNCTION
gaussdb=# call func(1);
 e1 | e2
----+----
    |
(1 row)
```

- 不支持修改自治事务的隔离级别。
- 不支持自治事务返回集合类型（setof）。

```
gaussdb=# drop table if exists test_in;
gaussdb=# create table test_in (id int,a date);
CREATE TABLE
gaussdb=# create table test_main (id int,a date);
CREATE TABLE
gaussdb=# insert into test_main values (1111,'2021-01-01'),(2222,'2021-02-02');
INSERT 0 2
gaussdb=# truncate test_in,test_main;
TRUNCATE TABLE
gaussdb=# CREATE OR REPLACE FUNCTION autonomous_f_022(num1  int) RETURNS SETOF test_in
LANGUAGE plpgsql AS $$
DECLARE
count int :=3;
test_row test_in%ROWTYPE;
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    while true
    loop
    if count=3 then
    null;
    else
    if count=2 then
        insert into test_main values (count,'2021-03-03');
        goto pos1;
    end if;
    end if;
    count=count-1;
    end loop;
    insert into test_main values (1000,'2021-04-04');
     <<pos1>>
     for test_row in select * from test_main
     loop
        return next test_row;
       end loop;
      return;
END;
$$
;
ERROR:  Autonomous transactions do not support RETURN SETOF.
DETAIL:  N/A
CONTEXT:  compilation of PL/pgSQL function "autonomous_f_022" near line 6
```


12 系统表和系统视图

12.1 系统表和系统视图概述

系统表是GaussDB存放结构元数据的地方，它是GaussDB数据库系统运行控制信息的来源，是数据库系统的核心组成部分。

系统视图提供了查询系统表和访问数据库内部状态的方法。

系统表和系统视图要么只对管理员可见，要么对所有用户可见。下面的系统表和视图有些标识了需要管理员权限，这些系统表和视图只有管理员可以查询。

用户可以删除后重新创建这些表、增加列、插入和更新数值，但是用户修改系统表会导致系统信息的不一致，从而导致系统控制紊乱。正常情况下不应该由用户手工修改系统表或系统视图，或者手工重命名系统表或系统视图所在的模式，而是由SQL语句关联的系统表操作自动维护系统表信息。

须知

- 不建议用户修改系统表和系统视图的权限。
- 用户应该禁止对系统表进行增删改等操作，人为对系统表的修改或破坏可能会导致系统各种异常情况甚至数据库不可用。
- 系统表和系统视图中的字段类型详见**数据类型**章节介绍。

12.2 系统表

12.2.1 GS_ASP

GS_ASP显示被持久化的ACTIVE SESSION PROFILE样本，该表只能在系统库下查询，在用户库下查询无数据。

表 12-1 GS_ASP 字段

名称	类型	描述
sampleid	bigint	采样ID。
sample_time	timestamp with time zone	采样的时间。
need_flush_sample	boolean	该样本是否需要刷新到磁盘。 ● t（true）：表示需要。 ● f（false）：表示不需要。
databaseid	oid	数据库ID。
thread_id	bigint	线程的ID。
sessionid	bigint	会话的ID。
start_time	timestamp with time zone	会话的启动时间。
event	text	具体的事件名称。
lwtid	integer	当前线程的轻量级线程号。
psessionid	bigint	streaming线程的父线程。
tlevel	integer	streaming线程的层级。与执行计划的层级（id）相对应。
smpid	integer	smp执行模式下并行线程的并行编号。
userid	oid	session用户的id。
application_name	text	应用的名字。
client_addr	inet	client端的地址。
client_hostname	text	client端的名字。
client_port	integer	客户端用于与后端通讯的TCP端口号。
query_id	bigint	debug query id。
unique_query_id	bigint	unique query id。
user_id	oid	unique query的key中的user_id。
cn_id	integer	表示下发该unique sql的节点id。unique query的key中的cn_id。
unique_query	text	规范化后的Unique SQL文本串。
locktag	text	会话等待锁信息，可通过locktag_decode解析。

名称	类型	描述
lockmode	text	会话等待锁模式： ● LW_EXCLUSIVE：排他锁 ● LW_SHARED：共享锁 ● LW_WAIT_UNTIL_FREE：等待LW_EXCLUSIVE可用
block_sessionid	bigint	如果会话正在等待锁，阻塞该会话获取锁的会话标识。
wait_status	text	描述event列的更多详细信息。
global_sessionid	text	全局会话ID。
xact_start_time	timestamp with time zone	事务开始时间。
query_start_time	timestamp with time zone	语句开始执行时间。
state	text	当前事务状态。 可能取值为： ● active: 后台正在执行一个查询。 ● idle in transaction: 后台在事务中。但事务中没有语句在执行。 ● idle in transaction (aborted): 后台在事务中，但事务中有语句执行失败。 ● fastpath function call: 后台正在执行一个fast-path函数。 ● disabled: 如果后台禁用track_activities，则报告这个状态。

12.2.2 GS_AUDITING_POLICY

GS_AUDITING_POLICY系统表记录统一审计的主体信息，每条记录对应一个设计策略。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

表 12-2 GS_AUDITING_POLICY 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
polname	name	策略名称，需要唯一，不可重复。

名称	类型	描述
polcomments	name	策略描述字段，记录策略相关的描述信息，通过COMMENTS关键字体现。
modifydate	timestamp without time zone	策略创建或修改的最新时间戳。
polenabled	boolean	用来表示策略启动开关。 • t（true）：表示策略启动。 • f（false）：表示策略没有启动。

12.2.3 GS_AUDITING_POLICY_ACCESS

GS_AUDITING_POLICY_ACCESS系统表记录与DML数据库相关操作的统一审计信息。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

表 12-3 GS_AUDITING_POLICY_ACCESS 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
accesstype	name	DML数据库操作相关类型。例如SELECT、INSERT、DELETE等。
labelname	name	资源标签名称。对应系统表13.2.2 GS_AUDITING_POLICY中的polname字段。
policyoid	oid	所属审计策略的Oid，对应系统表13.2.2 GS_AUDITING_POLICY中的oid。
modifydate	timestamp without time zone	创建或修改的最新时间戳。

12.2.4 GS_AUDITING_POLICY_FILTERS

GS_AUDITING_POLICY_FILTERS系统表记录统一审计相关的过滤策略相关信息，每条记录对应一个设计策略。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

表 12-4 GS_AUDITING_POLICY_FILTERS 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
filtertype	name	过滤类型。目前值仅为logical_expr。
labelname	name	名称。目前值仅为logical_expr。
policyoid	oid	所属审计策略的Oid，对应系统表13.2.2 GS_AUDITING_POLICY中的oid。
modifydate	timestamp without time zone	创建或修改的最新时间戳。
logicaloperator	text	过滤条件的逻辑字符串。

12.2.5 GS_AUDITING_POLICY_PRIVILEGES

GS_AUDITING_POLICY_PRIVILEGES系统表记录统一审计DDL数据库相关操作信息，每条记录对应一个设计策略。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

表 12-5 GS_AUDITING_POLICY_PRIVI 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
privilegetype	name	DDL数据库操作相关类型。例如CREATE、ALTER、DROP等。
labelname	name	资源标签名称。对应系统表13.2.2 GS_AUDITING_POLICY中的polname字段。
policyoid	oid	对应审计策略系统表13.2.2 GS_AUDITING_POLICY中的oid。
modifydate	timestamp without time zone	创建或修改的最新时间戳。

12.2.6 GS_CLIENT_GLOBAL_KEYS

GS_CLIENT_GLOBAL_KEYS系统表记录密态等值特性中客户端加密主密钥相关信息，每条记录对应一个客户端加密主密钥。

表 12-6 GS_CLIENT_GLOBAL_KEYS 字段

名称	类型	描述
oid	oid	行标识符（隐含字段）。
global_key_name	name	客户端加密主密钥（cmk）名称。
key_namespace	oid	包含此客户端加密主密钥（cmk）的命名空间OID。
key_owner	oid	客户端加密主密钥（cmk）的所有者。
key_acl	aclitem[]	创建该密钥时所拥有的访问权限。
create_date	timestamp without time zone	创建密钥的时间。

12.2.7 GS_CLIENT_GLOBAL_KEYS_ARGS

GS_CLIENT_GLOBAL_KEYS_ARGS系统表记录密态等值特性中客户端加密主密钥相关元数据信息，每条记录对应客户端加密主密钥的一个键值对信息。

表 12-7 GS_CLIENT_GLOBAL_KEYS_ARGS 字段

名称	类型	描述
oid	oid	行标识符（隐含字段）。
global_key_id	oid	客户端加密主密钥（cmk）oid。
function_name	name	值为encryption。
key	name	客户端加密主密钥（cmk）的元数据信息对应的名称。
value	bytea	客户端加密主密钥（cmk）的元数据信息名称的值。

12.2.8 GS_COLUMN_KEYS

GS_COLUMN_KEYS系统表记录密态等值特性中列加密密钥相关信息，每条记录对应一个列加密密钥。

表 12-8 GS_COLUMN_KEYS 字段

名称	类型	描述
oid	oid	行标识符（隐含字段）。
column_key_name	name	列加密密钥（cek）名称。

名称	类型	描述
column_key_distributed_id	oid	根据加密密钥（cek）全称域名hash值得到的id。
global_key_id	oid	外键。客户端加密主密钥（cmk）的OID。
key_namespace	oid	包含此列加密密钥（cek）的命名空间OID。
key_owner	oid	列加密密钥（cek）的所有者。
create_date	timestamp without time zone	创建列加密密钥的时间。
key_acl	aclitem[]	创建该列加密密钥时所拥有的访问权限。

12.2.9 GS_COLUMN_KEYS_ARGS

GS_COLUMN_KEYS_ARGS系统表记录密态等值特性中客户端加密主密钥相关元数据信息，每条记录对应客户端加密主密钥的一个键值对信息。

表 12-9 GS_COLUMN_KEYS_ARGS 字段

名称	类型	描述
oid	oid	行标识符（隐含字段）。
column_key_id	oid	列加密密钥（cek）oid。
function_name	name	值为encryption。
key	name	列加密密钥（cek）的元数据信息对应的名称。
value	bytea	列加密密钥（cek）的元数据信息名称的值。

12.2.10 GS_DATABASE_LINK

GS_DATABASE_LINK系统表是用于存储DATABASE LINK信息的系统表，主要记录的是DATABASE LINK对象的详细信息。只有具备sysadmin权限的用户才可以对该系统表进行读操作。

表 12-10 GS_DATABASE_LINK 字段

名称	类型	描述
oid	oid	当前DATABASE LINK对象的唯一id（隐含属性，必须明确选择）。
dlname	name	当前DATABASE LINK的名称。
dlowner	oid	当前DATABASE LINK的拥有者的id，为public则为0。
dlfdw	oid	当前DATABASE LINK的外部数据封装器的OID。
dlcreater	oid	当前DATABASE LINK创建者的id。
options	text[]	当前DATABASE LINK连接信息，使用"keyword=value"。
useroptions	text[]	当前DATABASE LINK连接远端所使用的用户信息，使用"keyword=value"。
dlacl	aclitem[]	当前DATABASE LINK访问权限。

12.2.11 GS_DB_PRIVILEGE

GS_DB_PRIVILEGE系统表记录ANY权限的授予情况，每条记录对应一条授权信息。

表 12-11 GS_DB_PRIVILEGE 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
roleid	oid	用户标识。
privilege_type	text	用户拥有的ANY权限，取值参考**表7-149**。
admin_option	boolean	是否具有privilege_type列记录的ANY权限的再授权权限。 • t：表示具有。 • f：表示不具有。

12.2.12 GS_DEPENDENCIES

GS_DEPENDENCIES系统表记录对象的依赖项信息，和**13.2.13 GS_DEPENDENCIES_OBJ**表是一个一对多的关系。

表 12-12 GS_DEPENDENCIES 字段

名称	类型	描述
schemaname	name	名称空间的名称。
packagename	name	package的名称。
refobjpos	integer	被依赖体引用的位置。 ● 1：类型。 ● 2：包头。 ● 4：函数头。 ● 8：函数体。 ● 16：包体。 ● 32：视图。
refobjoid	oid	被依赖体的oid。
objectname	text	依赖体名称。

12.2.13 GS_DEPENDENCIES_OBJ

GS_DEPENDENCIES_OBJ系统表记录对象的被依赖项详细信息

表 12-13 GS_DEPENDENCIES_OBJ 字段

名称	类型	描述
schemaname	name	名称空间的名称。
packagename	name	package的名称。
type	integer	被依赖体的类型。 ● 1：未知类型。 ● 2：变量。 ● 3：类型。 ● 4：函数。 ● 5：视图。 ● 6：函数头。
name	text	被依赖体名称。
objnode	pg_node_tree	被依赖体的详细信息。

12.2.14 GS_ENCRYPTED_COLUMNS

GS_ENCRYPTED_COLUMNS系统表记录密态等值特性中表的加密列相关信息，每条记录对应一条加密列信息。

表 12-14 GS_ENCRYPTED_COLUMNS 字段

名称	类型	描述
oid	oid	行标识符（隐含字段）。
rel_id	oid	表的OID。
column_name	name	加密列的名称。
column_key_id	oid	外键，列加密密钥的OID。
encryption_type	tinyint	加密类型，取值为2（DETERMINISTIC）或者1（RANDOMIZED）。
data_type_original_oid	oid	加密列的原始数据类型id，参考系统表PG_TYPE中的oid。
data_type_original_mod	integer	加密列的原始数据类型修饰符，参考系统表PG_ATTRIBUTE中的atttypmod。其值对那些不需要的类型data_type_original_mod通常为-1。
create_date	timestamp without time zone	创建加密列的时间。

12.2.15 GS_ENCRYPTED_PROC

GS_ENCRYPTED_PROC系统表提供了密态函数/存储过程函数参数、返回值的原始数据类型，加密列等信息。

表 12-15 GS_ENCRYPTED_PROC 字段

名称	类型	描述
oid	oid	行标识符（隐含字段）。
func_id	oid	function的oid，对应系统表13.2.84 PG_PROC中的oid行标识符。
prorettype_orig	integer	返回值的原始数据类型。
last_change	timestamp without time zone	密态函数信息上次修改的时间。
proargcachedcol	oidvector	函数INPUT参数对应的加密列的oid，对应系统表13.2.14 GS_ENCRYPTED_COLUMNS中的oid行标识符。
proallargtypes_orig	oid[]	所有函数参数的原始数据类型。

12.2.16 GS_GLOBAL_CONFIG

GS_GLOBAL_CONFIG记录了数据库实例初始化时，用户指定的参数值。除此之外，还存放了用户设置的弱口令，支持数据库初始用户通过ALTER和DROP语法对系统表中的参数进行写入、修改和删除。此系统表默认只有初始用户、系统管理员和安全管理员可以访问，其他用户默认无权访问。

表 12-16 GS_GLOBAL_CONFIG 字段

名称	类型	描述
name	name	数据库实例初始化时系统内置的指定参数名称、弱口令名称、或用户需要使用的参数。
value	text	数据库实例初始化时系统内置的指定参数值、弱口令名称、或用户需要使用的参数值。

12.2.17 GS_JOB_ARGUMENT

GS_JOB_ARGUMENT系统表提供了DBE_SCHEDULER定时任务和程序的参数属性。

表 12-17 GS_JOB_ARGUMENT 字段

名称	类型	描述
oid	oid	行标识符（隐含字段）。
argument_position	integer	定时任务或程序的参数位置。
argument_type	name	定时任务或程序的参数类型。
job_name	text	定时任务或程序名。
argument_name	text	定时任务或程序的参数名（定时任务继承了程序的参数名，所以为空）。
argument_value	text	定时任务的参数值（程序本身无法绑定值）。
default_value	text	程序的参数默认值。

12.2.18 GS_JOB_ATTRIBUTE

GS_JOB_ATTRIBUTE系统表提供了DBE_SCHEDULER定时任务的相关属性信息，其中包括定时任务，定时任务类，证书，授权，程序和调度的基本属性。新安装数据库实例普通用户无权限访问。

表 12-18 GS_JOB_ATTRIBUTE 字段

名称	类型	描述
oid	oid	行标识符（隐含字段）。

名称	类型	描述
job_name	text	定时任务，定时任务类，证书，程序和调度的名字，授权的用户名。
attribute_name	text	定时任务，定时任务类，证书，程序和调度的属性名，授权的内容。
attribute_value	text	定时任务，定时任务类，证书，程序和调度的属性值。

12.2.19 GS_MASKING_POLICY

GS_MASKING_POLICY系统表记录动态数据脱敏策略的主体信息，每条记录对应一个脱敏策略。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

表 12-19 GS_MASKING_POLICY 表字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
polname	name	策略名称，唯一不可重复。
polcomments	name	策略描述字段，记录策略相关的描述信息，通过COMMENTS关键字体现。
modifydate	timestamp without time zone	策略创建或修改的最新时间戳。
polenabled	boolean	策略启动开关。 ● t（true）：表示策略启动。 ● f（false）：表示策略没有启动。

12.2.20 GS_MASKING_POLICY_ACTIONS

GS_MASKING_POLICY_ACTIONS系统表记录动态数据脱敏策略中相应的脱敏策略包含的脱敏行为，一个脱敏策略对应着该表的一行或多行记录。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

表 12-20 GS_MASKING_POLICY_ACTIONS 表字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。

名称	类型	描述
actiontype	name	脱敏函数，标识脱敏策略使用的脱敏函数。
actparams	name	向脱敏函数中传递的参数信息。
actlabelname	name	被脱敏的label名称。
policyoid	oid	该条记录所属的脱敏策略oid，对应GS_MASKING_POLICY中的oid。
actmodifydate	timestamp without time zone	该条记录创建或修改的最新时间戳。

12.2.21 GS_MASKING_POLICY_FILTERS

GS_MASKING_POLICY_FILTERS系统表记录动态数据脱敏策略对应的用户过滤条件，当用户条件满足FILTER条件时，对应的脱敏策略才会生效。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

表 12-21 GS_MASKING_POLICY_FILTERS 表字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
filtertype	name	过滤类型。目前值仅为logical_expr。
filterlabelname	name	过滤范围。目前值仅为logical_expr。
policyoid	oid	该条用户过滤条件所属的脱敏策略oid，对应GS_MASKING_POLICY中的oid。
modifydate	timestamp without time zone	该条用户过滤条件创建或修改的最新时间戳。
logicaloperator	text	过滤条件的波兰表达式。

12.2.22 GS_MATVIEW

GS_MATVIEW系统表提供了关于数据库中每一个物化视图的信息。

表 12-22 GS_MATVIEW 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。

名称	类型	描述
matviewid	oid	物化视图的oid。
mapid	oid	物化视图map表的oid，map表为物化视图关联表，与物化视图一一对应。全量物化视图不存在对应的map表，该字段为0。
ivm	boolean	物化视图的类型，t为增量物化视图，f为全量物化视图。
needrefresh	boolean	保留字段。
refreshtime	timestamp without time zone	物化视图上一次刷新时间，若未刷新则为null。仅对增量物化视图维护该字段，全量物化视图为null。

12.2.23 GS_MATVIEW_DEPENDENCY

GS_MATVIEW_DEPENDENCY系统表提供了关于数据库中每一个增量物化视图、基表和mlog表的关联信息。全量物化视图不存在与基表对应的mlog表，不会写入记录。

表 12-23 GS_MATVIEW_DEPENDENCY 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
matviewid	oid	物化视图的oid。
relid	oid	物化视图基表的oid。
mlogid	oid	物化视图mlog表的oid，mlog表为物化视图日志表，与基表一一对应。
mxmin	integer	保留字段。

12.2.24 GS_MODEL_WAREHOUSE

GS_MODEL_WAREHOUSE系统表用于存储AI引擎训练模型，其中包含模型，训练过程的详细描述。

表 12-24 GS_MODEL_WAREHOUSE 字段

名称	数据类型	描述
oid	oid	隐含列。
modelname	name	唯一约束。

名称	数据类型	描述
modelowner	oid	模型拥有者的OID。
createtime	timestamp without time zone	模型创建的时间。
processedtuples	integer	训练涉及的元组数。
discardedtuples	integer	未参加训练的不合格元组数。
preprocesstime	real	数据预处理时长。
exectime	real	训练时长。
iterations	integer	迭代轮次。
outputtype	oid	模型输出的数据类型OID。
modeltype	text	AI算子的类型名称。
query	text	创建模型所执行的query语句。
modeldata	bytea	保存的二进制模型信息。
weight	real[]	目前只适用于GD算子模型。
hyperparametersnames	text[]	涉及的超参名称。
hyperparametersvalues	text[]	超参所对应的取值。
hyperparametersoids	oid[]	超参对应的数据类型OID。
coefnames	text[]	模型参数名称。
coefvalues	text[]	模型参数对应的取值。
coefoids	oid[]	模型参数对应的数据类型OID。
trainingscoresname	text[]	度量模型性能方法的名称。
trainingscoresvalue	real[]	度量模型性能方法的数值。
modeldescribe	text[]	模型的描述信息。

12.2.25 GS_OPT_MODEL

GS_OPT_MODEL是启用AiEngine执行计划时间预测功能时的数据表，记录机器学习模型的配置、训练结果、功能、对应系统函数、训练历史等相关信息。

表 12-25 GS_OPT_MODEL 字段

名称	类型	描述
template_name	name	机器学习模型的模板名，决定训练和预测调用的函数接口，目前只实现了rlstm，方便后续扩展。
model_name	name	模型的实例名，每个模型对应AiEngine在线学习进程中的一套参数、训练日志、模型系数。此列需为unique。
datname	name	该模型所服务的database名，每个模型只针对单个database。此参数决定训练时所使用的数据。
ip	name	AiEngine端所部署的host ip地址。
port	integer	AiEngine端所侦听的端口号。
max_epoch	integer	模型每次训练的迭代次数上限。
learning_rate	real	模型训练的学习速率，推荐缺省值1。
dim_red	real	模型特征维度降维系数。
hidden_units	integer	模型隐藏层神经元个数。如果训练发现模型长期无法收敛，可以适量提升本参数。
batch_size	integer	模型每次迭代时一个batch的大小，尽量设为大于等于训练数据总量的值，加快模型的收敛速度。
feature_size	integer	[不需设置] 模型特征的长度，用于触发重新训练，模型训练后该参数自动更新。
available	boolean	[不需设置]标识模型是否收敛。
Is_training	boolean	[不需设置]标识模型是否正在训练。

名称	类型	描述
label	"char"[]	模型的目标任务： • S: startup time • T: total time • R: rows • M: peak memory 目前受模型性能限制，推荐{S, T}或{R}。
max	bigint[]	[不需设置]标识模型各任务标签的最大值，用于触发重新训练。
acc	real[]	[不需设置]标识模型各任务的准确率。
description	text	模型注释。

12.2.26 GS_PACKAGE

GS_PACKAGE系统表记录PACKAGE内的信息。

表 12-26 GS_PACKAGE 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
pkgnamespace	oid	package所属schema。
pkgowner	oid	package的所属者。
pkgname	name	package的名字。
pkgspecsrc	text	package specification的内容。
pkgbodydeclsrc	text	package body的内容。
pkgbodyinitsrc	text	package init的内容。
pkgacl	aclitem[]	访问权限。
pkgsecdef	boolean	package是否是定义者权限。

12.2.27 GS_PLAN_TRACE

GS_PLAN_TRACE系统表是用于存储plan trace的系统表，主要记录的是DML语句生成计划过程的详情，只有初始用户才具有对该系统表进行写的权限，只要用户具备sysadmin权限就可以对该系统表进行读操作。

表 12-27 GS_PLAN_TRACE 字段

名称	类型	描述
query_id	text	当前请求的唯一id。
query	text	当前请求的sql语句，该字段大小不会超过系统参数track_activity_query_size指定的大小。
unique_sql_id	bigint	当前请求sql的唯一id。
plan	text	当前请求sql对应的查询计划文本，该字段大小不会超过10K。
plan_trace	text	当前请求sql对应的查询计划生成过程的明细，该字段大小不会超过300M。
owner	oid	当前请求sql用户的oid。
modifydate	timestamp with time zone	当前plan trace的更新时间（当前指的是 plan trace创建时间）。

12.2.28 GS_POLICY_LABEL

GS_POLICY_LABEL系统表记录资源标签配置信息，一个资源标签对应着一条或多条记录，每条记录标记了数据库资源所属的资源标签。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

FQDN（Fully Qualified Domain Name）标识了数据库资源所属的绝对路径。

表 12-28 GS_POLICY_LABEL 表字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
labelname	name	资源标签名称。
labeltype	name	资源标签类型，目前仅为RESOURCE。
fqdnnamespace	oid	被标识的数据库资源所属的namespace oid。
fqdnid	oid	被标识的数据库资源的oid，若数据库资源为列，则该列为所属表的oid。
relcolumn	name	列名，若被标识的数据库资源为列，该列指出列名，否则该列为空。

名称	类型	描述
fqdntype	name	被标识的数据库资源的类型名称，例如：schema, table, column, view等。

12.2.29 GS_RECYCLEBIN

gs_recyclebin描述了回收站对象的详细信息。

表 12-29 gs_recyclebin 字段

名称	类型	描述
oid	oid	系统列。
rcybaseid	oid	基表对象id，引用gs_recyclebin.oid。
rcydbid	oid	当前对象所属数据库oid。
rcyrelid	oid	当前对象oid。
rcyname	name	回收站对象名称，格式“BIN$unique_id$oid$0”，其中unique_id为最多16字符唯一标识，oid为对象标识符。
rcyoriginname	name	原始对象名称。
rcyoperation	"char"	操作类型。 ● d表示drop。 ● t表示truncate。
rcytype	integer	对象类型。 ● 0表示table。 ● 1表示index。 ● 2表示toast table。 ● 3表示toast index。 ● 4表示sequence，指serial、bigserial、smallserial、largeserial类型自动关联的序列对象。 ● 5表示partition。 ● 6表示global index。 ● 7表示物化视图。
rcyrecyclecsn	bigint	对象drop、truncate时csn。
rcyrecycletime	timestamp with time zone	对象drop、truncate时间。

名称	类型	描述
rcycreatecsn	bigint	对象创建时csn。
rcychangecsn	bigint	对象定义改变的csn。
rcynamespace	oid	包含这个关系的名字空间的OID。
rcyowner	oid	关系所有者。
rcytablespace	oid	这个关系存储所在的表空间。如果为0，则意味着使用该数据库的缺省表空间。如果关系在磁盘上没有文件，则这个字段没有什么意义。
rcyrelfilenode	oid	回收站对象在磁盘上的文件的名称，如果没有则为0，用于TRUNCATE对象恢复时物理文件还原。
rcycanrestore	boolean	是否可以被单独闪回。
rcycanpurge	boolean	是否可以被单独purge。
rcyfrozenxid	xid32	该表中所有在这个之前的事务ID已经被一个固定的（"frozen"）事务ID替换。
rcyfrozenxid64	xid	该表中所有在这个之前的事务ID已经被一个固定的（"frozen"）事务ID替换。

12.2.30 GS_SQL_PATCH

GS_SQL_PATCH系统表存储所有SQL_PATCH的状态信息。

表 12-30 GS_SQL_PATCH 字段

名称	类型	描述
patch_name	name	PATCH名称。
unique_sql_id	bigint	查询全局唯一ID。
owner	oid	PATCH的创建用户ID。
enable	boolean	PATCH是否生效。
status	"char"	PATCH的状态（预留字段）。
abort	boolean	是否是AbortHint。
hint_string	text	Hint文本。
hint_node	pg_node_tree	Hint解析&序列化的结果。
original_query	text	原始语句（预留字段）。

名称	类型	描述
patched_query	text	PATCH之后的语句（预留字段）。
original_query_tree	pg_node_tree	原始语句的解析结果（预留字段）。
patched_query_tree	pg_node_tree	PATCH之后语句的解析结果（预留字段）。
description	text	PATCH的备注。
parent_unique_sql_id	bigint	PATCH生效的SQL语句的外层语句的全局唯一ID，存储过程外的语句该值为0，存储过程内的语句该值为调用该存储过程语句的全局唯一ID。

12.2.31 GS_TXN_SNAPSHOT

GS_TXN_SNAPSHOT是“时间戳-CSN”映射表，周期性采样，并维护适当的时间范围，用于估算范围内的时间戳对应的CSN值。

表 12-31 GS_TXN_SNAPSHOT 字段

名称	类型	描述
snptime	timestamp with time zone	快照捕获时间。
snpxmin	bigint	快照xmin。
snpcsn	bigint	快照csn。
snpsnapshot	text	快照序列化文本。

12.2.32 GS_UID

GS_UID系统表存储了数据库中使用hasuids属性表的唯一标识元信息。

表 12-32 GS_UID 字段

名称	类型	描述
relid	oid	表的oid信息。
uid_backup	bigint	当前可以为表分配唯一标识的最大值。

12.2.33 GS_WORKLOAD_RULE

GS_WORKLOAD_RULE系统表存储与SQL限流规则相关的信息。该系统表没有权限限制，所有用户可查询。

表 12-33 GS_WORKLOAD_RULE 字段

名称	类型	描述
rule_id	bigint	限流规则标识列，系统自动生成。
rule_name	name	限流规则的名称，用于检索限流规则，不保证唯一性，可以为NULL。
databases	name[]	限流规则作用的数据库列表，为NULL表示所有库生效。
max_workload	bigint	限制规则设置的最大并发数。
is_valid	boolean	限流规则是否生效，超时的限流规则会设为false。
start_time	timestamp with time zone	限流规则开始的时间，为NULL表示从现在开始生效。
end_time	timestamp with time zone	限流规则结束的时间，为NULL表示一直生效。
rule_type	text	限流规则类型，当前仅支持：“sqlid”、“select”、“insert”、“update”、“delete”、“merge”、“resource”，其他的为非法值。
option_val	text[]	限流规则的参数值，包括：sqlid，关键字列表，资源限制情况。 详细请参见**gs_add_workload_rule**接口说明。
node_names	text[]	预留字段，限流规则生效的节点名称列表，当前不生效。
user_names	text[]	预留字段，限流规则生效的用户名称列表，当前不生效。

12.2.34 PG_AGGREGATE

PG_AGGREGATE系统表存储与聚集函数有关的信息。PG_AGGREGATE里的每条记录都是一条pg_proc里面的记录的扩展。PG_PROC记录承载该聚集的名称、输入和输出数据类型，以及其它一些和普通函数类似的信息。

表 12-34 PG_AGGREGATE 字段

名称	类型	引用	描述
aggfnoid	regproc	PG_PROC.proname	此聚集函数的PG_PROC proname。
aggtransfn	regproc	PG_PROC.proname	转换函数。
aggcollectfn	regproc	PG_PROC.proname	收集函数。
aggfinalfn	regproc	PG_PROC.proname	最终处理函数（如果没有则为零）。
aggsortop	oid	PG_OPERATOR.oid	关联排序操作符（如果没有则为零）。
aggtranstype	oid	PG_TYPE.oid	此聚集函数的内部转换（状态）数据的数据类型。 可能取值及其含义见于pg_type.h中type定义，主要分为多态（isPolymorphicType）和非多态两类。
agginitval	text	-	转换状态的初始值。这是一个文本数据域，它包含初始值的外部字符串表现形式。如果数据域是null，则转换状态值从null开始。
agginitcollect	text	-	收集状态的初始值。这是一个文本数据域，它包含初始值的外部字符串表现形式。如果数据域是null，则收集状态值从null开始。
aggkind	"char"	-	此聚集函数类型： • 'n'：表示Normal Agg • 'o'：表示Ordered Set Agg
aggnumdirectargs	smallint	-	Ordered Set Agg类型聚集函数的直接参数（非聚集相关参数）数量。对Normal Agg类型聚集函数，该值为0。

12.2.35 PG_AM

PG_AM系统表存储有关索引访问方法的信息。系统支持的每种索引访问方法都有一行。

表 12-35 PG_AM 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
amname	name	-	访问方法的名称。
amstrategies	smallint	-	访问方法的操作符策略个数，或者如果访问方法没有一个固定的操作符策略集则为0。
amsupport	smallint	-	访问方法的支持过程个数。
amcanorder	boolean	-	这种访问方式是否支持通过索引字段值的命令扫描排序。
amcanorderbyop	boolean	-	这种访问方式是否支持通过索引字段上操作符的结果的命令扫描排序。
amcanbackward	boolean	-	访问方式是否支持向后扫描。
amcanunique	boolean	-	访问方式是否支持唯一索引。
amcanmulticol	boolean	-	访问方式是否支持多字段索引。
amoptionalkey	boolean	-	访问方式是否支持第一个索引字段上没有任何约束的扫描。
amsearcharray	boolean	-	访问方式是否支持ScalarArrayOpExpr搜索。
amsearchnulls	boolean	-	访问方式是否支持IS NULL/NOT NULL搜索。
amstorage	boolean	-	是否允许索引存储的数据类型与列的数据类型不同。
amclusterable	boolean	-	是否允许在一个这种类型的索引上聚簇。
ampredlocks	boolean	-	是否允许这种类型的一个索引管理细粒度的谓词锁定。
amkeytype	oid	PG_TYPE.oid	存储在索引里数据的类型，如果不是一个固定的类型则为0。
aminsert	regproc	PG_PROC.proname	“插入这个行”函数。
ambeginscan	regproc	PG_PROC.proname	“准备索引扫描”函数。
amgettuple	regproc	PG_PROC.proname	“下一个有效行”函数，如果没有则为0。

名称	类型	引用	描述
amgetbitmap	regproc	PG_PROC.proname	“抓取所有的有效行” 函数，如果没有则为0。
amrescan	regproc	PG_PROC.proname	“（重新）开始索引扫描”函数。
amendscan	regproc	PG_PROC.proname	“索引扫描后清理” 函数。
ammarkpos	regproc	PG_PROC.proname	“标记当前扫描位置”函数。
amrestrpos	regproc	PG_PROC.proname	“恢复已标记的扫描位置”函数。
ammerge	regproc	PG_PROC.proname	“归并多个索引对象”函数。
ambuild	regproc	PG_PROC.proname	“建立新索引”函数。
ambuildempty	regproc	PG_PROC.proname	“建立空索引”函数。
ambulkdelete	regproc	PG_PROC.proname	批量删除函数。
amvacuumcleanup	regproc	PG_PROC.proname	VACUUM后的清理函数。
amcanreturn	regproc	PG_PROC.proname	检查是否索引支持唯一索引扫描的函数，如果没有则为0。
amcostestimate	regproc	PG_PROC.proname	估计一个索引扫描开销的函数。
amoptions	regproc	PG_PROC.proname	为一个索引分析和确认reloptions的函数。

12.2.36 PG_AMOP

PG_AMOP系统表存储有关和访问方法操作符族关联的信息。如果一个操作符是一个操作符族中的成员，则在这个表中会占据一行。一个族成员是一个search操作符或一个ordering操作符。一个操作符可以在多个族中出现，但是不能在一个族中的多个搜索位置或多个排序位置中出现。

表 12-36 PG_AMOP 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。

名称	类型	引用	描述
amopfamily	oid	PG_OPFAMILY.oid	这个项的操作符族。
amoplefttype	oid	PG_TYPE.oid	操作符的左输入类型。可能取值及其描述见于8.3 数据类型。
amoprighttype	oid	PG_TYPE.oid	操作符的右输入类型。可能取值及其描述见于8.3 数据类型。
amopstrategy	smallint	-	操作符策略数。
amoppurpose	"char"	-	操作符目的。 ● s：表示搜索。 ● o：表示排序。
amopopr	oid	PG_OPERATOR.oid	该操作符的OID。
amopmethod	oid	PG_AM.oid	索引访问方式操作符族。
amopsortfamily	oid	PG_OPFAMILY.oid	如果是一个排序操作符，则为这个项排序所依据的btree操作符族；如果是一个搜索操作符，则为0。

search操作符表明这个操作符族的一个索引可以被搜索，找到所有满足WHERE indexed_column operator constant的行。显然，这样的操作符必须返回布尔值，并且它的左输入类型必须匹配索引的字段数据类型。

ordering操作符表明这个操作符族的一个索引可以被扫描，返回以ORDER BY indexed_column operator constant顺序表示的行。这样的操作符可以返回任意可排序的数据类型，它的左输入类型也必须匹配索引的字段数据类型。 ORDER BY的确切的语义是由amopsortfamily字段指定的，该字段必须为操作符的返回类型引用一个btree操作符族。

12.2.37 PG_AMPROC

PG_AMPROC系统表存储有关与访问方法操作符族相关联的支持过程的信息。每个属于某个操作符族的支持过程都占有一行。

表 12-37 PG_AMPROC 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
amprocfamily	oid	PG_OPFAMILY.oid	该项的操作符族。

名称	类型	引用	描述
amproclefttype	oid	PG_TYPE.oid	相关操作符的左输入数据类型。可能取值及其描述见于8.3 数据类型。
amprocrighttype	oid	PG_TYPE.oid	相关操作符的右输入数据类型。可能取值及其描述见于8.3 数据类型。
amprocnum	smallint	-	支持过程编号。
amproc	regproc	PG_PROC.proname	过程的OID。

amproclefttype和amprocrighttype字段的习惯解释，标识一个特定支持过程支持的操作符的左和右输入类型。对于某些访问方式，匹配支持过程本身的输入数据类型，对其他的则不这样。有一个对索引的“缺省”支持过程的概念，amproclefttype和amprocrighttype都等于索引操作符类的opcintype。

12.2.38 PG_APP_WORKLOADGROUP_MAPPING

PG_APP_WORKLOADGROUP_MAPPING系统表提供了数据库负载映射组的信息。

表 12-38 PG_APP_WORKLOADGROUP_MAPPING 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
appname	name	应用名称。
workload_gpname	name	映射到的负载组名称。

12.2.39 PG_ATTRDEF

PG_ATTRDEF系统表存储列的默认值。

表 12-39 PG_ATTRDEF 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
adrelid	oid	该列的所属表。
adnum	smallint	该列的数目。
adbin	pg_node_tree	字段缺省值或生成表达式的内部表现形式。

名称	类型	描述
adsrc	text	可读缺省值或生成表达式的内部表现形式。
adgencol	"char"	标识该列是否为生成列。取值为's'表示该列为生成列，取值为'\0'表示该列为普通列，默认值为'\0'。

12.2.40 PG_ATTRIBUTE

PG_ATTRIBUTE系统表存储关于表字段的信息。

表 12-40 PG_ATTRIBUTE 字段

名称	类型	描述
attrelid	oid	此字段所属表。
attname	name	字段名。
atttypid	oid	字段类型。
attstattarget	integer	控制ANALYZE为这个字段积累的统计细节的级别。 • 零值表示不收集统计信息。 • 负数表示使用系统缺省的统计对象。 • 正数值的确切信息是和数据类型相关的。 对于标量数据类型，ATTSTATTARGET既是要收集的"最常用数值"的目标数目，也是要创建的柱状图的目标数量。
attlen	smallint	是本字段类型的**13.2.108 PG_TYPE**中typlen的拷贝。
attnum	smallint	字段编号。
attndims	integer	如果该字段是数组，则是维数，否则是0 。
attcacheoff	integer	在磁盘上的时候总是-1 ，但是如果加载入内存中的行描述器中，它可能会被更新以缓冲在行中字段的偏移量。
atttypmod	integer	记录创建新表时支持的类型特定的数据（比如一个varchar字段的最大长度）。它传递给类型相关的输入和长度转换函数当做第三个参数。其值对那些不需要ATTTYPMOD的类型通常为-1。
attbyval	boolean	这个字段类型的**13.2.108 PG_TYPE**中typbyval的拷贝。
attstorage	"char"	这个字段类型的**13.2.108 PG_TYPE**中typstorage的拷贝。
attalign	"char"	这个字段类型的**13.2.108 PG_TYPE**中typalign的拷贝。
attnotnull	boolean	这代表一个非空约束。可以改变这个字段以打开或者关闭这个约束。

名称	类型	描述
atthasdef	boolean	这个字段有一个缺省值，此时它对应13.2.47 PG_ATTRDEF表里实际定义此值的记录。
attisdropped	boolean	这个字段已经被删除了，不再有效。一个已经删除的字段物理上仍然存在表中，但会被分析器忽略，因此不能再通过SQL访问。
attislocal	boolean	这个字段是局部定义在关系中的。请注意一个字段可以同时是局部定义和继承的。
attcmprmode	tinyint	对某一列指定压缩方式。压缩方式包括： ● 0：ATT_CMPR_NOCOMPRESS，不压缩 ● 1：ATT_CMPR_DELTA，DELTA压缩算法 ● 2：ATT_CMPR_DICTIONARY，字典压缩算法 ● 3：ATT_CMPR_PREFIX，前缀压缩算法 ● 4：ATT_CMPR_NUMSTR，数字字符串压缩算法
attinhcount	integer	这个字段所拥有的直接父表的个数。如果一个字段的父表个数非零，则它就不能被删除或重命名。
attcollation	oid	对此列定义的校对列。
attacl	aclitem[]	列级访问权限控制。
attoptions	text[]	字段属性。目前支持以下两种属性： n_distinct，表示该字段的distinct值数量（不包含字表） n_distinct_inherited，表示该字段的distinct值数量（包含字表）
attfdwoptions	text[]	外表字段属性。当前支持的dist_fdw、file_fdw、log_fdw未使用外表字段属性。
attinitdefval	bytea	存储了此列默认的值表达式。行存表的ADD COLUMN需要使用此字段。
attkvtype	tinyint	对某一列指定key value类型。类型包括： 0：ATT_KV_UNDEFINED，默认。 1：ATT_KV_TAG，维度。 2：ATT_KV_FIELD，指标。 3：ATT_KV_TIMETAG，时间列。

12.2.41 PG_AUTHID

PG_AUTHID系统表存储有关数据库认证标识符（角色）的信息。角色把“用户”的概念包含在内。一个用户实际上就是一个rolcanlogin标志被设置的角色。任何角色（不管rolcanlogin设置与否）都能够把其他角色作为成员。

GaussDB中只有一份pg_authid，不是每个数据库有一份。需要有系统管理员权限才可以访问此系统表。

表 12-41 PG_AUTHID 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
rolname	name	角色名称。
rolsuper	boolean	角色是否是拥有最高权限的初始系统管理员。 ● t（true）：表示是。 ● f（false）：表示不是。
rolinherit	boolean	角色是否自动继承其所属角色的权限。 ● t（true）：表示自动继承。 ● f（false）：表示不自动继承。
rolcreaterole	boolean	角色是否可以创建更多角色。 ● t（true）：表示可以。 ● f（false）：表示不可以。
rolcreatedb	boolean	角色是否可以创建数据库。 ● t（true）：表示可以。 ● f（false）：表示不可以。
rolcatupdate	boolean	角色是否可以直接更新系统表。只有usesysid=10的初始系统管理员拥有此权限。其他用户无法获得此权限。 ● t（true）：表示可以。 ● f（false）：表示不可以。
rolcanlogin	boolean	角色是否可以登录，也就是说，这个角色可以给予会话认证标识符。 ● t（true）：表示可以。 ● f（false）：表示不可以。
rolreplication	boolean	角色是否具有复制权限。 ● t（true）：表示有。 ● f（false）：表示没有。
rolauditadmin	boolean	角色是否具有审计管理员权限。 ● t（true）：表示有。 ● f（false）：表示没有。
rolsystemadmin	boolean	角色是否具有系统管理员权限。 ● t（true）：表示有。 ● f（false）：表示没有。

名称	类型	描述
rolconnlimit	integer	对于可以登录的角色，限制其最大并发连接数量。 -1 表示没有限制。
rolpassword	text	密码（可能是加密的），如果没有密码，则为NULL。
rolvalidbegin	timestamp with time zone	账户的有效开始时间，如果没有开始时间，则为NULL。
rolvaliduntil	timestamp with time zone	账户的有效结束时间，如果没有结束时间，则为NULL。
rolrespool	name	用户所能够使用的resource pool。
roluseft	boolean	角色是否可以操作外表。 ● t（true）：表示可以。 ● f（false）：表示不可以。
rolparentid	oid	用户所在组用户的OID。
roltabspace	text	用户数据表的最大空间限额。
rolkind	"char"	特殊用户种类。 ● n：表示普通用户。 ● p：表示永久用户。
rolnodegroup	oid	该字段不支持。
roltempspace	text	用户临时表的最大空间限额，单位为KB。
rolspillspace	text	用户执行作业时下盘数据的最大空间限额，单位为KB。
rolexcpdata	text	用户可以设置的查询规则（当前未使用）。
rolmonitoradmin	boolean	角色是否具有监控管理员权限。 ● t（true）：表示有。 ● f（false）：表示没有。
roloperatoradmin	boolean	角色是否具有运维管理员权限。 ● t（true）：表示有。 ● f（false）：表示没有。
rolpolicyadmin	boolean	角色是否具有安全策略管理员权限。 ● t（true）：表示有。 ● f（false）：表示没有。

12.2.42 PG_AUTH_HISTORY

PG_AUTH_HISTORY系统表记录了角色的认证历史。需要有系统管理员权限才可以访问此系统表。

表 12-42 PG_AUTH_HISTORY 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
roloid	oid	角色标识。
passwordtime	timestamp with time zone	创建和修改密码的时间。
rolpassword	text	角色密码密文，加密方式由GUC参数password_encryption_type确定。

12.2.43 PG_AUTH_MEMBERS

PG_AUTH_MEMBERS系统表存储显示角色之间的成员关系。

表 12-43 PG_AUTH_MEMBERS 字段

名称	类型	描述
roleid	oid	拥有成员的角色ID。
member	oid	属于ROLEID角色的一个成员的角色ID。
grantor	oid	赋予此成员关系的角色ID。
admin_option	boolean	如果MEMBER可以把ROLEID角色的成员关系赋予其他角色则为真，不可以则为假。

12.2.44 PG_CAST

PG_CAST系统表存储数据类型之间的转化关系。

表 12-44 PG_CAST 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
castsource	oid	源数据类型的OID。
casttarget	oid	目标数据类型的OID。
castfunc	oid	转化函数的OID。如果为零表明不需要转化函数。

名称	类型	描述
castcontext	"char"	源数据类型和目标数据类型间的转化方式： ● 'e'：表示只能进行显式转化（使用CAST或::语法）。 ● 'i'：表示能进行隐式转化。 ● 'a'：表示类型间同时支持隐式和显式转化。
castmethod	"char"	转化方法： ● 'f'：使用castfunc字段中指定的函数进行转化。 ● 'b'：类型间是二进制强制转化，不使用castfunc。

12.2.45 PG_CLASS

PG_CLASS系统表存储数据库对象信息及其之间的关系。

表 12-45 PG_CLASS 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
relname	name	表、索引、视图等对象的名称。
relnamespace	oid	包含这个关系的名称空间的OID。
reltype	oid	对应这个表的行类型的数据类型（索引为零，因为索引没有pg_type记录）。
reloftype	oid	复合类型的OID，0表示其他类型。
relowner	oid	关系所有者。
relam	oid	如果行是索引，则就是所用的访问模式（B-tree等）。
relfilenode	oid	这个关系在磁盘上的文件的名称，如果没有则为0。
reltablespace	oid	这个关系存储所在的表空间。如果为零，则意味着使用该数据库的缺省表空间。如果关系在磁盘上没有文件，则这个字段没有什么意义。
relpages	double precision	以页（大小为BLCKSZ）为单位的此表在磁盘上的大小，它只是优化器用的一个近似值。
reltuples	double precision	表中行的数目，只是优化器使用的一个估计值。

名称	类型	描述
relallvisible	integer	被标识为全可见的表中的页的数量。此字段是优化器用来做SQL执行优化使用的。VACUUM、ANALYZE和一些DDL语句（例如，CREATE INDEX）会引起此字段更新。
reltoastrelid	oid	与此表关联的TOAST表的OID，如果没有则为0。 TOAST表在一个从属表里“离线”存储大字段。
reltoastidxid	oid	对于TOAST表是它的索引的OID，如果不是TOAST表则为0。
relhasindex	boolean	如果它是一个表而且至少有（或者最近有过）一个索引，则为真。 它是由CREATE INDEX设置的，但DROP INDEX不会立即将它清除。如果VACUUM进程检测一个表没有索引，将会把它将清理relhasindex字段，将值设置为假。
relisshared	boolean	如果该表在数据库中由所有数据库共享则为真。只有某些系统表（比如pg_database）是共享的。
relpersistence	"char"	• p：表示永久表。 • u：表示非日志表。 • t：表示临时表。 • g：表示全局临时表。
relkind	"char"	• r：表示普通表。 • i：表示索引。 • I：表示分区表GLOBAL索引。 • S：表示序列。 • L：表示长序列。 • v：表示视图。 • c：表示复合类型。 • t：表示TOAST表。 • f：表示外表。 • m：表示物化视图。
relnatts	smallint	关系中用户字段数目（除了系统字段以外）。在13.2.48 PG_ATTRIBUTE里肯定有相同数目对应行。
relchecks	smallint	表里的检查约束的数目，参阅13.2.55 PG_CONSTRAINT表。
relhasoids	boolean	如果为关系中每行都生成一个OID则为真，否则为假。
relhaspkey	boolean	如果这个表有一个（或者曾经有一个）主键则为真，否则为假。

名称	类型	描述
relhasrules	boolean	如表有规则就为真。是否有规则可参考系统表13.2.90 PG_REWRITE。
relhastriggers	boolean	True表示表中有触发器，或者曾经有过触发器。系统表13.2.102 PG_TRIGGER中记录了表和视图的触发器。
relhassubclass	boolean	如果有（或者曾经有）任何继承的子表为真，否则为假。
relcmprs	tinyint	表示是否启用表的压缩特性。需要特别注意，当且仅当批量插入才会触发压缩，普通的CRUD并不能够触发压缩。 • 0表示其他不支持压缩的表（主要是指系统表，不支持压缩属性的修改操作）。 • 1表示表数据的压缩特性为NOCOMPRESS或者无指定关键字。 • 2表示表数据的压缩特性为COMPRESS。
relhasclusterkey	boolean	是否有局部聚簇存储。 • true：表示有。 • false：表示没有。
relrowmovement	boolean	针对分区表进行update操作时，是否允许行迁移。 • true：表示允许行迁移。 • false：表示不允许行迁移。
parttype	"char"	表或者索引是否具有分区表的性质。 • p：表示带有分区表性质。 • n：表示没有分区表特性。 • s：表示该表为二级分区表。
relfrozenxid	xid32	该表中所有在这个之前的事务ID已经被一个固定的（"frozen"）事务ID替换。该字段用于跟踪此表是否需要为了防止事务ID重叠（或者允许收缩pg_clog）而进行清理。如果该关系不是表则为零（InvalidTransactionId）。 为保持前向兼容，保留此字段，新增relfrozenxid64用于记录此信息。
relacl	aclitem[]	访问权限。aclitem类型说明可以参考aclitem类型。 查询的回显结果为以下形式： *user1=privs/user2* 表示user2赋予user1的权限为*privs* *=privs/user3* 表示user3赋予public角色的权限为*privs* 其中user1，user2和user3为数据库中已存在的用户/角色名，privs为数据库中支持的权限。权限的参数说明请参见表12-46。

名称	类型	描述
reloptions	text[]	表或索引的访问方法，使用"keyword=value"格式的字符串。
relreplident	"char"	逻辑解码中解码列的标识： ● d = 默认（主键，如果存在）。 ● n = 无。 ● f = 所有列。 ● i = 索引的indisreplident被设置或者为默认。
relfrozenxid64	xid	该表中所有在这个之前的事务ID已经被一个固定的（"frozen"）事务ID替换。该字段用于跟踪此表是否需要为了防止事务ID重叠（或者允许收缩pg_clog）而进行清理。如果该关系不是表则为零（InvalidTransactionId）。 对于全局临时表，该字段无实际意义。各会话的全局临时表的relfrozenxid64可在pg_catalog.pg_gtt_relstats视图中查看。
relbucket	oid	当前表是否包含hash bucket分片。有效的OID指向pg_hashbucket表中记录的具体分片信息。NULL表示不包含hash bucket分片。
relbucketkey	int2vector	表示hash分区列信息，NULL表示不包含。
relminmxid	xid	该表中所有在这个之前的多事务ID已经被一个事务ID替换。该字段用于跟踪该表是否需要为了防止多事务ID重叠或者允许收缩pg_clog而进行清理。如果该关系不是表则为零（InvalidTransactionId）。

表 12-46 权限的参数说明

参数	参数说明
r	SELECT（读）
w	UPDATE（写）
a	INSERT（插入）
d	DELETE
D	TRUNCATE
x	REFERENCES
t	TRIGGER
X	EXECUTE
U	USAGE

参数	参数说明
C	CREATE
c	CONNECT
T	TEMPORARY
A	ALTER
P	DROP
m	COMMENT
i	INDEX
v	VACUUM
*	给前面权限的授权选项

12.2.46 PG_COLLATION

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

表 12-47 PG_COLLATION 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
collname	name	-	排序规则名（每个名称空间和编码唯一）。
collnamespace	oid	PG_NAMESPACE.oid	包含这个排序规则的名称空间的OID。
collowner	oid	PG_AUTHID.oid	排序规则的所有者。
collencoding	integer	-	排序规则可用的编码，兼容PostgreSQL所有的字符编码类型，如果适用于任意编码为-1。
collcollate	name	-	这个排序规则对象的LC_COLLATE。
collctype	name	-	这个排序规则对象的LC_CTYPE。
collpadattr	text	-	这个排序规则的填充属性。 ● NULL：不适用。 ● NO PAD：无填充。 ● PAD SPACE：末尾空白填充。

名称	类型	引用	描述
collisdef	boolean	-	这个排序规则是否是所属字符集的默认字符序。

12.2.47 PG_CONSTRAINT

PG_CONSTRAINT系统表存储表上的检查约束、主键和唯一约束。

表 12-48 PG_CONSTRAINT 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
conname	name	约束名称（不一定是唯一的）。
connamespace	oid	包含这个约束的名称空间的OID。
contype	"char"	● c：检查约束。 ● p：主键约束。 ● u：唯一约束。 ● t：触发器约束。 ● x：互斥约束。 ● f：外键约束。 ● s：聚簇约束。 ● i：无效约束。
condeferrable	boolean	这个约束是否可以推迟。 ● true：表示可以。 ● false：表示不可以。
condeferred	boolean	缺省时这个约束是否可以推迟。 ● true：表示可以。 ● false：表示不可以。
convalidated	boolean	约束是否有效。目前，只有外键和CHECK约束可将其设置为false。 ● true：表示有效。 ● false：表示无效。
conrelid	oid	这个约束所在的表，如果不是表约束则为0。
contypid	oid	这个约束所在的域，如果不是一个域约束则为0。
conindid	oid	与约束关联的索引ID。

名称	类型	描述
confrelid	oid	如果是外键，则为参考的表，否则为0。
confupdtype	"char"	外键更新动作代码。 ● a：没动作。 ● r：限制。 ● c：级联。 ● n：设置为null。 ● d：设置为缺省。
confdeltype	"char"	外键删除动作代码。 ● a：没动作。 ● r：限制。 ● c：级联。 ● n：设置为null。 ● d：设置为缺省。
confmatchtype	"char"	外键匹配类型。 ● f：全部。 ● p：部分。 ● u：未指定（在f的基础上允许匹配NULL值）。
conislocal	boolean	是否是为关系创建的本地约束。 ● true：表示是。 ● false：表示不是。
coninhcount	integer	约束直接继承父表的数目。继承父表数非零时，不能删除或重命名该约束。
connoinherit	boolean	是否可以被继承。 ● true：表示可以。 ● false：表示不可以。
consoft	boolean	是否为信息约束（Informational Constraint）。 ● true：表示是。 ● false：表示不是。
conopt	boolean	是否使用信息约束优化执行计划。 ● true：表示使用。 ● false：表示不使用。
conkey	smallint[]	如果是表约束，则是约束控制的字段列表。
confkey	smallint[]	如果是一个外键，是参考的字段的列表。

名称	类型	描述
conpfeqop	oid[]	如果是一个外键，是做PK=FK比较的相等操作符ID的列表。
conppeqop	oid[]	如果是一个外键，是做PK=PK比较的相等操作符ID的列表。
conffeqop	oid[]	如果是一个外键，是做FK=FK比较的相等操作符ID的列表。由于当前不支持外键，所以值为空。
conexclop	oid[]	如果是一个排他约束，是列的排他操作符ID列表。
conbin	pg_node_tree	如果是检查约束，那就是其表达式的内部形式。
consrc	text	如果是检查约束，则是表达式的可读形式。
conincluding	smallint[]	不用做约束，但是会包含在INDEX中的属性列。

须知

- consrc在被引用的对象改变之后不会被更新，它不会跟踪字段的名称修改。与其依赖这个字段，还是使用pg_get_constraintdef()来抽取一个检查约束的定义。
- 13.2.53 PG_CLASS的relchecks需要和在此表上为给定关系找到的检查约束的数目一致。

12.2.48 PG_CONVERSION

PG_CONVERSION系统表描述编码转换信息。

表 12-49 PG_CONVERSION 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
conname	name	-	转换名称（在一个名称空间里是唯一的）。
connamespace	oid	PG_NAMESPACE.oid	包含这个转换的名称空间的OID。
conowner	oid	PG_AUTHID.oid	编码转换的属主。
conforencoding	integer	-	源编码ID。
contoencoding	integer	-	目的编码ID。

名称	类型	引用	描述
conproc	regproc	PG_PROC.pronam e	转换过程。
condefault	boolean	-	如果这是缺省转换则为真，否则为假。

12.2.49 PG_DATABASE

PG_DATABASE系统表存储关于可用数据库的信息。

表 12-50 PG_DATABASE 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
datname	name	数据库名称。
datdba	oid	数据库所有人，通常为其创建者。
encoding	integer	数据库的字符编码方式。
datcollate	name	数据库使用的排序顺序。
datctype	name	数据库使用的字符分类。
datistemplate	boolean	是否允许作为模板数据库。
datallowconn	boolean	如果为真，表示用户可以连接到这个数据库。如果为假，则没有用户可以连接到这个数据库。这个字段用于保护template0数据库不被更改。
datconnlimit	integer	该数据库上允许的最大并发连接数，-1表示无限制。
datlastsysoid	oid	数据库里最后一个系统OID 。
datfrozenxid	xid32	用于跟踪该数据库是否需要为了防止事务ID重叠而进行清理。当前版本该字段已经废弃使用，为保持前向兼容，保留此字段，新增datfrozenxid64用于记录此信息。
dattablespace	oid	数据库的缺省表空间。
datcompatibility	name	数据库兼容模式，当前支持四种兼容模式：A、B、C、PG，分别表示兼容O、MY、TD和POSTGRES。
datacl	aclitem[]	访问权限。
datfrozenxid64	xid	用于跟踪该数据库是否需要为了防止事务ID重叠而进行清理。

名称	类型	描述
datminmxid	xid	该数据库中所有在这个之前的多事务ID已经被一个事务ID替换。这用于跟踪该数据库是否需要为了防止事务ID重叠或者允许收缩pg_clog而进行清理。它是此数据库中所有表的13.2.53 PG_CLASS中relminmxid的最小值。
dattimezone	name	数据库时区信息，默认为PRC时区。

12.2.50 PG_DB_ROLE_SETTING

PG_DB_ROLE_SETTING系统表存储数据库运行时每个角色与数据绑定的配置项的默认值 。

表 12-51 PG_DB_ROLE_SETTING 字段

名称	类型	描述
setdatabase	oid	配置项所对应的数据库，如果未指定数据库，则为0。
setrole	oid	配置项所对应的角色，如果未指定角色，则为0。
setconfig	text[]	运行时配置项的默认值。配置请联系管理员处理。

12.2.51 PG_DEFAULT_ACL

PG_DEFAULT_ACL系统表存储为新建对象设置的初始权限。

表 12-52 PG_DEFAULT_ACL 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
defaclrole	oid	与此权限相关的角色ID。
defaclnamespace	oid	与此权限相关的名称空间，如果没有，则为0。

名称	类型	描述
defaclobjtype	"char"	此权限的对象类型。 • r表示表或视图。 • S表示序列。 • f表示函数。 • T表示类型。 • K表示客户端主密钥。 • k表示列加密密钥。
defaclacl	aclitem[]	创建该类型时所拥有的访问权限。

12.2.52 PG_DEPEND

PG_DEPEND系统表记录数据库对象之间的依赖关系。这个信息允许DROP命令找出哪些其它对象必须由DROP CASCADE删除，或者是在DROP RESTRICT的情况下避免删除。

这个表的功能类似**PG_SHDEPEND**，用于记录那些在数据库之间共享的对象之间的依赖性关系。

表 12-53 PG_DEPEND 字段

名称	类型	引用	描述
classid	oid	**PG_CLASS**.oid	有依赖对象所在系统表的OID。
objid	oid	任意OID属性	指定的依赖对象的OID。
objsubid	integer	-	对于表字段，这个是该属性的字段数（objid和classid引用表本身）。对于所有其它对象类型，目前这个字段是0。
refclassid	oid	**PG_CLASS**.oid	被引用对象所在的系统表的OID。
refobjid	oid	任意OID属性	指定的被引用对象的OID。
refobjsubid	integer	-	对于表字段，这个是该字段的字段号（refobjid和refclassid引用表本身）。对于所有其它对象类型，目前这个字段是0。
deptype	"char"	-	一个定义这个依赖关系特定语义的代码。

在所有情况下，一个PG_DEPEND记录表示被引用的对象不能在有依赖的对象被删除前删除。不过，这里还有几种由deptype定义的情况：

- DEPENDENCY_NORMAL (n)：独立创建的对象之间的一般关系。有依赖的对象可以在不影响被引用对象的情况下删除。被引用对象只有在声明了CASCADE的情

况下删除，这时有依赖的对象也被删除。例子：一个表字段对其数据类型有一般依赖关系。

- DEPENDENCY_AUTO (a)：有依赖对象可以和被引用对象分别删除，并且如果删除了被引用对象则应该被自动删除（不管是RESTRICT或CASCADE模式）。例子：一个表上面的命名约束是在该表上的自动依赖关系，因此如果删除了表，它也会被删除。
- DEPENDENCY_INTERNAL (i)：有依赖的对象是作为被引用对象的一部分创建的，实际上只是它的内部实现的一部分。 DROP有依赖对象是不能直接允许的（将告诉用户发出一条删除被引用对象的DROP）。一个对被引用对象的DROP将传播到有依赖对象，不管是否声明了CASCADE。
- DEPENDENCY_EXTENSION (e)：依赖对象是被依赖对象extension的一个成员（请参见**PG_EXTENSION**）。依赖对象只可以通过在被依赖对象上DROP EXTENSION删除。函数上这个依赖类型和内部依赖一样动作，但是它为了清晰和简化gs_dump保持分开。

须知

扩展功能为内部使用功能，不建议用户使用。

- DEPENDENCY_PIN (p)：没有依赖对象；这种类型的记录标志着系统本身依赖于被引用对象，因此这个对象绝不能被删除。这种类型的记录只有在initdb的时候创建。有依赖对象的字段里是零。

12.2.53 PG_DESCRIPTION

PG_DESCRIPTION系统表可以给每个数据库对象存储一个可选的描述（注释）。许多内置的系统对象的描述提供了PG_DESCRIPTION的初始内容。

这个表的功能类似**PG_SHDESCRIPTION**，用于记录整个数据库范围内共享对象的注释。

表 12-54 PG_DESCRIPTION 字段

名称	类型	引用	描述
objoid	oid	任意OID属性	这条描述所描述的对象的OID。
classoid	oid	**PG_CLASS**.oid	这个对象出现的系统表的OID。
objsubid	integer	-	对于一个表字段的注释，它是字段号（objoid和classoid指向表自身）。对于其它对象类型，它是零。
description	text	-	对该对象描述的任意文本。

12.2.54 PG_DIRECTORY

PG_DIRECTORY系统表用于保存用户添加的directory对象，可以通过CREATE DIRECTORY语句向该表中添加记录，当enable_access_server_directory=off时，只允

许初始用户创建directory对象；当enable_access_server_directory=on时，具有SYSADMIN权限的用户和继承了内置角色gs_role_directory_create权限的用户可以创建directory对象。普通用户需要授权才能访问该表。

表 12-55 PG_DIRECTORY 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
dirname	name	目录对象的名称。
owner	oid	目录对象的所有者。
dirpath	text	目录路径。
diracl	aclitem[]	访问权限。

12.2.55 PG_ENUM

PG_ENUM系统表包含显示每个枚举类型值和标签的记录。给定枚举类型的内部表示实际上是PG_ENUM里面相关行的OID。

表 12-56 PG_ENUM 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
enumtypid	oid	PG_TYPE.oid	拥有这个枚举值的13.2.108 PG_TYPE记录的OID。
enumsortorder	real	-	这个枚举值在它的枚举类型中的排序位置。
enumlabel	name	-	这个枚举值的文本标签。

PG_ENUM行的OID跟着一个特殊规则：偶数的OID保证用和它们的枚举类型一样的排序顺序排序。也就是，如果两个偶数OID属于相同的枚举类型，那么较小的OID必须有较小enumsortorder值。奇数OID需要毫无关系的排序顺序。这个规则允许枚举比较例程在许多常见情况下避开目录查找。创建和修改枚举类型的例程只要可能就尝试分配偶数OID给枚举值。

当创建了一个枚举类型时，它的成员赋予了排序顺序位置1到n。但是随后添加的成员可能会分配enumsortorder的负值或分数值。对这些值的唯一要求是它们要正确的排序和在每个枚举类型中唯一。

12.2.56 PG_EXTENSION

PG_EXTENSION系统表存储关于所安装扩展的信息。GaussDB默认扩展是plpgsql、DIST_FDW、FILE_FDW、LOG_FDW、DBLINK_FDW、PACKAGES、

SECURITY_PLUGIN、GSSTAT_PLUGIN、PKG_DBE_RAW、PKG_DBE_OUTPUT、PKG_DBE_UTILITY和PKG_DBE_XML。该系统表为内部使用，不建议用户使用。

表 12-57 PG_EXTENSION

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
extname	name	扩展名。
extowner	oid	扩展的所有者。
extnamespace	oid	扩展导出对象的名称空间。
extrelocatable	boolean	标识此扩展是否可迁移到其他名称空间，true表示允许。
extversion	text	扩展的版本号。
extconfig	oid[]	扩展的配置信息。
extcondition	text[]	扩展配置信息的过滤条件。

12.2.57 PG_FOREIGN_DATA_WRAPPER

PG_FOREIGN_DATA_WRAPPER系统表存储外部数据封装器定义。一个外部数据封装器是在外部服务器上驻留外部数据的机制，是可以访问的。

表 12-58 PG_FOREIGN_DATA_WRAPPER 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
fdwname	name	-	外部数据封装器名。
fdwowner	oid	PG_AUTHID.oid	外部数据封装器的所有者。
fdwhandler	oid	PG_PROC.oid	引用一个负责为外部数据封装器提供扩展例程的处理函数。如果没有提供处理函数则为零。
fdwvalidator	oid	PG_PROC.oid	引用一个验证器函数，这个验证器函数负责验证给予外部数据封装器的选项、 外部服务器选项和使用外部数据封装器的用户映射的有效性。 如果没有提供验证器函数则为零。
fdwacl	aclitem[]	-	访问权限。

名称	类型	引用	描述
fdwoptions	text[]	-	外部数据封装器指定选项，使用“keyword=value”格式的字符串。

12.2.58 PG_FOREIGN_SERVER

PG_FOREIGN_SERVER系统表存储外部服务器定义。一个外部服务器描述了一个外部数据源，例如一个远程服务器。外部服务器通过外部数据封装器访问。

表 12-59 PG_FOREIGN_SERVER 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
srvname	name	-	外部服务器名。
srvowner	oid	PG_AUTHID.oid	外部服务器的所有者。
srvfdw	oid	PG_FOREIGN_DATA_WRAPPER.oid	这个外部服务器的外部数据封装器的OID。
srvtype	text	-	服务器的类型（可选）。
srvversion	text	-	服务器的版本（可选）。
srvacl	aclitem[]	-	访问权限。
srvoptions	text[]	-	外部服务器指定选项，使用“keyword=value”格式的字符串。

12.2.59 PG_HASHBUCKET

PG_HASHBUCKET系统表存储hash bucket信息。

表 12-60 PG_HASHBUCKET 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
bucketid	oid	对bucketvector计算的hash值，通过hash值可以加速对bucketvector的查找。
bucketcnt	integer	包含分片的个数。
bucketmapsize	integer	所有DN上包含的分片总数。

名称	类型	描述
bucketref	integer	预留字段，默认值为1。
bucketvector	oidvector_extend	记录此行bucket信息包含的所有bucket的id，在此列上建立唯一索引，具有相同bucketid信息的表共享同一行pg_hashbucket数据。

12.2.60 PG_INDEX

PG_INDEX系统表存储索引的一部分信息，其他的信息大多数在PG_CLASS中。

表 12-61 PG_INDEX 字段

名称	类型	描述
indexrelid	oid	这个索引在13.2.53 PG_CLASS里的记录的OID。
indrelid	oid	使用这个索引的表在13.2.53 PG_CLASS里的记录的OID。
indnatts	smallint	索引中的字段数目。
indisunique	boolean	是否为唯一索引。 • true：是个唯一索引。 • false：不是唯一索引。
indisprimary	boolean	该索引是否为该表的主键。 • true：该索引是该表的主键。这个字段为真的时候indisunique总是为真。 • false：该索引不是该表的主键。
indisexclusion	boolean	该索引是否支持排他约束。 • true：该索引支持排他约束。 • false：该索引不支持排他约束。
indimmediate	boolean	插入数据时是否立即进行唯一性检查。 • true：在插入数据时会立即进行唯一性检查。 • false：在插入数据时不会进行唯一性检查。
indisclustered	boolean	该表是否在这个索引上建簇。 • true：该表在这个索引上建簇。 • false：该表没有在这个索引上建簇。
indisusable	boolean	该索引对insert/select是否可用。 • true：该索引对insert/select可用。 • false：该索引对insert/select不可用。

名称	类型	描述
indisvalid	boolean	• true：该索引可以用于查询。 • false：该索引可能不完整，仍然必须在INSERT/UPDATE操作时进行更新，不过不能安全的用于查询。如果是唯一索引，则唯一属性也将不为真。
indcheckxmin	boolean	• true：查询不能使用索引，直到pg_index此行的xmin低于其快照的TransactionXmin，因为该表可能包含它们能看到的不兼容行断开的HOT链。 • false：查询可以用索引。
indisready	boolean	• true：此索引对插入数据是可用的。 • false：在插入或修改数据时忽略此索引。
indkey	int2vector	这是一个包含indnatts值的数组，这些数组值表示这个索引所建立的表字段。比如一个值为1 3的意思是第一个字段和第三个字段组成这个索引键字。这个数组里的零表明对应的索引属性是在这个表字段上的一个表达式，而不是一个简单的字段引用。
indcollation	oidvector	索引各列对应的排序规则的OID，详情请参考PG_COLLATION。
indclass	oidvector	对于索引键字里面的每个字段，这个字段都包含一个指向所使用的操作符类的OID，详情请参考PG_OPCLASS。
indoption	int2vector	存储列前标识位，该标识位是由索引的访问方法定义。
indexprs	pg_node_tree	表达式树（以nodeToString()形式表现）用于那些非简单字段引用的索引属性。它是一个列表，个数与indkey中的零值个数相同。如果所有索引属性都是简单的引用，则为空。
indpred	pg_node_tree	部分索引断言的表达式树（以nodeToString()的形式表现）。如果不是部分索引，则是空字符串。
indisreplident	boolean	此索引的列是否为逻辑解码的解码列。 • true：此索引的列成为逻辑解码的解码列。 • false：此索引的列不是逻辑解码的解码列。
indnkeyatts	smallint	索引中的总字段数，超出indnatts的部分不参与索引查询。

12.2.61 PG_INHERITS

PG_INHERITS系统表记录关于表继承层次的信息。数据库里每个直接的子系表都有一条记录。间接的继承可以通过追溯记录链来判断。

表 12-62 PG_INHERITS 字段

名称	类型	引用	描述
inhrelid	oid	PG_CLASS.oid	子表的OID。
inhparent	oid	PG_CLASS.oid	父表的OID。
inhseqno	integer	-	如果一个子表存在多个直系父表（多重继承），这个数字表明此继承字段的排列顺序。计数从1开始。

12.2.62 PG_JOB

PG_JOB系统表存储用户创建的定时任务的任务详细信息，定时任务线程定时轮询PG_JOB系统表中的时间，当任务到期会触发任务的执行，并更新PG_JOB表中的任务状态。该系统表属于Shared Relation，所有创建的job记录对所有数据库可见。普通用户需授权才能访问。

表 12-63 PG_JOB 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
job_id	bigint	作业ID，主键，是唯一的（有唯一索引）
current_postgres_pid	bigint	如果当前任务已被执行，那么此处记录运行此任务的线程ID。默认为 -1，表示此任务未被执行过。
log_user	name	创建者的UserName
priv_user	name	作业执行者的UserName
dbname	name	标识作业要在哪个数据库执行的数据库名称
node_name	name	标识当前作业是在哪个数据库主节点上创建和执行

名称	类型	描述
job_status	"char"	当前任务的执行状态，默认为's'，各取值含义： ● 'r'：running ● 's'：successfully finished ● 'f'：job failed ● 'd'：disable 当job连续执行失败16次，会将job_status自动设置为失效状态'd'，后续不再执行该job。 注：当用户将定时任务关闭（即：guc参数job_queue_processes为0时），由于监控job执行的线程不会启动，所以该状态不会根据job的实时状态进行设置，用户不需要关注此状态。只有当开启定时任务功能（即：guc参数job_queue_processes为非0时），系统才会根据当前job的实时状态刷新该字段值。
start_date	timestamp without time zone	作业第一次开始执行时间，时间精确到毫秒。
next_run_date	timestamp without time zone	定时任务下次执行的时间，时间精确到毫秒。
failure_count	smallint	失败计数，作业连续执行失败16次，不再继续执行。
interval	text	作业执行的重复时间间隔。
last_start_date	timestamp without time zone	上次运行开始时间，时间精确到毫秒。
last_end_date	timestamp without time zone	上次运行的结束时间，时间精确到毫秒。
last_suc_date	timestamp without time zone	上次成功运行的开始时间，时间精确到毫秒。
this_run_date	timestamp without time zone	正在运行任务的开始时间，时间精确到毫秒。
nspname	name	标识作业执行时的schema的名称。
job_name	text	DBE_SCHEDULER定时任务专用，定时任务名称。
end_date	timestamp without time zone	DBE_SCHEDULER定时任务专用，定时任务失效时间，时间精确到毫秒。

名称	类型	描述
enable	boolean	DBE_SCHEDULER定时任务专用，定时任务启用状态： ● true：启用 ● false：未启用
failure_msg	text	最新一次执行任务报错信息。

12.2.63 PG_JOB_PROC

PG_JOB_PROC系统表对应13.2.71 PG_JOB表中每个任务的作业内容（包括：PL/SQL代码块、匿名块）。将存储过程信息独立出来，如果放到PG_JOB中，被加载到共享内存的时候，会占用不必要的空间，所以在使用的时候再进行查询获取。普通用户需授权才能访问。

表 12-64 PG_JOB_PROC 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
job_id	integer	外键，关联13.2.71 PG_JOB中的job_id。
what	text	作业内容，DBE_SCHEDULER定时任务中的程序内容。
job_name	text	DBE_SCHEDULER定时任务专用，定时任务或程序名称。

12.2.64 PG_LANGUAGE

PG_LANGUAGE系统表登记编程语言，用户可以用这些语言或接口写函数或者存储过程。

表 12-65 PG_LANGUAGE 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性；必须明确选择）。
lanname	name	-	语言的名称。
lanowner	oid	PG_AUTHID.oid	语言的所有者。

名称	类型	引用	描述
lanispl	boolean	-	对于内部语言而言是假（比如SQL），对于用户定义的语言则是真。目前，gs_dump仍然使用该字段判断哪种语言需要转储，但是这些可能在将来被其它机制取代。
lanpltrusted	boolean	-	如果这是可信语言则为真，意味着系统相信它不会被授予任何正常SQL执行环境之外的权限。不可信语言为假，只有初始用户可以用不可信的语言创建函数。
lanplcallfoid	oid	PG_PROC.oid	对于非内部语言，这是指向该语言处理器的引用，语言处理器是一个特殊函数， 负责执行以某种语言写的所有函数。
laninline	oid	PG_PROC.oid	这个字段引用一个负责执行“inline”匿名代码块的函数（DO块）。如果不支持内联块则为零。
lanvalidator	oid	PG_PROC.oid	这个字段引用一个语言校验器函数，它负责检查新创建的函数的语法和有效性。如果没有提供校验器，则为零。
lanacl	aclitem[]	-	访问权限。

12.2.65 PG_LARGEOBJECT

PG_LARGEOBJECT系统表保存那些标记着“大对象”的数据。一个大对象是使用其创建时分配的OID标识的。每个大对象都分解成足够小的小段或者“页面”以便以行的形式存储在PG_LARGEOBJECT里。每页的数据定义为LOBLKSIZE。

需要有系统管理员权限才可以访问此系统表。

表 12-66 PG_LARGEOBJECT 字段

名称	类型	引用	描述
loid	oid	PG_LARGEOBJECT_METADATA.oid	包含本页的大对象的标识符。
pageno	integer	-	本页在其大对象数据中的页码从零开始计算。
data	bytea	-	存储在大对象中的实际数据。这些数据绝不会超过LOBLKSIZE字节，而且可能更少。

PG_LARGEOBJECT的每一行保存一个大对象的一个页面，从该对象内部的字节偏移（pageno * LOBLKSIZE）开始。这种实现允许松散的存储：页面可以丢失，而且可以

比LOBLKSIZE字节少（即使它们不是对象的最后一页）。大对象内丢失的部分读做零。

12.2.66 PG_LARGEOBJECT_METADATA

PG_LARGEOBJECT_METADATA系统表存储与大数据相关的元数据。实际的大对象数据存储在PG_LARGEOBJECT里。

表 12-67 PG_LARGEOBJECT_METADATA 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
lomowner	oid	PG_AUTHID.oid	大对象的所有者。
lomacl	aclitem[]	-	访问权限。

12.2.67 PG_NAMESPACE

PG_NAMESPACE系统表存储名称空间，即存储schema相关的信息。

表 12-68 PG_NAMESPACE 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
nspname	name	名称空间的名称。
nspowner	oid	名称空间的所有者。
nsptimeline	bigint	在数据库节点上创建此命名空间时的时间线。此字段为内部使用，仅在数据库节点上有效。
nspacl	aclitem[]	访问权限。
in_redistribution	"char"	是否处于重发布状态。
nspcollation	oid	名称空间的默认字符序（仅在sql_compatibility='b'时可能有值）。

12.2.68 PG_OBJECT

PG_OBJECT系统表存储限定类型对象（普通表，索引，序列，视图，存储过程和函数）的创建用户、创建时间和最后修改时间。

表 12-69 PG_OBJECT 字段

名称	类型	描述
object_oid	oid	对象标识符。
object_type	"char"	对象类型： ● r 表示普通表 ● i 表示索引 ● s 表示序列 ● v 表示视图 ● P 表示存储过程和函数 ● S 表示包头 ● B 表示包体
creator	oid	对象的所有者。
ctime	timestamp with time zone	对象的创建时间。
mtime	timestamp with time zone	对象的最后修改时间，修改行为包括ALTER操作和GRANT、REVOKE操作。
createcsn	bigint	对象创建时的CSN。
changecsn	bigint	对表或索引执行DDL操作时的CSN。
valid	boolean	对象的有效性，t为有效，f为无效。

须知

- 无法记录初始化数据库（initdb）过程中所创建或修改的对象，即PG_OBJECT无法查询到该对象记录。
- 对于升级前创建的对象，再次修改时会记录其修改时间（mtime），对表或索引执行DDL操作时会记录其所属事务的事务提交序列号（changecsn）。由于无法得知该对象创建时间，因此ctime和createcsn为空。
- ctime和mtime所记录的时间为用户当次操作所属事务的起始时间。
- 由扩容引起的对象修改时间也会被记录。
- createcsn和changecsn记录的是用户当次操作所属事务的事务提交序列号。
- enable_gtt_concurrent_truncate开启时，truncate全局临时表不会刷新mtime字段。
- 对象创建语句时存在未定义的对象，或所依赖的对象有修改或删除动作时，对象会为无效状态。

12.2.69 PG_OPCLASS

PG_OPCLASS系统表定义索引访问方法操作符类。

每个操作符类为一种特定数据类型和一种特定索引访问方法定义索引字段的语义。一个操作符类本质上指定一个特定的操作符族适用于一个特定的可索引的字段数据类型。索引的字段实际可用的族中的操作符集是接受字段的数据类型作为它们的左边的输入的那个。

表 12-70 PG_OPCLASS 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
opcmethod	oid	PG_AM.oid	操作符类所服务的索引访问方法。
opcname	name	-	这个操作符类的名称。
opcnamespace	oid	PG_NAMESPACE.oid	这个操作符类的名称空间。
opcowner	oid	PG_AUTHID.oid	操作符类属主。
opcfamily	oid	PG_OPFAMILY.oid	包含该操作符类的操作符族。
opcintype	oid	PG_TYPE.oid	操作符类索引的数据类型。
opcdefault	boolean	-	如果操作符类是opcintype的缺省，则为真。
opckeytype	oid	PG_TYPE.oid	索引数据的类型，如果和opcintype相同则为零。

一个操作符类的opcmethod必须匹配包含它的操作符族的opfmethod。

12.2.70 PG_OPERATOR

PG_OPERATOR系统表存储有关操作符的信息。

表 12-71 PG_OPERATOR 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
oprname	name	-	操作符的名称。
oprnamespace	oid	PG_NAMESPACE.oid	包含此操作符的名称空间的OID。
oprowner	oid	PG_AUTHID.oid	操作符所有者。

名称	类型	引用	描述
oprkind	"char"	-	• b：表示中缀(“两边”)。 • l：表示前缀(“左边”)。 • r：表示后缀(“右边”)。
oprcanmerge	boolean	-	这个操作符是否支持合并连接。 • t（true）：表示支持合并连接。 • f（false）：表示不支持合并连接。
oprcanhash	boolean	-	这个操作符是否支持Hash连接。 • t（true）：表示支持Hash连接。 • f（false）：表示不支持Hash连接。
oprleft	oid	PG_TYPE.oid	左操作数的类型。
oprright	oid	PG_TYPE.oid	右操作数的类型。
oprresult	oid	PG_TYPE.oid	结果类型。
oprcom	oid	PG_OPERATOR.oid	• 若存在，值为此操作符的交换符。 • 若不存在，值为0。
oprnegate	oid	PG_OPERATOR.oid	• 若存在，值为此操作符的反转器。 • 若不存在，值为0。
oprcode	regproc	PG_PROC.proname	实现这个操作符的函数。
oprrest	regproc	PG_PROC.proname	此操作符的约束选择性计算函数。
oprjoin	regproc	PG_PROC.proname	此操作符的连接选择性计算函数。

12.2.71 PG_OPFAMILY

PG_OPFAMILY系统表定义操作符族。

每个操作符族是一个操作符和相关支持例程的集合，其中的例程实现为一个特定的索引访问方式指定的语义。另外，族中的操作符都是“兼容的”，通过由访问方式指定的方法。操作符族的概念允许交叉数据类型操作符和索引一起使用，并且合理的使用访问方式的语义的知识。

表 12-72 PG_OPFAMILY 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
opfmethod	oid	PG_AM.oid	操作符族使用的索引方法。
opfname	name	-	这个操作符族的名称。
opfnamespace	oid	PG_NAMESPACE.oid	这个操作符的名称空间。
opfowner	oid	PG_AUTHID.oid	操作符族的所有者。

定义一个操作符族的大多数信息不在它的PG_OPFAMILY行里面，而是在相关的行PG_AMOP，PG_AMPROC和PG_OPCLASS里。

12.2.72 PG_PARTITION

PG_PARTITION系统表存储数据库内所有分区表（partitioned table）、分区（table partition）和分区索引（index partition）四类对象的信息。分区表索引（partitioned index）的信息不在PG_PARTITION系统表中保存。由于分区表（partitioned table）没有实际的物理文件，所以在pg_partition中不会记录其relfilenode，relpages，reltuples，reltoastrelid，reltoastidxid等信息。

表 12-73 PG_PARTITION 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
relname	name	分区表、分区、分区上toast表和分区索引的名称。
parttype	"char"	对象类型： • 'r'：partitioned table • 'p'：table partition • 's'：table subpartition • 'x'：index partition
parentid	oid	• 当对象为分区表或分区时，此字段表示分区表在PG_CLASS中的OID。 • 当对象为二级分区时，此字段标识其一级分区在PG_PARTITION中的OID。 • 当对象为index partition时，此字段表示所属分区表索引（partitioned index）的OID。
rangenum	integer	保留字段。
intervalnum	integer	保留字段。

名称	类型	描述
partstrategy	"char"	分区表分区策略，现在仅支持： ● 'r'：范围分区。 ● 'v'：数值分区。 ● 'i'：间隔分区。 ● 'l'：list分区。 ● 'h'：hash分区。 ● 'n'：无效分区。
relfilenode	oid	table partition、index partition、分区上toast表的物理存储位置。
reltablespace	oid	table partition、index partition、分区上toast表所属表空间的OID。
relpages	double precision	统计信息：table partition、index partition的数据页数量。
reltuples	double precision	统计信息：table partition、index partition的元组数。
relallvisible	integer	统计信息：table partition、index partition的可见数据页数。
reltoastrelid	oid	table partition所对应toast表的OID。
reltoastidxid	oid	table partition所对应toast表的索引的OID。
indextblid	oid	index partition对应table partition的OID。
indisusable	boolean	分区索引是否可用。
relfrozenxid	xid32	冻结事务ID号。 为保持前向兼容，保留此字段，新增relfrozenxid64用于记录此信息。
intspnum	integer	间隔分区所属表空间的个数。
partkey	int2vector	分区键的列号。
intervaltablespace	oidvector	间隔分区所属的表空间，间隔分区以round-robin方式落在这些表空间内。
interval	text[]	间隔分区的间隔值。
boundaries	text[]	范围分区和间隔分区的上边界。
transit	text[]	间隔分区的跳转点。
reloptions	text[]	设置partition的存储属性，与pg_class.reloptions的形态一样，用“keyword=value”格式的字符串来表示，目前用于在线扩容的信息搜集。

名称	类型	描述
relfrozenxid64	xid	冻结事务ID号。
relminmxid	xid	冻结多事务ID号。
partitionno	integer	用于维护分区表中的分区Map结构。 ● 当对象为分区时，此字段表示分区ID，从1开始自增。 ● 当对象为分区表时，此字段表示分区ID的最大值，并使用负值来特殊标记，该值会随着部分分区DDL语法不断递增。 ● 当对象为其他类型时，此字段为空值，没有任何含义。 partitionno是一个永久自增列，可以通过语法ALTER TABLE t_name RESET PARTITION或者VACUUM FULL命令重置/回收。
subpartitionno	integer	用于维护分区表中的二级分区Map结构。 ● 当对象为二级分区时，此字段表示二级分区ID，从1开始自增。 ● 当对象为二级分区表的一级分区时，此字段表示二级分区ID的最大值，并使用负值来特殊标记，该值会随着部分分区DDL语法不断递增。 ● 当对象为其他类型时，此字段为空值，没有任何含义。 subpartitionno是一个永久自增列，可以通过语法ALTER TABLE t_name RESET PARTITION或者VACUUM FULL命令重置/回收。

12.2.73 PG_PLTEMPLATE

PG_PLTEMPLATE系统表存储过程语言的“模板”信息。

表 12-74 PG_PLTEMPLATE 字段

名称	类型	描述
tmplname	name	这个模板所应用的语言的名称。
tmpltrusted	boolean	如果语言被认为是可信的，则为真。否则为假。
tmpldbacreate	boolean	如果语言是由数据库所有者创建的，则为真。否则为假。
tmplhandler	text	调用处理器函数的名称。
tmplinline	text	匿名块处理器的名称，若没有则为NULL。

名称	类型	描述
tmplvalidator	text	校验函数的名称，如果没有则为NULL。
tmpllibrary	text	实现语言的共享库的路径。
tmplacl	aclitem[]	模板的访问权限（未使用）。

12.2.74 PG_PROC

PG_PROC系统表存储函数或过程的信息。

表 12-75 PG_PROC 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
proname	name	函数名称。
pronamespace	oid	包含该函数名称空间的OID。
proowner	oid	函数的所有者。
prolang	oid	这个函数的实现语言或调用接口。
procost	real	估算的执行成本。
prorows	real	估算的影响行的数目。
provariadic	oid	参数元素的数据类型。
protransform	regproc	此函数的简化调用方式。
proisagg	boolean	函数是否是聚集函数。 ● t（true）：表示是。 ● f（false）：表示不是。
proiswindow	boolean	函数是否是窗口函数。 ● t（true）：表示是。 ● f（false）：表示不是。
prosecdef	boolean	函数是否是一个安全定义器（也就是一个“setuid”函数）。 ● t（true）：表示是。 ● f（false）：表示不是。
proleakproof	boolean	函数是否没有副作用。如果函数没有对参数进行防泄露处理，则会抛出错误。 ● t（true）：表示没副作用。 ● f（false）：表示有副作用。

名称	类型	描述
proisstrict	boolean	● t（true）：使用该函数时，如果入参有空值，则函数实际上不调用直接返回空。 ● f（false）：使用该函数时，即使入参有空值，也要调用。所以该函数必须处理空输入。
proretset	boolean	函数返回值是否为一个集合（也就是说，指定数据类型的多个数值）。 ● true：函数返回值是一个集合。 ● false：函数返回值不是一个集合。
provolatile	"char"	该函数的结果是否只依赖于它的输入参数，或者还会被外界因素影响。 ● i：“不可变的”（immutable）函数，对于相同的输入总是产生相同的结果。 ● s：稳定的”（stable）函数，（对于固定输入）其结果在一次扫描里不变。 ● v：“易变”（volatile）函数，其结果可能在任何时候变化v也用于那些有副作用的函数，因此调用它们无法得到优化。
pronargs	smallint	参数数目。
pronargdefaults	smallint	有默认值的参数数目。
prorettype	oid	返回值的数据类型。
proargtypes	oidvector	一个存放函数参数的数据类型的数组。数组里只包括输入参数（包括INOUT参数）此代表该函数的调用签名（接口）。
proallargtypes	oid[]	一个包含函数参数的数据类型的数组。数组里包括所有参数的类型（包括OUT和INOUT参数），如果所有参数都是IN参数，则这个字段就会是空。请注意数组下标是以1为起点的，而因为历史原因，proargtypes的下标起点为0。
proargmodes	"char"[]	一个保存函数参数模式的数组，编码如下： ● i表示IN参数。 ● o表示OUT参数。 ● b表示INOUT参数。 ● v表示VARIADIC参数。 如果所有参数都是IN参数，则这个字段为空。请注意，下标对应的是proallargtypes的位置，而不是proargtypes。
proargnames	text[]	一个保存函数参数的名称的数组。没有名称的参数在数组里设置为空字符串。如果没有一个参数有名称，这个字段将是空。请注意，此数组的下标对应proallargtypes而不是proargtypes。

名称	类型	描述
proargdefaults	pg_node_tree	默认值的表达式树。是PRONARGDEFAULTS元素的列表。
prosrc	text	描述函数或存储过程的定义。例如，对于解释型语言来说就是函数的源程序，或者一个连接符号，一个文件名，或者函数和存储过程创建时指定的其他任何函数体内容，具体取决于语言/调用习惯的实现。
probin	text	关于如何调用该函数的附加信息。同样，其含义也是和语言相关的。
proconfig	text[]	函数针对运行时配置变量的本地设置。
proacl	aclitem[]	访问权限。具体请参见**GRANT**和**REVOKE**。
prodefaultargpos	int2vector	函数具有默认值的入参的位置。
proshippable	boolean	表示该函数是否可以下推到数据库节点上执行，默认值是false。 ● 对于IMMUTABLE类型的函数，函数始终可以下推到数据库节点上执行。 ● 对于STABLE/VOLATILE类型的函数，仅当函数的属性是SHIPPABLE的时候，函数可以下推到数据库节点执行。
propackage	boolean	表示该函数是否支持重载，默认值是false。 ● t（true）：表示支持。 ● f（false）：表示不支持。
prokind	"char"	表示该对象为函数还是存储过程。 ● 'f'：表示该对象为函数。 ● 'p'：表示该对象为存储过程。
proargsrc	text	描述兼容A语法定义的函数或存储过程的参数输入字符串，包括参数注释。默认值为NULL。
proisprivate	boolean	描述函数是否是PACKAGE内的私有函数，默认为false。
propackageid	oid	函数所属的package oid，如果不在package内，则为0。
proargtypesext	oidvector_extend	当函数参数较多时，用来存放函数参数的数据类型的数组。数组里只包括输入参数（包括INOUT参数）,代表该函数的调用签名（接口）。
prodefaultargposext	int2vector_extend	当函数参数较多时，函数具有默认值的入参的位置。

名称	类型	描述
allargtypes	oidvector	用来存放存储过程参数的数据类型的数组，包含存储过程所有参数（入参、出参、INOUT参数）。
allargtypesext	oidvector_extend	当函数参数较多（大于666个）时，用来存放存储过程参数的数据类型的数组，包含所有参数（入参、出参、INOUT参数）。

说明

新建函数时，会向pg_proc表中插入数据，更新索引。当出入参个数很多时，索引的长度可能会超过页面的三分之一，进而可能会产生“Index row size xxx exceeds maximum xxx for index "pg_proc_proname_all_args_nsp_index"”的报错，此为预期情况。您可以通过减少参数个数，避免该报错的产生。

12.2.75 PG_RANGE

PG_RANGE系统表存储关于范围类型的信息，除了**PG_TYPE**里类型的记录。

表 12-76 PG_RANGE 字段

名称	类型	引用	描述
rngtypid	oid	**PG_TYPE**.oid	范围类型的OID。
rngsubtype	oid	**PG_TYPE**.oid	这个范围类型的元素类型（子类型）的OID。
rngcollation	oid	**PG_COLLATION**.oid	用于范围比较的排序规则的OID，如果没有则为零。
rngsubopc	oid	**PG_OPCLASS**.oid	用于范围比较的子类型的操作符类的OID。
rngcanonical	regproc	**PG_PROC**.proname	转换范围类型为规范格式的函数名，如果没有则为0。
rngsubdiff	regproc	**PG_PROC**.proname	表示用于计算范围两元素差值的函数的名字，该函数的返回值为双精度类型，如果不存在该函数则rngsubdiff字段值为0。

若元素类型是离散的，则rngcanonical决定用于范围类型的排序顺序；若元素类型是不可排序的，则rngsubopc决定用于范围类型的排序顺序；若元素类型是可排序的，则rngsubopc和rngcollation决定用于范围类型的排序顺序。

12.2.76 PG_REPLICATION_ORIGIN

PG_REPLICATION_ORIGIN系统表包含所有已创建的复制源，该表在数据库实例的所有数据库之间共享，即每个实例只有一份，而不是每个数据库一份。

表 12-77 PG_REPLICATION_ORIGIN 字段

名称	类型	描述
roident	oid	一个数据库实例范围内唯一的复制源标识符。
roname	text	外部的由用户定义的复制源名称。

12.2.77 PG_RESOURCE_POOL

PG_RESOURCE_POOL系统表提供了数据库资源池的信息。

表 12-78 PG_RESOURCE_POOL 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
respool_name	name	资源池名称。
mem_percent	integer	内存配置的百分比。
cpu_affinity	bigint	CPU绑定core的数值。
control_group	name	资源池所在的control group名称。
active_statements	integer	资源池上最大的并发数。
max_dop	integer	最大并发度。用作扩容的接口，表示数据重分布时，扫描并发度。
memory_limit	name	资源池最大的内存。
parentid	oid	父资源池OID。
io_limits	integer	每秒触发IO的次数上限。单位是万次/s。
io_priority	name	IO利用率高达90%时，重消耗IO作业进行IO资源管控时关联的优先级等级。
nodegroup	name	表示资源池所在的逻辑数据库的名称。（集中式不支持该字段）。
is_foreign	boolean	表示资源池是否用于控制逻辑数据库之外的用户。（集中式不支持该字段） • true：表示资源池用来控制不属于当前资源池的普通用户的资源。 • false：表示不控制不属于当前资源池的普通用户的资源。
max_worker	integer	只用于扩容的接口，表示扩容数据重分布时，表内插入并发度。

名称	类型	描述
max_connections	integer	最大连接数，用来限制资源池可使用的最大连接数。

注：max_dop和max_worker用于扩容，不适用于集中式。

12.2.78 PG_REWRITE

PG_REWRITE系统表存储表和视图定义的重写规则。

表 12-79 PG_REWRITE 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
rulename	name	规则名称。
ev_class	oid	使用这条规则的表名称。
ev_attr	smallint	这条规则适用的字段（目前总是为零，表示整个表）。
ev_type	"char"	规则适用的事件类型。 ● 1：SELECT ● 2：UPDATE ● 3：INSERT ● 4：DELETE
ev_enabled	"char"	用于控制复制的触发。 ● O：“origin”和“local”模式时触发。 ● D：禁用触发。 ● R：“replica”时触发。 ● A：任何模式是都会触发。
is_instead	boolean	如果该规则是INSTEAD规则，则为真，否则为假。
ev_qual	pg_node_tree	规则的资格条件的表达式树（以nodeToString（）形式存在）。
ev_action	pg_node_tree	规则动作的查询树（以nodeToString() 形式存在）。

12.2.79 PG_RLSPOLICY

PG_RLSPOLICY系统表存储行级访问控制策略。

表 12-80 PG_RLSPOLICY 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
polname	name	行级访问控制策略的名称。
polrelid	oid	行级访问控制策略作用的表对象oid。
polcmd	"char"	行级访问控制策略影响的SQL操作。
polpermissive	boolean	行级访问控制策略的属性。 ● t：表达式OR条件拼接。 ● f：表达式AND条件拼接。
polroles	oid[]	行级访问控制策略影响的用户oid列表，不指定表示影响所有的用户。
polqual	pg_node_tree	行级访问控制策略的表达式。

12.2.80 PG_SECLABEL

PG_SECLABEL系统表存储数据对象上的安全标签。

PG_SHSECLABEL的作用类似，只是它是用于在一个数据库内共享的数据库对象的安全标签上的。

表 12-81 PG_SECLABEL 字段

名称	类型	引用	描述
objoid	oid	任意OID属性	这个安全标签所属的对象的OID。
classoid	oid	PG_CLASS.oid	出现这个对象的系统目录的OID。
objsubid	integer	-	出现在这个对象中的列的序号。
provider	text	-	与这个标签相关的标签提供程序。
label	text	-	应用于这个对象的安全标签。

12.2.81 PG_SHDEPEND

PG_SHDEPEND系统表记录数据库对象和共享对象（比如角色）之间的依赖性关系。这些信息允许GaussDB保证在企图删除这些对象之前，这些对象是没有被引用的。

PG_DEPEND的作用类似，只是它是用于在一个数据库内部的对象的依赖性关系的。

PG_SHDEPEND是在数据库实例的所有数据库之间共享的，即每个实例只有一个，而不是每个数据库一个。

表 12-82 PG_SHDEPEND 字段

名称	类型	引用	描述
dbid	oid	PG_DATABASE.oid	依赖对象所在的数据库的OID，如果是共享对象，则为零。
classid	oid	PG_CLASS.oid	依赖对象所在的系统表的OID。
objid	oid	任意OID属性	指定的依赖对象的OID。
objsubid	integer	-	对于一个表字段，这是字段号（objid和classid参考表本身）。对于所有其他对象类型，这个字段为零。
refclassid	oid	PG_CLASS.oid	被引用对象所在的系统表的OID（必须是一个共享表）。
refobjid	oid	任意OID属性	指定的被引用对象的OID。
deptype	"char"	-	一段代码，定义了这个依赖性关系的特定语义；参阅下文。
objfile	text	-	用户定义函数库文件路径。

在任何情况下，一条PG_SHDEPEND记录就表明这个被引用的对象不能在未删除依赖对象的前提下删除。不过，deptype同时还标出了几种不同的子风格：

- SHARED_DEPENDENCY_OWNER (o)

  被引用的对象（必须是一个角色）是依赖对象的所有者。

- SHARED_DEPENDENCY_ACL (a)

  被引用的对象（必须是一个角色）在依赖对象的ACL（访问控制列表，也就是权限列表）里提到。SHARED_DEPENDENCY_ACL不会在对象的所有者头上添加的，因为所有者会有一个SHARED_DEPENDENCY_OWNER记录。

- SHARED_DEPENDENCY_PIN (p)

  这类记录标识系统自身依赖于该被依赖对象，因此这样的对象不能被删除。这种类型的记录只是由initdb创建。这样的依赖对象的字段都是零。

- SHARED_DEPENDENCY_ DBPRIV(d)

  被引用的对象（必须是一个角色）具有依赖对象所对应的ANY权限（指定的依赖对象的OID对应的是系统表13.2.11 GS_DB_PRIVILEGE中一行）。

12.2.82 PG_SHDESCRIPTION

PG_SHDESCRIPTION系统表为共享数据库对象存储可选的注释。可以使用COMMENT命令操作注释的内容，使用\d命令查看注释内容。

PG_DESCRIPTION提供了类似的功能，它记录了单个数据库中对象的注释。

PG_SHDESCRIPTION是在数据库实例的所有数据库之间共享的，即每个实例只有一个，而不是每个数据库一个。

表 12-83 PG_SHDESCRIPTION 字段

名称	类型	引用	描述
objoid	oid	任意OID属性	这条描述所描述的对象的OID。
classoid	oid	PG_CLASS.oid	这个对象出现的系统表的OID。
description	text	-	作为对该对象的描述的任意文本。

12.2.83 PG_SHSECLABEL

PG_SHSECLABEL系统表存储在共享数据库对象上的安全标签。安全标签可以用SECURITY LABEL命令操作。

查看安全标签的简单点的方法，请参阅PG_SECLABELS。

PG_SECLABEL的作用类似，只是它是用于在单个数据库内部的对象的安全标签的。

不同于大多数的系统表，PG_SHSECLABEL在GaussDB中的所有数据库中共享：每个GaussDB只有一个PG_SHSECLABEL，而不是每个数据库一个。

表 12-84 PG_SHSECLABEL 字段

名称	类型	引用	描述
objoid	oid	任意OID属性	这个安全标签所属的对象的OID。
classoid	oid	PG_CLASS.oid	出现这个对象的系统目录的OID。
provider	text	-	与这个标签相关的标签提供程序。
label	text	-	应用于这个对象的安全标签。

12.2.84 PG_SET

PG_SET系统表存储SET数据类型定义的元数据。

表 12-85 PG_SET 字段

名称	类型	描述
settypid	oid	SET数据类型的OID。
setnum	tinyint	SET数据类型的成员数量，最大64个成员。
setsortorder	tinyint	SET数据类型定义时成员的排序位置，从0开始编号。
setlabel	text	SET数据类型的成员名称。

12.2.85 PG_STATISTIC

PG_STATISTIC系统表存储有关该数据库中表和索引列的统计数据。默认只有系统管理员权限才可以访问此系统表，普通用户需要授权才可以访问。

表 12-86 PG_STATISTIC 字段

名称	类型	描述
starelid	oid	所描述字段所属的表或者索引。
starelkind	"char"	所属对象的类型。
staattnum	smallint	所描述字段在表中的编号，从1开始。
stainherit	boolean	是否统计有继承关系的对象。
stanullfrac	real	所描述字段中为NULL的记录的比率。
stawidth	integer	所描述字段非NULL记录的平均存储宽度，以字节计。
stadistinct	real	标识全局统计信息中数据库节点上字段里唯一的非NULL数据值的数目。 ● 一个大于零的数值是独立数值的实际数目。 ● 一个小于零的值是distinct值所占总行数的比例，比如stadistinct=-0.5时，它的实际distinct值是总行数*0.5。 ● 零值表示独立数值的数目未知。
stakindN	smallint	一个编码，表示这种类型的统计存储在pg_statistic行的第N个“槽位”。 N的取值范围：1～5
staopN	oid	一个用于生成这些存储在第N个“槽位”的统计信息的操作符。比如，一个柱面图槽位会显示<操作符，该操作符定义了该数据的排序顺序。 N的取值范围：1～5
stanumbersN	real[]	第N个“槽位”的相关类型的数值类型统计，如果该槽位和数值类型没有关系，则就是NULL。 N的取值范围：1～5
stavaluesN	anyarray	第N个“槽位”类型的字段数据值，如果该槽位类型不存储任何数据值，则就是NULL。每个数组的元素值实际上都是指定字段的数据类型，因此，除了把这些字段的类型定义成anyarray之外，没有更好的办法。 N的取值范围：1～5

名称	类型	描述
stadndistinct	real	标识DN1上字段里唯一的非NULL数据值的数目。 ● 一个大于零的数值是独立数值的实际数目。 ● 一个小于零的值是distinct值所占总行数的比例，比如stadistinct=-0.5时，它的实际distinct值是总行数*0.5。 ● 零值表示独立数值的数目未知。
staextinfo	text	统计信息的扩展信息。预留字段。

须知

PG_STATISTIC系统表存储了统计对象的一些敏感信息，如高频值MCV。系统管理员和授权后的其他用户可以通过访问PG_STATISTIC系统表查询到统计对象的这些敏感信息。

12.2.86 PG_STATISTIC_EXT

PG_STATISTIC_EXT系统表存储有关该数据库中表的扩展统计数据，包括多列统计数据和表达式统计数据（后续支持）。收集哪些扩展统计数据是由用户指定的。需要有系统管理员权限才可以访问此系统表。

表 12-87 PG_STATISTIC_EXT 字段

名称	类型	描述
starelid	oid	所描述字段所属的表或者索引。
starelkind	"char"	所属对象的类型，'c'表示表，'p'表示分区。
stainherit	boolean	是否统计有继承关系的对象。
stanullfrac	real	所描述字段中为NULL的记录的比率。
stawidth	integer	所描述字段非NULL记录的平均存储宽度，以字节计。
stadistinct	real	标识全局统计信息中数据库节点上字段里唯一的非NULL数据值的数目。 ● 一个大于零的数值是独立数值的实际数目。 ● 一个小于零的值是distinct值所占总行数的比例，比如stadistinct=-0.5时，它的实际distinct值是总行数*0.5。 ● 零值表示独立数值的数目未知。

名称	类型	描述
stadndistinct	real	标识DN1上字段里唯一的非NULL数据值的数目。 ● 一个大于零的数值是独立数值的实际数目。 ● 一个小于零的值是distinct值所占总行数的比例，比如stadistinct=-0.5时，它的实际distinct值是总行数*0.5。 ● 零值表示独立数值的数目未知。
stakindN	smallint	一个编码，表示这种类型的统计存储在pg_statistic行的第N个“槽位”。 N的取值范围：1～5
staopN	oid	一个用于生成这些存储在第N个“槽位”的统计信息的操作符。比如，一个柱面图槽位会显示<操作符，该操作符定义了该数据的排序顺序。 N的取值范围：1～5
stakey	int2vector	所描述的字段编号的数组。
stanumbersN	real[]	第N个“槽位”的相关类型的数值类型统计，如果该槽位和数值类型没有关系，则就是NULL。 N的取值范围：1～5
stavaluesN	anyarray	第N个“槽位”类型的字段数据值，如果该槽位类型不存储任何数据值，则就是NULL。每个数组的元素值实际上都是指定字段的数据类型，因此，除了把这些字段的类型定义成anyarray之外，没有更好的办法。 N的取值范围：1～5
staexprs	pg_node_tree	扩展统计信息对应的表达式。
stasource	"char"	扩展统计信息的来源： ● 'a'：表示自动创建，由GUC参数auto_statistic_ext_columns控制。 ● 'm'：表示手动创建，用户通过 analyze tablename ((column list)) 或者 alter table tablename add statistics ((column list)) 来创建。
stastatus	"char"	扩展统计信息的状态： ● 'a'：表示活跃可用。 ● 'd'：表示被禁用，相关信息不被收集，优化器在生成计划的时候也不使用。用户可以使用语法 alter table tablename disable/enable statistics((column list)) 来修改扩展统计信息的状态。

须知

PG_STATISTIC_EXT系统表存储了统计对象的一些敏感信息，如高频值MCV。系统管理员和授权后的其他用户可以通过访问PG_STATISTIC_EXT系统表查询到统计对象的这些敏感信息。

12.2.87 PG_SYNONYM

PG_SYNONYM系统表存储同义词对象名与其他数据库对象名间的映射信息。

表 12-88 PG_SYNONYM 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
synname	name	同义词名称。
synnamespace	oid	包含该同义词的名字空间的OID。
synowner	oid	同义词的所有者，通常是创建它的用户OID。
synobjschema	name	关联对象指定的模式名。
synobjname	name	关联对象名。
syndblinkname	name	关联DATABASE LINK对象名。

12.2.88 PG_TABLESPACE

PG_TABLESPACE系统表存储表空间信息。

表 12-89 PG_TABLESPACE 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
spcname	name	表空间名称。
spcowner	oid	表空间的所有者，通常是创建它的人。
spcacl	aclitem[]	访问权限。具体请参见GRANT和REVOKE。
spcoptions	text[]	表空间的选项。
spcmaxsize	text	可使用的最大磁盘空间大小，单位Byte。
relative	boolean	标识表空间指定的存储路径是否为相对路径。 • t（true）：表示是。 • f（false）：表示不是。

12.2.89 PG_TRIGGER

PG_TRIGGER系统表存储触发器信息。

表 12-90 PG_TRIGGER 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
tgrelid	oid	触发器所在表的OID。
tgname	name	触发器名。
tgfoid	oid	需要被触发器调用的函数。
tgtype	smallint	触发器类型。
tgenabled	"char"	O =触发器在“origin”和“local”模式下触发。 D =触发器被禁用。 R =触发器在“replica”模式下触发。 A =触发器始终触发。
tgisinternal	boolean	内部触发器标识，如果为true表示内部触发器。
tgconstrrelid	oid	完整性约束引用的表。
tgconstrindid	oid	完整性约束的索引。
tgconstraint	oid	约束触发器在pg_constraint中的OID。
tgdeferrable	boolean	约束触发器是为DEFERRABLE类型。
tginitdeferred	boolean	约束触发器是否为INITIALLY DEFERRED类型。
tgnargs	smallint	触发器函数入参个数。
tgattr	int2vector	当触发器指定列时的列号，未指定则为空数组。
tgargs	bytea	传递给触发器的参数。
tgqual	pg_node_tree	表示触发器的WHEN条件，如果没有则为null。
tgowner	oid	触发器的所有者。

12.2.90 PG_TS_CONFIG

PG_TS_CONFIG系统表包含表示文本搜索配置的记录。一个配置指定一个特定的文本搜索解析器和一个为了每个解析器的输出类型使用的字典的列表。

解析器在PG_TS_CONFIG记录中显示，但是字典映射的标记是由PG_TS_CONFIG_MAP里面的辅助记录定义的。

表 12-91 PG_TS_CONFIG 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
cfgname	name	-	文本搜索配置名。
cfgnamespace	oid	PG_NAMESPACE.oid	包含这个配置的名称空间的OID。
cfgowner	oid	PG_AUTHID.oid	配置的所有者。
cfgparser	oid	PG_TS_PARSER.oid	这个配置的文本搜索解析器的OID。
cfoptions	text[]	-	分词相关配置选项。

12.2.91 PG_TS_CONFIG_MAP

PG_TS_CONFIG_MAP系统表包含为每个文本搜索配置的解析器的每个输出符号类型，显示哪个文本搜索字典应该被咨询、以什么顺序搜索的记录。

表 12-92 PG_TS_CONFIG_MAP 字段

名称	类型	引用	描述
mapcfg	oid	PG_TS_CONFIG.oid	拥有这个映射记录的PG_TS_CONFIG记录的OID。
maptokentype	integer	-	由配置的解析器产生的一个符号类型值。
mapseqno	integer	-	在相同mapcfg或maptokentype值的情况下，该符号类型的顺序号。
mapdict	oid	PG_TS_DICT.oid	要咨询的文本搜索字典的OID。

12.2.92 PG_TS_DICT

PG_TS_DICT系统表包含定义文本搜索字典的记录。字典取决于文本搜索模板，该模板声明所有需要的实现函数；字典本身提供模板支持的用户可设置的参数的值。

这种分工允许字典通过非权限用户创建。参数由文本字符串dictinitoption指定，参数的格式和意义取决于模板。

表 12-93 PG_TS_DICT 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
dictname	name	-	文本搜索字典名。
dictnamespace	oid	PG_NAMESPACE.oid	包含这个字典的名称空间的OID。
dictowner	oid	PG_AUTHID.oid	字典的所有者。
dicttemplate	oid	PG_TS_TEMPLATE.oid	这个字典的文本搜索模板的OID。
dictinitoption	text	-	该模板的初始化选项字符串。

12.2.93 PG_TS_PARSER

PG_TS_PARSER系统表包含定义文本解析器的记录。解析器负责分裂输入文本为词位，并且为每个词位分配标记类型。新解析器必须由数据库系统管理员创建。

表 12-94 PG_TS_PARSER 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性；必须明确选择）。
prsname	name	-	文本搜索解析器名。
prsnamespace	oid	PG_NAMESPACE.oid	包含这个解析器的名称空间的OID。
prsstart	regproc	PG_PROC.proname	解析器的启动函数名。
prstoken	regproc	PG_PROC.proname	解析器的下一个标记函数名。
prsend	regproc	PG_PROC.proname	解析器的关闭函数名。
prsheadline	regproc	PG_PROC.proname	解析器的标题函数名。
prslextype	regproc	PG_PROC.proname	解析器的lextype函数名。

12.2.94 PG_TS_TEMPLATE

PG_TS_TEMPLATE系统表包含定义文本搜索模板的记录。模板是文本搜索字典的类的实现框架。因为模板必须通过C语言级别的函数实现，索引新模板的创建必须由数据库系统管理员创建。

表 12-95 PG_TS_TEMPLATE 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性；必须明确选择）。
tmplname	name	-	文本搜索模板名。
tmplnamespace	oid	PG_NAMESPACE.oid	包含这个模板的名称空间的OID。
tmplinit	regproc	PG_PROC.proname	模板的初始化函数名。
tmpllexize	regproc	PG_PROC.proname	模板的lexize函数名。

12.2.95 PG_TYPE

PG_TYPE系统表存储数据类型的相关信息。

表 12-96 PG_TYPE 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
typname	name	数据类型名称。
typnamespace	oid	包含这个类型的名称空间的OID。
typowner	oid	该类型的所有者。
typlen	smallint	对于定长类型是该类型内部表现形式的字节数目。对于变长类型是负数。 ● -1表示一种“变长”（有长度字属性的数据）。 ● -2表示这是一个NULL结尾的C字符串。
typbyval	boolean	● true：指定内部传递这个类型的数值时是传值。 ● false：指定内部传递这个类型的数值时是传引用。 如果该类型的typlen不是1、2、4、8， typbyval建议传引用，也可以传值。变长类型通常是传引用，也可以传值。

名称	类型	描述
typtype	"char"	• b：基础类型。 • c：复合类型（比如，一个表的行类型）。 • d：域类型。 • p：伪类型。 • u：未定义类型。 • o：集合类型。 参见typrelid和typbasetype。
typcategory	"char"	是数据类型的模糊分类，可用于解析器作为数据转换的依据。
typispreferred	boolean	• true：数据符合typcategory所指定的转换规则时进行转换。 • false：不进行转换。
typisdefined	boolean	• true：表示类型已定义。 • false：表示是一种尚未定义的类型的占位符，此时，除了该类型的名称、名称空间和OID之外没有可靠的信息。
typdelim	"char"	当分析数组输入时，分隔两个此类型数值的字符请注意该分隔符是与数组元素数据类型相关联的，而不是和数组数据类型关联。
typrelid	oid	如果是复合类型（请参见typtype），则这个字段指向13.2.53 PG_CLASS中定义该表的行。对于自由存在的复合类型，13.2.53 PG_CLASS记录并不表示一个表，但是总需要它来查找该类型连接的13.2.48 PG_ATTRIBUTE记录。对于非复合类型为零。
typelem	oid	如果不为0，则它标识pg_type里面的另外一行。当前类型可以当做一个产生类型为typelem的数组来描述。一个“真正的”数组类型是变长的（typlen= -1），但是一些定长的（typlen > 0）类型也拥有非零的typelem（比如name和point）。如果一个定长类型拥有一个typelem，则他的内部形式必须是typelem数据类型的某个数目的个数值，不能有其它数据。变长数组类型有一个该数组子过程定义的头（文件）。
typarray	oid	如果不为0，则表示在pg_type中有对应的类型记录。
typinput	regproc	输入转换函数（文本格式）。
typoutput	regproc	输出转换函数（文本格式）。
typreceive	regproc	输入转换函数（二进制格式），如果没有则为0。
typsend	regproc	输出转换函数（二进制格式），如果没有则为0。
typmodin	regproc	输入类型修改符函数，如果没有则为0。

名称	类型	描述
typmodout	regproc	输出类型修改符函数，如果没有则为0。
typanalyze	regproc	自定义的ANALYZE函数，如果使用标准函数，则为0。
typalign	"char"	当存储此类型的数值时要求的对齐性质。它应用于磁盘存储以及该值的大多数形式。如果数值是连续存放的，比如在磁盘上以完全的裸数据的形式存放时，则先在此类型的数据前填充空白，这样它就可以按照要求的界限存储。对齐引用是该序列中第一个数据的开头。可能的值包含： ● c：表示char对齐，也就是不需要对齐。 ● s：表示short对齐（在大多数机器上是2字节）。 ● i：表示int对齐（在大多数机器上是4字节）。 ● d：表示double对齐（在大多数机器上是8字节，但不一定是全部）。 **须知** 对于在系统表里使用的类型，在pg_type里定义的尺寸和对齐必须和编译器在一个表示表的一行的结构里的布局一样。
typstorage	"char"	指明一个变长类型（那些有typlen = -1）是否准备好应付非常规值，以及对这种属性的类型的缺省策略是什么。可能的值包含： ● p：数值总是以简单方式存储。 ● e：数值可以存储在一个“次要”关系中（如果该关系有这么一个，请参见13.2.53 PG_CLASS.reltoastrelid）。 ● m：数值可以以内联的压缩方式存储。 ● x：数值可以以内联的压缩方式或者在“次要”表里存储。 **须知** m域也可以移到从属表里存储，但只是最后的解决方法（e和x域先移走）。
typnotnull	boolean	该类型是否存在NOTNULL约束。目前只用于域。
typbasetype	oid	如果这是一个衍生类型（请参见typtype），则该标识作为这个类型的基础的类型。如果不是衍生类型则为零。
typtypmod	integer	● 域使用typtypmod记录要作用到它们的基础类型上的typmod（如果基础类型不使用typmod则为-1）。如果这种类型不是域，则为-1。 ● 如果该类型是数组类型则typtypmod是数组类型的最大容量。 ● 如果该类型是带索引的集合类型则typtypmod是该集合类型的索引最大长度。
typndims	integer	如果一个域是数组，则typndims是数组维数的数值。非域非数组域为零。

名称	类型	描述
typcollation	oid	指定类型的排序规则。取值参考PG_COLLATION系统表。如果为0，则表示不支持排序。
typdefaultbin	pg_node_tree	如果为非NULL，则它是该类型缺省表达式的nodeToString() 表现形式。目前这个字段只用于域。
typdefault	text	如果某类型没有相关缺省值，则取值是NULL。 • 如果typdefaultbin为非NULL，则typdefault必须包含一个typdefaultbin代表的缺省表达式。 • 如果typdefaultbin为NULL但typdefault不是，typdefault则是该类型缺省值的外部表现形式，可以把它作为该类型的输入，转换器生成一个常量。
typacl	aclitem[]	访问权限。
typelemmod	integer	• -1：表示集合和数组类型对应元素类型不需要typmod。 • >=0的数值：表示集合和数组类型对应元素类型的typmod值。 • NULL：表示集合和数组类型对应元素类型的typmod未知。 typmod通常指类型的最大长度。

12.2.96 PG_USER_MAPPING

PG_USER_MAPPING系统表存储从本地用户到远程的映射。

需要有系统管理员权限才可以访问此系统表。普通用户可以使用视图PG_USER_MAPPINGS进行查询。

表 12-97 PG_USER_MAPPING 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
umuser	oid	PG_AUTHID.oid	被映射的本地用户的OID，如果用户映射是公共的则为0。
umserver	oid	PG_FOREIGN_SERVER.oid	包含这个映射的外部服务器的OID。
umoptions	text[]	-	用户映射指定选项，使用“keyword=value”格式的字符串。

12.2.97 PG_USER_STATUS

PG_USER_STATUS系统表存储访问数据库用户的状态信息。需要有系统管理员权限才可以访问此系统表

表 12-98 PG_USER_STATUS 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
roloid	oid	角色的标识。
failcount	integer	尝试失败次数。
locktime	timestamp with time zone	默认显示角色的创建日期，如果角色被管理员锁定，或者登录失败次数超过阈值被锁定，则显示角色被锁定的日期。
rolstatus	smallint	角色的状态。 ● 0：正常状态。 ● 1：由于登录失败次数超过阈值被锁定了一定的时间。 ● 2：被管理员锁定。
permspace	bigint	角色已经使用的永久表存储空间大小。
tempspace	bigint	角色已经使用的临时表存储空间大小。
password expired	smallint	密码是否失效。 ● 0：密码有效。 ● 1：密码失效。

12.2.98 PGXC_CLASS

PGXC_CLASS系统表存储每张表的复制或分布信息。PGXC_CLASS系统表在集中式场景下只能查询表定义。

表 12-99 PGXC_CLASS 字段

名称	类型	描述
pcrelid	oid	表的OID。

名称	类型	描述
pclocatortype	"char"	定位器类型。 ● H：hash ● G：Range ● L：List ● M：Modulo ● N：Round Robin ● R：Replication
pchashalgorithm	smallint	使用哈希算法分布元组。
pchashbuckets	smallint	哈希容器的值。
pgroup	name	节点群的名称。
redistributed	"char"	表已经完成重分布。
redis_order	integer	重分布的顺序。该值等于0的表在本轮重分布过程中不进行重分布。
pcattnum	int2vector	用作分布键的列标号。
nodeoids	oidvector_extend	表分布的节点OID列表。
options	text	系统内部保留字段，存储扩展状态信息。

12.2.99 PGXC_GROUP

PGXC_GROUP系统表存储节点组信息。PGXC_GROUP系统表在集中式场景下只能查询表定义。

表 12-100 PGXC_GROUP 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
group_name	name	节点组名称。
in_redistribution	"char"	是否需要重分布。取值包括： ● n：表示NodeGroup没有再进行重分布。 ● y：表示NodeGroup是重分布过程中的源节点组。 ● t：表示NodeGroup是重分布过程中的目的节点组。
group_members	oidvector_extend	节点组的节点OID列表。

名称	类型	描述
group_buckets	text	分布数据桶的集合。
is_installation	boolean	是否安装子数据库实例。 ● t（true）：表示安装。 ● f（false）：表示不安装。
group_acl	aclitem[]	访问权限。
group_kind	"char"	node group类型，取值包括： ● i：表示installation node group。 ● n：表示普通非逻辑数据库实例node group。 ● v：表示逻辑数据库实例node group。 ● e：表示弹性数据库实例。
group_parent	oid	如果是子node group，该字段表示父node group的OID，如果是父node group，该字段值为空。

12.2.100 PGXC_NODE

PGXC_NODE系统表存储数据库实例节点信息。PGXC_NODE系统表仅在分布式场景下有具体含义，集中式只能查询表定义。

表 12-101 PGXC_NODE 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
node_name	name	节点名称。
node_type	"char"	节点类型。 ● C：协调节点。 ● D：数据节点。 ● S：数据节点的备节点。
node_port	integer	节点的端口号。
node_host	name	节点的主机名称或者IP（如配置为虚拟IP，则为虚拟IP）。
node_port1	integer	复制节点的端口号。
node_host1	name	复制节点的主机名称或者IP（如配置为虚拟IP，则为虚拟IP）。
hostis_primary	boolean	表明当前节点是否发生主备切换。 ● t（true）：表示发生。 ● f（false）：表示不发生。

名称	类型	描述
nodeis_primary	boolean	在replication表下，是否优选当前节点作为优先执行的节点进行非查询操作。 ● t（true）：表示优选。 ● f（false）：表示不优选。
nodeis_preferred	boolean	在replication表下，是否优选当前节点作为首选的节点进行查询。 ● t（true）：表示优选。 ● f（false）：表示不优选。
node_id	integer	节点标识符。由node_name经过hash函数计算后得到。
sctp_port	integer	主节点使用TCP代理通信库的数据通道侦听端口（当前版本已经不再支持SCTP通信库）。
control_port	integer	主节点使用TCP代理通信库的控制通道侦听端口。
sctp_port1	integer	备节点使用TCP代理通信库的数据通道侦听端口（当前版本已经不再支持SCTP通信库）。
control_port1	integer	备节点使用TCP代理通信库的控制通道侦听端口。
nodeis_central	boolean	表明当前节点是否为中心控制节点，对DN无效。 ● t（true）：表示是。 ● f（false）：表示不是。
nodeis_active	boolean	表明当前节点是否是正常状态，对DN无效。 ● t（true）：表示是。 ● f（false）：表示不是。

12.2.101 PGXC_SLICE

PGXC_SLICE表是针对range范围分布和list分布创建的系统表，用来记录分布具体信息，当前不支持range interval自动扩展分片，不过在系统表中预留。

集中式只能查询表定义。

表 12-102 PGXC_SLICE 字段

名称	类型	描述
relname	name	表名或者分片名，通过type区分
type	"char"	● 't'：relname是表名。 ● 's'：relname是分片的名字。

名称	类型	描述
strategy	"char"	● 'r'：为range分布表 ● 'l'：为list分布表 后续interval分片会扩展该值
relid	oid	该tuple隶属的分布表oid
referenceoid	oid	所参考分布表的oid,用于slice reference建表语法
sindex	integer	当为list分布表时，表示当前boundary在某个分片内的位置
interval	text[]	预留字段
transitboundary	text[]	预留字段
transitno	integer	预留字段
nodeoid	oid	当relname为分片名时，表示该分片的数据存放在哪个DN上，nodeoid表示这个DN的oid
boundaries	text[]	当relname为分片名时，对应该分片的边界值。
specified	boolean	当前分片对应的DN是否是用户在DDL中显示指定的。
sliceorder	integer	用户定义分片的顺序

12.2.102 PLAN_TABLE_DATA

PLAN_TABLE_DATA存储了用户通过执行EXPLAIN PLAN收集到的计划信息。与PLAN_TABLE视图不同的是PLAN_TABLE_DATA表存储了所有session和user执行EXPLAIN PLAN收集的计划信息。

表 12-103 PLAN_TABLE_DATA 字段

名称	类型	描述
session_id	text	表示插入该条数据的会话，由服务线程启动时间戳和服务线程ID组成。受非空约束限制。
user_id	oid	用户ID，用于标识触发插入该条数据的用户。受非空约束限制。
statement_id	character varying(30)	用户输入的查询标签。

名称	类型	描述
plan_id	bigint	查询标识。该标识在计划生成阶段自动产生，供内核工程师调试使用。
id	integer	计划中的节点编号。
operation	character varying(30)	操作描述。
options	character varying(255)	操作选项。
object_name	name	操作对应的对象名，来自于用户定义。
object_type	character varying(30)	对象类型。
object_owner	name	对象所属schema，来自于用户定义。
projection	character varying(4000)	操作输出的列信息。
cost	double precision	优化器对算子估算的执行代价。
cardinality	double precision	优化器对算子估算的结果行数。

说明

- PLAN_TABLE_DATA中包含了当前节点所有用户、所有会话的数据，仅管理员有访问权限。普通用户可以通过**PLAN_TABLE**视图查看属于自己的数据。
- PLAN_TABLE_DATA中的数据是用户通过执行EXPLAIN PLAN命令后由系统自动插入表中，因此禁止用户手动对数据进行插入或更新，否则会引起表中的数据混乱。需要对表中数据删除时，建议通过**PLAN_TABLE**视图。
- statement_id、object_name、object_owner和projection字段内容遵循用户定义的大小写存储，其它字段内容采用大写存储。

12.2.103 STATEMENT_HISTORY

获得当前节点的执行语句的信息。查询系统表必须具有sysadmin权限。只可在系统库中查询到结果，用户库中无法查询。

对于此系统表查询有如下约束：

- 必须在postgres库内查询，其它库中不存数据。
- 此系统表受track_stmt_stat_level控制，默认为"OFF,L0"，第一部分控制Full SQL，第二部分控制Slow SQL，具体字段记录级别见下表。考虑性能影响，更改该参数的值时建议通过set方式设置，使该参数仅对当前会话生效。

- 对于Slow SQL，当track_stmt_stat_level的值为非OFF时，且SQL执行时间超过log_min_duration_statement，会记录为慢SQL。

  说明

  当前版本暂不支持对FOR UPDATE关键字进行识别并归一化处理。例如：SELECT * FROM table; 与SELECT * FROM table FOR UPDATE WAIT N; 会被归一化处理为相同的归一化SQL，在query字段中体现。涉及FOR UPDATE关键字的SQL，可以通过query_plan字段进行区分，执行计划中会含有'lockRows'。

表 12-104 STATEMENT_HISTORY 字段

名称	类型	描述	记录级别
db_name	name	数据库名称。	L0
schema_name	name	schema名称。	L0
origin_node	integer	节点名称。	L0
user_name	name	用户名。	L0
application_name	text	用户发起的请求的应用程序名称。	L0
client_addr	text	用户发起的请求的客户端地址。	L0
client_port	integer	用户发起的请求的客户端端口。	L0
unique_query_id	bigint	归一化SQL ID。	L0
debug_query_id	bigint	唯一SQL ID。部分语句存在不唯一的情况，如Parse报文、DCL和TCL等语句的debug_query_id值为0。	L0
query	text	归一化SQL，track_stmt_parameter参数开启时，显示完整SQL。	L0
start_time	timestamp with time zone	语句启动的时间。	L0
finish_time	timestamp with time zone	语句结束的时间。	L0
slow_sql_threshold	bigint	语句执行时慢SQL的标准。	L0
transaction_id	bigint	事务ID。	L0
thread_id	bigint	执行线程ID。	L0
session_id	bigint	用户session id。	L0

名称	类型	描述	记录级别
n_soft_parse	bigint	软解析次数，n_soft_parse + n_hard_parse可能大于n_calls，因为子查询未计入n_calls。	L0
n_hard_parse	bigint	硬解析次数，n_soft_parse + n_hard_parse可能大于n_calls，因为子查询未计入n_calls。	L0
query_plan	text	语句执行计划。对于Slow SQL的query_plan，execution_time大于slow_sql_threshold时记录，发生死锁等场景query_plan有可能为空。	L0
n_returned_rows	bigint	SELECT返回的结果集行数。	L0
n_tuples_fetched	bigint	随机扫描行。	L0
n_tuples_returned	bigint	顺序扫描行。	L0
n_tuples_inserted	bigint	插入行。	L0
n_tuples_updated	bigint	更新行。	L0
n_tuples_deleted	bigint	删除行。	L0
n_blocks_fetched	bigint	buffer的块访问次数。	L0
n_blocks_hit	bigint	buffer的块命中次数。	L0
db_time	bigint	有效的DB时间花费，多线程将累加（单位：微秒）。	L0
cpu_time	bigint	CPU时间（单位：微秒）。	L0
execution_time	bigint	执行器内执行时间（单位：微秒）。	L0
parse_time	bigint	SQL解析时间（单位：微秒）。	L0
plan_time	bigint	SQL生成计划时间（单位：微秒）。	L0
rewrite_time	bigint	SQL重写时间（单位：微秒）。	L0
pl_execution_time	bigint	plpgsql上的执行时间（单位：微秒）。	L0
pl_compilation_time	bigint	plpgsql上的编译时间（单位：微秒）。	L0

名称	类型	描述	记录级别
data_io_time	bigint	IO上的时间花费（单位：微秒）。	L0
net_send_info	text	通过物理连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。集中式不支持该字段。	L0
net_recv_info	text	通过物理连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。集中式不支持该字段。	L0
net_stream_send_info	text	通过逻辑连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。集中式不支持该字段。	L0
net_stream_recv_info	text	通过逻辑连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。集中式不支持该字段。	L0
lock_count	bigint	加锁次数。	L0
lock_time	bigint	加锁耗时。	L1
lock_wait_count	bigint	加锁等待次数。	L0
lock_wait_time	bigint	加锁等待耗时。	L1
lock_max_count	bigint	最大持锁数量。	L0
lwlock_count	bigint	轻量级加锁次数（预留）。	L0
lwlock_wait_count	bigint	轻量级等锁次数。	L0
lwlock_time	bigint	轻量级加锁时间（预留）。	L1
lwlock_wait_time	bigint	轻量级等锁时间。	L1

名称	类型	描述	记录级别
details	bytea	等待事件和语句锁事件的列表。 记录级别的值>=L0时，开始记录等待事件的列表。显示当前语句event等待相关的统计信息。具体事件信息见**表12-351**、**15.3.67-表3 轻量级锁等待事件列表**、**表12-353**和**表12-354**。关于每种事务锁对业务的影响程度，请参考**LOCK**语法小节的详细描述。 记录级别的值是L2时，开始记录语句锁事件的列表，该列表按时间顺序记录事件，记录的数量受参数track_stmt_details_size的影响。 该字段为二进制，需要借助解析函数pg_catalog.statement_detail_decode读取，见（**表7-63**）。 事件包括： • 加锁开始 • 加锁结束 • 等锁开始 • 等锁结束 • 放锁开始 • 放锁结束 • 轻量级等锁开始 • 轻量级等锁结束	L0/L2
is_slow_sql	boolean	该SQL是否为slow SQL。 • t（true）：表示是。 • f（false）：表示不是。	L0
trace_id	text	驱动传入的trace id，与应用的一次请求相关联。	L0
advise	text	可能导致该SQL为slow SQL的风险信息。	L0
parent_unique_sql_id	bigint	当前语句的外层SQL的归一化SQL ID，对于存储过程内执行的语句，该值为调用存储过程语句的归一化SQL ID，存储过程外的语句该值为0。	L0

12.2.104 STREAMING_STREAM

STREAMING_STREAM系统表存储所有STREAM对象的元数据信息。

表 12-105 STREAMING_STREAM 字段

名称	类型	描述
relid	oid	STREAM对象的标识。
queries	bytea	该STREAM对应CONTVIEW的位图映射。

12.2.105 STREAMING_CONT_QUERY

STREAMING_CONT_QUERY系统表存储所有CONTVIEW对象的元数据信息。

表 12-106 STREAMING_CONT_QUERY 字段

名称	类型	描述
id	integer	CONTVIEW对象唯一的标识符，不可重复。
type	"char"	标识CONTVIEW的类型。 ● 'r'表示该CONTVIEW是基于行存存储模型。
relid	oid	CONTVIEW对象的标识。
defrelid	oid	CONTVIEW对应的持续计算规则VIEW的标识。
active	boolean	标识CONTVIEW是否处于持续计算状态。 ● t（true）：表示是。 ● f（false）：表示不是。
streamrelid	oid	CONTVIEW对应的STREAM的标识。
matrelid	oid	CONTVIEW对应物化表的标识。
lookupidxid	oid	CONTVIEW对应GROUP LOOK UP INDEX的标识，此字段内部使用，仅行存具有。
step_factor	smallint	标识CONTVIEW的步进模式。主要取值为0（无重叠窗口）和1（滑动窗口，步长为1）。
ttl	integer	CONTVIEW设置的ttl_interval参数值。
ttl_attno	smallint	CONTVIEW设置的TTL功能对应时间列的字段编号。
dictrelid	oid	CONTVIEW对应字典表的标识。
grpnum	smallint	CONTVIEW持续计算规则中维度列的个数，此字段内部使用。
grpidx	int2vector	CONTVIEW持续计算规则中维度列在TARGET LIST的索引，此字段内部使用。

12.3 系统视图

12.3.1 ADM_ARGUMENTS

ADM_ARGUMENTS视图显示所有存储过程或函数的参数信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-107 ADM_ARGUMENTS 字段

名称	类型	描述
owner	character varying(128)	函数或存储过程的所有者。
object_name	character varying(128)	函数或存储过程的名称。
package_name	character varying(128)	包名。
object_id	oid	函数或存储过程的OID。
overload	character varying(40)	表示该函数是该名称的第n个重载函数。
subprogram_id	numeric	包中函数或存储过程的位置。
argument_name	character varying(128)	参数名称。
position	numeric	该参数在参数列表中的位置，函数的返回值位置默认为0。
sequence	numeric	定义参数的顺序，从1开始，返回类型在前，然后是每个参数。
data_level	numeric	复合类型参数的嵌套深度，此列的值始终为0，因为每个参数现在只显示一行。
data_type	character varying(64)	参数的数据类型。
defaulted	character varying(1)	参数是否有默认值： ● Y：表示有默认值。 ● N：表示没有默认值。
default_value	text	暂不支持，值为NULL。
default_length	numeric	暂不支持，值为NULL。

名称	类型	描述
in_out	character varying(9)	参数出入属性： ● IN：表示入参。 ● OUT：表示出参。 ● IN_OUT：表示出入参。 ● VARIADIC：表示VARIADIC参数。
data_length	numeric	暂不支持，值为NULL。
data_precision	numeric	暂不支持，值为NULL。
data_scale	numeric	暂不支持，值为NULL。
radix	numeric	数字的参数基数，smallint,integer,bigint,numeric,float为10，其余值为NULL。
character_set_name	character varying(44)	暂不支持，值为NULL。
type_owner	character varying(128)	数据类型所有者。
type_name	character varying(128)	参数类型名，仅显示自定义类型。
type_subname	character varying(128)	暂不支持，值为NULL。
type_link	character varying(128)	暂不支持，值为NULL。
type_object_type	character varying(7)	由type_owner、type_name和type_subname列描述的类型的类型： ● TABLE：表示参数为表类型。 ● VIEW：表示参数为视图类型。 ● 其余暂为空。
pls_type	character varying(128)	对于数字类型参数，为参数的PL/SQL类型的名称，否则为空。
char_length	numeric	暂不支持，值为NULL。
char_used	character varying(1)	暂不支持，varchar，nvarchar2，bpchar，char类型置B，其余值为NULL。
origin_con_id	character varying(256)	暂不支持，值为0。

12.3.2 ADM_AUDIT_OBJECT

ADM_AUDIT_OBJECT显示数据库中所有对象的审计跟踪记录, 该视图同时存在于pg_catalog和sys schema下。默认只有系统管理员权限才可以访问,普通用户需要授权才可以访问。

表 12-108 ADM_AUDIT_OBJECT 字段

名称	类型	描述
os_username	character varying(255)	暂不支持，值为NULL。
username	character varying(128)	操作被审计的用户的名称，不是用户ID。
userhost	character varying(128)	暂不支持，值为NULL。
terminal	character varying(255)	暂不支持，值为NULL。
timestamp	timestamp(0) without time zone	在本地数据库会话时区中创建审计跟踪条目的日期和时间（审计会话创建的条目的用户登录日期和时间）。
owner	character varying(128)	受操作影响的对象的创建者。
obj_name	character varying(128)	受操作影响的对象的名称。
action_name	character varying(28)	DBA_AUDIT_TRAIL中的“操作”列中的数字代码对应的动作类型名称。 **说明** GaussDB的action_name字段与A数据库审计动作不一致。
new_owner	character varying(128)	暂不支持，值为NULL。
new_name	character varying(128)	暂不支持，值为NULL。

名称	类型	描述
ses_actions	character varying(19)	暂不支持，值为NULL。
comment_text	character varying(4000)	暂不支持，值为NULL。
sessionid	numeric	暂不支持，值为NULL。
entryid	numeric	暂不支持，值为NULL。
statementid	numeric	暂不支持，值为NULL。
returncode	numeric	暂不支持，值为NULL。
priv_used	character varying(40)	暂不支持，值为NULL。
client_id	character varying(128)	暂不支持，值为NULL。
econtext_id	character varying(64)	暂不支持，值为NULL。
session_cpu	numeric	暂不支持，值为NULL。
extended_timestamp	timestamp(6) with time zone	在UTC（协调世界时）时区创建审计跟踪条目的时间戳（审计会话创建的条目的用户登录时间戳）。
proxy_sessionid	numeric	暂不支持，值为NULL。
global_uid	character varying(32)	暂不支持，值为NULL。
instance_numeric	numeric	暂不支持，值为NULL。
os_process	character varying(16)	暂不支持，值为NULL。
transactionid	text	访问或修改对象事务的事务标识符。 **说明** GaussDB的transactionid字段与A数据库中transactionid数据的类型保持一致。
scn	numeric	暂不支持，值为NULL。
sql_bind	Ncharacter varying(2000)	暂不支持，值为NULL。

名称	类型	描述
sql_text	Ncharacter varying(2000)	查询的SQL文本。 **说明** GaussDB的sql_text字段为解析后sql描述语句，不完全与执行的sql语句相同。
obj_edition_name	character varying(128)	暂不支持，值为NULL。

12.3.3 ADM_AUDIT_SESSION

ADM_AUDIT_SESSION显示所有连接断开数据库审计信息，GaussDB审计信息主要通过pg_query_audit函数查询，该视图同时存在于PG_CATALOG和SYS schema下。仅拥有AUDITADMIN属性的用户才可以查看审计信息。

表 12-109 ADM_AUDIT_SESSION 字段

名称	类型	描述
os_username	character varying(255)	暂不支持，值为NULL。
username	character varying(128)	操作被审计的用户的名称，不是用户ID。
userhost	character varying(128)	暂不支持，值为NULL。
terminal	character varying(128)	暂不支持，值为NULL。
timestamp	timestamp(0) without time zone	创建审核跟踪条目的日期和时间（用户登录创建条目的日期和时间AUDIT SESSION）。
action_name	character varying(28)	DBA_AUDIT_TRAIL中的ACTION列中的数字代码对应的动作类型的名称。 **说明** GaussDB的action_name字段与A数据库审计动作不一致。
logoff_time	timestamp(0) without time zone	暂不支持，值为NULL。

名称	类型	描述
logoff_lread	numeric	暂不支持，值为NULL。
logoff_pread	numeric	暂不支持，值为NULL。
logoff_lwrite	numeric	暂不支持，值为NULL。
logoff_dlock	character varying(40)	暂不支持，值为NULL。
sessionid	numeric	暂不支持，值为NULL。
returncode	numeric	暂不支持，值为NULL。
client_id	character varying(128)	暂不支持，值为NULL。
session_cpu	numeric	暂不支持，值为NULL。
extended_timestamp	timestamp(6) with time zone	在UTC（协调世界时）时区创建审核跟踪条目的时间戳（创建条目的用户登录时间戳AUDIT SESSION）。
proxy_sessionid	numeric	暂不支持，值为NULL。
global_uid	character varying(32)	暂不支持，值为NULL。
instance_numeric	numeric	暂不支持，值为NULL。
os_process	character varying(16)	暂不支持，值为NULL。

12.3.4 ADM_AUDIT_STATEMENT

ADM_AUDIT_STATEMENT显示所有GRANT、REVOKE审计跟踪条目，GaussDB审计信息主要通过pg_query_audit函数查询，该视图同时存在于PG_CATALOG和SYS schema下。仅拥有AUDITADMIN属性的用户才可以查看审计信息。

表 12-110 ADM_AUDIT_STATEMENT 字段

名称	类型	描述
os_username	character varying(255)	暂不支持，值为NULL。

名称	类型	描述
username	character varying(128)	操作被审计的用户的名称，不是用户ID。
userhost	character varying(128)	暂不支持，值为NULL。
terminal	character varying(255)	暂不支持，值为NULL
timestamp	timestamp(0) without time zone	创建审核跟踪条目的日期和时间（用户登录创建条目的日期和时间AUDIT SESSION）。
owner	character varying(128)	暂不支持，值为NULL。
obj_name	character varying(128)	受操作影响的对象的名称。
action_name	character varying(28)	DBA_AUDIT_TRAIL中的ACTION列中的数字代码对应的动作类型的名称。 **说明** GaussDB的action_name字段与A数据库审计动作不一致。
new_name	character varying(128)	暂不支持，值为NULL。
obj_privilege	character varying(32)	暂不支持，值为NULL。
sys_privilege	character varying(40)	暂不支持，值为NULL。
admin_option	character varying(1)	暂不支持，值为NULL。
grantee	character varying(128)	暂不支持，值为NULL。
audit_option	character varying(40)	暂不支持，值为NULL。

名称	类型	描述
ses_actions	character varying(19)	暂不支持，值为NULL。
comment_text	character varying(4000)	暂不支持，值为NULL。
sessionid	numeric	暂不支持，值为NULL。
entryid	numeric	暂不支持，值为NULL。
statementid	numeric	暂不支持，值为NULL。
returncode	numeric	暂不支持，值为NULL。
priv_used	character varying(40)	暂不支持，值为NULL。
client_id	character varying(128)	暂不支持，值为NULL。
econtext_id	character varying(64)	暂不支持，值为NULL。
session_cpu	numeric	暂不支持，值为NULL。
extended_timestamp	timestamp(6) with time zone	在UTC（协调世界时）时区创建审核跟踪条目的时间戳（创建条目的用户登录时间戳AUDIT SESSION）。
proxy_sessionid	numeric	暂不支持，值为NULL。
global_uid	character varying(32)	暂不支持，值为NULL。
instance_number	numeric	暂不支持，值为NULL。
os_process	character varying(16)	暂不支持，值为NULL。
transactionid	text	访问或修改对象事务的事务标识符。 **说明** GaussDB的transactionid字段与A数据库中transactionid数据的类型保持一致。
scn	numeric	暂不支持，值为NULL。
sql_bind	nvarchar2(2000)	暂不支持，值为NULL。

名称	类型	描述
sql_text	character varying(2000)	查询的SQL文本。 **说明** GaussDB的sql_text字段为解析后的SQL描述语句，不完全与执行的SQL语句相同。
obj_edition_name	character varying(128)	暂不支持，值为NULL。

12.3.5 ADM_AUDIT_TRAIL

ADM_AUDIT_TRAIL显示所有标准审计跟踪条目，GaussDB审计信息主要通过pg_query_audit函数，该视图同时存在于PG_CATALOG和SYS schema下。仅拥有AUDITADMIN属性的用户才可以查看审计信息。GaussDB的action_name字段与A数据库审计动作不一致，transactionid字段与A数据库中transactionid数据的类型保持一致，GaussDB的sql_text字段为解析后sql描述语句，不完全与执行的sql语句相同。

表 12-111 ADM_AUDIT_TRAIL 字段

名称	类型	描述
os_username	character varying(255)	暂不支持，值为NULL。
username	character varying(128)	操作被审计的用户的名称，不是用户ID。
userhost	character varying(128)	暂不支持，值为NULL。
terminal	character varying(255)	暂不支持，值为NULL。
timestamp	timestamp(0) without time zone	在本地数据库会话时区中创建审计跟踪条目的日期和时间（审计会话创建的条目的用户登录日期和时间）。
owner	character varying(128)	受操作影响的对象的创建者。
obj_name	character varying(128)	受操作影响的对象的名称。
action	numeric	暂不支持，值为NULL。
action_name	character varying(28)	action列中的数字代码对应的action类型名称。 **说明** GaussDB的action_name字段与A数据库审计动作不一致。

名称	类型	描述
new_owner	character varying(128)	暂不支持，值为NULL。
new_name	character varying(128)	暂不支持，值为NULL。
obj_privilege	character varying(32)	暂不支持，值为NULL。
sys_privilege	character varying(40)	暂不支持，值为NULL。
admin_option	character varying(1)	暂不支持，值为NULL。
grantee	character varying(128)	暂不支持，值为NULL。
audit_option	character varying(40)	暂不支持，值为NULL。
ses_actions	character varying(19)	暂不支持，值为NULL。
logoff_time	timestamp(0) without time zone	暂不支持，值为NULL。
logoff_lread	numeric	暂不支持，值为NULL。
logoff_pread	numeric	暂不支持，值为NULL。
logoff_lwrite	numeric	暂不支持，值为NULL。
logoff_dlock	character varying(40)	暂不支持，值为NULL。
comment_text	character varying(4000)	暂不支持，值为NULL。
sessionid	numeric	暂不支持，值为NULL。
entryid	numeric	暂不支持，值为NULL。
statementid	numeric	暂不支持，值为NULL。
returncode	numeric	暂不支持，值为NULL。
priv_used	character varying(40)	暂不支持，值为NULL。
client_id	character varying(128)	暂不支持，值为NULL。
econtext_id	character varying(64)	暂不支持，值为NULL。

名称	类型	描述
session_cpu	numeric	暂不支持，值为NULL。
extended_timestamp	timestamp(6) with time zone	创建审计跟踪条目的时间戳（创建条目的用户登录时间戳UTC（协调通用）中的审计会话时间）时区。
proxy_sessionid	numeric	暂不支持，值为NULL。
global_uid	character varying(32)	暂不支持，值为NULL。
instance_number	numeric	暂不支持，值为NULL。
os_process	character varying(16)	暂不支持，值为NULL。
transactionid	text	访问或修改对象的事务的事务标识符。 **说明** GaussDB的transactionid字段与A数据库中transactionid数据的类型保持一致。
scn	numeric	暂不支持，值为NULL。
sql_bind	nvarchar2(2000)	暂不支持，值为NULL。
sql_text	nvarchar2	查询的SQL文本。 **说明** GaussDB的sql_text字段为解析后的SQL描述语句，不完全与执行的SQL语句相同。
obj_edition_name	character varying(128)	暂不支持，值为NULL。
dbid	numeric	暂不支持，值为NULL。
rls_info	clob	暂不支持，值为NULL。
current_user	character varying(128)	暂不支持，值为NULL。

12.3.6 ADM_COL_COMMENTS

ADM_COL_COMMENTS视图显示数据库表中字段的注释信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-112 ADM_COL_COMMENTS 字段

名称	类型	描述
owner	character varying(128)	表的所有者。

名称	类型	描述
table_name	character varying(128)	表名。
column_name	character varying(128)	列名。
comments	text	注释。
origin_con_id	numeric	暂不支持，值为0。
schema	character varying(64)	列所属的名称空间的名称。

12.3.7 ADM_COL_PRIVS

ADM_COL_PRIVS视图显示所有的列权限授予信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-113 ADM_COL_PRIVS 字段

名称	类型	描述
grantor	character varying(128)	执行授权的用户名。
owner	character varying(128)	对象的所有者。
grantee	character varying(128)	被授予权限的用户或角色的名称。
table_schema	character varying(128)	对象的Schema。
table_name	character varying(128)	对象的名称。
column_name	character varying(128)	列的名称。
privilege	character varying(40)	列的权限。
grantable	character varying(3)	是否授予特权。 ● YES：授予。 ● NO：不授予。
common	character varying(3)	暂不支持，值为NULL。
inherited	character varying(3)	暂不支持，值为NULL。

12.3.8 ADM_COLL_TYPES

ADM_COLL_TYPES视图显示所有集合类型的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-114 ADM_COLL_TYPES 字段

名称	类型	描述
owner	character varying(128)	集合的所有者。
type_name	character varying(128)	集合的名称。
coll_type	character varying(128)	集合的描述。
upper_bound	numeric	暂不支持，值为NULL。
elem_type_mod	character varying(7)	暂不支持，值为NULL。
elem_type_owner	character varying(128)	集合基于的元素类型的所有者。该值主要用于用户定义的类型。
elem_type_name	character varying(128)	集合所依据的数据类型或用户定义类型的名称。
length	numeric	暂不支持，值为NULL。
precision	numeric	暂不支持，值为NULL。
scale	numeric	暂不支持，值为NULL。
character_set_name	character varying(44)	暂不支持，值为NULL。
elem_storage	character varying(7)	暂不支持，值为NULL。
nulls_stored	character varying(3)	暂不支持，值为NULL。
char_used	character varying(1)	暂不支持，值为NULL。

12.3.9 ADM_CONS_COLUMNS

ADM_CONS_COLUMNS视图显示数据库表中约束的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-115 ADM_CONS_COLUMNS 字段

名称	类型	描述
owner	character varying(64)	约束创建者。

名称	类型	描述
constraint_name	character varying(64)	约束名。
table_name	character varying(64)	约束相关的表名。
column_name	character varying(64)	约束相关的列名。
position	smallint	表中列的位置。

12.3.10 ADM_CONSTRAINTS

ADM_CONSTRAINTS视图显示数据库表中约束的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-116 ADM_CONSTRAINTS 字段

名称	类型	描述
owner	character varying(64)	约束创建者。
constraint_name	character varying(64)	约束名。
constraint_type	text	约束类型： ● c：表示检查约束。 ● f：表示外键约束。 ● p：表示主键约束。 ● u：表示唯一约束。
table_name	character varying(64)	约束相关的表名。
index_owner	character varying(64)	约束相关的索引的所有者（只针对唯一约束和主键约束）。
index_name	character varying(64)	约束相关的索引名（只针对唯一约束和主键约束）。
status	character varying(8)	约束的状态。
generated	character varying(14)	暂不支持，值为NULL。
search_condition	text	暂不支持，值为NULL。
search_condition_vc	character varying(4000)	暂不支持，值为NULL。
r_owner	character varying(128)	暂不支持，值为NULL。
r_constraint_name	character varying(128)	暂不支持，值为NULL。

名称	类型	描述
delete_rule	character varying(9)	暂不支持，值为NULL。
con_deferrable	character varying(14)	暂不支持，值为NULL。
deferred	character varying(9)	暂不支持，值为NULL。
validated	character varying(13)	暂不支持，值为NULL。
bad	character varying(3)	暂不支持，值为NULL。
rely	character varying(4)	暂不支持，值为NULL。
last_change	timestamp(0) without time zone	暂不支持，值为NULL。
invalid	character varying(7)	暂不支持，值为NULL。
view_related	character varying(14)	暂不支持，值为NULL。
origin_con_id	character varying(256)	暂不支持，值为NULL。

12.3.11 ADM_DATA_FILES

ADM_DATA_FILES视图显示关于数据库文件的描述。默认只有系统管理员权限才可以访问此视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-117 ADM_DATA_FILES 字段

名称	类型	描述
tablespace_name	name	文件所属的表空间的名称。
bytes	double precision	文件的字节长度。

12.3.12 ADM_DEPENDENCIES

ADM_DEPENDENCIES视图显示数据库中的类型、表、视图、存储过程、函数、触发器之间的依赖关系。默认只有系统管理员权限才可以访问，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-118 ADM_DEPENDENCIES 字段

名称	类型	描述
owner	name	对象的所有者。
name	name	对象的名称。
type	character varying(18)	对象的类型。

名称	类型	描述
referenced_owner	name	被引用对象的所有者。
referenced_name	name	被引用对象的名称。
referenced_type	character varying(18)	被引用对象的类型。
referenced_link_name	character varying(128)	暂不支持，值为NULL。
dependency_type	character varying(4)	暂不支持，值为NULL。

12.3.13 ADM_DIRECTORIES

ADM_DIRECTORIES显示数据库中的所有目录对象。默认只有系统管理员权限才可以访问此视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-119 ADM_DIRECTORIES 字段

名称	类型	描述
owner	oid	目录的所有者。
directory_name	name	目录名称。
directory_path	text	目录的操作系统路径名。
origin_con_id	character varying(256)	暂不支持，值为NULL。

12.3.14 ADM_HIST_SNAPSHOT

ADM_HIST_SNAPSHOT视图记录当前系统中存储的WDR快照数据信息。默认只有系统管理员权限才可以访问，普通用户需要授权才可以访问。该视图位于PG_CATALOG和SYS Schema下。该视图只有在GUC参数enable_wdr_snapshot为on时才可以访问。访问PG_CATALOG.ADM_HIST_SNAPSHOT和SYS.ADM_HIST_SNAPSHOT视图除需要本视图访问权限外，还需要snapshot schema，snapshot table和tables_snap_timestamp表的访问权限。

表 12-120 ADM_HIST_SNAPSHOT 字段

名称	类型	描述
snap_id	bigint	唯一快照ID。
dbid	oid	快照的数据库ID。
instance_number	oid	暂不支持，同DBID。

名称	类型	描述
startup_time	timestamp(3) without time zone	实例启动时间。
begin_interval_time	timestamp without time zone	快照间隔开始的时间（即上次快照的结束时间）。
end_interval_time	timestamp without time zone	快照间隔结束的时间。拍摄快照的实际时间（即本次快照的结束时间）。
flush_elapsed	interval	执行快照所花费的时间。
snap_level	numeric	暂不支持，值为NULL。
error_count	numeric	暂不支持，值为NULL。
snap_flag	numeric	暂不支持，值为NULL。
snap_timezone	interval day to second(0)	快照时区，表示为与UTC（协调世界时）时区的偏移量。
begin_interval_time_tz	timestamp with time zone	快照间隔开始的时间（即上次快照的结束时间），带时区。
end_interval_time_tz	timestamp with time zone	快照间隔结束的时间。拍摄快照的实际时间（即本次快照的结束时间），带时区。
con_id	numeric	暂不支持，值为0。

12.3.15 ADM_HIST_SQL_PLAN

ADM_HIST_SQL_PLAN视图描述当前用户通过执行EXPLAIN PLAN收集到的计划信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在PG_CATALOG和SYS Schema下。

表 12-121 ADM_HIST_SQL_PLAN 字段

名称	类型	描述
dbid	text	数据库ID。
sql_id	character varying(30)	表示插入该条数据的会话，由服务线程启动时间戳和服务线程ID组成。受非空约束限制。
id	integer	分配给执行计划中的每个步骤编号。
plan_hash_value	bigint	查询标识。

名称	类型	描述
operation	character varying(30)	操作描述。
options	character varying(255)	操作选项。
object_name	name	操作对应的对象名，来自于用户定义。
object_node	character varying(128)	暂不支持，值为NULL。
object#	numeric	暂不支持，值为NULL。
object_owner	name	表或索引的对象编号。
object_alias	character varying(261)	暂不支持，值为NULL。
object_type	character varying(30)	对象类型。
optimizer	character varying(20)	暂不支持，值为NULL。
parent_id	numeric	暂不支持，值为NULL。
depth	numeric	暂不支持，值为NULL。
position	numeric	暂不支持，值为NULL。
search_columns	numeric	暂不支持，值为NULL。
cost	double precision	优化器对算子估算的执行代价。
cardinality	double precision	优化器对算子估算访问表记录基数大小。
bytes	numeric	暂不支持，值为NULL。
other_tag	character varying(35)	暂不支持，值为NULL。
partition_start	character varying(64)	暂不支持，值为NULL。
partition_stop	character varying(64)	暂不支持，值为NULL。
partition_id	numeric	暂不支持，值为NULL。
other	character varying(4000)	暂不支持，值为NULL。

名称	类型	描述
distribution	character varying(20)	暂不支持，值为NULL。
cpu_cost	numeric	暂不支持，值为NULL。
io_cost	numeric	暂不支持，值为NULL。
temp_space	numeric	暂不支持，值为NULL。
access_predicates	character varying(4000)	暂不支持，值为NULL。
filter_predicates	character varying(4000)	暂不支持，值为NULL。
projection	character varying(4000)	操作输出的列信息。
time	numeric	暂不支持，值为NULL。
qblock_name	character varying(128)	暂不支持，值为NULL。
remarks	character varying(4000)	暂不支持，值为NULL。
timestamp	timestamp(0) without time zone	暂不支持，值为NULL。
other_xml	clob	暂不支持，值为NULL。
con_dbid	text	容器数据库ID，目前与dbid取值相同。
con_id	numeric	容器ID，目前不支持容器，值为为0。

12.3.16 ADM_HIST_SQLSTAT

ADM_HIST_SQLSTAT视图描述当前节点的执行语句的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在PG_CATALOG和SYS schema下。

WDR Snapshot启动（即GUC参数enable_wdr_snapshot为on时）后，用户可以查看此视图中的数据。

表 12-122 ADM_HIST_SQLSTAT 字段

名称	类型	描述
instance_number	integer	快照的实例编号。
sql_id	bigint	查询标识。
plan_hash_value	integer	归一化SQL ID。
module	integer	包含第一次解析SQL语句时正在执行的模块的名称。
elapsed_time_delta	bigint	有效的DB时间花费，多线程将累加（单位：微秒）。
cpu_time_delta	bigint	CPU的时间消耗（单位：微秒）。
executions_delta	bigint	自从它被带入库缓存以来在此对象上发生的执行次数增量。
iowait_delta	bigint	I/O的时间花费（单位：微秒）。
apwait_delta	integer	应用程序等待时间的Delta值。
rows_processed_delta	bigint	SELECT返回的结果集行数。
snap_id	bigint	唯一快照ID。
parsing_schema_name	character varying	暂不支持，值为NULL。
disk_reads_delta	bigint	暂不支持，值为NULL。
buffer_reads_delta	bigint	暂不支持，值为NULL。
clwait_delta	bigint	暂不支持，值为NULL。

12.3.17 ADM_HIST_SQLTEXT

ADM_HIST_SQLTEXT视图描述当前节点的执行语句的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在PG_CATALOG和SYS Schema下。

WDR Snapshot启动（即GUC参数enable_wdr_snapshot为on时）后，用户可以查看此视图中的数据。

表 12-123 ADM_HIST_SQLTEXT 字段

名称	类型	描述
dbid	integer	数据库ID。
sql_id	bigint	查询标识。
sql_text	clob	查询对应文本。
command_type	integer	暂不支持，值为0。
con_dbid	integer	容器数据库ID，目前与dbid取值相同。
con_id	integer	容器ID，目前不支持容器，值为0。

12.3.18 ADM_IND_COLUMNS

ADM_IND_COLUMNS视图显示数据库中索引字段的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-124 ADM_IND_COLUMNS 字段

名称	类型	描述
index_owner	character varying(64)	索引的所有者。
index_name	character varying(64)	索引名。
table_owner	character varying(64)	表的所有者。
table_name	character varying(64)	表名。
column_name	name	列名。
column_position	smallint	索引中列的位置。
column_length	numeric	列的长度，如果列是变长类型，该字段取值为NULL。
char_length	numeric	列的最大字节长度。
descend	character varying	指示列是降序（DESC）排序还是升序（ASC）排序。
collated_column_id	numeric	暂不支持，值为NULL。

12.3.19 ADM_IND_EXPRESSIONS

ADM_IND_EXPRESSIONS视图显示数据库中表达式索引的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-125 ADM_IND_EXPRESSIONS 字段

名称	类型	描述
table_owner	character varying(64)	表的所有者。
table_name	character varying(64)	表名。
index_owner	character varying(64)	索引的所有者。
index_name	character varying(64)	索引名。
column_expression	text	定义列的基于函数的索引表达式。
column_position	smallint	索引中列的位置。

12.3.20 ADM_IND_PARTITIONS

ADM_IND_PARTITIONS视图显示数据库中所有一级分区表Local索引的索引分区信息。数据库中每个一级分区表的Local索引的索引分区（如果存在的话）都会在ADM_IND_PARTITIONS里有一行记录。默认只有系统管理员权限才可以访问，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-126 ADM_IND_PARTITIONS 字段

名称	类型	描述
index_owner	character varying(64)	索引分区所属分区表索引的所有者的名称。
index_name	character varying(64)	索引分区所属分区表索引的名称。
partition_name	character varying(64)	索引分区的名称。
def_tablespace_name	name	索引分区的表空间名称。

名称	类型	描述
high_value	text	索引分区所对应分区的上边界。 ● 对于范围分区和间隔分区，显示各分区的上边界值。 ● 对于列表分区，显示各分区的取值列表。 ● 对于哈希分区，显示各分区的编号。
index_partition_usable	boolean	索引分区是否可用。 ● t（true）：表示可用。 ● f（false）：表示不可用。
schema	character varying(64)	索引分区所属分区表索引的模式。
high_value_length	integer	索引分区所对应分区的边界的字符长度。
composite	character varying(3)	指示索引是否属于二级分区表上的本地索引，该表不存储二级分区信息，所以该值为NO。
subpartition_count	numeric	分区中的二级分区数，该表不存储二级分区信息，所以该值为0。
partition_position	numeric	索引分区在索引中的位置。
status	character varying(8)	指示索引分区是否可用。
tablespace_name	name	分区所在表空间的名称。
pct_free	numeric	块中最小可用空间百分比。
ini_trans	numeric	初始事务数，默认值为4，非USTORE分区表时为NULL。
max_trans	numeric	最大事务数，默认值为128，非USTORE分区表时为NULL。
initial_extent	numeric	暂不支持，值为NULL。
next_extent	numeric	暂不支持，值为NULL。
min_extent	numeric	暂不支持，值为NULL。

名称	类型	描述
max_extent	numeric	暂不支持，值为NULL。
max_size	numeric	暂不支持，值为NULL。
pct_increase	numeric	暂不支持，值为NULL。
freelists	numeric	暂不支持，值为NULL。
freelist_groups	numeric	暂不支持，值为NULL。
logging	character varying(7)	指示是否记录对索引的更改。
compression	character varying(13)	指示分区索引是否启用索引压缩。
blevel	numeric	暂不支持，值为NULL。
leaf_blocks	numeric	暂不支持，值为NULL。
distinct_keys	numeric	暂不支持，值为NULL。
avg_leaf_blocks_per_key	numeric	暂不支持，值为NULL。
avg_data_blocks_per_key	numeric	暂不支持，值为NULL。
clustering_factor	numeric	根据索引的值指示表中行的顺序。需要通过执行analyze进行统计。
num_rows	numeric	分区中的行数统计值。需要通过执行vacuum进行统计。
sample_size	numeric	暂不支持，值为NULL。
last_analyzed	timestamp with time zone	最近分析此分区的日期。
buffer_pool	character varying(7)	分区的实际缓冲池。
flash_cache	character varying(7)	暂不支持，值为NULL。
cell_flash_cache	character varying(7)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。
pct_direct_access	numeric	暂不支持，值为NULL。
global_stats	character varying(3)	暂不支持，值为NULL。
domidx_opstatus	character varying(6)	暂不支持，值为NULL。
parameters	character varying(1000)	暂不支持，值为NULL。
interval	character varying(3)	指示分区是否在间隔分区表的间隔节中。

名称	类型	描述
segment_created	character varying(3)	指示索引分区段是否已创建。
orphaned_entries	character varying(3)	暂不支持，值为NULL。

12.3.21 ADM_IND_SUBPARTITIONS

ADM_IND_SUBPARTITIONS视图显示数据库中所有二级分区表Local索引的索引分区信息（不包含分区表全局索引）。数据库中每个二级分区表的Local索引的索引分区（如果存在的话）都会在ADM_IND_SUBPARTITIONS里有一行记录。默认只有系统管理员权限才可以访问，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-127 ADM_IND_SUBPARTITIONS 字段

名称	类型	描述
index_owner	character varying(64)	索引分区所属分区表索引的所有者的名称。
index_name	character varying(64)	索引分区所属分区表索引的名称。
partition_name	character varying(64)	索引所在一级分区的名称。
subpartition_name	character varying(64)	索引所在二级分区的名称。
def_tablespace_name	name	索引分区的表空间名称。
high_value	text	索引分区所对应分区的边界值。 ● 对于范围分区和间隔分区，显示各分区的上边界值。 ● 对于列表分区，显示各分区的取值列表。 ● 对于哈希分区，显示各分区的编号。
index_partition_usable	boolean	索引分区是否可用。 ● t（true）：表示可用。 ● f（false）：表示不可用。
schema	character varying(64)	索引分区所属分区表索引的模式。

名称	类型	描述
high_value_length	integer	索引分区所对应分区的边界的字符长度。
partition_position	numeric	索引分区在索引中的位置。
subpartition_position	numeric	二级分区在分区中的位置。
status	character varying(8)	指示索引分区是否可用。
tablespace_name	name	索引分区的表空间名称。
pct_free	numeric	块中最小可用空间百分比。
ini_trans	numeric	初始事务数，默认值为4，非USTORE分区表时为NULL。
max_trans	numeric	最大事务数，默认值为128，非USTORE分区表时为NULL。
initial_extent	numeric	暂不支持，值为NULL。
next_extent	numeric	暂不支持，值为NULL。
min_extent	numeric	暂不支持，值为NULL。
max_extent	numeric	暂不支持，值为NULL。
max_size	numeric	暂不支持，值为NULL。
pct_increase	numeric	暂不支持，值为NULL。
freelists	numeric	暂不支持，值为NULL。
freelist_groups	numeric	暂不支持，值为NULL。
logging	character varying(7)	指示是否记录对索引的更改。
compression	character varying(13)	用于二级分区的压缩类型。
blevel	numeric	暂不支持，值为NULL。
leaf_blocks	numeric	暂不支持，值为NULL。
distinct_keys	numeric	暂不支持，值为NULL。
avg_leaf_blocks_per_key	numeric	暂不支持，值为NULL。
avg_data_blocks_per_key	numeric	暂不支持，值为NULL。

名称	类型	描述
clustering_factor	numeric	根据索引的值指示表中行的顺序。需要通过执行analyze进行统计。
num_rows	numeric	分区中的行数统计值。需要通过执行vacuum进行统计。
sample_size	numeric	暂不支持，值为NULL。
last_analyzed	timestamp with time zone	最近分析此分区的日期。
buffer_pool	character varying(7)	二级分区的缓冲池。
flash_cache	character varying(7)	暂不支持，值为NULL。
cell_flash_cache	character varying(7)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。
global_stats	character varying(3)	暂不支持，值为NULL。
interval	character varying(3)	指示分区是否在间隔分区表的间隔节中。
segment_created	character varying(3)	指示索引分区段是否已创建。
domidx_opstatus	character varying(6)	暂不支持，值为NULL。
parameters	character varying(1000)	暂不支持，值为NULL。

12.3.22 ADM_INDEXES

ADM_INDEXES视图显示数据库中所有索引的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-128 ADM_INDEXES 字段

名称	类型	描述
owner	character varying(64)	索引的所有者。
index_name	character varying(64)	索引名称。
table_name	character varying(64)	索引对应的表名。

名称	类型	描述
uniqueness	text	该索引是否为唯一索引。 ● UNIQUE：唯一索引。 ● NONUNIQUE：非唯一索引。
partitioned	character(3)	该索引是否具有分区表的性质。 ● Yes：索引具有分区表的性质。 ● No：索引不具有分区表的性质。
generated	character varying(1)	该索引的名称是否为系统生成。 ● y：索引名称为系统生成。 ● n：索引名称非系统生成。
index_type	character varying(27)	索引类型。 ● NORMAL：索引属性都是简单的引用，表达式树为空。 ● FUNCTION-BASED NORMAL：存在表达式树用于非简单字段引用的索引属性。
table_owner	character varying(128)	索引对象的所有者。
table_type	character(11)	索引对象的类型。 ● TABLE：索引对象为表类型。
tablespace_name	character varying(30)	包含索引的表空间名称。
status	character varying(8)	非分区索引状态。 ● VALID：非分区索引可以用于查询。 ● UNUSABLE：非分区索引不可用。 ● N/A：索引具有分区表性质。
compression	character varying(13)	暂不支持，值为NULL。
prefix_length	numeric	暂不支持，值为NULL。
ini_trans	numeric	暂不支持，值为NULL。

名称	类型	描述
max_trans	numeric	暂不支持，值为NULL。
initial_extent	numeric	暂不支持，值为NULL。
next_extent	numeric	暂不支持，值为NULL。
min_extents	numeric	暂不支持，值为NULL。
max_extents	numeric	暂不支持，值为NULL。
pct_increase	numeric	暂不支持，值为NULL。
pct_threshold	numeric	暂不支持，值为NULL。
include_column	numeric	暂不支持，值为NULL。
freelists	numeric	暂不支持，值为NULL。
freelist_groups	numeric	暂不支持，值为NULL。
pct_free	numeric	暂不支持，值为NULL。
logging	character varying(3)	暂不支持，值为NULL。
blevel	numeric	暂不支持，值为NULL。
leaf_blocks	numeric	暂不支持，值为NULL。
distinct_keys	numeric	暂不支持，值为NULL。
avg_leaf_blocks_per_key	numeric	暂不支持，值为NULL。
avg_data_blocks_per_key	numeric	暂不支持，值为NULL。
clustering_factor	numeric	暂不支持，值为NULL。
num_rows	numeric	暂不支持，值为NULL。
sample_size	numeric	暂不支持，值为NULL。
last_analyzed	timestamp(0) without time zone	暂不支持，值为NULL。
degree	character varying(40)	暂不支持，值为NULL。
instances	character varying(40)	暂不支持，值为NULL。
temporary	character varying(1)	暂不支持，值为NULL。
secondary	character varying(1)	暂不支持，值为NULL。
buffer_pool	character varying(7)	暂不支持，值为NULL。
flash_cache	character varying(7)	暂不支持，值为NULL。
cell_flash_cache	character varying(7)	暂不支持，值为NULL。

名称	类型	描述
user_stats	character varying(3)	暂不支持，值为NULL。
duration	character varying(15)	暂不支持，值为NULL。
pct_direct_access	numeric	暂不支持，值为NULL。
ityp_owner	character varying(128)	暂不支持，值为NULL。
ityp_name	character varying(128)	暂不支持，值为NULL。
parameters	character varying(1000)	暂不支持，值为NULL。
global_stats	character varying(3)	暂不支持，值为NULL。
domidx_status	character varying(12)	暂不支持，值为NULL。
domidx_opstatus	character varying(6)	暂不支持，值为NULL。
funcidx_status	character varying(8)	暂不支持，值为NULL。
join_index	character varying(3)	暂不支持，值为NULL。
iot_redundant_pkey_elim	character varying(3)	暂不支持，值为NULL。
dropped	character varying(3)	暂不支持，值为NULL。
visibility	character varying(9)	暂不支持，值为NULL。
domidx_management	character varying(14)	暂不支持，值为NULL。
segment_created	character varying(3)	暂不支持，值为NULL。
orphaned_entries	character varying(3)	暂不支持，值为NULL。
indexing	character varying(7)	暂不支持，值为NULL。
auto	character varying(3)	暂不支持，值为NULL。

12.3.23 ADM_OBJECTS

ADM_OBJECTS视图显示数据库中所有数据库对象的信息。默认只有系统管理员权限才可以访问，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-129 ADM_OBJECTS 字段

名称	类型	描述
owner	name	对象的所有者。
object_name	name	对象的名称。
object_id	oid	对象的OID。

名称	类型	描述
object_type	name	对象的类型。例如table，schema，index等。
namespace	oid	对象所在的命名空间。
temporary	character(1)	对象是否为临时对象。
status	character varying(7)	对象的状态。 ● valid：有效。 ● invalid：失效。
subobject_name	name	对象的子对象名称。
generated	character(1)	对象名称是否是系统生成。
created	timestamp with time zone	对象的创建时间。
last_ddl_time	timestamp with time zone	对象的最后修改时间。
default_collation	character varying(100)	对象的默认排序规则。
data_object_id	numeric	暂不支持，值为NULL。
timestamp	character varying(19)	暂不支持，值为NULL。
secondary	character varying(1)	暂不支持，值为NULL。
edition_name	character varying(128)	暂不支持，值为NULL。
sharing	character varying(18)	暂不支持，值为NULL。
editionable	character varying(1)	暂不支持，值为NULL。
oracle_maintained	character varying(1)	暂不支持，值为NULL。
application	character varying(1)	暂不支持，值为NULL。
duplicated	character varying(1)	暂不支持，值为NULL。

名称	类型	描述
sharded	character varying(1)	暂不支持，值为NULL。
created_appid	numeric	暂不支持，值为NULL。
modified_appid	numeric	暂不支持，值为NULL。
created_vsnid	numeric	暂不支持，值为NULL。
modified_vsnid	numeric	暂不支持，值为NULL。

须知

created和last_ddl_time支持的范围参见**PG_OBJECT**中的记录范围。

12.3.24 ADM_PART_COL_STATISTICS

ADM_PART_COL_STATISTICS视图显示数据库中所有表分区的列统计信息和直方图信息。默认只有系统管理员权限才可以访问，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-130 ADM_PART_COL_STATISTICS 字段

名称	类型	描述
owner	character varying(128)	分区表的所有者。
table_name	character varying(128)	表名。
partition_name	character varying(128)	表分区名称。
column_name	character varying(4000)	列名。
num_distinct	numeric	暂不支持，值为NULL。
low_value	raw	暂不支持，值为NULL。
high_value	raw	暂不支持，值为NULL。
density	numeric	暂不支持，值为NULL。
num_nulls	numeric	暂不支持，值为NULL。
num_buckets	numeric	暂不支持，值为NULL。
sample_size	numeric	暂不支持，值为NULL。
last_analyzed	date	暂不支持，值为NULL。
global_stats	character varying(3)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。

名称	类型	描述
notes	character varying(63)	暂不支持，值为NULL。
avg_col_len	numeric	暂不支持，值为NULL。
histogram	character varying(15)	暂不支持，值为NULL。
schema	character varying(64)	列所属的名称空间的名称。

12.3.25 ADM_PART_INDEXES

ADM_PART_INDEXES视图存储数据库中所有分区表索引的信息（不包含分区表全局索引）。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS schema下。

表 12-131 ADM_PART_INDEXES 字段

名称	类型	描述
def_tablespace_name	name	分区表索引的表空间名称。
index_owner	character varying(64)	分区表索引的所有者名称。
index_name	character varying(64)	分区表索引的名称。
partition_count	bigint	分区表索引的索引分区的个数。
partitioning_key_count	integer	分区表的分区键个数。
partitioning_type	text	分区表的分区策略。 **说明** 当前分区表策略支持范围见**CREATE TABLE PARTITION**。
schema	character varying(64)	分区表索引所属模式的名称。
table_name	character varying(64)	分区表索引所属的分区表名称。
subpartitioning_type	text	二级分区表的分区策略。如果分区表是一级分区表，则显示NONE。 **说明** 当前二级分区表策略支持范围见**CREATE TABLE SUBPARTITION**。
def_subpartition_count	integer	默认创建二级分区的个数。二级分区表为1，一级分区表为0。

名称	类型	描述
subpartitioning_key_count	integer	分区表二级分区键的个数。

12.3.26 ADM_PART_TABLES

ADM_PART_TABLES视图存储数据库中所有分区表的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS schema下。

表 12-132 ADM_PART_TABLES 字段

名称	类型	描述
table_owner	character varying(64)	分区表的所有者名称。
table_name	character varying(64)	分区表的名称。
partitioning_type	text	分区表的分区策略。 **说明** 当前分区表策略支持范围见 **CREATE TABLE PARTITION**。
partition_count	bigint	分区表的分区个数。
partitioning_key_count	integer	分区表的分区键个数。
def_tablespace_name	name	分区表的表空间名称。
schema	character varying(64)	分区表的模式。
subpartitioning_type	text	二级分区表的分区策略。如果分区表是一级分区表，则显示NONE。 **说明** 当前二级分区表策略支持范围见 **CREATE TABLE SUBPARTITION**。
def_subpartition_count	integer	默认创建二级分区的个数，二级分区表为1，一级分区表为0。
subpartitioning_key_count	integer	分区表二级分区键的个数。
status	character varying(8)	暂不支持，值为valid。
def_pct_free	numeric	添加分区时使用的PCTFREE默认值。

名称	类型	描述
def_pct_used	numeric	暂不支持，值为NULL。
def_ini_trans	numeric	添加分区时使用的INITRANS默认值。
def_max_trans	numeric	添加分区时使用的MAXTRANS默认值。
def_initial_extent	character varying(40)	暂不支持，值为NULL。
def_next_extent	character varying(40)	暂不支持，值为NULL。
def_min_extents	character varying(40)	暂不支持，值为NULL。
def_max_extents	character varying(40)	暂不支持，值为NULL。
def_max_size	character varying(40)	暂不支持，值为NULL。
def_pct_increase	character varying(40)	暂不支持，值为NULL。
def_freelists	numeric	暂不支持，值为NULL。
def_freelist_groups	numeric	暂不支持，值为NULL。
def_logging	character varying(7)	暂不支持，值为NULL。
def_compression	character varying(8)	添加分区时使用的默认压缩： • NONE • ENABLED • DISABLED
def_compress_for	character varying(30)	添加分区时使用的默认压缩。 **说明** 可用的压缩方法和压缩级别见 **WITH ({ storage_paramet...**。
def_buffer_pool	character varying(7)	暂不支持，值为DEFAULT。
def_flash_cache	character varying(7)	暂不支持，值为NULL。
def_cell_flash_cache	character varying(7)	暂不支持，值为NULL。
ref_ptn_constraint_name	character varying(128)	暂不支持，值为NULL。
interval	character varying(1000)	区间值字符串。
autolist	character varying(3)	暂不支持，值为NO。

名称	类型	描述
interval_subpartition	character varying(1000)	暂不支持，值为NULL。
autolist_subpartition	character varying(3)	暂不支持，值为NO。
is_nested	character varying(3)	暂不支持，值为NO。
def_segment_creation	character varying(4)	暂不支持段页式设置，当启用segment时，值为YES。
def_indexing	character varying(3)	暂不支持，值为ON。
def_inmemory	character varying(8)	暂不支持，值为NONE。
def_inmemory_priority	character varying(8)	暂不支持，值为NULL。
def_inmemory_distribute	character varying(15)	暂不支持，值为NULL。
def_inmemory_compression	character varying(17)	暂不支持，值为NULL。
def_inmemory_duplicate	character varying(13)	暂不支持，值为NULL。
def_read_only	character varying(3)	暂不支持，值为NO。
def_cellmemory	character varying(24)	暂不支持，值为NULL。
def_inmemory_service	character varying(12)	暂不支持，值为NULL。
def_inmemory_service_name	character varying(1000)	暂不支持，值为NULL。

12.3.27 ADM_PROCEDURES

ADM_PROCEDURES视图显示数据库中所有存储过程、函数和触发器的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-133 ADM_PROCEDURES 字段

名称	类型	描述
owner	character varying(64)	存储过程、或函数、或触发器、或包的所有者。
object_name	character varying(64)	存储过程、或函数、或触发器的名称，若为包中函数或存储过程，则为包名。

名称	类型	描述
procedure_name	character varying(128)	若object_name为包名，则为包中函数或存储过程名称，其余为空。
object_id	oid	存储过程、或函数、或触发器、或包的oid。
subprogram_id	numeric	如果为包中函数或存储过程，则表示在包中的位置，其余为空。
overload	character varying(40)	表示该函数是该名称的第n个重载函数。
object_type	character varying(13)	对象的类型名。
aggregate	character varying(3)	表示是否为聚合函数： ● YES：表示是。 ● NO：表示不是。
pipelined	character varying(3)	暂不支持，值为NO。
impltypeowner	character varying(128)	实现类型的所有者。
impltypename	character varying(128)	实现类型的名称。
parallel	character varying(3)	暂不支持，值为NO。
interface	character varying(3)	暂不支持，值为NO。
deterministic	character varying(3)	暂不支持，值为NO。
authid	character varying(12)	表示是使用创建者权限还是调用者权限： ● DEFINER：表示使用创建者权限。 ● CURRENT_USER：表示使用调用者权限。 因该字段与保留关键字冲突，调用该字段需加视图名。
result_cache	character varying(3)	暂不支持，值为NULL。
origin_con_id	character varying(256)	暂不支持，值为0。
polymorphic	character varying(5)	暂不支持，值为NULL。
argument_number	smallint	存储过程入参个数。

12.3.28 ADM_RECYCLEBIN

ADM_RECYCLEBIN显示所有回收站的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-134 ADM_RECYCLEBIN 字段

名称	类型	描述
owner	character varying(128)	对象的原始所有者名称。
object_name	character varying(128)	对象的新名称。
original_name	character varying(128)	对象的原始名称。
operation	character varying(9)	对对象执行的操作： ● 删除：对象已丢弃（对象不再需要）。 ● 清空：对象被清空。
type	character varying(25)	对象的类型。
ts_name	character varying(30)	对象所属的表空间名称
createtime	character varying(19)	创建对象的时间戳。
droptime	character varying(19)	删除对象的时间戳。
dropscn	numeric	将对象移动到回收站的事务的系统更改编号(SCN)。
partition_name	character varying(128)	已删除的分区的名称。
can_undrop	character varying(3)	对象是否可以闪回。
can_purge	character varying(3)	对象是否可以清除。
related	numeric	父对象的对象编号。

名称	类型	描述
base_object	numeric	基对象的对象编号。
purge_object	numeric	被清除的对象的对象编号。
space	numeric	对象使用的块数。

12.3.29 ADM_ROLE_PRIVS

ADM_ROLE_PRIVS视图显示授予所有用户和角色的角色的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS schema下。

表 12-135 ADM_ROLE_PRIVS 字段

名称	类型	描述
grantee	character varying(128)	被授予权限的用户或角色名称。
granted_role	character varying(128)	被授予的角色名称。
admin_option	character varying(3)	该授权是否包含ADMIN选项。 ● YES：包含ADMIN选项。 ● NO：不包含ADMIN选项。
delegate_option	character varying(3)	暂不支持，值为NULL。
default_role	character varying(3)	暂不支持，值为NULL。
os_granted	character varying(3)	暂不支持，值为NULL。
common	character varying(3)	暂不支持，值为NULL。
inherited	character varying(3)	暂不支持，值为NULL。

12.3.30 ADM_ROLES

ADM_ROLES视图显示数据库角色的相关信息。默认只有系统管理员权限才可以访问此系统视图。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-136 ADM_ROLES 字段

名称	类型	描述
role	character varying(128)	角色名称。
role_id	oid	角色ID号。
authentication_type	text	角色的身份验证机制。 ● password：需要密码验证。 ● NULL：不需要验证。
common	character varying(3)	暂不支持，值为NULL。
oracle_maintained	character varying(1)	暂不支持，值为NULL。
inherited	character varying(3)	暂不支持，值为NULL。
implicit	character varying(3)	暂不支持，值为NULL。
external_name	character varying(4000)	暂不支持，值为NULL。

12.3.31 ADM_SCHEDULER_JOB_ARGS

ADM_SCHEDULER_JOB_ARG视图显示数据库中所有任务的有关参数信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-137 ADM_SCHEDULER_JOB_ARGS 字段

名称	类型	描述
owner	character varying(128)	参数所属作业的拥有者。
job_name	character varying(128)	参数所属作业名。
argument_name	character varying(128)	参数名称。
argument_position	numeric	参数在参数列表中的位置。
argument_type	character varying(257)	参数的数据类型，可以是用户的自定义数据类型。
value	character varying(4000)	参数值。
anydata_value	character varying(4000)	暂不支持，值为NULL。

名称	类型	描述
out_argument	character varying(5)	保留字段，值为NULL。

12.3.32 ADM_SCHEDULER_JOBS

ADM_SCHEDULER_JOBS视图显示数据库中所有DBE_SCHEDULER定时任务的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-138 ADM_SCHEDULER_JOBS 字段

名称	类型	描述
owner	name	定时任务所有者。
job_name	text	定时任务名。
job_subname	character varying(128)	暂不支持，值为NULL。
job_style	text	定时任务行为模式。创建时指定，仅支持指定为“REGULAR”，不指定时为NULL。
job_creator	name	定时任务创建者。
client_id	character varying(65)	暂不支持，值为NULL。
global_uid	character varying(33)	暂不支持，值为NULL。
program_owner	character varying(4000)	定时任务引用的程序的所有者。
program_name	text	定时任务引用的程序的名称。
job_type	character varying(16)	定时任务内联程序类型，可用类型为： ● PLSQL_BLOCK：匿名存储过程块。 ● STORED_PROCEDURE：保存的存储过程。 ● EXTERNAL_SCRIPT：外部脚本。
job_action	text	定时任务的程序内容。
number_of_arguments	text	定时任务的参数个数。
schedule_owner	character varying(4000)	暂不支持，值为NULL。

名称	类型	描述
schedule_name	text	定时任务引用的调度的名称。
schedule_type	character varying(12)	暂不支持，值为NULL。
start_date	timestamp without time zone	定时任务的起始时间。
repeat_interval	text	定时任务的任务周期。
event_queue_owner	character varying(128)	暂不支持，值为NULL。
event_queue_name	character varying(128)	暂不支持，值为NULL。
event_queue_agent	character varying(523)	暂不支持，值为NULL。
event_condition	character varying(4000)	暂不支持，值为NULL。
event_rule	character varying(261)	暂不支持，值为NULL。
file_watcher_owner	character varying(261)	暂不支持，值为NULL。
file_watcher_name	character varying(261)	暂不支持，值为NULL。
end_date	timestamp without time zone	定时任务的失效时间。
job_class	text	定时任务所属的定时任务类的名称。
enabled	boolean	定时任务的启用状态。
auto_drop	text	定时任务的自动删除功能状态。
restart_on_recovery	character varying(5)	暂不支持，值为NULL。
restart_on_failure	character varying(5)	暂不支持，值为NULL。
state	"char"	定时任务的状态。
job_priority	numeric	暂不支持，值为NULL。
run_count	numeric	暂不支持，值为NULL。
uptime_run_count	numeric	暂不支持，值为NULL。
max_runs	numeric	暂不支持，值为NULL。
failure_count	smallint	定时任务失败次数统计。
uptime_failure_count	numeric	暂不支持，值为NULL。
max_failures	numeric	定时任务标记为破坏之前允许失败的最大次数。
retry_count	numeric	暂不支持，值为NULL。

名称	类型	描述
last_start_date	timestamp without time zone	定时任务上次拉起时间。
last_run_duration	interval day to second(6)	定时任务上次执行的时长。
next_run_date	timestamp without time zone	定时任务下次执行时间。
schedule_limit	interval day to second(0)	暂不支持，值为NULL。
max_run_duration	interval day to second(0)	暂不支持，值为NULL。
logging_level	character varying(11)	暂不支持，值为NULL。
store_output	character varying(5)	是否存储所有定时任务的输出信息。
stop_on_window_close	character varying(5)	暂不支持，值为NULL。
instance_stickiness	character varying(5)	暂不支持，值为NULL。
raise_events	character varying(4000)	暂不支持，值为NULL。
system	character varying(5)	暂不支持，值为NULL。
job_weight	numeric	暂不支持，值为NULL。
nls_env	character varying(4000)	暂不支持，值为NULL。
source	character varying(128)	暂不支持，值为NULL。
number_of_destinations	numeric	暂不支持，值为NULL。
destination_owner	character varying(261)	暂不支持，值为NULL。
destination	text	定时任务目标名称。
credential_owner	character varying(128)	暂不支持，值为NULL。
credential_name	text	定时任务证书名称。
instance_id	oid	当前数据库的OID。
deferred_drop	character varying(5)	暂不支持，值为NULL。
allow_runs_in_restricted_mode	character varying(5)	暂不支持，值为NULL。
comments	text	定时任务的备注。
flags	numeric	暂不支持，值为NULL。

名称	类型	描述
restartable	character varying(5)	暂不支持，值为NULL。
has_constraints	character varying(5)	暂不支持，值为NULL。
connect_credential_owner	character varying(128)	暂不支持，值为NULL。
connect_credential_name	character varying(128)	暂不支持，值为NULL。
fail_on_script_error	character varying(5)	暂不支持，值为NULL。

12.3.33 ADM_SCHEDULER_PROGRAM_ARGS

ADM_SCHEDULER_PROGRAM_ARG视图显示数据库中所有程序的有关参数信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-139 ADM_SCHEDULER_PROGRAM_ARGS 字段

名称	类型	描述
owner	character varying(128)	参数所属程序的拥有者。
program_name	character varying(128)	参数所属程序名。
argument_name	character varying(128)	参数名称。
argument_position	numeric	参数在参数列表中的位置。
argument_type	character varying(257)	参数的数据类型，可以是用户的自定义数据类型。
metadata_attribute	character varying(19)	暂不支持，值为NULL。
default_value	character varying(4000)	参数默认值。
default_anydata_value	character varying(4000)	暂不支持，值为NULL。
out_argument	character varying(5)	保留字段，值为NULL。

12.3.34 ADM_SCHEDULER_PROGRAMS

ADM_SCHEDULER_PROGRAMS视图显示数据库中所有可以调度的程序信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-140 ADM_SCHEDULER_PROGRAMS 字段

名称	类型	描述
owner	name	调度程序的所有者。
program_name	text	调度程序的名称。
program_type	character varying(16)	调度程序的类型，可用类型为： ● PLSQL_BLOCK：匿名存储过程块。 ● STORED_PROCEDURE：保存的存储过程。 ● EXTERNAL_SCRIPT：外部脚本。
program_action	text	调度程序执行的操作。
number_of_arguments	numeric	调度程序参数个数。
enabled	character varying(5)	调度程序是否启用。
comments	text	调度程序的备注。
detached	character varying(5)	暂不支持，值为NULL。
schedule_limit	interval day to second(0)	暂不支持，值为NULL。
priority	numeric	暂不支持，值为NULL。
weight	numeric	暂不支持，值为NULL。
max_runs	numeric	暂不支持，值为NULL。
max_failures	numeric	暂不支持，值为NULL。
max_run_duration	interval day to second(0)	暂不支持，值为NULL。
has_constraints	character varying(5)	暂不支持，值为NULL。
nls_env	character varying(4000)	暂不支持，值为NULL。

12.3.35 ADM_SCHEDULER_RUNNING_JOBS

ADM_SCHEDULER_RUNNING_JOBS视图显示数据库中所有正在执行的DBE_SCHEDULER定时任务的信息。默认只有系统管理员权限才可以访问此系统视

图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS schema下。

表 12-141 ADM_SCHEDULER_RUNNING_JOBS 字段

名称	类型	描述
owner	character varying(128)	定时任务所有者。
job_name	character varying(128)	定时任务名。
job_subname	character varying(128)	暂不支持，值为NULL。
job_style	character varying(17)	定时任务行为模式，创建时指定，仅支持指定为“REGULAR”，不指定时为NULL。
detached	character varying(5)	暂不支持，值为NULL。
session_id	numeric	执行该定时任务的会话ID。
slave_process_id	numeric	暂不支持，值为NULL。
slave_os_process_id	character varying(12)	执行该任务的进程号。
running_instance	numeric	暂不支持，值为NULL。
resource_consumer_group	character varying(32)	暂不支持，值为NULL。
elapsed_time	interval day to second(2)	定时任务本次已执行的时长。
cpu_used	interval day to second(2)	暂不支持，值为NULL。
destination_owner	character varying(261)	暂不支持，值为NULL。
destination	character varying(261)	定时任务目标名称。
credential_name	character varying(128)	定时任务证书名称。
credential_owner	character varying(128)	暂不支持，值为NULL。
log_id	numeric	暂不支持，值为NULL。

12.3.36 ADM_SEGMENTS

ADM_SEGMENTS视图显示数据库中所有段分配的存储空间。同时存在于PG_CATALOG和SYS Schema下。仅系统管理员可访问。

表 12-142 ADM_SEGMENTS 字段

名称	类型	描述
owner	character varying(128)	暂不支持，值为NULL。
segment_name	character varying(128)	暂不支持，值为NULL。
partition_name	character varying(128)	暂不支持，值为NULL。
segment_type	character varying(18)	暂不支持，值为NULL。
segment_subtype	character varying(10)	暂不支持，值为NULL。
tablespace_name	character varying(30)	暂不支持，值为NULL。
header_file	numeric	暂不支持，值为NULL。
header_block	numeric	暂不支持，值为NULL。
bytes	numeric	暂不支持，值为NULL。
blocks	numeric	暂不支持，值为NULL。
extents	numeric	暂不支持，值为NULL。
initial_extent	numeric	暂不支持，值为NULL。
next_extent	numeric	暂不支持，值为NULL。
min_extents	numeric	暂不支持，值为NULL。
max_extents	numeric	暂不支持，值为NULL。
max_size	numeric	暂不支持，值为NULL。
retention	character varying(7)	暂不支持，值为NULL。
minretention	numeric	暂不支持，值为NULL。
pct_increase	numeric	暂不支持，值为NULL。
freelists	numeric	暂不支持，值为NULL。
freelist_groups	numeric	暂不支持，值为NULL。
relative_fno	numeric	暂不支持，值为NULL。
buffer_pool	character varying(7)	暂不支持，值为NULL。
flash_cache	character varying(7)	暂不支持，值为NULL。
cell_flash_cache	character varying(7)	暂不支持，值为NULL。
inmemory	character varying(8)	暂不支持，值为NULL。
inmemory_priority	character varying(8)	暂不支持，值为NULL。
inmemory_distribute	character varying(15)	暂不支持，值为NULL。
inmemory_duplicate	character varying(13)	暂不支持，值为NULL。

名称	类型	描述
inmemory_compression	character varying(17)	暂不支持，值为NULL。
cellmemory	character varying(24)	暂不支持，值为NULL。

12.3.37 ADM_SEQUENCES

ADM_SEQUENCES视图显示数据库中所有序列的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-143 ADM_SEQUENCES 字段

名称	类型	描述
sequence_owner	character varying(64)	序列的所有者。
sequence_name	character varying(64)	序列名称。
min_value	int16	序列的最小值。
max_value	int16	序列的最大值。
increment_by	int16	序列的递增值。
last_number	int16	上一序列的值。
cache_size	int16	序列磁盘缓存大小。
cycle_flag	character(1)	序列是否是循环序列。取值范围： ● Y：是循环序列。 ● N：不是循环序列。

12.3.38 ADM_SOURCE

ADM_SOURCE视图显示数据库中所有存储过程、函数、触发器、包的定义信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-144 ADM_SOURCE 字段

名称	类型	描述
owner	name	对象的所有者。
name	name	对象名字。

名称	类型	描述
type	name	对象类型。取值范围：function、package、package body、procedure、trigger。
line	numeric	此行在定义信息中的行号。
text	text	存储对象的文本来源。
origin_con_id	character varying(256)	暂不支持，值为0。

12.3.39 ADM_SUBPART_COL_STATISTICS

ADM_SUBPART_COL_STATISTICS视图显示数据库中所有子分区的列统计信息和直方图信息。默认只有系统管理员权限才可以访问，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-145 ADM_SUBPART_COL_STATISTICS 字段

名称	类型	描述
owner	character varying(128)	表的所有者。
table_name	character varying(128)	表名。
subpartition_name	character varying(128)	子分区名称。
column_name	character varying(4000)	列名。
num_distinct	numeric	暂不支持，值为NULL。
low_value	raw	暂不支持，值为NULL。
high_value	raw	暂不支持，值为NULL。
density	numeric	暂不支持，值为NULL。
num_nulls	numeric	暂不支持，值为NULL。
num_buckets	numeric	暂不支持，值为NULL。
sample_size	numeric	暂不支持，值为NULL。
last_analyzed	timestamp(0) without time zone	暂不支持，值为NULL。
global_stats	character varying(3)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。
notes	character varying(41)	暂不支持，值为NULL。
avg_col_len	numeric	暂不支持，值为NULL。
histogram	character varying(15)	暂不支持，值为NULL。

名称	类型	描述
schema	character varying(64)	列所属的名称空间的名称。

12.3.40 ADM_SUBPART_KEY_COLUMNS

ADM_SUBPART_KEY_COLUMNS视图显示了数据库中所有的二级分区表或分区索引的分区键列的相关信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS schema下。

表 12-146 ADM_SUBPART_KEY_COLUMNS 字段

名称	类型	描述
owner	character varying(128)	二级分区表或索引的拥有者。
name	character varying(128)	二级分区表名或索引名。
object_type	character varying(128)	对象类型。 • 若分区为二级分区表，此列为table。 • 若分区为二级分区索引，此列为index。
column_name	character varying(4000)	二级分区表或索引的键列名。
column_position	numeric	列在分区中的位置。
collated_column_id	numeric	暂不支持，值为NULL。

12.3.41 ADM_SYNONYMS

ADM_SYNONYMS视图显示数据库中所有同义词的信息。默认只有系统管理员权限才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-147 ADM_SYNONYMS 字段

名称	类型	描述
owner	text	同义词的所有者。
schema_name	text	同义词所属模式名。
synonym_name	text	同义词的名称。

名称	类型	描述
table_owner	text	关联对象的所有者。尽管该列称为table_owner，但它拥有的该关联对象不一定是表，可以是任何数据库通用对象，例如视图、存储过程、同义词等。
table_name	text	关联对象名。尽管该列称为table_name，但此关联对象不一定是表，可以是任何数据库通用对象，例如视图、存储过程、同义词等。
table_schema_name	text	关联对象所属模式名。尽管该列称为table_schema_name，但此Schema下的该关联对象不一定是表，可以是任何数据库通用对象，例如视图、存储过程、同义词等。

12.3.42 ADM_SYS_PRIVS

ADM_SYS_PRIVS视图存储授予用户和角色的系统权限信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-148 ADM_SYS_PRIVS 字段

名称	类型	描述
grantee	character varying(128)	被授予权限的用户或角色的名称。
privilege	character varying(40)	系统权限。 系统权限包括rolsuper、rolinherit、rolcreaterole、rolcreatedb、rolcatupdate、rolcanlogin、rolreplication、rolauditadmin、rolsystemadmin、roluseft、rolmonitoradmin、roloperatoradmin、rolpolicyadmin。
admin_option	character varying(3)	表示该授权是否包含ADMIN选项。 ● YES：表示包含ADMIN选项。 ● NO：表示不包含ADMIN选项。
common	character varying(3)	暂不支持，值为NULL。
inherited	character varying(3)	暂不支持，值为NULL。

12.3.43 ADM_TAB_COL_STATISTICS

ADM_TAB_COL_STATISTICS视图显示从ADM_TAB_COLUMNS中提取的列统计信息和直方图信息。默认只有系统管理员权限才可以访问，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。该视图在LOW_VALUE、HIGH_VALUE字段，由于底层表结构不同原因，与A数据库取值有差异，当LOW_VALUE为高频值时，GaussDB的LOW_VALUE为次小值。当HIGH_VALUE为高频值时，GaussDB的HIGH_VALUE为次高值。HISTOGRAM字段，由于统计方式不同原因，与A数据库取值有差异，GaussDB只支持两种类型直方图frequency，equi-width。SCOPE字段，由于GaussDB不支持全局临时表统计原因，与A数据库取值有差异，GaussDB只支持本地临时表信息统计，默认置SHARED。

表 12-149 ADM_TAB_COL_STATISTICS 字段

名称	类型	描述
owner	character varying(128)	表的所有者。
table_name	character varying(128)	表名。
column_name	character varying(128)	列名。
num_distinct	numeric	列中不同值的数量。
low_value	raw	列中的低值。
high_value	raw	列中的高值。
density	numeric	• 如果COLUMN_NAME上有直方图，则此列将显示直方图中跨越少于2个端点的值的选择性。它不代表跨越2个或更多端点的值的选择性。 • 如果COLUMN_NAME上没有可用的直方图，则该列的值为1/NUM_DISTINCT。
num_nulls	numeric	列中空值数。
num_buckets	numeric	列的直方图的桶数。
sample_size	numeric	用于分析此列的样本量。
last_analyzed	timestamp(0) without time zone	最近分析此列的日期，数据库重启后，数据会丢失。
global_stats	character varying(3)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。
notes	character varying(99)	暂不支持，值为NULL。
avg_col_len	numeric	列的平均长度（以字节为单位）。

名称	类型	描述
histogram	character varying(15)	直方图是否存在以及存在的类型： ● NONE：表示不存在直方图。 ● FREQUENCY：表示频率直方图。 ● EQUI-WIDTH：表示等宽直方图。
scope	character varying(7)	对于在除全局临时表之外的任何表上收集的统计信息，该值是SHARED(表示统计信息在所有会话之间共享)。
schema	character varying(64)	列所属的名称空间的名称。

12.3.44 ADM_TAB_COLS

ADM_TAB_COLS视图显示表和视图列的相关信息。数据库中每个表和视图的每一个字段在ADM_TAB_COLS里有一行对应的数据。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。该视图与ADM_TAB_COLUMNS视图显示行数相同，仅存在字段差异。

表 12-150 ADM_TAB_COLS 字段

名称	类型	描述
owner	character varying(64)	表或视图的所有者。
table_name	character varying(128)	表或视图的名称。
column_name	character varying(128)	列名。
data_type	character varying(128)	列的数据类型，可以是用户自定义的数据类型。
data_type_mod	character varying(3)	暂不支持，值为NULL。
data_type_owner	character varying(128)	列的数据类型的所有者。
data_length	numeric	列的字节长度。
data_precision	numeric	数据类型的精度，对于numeric数据类型有效，其他类型为NULL。

名称	类型	描述
data_scale	numeric	小数点右边的位数，对于numeric数据类型有效，其他类型为0。
nullable	character varying(1)	该列是否允许为空，对于主键约束和非空约束，该值为n。
column_id	numeric	创建表时列的序号。
default_length	numeric	列的默认值字节长度。
data_default	text	列的默认值。
num_distinct	numeric	列中不同值的数量。
low_value	raw	列中的最小值。
high_value	raw	列中的最大值。
density	numeric	列密度。
num_nulls	numeric	列中空值数。
num_buckets	numeric	列的直方图的桶数。
last_analyzed	timestamp(0) without time zone	上次分析的日期。
sample_size	numeric	用于分析此列的样本量。
character_set_name	character varying(44)	暂不支持，值为NULL。
char_col_decl_length	numeric	字符类型列的声明长度。
global_stats	character varying(3)	暂不支持，值为NO。
user_stats	character varying(3)	暂不支持，值为NO。
avg_col_len	numeric	列的平均长度（单位字节）。
char_length	numeric	列的长度（以字符计），只对varchar，nvarchar2，bpchar，char类型有效。
char_used	character varying(1)	暂不支持，varchar，bpchar，char类型置B，nvarchar2类型置C，其余值为空。
v80_fmt_image	character varying(3)	暂不支持，值为NULL。
data_upgraded	character varying(3)	暂不支持，值为YES。
hidden_column	character varying(3)	暂不支持，值为NULL。

名称	类型	描述
virtual_column	character varying	指示列是否为虚拟列(即生成列)： YES：表示是。 NO：表示不是。
segment_column_id	numeric	暂不支持，值为NULL。
internal_column_id	numeric	列的内部序列号，内容同COLUMN_ID。
histogram	character varying(15)	直方图类型： ● NONE：表示不存在直方图。 ● FREQUENCY：表示频率直方图。 ● EQUI_WIDTH：表示等宽直方图。
qualified_col_name	character varying(64)	限定列名，同COLUMN_NAME。
user_generated	character varying(3)	暂不支持，值为YES。
default_on_null	character varying(3)	暂不支持，值为NULL。
identity_column	character varying(3)	暂不支持，值为NULL。
sensitive_column	character varying(3)	暂不支持，值为NULL。
evaluation_edition	character varying(128)	暂不支持，值为NULL。
unusable_before	character varying(128)	暂不支持，值为NULL。
unusable_beginning	character varying(128)	暂不支持，值为NULL。
collation	character varying(100)	列的排序规则。因该字段与保留关键字冲突，调用该字段需加视图名。
collated_column_id	numeric	暂不支持，值为NULL。
schema	character varying(64)	列所属的名称空间的名称。

12.3.45 ADM_TAB_HISTOGRAMS

ADM_TAB_HISTOGRAMS系统视图显示数据库所有表和视图的直方图信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-151 ADM_TAB_HISTOGRAMS 字段

名称	类型	描述
owner	character varying(128)	表的所有者。
table_name	character varying(128)	表名。
column_name	character varying(4000)	列名。
endpoint_number	numeric	直方图的桶号。
endpoint_value	numeric	暂不支持，值为NULL。
endpoint_actual_value	character varying(4000)	桶端点的实际值。
endpoint_actual_value_raw	raw	暂不支持，值为NULL。
endpoint_repeat_count	numeric	暂不支持，值为NULL。
scope	character varying(7)	暂不支持，值为SHARED。

12.3.46 ADM_TAB_PRIVS

ADM_TAB_PRIVS视图显示数据库中所有对象的授权信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-152 ADM_TAB_PRIVS 字段

名称	类型	描述
grantee	character varying(128)	被授予权限的用户或角色的名称。
owner	character varying(128)	对象的所有者。
table_name	character varying(128)	对象的名称。
grantor	character varying(128)	执行授权的用户名。
privilege	character varying(40)	对象上的权限，包括USAGE、UPDATE、DELETE、INSERT、CONNECT、SELECT、EXECUTE。

名称	类型	描述
grantable	character varying(3)	该授权是否包含GRANT选项。 ● YES：包含GRANT选项。 ● NO：不包含GRANT选项。
type	character varying(24)	对象的类型，包括NODE GROUP、COLUMN_ENCRYPTION_KEY、PACKAGE、COLUMN、TABLE、VIEW、SEQUENCE、TYPE、INDEX、DATABASE、DIRECTORY、FOREIGN DATA WRAPPER、FOREIGN SERVER、LANGUAGE、LARGE OBJECT、SCHEMA、TEMPLATE、FUNCTION、PROCEDURE、TABLESPACE。
hierarchy	character varying(3)	暂不支持，值为NULL。
common	character varying(3)	暂不支持，值为NULL。
inherited	character varying(3)	暂不支持，值为NULL。

12.3.47 ADM_TAB_STATISTICS

ADM_TAB_STATISTICS显示数据库中所有表的优化程序统计信息。该视图同时存在于pg_catalog和sys_schema下。默认只有系统管理员权限才可以访问，普通用户需要授权才可以访问。

表 12-153 ADM_TAB_STATISTICS 字段

名称	类型	描述
owner	character varying(128)	对象的所有者。
table_name	character varying(128)	表名。
partition_name	character varying(128)	暂不支持，值为NULL。

名称	类型	描述
partition_position	numeric	暂不支持，值为NULL。
subpartition_name	character varying(128)	暂不支持，值为NULL。
subpartition_position	numeric	暂不支持，值为NULL。
object_type	character varying(12)	对象类型： • TABLE • PARTITION • SUBPARTITION
num_rows	numeric	对象中的行数。
blocks	numeric	暂不支持，值为NULL。
empty_blocks	numeric	暂不支持，值为NULL。
avg_space	numeric	暂不支持，值为NULL。
chain_cnt	numeric	暂不支持，值为NULL。
avg_row_len	numeric	平均行长，包括行开销。
avg_space_freelist_blocks	numeric	暂不支持，值为NULL。
num_freelist_blocks	numeric	暂不支持，值为NULL。
avg_cached_blocks	numeric	暂不支持，值为NULL。
avg_cache_hit_ratio	numeric	暂不支持，值为NULL。
im_imcu_count	numeric	暂不支持，值为NULL。
im_block_count	numeric	暂不支持，值为NULL。
im_stat_update_time	timestamp(9) without time zone	暂不支持，值为NULL。
scan_rate	numeric	暂不支持，值为NULL。
sample_size	numeric	分析表格时使用的样本量。
last_analyzed	timestamp with time zone	最近分析表的日期。数据库重启后，数据会丢失。
global_stats	character varying(3)	暂不支持，值为NULL。

名称	类型	描述
user_stats	character varying(3)	暂不支持，值为NULL。
stattype_locked	character varying(5)	暂不支持，值为NULL。
stale_stats	character varying(7)	暂不支持，值为NULL。
notes	character varying(25)	暂不支持，值为NULL。
scope	character varying(7)	暂不支持，默认值SHARED。

12.3.48 ADM_TAB_STATS_HISTORY

ADM_TAB_STATS_HISTORY系统视图提供数据库所有表的表统计信息历史。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-154 ADM_TAB_STATS_HISTORY 字段

名称	类型	描述
owner	character varying(128)	对象的拥有者。
table_name	character varying(128)	表名。
partition_name	character varying(128)	暂不支持，值为NULL。
subpartition_name	character varying(128)	暂不支持，值为NULL。
stats_update_time	timestamp(6) with time zone	统计信息更新的时间，数据库重启后，数据会丢失。

12.3.49 ADM_TAB_SUBPARTITIONS

ADM_TAB_SUBPARTITIONS视图存储数据库下所有的二级分区信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS schema下。

表 12-155 ADM_TAB_SUBPARTITIONS 字段

名称	类型	描述
table_owner	character varying(64)	表的所有者。

名称	类型	描述
table_name	character varying(64)	关系表名称。
partition_name	character varying(64)	分区名称。
subpartition_name	character varying(64)	二级分区名称。
high_value	text	二级分区的边界值。 ● 对于范围分区和间隔分区，显示各分区的上边界值。 ● 对于列表分区，显示各分区的取值列表。 ● 对于哈希分区，显示各分区的编号。
tablespace_name	name	二级分区表的表空间名称。
schema	character varying(64)	名称空间的名称。
high_value_length	integer	二级分区的边界值的字符长度。

12.3.50 ADM_TABLES

ADM_TABLES视图显示关于数据库下的所有表信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-156 ADM_TABLES 字段

名称	类型	描述
owner	character varying(64)	表的所有者。
table_name	character varying(64)	表名称。
tablespace_name	character varying(64)	存储表的表空间名称。
dropped	character varying	当前表是否已删除： ● YES：表示已删除。 ● NO：表示未删除。
num_rows	numeric	表的估计行数。
status	character varying(8)	当前表是否有效。 ● VALID：表示当前表有效。 ● UNUSABLE：表示当前表不可用

名称	类型	描述
sample_size	numeric	分析表使用的样本数量。
temporary	character(1)	是否为临时表： ● Y：表示是临时表。 ● N：表示不是临时表。
pct_free	numeric	块中空闲空间的最小比例。
ini_trans	numeric	事务的初始数量。
max_trans	numeric	事务数量的最大值。
avg_row_len	integer	平均每行的字节数。
partitioned	character varying(3)	表是否为分区表。 ● YES：是分区表。 ● NO：不是分区表。
last_analyzed	timestamp with time zone	上次分析表的时间。
row_movement	character varying(8)	是否允许分区行移动。 ● ENABLED：允许分区行移动。 ● DISABLED：不允许分区行移动。
compression	character varying(8)	是否启用表压缩。 ● ENABLED：启用表压缩。 ● DISABLED：不启用表压缩。
duration	character varying(15)	临时表的期限。 ● NULL：表示非临时表。 ● sys$session：表示会话临时表。 ● sys$transaction：表示事务临时表。
logical_replication	character varying(8)	表是否启用逻辑复制。 ● ENABLED：启用逻辑复制。 ● DISABLED：不启用逻辑复制。

名称	类型	描述
external	character varying(3)	表是否为外表。 ● YES：是外表。 ● NO：不是外表。
logging	character varying(3)	表的更改是否记入日志。 ● YES：表的更改记录日志。 ● NO：表的更改不记录日志。
default_collation	character varying(100)	表的默认排序规则。 ● default
degree	character varying(10)	扫描表的实例数量。
table_lock	character varying(8)	是否启用表级锁。 ● ENABLED：启用表级锁。 ● DISABLED：不启用表级锁。
nested	character varying(3)	是否为嵌套表。 ● YES：是嵌套表。 ● NO：不是嵌套表。
buffer_pool	character varying(7)	表的默认缓冲池。
flash_cache	character varying(7)	暂不支持，值为NULL。
cell_flash_cache	character varying(7)	暂不支持，值为NULL。
skip_corrupt	character varying(8)	扫描表是否跳过损坏的块。 ● ENABLED：跳过损坏的块。 ● DISABLED：不跳过损坏的块。
has_identity	character varying(3)	表是否具有标识列。 ● YES：有标识列。 ● NO：没有标识列。
segment_created	character varying(3)	表段是否已被创建。 ● YES：表段已被创建。 ● NO：表段未被创建。

名称	类型	描述
monitoring	character varying(3)	是否跟踪表的修改。 ● YES：跟踪表的修改。 ● NO：不跟踪表的修改。
cluster_name	character varying(128)	暂不支持，值为NULL。
iot_name	character varying(128)	暂不支持，值为NULL。
pct_used	numeric	暂不支持，值为NULL。
initial_extent	numeric	暂不支持，值为NULL。
next_extent	numeric	暂不支持，值为NULL。
min_extents	numeric	暂不支持，值为NULL。
max_extents	numeric	暂不支持，值为NULL。
pct_increase	numeric	暂不支持，值为NULL。
freelists	numeric	暂不支持，值为NULL。
freelist_groups	numeric	暂不支持，值为NULL。
backed_up	character varying(1)	暂不支持，值为NULL。
blocks	numeric	暂不支持，值为NULL。
empty_blocks	numeric	暂不支持，值为NULL。
avg_space	numeric	暂不支持，值为NULL。
chain_cnt	numeric	暂不支持，值为NULL。
avg_space_freelist_blocks	numeric	暂不支持，值为NULL。
num_freelist_blocks	numeric	暂不支持，值为NULL。
instances	character varying(10)	暂不支持，值为NULL。
cache	character varying(5)	暂不支持，值为NULL。
iot_type	character varying(12)	暂不支持，值为NULL。
secondary	character varying(1)	暂不支持，值为NULL。
global_stats	character varying(3)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。
cluster_owner	character varying(30)	暂不支持，值为NULL。
dependencies	character varying(8)	暂不支持，值为NULL。
compression_for	character varying(30)	暂不支持，值为NULL。
read_only	character varying(3)	暂不支持，值为NULL。

名称	类型	描述
result_cache	character varying(7)	暂不支持，值为NULL。
clustering	character varying(3)	暂不支持，值为NULL。
activity_tracking	character varying(23)	暂不支持，值为NULL。
dml_timestamp	character varying(25)	暂不支持，值为NULL。
container_data	character varying(3)	暂不支持，值为NULL。
inmemory_priority	character varying(8)	暂不支持，值为NULL。
inmemory_distribute	character varying(15)	暂不支持，值为NULL。
inmemory_compression	character varying(17)	暂不支持，值为NULL。
inmemory_duplicate	character varying(13)	暂不支持，值为NULL。
duplicated	character varying(1)	暂不支持，值为NULL。
sharded	character varying(1)	暂不支持，值为NULL。
hybrid	character varying(3)	暂不支持，值为NULL。
cellmemory	character varying(24)	暂不支持，值为NULL。
containers_default	character varying(3)	暂不支持，值为NULL。
container_map	character varying(3)	暂不支持，值为NULL。
extended_data_link	character varying(3)	暂不支持，值为NULL。
extended_data_link_map	character varying(3)	暂不支持，值为NULL。
inmemory_service	character varying(12)	暂不支持，值为NULL。
inmemory_service_name	character varying(1000)	暂不支持，值为NULL。
container_map_object	character varying(3)	暂不支持，值为NULL。
memoptimize_read	character varying(8)	暂不支持，值为NULL。
memoptimize_write	character varying(8)	暂不支持，值为NULL。
has_sensitive_column	character varying(3)	暂不支持，值为NULL。
admit_null	character varying(3)	暂不支持，值为NULL。
data_link_dml_enabled	character varying(3)	暂不支持，值为NULL。
object_id_type	character varying(16)	暂不支持，值为NULL。
table_type_owner	character varying(128)	暂不支持，值为NULL。

名称	类型	描述
table_type	character varying(128)	暂不支持，值为NULL。
compress_for	character varying(30)	暂不支持，值为NULL。

12.3.51 ADM_TABLESPACES

ADM_TABLESPACES视图显示所有的表空间信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。A数据库与GaussDB逻辑结构特性不一致。

表 12-157 ADM_TABLESPACES 字段

名称	类型	描述
tablespace_name	character varying(64)	表空间名称。
block_size	numeric	暂不支持，值为NULL。
initial_extent	numeric	暂不支持，值为NULL。
next_extent	numeric	暂不支持，值为NULL。
min_extents	numeric	暂不支持，值为NULL。
max_extents	numeric	暂不支持，值为NULL。
max_size	numeric	暂不支持，值为NULL。
pct_increase	numeric	暂不支持，值为NULL。
min_extlen	numeric	暂不支持，值为NULL。
contents	character varying(9)	暂不支持，值为NULL。
status	character varying(9)	暂不支持，默认为ONLINE（在线）。
logging	character varying(9)	暂不支持，值为NULL。
force_logging	character varying(3)	暂不支持，值为NULL。
extent_management	character varying(10)	暂不支持，值为NULL。
allocation_type	character varying(9)	暂不支持，值为NULL。
plugged_in	character varying(3)	暂不支持，值为NULL。
segment_space_management	character varying(6)	暂不支持，值为NULL。
def_tab_compression	character varying(8)	暂不支持，值为NULL。
retention	character varying(11)	暂不支持，值为NULL。
bigfile	character varying(3)	暂不支持，值为NULL。

名称	类型	描述
predicate_evaluation	character varying(7)	暂不支持，值为NULL。
encrypted	character varying(3)	暂不支持，值为NULL。
compress_for	character varying(30)	暂不支持，值为NULL。
def_inmemory	character varying(8)	暂不支持，值为NULL。
def_inmemory_priority	character varying(8)	暂不支持，值为NULL。
def_inmemory_distribute	character varying(15)	暂不支持，值为NULL。
def_inmemory_compression	character varying(17)	暂不支持，值为NULL。
def_inmemory_duplicate	character varying(13)	暂不支持，值为NULL。
shared	character varying(12)	暂不支持，值为NULL。
def_index_compression	character varying(8)	暂不支持，值为NULL。
index_compress_for	character varying(13)	暂不支持，值为NULL。
def_cellmemory	character varying(14)	暂不支持，值为NULL。
def_inmemory_service	character varying(12)	暂不支持，值为NULL。
def_inmemory_service_name	character varying(1000)	暂不支持，值为NULL。
lost_write_protect	character varying(7)	暂不支持，值为NULL。
chunk_tablespace	character varying(1)	暂不支持，值为NULL。

12.3.52 ADM_TAB_COLUMNS

ADM_TAB_COLUMNS视图显示关于表和视图的字段的信息。数据库中每个表和视图的每一个字段在ADM_TAB_COLUMNS里有一行对应的数据。默认只有系统管理员权限才可以访问，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-158 ADM_TAB_COLUMNS 字段

名称	类型	描述
owner	character varying(64)	表的所有者。
table_name	character varying(64)	表的名称。
column_name	character varying(64)	列名。
data_type	character varying(128)	列的数据类型。

名称	类型	描述
data_type_mod	character varying(3)	暂不支持，值为NULL。
data_type_owner	character varying(128)	列的数据类型的所有者。
data_length	integer	列的字节长度。
data_precision	integer	数据类型的精度，对于numeric数据类型有效，其他类型为NULL。
data_scale	integer	小数点右边的位数，对于numeric数据类型有效，其他类型为0。
nullable	bpchar	该列是否允许为空。 • y：允许。 • n：不允许。对于主键约束和非空约束，该值为n。
column_id	integer	创建表时列的序号。
default_length	numeric	列的默认值字节长度，无默认值时为空。
data_default	text	列的默认值。
num_distinct	numeric	列中不同值的数量。
low_value	raw	列中的最小值。
high_value	raw	列中的最大值。
density	numeric	列密度。
num_nulls	numeric	列中空值数。
num_buckets	numeric	列的直方图的桶数。
last_analyzed	timestamp(0) without time zone	上次分析的日期。
sample_size	numeric	用于分析此列的样本量。
character_set_name	character varying(44)	暂不支持，值为NULL。
char_col_decl_length	numeric	字符类型列的声明长度。
global_stats	character varying(3)	暂不支持，值为NO。
user_stats	character varying(3)	暂不支持，值为NO。
avg_col_len	numeric	列的平均长度（单位字节）。
char_length	numeric	列的长度（以字符计），只对varchar，nvarchar2，bpchar，char类型有效。

名称	类型	描述
char_used	character varying(1)	暂不支持，varchar，bpchar，char类型置B，nvarchar2类型置C，其余置NULL。
v80_fmt_image	character varying(3)	暂不支持，值为NULL。
data_upgraded	character varying(3)	暂不支持，值为YES。
histogram	character varying(15)	直方图是否存在以及存在的类型： ● NONE：表示不存在直方图。 ● FREQUENCY：表示频率直方图。 ● EQUI_WIDTH：表示等宽直方图。
default_on_null	character varying(3)	暂不支持，值为NULL。
identity_column	character varying(3)	暂不支持，值为NULL。
sensitive_column	character varying(3)	暂不支持，值为NULL。
evaluation_edition	character varying(128)	暂不支持，值为NULL。
unusable_before	character varying(128)	暂不支持，值为NULL。
unusable_beginning	character varying(128)	暂不支持，值为NULL。
collation	character varying(100)	列的排序规则。因该字段与保留关键字冲突，调用该字段需加视图名。
comments	text	列的注释。
schema	character varying(64)	列所属的名称空间的名称。

12.3.53 ADM_TAB_COMMENTS

ADM_TAB_COMMENTS视图显示数据库下的所有表和视图的注释信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-159 ADM_TAB_COMMENTS 字段

名称	类型	描述
owner	character varying(64)	表或视图的所有者。
table_name	character varying(64)	表或视图的名称。

名称	类型	描述
comments	text	注释。
schema	character varying(64)	表所属的名称空间的名称。

12.3.54 ADM_TAB_PARTITIONS

ADM_TAB_PARTITIONS视图显示数据库下所有的一级分区信息（包括二级分区表）。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-160 ADM_TAB_PARTITIONS 字段

名称	类型	描述
table_owner	character varying(64)	表的所有者。
table_name	character varying(64)	关系表名称。
partition_name	character varying(64)	分区名称。
high_value	text	分区的边界值。 • 对于范围分区和间隔分区，显示各分区的上边界值。 • 对于列表分区，显示各分区的取值列表。 • 对于哈希分区，显示各分区的编号。
tablespace_name	name	分区表的表空间名称。
schema	character varying(64)	名称空间的名称。
subpartition_count	bigint	二级分区的个数。
high_value_length	integer	分区绑定值表达式长度。
composite	character varying(3)	表是否为二级分区表。
partition_position	numeric	分区在表中的位置。
pct_free	numeric	块中可用空间的最小百分比。
pct_used	numeric	暂不支持，值为NULL。
ini_trans	numeric	初始事务数，默认值为4，非USTORE分区表时为NULL。
max_trans	numeric	最大事务数，默认值为128，非USTORE分区表时为NULL。

名称	类型	描述
initial_extent	numeric	暂不支持，值为NULL。
next_extent	numeric	暂不支持，值为NULL。
min_extent	numeric	暂不支持，值为NULL。
max_extent	numeric	暂不支持，值为NULL。
max_size	numeric	暂不支持，值为NULL。
pct_increase	numeric	暂不支持，值为NULL。
freelists	numeric	暂不支持，值为NULL。
freelist_groups	numeric	暂不支持，值为NULL。
logging	character varying(7)	是否记录对表的更改。
compression	character varying(8)	表分区的实际压缩属性。
compress_for	character varying(30)	暂不支持，值为NULL。
num_rows	numeric	分区中的行数统计值。
blocks	numeric	暂不支持，值为NULL。
empty_blocks	numeric	暂不支持，值为NULL。
avg_space	numeric	暂不支持，值为NULL。
chain_cnt	numeric	暂不支持，值为NULL。
avg_row_len	numeric	暂不支持，值为NULL。
sample_size	numeric	暂不支持，值为NULL。
last_analyzed	timestamp with time zone	最近分析此分区的日期。
buffer_pool	character varying(7)	用于分区块的缓冲池。
flash_cache	character varying(7)	暂不支持，值为NULL。
cell_flash_cache	character varying(7)	暂不支持，值为NULL。
global_stats	character varying(3)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。
is_nested	character varying(3)	是否是嵌套表分区。
parent_table_partition	character varying(128)	暂不支持，值为NULL。
interval	character varying(3)	分区是否在间隔分区表的间隔节中。
segment_created	character varying(4)	表分区是否创建了段或未创建。

名称	类型	描述
indexing	character varying(4)	暂不支持，值为NULL。
read_only	character varying(4)	暂不支持，值为NULL。
inmemory	character varying(8)	暂不支持，值为NULL。
inmemory_priority	character varying(8)	暂不支持，值为NULL。
inmemory_distribute	character varying(15)	暂不支持，值为NULL。
inmemory_compression	character varying(17)	暂不支持，值为NULL。
inmemory_duplicate	character varying(13)	暂不支持，值为NULL。
cellmemory	character varying(24)	暂不支持，值为NULL。
inmemory_service	character varying(12)	暂不支持，值为NULL。
inmemory_service_name	character varying(100)	暂不支持，值为NULL。
memoptimize_read	character varying(8)	暂不支持，值为NULL。
memoptimize_write	character varying(8)	暂不支持，值为NULL。

12.3.55 ADM_TRIGGERS

ADM_TRIGGERS视图显示数据库中的触发器信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-161 ADM_TRIGGERS 字段

名称	类型	描述
owner	character varying(128)	触发器的所有者。
trigger_name	character varying(64)	触发器名称。
trigger_type	character varying	触发器触发的时机。取值范围：before statement、before each row、after statement、after each row、instead of。

名称	类型	描述
triggering_event	character varying	触发触发器的事件：update、insert、delete、truncate。
table_owner	character varying(64)	定义触发器的表的所有者。
base_object_type	character varying(18)	定义触发器的基础对象。取值范围：table、view。
table_name	character varying(64)	定义触发器的表或视图名称。
column_name	character varying(4000)	暂不支持，值为NULL。
referencing_name	character varying(422)	暂不支持，值为referencing new as new old as old。
when_clause	character varying(4000)	when子句的内容，必须值为true才能执行trigger_body。
status	character varying(64)	触发器的触发状态： ● O：触发器在“origin”和“local”模式下触发。 ● D：触发器被禁用。 ● R：触发器在“replica”模式下触发。 ● A：触发器始终触发。
description	character varying(4000)	触发器描述，用于重新创建触发器创建语句。
action_type	character varying(11)	触发器的动作类型，仅支持call。
trigger_body	text	触发器触发时执行的语句。
crossedition	character varying(7)	暂不支持，值为NULL。
before_statement	character varying(3)	暂不支持，值为NULL。
before_row	character varying(3)	暂不支持，值为NULL。
after_row	character varying(3)	暂不支持，值为NULL。
after_statement	character varying(3)	暂不支持，值为NULL。
instead_of_row	character varying(3)	暂不支持，值为NULL。

名称	类型	描述
fire_once	character varying(3)	暂不支持，值为NULL。
apply_server_only	character varying(3)	暂不支持，值为NULL。

12.3.56 ADM_TYPE_ATTRS

ADM_TYPE_ATTRS视图描述当前数据库对象类型的属性。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-162 ADM_TYPE_ATTRS 字段

名称	类型	描述
owner	oid	该类型的所有者。
type_name	name	数据类型名称。
attr_name	name	字段名。
attr_type_mod	integer	记录创建新表时支持类型特定的数据（比如一个varchar字段的最大长度）。它传递给类型相关的输入和长度转换函数作为第三个参数。其值对那些不需要ATTTYPMOD的类型通常为-1。
attr_type_owner	oid	该类型属性的所有者。
attr_type_name	name	数据类型属性名称。该字段记录的是转换后的类型名。
length	smallint	对于定长类型是该类型内部表现形式的字节数目。对于变长类型是负数。 ● -1表示一种“变长”（有长度字属性的数据）。 ● -2表示这是一个NULL结尾的C字符串。
precision	integer	数字类型的精度。
scale	integer	数字类型的范围。
character_set_name	character(1)	属性的字符集名称（c或n）。 ● c：CHAR_CS。 ● n：NCHAR_CS。
attr_no	smallint	属性编号。
inherited	character(1)	属性是否继承自超级类型（Y或N)。

名称	类型	描述
attr_length	integer	记录创建新表时支持类型特定的数据（比如一个varchar字段的最大长度）。对于raw类型，因内核实现原因，暂未记录。

12.3.57 ADM_TYPES

ADM_TYPES视图用于描述数据库中的所有对象类型。默认只有系统管理员权限才可以访问，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-163 ADM_TYPES 字段

名称	类型	描述
owner	character varying(128)	类型的所有者。
type_name	character varying(128)	类型名称。
type_oid	raw	类型的标识符（OID）。
typecode	character varying(128)	类型的类型代码。
attributes	numeric	类型中的属性数。
methods	numeric	暂不支持，值为0。
predefined	character varying(3)	表示该类型是否是内置类型。
incomplete	character varying(3)	表示该类型是否是不完整类型。
final	character varying(3)	暂不支持，值为NULL。
instantiable	character varying(3)	暂不支持，值为NULL。
persistable	character varying(3)	暂不支持，值为NULL。
supertype_owner	character varying(128)	暂不支持，值为NULL。
supertype_name	character varying(128)	暂不支持，值为NULL。
local_attributes	numeric	暂不支持，值为NULL。
local_methods	numeric	暂不支持，值为NULL。
typeid	raw	暂不支持，值为NULL。

12.3.58 ADM_USERS

ADM_USERS视图显示所有数据库用户的信息。默认只有系统管理员权限才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-164 ADM_USERS 字段

名称	类型	描述
username	character varying(128)	用户名称。
user_id	oid	用户ID。
account_status	character varying(32)	账户状态。 ● NULL：该账户是拥有最高权限的初始系统管理员。 ● 0：正常状态。 ● 1：由于登录失败次数超过阈值被锁定了一定的时间。 ● 2：被管理员锁定。
lock_date	timestamp with time zone	默认显示账户的创建日期，如果账户被管理员锁定，或者登录失败次数超过阈值被锁定，则显示账户被锁定的日期。初始系统管理员该字段为null。
expiry_date	timestamp with time zone	账户到期日期。
default_tablespace	character varying(4000)	数据的默认表空间。
temporary_tablespace	character varying(4000)	临时表的默认表空间的名称或表空间组的名称。
local_temp_tablespace	character varying(30)	暂不支持，默认值为NULL。
created	timestamp with time zone	用户创建日期。
profile	character varying(128)	暂不支持，默认值为NULL。
initial_rsrc_consumer_group	character varying(128)	暂不支持，默认值为NULL。
external_name	character varying(4000)	暂不支持，默认值为NULL。
password_versions	character varying(12)	显示账户密码的加密方式。取值为：MD5、SHA256或SM3。
editions_enabled	character varying(1)	暂不支持，默认值为NULL。
authentication_type	text	用户的身份验证机制。
proxy_only_connect	character varying(1)	暂不支持，默认值为NULL。

名称	类型	描述
common	character varying(3)	暂不支持，默认值为NULL。
last_login	timestamp with time zone	用户最后一次登录的时间。
oracle_maintained	character varying(1)	暂不支持，默认值为NULL。
inherited	character varying(3)	暂不支持，默认值为NULL。
default_collation	character varying(100)	用户Schema的默认字符序。
implicit	character varying(3)	暂不支持，默认值为NULL。
all_shard	character varying(3)	暂不支持，默认值为NULL。
password_change_date	timestamp with time zone	上次设置用户密码的日期。

12.3.59 ADM_VIEWS

ADM_VIEWS视图显示数据库中的视图信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-165 ADM_VIEWS 字段

名称	类型	描述
owner	character varying(64)	视图的所有者。
view_name	character varying(64)	视图名称。
text	text	视图文本。
text_length	integer	视图文本长度。
text_vc	character varying(4000)	视图创建语句。此列可能会截断视图文本。BEQUEATH子句将不会作为此视图中的TEXT_VC列的一部分出现。
type_text_length	numeric	暂不支持，值为NULL。
type_text	character varying(4000)	暂不支持，值为NULL。
oid_text_length	numeric	暂不支持，值为NULL。
oid_text	character varying(4000)	暂不支持，值为NULL。
view_type_owner	character varying(128)	暂不支持，值为NULL。
view_type	character varying(128)	暂不支持，值为NULL。

名称	类型	描述
superview_name	character varying(128)	暂不支持，值为NULL。
editioning_view	character varying(1)	暂不支持，值为NULL。
read_only	character varying(1)	暂不支持，值为NULL。
container_data	character varying(1)	暂不支持，值为NULL。
bequeath	character varying(12)	暂不支持，值为NULL。
origin_con_id	character varying(256)	暂不支持，值为NULL。
default_collation	character varying(100)	暂不支持，值为NULL。
containers_default	character varying(3)	暂不支持，值为NULL。
container_map	character varying(3)	暂不支持，值为NULL。
extended_data_link	character varying(3)	暂不支持，值为NULL。
extended_data_link_map	character varying(3)	暂不支持，值为NULL。
has_sensitive_column	character varying(3)	暂不支持，值为NULL。
admit_null	character varying(3)	暂不支持，值为NULL。
pdb_local_only	character varying(3)	暂不支持，值为NULL。

12.3.60 DB_ALL_TABLES

DB_ALL_TABLES视图显示当前用户所能访问的表和视图的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-166 DB_ALL_TABLES 字段

名称	类型	描述
owner	name	表或视图的所有者。
table_name	name	表或视图的名称。
tablespace_name	name	表或视图所在的表空间。

12.3.61 DB_ARGUMENTS

DB_ARGUMENTS视图显示当前用户可访问的存储过程和函数的参数信息。该视图同时存在于PG_CATALOG和SYS Schema下。该视图所有用户都可以访问，显示当前用户可访问的所有信息。

表 12-167 DB_ARGUMENTS 字段

名称	类型	描述
owner	character varying(128)	函数或存储过程的所有者。
object_name	character varying(128)	函数或存储过程的名称。
package_name	character varying(128)	包名。
object_id	oid	函数或存储过程的OID。
overload	character varying(40)	表示该函数是该名称的第n个重载函数。
subprogram_id	numeric	包中函数或存储过程的位置。
argument_name	character varying(128)	参数名称。
position	numeric	该参数在参数列表中的位置，函数的返回值位置默认为0。
sequence	numeric	定义参数的顺序，从1开始，返回类型在前，然后是每个参数。
data_level	numeric	复合类型参数的嵌套深度，此列的值始终为0，因为每个参数现在只显示一行。
data_type	character varying(64)	参数的数据类型。
defaulted	character varying(1)	参数是否有默认值： ● Y：表示有默认值。 ● N：表示没有默认值。
default_value	text	暂不支持，值为NULL。
default_length	numeric	暂不支持，值为NULL。
in_out	character varying(9)	参数出入属性： ● IN：表示入参。 ● OUT：表示出参。 ● IN_OUT：表示出入参。 ● VARIADIC：表示VARIADIC参数。
data_length	numeric	暂不支持，值为NULL。
data_precision	numeric	暂不支持，值为NULL。

名称	类型	描述
data_scale	numeric	暂不支持，值为NULL。
radix	numeric	数字的参数基数，smallint、integer、bigint、numeric、float为10，其余为空。
character_set_name	character varying(44)	暂不支持，值为NULL。
type_owner	character varying(128)	数据类型所有者。
type_name	character varying(128)	参数类型名，仅显示自定义类型。
type_subname	character varying(128)	暂不支持，值为NULL。
type_link	character varying(128)	暂不支持，值为NULL。
type_object_type	character varying(7)	由type_owner、type_name和type_subname列描述的类型的类型： ● TABLE：表示参数为表类型。 ● VIEW：表示参数为视图类型。 ● 其余置NULL。
pls_type	character varying(128)	对于数字类型参数，为参数的PL/SQL类型的名称，否则为空。
char_length	numeric	暂不支持，值为NULL。
char_used	character varying(1)	暂不支持，varchar，nvarchar2，bpchar，char类型置B，其余值为NULL。
origin_con_id	character varying(256)	暂不支持，值为0。

12.3.62 DB_COL_COMMENTS

DB_COL_COMMENTS视图显示当前用户可访问的表中字段的注释信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-168 DB_COL_COMMENTS 字段

名称	类型	描述
owner	character varying(64)	表的所有者。

名称	类型	描述
table_name	character varying(64)	表名。
column_name	character varying(64)	列名。
comments	text	注释。
schema	character varying(64)	列所属的名称空间的名称。

12.3.63 DB_COL_PRIVS

DB_COL_PRIVS视图显示以下授权信息：

- 当前用户作为对象所有者、授予者或被授予者时的列权限授予信息。
- 已启用角色或PUBLIC角色作为被授予者时的列权限授予信息。

默认所有用户都可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-169 DB_COL_PRIVS 字段

名称	类型	描述
grantor	character varying(128)	执行授权的用户名。
owner	character varying(128)	对象的所有者。
grantee	character varying(128)	被授予权限的用户或角色的名称。
table_schema	character varying(128)	对象的Schema。
table_name	character varying(128)	对象的名称。
column_name	character varying(128)	列的名称。
privilege	character varying(40)	列的权限。
grantable	character varying(3)	是否授予特权。 • YES：授予特权。 • NO：不授予特权。
common	character varying(3)	暂不支持，值为NULL。
inherited	character varying(3)	暂不支持，值为NULL。

12.3.64 DB_COLL_TYPES

DB_COLL_TYPES视图显示当前用户可访问的所有集合类型的信息。默认所有用户都可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-170 DB_COLL_TYPES 字段

名称	类型	描述
owner	character varying(128)	集合的所有者。
type_name	character varying(128)	集合的名称。
coll_type	character varying(128)	集合的描述。
upper_bound	numeric	暂不支持，值为NULL。
elem_type_mod	character varying(7)	元素的类型修饰。
elem_type_owner	character varying(128)	集合基于的元素类型的所有者。该值主要用于用户定义的类型。
elem_type_name	character varying(128)	集合所依据的数据类型或用户定义类型的名称。
length	numeric	暂不支持，值为NULL。
precision	numeric	暂不支持，值为NULL。
scale	numeric	暂不支持，值为NULL。
character_set_name	character varying(44)	暂不支持，值为NULL。
elem_storage	character varying(7)	暂不支持，值为NULL。
nulls_stored	character varying(3)	暂不支持，值为NULL。
char_used	character varying(1)	暂不支持，值为NULL。

12.3.65 DB_CONS_COLUMNS

DB_CONS_COLUMNS视图显示当前用户可访问的约束字段的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-171 DB_CONS_COLUMNS 字段

名称	类型	描述
owner	character varying(64)	约束创建者。
constraint_name	character varying(64)	约束名。
table_name	character varying(64)	约束相关的表名。
column_name	character varying(64)	约束相关的列名。

名称	类型	描述
position	smallint	表中列的位置。

12.3.66 DB_CONSTRAINTS

DB_CONSTRAINTS视图显示当前用户可访问的约束的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-172 DB_CONSTRAINTS 字段

名称	类型	描述
owner	character varyiny(64)	约束创建者。
constraint_name	character varying(64)	约束名。
constraint_type	text	约束类型： ● c表示检查约束。 ● f表示外键约束。 ● p表示主键约束。 ● u表示唯一约束。
table_name	character varying(64)	约束相关的表名。
index_owner	character varying(64)	约束相关的索引的所有者（只针对唯一约束和主键约束）。
index_name	character varying(64)	约束相关的索引名（只针对唯一约束和主键约束）。

12.3.67 DB_DEPENDENCIES

DB_DEPENDENCIES视图显示当前用户可访问的类型、表、视图、存储过程、函数、触发器之间的依赖关系。所有用户都可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-173 DB_DEPENDENCIES 字段

名称	类型	描述
owner	name	对象的所有者。
name	name	对象的名称。
type	character varying(18)	对象的类型。

名称	类型	描述
referenced_owner	name	被引用对象的所有者。
referenced_name	name	被引用对象的名称。
referenced_type	character varying(18)	被引用对象的类型。
referenced_link_name	character varying(128)	暂不支持，值为NULL。
dependency_type	character varying(4)	暂不支持，值为NULL。

12.3.68 DB_ERRORS

DB_ERRORS视图显示用户可访问存储对象的最新编译错误信息。默认所有用户都可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-174 DB_ERRORS 字段

名称	类型	描述
owner	character varying(128)	对象的所有者。
name	character varying(128)	对象的名称。
type	character varying(12)	对象类型。
sequence	numeric	序列号。
line	numeric	发生错误的行号。
position	numeric	暂不支持，值为NULL。
text	character varying(4000)	错误文本。
attribute	character varying(9)	属性标记，错误（ERROR）。
message_number	numeric	暂不支持，值为NULL。

12.3.69 DB_IND_COLUMNS

DB_IND_COLUMNS视图存储了当前用户可访问的所有索引的字段信息。该视图同时存在于PG_CATALOG和SYS schema下。

表 12-175 DB_IND_COLUMNS 字段

名称	类型	描述
index_owner	character varying(64)	索引的所有者。
index_name	character varying(64)	索引名。
table_owner	character varying(64)	表的所有者。
table_name	character varying(64)	表名。
column_name	name	列名。
column_position	smallint	索引中列的位置。
column_length	numeric	列的长度。
char_length	numeric	列的长度（字符类型）。
descend	character varying(4)	列的排序方式：降序（DESC）、升序（ASC）。
collated_column_id	numeric	此列提供语言排序的列的内部序列号。

12.3.70 DB_IND_EXPRESSIONS

DB_IND_EXPRESSIONS视图显示当前用户可访问的表达式索引的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-176 DB_IND_EXPRESSIONS 字段

名称	类型	描述
index_owner	character varying(64)	索引的所有者。
index_name	character varying(64)	索引名。
table_owner	character varying(64)	表的所有者。
table_name	character varying(64)	表名。
column_expression	text	定义列的基于函数的索引表达式。
column_position	smallint	索引中列的位置。

12.3.71 DB_IND_PARTITIONS

DB_IND_PARTITIONS视图显示当前用户所能访问的一级分区表Local索引的索引分区信息（不包含分区表全局索引）。所有用户都可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-177 DB_IND_PARTITIONS 字段

名称	类型	描述
index_owner	character varying(64)	索引分区所属分区表索引的所有者的名称。
index_name	character varying(64)	索引分区所属分区表索引的名称。
partition_name	character varying(64)	索引分区的名称。
def_tablespace_name	name	索引分区的表空间名称。
high_value	text	索引分区所对应分区的上边界。
index_partition_usable	boolean	索引分区是否可用。 ● t（true）：表示可用。 ● f（false）：表示不可用。
schema	character varying(64)	索引分区所属分区表索引的模式。
high_value_length	integer	索引分区所对应分区的边界的字符长度。
composite	character varying(3)	索引是否属于二级分区表上的本地索引，该表不存储二级分区信息，所以该值为NO。
subpartition_count	numeric	分区中的二级分区数，该表不存储二级分区信息，所以该值为0。
partition_position	numeric	索引分区在索引中的位置。
status	character varying(8)	索引分区是否可用。
tablespace_name	name	分区所在表空间的名称。
pct_free	numeric	块中最小可用空间百分比。
ini_trans	numeric	初始事务数，默认值为4，非USTORE分区表时为NULL。
max_trans	numeric	最大事务数，默认值为128，非USTORE分区表时为NULL。
initial_extent	numeric	暂不支持，值为NULL。
next_extent	numeric	暂不支持，值为NULL。

名称	类型	描述
min_extent	numeric	暂不支持，值为NULL。
max_extent	numeric	暂不支持，值为NULL。
max_size	numeric	暂不支持，值为NULL。
pct_increase	numeric	暂不支持，值为NULL。
freelists	numeric	暂不支持，值为NULL。
freelist_groups	numeric	暂不支持，值为NULL。
logging	character varying(7)	是否记录对索引的更改。
compression	character varying(13)	分区索引是否启用索引压缩。
blevel	numeric	暂不支持，值为NULL。
leaf_blocks	numeric	暂不支持，值为NULL。
distinct_keys	numeric	暂不支持，值为NULL。
avg_leaf_blocks_per_key	numeric	暂不支持，值为NULL。
avg_data_blocks_per_key	numeric	暂不支持，值为NULL。
clustering_factor	numeric	根据索引的值表示表中行的顺序。需要通过执行analyze进行统计。
num_rows	numeric	分区中的行数。需要通过执行vacuum进行统计。
sample_size	numeric	暂不支持，值为NULL。
last_analyzed	timestamp with time zone	最近分析此分区的日期。
buffer_pool	character varying(7)	分区的实际缓冲池。
flash_cache	character varying(7)	暂不支持，值为NULL。
cell_flash_cache	character varying(7)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。
pct_direct_access	numeric	暂不支持，值为NULL。
global_stats	character varying(3)	暂不支持，值为NULL。
domidx_opstatus	character varying(6)	暂不支持，值为NULL。
parameters	character varying(1000)	暂不支持，值为NULL。
interval	character varying(3)	分区是否在间隔分区表的间隔节中。

名称	类型	描述
segment_created	character varying(3)	索引分区段是否已创建。
orphaned_entries	character varying(3)	暂不支持，值为NULL。

12.3.72 DB_IND_SUBPARTITIONS

DB_IND_SUBPARTITIONS视图显示当前用户所能访问的二级分区表Local索引的索引分区信息（不包含分区表全局索引）。所有用户都可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-178 DB_IND_SUBPARTITIONS 字段

名称	类型	描述
index_owner	character varying(64)	索引分区所属分区表索引的所有者的名称。
index_name	character varying(64)	索引分区所属分区表索引的名称。
partition_name	character varying(64)	索引所在一级分区的名称。
subpartition_name	character varying(64)	索引所在二级分区的名称。
def_tablespace_name	name	索引分区的表空间名称。
high_value	text	索引分区所对应分区的边界值。
index_partition_usable	boolean	索引分区是否可用。 ● t（true）：可用。 ● f（false）：不可用。
schema	character varying(64)	索引分区所属分区表索引的模式。
high_value_length	integer	索引分区所对应分区的边界的字符长度。
partition_position	numeric	索引分区在索引中的位置。
subpartition_position	numeric	二级分区在分区中的位置。
status	character varying(8)	指示索引分区是否可用。
tablespace_name	name	索引分区的表空间名称。

名称	类型	描述
pct_free	numeric	块中最小可用空间百分比。
ini_trans	numeric	初始事务数，默认值为4，非USTORE分区表时为NULL。
max_trans	numeric	最大事务数，默认值为128，非USTORE分区表时为NULL。
initial_extent	numeric	暂不支持，值为NULL。
next_extent	numeric	暂不支持，值为NULL。
min_extent	numeric	暂不支持，值为NULL。
max_extent	numeric	暂不支持，值为NULL。
max_size	numeric	暂不支持，值为NULL。
pct_increase	numeric	暂不支持，值为NULL。
freelists	numeric	暂不支持，值为NULL。
freelist_groups	numeric	暂不支持，值为NULL。
logging	character varying(7)	是否记录对索引的更改。
compression	character varying(13)	用于二级分区的压缩类型。
blevel	numeric	暂不支持，值为NULL。
leaf_blocks	numeric	暂不支持，值为NULL。
distinct_keys	numeric	暂不支持，值为NULL。
avg_leaf_blocks_per_key	numeric	暂不支持，值为NULL。
avg_data_blocks_per_key	numeric	暂不支持，值为NULL。
clustering_factor	numeric	根据索引的值表示表中行的顺序。需要通过执行analyze进行统计。
num_rows	numeric	二级分区中的行数。需要通过执行vacuum进行统计。
sample_size	numeric	暂不支持，值为NULL。
last_analyzed	timestamp with time zone	最近分析此分区的日期。
buffer_pool	character varying(7)	二级分区的缓冲池。

名称	类型	描述
flash_cache	character varying(7)	暂不支持，值为NULL。
cell_flash_cache	character varying(7)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。
global_stats	character varying(3)	暂不支持，值为NULL。
interval	character varying(3)	分区是否在间隔分区表的间隔节中。
segment_created	character varying(3)	索引分区段是否已创建。
domidx_opstatus	character varying(6)	暂不支持，值为NULL。
parameters	character varying(1000)	暂不支持，值为NULL。

12.3.73 DB_INDEXES

DB_INDEXES视图显示当前用户可访问的索引信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-179 DB_INDEXES 字段

名称	类型	描述
owner	character varying(64)	索引的所有者。
index_name	character varying(64)	索引名称。
table_name	character varying(64)	索引对应的表名。
uniqueness	text	表示该索引是否为唯一索引。 ● UNIQUE：唯一索引。 ● NONUNIQUE：非唯一索引。
partitioned	character(3)	表示该索引是否具有分区表的性质。 ● Yes：索引具有分区表的性质。 ● No：索引不具有分区表的性质。

名称	类型	描述
generated	character varying(1)	表示该索引的名称是否为系统生成。 ● y：索引名称为系统生成。 ● n：索引名称非系统生成。
index_type	character varying(27)	索引类型。 ● NORMAL：索引属性都是简单引用，表达式树为空。 ● FUNCTION-BASED NORMAL：存在表达式树用于非简单字段引用的索引属性。
table_owner	character varying(128)	索引对象的所有者。
table_type	character(11)	索引对象的类型。 ● TABLE：索引对象为表类型。
tablespace_name	character varying(30)	包含索引的表空间名称。
status	character varying(8)	非分区索引状态。 ● VALID：非分区索引可以用于查询。 ● UNUSABLE：非分区索引不可用。 ● N/A：索引具有分区表性质。
compression	character varying(13)	暂不支持，值为NULL。
prefix_length	numeric	暂不支持，值为NULL。
ini_trans	numeric	暂不支持，值为NULL。
max_trans	numeric	暂不支持，值为NULL。
initial_extent	numeric	暂不支持，值为NULL。
next_extent	numeric	暂不支持，值为NULL。
min_extents	numeric	暂不支持，值为NULL。
max_extents	numeric	暂不支持，值为NULL。
pct_increase	numeric	暂不支持，值为NULL。
pct_threshold	numeric	暂不支持，值为NULL。

名称	类型	描述
include_column	numeric	暂不支持，值为NULL。
freelists	numeric	暂不支持，值为NULL。
freelist_groups	numeric	暂不支持，值为NULL。
pct_free	numeric	暂不支持，值为NULL。
logging	character varying(3)	暂不支持，值为NULL。
blevel	numeric	暂不支持，值为NULL。
leaf_blocks	numeric	暂不支持，值为NULL。
distinct_keys	numeric	暂不支持，值为NULL。
avg_leaf_blocks_per_key	numeric	暂不支持，值为NULL。
avg_data_blocks_per_key	numeric	暂不支持，值为NULL。
clustering_factor	numeric	暂不支持，值为NULL。
num_rows	numeric	暂不支持，值为NULL。
sample_size	numeric	暂不支持，值为NULL。
last_analyzed	timestamp(0) without time zone	暂不支持，值为NULL。
degree	character varying(40)	暂不支持，值为NULL。
instances	character varying(40)	暂不支持，值为NULL。
temporary	character varying(1)	暂不支持，值为NULL。
secondary	character varying(1)	暂不支持，值为NULL。
buffer_pool	character varying(7)	暂不支持，值为NULL。
flash_cache	character varying(7)	暂不支持，值为NULL。
cell_flash_cache	character varying(7)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。
duration	character varying(15)	暂不支持，值为NULL。
pct_direct_access	numeric	暂不支持，值为NULL。
ityp_owner	character varying(128)	暂不支持，值为NULL。
ityp_name	character varying(128)	暂不支持，值为NULL。
parameters	character varying(1000)	暂不支持，值为NULL。
global_stats	character varying(3)	暂不支持，值为NULL。

名称	类型	描述
domidx_status	character varying(12)	暂不支持，值为NULL。
domidx_opstatus	character varying(6)	暂不支持，值为NULL。
funcidx_status	character varying(8)	暂不支持，值为NULL。
join_index	character varying(3)	暂不支持，值为NULL。
iot_redundant_pkey_elim	character varying(3)	暂不支持，值为NULL。
dropped	character varying(3)	暂不支持，值为NULL。
visibility	character varying(9)	暂不支持，值为NULL。
domidx_management	character varying(14)	暂不支持，值为NULL。
segment_created	character varying(3)	暂不支持，值为NULL。
orphaned_entries	character varying(3)	暂不支持，值为NULL。
indexing	character varying(7)	暂不支持，值为NULL。
auto	character varying(3)	暂不支持，值为NULL。

12.3.74 DB_OBJECTS

DB_OBJECTS视图显示当前用户可访问的数据库对象的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-180 DB_OBJECTS 字段

名称	类型	描述
owner	name	对象的所有者。
object_name	name	对象的名称。
object_id	oid	对象的OID。
object_type	name	对象的类型。
namespace	oid	对象所在的命名空间的ID。
temporary	character(1)	对象是否为临时对象。
status	character varying(7)	对象的状态。
subobject_name	name	对象的子对象名称。
generated	character(1)	对象名称是否是系统生成。

名称	类型	描述
created	timestamp with time zone	对象的创建时间。
last_ddl_time	timestamp with time zone	对象的最后修改时间。
default_collation	character varying(100)	对象的默认排序规则。
data_object_id	numeric	暂不支持，值为NULL。
timestamp	character varying(19)	暂不支持，值为NULL。
secondary	character varying(1)	暂不支持，值为NULL。
edition_name	character varying(128)	暂不支持，值为NULL。
sharing	character varying(18)	暂不支持，值为NULL。
editionable	character varying(1)	暂不支持，值为NULL。
oracle_maintained	character varying(1)	暂不支持，值为NULL。
application	character varying(1)	暂不支持，值为NULL。
duplicated	character varying(1)	暂不支持，值为NULL。
sharded	character varying(1)	暂不支持，值为NULL。
created_appid	numeric	暂不支持，值为NULL。
modified_appid	numeric	暂不支持，值为NULL。
created_vsnid	numeric	暂不支持，值为NULL。
modified_vsnid	numeric	暂不支持，值为NULL。

须知

created和last_ddl_time支持的范围参见**PG_OBJECT**中的记录范围。

12.3.75 DB_PART_COL_STATISTICS

DB_PART_COL_STATISTICS视图显示当前用户可访问的表分区的列统计信息和直方图信息。所有用户都可以访问该视图。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-181 DB_PART_COL_STATISTICS 字段

名称	类型	描述
owner	character varying(128)	分区表的所有者。
table_name	character varying(128)	表名。
partition_name	character varying(128)	表分区名称。
column_name	character varying(4000)	列名。
num_distinct	numeric	暂不支持，值为NULL。
low_value	raw	暂不支持，值为NULL。
high_value	raw	暂不支持，值为NULL。
density	numeric	暂不支持，值为NULL。
num_nulls	numeric	暂不支持，值为NULL。
num_buckets	numeric	暂不支持，值为NULL。
sample_size	numeric	暂不支持，值为NULL。
last_analyzed	date	暂不支持，值为NULL。
global_stats	character varying(3)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。
notes	character varying(63)	暂不支持，值为NULL。
avg_col_len	numeric	暂不支持，值为NULL。
histogram	character varying(15)	暂不支持，值为NULL。
schema	character varying(64)	列所属的名称空间的名称。

12.3.76 DB_PART_INDEXES

DB_PART_INDEXES视图存储当前用户所能访问的分区表索引的信息（不包含分区表全局索引）。该视图同时存在于PG_CATALOG和SYS schema下。

表 12-182 DB_PART_INDEXES 字段

名称	类型	描述
def_tablespace_name	name	分区表索引的表空间名称。
index_owner	character varying(64)	分区表索引的所有者名称。
index_name	character varying(64)	分区表索引的名称。
partition_count	bigint	分区表索引的索引分区的个数。
partitioning_key_count	integer	分区表的分区键个数。
partitioning_type	text	分区表的分区策略。 **说明** 当前分区表策略支持范围见**CREATE TABLE PARTITION**。
schema	character varying(64)	分区表索引所属模式名称。
table_name	character varying(64)	分区表索引所属的分区表名称。
subpartitioning_type	text	二级分区表的分区策略。如果分区表是一级分区表，则显示NONE。 **说明** 当前二级分区表策略支持范围见**CREATE TABLE SUBPARTITION**。
def_subpartition_count	integer	默认创建二级分区的个数，二级分区表为1，一级分区表为0。
subpartitioning_key_count	integer	分区表二级分区键的个数。

12.3.77 DB_PART_KEY_COLUMNS

DB_PART_KEY_COLUMNS视图显示了当前用户可访问的分区表或分区索引的分区键列的相关信息。该视图所有用户可访问，显示当前用户可访问的所有信息。该视图同时存在于PG_CATALOG和SYS schema下。

表 12-183 DB_PART_KEY_COLUMNS 字段

名称	类型	描述
owner	character varying(128)	分区表或索引的拥有者。
name	character varying(128)	分区表名或索引名。

名称	类型	描述
object_type	character varying(128)	对象类型。 • 若分区为分区表，此列为table。 • 若分区为分区索引，此列为index。
column_name	character varying(4000)	分区表或索引的键列名。
column_position	numeric	列在分区中的位置。
collated_column_id	numeric	暂不支持，值为NULL。

12.3.78 DB_PART_TABLES

DB_PART_TABLES视图显示当前用户所能访问的分区表的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-184 DB_PART_TABLES 字段

名称	类型	描述
table_owner	character varying(64)	分区表的所有者名称。
table_name	character varying(64)	分区表的名称。
partitioning_type	text	分区表的分区策略。 **说明** 当前分区表策略支持范围见 **CREATE TABLE PARTITION**。
partition_count	bigint	分区表的分区个数。
partitioning_key_count	integer	分区表的分区键个数。
def_tablespace_name	name	分区表的表空间名称。
schema	character varying(64)	分区表的模式。
subpartitioning_type	text	二级分区表的分区策略。如果分区表是一级分区表，则显示NONE。 **说明** 当前二级分区表策略支持范围见 **CREATE TABLE SUBPARTITION**。
def_subpartition_count	integer	默认创建二级分区的个数，二级分区表为1，一级分区表为0。

名称	类型	描述
subpartitioning_key_count	integer	分区表二级分区键的个数。
status	character varying(8)	暂不支持，值为valid。
def_pct_free	numeric	添加分区时使用的PCTFREE默认值。
def_pct_used	numeric	暂不支持，值为NULL。
def_ini_trans	numeric	添加分区时使用的INITRANS默认值。
def_max_trans	numeric	添加分区时使用的MAXTRANS默认值。
def_initial_extent	character varying(40)	暂不支持，值为NULL。
def_next_extent	character varying(40)	暂不支持，值为NULL。
def_min_extents	character varying(40)	暂不支持，值为NULL。
def_max_extents	character varying(40)	暂不支持，值为NULL。
def_max_size	character varying(40)	暂不支持，值为NULL。
def_pct_increase	character varying(40)	暂不支持，值为NULL。
def_freelists	numeric	暂不支持，值为NULL。
def_freelist_groups	numeric	暂不支持，值为NULL。
def_logging	character varying(7)	暂不支持，值为NULL。
def_compression	character varying(8)	添加分区时使用的默认压缩： • NONE • ENABLED • DISABLED
def_compress_for	character varying(30)	添加分区时使用的默认压缩。 **说明** 可用的压缩方法和压缩级别见 **WITH ({ storage_paramet...**。
def_buffer_pool	character varying(7)	暂不支持，值为DEFAULT。
def_flash_cache	character varying(7)	暂不支持，值为NULL。
def_cell_flash_cache	character varying(7)	暂不支持，值为NULL。

名称	类型	描述
ref_ptn_constraint_name	character varying(128)	暂不支持，值为NULL。
interval	character varying(1000)	区间值字符串。
autolist	character varying(3)	暂不支持，值为NO。
interval_subpartition	character varying(1000)	暂不支持，值为NULL。
autolist_subpartition	character varying(3)	暂不支持，值为NO。
is_nested	character varying(3)	暂不支持，值为NO。
def_segment_creation	character varying(4)	暂不支持段页式设置，当启用segment时，值为YES。
def_indexing	character varying(3)	暂不支持，值为ON。
def_inmemory	character varying(8)	暂不支持，值为NONE。
def_inmemory_priority	character varying(8)	暂不支持，值为NULL。
def_inmemory_distribute	character varying(15)	暂不支持，值为NULL。
def_inmemory_compression	character varying(17)	暂不支持，值为NULL。
def_inmemory_duplicate	character varying(13)	暂不支持，值为NULL。
def_read_only	character varying(3)	暂不支持，值为NO。
def_cellmemory	character varying(24)	暂不支持，值为NULL。
def_inmemory_service	character varying(12)	暂不支持，值为NULL。
def_inmemory_service_name	character varying(1000)	暂不支持，值为NULL。

12.3.79 DB_PROCEDURES

DB_PROCEDURES视图显示当前用户可访问的所有存储过程和函数的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-185 DB_PROCEDURES 字段

名称	类型	描述
owner	name	对象的所有者。
object_name	name	对象的名称。

12.3.80 DB_SCHEDULER_JOB_ARGS

DB_SCHEDULER_JOB_ARG视图显示当前用户可访问任务的有关参数信息。该视图所有用户可访问，显示当前用户可访问的所有信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-186 DB_SCHEDULER_JOB_ARGS 字段

名称	类型	描述
owner	character varying(128)	参数所属作业的拥有者。
job_name	character varying(128)	参数所属作业名。
argument_name	character varying(128)	参数名称。
argument_position	numeric	参数在参数列表中的位置。
argument_type	character varying(257)	参数的数据类型，可以是用户的自定义数据类型。
value	character varying(4000)	参数值。
anydata_value	character varying(4000)	暂不支持，值为NULL。
out_argument	character varying(5)	保留字段，值为NULL。

12.3.81 DB_SCHEDULER_PROGRAM_ARGS

DB_SCHEDULER_PROGRAM_ARG视图显示当前用户可访问程序的有关参数信息。该视图所有用户可访问，显示当前用户可访问的所有信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-187 DB_SCHEDULER_PROGRAM_ARGS 字段

名称	类型	描述
owner	character varying(128)	参数所属程序的拥有者。
program_name	character varying(128)	参数所属程序名。
argument_name	character varying(128)	参数名称。
argument_position	numeric	参数在参数列表中的位置。
argument_type	character varying(257)	参数的数据类型，可以是用户的自定义数据类型。
metadata_attribute	character varying(19)	暂不支持，值为NULL。
default_value	character varying(4000)	参数默认值。
anydata_default_value	character varying(4000)	暂不支持，值为NULL。
out_argument	character varying(5)	保留字段，值为NULL。

12.3.82 DB_SEQUENCES

DB_SEQUENCES视图显示当前用户能够访问的所有序列的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-188 DB_SEQUENCES 字段

名称	类型	描述
sequence_owner	name	序列所有者。
sequence_name	name	序列的名称。
min_value	int16	序列最小值。
max_value	int16	序列最大值。
increment_by	int16	序列的增量。
cycle_flag	character(1)	序列是否是循环序列。取值范围： ● Y：循环序列。 ● N：不是循环序列。

名称	类型	描述
order_flag	character varying(1)	标志序列是否按照请求顺序发生，暂不支持，值为NULL。
last_number	int16	上一序列的值。
cache_size	int16	序列磁盘缓存大小。
scale_flag	character varying(1)	标志是否为可扩展序列，暂不支持，值为NULL。
extend_flag	character varying(1)	标志可扩展序列生成的值是否超出序列最大值、最小值范围。暂不支持，值为NULL。
sharded_flag	character varying(1)	标志是否是分片序列，暂不支持，值为NULL。
session_flag	character varying(1)	标志序列是否是会话私有，暂不支持，值为NULL。
keep_value	character varying(1)	标志在失败后的replay期间是否保留序列值，暂不支持，值为NULL。

12.3.83 DB_SOURCE

DB_SOURCE视图显示当前用户可访问的存储过程、函数、触发器、包的定义信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-189 DB_SOURCE 字段

名称	类型	描述
owner	name	对象的所有者。
name	name	对象名字。
type	name	对象类型。取值范围：function、package、package body、procedure、trigger。
line	numeric	此行在定义信息中的行号。
text	text	存储对象的文本来源。
origin_con_id	character varying(256)	暂不支持，值为0。

12.3.84 DB_SUBPART_COL_STATISTICS

DB_SUBPART_COL_STATISTICS视图存储了当前用户可访问的分区对象的子分区的列统计信息和直方图信息。所有用户都可以访问该视图.该视图同时存在于pg_catalog和sys schema下。

名称	类型	描述
owner	character varying(128)	表的所有者。
table_name	character varying(128)	表名。
subpartition_name	character varying(128)	表子分区名称。
column_name	character varying(4000)	列名。
num_distinct	numeric	暂不支持，值为NULL。
low_value	raw	暂不支持，值为NULL。
high_value	raw	暂不支持，值为NULL。
density	numeric	暂不支持，值为NULL。
num_nulls	numeric	暂不支持，值为NULL。
num_buckets	numeric	暂不支持，值为NULL。
sample_size	numeric	暂不支持，值为NULL。
last_analyzed	timestamp(0) without time zone	暂不支持，值为NULL。
global_stats	character varying(3)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。
notes	character varying(41)	暂不支持，值为NULL。
avg_col_len	numeric	暂不支持，值为NULL。
histogram	character varying(15)	暂不支持，值为NULL。
schema	character varying(64)	列所属的名称空间的名称。

12.3.85 DB_SUBPART_KEY_COLUMNS

DB_SUBPART_KEY_COLUMNS视图显示了当前用户可访问的二级分区表或分区索引的分区键列的相关信息。该视图所有用户可访问，显示当前用户可访问的所有信息。该视图同时存在于PG_CATALOG和SYS schema下。

表 12-190 DB_SUBPART_KEY_COLUMNS 字段

名称	类型	描述
owner	character varying(128)	二级分区表或索引的拥有者。
name	character varying(128)	二级分区表名或索引名。

名称	类型	描述
object_type	character varying(128)	对象类型。 ● 若分区为分区表，此列为table。 ● 若分区为分区索引，此列为index。
column_name	character varying(4000)	二级分区表或索引的键列名。
column_position	numeric	列在分区中的位置。
collated_column_id	numeric	暂不支持，值为NULL。

12.3.86 DB_SYNONYMS

DB_SYNONYMS视图显示当前用户可访问的所有同义词信息。

表 12-191 DB_SYNONYMS 字段

名称	类型	描述
owner	text	同义词的所有者。
schema_name	text	同义词所属模式名。
synonym_name	text	同义词的名称。
table_owner	text	关联对象的所有者。尽管该列称为table_owner，但它拥有的该关联对象不一定是表，可以是任何数据库通用对象，例如视图、存储过程、同义词等。
table_name	text	关联对象名。尽管该列称为table_name，但此关联对象不一定是表，可以是任何数据库通用对象，例如视图、存储过程、同义词等。
table_schema_name	text	关联对象所属模式名。尽管该列称为table_schema_name，但此schema下的该关联对象不一定是表，可以是任何数据库通用对象，例如视图、存储过程、同义词等。
db_link	character varying(128)	保留字段，值为NULL。
origin_con_id	character varying(256)	暂不支持，值为0。

12.3.87 DB_TAB_COL_STATISTICS

DB_TAB_COL_STATISTICS视图显示从DB_TAB_COLUMNS中提取的列统计信息和直方图信息。所有用户都可以访问该视图。该视图同时存在于PG_CATALOG和SYS Schema

下。该视图在LOW_VALUE、HIGH_VALUE字段，由于底层表结构不同原因，与A数据库取值有差异，当LOW_VALUE为高频值时，GaussDB的LOW_VALUE为次小值。当HIGH_VALUE为高频值时，GaussDB的HIGH_VALUE为次高值。HISTOGRAM字段，由于统计方式不同原因，与A数据库取值有差异，GaussDB只支持两种类型直方图frequency，equi-width。SCOPE字段，由于GaussDB不支持全局临时表统计原因，与A数据库取值有差异，GaussDB只支持本地临时表信息统计，默认置SHARED。

表 12-192 DB_TAB_COL_STATISTICS 字段

名称	类型	描述
owner	character varying(128)	表的所有者。
table_name	character varying(128)	表名。
column_name	character varying(128)	列名。
num_distinct	numeric	列中不同值的数量。
low_value	raw	列中的低值。
high_value	raw	列中的高值。
density	numeric	• 如果COLUMN_NAME上有直方图，则此列将显示直方图中跨越少于2个端点的值的选择性。它不代表跨越2个或更多端点的值的选择性。 • 如果COLUMN_NAME上没有可用的直方图，则该列的值为1/NUM_DISTINCT。
num_nulls	numeric	列中空值数。
num_buckets	numeric	列的直方图中的桶数。
sample_size	numeric	用于分析此列的样本量。
last_analyzed	timestamp(0) without time zone	最近分析此列的日期。数据库重启后，数据会丢失。
global_stats	character varying(3)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。
notes	character varying(99)	暂不支持，值为NULL。
avg_col_len	numeric	列的平均长度（以字节为单位）。
histogram	character varying(15)	表示直方图是否存在以及存在的类型： • NONE：表示不存在直方图。 • FREQUENCY：表示频率直方图。 • EQUI-WIDTH：表示等宽直方图。

名称	类型	描述
scope	character varying(7)	该值SHARED用于在除全局临时表之外的任何表上收集的统计信息，值为SHARED。
schema	character varying(64)	列所属的名称空间的名称。

12.3.88 DB_TAB_COLUMNS

DB_TAB_COLUMNS视图显示当前用户可访问的表和视图的列的描述信息。该视图同时存在于PG_CATALOG和SYS Schema下。该视图所有用户可访问，显示当前用户可访问的所有信息。

表 12-193 DB_TAB_COLUMNS 字段

名称	类型	描述
owner	character varying(64)	表的所有者。
table_name	character varying(64)	表的名称。
column_name	character varying(64)	列的名称。
data_type	character varying(128)	列的数据类型。
data_type_mod	character varying(3)	暂不支持，值为NULL。
data_type_owner	character varying(128)	列的数据类型的所有者。
data_length	integer	列的字节长度。
data_precision	integer	数据类型的精度，对于numeric数据类型有效，其他类型为NULL。
data_scale	integer	小数点右边的位数，对于numeric数据类型有效，其他类型为0。
nullable	bpchar	该列是否允许为空，对于主键约束和非空约束，该值为n。
column_id	integer	创建表时列的序号。
default_length	numeric	列的默认值字节长度。
data_default	text	列的默认值。
num_distinct	numeric	列中不同值的数量。

名称	类型	描述
low_value	raw	列中的最小值。
high_value	raw	列中的最大值。
density	numeric	列密度。
num_nulls	numeric	列中空值数。
num_buckets	numeric	列的直方图的桶数。
last_analyzed	timestamp(0) without time zone	上次分析的日期。
sample_size	numeric	用于分析此列的样本量。
character_set_name	character varying(44)	暂不支持，值为NULL。
char_col_decl_length	numeric	字符类型列的声明长度。
global_stats	character varying(3)	暂不支持，值为NO。
user_stats	character varying(3)	暂不支持，值为NO。
avg_col_len	numeric	列的平均长度（单位字节）。
char_length	numeric	列的长度（以字符计），只对varchar，nvarchar2，bpchar，char类型有效。
char_used	character varying(1)	暂不支持，varchar，bpchar，char类型置B，nvarchar2类型置C，其余值为NULL。
v80_fmt_image	character varying(3)	暂不支持，值为NULL。
data_upgraded	character varying(3)	暂不支持，值为YES。
histogram	character varying(15)	表示直方图是否存在以及存在的类型： ● NONE：表示不存在直方图。 ● FREQUENCY：表示频率直方图。 ● EQUI_WIDTH：表示等宽直方图。
default_on_null	character varying(3)	暂不支持，值为NULL。
identity_column	character varying(3)	暂不支持，值为NULL。
evaluation_edition	character varying(128)	暂不支持，值为NULL。

名称	类型	描述
unusable_before	character varying(128)	暂不支持，值为NULL。
unusable_beginning	character varying(128)	暂不支持，值为NULL。
collation	character varying(100)	列的排序规则。因该字段与保留关键字冲突，调用该字段需加视图名。
comments	text	注释。
schema	character varying(64)	列所属的名称空间的名称。

12.3.89 DB_TAB_COMMENTS

DB_TAB_COMMENTS视图显示当前用户可访问的所有表和视图的注释信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-194 DB_TAB_COMMENTS 字段

名称	类型	描述
owner	character varying(128)	表或视图的所有者。
table_name	character varying(128)	表或视图的名称。
table_type	character varying(11)	对象类型。
comments	text	注释。
origin_con_id	numeric	暂不支持，值为0。
schema	character varying(64)	表所属的名称空间的名称。

12.3.90 DB_TAB_HISTOGRAMS

DB_TAB_HISTOGRAMS视图显示当前用户可访问的表或视图的直方图统计信息，即表各列数据的分布情况。所有用户都可以访问该视图。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-195 DB_TAB_HISTOGRAMS 字段

名称	类型	描述
owner	character varying(128)	表的拥有者。
table_name	character varying(128)	表名。
column_name	character varying(4000)	列名。

名称	类型	描述
endpoint_number	numeric	直方图的桶号。
endpoint_value	numeric	暂不支持，值为NULL。
endpoint_actual_value	character varying(4000)	桶端点的实际值。
endpoint_actual_value_raw	raw	暂不支持，值为NULL。
endpoint_repeat_count	numeric	暂不支持，值为NULL。
scope	character varying(7)	暂不支持，值为SHARED。

12.3.91 DB_TAB_PARTITIONS

DB_TAB_PARTITIONS视图显示当前用户所能访问的一级分区信息（包括二级分区表）。所有用户都可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-196 DB_TAB_PARTITIONS 字段

名称	类型	描述
table_owner	character varying(64)	表的所有者。
table_name	character varying(64)	关系表名称。
partition_name	character varying(64)	分区名称。
high_value	text	分区的边界值。 • 对于范围分区和间隔分区，显示各分区的上边界值。 • 对于列表分区，显示各分区的取值列表。 • 对于哈希分区，显示各分区的编号。
tablespace_name	name	分区表的表空间名称。
schema	character varying(64)	名称空间的名称。
subpartition_count	bigint	二级分区的个数。
high_value_length	integer	分区绑定值表达式长度。
composite	character varying(3)	表是否为二级分区表。
partition_position	numeric	分区在表中的位置。

名称	类型	描述
pct_free	numeric	块中可用空间的最小百分比。
pct_used	numeric	暂不支持，值为NULL。
ini_trans	numeric	初始事务数，默认值为4，非USTORE分区表时为NULL。
max_trans	numeric	最大事务数，默认值为128，非USTORE分区表时为NULL。
initial_extent	numeric	暂不支持，值为NULL。
next_extent	numeric	暂不支持，值为NULL。
min_extent	numeric	暂不支持，值为NULL。
max_extent	numeric	暂不支持，值为NULL。
max_size	numeric	暂不支持，值为NULL。
pct_increase	numeric	暂不支持，值为NULL。
freelists	numeric	暂不支持，值为NULL。
freelist_groups	numeric	暂不支持，值为NULL。
logging	character varying(7)	是否记录对表的更改。
compression	character varying(8)	表分区的实际压缩属性。
compress_for	character varying(30)	暂不支持，值为NULL。
num_rows	numeric	分区中的行数。
blocks	numeric	暂不支持，值为NULL。
empty_blocks	numeric	暂不支持，值为NULL。
avg_space	numeric	暂不支持，值为NULL。
chain_cnt	numeric	暂不支持，值为NULL。
avg_row_len	numeric	暂不支持，值为NULL。
sample_size	numeric	暂不支持，值为NULL。
last_analyzed	timestamp with time zone	最近分析此分区的日期。
buffer_pool	character varying(7)	用于分区块的缓冲池。
flash_cache	character varying(7)	暂不支持，值为NULL。
cell_flash_cache	character varying(7)	暂不支持，值为NULL。
global_stats	character varying(3)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。

名称	类型	描述
is_nested	character varying(3)	这是否是嵌套表分区。
parent_table_partition	character varying(128)	暂不支持，值为NULL。
interval	character varying(3)	分区是否在间隔分区表的间隔节中。
segment_created	character varying(4)	表分区是否创建了段或未创建。
indexing	character varying(4)	暂不支持，值为NULL。
read_only	character varying(4)	暂不支持，值为NULL。
inmemory	character varying(8)	暂不支持，值为NULL。
inmemory_priority	character varying(8)	暂不支持，值为NULL。
inmemory_distribute	character varying(15)	暂不支持，值为NULL。
inmemory_compression	character varying(17)	暂不支持，值为NULL。
inmemory_duplicate	character varying(13)	暂不支持，值为NULL。
cellmemory	character varying(24)	暂不支持，值为NULL。
inmemory_service	character varying(12)	暂不支持，值为NULL。
inmemory_service_name	character varying(100)	暂不支持，值为NULL。
memoptimize_read	character varying(8)	暂不支持，值为NULL。
memoptimize_write	character varying(8)	暂不支持，值为NULL。

12.3.92 DB_TAB_STATS_HISTORY

DB_TAB_STATS_HISTORY视图显示表统计信息涉及的表、分区和子分区以及执行收集表统计信息的时间。所有用户都可以访问该视图。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-197 DB_TAB_STATS_HISTORY 字段

名称	类型	描述
owner	character varying(128)	对象的拥有者。

名称	类型	描述
table_name	character varying(128)	表名。
partition_name	character varying(128)	暂不支持，值为NULL。
subpartition_name	character varying(128)	暂不支持，值为NULL。
stats_update_time	timestamp(6) with time zone	统计信息更新的时间，数据库重启后，数据会丢失。

12.3.93 DB_TAB_SUBPARTITIONS

DB_TAB_SUBPARTITIONS视图显示当前用户所能访问的二级分区表的信息。所有用户都可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-198 DB_TAB_SUBPARTITIONS 字段

名称	类型	描述
table_owner	character varying(64)	表的所有者。
table_name	character varying(64)	关系表名称。
partition_name	character varying(64)	分区名称。
subpartition_name	character varying(64)	二级分区名称。
high_value	text	二级分区的边界值。 • 对于范围分区和间隔分区，显示各分区的上边界值。 • 对于列表分区，显示各分区的取值列表。 • 对于哈希分区，显示各分区的编号。
tablespace_name	name	二级分区表的表空间名称。
schema	character varying(64)	名称空间的名称。
high_value_length	integer	二级分区边界值的字符长度。
partition_position	numeric	分区在表中的位置。
subpartition_position	numeric	二级分区在分区中的位置。
pct_free	numeric	块中可用空间的最小百分比。
pct_used	numeric	暂不支持，值为NULL。

名称	类型	描述
ini_trans	numeric	初始事务数，默认值为4，非USTORE分区表时为NULL。
max_trans	numeric	最大事务数，默认值为128，非USTORE分区表时为NULL。
initial_extent	numeric	暂不支持，值为NULL。
next_extent	numeric	暂不支持，值为NULL。
min_extent	numeric	暂不支持，值为NULL。
max_extent	numeric	暂不支持，值为NULL。
max_size	numeric	暂不支持，值为NULL。
pct_increase	numeric	暂不支持，值为NULL。
freelists	numeric	暂不支持，值为NULL。
freelist_groups	numeric	暂不支持，值为NULL。
logging	character varying(3)	是否记录对表的更改。
compression	character varying(8)	表分区的实际压缩属性。
compress_for	character varying(30)	暂不支持，值为NULL。
num_rows	numeric	二级分区中的行数。
blocks	numeric	暂不支持，值为NULL。
empty_blocks	numeric	暂不支持，值为NULL。
avg_space	numeric	暂不支持，值为NULL。
chain_cnt	numeric	暂不支持，值为NULL。
avg_row_len	numeric	暂不支持，值为NULL。
sample_size	numeric	暂不支持，值为NULL。
last_analyzed	timestamp with time zone	最近分析此分区的日期。
buffer_pool	character varying(7)	二级分区的缓冲池。
flash_cache	character varying(7)	暂不支持，值为NULL。
cell_flash_cache	character varying(7)	暂不支持，值为NULL。
global_stats	character varying(3)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。
interval	character varying(3)	分区是否在间隔分区表的间隔节中。
segment_created	character varying(3)	表二级分区是否创建了段。

名称	类型	描述
indexing	character varying(3)	暂不支持，值为NULL。
read_only	character varying(3)	暂不支持，值为NULL。
inmemory	character varying(8)	暂不支持，值为NULL。
inmemory_priority	character varying(8)	暂不支持，值为NULL。
inmemory_distribute	character varying(15)	暂不支持，值为NULL。
inmemory_compression	character varying(17)	暂不支持，值为NULL。
inmemory_duplicate	character varying(13)	暂不支持，值为NULL。
inmemory_service	character varying(12)	暂不支持，值为NULL。
inmemory_service_name	character varying(1000)	暂不支持，值为NULL。
cellmemory	character varying(24)	暂不支持，值为NULL。
memoptimize_read	character varying(8)	暂不支持，值为NULL。
memoptimize_write	character varying(8)	暂不支持，值为NULL。

12.3.94 DB_TABLES

DB_TABLES视图显示当前用户可访问的所有表。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-199 DB_TABLES 字段

名称	类型	描述
owner	character varying(64)	表的所有者。
table_name	character varying(64)	表名。
tablespace_name	character varying(64)	存储表的表空间名称。
num_rows	numeric	表的估计行数。

名称	类型	描述
status	character varying(8)	当前表是否有效。 ● VALID：表示当前表有效。 ● UNUSABLE：表示当前表不可用。
sample_size	numeric	分析表使用的样本数量。
temporary	character(1)	表是否为临时表： ● Y：表示是临时表。 ● N：表示不是临时表。
dropped	character varying	当前表是否已删除： ● YES：表示已删除。 ● NO：表示未删除。
pct_free	numeric	块中空闲空间的最小比例。
ini_trans	numeric	事务的初始数量。
max_trans	numeric	事务数量的最大值。
avg_row_len	integer	平均每行的字节数。
partitioned	character varying(3)	表是否为分区表。 ● YES：是分区表。 ● NO：不是分区表。
last_analyzed	timestamp with time zone	上次分析表的时间。
row_movement	character varying(8)	是否允许分区行移动。 ● ENABLED：允许分区行移动。 ● DISABLED：不允许分区行移动。
compression	character varying(8)	是否启用表压缩。 ● ENABLED：启用表压缩。 ● DISABLED：不启用表压缩。
duration	character varying(15)	临时表的期限。 ● NULL：表示非临时表。 ● sys$session：表示会话临时表。 ● sys$transaction：表示事务临时表。

名称	类型	描述
logical_replication	character varying(8)	表是否启用逻辑复制。 ● ENABLED：启用逻辑复制。 ● DISABLED：不启用逻辑复制。
external	character varying(3)	表是否为外表。 ● YES：是外表。 ● NO：不是外表。
logging	character varying(3)	表的更改是否记入日志。 ● YES：表的更改记录日志。 ● NO：表的更改不记录日志。
default_collation	character varying(100)	表的默认排序规则。 ● default
degree	character varying(10)	扫描表的实例数量。
table_lock	character varying(8)	是否启用表级锁。 ● ENABLED：启用表级锁。 ● DISABLED：不启用表级锁。
nested	character varying(3)	是否为嵌套表。 ● YES：是嵌套表。 ● NO：不是嵌套表。
buffer_pool	character varying(7)	表的默认缓冲池。
flash_cache	character varying(7)	暂不支持，值为NULL。
cell_flash_cache	character varying(7)	暂不支持，值为NULL。
skip_corrupt	character varying(8)	扫描表是否跳过损坏的块。 ● ENABLED：跳过损坏的块。 ● DISABLED：不跳过损坏的块。
has_identity	character varying(3)	表是否具有标识列。 ● YES：有标识列。 ● NO：没有标识列。
segment_created	character varying(3)	表段是否已被创建。 ● YES：表段已被创建。 ● NO：表段未被创建。

名称	类型	描述
monitoring	character varying(3)	是否跟踪表的修改。 ● YES：跟踪表的修改。 ● NO：不跟踪表的修改。
cluster_name	character varying(128)	暂不支持，值为NULL。
iot_name	character varying(128)	暂不支持，值为NULL。
pct_used	numeric	暂不支持，值为NULL。
initial_extent	numeric	暂不支持，值为NULL。
next_extent	numeric	暂不支持，值为NULL。
min_extents	numeric	暂不支持，值为NULL。
max_extents	numeric	暂不支持，值为NULL。
pct_increase	numeric	暂不支持，值为NULL。
freelists	numeric	暂不支持，值为NULL。
freelist_groups	numeric	暂不支持，值为NULL。
backed_up	character varying(1)	暂不支持，值为NULL。
blocks	numeric	暂不支持，值为NULL。
empty_blocks	numeric	暂不支持，值为NULL。
avg_space	numeric	暂不支持，值为NULL。
chain_cnt	numeric	暂不支持，值为NULL。
avg_space_freelist_blocks	numeric	暂不支持，值为NULL。
num_freelist_blocks	numeric	暂不支持，值为NULL。
instances	character varying(10)	暂不支持，值为NULL。
cache	character varying(5)	暂不支持，值为NULL。
iot_type	character varying(12)	暂不支持，值为NULL。
secondary	character varying(1)	暂不支持，值为NULL。
global_stats	character varying(3)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。
cluster_owner	character varying(30)	暂不支持，值为NULL。
dependencies	character varying(8)	暂不支持，值为NULL。
compression_for	character varying(30)	暂不支持，值为NULL。
read_only	character varying(3)	暂不支持，值为NULL。

名称	类型	描述
result_cache	character varying(7)	暂不支持，值为NULL。
clustering	character varying(3)	暂不支持，值为NULL。
activity_tracking	character varying(23)	暂不支持，值为NULL。
dml_timestamp	character varying(25)	暂不支持，值为NULL。
container_data	character varying(3)	暂不支持，值为NULL。
inmemory_priority	character varying(8)	暂不支持，值为NULL。
inmemory_distribute	character varying(15)	暂不支持，值为NULL。
inmemory_compression	character varying(17)	暂不支持，值为NULL。
inmemory_duplicate	character varying(13)	暂不支持，值为NULL。
duplicated	character varying(1)	暂不支持，值为NULL。
sharded	character varying(1)	暂不支持，值为NULL。
hybrid	character varying(3)	暂不支持，值为NULL。
cellmemory	character varying(24)	暂不支持，值为NULL。
containers_default	character varying(3)	暂不支持，值为NULL。
container_map	character varying(3)	暂不支持，值为NULL。
extended_data_link	character varying(3)	暂不支持，值为NULL。
extended_data_link_map	character varying(3)	暂不支持，值为NULL。
inmemory_service	character varying(12)	暂不支持，值为NULL。
inmemory_service_name	character varying(1000)	暂不支持，值为NULL。
container_map_object	character varying(3)	暂不支持，值为NULL。
memoptimize_read	character varying(8)	暂不支持，值为NULL。
memoptimize_write	character varying(8)	暂不支持，值为NULL。
has_sensitive_column	character varying(3)	暂不支持，值为NULL。
admit_null	character varying(3)	暂不支持，值为NULL。
data_link_dml_enabled	character varying(3)	暂不支持，值为NULL。
object_id_type	character varying(16)	暂不支持，值为NULL。
table_type_owner	character varying(128)	暂不支持，值为NULL。
table_type	character varying(128)	暂不支持，值为NULL。

名称	类型	描述
compress_for	character varying(30)	暂不支持，值为NULL。

12.3.95 DB_TRIGGERS

DB_TRIGGERS视图显示当前用户能访问到的触发器信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-200 DB_TRIGGERS 字段

名称	类型	描述
trigger_name	character varying(64)	触发器名称。
table_owner	character varying(64)	角色名称。
table_name	character varying(64)	关系表名称。
status	character varying(64)	触发器的触发状态： ● O：触发器在“origin”和“local”模式下触发。 ● D：触发器被禁用。 ● R：触发器在“replica”模式下触发。 ● A：触发器始终触发。

12.3.96 DB_TYPES

DB_TYPES视图显示当前用户可访问的对象类型的信息。所有用户都可以访问该视图。该视图同时存在于PG_CATALOG和SYS Schema下

表 12-201 DB_TYPES 字段

名称	类型	描述
owner	character varying(128)	类型的所有者。
type_name	character varying(128)	类型名称。
type_oid	raw	类型的标识符（OID）。
typecode	character varying(128)	类型的类型代码。
attributes	numeric	类型中的属性数。
methods	numeric	暂不支持，值为0。
predefined	character varying(3)	表示该类型是否是内置类型。

名称	类型	描述
incomplete	character varying(3)	表示类型是否为不完整类型。
final	character varying(3)	暂不支持，值为NULL。
instantiable	character varying(3)	暂不支持，值为NULL。
persistable	character varying(3)	暂不支持，值为NULL。
supertype_owner	character varying(128)	暂不支持，值为NULL。
supertype_name	character varying(128)	暂不支持，值为NULL。
local_attributes	numeric	暂不支持，值为NULL。
local_methods	numeric	暂不支持，值为NULL。
typeid	raw	暂不支持，值为NULL。

12.3.97 DB_USERS

DB_USERS视图显示记录数据库中所有用户，但不对用户信息进行详细的描述。默认只有系统管理员可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-202 DB_USERS 字段

名称	类型	描述
user_id	oid	用户的OID。
username	name	用户的名称。

12.3.98 DB_VIEWS

DB_VIEWS视图显示当前用户可访问的所有视图描述信息。该视图同时存在于PG_CATALOG和SYS schema下。

表 12-203 DB_VIEWS 字段

名称	类型	描述
owner	name	视图的所有者。
view_name	name	视图的名称。
text	text	视图文本。
text_length	integer	视图文本长度。

名称	类型	描述
TEXT_VC	character varying(4000)	视图创建语句。此列可能会截断视图文本。BEQUEATH子句将不会作为此视图中的TEXT_VC列的一部分出现。
type_text_length	numeric	暂不支持，值为NULL。
type_text	character varying(4000)	暂不支持，值为NULL。
oid_text_length	numeric	暂不支持，值为NULL。
oid_text	character varying(4000)	暂不支持，值为NULL。
view_type_owner	character varying(128)	暂不支持，值为NULL。
view_type	character varying(128)	暂不支持，值为NULL。
superview_name	character varying(128)	暂不支持，值为NULL。
editioning_view	character varying(1)	暂不支持，值为NULL。
read_only	character varying(1)	暂不支持，值为NULL。
container_data	character varying(1)	暂不支持，值为NULL。
bequeath	character varying(12)	暂不支持，值为NULL。
origin_con_id	character varying(256)	暂不支持，值为NULL。
default_collation	character varying(100)	暂不支持，值为NULL。
containers_default	character varying(3)	暂不支持，值为NULL。
container_map	character varying(3)	暂不支持，值为NULL。
extended_data_link	character varying(3)	暂不支持，值为NULL。
extended_data_link_map	character varying(3)	暂不支持，值为NULL。

名称	类型	描述
has_sensitive_column	character varying(3)	暂不支持，值为NULL。
admit_null	character varying(3)	暂不支持，值为NULL。
pdb_local_only	character varying(3)	暂不支持，值为NULL。

12.3.99 DICT

DICT视图显示数据库中的数据字典表和系统视图的描述信息。所有用户都可以访问，该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-204 DICT 字段

名称	类型	描述
table_name	character varying(128)	对象的名称。
comments	character varying(4000)	对象上的文本注释。

12.3.100 DICTIONARY

DICTIONARY视图显示数据库中的数据字典表和系统视图的描述信息。所有用户都可以访问，该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-205 DICTIONARY 字段

名称	类型	描述
table_name	character varying(128)	对象的名称。
comments	character varying(4000)	对象上的文本注释。

12.3.101 DV_SESSION_LONGOPS

DV_SESSION_LONGOPS视图显示当前正在执行的操作的进度。该视图需要授权访问。

表 12-206 DV_SESSION_LONGOPS 字段

名称	类型	描述
sid	bigint	当前正在执行的后台进程的OID。
serial#	integer	当前正在执行的后台进程的序号，在GaussDB中为0。
sofar	integer	目前完成的工作量，在GaussDB中为空。
totalwork	integer	工作总量，在GaussDB中为空。

12.3.102 DV_SESSIONS

DV_SESSIONS视图显示当前所有活动的后台线程的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。

表 12-207 DV_SESSIONS 字段

名称	类型	描述
sid	bigint	当前活动的后台线程的OID。
serial#	integer	当前活动的后台线程的序号，在GaussDB中为0。
user#	oid	登录此后台线程的用户的OID。oid为0表示此后台线程为全局辅助线程（auxiliary）。
username	name	登录此后台线程的用户名。username为空表示此后台线程为全局辅助线程（auxiliary）。 可以通过和pg_stat_get_activity()关联查询，识别出application_name。 例如： SELECT s.*,a.application_name FROM DV_SESSIONS AS s LEFT JOIN pg_stat_get_activity(NULL) AS a ON s.sid=a.sessionid;

12.3.103 GS_ALL_CONTROL_GROUP_INFO

GS_ALL_CONTROL_GROUP_INFO视图显示数据库内所有的控制组信息。

表 12-208 GS_ALL_CONTROL_GROUP_INFO 字段

名称	类型	描述
name	text	控制组的名称。

名称	类型	描述
type	text	控制组的类型。 ● GROUP_NONE：无分组。 ● GROUP_TOP：顶级分组。 ● GROUP_CLASS：该资源的类分组，不控制任何线程。 ● GROUP_BAKWD：后端线程控制组。 ● GROUP_DEFWD：默认控制组，仅控制该级别的查询线程。 ● GROUP_TSWD：每个用户的分时控制组，控制最底层的查询线程。
gid	bigint	控制组ID。
classgid	bigint	Workload所属Class的控制组ID。
class	text	Class控制组。
workload	text	Workload控制组。
shares	bigint	控制组分配的CPU资源配额。
limits	bigint	控制组分配的CPU资源限额。
wdlevel	bigint	Workload控制组层级。
cpucores	text	控制组使用的CPU核的信息。

12.3.104 GS_ALL_PREPARED_STATEMENTS

GS_ALL_PREPARED_STATEMENTS视图显示所有会话中可用的预备语句的信息。默认只有系统管理员权限才可以访问此视图。

表 12-209 GS_ALL_PREPARED_STATEMENTS 字段

名称	类型	描述
pid	bigint	后台线程ID。 **说明** 线程池模式下pid显示的是当前会话绑定的线程ID，当会话在不同线程上执行时pid会随之改变。线程池模式下statement与sessionid相关联，与pid无关联，关联查询时建议使用sessionid。
sessionid	bigint	当前会话ID。
global_sessionid	text	全局会话ID。
name	text	预备语句的标识符。

名称	类型	描述
statement	text	创建该预备语句的查询字符串。 ● 对于从SQL创建的预备语句而言是客户端提交的PREPARE语句。 ● 对于通过前/后端协议创建的预备语句而言是预备语句自身的文本。
prepare_time	timestamp with time zone	创建该预备语句的时间戳。
parameter_types	regtype[]	该预备语句期望的参数类型，以regtype类型的数组格式出现。与该数组元素相对应的OID可以通过把regtype转换为OID值得到。
from_sql	boolean	● 如果该预备语句是通过PREPARE语句创建的则为true。 ● 如果是通过前/后端协议创建的则为false。

12.3.105 GS_AUDITING

GS_AUDITING视图显示对数据库相关操作的所有审计信息。需要有系统管理员或安全策略管理员权限才可以访问此视图。

表 12-210 GS_AUDITING 字段

名称	类型	描述
polname	name	策略名称，需要唯一，不可重复。
pol_type	text	审计策略类型。取值范围： ● access：审计DML操作。 ● privilege：审计DDL操作。
polenabled	boolean	策略是否启动。 ● t（true）：启动。 ● f（false）：不启动。
access_type	name	DML数据库操作相关类型。例如SELECT、INSERT、DELETE等。
label_name	name	资源标签名称。对应系统表gs_auditing_policy中的polname字段。
priv_object	text	数据库资产的路径。
filter_name	text	过滤条件的逻辑字符串。

12.3.106 GS_AUDITING_ACCESS

GS_AUDITING_ACCESS视图显示对数据库DML相关操作的所有审计信息。需要有系统管理员或安全策略管理员权限才可以访问此视图。

表 12-211 GS_AUDITING_ACCESS 字段

名称	类型	描述
polname	name	策略名称，需要唯一，不可重复。
pol_type	text	审计策略类型，值为‘access’，表示审计DML操作。
polenabled	boolean	策略是否启动。 ● t（true）：启动。 ● f（false）：不启动。
access_type	name	DML数据库操作相关类型。例如SELECT、INSERT、DELETE等。
label_name	name	资源标签名称。对应系统表gs_auditing_policy中的polname字段。
access_object	text	数据库资产的路径。
filter_name	text	过滤条件的逻辑字符串。

12.3.107 GS_AUDITING_PRIVILEGE

GS_AUDITING_PRIVILEGE视图显示对数据库DDL相关操作的所有审计信息。需要有系统管理员或安全策略管理员权限才可以访问此视图。

表 12-212 GS_AUDITING_PRIVILEGE 字段

名称	类型	描述
polname	name	策略名称，需要唯一，不可重复。
pol_type	text	审计策略类型，值为‘privilege’，表示审计DDL操作。
polenabled	boolean	策略是否启动。 ● t（true）：表示启动。 ● f（false）：表示不启动。
access_type	name	DDL数据库操作相关类型。例如CREATE、ALTER、DROP等。
label_name	name	资源标签名称。对应系统表gs_auditing_policy中的polname字段。

名称	类型	描述
priv_object	text	包含数据库对象的全称域名。
filter_name	text	过滤条件的逻辑字符串。

12.3.108 GS_CLUSTER_RESOURCE_INFO

集中式不支持该视图。

12.3.109 GS_COMM_LISTEN_ADDRESS_EXT_INFO

GS_COMM_LISTEN_ADDRESS_EXT_INFO描述了查询连接扩展IP的相关线程、会话、socket等DFX信息。当前不支持查询该视图。

表 12-213 GS_COMM_LISTEN_ADDRESS_EXT_INFO 字段

名称	类型	描述
node_name	text	当前实例名。
app	text	当前连接DN的客户端。
tid	bigint	当前线程的线程号。
lwtid	integer	当前线程的轻量级线程号。
query_id	bigint	当前线程的查询ID。
socket	integer	当前物理连接的socket fd。
remote_ip	text	当前连接对端IP。
remote_port	text	当前连接对端port。
local_ip	text	当前连接本端IP。
local_port	text	当前连接本端port。

12.3.110 GS_COMM_PROXY_THREAD_STATUS

GS_COMM_PROXY_THREAD_STATUS视图用来显示代理通信库comm_proxy的数据收发包统计。只有集中式数据库在安装阶段启动用户态网络部署形态，同时comm_proxy_attr参数的enable_dfx配置为true，才会具体显示数据comm_proxy的数据收发统计，其他场景该视图不支持查询。

表 12-214 GS_COMM_PROXY_THREAD_STATUS 字段

名称	类型	描述
ProxyThreadId	bigint	当前网络代理线程comm_proxy的线程id。

名称	类型	描述
ProxyCpuAffinity	text	当前网络代理线程comm_proxy的NUMA-CPU亲和性，表示所在NUMA和CPU ID。
ThreadStartTime	text	当前网络代理线程comm_proxy的启动时间。
RxPckNums	bigint	当前网络代理线程comm_proxy收包数量。
TxPckNums	bigint	当前网络代理线程comm_proxy发包数量。
RxPcks	float	当前网络代理线程comm_proxy每秒收包数量。
TxPcks	float	当前网络代理线程comm_proxy每秒发包数量。

12.3.111 GS_DB_LINKS

GS_DB_LINKS系统视图存储DATABASE LINK对象的相关信息，用户可以查看属于自己和PUBLIC级别的DATABASE LINK信息。

表 12-215 GS_DB_LINKS 字段

名称	类型	描述
dblinkid	oid	当前DATABASE LINK对象的OID。
dlname	name	当前DATABASE LINK对象的名称。
dlowner	oid	当前DATABASE LINK对象拥有者的ID。对象拥有者为public时值为0。
dlownername	name	当前DATABASE LINK对象拥有者的名称。
options	text[]	当前DATABASE LINK对象的连接信息，使用“keyword=value”格式的字符串。
useroptions	text	当前DATABASE LINK对象连接的远端用户信息。
heterogeneous	text	暂不支持，值为NULL。
protocol	text	暂不支持，值为NULL。
opencursors	text	暂不支持，值为NULL。
intransaction	boolean	当前DATABASE LINK对象是否在事务中。
updatasent	boolean	当前DATABASE LINK对象是否使用了更新数据的语句。

12.3.112 GS_DB_PRIVILEGES

GS_DB_PRIVILEGES系统视图记录ANY权限的授予情况，每条记录对应一条授权信息。

表 12-216 GS_DB_PRIVILEGES 字段

名称	类型	描述
rolename	name	用户名。
privilege_type	text	用户拥有的ANY权限，取值参考**表 7-149**。
admin_option	text	是否具有privilege_type列记录的ANY权限的再授权权限。 • yes：表示具有。 • no：表示不具有。

12.3.113 GS_FILE_STAT

GS_FILE_STAT视图通过对数据文件I/O的统计，反映数据的I/O性能，用以发现I/O操作异常等性能问题。

表 12-217 GS_FILE_STAT 字段

名称	类型	描述
filenum	oid	文件标识。
dbid	oid	数据库标识。
spcid	oid	表空间标识。
phyrds	bigint	读物理文件的数目。
phywrts	bigint	写物理文件的数目。
phyblkrd	bigint	读物理文件块的数目。
phyblkwrt	bigint	写物理文件块的数目。
readtim	bigint	读文件的总时长，单位微秒。
writetim	bigint	写文件的总时长，单位微秒。
avgiotim	bigint	读写文件的平均时长，单位微秒。
lstiotim	bigint	最后一次读文件时长，单位微秒。
miniotim	bigint	读写文件的最小时长，单位微秒。
maxiowtm	bigint	读写文件的最大时长，单位微秒。

12.3.114 GS_GET_CONTROL_GROUP_INFO

集中式不支持该视图。

表 12-218 GS_GET_CONTROL_GROUP_INFO 字段

名称	类型	描述
group_name	text	控制组的名称。
group_type	text	控制组的类型。 ● GROUP_NONE：无分组。 ● GROUP_TOP：顶级分组。 ● GROUP_CLASS：该资源的类分组，不控制任何线程。 ● GROUP_BAKWD：后端线程控制组。 ● GROUP_DEFWD：默认控制组，仅控制该级别的查询线程。 ● GROUP_TSWD：每个用户的分时控制组，控制最底层的查询线程。
gid	bigint	控制组ID。
classgid	bigint	Workload所属Class的控制组ID。
class	text	Class控制组。
group_workload	text	Workload控制组。
shares	bigint	控制组分配的CPU资源配额。
limits	bigint	控制组分配的CPU资源限额。
wdlevel	bigint	Workload控制组层级。
cpucores	text	控制组使用的CPU核的信息。
nodegroup	text	node group的名称。
group_kind	text	node group的类型，取值包括i，n，v，e。 ● i：表示installation node group。 ● n：表示普通集群node group。 ● e：表示弹性集群。

12.3.115 GS_GET_LISTEN_ADDRESS_EXT_INFO

GS_GET_LISTEN_ADDRESS_EXT_INFO视图描述了查询当前实例扩展IP配置信息。当前不支持查询该视图。

表 12-219 GS_GET_LISTEN_ADDRESS_EXT_INFO 字段

名称	类型	描述
node_name	text	当前实例名。
host	text	当前实例侦听IP地址。
port	bigint	当前实例侦听的port。
ext_listen_ip	text	当前实例配置扩展IP地址。

12.3.116 GS_GLC_MEMORY_DETAIL

GS_GLC_MEMORY_DETAIL视图显示所有数据库内global plpgsql cache全局缓存的内存使用情况，仅在enable_global_plsqlcache=on时可用。

表 12-220 GS_GLC_MEMORY_DETAIL 字段

名称	类型	描述
contextname	text	内存对象名。
database	text	内存对象所属的数据库，"pkg_bucket" 与"func_bucket"显示为"NULL"。
schema	text	内存对象所属的模式，"pkg_bucket" 与"func_bucket"显示为"NULL"。
type	text	对象类型： ● "pkg_bucket"：代表该对象为package对象的父节点。 ● "func_bucket"：代表该对象为函数或存储过程的父节点。 ● "pkg"：代表该对象为package对象。 ● "func"：代表该对象为函数或存储过程。
status	text	缓存对象当前状态，可用显示为"valid"，不可用显示为"invalid"。"pkg_bucket"与"func_bucket"对象无状态，显示为"NULL"。如果缓存对象为package，会显示缓存对象是包头还是包体，以"valid"为例，包头显示为"valid:spec"，包体显示为"valid:body"。
location	text	缓存对象当前位置，在缓存哈希表内显示为"in_global_hash_table"，在失效链表中显示为"in_global_expired_list"。 "pkg_bucket"与"func_bucket"对象显示为"NULL"。
env	int8	创建对象时的环境参数，即behavior_compat_flags的值。"pkg_bucket" 与"func_bucket"对象显示为0。
usedsize	int8	缓存对象大小。

名称	类型	描述
usecount	int8	有多少对象正在引用该全局缓存，当没有对象引用该全局编译产物时，数量为0。

12.3.117 GS_GLOBAL_ARCHIVE_STATUS

GS_GLOBAL_ARCHIVE_STATUS视图描述DN和所有分片的归档进度，获取分片名称（node_name）、归档位置（restart_lsn），实际进行归档的主/备机名称（archive_node）和当前日志位置（current_xlog_location）。查询此视图需要数据库开启归档功能，并从主DN节点进行查询。

表 12-221 GS_GLOBAL_ARCHIVE_STATUS 字段

名称	类型	描述
node_name	text	分片名称。
restart_lsn	text	归档位置。
archive_node	text	实际进行归档的主/备机名称。
current_xlog_location	text	当前日志位置。

12.3.118 GS_GSC_MEMORY_DETAIL

GS_GSC_MEMORY_DETAIL视图显示当前节点当前进程的全局SysCache的内存占用情况，仅在开启GSC的模式下有数据。

需要注意的是，由于这个查询是以数据库内存上下文分隔的，因此会缺少一部分内存的统计，缺失的内存统计对应的内存上下文名称为GlobalSysDBCache。

表 12-222 GS_GSC_MEMORY_DETAIL 字段

名称	类型	描述
db_id	text	数据库id。
totalsize	numeric	共享内存总大小，单位Byte。
freesize	numeric	共享内存剩余大小，单位Byte。
usedsize	numeric	共享内存使用大小，单位Byte。

12.3.119 GS_INSTANCE_TIME

提供当前集节点下的各种时间消耗信息，主要分为以下类型:

- DB_TIME: 作业在多核下的有效时间花销。
- CPU_TIME：CPU的时间花销。
- EXECUTION_TIME：执行器内的时间花销。
- PARSE_TIME：SQL解析的时间花销。
- PLAN_TIME：生成Plan的时间花销。
- REWRITE_TIME：SQL重写的时间花销。
- PL_EXECUTION_TIME ： PL/SQL（存储过程）执行的时间花销。
- PL_COMPILATION_TIME：PL/SQL（存储过程）编译的时间花销。
- NET_SEND_TIME：网络上的时间花销。
- DATA_IO_TIME：I/O的时间花销。

表 12-223 GS_INSTANCE_TIME 字段

名称	类型	描述
stat_id	integer	统计编号。
stat_name	text	类型名称。
value	bigint	时间值（单位：微秒）。

12.3.120 GS_LABELS

GS_LABELS视图显示所有已配置的资源标签信息。需要有系统管理员或安全策略管理员权限才可以访问此视图。

表 12-224 GS_LABELS 字段

名称	类型	描述
labelname	name	资源标签的名称。
labeltype	name	资源标签的类型。对应系统表GS_POLICY_LABEL中的labeltype字段。
fqdntype	name	数据库资源的类型。如table、schema、index等。
schemaname	name	数据库资源所属的schema名称。
fqdnname	name	数据库资源名称。
columnname	name	数据库资源列名称，若标记的数据库资源不为表的列则该项为空。

12.3.121 GS_LSC_MEMORY_DETAIL

GS_LSC_MEMORY_DETAIL视图显示所有线程的本地SysCache的内存占用情况，以MemoryContext节点来统计，仅在开启GSC的模式下有数据。

表 12-225 GS_LSC_MEMORY_DETAIL 字段

名称	类型	描述
threadid	text	线程启动时间+线程标识（字符串信息为timestamp.sessionid）。
tid	bigint	线程标识。
thrdtype	text	线程类型。可以是系统内存在的任何线程类型，如postgresql、wlmmonitor等。
contextname	text	内存上下文名称。
level	smallint	当前上下文在整体内存上下文中的层级。
parent	text	父内存上下文名称。
totalsize	bigint	当前内存上下文的内存总数，单位Byte。
freesize	bigint	当前内存上下文中已释放的内存总数，单位Byte。
usedsize	bigint	当前内存上下文中已使用的内存总数，单位Byte。

12.3.122 GS_MASKING

GS_MASKING视图显示所有已配置的动态脱敏策略信息。需要有系统管理员或安全策略管理员权限才可以访问此视图。

表 12-226 GS_MASKING 字段

名称	类型	描述
polname	name	脱敏策略名称。
polenabled	boolean	脱敏策略开关。
maskaction	name	脱敏函数。
labelname	name	脱敏函数作用的标签名称。
masking_object	text	脱敏数据库资源对象。
filter_name	text	过滤条件的逻辑表达式。

12.3.123 GS_MATVIEWS

GS_MATVIEWS视图提供了关于数据库中每一个物化视图的信息。

表 12-227 GS_MATVIEWS 字段

名称	类型	引用	描述
schemaname	name	PG_NAMESPACE.nspname	物化视图的模式名。
matviewname	name	PG_CLASS.relname	物化视图名。
matviewowner	name	PG_AUTHID.Erolname	物化视图的所有者。
tablespace	name	PG_TABLESPACE.spcname	物化视图的表空间名（如果使用数据库默认表空间则为空）。
hasindexes	boolean	-	如果物化视图有（或者最近有过）任何索引，则此列为真。
definition	text	-	物化视图的定义（一个重构的SELECT查询）。

12.3.124 GS_MY_PLAN_TRACE

GS_MY_PLAN_TRACE是系统表GS_PLAN_TRACE的视图，该视图主要用来查看当前用户的plan trace。

表 12-228 GS_MY_PLAN_TRACE 字段

名称	类型	描述
query_id	text	当前请求的唯一id。
query	text	当前请求的sql语句，该字段大小不会超过系统参数track_activity_query_size指定的大小。
unique_sql_id	bigint	当前请求sql的唯一id。
plan	text	当前请求sql对应的查询计划文本。该字段大小不会超过10K。
plan_trace	text	当前请求sql对应的查询计划生成过程的明细，该字段大小不会超过300M。
modifydate	timestamp with time zone	当前plan trace的更新时间（当前指的是 plan trace创建时间）。

12.3.125 GS_OS_RUN_INFO

GS_OS_RUN_INFO视图显示当前操作系统运行状态的信息。

表 12-229 GS_OS_RUN_INFO 字段

名称	类型	描述
id	integer	编号。
name	text	操作系统运行状态名称。
value	numeric	操作系统运行状态值。
comments	text	操作系统运行状态注释。
cumulative	boolean	操作系统运行状态的值是否为累加值。

12.3.126 GS_REDO_STAT

GS_REDO_STAT视图显示会话线程的日志回放情况。

表 12-230 GS_REDO_STAT 字段

名称	类型	描述
phywrts	bigint	日志回放过程中写数据的次数。
phyblkwrt	bigint	日志回放过程中写数据的块数。
writetim	bigint	日志回放过程中写数据所耗的总时间。
avgiotim	bigint	日志回放过程中写一次数据的平均消耗时间。
lstiotim	bigint	日志回放过程中最后一次写数据消耗的时间。
miniotim	bigint	日志回放过程中单次写数据消耗的最短时间。
maxiowtm	bigint	日志回放过程中单次写数据消耗的最长时间。

12.3.127 GS_SESSION_ALL_SETTINGS

GS_SESSION_ALL_SETTINGS显示本节点上所有session的全量GUC参数配置。该视图只有sysadmin或者monadmin权限可以查看。

表 12-231 GS_SESSION_ALL_SETTINGS 字段

名称	类型	描述
sessionid	bigint	会话的ID。
pid	bigint	后端线程的ID。
name	text	参数名称。
setting	text	参数当前值。

名称	类型	描述
unit	text	参数的隐式单位。

12.3.128 GS_SESSION_MEMORY

GS_SESSION_MEMORY视图显示Session级别的内存使用情况，包含执行作业在数据节点上gaussdb线程和Stream线程分配的所有内存。当GUC参数enable_memory_limit的值为off时，本视图不可用。

表 12-232 GS_SESSION_MEMORY 字段

名称	类型	描述
sessid	text	线程启动时间+线程标识。
init_mem	integer	当前正在执行作业进入执行器前已分配的内存，单位MB。
used_mem	integer	当前正在执行作业已分配的内存，单位MB。
peak_mem	integer	当前正在执行作业已分配的内存峰值，单位MB。

12.3.129 GS_SESSION_MEMORY_CONTEXT

GS_SESSION_MEMORY_CONTEXT视图显示所有会话的内存使用情况，以MemoryContext节点来统计。当GUC参数enable_memory_limit或enable_thread_pool的值为off时，本视图不可用。

其中内存上下文“TempSmallContextGroup”，记录当前线程中所有内存上下文字段“totalsize”小于8192字节的信息汇总，并且内存上下文统计计数记录到“usedsize”字段中。所以在视图中，“TempSmallContextGroup”内存上下文中的“totalsize”和“freesize”是该线程中所有内存上下文“totalsize”小于8192字节的汇总总和，usedsize字段表示统计的内存上下文个数。

表 12-233 GS_SESSION_MEMORY_CONTEXT 字段

名称	类型	描述
sessid	text	会话启动时间+会话标识（字符串信息为timestamp.sessionid）。
threadid	bigint	会话绑定的线程标识，如果未绑定线程，该值为-1。
contextname	text	内存上下文名称。
level	smallint	当前上下文在整体内存上下文中的层级。
parent	text	父内存上下文名称。

名称	类型	描述
totalsize	bigint	当前内存上下文的内存总数，单位Byte。
freesize	bigint	当前内存上下文中已释放的内存总数，单位Byte。
usedsize	bigint	当前内存上下文中已使用的内存总数，单位Byte；“TempSmallContextGroup”内存上下文中该字段含义为统计计数。

⚠ 注意

该视图为运维视图，用于定位内存问题时使用，不要并发查询该视图，并发查询该视图会随着并发数的增多导致新连接接入等待时间增加，长时间无法接入。

12.3.130 GS_SESSION_MEMORY_DETAIL

GS_SESSION_MEMORY_DETAIL显示会话的内存使用情况，以MemoryContext节点来统计。当开启线程池（enable_thread_pool = on）时，该视图包含所有线程和会话的内存使用情况。当GUC参数enable_memory_limit的值为off时，本视图不可用。

其中内存上下文“TempSmallContextGroup”，记录当前线程中所有内存上下文字段“totalsize”小于8192字节的信息汇总，并且内存上下文统计计数记录到“usedsize”字段中。所以在视图中，“TempSmallContextGroup”内存上下文中的“totalsize”和“freesize”是该线程中所有内存上下文“totalsize”小于8192字节的汇总总和，usedsize字段表示统计的内存上下文个数。

可通过"SELECT * FROM gs_session_memctx_detail(threadid, '');"将某个线程所有内存上下文信息记录到“$GAUSSLOG/pg_log/${node_name}/dumpmem”目录下的“threadid_timestamp.log”文件中。其中threadid可通过下表sessid中获得。

表 12-234 GS_SESSION_MEMORY_DETAIL 字段

名称	类型	描述
sessid	text	1. 关闭线程池（enable_thread_pool = off）时该字段表示线程启动时间+session标识（字符串信息为timestamp.sessionid）。 2. 开启线程池（enable_thread_pool = on）时，内存上下文是线程级别的，则对应的该字段表示线程启动时间+线程标识（字符串信息为timestamp.threadid），内存上下文是session级别的，则对应的该字段表示线程启动时间+session标识（字符串信息为timestamp.sessionid）。
sesstype	text	线程名称。
contextname	text	内存上下文名称。

名称	类型	描述
level	smallint	当前上下文在整体内存上下文中的层级。
parent	text	父内存上下文名称。
totalsize	bigint	当前内存上下文的内存总数，单位Byte。
freesize	bigint	当前内存上下文中已释放的内存总数，单位Byte。
usedsize	bigint	当前内存上下文中已使用的内存总数，单位Byte；“TempSmallContextGroup”内存上下文中该字段含义为统计计数。

⚠ 注意

该视图为运维视图，用于定位内存问题时使用，不要并发查询该视图，并发查询该视图会随着并发数的增多导致新连接接入等待时间增加，长时间无法接入。

12.3.131 GS_SESSION_STAT

GS_SESSION_STAT视图以会话线程或AutoVacuum线程为单位，显示会话状态信息。

表 12-235 GS_SESSION_STAT 字段

名称	类型	描述
sessid	text	线程标识+线程启动时间。
statid	integer	统计会话编号。
statname	text	统计会话名称。
statunit	text	统计会话单位。
value	bigint	统计会话值。

12.3.132 GS_SESSION_TIME

GS_SESSION_TIME视图显示会话线程的运行时间信息，及各执行阶段所消耗时间。

表 12-236 GS_SESSION_TIME 字段

名称	类型	描述
sessid	text	线程标识+线程启动时间。
stat_id	integer	统计会话编号。

名称	类型	描述
stat_name	text	统计会话类型名称。
value	bigint	统计会话值。

12.3.133 GS_SQL_COUNT

GS_SQL_COUNT视图显示数据库当前节点当前时刻执行的五类语句（SELECT、INSERT、UPDATE、DELETE、MERGE INTO）统计信息。

- 普通用户查询GS_SQL_COUNT视图仅能看到该用户当前节点的统计信息；管理员权限用户查询GS_SQL_COUNT视图则能看到所有用户当前节点的统计信息。
- 当数据库或该节点重启时，计数将清零，并重新开始计数 。
- 计数以节点收到的查询数为准，包括数据库内部进行的查询。

表 12-237 GS_SQL_COUNT 字段

名称	类型	描述
node_name	name	节点名称。
user_name	name	用户名。
select_count	bigint	SELECT语句统计结果。
update_count	bigint	UPDATE语句统计结果。
insert_count	bigint	INSERT语句统计结果。
delete_count	bigint	DELETE语句统计结果。
mergeinto_count	bigint	MERGE INTO语句统计结果。
ddl_count	bigint	DDL语句的数量。
dml_count	bigint	DML语句的数量。
dcl_count	bigint	DML语句的数量。
total_select_elapse	bigint	总SELECT的时间花费（单位：微秒）。
avg_select_elapse	bigint	平均SELECT的时间花费（单位：微秒）。
max_select_elapse	bigint	最大SELECT的时间花费（单位：微秒）。
min_select_elapse	bigint	最小SELECT的时间花费（单位：微秒）。
total_update_elapse	bigint	总UPDATE的时间花费（单位：微秒）。
avg_update_elapse	bigint	平均UPDATE的时间花费（单位：微秒）。

名称	类型	描述
max_update_elapse	bigint	最大UPDATE的时间花费（单位：微秒）。
min_update_elapse	bigint	最小UPDATE的时间花费（单位：微秒）。
total_insert_elapse	bigint	总INSERT的时间花费（单位：微秒）。
avg_insert_elapse	bigint	平均INSERT的时间花费（单位：微秒）。
max_insert_elapse	bigint	最大INSERT的时间花费（单位：微秒）。
min_insert_elapse	bigint	最小INSERT的时间花费（单位：微秒）。
total_delete_elapse	bigint	总DELETE的时间花费（单位：微秒）。
avg_delete_elapse	bigint	平均DELETE的时间花费（单位：微秒）。
max_delete_elapse	bigint	最大DELETE的时间花费（单位：微秒）。
min_delete_elapse	bigint	最小DELETE的时间花费（单位：微秒）。

12.3.134 GS_STAT_ALL_PARTITIONS

GS_STAT_ALL_PARTITIONS视图包含当前数据库中所有分区表每个分区的信息，每个分区各占一行，显示该分区访问情况的统计信息，此视图信息通过gs_stat_get_all_partitions_stats()函数查询。

表 12-238 GS_STAT_ALL_PARTITIONS 字段

名称	类型	描述
partition_oid	oid	分区的OID。
schemaname	name	该分区所在表的模式名。
relname	name	该分区所在表的表名。
partition_name	name	该分区所在一级分区名。
sub_partition_name	name	该分区所在二级分区名。
seq_scan	bigint	该分区发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。

名称	类型	描述
idx_scan	bigint	该分区发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	热更新行数（比如没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计非活跃行数。
last_vacuum	timestamp with time zone	该分区最后一次被清理的时间。
last_autovacuum	timestamp with time zone	该分区最后一次被autovacuum守护线程清理的时间。
last_analyze	timestamp with time zone	该分区最后一次被分析的时间。
last_autoanalyze	timestamp with time zone	该分区最后一次被autovacuum守护线程分析的时间。
vacuum_count	bigint	该分区被清理的次数。
autovacuum_count	bigint	该分区被autovacuum守护线程清理的次数。
analyze_count	bigint	该分区被分析的次数。
autoanalyze_count	bigint	该分区被autovacuum守护线程分析的次数。

12.3.135 GS_STAT_XACT_ALL_PARTITIONS

GS_STAT_XACT_ALL_PARTITIONS视图显示命名空间中所有分区表分区的事务状态信息，此视图信息通过gs_stat_get_xact_all_partitions_stats()函数查询。

表 12-239 GS_STAT_XACT_ALL_PARTITIONS 字段

名称	类型	描述
partition_oid	oid	分区的OID。
schemaname	name	该分区的模式名。
relname	name	该分区所在表的表名。

名称	类型	描述
partition_name	name	该分区所在一级分区名。
sub_partition_name	name	该分区所在二级分区名。
seq_scan	bigint	该分区发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该分区发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	热更新行数（比如没有更新所需的单独索引）。

12.3.136 GS_STATIO_ALL_PARTITIONS

GS_STATIO_ALL_PARTITIONS视图包含当前数据库中每个分区表分区的I/O统计信息，此视图信息由gs_statio_get_all_partitions_stats()函数查询得到。

表 12-240 GS_STATIO_ALL_PARTITIONS 字段

名称	类型	描述
partition_oid	oid	分区OID。
schemaname	name	该分区模式名。
relname	name	该分区所在表的表名。
partition_name	name	该分区所在一级分区名。
sub_partition_name	name	该分区所在二级分区名。
heap_blks_read	bigint	从该分区中读取的磁盘块数。
heap_blks_hit	bigint	该分区缓存命中数。
idx_blks_read	bigint	从分区中所有索引读取的磁盘块数。
idx_blks_hit	bigint	分区中所有索引命中缓存数。
toast_blks_read	bigint	该分区的TOAST表分区读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该分区的TOAST表分区命中缓冲区数（如果存在）。

名称	类型	描述
tidx_blks_read	bigint	该分区的TOAST表分区索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该分区的TOAST表分区索引命中缓冲区数（如果存在）。

12.3.137 GS_THREAD_MEMORY_CONTEXT

GS_THREAD_MEMORY_CONTEXT视图显示所有线程的内存使用情况，以MemoryContext节点来统计。该视图在关闭线程池（enable_thread_pool = off）时等价于GS_SESSION_MEMORY_DETAIL视图。当GUC参数enable_memory_limit的值为off时，本视图不可用。

其中内存上下文“TempSmallContextGroup”，记录当前线程中所有内存上下文字段“totalsize”小于8192字节的信息汇总，并且内存上下文统计计数记录到“usedsize”字段中。所以在视图中，“TempSmallContextGroup”内存上下文中的“totalsize”和“freesize”是该线程中所有内存上下文“totalsize”小于8192字节的汇总总和，usedsize字段表示统计的内存上下文个数。

表 12-241 GS_THREAD_MEMORY_CONTEXT 字段

名称	类型	描述
threadid	text	线程启动时间+线程标识（字符串信息为timestamp.sessionid）。
tid	bigint	线程标识。
thrdtype	text	线程类型。
contextname	text	内存上下文名称。
level	smallint	当前上下文在整体内存上下文中的层级。
parent	text	父内存上下文名称。
totalsize	bigint	当前内存上下文的内存总数，单位Byte。
freesize	bigint	当前内存上下文中已释放的内存总数，单位Byte。
usedsize	bigint	当前内存上下文中已使用的内存总数，单位Byte；“TempSmallContextGroup”内存上下文中该字段含义为统计计数。

12.3.138 GS_TOTAL_MEMORY_DETAIL

GS_TOTAL_MEMORY_DETAIL视图显示当前数据库节点的内存使用情况，单位为MB。当GUC参数enable_memory_limit的值为off时，本视图不可用。

表 12-242 GS_TOTAL_MEMORY_DETAIL 字段

名称	类型	描述
nodename	text	节点名称。
memorytype	text	内存类型，包括以下几种： ● max_process_memory：GaussDB实例所占用的内存大小。 ● process_used_memory：GaussDB进程所使用的内存大小。 ● max_dynamic_memory：最大动态内存。 ● dynamic_used_memory：已使用的动态内存。 ● dynamic_peak_memory：内存的动态峰值。 ● dynamic_used_shrctx：最大动态共享内存上下文。 ● dynamic_peak_shrctx：共享内存上下文的动态峰值。 ● max_backend_memory：使用HA端口执行业务可使用的最大内存上限。 ● backend_used_memory：使用HA端口执行业务已使用的内存。 ● max_shared_memory：最大共享内存。 ● shared_used_memory：已使用的共享内存。 ● max_sctpcomm_memory：通信库所允许使用的最大内存。 ● sctpcomm_used_memory：通信库已使用的内存大小。 ● sctpcomm_peak_memory：通信库的内存峰值。 ● other_used_memory：其他已使用的内存大小。
memorymbytes	integer	内存类型分配内存的大小。

12.3.139 GS_TOTAL_NODEGROUP_MEMORY_DETAIL

GS_TOTAL_NODEGROUP_MEMORY_DETAIL返回当前数据库逻辑实例使用内存的信息，单位为MB，若GUC参数enable_memory_limit设置为off，则该视图不能使用。

表 12-243 GS_TOTAL_NODEGROUP_MEMORY_DETAIL 字段

名称	类型	描述
ngname	text	逻辑实例名称。

名称	类型	描述
memorytype	text	内存类型，包括以下几种： • ng_total_memory：该逻辑实例的总内存大小。 • ng_used_memory：该逻辑实例的实际使用内存大小。 • ng_estimate_memory：该逻辑实例的估算使用内存大小。 • ng_foreignrp_memsize：该逻辑实例的外部资源池的总内存大小。 • ng_foreignrp_usedsize：该逻辑实例的外部资源池实际使用内存大小。 • ng_foreignrp_peaksize：该逻辑实例的外部资源池使用内存的峰值。 • ng_foreignrp_mempct：该逻辑实例的外部资源池占该逻辑实例总内存大小的百分比。 • ng_foreignrp_estmsize：该逻辑实例的外部资源池估算使用内存大小。
memorymbytes	integer	内存类型分配内存的大小。

12.3.140 GS_WLM_WORKLOAD_RECORDS

集中式不支持该视图。

12.3.141 GS_WORKLOAD_RULE_STAT

GS_WORKLOAD_RULE_STAT系统视图记录SQL限流规则相关的信息。只有具有sysadmin权限的用户才可以访问此系统视图。

表 12-244 GS_WORKLOAD_RULE_STAT 字段

名称	类型	描述
rule_id	bigint	限流规则ID。
rule_name	name	限流规则的名称，用于检索限流规则。
databases	name[]	限流规则作用的数据库列表，为NULL表示所有库生效。
rule_type	text	限流规则类型，当前仅支持：“sqlid”、“select”、“insert”、“update”、“delete”、“merge”、“resource”，其他取值为非法值。
start_time	timestamp with time zone	限流规则开始的时间，为NULL表示从现在开始生效。

名称	类型	描述
end_time	timestamp with time zone	限流规则结束的时间，为NULL表示一直生效。
max_workload	bigint	限制规则设置的最大并发数。
option_val	text[]	限流规则的参数值，包括：sqlid，关键字列表，资源限制情况。 详细请参见**gs_add_workload_rule**接口说明。
is_valid	boolean	限流规则是否生效，超时的限流规则会设为false。
validate_count	bigint	限流规则拦截SQL的次数。
node_names	text[]	预留字段，限流规则生效的节点名称列表，当前不生效。
user_names	text[]	预留字段，限流规则生效的用户名称列表，当前不生效。

12.3.142 GV_INSTANCE

GV_INSTANCE视图显示当前数据库实例的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-245 GV_INSTANCE 字段

名称	类型	描述
inst_id	oid	当前数据库oid。
instance_number	oid	当前数据库oid。
instance_name	character varying(16)	当前数据库名。
host_name	character varying(64)	暂不支持，值为NULL。
version	character varying(17)	暂不支持，值为NULL。
version_legacy	character varying(17)	暂不支持，值为NULL。
version_full	character varying(17)	暂不支持，值为NULL。
startup_time	timestamp(0) without time zone	暂不支持，值为NULL。
status	character varying(12)	暂不支持，值为NULL。
parallel	character varying(3)	暂不支持，值为NULL。

名称	类型	描述
thread#	numeric	暂不支持，值为NULL。
archiver	character varying(7)	暂不支持，值为NULL。
log_switch_wait	character varying(15)	暂不支持，值为NULL。
logins	character varying(10)	暂不支持，值为NULL。
shutdown_pending	character varying(3)	暂不支持，值为NULL。
database_status	character varying(17)	暂不支持，值为NULL。
instance_role	character varying(18)	暂不支持，值为NULL。
active_state	character varying(9)	暂不支持，值为NULL。
blocked	character varying(3)	暂不支持，值为NULL。
con_id	numeric	暂不支持，值为NULL。
instance_mode	character varying(11)	暂不支持，值为NULL。
edition	character varying(7)	暂不支持，值为NULL。
family	character varying(80)	暂不支持，值为NULL。
database_type	character varying(15)	暂不支持，值为NULL。

12.3.143 GV_SESSION

GV_SESSION视图存储当前会话的所有会话信息。该视图只有管理员可以访问，普通用户需要授权才能访问，该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-246 GV_SESSION 字段

名称	类型	描述
inst_id	numeric	暂不支持，值为NULL。
saddr	raw	暂不支持，值为NULL。
sid	bigint	会话ID。
serial#	integer	当前活动的后台线程的序号。
audsid	numeric	暂不支持，值为NULL。
paddr	raw	暂不支持，值为NULL。
schema#	numeric	暂不支持，值为NULL。
schemaname	name	登录该后台的用户名。

名称	类型	描述
user#	oid	登录此后台线程的用户的OID。oid 为0表示此后台线程为全局辅助线程(auxiliary)。
username	name	登录此后台线程的用户名。username为空表示此后台线程为全局辅助线程(auxiliary)。
command	numeric	暂不支持，值为NULL。
ownerid	numeric	暂不支持，值为NULL。
taddr	character varying(16)	暂不支持，值为NULL。
lockwait	character varying(16)	暂不支持，值为NULL。
machine	text	客户端的主机名，这个字段是通过client_addr的反向DNS查找得到。这个字段只有在启动log_hostname且使用IP连接时才非空。
sql_id	bigint	sql的oid。
client_info	text	客户端信息。
event	text	语句当前排队状态。可能值是： ● waiting in queue：表示语句在排队中。 ● 空：表示语句正在运行。
sql_exec_start	timestamp with time zone	sql执行开始时间。
program	text	连接到该后台的应用名。
status	text	该后台当前总体状态。可能值是： ● active：后台正在执行一个查询。 ● idle：后台正在等待一个新的客户端命令。 ● idle in transaction：后台在事务中，但事务中没有语句在执行。 ● idle in transaction (aborted)：后台在事务中，但事务中有语句执行失败。 ● fastpath function call：后台正在执行一个fast-path函数。 ● disabled：如果后台禁用track_activities，则报告这个状态。
server	character varying(9)	暂不支持，值为NULL。
pdml_status	character varying(8)	当前会话是否启用DML的并行执行。

名称	类型	描述
port	numeric	当前会话的端口号。
process	character varying(24)	当前会话的进程号。
logon_time	timestamp(0) without time zone	当前会话的登录时间。
last_call_et	integer	当前会话上次状态发生改变的时长。
osuser	character varying(128)	暂不支持，值为NULL。
terminal	character varying(30)	暂不支持，值为NULL。
type	character varying(10)	暂不支持，值为NULL。
sql_address	raw	暂不支持，值为NULL。
sql_hash_value	numeric	暂不支持，值为NULL。
sql_child_number	numeric	暂不支持，值为NULL。
sql_exec_id	numeric	暂不支持，值为NULL。
prev_sql_addr	raw	暂不支持，值为NULL。
prev_hash_value	numeric	暂不支持，值为NULL。
prev_sql_id	character varying(13)	暂不支持，值为NULL。
prev_child_number	numeric	暂不支持，值为NULL。
prev_exec_start	timestamp(0) without time zone	暂不支持，值为NULL。
prev_exec_id	numeric	暂不支持，值为NULL。
plsql_entry_object_id	numeric	暂不支持，值为NULL。
plsql_entry_subprogram_id	numeric	暂不支持，值为NULL。

名称	类型	描述
plsql_object_id	numeric	暂不支持，值为NULL。
plsql_subprogram_id	numeric	暂不支持，值为NULL。
module	character varying(64)	暂不支持，值为NULL。
module_hash	numeric	暂不支持，值为NULL。
action	character varying(64)	暂不支持，值为NULL。
action_hash	numeric	暂不支持，值为NULL。
fixed_table_sequence	numeric	暂不支持，值为NULL。
row_wait_obj#	numeric	暂不支持，值为NULL。
row_wait_file#	numeric	暂不支持，值为NULL。
row_wait_block#	numeric	暂不支持，值为NULL。
row_wait_row#	numeric	暂不支持，值为NULL。
top_level_call#	numeric	暂不支持，值为NULL。
pdml_enabled	character varying(3)	暂不支持，值为NULL。
failover_type	character varying(13)	暂不支持，值为NULL。
failover_method	character varying(10)	暂不支持，值为NULL。
failed_over	character varying(3)	暂不支持，值为NULL。
resource_consumer_group	character varying(32)	暂不支持，值为NULL。
pddl_status	character varying(8)	暂不支持，值为NULL。
pq_status	character varying(8)	暂不支持，值为NULL。

名称	类型	描述
current_queue_duration	numeric	暂不支持，值为NULL。
client_identifier	character varying(64)	暂不支持，值为NULL。
blocking_session_status	character varying(11)	暂不支持，值为NULL。
blocking_instance	numeric	暂不支持，值为NULL。
blocking_session	numeric	暂不支持，值为NULL。
final_blocking_session_status	character varying(11)	暂不支持，值为NULL。
final_blocking_instance	numeric	暂不支持，值为NULL。
final_blocking_session	numeric	暂不支持，值为NULL。
seq#	numeric	暂不支持，值为NULL。
event#	numeric	暂不支持，值为NULL。
p1text	character varying(64)	暂不支持，值为NULL。
p1	numeric	暂不支持，值为NULL。
p1raw	raw	暂不支持，值为NULL。
p2text	character varying(64)	暂不支持，值为NULL。
p2	numeric	暂不支持，值为NULL。
p2raw	raw	暂不支持，值为NULL。
p3text	character varying(64)	暂不支持，值为NULL。
p3	numeric	暂不支持，值为NULL。
p3raw	raw	暂不支持，值为NULL。
wait_class_id	numeric	暂不支持，值为NULL。
wait_class#	numeric	暂不支持，值为NULL。
wait_class	character varying(64)	暂不支持，值为NULL。

名称	类型	描述
wait_time	numeric	暂不支持，值为NULL。
seconds_in_wait	numeric	暂不支持，值为NULL。
state	character varying(19)	暂不支持，值为NULL。
wait_time_micro	numeric	暂不支持，值为NULL。
time_remaining_micro	numeric	暂不支持，值为NULL。
time_since_last_wait_micro	numeric	暂不支持，值为NULL。
service_name	character varying(64)	暂不支持，值为NULL。
sql_trace	character varying(8)	暂不支持，值为NULL。
sql_trace_waits	character varying(5)	暂不支持，值为NULL。
sql_trace_binds	character varying(5)	暂不支持，值为NULL。
sql_trace_plan_stats	character varying(10)	暂不支持，值为NULL。
session_edition_id	numeric	暂不支持，值为NULL。
creator_addr	raw	暂不支持，值为NULL。
creator_serial#	numeric	暂不支持，值为NULL。
ecid	character varying(64)	暂不支持，值为NULL。
sql_translation_profile_id	numeric	暂不支持，值为NULL。
pga_tunable_mem	numeric	暂不支持，值为NULL。
shard_ddl_status	character varying(8)	暂不支持，值为NULL。
con_id	numeric	暂不支持，值为NULL。

名称	类型	描述
external_name	character varying(1024)	暂不支持，值为NULL。
plsql_debugger_connected	character varying(5)	暂不支持，值为NULL。

12.3.144 MPP_TABLES

MPP_TABLES视图显示信息如下。

表 12-247 MPP_TABLES 字段

名称	类型	描述
schemaname	name	表的模式名。
tablename	name	表名。
tableowner	name	表的所有者。
tablespace	name	表所在的表空间。
pgroup	name	节点群的名称。
nodeoids	oidvector_extend	表分布的节点OID列表。

12.3.145 MY_COL_COMMENTS

MY_COL_COMMENTS视图显示当前用户下表的列注释信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-248 MY_COL_COMMENTS 字段

名称	类型	描述
owner	character varying(128)	表的所有者。
table_name	character varying(128)	表的名称。
column_name	character varying(128)	列名称。
comments	text	注释。
origin_con_id	numeric	暂不支持，值为0。

名称	类型	描述
schema	character varying(64)	列所属的名称空间的名称。

12.3.146 MY_COL_PRIVS

MY_COL_PRIVS视图显示当前用户作为对象所有者、授权者或被授予者时的列权限授予信息。默认所有用户都可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-249 MY_COL_PRIVS 字段

名称	类型	描述
grantor	character varying(128)	执行授权的用户名。
owner	character varying(128)	对象的所有者。
grantee	character varying(128)	被授予权限的用户或角色的名称。
table_schema	character varying(128)	对象的Schema。
table_name	character varying(128)	对象的名称。
column_name	character varying(128)	列的名称。
privilege	character varying(40)	列的权限。
grantable	character varying(3)	是否授予特权。 ● YES：授予特权。 ● NO：不授予特权。
common	character varying(3)	暂不支持，值为NULL。
inherited	character varying(3)	暂不支持，值为NULL。

12.3.147 MY_COLL_TYPES

MY_COLL_TYPES视图显示当前用户创建的集合类型信息。默认所有用户都可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-250 MY_COLL_TYPES 字段

名称	类型	描述
owner	character varying(128)	集合的所有者。
type_name	character varying(128)	集合的名称。
coll_type	character varying(128)	集合的描述。

名称	类型	描述
upper_bound	numeric	暂不支持，值为NULL。
elem_type_mod	character varying(7)	元素的类型修饰。
elem_type_owner	character varying(128)	集合基于的元素类型的所有者。该值主要用于用户定义的类型。
elem_type_name	character varying(128)	集合所依据的数据类型或用户定义类型的名称。
length	numeric	暂不支持，值为NULL。
precision	numeric	暂不支持，值为NULL。
scale	numeric	暂不支持，值为NULL。
character_set_name	character varying(44)	暂不支持，值为NULL。
elem_storage	character varying(7)	暂不支持，值为NULL。
nulls_stored	character varying(3)	暂不支持，值为NULL。
char_used	character varying(1)	暂不支持，值为NULL。

12.3.148 MY_CONS_COLUMNS

MY_CONS_COLUMNS视图显示当前用户下表中主键约束列的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-251 MY_CONS_COLUMNS 字段

名称	类型	描述
owner	character varying(64)	约束创建者。
table_name	character varying(64)	约束相关的表名。
column_name	character varying(64)	约束相关的列名。
constraint_name	character varying(64)	约束名。
position	smallint	表中列的位置。

12.3.149 MY_CONSTRAINTS

MY_CONSTRAINTS视图显示当前用户下表中约束的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-252 MY_CONSTRAINTS 字段

名称	类型	描述
owner	character varyiny(64)	约束创建者。
constraint_name	vcharacter varying(64)	约束名。
constraint_type	text	约束类型： ● c表示检查约束。 ● f表示外键约束。 ● p表示主键约束。 ● u表示唯一约束。
table_name	character varying(64)	约束相关的表名。
index_owner	character varying(64)	约束相关的索引的所有者（只针对唯一约束和主键约束）。
index_name	character varying(64)	约束相关的索引的名称（只针对唯一约束和主键约束）。

12.3.150 MY_DEPENDENCIES

MY_DEPENDENCIES显示用户拥有对象对其他对象的依赖关系。所有用户都可以访问，该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-253 MY_DEPENDENCIES 字段

名称	类型	描述
name	character varying(128)	对象的名称。
type	character varying(18)	对象类型。
referenced_owner	character varying(128)	被引用对象的所有者。
referenced_name	character varying(128)	被引用对象的名称。

名称	类型	描述
referenced_type	character varying(18)	被引用对象的类型。
referenced_link_name	character varying(128)	指向父对象的链接的名称（如果是远程）。
dependency_type	character varying(4)	指示依赖关系是否为REF依赖关系。

12.3.151 MY_ERRORS

MY_ERRORS视图显示用户拥有的存储对象的最新编译错误信息。该视图所有用户可访问，仅可查看当前用户所属信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-254 MY_ERRORS 字段

名称	类型	描述
name	character varying(128)	对象的名称。
type	character varying(12)	对象类型： • PROCEDURE • FUNCTION • PACKAGE • PACKAGE BODY
sequence	numeric	序列号。
line	numeric	发生错误的行号。
position	numeric	发生错误的行中的位置。
text	character varying(4000)	错误文本。
attribute	character varying(9)	属性标记：错误（ERROR）。
message_number	numeric	暂不支持，值为NULL。

12.3.152 MY_IND_COLUMNS

MY_IND_COLUMNS视图显示当前用户下所有索引的字段信息。所有用户都可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-255 MY_IND_COLUMNS 字段

名称	类型	描述
index_owner	character varying(64)	索引的所有者。
index_name	character varying(64)	索引名。
table_owner	character varying(64)	表的所有者。
table_name	character varying(64)	表名。
column_name	name	列名。
column_position	smallint	索引中列的位置。
column_length	numeric	列的长度，如果列是变长类型，该字段取值为NULL。
char_length	numeric	列的最大字节长度。
descend	character varying	表示列是按降序（DESC）还是升序（ASC）排序。
collated_column_id	numeric	暂不支持，值为NULL。

12.3.153 MY_IND_EXPRESSIONS

MY_IND_EXPRESSIONS视图显示当前用户下基于函数的表达式索引的信息。所有用户都可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-256 MY_IND_EXPRESSIONS 字段

名称	类型	描述
table_owner	character varying(64)	表的所有者。
table_name	character varying(64)	表名。
index_owner	character varying(64)	索引的所有者。
index_name	character varying(64)	索引名。
column_expression	text	定义列的基于函数的索引表达式。
column_position	smallint	索引中列的位置。

12.3.154 MY_IND_PARTITIONS

MY_IND_PARTITIONS视图存储当前用户下一级分区表Local索引的索引分区信息（不包含分区表全局索引）。所有用户都可以访问。该视图同时存在于PG_CATALOG和SYS schema下。

表 12-257 MY_IND_PARTITIONS 字段

名称	类型	描述
index_owner	character varying(64)	索引分区所属分区表索引的所有者的名称。
index_name	character varying(64)	索引分区所属分区表索引的名称。
partition_name	character varying(64)	索引分区的名称。
def_tablespace_name	name	索引分区的表空间名称。
high_value	text	索引分区所对应分区的上边界。
index_partition_usable	boolean	索引分区是否可用： ● t（true）：可用。 ● f（false）：不可用。
schema	character varying(64)	索引分区所属分区表索引的模式。
high_value_length	integer	索引分区所对应分区的边界的字符长度。
composite	character varying(3)	索引是否属于二级分区表上的本地索引，该表不存储二级分区信息，所以该值为NO。
subpartition_count	numeric	分区中的二级分区数，该表不存储二级分区信息，所以该值为0。
partition_position	numeric	索引分区在索引中的位置。
status	character varying(8)	索引分区是否可用。
tablespace_name	name	分区所在表空间的名称。
pct_free	numeric	块中最小可用空间百分比。
ini_trans	numeric	初始事务数，默认值为4，非USTORE分区表时为NULL。
max_trans	numeric	最大事务数，默认值为128，非USTORE分区表时为NULL。
initial_extent	numeric	暂不支持，值为NULL。
next_extent	numeric	暂不支持，值为NULL。

名称	类型	描述
min_extent	numeric	暂不支持，值为NULL。
max_extent	numeric	暂不支持，值为NULL。
max_size	numeric	暂不支持，值为NULL。
pct_increase	numeric	暂不支持，值为NULL。
freelists	numeric	暂不支持，值为NULL。
freelist_groups	numeric	暂不支持，值为NULL。
logging	character varying(7)	是否记录对索引的更改。
compression	character varying(13)	分区索引是否启用索引压缩。
blevel	numeric	暂不支持，值为NULL。
leaf_blocks	numeric	暂不支持，值为NULL。
distinct_keys	numeric	暂不支持，值为NULL。
avg_leaf_blocks_per_key	numeric	暂不支持，值为NULL。
avg_data_blocks_per_key	numeric	暂不支持，值为NULL。
clustering_factor	numeric	根据索引的值表示表中行的顺序。需要通过执行analyze进行统计。
num_rows	numeric	分区中的行数。需要通过执行vacuum进行统计。
sample_size	numeric	暂不支持，值为NULL。
last_analyzed	timestamp with time zone	最近分析此分区的日期。数据库重启后，数据会丢失。
buffer_pool	character varying(7)	分区的实际缓冲池。
flash_cache	character varying(7)	暂不支持，值为NULL。
cell_flash_cache	character varying(7)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。
pct_direct_access	numeric	暂不支持，值为NULL。
global_stats	character varying(3)	暂不支持，值为NULL。
domidx_opstatus	character varying(6)	暂不支持，值为NULL。
parameters	character varying(1000)	暂不支持，值为NULL。
interval	character varying(3)	分区是否在间隔分区表的间隔节中。

名称	类型	描述
segment_created	character varying(3)	索引分区段是否已创建。
orphaned_entries	character varying(3)	暂不支持，值为NULL。

12.3.155 MY_IND_SUBPARTITIONS

MY_IND_SUBPARTITIONS描述了当前用户拥有的二级分区表Local索引的索引分区信息（不包含分区表全局索引）。所有用户都可以访问。该视图同时存在于PG_CATALOG和SYS schema下。

表 12-258 MY_IND_SUBPARTITIONS 字段

名称	类型	描述
index_owner	character varying(64)	索引分区所属分区表索引的所有者的名称。
index_name	character varying(64)	索引分区所属分区表索引的名称。
partition_name	character varying(64)	索引所在一级分区的名称。
subpartition_name	character varying(64)	索引所在二级分区的名称。
def_tablespace_name	name	索引分区的表空间名称。
high_value	text	索引分区所对应分区的边界值。
index_partition_usable	boolean	索引分区是否可用： ● t（true）：可用。 ● f（false）：不可用。
schema	character varying(64)	索引分区所属分区表索引的模式。
high_value_length	integer	索引分区所对应分区的边界的字符长度。
partition_position	numeric	索引分区在索引中的位置。
subpartition_position	numeric	二级分区在分区中的位置。
status	character varying(8)	索引分区是否可用。
tablespace_name	name	索引分区的表空间名称。

名称	类型	描述
pct_free	numeric	块中最小可用空间百分比。
ini_trans	numeric	初始事务数，默认值为4，非USTORE分区表时为NULL。
max_trans	numeric	最大事务数，默认值为128，非USTORE分区表时为NULL。
initial_extent	numeric	暂不支持，值为NULL。
next_extent	numeric	暂不支持，值为NULL。
min_extent	numeric	暂不支持，值为NULL。
max_extent	numeric	暂不支持，值为NULL。
max_size	numeric	暂不支持，值为NULL。
pct_increase	numeric	暂不支持，值为NULL。
freelists	numeric	暂不支持，值为NULL。
freelist_groups	numeric	暂不支持，值为NULL。
logging	character varying(7)	是否记录对索引的更改。
compression	character varying(13)	用于二级分区的压缩类型。
blevel	numeric	暂不支持，值为NULL。
leaf_blocks	numeric	暂不支持，值为NULL。
distinct_keys	numeric	暂不支持，值为NULL。
avg_leaf_blocks_per_key	numeric	暂不支持，值为NULL。
avg_data_blocks_per_key	numeric	暂不支持，值为NULL。
clustering_factor	numeric	根据索引的值表示表中行的顺序。需要通过执行analyze进行统计。
num_rows	numeric	二级分区中的行数。需要通过执行vacuum进行统计。
sample_size	numeric	暂不支持，值为NULL。
last_analyzed	timestamp with time zone	最近分析此分区的日期。数据库重启后，数据会丢失。
buffer_pool	character varying(7)	二级分区的缓冲池。

名称	类型	描述
flash_cache	character varying(7)	暂不支持，值为NULL。
cell_flash_cache	character varying(7)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。
global_stats	character varying(3)	暂不支持，值为NULL。
interval	character varying(3)	分区是否在间隔分区表的间隔节中。
segment_created	character varying(3)	索引分区段是否已创建。
domidx_opstatus	character varying(6)	暂不支持，值为NULL。
parameters	character varying(1000)	暂不支持，值为NULL。

12.3.156 MY_INDEXES

MY_INDEXES视图显示当前用户的索引信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-259 MY_INDEXES 字段

名称	类型	描述
owner	character varying(64)	索引的所有者。
index_name	character varying(64)	索引名称。
table_name	character varying(64)	索引对应的表名。
uniqueness	text	表示该索引是否为唯一索引。 ● UNIQUE：唯一索引。 ● NONUNIQUE：非唯一索引。
partitioned	character(3)	表示该索引是否具有分区表的性质。 ● Yes：索引具有分区表的性质。 ● No：索引不具有分区表的性质。

名称	类型	描述
generated	character varying(1)	表示该索引的名称是否为系统生成。 ● y：索引名称为系统生成。 ● n：索引名称非系统生成。
index_type	character varying(27)	索引类型。 ● NORMAL：索引属性都是简单的引用，表达式树为空。 ● FUNCTION-BASED NORMAL：存在表达式树用于非简单字段引用的索引属性。
table_owner	character varying(128)	索引对象的所有者。
table_type	character(11)	索引对象的类型。 ● TABLE：索引对象为表类型。
tablespace_name	character varying(30)	包含索引的表空间名称。
status	character varying(8)	非分区索引状态。 ● VALID：非分区索引可以用于查询。 ● UNUSABLE：非分区索引不可用。 ● N/A：索引具有分区表性质。
compression	character varying(13)	暂不支持，值为NULL。
prefix_length	numeric	暂不支持，值为NULL。
ini_trans	numeric	暂不支持，值为NULL。
max_trans	numeric	暂不支持，值为NULL。
initial_extent	numeric	暂不支持，值为NULL。
next_extent	numeric	暂不支持，值为NULL。
min_extents	numeric	暂不支持，值为NULL。
max_extents	numeric	暂不支持，值为NULL。
pct_increase	numeric	暂不支持，值为NULL。
pct_threshold	numeric	暂不支持，值为NULL。

名称	类型	描述
include_column	numeric	暂不支持，值为NULL。
freelists	numeric	暂不支持，值为NULL。
freelist_groups	numeric	暂不支持，值为NULL。
pct_free	numeric	暂不支持，值为NULL。
logging	character varying(3)	暂不支持，值为NULL。
blevel	numeric	暂不支持，值为NULL。
leaf_blocks	numeric	暂不支持，值为NULL。
distinct_keys	numeric	暂不支持，值为NULL。
avg_leaf_blocks_per_key	numeric	暂不支持，值为NULL。
avg_data_blocks_per_key	numeric	暂不支持，值为NULL。
clustering_factor	numeric	暂不支持，值为NULL。
num_rows	numeric	暂不支持，值为NULL。
sample_size	numeric	暂不支持，值为NULL。
last_analyzed	timestamp(0) without time zone	暂不支持，值为NULL。
degree	character varying(40)	暂不支持，值为NULL。
instances	character varying(40)	暂不支持，值为NULL。
temporary	character varying(1)	暂不支持，值为NULL。
secondary	character varying(1)	暂不支持，值为NULL。
buffer_pool	character varying(7)	暂不支持，值为NULL。
flash_cache	character varying(7)	暂不支持，值为NULL。
cell_flash_cache	character varying(7)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。
duration	character varying(15)	暂不支持，值为NULL。
pct_direct_access	numeric	暂不支持，值为NULL。
ityp_owner	character varying(128)	暂不支持，值为NULL。
ityp_name	character varying(128)	暂不支持，值为NULL。
parameters	character varying(1000)	暂不支持，值为NULL。
global_stats	character varying(3)	暂不支持，值为NULL。

名称	类型	描述
domidx_status	character varying(12)	暂不支持，值为NULL。
domidx_opstatus	character varying(6)	暂不支持，值为NULL。
funcidx_status	character varying(8)	暂不支持，值为NULL。
join_index	character varying(3)	暂不支持，值为NULL。
iot_redundant_pkey_elim	character varying(3)	暂不支持，值为NULL。
dropped	character varying(3)	暂不支持，值为NULL。
visibility	character varying(9)	暂不支持，值为NULL。
domidx_management	character varying(14)	暂不支持，值为NULL。
segment_created	character varying(3)	暂不支持，值为NULL。
orphaned_entries	character varying(3)	暂不支持，值为NULL。
indexing	character varying(7)	暂不支持，值为NULL。
auto	character varying(3)	暂不支持，值为NULL。

12.3.157 MY_JOBS

MY_JOBS视图显示当前用户拥有的定时任务的详细信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-260 MY_JOBS 字段

名称	类型	描述
job	bigint	作业ID。
log_user	name	创建者的UserName。
priv_user	name	作业执行者的UserName。
dbname	name	创建作业的数据库名称。
schema_user	name	定时任务的默认模式名。
start_date	timestamp without time zone	作业第一次开始执行的时间。
start_suc	text	作业第一次成功执行的时间。
last_date	timestamp without time zone	上次运行的开始时间。
last_suc	text	上次成功运行的开始时间。

名称	类型	描述
last_sec	text	上次成功运行的开始时间，提供兼容性支持。
this_date	timestamp without time zone	正在运行任务的此次开始时间。
this_suc	text	正在运行任务的此次开始时间。
this_sec	text	正在运行任务的此次开始时间，提供兼容性支持。
next_date	timestamp without time zone	任务下次执行时间。
next_suc	text	任务下次执行时间。
next_sec	text	任务下次执行时间，提供兼容性支持。
total_time	numeric	任务最近一次的执行时长。
broken	text	如果任务状态为d，则为'y'，否则为'n'。
status	"char"	本步骤的执行状态，取值范围：（'r'、's'、'f'、'd'），默认为'r'，取值含义： ● r：运行中。 ● s：执行成功。 ● f：执行失败。 ● d：取消执行。
interval	text	用来计算下次运行时间的时间表达式，如果为null则表示定时任务只执行一次。
failures	smallint	失败计数，若作业连续执行失败16次，则不再继续执行。
what	text	可执行的作业。
nls_env	character varying(4000)	暂不支持，值为NULL。
misc_env	raw	暂不支持，值为NULL。
instance	numeric	暂不支持，值为NULL。

12.3.158 MY_OBJECTS

MY_OBJECTS视图描述了当前用户所属的数据库对象信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-261 MY_OBJECTS 字段

名称	类型	描述
object_name	name	对象的名称。
object_id	oid	对象的OID。
object_type	name	对象的类型，包括TABLE、INDEX、SEQUENCE、VIEW。
namespace	oid	对象所属的名称空间。
temporary	character(1)	对象是否为临时对象。
status	character varying(7)	对象的状态。
subobject_name	name	对象的子对象名称。
generated	character(1)	对象名称是否是系统生成。
created	timestamp with time zone	对象的创建时间。
last_ddl_time	timestamp with time zone	对象的最后修改时间。
default_collation	character varying(100)	对象的默认排序规则。
data_object_id	numeric	暂不支持，值为NULL。
timestamp	character varying(19)	暂不支持，值为NULL。
secondary	character varying(1)	暂不支持，值为NULL。
edition_name	character varying(128)	暂不支持，值为NULL。
sharing	character varying(18)	暂不支持，值为NULL。
editionable	character varying(1)	暂不支持，值为NULL。
oracle_maintained	character varying(1)	暂不支持，值为NULL。

名称	类型	描述
application	character varying(1)	暂不支持，值为NULL。
duplicated	character varying(1)	暂不支持，值为NULL。
sharded	character varying(1)	暂不支持，值为NULL。
created_appid	numeric	暂不支持，值为NULL。
modified_appid	numeric	暂不支持，值为NULL。
created_vsnid	numeric	暂不支持，值为NULL。
modified_vsnid	numeric	暂不支持，值为NULL。

须知

created和last_ddl_time支持的范围参见PG_OBJECT中的记录范围。

12.3.159 MY_PART_COL_STATISTICS

MY_PART_COL_STATISTICS视图显示当前用户拥有的表分区的列统计信息和直方图信息。所有用户可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-262 MY_PART_COL_STATISTICS 字段

名称	类型	描述
table_name	character varying(128)	表名。
partition_name	character varying(128)	表分区名称。
column_name	character varying(4000)	列名。
num_distinct	numeric	暂不支持，值为NULL。
low_value	raw	暂不支持，值为NULL。
high_value	raw	暂不支持，值为NULL。
density	numeric	暂不支持，值为NULL。
num_nulls	numeric	暂不支持，值为NULL。
num_buckets	numeric	暂不支持，值为NULL。
sample_size	numeric	暂不支持，值为NULL。
last_analyzed	date	暂不支持，值为NULL。
global_stats	character varying(3)	暂不支持，值为NULL。

名称	类型	描述
user_stats	character varying(3)	暂不支持，值为NULL。
notes	character varying(63)	暂不支持，值为NULL。
avg_col_len	numeric	暂不支持，值为NULL。
histogram	character varying(15)	暂不支持，值为NULL。
schema	character varying(64)	列所属的名称空间的名称。

12.3.160 MY_PART_INDEXES

MY_PART_INDEXES视图存储当前用户下分区表索引的信息。该视图同时存在于PG_CATALOG和SYS schema下。

表 12-263 MY_PART_INDEXES 字段

名称	类型	描述
def_tablespace_name	name	分区表索引的表空间名称。
index_owner	character varying(64)	分区表索引的所有者名称。
index_name	character varying(64)	分区表索引的名称。
partition_count	bigint	分区表索引的索引分区的个数。
partitioning_key_count	integer	分区表的分区键个数。
partitioning_type	text	分区表的分区策略。 **说明** 当前分区表策略支持范围见**CREATE TABLE PARTITION**。
schema	character varying(64)	分区表索引的模式。
table_name	character varying(64)	分区表索引所属的分区表名称。
subpartitioning_type	text	二级分区表的分区策略。如果分区表是一级分区表，则显示NONE。 **说明** 当前二级分区表策略支持范围见**CREATE TABLE SUBPARTITION**。
def_subpartition_count	integer	默认创建二级分区的个数，二级分区表为1，一级分区表为0。

名称	类型	描述
subpartitioning_key_count	integer	分区表二级分区键的个数。

12.3.161 MY_PART_KEY_COLUMNS

MY_PART_KEY_COLUMNS视图显示了当前用户拥有的分区表或分区索引的分区键列的相关信息。该视图所有用户可访问，仅可查看当前用户所属信息。该视图同时存在于PG_CATALOG和SYS schema下。

表 12-264 MY_PART_KEY_COLUMNS 字段

名称	类型	描述
name	character varying(128)	分区表名或索引名。
object_type	character varying(128)	对象类型。 • 若分区为分区表，此列为table。 • 若分区为分区索引，此列为index。
column_name	character varying(4000)	分区表或索引的键列名。
column_position	numeric	列在分区中的位置。
collated_column_id	numeric	暂不支持，值为NULL。

12.3.162 MY_PART_TABLES

MY_PART_TABLES视图存储当前用户下分区表的信息。该视图同时存在于PG_CATALOG和SYS schema下。

表 12-265 MY_PART_TABLES 字段

名称	类型	描述
table_owner	character varying(64)	分区表的所有者名称。
table_name	character varying(64)	分区表的名称。
partitioning_type	text	分区表的分区策略。 **说明** 当前分区表策略支持范围见 **CREATE TABLE PARTITION**。
partition_count	bigint	分区表的分区个数。

名称	类型	描述
partitioning_key_count	integer	分区表的分区键个数。
def_tablespace_name	name	分区表的表空间名称。
schema	character varying(64)	分区表的模式。
subpartitioning_type	text	二级分区表的分区策略。如果分区表是一级分区表，则显示NONE。 **说明** 当前二级分区表策略支持范围见CREATE TABLE SUBPARTITION。
def_subpartition_count	integer	默认创建二级分区的个数，二级分区表为1，一级分区表为0。
subpartitioning_key_count	integer	分区表二级分区键的个数。
status	character varying(8)	暂不支持，值为valid。
def_pct_free	numeric	添加分区时使用的PCTFREE默认值。
def_pct_used	numeric	暂不支持，值为NULL。
def_ini_trans	numeric	添加分区时使用的INITRANS默认值。
def_max_trans	numeric	添加分区时使用的MAXTRANS默认值。
def_initial_extent	character varying(40)	暂不支持，值为NULL。
def_next_extent	character varying(40)	暂不支持，值为NULL。
def_min_extents	character varying(40)	暂不支持，值为NULL。
def_max_extents	character varying(40)	暂不支持，值为NULL。
def_max_size	character varying(40)	暂不支持，值为NULL。
def_pct_increase	character varying(40)	暂不支持，值为NULL。
def_freelists	numeric	暂不支持，值为NULL。
def_freelist_groups	numeric	暂不支持，值为NULL。
def_logging	character varying(7)	暂不支持，值为NULL。
def_compression	character varying(8)	添加分区时使用的默认压缩： • NONE • ENABLED • DISABLED

名称	类型	描述
def_compress_for	character varying(30)	添加分区时使用的默认压缩。 **说明** 可用的压缩方法和压缩级别见 **WITH ({ storage_paramet...**。
def_buffer_pool	character varying(7)	暂不支持，值为DEFAULT。
def_flash_cache	character varying(7)	暂不支持，值为NULL。
def_cell_flash_cache	character varying(7)	暂不支持，值为NULL。
ref_ptn_constraint_name	character varying(128)	暂不支持，值为NULL。
interval	character varying(1000)	区间值字符串。
autolist	character varying(3)	暂不支持，值为NO。
interval_subpartition	character varying(1000)	暂不支持，值为NULL。
autolist_subpartition	character varying(3)	暂不支持，值为NO。
is_nested	character varying(3)	暂不支持，值为NO。
def_segment_creation	character varying(4)	暂不支持段页式设置，当启用segment时，值为YES。
def_indexing	character varying(3)	暂不支持，值为ON。
def_inmemory	character varying(8)	暂不支持，值为NONE。
def_inmemory_priority	character varying(8)	暂不支持，值为NULL。
def_inmemory_distribute	character varying(15)	暂不支持，值为NULL。
def_inmemory_compression	character varying(17)	暂不支持，值为NULL。
def_inmemory_duplicate	character varying(13)	暂不支持，值为NULL。
def_read_only	character varying(3)	暂不支持，值为NO。
def_cellmemory	character varying(24)	暂不支持，值为NULL。
def_inmemory_service	character varying(12)	暂不支持，值为NULL。
def_inmemory_service_name	character varying(1000)	暂不支持，值为NULL。

12.3.163 MY_PROCEDURES

MY_PROCEDURES视图描述了关于当前用户拥有的存储过程、函数或触发器的信息。该视图同时存在于PG_CATALOG和SYS Schema下。该视图所有用户可访问，仅可查看当前用户所属信息。

表 12-266 MY_PROCEDURES 字段

名称	类型	描述
owner	character varying(64)	存储过程、函数或触发器或包的所有者。
object_name	character varying(64)	存储过程、函数或触发器的名称，若为包内函数或存储过程，则为包名。
procedure_name	character varying(128)	若object_name为包名，则显示包内函数或存储过程。
object_id	oid	存储过程、函数或触发器或包的oid。
subprogram_id	numeric	包内函数或存储过程的位置。
overload	character varying(40)	第n个重载函数。
object_type	character varying(13)	对象的类型。
aggregate	character varying(3)	表示是否为聚合函数： ● YES：表示是。 ● NO：表示不是。
pipelined	character varying(3)	暂不支持，值为NO。
impltypeowner	character varying(128)	实现类型的所有者。
impltypename	character varying(128)	实现类型的名称。
parallel	character varying(3)	暂不支持，值为NO。
interface	character varying(3)	暂不支持，值为NO。
deterministic	character varying(3)	暂不支持，值为NO。
authid	character varying(12)	表示是使用创建者权限还是调用者权限： ● DEFINER：表示使用创建者权限。 ● CURRENT_USER：表示使用调用者权限。 因该字段与保留关键字冲突，调用该字段需加视图名。
result_cache	character varying(3)	暂不支持，值为NULL。

名称	类型	描述
origin_con_id	character varying(256)	暂不支持，值为0。
polymorphic	character varying(5)	暂不支持，值为NULL。
argument_number	smallint	存储过程入参个数。

12.3.164 MY_RECYCLEBIN

MY_RECYCLEBIN显示当前用户拥有的回收站信息。所有用户都可以访问，该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-267 MY_RECYCLEBIN 字段

名称	类型	描述
object_name	character varying(128)	对象的新名称。
original_name	character varying(128)	对象的原始名称。
operation	character varying(9)	对对象执行的操作： ● DROP：对象已删除（对象不再需要）。 ● TRUNCATE：对象被清空。
type	character varying(25)	对象的类型。
ts_name	character varying(30)	对象所属的表空间名称。
createtime	character varying(19)	创建对象的时间戳。
droptime	character varying(19)	删除对象的时间戳。
dropscn	numeric	将对象移动到回收站的事务的系统更改编号(SCN)。
partition_name	character varying(128)	已删除的分区的名称。

名称	类型	描述
can_undrop	character varying(3)	指示对象是否可以闪回。
can_purge	character varying(3)	指示对象是否可以清除。
related	numeric	父对象的对象编号。
base_object	numeric	基对象的对象编号。
purge_object	numeric	被清除的对象的对象编号。
space	numeric	对象使用的块数。

12.3.165 MY_ROLE_PRIVS

MY_ROLE_PRIVS视图显示授予当前用户角色（包括public角色）的权限的信息。默认所有用户都可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-268 MY_ROLE_PRIVS 字段

名称	类型	描述
grantee	character varying(128)	被授予权限的用户或角色名称。
granted_role	character varying(128)	被授予的角色。
admin_option	character varying(3)	该授权是否包含ADMIN选项。 ● YES：包含ADMIN选项。 ● NO：不包含ADMIN选项。
delegate_option	character varying(3)	暂不支持，值为NULL。
default_role	character varying(3)	暂不支持，值为NULL。
os_granted	character varying(3)	暂不支持，值为NULL。
common	character varying(3)	暂不支持，值为NULL。
inherited	character varying(3)	暂不支持，值为NULL。

12.3.166 MY_SCHEDULER_JOB_ARGS

MY_SCHEDULER_JOB_ARG视图显示当前用户拥有的任务的有关参数信息。该视图所有用户可访问，仅可查看当前用户所属信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-269 MY_SCHEDULER_JOB_ARGS 字段

名称	类型	描述
job_name	character varying(128)	参数所属作业名。
argument_name	character varying(128)	参数名称。
argument_position	numeric	参数在参数列表中的位置。
argument_type	character varying(257)	参数的数据类型。
value	character varying(4000)	参数值。
anydata_value	character varying(4000)	暂不支持，值为NULL。
out_argument	character varying(5)	保留字段，值为NULL。

12.3.167 MY_SCHEDULER_PROGRAM_ARGS

MY_SCHEDULER_PROGRAM_ARG视图显示了当前用户拥有的程序的有关参数信息。该视图所有用户可访问，仅可查看当前用户所属信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-270 MY_SCHEDULER_PROGRAM_ARGS 字段

名称	类型	描述
program_name	character varying(128)	参数所属程序名。
argument_name	character varying(128)	参数名称。
argument_position	numeric	参数在参数列表中的位置。
argument_type	character varying(257)	参数的数据类型。
metadata_attribute	character varying(19)	暂不支持，值为NULL。
default_value	character varying(4000)	参数默认值。
default_anydata_value	character varying(4000)	暂不支持，值为NULL。

名称	类型	描述
out_argument	character varying(5)	保留字段，值为NULL。

12.3.168 MY_SEQUENCES

MY_SEQUENCES视图显示当前用户的序列信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-271 MY_SEQUENCES 字段

名称	类型	描述
sequence_owner	name	序列所有者。
sequence_name	name	序列的名称。
min_value	int16	序列最小值。
max_value	int16	序列最大值。
increment_by	int16	序列的增量。
cycle_flag	character(1)	表示序列是否是循环序列，取值为Y或N： ● Y表示是循环序列。 ● N表示不是循环序列。
order_flag	character varying(1)	表示序列是否是按照请求顺序发生，暂不支持，值为NULL。
last_number	int16	上一序列的值。
cache_size	int16	序列磁盘缓存大小。
scale_flag	character varying(1)	表示是否为可扩展序列，暂不支持，值为NULL。
extend_flag	character varying(1)	表示可扩展序列生成的值是否超出序列最大值、最小值范围，暂不支持，值为NULL。
sharded_flag	character varying(1)	表示是否是分片序列，暂不支持，值为NULL。
session_flag	character varying(1)	表示序列是否是会话私有，暂不支持，值为NULL。
keep_value	character varying(1)	表示在失败后的replay期间是否保留序列值，暂不支持，值为NULL。

12.3.169 MY_SOURCE

MY_SOURCE视图显示当前用户拥有的存储过程、函数、触发器、包的定义信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-272 MY_SOURCE 字段

名称	类型	描述
owner	name	对象的所有者。
name	name	对象名字。
type	name	对象类型：function、package、package body、procedure、trigger。
line	numeric	此行来源的行号。
text	text	存储对象的文本来源。
origin_con_id	character varying(256)	暂不支持，值为0。

12.3.170 MY_SUBPART_COL_STATISTICS

MY_SUBPART_COL_STATISTICS视图存储了当前用户拥有的分区对象的子分区的列统计信息和直方图信息。所有用户都可以访问该视图。该视图同时存在于pg_catalog和sys schema下。

名称	类型	描述
table_name	character varying(128)	表名。
subpartition_name	character varying(128)	表子分区名称。
column_name	character varying(4000)	列名。
num_distinct	numeric	暂不支持，值为NULL。
low_value	raw	暂不支持，值为NULL。
high_value	raw	暂不支持，值为NULL。
density	numeric	暂不支持，值为NULL。
num_nulls	numeric	暂不支持，值为NULL。
num_buckets	numeric	暂不支持，值为NULL。
sample_size	numeric	暂不支持，值为NULL。
last_analyzed	date	暂不支持，值为NULL。
global_stats	character varying(3)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。

名称	类型	描述
notes	character varying(41)	暂不支持，值为NULL。
avg_col_len	numeric	暂不支持，值为NULL。
histogram	character varying(15)	暂不支持，值为NULL。
schema	character varying(64)	列所属的名称空间的名称。

12.3.171 MY_SUBPART_KEY_COLUMNS

MY_SUBPART_KEY_COLUMNS视图显示了当前用户所拥有的二级分区表或分区索引的分区键列的相关信息。该视图所有用户可访问，仅可查看当前用户所属信息。该视图同时存在于PG_CATALOG和SYS schema下。

表 12-273 MY_SUBPART_KEY_COLUMNS 字段

名称	类型	描述
name	character varying(128)	二级分区表名或索引名。
object_type	character varying(128)	对象类型。 • 若分区为分区表，此列为table。 • 若分区为分区索引，此列为index。
column_name	character varying(4000)	二级分区表或索引的键列名。
column_position	numeric	列在分区中的位置。
collated_column_id	numeric	暂不支持，值为NULL。

12.3.172 MY_SYNONYMS

MY_SYNONYMS视图显示当前模式下同义词的信息。该视图所有用户可访问，仅可查看当前用户所属信息。该视图同时存在于PG_CATALOG和SYS schema下。

表 12-274 MY_SYNONYMS 字段

名称	类型	描述
schema_name	text	同义词所属模式名。
synonym_name	text	同义词的名称。

名称	类型	描述
table_owner	text	关联对象的所有者。尽管该列称为table_owner，但它拥有的该关联对象不一定是表，可以是任何数据库通用对象，例如视图、存储过程、同义词等。
table_name	text	关联对象名。尽管该列称为table_name，但此关联对象不一定是表，可以是任何数据库通用对象，例如视图、存储过程、同义词等。
table_schema_name	text	关联对象所属模式名。尽管该列称为table_schema_name，但此schema下的该关联对象不一定是表，可以是任何数据库通用对象，例如视图、存储过程、同义词等。
db_link	character varying(128)	保留字段，值为NULL。
origin_con_id	character varying(256)	暂不支持，值为0。

12.3.173 MY_SYS_PRIVS

MY_SYS_PRIVS视图显示授予当前用户的系统权限信息。默认所有用户都可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-275 MY_SYS_PRIVS 字段

名称	类型	描述
grantee	character varying(128)	被授予权限的用户或角色的名称。
privilege	character varying(40)	系统权限。
admin_option	character varying(3)	该授权是否包含ADMIN选项。 ● YES：包含ADMIN选项。 ● NO：不包含ADMIN选项。
common	character varying(3)	暂不支持，值为NULL。
inherited	character varying(3)	暂不支持，值为NULL。

12.3.174 MY_TAB_COL_STATISTICS

MY_TAB_COL_STATISTICS视图显示从MY_TAB_COLUMNS 中提取的列统计信息和直方图信息。所有用户都可以访问该视图。该视图同时存在于PG_CATALOG和SYS Schema下。该视图在LOW_VALUE、HIGH_VALUE字段，由于底层表结构不同原因，与A数据库取值有差异，当LOW_VALUE为高频值时，GaussDB的LOW_VALUE为次小值。当HIGH_VALUE为高频值时，GaussDB的HIGH_VALUE为次高值。HISTOGRAM字段，由于统计方式不同原因，与A数据库取值有差异，GaussDB只支持两种类型直方图frequency，equi-width。SCOPE字段，由于GaussDB不支持全局临时表统计原因，与A数据库取值有差异，GaussDB只支持本地临时表信息统计，默认置SHARED。

表 12-276 MY_TAB_COL_STATISTICS 字段

名称	类型	描述
table_name	character varying(128)	表名。
column_name	character varying(128)	列名。
num_distinct	numeric	列中不同值的数量。
low_value	raw	列中的低值。
high_value	raw	列中的高值。
density	numeric	• 如果COLUMN_NAME上有直方图，则此列将显示直方图中跨越少于2个端点的值的选择性。它不代表跨越2个或更多端点的值的选择性。 • 如果COLUMN_NAME上没有可用的直方图，则该列的值为1/NUM_DISTINCT。
num_nulls	numeric	列中空值数。
num_buckets	numeric	列的直方图中的桶数。
sample_size	numeric	用于分析此列的样本量。数据库重启后，数据会丢失。
last_analyzed	timestamp(0) without time zone	最近分析此列的日期。
global_stats	character varying(3)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。
notes	character varying(99)	暂不支持，值为NULL。
avg_col_len	numeric	列的平均长度（以字节为单位）。

名称	类型	描述
histogram	character varying(15)	表示直方图是否存在，如果存在的话是什么类型： ● NONE：表示不存在直方图。 ● FREQUENCY：表示频率直方图。 ● EQUI-WIDTH：表示等宽直方图。
scope	character varying(7)	对于在除全局临时表之外的任何表上收集的统计信息，该值是SHARED(表示统计信息在所有会话之间共享)。
schema	character varying(64)	列所属的名称空间的名称。

12.3.175 MY_TAB_COLUMNS

MY_TAB_COLUMNS视图显示当前用户拥有的表和视图的字段信息。该视图同时存在于PG_CATALOG和SYS Schema下。该视图所有用户可访问，仅显示该用户所属的信息。

表 12-277 MY_TAB_COLUMNS 字段

名称	类型	描述
owner	character varying(64)	表的所有者。
table_name	character varying(64)	表名称。
column_name	character varying(64)	列名。
data_type	character varying(128)	列的数据类型。
data_type_mod	character varying(3)	暂不支持，值为NULL。
data_type_owner	character varying(128)	列的数据类型的所有者。
data_length	integer	列的字节长度。
data_precision	integer	数据类型的精度，对于numeric数据类型有效，其他类型为NULL。
data_scale	integer	小数点右边的位数，对于numeric数据类型有效，其他类型为0。
nullable	bpchar	该列是否允许为空，对于主键约束和非空约束，该值为n。
column_id	integer	创建表时列的序号。
default_length	numeric	列的默认值字节长度。
data_default	text	列的默认值。

名称	类型	描述
num_distinct	numeric	列中不同值的数量。
low_value	raw	列中的最小值。
high_value	raw	列中的最大值。
density	numeric	列密度。
num_nulls	numeric	列中空值数。
num_buckets	numeric	列的直方图的桶数。
last_analyzed	timestamp(0) without time zone	上次分析的日期。
sample_size	numeric	用于分析此列的样本量。
character_set_name	character varying(44)	暂不支持，值为NULL。
char_col_decl_length	numeric	字符类型列的声明长度。
global_stats	character varying(3)	暂不支持，值为NO。
user_stats	character varying(3)	暂不支持，值为NO。
avg_col_len	numeric	列的平均长度（单位字节）。
char_length	numeric	列的长度（单位字符），只对varchar，nvarchar2，bpchar，char类型有效。
char_used	character varying(1)	暂不支持，varchar，bpchar，char类型置B，nvarchar2类型置C，其余置NULL。
v80_fmt_image	character varying(3)	暂不支持，值为NULL。
data_upgraded	character varying(3)	暂不支持，值为YES。
histogram	character varying(15)	表示直方图是否存在，如果存在的话是什么类型： ● NONE：表示不存在直方图。 ● FREQUENCY：表示频率直方图。 ● EQUI_WIDTH：表示等宽直方图。
default_on_null	character varying(3)	暂不支持，值为NULL。
identity_column	character varying(3)	暂不支持，值为NULL。
sensitive_column	character varying(3)	暂不支持，值为NULL。

名称	类型	描述
evaluation_edition	character varying(128)	暂不支持，值为NULL。
unusable_before	character varying(128)	暂不支持，值为NULL。
unusable_beginning	character varying(128)	暂不支持，值为NULL。
collation	character varying(100)	列的排序规则。因该字段与保留关键字冲突，调用该字段需加视图名。
comments	text	列的注释。
schema	character varying(64)	列所属的名称空间的名称。

12.3.176 MY_TAB_COMMENTS

MY_TAB_COMMENTS视图显示当前用户拥有的所有表和视图的注释信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-278 MY_TAB_COMMENTS 字段

名称	类型	描述
owner	character varying(64)	表或视图的所有者。
table_name	character varying(64)	表或视图的名称。
comments	text	注释。
schema	character varying(64)	表所属的名称空间的名称。

12.3.177 MY_TAB_HISTOGRAMS

MY_TAB_HISTOGRAMS系统视图显示当前用户拥有的表或视图的直方图信息。所有用户都可以访问该视图。该视图同时存在于pg_catalog和sys schema下。

名称	类型	描述
table_name	character varying(128)	表名。
column_name	character varying(4000)	列名或者对象类型列的属性。
endpoint_number	numeric	直方图的桶号。
endpoint_value	numeric	暂不支持，值为NULL。
endpoint_actual_value	character varying(4000)	桶端点的实际值。

名称	类型	描述
endpoint_actual_value_raw	raw	暂不支持，值为NULL。
endpoint_repeat_count	numeric	暂不支持，值为NULL。
scope	character varying(7)	暂不支持，值为SHARED。

12.3.178 MY_TAB_PARTITIONS

MY_TAB_PARTITIONS视图存储当前用户下所有一级分区信息。当前用户下每个分区表的一级分区都会在MY_TAB_PARTITIONS中有一条记录。所有用户都可以访问。该视图同时存在于PG_CATALOG和SYS schema下。

表 12-279 MY_TAB_PARTITIONS 字段

名称	类型	描述
table_owner	character varying(64)	表的所有者。
table_name	character varying(64)	关系表名称。
partition_name	character varying(64)	分区名称。
high_value	text	分区的边界值。 • 对于范围分区和间隔分区，显示各分区的上边界值。 • 对于列表分区，显示各分区的取值列表。 • 对于哈希分区，显示各分区的编号。
tablespace_name	name	分区表的表空间名称。
schema	character varying(64)	名称空间的名称。
subpartition_count	bigint	二级分区的个数。
high_value_length	integer	分区绑定值表达式长度。
composite	character varying(3)	表是否为二级分区表。
partition_position	numeric	分区在表中的位置。
pct_free	numeric	块中可用空间的最小百分比。
pct_used	numeric	暂不支持，值为NULL。
ini_trans	numeric	初始事务数，默认值为4，非USTORE分区表时为NULL。

名称	类型	描述
max_trans	numeric	最大事务数，默认值为128，非USTORE分区表时为NULL。
initial_extent	numeric	暂不支持，值为NULL。
next_extent	numeric	暂不支持，值为NULL。
min_extent	numeric	暂不支持，值为NULL。
max_extent	numeric	暂不支持，值为NULL。
max_size	numeric	暂不支持，值为NULL。
pct_increase	numeric	暂不支持，值为NULL。
freelists	numeric	暂不支持，值为NULL。
freelist_groups	numeric	暂不支持，值为NULL。
logging	character varying(7)	是否记录对表的更改。
compression	character varying(8)	表分区的实际压缩属性。
compress_for	character varying(30)	暂不支持，值为NULL。
num_rows	numeric	分区中的行数。
blocks	numeric	暂不支持，值为NULL。
empty_blocks	numeric	暂不支持，值为NULL。
avg_space	numeric	暂不支持，值为NULL。
chain_cnt	numeric	暂不支持，值为NULL。
avg_row_len	numeric	暂不支持，值为NULL。
sample_size	numeric	暂不支持，值为NULL。
last_analyzed	timestamp with time zone	最近分析此分区的日期。
buffer_pool	character varying(7)	用于分区块的缓冲池。
flash_cache	character varying(7)	暂不支持，值为NULL。
cell_flash_cache	character varying(7)	暂不支持，值为NULL。
global_stats	character varying(3)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。
is_nested	character varying(3)	是否是嵌套表分区。
parent_table_partition	character varying(128)	暂不支持，值为NULL。
interval	character varying(3)	分区是否在间隔分区表的间隔节中。

名称	类型	描述
segment_created	character varying(4)	表分区是否创建了段或未创建。
indexing	character varying(4)	暂不支持，值为NULL。
read_only	character varying(4)	暂不支持，值为NULL。
inmemory	character varying(8)	暂不支持，值为NULL。
inmemory_priority	character varying(8)	暂不支持，值为NULL。
inmemory_distribute	character varying(15)	暂不支持，值为NULL。
inmemory_compression	character varying(17)	暂不支持，值为NULL。
inmemory_duplicate	character varying(13)	暂不支持，值为NULL。
cellmemory	character varying(24)	暂不支持，值为NULL。
inmemory_service	character varying(12)	暂不支持，值为NULL。
inmemory_service_name	character varying(100)	暂不支持，值为NULL。
memoptimize_read	character varying(8)	暂不支持，值为NULL。
memoptimize_write	character varying(8)	暂不支持，值为NULL

12.3.179 MY_TAB_STATISTICS

MY_TAB_STATISTICS视图显示数据库中有关当前用户拥有的表的统计信息。所有用户均可访问该视图，该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-280 MY_TAB_STATISTICS 字段

名称	类型	描述
table_name	character varying(128)	表名。
partition_name	character varying(128)	暂不支持，值为NULL。
partition_position	numeric	暂不支持，值为NULL。
subpartition_name	character varying(128)	暂不支持，值为NULL。

名称	类型	描述
subpartition_position	numeric	暂不支持，值为NULL。
object_type	character varying(12)	对象类型： ● TABLE ● PARTITION ● SUBPARTITION
num_rows	numeric	对象中的行数。
blocks	numeric	暂不支持，值为NULL。
empty_blocks	numeric	暂不支持，值为NULL。
avg_space	numeric	暂不支持，值为NULL。
chain_cnt	numeric	暂不支持，值为NULL。
avg_row_len	numeric	平均行长，包括行开销。
avg_space_freelist_blocks	numeric	暂不支持，值为NULL。
num_freelist_blocks	numeric	暂不支持，值为NULL。
avg_cached_blocks	numeric	暂不支持，值为NULL。
avg_cache_hit_ratio	numeric	暂不支持，值为NULL。
im_imcu_count	numeric	暂不支持，值为NULL。
im_block_count	numeric	暂不支持，值为NULL。
im_stat_update_time	timestamp(9) without time zone	暂不支持，值为NULL。
scan_rate	numeric	暂不支持，值为NULL。
sample_size	numeric	分析表格时使用的样本量。
last_analyzed	timestamp with time zone	最近分析表的日期。数据库重启后，数据会丢失。
global_stats	character varying(3)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。
stattype_locked	character varying(5)	暂不支持，值为NULL。

名称	类型	描述
stale_stats	character varying(7)	暂不支持，值为NULL。
notes	character varying(25)	暂不支持，值为NULL。
scope	character varying(7)	暂不支持，默认值SHARED。

12.3.180 MY_TAB_STATS_HISTORY

MY_TAB_STATS_HISTORY视图提供当前用户所拥有的表的表统计信息历史。所有用户都可以访问该视图。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-281 MY_TAB_STATS_HISTORY 字段

名称	类型	描述
table_name	character varying(128)	表名。
partition_name	character varying(128)	暂不支持，值为NULL。
subpartition_name	character varying(128)	暂不支持，值为NULL。
stats_update_time	timestamp(6) with time zone	统计信息更新的时间，数据库重启后，数据会丢失。

12.3.181 MY_TAB_SUBPARTITIONS

MY_TAB_SUBPARTITIONS视图显示当前用户可访问的所有子分区表信息，显示子分区名称、所属表和分区的名称、其存储属性以及DBMS_STATS包生成的统计信息。该视图所有用户可访问，仅可查看当前用户所属信息。该视图同时存在于PG_CATALOG和SYS schema下。

表 12-282 MY_TAB_SUBPARTITIONS 字段

名称	类型	描述
table_owner	character varying(64)	表的所有者。
schema	character varying(64)	模式名称。
table_name	character varying(64)	表名。

名称	类型	描述
partition_name	character varying(64)	分区名称。
subpartition_name	character varying(64)	子分区名称。
high_value	text	子分区绑定值表达式。
high_value_length	integer	子分区绑定值表达式长度。
partition_position	numeric	分区在表内的位置。
subpartition_position	numeric	子分区在分区中的位置。
tablespace_name	name	子分区所在的表空间名称。
pct_free	numeric	块中最小可用空间百分比。
pct_used	numeric	暂不支持，值为NULL。
ini_trans	numeric	初始事务数。
max_trans	numeric	最大事务数。
initial_extent	numeric	暂不支持，值为NULL。
next_extent	numeric	暂不支持，值为NULL。
min_extent	numeric	暂不支持，值为NULL。
max_extent	numeric	暂不支持，值为NULL。
max_size	numeric	暂不支持，值为NULL。
pct_increase	numeric	暂不支持，值为NULL。
freelists	numeric	暂不支持，值为NULL。
freelist_groups	numeric	暂不支持，值为NULL。
logging	character varying(7)	是否记录对表的更改： • YES • NO
compression	character varying(8)	子分区是否压缩。取值范围：ENABLED、DISABLED。
compress_for	character varying(30)	暂不支持，值为NULL。
num_rows	numeric	暂不支持，值为NULL。
blocks	numeric	暂不支持，值为NULL。
empty_blocks	numeric	暂不支持，值为NULL。

名称	类型	描述
avg_space	numeric	暂不支持，值为NULL。
chain_cnt	numeric	暂不支持，值为NULL。
avg_row_len	numeric	暂不支持，值为NULL。
sample_size	numeric	暂不支持，值为NULL。
last_analyzed	timestamp with time zone	最近分析此表的日期。数据库重启后，数据会丢失。
buffer_pool	character varying(7)	子分区的缓冲池： ● DEFAULT ● KEEP ● RECYCLE ● NULL
flash_cache	character varying(7)	暂不支持，值为NULL。
cell_flash_cache	character varying(7)	暂不支持，值为NULL。
global_stats	character varying(3)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。
interval	character varying(3)	指示分区是在间隔分区表的间隔节中（YES），还是在范围节中（NO）。
segment_created	character varying(3)	表子分区段是否已创建： ● YES：已创建。 ● NO：未创建。 ● N/A：此表没有子分区。
indexing	character varying(3)	暂不支持，值为NULL。
read_only	character varying(5)	暂不支持，值为NULL。
inmemory	character varying(8)	暂不支持，值为NULL。
inmemory_priority	character varying(8)	暂不支持，值为NULL。
inmemory_distribute	character varying(15)	暂不支持，值为NULL。

名称	类型	描述
inmemory_compression	character varying(17)	暂不支持，值为NULL。
inmemory_duplicate	character varying(13)	暂不支持，值为NULL。
inmemory_service	character varying(12)	暂不支持，值为NULL。
inmemory_service_name	character varying(1000)	暂不支持，值为NULL。
cellmemory	character varying(24)	暂不支持，值为NULL。
memoptimize_read	character varying(8)	暂不支持，值为NULL。
memoptimize_write	character varying(8)	暂不支持，值为NULL。

12.3.182 MY_TABLES

MY_TABLES视图显示当前用户拥有的表的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-283 MY_TABLES 字段

名称	类型	描述
owner	character varying(64)	表的所有者。
table_name	character varying(64)	表名称。
tablespace_name	character varying(64)	存储表的表空间名称。
dropped	character varying	当前记录是否已删除： • yes表示已删除。 • no表示未删除。
num_rows	numeric	表的估计行数。
status	character varying(8)	当前记录是否有效： • valid表示有效。
sample_size	numeric	分析表使用的样本数量。
temporary	character(1)	是否为临时表。 • y表示是临时表。 • n表示不是临时表。

名称	类型	描述
pct_free	numeric	块中空闲空间的最小比例。
ini_trans	numeric	事务的初始数量。
max_trans	numeric	事务数量的最大值。
avg_row_len	integer	平均每行的字节数。
partitioned	character varying(3)	表是否为分区表。
last_analyzed	timestamp with time zone	上次分析表的时间。数据库重启后，数据会丢失。
row_movement	character varying(8)	是否允许分区行移动。
compression	character varying(8)	是否启用表压缩。
duration	character varying(15)	临时表的期限。
logical_replication	character varying(8)	表是否启用逻辑复制。
external	character varying(3)	表是否为外表。
logging	character varying(3)	表的更改是否记入日志。
default_collation	character varying(100)	表的默认排序规则。
degree	character varying(10)	扫描表的实例数量。
table_lock	character varying(8)	是否启用表级锁。
nested	character varying(3)	是否为嵌套表。
buffer_pool	character varying(7)	表的默认缓冲池。
flash_cache	character varying(7)	暂不支持，值为NULL。
cell_flash_cache	character varying(7)	暂不支持，值为NULL。
skip_corrupt	character varying(8)	扫描表是否跳过损坏的块。
has_identity	character varying(3)	表是否具有标识列。
segment_created	character varying(3)	表段是否已被创建。
monitoring	character varying(3)	是否跟踪表的修改。
cluster_name	character varying(128)	暂不支持，值为NULL。
iot_name	character varying(128)	暂不支持，值为NULL。
pct_used	numeric	暂不支持，值为NULL。
initial_extent	numeric	暂不支持，值为NULL。
next_extent	numeric	暂不支持，值为NULL。
min_extents	numeric	暂不支持，值为NULL。

名称	类型	描述
max_extents	numeric	暂不支持，值为NULL。
pct_increase	numeric	暂不支持，值为NULL。
freelists	numeric	暂不支持，值为NULL。
freelist_groups	numeric	暂不支持，值为NULL。
backed_up	character varying(1)	暂不支持，值为NULL。
blocks	numeric	暂不支持，值为NULL。
empty_blocks	numeric	暂不支持，值为NULL。
avg_space	numeric	暂不支持，值为NULL。
chain_cnt	numeric	暂不支持，值为NULL。
avg_space_freelist_blocks	numeric	暂不支持，值为NULL。
num_freelist_blocks	numeric	暂不支持，值为NULL。
instances	character varying(10)	暂不支持，值为NULL。
cache	character varying(5)	暂不支持，值为NULL。
iot_type	character varying(12)	暂不支持，值为NULL。
secondary	character varying(1)	暂不支持，值为NULL。
global_stats	character varying(3)	暂不支持，值为NULL。
user_stats	character varying(3)	暂不支持，值为NULL。
cluster_owner	character varying(30)	暂不支持，值为NULL。
dependencies	character varying(8)	暂不支持，值为NULL。
compression_for	character varying(30)	暂不支持，值为NULL。
read_only	character varying(3)	暂不支持，值为NULL。
result_cache	character varying(7)	暂不支持，值为NULL。
clustering	character varying(3)	暂不支持，值为NULL。
activity_tracking	character varying(23)	暂不支持，值为NULL。
dml_timestamp	character varying(25)	暂不支持，值为NULL。
container_data	character varying(3)	暂不支持，值为NULL。
inmemory_priority	character varying(8)	暂不支持，值为NULL。
inmemory_distribute	character varying(15)	暂不支持，值为NULL。

名称	类型	描述
inmemory_compression	character varying(17)	暂不支持，值为NULL。
inmemory_duplicate	character varying(13)	暂不支持，值为NULL。
duplicated	character varying(1)	暂不支持，值为NULL。
sharded	character varying(1)	暂不支持，值为NULL。
hybrid	character varying(3)	暂不支持，值为NULL。
cellmemory	character varying(24)	暂不支持，值为NULL。
containers_default	character varying(3)	暂不支持，值为NULL。
container_map	character varying(3)	暂不支持，值为NULL。
extended_data_link	character varying(3)	暂不支持，值为NULL。
extended_data_link_map	character varying(3)	暂不支持，值为NULL。
inmemory_service	character varying(12)	暂不支持，值为NULL。
inmemory_service_name	character varying(1000)	暂不支持，值为NULL。
container_map_object	character varying(3)	暂不支持，值为NULL。
memoptimize_read	character varying(8)	暂不支持，值为NULL。
memoptimize_write	character varying(8)	暂不支持，值为NULL。
has_sensitive_column	character varying(3)	暂不支持，值为NULL。
admit_null	character varying(3)	暂不支持，值为NULL。
data_link_dml_enabled	character varying(3)	暂不支持，值为NULL 。
object_id_type	character varying(16)	暂不支持，值为NULL。
table_type_owner	character varying(128)	暂不支持，值为NULL。
table_type	character varying(128)	暂不支持，值为NULL。
compress_for	character varying(30)	暂不支持，值为NULL。

12.3.183 MY_TABLESPACES

MY_TABLESPACES视图显示用户拥有存储对象的表空间的描述信息。默认所有用户都可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。A数据库与GaussDB逻辑结构特性不一致。

表 12-284 MY_TABLESPACES 字段

名称	类型	描述
tablespace_name	character varying(64)	表空间名称。
block_size	numeric	暂不支持，值为NULL。
initial_extent	numeric	暂不支持，值为NULL。
next_extent	numeric	暂不支持，值为NULL。
min_extents	numeric	暂不支持，值为NULL。
max_extents	numeric	暂不支持，值为NULL。
max_size	numeric	暂不支持，值为NULL。
pct_increase	numeric	暂不支持，值为NULL。
min_extlen	numeric	暂不支持，值为NULL。
contents	character varying(9)	暂不支持，值为NULL。
status	character varying(9)	表空间状态，默认为ONLINE（在线）。
logging	character varying(9)	暂不支持，值为NULL。
force_logging	character varying(3)	暂不支持，值为NULL。
extent_management	character varying(10)	暂不支持，值为NULL。
allocation_type	character varying(9)	暂不支持，值为NULL。
segment_space_management	character varying(6)	暂不支持，值为NULL。
def_tab_compression	character varying(8)	暂不支持，值为NULL。
retention	character varying(11)	暂不支持，值为NULL。
bigfile	character varying(3)	暂不支持，值为NULL。
predicate_evaluation	character varying(7)	暂不支持，值为NULL。
encrypted	character varying(3)	暂不支持，值为NULL。
compress_for	character varying(30)	暂不支持，值为NULL。
def_inmemory	character varying(8)	暂不支持，值为NULL。
def_inmemory_priority	character varying(8)	暂不支持，值为NULL。

名称	类型	描述
def_inmemory_distribute	character varying(15)	暂不支持，值为NULL。
def_inmemory_compression	character varying(17)	暂不支持，值为NULL。
def_inmemory_duplicate	character varying(13)	暂不支持，值为NULL。
shared	character varying(12)	暂不支持，值为NULL。
def_index_compression	character varying(8)	暂不支持，值为NULL。
index_compress_for	character varying(13)	暂不支持，值为NULL。
def_cellmemory	character varying(14)	暂不支持，值为NULL。
def_inmemory_service	character varying(12)	暂不支持，值为NULL。
def_inmemory_service_name	character varying(1000)	暂不支持，值为NULL。
lost_write_protect	character varying(7)	暂不支持，值为NULL。
chunk_tablespace	character varying(1)	暂不支持，值为NULL。

12.3.184 MY_TRIGGERS

MY_TRIGGERS视图显示当前用户拥有的触发器信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-285 MY_TRIGGERS 字段

名称	类型	描述
owner	character varying(128)	触发器的所有者。
trigger_name	character varying(64)	触发器名称。
trigger_type	character varying	触发器触发的时机：before statement，before each row，after statement，after each row，instead of。
triggering_event	character varying	触发触发器的事件：update, insert, delete,truncate。
table_owner	character varying(64)	定义触发器的表的所有者。
base_object_type	character varying(18)	定义触发器的基础对象：table，view。
table_name	character varying(64)	定义触发器的表或视图名称。
column_name	character varying(4000)	暂不支持，值为NULL。

名称	类型	描述
referencing_name	character varying(422)	暂不支持，值为referencing new as new old as old。
when_clause	character varying(4000)	when的内容，必须评估TRUE为TRIGGER_BODY执行。
status	character varying(64)	• O：触发器在“origin”和“local”模式下触发。 • D：触发器被禁用。 • R：触发器在“replica”模式下触发。 • A：触发器始终触发。
description	character varying(4000)	触发器描述；用于重新创建触发器创建语句。
action_type	character varying(11)	触发体的动作类型，仅支持call。
trigger_body	text	触发器触发时执行的语句。
crossedition	character varying(7)	暂不支持，值为NULL。
before_statement	character varying(3)	暂不支持，值为NULL。
before_row	character varying(3)	暂不支持，值为NULL。
after_row	character varying(3)	暂不支持，值为NULL。
after_statement	character varying(3)	暂不支持，值为NULL。
instead_of_row	character varying(3)	暂不支持，值为NULL。
fire_once	character varying(3)	暂不支持，值为NULL。
apply_server_only	character varying(3)	暂不支持，值为NULL。

12.3.185 MY_TYPE_ATTRS

MY_TYPE_ATTRS视图显示数据库中当前用户所拥有的所有类型的属性。所有用户均可访问该视图，该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-286 MY_TYPE_ATTRS 字段

名称	类型	描述
type_name	character varying(128)	数据类型名称。

名称	类型	描述
attr_name	character varying(128)	属性名称。
attr_type_mod	character varying(7)	属性的类型修饰符： ● REF ● POINT
attr_type_owner	character varying(128)	属性类型的所有者。
attr_type_name	character varying(128)	属性类型的名称。
length	numeric	CHAR属性的长度，或VARCHAR和character varying属性的最大长度。
precision	numeric	数字或DECIMAL属性的十进制精度，或FLOAT属性的二进制精度。
scale	numeric	numeric或DECIMAL属性的小数位。
character_set_name	character varying(44)	属性的字符集名称（Char_CS或NCHAR_CS）。
attr_no	numeric	类型规范或CREATE TYPE语句中指定的属性的语法顺序编号或位置（不用作ID编号）。
inherited	character varying(3)	表示属性是否继承自超类型。 ● YES：表示继承自超类型。 ● NO：表示没有继承自超类型。

12.3.186 MY_TYPES

MY_TYPES视图用于描述当前用户所拥有的所有对象类型。所有用户都可以访问该视图。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-287 MY_TYPES 字段

名称	类型	描述
type_name	character varying(128)	类型名称。
type_oid	raw	类型的标识符(OID)。
typecode	character varying(128)	类型的类型代码。
attributes	numeric	类型中的属性数。
methods	numeric	暂不支持，值为0。

名称	类型	描述
predefined	character varying(3)	表示该类型是否是内置类型。
incomplete	character varying(3)	表示类型是否为不完整类型。
final	character varying(3)	暂不支持，值为NULL。
instantiable	character varying(3)	暂不支持，值为NULL。
persistable	character varying(3)	暂不支持，值为NULL。
supertype_owner	character varying(128)	暂不支持，值为NULL。
supertype_name	character varying(128)	暂不支持，值为NULL。
local_attributes	numeric	暂不支持，值为NULL。
local_methods	numeric	暂不支持，值为NULL。
typeid	raw	暂不支持，值为NULL。

12.3.187 MY_VIEWS

MY_VIEWS视图显示当前用户的所有视图信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-288 MY_VIEWS 字段

名称	类型	描述
owner	character varying(64)	视图的所有者。
view_name	character varying(64)	视图名称。
text	text	视图文本。
text_length	integer	视图文本长度。
TEXT_VC	character varying(4000)	视图创建语句。此列可能会截断视图文本。BEQUEATH子句将不会作为此视图中的TEXT_VC列的一部分出现。
type_text_length	numeric	暂不支持，值为NULL。
type_text	character varying(4000)	暂不支持，值为NULL。
oid_text_length	numeric	暂不支持，值为NULL。
oid_text	character varying(4000)	暂不支持，值为NULL。
view_type_owner	character varying(128)	暂不支持，值为NULL。

名称	类型	描述
view_type	character varying(128)	暂不支持，值为NULL。
superview_name	character varying(128)	暂不支持，值为NULL。
editioning_view	character varying(1)	暂不支持，值为NULL。
read_only	character varying(1)	暂不支持，值为NULL。
container_data	character varying(1)	暂不支持，值为NULL。
bequeath	character varying(12)	暂不支持，值为NULL。
origin_con_id	character varying(256)	暂不支持，值为NULL。
default_collation	character varying(100)	暂不支持，值为NULL。
containers_default	character varying(3)	暂不支持，值为NULL。
container_map	character varying(3)	暂不支持，值为NULL。
extended_data_link	character varying(3)	暂不支持，值为NULL。
extended_data_link_map	character varying(3)	暂不支持，值为NULL。
has_sensitive_column	character varying(3)	暂不支持，值为NULL。
admit_null	character varying(3)	暂不支持，值为NULL。
pdb_local_only	character varying(3)	暂不支持，值为NULL。

12.3.188 NLS_DATABASE_PARAMETERS

NLS_DATABASE_PARAMETERS列出数据库服务器端的永久NLS参数。该视图同时存在于PG_CATALOG和SYS Schema下。所有用户都可以访问。由于数据库内核不同、参数的设置格式不同等原因，该视图对相同参数的查询结果可能会和A数据库有明显差异。

表 12-289 NLS_DATABASE_PARAMETERS 字段

名称	类型	描述
parameter	character varying(128)	参数名。
value	character varying(64)	参数值。

12.3.189 NLS_INSTANCE_PARAMETERS

NLS_INSTANCE_PARAMETERS列出数据库客户端的永久NLS参数。该视图同时存在于PG_CATALOG和SYS Schema下。所有用户都可以访问。由于数据库内核不同、参数的设置格式不同等原因，该视图对相同参数的查询结果中可能会和A数据库有明显差异。

表 12-290 NLS_INSTANCE_PARAMETERS 字段

名称	类型	描述
parameter	character varying(128)	参数名。
value	character varying(64)	参数值。

12.3.190 PG_AVAILABLE_EXTENSION_VERSIONS

PG_AVAILABLE_EXTENSION_VERSIONS视图显示数据库中某些特性的扩展版本信息。该视图为内部使用，不建议用户使用。

表 12-291 PG_AVAILABLE_EXTENSION_VERSIONS 字段

名称	类型	描述
name	name	扩展名。
version	text	版本名。
installed	boolean	如果这个扩展的版本是当前已经安装了的则为真。
superuser	boolean	如果只允许系统管理员安装这个扩展则为真。
relocatable	boolean	如果扩展可以重新加载到另一个模式则为真。
schema	name	扩展必须安装到的模式名，如果部分或全部可重新定位则为NULL。
requires	name[]	先决条件扩展的名称，如果没有则为NULL。
comment	text	扩展的控制文件中的评论。

12.3.191 PG_AVAILABLE_EXTENSIONS

PG_AVAILABLE_EXTENSIONS视图显示数据库中某些特性的扩展信息。该视图为内部使用，不建议用户使用。

表 12-292 PG_AVAILABLE_EXTENSIONS 字段

名称	类型	描述
name	name	扩展名。
default_version	text	缺省版本的名称，如果没有指定则为NULL。
installed_version	text	扩展当前安装版本，如果没有安装任何版本则为NULL。

名称	类型	描述
comment	text	扩展的控制文件中的评论。

12.3.192 PG_COMM_DELAY

PG_COMM_DELAY视图展示单个DN的通信库时延状态。

表 12-293 PG_COMM_DELAY 字段

名称	类型	描述
node_name	text	节点名称。
remote_name	text	连接对端节点名称。
remote_host	text	连接对端IP地址。
stream_num	integer	当前物理连接使用的stream逻辑连接数量。
min_delay	integer	当前物理连接一分钟内探测到的最小时延，单位微秒。 **说明** 负数结果无效，请重新等待时延状态更新后再执行。
average	integer	当前物理连接一分钟内探测时延的平均值，单位微秒。
max_delay	integer	当前物理连接一分钟内探测到的最大时延，单位微秒。

12.3.193 PG_COMM_RECV_STREAM

PG_COMM_RECV_STREAM视图展示单个DN上所有的通信库接收流状态。

表 12-294 PG_COMM_RECV_STREAM 字段

名称	类型	描述
node_name	text	节点名称。
local_tid	bigint	使用此通信流的线程ID。
remote_name	text	连接对端节点名称。
remote_tid	bigint	连接对端线程ID。
idx	integer	通信对端DN在本DN内的标识编号。
sid	integer	通信流在物理连接中的标识编号。
tcp_sock	integer	通信流所使用的tcp通信socket。

名称	类型	描述
state	text	通信流当前的状态。 ● UNKNOWN：表示当前逻辑连接状态未知。 ● READY：表示逻辑连接已就绪。 ● RUN：表示逻辑连接发送报文正常。 ● HOLD：表示逻辑连接发送报文等待中。 ● CLOSED：表示关闭逻辑连接。 ● TO_CLOSED：表示将会关闭逻辑连接。
query_id	bigint	通信流对应的debug_query_id编号。
pn_id	integer	通信流所执行查询的plan_node_id编号。
send_smp	integer	通信流所执行查询send端的smpid编号。
recv_smp	integer	通信流所执行查询recv端的smpid编号。
recv_bytes	bigint	通信流接收的数据总量，单位Byte。
time	bigint	通信流当前生命周期使用时长，单位ms。
speed	bigint	通信流的平均接收速率，单位Byte/s。
quota	bigint	通信流当前的通信配额值，单位Byte。
buff_usize	bigint	通信流当前缓存的数据大小，单位Byte。

12.3.194 PG_COMM_SEND_STREAM

PG_COMM_SEND_STREAM视图展示单个DN上所有的通信库发送流状态。

表 12-295 PG_COMM_SEND_STREAM 字段

名称	类型	描述
node_name	text	节点名称。
local_tid	bigint	使用此通信流的线程ID。
remote_name	text	连接对端节点名称。
remote_tid	bigint	连接对端线程ID。
idx	integer	通信对端DN在本DN内的标识编号。
sid	integer	通信流在物理连接中的标识编号。
tcp_sock	integer	通信流所使用的tcp通信socket。

名称	类型	描述
state	text	通信流当前的状态。 ● UNKNOWN：表示当前逻辑连接状态未知。 ● READY：表示逻辑连接已就绪。 ● RUN：表示逻辑连接发送报文正常。 ● HOLD：表示逻辑连接发送报文等待中。 ● CLOSED：表示关闭逻辑连接。 ● TO_CLOSED：表示将会关闭逻辑连接。
query_id	bigint	通信流对应的debug_query_id编号。
pn_id	integer	通信流所执行查询的plan_node_id编号。
send_smp	integer	通信流所执行查询send端的smpid编号。
recv_smp	integer	通信流所执行查询recv端的smpid编号。
send_bytes	bigint	通信流发送的数据总量，单位Byte。
time	bigint	通信流当前生命周期使用时长，单位ms。
speed	bigint	通信流的平均发送速率，单位Byte/s。
quota	bigint	通信流当前的通信配额值，单位Byte。
wait_quota	bigint	通信流等待quota值产生的额外时间开销，单位ms。

12.3.195 PG_COMM_STATUS

PG_COMM_STATUS视图展示单个DN的通信库状态。

表 12-296 PG_COMM_STATUS 字段

名称	类型	描述
node_name	text	节点名称。
rxpck_rate	integer	节点通信库接收速率，单位Byte/s。
txpck_rate	integer	节点通信库发送速率，单位Byte/s。
rxkbyte_rate	bigint	节点通信库接收速率，单位KByte/s。
txkbyte_rate	bigint	节点通信库发送速率，单位KByte/s。
buffer	bigint	cmailbox的buffer大小。
memkbyte_libcomm	bigint	libcomm进程通信内存大小，单位Byte。

名称	类型	描述
memkbyte_libpq	bigint	libpq进程通信内存大小，单位Byte。
used_pm	integer	postmaster线程实时使用率。
used_sflow	integer	gs_sender_flow_controller线程实时使用率。
used_rflow	integer	gs_receiver_flow_controller线程实时使用率。
used_rloop	integer	多个gs_receivers_loop线程中高的实时使用率。
stream	integer	当前使用的逻辑连接总数。

12.3.196 PG_CONTROL_GROUP_CONFIG

PG_CONTROL_GROUP_CONFIG视图显示系统的控制组配置信息。查询该视图需要sysadmin权限。

表 12-297 PG_CONTROL_GROUP_CONFIG 字段

名称	类型	描述
pg_control_group_config	text	控制组的配置信息。

12.3.197 PG_CURSORS

PG_CURSORS视图列出了当前可用的游标。

表 12-298 PG_CURSORS 字段

名称	类型	描述
name	text	游标名。
statement	text	声明该游标时的查询语句。
is_holdable	boolean	如果该游标是持久的（就是在声明该游标的事务结束后仍然可以访问该游标）则为TRUE，否则为FALSE。
is_binary	boolean	如果该游标被声明为BINARY则为TRUE，否则为FALSE。
is_scrollable	boolean	如果该游标可以滚动（就是允许以不连续的方式检索）则为TRUE，否则为FALSE。
creation_time	timestamp with time zone	声明该游标的时间戳。

12.3.198 PG_EXT_STATS

PG_EXT_STATS视图可用来访问存储在PG_STATISTIC_EXT表里面的扩展统计信息。扩展统计信息目前包括多列统计信息。

表 12-299 PG_EXT_STATS 字段

名称	类型	引用	描述
schemaname	name	PG_NAMESPACE.nspname	表的模式名。
tablename	name	PG_CLASS.relname	表名。
attname	int2vector	PG_STATISTIC_EXT.stakey	统计信息扩展的多列信息。
inherited	boolean	-	暂不支持继承表，该字段为false。
null_frac	real	-	记录中字段组合为空的百分比。
avg_width	integer	-	字段组合记录以字节记的平均宽度。
n_distinct	real	-	• 如果大于零，表示字段组合中独立数值的估计数目。 • 如果小于零，表示独立数值的数目除以行数后乘-1得到的负数。比如，-1表示一个字段组合中独立数值的个数和行数相同。 1. 用负数形式是因为ANALYZE认为独立数值的数目是随着表增长而增长； 2. 正数的形式用于在字段看上去好像有固定的可能值数目的情况下。 • 如果等于零，表示独立数值的数目未知。
n_dndistinct	real	-	标识dn1上字段组合中非NULL的独立数值的数目。 • 如果大于零，表示独立数值的实际数目。 • 如果小于零，表示独立数值的数目除以行数后乘-1得到的负数。比如，一个字段组合的数值平均出现概率为两次，则可以表示为n_dndistinct=-0.5。 • 如果等于零，表示独立数值的数目未知。

名称	类型	引用	描述
most_common_vals	anyarray	-	一个字段组合里最常用数值的列表。如果该字段组合不存在最常用数值，则为NULL。本列保存的多列常用数值均不为NULL。
most_common_freqs	real[]	-	一个记录字段组合里最常用数值的出现频率的列表，频率由每个数值出现的次数除以行数得到。如果most_common_vals取值为NULL，则该字段取值也为NULL。
most_common_vals_null	anyarray	-	一个字段组合里最常用数值的列表。如果该字段组合不存在最常用数值，则为NULL。本列保存的多列常用数值中至少有一个值为NULL。
most_common_freqs_null	real[]	-	一个记录字段组合里最常用数值的出现频率的列表，频率由每个数值出现的次数除以行数得到。如果most_common_vals_null取值为NULL，则该字段取值也为NULL。
histogram_bounds	anyarray	-	直方图的边界值列表。

12.3.199 PG_GET_SENDERS_CATCHUP_TIME

PG_GET_SENDERS_CATCHUP_TIME视图显示数据库节点上当前活跃的主备发送线程的追赶信息。

表 12-300 PG_GET_SENDERS_CATCHUP_TIME 字段

名称	类型	描述
pid	bigint	当前sender的线程ID。
lwpid	integer	当前sender的lwpid。
local_role	text	本地的角色。
peer_role	text	对端的角色。

名称	类型	描述
state	text	当前sender的复制状态。 ● Startup：启动状态。 ● Backup：备份状态。 ● Catchup：追赶状态，表示备节点正在追赶主节点。 ● Streaming：流复制状态，当备节点追上主节点后维持Streaming状态。
type	text	当前sender的类型。 ● Wal：预写入日志类型。 ● Data：数据类型。
catchup_start	timestamp with time zone	catchup启动的时间。
catchup_end	timestamp with time zone	catchup结束的时间。

12.3.200 PG_GROUP

PG_GROUP视图可用来查看数据库认证角色和组之间的成员关系。

表 12-301 PG_GROUP 字段

名称	类型	描述
groname	name	组的名称。
grosysid	oid	组的ID。
grolist	oid[]	包含该组里面所有角色的ID的数组。

12.3.201 PG_GTT_ATTACHED_PIDS

PG_GTT_ATTACHED_PIDS视图可用来查看哪些会话正在使用全局临时表，调用pg_get_attached_pid()函数。

表 12-302 PG_GTT_ATTACHED_PIDS 字段

名称	类型	描述
schemaname	name	schema名称。
tablename	name	全局临时表名称。

名称	类型	描述
relid	oid	全局临时表的oid。
pids	bigint[]	线程pid列表。
sessionids	bigint[]	会话id列表。

12.3.202 PG_GTT_RELSTATS

PG_GTT_RELSTATS视图可用来查看当前会话所有全局临时表的基本信息，调用pg_get_gtt_relstats()函数。

表 12-303 PG_GTT_RELSTATS 字段

名称	类型	描述
schemaname	name	schema名称。
tablename	name	全局临时表名称。
relfilenode	oid	文件对象的ID。
relpages	integer	全局临时表的磁盘页面数。
reltuples	real	全局临时表的记录数。
relallvisible	integer	被标识为全可见的页面数。
relfrozenxid	xid	该表中所有在这个之前的事务ID已经被一个固定的（frozen）事务ID替换。
relminmxid	xid	预留接口，暂未启用。

12.3.203 PG_GTT_STATS

PG_GTT_STATS视图可用来查看当前会话所有全局临时表的单列统计信息，调用pg_get_gtt_statistics()函数。

表 12-304 PG_GTT_STATS 字段

名称	类型	描述
schemaname	name	schema名称。
tablename	name	全局临时表名称。
attname	name	属性名称。
inherited	boolean	是否统计有继承关系的对象。
null_frac	real	该字段中为NULL的记录的比率。

名称	类型	描述
avg_width	integer	非NULL记录的平均存储宽度，以字节计算。
n_distinct	real	标识全局统计信息中字段里唯一的非NULL数据值的数目。
most_common_vals	text[]	高频值列表，按照出现的频率排序。
most_common_freqs	real[]	高频值的频率。
histogram_bounds	text[]	等频直方图描述列中的数据分布（不包含高频值）。
correlation	real	相关系数。
most_common_elems	text[]	类型高频值列表，用于数组类型或一些其他类型。
most_common_elem_freqs	real[]	类型高频值的频率。
elem_count_histogram	real[]	数组类型直方图。

12.3.204 PG_INDEXES

PG_INDEXES视图显示数据库中每个索引的信息。

表 12-305 PG_INDEXES 字段

名称	类型	引用	描述
schemaname	name	PG_NAMESPACE.nspname	包含表和索引的模式的名称。
tablename	name	PG_CLASS.relname	此索引所在的表的名称。
indexname	name	PG_CLASS.relname	索引的名称。
tablespace	name	PG_TABLESPACE.nspname	包含索引的表空间的名称。
indexdef	text	-	索引定义（一个重建的CREATE INDEX命令）。

12.3.205 PG_LOCKS

PG_LOCKS视图存储各打开事务所持有的锁信息。

表 12-306 PG_LOCKS 字段

名称	类型	引用	描述
locktype	text	-	被锁定对象的类型：relation、extend，page、tuple、transactionid、virtualxid、object、userlock、advisory。
database	oid	PG_DATABASE.oid	被锁定对象所在数据库的OID。 • 如果被锁定的对象是共享对象，则OID为0。 • 如果被锁定的对象是一个事务，则OID为NULL。
relation	oid	PG_CLASS.oid	关系的OID，如果锁定的对象不是关系，也不是关系的一部分，则为NULL。
page	integer	-	关系内部的页面编号，如果对象不是关系页或者不是行页，则为NULL。
tuple	smallint	-	页面里边的行编号，如果对象不是行，则为NULL。
bucket	integer	-	子表对应的bucket number。如果目标不是表的话，则为NULL。
virtualxid	text	-	虚拟事务的ID，如果对象不是一个虚拟事务，则为NULL。
transactionid	xid	-	事务的ID，如果对象不是一个事务，则为NULL。
classid	oid	PG_CLASS.oid	包含该对象的系统表的OID，如果对象不是普通的数据库对象，则为NULL。
objid	oid	-	对象在其系统表内的OID，如果对象不是普通数据库对象，则为NULL。
objsubid	smallint	-	对于表的一个字段，这是字段编号；对于其他对象类型，这个字段是0；如果这个对象不是普通数据库对象，则为NULL。
virtualtransaction	text	-	持有此锁或者在等待此锁的虚拟事务的ID。
pid	bigint	-	持有或者等待这个锁的服务器逻辑线程的ID。如果锁是被一个预备事务持有的，则为NULL。
sessionid	bigint	-	持有或者等待这个锁的会话的ID。
mode	text	-	这个线程持有的或者是期望的锁模式。

名称	类型	引用	描述
granted	boolean	-	• 如果锁是持有锁，则为TRUE。 • 如果锁是等待锁，则为FALSE。
fastpath	boolean	-	如果通过fast-path获得锁，则为TRUE；如果通过主要的锁表获得，则为FALSE。
locktag	text	-	会话等待锁信息，可通过locktag_decode()函数解析。
global_sessionid	text	-	全局会话ID。

12.3.206 PG_NODE_ENV

PG_NODE_ENV视图显示当前节点的环境变量信息。

表 12-307 PG_NODE_ENV 字段

名称	类型	描述
node_name	text	当前节点的名称。
host	text	当前节点的主机名称。
process	integer	当前节点的进程号。
port	integer	当前节点的端口号。
installpath	text	当前节点的安装目录。
datapath	text	当前节点的数据目录。
log_directory	text	当前节点的日志目录。

12.3.207 PG_OS_THREADS

PG_OS_THREADS视图提供当前节点下所有线程的状态信息。

表 12-308 PG_OS_THREADS 字段

名称	类型	描述
node_name	text	当前节点的名称。
pid	bigint	当前节点进程中正在运行的线程号。
lwpid	integer	与pid对应的轻量级线程号。
thread_name	text	与pid对应的线程名称。

名称	类型	描述
creation_time	timestamp with time zone	与pid对应的线程创建的时间。

12.3.208 PG_PREPARED_STATEMENTS

PG_PREPARED_STATEMENTS视图显示当前会话所有可用的预备语句的信息。

表 12-309 PG_PREPARED_STATEMENTS 字段

名称	类型	描述
name	text	预备语句的标识符。
statement	text	创建该预备语句的查询字符串。对于从SQL创建的预备语句而言是客户端提交的PREPARE语句；对于通过前/后端协议创建的预备语句而言是预备语句自身的文本。
prepare_time	timestamp with time zone	创建该预备语句的时间戳。
parameter_types	regtype[]	该预备语句期望的参数类型，以regtype类型的数组格式出现。与该数组元素相对应的OID可以通过把regtype转换为OID值得到。
from_sql	boolean	• 如果该预备语句是通过PREPARE语句创建的则为true。 • 如果是通过前/后端协议创建的则为false。

12.3.209 PG_PREPARED_XACTS

PG_PREPARED_XACTS视图显示当前准备好进行两阶段提交的事务的信息。

表 12-310 PG_PREPARED_XACTS 字段

名称	类型	引用	描述
transaction	xid	-	预备事务的数字标识。
gid	text	-	预备事务的全局标识。
prepared	timestamp with time zone	-	事务准备好提交的时间。
owner	name	PG_AUTHID.rolname	执行该事务的用户的名称。

名称	类型	引用	描述
database	name	PG_DATABASE.datname	执行该事务的数据库的名称。

12.3.210 PG_REPLICATION_ORIGIN_STATUS

PG_REPLICATION_ORIGIN_STATUS视图可用来查看复制源的复制状态。

表 12-311 PG_REPLICATION_ORIGIN_STATUS 字段

名称	类型	描述
local_id	oid	复制源ID。
external_id	text	复制源名称。
remote_lsn	text	复制源的lsn位置。
local_lsn	text	本地的lsn位置。

12.3.211 PG_REPLICATION_SLOTS

PG_REPLICATION_SLOTS视图显示复制槽的信息。

表 12-312 PG_REPLICATION_SLOTS 字段

名称	类型	描述
slot_name	text	复制槽的名称。
plugin	text	逻辑复制槽对应的输出插件名称。
slot_type	text	复制槽的类型。 • physical：物理复制槽。 • logical：逻辑复制槽。
datoid	oid	复制槽所在的数据库OID。
database	name	复制槽所在的数据库名称。
active	boolean	复制槽是否为激活状态。 • t（true）：表示是。 • f（false）：表示不是。
xmin	xid	数据库需要为复制槽保留的最早事务的事务号。
catalog_xmin	xid	数据库需要为逻辑复制槽保留的最早的涉及系统表的事务的事务号。

名称	类型	描述
restart_lsn	text	复制槽需要的最早xlog的物理位置。
dummy_standby	boolean	预留参数。
confirmed_flush	text	逻辑复制槽专用，客户端确认接收到的日志位置。
confirmed_csn	xid	逻辑复制槽专用，客户端确认接收到的日志中最后一个事务对应的CSN。

示例：

```
gaussdb=# select * from pg_replication_slots;
 slot_name |      plugin      | slot_type | datoid | database | active | xmin | catalog_xmin | restart_lsn |
dummy_standby | confirmed_flush | confirmed_csn
-----------+------------------+-----------+--------+----------+--------+------+--------------+-------------+---------------
+-----------------+---------------
 dn_6002   |                  | physical  |      0 |          | t      |      |              | 0/3622B528  | f               |                 |
 dn_6003   |                  | physical  |      0 |          | t      |      |              | 0/3622B528  | f               |                 |
 slot_lsn  | mppdb_decoding | logical   | 131072 | db_test  | f      |      |        66658 | 0/36252350  | f               |
0/362523D0      |
 slot_test | mppdb_decoding | logical   | 131072 | db_test  | f      |      |        66658 | 0/36251718  | f
|                 |      10025527
(4 rows)
```

须知

在DN上执行查询，LSN序逻辑复制槽的confirmed_csn查询结果为空，CSN序逻辑复制槽的confirmed_flush查询结果为空。

12.3.212 PG_RLSPOLICIES

PG_RLSPOLICIES视图显示行级访问控制策略的信息。初始化用户和具有sysadmin属性的用户可以查看全部的策略信息，其他用户只能查看自己所拥有表上的策略信息。

表 12-313 PG_RLSPOLICIES 字段

名称	类型	描述
schemaname	name	行级访问控制策略作用的表对象所属的模式名称。
tablename	name	行级访问控制策略作用的表对象名称。
policyname	name	行级访问控制策略名称。
policypermissive	text	行级访问控制策略的表达式拼接方式。取值范围： • PERMISSIVE：宽容性策略，用OR表达式拼接。 • RESTRICTIVE：限制性策略，用AND表达式拼接。

名称	类型	描述
policyroles	name[]	行级访问控制策略影响的用户列表，不指定表示影响所有的用户。
policycmd	text	行级访问控制策略影响的SQL操作。
policyqual	text	行级访问控制策略的表达式。

12.3.213 PG_ROLES

PG_ROLES视图显示数据库角色的相关信息，初始化用户和具有sysadmin属性或createrole属性的用户可以查看全部角色的信息，其他用户只能查看自己的信息。

表 12-314 PG_ROLES 字段

名称	类型	引用	描述
rolname	name	-	角色名称。
rolsuper	boolean	-	该角色是否是拥有最高权限的初始系统管理员。
rolinherit	boolean	-	该角色是否继承角色的权限。
rolcreaterole	boolean	-	该角色是否可以创建其他的角色。
rolcreatedb	boolean	-	该角色是否可以创建数据库。
rolcatupdate	boolean	-	该角色是否可以直接更新系统表。只有usesysid=10的初始系统管理员拥有此权限。其他用户无法获得此权限。
rolcanlogin	boolean	-	该角色是否可以登录数据库。
rolreplication	boolean	-	该角色是否可以复制。
rolauditadmin	boolean	-	该角色是否为审计管理员。
rolsystemadmin	boolean	-	该角色是否为系统管理员。
rolconnlimit	integer	-	对于可以登录的角色，表示该角色允许发起的最大并发连接数。 -1表示无限制。
rolpassword	text	-	密文存储后的用户密码，始终为********。
rolvalidbegin	timestamp with time zone	-	账户的有效开始时间。如果没有设置有效开始时间，则为NULL。

名称	类型	引用	描述
rolvaliduntil	timestamp with time zone	-	账户的有效结束时间。如果没有设置有效结束时间，则为NULL。
rolrespool	name	-	用户可以使用的资源池。
rolparentid	oid	PG_AUTHID.rolparentid	用户所在组用户的OID。
roltabspace	text	-	用户永久表的存储空间限额，单位为kB。
roltempspace	text	-	用户临时表的存储空间限额，单位为kB。
rolspillspace	text	-	用户算子的落盘空间限额，单位为kB。
rolconfig	text[]	PG_DB_ROLE_SETTING.setconfig	运行时配置项的默认值。
oid	oid	PG_AUTHID.oid	角色的ID。
roluseft	boolean	PG_AUTHID.roluseft	角色是否可以操作外表。
rolkind	"char"	-	角色类型。 • n：普通用户，即非永久用户 • p：永久用户
nodegroup	name	-	该字段不支持。
rolmonitoradmin	boolean	-	该角色是否为监控管理员。
roloperatoradmin	boolean	-	该角色是否为运维管理员。
rolpolicyadmin	boolean	-	该角色是否为安全策略管理员。

12.3.214 PG_RULES

PG_RULES视图可用来查询重写规则的有用信息。

表 12-315 PG_RULES 字段

名称	类型	描述
schemaname	name	表的模式名称。
tablename	name	规则作用的表的名称。
rulename	name	规则的名称。
definition	text	规则的定义（由CREATE语句重新构造得来）。

12.3.215 PG_RUNNING_XACTS

PG_RUNNING_XACTS视图显示当前节点运行事务的信息。

表 12-316 PG_RUNNING_XACTS 字段

名称	类型	描述
handle	integer	事务对应的事务管理器中的槽位句柄，该值恒为-1。
gxid	xid	事务id号。
state	tinyint	事务状态。 ● 3：prepared。 ● 0：starting。
node	text	节点名称。
xmin	xid	节点上当前数据涉及的最小事务号。
vacuum	boolean	表示当前事务是否是lazy vacuum事务。 ● t（true）：表示是。 ● f（false）：表示不是。
timeline	bigint	数据库重启次数。
prepare_xid	xid	处于prepared状态的事务的id，若事务不在prepared状态，值为0。
pid	bigint	事务对应的线程id。
next_xid	xid	本地活跃事务最小CSN值。

12.3.216 PG_SECLABELS

PG_SECLABELS视图显示安全标签的信息。

表 12-317 PG_SECLABELS 字段

名称	类型	引用	描述
objoid	oid	任意OID属性	该安全标签所属的对象的OID。
classoid	oid	PG_CLASS.oid	该安全标签所属的对象所在系统表的OID。
objsubid	integer	-	对于一个在表字段上的安全标签，是字段序号（引用表本身的objoid和classoid）。对于所有其他对象类型，这个字段为0。
objtype	text	-	该标签所属的对象的类型，文本格式。例如： • table：表类型。 • column：列类型。
objnamespace	oid	PG_NAMESPACE.oid	该对象的名称空间的OID，如果不适用，取值为NULL。
objname	text	-	该标签所属的对象的名称，文本格式。
provider	text	PG_SECLABEL.provider	该标签的提供者。
label	text	PG_SECLABEL.label	安全标签名称。

12.3.217 PG_SETTINGS

PG_SETTINGS视图显示数据库运行时参数的相关信息。

表 12-318 PG_SETTINGS 字段

名称	类型	描述
name	text	参数名称。
setting	text	参数当前值。
unit	text	参数的单位。
category	text	参数的逻辑组。
short_desc	text	参数的简单描述。
extra_desc	text	参数的详细描述。
context	text	设置参数值的上下文，包括internal、postmaster、sighup、backend、superuser、user。

名称	类型	描述
vartype	text	参数类型，包括bool、enum、integer、real、string。
source	text	参数的赋值方式。
min_val	text	参数最小值。如果参数类型不是数值型，那么该字段值为null。
max_val	text	参数最大值。如果参数类型不是数值型，那么该字段值为null。
enumvals	text[]	enum类型参数合法值。如果参数类型不是enum型，那么该字段值为null。
boot_val	text	数据库启动时参数默认值。
reset_val	text	数据库重置时参数默认值。
sourcefile	text	设置参数值的配置文件。如果参数不是通过配置文件赋值，那么该字段值为null。
sourceline	integer	设置参数值的配置文件的行号。如果参数不是通过配置文件赋值，那么该字段值为null。

12.3.218 PG_SHADOW

PG_SHADOW视图显示了所有在**PG_AUTHID**中标记了rolcanlogin的角色的属性，只有系统管理员权限才可以访问此系统视图。

该视图的信息与**PG_USER**是基本一致的，区别在于后者对密码做了敏感化处理，统一显示为********。

表 12-319 PG_SHADOW 字段

名称	类型	引用	描述
usename	name	**PG_AUTHID**.rolname	用户名。
usesysid	oid	**PG_AUTHID**.oid	用户的ID。
usecreatedb	boolean	-	用户是否可以创建数据库。 • t（true）：表示是。 • f（false）：表示否。
usesuper	boolean	-	用户是否是系统管理员。 • t（true）：表示是。 • f（false）：表示否。

名称	类型	引用	描述
usecatupd	boolean	-	用户是否可以更新视图。即使是系统管理员，如果该字段的值为假，也不能更新视图。 ● t（true）：表示是。 ● f（false）：表示否。
userepl	boolean	-	用户是否可以复制数据流。 ● t（true）：表示是。 ● f（false）：表示否。
passwd	text	PG_AUTHID.rolpassword	密码密文，如果没有密码，则为NULL。
valbegin	timestamp with time zone	-	账户的有效开始时间。如果没有设置有效开始时间，则为NULL。
valuntil	timestamp with time zone	-	账户的有效结束时间。如果没有设置有效结束时间，则为NULL。
respool	name	-	用户所在的资源池。
parent	oid	-	父用户OID。
spacelimit	text	-	永久表的存储空间限额，单位KB。
useconfig	text[]	PG_DB_ROLE_SETTING.setconfig	运行时配置项的默认值。
tempspacelimit	text	-	临时表的存储空间限额，单位KB。
spillspacelimit	text	-	算子的落盘空间限额，单位KB。
usemonitoradmin	boolean	-	用户是否是监控管理员。 ● t（true）：表示是。 ● f（false）：表示否。
useoperatoradmin	boolean	-	用户是否是运维管理员。 ● t（true）：表示是。 ● f（false）：表示否。
usepolicyadmin	boolean	-	用户是否是安全策略管理员。 ● t（true）：表示是。 ● f（false）：表示否。

12.3.219 PG_STAT_ACTIVITY

PG_STAT_ACTIVITY视图显示和当前用户查询相关的信息，字段保存的是上一次执行的信息。

表 12-320 PG_STAT_ACTIVITY 字段

名称	类型	描述
datid	oid	用户会话在后台连接到的数据库OID。
datname	name	用户会话在后台连接到的数据库名称。
pid	bigint	后台线程ID。
sessionid	bigint	会话ID。
usesysid	oid	登录该后台的用户OID。
usename	name	登录该后台的用户名。
application_name	text	连接到该后台的应用名。
client_addr	inet	连接到该后台的客户端的IP地址。 如果此字段取值是null，表明是通过服务器机器上UNIX套接字连接客户端或者这是内部进程，如autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过client_addr的反向DNS查找得到。这个字段只有在启动log_hostname且使用IP连接时才非空。
client_port	integer	客户端用于与后台通讯的TCP端口号，如果使用Unix套接字，则为-1。
backend_start	timestamp with time zone	该会话开始的时间，即客户端连接服务器的时间。
xact_start	timestamp with time zone	当前活跃事务开始的时间，如果没有事务是活跃的，则为null。如果当前查询是首个事务，则这列等同于query_start列。
query_start	timestamp with time zone	当前活跃查询开始的时间， 如果state的值不是active，则这个值是上一个查询的开始时间。如果是存储过程、函数、package，则显示的是第一个查询时间，不会随着存储过程内语句运行而改变。
state_change	timestamp with time zone	上次状态改变的时间。
waiting	boolean	如果后台当前正等待锁则为true。

名称	类型	描述
enqueue	text	该字段不支持。
state	text	该后台当前总体状态。可能值是： ● active：后台正在执行一个查询。 ● idle：后台正在等待一个新的客户端命令。 ● idle in transaction：后台在事务中，但事务中没有语句在执行。 ● idle in transaction (aborted)：后台在事务中，但事务中有语句执行失败。 ● fastpath function call：后台正在执行一个fast-path函数。 ● disabled：如果后台禁用track_activities，则报告这个状态。 **说明** 普通用户只能查看到自己账户所对应的会话状态。即其他账户的state信息为空。例如以judy用户连接数据库后，在pg_stat_activity中查看到的普通用户joe及初始用户omm的state信息为空： SELECT datname, usename, usesysid, state,pid FROM pg_stat_activity; datname | usename | usesysid | state | pid ----------+---------+----------+--------+----------------- testdb | omm | 10 | | 139968752121616 testdb | omm | 10 | | 139968903116560 db_tpcc | judy | 16398 | active | 139968391403280 testdb | omm | 10 | | 139968643069712 testdb | omm | 10 | | 139968680818448 testdb | joe | 16390 | | 139968563377936 (6 rows)
resource_pool	name	用户使用的资源池。
query_id	bigint	查询语句的ID。
query	text	该后台的最新查询。如果state状态是active（活跃的），此字段显示当前正在执行的查询。所有其他情况表示上一个查询。
connection_info	text	json格式字符串，记录当前连接数据库的驱动类型、驱动版本号、当前驱动的部署路径、进程属主用户等信息（参见GUC参数connection_info）。

名称	类型	描述
unique_sql_id	bigint	语句的unique sql id。
trace_id	text	驱动传入的trace id，与应用的一次请求相关联。
top_xid	xid	事务的顶层事务ID。
current_xid	xid	事务的当前事务ID。
xlog_quantity	bigint	事务当前使用的Xlog量，单位为字节。

12.3.220 PG_STAT_ACTIVITY_NG

PG_STAT_ACTIVITY_NG视图显示在当前用户所属的逻辑数据库实例下，所有查询的相关信息。

表 12-321 PG_STAT_ACTIVITY_NG 字段

名称	类型	描述
datid	oid	用户会话在后台连接到的数据库OID。
datname	name	用户会话在后台连接到的数据库名称。
pid	bigint	后台线程ID。
sessionid	bigint	会话ID。
usesysid	oid	登录该后台的用户OID。
usename	name	登录该后台的用户名。
application_name	text	连接到该后台的应用名。
client_addr	inet	连接到该后台的客户端的IP地址。 如果该字段取值是null，表明是通过服务器机器上UNIX套接字连接客户端或者这是内部线程，如autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过client_addr的反向DNS查找得到。这个字段只有在启动log_hostname且使用IP连接时才非空。
client_port	integer	客户端用于与后台通讯的TCP端口号，如果使用Unix套接字，则为-1。
backend_start	timestamp with time zone	该会话开始的时间，即当客户端连接服务器的时间。

名称	类型	描述
xact_start	timestamp with time zone	当前活跃事务开始的时间，如果没有事务是活跃的，则为null。如果当前查询是首个事务，则这列等同于query_start列。
query_start	timestamp with time zone	当前活跃查询开始的时间， 如果state的值不是active，则这个值是上一个查询的开始时间。如果是存储过程、函数、package，则显示的是第一个查询时间，不会随着存储过程内语句运行而改变。
state_change	timestamp with time zone	上次状态改变的时间。
waiting	boolean	如果后台当前正等待锁则为true。否则为false。
enqueue	text	语句当前排队状态。可能值是： ● waiting in queue：表示语句在排队中。 ● 空：表示语句正在运行。

名称	类型	描述
state	text	该后台当前总体状态。可能值是： ● active：后台正在执行一个查询。 ● idle：后台正在等待一个新的客户端命令。 ● idle in transaction：后台在事务中，但事务中没有语句在执行。 ● idle in transaction (aborted)：后台在事务中，但事务中有语句执行失败。 ● fastpath function call：后台正在执行一个fast-path函数。 ● disabled：如果后台禁用track_activities，则报告这个状态。 **说明** 普通用户只能查看到自己账户所对应的会话状态。即其他账户的state信息为空。例如以judy用户连接数据库后，在pg_stat_activity中查看到的普通用户joe及初始用户omm的state信息为空： SELECT datname, usename, usesysid, state,pid FROM pg_stat_activity_ng; datname | usename | usesysid | state | pid ----------+---------+----------+--------+----------------- testdb | omm | 10 | | 139968752121616 testdb | omm | 10 | | 139968903116560 db_tpcds | judy | 16398 | active | 139968391403280 testdb | omm | 10 | | 139968643069712 testdb | omm | 10 | | 139968680818448 testdb | joe | 16390 | | 139968563377936 (6 rows)
resource_pool	name	用户使用的资源池。
query_id	bigint	查询语句的ID。
query	text	该后台的最新查询。如果state状态是active（活跃的），此字段显示当前正在执行的查询。所有其他情况表示上一个查询。
node_group	text	语句所属用户对应的逻辑数据库实例。

12.3.221 PG_STAT_ALL_INDEXES

PG_STAT_ALL_INDEXES视图可用来查询当前数据库中的每个索引行，显示访问特定索引的统计。

索引可以通过简单的索引扫描或位图索引扫描进行使用。位图扫描中几个索引的输出可以通过AND或者OR规则进行组合，因此当使用位图扫描的时候，很难将独立堆行抓取与特定索引进行组合，因此，每一次位图扫描都会增加
pg_stat_all_indexes.idx_tup_read使用索引的计数，并且增加
pg_stat_all_tables.idx_tup_fetch表的计数，但不影响
pg_stat_all_indexes.idx_tup_fetch。

表 12-322 PG_STAT_ALL_INDEXES 字段

名称	类型	描述
relid	oid	索引所在的表的OID。
indexrelid	oid	索引的OID。
schemaname	name	索引的模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	通过索引上扫描返回的索引项数。
idx_tup_fetch	bigint	使用该索引的简单索引扫描在原表中抓取的活跃表行数。

12.3.222 PG_STAT_ALL_TABLES

PG_STAT_ALL_TABLES视图可用来查询当前数据库中每个表的信息（包括TOAST表），显示特定表的统计信息。

表 12-323 PG_STAT_ALL_TABLES 字段

名称	类型	描述
relid	oid	表的OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。

名称	类型	描述
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（比如没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计不活跃行数。
last_vacuum	timestamp with time zone	最后一次清理该表的时间。
last_autovacuum	timestamp with time zone	这个表上次被autovacuum守护线程清理的时间。
last_analyze	timestamp with time zone	上次分析该表的时间。
last_autoanalyze	timestamp with time zone	这个表上次被autovacuum守护线程分析的时间。
vacuum_count	bigint	这个表被清理的次数。
autovacuum_count	bigint	这个表被autovacuum清理的次数。
analyze_count	bigint	这个表被分析的次数。
autoanalyze_count	bigint	这个表被autovacuum守护进程分析的次数。
last_data_changed	timestamp with time zone	记录这个表上一次数据发生变化的时间（引起数据变化的操作包括对表的修改insert/update/delete/truncate和对表的分区(partition/subpartition)的修改exchange/truncate/drop），该列数据仅在本地数据库主节点记录。

12.3.223 PG_STAT_BAD_BLOCK

PG_STAT_BAD_BLOCK视图显示自节点启动后，读取数据时出现Page校验失败的统计信息。

表 12-324 PG_STAT_BAD_BLOCK 字段

名称	类型	描述
nodename	text	节点名。

名称	类型	描述
databaseid	integer	数据库OID。
tablespaceid	integer	表空间OID。
relfilenode	integer	文件对象ID。
bucketid	smallint	一致性hash bucket ID。
forknum	integer	文件类型。取值如下： 0：数据主文件。 1：FSM文件。 2：VM文件。 3：BCM文件。
error_count	integer	出现校验失败的次数。
first_time	timestamp with time zone	第一次出现时间。
last_time	timestamp with time zone	最后一次出现时间。

12.3.224 PG_STAT_BGWRITER

PG_STAT_BGWRITER视图显示后端写进程活动的统计信息。

表 12-325 PG_STAT_BGWRITER 字段

名称	类型	描述
checkpoints_timed	bigint	定期执行的检查点数。
checkpoints_req	bigint	主动执行的检查点数。
checkpoint_write_time	double precision	将文件写入到磁盘时，在检查点处理部分花费的时间总量，以毫秒为单位。
checkpoint_sync_time	double precision	将文件同步到磁盘时，在检查点处理部分花费的时间总量，以毫秒为单位。
buffers_checkpoint	bigint	检查点写入的缓冲区的数量。
buffers_clean	bigint	后端写进程写入的缓冲区的数量。
maxwritten_clean	bigint	后端写进程因写入的缓冲区过多导致的清理扫描停止的次数。

名称	类型	描述
buffers_backend	bigint	后端直接写入的缓冲区的数量。
buffers_backend_fsync	bigint	后端自己执行fsync调用的次数（通常情况下，即使后端自己执行了这些写入动作，后端写进程也会再处理一次）。
buffers_alloc	bigint	分配的缓冲区数量。
stats_reset	timestamp with time zone	这些统计被重置的时间。

12.3.225 PG_STAT_DATABASE

PG_STAT_DATABASE视图显示GaussDB中每个数据库的统计信息。

表 12-326 PG_STAT_DATABASE 字段

名称	类型	描述
datid	oid	数据库的OID。
datname	name	数据库的名称。
numbackends	integer	当前连接到该数据库的后端数。 这是该视图中唯一一个返回当前状态值的字段，其他字段返回的都是自上次重置之后的累计值。
xact_commit	bigint	该数据库中已经提交的事务数。
xact_rollback	bigint	该数据库中已经回滚的事务数。
blks_read	bigint	在该数据库中读取的磁盘块的数量。
blks_hit	bigint	已在缓冲区缓存中找到磁盘块的次数，因此不需要读取（只统计在缓冲区缓存找到的，不包括在操作系统的文件系统缓存中找到的）。
tup_returned	bigint	通过数据库查询返回的行数。
tup_fetched	bigint	通过数据库查询抓取的行数。
tup_inserted	bigint	通过数据库查询插入的行数。
tup_updated	bigint	通过数据库查询更新的行数。
tup_deleted	bigint	通过数据库查询删除的行数。
conflicts	bigint	由于数据库恢复冲突取消的查询数量（只在备用服务器发生的冲突）。请参见PG_STAT_DATABASE_CONFLICTS获取更多信息。

名称	类型	描述
temp_files	bigint	通过数据库查询创建的临时文件数量。计算所有临时文件， 无论该临时文件为什么创建（比如排序或者哈希）， 也不管log_temp_files参数如何设置。
temp_bytes	bigint	通过数据库查询写入临时文件的数据总量。计算所有临时文件，无论该临时文件为什么创建，也不管log_temp_files参数如何设置。
deadlocks	bigint	该数据库中检测到的死锁数。
blk_read_time	double precision	通过数据库后端读取数据文件块花费的时间，以毫秒计算。
blk_write_time	double precision	通过数据库后端写入数据文件块花费的时间，以毫秒计算。
stats_reset	timestamp with time zone	当前状态统计被重置的时间。

12.3.226 PG_STAT_DATABASE_CONFLICTS

PG_STAT_DATABASE_CONFLICTS视图显示数据库冲突状态的统计信息。

表 12-327 PG_STAT_DATABASE_CONFLICTS 字段

名称	类型	描述
datid	oid	数据库标识。
datname	name	数据库名称。
confl_tablespace	bigint	冲突的表空间的数目。
confl_lock	bigint	冲突的锁数目。
confl_snapshot	bigint	冲突的快照数目。
confl_bufferpin	bigint	冲突的缓冲区数目。
confl_deadlock	bigint	冲突的死锁数目。

12.3.227 PG_STAT_REPLICATION

PG_STAT_REPLICATION视图显示日志同步线程的信息，如发起端发送日志位置，接收端接收日志位置等。

表 12-328 PG_STAT_REPLICATION 字段

名称	类型	描述
pid	bigint	线程的PID。
usesysid	oid	用户系统ID。
usename	name	用户名。
application_name	text	程序名称。
client_addr	inet	客户端地址。
client_hostname	text	客户端名。
client_port	integer	客户端端口。
backend_start	timestamp with time zone	程序启动时间。
state	text	日志同步线程的状态： ● startup：线程正在启动。 ● catchup：线程正在建立备用服务器和主服务器的连接。 ● streaming：线程已建立备用服务器和主服务器的连接，正在进行数据的流复制。 ● backup：线程正在发送备份。 ● stopping：线程正在停止。
sender_sent_location	text	发送端发送日志位置。
receiver_write_location	text	接收端write日志位置。
receiver_flush_location	text	接收端flush日志位置。
receiver_replay_location	text	接收端replay日志位置。
sync_priority	integer	同步复制的优先级（0表示异步）。

名称	类型	描述
sync_state	text	同步状态： ● async：异步复制。 ● sync：同步复制。 ● potential：该备用服务器现在是异步的，但假如一个当前的同步服务器发生故障，该服务器会变成同步的。 ● quorum：在同步与异步之间切换，保证备机中有大于一定数量的同步备机，同步备机数量一般为(n+1)/2-1，n为总副本个数。是否为同步备机取决于是否先接到了日志。详情可参考synchronous_standby_names参数描述。

12.3.228 PG_STAT_SYS_INDEXES

PG_STAT_SYS_INDEXES视图显示pg_catalog、information_schema模式中所有系统表的索引状态信息。

表 12-329 PG_STAT_SYS_INDEXES 字段

名称	类型	描述
relid	oid	该索引所在的表的OID。
indexrelid	oid	索引的OID。
schemaname	name	索引的模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	该索引上扫描返回的索引项数。
idx_tup_fetch	bigint	使用该索引的简单索引扫描在原表中抓取的活跃行数。

12.3.229 PG_STAT_SYS_TABLES

PG_STAT_SYS_TABLES视图显示pg_catalog、information_schema模式的所有命名空间中系统表的统计信息。

表 12-330 PG_STAT_SYS_TABLES 字段

名称	类型	描述
relid	oid	表的OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（比如没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计不活跃行数。
last_vacuum	timestamp with time zone	上次手动清理该表的时间（不计算VACUUM FULL）。
last_autovacuum	timestamp with time zone	上次被autovacuum守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析这个表的时间。
last_autoanalyze	timestamp with time zone	上次被autovacuum守护进程分析的时间。
vacuum_count	bigint	这个表被手动清理的次数（不计算VACUUM FULL）。
autovacuum_count	bigint	这个表被autovacuum清理的次数。
analyze_count	bigint	这个表被手动分析的次数。
autoanalyze_count	bigint	这个表被autovacuum守护进程分析的次数。

名称	类型	描述
last_data_changed	timestamp with time zone	这个表数据最近修改时间。

12.3.230 PG_STAT_USER_FUNCTIONS

PG_STAT_USER_FUNCTIONS视图显示命名空间中用户自定义函数（函数语言为非内部语言）的状态信息。

表 12-331 PG_STAT_USER_FUNCTIONS 字段

名称	类型	描述
funcid	oid	函数标识。
schemaname	name	模式的名称。
funcname	name	函数名称。
calls	bigint	函数被调用的次数。
total_time	double precision	函数的总执行时长。
self_time	double precision	当前线程调用函数的总的时长。

12.3.231 PG_STAT_USER_INDEXES

PG_STAT_USER_INDEXES视图显示数据库中用户自定义普通表和toast表的索引状态信息。

表 12-332 PG_STAT_USER_INDEXES 字段

名称	类型	描述
relid	oid	该索引所在的表的OID。
indexrelid	oid	索引的OID。
schemaname	name	索引的模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	该索引上扫描返回的索引项数。
idx_tup_fetch	bigint	使用该索引的简单索引扫描在原表中抓取的活跃行数。

12.3.232 PG_STAT_USER_TABLES

PG_STAT_USER_TABLES视图显示所有命名空间中用户自定义普通表和toast表的状态信息。

表 12-333 PG_STAT_USER_TABLES 字段

名称	类型	描述
relid	oid	表的OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（即没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计不活跃行数。
last_vacuum	timestamp with time zone	上次手动清理该表的时间（不计算VACUUM FULL）。
last_autovacuum	timestamp with time zone	上次被autovacuum守护线程清理的时间。
last_analyze	timestamp with time zone	上次手动分析这个表的时间。
last_autoanalyze	timestamp with time zone	上次被autovacuum守护线程分析的时间。
vacuum_count	bigint	这个表被手动清理的次数（不计算VACUUM FULL）。
autovacuum_count	bigint	这个表被autovacuum清理的次数。
analyze_count	bigint	这个表被手动分析的次数。
autoanalyze_count	bigint	这个表被autovacuum守护进程分析的次。

名称	类型	描述
last_data_changed	timestamp with time zone	这个表数据最近修改时间。

12.3.233 PG_STAT_XACT_ALL_TABLES

PG_STAT_XACT_ALL_TABLES视图显示命名空间中所有普通表和toast表的事务状态信息。

表 12-334 PG_STAT_XACT_ALL_TABLES 字段

名称	类型	描述
relid	oid	表的OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（比如没有更新所需的单独索引）。

12.3.234 PG_STAT_XACT_SYS_TABLES

PG_STAT_XACT_SYS_TABLES视图显示命名空间中系统表的事务状态信息。

表 12-335 PG_STAT_XACT_SYS_TABLES 字段

名称	类型	描述
relid	oid	表的OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。

名称	类型	描述
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（比如没有更新所需的单独索引）。

12.3.235 PG_STAT_XACT_USER_FUNCTIONS

PG_STAT_XACT_USER_FUNCTIONS视图包含每个函数的执行的统计信息。

表 12-336 PG_STAT_XACT_USER_FUNCTIONS 字段

名称	类型	描述
funcid	oid	函数标识。
schemaname	name	模式的名称。
funcname	name	函数名称。
calls	bigint	函数被调用的次数。
total_time	double precision	函数的总执行时长。
self_time	double precision	当前线程调用函数的总的时长。

12.3.236 PG_STAT_XACT_USER_TABLES

PG_STAT_XACT_USER_TABLES视图显示命名空间中用户表的事务状态信息。

表 12-337 PG_STAT_XACT_USER_TABLES 字段

名称	类型	描述
relid	oid	表的OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。

名称	类型	描述
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（比如没有更新所需的单独索引）。

12.3.237 PG_STATS

PG_STATS视图可用来查看存储在pg_statistic表里面的单列统计信息。该视图记录的统计信息更新时间间隔由参数autovacuum_naptime设置。

表 12-338 PG_STATS 字段

名称	类型	引用	描述
schemaname	name	PG_NAMESPACE.nspname	包含表的模式名。
tablename	name	PG_CLASS.relname	表名。
attname	name	PG_ATTRIBUTE.attname	字段的名称。
inherited	boolean	-	暂不支持继承表，该字段为false。
null_frac	real	-	记录中字段为空的百分比。
avg_width	integer	-	字段记录以字节记的平均宽度。
n_distinct	real	-	• 如果大于零，表示字段中独立数值的估计数目。 • 如果小于零，表示独立数值的数目除以行数后乘-1得到的负数。比如，-1表示一个唯一字段，独立数值的个数和行数相同。 1. 用负数形式是因为ANALYZE认为独立数值的数目是随着表增长而增长； 2. 正数的形式用于在字段看上去好像有固定的可能值数目的情况下。 • 如果等于零，表示独立数值的数目未知。

名称	类型	引用	描述
n_dndistinct	real	-	标识dn1上字段中非NULL的独立数值的数目。 ● 如果大于零，表示独立数值的实际数目。 ● 如果小于零，表示独立数值的数目除以行数后乘-1得到的负数。比如，一个字段的数值平均出现概率为两次，则可以表示为n_dndistinct=-0.5。 ● 如果等于零，表示独立数值的数目未知。
most_common_vals	anyarray	-	一个字段里最常用数值的列表。如果该字段不存在最常用数值，则为NULL。
most_common_freqs	real[]	-	一个记录字段里最常用数值的出现频率的列表，频率由每个数值出现的次数除以行数得到。如果most_common_vals是NULL，则为NULL。
histogram_bounds	anyarray	-	由排除了空值和MCV值之外的取值组成的等频直方图。如果某个数值出现在most_common_vals中，则不出现在直方图里。如果字段数据类型没有<操作符或者most_common_vals列表包含了该字段所有取值，则这个字段的直方图信息为NULL。
correlation	real	-	字段值的物理行序和逻辑行序的相关性。取值范围从-1到+1。该值接近-1或者+1的时候，因为减少了对磁盘的随机访问，索引扫描的开销比接近零的时候更少。如果字段数据类型没有<操作符，则这个字段的相关性为NULL。
most_common_elems	anyarray	-	一个最常用的非空元素的列表。
most_common_elem_freqs	real[]	-	一个记录最常用的非空元素的出现频率的列表。
elem_count_histogram	real[]	-	对于独立的非空元素的统计直方图。

12.3.238 PG_STATIO_ALL_INDEXES

PG_STATIO_ALL_INDEXES视图可用来查询当前数据库中的每个索引行的信息， 显示特定索引的I/O的统计信息。

表 12-339 PG_STATIO_ALL_INDEXES 字段

名称	类型	描述
relid	oid	索引的表的OID。
indexrelid	oid	该索引的OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	bigint	从索引中读取的磁盘块数。
idx_blks_hit	bigint	索引命中缓存数。

12.3.239 PG_STATIO_ALL_SEQUENCES

PG_STATIO_ALL_SEQUENCES视图显示当前数据库中每个序列的I/O的统计信息。

表 12-340 PG_STATIO_ALL_SEQUENCES 字段

名称	类型	描述
relid	oid	序列OID。
schemaname	name	序列的模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列命中缓存数。

12.3.240 PG_STATIO_ALL_TABLES

PG_STATIO_ALL_TABLES视图可用来查询当前数据库中每个表（包括TOAST表）的I/O统计信息。

表 12-341 PG_STATIO_ALL_TABLES 字段

名称	类型	描述
relid	oid	表OID。

名称	类型	描述
schemaname	name	该表的模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表命中缓存数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	从该表的TOAST表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的TOAST表命中缓存数（如果存在）。
tidx_blks_read	bigint	从该表的TOAST表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该表的TOAST表索引命中缓存数（如果存在）。

12.3.241 PG_STATIO_SYS_INDEXES

PG_STATIO_SYS_INDEXES视图显示命名空间中所有系统表索引的I/O状态信息。

表 12-342 PG_STATIO_SYS_INDEXES 字段

名称	类型	描述
relid	oid	该索引所在的表的OID。
indexrelid	oid	该索引的OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	bigint	从索引中读取的磁盘块数。
idx_blks_hit	bigint	索引命中缓存数。

12.3.242 PG_STATIO_SYS_SEQUENCES

PG_STATIO_SYS_SEQUENCES视图显示命名空间中所有序列的I/O状态信息。

表 12-343 PG_STATIO_SYS_SEQUENCES 字段

名称	类型	描述
relid	oid	序列OID。
schemaname	name	序列的模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列命中缓存数。

12.3.243 PG_STATIO_SYS_TABLES

PG_STATIO_SYS_TABLES视图显示命名空间中所有系统表的I/O状态信息。

表 12-344 PG_STATIO_SYS_TABLES 字段

名称	类型	描述
relid	oid	表OID。
schemaname	name	该表的模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表命中缓存数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	从该表的TOAST表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的TOAST表命中缓存数（如果存在）。
tidx_blks_read	bigint	从该表的TOAST表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该表的TOAST表索引命中缓存数（如果存在）。

12.3.244 PG_STATIO_USER_INDEXES

PG_STATIO_USER_INDEXES视图显示命名空间中所有用户关系表索引的I/O状态信息。

表 12-345 PG_STATIO_USER_INDEXES 字段

名称	类型	描述
relid	oid	该索引所在的表的OID。
indexrelid	oid	该索引的OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名.
indexrelname	name	索引名称。
idx_blks_read	bigint	从索引中读取的磁盘块数。
idx_blks_hit	bigint	索引命中缓存数。

12.3.245 PG_STATIO_USER_SEQUENCES

PG_STATIO_USER_SEQUENCES视图显示命名空间中所有用户关系表序列的I/O状态信息。

表 12-346 PG_STATIO_USER_SEQUENCES 字段

名称	类型	描述
relid	oid	序列OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列命中缓存数。

12.3.246 PG_STATIO_USER_TABLES

PG_STATIO_USER_TABLES视图显示命名空间中所有用户关系表的I/O状态信息。

表 12-347 PG_STATIO_USER_TABLES 字段

名称	类型	描述
relid	oid	表OID。
schemaname	name	该表的模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。

名称	类型	描述
heap_blks_hit	bigint	该表命中缓存数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	从该表的TOAST表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的TOAST表命中缓存数（如果存在）。
tidx_blks_read	bigint	从该表的TOAST表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该表的TOAST表索引命中缓存数（如果存在）。

12.3.247 PG_TABLES

PG_TABLES视图可用来查询对数据库中每个表的有用信息。

表 12-348 PG_TABLES 字段

名称	类型	引用	描述
schemaname	name	PG_NAMESPACE.nspname	表的模式名。
tablename	name	PG_CLASS.relname	表名。
tableowner	name	pg_get_userbyid(PG_CLASS.relowner)	表的所有者。
tablespace	name	PG_TABLESPACE.spcname	包含表的表空间，默认为NULL。
hasindexes	boolean	PG_CLASS.relhasindex	如果表上有索引（或者最近拥有）则为TRUE，否则为FALSE。
hasrules	boolean	PG_CLASS.relhasruls	如果表上有规则，则为TRUE，否则为FALSE。
hastriggers	boolean	PG_CLASS.RELHASTRIGGERS	如果表上有触发器，则为TRUE，否则为FALSE。
tablecreator	name	pg_get_userbyid(PG_OBJECT.creator)	表的所有者。
created	timestamp with time zone	PG_OBJECT.ctime	表的创建时间。

名称	类型	引用	描述
last_ddl_time	timestamp with time zone	PG_OBJECT.mtime	最后一次对该表执行DDL操作的时间。

12.3.248 PG_TDE_INFO

PG_TDE_INFO视图显示整个数据库的加密信息。

表 12-349 PG_TDE_INFO 字段

名称	类型	描述
is_encrypt	boolean	是否是加密数据库。 • f：非加密数据库。 • t：加密数据库。
g_tde_algo	text	加密算法。 • SM4-CTR-128 • AES-CTR-128
remain	text	保留字段。

12.3.249 PG_THREAD_WAIT_STATUS

通过PG_THREAD_WAIT_STATUS视图可以检测当前实例中工作线程（backend thread）以及辅助线程（auxiliary thread）的阻塞等待情况。

表 12-350 PG_THREAD_WAIT_STATUS 字段

名称	类型	描述
node_name	text	当前节点的名称。
db_name	text	数据库名称。
thread_name	text	线程名称。
query_id	bigint	查询ID，对应debug_query_id。
tid	bigint	当前线程的线程号。
sessionid	bigint	当前会话ID。
lwtid	integer	当前线程的轻量级线程号。
psessionid	bigint	父会话ID。

名称	类型	描述
tlevel	integer	streaming线程的层级。
smpid	integer	并行线程的ID。
wait_status	text	当前线程的等待状态。等待状态的详细信息请参见**表12-351**。
wait_event	text	如果wait_status是acquire lock、acquire lwlock、wait io三种类型，此列描述具体的锁、轻量级锁、IO的信息。否则是空。
locktag	text	当前线程正等待获取的锁的信息。
lockmode	text	当前线程正等待获取的锁的模式。包含表级锁、行级锁、页级锁下的各模式。
block_sessionid	bigint	阻塞当前线程获取锁的会话标识。
global_sessionid	text	全局会话ID。

wait_status列的等待状态有以下状态。

表 12-351 等待状态列表

wait_status值	含义
none	没在等任意事件。
acquire lock	等待加锁，要么加锁成功，要么加锁等待超时。
acquire lwlock	等待获取轻量级锁。
wait io	等待IO完成。
wait cmd	等待完成读取网络通信包。
wait pooler get conn	等待pooler完成获取连接。
wait pooler abort conn	等待pooler完成终止连接。
wait pooler clean conn	等待pooler完成清理连接。
pooler create conn: [nodename], total N	等待pooler建立连接，当前正在与nodename指定节点建立连接，且仍有N个连接等待建立。
get conn	获取到其他节点的连接。

wait_status值	含义
set cmd: [nodename]	在连接上执行SET/RESET/TRANSACTION BLOCK LEVEL PARA SET/SESSION LEVEL PARA SET，当前正在nodename指定节点上执行。
cancel query	取消某连接上正在执行的SQL语句。
stop query	停止某连接上正在执行的查询。
wait node: [nodename](plevel), total N, [phase]	等待接收与某节点的连接上的数据，当前正在等待nodename节点plevel线程的数据，且仍有N个连接的数据待返回。如果状态包含phase信息，则可能的阶段状态有： ● begin：表示处于事务开始阶段。 ● commit：表示处于事务提交阶段。 ● rollback：表示处于事务回滚阶段。
wait transaction sync: xid	等待xid指定事务同步。
wait wal sync	等待特定LSN的wal log完成到备机的同步。
wait data sync	等待完成数据页到备机的同步。
wait data sync queue	等待把行存的数据页放入同步队列。
flush data: [nodename](plevel), [phase]	等待向网络中nodename指定节点的plevel对应线程发送数据。如果状态包含phase信息，则可能的阶段状态为wait quota，即当前通信流正在等待quota值。 quota即流控大小，表示网络通道可接收的数据大小。 wait quota具体表示发送端（数据生产者）等待接收端（数据消费者）发送当前连接的quota信息。
stream get conn: [nodename], total N	初始化stream flow时，等待与nodename节点的consumer对象建立连接，且当前有N个待建连对象。
wait producer ready: [nodename](plevel), total N	初始化stream flow时，等待每个producer都准备好，当前正在等待nodename节点plevel对应线程的producer对象准备好，且仍有N个producer对象处于等待状态。
synchronize quit	stream plan结束时，等待stream线程组内的线程统一退出。
wait stream nodegroup destroy	stream plan结束时，等待销毁stream node group。
wait active statement	等待作业执行，正在资源负载管控中。

wait_status值	含义
analyze: [relname], [phase]	当前正在对表relname执行analyze。如果状态包含phase信息，则为autovacuum，表示是数据库自动开启AutoVacuum线程执行的analyze分析操作。
vacuum: [relname], [phase]	当前正在对表relname执行vacuum。如果状态包含phase信息，则为autovacuum，表示是数据库自动开启AutoVacuum线程执行的vacuum清理操作。
vacuum full: [relname]	当前正在对表relname执行vacuum full清理。
create index	当前正在创建索引。
HashJoin - [build hash | write file]	当前是HashJoin算子，主要关注耗时的执行阶段。 ● build hash：表示当前HashJoin算子正在建立哈希表。 ● write file：表示当前HashJoin算子正在将数据写入磁盘。
HashAgg - [build hash | write file]	当前是HashAgg算子，主要关注耗时的执行阶段。 ● build hash：表示当前HashAgg算子正在建立哈希表。 ● write file：表示当前HashAgg算子正在将数据写入磁盘。
HashSetop - [build hash | write file]	当前是HashSetop算子，主要关注耗时的执行阶段。 ● build hash：表示当前HashSetop算子正在建立哈希表。 ● write file：表示当前HashSetop算子正在将数据写入磁盘。
Sort | Sort - [fetch tuple | write file]	当前是Sort算子做排序，fetch tuple表示Sort算子正在获取tuple，write file表示Sort算子正在将数据写入磁盘。
Material | Material - write file	当前是Material算子，write file表示Material算子正在将数据写入磁盘。
NestLoop	当前是NestLoop算子。
wait memory	等待内存获取。
wait sync consumer next step	Stream算子等待消费者执行。
wait sync producer next step	Stream算子等待生产者执行。
vacuum gpi	vacuum或者autovacuum流程中global partition index清理。

wait_status值	含义
standby read recovery conflict	备机只读与日志回放产生冲突。
standby get snapshot	备机只读获取快照。
prune table	等待堆表清理历史删除数据。
prune index	等待索引清理历史删除数据。
wait reserve td	等待分配ustore事务槽。
wait td rollback	等待ustore事务槽回滚。
wait available td	等待ustore可用的事务槽。
wait transaction rollback	等待事务回滚。
wait sync bgworkers	等待并行创建索引的子线程完成本地扫描和排序。
security audit write pipe	等待将审计日志写入管道。
wait fetch undo record	等待读取目标undo记录。
wait heap hot search buffer	等待通过hot链读取满足快照的Astore元组。
wait exclusive lwlock	防饿死机制触发，新的轻量级锁加锁请求等待之前被阻塞的轻量级锁获取到。

当wait_status为acquire lwlock、acquire lock或者wait io时，表示有等待事件。正在等待获取wait_event列对应类型的轻量级锁、事务锁，或者正在进行IO。

其中，wait_status值为acquire lwlock（轻量级锁）时对应的wait_event等待事件类型与描述信息如下。（wait_event为extension时，表示此时的轻量级锁是动态分配的锁，未被监控。）

表 12-352 轻量级锁等待事件列表

wait_event类型	类型描述
ShmemIndexLock	用于保护共享内存中的主索引哈希表。
OidGenLock	用于避免不同线程产生相同的OID。
XidGenLock	用于避免两个事务获得相同的xid。
ProcArrayLock	用于避免并发访问或修改ProcArray共享数组。
SInvalReadLock	用于避免与清理失效消息并发执行。
SInvalWriteLock	用于避免与其它写失效消息、清理失效消息并发执行。
WALInsertLock	用于避免与其它WAL插入操作并发执行。
WALWriteLock	用于避免并发WAL写盘。

wait_event类型	类型描述
ControlFileLock	用于避免pg_control文件的读写并发、写写并发。
CheckpointLock	用于避免多个checkpoint并发执行。
CLogControlLock	用于避免并发访问或者修改Clog控制数据结构。
SubtransControlLock	用于避免并发访问或者修改子事务控制数据结构。
MultiXactGenLock	用于串行分配唯一MultiXact id。
MultiXactOffsetControlLock	用于避免对pg_multixact/offset的写写并发和读写并发。
MultiXactMemberControlLock	用于避免对pg_multixact/members的写写并发和读写并发。
RelCacheInitLock	用于失效消息场景对init文件进行操作时加锁。
CheckpointerCommLock	用于向checkpointer发起文件刷盘请求场景，需要串行的向请求队列插入请求结构。
TwoPhaseStateLock	用于避免并发访问或者修改两阶段信息共享数组。
TablespaceCreateLock	用于确定tablespace是否已经存在。
BtreeVacuumLock	用于防止vacuum清理B-tree中还在使用的页面。
AutovacuumLock	用于串行化访问autovacuum worker数组。
AutovacuumScheduleLock	用于串行化分配需要vacuum的table。
AutoanalyzeLock	用于获取和释放允许执行Autoanalyze的任务资源。
SyncScanLock	用于确定heap扫描时某个relfilenode的起始位置。
NodeTableLock	用于保护存放数据库节点信息的共享结构。
PoolerLock	用于保证两个线程不会同时从连接池里取到相同的连接。
RelationMappingLock	用于等待更新系统表到存储位置之间映射的文件。
Async Ctl	用于保护Async buffer。
AsyncCtlLock	用于避免并发访问或者修改共享通知状态。
AsyncQueueLock	用于避免并发访问或者修改共享通知信息队列。
SerializableXactHashLock	用于避免对于可串行事务共享结构的写写并发和读写并发。

wait_event类型	类型描述
SerializableFinishedListLock	用于避免对于已完成可串行事务共享链表的写写并发和读写并发。
SerializablePredicateLockListLock	用于保护对于可串行事务持有的锁链表。
OldSerXidLock	用于保护记录冲突可串行事务的结构。
FileStatLock	用于保护存储统计文件信息的数据结构。
SyncRepLock	用于在主备复制时保护xlog同步信息。
DataSyncRepLock	用于在主备复制时保护数据页同步信息。
MetaCacheSweepLock	用于元数据循环淘汰。
ExtensionConnectorLibLock	用于初始化ODBC连接场景，在加载与卸载特定动态库时进行加锁。
SearchServerLibLock	用于GPU加速场景初始化加载特定动态库时，对读文件操作进行加锁。
LsnXlogChkFileLock	用于串行更新特定结构中记录的主备机的xlog flush位置点。
ReplicationSlotAllocationLock	用于主备复制时保护主机端的流复制槽的分配。
ReplicationSlotControlLock	用于主备复制时避免并发更新流复制槽状态。
ResourcePoolHashLock	用于避免并发访问或者修改资源池哈希表。
WorkloadStatHashLock	用于避免并发访问或者修改包含数据库主节点的SQL请求构成的哈希表。
WorkloadIoStatHashLock	用于避免并发访问或者修改用于统计当前数据库节点的IO信息的哈希表。
WorkloadCGroupHashLock	用于避免并发访问或者修改Cgroup信息构成的哈希表。
OBSGetPathLock	用于避免对obs路径的写写并发和读写并发。
WorkloadRecordLock	用于避免并发访问或修改在内存自适应管理时对数据库主节点收到请求构成的哈希表。
WorkloadIOUtilLock	用于保护记录iostat，CPU等负载信息的结构。
WorkloadNodeGroupLock	用于避免并发访问或者修改内存中的nodegroup信息构成的哈希表。
JobShmemLock	用于定时任务功能中保护定时读取的全局变量。
OBSRuntimeLock	用于获取环境变量，如GAUSSHOME。
CriticalCacheBuildLock	用于从共享或者本地缓存初始化文件中加载cache的场景。

wait_event类型	类型描述
WaitCountHashLock	用于保护用户语句计数功能场景中的共享结构。
BufMappingLock	用于保护对共享缓冲映射表的操作。
LockMgrLock	用于保护常规锁结构信息。
PredicateLockMgrLock	用于保护可串行事务锁结构信息。
OperatorRealTLock	用于避免并发访问或者修改记录算子级实时数据的全局结构。
OperatorHistLock	用于避免并发访问或者修改记录算子级历史数据的全局结构。
SessionRealTLock	用于避免并发访问或者修改记录query级实时数据的全局结构。
SessionHistLock	用于避免并发访问或者修改记录query级历史数据的全局结构。
BarrierLock	用于保证当前只有一个线程在创建Barrier。
dummyServerInfoCacheLock	用于保护缓存加速数据库连接信息的全局哈希表。
RPNumberLock	用于加速GaussDB的数据库节点对正在执行计划的任务线程的计数。
CBMParseXlogLock	Cbm解析xlog时的保护锁
RelfilenodeReuseLock	避免错误地取消已重用的列属性文件的链接。
RcvWriteLock	防止并发调用WalDataRcvWrite。
PercentileLock	用于保护全局PercentileBuffer
CSNBufMappingLock	保护csn页面
UniqueSQLMappingLock	用于保护uniquesql hash table
DelayDDLLock	防止并发ddl。
CLOG Ctl	用于避免并发访问或者修改Clog控制数据结构
Async Ctl	保护Async buffer
MultiXactOffset Ctl	保护MultiXact offet的slru buffer
MultiXactMember Ctl	保护MultiXact member的slrubuffer
OldSerXid SLRU Ctl	保护old xids的slru buffer
ReplicationSlotLock	用于保护ReplicationSlot
PGPROCLock	用于保护pgproc
MetaCacheLock	用于保护MetaCache

wait_event类型	类型描述
DataCacheLock	用于保护datacache
InstrUserLock	用于保护InstrUserHTAB。
BadBlockStatHashLock	用于保护global_bad_block_stat hash表。
BufFreelistLock	用于保证共享缓冲区空闲列表操作的原子性。
AddinShmemInitLock	保护共享内存对象的初始化。
AlterPortLock	保护协调节点更改注册端口号的操作。
FdwPartitionCaheLock	HDFS分区表缓冲区的管理锁。
DfsConnectorCacheLock	DFSConnector缓冲区的管理锁。
DfsSpaceCacheLock	HDFS表空间管理缓冲区的管理锁。
FullBuildXlogCopyStartPtrLock	用于保护全量Build中Xlog拷贝的操作。
DfsUserLoginLock	用于HDFS用户登录以及认证。
LogicalReplicationSlotPersistentDataLock	用于保护逻辑复制过程中复制槽位的数据。
PgfdwLock	用于管理实例向Foreign server建立连接。
InstanceTimeLock	用于获取实例中会话的时间信息。
XlogRemoveSegLock	保护Xlog段文件的回收操作。
DnUsedSpaceHashLock	用于更新会话对应的空间使用信息。
CsnMinLock	用于计算CSNmin。
GPCCommitLock	用于保护全局Plan Cache hash表的添加操作。
GPCClearLock	用于保护全局Plan Cache hash表的清除操作。
GPCTimelineLock	用于保护全局Plan Cache hash表检查Timeline的操作。
InstanceRealTLock	用于保护共享实例统计信息hash表的更新操作。
CLogBufMappingLock	用于提交日志缓存管理。
GPCMappingLock	用于全局Plan Cache缓存管理。
GPCPrepareMappingLock	用于全局Plan Cache缓存管理。
BufferIOLock	保护共享缓冲区页面的IO操作。
BufferContentLock	保护共享缓冲区页面内容的读取、修改。
CSNLOG Ctl	用于CSN日志管理。
DoubleWriteLock	用于双写的管理操作。
RowPageReplicationLock	用于管理行存储的数据页复制。

wait_event类型	类型描述
MatviewSeqnoLock	用于物化视图缓存管理。
GPRCMappingLock	用于管理自治事务全局缓存hash表的访问和修改操作。
extension	其他轻量锁。
wait active statement	等待作业执行，正在资源负载管控中。
wait memory	等待内存获取。
IOStatLock	用于资源管理IO统计信息哈希表并发维护操作。
StartBlockMappingLock	用于globalstat从pgstat获取startblockarray等信息
PldebugLock	用于存储过程调试并发维护操作
DataFileIdCacheLock	管理共享内存中存储数据文件描述符的哈希表的并发访存。
GTMHostInfoLock	保护共享GTM主机信息的并发访存。
TwoPhaseStatePartLock	保护（各个分区）两阶段事务状态信息。
WALBufMappingLock	保护共享内中各个wal缓存页面与lsn偏移的映射关系。
UndoZoneLock	保护undozone的并发访存。
RollbackReqHashLock	管理共享内存中存储回滚请求信息的哈希表的并发访存。
UHeapStatLock	保护ustore统计信息的并发访存。
WALWritePaxosLock	保护向paxos复制组件写wal日志的并发顺序。
SyncPaxosLock	保护paxos同步队列的并发访存。
BackgroundWorkerLock	保护background worker的并发顺序。
HadrSwitchoverLock	保护容灾切换的并发顺序。
HashUidLock	保护uid分配的并发顺序。
ParallelDecodeLock	保护并行解码的并发顺序。
XLogMaxCSNLock	保护容灾模式下本地最大可恢复CSN信息。
DisasterCacheLock	保护共享内存中容灾信息缓存的并发访存。
MaxCSNArrayLock	保护共享内存中各个分片备机CSN恢复进度信息。
RepairBadBlockStatHashLock	保护共享内存中损坏页面哈希表的并发访存。
HypoIndexLock	虚拟索引创建、删除、重置等动作中使用的轻量级锁。

wait_event类型	类型描述
XGBoostLibLock	DB4AI特性调用xgboost库时启用的锁。
DropArchiveSlotLock	保护删除归档槽信息的并发顺序
ProcXactMappingLock	保护事务号到线程信息映射哈希表的并发访存。
UndoPerZoneLock	保护每个undozone内信息的并发访存。
UndoSpaceLock	保护undospace的并发访存。
SnapshotBlockLock	控制快照备份与页面刷盘的并发顺序。
DWSingleFlushFirstLock	控制页式单页面双写文件的并发顺序。
RestartPointQueueLock	控制备机restart Point数组的并发访存。
UnlinkRelHashTblLock	保护共享内存中待删除文件哈希表的并发访存。
UnlinkRelForkHashTblLock	保护共享内存中待删除文件fork哈希表的并发访存。
WALFlushWait	保护日志刷盘的并发顺序。
WALConsensusWait	保护日志达成一致才进行事务提交或日志回放操作。
WALBufferInitWait	保护wal共享内存页面的初始化和刷盘顺序。
WALInitSegment	保护wal日志段文件的初始化顺序。
PgwrSyncQueueLock	保护待刷盘文件队列的并发访存。
BarrierHashTblLock	保护共享内存中barrier列表信息的并发访存。
PageRepairHashTblLock	保护页面修复哈希表的并发访存。
FileRepairHashTblLock	保护文件修复哈希表的并发访存。
InstrUserLockId	对保护用户登录推出哈希表并发修改加锁。
GsStackLock	控制gs_tack函数不会被并发调用。
InstrStmtTrackCtlLock	在动态开启全量SQL时，保护哈希表的并发访存。
CaptureViewFileHashLock	开启性能视图采集时，保护哈希表的并发访存。
UniqueSqlEvictLock	开启Unique SQL回收时，保护哈希表的并发访存。
gtt_shared_ctl	用于保护全局临时表共享哈希表并发读写。
AuditIndexFileLock	控制审计日志index文件的并发读写。
LWTRANCHE_ACCOUNT_TABLE	控制账户锁定状态hash表的并发读写。

当wait_status值为wait io时对应的wait_event等待事件类型与描述信息如下。

表 12-353 IO 等待事件列表

wait_event类型	类型描述
BufFileRead	从临时文件中读取数据到指定buffer。
BufFileWrite	向临时文件中写入指定buffer中的内容。
ControlFileRead	读取pg_control文件。主要在数据库启动、执行checkpoint和主备校验过程中发生。
ControlFileSync	将pg_control文件持久化到磁盘。数据库初始化时发生。
ControlFileSyncUpdate	将pg_control文件持久化到磁盘。主要在数据库启动、执行checkpoint和主备校验过程中发生。
ControlFileWrite	写入pg_control文件。数据库初始化时发生。
ControlFileWriteUpdate	更新pg_control文件。主要在数据库启动、执行checkpoint和主备校验过程中发生。
CopyFileRead	copy文件时读取文件内容。
CopyFileWrite	copy文件时写入文件内容。
DataFileExtend	扩展文件时向文件写入内容。
DataFileFlush	将表数据文件持久化到磁盘
DataFileImmediateSync	将表数据文件立即持久化到磁盘。
DataFilePrefetch	异步读取表数据文件。
DataFileRead	同步读取表数据文件。
DataFileSync	将表数据文件的修改持久化到磁盘。
DataFileTruncate	表数据文件truncate。
DataFileWrite	向表数据文件写入内容。
LockFileAddToDataDirRead	读取“postmaster.pid”文件。
LockFileAddToDataDirSync	将“postmaster.pid”内容持久化到磁盘。
LockFileAddToDataDirWrite	将pid信息写到“postmaster.pid”文件。
LockFileCreateRead	读取LockFile文件“%s.lock”。
LockFileCreateSync	将LockFile文件“%s.lock”内容持久化到磁盘。
LockFileCreateWRITE	将pid信息写到LockFile文件“%s.lock”。
RelationMapRead	读取系统表到存储位置之间的映射文件
RelationMapSync	将系统表到存储位置之间的映射文件持久化到磁盘。

wait_event类型	类型描述
RelationMapWrite	写入系统表到存储位置之间的映射文件。
ReplicationSlotRead	读取流复制槽文件。重新启动时发生。
ReplicationSlotRestoreSync	将流复制槽文件持久化到磁盘。重新启动时发生。
ReplicationSlotSync	checkpoint时将流复制槽临时文件持久化到磁盘。
ReplicationSlotWrite	checkpoint时写流复制槽临时文件。
SLRUFlushSync	将pg_clog、pg_subtrans和pg_multixact文件持久化到磁盘。主要在执行checkpoint和数据库停机时发生。
SLRURead	读取pg_clog、pg_subtrans和pg_multixact文件。
SLRUSync	将脏页写入文件pg_clog、pg_subtrans和pg_multixact并持久化到磁盘。主要在执行checkpoint和数据库停机时发生。
SLRUWrite	写入pg_clog、pg_subtrans和pg_multixact文件。
TimelineHistoryRead	读取timeline history文件。在数据库启动时发生。
TimelineHistorySync	将timeline history文件持久化到磁盘。在数据库启动时发生。
TimelineHistoryWrite	写入timeline history文件。在数据库启动时发生。
TwophaseFileRead	读取pg_twophase文件。在两阶段事务提交、两阶段事务恢复时发生。
TwophaseFileSync	将pg_twophase文件持久化到磁盘。在两阶段事务提交、两阶段事务恢复时发生。
TwophaseFileWrite	写入pg_twophase文件。在两阶段事务提交、两阶段事务恢复时发生。
WALBootstrapSync	将初始化的WAL文件持久化到磁盘。在数据库初始化发生。
WALBootstrapWrite	写入初始化的WAL文件。在数据库初始化发生。
WALCopyRead	读取已存在的WAL文件并进行复制时产生的读操作。在执行归档恢复完后发生。
WALCopySync	将复制的WAL文件持久化到磁盘。在执行归档恢复完后发生。
WALCopyWrite	读取已存在WAL文件并进行复制时产生的写操作。在执行归档恢复完后发生。
WALInitSync	将新初始化的WAL文件持久化磁盘。在日志回收或写日志时发生。
WALInitWrite	将新创建的WAL文件初始化为0。在日志回收或写日志时发生。

wait_event类型	类型描述
WALRead	从xlog日志读取数据。两阶段文件redo相关的操作产生。
WALSyncMethodAssign	将当前打开的所有WAL文件持久化到磁盘。
WALWrite	写入WAL文件。
WALBufferAccess	WAL Buffer访问（出于性能考虑，内核代码里只统计访问次数，未统计其访问耗时）。
WALBufferFull	WAL Buffer满时，写wal文件相关的处理。
DoubleWriteFileRead	双写 文件读取。
DoubleWriteFileSync	双写 文件强制刷盘。
DoubleWriteFileWrite	双写 文件写入。
PredoProcessPending	并行日志回放中当前记录回放等待其它记录回放完成。
PredoApply	并行日志回放中等待当前工作线程等待其他线程回放至本线程LSN。
DisableConnectFileRead	HA锁分片逻辑文件读取。
DisableConnectFileSync	HA锁分片逻辑文件强制刷盘。
DisableConnectFileWrite	HA锁分片逻辑文件写入。
BufHashTableSearch	共享缓冲区hash表搜索（可能会触发页面淘汰）。
buffer_strategy_get	策略化缓冲区页面获取（可能会触发页面淘汰）。
UndoFileExtend	undo文件扩展。
UndoFilePrefetch	undo文件预取。
UndoFileRead	undo文件读取。
UndoFileWrite	undo文件写。
UndoFileSync	undo文件刷盘。
UndoFileUnlink	undo文件删除。
UndoMetaSync	undo元数据文件刷盘。
DWSingleFlushGetPos	单页面双写文件查找可用位置。
DWSingleFlushWrite	单页面双写文件刷盘。
MPFL_INIT	初始化max_page_flush_lsn。
MPFL_READ	读取max_page_flush_lsn。
MPFL_WRITE	写max_page_flush_lsn。
OBSList	遍历OBS目录。

wait_event类型	类型描述
OBSRead	读取OBS对象。
OBSWrite	写入OBS对象。
LOGCTRL_SLEEP	等待备机回放追赶。
ShareStorageWalRead	共享盘读取日志文件。
ShareStorageWalWrite	共享盘写入日志文件。
ShareStorageCtlInfoRead	共享盘读取控制信息。
ShareStorageCtlInfoWrite	共享盘写入控制信息。

当wait_status值为acquire lock（事务锁）时对应的wait_event等待事件类型与描述信息如下。

表 12-354 事务锁等待事件列表

wait_event类型	类型描述
relation	对表加锁。
extend	对表扩展空间时加锁。
partition	对分区表加锁。
partition_seq	对分区表的分区加锁。
page	对表页面加锁。
tuple	对页面上的tuple加锁。
transactionid	对事务ID加锁。
virtualxid	对虚拟事务ID加锁。
object	加对象锁。
userlock	加用户锁。
advisory	加advisory锁。
filenode	对文件名加锁，控制文件级操作的并发顺序。
subtransactionid	对子事务号加锁。
tuple_uid	对元组uid隐藏字段加锁。

12.3.250 PG_TIMEZONE_ABBREVS

PG_TIMEZONE_ABBREVS视图显示所有可用时区的信息。

表 12-355 PG_TIMEZONE_ABBREVS 字段

名称	类型	描述
abbrev	text	时区名缩写。
utc_offset	interval	相对于UTC的偏移量。
is_dst	boolean	如果当前正实行夏令时则为TRUE，否则为FALSE。

12.3.251 PG_TIMEZONE_NAMES

PG_TIMEZONE_NAMES视图显示所有能够被SET TIMEZONE语法识别的时区名及其缩写、UTC偏移量、是否实行夏令时。

表 12-356 PG_TIMEZONE_NAMES 字段

名称	类型	描述
name	text	时区名。
abbrev	text	时区名缩写。
utc_offset	interval	相对于UTC的偏移量。
is_dst	boolean	如果当前正实行夏令时则为TRUE，否则为FALSE。

12.3.252 PG_TOTAL_MEMORY_DETAIL

PG_TOTAL_MEMORY_DETAIL视图显示某个数据库节点内存使用情况。

表 12-357 PG_TOTAL_MEMORY_DETAIL 字段

名称	类型	描述
nodename	text	节点名称。
memorytype	text	内存的名称。
memorymbytes	integer	内存使用的大小，单位为MB。

12.3.253 PG_TOTAL_USER_RESOURCE_INFO

PG_TOTAL_USER_RESOURCE_INFO视图显示所有用户资源使用情况，需要使用管理员用户进行查询。此视图在GUC参数use_workload_manager为on时才有效。其中，IO相关监控项在参数enable_logical_io_statistics为on时才有效。

表 12-358 PG_TOTAL_USER_RESOURCE_INFO 字段

名称	类型	描述
username	name	用户名。
used_memory	integer	正在使用的内存大小，单位MB。
total_memory	integer	可以使用的内存大小，单位MB。值为0表示未限制最大可用内存，其限制取决于数据库最大可用内存。
used_cpu	double precision	正在使用的CPU核数（仅统计复杂作业CPU使用情况，且该值为相关控制组的CPU使用统计值）。
total_cpu	integer	在该机器节点上，用户关联控制组的CPU核数总和。
used_space	bigint	已使用的永久表存储空间大小，单位kB。
total_space	bigint	可使用的永久表存储空间大小，单位kB，值为-1表示未限制最大存储空间。
used_temp_space	bigint	已使用的临时空间大小，单位kB。
total_temp_space	bigint	可使用的临时空间总大小，单位kB，值为-1表示未限制。
used_spill_space	bigint	已使用的算子落盘空间大小，单位kB。
total_spill_space	bigint	可使用的算子落盘空间总大小，单位kB，值为-1表示未限制。
read_kbytes	bigint	数据库主节点：过去5秒内，该用户在数据库节点上复杂作业read的字节总数（单位kB）。 数据库节点：实例启动至当前时间为止，该用户复杂作业read的字节总数（单位kB）。
write_kbytes	bigint	数据库主节点：过去5秒内，该用户在数据库节点上复杂作业write的字节总数（单位kB）。 数据库节点：实例启动至当前时间为止，该用户复杂作业write的字节总数（单位kB）。
read_counts	bigint	数据库主节点：过去5秒内，该用户在数据库节点上复杂作业read的次数之和（单位次）。 数据库节点：实例启动至当前时间为止，该用户复杂作业read的次数之和（单位次）。
write_counts	bigint	数据库主节点：过去5秒内，该用户在数据库节点上复杂作业write的次数之和（单位次）。 数据库节点：实例启动至当前时间为止，该用户复杂作业write的次数之和（单位次）。

名称	类型	描述
read_speed	double precision	数据库主节点：过去5秒内，该用户在单个数据库节点上复杂作业read平均速率（单位kB/s）。 数据库节点：过去5秒内，该用户在该数据库节点上复杂作业read平均速率（单位kB/s）。
write_speed	double precision	数据库主节点：过去5秒内，该用户在单个数据库节点上复杂作业write平均速率（单位kB/s）。 数据库节点：过去5秒内，该用户在该数据库节点上复杂作业write平均速率（单位kB/s）。

12.3.254 PG_TOTAL_USER_RESOURCE_INFO_OID

PG_TOTAL_USER_RESOURCE_INFO_OID视图显示所有用户资源使用情况，需要使用管理员用户进行查询。此视图在GUC参数use_workload_manager为on时才有效。

表 12-359 PG_TOTAL_USER_RESOURCE_INFO_OID 字段

名称	类型	描述
userid	oid	用户ID。
used_memory	integer	正在使用的内存大小，单位MB。
total_memory	integer	可以使用的内存大小，单位MB。值为0表示未限制最大可用内存，其限制取决于数据库最大可用内存。
used_cpu	double precision	正在使用的CPU核数。
total_cpu	integer	在该机器节点上，用户关联控制组的CPU核数总和。
used_space	bigint	已使用的存储空间大小，单位kB。
total_space	bigint	可使用的存储空间大小，单位kB，值为-1表示未限制最大存储空间。
used_temp_space	bigint	已使用的临时空间大小，单位kB
total_temp_space	bigint	可使用的临时空间总大小，单位kB，值为-1表示未限制。
used_spill_space	bigint	已使用的下盘空间大小。单位kB。
total_spill_space	bigint	可使用的下盘空间总大小，单位kB，值为-1表示未限制。
read_kbytes	bigint	读磁盘数据量，单位kB。

名称	类型	描述
write_kbytes	bigint	写磁盘数据量，单位kB。
read_counts	bigint	读磁盘次数。
write_counts	bigint	写磁盘次数。
read_speed	double precision	读磁盘速率，单位B/ms。
write_speed	double precision	写磁盘速率，单位B/ms。

12.3.255 PG_USER

PG_USER视图显示数据库用户的信息，默认只有初始化用户和具有sysadmin属性的用户可以查看，其余用户需要赋权后才可以查看。

表 12-360 PG_USER 字段

名称	类型	描述
usename	name	用户名。
usesysid	oid	用户的ID。
usecreatedb	boolean	用户是否可以创建数据库。 ● t（true）：表示是。 ● f（false）：表示否。
usesuper	boolean	用户是否是拥有最高权限的初始系统管理员。 ● t（true）：表示是。 ● f（false）：表示否。
usecatupd	boolean	用户是否可以直接更新系统表。只有usesysid=10的初始系统管理员拥有此权限。其他用户无法获得此权限。 ● t（true）：表示是。 ● f（false）：表示否。
userepl	boolean	用户是否可以复制数据流。 ● t（true）：表示是。 ● f（false）：表示否。
passwd	text	密文存储后的用户密码，始终为********。
valbegin	timestamp with time zone	账户的有效开始时间。如果没有设置有效开始时间，则为NULL。

名称	类型	描述
valuntil	timestamp with time zone	账户的有效结束时间。如果没有设置有效结束时间，则为NULL。
respool	name	用户所在的资源池。
parent	oid	父用户OID。
spacelimit	text	永久表的存储空间限额，单位KB。
useconfig	text[]	运行时配置项的默认值。
tempspacelimit	text	临时表的存储空间限额，单位KB。
spillspacelimit	text	算子的落盘空间限额，单位KB。
nodegroup	name	用户关联的逻辑数据库名称，如果该用户没有管理逻辑数据库，则该字段为空。
usemonitoradmin	boolean	用户是否是监控管理员。 • t（true）：表示是。 • f（false）：表示否。
useoperatoradmin	boolean	用户是否是运维管理员。 • t（true）：表示是。 • f（false）：表示否。
usepolicyadmin	boolean	用户是否是安全策略管理员。 • t（true）：表示是。 • f（false）：表示否。

12.3.256 PG_USER_MAPPINGS

PG_USER_MAPPINGS视图显示用户映射的信息。

该视图只是系统表PG_USER_MAPPING可读部分的视图化表现，如果用户无权使用系统表且查询该视图时，有些选项字段会显示为空。普通用户需要授权才可以访问。

表 12-361 PG_USER_MAPPINGS 字段

名称	类型	引用	描述
umid	oid	PG_USER_MAPPING.oid	用户映射的OID。
srvid	oid	PG_FOREIGN_SERVER.oid	包含这个映射的外部服务器的OID。
srvname	name	PG_FOREIGN_SERVER.srvname	外部服务器的名称。

名称	类型	引用	描述
umuser	oid	PG_AUTHID.oid	被映射的本地角色的OID，如果用户映射是公共的则为0。
usename	name	-	被映射的本地用户的名称。
umoptions	text[]	-	如果当前用户是外部服务器的所有者，则为用户映射指定选项，使用“keyword=value”字符串，否则为NULL。

12.3.257 PG_VARIABLE_INFO

PG_VARIABLE_INFO视图用于查询数据库中当前节点的xid、oid的状态。

表 12-362 PG_VARIABLE_INFO 字段

名称	类型	描述
node_name	text	节点名称。
next_oid	oid	该节点下一次生成的oid。
next_xid	xid	该节点下一次生成的事务号。
oldest_xid	xid	该节点最旧的事务号。
xid_vac_limit	xid	强制autovacuum的临界点。
oldest_xid_db	oid	该节点datafrozenxid最小的数据库oid。
last_extend_csn_logpage	xid	最后一次扩展csnlog的页面号。
start_extend_csn_logpage	xid	csnlog扩展的起始页面号。
next_commit_seqno	xid	该节点下次生成的csn号。
latest_completed_xid	xid	该节点提交或者回滚后节点上的最新事务号。
startup_max_xid	xid	该节点关机前的最后一个事务号。

12.3.258 PG_VIEWS

PG_VIEWS视图显示数据库中每个视图的有用信息。

表 12-363 PG_VIEWS 字段

名称	类型	引用	描述
schemaname	name	PG_NAMESPACE.nspname	视图的模式名。
viewname	name	PG_CLASS.relname	视图名。
viewowner	name	PG_AUTHID.Erolname	视图的所有者。
definition	text	-	视图的定义。

12.3.259 PGXC_PREPARED_XACTS

PGXC_PREPARED_XACTS视图显示当前处于prepared阶段的两阶段事务。只有system admin和monitor admin用户有权限查看。

表 12-364 PGXC_PREPARED_XACTS 字段

名称	类型	描述
pgxc_prepared_xact	text	当前处于prepared阶段的两阶段事务。

12.3.260 PGXC_THREAD_WAIT_STATUS

集中式不支持该视图。

12.3.261 PLAN_TABLE

PLAN_TABLE显示用户通过执行EXPLAIN PLAN收集到的计划信息。计划信息的生命周期是session级别，session退出后相应的数据将被清除。同时不同session和不同user间的数据是相互隔离的。该视图同时存在PG_CATALOG和SYS schema下。

表 12-365 PLAN_TABLE 字段

名称	类型	描述
statement_id	character varying(30)	用户输入的查询标签。
plan_id	bigint	查询标识。
id	integer	查询生成的计划中的每一个执行算子的编号。
operation	character varying(30)	计划中算子的操作描述。
options	character varying(255)	操作选项。

名称	类型	描述
object_name	name	操作对应的对象名，非查询中使用到的对象别名。来自于用户定义。
object_type	character varying(30)	对象类型。
object_owner	name	对象所属schema，来自于用户定义。
projection	character varying(4000)	操作输出的列信息。
cost	double precision	优化器对算子估算的执行代价。
cardinality	double precision	优化器对算子估算的结果行数。
remarks	character varying(4000)	暂不支持，值为NULL。
timestamp	timestamp(0) without time zone	暂不支持，值为NULL。
object_node	character varying(128)	暂不支持，值为NULL。
object_alias	character varying(261)	暂不支持，值为NULL。
object_instance	numeric	暂不支持，值为NULL。
optimizer	character varying(255)	暂不支持，值为NULL。
search_columns	numeric	暂不支持，值为NULL。
parent_id	numeric	暂不支持，值为NULL。
depth	numeric	暂不支持，值为NULL。
position	numeric	暂不支持，值为NULL。
bytes	numeric	暂不支持，值为NULL。
other_tag	character varying(255)	暂不支持，值为NULL。
partition_start	character varying(255)	暂不支持，值为NULL。
partition_stop	character varying(255)	暂不支持，值为NULL。
partition_id	numeric	暂不支持，值为NULL。
other	character varying	暂不支持，值为NULL。
other_xml	clob	暂不支持，值为NULL。

名称	类型	描述
distribution	character varying(20)	暂不支持，值为NULL。
cpu_cost	numeric	暂不支持，值为NULL。
io_cost	numeric	暂不支持，值为NULL。
temp_space	numeric	暂不支持，值为NULL。
access_predicates	character varying(4000)	暂不支持，值为NULL。
filter_predicates	character varying(4000)	暂不支持，值为NULL。
time	numeric	暂不支持，值为NULL。
qblock_name	character varying(128)	暂不支持，值为NULL。

说明

- object_type取值范围为PG_CLASS中定义的relkind类型（TABLE普通表，INDEX索引，SEQUENCE序列，VIEW视图，COMPOSITE TYPE复合类型，TOASTVALUE TOAST表）和计划使用到的rtekind(SUBQUERY, JOIN, FUNCTION, VALUES, CTE, REMOTE_QUERY)。
- object_owner对于RTE来说是计划中使用的对象描述，非用户定义的类型不存在object_owner。
- statement_id、object_name、object_owner、projection字段内容遵循用户定义的大小写存储，其它字段内容采用大写存储。
- 支持用户对PLAN_TABLE进行SELECT和DELETE操作，不支持其它DML操作。

12.3.262 ROLE_ROLE_PRIVS

ROLE_ROLE_PRIVS视图显示授予其他角色的角色，仅提供用户有权访问的角色的信息。默认所有用户都可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-366 ROLE_ROLE_PRIVS 字段

名称	类型	描述
role	character varying(128)	角色名称。
granted_role	character varying(128)	被授予的角色。
admin_option	character varying(3)	该授权是否包含ADMIN选项。 • YES：包含ADMIN选项。 • NO：不包含ADMIN选项。

名称	类型	描述
common	character varying(3)	暂不支持，值为NULL。
inherited	character varying(3)	暂不支持，值为NULL。

12.3.263 ROLE_SYS_PRIVS

ROLE_SYS_PRIVS视图显示授予角色的系统特权信息，仅提供用户有权访问的角色的信息。默认所有用户都可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-367 ROLE_SYS_PRIVS 字段

名称	类型	描述
role	character varying(128)	角色名称。
privilege	character varying(40)	授予角色的系统权限。 系统权限包括rolsuper、rolinherit、rolcreaterole、rolcreatedb、rolcatupdate、rolcanlogin、rolreplication、rolauditadmin、rolsystemadmin、roluseft、rolmonitoradmin、roloperatoradmin、rolpolicyadmin。
admin_option	character varying(3)	表示该授权是否包含ADMIN选项。 ● YES：表示包含ADMIN选项。 ● NO：表示不包含ADMIN选项。
common	character varying(3)	暂不支持，值为NULL。
inherited	character varying(3)	暂不支持，值为NULL。

12.3.264 ROLE_TAB_PRIVS

ROLE_TAB_PRIVS视图显示授予角色的对象权限信息，仅提供用户有权访问的角色的信息。默认所有用户都可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-368 ROLE_TAB_PRIVS 字段

名称	类型	描述
role	character varying(128)	角色名称。

名称	类型	描述
owner	character varying(128)	对象的所有者。
table_name	character varying(128)	对象的名称。对象类型包括表、包、索引、序列等等。
column_name	character varying(128)	暂不支持，值为NULL。
privilege	character varying(40)	对象上的权限，包括USAGE、UPDATE、DELETE、INSERT、CONNECT、SELECT、EXECUTE。
grantable	character varying(3)	表示该授权是否包含GRANT选项。 ● YES：表示包含GRANT选项。 ● NO：表示不包含GRANT选项。
common	character varying(3)	暂不支持，值为NULL。
inherited	character varying(3)	暂不支持，值为NULL。

12.3.265 SYS_DUMMY

SYS_DUMMY视图是数据库根据数据字典自动创建的，它只有一个文本字段，且只有一行，用于保存表达式计算结果。任何用户都可以访问它。该视图同时存在于PG_CATALOG和SYS schema下。

表 12-369 SYS_DUMMY 字段

名称	类型	描述
dummy	text	表达式计算结果。

12.3.266 V_INSTANCE

V_INSTANCE视图显示当前数据库的实例信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-370 V_INSTANCE 字段

名称	类型	描述
instance_number	oid	当前数据库oid。
instance_name	character varying(16)	当前数据库名。

名称	类型	描述
host_name	character varying(64)	主机名。
version	character varying(17)	暂不支持，值为NULL。
version_legacy	character varying(17)	暂不支持，值为NULL。
version_full	character varying(17)	暂不支持，值为NULL。
startup_time	timestamp(0) without time zone	暂不支持，值为NULL。
status	character varying(12)	暂不支持，值为NULL。
parallel	character varying(3)	暂不支持，值为NULL。
thread#	numeric	暂不支持，值为NULL。
archiver	character varying(7)	暂不支持，值为NULL。
log_switch_wait	character varying(15)	暂不支持，值为NULL。
logins	character varying(10)	暂不支持，值为NULL。
shutdown_pending	character varying(3)	暂不支持，值为NULL。
database_status	character varying(17)	暂不支持，值为NULL。
instance_role	character varying(18)	暂不支持，值为NULL。
active_state	character varying(9)	暂不支持，值为NULL。
blocked	character varying(3)	暂不支持，值为NULL。
con_id	numeric	暂不支持，值为NULL。
instance_mode	character varying(11)	暂不支持，值为NULL。
edition	character varying(7)	暂不支持，值为NULL。
family	character varying(80)	暂不支持，值为NULL。
database_type	character varying(15)	暂不支持，值为NULL。

12.3.267 V_MYSTAT

V_MYSTAT视图显示数据库所有会话的统计信息。该视图只有系统管理员可以访问，普通用户需要授权才能访问，该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-371 V_MYSTAT 字段

名称	类型	描述
sid	numeric	当前会话的ID。
statistic#	numeric	暂不支持，值为NULL。

名称	类型	描述
value	numeric	暂不支持，值为NULL。
con_id	numeric	暂不支持，值为NULL。

12.3.268 V_SESSION

V_SESSION视图显示当前所有会话的信息，该视图只有系统管理员可以访问，普通用户需要授权才能访问，该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-372 V_SESSION 字段

名称	类型	描述
saddr	raw	暂不支持，值为NULL。
sid	bigint	会话ID。
serial#	integer	当前活动的后台线程的序号，在GaussDB中为0。
audsid	numeric	暂不支持，值为NULL。
paddr	raw	暂不支持，值为NULL。
schema#	numeric	暂不支持，值为NULL。
schemaname	name	登录该后台的用户名。
user#	oid	登录此后台线程的用户的OID。oid 为0表示此后台线程为全局辅助线程（auxiliary）。
username	name	登录此后台线程的用户名。username为空表示此后台线程为全局辅助线程（auxiliary）。
command	numeric	暂不支持，值为NULL。
ownerid	numeric	暂不支持，值为NULL。
taddr	character varying(16)	暂不支持，值为NULL。
lockwait	character varying(16)	暂不支持，值为NULL。
machine	text	客户端的主机名，这个字段是通过client_addr的反向DNS查找得到。这个字段只有在启动log_hostname且使用IP连接时才非空。
sql_id	bigint	查询语句的ID。
client_info	text	客户端信息。

名称	类型	描述
event	text	语句当前排队状态。可能值是： ● waiting in queue：表示语句在排队中。 ● 空：表示语句正在运行。
sql_exec_start	timestamp with time zone	开始当前活跃查询的时间，如果state的值不是active，则这个值是上一个查询的开始时间。
program	text	连接到该后台的应用名。
status	text	该后台当前总体状态。可能值是： ● active：后台正在执行一个查询。 ● idle：后台正在等待一个新的客户端命令。 ● idle in transaction：后台在事务中，但事务中没有语句在执行。 ● idle in transaction (aborted)：后台在事务中，但事务中有语句执行失败。 ● fastpath function call：后台正在执行一个fast-path函数。 ● disabled：如果后台禁用track_activities，则报告这个状态。
server	character varying(9)	暂不支持，值为NULL。
pdml_status	character varying(8)	当前会话是否启用DML的并行执行。
port	numeric	当前会话的端口号。
process	character varying(24)	当前会话的进程号。
logon_time	timestamp(0) without time zone	当前会话的登录时间。
last_call_et	integer	当前会话上次状态发生改变的时长。
osuser	character varying(128)	暂不支持，值为NULL。
terminal	character varying(30）	暂不支持，值为NULL。
type	character varying(10)	暂不支持，值为NULL。
sql_address	raw	暂不支持，值为NULL。
sql_hash_value	numeric	暂不支持，值为NULL。

名称	类型	描述
sql_child_number	numeric	暂不支持，值为NULL。
sql_exec_id	numeric	暂不支持，值为NULL。
prev_sql_addr	raw	暂不支持，值为NULL。
prev_hash_value	numeric	暂不支持，值为NULL。
prev_sql_id	character varying(13)	暂不支持，值为NULL。
prev_child_number	numeric	暂不支持，值为NULL。
prev_exec_start	timestamp(0) without time zone	暂不支持，值为NULL。
prev_exec_id	numeric	暂不支持，值为NULL。
plsql_entry_object_id	numeric	暂不支持，值为NULL。
plsql_entry_subprogram_id	numeric	暂不支持，值为NULL。
plsql_object_id	numeric	暂不支持，值为NULL。
plsql_subprogram_id	numeric	暂不支持，值为NULL。
module	character varying(64)	暂不支持，值为NULL。
module_hash	numeric	暂不支持，值为NULL。
action	character varying(64)	暂不支持，值为NULL。
action_hash	numeric	暂不支持，值为NULL。
fixed_table_sequence	numeric	暂不支持，值为NULL。
row_wait_obj#	numeric	暂不支持，值为NULL。
row_wait_file#	numeric	暂不支持，值为NULL。

名称	类型	描述
row_wait_block#	numeric	暂不支持，值为NULL。
row_wait_row#	numeric	暂不支持，值为NULL。
top_level_call#	numeric	暂不支持，值为NULL。
pdml_enabled	character varying(3)	暂不支持，值为NULL。
failover_type	character varying(13)	暂不支持，值为NULL。
failover_method	character varying(10)	暂不支持，值为NULL。
failed_over	character varying(3)	暂不支持，值为NULL。
resource_consumer_group	character varying(32)	暂不支持，值为NULL。
pddl_status	character varying(8)	暂不支持，值为NULL。
pq_status	character varying(8)	暂不支持，值为NULL。
current_queue_duration	numeric	暂不支持，值为NULL。
client_identifier	character varying(64)	暂不支持，值为NULL。
blocking_session_status	character varying(11)	暂不支持，值为NULL。
blocking_instance	numeric	暂不支持，值为NULL。
blocking_session	numeric	暂不支持，值为NULL。
final_blocking_session_status	character varying(11)	暂不支持，值为NULL。
final_blocking_instance	numeric	暂不支持，值为NULL。
final_blocking_session	numeric	暂不支持，值为NULL。

名称	类型	描述
seq#	numeric	暂不支持，值为NULL。
event#	numeric	暂不支持，值为NULL。
p1text	character varying(64)	暂不支持，值为NULL。
p1	numeric	暂不支持，值为NULL。
p1raw	raw	暂不支持，值为NULL。
p2text	character varying(64)	暂不支持，值为NULL。
p2	numeric	暂不支持，值为NULL。
p2raw	raw	暂不支持，值为NULL。
p3text	character varying(64)	暂不支持，值为NULL。
p3	numeric	暂不支持，值为NULL。
p3raw	raw	暂不支持，值为NULL。
wait_class_id	numeric	暂不支持，值为NULL。
wait_class#	numeric	暂不支持，值为NULL。
wait_class	character varying(64)	暂不支持，值为NULL。
wait_time	numeric	暂不支持，值为NULL。
seconds_in_wait	numeric	暂不支持，值为NULL。
state	character varying(19)	暂不支持，值为NULL。
wait_time_micro	numeric	暂不支持，值为NULL。
time_remaining_micro	numeric	暂不支持，值为NULL。
time_since_last_wait_micro	numeric	暂不支持，值为NULL。
service_name	character varying(64)	暂不支持，值为NULL。
sql_trace	character varying(8)	暂不支持，值为NULL。

名称	类型	描述
sql_trace_waits	character varying(5)	暂不支持，值为NULL。
sql_trace_binds	character varying(5)	暂不支持，值为NULL。
sql_trace_plan_stats	character varying(10)	暂不支持，值为NULL。
session_edition_id	numeric	暂不支持，值为NULL。
creator_addr	raw	暂不支持，值为NULL。
creator_serial#	numeric	暂不支持，值为NULL。
ecid	character varying(64)	暂不支持，值为NULL。
sql_translation_profile_id	numeric	暂不支持，值为NULL。
pga_tunable_mem	numeric	暂不支持，值为NULL。
shard_ddl_status	character varying(8)	暂不支持，值为NULL。
con_id	numeric	暂不支持，值为NULL。
external_name	character varying(1024)	暂不支持，值为NULL。
plsql_debugger_connected	character varying(5)	暂不支持，值为NULL。

12.3.269 V$GLOBAL_TRANSACTION

V$GLOBAL_TRANSACTION视图显示有关当前活动全局事务的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-373 V$GLOBAL_TRANSACTION 字段

名称	类型	描述
formatid	numeric	全局事务格式标识符。暂不支持，值为NULL。
globalid	raw	全局事务标识符。

名称	类型	描述
branchid	raw	全局事务分支标识符。暂不支持，值为NULL。全局事务的每一个单独事务称为分支。
branches	numeric	全局事务分支总数。
refcount	numeric	全局事务的同级数（必须与分支相同）。
preparecount	numeric	已准备的全局事务分支数。当system_view_version参数大于0时，不存在已准备的全局事务分支时为0，否则为NULL。
state	character varying(38)	全局事务的分支的状态。
flags	numeric	状态的数字表示形式。
coupling	character varying(15)	指示分支是自由(`FREE`)、松散耦合(`LOOSELY COUPLED`)、还是紧密耦合(TIGHTLY COUPLED)。暂不支持，值为NULL。
con_id	numeric	与数据相关的容器的 ID。暂不支持，值为0。

12.3.270 V$NLS_PARAMETERS

V$NLS_PARAMETERS视图显示数据库当前配置的NLS（National Language Support）参数和参数的值。所有用户都可以访问，该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-374 V$NLS_PARAMETERS 字段

名称	类型	描述
parameter	character varying(64)	NLS（National Language Support）参数名。
value	character varying(64)	NLS（National Language Support）参数的值。
con_id	numeric	暂不支持，值为0。

12.3.271 V$SESSION_WAIT

V$SESSION_WAIT视图显示了每一个用户每一个会话的当前正在等待的事件或者最后一次等待的事件。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-375 V$SESSION_WAIT 字段

名称	类型	描述
sid	numeric	会话识别标记，映射到V$SESSION.SID字段。
seq#	numeric	暂不支持，值为NULL。
event	character varying(64)	如果会话在等待中，则显示目前在等待的资源或者事件，如果会话没有在等待，则显示最后一次等待的资源或者事件。
p1text	character varying(64)	暂不支持，值为NULL。
p1	numeric	暂不支持，值为NULL。
p1raw	raw	暂不支持，值为NULL。
p2text	character varying(64)	暂不支持，值为NULL。
p2	numeric	暂不支持，值为NULL。
p2raw	raw	暂不支持，值为NULL。
p3text	character varying(64)	暂不支持，值为NULL。
p3	numeric	暂不支持，值为NULL。
p3raw	raw	暂不支持，值为NULL。
wait_class_id	numeric	暂不支持，值为NULL。
wait_class#	numeric	暂不支持，值为NULL。
wait_class	character varying(64)	等待事件的种类命名。
wait_time	numeric	如果会话当前正在等待，则值为 0。如果会话不在等待中，则值如下所示： ● >0：值是最后一次等待的持续时间（以百分之一秒为单位）。 ● -1：最后一次等待的持续时间不到百分之一秒。 ● -2：参数TIMED_STATISTICS设置为 false。 此列已被弃用，取而代之的是列为WAIT_TIME_MICRO和STATE。

名称	类型	描述
second_in_wait	numeric	如果会话当前正在等待，则该值是等待当前等待的时间量。如果会话未处于等待状态，则该值是自上次等待开始以来的时间量。 此列已被弃用，取而代之的是WAIT_TIME_MICRO列和TIME_SINCE_LAST_WAIT_MICRO列。
state	character varying(64)	等待状态： ● WAITING：会话当前正在等待。 ● WAITED UNKNOWN TIME：最后等待的持续时间未知；这是参数TIMED_STATISTICS设置为false时的值。 ● WAITED SHORT TIME：最后一次等待不到百分之一秒。 ● WAITED KNOWN TIME：在WAIT_TIME列中指定的最后一次等待的持续时间。
wait_time_micro	numeric	等待的时间（以微秒为单位）。如果会话当前正在等待，则该值是在当前等待中花费的时间。如果会话当前未处于等待状态，则该值是上次等待的等待时间量。
time_remaining_micro	numeric	暂不支持，值为NULL。
time_since_last_wait_micro	numeric	自上次等待结束以来经过的时间（以微秒为单位）。如果会话当前处于等待状态，则值为 0。
con_id	numeric	暂不支持，值为0。

12.3.272 V$SYSSTAT

V$SYSSTAT视图显示自数据库实例启动那刻起就开始累计的全实例的资源使用情况。默认只有初始用户或监控管理员可以访问，其它用户需授予MONADMIN权限才可访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-376 V$SYSSTAT 字段

名称	类型	描述
statistic#	numeric	统计编号。
name	character varying(64)	统计名称。
class	numeric	暂不支持，值为NULL。
value	numeric	统计值。

名称	类型	描述
stat_id	numeric	暂不支持，值为NULL。
con_id	numeric	暂不支持，值为0。

12.3.273 V$SYSTEM_EVENT

V$SYSTEM_EVENT视图显示有关事件总等待的信息（自实例启动后各个等待事件的概括）。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-377 V$SYSTEM_EVENT 字段

名称	类型	描述
event	character varying(64)	等待事件的名称。
total_waits	numeric	等待事件的总次数。
total_timeouts	numeric	事件的超时总数。
time_waited	numeric	等待事件的总时间（以百分之一秒为单位）。
average_wait	numeric	等待事件的平均时间（以百分之一秒为单位）。
time_waited_micro	numeric	等待事件的总时间（以微秒为单位）。
total_waits_fg	numeric	暂不支持，值为NULL。
total_timeouts_fg	numeric	暂不支持，值为NULL。
time_waited_fg	numeric	暂不支持，值为NULL。
average_wait_fg	numeric	暂不支持，值为NULL。
time_waited_micro_fg	numeric	暂不支持，值为NULL。
event_id	numeric	暂不支持，值为NULL。
wait_class_id	numeric	暂不支持，值为NULL。
wait_class#	numeric	暂不支持，值为NULL。
wait_class	character varying(64)	等待事件的等待类名称。
con_id	numeric	暂不支持，值为0。

12.3.274 V$VERSION

V$VERSION视图显示数据库的版本号。所有用户都可以访问，该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-378 V$VERSION 字段

名称	类型	描述
banner	character varying(80)	组件名称和版本号。
banner_full	character varying(160)	显示数据库版本和版本号。
banner_legacy	character varying(80)	显示数据库版本。
con_id	numeric	暂不支持，值为0。

12.4 废弃

12.4.1 系统视图

12.4.1.1 GET_GLOBAL_PREPARED_XACTS

集中式不支持该视图。

12.4.1.2 PG_GET_INVALID_BACKENDS

不支持查询该视图，报错提示：“Un-support feature”。

13 Schema

GaussDB的Schema如下表所示。

说明

数据库禁止在提供功能接口的Schema下创建用户的业务数据，包括但不限于表、函数等（dbe_*, pkg_*）。

表 13-1 GaussDB 支持的 Schema

Schema名称	描述
db4ai	用于管理AI训练中不同版本的数据信息。
dbe_perf	DBE_PERF Schema内视图主要用来诊断性能问题，也是WDR Snapshot的数据来源。数据库安装后，默认只有初始用户和监控管理员具有模式dbe_perf的权限，有权查看该模式下的视图和函数。
dbe_pldebugger	用于调试PL/SQL函数及存储过程。
snapshot	用于管理WDR snapshot的相关的数据信息，默认初始化用户或监控管理员用户可以访问。
sqladvisor	用于分布列推荐，集中式不可用。
sys	用于提供系统信息视图接口。
pg_catalog	用于维护系统的catalog信息，包含系统表和所有内置数据类型、函数、操作符。
pg_toast	用于存储大对象（系统内部使用）。
public	公共模式，用于存储公共对象。search_path参数缺省时，如果存在用户同名的模式则将创建的表（以及其他对象）默认创建到同名模式下，不存在用户同名模式则自动放入public模式。
pkg_service	用于管理package服务相关信息。
pkg_util	用于管理package工具相关信息。

Schema名称	描述
dbe_raw	高级功能包dbe_raw，用于raw类型数据的转化、取子串、求长度等操作。
dbe_session	高级功能包dbe_session，用于设置指定属性的值（value），并支持用户查询校验。
dbe_lob	高级功能包dbe_lob，用于大文件（clob/blob）的读取、写入、复制等操作。
dbe_match	高级功能包dbe_match，用于字符串相似度的比较。
dbe_task	高级功能包dbe_task，用于作业任务的调度包括提交任务、取消任务、同步任务状态、更新任务信息等可以使数据库定期执行特定的任务。
dbe_sql	高级功能包dbe_sql，用于执行动态sql，可以在应用的运行时间构建查询和其他命令。
dbe_file	高级功能包dbe_file，用于数据库外部文件的读取、复制、写入、删除、重命名等。
dbe_output	高级功能包dbe_output，用于打印输出信息。
dbe_random	高级功能包dbe_random，用于生成随机种子和随机数。
dbe_application_info	高级功能包dbe_application_info，用于记录客户端信息。
dbe_utility	高级功能包dbe_utility，用于存储过程调用调试工具，例如打印错误堆栈等。
dbe_scheduler	高级功能包dbe_scheduler，用于创建定时任务，通过程序(program)、调度(schedule)使数据库定期执行特定的任务。也可以通过授权、提供证书执行数据库外部任务。
information_schema	用于存储有关当前数据库中定义的对象的信息。
dbe_pldeveloper	用户存储过程编译调试。
dbe_sql_util	SQL运维功能，目前包含SQL Patch的运维接口。

13.1 Information Schema

信息模式本身是一个名为information_schema的模式。这个模式自动存在于所有数据库中。信息模式由一组视图构成，它们包含定义在当前数据库中对象的信息。这个模式的拥有者是初始数据库用户，但是所有用户仅有使用权限，没有创建表、函数等对象的权限。

信息模式兼容PG，包括：constraint_table_usage、domain_constraints、domain_udt_usage、domains、enabled_roles、key_column_usage、parameters、referential_constraints、applicable_roles、administrable_role_authorizations、

attributes、character_sets、check_constraint_routine_usage、check_constraints、collations、collation_character_set_applicability、column_domain_usage、column_privileges、column_udt_usage、columns、constraint_column_usage、role_column_grants、routine_privileges、role_routine_grants、routines、schemata、sequences、table_constraints、table_privileges、role_table_grants、tables、triggered_update_columns、triggers、udt_privileges、role_udt_grants、usage_privileges、role_usage_grants、user_defined_types、view_column_usage、view_routine_usage、view_table_usage、views、data_type_privileges、element_types、column_options、foreign_data_wrapper_options、foreign_data_wrappers、foreign_server_options、foreign_servers、foreign_table_options、foreign_tables、user_mapping_options、user_mappings、sql_features、sql_implementation_info、sql_languages、sql_packages、sql_parts、sql_sizing、sql_sizing_profiles。

下面章节只显示未在上述描述内的视图信息。

13.1.1 _PG_FOREIGN_DATA_WRAPPERS

显示外部数据封装器的信息。该视图只有sysadmin权限可以查看。

表 13-2 _PG_FOREIGN_DATA_WRAPPERS 字段

名称	类型	描述
oid	oid	外部数据封装器的oid。
fdwowner	oid	外部数据封装器的所有者的oid。
fdwoptions	text[]	外部数据封装器指定选项，使用“keyword=value”格式的字符串。
foreign_data_wrapper_catalog	information_schema.sql_identifier	外部封装器所在的数据库名称（永远为当前数据库）。
foreign_data_wrapper_name	information_schema.sql_identifier	外部数据封装器名称。
authorization_identifier	information_schema.sql_identifier	外部数据封装器所有者的角色名称。
foreign_data_wrapper_language	information_schema.character_data	外部数据封装器的实现语言。

13.1.2 _PG_FOREIGN_SERVERS

显示外部服务器的信息。该视图只有sysadmin权限可以查看。

表 13-3 _PG_FOREIGN_SERVERS 字段

名称	类型	描述

oid	oid	外部服务器的oid。
srvoptions	text[]	外部服务器指定选项，使用“keyword=value”格式的字符串。
foreign_server_catalog	information_schema.sql_identifier	外部服务器所在database名称（永远为当前数据库）。
foreign_server_name	information_schema.sql_identifier	外部服务器名称。
foreign_data_wrapper_catalog	information_schema.sql_identifier	外部数据封装器所在database名称（永远为当前数据库）。
foreign_data_wrapper_name	information_schema.sql_identifier	外部数据封装器名称。
foreign_server_type	information_schema.character_data	外部服务器的类型。
foreign_server_version	information_schema.character_data	外部服务器的版本。
authorization_identifier	information_schema.sql_identifier	外部服务器的所有者的角色名称。

13.1.3 _PG_FOREIGN_TABLE_COLUMNS

显示外部表的列信息。该视图只有sysadmin权限可以查看。

表 13-4 _PG_FOREIGN_TABLE_COLUMNS 字段

名称	类型	描述
nspname	name	schema名称。
relname	name	表名称。
attname	name	列名称。
attfdwoptions	text[]	外部数据封装器的属性选项，使用“keyword=value”格式的字符串。

13.1.4 _PG_FOREIGN_TABLES

存储所有的定义在本数据库的外部表信息。只显示当前用户有权访问的外部表信息。该视图只有sysadmin权限可以查看。

表 13-5 _PG_FOREIGN_TABLES 字段

名称	类型	描述
foreign_table_catalog	information_schema.sql_identifier	外部表所在的数据库名称（永远是当前数据库）。
foreign_table_schema	name	外部表的schema名称。
foreign_table_name	name	外部表的名称。
ftoptions	text[]	外部表的可选项。
foreign_server_catalog	information_schema.sql_identifier	外部服务器所在的数据库名称（永远是当前数据库）。
foreign_server_name	information_schema.sql_identifier	外部服务器的名称。
authorization_identifier	information_schema.sql_identifier	所有者的角色名称。

13.1.5 _PG_USER_MAPPINGS

存储从本地用户到远程的映射。该视图只有sysadmin权限可以查看。

表 13-6 _PG_USER_MAPPINGS 字段

名称	类型	描述
oid	oid	从本地用户到远程的映射的oid。
umoptions	text[]	用户映射指定选项，使用"keyword=value”格式的字符串。
umuser	oid	被映射的本地用户的oid，如果用户映射是公共的则为0。
authorization_identifier	information_schema.sql_identifier	本地用户角色名称。
foreign_server_catalog	information_schema.sql_identifier	外部服务器所在的数据库名称(永远是当前数据库)
foreign_server_name	information_schema.sql_identifier	外部服务器名称。
srvowner	information_schema.sql_identifier	外部服务器所有者。

13.1.6 INFORMATION_SCHEMA_CATALOG_NAME

用来显示当前所在的database的名称。

表 13-7 INFORMATION_SCHEMA_CATALOG_NAME 字段

名称	类型	描述
catalog_name	information_schema.sql_identifier	当前database的名称。

13.2 DBE_PERF Schema

DBE_PERF Schema内视图主要用来诊断性能问题，也是WDR Snapshot的数据来源。数据库安装后，默认只有初始用户和监控管理员具有模式dbe_perf的权限。若是由旧版本升级而来，为保持权限的前向兼容，模式dbe_perf的权限与旧版本保持一致。从OS、Instance、Memory等多个维度划分组织视图，并且符合如下命名规范：

- GLOBAL_开头的视图，代表从数据库节点请求数据，并将数据追加对外返回，不会处理数据。
- SUMMARY_开头的视图，代表是将数据库内的数据概述，多数情况下是返回数据库节点（有时只有数据库主节点的）的数据，会对数据进行加工和汇聚。
- 非这两者开头的视图，一般代表本地视图，不会向其它数据库节点请求数据。

13.2.1 OS

13.2.1.1 OS_RUNTIME

显示当前操作系统运行的状态信息。

表 13-8 OS_RUNTIME 字段

名称	类型	描述
id	integer	编号。
name	text	操作系统运行状态名称。
value	numeric	操作系统运行状态值。
comments	text	操作系统运行状态注释。
cumulative	boolean	操作系统运行状态的值是否为累加值。

13.2.1.2 GLOBAL_OS_RUNTIME

提供数据库中所有正常节点下的操作系统运行状态信息。

表 13-9 GLOBAL_OS_RUNTIME 字段

名称	类型	描述
node_name	name	节点名称。
id	integer	编号。
name	text	操作系统运行状态名称。
value	numeric	操作系统运行状态值。
comments	text	操作系统运行状态注释。
cumulative	boolean	操作系统运行状态的值是否为累加值。

13.2.1.3 OS_THREADS

提供当前节点下所有线程的状态信息。

表 13-10 OS_THREADS 字段

名称	类型	描述
node_name	text	当前节点的名称。
pid	bigint	当前节点进程中正在运行的线程号。
lwpid	integer	与pid对应的轻量级线程号。
thread_name	text	与pid对应的线程名称。
creation_time	timestamp with time zone	与pid对应的线程创建的时间。

13.2.1.4 GLOBAL_OS_THREADS

提供数据库中所有正常节点下的线程状态信息。

表 13-11 GLOBAL_OS_THREADS 字段

名称	类型	描述
node_name	text	节点名称。
pid	bigint	当前节点进程中正在运行的线程号。
lwpid	integer	与pid对应的轻量级线程号。
thread_name	text	与pid对应的线程名称。

名称	类型	描述
creation_time	timestamp with time zone	与pid对应的线程创建的时间。

13.2.1.5 NODE_NAME

提供数据库中所有正常节点的名称。

表 13-12 NODE_NAME 字段

名称	类型	描述
node_name	name	节点名称。

13.2.2 Instance

13.2.2.1 INSTANCE_TIME

提供当前数据库节点下的各种时间消耗信息，主要分为以下类型：

- DB_TIME：作业在多核下的有效时间花销。
- CPU_TIME：CPU的时间花销。
- EXECUTION_TIME：执行器内的时间花销。
- PARSE_TIME：SQL解析的时间花销。
- PLAN_TIME：生成Plan的时间花销。
- REWRITE_TIME：SQL重写的时间花销。
- PL_EXECUTION_TIME ： PL/SQL（存储过程）执行的时间花销。
- PL_COMPILATION_TIME：PL/SQL（存储过程）编译的时间花销。
- NET_SEND_TIME：网络上的时间花销。
- DATA_IO_TIME：I/O上的时间花销。

表 13-13 INSTANCE_TIME 字段

名称	类型	描述
stat_id	integer	统计编号。
stat_name	text	类型名称。
value	bigint	时间值（单位：微秒）。

13.2.2.2 GLOBAL_INSTANCE_TIME

提供数据库中所有正常节点下的各种时间消耗信息（时间类型见instance_time视图）。

表 13-14 GLOBAL_INSTANCE_TIME 字段

名称	类型	描述
node_name	name	节点名称。
stat_id	integer	统计编号。
stat_name	text	类型名称。
value	bigint	时间值（单位：微秒）。

13.2.3 Memory

13.2.3.1 GS_SHARED_MEMORY_DETAIL

查询当前节点所有已产生的共享内存上下文的使用信息。

表 13-15 GS_SHARED_MEMORY_DETAIL 字段

名称	类型	描述
contextname	text	内存上下文的名称。
level	smallint	内存上下文的级别。
parent	text	上级内存上下文。
totalsize	bigint	共享内存总大小（单位：字节）。
freesize	bigint	共享内存剩余大小（单位：字节）。
usedsize	bigint	共享内存使用大小（单位：字节）。

13.2.3.2 GLOBAL_MEMORY_NODE_DETAIL

显示当前数据库中所有正常节点下的内存使用情况。

表 13-16 GLOBAL_MEMORY_NODE_DETAIL 字段

名称	类型	描述
nodename	text	节点名称。

名称	类型	描述
memorytype	text	内存使用的名称。 ● max_process_memory：数据库节点可用内存的最大值。 ● process_used_memory：进程所使用的内存大小。 ● max_dynamic_memory：最大动态内存。 ● dynamic_used_memory：已使用的动态内存。 ● dynamic_peak_memory：内存的动态峰值。 ● dynamic_used_shrctx：已使用的动态共享内存上下文。 ● dynamic_peak_shrctx：共享内存上下文的动态峰值。 ● max_shared_memory：最大共享内存。 ● shared_used_memory：已使用的共享内存。 ● max_sctpcomm_memory：TCP代理通信所允许使用的最大内存。 ● sctpcomm_used_memory：TCP代理通信已使用的内存大小。 ● sctpcomm_peak_memory：TCP代理通信的内存峰值。 ● other_used_memory：其他已使用的内存大小。 ● gpu_max_dynamic_memory：GPU最大动态内存。 ● gpu_dynamic_used_memory：GPU已使用的动态内存。 ● gpu_dynamic_peak_memory：GPU内存的动态峰值。 ● pooler_conn_memory：连接池申请内存计数。 ● pooler_freeconn_memory：连接池空闲连接的内存计数。 ● storage_compress_memory：存储模块压缩使用的内存大小。 ● udf_reserved_memory：UDF预留的内存大小。
memorymbytes	integer	内存使用的大小，单位为MB。

13.2.3.3 GLOBAL_SHARED_MEMORY_DETAIL

查询数据库中所有正常节点下的共享内存上下文的使用信息。

表 13-17 GLOBAL_SHARED_MEMORY_DETAIL 字段

名称	类型	描述
node_name	name	节点名称。
contextname	text	内存上下文的名称。
level	smallint	内存上下文的级别。
parent	text	上级内存上下文。
totalsize	bigint	共享内存总大小（单位：字节）。
freesize	bigint	共享内存剩余大小（单位：字节）。
usedsize	bigint	共享内存使用大小（单位：字节）。

13.2.3.4 MEMORY_NODE_DETAIL

显示当前数据库节点内存使用情况。

表 13-18 MEMORY_NODE_DETAIL 字段

名称	类型	描述
nodename	text	节点名称。

名称	类型	描述
memorytype	text	内存的名称。 ● max_process_memory：数据库节点可用内存的最大值。 ● process_used_memory：进程所使用的内存大小。 ● max_dynamic_memory：最大动态内存。 ● dynamic_used_memory：已使用的动态内存。 ● dynamic_peak_memory：内存的动态峰值。 ● dynamic_used_shrctx：已使用的动态共享内存上下文。 ● dynamic_peak_shrctx：共享内存上下文的动态峰值。 ● max_shared_memory：最大共享内存。 ● shared_used_memory：已使用的共享内存。 ● max_sctpcomm_memory：TCP代理通信所允许使用的最大内存。 ● sctpcomm_used_memory：TCP代理通信已使用的内存大小。 ● sctpcomm_peak_memory：TCP代理通信的内存峰值。 ● other_used_memory：其他已使用的内存大小。 ● gpu_max_dynamic_memory：GPU最大动态内存。 ● gpu_dynamic_used_memory：GPU已使用的动态内存。 ● gpu_dynamic_peak_memory：GPU内存的动态峰值。 ● pooler_conn_memory：连接池申请内存计数。 ● pooler_freeconn_memory：连接池空闲连接的内存计数。 ● storage_compress_memory：存储模块压缩使用的内存大小。 ● udf_reserved_memory：UDF预留的内存大小。
memorymbytes	integer	内存使用的大小，单位为MB。

13.2.3.5 SHARED_MEMORY_DETAIL

查询当前节点所有已产生的共享内存上下文的使用信息。

表 13-19 表 1 SHARED_MEMORY_DETAIL 字段

名称	类型	描述
contextname	text	内存上下文的名称。
level	smallint	内存上下文的级别。
parent	text	上级内存上下文。
totalsize	bigint	共享内存总大小(单位：字节)。
freesize	bigint	共享内存剩余大小(单位：字节)。
usedsize	bigint	共享内存使用大小(单位：字节)。

13.2.4 File

13.2.4.1 FILE_IOSTAT

通过对数据文件I/O的统计，反映数据的I/O性能，用以发现I/O操作异常等性能问题。

表 13-20 FILE_IOSTAT 字段

名称	类型	描述
filenum	oid	文件标识。
dbid	oid	数据库标识。
spcid	oid	表空间标识。
phyrds	bigint	读物理文件的数目。
phywrts	bigint	写物理文件的数目。
phyblkrd	bigint	读物理文件块的数目。
phyblkwrt	bigint	写物理文件块的数目。
readtim	bigint	读文件的总时长（单位：微秒）。
writetim	bigint	写文件的总时长（单位：微秒）。
avgiotim	bigint	读写文件的平均时长（单位：微秒）。
lstiotim	bigint	最后一次读文件时长（单位：微秒）。
miniotim	bigint	读写文件的最小时长（单位：微秒）。
maxiowtm	bigint	读写文件的最大时长（单位：微秒）。

13.2.4.2 SUMMARY_FILE_IOSTAT

通过数据库内对数据文件I/O统计的汇总结果，反映数据的I/O性能，用以发现I/O操作异常等性能问题。

其中phyrds、phywrts、phyblkrd、phyblkwrt、readtim、writetim字段按照各节点的数据累加求和，avgiotim为各节点的平均值（总时长/总次数），lstiotim、maxiowtm取各节点的最大值，miniotim取各节点的最小值。

表 13-21 SUMMARY_FILE_IOSTAT 字段

名称	类型	描述
filenum	oid	文件标识。
dbid	oid	数据库标识。
spcid	oid	表空间标识。
phyrds	numeric	读物理文件的数目。
phywrts	numeric	写物理文件的数目。
phyblkrd	numeric	读物理文件块的数目。
phyblkwrt	numeric	写物理文件块的数目。
readtim	numeric	读文件的总时长（单位：微秒）。
writetim	numeric	写文件的总时长（单位：微秒）。
avgiotim	bigint	读写文件的平均时长（单位：微秒）。
lstiotim	bigint	最后一次读文件时长（单位：微秒）。
miniotim	bigint	读写文件的最小时长（单位：微秒）。
maxiowtm	bigint	读写文件的最大时长（单位：微秒）。

13.2.4.3 GLOBAL_FILE_IOSTAT

显示数据库内所有节点的数据文件I/O 统计信息。

表 13-22 GLOBAL_FILE_IOSTAT 字段

名称	类型	描述
node_name	name	节点名称。
filenum	oid	文件标识。
dbid	oid	数据库标识。
spcid	oid	表空间标识。
phyrds	bigint	读物理文件的数目。

名称	类型	描述
phywrts	bigint	写物理文件的数目。
phyblkrd	bigint	读物理文件块的数目。
phyblkwrt	bigint	写物理文件块的数目。
readtim	bigint	读文件的总时长（单位：微秒）。
writetim	bigint	写文件的总时长（单位：微秒）。
avgiotim	bigint	读写文件的平均时长（单位：微秒）。
lstiotim	bigint	最后一次读文件时长（单位：微秒）。
miniotim	bigint	读写文件的最小时长（单位：微秒）。
maxiowtm	bigint	读写文件的最大时长（单位：微秒）。

13.2.4.4 FILE_REDO_IOSTAT

本节点Redo(WAL)相关的统计信息。

表 13-23 FILE_REDO_IOSTAT 字段

名称	类型	描述
phywrts	bigint	向wal buffer中写的次数。
phyblkwrt	bigint	向wal buffer中写的block的块数。
writetim	bigint	向xLog文件中写操作的时间（单位：微秒）。
avgiotim	bigint	平均写xLog的时间（writetim/phywrts，单位：微秒）。
lstiotim	bigint	最后一次写xLog的时间（单位：微秒）。
miniotim	bigint	最小的写xLog时间（单位：微秒）。
maxiowtm	bigint	最大的写xLog时间（单位：微秒）。

13.2.4.5 SUMMARY_FILE_REDO_IOSTAT

数据库内汇总所有节点的Redo(WAL）相关的统计信息。其中phywrts、phyblkwrt、writetim字段按照各节点的数据累加求和，avgiotim为各节点的平均值（汇总的writetim/汇总的phywrts），lstiotim、maxiowtm取各节点的最大值，miniotim取各节点的最小值。

表 13-24 SUMMARY_FILE_REDO_IOSTAT 字段

名称	类型	描述
phywrts	numeric	向wal buffer中写的次数。
phyblkwrt	numeric	向wal buffer中写的block的块数。
writetim	numeric	向xLog文件中写操作的时间（单位：微秒）。
avgiotim	bigint	平均写xLog的时间（writetim/phywrts，单位：微秒）。
lstiotim	bigint	最后一次写xLog的时间（单位：微秒）。
miniotim	bigint	最小的写xLog时间（单位：微秒）。
maxiowtm	bigint	最大的写xLog时间（单位：微秒）。

13.2.4.6 GLOBAL_FILE_REDO_IOSTAT

显示数据库内各节点的Redo(WAL)相关统计信息。

表 13-25 GLOBAL_FILE_REDO_IOSTAT 字段

名称	类型	描述
node_name	name	节点名称。
phywrts	bigint	向wal buffer中写的次数。
phyblkwrt	bigint	向wal buffer中写的block的块数。
writetim	bigint	向xLog文件中写操作的时间（单位：微秒）。
avgiotim	bigint	平均写xLog的时间（writetim/phywrts，单位：微秒）。
lstiotim	bigint	最后一次写xLog的时间（单位：微秒）。
miniotim	bigint	最小的写xLog时间（单位：微秒）。
maxiowtm	bigint	最大的写xLog时间（单位：微秒）。

13.2.4.7 LOCAL_REL_IOSTAT

获取当前节点中数据文件I/O状态的累计值，显示为所有数据文件I/O状态的总和。

表 13-26 LOCAL_REL_IOSTAT 字段

名称	类型	描述
phyrds	bigint	读物理文件的数目。
phywrts	bigint	写物理文件的数目。
phyblkrd	bigint	读物理文件的块的数目。
phyblkwrt	bigint	写物理文件的块的数目。

13.2.4.8 GLOBAL_REL_IOSTAT

获取所有节点上的数据文件I/O统计信息。

表 13-27 GLOBAL_REL_IOSTAT 字段

名称	类型	描述
node_name	name	节点名称。
phyrds	bigint	读物理文件的数目。
phywrts	bigint	写物理文件的数目。
phyblkrd	bigint	读物理文件块的数目。
phyblkwrt	bigint	写物理文件块的数目。

13.2.4.9 SUMMARY_REL_IOSTAT

获取所有节点上的数据文件I/O 统计信息的汇总求和结果。

表 13-28 SUMMARY_REL_IOSTAT 字段

名称	类型	描述
phyrds	numeric	读物理文件的数目。
phywrts	numeric	写物理文件的数目。
phyblkrd	numeric	读物理文件的块的数目。
phyblkwrt	numeric	写物理文件的块的数目。

13.2.5 Object

13.2.5.1 STAT_USER_TABLES

显示当前节点所有Schema中用户自定义普通表的状态信息。

表 13-29 STAT_USER_TABLES 字段

名称	类型	描述
relid	oid	表的OID。
schemaname	name	该表所在的Schema名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（即没有更新索引列的行数）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计不活跃行数。
last_vacuum	timestamp with time zone	最后一次该表是手动清理的（不计算VACUUM FULL）时间。
last_autovacuum	timestamp with time zone	上次被autovacuum守护线程清理的时间。
last_analyze	timestamp with time zone	上次手动分析该表的时间。
last_autoanalyze	timestamp with time zone	上次被autovacuum守护线程分析的时间。
vacuum_count	bigint	该表被手动清理的次数（不计算VACUUM FULL）。
autovacuum_count	bigint	该表被autovacuum清理的次数。
analyze_count	bigint	该表被手动分析的次数。
autoanalyze_count	bigint	该表被autovacuum守护线程分析的次数。

13.2.5.2 SUMMARY_STAT_USER_TABLES

显示数据库各节点所有Schema中用户自定义普通表的状态信息的汇总求和结果（其中timestamp类型字段不进行求和，仅取所有节点该字段的最新值）。

表 13-30 SUMMARY_STAT_USER_TABLES

名称	类型	描述
schemaname	name	此表所在的Schema名。
relname	name	表名。
seq_scan	numeric	此表发起的顺序扫描数。
seq_tup_read	numeric	顺序扫描抓取的活跃行数。
idx_scan	numeric	此表发起的索引扫描数。
idx_tup_fetch	numeric	索引扫描抓取的活跃行数。
n_tup_ins	numeric	插入行数。
n_tup_upd	numeric	更新行数。
n_tup_del	numeric	删除行数。
n_tup_hot_upd	numeric	HOT更新行数（即没有更新索引列的行数）。
n_live_tup	numeric	估计活跃行数。
n_dead_tup	numeric	估计不活跃行数。
last_vacuum	timestamp with time zone	最后一次此表是手动清理的（不计算VACUUM FULL）时间。
last_autovacuum	timestamp with time zone	上次被autovacuum守护线程清理的时间。
last_analyze	timestamp with time zone	上次手动分析这个表的时间。
last_autoanalyze	timestamp with time zone	上次被autovacuum守护线程分析的时间。
vacuum_count	numeric	这个表被手动清理的次数（不计算VACUUM FULL）。
autovacuum_count	numeric	这个表被autovacuum清理的次数。
analyze_count	numeric	这个表被手动分析的次数。
autoanalyze_count	numeric	这个表被autovacuum守护线程分析的次数。

13.2.5.3 GLOBAL_STAT_USER_TABLES

显示数据库各节点所有Schema中用户自定义普通表的状态信息（不汇总）。

表 13-31 GLOBAL_STAT_USER_TABLES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	表的OID。
schemaname	name	此表所在的Schema名。
relname	name	表名。
seq_scan	bigint	此表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	此表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（即没有更新索引列的行数）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计不活跃行数。
last_vacuum	timestamp with time zone	最后一次此表是手动清理的（不计算VACUUM FULL）时间。
last_autovacuum	timestamp with time zone	上次被autovacuum守护线程清理的时间。
last_analyze	timestamp with time zone	上次手动分析这个表的时间。
last_autoanalyze	timestamp with time zone	上次被autovacuum守护线程分析的时间。
vacuum_count	bigint	这个表被手动清理的次数（不计算VACUUM FULL）。
autovacuum_count	bigint	这个表被autovacuum清理的次数。
analyze_count	bigint	这个表被手动分析的次数。
autoanalyze_count	bigint	这个表被autovacuum守护线程分析的次数。

13.2.5.4 STAT_USER_INDEXES

显示数据库中当前节点用户自定义普通表的索引状态信息。

表 13-32 STAT_USER_INDEXES 字段

名称	类型	描述
relid	oid	此索引的表的OID。
indexrelid	oid	索引的OID。
schemaname	name	索引所在的Schema名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	该索引上执行的索引扫描次数。
idx_tup_read	bigint	该索引上扫描返回的索引项数。
idx_tup_fetch	bigint	使用该索引的简单索引扫描在原表中抓取的活跃行数。

13.2.5.5 SUMMARY_STAT_USER_INDEXES

显示数据库各节点所有Schema中用户自定义普通表的索引状态信息的汇总求和结果。

表 13-33 SUMMARY_STAT_USER_INDEXES 字段

名称	类型	描述
schemaname	name	索引所在的Schema名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	numeric	该索引上执行的索引扫描次数。
idx_tup_read	numeric	该索引上扫描返回的索引项数。
idx_tup_fetch	numeric	使用该索引的简单索引扫描在原表中抓取的活跃行数。

13.2.5.6 GLOBAL_STAT_USER_INDEXES

显示数据库各节点所有Schema中用户自定义普通表的索引状态信息（不同节点数据不汇总求和）。

表 13-34 GLOBAL_STAT_USER_INDEXES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	这个索引的表的OID。
indexrelid	oid	索引的OID。
schemaname	name	索引所在的Schema名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	该索引上执行的索引扫描数。
idx_tup_read	bigint	该索引上扫描返回的索引项数。
idx_tup_fetch	bigint	使用该索引的简单索引扫描在原表中抓取的活跃行数。

13.2.5.7 STAT_SYS_TABLES

显示当前节点内pg_catalog、information_schema以及pg_toast模式下所有系统表的状态信息。

表 13-35 STAT_SYS_TABLES 字段

名称	类型	描述
relid	oid	表的OID。
schemaname	name	该表所在的Schema名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（即没有更新索引列的行数）。

名称	类型	描述
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计不活跃行数。
last_vacuum	timestamp with time zone	最后一次该表是手动清理的（不计算VACUUM FULL）时间。
last_autovacuum	timestamp with time zone	上次被autovacuum守护线程清理的时间。
last_analyze	timestamp with time zone	上次手动分析该表的时间。
last_autoanalyze	timestamp with time zone	上次被autovacuum守护线程分析的时间。
vacuum_count	bigint	这个表被手动清理的次数（不计算VACUUM FULL）。
autovacuum_count	bigint	该表被autovacuum清理的次数。
analyze_count	bigint	该表被手动分析的次数。
autoanalyze_count	bigint	该表被autovacuum守护线程分析的次数。

13.2.5.8 SUMMARY_STAT_SYS_TABLES

显示数据库各节点pg_catalog、information_schema以及pg_toast模式下所有系统表的状态信息的汇总求和结果（对每个节点下系统表的状态信息汇总求和，其中timestamp类型字段不进行求和，仅取所有节点该字段的最新值）。

表 13-36 SUMMARY_STAT_SYS_TABLES 字段

名称	类型	描述
schemaname	name	此表所在的Schema名。
relname	name	表名。
seq_scan	numeric	此表发起的顺序扫描数。
seq_tup_read	numeric	顺序扫描抓取的活跃行数。
idx_scan	numeric	此表发起的索引扫描数。
idx_tup_fetch	numeric	索引扫描抓取的活跃行数。
n_tup_ins	numeric	插入行数。
n_tup_upd	numeric	更新行数。
n_tup_del	numeric	删除行数。

名称	类型	描述
n_tup_hot_upd	numeric	HOT更新行数（即没有更新索引列的行数）。
n_live_tup	numeric	估计活跃行数。
n_dead_tup	numeric	估计不活跃行数。
last_vacuum	timestamp with time zone	最后一次此表是手动清理的（不计算VACUUM FULL）时间。
last_autovacuum	timestamp with time zone	上次被autovacuum守护线程清理的时间。
last_analyze	timestamp with time zone	上次手动分析这个表的时间。
last_autoanalyze	timestamp with time zone	上次被autovacuum守护线程分析的时间。
vacuum_count	numeric	这个表被手动清理的次数（不计算VACUUM FULL）。
autovacuum_count	numeric	这个表被autovacuum清理的次数。
analyze_count	numeric	这个表被手动分析的次数。
autoanalyze_count	numeric	这个表被autovacuum守护线程分析的次数。

13.2.5.9 GLOBAL_STAT_SYS_TABLES

显示数据库各节点pg_catalog、information_schema以及pg_toast模式下所有系统表的统计信息（不同节点数据不汇总求和）。

表 13-37 GLOBAL_STAT_SYS_TABLES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	表的OID。
schemaname	name	此表所在的Schema名。
relname	name	表名。
seq_scan	bigint	此表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	此表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。

名称	类型	描述
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（即没有更新索引列的行数）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计不活跃行数。
last_vacuum	timestamp with time zone	最后一次此表是手动清理的（不计算VACUUM FULL）时间。
last_autovacuum	timestamp with time zone	上次被autovacuum守护线程清理的时间。
last_analyze	timestamp with time zone	上次手动分析这个表的时间。
last_autoanalyze	timestamp with time zone	上次被autovacuum守护线程分析的时间。
vacuum_count	bigint	这个表被手动清理的次数（不计算VACUUM FULL）。
autovacuum_count	bigint	这个表被autovacuum清理的次数。
analyze_count	bigint	这个表被手动分析的次数。
autoanalyze_count	bigint	这个表被autovacuum守护线程分析的次数。

13.2.5.10 STAT_SYS_INDEXES

显示当前节点pg_catalog、information_schema以及pg_toast模式中所有系统表的索引状态信息。

表 13-38 STAT_SYS_INDEXES 字段

名称	类型	描述
relid	oid	此索引的表的OID。
indexrelid	oid	索引的OID。
schemaname	name	索引所在的Schema名。
relname	name	索引的表名。

名称	类型	描述
indexrelname	name	索引名。
idx_scan	bigint	该索引上执行的索引扫描次数。
idx_tup_read	bigint	该索引上扫描返回的索引项数。
idx_tup_fetch	bigint	使用该索引的简单索引扫描在原表中抓取的活跃行数。

13.2.5.11 SUMMARY_STAT_SYS_INDEXES

显示数据库各节点pg_catalog、information_schema以及pg_toast模式中所有系统表的索引状态信息的汇总求和结果。

表 13-39 SUMMARY_STAT_SYS_INDEXES 字段

名称	类型	描述
schemaname	name	索引所在的Schema名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	numeric	该索引上执行的索引扫描次数。
idx_tup_read	numeric	该索引上扫描返回的索引项数。
idx_tup_fetch	numeric	使用该索引的简单索引扫描在原表中抓取的活跃行数。

13.2.5.12 GLOBAL_STAT_SYS_INDEXES

显示数据库各节点pg_catalog、information_schema以及pg_toast模式中所有系统表的索引状态信息（不同节点数据不汇总求和）。

表 13-40 GLOBAL_STAT_SYS_INDEXES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	这个索引的表的OID。
indexrelid	oid	索引的OID。

名称	类型	描述
schemaname	name	索引所在的Schema名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	该索引上执行的索引扫描次数。
idx_tup_read	bigint	该索引上扫描返回的索引项数。
idx_tup_fetch	bigint	使用该索引的简单索引扫描在原表中抓取的活跃行数。

13.2.5.13 STAT_ALL_TABLES

显示数据库当前节点每个表（包括TOAST表）的状态信息。

表 13-41 STAT_ALL_TABLES 字段

名称	类型	描述
relid	oid	表的OID。
schemaname	name	该表所在的Schema名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（即没有更新索引列的行数）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计不活跃行数。
last_vacuum	timestamp with time zone	最后一次该表是手动清理的（不计算VACUUM FULL）的时间。

名称	类型	描述
last_autovacuum	timestamp with time zone	上次被autovacuum守护线程清理的时间。
last_analyze	timestamp with time zone	上次手动分析该表的时间。
last_autoanalyze	timestamp with time zone	上次被autovacuum守护线程分析时间。
vacuum_count	bigint	该表被手动清理的次数（不计算VACUUM FULL）。
autovacuum_count	bigint	该表被autovacuum清理的次数。
analyze_count	bigint	该表被手动分析的次数。
autoanalyze_count	bigint	该表被autovacuum守护线程分析的次数。

13.2.5.14 SUMMARY_STAT_ALL_TABLES

显示数据库各节点中每个表（包括TOAST表）的统计信息的汇总求和结果（其中timestamp类型字段不进行求和，仅取所有节点该字段的最新值）。

表 13-42 SUMMARY_STAT_ALL_TABLES 字段

名称	类型	描述
schemaname	name	此表所在的Schema名。
relname	name	表名。
seq_scan	numeric	此表发起的顺序扫描数。
seq_tup_read	numeric	顺序扫描抓取的活跃行数。
idx_scan	numeric	此表发起的索引扫描数。
idx_tup_fetch	numeric	索引扫描抓取的活跃行数。
n_tup_ins	numeric	插入行数。
n_tup_upd	numeric	更新行数。
n_tup_del	numeric	删除行数。
n_tup_hot_upd	numeric	HOT更新行数（即没有更新索引列的行数）。
n_live_tup	numeric	估计活跃行数。
n_dead_tup	numeric	估计不活跃行数。

名称	类型	描述
last_vacuum	timestamp with time zone	最后一次此表是手动清理的（不计算VACUUM FULL）的时间。
last_autovacuum	timestamp with time zone	上次被autovacuum守护线程清理的时间。
last_analyze	timestamp with time zone	上次手动分析这个表的时间。
last_autoanalyze	timestamp with time zone	上次被autovacuum守护线程分析时间。
vacuum_count	numeric	这个表被手动清理的次数（不计算VACUUM FULL）。
autovacuum_count	numeric	这个表被autovacuum清理的次数。
analyze_count	numeric	这个表被手动分析的次数。
autoanalyze_count	numeric	这个表被autovacuum守护线程分析的次数。

13.2.5.15 GLOBAL_STAT_ALL_TABLES

显示数据库各节点中每个表（包括TOAST表）的统计信息（不同节点数据不汇总求和）。

表 13-43 GLOBAL_STAT_ALL_TABLES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	表的OID。
schemaname	name	此表所在的Schema名。
relname	name	表名。
seq_scan	bigint	此表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	此表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。

名称	类型	描述
n_tup_hot_upd	bigint	HOT更新行数（即没有更新索引列的行数）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计不活跃行数。
last_vacuum	timestamp with time zone	最后一次此表是手动清理的（不计算VACUUM FULL）的时间。
last_autovacuum	timestamp with time zone	上次被autovacuum守护线程清理的时间。
last_analyze	timestamp with time zone	上次手动分析这个表的时间。
last_autoanalyze	timestamp with time zone	上次被autovacuum守护线程分析时间。
vacuum_count	bigint	这个表被手动清理的次数（不计算VACUUM FULL）。
autovacuum_count	bigint	这个表被autovacuum清理的次数。
analyze_count	bigint	这个表被手动分析的次数。
autoanalyze_count	bigint	这个表被autovacuum守护线程分析的次数。

13.2.5.16 STAT_ALL_INDEXES

显示数据库当前节点中的每个索引的访问信息。

表 13-44 STAT_ALL_INDEXES 字段

名称	类型	描述
relid	oid	这个索引的表的OID。
indexrelid	oid	索引的OID。
schemaname	name	索引所在的Schema名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	该索引上执行的索引扫描次数。
idx_tup_read	bigint	该索引上扫描返回的索引项数。

名称	类型	描述
idx_tup_fetch	bigint	使用该索引的简单索引扫描在原表中抓取的活跃行数。

13.2.5.17 SUMMARY_STAT_ALL_INDEXES

显示GaussDB数据库各节点的每个索引的访问信息的汇总求和结果。

表 13-45 SUMMARY_STAT_ALL_INDEXES 字段

名称	类型	描述
schemaname	name	索引所在的Schema名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	numeric	该索引上执行的索引扫描次数。
idx_tup_read	numeric	该索引上扫描返回的索引项数。
idx_tup_fetch	numeric	使用该索引的简单索引在原表中扫描抓取的活跃行数。

13.2.5.18 GLOBAL_STAT_ALL_INDEXES

显示数据库各节点中的每个索引的访问信息（每个索引在每个节点下的状态信息不汇总）。

表 13-46 GLOBAL_STAT_ALL_INDEXES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	这个索引的表的OID。
indexrelid	oid	索引的OID。
schemaname	name	索引中所在的Schema名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	该索引上执行的索引扫描次数。

名称	类型	描述
idx_tup_read	bigint	该索引上扫描返回的索引项数。
idx_tup_fetch	bigint	使用该索引的简单索引扫描在原表中抓取的活跃行数。

13.2.5.19 STAT_DATABASE

显示数据库当前节点的统计信息。

表 13-47 STAT_DATABASE 字段

名称	类型	描述
datid	oid	数据库的OID。
datname	name	此数据库的名称。
numbackends	integer	当前连接到该数据库的后端数。
xact_commit	bigint	此数据库中已经提交的事务数。
xact_rollback	bigint	此数据库中已经回滚的事务数。
blks_read	bigint	在这个数据库中读取的磁盘块的数量。
blks_hit	bigint	高速缓存中已经命中的磁盘块的次数，这种情况下不需要从磁盘读取（高速缓存只包括数据库缓冲区高速缓存，不包括操作系统的文件系统缓存）。
tup_returned	bigint	该数据库中顺序扫描获取的活跃行数和索引扫描返回的索引行数。
tup_fetched	bigint	当前数据库通过索引返回的行数。
tup_inserted	bigint	插入的行数。
tup_updated	bigint	更新的行数。
tup_deleted	bigint	删除的行数。
conflicts	bigint	由于与数据库回放发生冲突而取消的查询数量（冲突仅在备机上发生）。请参见STAT_DATABASE_CONFLICTS获取更多信息。

名称	类型	描述
temp_files	bigint	该数据库中查询语句创建的临时文件数量。统计所有临时文件，不受GUC参数log_temp_files设置值影响。
temp_bytes	bigint	该数据库中查询语句写入临时文件的数据总量。统计所有临时文件，不受GUC参数log_temp_files设置值影响。
deadlocks	bigint	在该数据库中检索的死锁数。
blk_read_time	double precision	通过数据库后端读取数据文件块花费的时间，以毫秒计算。
blk_write_time	double precision	通过数据库后端写入数据文件块花费的时间，以毫秒计算。
stats_reset	timestamp with time zone	重置当前状态统计的时间。

13.2.5.20 SUMMARY_STAT_DATABASE

显示数据库各节点的状态统计信息的汇总求和结果（其中timestamp类型字段不进行求和，仅取所有节点该字段的最新值）。

表 13-48 SUMMARY_STAT_DATABASE

名称	类型	描述
datname	name	数据库的名称。
numbackends	bigint	当前连接到该数据库的后端数。
xact_commit	numeric	此数据库中已经提交的事务数。
xact_rollback	numeric	此数据库中已经回滚的事务数。
blks_read	numeric	在这个数据库中读取的磁盘块的数量。
blks_hit	numeric	高速缓存中已经命中的磁盘块的次数，这种情况下不需要从磁盘读读取（高速缓存只包括GaussDB数据库缓冲区高速缓存，不包括操作系统的文件系统缓存）。
tup_returned	numeric	该数据库中顺序扫描获取的活跃行数和索引扫描返回的索引行数。
tup_fetched	numeric	当前数据库通过索引返回的行数。
tup_inserted	bigint	插入的行数。
tup_updated	bigint	更新的行数。

名称	类型	描述
tup_deleted	bigint	删除的行数。
conflicts	bigint	由于与数据库回放发生冲突而取消的查询数量（冲突仅在备机上发生）。请参见STAT_DATABASE_CONFLICTS获取更多信息。
temp_files	numeric	该数据库查询语句创建的临时文件数量。统计所有临时文件，不受GUC参数log_temp_files设置值影响。
temp_bytes	numeric	该数据库查询语句写入临时文件的数据总量。统计所有临时文件，不受GUC参数log_temp_files设置值影响。
deadlocks	bigint	在该数据库中检索的死锁数。
blk_read_time	double precision	通过数据库后端读取数据文件块花费的时间，以毫秒计算。
blk_write_time	double precision	通过数据库后端写入数据文件块花费的时间，以毫秒计算。
stats_reset	timestamp with time zone	重置当前状态统计的时间。

13.2.5.21 GLOBAL_STAT_DATABASE

显示数据库各节点的统计信息（不同节点下数据库的状态信息不汇总）。

表 13-49 GLOBAL_STAT_DATABASE 字段

名称	类型	描述
node_name	name	节点名称。
datid	oid	数据库的OID。
datname	name	数据库的名称。
numbackends	integer	当前连接到该数据库的后端数。
xact_commit	bigint	此数据库中已经提交的事务数。
xact_rollback	bigint	此数据库中已经回滚的事务数。
blks_read	bigint	在这个数据库中读取的磁盘块的数量。

名称	类型	描述
blks_hit	bigint	高速缓存中已经命中的磁盘块的次数，这种情况下不需要从磁盘读取（高速缓存只包括数据库内核缓冲区高速缓存，不包括操作系统的文件系统缓存）。
tup_returned	bigint	该数据库中顺序扫描获取的活跃行数和索引扫描返回的索引行数。
tup_fetched	bigint	当前数据库通过索引返回的行数。
tup_inserted	bigint	插入的行数。
tup_updated	bigint	更新的行数。
tup_deleted	bigint	删除的行数。
conflicts	bigint	由于与数据库回放发生冲突而取消的查询数量（冲突仅在备机上发生）。请参见STAT_DATABASE_CONFLICTS获取更多信息。
temp_files	bigint	该数据库查询语句创建的临时文件数量。统计所有临时文件，不受GUC参数log_temp_files设置值影响。
temp_bytes	bigint	该数据库查询语句写入临时文件的数据总量。统计所有临时文件，不受GUC参数log_temp_files设置值影响。
deadlocks	bigint	在该数据库中检索的死锁数。
blk_read_time	double precision	通过数据库后端读取数据文件块花费的时间，以毫秒计算。
blk_write_time	double precision	通过数据库后端写入数据文件块花费的时间，以毫秒计算。
stats_reset	timestamp with time zone	重置当前状态统计的时间。

13.2.5.22 STAT_DATABASE_CONFLICTS

显示数据库当前节点冲突状态的统计信息。

表 13-50 STAT_DATABASE_CONFLICTS 字段

名称	类型	描述
datid	oid	数据库标识。
datname	name	数据库名称。

名称	类型	描述
confl_tablespace	bigint	冲突的表空间的数目。
confl_lock	bigint	冲突的锁数目。
confl_snapshot	bigint	冲突的快照数目。
confl_bufferpin	bigint	冲突的缓冲区数目。
confl_deadlock	bigint	冲突的死锁数目。

13.2.5.23 SUMMARY_STAT_DATABASE_CONFLICTS

显示数据库各节点冲突状态的统计信息的汇总求和结果。

表 13-51 SUMMARY_STAT_DATABASE_CONFLICTS 字段

名称	类型	描述
datname	name	数据库名称。
confl_tablespace	bigint	冲突的表空间的数目。
confl_lock	bigint	冲突的锁数目。
confl_snapshot	bigint	冲突的快照数目。
confl_bufferpin	bigint	冲突的缓冲区数目。
confl_deadlock	bigint	冲突的死锁数目。

13.2.5.24 GLOBAL_STAT_DATABASE_CONFLICTS

显示数据库各节点冲突状态的统计信息（不同节点下，每个数据库状态信息不汇总求和）。

表 13-52 GLOBAL_STAT_DATABASE_CONFLICTS 字段

名称	类型	描述
node_name	name	节点名称。
datid	oid	数据库标识。
datname	name	数据库名称。
confl_tablespace	bigint	冲突的表空间的数目。

名称	类型	描述
confl_lock	bigint	冲突的锁数目。
confl_snapshot	bigint	冲突的快照数目。
confl_bufferpin	bigint	冲突的缓冲区数目。
confl_deadlock	bigint	冲突的死锁数目。

13.2.5.25 STAT_XACT_ALL_TABLES

显示当前节点所有Schema中所有普通表和toast表的事务状态信息。

表 13-53 STAT_XACT_ALL_TABLES 字段

名称	类型	描述
relid	oid	表的OID。
schemaname	name	该表所在的Schema名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（即没有更新索引列的行数）。

13.2.5.26 SUMMARY_STAT_XACT_ALL_TABLES

显示数据库各节点所有Schema中所有普通表和toast表的事务状态信息的汇总求和结果。

表 13-54 SUMMARY_STAT_XACT_ALL_TABLES 字段

名称	类型	描述
schemaname	name	此表所在的Schema名。
relname	name	表名。
seq_scan	numeric	此表发起的顺序扫描数。
seq_tup_read	numeric	顺序扫描抓取的活跃行数。
idx_scan	numeric	此表发起的索引扫描数。
idx_tup_fetch	numeric	索引扫描抓取的活跃行数。
n_tup_ins	numeric	插入行数。
n_tup_upd	numeric	更新行数。
n_tup_del	numeric	删除行数。
n_tup_hot_upd	numeric	HOT更新行数（即没有更新索引列的行数）。

13.2.5.27 GLOBAL_STAT_XACT_ALL_TABLES

显示数据库各节点所有Schema中所有普通表和toast表的事务状态信息（不同节点下表的事务状态信息不进行汇总求和）。

表 13-55 GLOBAL_STAT_XACT_ALL_TABLES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	表的OID。
schemaname	name	此表所在的Schema名。
relname	name	表名。
seq_scan	bigint	此表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	此表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。

名称	类型	描述
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（即没有更新索引列的行数）。

13.2.5.28 STAT_XACT_SYS_TABLES

显示当前节点Schema系统表的事务状态信息。

表 13-56 STAT_XACT_SYS_TABLES 字段

名称	类型	描述
relid	oid	表的OID。
schemaname	name	该表所在的Schema名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（即没有更新索引列的行数）。

13.2.5.29 SUMMARY_STAT_XACT_SYS_TABLES

显示数据库各节点的Schema中系统表的事务状态信息的汇总求和结果。

表 13-57 SUMMARY_STAT_XACT_SYS_TABLES 字段

名称	类型	描述
schemaname	name	此表所在的Schema名。
relname	name	表名。

名称	类型	描述
seq_scan	numeric	此表发起的顺序扫描数。
seq_tup_read	numeric	顺序扫描抓取的活跃行数。
idx_scan	numeric	此表发起的索引扫描数。
idx_tup_fetch	numeric	索引扫描抓取的活跃行数。
n_tup_ins	numeric	插入行数。
n_tup_upd	numeric	更新行数。
n_tup_del	numeric	删除行数。
n_tup_hot_upd	numeric	HOT更新行数（即没有更新索引列的行数）。

13.2.5.30 GLOBAL_STAT_XACT_SYS_TABLES

显示数据库各节点Schema中系统表的事务状态信息（不同节点下表的事务状态信息不进行汇总求和）。

表 13-58 GLOBAL_STAT_XACT_SYS_TABLES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	表的OID。
schemaname	name	此表所在的Schema名。
relname	name	表名。
seq_scan	bigint	此表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	此表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（即没有更新索引列的行数）。

13.2.5.31 STAT_XACT_USER_TABLES

显示当前节点Schema中用户表的事务状态信息。

表 13-59 STAT_XACT_USER_TABLES 字段

名称	类型	描述
relid	oid	表的OID。
schemaname	name	该表所在的Schema名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（即没有更新索引列的行数）。

13.2.5.32 SUMMARY_STAT_XACT_USER_TABLES

显示数据库各节点Schema中用户表的事务状态信息的汇总求和结果。

表 13-60 SUMMARY_STAT_XACT_USER_TABLES 字段

名称	类型	描述
schemaname	name	此表所在的Schema名。
relname	name	表名。
seq_scan	numeric	此表发起的顺序扫描数。
seq_tup_read	numeric	顺序扫描抓取的活跃行数。
idx_scan	numeric	此表发起的索引扫描数。
idx_tup_fetch	numeric	索引扫描抓取的活跃行数。
n_tup_ins	numeric	插入行数。

名称	类型	描述
n_tup_upd	numeric	更新行数。
n_tup_del	numeric	删除行数。
n_tup_hot_upd	numeric	HOT更新行数（即没有更新索引列的行数）。

13.2.5.33 GLOBAL_STAT_XACT_USER_TABLES

显示数据库各节点Schema中用户表的事务状态信息（不同节点下表的事务状态信息不进行汇总求和）。

表 13-61 GLOBAL_STAT_XACT_USER_TABLES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	表的OID。
schemaname	name	此表所在的Schema名。
relname	name	表名。
seq_scan	bigint	此表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	此表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（即没有更新索引列的行数）。

13.2.5.34 STAT_XACT_USER_FUNCTIONS

显示当前节点本事务内函数执行的统计信息。

表 13-62 STAT_XACT_USER_FUNCTIONS 字段

名称	类型	描述
funcid	oid	函数标识。
schemaname	name	函数所在Schema名。
funcname	name	函数名称。
calls	bigint	函数被调用的次数。
total_time	double precision	此函数及其调用的所有其他函数所花费的总时间。
self_time	double precision	在此函数本身中花费的时间（不包括它调用的其他函数）。

13.2.5.35 SUMMARY_STAT_XACT_USER_FUNCTIONS

显示数据库各节点，本事务内函数执行的统计信息的汇总求和结果。

表 13-63 SUMMARY_STAT_XACT_USER_FUNCTIONS 字段

名称	类型	描述
schemaname	name	函数所在Schema名。
funcname	name	函数名称。
calls	numeric	函数被调用的次数。
total_time	double precision	此函数及其调用的所有其他函数所花费的总时间。
self_time	double precision	在此函数本身中花费的时间（不包括它调用的其他函数）。

13.2.5.36 GLOBAL_STAT_XACT_USER_FUNCTIONS

显示数据库各节点，本事务内函数执行的统计信息（不同节点下的数据不进行汇总求和）。

表 13-64 GLOBAL_STAT_XACT_USER_FUNCTIONS 字段

名称	类型	描述
node_name	name	节点名称。
funcid	oid	函数标识。

名称	类型	描述
schemaname	name	函数所在Schema名。
funcname	name	函数名称。
calls	bigint	函数被调用的次数。
total_time	double precision	此函数及其调用的所有其他函数所花费的总时间。
self_time	double precision	在此函数本身中花费的总时间（不包括它调用的其他函数）。

13.2.5.37 STAT_BAD_BLOCK

获得当前节点表、索引等文件的读取失败信息。

表 13-65 STAT_BAD_BLOCK 字段

名称	类型	描述
nodename	text	节点名称。
databaseid	integer	database的oid。
tablespaceid	integer	tablespace的oid。
relfilenode	integer	relation的file node。
bucketid	smallint	一致性hash bucket ID。
forknum	integer	fork编号。
error_count	integer	error的数量。
first_time	timestamp with time zone	页面损坏第一次出现的时间。
last_time	timestamp with time zone	页面损坏最后出现的时间。

13.2.5.38 SUMMARY_STAT_BAD_BLOCK

获得数据库各节点的表、索引等文件的读取失败信息的汇总求和结果（其中first_time取最早的时间，last_time取最新的时间）。

表 13-66 SUMMARY_STAT_BAD_BLOCK 字段

名称	类型	描述
databaseid	integer	database的oid。
tablespaceid	integer	tablespace的oid。
relfilenode	integer	relation的file node。
forknum	bigint	fork编号。
error_count	bigint	error的数量。
first_time	timestamp with time zone	页面损坏第一次出现的时间。
last_time	timestamp with time zone	页面损坏最后出现的时间。

13.2.5.39 GLOBAL_STAT_BAD_BLOCK

获得数据库各节点的表、索引等文件的读取失败信息（不同节点下各文件的读取失败信息不进行汇总求和）。

表 13-67 GLOBAL_STAT_BAD_BLOCK 字段

名称	类型	描述
node_name	text	节点名称。
databaseid	integer	database的oid。
tablespaceid	integer	tablespace的oid。
relfilenode	integer	relation的file node。
forknum	integer	fork编号。
error_count	integer	error的数量。
first_time	timestamp with time zone	页面损坏第一次出现的时间。
last_time	timestamp with time zone	页面损坏最后出现的时间。

13.2.5.40 STAT_USER_FUNCTIONS

显示当前节点的Schema中用户自定义函数（函数语言为非内部语言）的状态信息。

表 13-68 STAT_USER_FUNCTIONS 字段

名称	类型	描述
funcid	oid	函数标识。
schemaname	name	Schema的名称。
funcname	name	用户自定义函数的名称。
calls	bigint	函数被调用的次数。
total_time	double precision	调用此函数花费的总时间，包含调用其它函数的时间（单位：毫秒）。
self_time	double precision	调用此函数本身花费的时间，不包含调用其它函数的时间（单位：毫秒）。

13.2.5.41 SUMMARY_STAT_USER_FUNCTIONS

显示数据库各节点的用户自定义函数的相关统计信息的汇总求和结果。

表 13-69 SUMMARY_STAT_USER_FUNCTIONS 字段

名称	类型	描述
schemaname	name	Schema的名称。
funcname	name	用户自定义函数的名称。
calls	numeric	该函数被调用的次数。
total_time	double precision	调用此函数的总时间花费，包含调用其它函数的时间（单位：毫秒）。
self_time	double precision	调用此函数本身时间的花费，不包含调用其它函数的时间（单位：毫秒）。

13.2.5.42 GLOBAL_STAT_USER_FUNCTIONS

显示数据库各个节点的用户自定义函数的统计信息（不同节点下的统计信息不进行汇总求和）。

表 13-70 GLOBAL_STAT_USER_FUNCTIONS 字段

名称	类型	描述
node_name	name	节点名称。
funcid	oid	函数的id。

名称	类型	描述
schemaname	name	此函数所在Schema的名称。
funcname	name	用户自定义函数的名称。
calls	bigint	该函数被调用的次数。
total_time	double precision	调用此函数花费的总时间，包含调用其它函数的时间（单位：毫秒）。
self_time	double precision	调用此函数本身花费的时间，不包含调用其它函数的时间（单位：毫秒）。

13.2.6 Workload

13.2.6.1 WORKLOAD_SQL_COUNT

显示当前节点workload上的SQL数量分布。

表 13-71 WORKLOAD_SQL_COUNT 字段

名称	类型	描述
workload	name	负载名称。
select_count	bigint	select数量。
update_count	bigint	update数量。
insert_count	bigint	insert数量。
delete_count	bigint	delete数量。
ddl_count	bigint	ddl数量。
dml_count	bigint	dml数量。
dcl_count	bigint	dcl数量。

13.2.6.2 SUMMARY_WORKLOAD_SQL_COUNT

显示数据库内各数据库主节点的workload上的SQL数量分布。

表 13-72 SUMMARY_WORKLOAD_SQL_COUNT 字段

名称	类型	描述
node_name	name	节点名称。
workload	name	负载名称。
select_count	bigint	select数量。
update_count	bigint	update数量。
insert_count	bigint	insert数量。
delete_count	bigint	delete数量。
ddl_count	bigint	ddl数量。
dml_count	bigint	dml数量。
dcl_count	bigint	dcl数量。

13.2.6.3 WORKLOAD_TRANSACTION

当前节点上负载的事务信息。

表 13-73 WORKLOAD_TRANSACTION 字段

名称	类型	描述
workload	name	负载的名称。
commit_counter	bigint	用户事务commit数量。
rollback_counter	bigint	用户事务rollback数量。
resp_min	bigint	用户事务最小响应时间（单位：微秒）。
resp_max	bigint	用户事务最大响应时间（单位：微秒）。
resp_avg	bigint	用户事务平均响应时间（单位：微秒）。
resp_total	bigint	用户事务总响应时间（单位：微秒）。
bg_commit_counter	bigint	后台事务commit数量。
bg_rollback_counter	bigint	后台事务rollback数量。
bg_resp_min	bigint	后台事务最小响应时间（单位：微秒）。
bg_resp_max	bigint	后台事务最大响应时间（单位：微秒）。

名称	类型	描述
bg_resp_avg	bigint	后台事务平均响应时间（单位：微秒）。
bg_resp_total	bigint	后台事务总响应时间（单位：微秒）。

13.2.6.4 SUMMARY_WORKLOAD_TRANSACTION

显示数据库内汇聚的负载事务信息。

表 13-74 SUMMARY_WORKLOAD_TRANSACTION 字段

名称	类型	描述
workload	name	负载的名称。
commit_counter	numeric	用户事务commit数量。
rollback_counter	numeric	用户事务rollback数量。
resp_min	bigint	用户事务最小响应时间（单位：微秒）。
resp_max	bigint	用户事务最大响应时间（单位：微秒）。
resp_avg	bigint	用户事务平均响应时间（单位：微秒）。
resp_total	numeric	用户事务总响应时间（单位：微秒）。
bg_commit_counter	numeric	后台事务commit数量。
bg_rollback_counter	numeric	后台事务rollback数量。
bg_resp_min	bigint	后台事务最小响应时间（单位：微秒）。
bg_resp_max	bigint	后台事务最大响应时间（单位：微秒）。
bg_resp_avg	bigint	后台事务平均响应时间（单位：微秒）。
bg_resp_total	numeric	后台事务总响应时间（单位：微秒）。

13.2.6.5 GLOBAL_WORKLOAD_TRANSACTION

显示各节点上的workload的负载信息。

表 13-75 GLOBAL_WORKLOAD_TRANSACTION 字段

名称	类型	描述
node_name	name	节点名称。
workload	name	负载的名称。

名称	类型	描述
commit_counter	bigint	用户事务commit数量。
rollback_counter	bigint	用户事务rollback数量。
resp_min	bigint	用户事务最小响应时间（单位：微秒）。
resp_max	bigint	用户事务最大响应时间（单位：微秒）。
resp_avg	bigint	用户事务平均响应时间（单位：微秒）。
resp_total	bigint	用户事务总响应时间（单位：微秒）。
bg_commit_counter	bigint	后台事务commit数量。
bg_rollback_counter	bigint	后台事务rollback数量。
bg_resp_min	bigint	后台事务最小响应时间（单位：微秒）。
bg_resp_max	bigint	后台事务最大响应时间（单位：微秒）。
bg_resp_avg	bigint	后台事务平均响应时间（单位：微秒）。
bg_resp_total	bigint	后台事务总响应时间（单位：微秒）。

13.2.6.6 WORKLOAD_SQL_ELAPSE_TIME

WORKLOAD_SQL_ELAPSE_TIME用来统计workload（业务负载）上的SUID（查询/更新/插入/删除）信息。

表 13-76 WORKLOAD_SQL_ELAPSE_TIME 字段

名称	类型	描述
workload	name	workload（业务负载）名称。
total_select_elapse	bigint	总select的时间花费（单位：微秒）。
max_select_elapse	bigint	最大select的时间花费（单位：微秒）。
min_select_elapse	bigint	最小select的时间花费（单位：微秒）。
avg_select_elapse	bigint	平均select的时间花费（单位：微秒）。
total_update_elapse	bigint	总update的时间花费（单位：微秒）。
max_update_elapse	bigint	最大update的时间花费（单位：微秒）。
min_update_elapse	bigint	最小update的时间花费（单位：微秒）。
avg_update_elapse	bigint	平均update的时间花费（单位：微秒）。
total_insert_elapse	bigint	总insert的时间花费（单位：微秒）。

名称	类型	描述
max_insert_elapse	bigint	最大insert的时间花费（单位：微秒）。
min_insert_elapse	bigint	最小insert的时间花费（单位：微秒）。
avg_insert_elapse	bigint	平均insert的时间花费（单位：微秒）。
total_delete_elapse	bigint	总delete的时间花费（单位：微秒）。
max_delete_elapse	bigint	最大delete的时间花费（单位：微秒）。
min_delete_elapse	bigint	最小delete的时间花费（单位：微秒）。
avg_delete_elapse	bigint	平均delete的时间花费（单位：微秒）。

13.2.6.7 SUMMARY_WORKLOAD_SQL_ELAPSE_TIME

SUMMARY_WORKLOAD_SQL_ELAPSE_TIME用来统计数据库主节点上workload（业务）负载的SUID（查询/更新/插入/删除）信息。

表 13-77 SUMMARY_WORKLOAD_SQL_ELAPSE_TIME 字段

名称	类型	描述
node_name	name	数据库进程名称。
workload	name	workload（业务负载）名称。
total_select_elapse	bigint	总select的时间花费（单位：微秒）。
max_select_elapse	bigint	最大select的时间花费（单位：微秒）。
min_select_elapse	bigint	最小select的时间花费（单位：微秒）。
avg_select_elapse	bigint	平均select的时间花费（单位：微秒）。
total_update_elapse	bigint	总update的时间花费（单位：微秒）。
max_update_elapse	bigint	最大update的时间花费（单位：微秒）。
min_update_elapse	bigint	最小update的时间花费（单位：微秒）。
avg_update_elapse	bigint	平均update的时间花费（单位：微秒）。
total_insert_elapse	bigint	总insert的时间花费（单位：微秒）。
max_insert_elapse	bigint	最大insert的时间花费（单位：微秒）。
min_insert_elapse	bigint	最小insert的时间花费（单位：微秒）。
avg_insert_elapse	bigint	平均insert的时间花费（单位：微秒）。
total_delete_elapse	bigint	总delete的时间花费（单位：微秒）。
max_delete_elapse	bigint	最大delete的时间花费（单位：微秒）。

名称	类型	描述
min_delete_elapse	bigint	最小delete的时间花费（单位：微秒）。
avg_delete_elapse	bigint	平均delete的时间花费（单位：微秒）。

13.2.6.8 USER_TRANSACTION

USER_TRANSACTION用来统计用户执行的事务信息。monadmin用户能看到所有用户执行事务的信息

表 13-78 USER_TRANSACTION 字段

名称	类型	描述
username	name	用户的名称。
commit_counter	bigint	用户事务commit数量。
rollback_counter	bigint	用户事务rollback数量。
resp_min	bigint	用户事务最小响应时间（单位：微秒）。
resp_max	bigint	用户事务最大响应时间（单位：微秒）。
resp_avg	bigint	用户事务平均响应时间（单位：微秒）。
resp_total	bigint	用户事务总响应时间（单位：微秒）。
bg_commit_counter	bigint	后台事务commit数量。
bg_rollback_counter	bigint	后台事务rollback数量。
bg_resp_min	bigint	后台事务最小响应时间（单位：微秒）。
bg_resp_max	bigint	后台事务最大响应时间（单位：微秒）。
bg_resp_avg	bigint	后台事务平均响应时间（单位：微秒）。
bg_resp_total	bigint	后台事务总响应时间（单位：微秒）。

13.2.6.9 GLOBAL_USER_TRANSACTION

GLOBAL_USER_TRANSACTION用来统计全局用户执行的事务信息。

表 13-79 GLOBAL_USER_TRANSACTION 字段

名称	类型	描述
node_name	name	节点名称。

名称	类型	描述
username	name	用户的名称。
commit_counter	bigint	用户事务commit数量。
rollback_counter	bigint	用户事务rollback数量。
resp_min	bigint	用户事务最小响应时间（单位：微秒）。
resp_max	bigint	用户事务最大响应时间（单位：微秒）。
resp_avg	bigint	用户事务平均响应时间(单位：微秒）。
resp_total	bigint	用户事务总响应时间（单位：微秒）。
bg_commit_counter	bigint	后台事务commit数量。
bg_rollback_counter	bigint	后台事务rollback数量。
bg_resp_min	bigint	后台事务最小响应时间（单位：微秒）。
bg_resp_max	bigint	后台事务最大响应时间（单位：微秒）。
bg_resp_avg	bigint	后台事务平均响应时间（单位：微秒）。
bg_resp_total	bigint	后台事务总响应时间（单位：微秒）。

13.2.7 Session/Thread

13.2.7.1 SESSION_STAT

当前节点以会话线程或AutoVacuum线程为单位，统计会话状态信息。

表 13-80 SESSION_STAT 字段

名称	类型	描述
sessid	text	线程启动时间+线程标识。
statid	integer	统计编号。
statname	text	统计会话名称。
statunit	text	统计会话单位。
value	bigint	统计会话值。

13.2.7.2 GLOBAL_SESSION_STAT

各节点上以会话线程或AutoVacuum线程为单位，统计会话状态信息。

表 13-81 GLOBAL_SESSION_STAT 字段

名称	类型	描述
node_name	name	节点名称。
sessid	text	线程启动时间+线程标识。
statid	integer	统计编号。
statname	text	统计会话名称。
statunit	text	统计会话单位。
value	bigint	统计会话值。

13.2.7.3 SESSION_TIME

用于统计当前节点会话线程的运行时间信息，及各执行阶段所消耗时间。

表 13-82 SESSION_TIME 字段

名称	类型	描述
sessid	text	线程启动时间+线程标识。
stat_id	integer	统计编号。
stat_name	text	会话类型名称。
value	bigint	会话值。

13.2.7.4 GLOBAL_SESSION_TIME

用于统计各节点会话线程的运行时间信息，及各执行阶段所消耗时间。

表 13-83 GLOBAL_SESSION_TIME 字段

名称	类型	描述
node_name	name	节点名称。
sessid	text	线程启动时间+线程标识。
stat_id	integer	统计编号。
stat_name	text	会话类型名称。
value	bigint	会话值。

13.2.7.5 SESSION_MEMORY

统计Session级别的内存使用情况，包含执行作业在数据节点上GaussDB线程和Stream线程分配的所有内存，单位为MB。

表 13-84 SESSION_MEMORY 字段

名称	类型	描述
sessid	text	线程启动时间+线程标识。
init_mem	integer	当前正在执行作业进入执行器前已分配的内存。
used_mem	integer	当前正在执行作业已分配的内存。
peak_mem	integer	当前正在执行作业已分配的内存峰值。

13.2.7.6 GLOBAL_SESSION_MEMORY

统计各节点的Session级别的内存使用情况，包含执行作业在数据节点上GaussDB线程和Stream线程分配的所有内存，单位为MB。

表 13-85 GLOBAL_SESSION_MEMORY 字段

名称	类型	描述
node_name	name	节点名称。
sessid	text	线程启动时间+线程标识。
init_mem	integer	当前正在执行作业进入执行器前已分配的内存。
used_mem	integer	当前正在执行作业已分配的内存。
peak_mem	integer	当前正在执行作业已分配的内存峰值。

13.2.7.7 SESSION_MEMORY_DETAIL

统计线程的内存使用情况，以MemoryContext节点来统计。

表 13-86 SESSION_MEMORY_DETAIL 字段

名称	类型	描述
sessid	text	线程启动时间+线程标识。
sesstype	text	线程名称。
contextname	text	内存上下文名称。
level	smallint	内存上下文的重要级别。
parent	text	父级内存上下文名称。

名称	类型	描述
totalsize	bigint	总申请内存大小（单位：字节）。
freesize	bigint	空闲内存大小（单位：字节）。
usedsize	bigint	使用内存大小（单位：字节）。

13.2.7.8 GLOBAL_SESSION_MEMORY_DETAIL

统计各节点的线程的内存使用情况，以MemoryContext节点来统计。

表 13-87 GLOBAL_SESSION_MEMORY_DETAIL 字段

名称	类型	描述
node_name	name	节点名称。
sessid	text	线程启动时间+线程标识。
sesstype	text	线程名称。
contextname	text	内存上下文名称。
level	smallint	内存上下文的重要级别。
parent	text	父级内存上下文名称。
totalsize	bigint	总申请内存大小（单位：字节）。
freesize	bigint	空闲内存大小（单位：字节）。
usedsize	bigint	使用内存大小（单位：字节）。

13.2.7.9 SESSION_STAT_ACTIVITY

显示当前节点上正在运行的线程相关的信息。

表 13-88 SESSION_STAT_ACTIVITY 字段

名称	类型	描述
datid	oid	用户会话在后台连接到的数据库OID。
datname	name	用户会话在后台连接到的数据库名称。
pid	bigint	后台线程ID。
usesysid	oid	登录该后台的用户OID。
usename	name	登录该后台的用户名。
application_name	text	连接到该后台的应用名。

名称	类型	描述
client_addr	inet	连接到该后台的客户端的IP地址。如果此字段是null，它表明通过服务器机器上UNIX套接字连接客户端或者这是内部进程，如autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过client_addr的反向DNS查找得到。这个字段只有在启动log_hostname且使用IP连接时才非空。
client_port	integer	客户端用于与后台通讯的TCP端口号，如果使用Unix套接字，则为-1。
backend_start	timestampwith time zone	该过程开始的时间，即当客户端连接服务器时间。
xact_start	timestampwith time zone	启动当前事务的时间，如果没有事务是活跃的，则为null。如果当前查询是首个事务，则这列等同于query_start列。
query_start	timestampwith time zone	开始当前活跃查询的时间，如果state的值不是active，则这个值是上一个查询的开始时间。如果是存储过程、函数、package，则查询的是第一个查询时间，不会随着存储过程内语句运行而改变。
state_change	timestampwith time zone	上次状态改变的时间。
waiting	boolean	如果后台当前正等待锁则为true。
enqueue	text	该字段不支持。

名称	类型	描述
state	text	该后台当前总体状态。可能值是： ● active：后台正在执行一个查询。 ● idle：后台正在等待一个新的客户端命令。 ● idle in transaction：后台在事务中，但是目前无法执行查询。 ● idle in transaction (aborted)：这个状态除说明事务中有某个语句导致了错误外，类似于idle in transaction ● fastpath function call：后台正在执行一个fast-path函数。 ● disabled：如果后台禁用track_activities，则报告这个状态。 **说明** 普通用户只能查看到自己账户所对应的会话状态。即其他账户的state信息为空。例如以judy用户连接数据库后，在pg_stat_activity中查看到的普通用户joe及初始用户omm的stat信息为空。 gaussdb=# SELECT datname, usename, usesysid,state,pid FROM pg_stat_activity; datname | usename | usesysid | state | pid ----------+---------+----------+-------- +-----------------testdb | omm | 10 | | 139968752121616 testdb | omm | 10 | | 139968903116560 db_tpcds | judy | 16398 | active | 139968391403280 testdb | omm | 10 | | 139968643069712 testdb | omm | 10 | | 139968680818448 testdb | joe | 16390 | | 139968563377936 (6 rows)
resource_pool	name	用户使用的资源池。
query_id	bigint	查询语句的ID。
query	text	该后台的最新查询。如果state状态是active（活跃的），此字段显示当前正在执行的查询。所有其他情况表示上一个查询。
unique_sql_id	bigint	语句的unique sql id。
trace_id	text	驱动传入的trace id，与应用的一次请求相关联。

13.2.7.10 GLOBAL_SESSION_STAT_ACTIVITY

显示数据库内各节点上正在运行的线程相关的信息。

表 13-89 GLOBAL_SESSION_STAT_ACTIVITY 字段

名称	类型	描述
coorname	text	数据库进程名称。
datid	oid	用户会话在后台连接到的数据库OID。
datname	text	用户会话在后台连接到的数据库名称。
pid	bigint	后台线程ID。
usesysid	oid	登录该后台的用户OID。
usename	text	登录该后台的用户名。
application_name	text	连接到该后台的应用名。
client_addr	inet	连接到该后台的客户端的IP地址。如果此字段是null，它表明通过服务器机器上UNIX套接字连接客户端或者这是内部进程，如autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过client_addr的反向DNS查找得到。这个字段只有在启动log_hostname且使用IP连接时才非空。
client_port	integer	客户端用于与后台通讯的TCP端口号，如果使用Unix套接字，则为-1。
backend_start	timestampwith time zone	该过程开始的时间，即当客户端连接服务器时间。
xact_start	timestampwith time zone	启动当前事务的时间，如果没有事务是活跃的，则为null。如果当前查询是首个事务，则这列等同于query_start列。
query_start	timestampwith time zone	开始当前活跃查询的时间，如果state的值不是active，则这个值是上一个查询的开始时间。如果是存储过程、函数、package，则查询的是第一个查询时间，不会随着存储过程内语句运行而改变。
state_change	timestampwith time zone	上次状态改变的时间。
waiting	boolean	如果后台当前正等待锁则为true。
enqueue	text	该字段不支持。

名称	类型	描述
state	text	该后台当前总体状态。可能值是： • active：后台正在执行一个查询。 • idle：后台正在等待一个新的客户端命令。 • idle in transaction：后台在事务中，但是目前无法执行查询。 • idle in transaction (aborted)：后台在事务中，但事务中有语句执行失败。 • fastpath function call：后台正在执行一个fast-path函数。 • disabled：如果后台禁用track_activities，则报告这个状态。 **说明** 普通用户只能查看到自己账户所对应的会话状态。即其他账户的state信息为空。例如以judy用户连接数据库后，在pg_stat_activity中查看到的普通用户joe及初始用户omm的state信息为空。 gaussdb=# SELECT datname, username, usesysid,state,pid FROM pg_stat_activity; datname | username | usesysid | state | pid ----------+----------+----------+-------- +-----------------testdb | omm | 10 | | 139968752121616 testdb | omm | 10 | | 139968903116560 db_tpcds | judy | 16398 | active | 139968391403280 testdb | omm | 10 | | 139968643069712 testdb | omm | 10 | | 139968680818448 testdb | joe | 16390 | | 139968563377936 (6 rows)
resource_pool	name	用户使用的资源池。
query_id	bigint	查询语句的ID。
query	text	该后台的最新查询。如果state状态是active（活跃的），此字段显示当前正在执行的查询。所有其他情况表示上一个查询。
unique_sql_id	bigint	语句的unique sql id。
trace_id	text	驱动传入的trace id，与应用的一次请求相关联。

13.2.7.11 THREAD_WAIT_STATUS

通过该视图可以检测当前节点中工作线程（backend thread）以及辅助线程（auxiliary thread）的阻塞等待情况，具体事件信息请参见**15.3.67-表2 等待状态列**

表、15.3.67-表3 轻量级锁等待事件列表、15.3.67-表4 IO等待事件列表和15.3.67-表5 事务锁等待事件列表。

表 13-90 THREAD_WAIT_STATUS 字段

名称	类型	描述
node_name	text	当前节点的名称。
db_name	text	数据库名称。
thread_name	text	线程名称。
query_id	bigint	查询ID，对应debug_query_id。
tid	bigint	当前线程的线程号。
sessionid	bigint	session的ID。
lwtid	integer	当前线程的轻量级线程号。
psessionid	bigint	streaming线程的父线程。
tlevel	integer	streaming线程的层级。
smpid	integer	并行线程的ID。
wait_status	text	当前线程的等待状态。等待状态的详细信息请参见15.3.67-表2 等待状态列表。
wait_event	text	如果wait_status是acquire lock、acquire lwlock、wait io三种类型，此列描述具体的锁、轻量级锁、I/O的信息；否则为空。
locktag	text	当前线程正在等待锁的信息。
lockmode	text	当前线程正等待获取的锁模式。包含表级锁、行级锁、页级锁下的各模式。
block_sessionid	bigint	阻塞当前线程获取锁的会话标识。
global_sessionid	text	全局会话ID。

13.2.7.12 GLOBAL_THREAD_WAIT_STATUS

通过该视图可以检测所有节点上工作线程（backend thread）以及辅助线程（auxiliary thread）的阻塞等待情况。具体事件信息请参见15.3.67-表2 等待状态列表、15.3.67-表3 轻量级锁等待事件列表、15.3.67-表4 IO等待事件列表和15.3.67-表5 事务锁等待事件列表。

通过GLOBAL_THREAD_WAIT_STATUS视图，可以查看数据库全局各个节点上所有SQL语句产生的线程之间的调用层次关系，以及各个线程的阻塞等待状态，从而更容易定位hang以及类似现象的原因。

GLOBAL_THREAD_WAIT_STATUS视图和THREAD_WAIT_STATUS视图列定义完全相同，这是由于GLOBAL_THREAD_WAIT_STATUS视图本质是到数据库中各个节点上查询THREAD_WAIT_STATUS视图汇总的结果。

表 13-91 GLOBAL_THREAD_WAIT_STATUS 字段

名称	类型	描述
node_name	text	节点名称。
db_name	text	数据库名称。
thread_name	text	线程名称。
query_id	bigint	查询ID，对应debug_query_id。
tid	bigint	当前线程的线程号。
sessionid	bigint	session的ID。
lwtid	integer	当前线程的轻量级线程号。
psessionid	bigint	streaming线程的父线程。
tlevel	integer	streaming线程的层级。
smpid	integer	并行线程的ID。
wait_status	text	当前线程的等待状态。等待状态的详细信息请参见15.3.67-表2 等待状态列表。
wait_event	text	如果wait_status是acquire lock、acquire lwlock、wait io三种类型，此列描述具体的锁、轻量级锁、I/O的信息。否则是空。
locktag	text	当前线程正在等待锁的信息。
lockmode	text	当前线程正等待获取的锁模式。包含表级锁、行级锁、页级锁下的各模式。
block_sessionid	bigint	阻塞当前线程获取锁的会话标识。
global_sessionid	text	全局会话ID。

13.2.7.13 LOCAL_THREADPOOL_STATUS

LOCAL_THREADPOOL_STATUS视图显示线程池下工作线程及会话的状态信息。该视图仅在线程池开启（enable_thread_pool = on）时生效。

表 13-92 LOCAL_THREADPOOL_STATUS 字段

名称	类型	描述
node_name	text	节点名称。
group_id	integer	线程池组ID。
bind_numa_id	integer	该线程池组绑定的NUMA ID。
bind_cpu_number	integer	该线程池组绑定的CPU信息。如果未绑定CPU，该值为NULL。
listener	integer	该线程池组的Listener线程数量。
worker_info	text	线程池中线程相关信息，包括以下信息： ● default：该线程池组中的初始线程数量。 ● new：该线程池组中新增线程的数量。 ● expect：该线程池组中预期线程的数量。 ● actual：该线程池组中实际线程的数量。 ● idle：该线程池组中空闲线程的数量。 ● pending：该线程池组中等待线程的数量。
session_info	text	线程池中会话相关信息，包括以下信息： ● total：该线程池组中所有的会话数量。 ● waiting：该线程池组中等待调度的会话数量。 ● running：该线程池中正在执行的会话数量。 ● idle：该线程池组中空闲的会话数量。
stream_info	text	stream池相关信息，包含以下信息： ● total：该stream池组中所有的线程数量。 ● running：该stream池中正在执行的线程数量。 ● idle：该stream池组中空闲的线程数量。

13.2.7.14 GLOBAL_THREADPOOL_STATUS

GLOBAL_THREADPOOL_STATUS视图显示在所有节点上的线程池中工作线程及会话的状态信息。具体的字段表13-92。

13.2.7.15 LOCAL_ACTIVE_SESSION

LOCAL_ACTIVE_SESSION视图显示本节点上的ACTIVE SESSION PROFILE内存中的样本。

表 13-93 LOCAL_ACTIVE_SESSION 字段

名称	类型	描述
sampleid	bigint	采样ID。
sample_time	timestamp with time zone	采样的时间。
need_flush_sample	boolean	该样本是否需要刷新到磁盘。
databaseid	oid	数据库ID
thread_id	bigint	线程的ID。
sessionid	bigint	会话的ID。
start_time	timestamp with time zone	会话的启动时间。
event	text	具体的事件名称。
lwtid	integer	当前线程的轻量级线程号。
psessionid	bigint	streaming线程的父线程。
tlevel	integer	streaming线程的层级。与执行计划的层级（id）相对应。
smpid	integer	smp执行模式下并行线程的并行编号。
userid	oid	session用户的id。
application_name	text	应用的名称。
client_addr	inet	client端的地址。
client_hostname	text	client端的名称。
client_port	integer	客户端用于与后端通讯的TCP端口号。
query_id	bigint	debug query id。
unique_query_id	bigint	unique query id。
user_id	oid	unique query的key中的user_id。
cn_id	integer	cn id，在DN上表示下发该unique sql的节点id，unique query的key中的cn_id。
unique_query	text	规范化后的UniqueSQL文本串。
locktag	text	会话等待锁信息，可通过locktag_decode解析。

名称	类型	描述
lockmode	text	会话等待锁模式。
block_sessionid	bigint	如果会话正在等待锁，阻塞该会话获取锁的会话标识。
final_block_sessionid	bigint	表示源头阻塞会话id。
wait_status	text	描述event列的更多详细信息。
global_sessionid	text	全局会话ID
xact_start_time	timestamp with time zone	事务开始时间。
query_start_time	timestamp with time zone	语句开始执行时间。
state	text	当前语句状态。 可能取值为：active, idle in transaction, fastpath function call, idle in transaction (aborted), disabled, retrying。

13.2.8 Transaction

13.2.8.1 TRANSACTIONS_PREPARED_XACTS

显示当前准备好进行两阶段提交的事务的信息。

表 13-94 TRANSACTIONS_PREPARED_XACTS 字段

名称	类型	描述
transaction	xid	预备事务的数字事务标识。
gid	text	赋予该事务的全局事务标识。
prepared	timestamp with time zone	事务准备好提交的时间。
owner	name	执行该事务的用户的名称。
database	name	执行该事务所在的数据库名。

13.2.8.2 SUMMARY_TRANSACTIONS_PREPARED_XACTS

显示数据库中数据库主节点当前准备好进行两阶段提交的事务的信息。

表 13-95 SUMMARY_TRANSACTIONS_PREPARED_XACTS 字段

名称	类型	描述
transaction	xid	预备事务的数字事务标识。
gid	text	赋予该事务的全局事务标识。
prepared	timestamp with time zone	事务准备好提交的时间。
owner	name	执行该事务的用户的名称。
database	name	执行该事务所在的数据库名。

13.2.8.3 GLOBAL_TRANSACTIONS_PREPARED_XACTS

显示各节点当前准备好进行两阶段提交的事务的信息。

表 13-96 GLOBAL_TRANSACTIONS_PREPARED_XACTS 字段

名称	类型	描述
transaction	xid	预备事务的数字事务标识。
gid	text	赋予该事务的全局事务标识。
prepared	timestamp with time zone	事务准备好提交的时间。
owner	name	执行该事务的用户的名称。
database	name	执行该事务所在的数据库名。

13.2.8.4 TRANSACTIONS_RUNNING_XACTS

显示当前节点运行事务的信息。

表 13-97 TRANSACTIONS_RUNNING_XACTS 字段

名称	类型	描述
handle	integer	事务对应的事务管理器中的槽位句柄，该值恒为-1。
gxid	xid	事务id号。
state	tinyint	事务状态（3：prepared或者0：starting）。
node	text	节点名称。
xmin	xid	节点上当前数据涉及的最小事务号xmin。
vacuum	boolean	标志当前事务是否是lazy vacuum事务。

名称	类型	描述
timeline	bigint	标志数据库重启次数。
prepare_xid	xid	处于prepared状态的事务的id号，若不在prepared状态，值为0。
pid	bigint	事务对应的线程id。
next_xid	xid	本地活跃事务最小CSN值。

13.2.8.5 SUMMARY_TRANSACTIONS_RUNNING_XACTS

显示集群中各个节点运行事务的信息，字段内容和transactions_running_xacts一致。

表 13-98 SUMMARY_TRANSACTIONS_RUNNING_XACTS 字段

名称	类型	描述
handle	integer	事务对应的事务管理器中的槽位句柄，该值恒为-1。
gxid	xid	事务id号。
state	tinyint	事务状态（3：prepared或者0：starting）。
node	text	节点名称。
xmin	xid	节点上当前数据涉及的最小事务号xmin。
vacuum	boolean	标志当前事务是否是lazy vacuum事务。
timeline	bigint	标志数据库重启次数。
prepare_xid	xid	处于prepared状态的事务的id号，若不在prepared状态，值为0。
pid	bigint	事务对应的线程id。
next_xid	xid	本地活跃事务最小CSN值。

13.2.8.6 GLOBAL_TRANSACTIONS_RUNNING_XACTS

显示集群中各个节点运行事务的信息。

表 13-99 GLOBAL_TRANSACTIONS_RUNNING_XACTS 字段

名称	类型	描述
handle	integer	事务对应的事务管理器中的槽位句柄，该值恒为-1。
gxid	xid	事务id号。
state	tinyint	事务状态（3：prepared或者0：starting）。

名称	类型	描述
node	text	节点名称。
xmin	xid	节点上当前数据涉及的最小事务号xmin。
vacuum	boolean	标志当前事务是否是lazy vacuum事务。
timeline	bigint	标志数据库重启次数。
prepare_xid	xid	处于prepared状态的事务的id号，若不在prepared状态，值为0。
pid	bigint	事务对应的线程id。
next_xid	xid	本地活跃事务最小CSN值。

13.2.9 Query

13.2.9.1 STATEMENT

获得当前节点的执行语句（归一化SQL）的信息。数据库主节点上可以看到此数据库主节点接收到的归一化的SQL的全量统计信息（包含数据库节点）；数据库节点上仅可看到归一化的SQL的此节点执行的统计信息。

说明

- 不同的savepoint_name所生成的unique_sql_id不同，大量使用savepoint_name时会导致系统中产生的unique_sql_id信息快速上涨，若unique_sql_id数量高于instr_unique_sql_count数量时，新产生的unique_sql_id信息将不被统计。
- 当前版本暂不支持对FOR UPDATE关键字进行识别并归一化处理。例如：SELECT * FROM table; 与SELECT * FROM table FOR UPDATE WAIT N; 会被归一化处理为相同的归一化SQL，在query字段中体现。

表 13-100 STATEMENT 字段

名称	类型	描述
node_name	name	数据库进程名称。
node_id	integer	节点的ID。
user_name	name	用户名称。
user_id	oid	用户OID。
unique_sql_id	bigint	归一化的SQL ID。
query	text	归一化的SQL。 备注：长度受track_activity_query_size控制。
n_calls	bigint	调用次数。

名称	类型	描述
min_elapse_time	bigint	SQL在内核内的最小运行时间（单位：微秒）。
max_elapse_time	bigint	SQL在内核内的最大运行时间（单位：微秒）。
total_elapse_time	bigint	SQL在内核内的总运行时间（单位：微秒）。
n_returned_rows	bigint	SELECT返回的结果集行数。
n_tuples_fetched	bigint	随机扫描行。
n_tuples_returned	bigint	顺序扫描行。
n_tuples_inserted	bigint	插入行。
n_tuples_updated	bigint	更新行。
n_tuples_deleted	bigint	删除行。
n_blocks_fetched	bigint	buffer的块访问次数。
n_blocks_hit	bigint	buffer的块命中次数。
n_soft_parse	bigint	软解析次数，n_soft_parse + n_hard_parse可能大于n_calls，因为子查询未计入n_calls。
n_hard_parse	bigint	硬解析次数，n_soft_parse + n_hard_parse可能大于n_calls，因为子查询未计入n_calls。
db_time	bigint	有效的DB时间花费，多线程将累加（单位：微秒）。
cpu_time	bigint	CPU时间（单位：微秒）。
execution_time	bigint	执行器内执行时间（单位：微秒）。
parse_time	bigint	SQL解析时间（单位：微秒）。
plan_time	bigint	SQL生成计划时间（单位：微秒）。
rewrite_time	bigint	SQL重写时间（单位：微秒）。
pl_execution_time	bigint	plpgsql上的执行时间（单位：微秒）。
pl_compilation_time	bigint	plpgsql上的编译时间（单位：微秒）。
data_io_time	bigint	IO上的时间花费（单位：微秒）。
net_send_info	text	通过物理连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。单机模式下不支持该字段。
net_recv_info	text	通过物理连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。单机模式下不支持该字段。

名称	类型	描述
net_stream_send_info	text	通过逻辑连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。单机模式下不支持该字段。
net_stream_recv_info	text	通过逻辑连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。单机模式下不支持该字段。
last_updated	timestamp with time zone	最后一次更新该语句的时间。
sort_count	bigint	排序执行的次数。
sort_time	bigint	排序执行的时间（单位：微秒）。
sort_mem_used	bigint	排序过程中使用的work memory大小（单位：KB）。
sort_spill_count	bigint	排序过程中，若发生落盘，写文件的次数。
sort_spill_size	bigint	排序过程中，若发生落盘，使用的文件大小（单位：KB）。
hash_count	bigint	hash执行的次数。
hash_time	bigint	hash执行的时间（单位：微秒）。
hash_mem_used	bigint	hash过程中使用的work memory大小（单位：KB）。
hash_spill_count	bigint	hash过程中，若发生落盘，写文件的次数。
hash_spill_size	bigint	hash过程中，若发生落盘，使用的文件大小（单位：KB）。
parent_unique_sql_id	bigint	父语句的unique_sql_id，非存储过程子语句该值为0。

说明

n_calls表示实际调用次数，对于存储过程内的fetch语句，fetch语句的实际触发次数，对应该cursor实际执行的语句的n_calls的增加次数。

13.2.9.2 SUMMARY_STATEMENT

获得各数据库主节点的执行语句（归一化SQL）的全量信息（包含数据库节点）。

说明

当前版本暂不支持对FOR UPDATE关键字进行识别并归一化处理。例如：SELECT * FROM table;与SELECT * FROM table FOR UPDATE WAIT N; 会被归一化处理为相同的归一化SQL，在query字段中体现。

表 13-101 SUMMARY_STATEMENT 字段

名称	类型	描述
node_name	name	数据库进程名称。
node_id	integer	节点的ID。
user_name	name	用户名称。
user_id	oid	用户OID。
unique_sql_id	bigint	归一化的SQL ID。
query	text	归一化的SQL。 备注：长度受track_activity_query_size控制。
n_calls	bigint	调用次数。
min_elapse_time	bigint	SQL在内核内的最小运行时间（单位：微秒）。
max_elapse_time	bigint	SQL在内核内的最大运行时间（单位：微秒）。
total_elapse_time	bigint	SQL在内核内的总运行时间（单位：微秒）。
n_returned_rows	bigint	SELECT返回的结果集行数。
n_tuples_fetched	bigint	随机扫描行。
n_tuples_returned	bigint	顺序扫描行。
n_tuples_inserted	bigint	插入行。
n_tuples_updated	bigint	更新行。
n_tuples_deleted	bigint	删除行。
n_blocks_fetched	bigint	buffer的块访问次数。
n_blocks_hit	bigint	buffer的块命中次数。
n_soft_parse	bigint	软解析次数。
n_hard_parse	bigint	硬解析次数。
db_time	bigint	有效的DB时间花费，多线程将累加（单位：微秒）。
cpu_time	bigint	CPU时间（单位：微秒）。
execution_time	bigint	执行器内执行时间（单位：微秒）。
parse_time	bigint	SQL解析时间（单位：微秒）。
plan_time	bigint	SQL生成计划时间（单位：微秒）。
rewrite_time	bigint	SQL重写时间（单位：微秒）。

名称	类型	描述
pl_execution_time	bigint	plpgsql上的执行时间（单位：微秒）。
pl_compilation_time	bigint	plpgsql上的编译时间（单位：微秒）。
data_io_time	bigint	I/O上的时间花费（单位：微秒）。
net_send_info	text	通过物理连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。单机模式下不支持该字段。
net_recv_info	text	通过物理连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。单机模式下不支持该字段。
net_stream_send_info	text	通过逻辑连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。单机模式下不支持该字段。
net_stream_recv_info	text	通过逻辑连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。单机模式下不支持该字段。
last_updated	timestamp with time zone	最后一次更新该语句的时间。
sort_count	bigint	排序执行的次数。
sort_time	bigint	排序执行的时间（单位：微秒）。
sort_mem_used	bigint	排序过程中使用的work memory大小（单位：KB）。
sort_spill_count	bigint	排序过程中，若发生落盘，写文件的次数。
sort_spill_size	bigint	排序过程中，若发生落盘，使用的文件大小（单位：KB）。
hash_count	bigint	hash执行的次数。
hash_time	bigint	hash执行的时间（单位：微秒）。
hash_mem_used	bigint	hash过程中使用的work memory大小（单位：KB）。
hash_spill_count	bigint	hash过程中，若发生落盘，写文件的次数。
hash_spill_size	bigint	hash过程中，若发生落盘，使用的文件大小（单位：KB）。
parent_unique_sql_id	bigint	父语句的unique_sql_id，非存储过程子语句该值为0。

13.2.9.3 STATEMENT_COUNT

显示数据库当前节点当前时刻执行的五类语句（SELECT、INSERT、UPDATE、DELETE、MERGE INTO）和（DDL、DML、DCL）统计信息。

说明

管理员权限用户查询STATEMENT_COUNT视图则能看到所有用户当前节点的统计信息。当数据库或该节点重启时，计数将清零，并重新开始计数。计数以节点收到的查询数为准，数据库内部进行的查询。例如，数据库主节点收到一条查询，若下发多条查询数据库节点，那将在数据库节点上进行相应次数的计数。

表 13-102 STATEMENT_COUNT 字段

名称	类型	描述
node_name	text	数据库进程名称。
user_name	text	用户名。
select_count	bigint	select语句统计结果。
update_count	bigint	update语句统计结果。
insert_count	bigint	insert语句统计结果。
delete_count	bigint	delete语句统计结果。
mergeinto_count	bigint	merge into语句统计结果。
ddl_count	bigint	DDL语句的数量。
dml_count	bigint	DML语句的数量。
dcl_count	bigint	DCL语句的数量。
total_select_elapse	bigint	总select的时间花费（单位：微秒）。
avg_select_elapse	bigint	平均select的时间花费（单位：微秒）。
max_select_elapse	bigint	最大select的时间花费（单位：微秒）。
min_select_elapse	bigint	最小select的时间花费（单位：微秒）。
total_update_elapse	bigint	总update的时间花费（单位：微秒）。
avg_update_elapse	bigint	平均update的时间花费（单位：微秒）。
max_update_elapse	bigint	最大update的时间花费（单位：微秒）。
min_update_elapse	bigint	最小update的时间花费（单位：微秒）。
total_insert_elapse	bigint	总insert的时间花费（单位：微秒）。
avg_insert_elapse	bigint	平均insert的时间花费（单位：微秒）。
max_insert_elapse	bigint	最大insert的时间花费（单位：微秒）。
min_insert_elapse	bigint	最小insert的时间花费（单位：微秒）。
total_delete_elapse	bigint	总delete的时间花费（单位：微秒）。

名称	类型	描述
avg_delete_elapse	bigint	平均delete的时间花费（单位：微秒）。
max_delete_elapse	bigint	最大delete的时间花费（单位：微秒）。
min_delete_elapse	bigint	最小delete的时间花费（单位：微秒）。

13.2.9.4 GLOBAL_STATEMENT_COUNT

显示数据库各节点当前时刻执行的五类语句（SELECT、INSERT、UPDATE、DELETE、MERGE INTO）和(DDL、DML、DCL)统计信息。

表 13-103 GLOBAL_STATEMENT_COUNT 字段

名称	类型	描述
node_name	text	节点名称。
user_name	text	用户名。
select_count	bigint	select语句统计结果。
update_count	bigint	update语句统计结果。
insert_count	bigint	insert语句统计结果。
delete_count	bigint	delete语句统计结果。
mergeinto_count	bigint	merge into语句统计结果。
ddl_count	bigint	DDL语句的数量。
dml_count	bigint	DML语句的数量。
dcl_count	bigint	DCL语句的数量。
total_select_elapse	bigint	总select的时间花费（单位：微秒）。
avg_select_elapse	bigint	平均select的时间花费（单位：微秒）。
max_select_elapse	bigint	最大select的时间花费（单位：微秒)。
min_select_elapse	bigint	最小select的时间花费（单位：微秒）。
total_update_elapse	bigint	总update的时间花费（单位：微秒)。
avg_update_elapse	bigint	平均update的时间花费（单位：微秒）。
max_update_elapse	bigint	最大update的时间花费（单位：微秒）。
min_update_elapse	bigint	最小update的时间花费（单位：微秒）。
total_insert_elapse	bigint	总insert的时间花费（单位：微秒）。
avg_insert_elapse	bigint	平均insert的时间花费（单位：微秒）。

名称	类型	描述
max_insert_elapse	bigint	最大insert的时间花费（单位：微秒)。
min_insert_elapse	bigint	最小insert的时间花费（单位：微秒）。
total_delete_elapse	bigint	总delete的时间花费（单位：微秒）。
avg_delete_elapse	bigint	平均delete的时间花费（单位：微秒）。
max_delete_elapse	bigint	最大delete的时间花费（单位：微秒）。
min_delete_elapse	bigint	最小delete的时间花费（单位：微秒）。

13.2.9.5 SUMMARY_STATEMENT_COUNT

显示数据库汇聚各节点（数据库节点）当前时刻执行的五类语句（SELECT、INSERT、UPDATE、DELETE、MERGE INTO）和（DDL、DML、DCL）统计信息。

表 13-104 SUMMARY_STATEMENT_COUNT 字段

名称	类型	描述
user_name	text	用户名。
select_count	numeric	select语句统计结果。
update_count	numeric	update语句统计结果。
insert_count	numeric	insert语句统计结果。
delete_count	numeric	delete语句统计结果。
mergeinto_count	numeric	merge into语句统计结果。
ddl_count	numeric	DDL语句的数量。
dml_count	numeric	DML语句的数量。
dcl_count	numeric	DCL语句的数量。
total_select_elapse	numeric	总select的时间花费（单位：微秒）。
avg_select_elapse	bigint	平均select的时间花费（单位：微秒）。
max_select_elapse	bigint	最大select的时间花费（单位：微秒）。
min_select_elapse	bigint	最小select的时间花费（单位：微秒）。
total_update_elapse	numeric	总update的时间花费（单位：微秒）。
avg_update_elapse	bigint	平均update的时间花费（单位：微秒）。
max_update_elapse	bigint	最大update的时间花费（单位：微秒）。
min_update_elapse	bigint	最小update的时间花费（单位：微秒）。

名称	类型	描述
total_insert_elapse	numeric	总insert的时间花费（单位：微秒）。
avg_insert_elapse	bigint	平均insert的时间花费（单位：微秒）。
max_insert_elapse	bigint	最大insert的时间花费（单位：微秒）。
min_insert_elapse	bigint	最小insert的时间花费（单位：微秒）。
total_delete_elapse	numeric	总delete的时间花费（单位：微秒）。
avg_delete_elapse	bigint	平均delete的时间花费（单位：微秒）。
max_delete_elapse	bigint	最大delete的时间花费（单位：微秒）。
min_delete_elapse	bigint	最小delete的时间花费（单位：微秒）。

13.2.9.6 STATEMENT_RESPONSETIME_PERCENTILE

获取数据库SQL响应时间P80，P95分布信息。

表 13-105 STATEMENT_RESPONSETIME_PERCENTILE 的字段

名称	类型	描述
p80	bigint	数据库80%的SQL的响应时间（单位：微秒）。
p95	bigint	数据库95%的SQL的响应时间（单位：微秒）。

13.2.9.7 STATEMENT_HISTORY

获得当前节点的执行语句的信息。只可在系统库中查询到结果，用户库中无法查询。

说明

当前版本暂不支持对FOR UPDATE关键字进行识别并归一化处理。例如：SELECT * FROM table;与SELECT * FROM table FOR UPDATE WAIT N; 会被归一化处理为相同的归一化SQL，在query字段中体现。涉及FOR UPDATE关键字的SQL，可以通过query_plan字段进行区分，执行计划中会含有'lockRows'。

表 13-106 STATEMENT_HISTORY 字段

名称	类型	描述
db_name	name	数据库名称。
schema_name	name	schema名称。
origin_node	integer	节点名称。

名称	类型	描述
user_name	name	用户名。
application_name	text	用户发起的请求的应用程序名称。
client_addr	text	用户发起的请求的客户端地址。
client_port	integer	用户发起的请求的客户端端口。
unique_query_id	bigint	归一化SQL ID。
debug_query_id	bigint	唯一SQL ID。
query	text	归一化SQL。
start_time	timestamp with time zone	语句启动的时间。
finish_time	timestamp with time zone	语句结束的时间。
slow_sql_threshold	bigint	语句执行时慢SQL的标准。
transaction_id	bigint	事务ID。
thread_id	bigint	执行线程ID。
session_id	bigint	用户session id。
n_soft_parse	bigint	软解析次数，n_soft_parse + n_hard_parse可能大于n_calls，因为子查询未计入n_calls。
n_hard_parse	bigint	硬解析次数，n_soft_parse + n_hard_parse可能大于n_calls，因为子查询未计入n_calls。
query_plan	text	语句执行计划。
n_returned_rows	bigint	SELECT返回的结果集行数。
n_tuples_fetched	bigint	随机扫描行。
n_tuples_returned	bigint	顺序扫描行。
n_tuples_inserted	bigint	插入行。
n_tuples_updated	bigint	更新行。
n_tuples_deleted	bigint	删除行。

名称	类型	描述
n_blocks_fetched	bigint	buffer的块访问次数。
n_blocks_hit	bigint	buffer的块命中次数。
db_time	bigint	有效的DB时间花费，多线程将累加（单位：微秒）。
cpu_time	bigint	CPU时间（单位：微秒）。
execution_time	bigint	执行器内执行时间（单位：微秒）。
parse_time	bigint	SQL解析时间（单位：微秒）。
plan_time	bigint	SQL生成计划时间（单位：微秒）。
rewrite_time	bigint	SQL重写时间（单位：微秒）。
pl_execution_time	bigint	plpgsql上的执行时间（单位：微秒）。
pl_compilation_time	bigint	plpgsql上的编译时间（单位：微秒）。
data_io_time	bigint	IO上的时间花费（单位：微秒）。
net_send_info	text	通过物理连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析SQL在分布式系统下的网络开销，单机模式下不支持该字段。例如：{"time":xxx, "n_calls":xxx, "size":xxx}。
net_recv_info	text	通过物理连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析SQL在分布式系统下的网络开销，单机模式下不支持该字段。例如：{"time":xxx, "n_calls":xxx, "size":xxx}。
net_stream_send_info	text	通过逻辑连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析SQL在分布式系统下的网络开销，单机模式下不支持该字段。例如：{"time":xxx, "n_calls":xxx, "size":xxx}。
net_stream_recv_info	text	通过逻辑连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析SQL在分布式系统下的网络开销，单机模式下不支持该字段。例如：{"time":xxx, "n_calls":xxx, "size":xxx}。
lock_count	bigint	加锁次数。
lock_time	bigint	加锁耗时。
lock_wait_count	bigint	加锁等待次数。

名称	类型	描述
lock_wait_time	bigint	加锁等待耗时。
lock_max_count	bigint	最大持锁数量。
lwlock_count	bigint	轻量级加锁次数（预留）。
lwlock_wait_count	bigint	轻量级等锁次数。
lwlock_time	bigint	轻量级加锁时间（预留）。
lwlock_wait_time	bigint	轻量级加锁时间。
details	bytea	等待事件和语句锁事件的列表。 记录级别的值>=L0时，开始记录等待事件的列表。显示当前语句event等待相关的统计信息。具体事件信息见**15.3.67-表2 等待状态列表**、**15.3.67-表3 轻量级锁等待事件列表**、**15.3.67-表4 IO等待事件列表**和**15.3.67-表5 事务锁等待事件列表**。关于每种事务锁对业务的影响程度，请参考**LOCK**语法小节的详细描述。 记录级别的值是L2时，开始记录语句锁事件的列表，该列表按时间顺序记录事件，记录的数量受参数track_stmt_details_size的影响。 事件包括： • 加锁开始 • 加锁结束 • 等锁开始 • 等锁结束 • 放锁开始 • 放锁结束 • 轻量级等锁开始 • 轻量级等锁结束
is_slow_sql	boolean	该SQL是否为slow SQL
trace_id	text	驱动传入的trace id，与应用的一次请求相关联。
advise	text	可能导致该SQL为slow SQL的风险信息。 **说明** 如果语句通过SQLBypass方式执行（可通过explain查看计划判断），可以认为无慢SQL风险，本字段信息可能会缺失。

名称	类型	描述
parent_unique_sql_id	bigint	当前语句的外层SQL的归一化SQL ID，对于存储过程内执行的语句，该值为调用存储过程语句的归一化SQL ID，存储过程外的语句该值为0。

13.2.10 Cache/IO

13.2.10.1 STATIO_USER_TABLES

STATIO_USER_TABLES视图显示命名空间中所有用户关系表的I/O状态信息。

表 13-107 STATIO_USER_TABLES 字段

名称	类型	描述
relid	oid	表OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	该表的TOAST表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的TOAST表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	该表的TOAST表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该表的TOAST表索引命中缓冲区数（如果存在）。

13.2.10.2 SUMMARY_STATIO_USER_TABLES

SUMMARY_STATIO_USER_TABLES视图显示数据库内汇聚的命名空间中所有用户关系表的I/O状态信息。

表 13-108 SUMMARY_STATIO_USER_TABLES 字段

名称	类型	描述
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	numeric	从该表中读取的磁盘块数。
heap_blks_hit	numeric	此表缓存命中数。
idx_blks_read	numeric	从表中所有索引读取的磁盘块数。
idx_blks_hit	numeric	表中所有索引命中缓存数。
toast_blks_read	numeric	此表的TOAST表读取的磁盘块数（如果存在）。
toast_blks_hit	numeric	此表的TOAST表命中缓冲区数（如果存在）。
tidx_blks_read	numeric	此表的TOAST表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	numeric	此表的TOAST表索引命中缓冲区数（如果存在）。

13.2.10.3 GLOBAL_STATIO_USER_TABLES

GLOBAL_STATIO_USER_TABLES视图显示各节点的命名空间中所有用户关系表的I/O状态信息。

表 13-109 GLOBAL_STATIO_USER_TABLES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	表OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	此表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。

名称	类型	描述
toast_blks_read	bigint	此表的TOAST表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	此表的TOAST表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	此表的TOAST表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	此表的TOAST表索引命中缓冲区数（如果存在）。

13.2.10.4 STATIO_USER_INDEXES

STATIO_USER_INDEXES视图显示当前节点命名空间中所有用户关系表索引的I/O状态信息。

表 13-110 STATIO_USER_INDEXES 字段

名称	类型	描述
relid	oid	索引的表的OID。
indexrelid	oid	该索引的OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	bigint	从索引中读取的磁盘块数。
idx_blks_hit	bigint	索引命中缓存数。

13.2.10.5 SUMMARY_STATIO_USER_INDEXES

SUMMARY_STATIO_USER_INDEXES视图显示数据库内汇聚的命名空间中所有用户关系表索引的I/O状态信息。

表 13-111 SUMMARY_STATIO_USER_INDEXES 字段

名称	类型	描述
schemaname	name	该索引的模式名。
relname	name	该索引的表名。

名称	类型	描述
indexrelname	name	索引名称。
idx_blks_read	numeric	从索引中读取的磁盘块数。
idx_blks_hit	numeric	索引命中缓存数。

13.2.10.6 GLOBAL_STATIO_USER_INDEXES

GLOBAL_STATIO_USER_INDEXES视图显示各节点的命名空间中所有用户关系表索引的I/O状态信息。

表 13-112 GLOBAL_STATIO_USER_INDEXES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	索引的表的OID。
indexrelid	oid	该索引的OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	numeric	从索引中读取的磁盘块数。
idx_blks_hit	numeric	索引命中缓存数。

13.2.10.7 STATIO_USER_SEQUENCES

STATIO_USER_SEQUENCE视图显示当前节点的命名空间中所有用户关系表类型为序列的I/O状态信息。

表 13-113 STATIO_USER_SEQUENCE 字段

名称	类型	描述
relid	oid	序列OID。
schemaname	name	序列中模式名。
relname	name	序列名。

名称	类型	描述
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

13.2.10.8 SUMMARY_STATIO_USER_SEQUENCES

SUMMARY_STATIO_USER_SEQUENCES视图显示数据库内汇聚的命名空间中所有用户关系表类型为序列的I/O状态信息。

表 13-114 SUMMARY_STATIO_USER_SEQUENCES 字段

名称	类型	描述
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	numeric	从序列中读取的磁盘块数。
blks_hit	numeric	序列中缓存命中数。

13.2.10.9 GLOBAL_STATIO_USER_SEQUENCES

GLOBAL_STATIO_USER_SEQUENCES视图显示各节点的命名空间中所有用户关系表类型为序列的I/O状态信息。

表 13-115 GLOBAL_STATIO_USER_SEQUENCES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	序列OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

13.2.10.10 STATIO_SYS_TABLES

STATIO_SYS_TABLES视图显示命名空间中所有系统表的I/O状态信息。

表 13-116 STATIO_SYS_TABLES 字段

名称	类型	描述
relid	oid	表OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	该表的TOAST表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的TOAST表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	该表的TOAST表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该表的TOAST表索引命中缓冲区数（如果存在）。

13.2.10.11 SUMMARY_STATIO_SYS_TABLES

SUMMARY_STATIO_SYS_TABLES视图显示数据库内汇聚的命名空间中所有系统表的I/O状态信息。

表 13-117 SUMMARY_STATIO_SYS_TABLES 字段

名称	类型	描述
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	numeric	从该表中读取的磁盘块数。
heap_blks_hit	numeric	此表缓存命中数。
idx_blks_read	numeric	从表中所有索引读取的磁盘块数。
idx_blks_hit	numeric	表中所有索引命中缓存数。

名称	类型	描述
toast_blks_read	numeric	此表的TOAST表读取的磁盘块数（如果存在）。
toast_blks_hit	numeric	此表的TOAST表命中缓冲区数（如果存在）。
tidx_blks_read	numeric	此表的TOAST表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	numeric	此表的TOAST表索引命中缓冲区数（如果存在）。

13.2.10.12 GLOBAL_STATIO_SYS_TABLES

GLOBAL_STATIO_SYS_TABLES视图显示各节点的命名空间中所有系统表的I/O状态信息。

表 13-118 GLOBAL_STATIO_SYS_TABLES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	表OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	此表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	此表的TOAST表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	此表的TOAST表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	此表的TOAST表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	此表的TOAST表索引命中缓冲区数（如果存在）。

13.2.10.13 STATIO_SYS_INDEXES

STATIO_SYS_INDEXES显示命名空间中所有系统表索引的I/O状态信息。

表 13-119 STATIO_SYS_INDEXES 字段

名称	类型	描述
relid	oid	索引的表的OID。
indexrelid	oid	该索引的OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	bigint	从索引中读取的磁盘块数。
idx_blks_hit	bigint	索引命中缓存数。

13.2.10.14 SUMMARY_STATIO_SYS_INDEXES

SUMMARY_STATIO_SYS_INDEXES视图显示数据库内汇聚的命名空间中所有系统表索引的I/O状态信息。

表 13-120 SUMMARY_STATIO_SYS_INDEXES 字段

名称	类型	描述
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	numeric	从索引中读取的磁盘块数。
idx_blks_hit	numeric	索引命中缓存数。

13.2.10.15 GLOBAL_STATIO_SYS_INDEXES

GLOBAL_STATIO_SYS_INDEXES视图显示各节点的命名空间中所有系统表索引的I/O状态信息。

表 13-121 GLOBAL_STATIO_SYS_INDEXES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	索引的表的OID。
indexrelid	oid	该索引的OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	numeric	从索引中读取的磁盘块数。
idx_blks_hit	numeric	索引命中缓存数。

13.2.10.16 STATIO_SYS_SEQUENCES

STATIO_SYS_SEQUENCES显示命名空间中所有系统序列的I/O状态信息。

表 13-122 STATIO_SYS_SEQUENCES 字段

名称	类型	描述
relid	oid	序列OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

13.2.10.17 SUMMARY_STATIO_SYS_SEQUENCES

SUMMARY_STATIO_SYS_SEQUENCES视图显示数据库内汇聚的命名空间中所有系统序列的I/O状态信息。

表 13-123 SUMMARY_STATIO_SYS_SEQUENCES 字段

名称	类型	描述
schemaname	name	序列中模式名。

名称	类型	描述
relname	name	序列名。
blks_read	numeric	从序列中读取的磁盘块数。
blks_hit	numeric	序列中缓存命中数。

13.2.10.18 GLOBAL_STATIO_SYS_SEQUENCES

GLOBAL_STATIO_SYS_SEQUENCES视图显示各节点的命名空间中所有系统序列的I/O状态信息。

表 13-124 GLOBAL_STATIO_SYS_SEQUENCES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	序列OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

13.2.10.19 STATIO_ALL_TABLES

STATIO_ALL_TABLES视图将包含数据库中每个表（包括TOAST表）的一行，显示出特定表I/O的统计。

表 13-125 STATIO_ALL_TABLES 字段

名称	类型	描述
relid	oid	表OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表缓存命中数。

名称	类型	描述
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	该表的TOAST表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的TOAST表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	该表的TOAST表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该表的TOAST表索引命中缓冲区数（如果存在）。

13.2.10.20 SUMMARY_STATIO_ALL_TABLES

SUMMARY_STATIO_ALL_TABLES视图将包含数据库内汇聚的数据库中每个表（包括TOAST表）的I/O的统计。

表 13-126 SUMMARY_STATIO_ALL_TABLES 字段

名称	类型	描述
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	numeric	从该表中读取的磁盘块数。
heap_blks_hit	numeric	此表缓存命中数。
idx_blks_read	numeric	从表中所有索引读取的磁盘块数。
idx_blks_hit	numeric	表中所有索引命中缓存数。
toast_blks_read	numeric	此表的TOAST表读取的磁盘块数（如果存在）。
toast_blks_hit	numeric	此表的TOAST表命中缓冲区数（如果存在）。
tidx_blks_read	numeric	此表的TOAST表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	numeric	此表的TOAST表索引命中缓冲区数（如果存在）。

13.2.10.21 GLOBAL_STATIO_ALL_TABLES

GLOBAL_STATIO_ALL_TABLES视图将包含各节点的数据库中每个表（包括TOAST表）的I/O的统计。

表 13-127 GLOBAL_STATIO_ALL_TABLES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	表OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	此表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	此表的TOAST表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	此表的TOAST表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	此表的TOAST表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	此表的TOAST表索引命中缓冲区数（如果存在）。

13.2.10.22 STATIO_ALL_INDEXES

STATIO_ALL_INDEXES视图包含数据库中的每个索引行，显示特定索引的I/O的统计。

表 13-128 STATIO_ALL_INDEXES 字段

名称	类型	描述
relid	oid	索引的表的OID。
indexrelid	oid	该索引的OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。

名称	类型	描述
idx_blks_read	bigint	从索引中读取的磁盘块数。
idx_blks_hit	bigint	索引命中缓存数。

13.2.10.23 SUMMARY_STATIO_ALL_INDEXES

SUMMARY_STATIO_ALL_INDEXES视图包含数据库内汇聚的数据库中的每个索引行，显示特定索引的I/O的统计。

表 13-129 SUMMARY_STATIO_ALL_INDEXES 字段

名称	类型	描述
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	numeric	从索引中读取的磁盘块数。
idx_blks_hit	numeric	索引命中缓存数。

13.2.10.24 GLOBAL_STATIO_ALL_INDEXES

GLOBAL_STATIO_ALL_INDEXES视图包含各节点的数据库中的每个索引行，显示特定索引的I/O的统计。

表 13-130 GLOBAL_STATIO_ALL_INDEXES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	索引的表的OID。
indexrelid	oid	该索引的OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	numeric	从索引中读取的磁盘块数。

名称	类型	描述
idx_blks_hit	numeric	索引命中缓存数。

13.2.10.25 STATIO_ALL_SEQUENCES

STATIO_ALL_SEQUENCES视图包含数据库中每个序列的每一行，显示特定序列关于I/O的统计。

表 13-131 STATIO_ALL_SEQUENCES 字段

名称	类型	描述
relid	oid	序列OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

13.2.10.26 SUMMARY_STATIO_ALL_SEQUENCES

SUMMARY_STATIO_ALL_SEQUENCES视图包含数据库内汇聚的数据库中每个序列的每一行,显示特定序列关于I/O的统计。

表 13-132 SUMMARY_STATIO_ALL_SEQUENCES 字段

名称	类型	描述
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	numeric	从序列中读取的磁盘块数。
blks_hit	numeric	序列中缓存命中数。

13.2.10.27 GLOBAL_STATIO_ALL_SEQUENCES

GLOBAL_STATIO_ALL_SEQUENCES包含各节点的数据库中每个序列的每一行，显示特定序列关于I/O的统计。

表 13-133 GLOBAL_STATIO_ALL_SEQUENCES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	序列OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

13.2.11 Utility

13.2.11.1 REPLICATION_STAT

REPLICATION_STAT用于描述日志同步状态信息，如发起端发送日志位置、接收端接收日志位置等。

表 13-134 REPLICATION_STAT 字段

名称	类型	描述
pid	bigint	线程的PID。
usesysid	oid	用户系统ID。
usename	name	用户名。
application_name	text	程序名称。
client_addr	inet	客户端地址。
client_hostname	text	客户端名。
client_port	integer	客户端端口。
backend_start	timestamp with time zone	程序启动时间。
state	text	日志复制的状态： • 追赶状态 • 一致的流状态
sender_sent_location	text	发送端发送日志位置。
receiver_write_location	text	接收端write日志位置。
receiver_flush_location	text	接收端flush日志位置。

名称	类型	描述
receiver_replay_location	text	接收端replay日志位置。
sync_priority	integer	同步复制的优先级（0表示异步）。
sync_state	text	同步状态： ● 异步复制 ● 同步复制 ● 潜在同步者

13.2.11.2 GLOBAL_REPLICATION_STAT

GLOBAL_REPLICATION_STAT视图用于获得各节点描述日志同步状态信息，如发起端发送日志位置、接收端接收日志位置等。

表 13-135 GLOBAL_REPLICATION_STAT 字段

名称	类型	描述
node_name	name	节点名称。
pid	bigint	线程的PID。
usesysid	oid	用户系统ID。
usename	name	用户名。
application_name	text	程序名称。
client_addr	inet	客户端地址。
client_hostname	text	客户端名。
client_port	integer	客户端端口。
backend_start	timestamp with time zone	程序启动时间。
state	text	日志复制的状态： ● 追赶状态 ● 一致的流状态
sender_sent_location	text	发送端发送日志位置。
receiver_write_location	text	接收端write日志位置。
receiver_flush_location	text	接收端flush日志位置。

名称	类型	描述
receiver_replay_location	text	接收端replay日志位置。
sync_priority	integer	同步复制的优先级（0表示异步）。
sync_state	text	同步状态： ● 异步复制 ● 同步复制 ● 潜在同步者

13.2.11.3 REPLICATION_SLOTS

REPLICATION_SLOTS视图用于查看复制槽的信息。

表 13-136 REPLICATION_SLOTS 字段

名称	类型	描述
slot_name	text	复制槽的名称。
plugin	text	逻辑复制槽对应的输出插件名称。
slot_type	text	复制槽的类型。 ● physical：物理复制槽。 ● logical：逻辑复制槽。
datoid	oid	复制槽所在的数据库OID。
database	name	复制槽所在的数据库名称。
active	boolean	复制槽是否为激活状态。 ● t（true）： 表示是。 ● f（false）： 表示不是。
xmin	xid	数据库须为复制槽保留的最早事务的事务号。
catalog_xmin	xid	数据库须为逻辑复制槽保留的最早的涉及系统表的事务的事务号。
restart_lsn	text	复制槽需要的最早xlog的物理位置。
dummy_standby	boolean	预留参数。

13.2.11.4 GLOBAL_REPLICATION_SLOTS

GLOBAL_REPLICATION_SLOTS视图用于查看数据库各节点的复制槽的信息。

表 13-137 GLOBAL_REPLICATION_SLOTS 字段

名称	类型	描述
node_name	name	节点名称。
slot_name	text	复制槽的名称。
plugin	text	逻辑复制槽对应的输出插件名称。
slot_type	text	复制槽的类型。 ● physical：物理复制槽。 ● logical：逻辑复制槽。
datoid	oid	复制槽所在的数据库OID。
database	name	复制槽所在的数据库名称。
active	boolean	复制槽是否为激活状态。 ● t（true）：表示是。 ● f（false）：表示不是。
x_min	xid	数据库须为复制槽保留的最早事务的事务号。
catalog_xmin	xid	数据库须为逻辑复制槽保留的最早的涉及系统表的事务的事务号。
restart_lsn	text	复制槽需要的最早xlog的物理位置。
dummy_standby	boolean	预留参数。

13.2.11.5 BGWRITER_STAT

BGWRITER_STAT视图显示关于后端写进程活动的统计信息。

表 13-138 BGWRITER_STAT 字段

名称	类型	描述
checkpoints_timed	bigint	执行的定期检查点数。
checkpoints_req	bigint	执行的需求检查点数。
checkpoint_write_time	double precision	花费在检查点处理部分的时间总量，其中文件被写入到磁盘，以毫秒为单位。
checkpoint_sync_time	double precision	花费在检查点处理部分的时间总量，其中文件被同步到磁盘，以毫秒为单位。
buffers_checkpoint	bigint	检查点写缓冲区数量。

名称	类型	描述
buffers_clean	bigint	后端写进程写缓冲区数量。
maxwritten_clean	bigint	后端写进程停止清理扫描时间数，因为它写了太多缓冲区。
buffers_backend	bigint	通过后端直接写缓冲区数。
buffers_backend_fsync	bigint	后端不得不执行自己的fsync调用的时间数（通常后端写进程处理这些即使后端确实自己写）。
buffers_alloc	bigint	分配的缓冲区数量。
stats_reset	timestamp with time zone	这些统计被重置的时间。

13.2.11.6 GLOBAL_BGWRITER_STAT

GLOBAL_BGWRITER_STAT视图显示各节点关于后端写进程活动的统计信息。

表 13-139 GLOBAL_BGWRITER_STAT 字段

名称	类型	描述
node_name	name	节点名称。
checkpoints_timed	bigint	执行的定期检查点数。
checkpoints_req	bigint	执行的需求检查点数。
checkpoint_write_time	double precision	花费在检查点处理部分的时间总量，其中文件被写入到磁盘，以毫秒为单位。
checkpoint_sync_time	double precision	花费在检查点处理部分的时间总量，其中文件被同步到磁盘，以毫秒为单位。
buffers_checkpoint	bigint	检查点写缓冲区数量。
buffers_clean	bigint	后端写进程写缓冲区数量。
maxwritten_clean	bigint	后端写进程停止清理扫描时间数，因为它写了太多缓冲区。
buffers_backend	bigint	通过后端直接写缓冲区数。
buffers_backend_fsync	bigint	后端不得不执行自己的fsync调用的时间数（通常后端写进程处理这些即使后端确实自己写）。

名称	类型	描述
buffers_alloc	bigint	分配的缓冲区数量。
stats_reset	timestamp with time zone	这些统计被重置的时间。

13.2.11.7 GLOBAL_CKPT_STATUS

GLOBAL_CKPT_STATUS视图用于显示数据库所有实例的检查点信息和各类日志刷页情况。

表 13-140 GLOBAL_CKPT_STATUS 字段

名称	类型	描述
node_name	text	节点名称。
ckpt_redo_point	text	当前实例的检查点。
ckpt_clog_flush_num	bigint	从启动到当前时间clog刷盘页面数。
ckpt_csnlog_flush_num	bigint	从启动到当前时间csnlog刷盘页面数。
ckpt_multixact_flush_num	bigint	从启动到当前时间multixact刷盘页面数。
ckpt_predicate_flush_num	bigint	从启动到当前时间predicate刷盘页面数。
ckpt_twophase_flush_num	bigint	从启动到当前时间twophase刷盘页面数。

13.2.11.8 GLOBAL_DOUBLE_WRITE_STATUS

GLOBAL_DOUBLE_WRITE_STATUS视图显示数据库所有实例的双写文件的情况。它是由每个节点的local_double_write_stat视图组成，属性完全一致。

表 13-141 GLOBAL_DOUBLE_WRITE_STATUS 字段

名称	类型	描述
node_name	text	节点名称。
curr_dwn	bigint	当前双写文件的序列号。
curr_start_page	bigint	当前双写文件恢复起始页面。
file_trunc_num	bigint	当前双写文件复用的次数。
file_reset_num	bigint	当前双写文件写满后发生重置的次数。

名称	类型	描述
total_writes	bigint	当前双写文件总的I/O次数。
low_threshold_writes	bigint	低效率写双写文件的I/O次数（一次I/O刷页数量少于16页面）。
high_threshold_writes	bigint	高效率写双写文件的I/O次数（一次I/O刷页数量多于一批，421个页面）。
total_pages	bigint	当前刷页到双写文件区的总的页面个数。
low_threshold_pages	bigint	低效率刷页的页面个数。
high_threshold_pages	bigint	高效率刷页的页面个数。
file_id	bigint	当前双写文件的id号。

13.2.11.9 GLOBAL_PAGEWRITER_STATUS

GLOBAL_PAGEWRITER_STATUS视图显示数据库实例的刷页信息和检查点信息。

表 13-142 GLOBAL_PAGEWRITER_STATUS 字段

名称	类型	描述
node_name	text	节点名称。
pgwr_actual_flush_total_num	bigint	从启动到当前时间总计刷脏页数量。
pgwr_last_flush_num	integer	上一批刷脏页数量。
remain_dirty_page_num	bigint	当前预计还剩余多少脏页。
queue_head_page_rec_lsn	text	当前实例的脏页队列第一个脏页的recovery_lsn。
queue_rec_lsn	text	当前实例的脏页队列的recovery_lsn。
current_xlog_insert_lsn	text	当前实例xLog写入的位置。
ckpt_redo_point	text	当前实例的检查点。

13.2.11.10 GLOBAL_RECORD_RESET_TIME

GLOBAL_RECORD_RESET_TIME用于获取数据库的“重置（重启，主备倒换，数据库删除）时间”的统计信息。

表 13-143 GLOBAL_RECORD_RESET_TIME 字段

名称	类型	描述
node_name	text	节点名称。
reset_time	timestamp with time zone	重置时间点。

13.2.11.11 GLOBAL_REDO_STATUS

GLOBAL_REDO_STATUS视图显示数据库实例的日志回放情况。

表 13-144 GLOBAL_REDO_STATUS 字段

名称	类型	描述
node_name	text	节点名称。
redo_start_ptr	bigint	当前实例日志回放的起始点。
redo_start_time	bigint	当前实例日志回放的起始UTC时间。
redo_done_time	bigint	当前实例日志回放的结束UTC时间。
curr_time	bigint	当前实例的当前UTC时间。
min_recovery_point	bigint	当前实例日志的完成回放后可对外提供服务的最小一致性点。
read_ptr	bigint	当前实例日志的读取位置。
last_replayed_read_ptr	bigint	当前实例的日志回放位置。
recovery_done_ptr	bigint	当前实例启动完成时的回放位置。
read_xlog_io_counter	bigint	当前实例读取回放日志的I/O次数计数。
read_xlog_io_total_dur	bigint	当前实例读取回放日志的I/O总用时。
read_data_io_counter	bigint	当前实例日志回放过程中读取数据页面的I/O次数计数。
read_data_io_total_dur	bigint	当前实例日志回放过程中读取数据页面的I/O总用时。
write_data_io_counter	bigint	当前实例日志回放过程中写数据页面的I/O次数计数。
write_data_io_total_dur	bigint	当前实例日志回放过程中写数据页面的I/O总用时。

名称	类型	描述
process_pending_counter	bigint	当前实例日志回放过程中日志分发线程的同步次数计数。
process_pending_total_dur	bigint	当前实例日志回放过程中日志分发线程的同步总用时。
apply_counter	bigint	当前实例日志回放过程中回放线程的同步次数计数。
apply_total_dur	bigint	当前实例日志回放过程中回放线程的同步总用时。
speed	bigint	当前实例日志回放速率，每回放256MB日志该值更新一次，单位byte/s。 建议使用cm_ctl query -rv命令来获取更精确的备机回放速度（cm_ctl命令请参考《工具参考》中“统一数据库管理工具”章节）。
local_max_ptr	bigint	当前实例启动成功后本地收到的回放日志的最大值。
primary_flush_ptr	bigint	主机落盘日志的位置。
worker_info	text	当前实例回放线程信息，若没有开并行回放则该值为空。

13.2.11.12 GLOBAL_RECOVERY_STATUS

GLOBAL_RECOVERY_STATUS视图显示关于主机和备机的日志流控信息。

表 13-145 GLOBAL_RECOVERY_STATUS 字段

名称	类型	描述
node_name	text	节点名称，包含主机和备机。
standby_node_name	text	备机节点的名称。
source_ip	text	主机的IP地址。
source_port	integer	主机的端口号。
dest_ip	text	备机的IP地址。
dest_port	integer	备机的端口号。
current_rto	bigint	备机当前的日志流控时间，单位秒。
target_rto	bigint	备机通过GUC参数设置的预期流控时间，单位秒。

名称	类型	描述
current_sleep_time	bigint	为了达到此RTO预期，主机所需要执行的睡眠时间，单位微秒。

13.2.11.13 CLASS_VITAL_INFO

CLASS_VITAL_INFO视图用于做WDR时校验相同的表或者索引的oid是否一致。

表 13-146 CLASS_VITAL_INFO 字段

名称	类型	描述
relid	oid	表的oid。
schemaname	name	schema名称。
relname	name	表名。
relkind	"char"	表示对象类型，取值范围如下： ● r：表示普通表。 ● t：表示toast表。 ● i：表示索引。

13.2.11.14 USER_LOGIN

USER_LOGIN用来记录用户登录和退出次数的相关信息。

表 13-147 USER_LOGIN 字段

名称	类型	描述
node_name	text	节点名称。
user_name	text	用户名称。
user_id	integer	用户oid（同pg_authid中的oid字段)。
login_counter	bigint	登录次数。
logout_counter	bigint	退出次数。

13.2.11.15 SUMMARY_USER_LOGIN

SUMMARY_USER_LOGIN用来记录数据库主节点上用户登录和退出次数的相关信息。

表 13-148 SUMMARY_USER_LOGIN 字段

名称	类型	描述
node_name	text	节点名称。
user_name	text	用户名称。
user_id	integer	用户oid（同pg_authid中的oid字段）。
login_counter	bigint	登录次数。
logout_counter	bigint	退出次数。

13.2.11.16 GLOBAL_SINGLE_FLUSH_DW_STATUS

GLOBAL_SINGLE_FLUSH_DW_STATUS视图显示数据库所有实例单页面淘汰双写文件信息。显示内容中，/前是第一个版本双写文件刷页情况，/后是第二个版本双写文件刷页情况。

表 13-149 GLOBAL_SINGLE_FLUSH_DW_STATUS 字段

名称	类型	描述
node_name	text	节点名称。
curr_dwn	text	当前双写文件的序列号。
curr_start_page	text	当前双写文件start位置。
total_writes	text	当前双写文件总计写数据页面个数。
file_trunc_num	text	当前双写文件复用的次数。
file_reset_num	text	当前双写文件写满后发生重置的次数。

13.2.11.17 GLOBAL_CANDIDATE_STATUS

GLOBAL_CANDIDATE_STATUS视图显示整个数据库所有实例候选buffer个数和buffer淘汰信息。

表 13-150 GLOBAL_GET_BGWRITER_STATUS 字段

名称	类型	描述
node_name	text	节点名称。
candidate_slots	integer	当前Normal Buffer Pool候选buffer链中页面个数。
get_buf_from_list	bigint	Normal Buffer Pool，buffer淘汰从候选buffer链中获取页面的次数。

名称	类型	描述
get_buf_clock_sweep	bigint	Normal Buffer Pool，buffer淘汰从原淘汰方案中获取页面的次数。
seg_candidate_slots	integer	当前Segment Buffer Pool候选buffer链中页面个数。
seg_get_buf_from_list	bigint	Segment Buffer Pool，buffer淘汰从候选buffer链中获取页面的次数。
seg_get_buf_clock_sweep	bigint	Segment Buffer Pool，buffer淘汰从原淘汰方案中获取页面的次数。

13.2.11.18 PARALLEL_DECODE_STATUS

PARALLEL_DECODE_STATUS视图用于查看当前节点上的复制槽的并行解码信息。

表 13-151 PARALLEL_DECODE_STATUS 字段

名称	类型	描述
slot_name	text	复制槽的名称。
parallel_decode_num	integer	该复制槽的并行解码线程数。
read_change_queue_length	text	将每个解码线程读取日志队列的当前长度拼接后输出。
decode_change_queue_length	text	将每个解码线程解码结果队列的当前长度拼接后输出。
reader_lsn	text	当前reader线程读取的日志位置。
working_txn_cnt	bigint	当前拼接-发送线程中正在拼接的事务个数。
working_txn_memory	bigint	拼接-发送线程中拼接事务占用总内存（单位字节）。
decoded_time	timestamptz	该复制槽最新解码到的WAL日志时间。

13.2.11.19 GLOBAL_PARALLEL_DECODE_STATUS

GLOBAL_PARALLEL_DECODE_STATUS视图用于查看当前节点上的复制槽的并行解码信息。

表 13-152 GLOBAL_PARALLEL_DECODE_STATUS 字段

名称	类型	描述
node_name	name	节点名称。
slot_name	text	复制槽的名称。
parallel_decode_num	integer	该复制槽的并行解码线程数。
read_change_queue_length	text	将每个解码线程读取日志队列的当前长度拼接后输出。
decode_change_queue_length	text	将每个解码线程解码结果队列的当前长度拼接后输出。
reader_lsn	text	当前reader线程读取的日志位置。
working_txn_cnt	bigint	当前拼接-发送线程中正在拼接的事务个数。
working_txn_memory	bigint	拼接-发送线程中拼接事务占用总内存（单位字节）。
decoded_time	timestamptz	该复制槽最新解码到的WAL日志时间。

13.2.11.20 PARALLEL_DECODE_THREAD_INFO

PARALLEL_DECODE_THREAD_INFO视图用于查看当前节点上进行并行解码的线程信息。

表 13-153 PARALLEL_DECODE_THREAD_INFO 字段

名称	类型	描述
thread_id	bigint	线程id。
slot_name	text	复制槽名。
thread_type	text	线程种类（共三种，sender代表发送线程，reader代表读取线程，decoder代表解码线程）。
seq_number	integer	当前复制槽中同种线程的序号（从1开始）。

13.2.11.21 GLOBAL_PARALLEL_DECODE_THREAD_INFO

GLOBAL_PARALLEL_DECODE_THREAD_INFO视图用于查看当前节点上的复制槽的并行解码线程信息。

表 13-154 GLOBAL_PARALLEL_DECODE_THREAD_INFO 字段

名称	类型	描述
node_name	name	节点名称。
thread_id	bigint	线程id。
slot_name	text	复制槽名。
thread_type	text	线程种类（共三种，sender代表发送线程，reader代表读取线程，decoder代表解码线程）。
seq_number	integer	当前复制槽中同种线程的序号（从1开始）。

13.2.12 Lock

13.2.12.1 LOCKS

LOCKS视图用于查看各打开事务所持有的锁信息。

表 13-155 LOCKS 字段

名称	类型	描述
locktype	text	被锁定对象的类型：relation，extend，page，tuple，transactionid，virtualxid，object，userlock，advisory。
database	oid	被锁定对象所在数据库的OID： ● 如果被锁定的对象是共享对象，则OID为0。 ● 如果是一个事务ID，则为NULL。
relation	oid	关系的OID，如果锁定的对象不是关系，也不是关系的一部分，则为NULL。
page	integer	关系内部的页面编号，如果对象不是关系页或者不是行页，则为NULL。
tuple	smallint	页面里边的行编号，如果对象不是行，则为NULL。
bucket	integer	哈希桶号。
virtualxid	text	事务的虚拟ID，如果对象不是一个虚拟事务ID，则为NULL。
transactionid	xid	事务的ID，如果对象不是一个事务ID，则为NULL。

名称	类型	描述
classid	oid	包含该对象的系统表的OID，如果对象不是普通的数据库对象，则为NULL。
objid	oid	对象在其系统表内的OID，如果对象不是普通数据库对象，则为NULL。
objsubid	smallint	对于表的一个字段，这是字段编号；对于其他对象类型，这个字段是0；如果这个对象不是普通数据库对象，则为NULL。
virtualtransaction	text	持有此锁或者在等待此锁的事务的虚拟ID。
pid	bigint	持有或者等待这个锁的服务器线程的逻辑ID。如果锁是被一个预备事务持有的，则为NULL。
sessionid	bigint	持有或者等待这个锁的会话ID。如果锁是被一个预备事务持有的，则为NULL。
mode	text	这个线程持有的或者是期望的锁模式。
granted	boolean	● 如果锁是持有锁，则为TRUE。 ● 如果锁是等待锁，则为FALSE。
fastpath	boolean	如果通过fast-path获得锁，则为TRUE；如果通过主要的锁表获得，则为FALSE。
locktag	text	会话等待锁信息，可通过locktag_decode()函数解析。
global_sessionid	text	全局会话ID。

13.2.12.2 GLOBAL_LOCKS

GLOBAL_LOCKS视图用于查看各节点各打开事务所持有的锁信息。

表 13-156 GLOBAL_LOCKS 字段

名称	类型	描述
node_name	name	节点名称。
locktype	text	被锁定对象的类型：relation，extend，page，tuple，transactionid，virtualxid，object，userlock，advisory。
database	oid	被锁定对象所在数据库的OID： ● 如果被锁定的对象是共享对象，则OID为0。 ● 如果是一个事务ID，则为NULL。

名称	类型	描述
relation	oid	关系的OID，如果锁定的对象不是关系，也不是关系的一部分，则为NULL。
page	integer	关系内部的页面编号，如果对象不是关系页或者不是行页，则为NULL。
tuple	smallint	页面里边的行编号，如果对象不是行，则为NULL。
virtualxid	text	事务的虚拟ID，如果对象不是一个虚拟事务ID，则为NULL。
transactionid	xid	事务的ID，如果对象不是一个事务ID，则为NULL。
classid	oid	包含该对象的系统表的OID，如果对象不是普通的数据库对象，则为NULL。
objid	oid	对象在其系统表内的OID，如果对象不是普通数据库对象，则为NULL。
objsubid	smallint	对于表的一个字段，这是字段编号；对于其他对象类型，这个字段是0；如果这个对象不是普通数据库对象，则为NULL。
virtualtransaction	text	持有此锁或者在等待此锁的事务的虚拟ID。
pid	bigint	持有或者等待这个锁的服务器线程的逻辑ID。如果锁是被一个预备事务持有的，则为NULL。
mode	text	这个线程持有的或者是期望的锁模式。
granted	boolean	• 如果锁是持有锁，则为TRUE。 • 如果锁是等待锁，则为FALSE。
fastpath	boolean	如果通过fast-path获得锁，则为TRUE；如果通过主要的锁表获得，则为FALSE。

13.2.13 Wait Events

13.2.13.1 WAIT_EVENTS

WAIT_EVENTS显示当前节点的event的等待相关的统计信息。具体事件信息见**表12-351**、**表12-352**、**表12-353**和**表12-354**。关于每种事务锁对业务的影响程度，请参考**LOCK**语法小节的详细描述。

表 13-157 WAIT_EVENTS 字段

名称	类型	描述
nodename	text	数据库进程名称。
type	text	event类型。
event	text	event名称。
wait	bigint	等待次数。
failed_wait	bigint	失败的等待次数。
total_wait_time	bigint	总等待时间（单位：微秒）。
avg_wait_time	bigint	平均等待时间（单位：微秒）。
max_wait_time	bigint	最大等待时间（单位：微秒）。
min_wait_time	bigint	最小等待时间（单位：微秒）。
last_updated	timestamp with time zone	最后一次更新该事件的时间。

13.2.13.2 GLOBAL_WAIT_EVENTS

GLOBAL_WAIT_EVENTS视图显示各节点的event的等待相关的统计信息。

表 13-158 GLOBAL_WAIT_EVENTS 字段

名称	类型	描述
nodename	text	数据库进程名称。
type	text	event类型。
event	text	event名称。
wait	bigint	等待次数。
failed_wait	bigint	失败的等待次数。
total_wait_time	bigint	总等待时间（单位：微秒）。
avg_wait_time	bigint	平均等待时间（单位：微秒）。
max_wait_time	bigint	最大等待时间（单位：微秒）。
min_wait_time	bigint	最小等待时间（单位：微秒）。
last_updated	timestamp with time zone	最后一次更新该事件的时间。

13.2.14 Configuration

13.2.14.1 CONFIG_SETTINGS

CONFIG_SETTINGS视图显示数据库运行时参数的相关信息。

表 13-159 CONFIG_SETTINGS 字段

名称	类型	描述
name	text	参数名称。
setting	text	参数当前值。
unit	text	参数的隐式结构。
category	text	参数的逻辑组。
short_desc	text	参数的简单描述。
extra_desc	text	参数的详细描述。
context	text	设置参数值的上下文，包括internal，postmaster，sighup，backend，superuser，user。
vartype	text	参数类型，包括bool，enum，integer，real，string。
source	text	参数的赋值方式。
min_val	text	参数最大值。如果参数类型不是数值型，那么该字段值为null。
max_val	text	参数最小值。如果参数类型不是数值型，那么该字段值为null。
enumvals	text[]	enum类型参数合法值。如果参数类型不是enum型，那么该字段值为null。
boot_val	text	数据库启动时参数默认值。
reset_val	text	数据库重置时参数默认值。
sourcefile	text	设置参数值的配置文件。如果参数不是通过配置文件赋值，那么该字段值为null。
sourceline	integer	设置参数值的配置文件的行号。如果参数不是通过配置文件赋值，那么该字段值为null。

13.2.14.2 GLOBAL_CONFIG_SETTINGS

GLOBAL_CONFIG_SETTINGS显示各节点数据库运行时参数的相关信息。

表 13-160 GLOBAL_CONFIG_SETTINGS 的字段

名称	类型	描述
node_name	text	节点名称。
name	text	参数名称。
setting	text	参数当前值。
unit	text	参数的隐式结构。
category	text	参数的逻辑组。
short_desc	text	参数的简单描述。
extra_desc	text	参数的详细描述。
context	text	设置参数值的上下文，包括internal，postmaster，sighup，backend，superuser，user。
vartype	text	参数类型，包括bool，enum，integer，real，string。
source	text	参数的赋值方式。
min_val	text	参数最小值。如果参数类型不是数值型，那么该字段值为null。
max_val	text	参数最大值。如果参数类型不是数值型，那么该字段值为null。
enumvals	text[]	enum类型参数合法值。如果参数类型不是enum型，那么该字段值为null。
boot_val	text	数据库启动时参数默认值。
reset_val	text	数据库重置时参数默认值。
sourcefile	text	设置参数值的配置文件。如果参数不是通过配置文件赋值，那么该字段值为null。
sourceline	integer	设置参数值的配置文件的行号。如果参数不是通过配置文件赋值，那么该字段值为null。

13.2.15 Operator

13.2.15.1 OPERATOR_HISTORY_TABLE

OPERATOR_HISTORY_TABLE系统视图显示执行作业结束后的算子相关的记录。此数据是从内核中转储到系统视图中的数据。

表 13-161 OPERATOR_HISTORY_TABLE 的字段

名称	类型	描述
queryid	bigint	语句执行使用的内部query_id。
pid	bigint	后端线程id。
plan_node_id	integer	查询对应的执行计划的plan node id。
plan_node_name	text	对应于plan_node_id的算子的名称。
start_time	timestamp with time zone	该算子处理第一条数据的开始时间。
duration	bigint	该算子到结束时候总的执行时间（ms）。
query_dop	integer	当前算子执行时的并行度。
estimated_rows	bigint	优化器估算的行数信息。
tuple_processed	bigint	当前算子返回的元素个数。
min_peak_memory	integer	当前算子在数据库节点上的最小内存峰值（MB）。
max_peak_memory	integer	当前算子在数据库节点上的最大内存峰值（MB）。
average_peak_memory	integer	当前算子在数据库节点上的平均内存峰值（MB）。
memory_skew_percent	integer	当前算子在数据库节点间的内存使用倾斜率。
min_spill_size	integer	若发生下盘，数据库节点上下盘的最小数据量（MB），默认为0。
max_spill_size	integer	若发生下盘，数据库节点上下盘的最大数据量（MB），默认为0。
average_spill_size	integer	若发生下盘，数据库节点上下盘的平均数据量（MB)，默认为0。
spill_skew_percent	integer	若发生下盘，数据库节点间下盘倾斜率。
min_cpu_time	bigint	该算子在数据库节点上的最小执行时间（ms）。
max_cpu_time	bigint	该算子在数据库节点上的最大执行时间（ms）。
total_cpu_time	bigint	该算子在数据库节点上的总执行时间（ms）。
cpu_skew_percent	integer	数据库节点间执行时间的倾斜率。

名称	类型	描述
warning	text	主要显示如下几类告警信息： • Sort/SetOp/HashAgg/HashJoin spill • Spill file size large than 256MB • Broadcast size large than 100MB • Early spill • Spill times is greater than 3 • Spill on memory adaptive • Hash table conflict

13.2.15.2 OPERATOR_HISTORY

OPERATOR_HISTORY视图显示的是当前用户数据库主节点上执行作业结束后的算子的相关记录。

13.2.15.3 OPERATOR_RUNTIME

OPERATOR_RUNTIME视图显示当前用户正在执行的作业的算子相关信息。

表 13-162 OPERATOR_RUNTIME 的字段

名称	类型	描述
queryid	bigint	语句执行使用的内部query_id。
pid	bigint	后端线程id。
plan_node_id	integer	查询对应的执行计划的plan node id。
plan_node_name	text	对应于plan_node_id的算子的名称。
start_time	timestamp with time zone	该算子处理第一条数据的开始时间。
duration	bigint	该算子到结束时候总的执行时间（ms）。
status	text	当前算子的执行状态，包括finished和running。
query_dop	integer	当前算子执行时的并行度。
estimated_rows	bigint	优化器估算的行数信息。
tuple_processed	bigint	当前算子返回的元素个数。
min_peak_memory	integer	当前算子在数据库节点上的最小内存峰值（MB）。

名称	类型	描述
max_peak_memory	integer	当前算子在数据库节点上的最大内存峰值（MB）。
average_peak_memory	integer	当前算子在数据库节点上的平均内存峰值（MB)。
memory_skew_percent	integer	当前算子在数据库节点的内存使用倾斜率。
min_spill_size	integer	若发生下盘，数据库节点上下盘的最小数据量（MB），默认为0。
max_spill_size	integer	若发生下盘，数据库节点上下盘的最大数据量（MB），默认为0。
average_spill_size	integer	若发生下盘，数据库节点上下盘的平均数据量（MB)，默认为0。
spill_skew_percent	integer	若发生下盘，数据库节点间下盘倾斜率。
min_cpu_time	bigint	该算子在数据库节点上的最小执行时间（ms)。
max_cpu_time	bigint	该算子在数据库节点上的最大执行时间（ms)。
total_cpu_time	bigint	该算子在数据库节点上的总执行时间（ms)。
cpu_skew_percent	integer	数据库节点间执行时间的倾斜率。
warning	text	主要显示如下几类告警信息： ● Sort/SetOp/HashAgg/HashJoin spill ● Spill file size large than 256MB ● Broadcast size large than 100MB ● Early spill ● Spill times is greater than 3 ● Spill on memory adaptive ● Hash table conflict

13.2.15.4 GLOBAL_OPERATOR_HISTORY

GLOBAL_OPERATOR_HISTORY系统视图显示的是当前用户在数据库主节点上执行作业结束后的算子的相关记录。

表 13-163 GLOBAL_OPERATOR_HISTORY 的字段

名称	类型	描述
queryid	bigint	语句执行使用的内部query_id。

名称	类型	描述
pid	bigint	后端线程id。
plan_node_id	integer	查询对应的执行计划的plan node id。
plan_node_name	text	对应于plan_node_id的算子的名称。
start_time	timestamp with time zone	该算子处理第一条数据的开始时间。
duration	bigint	该算子到结束时候总的执行时间（ms）。
query_dop	integer	当前算子执行时的并行度。
estimated_rows	bigint	优化器估算的行数信息。
tuple_processed	bigint	当前算子返回的元素个数。
min_peak_memory	integer	当前算子在数据库节点上的最小内存峰值（MB）。
max_peak_memory	integer	当前算子在数据库节点上的最大内存峰值（MB）。
average_peak_memory	integer	当前算子在数据库节点上的平均内存峰值（MB）。
memory_skew_percent	integer	当前算子在数据库节点间的内存使用倾斜率。
min_spill_size	integer	若发生下盘，数据库节点上下盘的最小数据量（MB），默认为0。
max_spill_size	integer	若发生下盘，数据库节点上下盘的最大数据量（MB），默认为0。
average_spill_size	integer	若发生下盘，数据库节点上下盘的平均数据量（MB)，默认为0。
spill_skew_percent	integer	若发生下盘，数据库节点间下盘倾斜率。
min_cpu_time	bigint	该算子在数据库节点上的最小执行时间（ms）。
max_cpu_time	bigint	该算子在数据库节点上的最大执行时间（ms）。
total_cpu_time	bigint	该算子在数据库节点上的总执行时间（ms）。
cpu_skew_percent	integer	数据库节点间执行时间的倾斜率。

名称	类型	描述
warning	text	主要显示如下几类告警信息： 1. Sort/SetOp/HashAgg/HashJoin spill 2. Spill file size large than 256MB 3. Broadcast size large than 100MB 4. Early spill 5. Spill times is greater than 3 6. Spill on memory adaptive 7. Hash table conflict

13.2.15.5 GLOBAL_OPERATOR_HISTORY_TABLE

GLOBAL_OPERATOR_HISTORY_TABLE视图显示数据库主节点执行作业结束后的算子相关的记录。此数据是从内核中转储到系统表GS_WLM_OPERATOR_INFO中的数据。该视图是查询数据库主节点系统表GS_WLM_OPERATOR_INFO的汇聚视图。表字段同**表13-163**。

13.2.15.6 GLOBAL_OPERATOR_RUNTIME

GLOBAL_OPERATOR_RUNTIME视图显示当前用户在数据库主节点上正在执行的作业的算子相关信息。

表 13-164 GLOBAL_OPERATOR_RUNTIME 的字段

名称	类型	描述
queryid	bigint	语句执行使用的内部query_id。
pid	bigint	后端线程id。
plan_node_id	integer	查询对应的执行计划的plan node id。
plan_node_name	text	对应于plan_node_id的算子的名称。
start_time	timestamp with time zone	该算子处理第一条数据的开始时间。
duration	bigint	该算子到结束时候总的执行时间（ms）。
status	text	当前算子的执行状态，包括finished和running。
query_dop	integer	当前算子执行时的并行度。
estimated_rows	bigint	优化器估算的行数信息。
tuple_processed	bigint	当前算子返回的元素个数。

名称	类型	描述
min_peak_memory	integer	当前算子在数据库节点上的最小内存峰值（MB）。
max_peak_memory	integer	当前算子在数据库节点上的最大内存峰值（MB）。
average_peak_memory	integer	当前算子在数据库节点上的平均内存峰值（MB）。
memory_skew_percent	integer	当前算子在数据库节点的内存使用倾斜率。
min_spill_size	integer	若发生下盘，数据库节点上下盘的最小数据量（MB），默认为0。
max_spill_size	integer	若发生下盘，数据库节点上下盘的最大数据量（MB），默认为0。
average_spill_size	integer	若发生下盘，数据库节点上下盘的平均数据量（MB)，默认为0。
spill_skew_percent	integer	若发生下盘，数据库节点间下盘倾斜率。
min_cpu_time	bigint	该算子在数据库节点上的最小执行时间（ms）。
max_cpu_time	bigint	该算子在数据库节点上的最大执行时间（ms）。
total_cpu_time	bigint	该算子在数据库节点上的总执行时间（ms）。
cpu_skew_percent	integer	数据库节点间执行时间的倾斜率。
warning	text	主要显示如下几类告警信息： ● Sort/SetOp/HashAgg/HashJoin spill ● Spill file size large than 256MB ● Broadcast size large than 100MB ● Early spill ● Spill times is greater than 3 ● Spill on memory adaptive ● Hash table conflict

13.2.16 Workload Manager

13.2.16.1 WLM_USER_RESOURCE_CONFIG

WLM_USER_RESOURCE_CONFIG视图显示用户的资源配置信息。

表 13-165 WLM_USER_RESOURCE_CONFIG 字段

名称	类型	描述
userid	oid	用户oid。
username	name	用户名称。
sysadmin	boolean	是否是sysadmin。
rpoid	oid	资源池的oid。
respool	name	资源池的名称。
parentid	oid	父用户的oid。
totalspace	bigint	占用总空间大小。
spacelimit	bigint	空间大上限。
childcount	integer	子用户数量。
childlist	text	子用户的列表。

13.2.16.2 WLM_USER_RESOURCE_RUNTIME

WLM_USER_RESOURCE_RUNTIME视图显示所有用户资源使用情况，需要使用管理员用户进行查询。此视图在GUC参数“use_workload_manager”为“on”时才有效。

表 13-166 WLM_USER_RESOURCE_RUNTIME 字段

名称	类型	描述
username	name	用户名。
used_memory	integer	正在使用的内存大小，单位MB。
total_memory	integer	可以使用的内存大小，单位MB。值为0表示未限制最大可用内存，其限制取决于数据库最大可用内存。
used_cpu	integer	正在使用的CPU核数。
total_cpu	integer	在该机器节点上，用户关联控制组的CPU核数总和。
used_space	bigint	已使用的存储空间大小，单位KB。
total_space	bigint	可使用的存储空间大小，单位KB。值为-1表示未限制最大存储空间。
used_temp_space	bigint	已使用的临时空间大小（预留字段，暂未使用），单位KB。
total_temp_space	bigint	可使用的临时空间大小（预留字段，暂未使用），单位KB。值为-1表示未限制最大临时存储空间。

名称	类型	描述
used_spill_space	bigint	已使用的下盘空间大小（预留字段，暂未使用），单位KB。
total_spill_space	bigint	可使用的下盘空间大小（预留字段，暂未使用），单位KB。值为-1表示未限制最大下盘空间。

13.2.17 Global Plancache

GPC相关视图在enable_global_plancache打开的状态下才有效。

13.2.17.1 GLOBAL_PLANCACHE_STATUS

GLOBAL_PLANCACHE_STATUS视图显示GPC全局计划缓存状态信息。

表 13-167 GLOBAL_PLANCACHE_STATUS 字段

名称	类型	描述
nodename	text	所属节点名称。
query	text	查询语句text。
refcount	integer	被引用次数。
valid	bool	是否合法。
databaseid	oid	所属数据库id。
schema_name	text	所属schema。
params_num	integer	参数数量。
func_id	oid	该plancache所在存储过程oid，如果不属于存储过程则为0。
pkg_id	oid	该plancache所在存储过程所属的Package，如果不属于Package则为0。
stmt_id	integer	显示存储过程内语句计划的序号。

13.2.17.2 GLOBAL_PLANCACHE_CLEAN

GLOBAL_PLANCACHE_CLEAN视图用于清理所有节点上无人使用的全局计划缓存。返回值为Boolean类型。

13.2.18 RTO & RPO

13.2.18.1 global_rto_status

global_rto_status视图显示关于主机和备机的日志流控信息（本节点除外、备DN上不可使用）。

表 13-168 global_rto_status 字段

参数	类型	描述
node_name	text	节点的名称，包含主机和备机。
rto_info	text	流控的信息，包含了备机当前的日志流控时间（单位：秒），备机通过GUC参数设置的预期流控时间（单位：秒），为了达到这个预期主机所需要的睡眠时间（单位：微秒）。

13.2.18.2 global_streaming_hadr_rto_and_rpo_stat

global_streaming_hadr_rto_and_rpo_stat视图显示流式容灾的主数据库实例和备数据库实例日志流控信息（只可在主数据库实例的主DN使用，备DN以及备数据库实例上均不可获取到统计信息）。

表 13-169 global_streaming_hadr_rto_and_rpo_stat 参数说明

参数	类型	描述
hadr_sender_node_name	text	节点的名称，包含主数据库实例和备数据库实例首备。
hadr_receiver_node_name	text	备数据库实例首备名称。
current_rto	int	流控的信息，当前主备数据库实例的日志rto时间（单位：秒）。
target_rto	int	流控的信息，目标主备数据库实例间的rto时间（单位：秒）。
current_rpo	int	流控的信息，当前主备数据库实例的日志rpo时间（单位：秒）。
target_rpo	int	流控的信息，目标主备数据库实例间的rpo时间（单位：秒）。
rto_sleep_time	int	RTO流控信息，为了达到目标rto，预期主机walsender所需要的睡眠时间（单位：微秒）。
rpo_sleep_time	int	RPO流控信息，为了达到目标rpo，预期主机xlogInsert所需要的睡眠时间（单位：微秒）。

13.2.19 AI Watchdog

13.2.19.1 ai_watchdog_monitor_status

表 13-170 ai_watchdog_monitor_status 参数说明

参数	类型	描述
metric_name	text	metric指标名称： ● tps：TPS。 ● tps_hourly：每小时的TPS均值。 ● shared_used_mem：共享内存使用量（MB）。 ● dynamic_used_shrctx：共享内存上下文使用量（MB）。 ● other_used_mem：其他内存使用量（MB）。 ● process_used_mem：系统常驻内存使用量（MB）。 ● dynamic_used_mem：动态内存使用量（MB）。 ● malloc_failures：每个采集间隔内的内存分配失败次数。 ● D_state_rate：D状态线程比例。 ● R_state_rate：R状态线程比例。 ● S_state_rate：S状态线程比例。 ● db_state：数据库的状态（68表示D、82表示R、83表示S）。 ● cpu_usage：CPU使用率，上限100。 ● disk_io：两个采集间隔内的磁盘IO延迟。 ● network_io：两个采集间隔内的网络IO延迟。 ● threadpool_usage：线程池使用率。 ● threadpool_hang_rate：线程池group处于hang状态的比例。
max_length	int	采集队列长度。
current_length	int	当前采集到的样本数。
collection_interval	int	采集间隔，单位秒。
latest_value	int	上次采集到的值，没采集到为null。
last_report	timestamp	上次采集时刻。

13.2.19.2 ai_watchdog_detection_warnings

表 13-171 ai_watchdog_detection_warnings 参数

参数	类型	描述
event	text	事件名称。
cause	text	事件原因。
details	text	事件详情。
time	timestamp	报告时刻。
need_to_handle	bool	是否需要自动处理。

13.2.19.3 ai_watchdog_parameters

表 13-172 ai_watchdog_parameters 参数

参数	类型	描述
name	text	参数名称，包括如下常用参数： ● enable_ai_watchdog：是否开启本功能。 ● ai_watchdog_max_consuming_time_ms：最大耗时。 ● ai_watchdog_used_memory_kb：本功能当前内存使用。 ● ai_watchdog_detection_times：已检测次数。 ● enable_self_healing：发现问题后是否可以自愈。 ● oom_detected_times：已检测到的OOM次数。 ● hang_detected_times：已检测到的hang次数。 ● enable_oom_detection：是否自动启动了OOM探测功能。 ● in_wait_time：是否处于等待时间。 ● other_used_memory_has_risk：其他内存使用部分是否存在风险。 ● shared_used_mem_has_risk：共享内存上下文使用是否存在风险。 ● dynamic_used_shrctx_has_risk：动态内存使用是存在风险。

参数	类型	描述
value	text	参数值

13.2.20 废弃

13.2.20.1 Query

13.2.20.1.1 GS_SLOW_QUERY_INFO

GS_SLOW_QUERY_INFO视图显示当前节点上已经转储的慢查询信息。此数据是从内核中转储到系统表中的数据。当设置GUC参数**enable_resource_record**为on时，系统会定时（周期为3分钟）将内核中query信息导入GS_WLM_SESSION_QUERY_INFO_ALL系统表，开启此功能会占用系统存储空间并对性能有一定影响。用户通过查询GS_SLOW_QUERY_INFO视图，可以查看已经转储的慢查询信息，本版本中已废弃。

表 13-173 GS_SLOW_QUERY_INFO 字段

名称	类型	描述
dbname	text	数据库名称。
schemaname	text	schema名称。
nodename	text	节点名称。
username	text	用户名。
queryid	bigint	归一化ID。
query	text	query语句。
start_time	timestamp with time zone	开始执行时间。
finish_time	timestamp with time zone	结束执行时间。
duration	bigint	执行持续时间（毫秒）。
query_plan	text	计划信息。
n_returned_rows	bigint	Select返回的结果集行数。
n_tuples_fetched	bigint	随机扫描行数。
n_tuples_returned	bigint	顺序扫描行数。
n_tuples_inserted	bigint	插入行数。
n_tuples_updated	bigint	更新行数。

名称	类型	描述
n_tuples_deleted	bigint	删除行数。
n_blocks_fetched	bigint	Cache加载次数。
n_blocks_hit	bigint	Cache命中数。
db_time	bigint	有效的DB时间花费，多线程将累加（单位：微秒）。
cpu_time	bigint	CPU时间（单位：微秒）。
execution_time	bigint	执行器内执行时间（单位：微秒）。
parse_time	bigint	SQL解析时间（单位：微秒）。
plan_time	bigint	SQL生成计划时间（单位：微秒）。
rewrite_time	bigint	SQL重写时间（单位：微秒）。
pl_execution_time	bigint	plpgsql上的执行时间（单位：微秒）。
pl_compilation_time	bigint	plpgsql上的编译时间（单位：微秒）。
net_send_time	bigint	网络上的时间花费（单位：微秒）。
data_io_time	bigint	IO上的时间花费(单位：微秒)。

13.2.20.1.2 GS_SLOW_QUERY_HISTORY

GS_SLOW_QUERY_HISTORY显示当前节点上未转储的慢查询信息。具体字段信息请参考GS_SLOW_QUERY_INFO。本版本中已废弃。

13.2.20.1.3 GLOBAL_SLOW_QUERY_HISTORY

GS_SLOW_QUERY_HISTORY显示所有节点上未转储的慢查询信息，本版本中已废弃。具体字段信息请参考GS_SLOW_QUERY_INFO。

13.2.20.1.4 GLOBAL_SLOW_QUERY_INFO

GS_SLOW_QUERY_HISTORY显示所有节点上已经转储的慢查询信息，本版本中已废弃。具体字段信息请参考GS_SLOW_QUERY_INFO。

13.3 DBE_PLDEBUGGER Schema

DBE_PLDEBUGGER Schema下的系统函数用于调试存储过程，目前支持的接口及其描述如下所示。仅管理员有权限执行这些调试接口，但无权限在该schema上修改和创建新函数。普通用户只能调试在public schema或用户创建schema下的非系统函数，禁止普通用户调试系统函数。

须知

当在函数体中创建用户时，调用attach、next、continue、 info_code、step、info_breakpoint、backtrace、 finish中会返回密码的明文。因此不建议用户在函数体中创建用户。

对应权限角色为gs_role_pldebugger，可以由管理员用户通过如下命令将debugger权限赋权给该用户。

```
GRANT gs_role_pldebugger to user;
```

需要有两个客户端连接数据库，一个客户端负责执行调试接口作为debug端，另一个客户端执行调试函数，控制server端存储过程执行。示例如下。

- 准备调试

  通过PG_PROC，查找到待调试存储过程的oid，并执行DBE_PLDEBUGGER.turn_on(oid)。本客户端就会作为server端使用。

  ```
  gaussdb=# CREATE OR REPLACE PROCEDURE test_debug ( IN  x INT)
  AS
  BEGIN
      INSERT INTO t1 (a) VALUES (x);
      DELETE FROM t1 WHERE a = x;
  END;
  /
  CREATE PROCEDURE
  gaussdb=# SELECT OID FROM PG_PROC WHERE PRONAME='test_debug';
    oid
  -------
   16389
  (1 row)
  gaussdb=# SELECT * FROM DBE_PLDEBUGGER.turn_on(16389);
   nodename | port
  ----------+------
   datanode |    0
  (1 row)
  ```

- 开始调试

  server端执行存储过程，会在存储过程内第一条SQL语句前hang住，等待debug端发送的调试消息。仅支持直接执行存储过程的调试，不支持通过trigger调用执行的存储过程调试。

  ```
  gaussdb=# call test_debug(1);
  ```

  再起一个客户端，作为debug端，通过turn_on返回的数据，调用DBE_PLDEBUGGER.attach关联到该存储过程上进行调试。

  ```
  gaussdb=# SELECT * FROM DBE_PLDEBUGGER.attach('datanode',0);
   funcoid |  funcname  | lineno |              query
  ---------+------------+--------+---------------------------------
    16389 | test_debug |      3 |   INSERT INTO t1 (a) VALUES (x);
  (1 row)
  ```

  在执行attach的客户端调试，执行下一条statement。


```
gaussdb=# SELECT * FROM DBE_PLDEBUGGER.next();
 funcoid |  funcname  | lineno |        query
---------+------------+--------+----------------------
   16389 | test_debug |      0 | [EXECUTION FINISHED]
(1 row)
```

在执行attach的客户端调试，可以执行以下变量操作

```
gaussdb=# SELECT * FROM DBE_PLDEBUGGER.info_locals(); --打印全部变量
 varname | vartype | value | package_name | isconst
---------+---------+-------+--------------+---------
 x       | int4    | 1     |              | f
(1 row)
gaussdb=# SELECT * FROM DBE_PLDEBUGGER.set_var('x', 2); --变量赋值
 set_var
---------
 t
(1 row)
gaussdb=# SELECT * FROM DBE_PLDEBUGGER.print_var('x'); --打印单个变量
 varname | vartype | value | package_name | isconst
---------+---------+-------+--------------+---------
 x       | int4    | 2     |              | f
(1 row)
```

直接执行完成当前正在调试的存储过程。

```
gaussdb=# SELECT * FROM DBE_PLDEBUGGER.continue();
 funcoid |  funcname  | lineno |        query
---------+------------+--------+----------------------
   16389 | test_debug |      0 | [EXECUTION FINISHED]
(1 row)
```

直接退出当前正在调试的存储过程，不执行尚未执行的语句。

```
gaussdb=# SELECT * FROM DBE_PLDEBUGGER.abort();
 abort
-------
 t
(1 row)
```

client端查看代码信息并识别可以设置断点行号。

```
gaussdb=# SELECT * FROM DBE_PLDEBUGGER.info_code(16389);
 lineno |                          query                          | canbreak
--------+---------------------------------------------------------+----------
        | CREATE OR REPLACE PROCEDURE public.test_debug( IN  x INT) | f
      1 | AS  DECLARE                                               | f
      2 | BEGIN                                                     | f
      3 |     INSERT INTO t1 (a) VALUES (x);                        | t
      4 |     DELETE FROM t1 WHERE a = x;                           | t
      5 | END;                                                      | f
      6 | /                                                         | f
(7 rows)
```

设置断点。

```
gaussdb=# SELECT * FROM DBE_PLDEBUGGER.add_breakpoint(16389,4);
 breakpointno
--------------
            0
(1 row)
```

查看断点信息。

```
gaussdb=# SELECT * FROM DBE_PLDEBUGGER.info_breakpoints();
 breakpointno | funcoid | lineno |              query               | enable
--------------+---------+--------+----------------------------------+--------
            0 |   16389 |      4 |     DELETE FROM t1 WHERE a = x; | t
(1 row)
```

执行至断点。

```
gaussdb=# SELECT * FROM DBE_PLDEBUGGER.continue();
 funcoid |  funcname  | lineno |              query
---------+------------+--------+----------------------------------
```



```
  16389 | test_debug |     4 |     DELETE FROM t1 WHERE a = x;
(1 row)
```

存储过程执行结束后，调试会自动退出，再进行调试需要重新attach关联。如果server端不需要继续调试，可执行turn_off关闭，或退出session。具体调试接口请见下面列表。

表 13-174 DBE_PLDEBUGGER

接口名称	描述
DBE_PLDEBUGGER.turn_on	server端调用，标记存储过程可以调试，调用后执行该存储过程时会hang住等待调试信息。
DBE_PLDEBUGGER.turn_off	server端调用，标记存储过程关闭调试。
DBE_PLDEBUGGER.local_debug_server_info	server端调用，打印本session内所有已turn_on的存储过程。
DBE_PLDEBUGGER.attach	debug端调用，关联到正在调试存储过程。
DBE_PLDEBUGGER.info_locals	debug端调用，打印正在调试的存储过程中的变量当前值。
DBE_PLDEBUGGER.next	debug端调用，单步执行。
DBE_PLDEBUGGER.continue	debug端调用，继续执行，直到断点或存储过程结束。
DBE_PLDEBUGGER.abort	debug端调用，停止调试，server端报错长跳转。
DBE_PLDEBUGGER.print_var	debug端调用，打印正在调试的存储过程中指定的变量当前值。
DBE_PLDEBUGGER.info_code	debug和server端都可以调用，打印指定存储过程的源语句和各行对应的行号。
DBE_PLDEBUGGER.step	debug端调用，单步进入执行。
DBE_PLDEBUGGER.add_breakpoint	debug端调用，新增断点。
DBE_PLDEBUGGER.delete_breakpoint	debug端调用，删除断点。
DBE_PLDEBUGGER.info_breakpoints	debug端调用，查看当前的所有断点。
DBE_PLDEBUGGER.backtrace	debug端调用，查看当前的调用栈。
DBE_PLDEBUGGER.enable_breakpoint	debug端调用，激活被禁用的断点。
DBE_PLDEBUGGER.disable_breakpoint	debug端调用，禁用已激活的断点。

接口名称	描述
DBE_PLDEBUGGER.finish	debug端调用，继续调试，直到断点或返回上一层调用栈。
DBE_PLDEBUGGER.set_var	debug端调用，为变量进行赋值操作。

13.3.1 DBE_PLDEBUGGER.turn_on

该函数用于标记某一存储过程为可调试，执行turn_on后server端可以执行该存储过程来进行调试。需要用户根据系统表PG_PROC手动获取存储过程oid，传入函数中。turn_on后本session内执行该存储过程会停在第一条sql前等待debug端的调试操作。该设置会在session断连后默认被清理掉。目前不支持对启用自治事务的存储过程/函数进行调试。

函数原型为：

```
DBE_PLDEBUGGER.turn_on(Oid)
RETURN Record;
```

表 13-175 turn_on 入参和返回值列表

名称	类型	描述
func_oid	IN oid	函数oid
nodename	OUT text	节点名称
port	OUT integer	连接端口号

13.3.2 DBE_PLDEBUGGER.turn_off

用于去掉turn_on添加的调试标记，返回值表示成功或失败。可通过DBE_PLDEBUGGER.local_debug_server_info查找已经turn_on的存储过程oid。

函数原型为：

```
DBE_PLDEBUGGER.turn_off(Oid)
RETURN boolean;
```

表 13-176 turn_off 入参和返回值列表

名称	类型	描述
func_oid	IN oid	函数oid
turn_off	OUT boolean	turn off是否成功

13.3.3 DBE_PLDEBUGGER.local_debug_server_info

用于查找当前连接中已经turn_on的存储过程oid。便于用户确认在调试哪些存储过程，需要通过funcoid和pg_proc配合使用。

表 13-177 local_debug_server_info 返回值列表

名称	类型	描述
nodename	OUT text	节点名称
port	OUT bigint	端口号
funcoid	OUT oid	存储过程oid

13.3.4 DBE_PLDEBUGGER.attach

server端执行存储过程，停在第一条语句前，等待debug端关联。debug端调用attach，传入nodename和port，关联到该存储过程上。

如果调试过程中报错，attach会自动失效；如果调试过程中attach到其他存储过程上，当前attach的调试也会失效。

表 13-178 attach 入参和返回值列表

名称	类型	描述
nodename	IN text	节点名称
port	IN integer	连接端口号
funcoid	OUT oid	函数id
funcname	OUT text	函数名
lineno	OUT integer	当前调试运行的下一行行号
query	OUT text	当前调试的下一行函数源码

13.3.5 DBE_PLDEBUGGER.info_locals

debug端调试过程中，调用info_locals，打印当前存储过程内变量。该函数入参frameno表示查询遍历的栈层数，支持无入参调用，缺省为查看最上层栈变量。

表 13-179 info_locals 入参和返回值列表

名称	类型	描述
frameno	IN integer（可选）	指定的栈层数，缺省为最顶层
varname	OUT text	变量名
vartype	OUT text	变量类型
value	OUT text	变量值
package_name	OUT text	变量对应的package名，非package时为空
isconst	OUT boolean	是否为常量

13.3.6 DBE_PLDEBUGGER.next

执行存储过程中当前的sql，返回执行的下一条的行数和对应query。

表 13-180 next 返回值列表

名称	类型	描述
funcoid	OUT oid	函数id
funcname	OUT text	函数名
lineno	OUT integer	当前调试运行的下一行行号
query	OUT text	当前调试的下一行函数源码

13.3.7 DBE_PLDEBUGGER.continue

执行当前存储过程，直到下一个断点或结束，返回执行的下一条的行数和对应query。

函数原型为：

```
DBE_PLDEBUGGER.continue()
RETURN Record;
```

表 13-181 continue 返回值列表

名称	类型	描述
funcoid	OUT oid	函数id
funcname	OUT text	函数名

名称	类型	描述
lineno	OUT integer	当前调试运行的下一行行号
query	OUT text	当前调试的下一行函数源码

13.3.8 DBE_PLDEBUGGER.abort

令server端执行的存储过程报错跳出。返回值表示是否成功发送abort。

函数原型为：

```
DBE_PLDEBUGGER.abort()
RETURN boolean;
```

表 13-182 abort 返回值列表

名称	类型	描述
abort	OUT boolean	表示成功或失败

13.3.9 DBE_PLDEBUGGER.print_var

debug端调试过程中，调用print_var，打印当前存储过程内变量中指定的变量名及其取值。该函数入参frameno表示查询遍历的栈层数，支持不加入该参数调用，缺省为查看最上层栈变量。

表 13-183 print_var 入参和返回值列表

名称	类型	描述
var_name	IN text	变量
frameno	IN integer（可选）	指定的栈层数，缺省为最顶层
varname	OUT text	变量名
vartype	OUT text	变量类型
value	OUT text	变量值
package_name	OUT text	变量对应的package名，预留使用，当前均为空
isconst	OUT boolean	是否为常量

13.3.10 DBE_PLDEBUGGER.info_code

debug端调试过程中，调用info_code，查看指定存储过程的源语句和各行对应的行号，行号从函数体开始，函数头部分行号为空。

表 13-184 info_code 入参和返回值列表

名称	类型	描述
funcoid	IN oid	函数ID
lineno	OUT integer	行号
query	OUT text	源语句
canbreak	OUT bool	当前行是否支持断点

13.3.11 DBE_PLDEBUGGER.step

debug端调试过程中，如果当前执行的是一个存储过程，则进入该存储过程继续调试，返回该存储过程第一行的行号等信息，如果当前执行的不是存储过程，则和next行为一致，执行该sql后返回下一行的行号等信息。

表 13-185 step 入参和返回值列表

名称	类型	描述
funcoid	OUT oid	函数ID
funcname	OUT text	函数名
lineno	OUT integer	当前调试运行的下一行行号
query	OUT text	当前调试的下一行函数源码

13.3.12 DBE_PLDEBUGGER.add_breakpoint

debug端调试过程中，调用add_breakpoint增加新的断点。如果返回-1则说明指定的断点不合法，请参考**DBE_PLDEBUGGER.info_code**的canbreak字段确定断点合适的位置。

表 13-186 add_breakpoint 入参和返回值列表

名称	类型	描述
funcoid	IN text	函数ID
lineno	IN integer	行号

名称	类型	描述
breakpointno	OUT integer	断点编号

13.3.13 DBE_PLDEBUGGER.delete_breakpoint

debug端调试过程中，调用delete_breakpoint删除已有的断点。

表 13-187 delete_breakpoint 入参和返回值列表

名称	类型	描述
breakpointno	IN integer	断点编号
result	OUT bool	是否成功

13.3.14 DBE_PLDEBUGGER.info_breakpoints

debug端调试过程中，调用info_breakpoints，查看当前的函数断点。

表 13-188 info_breakpoints 返回值列表

名称	类型	描述
breakpointno	OUT integer	断点编号
funcoid	OUT oid	函数ID
lineno	OUT integer	行号
query	OUT text	断点内容
enable	OUT boolean	是否有效

13.3.15 DBE_PLDEBUGGER.backtrace

debug端调试过程中，调用backtrace，查看当前的调用堆栈。

表 13-189 backtrace 返回值列表

名称	类型	描述
frameno	OUT integer	调用栈编号
funcname	OUT text	函数名
lineno	OUT integer	行号
query	OUT text	断点内容

名称	类型	描述
funcoid	OUT oid	函数oid

13.3.16 DBE_PLDEBUGGER.enable_breakpoint

debug端调试过程中，调用enable_breakpoint激活已被禁用的断点。

表 13-190 enable_breakpoint 入参和返回值列表

名称	类型	描述
breakpointno	IN integer	断点编号
result	OUT bool	是否成功

13.3.17 DBE_PLDEBUGGER.disable_breakpoint

debug端调试过程中，调用disable_breakpoint禁用已被激活的断点。

表 13-191 disable_breakpoint 入参和返回值列表

名称	类型	描述
breakpointno	IN integer	断点编号
result	OUT bool	是否成功

13.3.18 DBE_PLDEBUGGER.finish

执行存储过程中当前的SQL直到下一个断点触发或执行到上层栈的下一行。

表 13-192 finish 入参和返回值列表

名称	类型	描述
funcoid	OUT oid	函数id
funcname	OUT text	函数名
lineno	OUT integer	当前调试运行的下一行行号
query	OUT text	当前调试的下一行函数源码

13.3.19 DBE_PLDEBUGGER.set_var

将指定的调试的存储过程中最上层栈上的变量修改为入参的取值。如果存储过程中包含同名的变量，set_var只支持第一个变量值的设置。

表 13-193 set_var 入参和返回值列表

名称	类型	描述
var_name	IN text	变量名
value	IN text	修改值
result	OUT boolean	结果，是否成功

13.4 DB4AI Schema

DB4AI模式在AI特性中主要是用来存储和管理数据集版本。模式中保存数据表的原始视图快照，以及每一个数据版本的更改记录和版本快照的管理信息。模式面向普通用户，用户可在该模式下查找特性DB4AI.SNAPSHOT创建的快照版本信息。

13.4.1 DB4AI.SNAPSHOT

SNAPSHOT表记录当前用户通过特性DB4AI.SNAPSHOT存储的快照。

表 13-194 db4ai.snapshot 表属性

名称	类型	描述	实例
id	bigint	当前快照的ID。	1
parent_id	bigint	父快照的ID。	0
matrix_id	bigint	CSS模式下快照的矩阵ID，否则为NULL。	0
root_id	bigint	初始快照的ID，通过db4ai.create_snapshot()从操作数据构建。	0
schema	name	导出快照视图的模式。	public
name	name	快照的名称，包括版本后缀。	example0@1.1.0
owner	name	创建此快照的用户的名称。	nw
commands	text[]	记录如何从其根快照生成到此快照的SQL语句的完整列表。	{DELETE,"WHERE id > 7"}
comment	text	快照说明。	inherits from @1.0.0

名称	类型	描述	实例
published	boolean	TRUE，当且仅当快照当前已发布。	f
archived	boolean	TRUE，当且仅当快照当前已存档。	f
created	timestamp without time zone	快照创建日期的时间戳。	2021-08-25 10:59:52.955604
row_count	bigint	此快照中的数据行数。	8

13.4.2 DB4AI.CREATE_SNAPSHOT

CREATE_SNAPSHOT是DB4AI特性用于创建快照的接口函数。通过语法CREATE SNAPSHOT调用。

表 13-195 DB4AI.CREATE_SNAPSHOT 入参和返回值列表

参数	类型	描述
i_schema	IN NAME	快照存储的模式名字，默认值是当前用户或者PUBLIC
i_name	IN NAME	快照名称
i_commands	IN TEXT[]	定义数据获取的SQL命令
i_vers	IN NAME	版本后缀
i_comment	IN TEXT	快照描述
res	OUT db4ai.snapshot_name	结果

13.4.3 DB4AI.CREATE_SNAPSHOT_INTERNAL

CREATE_SNAPSHOT_INTERNAL是db4ai.create_snapshot函数的内置执行函数。函数存在信息校验，无法直接调用。

表 13-196 DB4AI.CREATE_SNAPSHOT_INTERNAL 入参和返回值列表

参数	类型	描述
s_id	IN BIGINT	快照ID
i_schema	IN NAME	快照存储的名字空间

参数	类型	描述
i_name	IN NAME	快照名称
i_commands	IN TEXT[]	定义数据获取的SQL命令
i_comment	IN TEXT	快照描述
i_owner	IN NAME	快照拥有者

13.4.4 DB4AI.PREPARE_SNAPSHOT

PREPARE_SNAPSHOT是DB4AI特性中数据准备模型训练和解释快照进行协作。快照为所有应用更改的数据和文档提供了完整的序列。通过语法PREPARE SNAPSHOT调用。

表 13-197 DB4AI.PREPARE_SNAPSHOT 入参和返回值列表

参数	类型	描述
i_schema	IN NAME	快照存储的模式名字，默认值是当前用户或者PUBLIC
i_parent	IN NAME	父快照名称
i_commands	IN TEXT[]	定义快照修改的DDL和DML命令
i_vers	IN NAME	版本后缀
i_comment	IN TEXT	此数据策展单元的说明
res	OUT db4ai.snapshot_name	结果

13.4.5 DB4AI.PREPARE_SNAPSHOT_INTERNAL

PREPARE_SNAPSHOT_INTERNAL是db4ai.prepare_snapshot函数的内置执行函数。函数存在信息校验，无法直接调用。

表 13-198 DB4AI.PREPARE_SNAPSHOT_INTERNAL 入参和返回值列表

参数	类型	描述
s_id	IN BIGINT	快照ID
p_id	IN BIGINT	父快照ID
m_id	IN BIGINT	矩阵id
r_id	IN BIGINT	根快照ID

参数	类型	描述
i_schema	IN NAME	快照模式
i_name	IN NAME	快照名称
i_commands	IN TEXT[]	定义快照修改的DDL和DML命令
i_comment	IN TEXT	快照描述
i_owner	IN NAME	快照所有者
i_idx	INOUT INT	exec_cmds的索引
i_exec_cmds	INOUT TEXT[]	用于执行的DDL和DML
i_mapping	IN NAME[]	将用户列映射到备份列；如果不为NULL，则生成规则

13.4.6 DB4AI.ARCHIVE_SNAPSHOT

ARCHIVE_SNAPSHOT是DB4AI特性用于存档快照的接口函数。通过语法ARCHIVE SNAPSHOT调用。生效后的快照无法参与训练等任务。

表 13-199 DB4AI.ARCHIVE_SNAPSHOT 入参和返回值列表

参数	类型	描述
i_schema	IN NAME	快照存储的模式名字，默认值是当前用户
i_name	IN NAME	快照名称
res	OUT db4ai.snapshot_name	结果

13.4.7 DB4AI.PUBLISH_SNAPSHOT

PUBLISH_SNAPSHOT是DB4AI特性用于发布快照的接口函数。通过语法PUBLISH SNAPSHOT调用。

表 13-200 DB4AI.PUBLISH_SNAPSHOT 入参和返回值列表

参数	类型	描述
i_schema	IN NAME	快照存储的模式名字，默认值是当前用户或者PUBLIC

参数	类型	描述
i_name	IN NAME	快照名称
res	OUT db4ai.snapshot_name	结果

13.4.8 DB4AI.MANAGE_SNAPSHOT_INTERNAL

MANAGE_SNAPSHOT_INTERNAL是DB4AI.PUBLISH_SNAPSHOT和DB4AI.ARCHIVE_SNAPSHOT函数的内置执行函数。函数存在信息校验，无法直接调用。

表 13-201 DB4AI.MANAGE_SNAPSHOT_INTERNAL 入参和返回值列表

参数	类型	描述
i_schema	IN NAME	快照存储的模式名字
i_name	IN NAME	快照名称
publish	IN BOOLEAN	是否是发布状态
res	OUT db4ai.snapshot_name	结果

13.4.9 DB4AI.SAMPLE_SNAPSHOT

SAMPLE_SNAPSHOT是DB4AI特性用于对基数据进行采样生成快照的接口函数。通过语法SAMPLE SNAPSHOT调用。

表 13-202 DB4AI.SAMPLE_SNAPSHOT 入参和返回值列表

参数	类型	描述
i_schema	IN NAME	快照存储的模式名字
i_parent	IN NAME	父快照名称
i_sample_infixes	IN NAME[]	示例快照名称中缀
i_sample_ratios	IN NUMBER[]	每个样本的大小，作为父集的比率
i_stratify	IN NAME[]	分层策略
i_sample_comments	IN TEXT[]	示例快照描述
res	OUT db4ai.snapshot_name	结果

13.4.10 DB4AI.PURGE_SNAPSHOT

PURGE_SNAPSHOT是DB4AI特性用于删除快照的接口函数。通过语法PURGE SNAPSHOT调用。

表 13-203 DB4AI.PURGE_SNAPSHOT 入参和返回值列表

参数	类型	描述
i_schema	IN NAME	快照存储的模式名字
i_name	IN NAME	快照名称
res	OUT db4ai.snapshot_name	结果

13.4.11 DB4AI.PURGE_SNAPSHOT_INTERNAL

PURGE_SNAPSHOT_INTERNAL是DB4AI.PURGE_SNAPSHOT函数的内置执行函数。函数存在信息校验，无法直接调用

表 13-204 DB4AI.PURGE_SNAPSHOT_INTERNAL 入参和返回值列表

参数	类型	描述
i_schema	IN NAME	快照存储的模式名字
i_name	IN NAME	快照名称

13.5 DBE_PLDEVELOPER

DBE_PLDEVELOPER下系统表用于记录PL/SQL包、函数及存储过程编译过程中需要记录的信息。

13.5.1 DBE_PLDEVELOPER.gs_source

用于记录PL/SQL对象（存储过程、函数、包、包体）编译相关信息，具体内容见下列字段描述。

打开plsql_show_all_error参数后，会把成功或失败的PL/SQL对象编译信息记录在此表中，如果关闭plsql_show_all_error参数则只会将正确的编译相关信息插入此表中。

⚠ 注意

1. gs_source表中只记录用户定义的原始对象语句，即使用户使用了ALTER改变了创建的SCHEMA或者名字，gs_source表中的信息也不会发生变化，如果用户更改了对象的SCHEMA或者名字，会导致用户在删除对象后，对象仍存在于gs_source表中。
2. gs_source表中的owner指创建的用户，不是用户创建存储过程或者package时指定的用户。
3. 数据库默认情况下没有对gs_source表中设置行级访问控制，如果用户想使用数据库隔离性特性，请参考以下语句，自行添加行级访问控制。

```
ALTER TABLE dbe_pldeveloper.gs_source ENABLE ROW LEVEL SECURITY;
CREATE ROW LEVEL SECURITY POLICY all_data_rls ON dbe_pldeveloper.gs_source USING(owner =
(select oid from pg_roles where rolname=current_user));
```

表 13-205 DBE_PLDEVELOPER.gs_source 字段

名称	类型	描述
id	oid	对象的ID。
owner	bigint	对象创建用户ID。
nspid	oid	对象的模式ID。
name	name	对象名。
type	text	对象类型（procedure/function/package/package body）。
status	boolean	是否创建成功。
src	text	对象创建的原始语句。

13.5.2 DBE_PLDEVELOPER.gs_errors

用于记录PL/SQL对象（存储过程、函数、包、包体）编译过程中遇到的报错信息，具体内容见下列字段描述。

打开plsql_show_all_error参数后，如果编译过程中存在报错，则会跳过报错继续编译并把报错信息记录在gs_errors中，如果关闭plsql_show_all_error参数，且behavior_compat_options参数不为skip_insert_gs_source，则会直接将报错信息插入此表中。

该表的owner是创建的用户，修改存储过程或者package的owner不会修改该表信息。

表 13-206 DBE_PLDEVELOPER.gs_errors 字段

名称	类型	描述
id	oid	对象的ID。
owner	bigint	对象创建用户ID。
nspid	oid	对象的模式ID。

name	name	对象名。
type	text	对象类型（procedure/function/package/package body）。
line	integer	行号。
src	text	报错信息。

注意

1. 创建包头时的开头（as/is之前）和结尾（end之后），如果出现错误，不会记录在gs_errors表格里面，会直接在客户端返回错误的行号和该行具体内容，返回的行号不一定准确。个别is本身和end本身错误场景也不会记录在gs_errors表格里面。
2. 创建包体时的开头（as/is之前）和结尾（end之后），如果出现错误，不会记录在gs_errors表格里面，会直接在客户端返回错误的行号和该行具体内容，返回的行号不一定准确。个别is本身和end本身错误场景也不会记录在gs_errors表格里面。
3. 创建包体时，函数或者存储过程的结尾（end之后），如果出现错误，不会记录在gs_errors表格里面，会直接在客户端返回错误的行号和该行具体内容，行号不一定准确。
4. 创建包体时，函数或者存储过程的开头（as/is 以及 as/is之前）如果出现错误，报错行数不准确。
5. 创建包头时，变量声明少分号，会记录在gs_errors表格里面，记录的报错行号不准确。开启参数不会记录。
6. 包体内存储过程或函数内部，自治事务标识符PRAGMA AUTOMOUS_TRANSACTION声明错误时，不能确保是否能记录在gs_errors表里。
7. 客户端直接报错，但是gs_errors表格未记录的情况，如果客户端报错行号不对，本需求不纠正原本报错的行号。
8. 对于类似 if ....then、for....loop、when .... then 语句中间的错误或EXCEPTION本身错误，报错行号在本行，而不是下一个分号所在行。
9. 包内存储过程或函数内部begin本身错误的场景，报错行数不准。

13.6 DBE_SQL_UTIL Schema

DBE_SQL_UTIL模式存储了用于管理SQL PATCH的工具，包括创建、删除、开启、禁用SQL PATCH等系统函数。普通用户只有usage权限，没有create、alter、drop、comment等权限。

DBE_SQL_UTIL Schema使用请参考**使用SQL PATCH进行调优**。

13.6.1 DBE_SQL_UTIL.create_hint_sql_patch

create_hint_sql_patch是用于创建调优SQL PATCH的接口函数，返回执行是否成功。

限制仅初始用户、sysadmin、opradmin、monadmin用户有权限调用。

表 13-207 DBE_SQL_UTIL.create_hint_sql_patch 入参和返回值列表

参数	类型	描述
patch_name	IN name	PATCH名称。
unique_sql_id	IN bigint	查询全局唯一ID。
hint_string	IN text	Hint文本。
description	IN text	PATCH的备注，默认值为NULL。
enabled	IN bool	PATCH是否生效，默认值为true。
result	OUT bool	执行是否成功。

13.6.2 DBE_SQL_UTIL.create_abort_sql_patch

create_abort_sql_patch是用于创建避险SQL PATCH的接口函数，返回执行是否成功。

限制仅初始用户、sysadmin、opradmin、monadmin用户有权限调用。

表 13-208 DBE_SQL_UTIL.create_abort_sql_patch 入参和返回值列表

参数	类型	描述
patch_name	IN name	PATCH名称。
unique_sql_id	IN bigint	查询全局唯一ID。
description	IN text	PATCH的备注，默认值为NULL。
enabled	IN bool	PATCH是否生效，默认值为true。
result	OUT bool	执行是否成功。

13.6.3 DBE_SQL_UTIL.drop_sql_patch

drop_sql_patch是用于删除SQL PATCH的接口函数，返回执行是否成功。

限制仅初始用户、sysadmin、opradmin、monadmin用户有权限调用。

表 13-209 DBE_SQL_UTIL.drop_sql_patch 入参和返回值列表

参数	类型	描述
patch_name	IN name	PATCH名称。
result	OUT bool	执行是否成功。

13.6.4 DBE_SQL_UTIL.enable_sql_patch

enable_sql_patch是用于开启SQL PATCH的接口函数，返回执行是否成功。

限制仅初始用户、sysadmin、opradmin、monadmin用户有权限调用。

表 13-210 DBE_SQL_UTIL.enable_sql_patch 入参和返回值列表

参数	类型	描述
patch_name	IN name	PATCH名称。
result	OUT bool	执行是否成功。

13.6.5 DBE_SQL_UTIL.disable_sql_patch

disable_sql_patch是用于禁用SQL PATCH的接口函数，返回执行是否成功。

限制仅初始用户、sysadmin、opradmin、monadmin用户有权限调用。

表 13-211 DBE_SQL_UTIL.disable_sql_patch 入参和返回值列表

参数	类型	描述
patch_name	IN name	PATCH名称。
result	OUT bool	执行是否成功。

13.6.6 DBE_SQL_UTIL.show_sql_patch

show_sql_patch是用于显示给定patch_name对应的SQL PATCH的接口函数，返回运行结果。

限制仅初始用户、sysadmin、opradmin、monadmin用户有权限调用。

表 13-212 DBE_SQL_UTIL.show_sql_patch 入参和返回值列表

参数	类型	描述
patch_name	IN name	PATCH名称。
unique_sql_id	OUT bigint	查询全局唯一ID。
enabled	OUT bool	PATCH是否生效。
abort	OUT bool	是否是AbortHint。
hint_str	OUT text	Hint文本。

13.6.7 DBE_SQL_UTIL.create_hint_sql_patch

create_hint_sql_patch是用于创建调优SQL PATCH的接口函数，返回执行是否成功。本函数是原函数的重载函数，支持通过parent_unique_sql_id值限制hint patch的生效范围。

限制仅初始用户、sysadmin、opradmin、monadmin用户有权限调用。

表 13-213 DBE_SQL_UTIL.create_hint_sql_patch 重载函数入参和返回值列表

参数	类型	描述
patch_name	IN name	PATCH名称。
unique_sql_id	IN bigint	查询全局唯一ID。
parent_unique_sql_id	IN bigint	标识外层SQL语句的全局唯一ID，值为0时表示限制存储过程外语句SQL PATCH生效；非0值表示限制特定存储过程生效。
hint_string	IN text	Hint文本。
description	IN text	PATCH的备注，默认值为NULL。
enabled	IN bool	PATCH是否生效，默认值为true。
result	OUT bool	执行是否成功。

13.6.8 DBE_SQL_UTIL.create_abort_sql_patch

create_abort_sql_patch是用于创建避险SQL PATCH的接口函数，返回执行是否成功。本函数是原函数的重载函数，支持通过parent_unique_sql_id值限制abort patch的生效范围。

限制仅初始用户、sysadmin、opradmin、monadmin用户有权限调用。

表 13-214 DBE_SQL_UTIL.create_abort_sql_patch 重载函数入参和返回值列表

参数	类型	描述
patch_name	IN name	PATCH名称。
unique_sql_id	IN bigint	查询全局唯一ID。
parent_unique_sql_id	IN bigint	标识外层SQL语句的全局唯一ID，值为0时表示限制存储过程外语句SQL PATCH生效；非0值表示限制特定存储过程生效。
description	IN text	PATCH的备注，默认值为NULL。

参数	类型	描述
enabled	IN bool	PATCH是否生效，默认值为true。
result	OUT bool	执行是否成功。

14 配置运行参数

14.1 查看参数

GaussDB安装后，有一套默认的运行参数，为了使GaussDB与业务的配合度更高，用户需要根据业务场景和数据量的大小进行GUC参数调整。

操作步骤

步骤1 连接数据库。

步骤2 查看数据库运行参数当前取值。

- 方法一：使用SHOW命令。
  - 使用如下命令查看单个参数：
    ```
    gaussdb=# SHOW server_version;
    ```
    server_version显示数据库版本信息的参数。
  - 使用如下命令查看所有参数：
    ```
    gaussdb=#  SHOW ALL;
    ```
- 方法二：使用pg_settings视图。
  - 使用如下命令查看单个参数：
    ```
    gaussdb=# SELECT * FROM pg_settings WHERE NAME='server_version';
    ```
  - 使用如下命令查看所有参数：
    ```
    gaussdb=# SELECT * FROM pg_settings;
    ```

----结束

示例

查看客户端的字符编码类型。

```
gaussdb=# SHOW SHOW client_encoding;
 SHOW client_encoding
----------------
 UTF8
(1 row)
```


14.2 设置参数

GaussDB支持在管理控制台修改部分参数，建议在管理控制台上修改指定参数，如果需要修改的参数在管理控制台无法修改，请提前评估风险后再联系客服进行修改。

背景信息

GaussDB提供了多种修改GUC参数的方法，用户可以方便地针对数据库、用户、会话进行设置。

- 参数名称不区分大小写。
- 参数取值有整型、浮点型、字符串、布尔型和枚举型五类。
  - 布尔值可以是（on，off）、（true，false）、（yes，no）或者（1，0），且不区分大小写。
  - 枚举类型的取值是在系统表pg_settings的enumvals字段取值定义的。
- 对于有单位的参数，在设置时请指定单位，否则将使用默认的单位。
  - 参数的默认单位在系统表pg_settings的unit字段定义的。
  - 内存单位有：KB（千字节）、MB（兆字节）和GB（吉字节）。
  - 时间单位：ms（毫秒）、s（秒）、min（分钟）、h（小时）和d（天）。

具体参数说明请参见**GUC参数说明**。

GUC 参数设置

GaussDB提供了六类GUC参数，具体分类和设置方式请参考**表14-1**：

表 14-1 GUC 参数分类

参数类型	说明	设置方式
INTERNAL	固定参数，在创建数据库的时候确定，用户无法修改，只能通过show语法或者pg_settings视图进行查看。	无
POSTMASTER	数据库服务端参数，在数据库启动时确定，可以通过配置文件指定。	支持**表14-2**中的方式一。
SIGHUP	数据库全局参数，可在数据库启动时设置或者在数据库启动后，发送指令重新加载。	支持**表14-2**中的方式一、方式二。
BACKEND	会话连接参数。在创建会话连接时指定，连接建立后无法修改。连接断掉后参数失效。内部使用参数，不推荐用户设置。	支持**表14-2**中的方式一、方式二。 **说明** 设置该参数后，下一次建立会话连接时生效。

参数类型	说明	设置方式
SUSET	数据库管理员参数。可在数据库启动时、数据库启动后或者数据库管理员通过SQL进行设置。	支持表14-2中的方式一、方式二或由数据库管理员通过方式三设置。
USERSET	普通用户参数。可被任何用户在任何时刻设置。	支持表14-2中的方式一、方式二或方式三设置。

GaussDB提供了三种方式来修改GUC参数，具体操作请参考表14-2：

表 14-2 GUC 参数设置方式

序号	设置方法
方式一	1. 登录管理控制台。 2. 在“实例管理”页面，选择指定的实例，单击实例名称，进入实例基本信息页面。 3. 在左侧导航栏单击“参数修改”，进入参数修改页面，在该页面修改参数。 如果需要修改的参数在管理该控制台无法修改，请提前评估风险后再联系客服进行修改。 4. 重启数据库使参数生效。 **说明** 重启数据库集群操作会导致用户执行操作中断，请在操作之前规划好合适的执行窗口。
方式二	1. 登录管理控制台。 2. 在“实例管理”页面，选择指定的实例，单击实例名称，进入实例基本信息页面。 3. 在左侧导航栏单击“参数修改”，进入参数修改页面，在该页面修改参数。 如果需要修改的参数在管理该控制台无法修改，请提前评估风险后再联系客服进行修改。
方式三	修改会话级别的参数。 • 设置会话级别的参数 gaussdb=# **SET** *paraname* **TO** *value;* 修改本次会话中的取值。退出会话后，设置将失效。

⚠ 注意

使用方式一和方式二设置参数时，若所设参数不属于当前环境，数据库会提示参数不在支持范围内的相关信息。

如果需要修改的参数无法通过以上方式设置，请提前评估风险后再联系客服进行修改。

14.3 GUC 参数说明

14.3.1 GUC 使用说明

数据库提供了许多运行参数，配置这些参数可以影响数据库系统的行为。在修改这些参数时请确保用户理解了这些参数对数据库的影响，否则可能会导致无法预料的结果。

注意事项

- 参数中如果取值范围为字符串，此字符串应遵循操作系统的路径和文件名命名规则。
- 取值范围最大值为INT_MAX的参数，此选项最大值跟所在的操作系统有关。
- 取值范围最大值为DBL_MAX的参数，此选项最大值跟所在的操作系统有关。

14.3.2 文件位置

数据库安装后会自动生成三个配置文件（postgresql.conf、pg_hba.conf和pg_ident.conf），并统一存放在数据目录（data）下。用户可以使用本节介绍的方法修改配置文件的名称和存放路径。

修改任意一个配置文件的存放目录时，postgresql.conf里的data_directory参数必须设置为实际数据目录（data）。

须知

考虑到配置文件修改一旦出错对数据库的影响很大，不建议安装后再修改本节的配置文件。

data_directory

参数说明：设置GaussDB的数据目录（data目录），仅sysadmin用户可以访问。此参数可以通过如下方式指定。

- 在安装GaussDB时指定。
- 该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，长度大于0

默认值：安装时指定，如果在安装时不指定，则默认不初始化数据库。

config_file

参数说明：设置主服务器配置文件名称（postgresql.conf）。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置

取值范围：字符串，长度大于0

默认值：postgresql.conf(实际安装可能带有绝对目录)

hba_file

参数说明：设置基于主机认证（HBA）的配置文件（pg_hba.conf）。此参数只能在配置文件postgresql.conf中指定，仅sysadmin用户可以访问。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

默认值：pg_hba.conf（实际安装可能带有绝对目录）

ident_file

参数说明：设置用于客户端认证的配置文件的名称（pg_ident.conf），仅sysadmin用户可以访问。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

默认值：pg_ident.conf（实际安装可能带有绝对目录）

external_pid_file

参数说明：声明可被服务器管理程序使用的额外PID文件，仅sysadmin用户可以访问。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

这个参数只能在数据库服务重新启动后生效。

取值范围：字符串

默认值：空

enable_default_cfunc_libpath

参数说明：设置GaussDB创建C函数时的so文件是否使用默认路径。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

on：代表创建C函数时，so文件必须放在指定的目录（$libdir/proc_srclib）下。

off：代表创建C函数时，so文件可以放在任意可访问的目录下。

默认值：on

须知

参数设置成off时，.so文件可以放在任意可访问的目录下或使用系统自带的.so，存在安全风险，不建议使用。

14.3.3 连接和认证

14.3.3.1 连接设置

介绍设置客户端和服务器连接方式相关的参数。

light_comm

参数说明：指定服务器是否使用轻量通信方式。

该参数指定服务器是否使用基于轻量锁和非阻塞socket的通信方式。该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- TRUE / ON：表示指定使用轻量通信方式。
- FALSE / OFF：表示指定不使用轻量通信方式。

默认值：FALSE / OFF

listen_addresses

参数说明：声明服务器侦听客户端的TCP/IP地址。

该参数指定GaussDB服务器使用哪些IP地址进行侦听，如IPV4。服务器主机上可能存在多个网卡，每个网卡可以绑定多个IP地址，该参数就是控制GaussDB绑定在哪个或者哪几个IP地址上。而客户端则可以通过该参数中指定的IP地址来连接GaussDB或者给GaussDB发送请求。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：

- 主机名或IP地址，多个值之间用英文逗号分隔。
- 星号“*”或“0.0.0.0”表示侦听所有IP地址。配置侦听所有IP地址存在安全风险，不推荐用户使用。必须与有效地址结合使用（比如本地IP等），否则，可能造成Build失败的问题。同时，主备环境下配置为“*”或“0.0.0.0”时，主节点数据库路径下postgresql.conf文件中的localport端口号不能为数据库dataPortBase+1，否则会导致数据库无法启动。
- 当该参数值非法时（如包含非法IP、非法字符，参数值为空，或重复侦听），进程将启动失败。

默认值：数据库实例安装好后，根据public_cloud.conf配置文件中不同实例的IP地址配置不同默认值。DN的默认参数值为：listen_addresses = 'data.net网卡对应的IP地址'。

说明

- public_cloud.conf文件保存的网卡信息，包括：mgr.net（管理网卡）、data.net（数据网卡）、virtual.net（虚拟网卡）。
- 支持gs_guc reload方式进行设置该参数值，GaussDB内核会按照“侦听新IP、保留重复IP的侦听、关闭无效IP侦听”的策略，动态侦听该参数配置的IP。若存在非法IP（如本机未配置此IP、无效的IP值、该IP已被侦听等）时，内核侦听IP失败，此时会出现listen_addresses设置值和实际侦听IP的状况不匹配。
- 若通过gs_guc reload方式设置参数值，且参数值中均为非法IP时，侦听新IP失败，同时会取消全部旧IP的侦听。若参数为空时，直接拦截而保持旧IP侦听。
- 若在进程启动参数中配置listen_addresses后，该参数将强制指定为设置值而不会被reload修改。
- 若设置新参数值和已设置旧参数值相同时，将不再进行侦听动作。故设置listen_addresses参数值且列表中某非法IP时，内核侦听此IP失败；待IP变为合法后，再次设置相同参数值，由于前后两次参数值相同，则不会侦听此IP。此时需通过gs_guc reload方式将此IP从参数值中去除，并再次设置添加该IP，才会成功侦听。
- 通过gs_guc reload方式刷新该参数后，若不再侦听某IP后，需再执行gs_validate_ext_listen_ip函数进行清理连接操作，具体入参及执行详见该函数说明。
- 禁止gs_guc工具通过"-N all"方式reload刷新所有DN的listen_addresses参数值，仅允许单DN设置。
- 动态修改listen_addresses属于高危操作，若配置有误可能导致数据库无法接受新连接，进而影响业务，请谨慎操作。

local_bind_address

参数说明：声明当前节点连接数据库其他节点绑定的本地IP地址。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

默认值：数据库实例安装好后，根据public_cloud.conf配置文件中不同实例的IP地址配置不同默认值。DN的默认参数值为：local_bind_address = 'data.net网卡对应的IP地址'。

说明

public_cloud.conf文件保存的网卡信息，包括：mgr.net（管理网卡）、data.net（数据网卡）、virtual.net（虚拟网卡）

port

参数说明：GaussDB服务侦听的TCP端口号。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

说明

该参数由安装时的配置文件指定，请勿轻易修改，否则修改后会影响数据库正常通信。

取值范围：整型，1～65535

说明

- 设置端口号时，请设置一个未被占用的端口号。设置多个实例的端口号，不可冲突。
- 1~1023为操作系统保留端口号，请不要使用。
- 通过配置文件安装数据库实例时，配置文件中的端口号需要注意通信矩阵预留端口。如：DN还需保留dataPortBase+1作为内部工具使用端口，保留dataPortBase+6作为流引擎（由于规格变更，当前版本已经不再支持本特性，请不要使用）消息队列通信端口等。故数据库实例安装阶段，port最大值为：DN可设置65529，同时需要保证端口号不冲突。

默认值：5432（实际值由安装时的配置文件指定）

max_connections

参数说明：允许和数据库连接的最大并发连接数。此参数会影响数据库的并发能力。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型。最小值为10（要大于max_wal_senders），理论最大值为262143，实际最大值为动态值，计算公式为“262143 - job_queue_processes - autovacuum_max_workers - AUXILIARY_BACKENDS - AV_LAUNCHER_PROCS - max_inner_tool_connections - max_concurrent_autonomous_transactions - min(max(newValue/4,64),1024)”，**job_queue_processes**、**autovacuum_max_workers**、**max_inner_tool_connections**和**max_concurrent_autonomous_transactions**的值取决于对应GUC参数的设置，AUXILIARY_BACKENDS为预留辅助线程数固定为20，AV_LAUNCHER_PROCS为预留autovacuum的launcher线程数固定为2，min(max(newValue/4,64),1024)公式中newValue为新设置的值。

在不同实例的内存规格下，

表 14-3 不同实例的内存规格的参数取值范围

内存规格	DN参数取值范围
< 32GB	[10, 100]
[32GB, 64GB)	[10, 200]
[64GB, 128GB)	[10, 2048]
[128GB, 256GB)	[10, 5000]
[256GB, 480GB)	[10, 11000]
[480GB, 512GB)	[10, 24000]
[512GB, 640GB)	[10, 25000]
[640GB, 768GB)	[10, 34000]
[768GB, 1024GB)	[10, 40000]
[1024GB, 1536GB)	[10, 55000]
[1536GB, 2048GB)	[10, 85000]
>= 2048GB	[10, 110000]

默认值：

85000（196核CPU/1536G内存）；55000（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存）；40000（96核CPU/768G内存）；34000（80核CPU/640G内存）；25000（64核CPU/512G内存）；24000（60核CPU/480G内存）；11000（32核CPU/256G内存）；5000（16核CPU/128G内存）；2048（8核CPU/64G内存）；100（4核CPU/32G内存，4核CPU/16G内存）

设置建议：

数据库主节点中此参数建议保持默认值。

配置不当时影响：

- 若配置max_connections过大，超过计算公式所描述的最大动态值，会出现节点拉起失败问题，报错提示“invalid value for parameter "max_connections"”；或在拉起时申请内存失败，报错提示“Cannot allocate memory”。
- 若未按照对外出口规格配置仅调大max_connections参数值，未同比例调整内存参数。业务压力大时，容易出现内存不足，报错提示“memory is temporarily unavailable”。

说明

- 对于管理员用户的连接数限制会略超过max_connections设置，目的是为了让管理员在连接被普通用户占满后仍可以连接上数据库，再超过一定范围（sysadmin_reserved_connections参数）后才会报错。即管理员用户的最大连接数等于max_connections + sysadmin_reserved_connections。
- 对于普通用户来说，由于内部作业也会使用一些链接，因此会略小于max_connections，具体值取决于内部链接个数。

max_inner_tool_connections

参数说明：允许和数据库连接的工具的最大并发连接数。此参数会影响GaussDB的工具连接并发能力。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，最小值为1，最大值为MIN(262143, max_connections)，max_connections的计算方法见上文。

默认值：

50（196核CPU/1536G内存，128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，64核CPU/512G内存，60核CPU/480G内存，32核CPU/256G内存，16核CPU/128G内存，8核CPU/64G内存，4核CPU/32G内存）；10（4核CPU/16G内存）

如果该默认值超过内核支持的最大值（在执行gs_initdb的时候判断），系统会提示错误。

设置建议：

数据库主节点中此参数建议保持默认值。

增大此参数可能导致GaussDB要求更多的SystemV共享内存或者信号量，可能超过操作系统缺省配置的最大值。这种情况下，请酌情对数值加以调整。

sysadmin_reserved_connections

参数说明：为管理员用户预留的最少连接数，不建议设置过大。该参数和max_connections参数配合使用，管理员用户的最大连接数等于max_connections + sysadmin_reserved_connections。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，最小值为0，最大值为MIN(262143, max_connections)，max_connections的计算方法见上文。

默认值：3

注意：当启用线程池功能时，若线程池占满将形成处理瓶颈，导致管理员预留连接无法正常建立；作为逃生手段，此时可使用gsql通过主端口+1端口号连入，清理无用会话，即可正常连入。

service_reserved_connections

参数说明：为后台运维用户（带有persistence属性）预留的最少连接数，不建议设置过大。该参数和max_connections参数配合使用，运维用户的最大连接数等于max_connections + service_reserved_connections。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，最小值为0，最大值为262143。

默认值：10

注意：如果设置过小，会导致在max_connections满的情况下，该运维用户（带有persistence属性）无法连接数据库，作业无法正常执行。

unix_socket_directory

参数说明：设置GaussDB服务器侦听客户端连接的Unix域套接字目录。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

该参数的长度限制于操作系统的长度，超过该限制将会导致Unix-domain socket path "xxx" is too long的问题。

取值范围：字符串

默认值：空字符串（实际值由安装时配置文件指定）

unix_socket_group

参数说明：设置Unix域套接字的所属组（套接字的所属用户总是启动服务器的用户）。可以与选项**unix_socket_permissions**一起用于对套接字进行访问控制。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，其中空字符串表示当前用户的缺省组。

默认值：空字符串

unix_socket_permissions

参数说明：设置Unix域套接字的访问权限。

Unix域套接字使用普通的Unix文件系统权限集。这个参数的值应该是数值的格式（chmod和umask命令可接受的格式）。如果使用自定义的八进制格式，数字必须以0开头。

建议设置为0770（只有当前连接数据库的用户和同组的人可以访问）或者0700（只有当前连接数据库的用户自己可以访问，同组或者其他人都没有权限）。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：0000-0777

默认值：0700

说明

在Linux中，文档具有十个属性，其中第一个属性为文档类型，后面九个为权限属性，分别为Owner，Group及Others这三个组别的read、write、execute属性。

文档的权限属性分别简写为r，w，x，这九个属性三个为一组，也可以使用数字来表示文档的权限，对照表如下：

r：4

w: 2

x：1

-：0

同一组（owner/group/others）的三个属性是累加的。

例如，-rwxrwx---表示这个文档的权限为：

owner = rwx = 4+2+1 = 7

group = rwx = 4+2+1 = 7

others = --- = 0+0+0 = 0

所以其权限为0770。

application_name

参数说明：当前连接请求当中，所使用的客户端名称。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

在备机请求主机进行日志复制时，如果该参数非空串，那么会被用来作为备机在主机上的流复制槽名字。此时，如果该参数长度超过61个字节，那么流复制槽名字只会截取使用前61个字节的字符。

取值范围：字符串。

默认值：空字符串(连接到后端的应用名，以实际安装为准)

connection_info

参数说明：连接数据库的驱动类型、驱动版本号、当前驱动的部署路径和进程属主用户。

该参数属于USERSET类型参数，属于运维类参数，不建议用户设置。

取值范围：字符串。

默认值：空字符串。

说明

- 空字符串，表示当前连接数据库的驱动不支持自动设置connection_info参数或应用程序未设置。
- 驱动连接数据库的时候自行拼接的connection_info参数格式如下：
  ```
  {"driver_name":"ODBC","driver_version": "(GaussDB Kernel 503.1.XXX build 290d125f) compiled at 2020-05-08 02:59:43 commit 2143 last mr 131 release","driver_path":"/usr/local/lib/psqlodbcw.so","os_user":"omm"}
  ```
  默认显示driver_name和driver_version，driver_path和os_user的显示由用户控制（参见《开发指南》中“应用程序开发教程 > 基于JDBC开发 > 连接数据库”章节和《开发指南》中“应用程序开发教程 > 基于ODBC开发 > Linux下配置数据源”章节）。

14.3.3.2 安全和认证（postgresql.conf）

介绍设置客户端和服务器的安全认证方式的相关参数。

authentication_timeout

参数说明：完成客户端认证的最长时间。如果一个客户端没有在这段时间里完成与服务器端的认证，则服务器自动中断与客户端的连接，这样就避免了出问题的客户端无限制地占用连接数。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，最小值为1，最大值为600，最小单位为s。

默认值：1min

auth_iteration_count

参数说明：认证加密信息生成过程中使用的迭代次数。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，2048-134217728。

默认值：10000

须知

迭代次数设置过小会降低口令存储的安全性，设置过大会导致认证、用户创建等涉及口令加密的场景性能劣化，请根据实际硬件条件合理设置迭代次数，推荐采用默认迭代次数。

session_authorization

参数说明：当前会话的用户标识。

该参数属于USERSET类型参数，只能通过《开发指南》中“SQL参考 > SQL语法 > SET SESSION AUTHORIZATION”章节语法设置，不支持直接设置。

取值范围：字符串。

默认值：NULL

session_timeout

参数说明：表明与服务器建立链接后，不进行任何操作的最长时间。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0-86400，最小单位为s，0表示关闭超时设置。

默认值：1800s

须知

GaussDB gsql客户端中有自动重连机制，所以针对初始化用户本地连接，超时后gsql表现的现象为断开后重连。

ssl

参数说明：启用SSL连接。请在使用这个选项之前阅读**通过gsql连接实例**章节。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示启用SSL连接。
- off表示不启用SSL连接。

须知

开启此参数需要同时配置ssl_cert_file、ssl_key_file和ssl_ca_file等参数及对应文件，如果使用国密认证，还需要确保ssl_enc_cert_file和ssl_enc_key_file参数配置正确，不正确的配置可能会导致数据库无法正常启动。

默认值：on

require_ssl

参数说明：设置服务器端是否强制要求SSL连接，该参数只有当参数**ssl**为on时才有效。请在使用这个选项之前阅读**通过gsql连接实例**章节。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示服务器端强制要求SSL连接。
- off表示服务器端对是否通过SSL连接不作强制要求。

须知

GaussDB目前支持SSL的场景为客户端连接数据库主节点场景，该参数目前建议只在数据库主节点中开启。

默认值：off

ssl_ciphers

参数说明：指定SSL支持的加密算法列表，仅sysadmin用户可以访问。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，如果指定多个加密算法，加密算法之间需要以分号分隔。

默认值：ALL

须知

ssl_ciphers设置错误会导致数据库不能正常启动。

ssl_renegotiation_limit

参数说明：指定在会话密钥重新协商之前，通过SSL加密通道可以传输的流量。这个重新协商流量限制机制可以减少攻击者针对大量数据使用密码分析法破解密钥的几率，但是也带来较大的性能损失。流量是指发送和接受的流量总和。使用SSL重协商机制可能引入其他风险，因此已禁用SSL重协商机制，为保持版本兼容保留此参数，修改参数配置不再起作用。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，最小值为0，最大值为2147483647。单位为KB。其中0表示禁用重新协商机制。

默认值：0

ssl_cert_file

参数说明：指定包含SSL服务器证书的文件的名称，其相对路径是相对于数据目录的。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

默认值：server.crt

ssl_key_file

参数说明：指定包含SSL私钥的文件名称，其相对路径是相对于数据目录的。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

默认值：server.key

ssl_enc_cert_file

参数说明：指定包含SSL服务器国密加密证书的文件的名称，其相对路径是相对于数据目录的。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

默认值：server_enc.crt

ssl_enc_key_file

参数说明：指定包含SSL私钥的文件名称，其相对路径是相对于数据目录的。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

默认值：server_enc.key

ssl_ca_file

参数说明：指定包含CA信息的文件的名称，其相对路径是相对于数据目录的。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，其中空字符串表示没有CA文件被加载，不进行客户端证书验证。

默认值：cacert.pem

ssl_crl_file

参数说明：证书吊销列表，如果客户端证书在该列表中，则当前客户端证书被视为无效证书。必须使用相对路径，相对路径是相对于数据目录的。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，空字符串表示没有吊销列表。

默认值：空

krb_server_keyfile

参数说明：指定Kerberos服务主配置文件的位置。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

默认值：空

krb_srvname

参数说明：设置Kerberos服务名。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

默认值：postgres

krb_caseins_users

参数说明：设置Kerberos用户名是否大小写敏感。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示大小写不敏感
- off表示大小写敏感

默认值：off

modify_initial_password

参数说明：当GaussDB安装成功后，数据库中仅存在一个初始用户（UID为10的用户）。客户通过该账户初次登录数据库进行操作时，该参数决定是否要对该初始账户的密码进行修改。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

如果安装过程中未指定初始用户密码，则安装后初始用户密码默认为空，执行其他操作前需要先通过gsql客户端设置初始用户的密码。此参数功能不再生效，保留此参数仅为兼容升级场景。

取值范围：布尔型

- on表示数据库安装成功后初始用户首次登录操作前需要修改初始密码。
- off表示数据库安装成功后初始用户无需修改初始密码即可进行操作。

默认值：off

password_policy

参数说明：在使用CREATE ROLE/USER或者ALTER ROLE/USER命令创建或者修改GaussDB账户时，该参数决定是否进行密码复杂度检查。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

从安全性考虑，请勿关闭密码复杂度策略。

取值范围：0、1

- 0表示不采用密码复杂度校验策略。
- 1表示采用默认密码复杂度校验策略。

默认值：1

password_reuse_time

参数说明：在使用ALTER USER或者ALTER ROLE修改用户密码时，该参数指定是否对新密码进行可重用天数检查。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

修改密码时会检查配置参数**password_reuse_time**和**password_reuse_max**。

- 当**password_reuse_time**和**password_reuse_max**都为正数时，只要满足其中任一个，即可认为密码可以重用。
- 当**password_reuse_time**为0时，表示不限制密码重用天数，仅限制密码重用次数。
- 当**password_reuse_max**为0时，表示不限制密码重用次数，仅限制密码重用天数。
- 当**password_reuse_time**和**password_reuse_max**都为0时，表示不对密码重用进行限制。

取值范围：浮点型（天），最小值为0，最大值为3650。

- 0表示不检查密码可重用的天数。
- 正数表示新密码不能为该值指定的天数内使用过的密码。

默认值：0

password_reuse_max

参数说明：在使用ALTER USER或者ALTER ROLE修改用户密码时，该参数指定是否对新密码进行可重用次数检查，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

修改密码时会检查配置参数**password_reuse_time**和**password_reuse_max**。

- 当**password_reuse_time**和**password_reuse_max**都为正数时，只要满足其中任一个，即可认为密码可以重用。
- 当**password_reuse_time**为0时，表示不限制密码重用天数，仅限制密码重用次数。
- 当**password_reuse_max**为0时，表示不限制密码重用次数，仅限制密码重用天数。
- 当**password_reuse_time**和**password_reuse_max**都为0时，表示不对密码重用进行限制。

取值范围：整型，最小值为0，最大值为1000。

- 0表示不检查密码可重用次数。
- 正整数表示新密码不能为该值指定的次数内使用过的密码。

默认值：0

password_lock_time

参数说明：该参数指定账户被锁定后自动解锁的时间。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

password_lock_time和**failed_login_attempts**必须都为正数时锁定和解锁功能才能生效。

取值范围：浮点型，最小值为0，最大值为365，单位为天。

- 0表示密码验证失败时，自动锁定功能不生效。
- 正数表示账户被锁定后，当锁定时间超过password_lock_time设定的值时，账户将会被自行解锁。

默认值：1

failed_login_attempts

参数说明：在任意时候，如果输入密码错误的次数达到failed_login_attempts则当前账户被锁定，password_lock_time秒后被自动解锁，仅sysadmin用户可以访问。例如，登录时输入密码失败，ALTER USER时修改密码失败等。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

failed_login_attempts和**password_lock_time**必须都为正数时锁定和解锁功能才能生效。

取值范围：整型，最小值为0，最大值为1000。

- 0表示自动锁定功能不生效。
- 正整数表示当错误密码次数达到failed_login_attempts设定的值时，当前账户将被锁定。

默认值：10

password_encryption_type

参数说明：该字段决定采用何种加密方式对用户密码进行加密存储。修改此参数的配置不会自动触发已有用户密码加密方式的修改，只会影响新创建用户或修改用户密码操作。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：0、1、2、3

- 0表示采用md5方式对密码加密。
- 1表示采用sha256和md5两种方式分别对密码加密。

- 2表示采用sha256方式对密码加密。
- 3表示采用sm3方式对密码加密。

须知

MD5加密算法安全性低，存在安全风险，不建议使用。

默认值：2

password_min_length

参数说明：该字段决定账户密码的最小长度，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，6~999个字符。

默认值：8

password_max_length

参数说明：该字段决定账户密码的最大长度，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，6~999个字符。

默认值：32

password_min_uppercase

参数说明：该字段决定账户密码中至少需要包含大写字母个数，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~999

- 0表示没有限制。
- 1~999表示创建账户所指定的密码中至少需要包含大写字母个数。

默认值：0

password_min_lowercase

参数说明：该字段决定账户密码中至少需要包含小写字母的个数，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~999

- 0表示没有限制。
- 1~999表示创建账户所指定的密码中至少需要包含小写字母个数。

默认值：0

password_min_digital

参数说明：该字段决定账户密码中至少需要包含数字的个数，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~999

- 0表示没有限制。
- 1~999表示创建账户所指定的密码中至少需要包含数字个数。

默认值：0

password_min_special

参数说明：该字段决定账户密码中至少需要包含特殊字符个数，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~999

- 0表示没有限制。
- 1~999表示创建账户所指定的密码中至少需要包含特殊字符个数。

默认值：0

password_effect_time

参数说明：该字段决定账户密码的有效时间。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：浮点型，最小值为0，最大值为999，单位为天。

- 0表示不开启有效期限制功能。
- 1~999表示创建账户所指定的密码有效期，临近或超过有效期系统会提示用户修改密码。

默认值：0

password_notify_time

参数说明：该字段决定账户密码到期前提醒的天数。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，最小值为0，最大值为999，单位为天。

- 0表示不开启提醒功能。
- 1~999表示账户密码到期前提醒的天数。

默认值：7

ssl_cert_notify_time

参数说明：SSL服务器证书到期前提醒的天数。建立连接初始化ssl证书时，若当前时间距离证书到期时间小于设定值，则在日志中打印过期提醒。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，最小值为7，最大值为180，单位为天。

默认值：90

14.3.3.3 通信库参数

本节介绍通信库相关的参数设置及取值范围等内容。

tcp_keepalives_idle

参数说明：在支持TCP_KEEPIDLE套接字选项的系统上，设置发送活跃信号的间隔秒数。不设置发送保持活跃信号，连接就会处于闲置状态。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 如果操作系统不支持TCP_KEEPIDLE选项 ，这个参数的值必须为0。
- 在通过UNIX域套接字进行的连接的操作系统上，这个参数将被忽略。
- 将该值设置为0时，将使用系统的值。
- 该参数在不同的会话之间不共享，也就是说不同的会话连接可能有不同的值。
- 查看该参数时查出来的是当前会话连接内的参数值，而不是GUC副本的值。

取值范围：0-3600，单位为s。

默认值：60

tcp_keepalives_interval

参数说明：在支持TCP_KEEPINTVL套接字选项的操作系统上，以秒数声明在重新传输之间等待响应的时间。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：0-180，单位为s。

默认值：

须知

- 如果操作系统不支持TCP_KEEPINTVL选项，这个参数的值必须为0。
- 在通过UNIX域套接字进行的连接的操作系统上，这个参数将被忽略。
- 将该值设置为0时，将使用系统的值。
- 该参数在不同的会话之间不共享，也就是说不同的会话连接可能有不同的值。
- 查看该参数时查出来的是当前会话连接内的参数值，而不是GUC副本的值。

tcp_keepalives_count

参数说明：在支持TCP_KEEPCNT套接字选项的操作系统上，设置GaussDB服务端在断开与客户端连接之前可以等待的保持活跃信号个数。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 如果操作系统不支持TCP_KEEPCNT选项，这个参数的值必须为0。
- 在通过UNIX域套接字进行连接的操作系统上，这个参数将被忽略。
- 将该值设置为0时，将使用系统的值。
- 该参数在不同的会话之间不共享，也就是说不同的会话连接可能有不同的值。
- 查看该参数时查出来的是当前会话连接内的参数值，而不是GUC副本的值。

取值范围：0-100，其中0表示GaussDB未收到客户端反馈的保持活跃信号则立即断开连接。

默认值：20

tcp_user_timeout

参数说明：在支持TCP_USER_TIMEOUT套接字选项的操作系统上，设置GaussDB在发送数据时，指定传输的数据在TCP连接被强制关闭之前可以保持未确认状态的最大时长。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 如果操作系统不支持TCP_USER_TIMEOUT选项，这个参数的值将不生效，默认为0。
- 在通过UNIX域套接字进行连接的操作系统上，这个参数将被忽略。

取值范围：0-3600000，单位为ms。其中0表示跟随操作系统设置。

默认值：0

注意，不同操作系统内核下，这个参数生效结果将不同：

- aarch64 EulerOS（Linux内核版本：4.19），超时时间即为该参数设置值。
- x86 Euler2.5（Linux内核版本：3.10），超时时间不是该参数设置值，而是不同区间的最大值，即超时时间取值为：tcp_user_timeout设置值所处“Linux TCP重传总耗时”区间的上限最大值。例如：tcp_user_timeout=40000时，重传总耗时为51秒。

  表 14-4 x86 Euler2.5（Linux 内核版本：3.10）tcp_user_timeout 参数取值示意

Linux TCP重传次数	Linux TCP重传总耗时区间（秒）	tcp_user_timeout设置举例（毫秒）	实际Linux TCP重传总耗时（秒）
1	(0.2,0.6]	400	0.6
2	(0.6,1.4]	1000	1.4
3	(1.4,3]	2000	3
4	(3,6.2]	4000	6.2
5	(6.2,12.6]	10000	12.6
6	(12.6,25.4]	20000	25.4
7	(25.4,51]	40000	51
8	(51,102.2]	80000	102.2
9	(102.2,204.6]	150000	204.6
10	(204.6,324.6]	260000	324.6
11	(324.6,444.6]	400000	444.6

  注：TCP每次重传耗时随重传次数指数增加，当TCP一次重传到达120秒后，后续每次重传都将耗时120秒不再变化。

comm_proxy_attr

参数说明：通信代理库相关参数配置。

说明

- 该参数仅支持欧拉2.9系统下的集中式ARM单机。
- 本功能在线程池开启状态下生效，即enable_thread_pool为on。
- 配置该参数时需同步配置GUC参数local_bind_address为libos_kni的网卡IP。
- 参数模板：comm_proxy_attr = '{enable_libnet:true, enable_dfx:false, numa_num:4, numa_bind:[[30,31],[62,63],[94,95],[126,127]]}'
- 可配置参数说明。
  - enable_libnet：是否开启用户态协议，取值范围：true、false。
  - enable_dfx：是否开启通信代理库视图，取值范围：true、false。
  - numa_num：机器环境中numa的数量，支持2P、4P服务器，取值范围：4、8。
  - numa_bind：代理线程绑核参数，每个numa两个CPU绑核，共numa_num组，取值范围：[0，cpu数-1]。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，长度大于0。

默认值：'none'

14.3.4 资源消耗

14.3.4.1 内存

介绍与内存相关的参数设置。

须知

这些参数只能在数据库服务重新启动后生效，local_syscache_threshold除外。

memorypool_enable

参数说明：设置是否允许使用内存池。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许使用内存池。
- off表示不允许使用内存池。

默认值：off

memorypool_size

参数说明：设置内存池大小。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，128*1024～INT_MAX/2，单位为KB。

默认值：512MB

enable_memory_limit

参数说明：启用逻辑内存管理模块。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示启用逻辑内存管理模块。
- off表示不启用逻辑内存管理模块。

默认值：on

⚠ 注意

- GaussDB强制把enable_memory_limit设置为off。其中元数据是GaussDB内部使用的内存，和部分并发参数，如max_connections，thread_pool_attr，max_prepared_transactions等参数相关。
- 当该值为off时，不对数据库使用的内存做限制，在大并发或者复杂查询时，使用内存过多，可能导致操作系统OOM问题。

max_process_memory

参数说明：设置一个数据库节点可用的最大物理内存。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，2*1024*1024～INT_MAX，单位为KB。

默认值：

1400GB（196核CPU/1536G内存）；900GB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存）；680GB（96核CPU/768G内存）；560GB（80核CPU/640G内存）；450GB（64核CPU/512G内存）；420GB（60核CPU/480G内存）；200GB（32核CPU/256G内存）；90GB（16核CPU/128G内存）；40GB（8核CPU/64G内存）；20GB（4核CPU/32G内存）；10GB（4核CPU/16G内存）

设置建议：

数据库节点上该数值需要根据系统物理内存及单节点部署主数据库节点个数决定。建议计算公式如下：(物理内存大小 - vm.min_free_kbytes) * 0.7 / (1 + 主节点个数)。该系数的目的是尽可能保证系统的可靠性，不会因数据库内存膨胀导致节点OOM。这个公式中提到vm.min_free_kbytes，其含义是预留操作系统内存供内核使用，通常用作操作系统内核中通信收发内存分配，至少为5%内存。即，max_process_memory = 物理内存 * 0.665 / (1 + 主节点个数)。

⚠ 注意

当该值设置不合理，即大于服务器物理内存，可能导致操作系统OOM问题。

enable_memory_context_control

参数说明：启用检查内存上下文是否超过给定限制的功能。仅适用于DEBUG版本。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示启用最大内存上下文限制检查功能。
- off表示关闭最大内存上下文限制检查功能。

默认值：off

uncontrolled_memory_context

参数说明：启用检查内存上下文是否超过给定限制的功能时，设置不受此功能约束。仅适用于DEBUG版本。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

查询时会在参数值的最前面添加标题含义字符串“MemoryContext white list:"。

取值范围：字符串

默认值：空

shared_buffers

参数说明：设置GaussDB使用的共享内存大小。增加此参数的值会使GaussDB比系统默认设置需要更多的System V共享内存。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，16 ~ 1073741823，单位为8KB。

shared_buffers需要设置为BLCKSZ的整数倍，BLCKSZ目前设置为8KB，即shared_buffers需要设置为8KB整数倍。改变BLCKSZ的值会改变最小值。

默认值：

560GB（196核CPU/1536G内存）；360GB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存）；270GB（96核CPU/768G内存）；220GB（80核CPU/640G内存）；180GB（64核CPU/512G内存）；160GB（60核CPU/480G内存）；80GB（32核CPU/256G内存）；36GB（16核CPU/128G内存）；16GB（8核CPU/64G内存）；8GB（4核CPU/32G内存）；4GB（4核CPU/16G内存）

设置建议：

1. 建议设置shared_buffers值为内存的40%以内。
2. 如果设置较大的shared_buffers需要同时增加checkpoint_segments的值，因为写入大量新增、修改数据需要消耗更多的时间周期。
3. 如果调整shared_buffers参数之后，导致进程重启失败，请参考启动失败的报错信息，采用以下解决方案之一：
   a. 对应调整操作系统kernel.shmall、kernel.shmmax、kernel.shmmin参数，调整方式请参考《安装指南》的配置操作系统其他参数小节。
   b. 执行free -g观察操作系统可用内存和swap空间是否足够，如果内存明显不足，请手动停止其他比较占用内存的用户程序。
   c. 避免设置明显不合理（过大或过小）的shared_buffers值。

segment_buffers

参数说明：预留参数，暂不支持。

bulk_write_ring_size

参数说明：大批量数据写入触发时（例如copy动作），该操作使用的环形缓冲区大小。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，16384 ~ 2147483647，单位为KB。

默认值：2GB

设置建议：建议导入压力大的场景中增加数据库节点中此参数配置。

standby_shared_buffers_fraction

参数说明：备实例所在服务器使用shared_buffers内存缓冲区大小的比例。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：双精度浮点型，0.1~1.0

默认值：1

temp_buffers

参数说明：设置每个数据库会话使用的LOCAL临时缓冲区的大小。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

在每个会话的第一次使用临时表之前可以改变temp_buffers的值，之后的设置将是无效的。

一个会话将按照temp_buffers给出的限制，根据需要分配临时缓冲区。如果在一个并不需要大量临时缓冲区的会话里设置一个大的数值，其开销只是一个缓冲区描述符的大小。当缓冲区被使用，就会额外消耗8192字节。

取值范围：整型，100~1073741823，单位为8KB。

默认值：1MB

max_prepared_transactions

参数说明：设置可以同时处于"预备"状态的事务的最大数目。增加此参数的值会使GaussDB比系统默认设置需要更多的System V共享内存。

当GaussDB部署为主备双机时，在备机上此参数的设置必须要高于或等于主机上的，否则无法在备机上进行查询操作。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~262143。

默认值：

200（196核CPU/1536G内存，128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，64核CPU/512G内存，60核CPU/480G内存，32核CPU/256G内存，16核CPU/128G内存，8核CPU/64G内存，4核CPU/32G内存）；0（4核CPU/16G内存）

说明

一般不需要对事务显式进行PREPARE操作，如果业务有对事务进行显式PREPARE操作，为避免在准备步骤失败，需调大该值，大于需要进行PREPARE业务的并发数。

work_mem

参数说明：设置内部排序操作和Hash表在开始写入临时磁盘文件之前使用的内存大小。ORDER BY，DISTINCT和merge joins都要用到排序操作。Hash表在散列连接、散列为基础的聚集、散列为基础的IN子查询处理中都要用到。

对于复杂的查询，可能会同时并发运行好几个排序或者散列操作，每个都可以使用此参数所声明的内存量，不足时会使用临时文件。同样，好几个正在运行的会话可能会同时进行排序操作。因此使用的总内存可能是work_mem的好几倍。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，64~2147483647，单位为KB。

默认值：

280MB（196核CPU/1536G内存）；256MB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存）；128MB（80核CPU/640G内存，64核CPU/512G内存，60核CPU/480G内存，32核CPU/256G内存，16核CPU/128G内存）；64MB（8核CPU/64G内存）；32MB（4核CPU/32G内存）；16MB（4核CPU/16G内存）

须知

设置建议：

依据查询特点和并发来确定，一旦work_mem限定的物理内存不够，算子运算数据将写入临时表空间，带来5-10倍的性能下降，查询响应时间从秒级下降到分钟级。

- 对于串行无并发的复杂查询场景，平均每个查询有5-10关联操作，建议work_mem=50%内存/10。
- 对于串行无并发的简单查询场景，平均每个查询有2-5个关联操作，建议work_mem=50%内存/5。
- 对于并发场景，建议work_mem=串行下的work_mem/物理并发数。
- 对于BitmapScan的哈希表也会受到work_mem的限制，但不会被严格管控下盘。完全Lossify的情况下，哈希表每占用1MB的内存，对应一次BitmapHeapScan的16GB的页面（Ustore为32GB），达到work_mem上限后，会按此比例随数据访问量线性增长。

query_mem

参数说明：设置执行作业所使用的内存。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：0，或大于32M的整型，默认单位为KB。

默认值：0

须知

- 如果设置的query_mem值大于0，在生成执行计划时，优化器会将作业的估算内存调整为该值。
- 如果设置值为负数或小于32MB，将设置为默认值0，此时优化器不会根据该值调整作业的估算内存。

query_max_mem

参数说明：设置执行作业所能够使用的最大内存。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：0，或大于32M的整型，默认单位为KB。

默认值：0

须知

- 如果设置的query_max_mem值大于0，当作业执行时所使用内存超过该值时，将报错退出。
- 如果设置值为负数或小于32M，将设置为默认值0，此时不会根据该值限制作业的内存使用。

maintenance_work_mem

参数说明：设置在维护性操作（比如VACUUM、CREATE INDEX、ALTER TABLE ADD FOREIGN KEY等）中可使用的最大的内存。该参数的设置会影响VACUUM、VACUUM FULL、CLUSTER、CREATE INDEX的执行效率。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1024~INT_MAX，单位为KB。

默认值：

2GB（196核CPU/1536G内存，128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，64核CPU/512G内存，60核CPU/480G内存）；1GB（32核CPU/256G内存）；512MB（16核CPU/128G内存）；256MB（8核CPU/64G内存）；128MB（4核CPU/32G内存）；64MB（4核CPU/16G内存）

须知

设置建议：

- 建议设置此参数的值大于**work_mem**，可以改进清理和恢复数据库转储的速度。因为在一个数据库会话里，任意时刻只有一个维护性操作可以执行，并且在执行维护性操作时不会有太多的会话。
- 当**自动清理**线程运行时，**autovacuum_max_workers**倍数的内存将会被分配，所以此时设置maintenance_work_mem的值应该不小于**work_mem**。
- 如果进行大数据量的cluster等，可以在session中调大该值。

max_stack_depth

参数说明：设置GaussDB执行堆栈的最大安全深度。需要这个安全界限是因为在服务器里，并非所有程序都检查了堆栈深度，只是在可能递规的过程，比如表达式计算这样的过程里面才进行检查。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，100~INT_MAX，单位为KB。

默认值：

- （ulimit -s的设置）- 640 KB的值大于等于2MB时，此参数的默认值为2MB。
- （ulimit -s的设置）- 640 KB的值小于2MB时，此参数的默认值为（ulimit -s的设置）- 640 KB。

须知

设置原则：

- 数据库需要预留640KB堆栈深度，因此，此参数的最佳设置是等于操作系统内核允许的最大值（就是ulimit -s的设置）- 640KB。
- 数据库未运行前设置的该参数值大于（ulimit -s的设置）- 640 KB时会导致数据库启动失败；数据库运行阶段设置该参数值大于（ulimit -s的设置）- 640 KB时该值不生效。
- 若（ulimit -s的设置）-640KB小于此参数取值范围的最小值时会导致数据库启动失败。
- 如果设置此参数的值大于实际的内核限制，则一个正在运行的递归函数可能会导致一个独立的服务器进程崩溃。
- 因为并非所有的操作都能够检测，所以建议用户在此设置一个明确的值。
- 默认值最大为2MB，这个值相对比较小，不容易导致系统崩溃。

bulk_read_ring_size

参数说明：大批量数据查询时（例如大表扫描），该操作使用的环形缓冲区大小。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，256~2147483647，单位为KB。

默认值：16MB

enable_early_free

参数说明：控制是否可以实现算子内存的提前释放。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示支持算子内存提前释放。
- off表示不支持算子内存提前释放。

默认值：on

local_syscache_threshold

参数说明：系统表cache在单个session缓存的大小。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

如果enable_global_plancache已打开，为保证GPC生效，local_syscache_threshold设置值小于16MB时不会生效，最小为16MB。

如果enable_global_syscache和enable_thread_pool打开，该参数描述的是当前线程和绑定到当前线程上的session缓存的总大小。

取值范围：整型，1*1024～512*1024，单位为KB。

默认值：

32MB（196核CPU/1536G内存）；16MB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，64核CPU/512G内存，60核CPU/480G内存，32核CPU/256G内存，16核CPU/128G内存，8核CPU/64G内存，4核CPU/32G内存，4核CPU/16G内存）

memory_trace_level

参数说明：动态内存使用超过最大动态内存的90%后，记录内存申请信息的管控等级。该参数仅在use_workload_manager和enable_memory_limit打开时生效。该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：枚举型

- none：表示不记录内存申请信息。
- level1：动态内存使用超过最大动态内存的90%后，会记录以下信息，并将记录的内存信息保存在$GAUSSLOG/mem_log目录下。
  - 全局内存概况。
  - instance，session，thread三种类型的所有内存上下文中内存占用前20的内存上下文的内存使用情况。
  - 每个内存上下文的totalsize、freesize字段。
- level2：动态内存使用超过最大动态内存的90%后，会记录以下信息，并将记录的内存信息保存在$GAUSSLOG/mem_log目录下。
  - 全局内存概况。

  - instance，session，thread三种类型的所有内存上下文中内存占用前20的内存上下文的内存使用情况。
  - 每个内存上下文的totalsize，freesize字段。
  - 每个内存上下文上所有内存申请的详细信息，包含申请内存所在的文件，行号和大小。

默认值：level1

须知

- 该参数设置为level2后，会记录每个内存上下文的内存申请详情（file，line，size字段），会对性能影响较大，需慎重设置。
- 记录的内存快照信息可以通过系统函数gs_get_history_memory_detail(cstring)查询，函数详情请参考《开发指南》的“SQL参考 > 函数和操作符 > 统计信息函数”章节查询。
- 记录的内存上下文是经过将同一类型所有重名的内存上下文进行汇总之后得到的。

resilience_memory_reject_percent

参数说明：用于控制内存过载逃生的动态内存占用百分比。该参数仅在GUC参数use_workload_manager和enable_memory_limit打开时生效。该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，长度大于0

该参数分为recover_memory_percent,、overload_memory_percent 2部分，这2个部分的具体含义如下：

- recover_memory_percent：内存从过载状态恢复正常状态的动态内存使用占最大动态内存的百分比，当动态内存使用小于最大动态内存乘以该值对应的百分比后，停止过载逃生并放开新连接接入，取值为0~100，设置为多少表示百分之多少。
- overload_memory_percent：内存过载时动态内存使用占最大动态内存的百分比，当动态内存使用大于最大动态内存乘以该值对应的百分比后，表示当前内存已经过载，触发过载逃生kill会话并禁止新连接接入，取值为0~100，设置为多少表示百分之多少。

默认值：'0,0'，表示关闭内存过载逃生功能。

示例：

```
resilience_memory_reject_percent = '70,90'
```

表示内存使用超过最大内存上限的90%后禁止新连接接入并kill堆积的会话，kill会话过程中内存恢复到最大内存的70%以下时停止kill会话并允许新连接接入。

须知

- 最大动态内存和已使用的动态内存可以通过gs_total_memory_detail视图查询获得，最大动态内存：max_dynamic_memory，已使用的动态内存：dynamic_used_memory。
- 该参数如果设置的百分比过小，则会频繁触发内存过载逃生流程，会使正在执行的会话被强制退出，新连接短时间接入失败，需要根据实际内存使用情况慎重设置。
- recover_memory_percent和overload_memory_percent的值可以同时为0，除此之外，recover_memory_percent的值必须要小于overload_memory_percent，否则会设置不生效。

resilience_escape_user_permissions

参数说明：设置用户权限，以逗号分隔，可以设置多个，设置多个则表示多个特殊权限的用户都支持逃生能力，只设置一个则只针对一个特权用户进行逃生。sysadmin控制sysadmin用户的作业是否会被该逃生功能进行cancel处理；monadmin控制monadmin用户的作业是否会被该逃生功能进行cancel处理；默认为空，表示关闭sysadmin和monadmin用户的逃生能力。当前取值仅支持sysadmin，monadmin或者空字符串。该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，长度大于0。

该参数目前只支持三个取值：sysadmin，monadmin或''，这几个值的具体含义如下：

- sysadmin：控制sysadmin用户的作业是否会被该逃生功能进行cancel处理。
- monadmin：控制monadmin用户的作业是否会被该逃生功能进行cancel处理。
- ''：关闭sysadmin和monadmin用户的逃生能力。

默认值：''，关闭sysadmin和monadmin用户的逃生能力。

示例：

```
resilience_escape_user_permissions = 'sysadmin,monadmin'
```

表示同时开启sysadmin和monadmin用户的逃生功能。

须知

- 该参数可以同时设置多个值，以逗号分隔，例如resilience_escape_user_permissions = 'sysadmin,monadmin'，也可以只设置一个值，例如resilience_escape_user_permissions = 'monadmin'。
- 若该参数多次设置，以最新的设置生效。
- 该参数设置为取值范围中的任意值，普通用户都支持该逃生功能。
- 当用户同时具有sysadmin和monadmin时，resilience_escape_user_permissions必须要同时设置'sysadmin,monadmin'才能触发该用户的逃生功能。

14.3.4.2 磁盘空间

介绍与磁盘空间相关的参数，用于限制临时文件所占用的磁盘空间。

sql_use_spacelimit

参数说明：限制单个SQL在单个数据库节点上，触发落盘操作时，落盘文件的空间大小，管控的空间包括普通表、临时表以及中间结果集落盘占用的空间，对初始用户不生效。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，-1~2147483647，单位为KB。其中-1表示没有限制。

默认值：-1

temp_file_limit

参数说明：限制一个会话中，触发下盘操作时，下盘文件占用的空间大小。例如一次会话中，排序和哈希表使用的临时文件，或者游标占用的临时文件。

此设置为会话级别的下盘文件控制。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

SQL查询执行时使用的临时表空间不在此限制。

取值范围：整型，-1~2147483647，单位为KB。其中-1表示没有限制。

默认值：-1

14.3.4.3 内核资源使用

介绍与操作系统内核相关的参数，这些参数是否生效依赖于操作系统的设置。

max_files_per_process

参数说明：设置每个服务器进程允许同时打开的最大文件数目。如果操作系统内核强制一个合理的数目，则不需要设置。

但是在一些平台上（特别是大多数BSD系统），内核允许独立进程打开比系统真正可以支持的数目大得多的文件数。如果用户发现有的“Too many open files”这样的失败现象，请尝试缩小这个设置。通常情况下需要满足，系统FD（file descriptor）数量>=最大并发数*数据库节点个数*max_files_per_process*3。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，25~2147483647。

默认值：1024

shared_preload_libraries

参数说明：此参数用于声明一个或者多个在服务器启动的时候预先装载的共享库，多个库名称之间用逗号分隔，仅sysadmin用户可以访问。比如'$libdir/mylib'会在加载标准库目录中的库文件之前预先加载mylib.so（某些平台上可能是mylib.sl）库文件。

可以用这个方法预先装载GaussDB的存储过程库，通常是使用'$libdir/plXXX'语法。XXX只能是pgsql，perl，tcl，python之一。

通过预先装载一个共享库并在需要的时候初始化它，可以避免第一次使用这个库的加载时间。但是启动每个服务器进程的时间可能会增加，即使进程从来没有使用过这些库。因此建议对那些将被大多数会话使用的库才使用这个选项。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 如果被声明的库不存在，GaussDB服务将会启动失败。
- 每一个支持GaussDB的库都有一个特殊的标记用于保证兼容性。因此，不支持GaussDB的库不能用这种方法加载。

取值范围：字符串

默认值：security_plugin

14.3.4.4 基于开销的清理延迟

这个特性的目的是允许管理员减少VACUUM和ANALYZE语句在并发活动的数据库上的I/O影响。比如，像VACUUM和ANALYZE这样的维护语句并不需要迅速完成，并且不希望它们严重干扰系统执行其他的数据库操作。基于开销的清理延迟为管理员提供了一个实现这个目的手段。

须知

有些清理操作会持有关键的锁，这些操作应该尽快结束并释放锁。所以GaussDB的机制是，在这类操作过程中，基于开销的清理延迟不会发生作用。为了避免在这种情况下的长延时，实际的开销限制取下面两者之间的较大值：

- vacuum_cost_delay*accumulated_balance/vacuum_cost_limit
- vacuum_cost_delay*4

背景信息

在ANALYZE | ANALYSE（详见《开发指南》中“SQL参考 > SQL语法 > ANALYZE | ANALYSE”章节）和VACUUM（详见《开发指南》中“SQL参考 > SQL语法 > VACUUM”章节）语句执行过程中，系统维护一个内部的记数器，跟踪所执行的各种I/O操作的近似开销。如果积累的开销达到了vacuum_cost_limit声明的限制，则执行这个操作的进程将睡眠vacuum_cost_delay指定的时间。然后它会重置记数器然后继续执行。

这个特性是缺省关闭的。如需开启，需要把vacuum_cost_delay变量设置为一个非零值。

vacuum_cost_delay

参数说明：指定开销超过vacuum_cost_limit的值时，进程睡眠的时间。

要注意在许多系统上，睡眠的有效分辨率是10毫秒。因此把vacuum_cost_delay设置为一个不是10的整数倍的数值与将它设置为下一个10的整数倍作用相同。

此参数一般设置较小，常见的设置是10或20毫秒。调整此特性资源占用率时，最好是调整其他参数，而不是此参数。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~100，单位为ms，正数值表示打开基于开销的清理延迟特性；0表示关闭基于开销的清理延迟特性。

默认值：1

vacuum_cost_page_hit

参数说明：清理一个在共享缓存里找到的缓冲区的预计开销。表示锁住缓冲池、查找共享的Hash表、扫描页面内容的开销。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~10000。

默认值：1

vacuum_cost_page_miss

参数说明：清理一个要从磁盘上读取的缓冲区的预计开销。表示锁住缓冲池、查找共享Hash表、从磁盘读取需要的数据块、扫描它的内容的开销。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~10000。

默认值：10

vacuum_cost_page_dirty

参数说明：清理修改一个原先是干净的块的预计开销。表示把一个脏的磁盘块再次刷新到磁盘上的额外开销。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~10000

默认值：20

vacuum_cost_limit

参数说明：设置清理进程休眠的开销限制。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1~10000。

默认值：1000

14.3.4.5 后端写进程

介绍后端写（background writer）进程的参数配置。后端写进程的功能就是把共享缓冲区中的脏数据（指共享缓冲区中新增或者修改的内容）写入到磁盘。目的是让数据

库进程在进行用户查询时可以很少或者几乎不等待写动作的发生（写动作由后端写进程完成）。

此机制同样也减少了检查点造成的性能下降。后端写进程将持续的把脏页面刷新到磁盘上，所以在检查点到来的时候，只有几个页面需要刷新到磁盘上。但是这样还是增加了I/O的总净负荷，因为以前的检查点间隔里，一个重复弄脏的页面可能只会冲刷一次，而同一个间隔里，后端写进程可能会写好几次。在大多数情况下，连续的低负荷要比周期性的尖峰负荷好，但是在本节讨论的参数可以用于按实际需要调节其行为。

bgwriter_delay

参数说明：设置后端写进程写"脏"共享缓冲区之间的时间间隔。每一次，后端写进程都会为一些脏的缓冲区发出写操作，全量checkpoint模式用bgwriter_lru_maxpages参数控制每次写的量，然后休眠bgwriter_delay毫秒后才再次启动；增量checkpoint模式下，根据设定candidate_buf_percent_target计算目标空闲缓冲页面个数，不足时每隔bgwriter_delay毫秒刷一批页面下盘，刷页个数根据目标差距百分比计算，会根据max_io_capacity限制最大数量。

在许多系统上，休眠延时的有效分辨率是10毫秒。因此，设置一个不是10的倍数的数值与把它设置为下一个10的倍数是一样的效果。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，10~10000，单位为毫秒。

默认值：2s

设置建议：在数据写压力比较大的场景中可以尝试减小该值以降低checkpoint的压力。

candidate_buf_percent_target

参数说明：设置用于增量检查点打开时，候选buffer链中可用buffer数目占据shared_buffer内存缓冲区百分比的期望值，当前候选链中的数目少于目标值时，bgwriter线程会启动将满足条件的脏页刷盘。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：双精度浮点类型，0.1～0.85

默认值：0.3

bgwriter_lru_maxpages

参数说明：设置后端写进程每次可写入磁盘的“脏”缓存区的个数。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0～1000

说明

此参数设置为0表示禁用后端写功能，禁用后端写功能不会对checkpoints产生影响。

默认值：100

bgwriter_lru_multiplier

参数说明：通过与已使用缓存区数目的乘积评估下次服务器需要的缓存区数目。

写“脏”缓存区到磁盘的数目取决于服务器最近几次使用的缓存区数目。最近的buffers数目的平均值乘以bgwriter_lru_multiplier是为了评估下次服务器进程需要的buffers数目。在有足够多的干净的、可用的缓存区之前，后端写进程会一直写“脏”缓存区的（每次写的缓存区数目不会超过bgwriter_lru_maxpages的值）。

设置bgwriter_lru_multiplier的值为1.0表示一种“实时”策略，其作用是精准预测下次写“脏”缓冲区的数目。设置为较大的值可以应对突然的需求高峰，而较小的值则可以让服务器进程执行更多的写操作。

设置较小的bgwriter_lru_maxpages和bgwriter_lru_multiplier会减小后端写进程导致的额外I/O开销，但是服务器进程必须自己发出写操作，增加了对查询的响应时间。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：浮点型，0~10。

默认值：2

pagewriter_thread_num

参数说明：设置用于增量检查点打开后后台刷页的线程数，主要是按照脏页置脏的顺序刷盘，用于推进recovery点。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1～16

默认值：4

dirty_page_percent_max

参数说明：设置用于增量检查点打开后脏页数量占shared_buffers的百分比。达到这个设定值时，后台刷页线程将以设置的max_io_capacity计算出的最大值刷脏页。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：浮点型，0.1～1

默认值：0.9

pagewriter_sleep

参数说明：设置用于增量检查点打开后，pagewriter线程每隔pagewriter_sleep的时间刷一批脏页下盘。当脏页占据shared_buffers的比例达到dirty_page_percent_max时，每批页面数量以设定的max_io_capacity计算出的值刷页，其余情况每批页面数量按比例相对减少。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0～3600000（毫秒）

默认值：2000ms（2s）

max_io_capacity

参数说明：设置后端写进程批量刷页每秒的I/O上限，需要根据具体业务场景和机器磁盘I/O能力进行设置。要求RTO很短时间或者数据量比共享内存大多倍的情况，业务访问数据量又是随机访问时，该值不宜过小。该参数设置较小会减小后端写进程刷页个数，如果业务触发页面淘汰多时，该值设置小会影响业务。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，30720~10485760。单位是KB。

默认值：512000KB（500MB）

enable_consider_usecount

参数说明：设置backend线程在页面置换时是否考虑页面热度，建议大容量场景下开启此参数。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on/true表示考虑页面热度。
- off/false表示不考虑页面热度。

默认值：off

dw_file_num

参数说明：设置批量双写文件的数量，该值与pagewriter_thread_num有关，不会大于pagwriter_thread_num，如果设置过大，内部会纠正为pagewriter_thread_num。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1～16

默认值：1

dw_file_size

参数说明：设置每个批量双写文件的大小。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，32～256

默认值：256

14.3.4.6 异步 I/O

checkpoint_flush_after

参数说明：设置checkpointer线程刷页个数超过设定的阈值时，告知操作系统开始将操作系统缓存中的页面异步刷盘。GaussDB中，磁盘页大小为8KB。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~256（0表示关闭异步刷盘功能）。例如，取值32，表示checkpointer线程连续写32个磁盘页，即32*8=256KB磁盘空间后会进行异步刷盘。

默认值：256KB

bgwriter_flush_after

参数说明：设置background writer线程刷页个数超过设定的阈值时，告知操作系统开始将操作系统缓存中的页面异步刷盘。GaussDB中，磁盘页大小为8KB。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~256（0表示关闭异步刷盘功能），单位页面（8KB）。例如，取值64，表示background writer线程连续写64个磁盘页，即64*8=512KB磁盘空间后会进行异步刷盘。

默认值：512KB（即64个页面）

backend_flush_after

参数说明：设置backend线程刷页个数超过设定的阈值时，告知操作系统开始将操作系统缓存中的页面异步刷盘。GaussDB中，磁盘页大小为8KB。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~256（0表示关闭异步刷盘功能），单位页面（8KB）。例如，取值64，表示backend线程连续写64个磁盘页，即64*8=512KB磁盘空间后会进行异步刷盘。

默认值：0

14.3.5 数据导入导出

介绍导入导出的相关参数。

raise_errors_if_no_files

参数说明：导入时是否区分“导入文件记录数为空”和“导入文件不存在”。raise_errors_if_no_files=TRUE，则“导入文件不存在”的时候，GaussDB将抛出“文件不存在的”错误。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示导入时区分“导入文件记录数为空”和“导入文件不存在”。
- off表示导入时不区分“导入文件记录数为空”和“导入文件不存在”。

默认值：off

partition_mem_batch

参数说明：为了优化对列存分区表的批量插入，在批量插入过程中会对数据进行缓存后再批量写盘。通过partition_mem_batch可指定缓存个数。该值设置过大，将消耗较多系统内存资源；设置过小，将降低系统列存分区表批量插入性能。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：1~ 65535

默认值：256

partition_max_cache_size

参数说明：为了优化对列存分区表的批量插入，在批量插入过程中会对数据进行缓存后再批量写盘。通过partition_max_cache_size可指定数据缓存区大小。该值设置过大，将消耗较多系统内存资源；设置过小，将降低列存分区表批量插入性能。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：

列存分区表：4096~ INT_MAX / 2，最小单位为KB。

默认值：2GB

enable_delta_store

参数说明：为了增强列存单条数据导入的性能和解决磁盘冗余问题，可通过此参数选择是否开启支持列存delta表功能。该参数开启时，数据导入列存表，会根据表定义时指定的《开发指南》中“SQL参考 > SQL语法 > CREATE TABLE”章节中的DELTAROW_THRESHOLD内容决定数据进入delta表存储还是主表CU存储，当数据量小于DELTAROW_THRESHOLD时，数据进入delta表。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：

- on表示开启列存delta表功能。
- off表示不开启列存delta表功能。

默认值：off

safe_data_path

参数说明：设置初始用户以外的路径前缀限制，目前包括copy和高级包路径限制。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串（不超过4096个字符）

默认值：NULL

⚠ 注意

- 如果safe_data_path目录下存在软链接文件，则会按软链接实际指向的文件路径进行处理，实际路径如果不在safe_data_path下会报错处理。
- 如果safe_data_path目录下存在硬链接文件，则可以正常使用。为安全起见，请谨慎使用硬链接文件，切勿在safe_data_path目录下创建指向目录以外的硬链接文件，并确保safe_data_path目录权限最小化。

enable_copy_server_files

参数说明：是否开启copy服务器端文件的权限。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启copy服务端文件的权限。
- off表示不开启copy服务端文件的权限。

默认值：off

须知

当参数enable_copy_server_files关闭时，只允许初始用户执行COPY FROM FILENAME或COPY TO FILENAME命令，当参数enable_copy_server_files打开，允许具有SYSADMIN权限的用户或继承了内置角色gs_role_copy_files权限的用户执行。

14.3.6 预写式日志

14.3.6.1 设置

wal_level

参数说明：设置写入WAL信息量的级别，不能为空或被注释掉。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 如果需要启用WAL日志归档和主备机的数据流复制，必须将此参数设置为archive、hot_standby或者logical。
- 如果此参数设置为minimal，archive_mode必须设置为off，hot_standby必须设置为off，max_wal_senders参数设置为0，且需为单机环境，否则将导致数据库无法启动。
- 如果此参数设置为archive，hot_standby必须设置为off，否则将导致数据库无法启动。但是，hot_standby在双机环境中不能设置为off，具体参见hot_standby参数说明。

取值范围：枚举类型

- minimal

  优点：一些重要操作（包括创建表、创建索引、簇操作和表的复制）都能安全的跳过，这样就可以使操作变得更快。

  缺点：WAL仅提供从数据库服务器崩溃或者紧急关闭状态恢复时所需要的基本信息，无法用WAL归档日志恢复数据。
- archive

这个参数增加了WAL归档需要的日志信息，从而可以支持数据库的归档恢复。

- hot_standby
  - 这个参数进一步增加了在备机上运行的SQL查询的信息，这个参数只能在数据库服务重新启动后生效。
  - 为了在备机上开启只读查询，wal_level必须在主机上设置成hot_standby，并且备机必须打开hot_standby参数。hot_standby和archive级别之间的性能只有微小的差异，如果它们的设置对产品的性能影响有明显差异，欢迎反馈。
- logical

  这个参数表示WAL日志支持逻辑复制。

默认值：hot_standby

fsync

参数说明：设置GaussDB服务器是否使用fsync()系统函数（请参见**wal_sync_method**）确保数据的更新及时写入物理磁盘中。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 使用fsync()系统函数可以保证在操作系统或者硬件崩溃的情况下将数据恢复到一个已知的状态。
- 如果将此参数关闭，可能会在系统崩溃时无法恢复原来的数据，导致数据库不可用。

取值范围：布尔型

- on表示使用fsync()系统函数。
- off表示不使用fsync()系统函数。

默认值：on

synchronous_commit

参数说明：设置当前事务的同步方式。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

通常情况下，一个事务产生的日志的同步顺序如下：

1. 主机将日志内容写入本地内存。
2. 主机将本地内存中的日志写入本地文件系统。
3. 主机将本地文件系统中的日志内容刷盘。
4. 主机将日志内容发送给备机。
5. 备机接收到日志内容，存入备机内存。
6. 备机将备机内存中的日志写入备机文件系统。
7. 备机将备机文件系统中的日志内容刷盘。
8. 备机回放日志，完成对数据文件的增量更新。

取值范围：枚举类型

- on：表示主机事务提交需要等待备机将对应日志刷新到磁盘。
- off：表示主机事务提交无需等待主机自身将对应日志刷新到磁盘，通常也称为异步提交。
- local：表示主机事务提交需要等待主机自身将对应日志刷新到磁盘，通常也称为本地提交。
- remote_write：表示主机事务提交需要等待备机将对应日志写到文件系统（无需刷新到磁盘）。
- remote_receive：表示主机事务提交需要等待备机接收到对应日志数据（无需写入文件系统）。
- remote_apply：表示主机事务提交需要等待备机完成对应日志的回放操作。
- true：同on。
- false：同off。
- yes：同on。
- no：同off。
- 1：同on。
- 0：同off。
- 2：同remote_apply。

默认值：on

wal_sync_method

参数说明：设置向磁盘强制更新WAL数据的方法。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

如果将**fsync**关闭，这个参数的设置就没有意义，因为所有数据更新都不会强制写入磁盘。

取值范围：枚举类型

- open_datasync表示用带O_DSYNC选项的open()打开“WAL”文件。
- fdatasync表示每次提交的时候都调用fdatasync()（支持suse10和suse11）。
- fsync_writethrough表示每次提交的时候调用fsync()强制把缓冲区任何数据写入磁盘。

  说明

  由于历史原因，Windows平台支持将wal_sync_method设置为fsync_writethrough。在windows平台上fsync_writethrough和fsync等效。
- fsync表示每次提交的时候调用fsync()（支持suse10和suse11）。
- open_sync表示用带O_SYNC选项的open()写“WAL”文件（支持suse10和suse11）。

> **说明**
>
> 不是所有的平台都支持以上参数。

默认值：fdatasync

full_page_writes

参数说明：设置GaussDB服务器在检查点之后对页面的第一次修改时，是否将每个磁盘页面的全部内容写到WAL日志中。当增量检查点开关和enable_double_write同时打开时，则不使用full_page_writes。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 设置这个参数是因为在操作系统崩溃过程中可能磁盘页面只写入了一部分内容，从而导致在同一个页面中包含新旧数据的混合。在崩溃后的恢复期间，由于在WAL日志中存储的行变化信息不够完整，因此无法完全恢复该页。把完整的页面影像保存下来就可以保证页面被正确还原，代价是增加了写入WAL日志的数据量。
- 关闭此参数，在系统崩溃的时候，可能无法恢复原来的数据。如果服务器硬件的特质（比如电池供电的磁盘控制器）可以减小部分页面的写入风险，或者文件系统特性支持（比如ReiserFS 4），并且清楚知道写入风险在一个可以接受的范畴，可以关闭这个参数。

取值范围：布尔型

- on表示启用此特性。
- off表示关闭此特性。

默认值：on

wal_log_hints

参数说明：设置在检查点之后对页面的第一次修改为页面上元组hint bits的修改时，是否将整个页面的全部内容写到WAL日志中。不推荐用户修改此设置。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示整个页面全部内容写到WAL日志中。
- off表示整个页面内容不会写到WAL日志中。

默认值：on

wal_buffers

参数说明：设置用于存放WAL数据的共享内存空间的XLOG_BLCKSZ数，XLOG_BLCKSZ的大小默认为8KB。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：-1~2^{18}，最小值为-1，最大值为262144，单位为8KB。

- 如果设置为-1，表示wal_buffers的大小随着参数shared_buffers自动调整，为shared_buffers的1/64，最小值为8个XLOG_BLCKSZ，最大值为2048个XLOG_BLCKSZ，自动调整后的值小于最小值时会调整为最小值，大于最大值时会调整为最大值。
- 如果设置为其他值，当小于4时，会被默认设置为4。

默认值：

1GB（196核CPU/1536G内存，128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，64核CPU/512G内存，60核CPU/480G内存，32核CPU/256G内存）；512MB（16核CPU/128G内存）；256MB（8核CPU/64G内存）；128MB（4核CPU/32G内存）；16MB（4核CPU/16G内存）

设置建议：每次事务提交时，WAL缓冲区的内容都写入到磁盘中，因此设置为很大的值不会带来明显的性能提升。如果将它设置成几百兆，就可以在有很多即时事务提交的服务器上提高写入磁盘的性能。根据经验来说，默认值可以满足大多数的情况。

wal_writer_delay

参数说明：WalWriter进程的写间隔时间。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

如果时间过长可能造成WAL缓冲区的内存不足，时间过短会引起WAL不断写入，增加磁盘I/O负担。

取值范围：整型， 1～10000（毫秒）

默认值：200ms

commit_delay

参数说明：表示一个已经提交的数据在WAL缓冲区中存放的时间。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 设置为非 0 值时事务执行commit后不会立即写入WAL中，而仍存放在WAL缓冲区中，等待WalWriter进程周期性写入磁盘。
- 如果系统负载很高，在延迟时间内，其他事务可能已经准备好提交。但如果没有事务准备提交，这个延迟就是在浪费时间。

取值范围：整型， 0～100000（微秒），其中0表示无延迟。

默认值：0

commit_siblings

参数说明：当一个事务发出提交请求时，如果数据库中正在执行的事务数量大于此参数的值，则该事务将等待一段时间（**commit_delay**的值），否则该事务则直接写入WAL。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型， 0～1000

默认值：5

wal_block_size

参数说明：说明WAL日志段文件中日志页面的大小。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：整型，单位为Byte。

默认值：8192

wal_segment_size

参数说明：说明WAL日志段文件的大小。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：整型，单位为8KB。

默认值：16MB (2048 * 8KB)

walwriter_cpu_bind

参数说明：绑定到WAL写入线程的CPU核，与thread pool配合使用。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，-1~核数减1。

默认值：-1

wal_sender_bind_cpu_attr

参数说明：用于控制日志发送的绑核操作，仅sysadmin用户可以访问。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

- 'nobind'：线程不做绑核。
- 'cpuorderbind: 8-12': 从8号核开始一个线程绑定一个CPU，区间内核不足就不参与绑定。建议区间大小设置为大于等于参数max_wal_senders。

取值范围：字符串，长度大于0，该参数不区分大小写。

默认值：'nobind'

walwriteraux_bind_cpu

参数说明：绑定到辅助写日志线程的CPU核

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，-1~核数减1。。

默认值：-1

walwriter_sleep_threshold

参数说明：xlogflusher进入sleep之前空闲xlog刷新的次数，达到阈值会休眠。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1-50000。

默认值：500（196核CPU/1536G内存，128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，64核CPU/512G内存，60核CPU/480G内存）；50（32核CPU/256G内存，16核CPU/128G内存，8核CPU/64G内存，4核CPU/32G内存，4核CPU/16G内存）

wal_file_init_num

参数说明：WAL编写器将创建的xlog段文件的数量。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~1000000。

默认值：10（196核CPU/1536G内存，128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，64核CPU/512G内存，60核CPU/480G内存）；0（32核CPU/256G内存，16核CPU/128G内存，8核CPU/64G内存，4核CPU/32G内存，4核CPU/16G内存）

xlog_file_path

参数说明：双数据库实例共享存储场景下，xlog日志共享盘的路径。本参数在数据库系统初始化时由OM进行配置，不建议用户自行修改。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串。

默认值：NULL

xlog_file_size

参数说明：双数据库实例共享存储场景下，xlog日志共享盘的大小。本参数在数据库系统初始化时由OM进行配置，不建议用户自行修改。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：长整型，5053733504~576460752303423487，单位是字节。

默认值：549755813888

xlog_lock_file_path

参数说明：双数据库实例共享存储场景下，xlog日志共享盘抢占的锁文件的路径。本参数在数据库系统初始化时由OM进行配置，不建议用户自行修改。

该参数属于POSTMASTER类型参数，请参考表14-1中对应设置方法进行设置。

取值范围：字符串。

默认值：NULL

force_promote

参考说明：备机强切功能开关。

备机强切在数据库实例故障状态下，以丢失部分数据为代价换取数据库实例尽可能快的恢复服务；是数据库实例状态为不可用时的一个逃生方法，不建议频繁触发。如果操作者不清楚备机强切后丢失数据对业务的影响，请勿使用本功能。

使用时需要分别在DN和cmserver开启并重启数据库实例生效，备机强切功能参考《故障处理》中“应急处理 > 备机强切”章节。

取值范围：整型，0或1

0表示关闭，1表示开启

默认值：0

wal_debug

参数说明：允许输出wal相关的调试信息。仅在编译时开启WAL_DEBUG编译宏时可用。

该参数属于SUSET类型参数，请参考表14-1中对应设置方法进行设置。

取值范围：布尔类型

默认值：false

wal_flush_timeout

参数说明：遍历WalInsertStatusEntryTbl的超时时间。Xlog刷盘自适应控制的刷盘IO遍历WalInsertStatusEntryTbl等待的最大时间。

该参数属于SIGHUP类型参数，请参考表14-1中对应设置方法进行设置。

须知

如果时间过长可能造成Xlog刷盘频率降低，降低Xlog处理性能。

取值范围：整型， 0 ~ 90000000（微秒）

默认值：

2us（196核CPU/1536G内存，128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，64核CPU/512G内存，60核CPU/480G内存，32核CPU/256G内存，16核CPU/128G内存，8核CPU/64G内存，4核CPU/32G内存）；800us（4核CPU/16G内存）

wal_flush_delay

参数说明：遍历WalInsertStatusEntryTbl时，遇到WAL_NOT_COPIED状态entry时等待的时间间隔。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型， 0 ~ 90000000（微秒）

默认值：

1us（196核CPU/1536G内存，128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，64核CPU/512G内存，60核CPU/480G内存，32核CPU/256G内存，16核CPU/128G内存，8核CPU/64G内存，4核CPU/32G内存）；100us（4核CPU/16G内存）

wal_receiver_bind_cpu

参数说明：WAL receiver 线程绑定cpu的核号。

默认是-1，-1是无效值，就是不绑核。

虽然数值范围为INT_MAX，但是实际和所用设备核数相关，所设定CPU核号不存在时报错中断。

多个线程允许绑定一个核，但是性能下降，不建议。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：最大值INT_MAX ，最小值-1。

默认值：-1

wal_rec_writer_bind_cpu

参数说明：WAL rec writer 线程绑定cpu的核号。

默认是-1，-1是无效值，就是不绑核。

虽然数值范围为INT_MAX，但是实际和所用设备核数相关，所设定CPU核号不存在时报错中断。

多个线程允许绑定一个核，但是性能下降，不建议。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：最大值INT_MAX ，最小值-1。

默认值：-1

14.3.6.2 检查点

checkpoint_segments

参数说明：设置**checkpoint_timeout**周期内所保留的最少WAL日志段文件数量。每个日志文件大小为16MB。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，最小值1

提升此参数可加快大数据的导入速度，但需要结合**checkpoint_timeout**、**shared_buffers**这两个参数统一考虑。这个参数同时影响WAL日志段文件复用数量，通常情况下pg_xlog文件夹下最大的复用文件个数为2倍的checkpoint_segments个，复用的文件被改名为后续即将使用的WAL日志段文件，不会被真正删除。

默认值：1024

checkpoint_timeout

参数说明：设置自动WAL检查点之间的最长时间。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型， 30～3600（秒）

在提升**checkpoint_segments**以加快大数据导入的场景也需将此参数调大，同时这两个参数提升会加大**shared_buffers**的负担，需要综合考虑。

默认值：15min

checkpoint_completion_target

参数说明：指定检查点完成的目标。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：双精度浮点类型，0.0～1.0

默认值：0.5

说明

默认值0.5表示每个checkpoint需要在checkpoints间隔时间的50%内完成。

checkpoint_warning

参数说明：如果由于填充检查点段文件导致检查点发生的时间间隔接近这个参数表示的秒数，就向服务器日志发送一个建议增加**checkpoint_segments**值的消息。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~INT_MAX（秒），其中0表示关闭警告。

默认值：5min

推荐值：5min

checkpoint_wait_timeout

参数说明：设置请求检查点等待checkpointer线程启动的最长时间。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，2～3600（秒）

默认值：1min

enable_incremental_checkpoint

参数说明：增量检查点开关。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

默认值：on

enable_double_write

参数说明：双写开关。当增量检查点开关打开时，同时enable_double_write打开，则使用enable_double_write双写特性保护，不再使用full_page_writes防止半页写问题。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

默认值：on

incremental_checkpoint_timeout

参数说明：增量检查点开关打开之后，设置自动WAL检查点之间的最长时间。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1~3600（秒）

默认值：1min

enable_xlog_prune

参数说明：设置在任一备机断联时，主机是否根据xlog日志的大小超过参数max_size_for_xlog_prune的值而回收日志。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- 设置为on时，如果任一备机断联时，主机回收日志。
- 设置为off时，如果任一备机断联时，主机不回收日志。

默认值：on

max_size_for_xlog_prune

参数说明：在enable_xlog_prune打开时生效，工作机制如下：

1. 如果replconninfo系列guc参数配置的所有备机都连着主机，那么该参数实际不起作用。
2. 如果replconninfo系列guc参数配置的备机至少有一个没有连着主机，那么该参数生效：当主机历史日志数量大于该参数值，会强制回收。例外：在同步提交模式下（即synchronous_commit参数非local或off时），如果还存在连着的备机，那么主机会考虑保留满足多数派备机中最小日志接受位置的日志，这种情况下，保留的日志可能会多余max_size_for_xlog_prune参数值。
3. 如果有任何一个备机正在build，那么该参数不会生效，主机日志会全量保留，防止build操作由于日志回收重复失败。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0～2147483647，单位为KB

默认值：256GB

14.3.6.3 日志回放

recovery_time_target

参数说明：设置recovery_time_target秒能够让备机完成日志写入和回放。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~3600（秒）

0是指不开启日志流控，1~3600是指备机能够在recovery_time_target时间内完成日志的写入和回放，可以保证主机与备机切换时能够在recovery_time_target秒完成日志写入和回放，保证备机能够快速升主机。recovery_time_target设置时间过小会影响主机的性能，设置过大会失去流控效果。

默认值：60

recovery_max_workers

参数说明：设置最大并行回放线程个数。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~20

默认值：4（安装工具默认设置为4，以获得更好的性能）

queue_item_size

参数说明：设定每个redo replay 线程的任务队列最大长度。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：最大值65535 ，最小值1。

默认值：560

recovery_parse_workers

参数说明：是极致RTO特性中ParseRedoRecord线程的数量。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1~16

仅在开启极致RTO情况下可以设置recovery_parse_workers为>1。需要配合recovery_redo_workers使用。若同时开启recovery_parse_workers和recovery_max_workers，以开启极致RTO的recovery_parse_workers为准，并行回放特性失效。极致RTO不再自带流控，流控统一由**recovery_time_target**参数来控制。

默认值：1

说明

打开极致RTO后，备机会额外启动recovery_parse_workers * (recovery_redo_workers + 2) + 5个线程，占用更多的CPU、内存和IO资源。请根据实际硬件配置设置参数，避免因为参数设置过大，导致CPU和内存占用过高，启动异常。

recovery_redo_workers

参数说明：是极致RTO特性中每个ParseRedoRecord线程对应的PageRedoWorker数量。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1~8

需要配合recovery_parse_workers使用。在配合recovery_parse_workers使用时，只有recovery_parse_workers大于1，recovery_redo_workers参数才生效。

默认值：1

说明

从V500R001C00版本升级到V500R001C10及其后续版本后，建议根据环境的CPU个数进行参数设置，并重启DN。

表 14-5 不同 CPU 内存和部署模式下的参数设置参考

编号	CPU个数	内存	是否分布式混部	recovery_parse_workers	recovery_redo_workers	回放线程总数	备注
1	4	-	-	1	1	-	不推荐设置。
2	8	-	是	1	1	-	不推荐设置。
3	8	64	否	1	1	-	不推荐设置。
4	16	128	是	1	1	-	不推荐设置。
5	16	128	否	2	3	15	-
6	32	256	是	2	2	13	-
7	32	256	否	2	8	25	-
8	64	512	是	2	4	17	-
9	64	512	否	2	8	25	大于此规格的均按照此推荐值设置。
10	96	768	-	2	8	25	大于此规格的均按照此推荐值设置。

recovery_parallelism

参数说明：查询实际回放线程个数，该参数为只读参数，无法修改。

该参数属于POSTMASTER类型参数，受recovery_max_workers以及recovery_parse_workers参数影响，任意一值大于0时，recover_parallelism将被重新计算。

取值范围：整型，1~2147483647

默认值：1

enable_page_lsn_check

参数说明：数据页lsn检查开关。回放时，检查数据页当前的lsn是否是期望的lsn。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

默认值：on

recovery_min_apply_delay

参数说明：设置备节点回放的延迟时间。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 此参数主节点设置无效，必须设置在需要延迟的备节点上，推荐设置在异步备上，设置了延时的异步备如果升主RTO时间会比较长。
- 延迟时间是根据主服务器上事务提交的时间戳与备机上的当前时间来计算，因此需要保证主备系统时钟一致。
- 延迟时间设置过长时，可能会导致该备机XLOG文件所在的磁盘满，需要平衡考虑磁盘大小来设置延迟时间。
- 没有事务的操作不会被延迟。
- 主备切换之后，原主机若需延迟，需要再手动配置此参数。
- 当synchronous_commit被设置为remote_apply时，同步复制会受到这个延时的影响，每一个COMMIT都需要等待备机回放结束后才会返回。
- 使用这个特性也会让hot_standby_feedback被延迟，这可能导致主服务器的膨胀，两者一起使用时要小心。
- 主机执行了持有AccessExclusive锁的DDL操作，比如DROP和TRUNCATE操作，在备机延迟回放该条记录期间，在备机上对该操作对象执行查询操作会等待锁释放之后才会返回。

取值范围：整型，0~INT_MAX，单位为毫秒。

默认值：0（不增加延迟）

redo_bind_cpu_attr

参数说明：用于控制回放线程的绑核操作，仅sysadmin用户可以访问。该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，长度大于0，该参数不区分大小写。

- 'nobind'：线程不做绑核。
- 'nodebind: 1, 2'：利用NUMA组1,2中的CPU core进行绑核。
- 'cpubind: 0-30'：利用0-30号CPU core进行绑核。
- 'cpuorderbind: 16-32': 从16号核开始开始一个线程绑定一个CPU，区间内核不足就不参与绑定。建议区间大小设置为大于等于recovery_parallelism + 1。

默认值：'nobind'

14.3.6.4 归档

archive_mode

参数说明：表示是否进行归档操作。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 当**wal_level**设置成minimal时，archive_mode参数无法使用。
- 无论是同步备机还是异步备机都能够开启归档，归档开启的方式与单机开启归档一致，将archive_mode置为on，并设置正确的archive_dest或者archive_command即可。
- 若未开启最大可用模式以及有同步备机与主机断开连接时，主机会因为业务阻塞的原因无法给备机发送归档的位置，从而导致归档失败。

取值范围: 布尔型

- on表示进行归档。
- off表示不进行归档。

默认值：off

archive_command

参数说明：由管理员设置的用于归档WAL日志的命令，建议归档路径为绝对路径。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 当archive_dest和archive_command同时配置时，WAL日志优先保存到archive_dest所设置的目录中，archive_command配置的命令不生效。
- 字符串中任何%p都被要归档的文件的绝对路径代替，而任何%f都只被该文件名代替（相对路径都相对于数据目录的）。如果需要在命令里嵌入%字符就必须双写%。
- 这个命令当且仅当成功的时候才返回零。示例如下：
  ```
  archive_command = 'cp --remove-destination %p /mnt/server/archivedir/%f'
  ```
- --remove-destination选项作用为：拷贝前如果目标文件已存在，会先删除已存在的目标文件，然后执行拷贝操作。
- 如果归档命令有多条，则需将其写入SHELL脚本文件中，然后将archive_command配置为执行该脚本的命令。示例如下：
  ```
  --假设多条命令如下。
  test ! -f dir/%f && cp %p dir/%f
  --则test.sh脚本内容如下。
  test ! -f dir/$2 && cp $1 dir/$2
  --归档命令如下。
  archive_command='sh dir/test.sh %p %f'
  ```

取值范围：字符串

默认值：(disabled)

archive_dest

参数说明：由管理员设置的用于归档WAL日志的目录，建议归档路径为绝对路径。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 当archive_dest和archive_command同时配置时，WAL日志优先保存到archive_dest所设置的目录中，archive_command配置的命令不生效。
- 字符串中如果是相对路径为相对于数据目录的。示例如下。
  ```
  archive_dest = '/mnt/server/archivedir/'
  ```

取值范围：字符串

默认值：空字符串

archive_timeout

参数说明：表示归档周期。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 超过该参数设定的时间时强制切换WAL段。
- 由于强制切换而提早关闭的归档文件仍然与完整的归档文件长度相同。因此，将archive_timeout设为很小的值将导致占用巨大的归档存储空间，建议将archive_timeout设置为60秒。

取值范围：整型，0 ~ 1073741823，单位为秒。其中0表示禁用该功能。

默认值：0

archive_interval

参数说明：表示归档间隔时间。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 超过该参数设定的时间时强制归档日志文件。
- 由于归档有I/O操作，不可过于频繁地归档，也不能设置较大影响PITR的RPO建议使用默认值。

取值范围：整型，1 ~ 1000，单位为秒。

默认值：1

time_to_target_rpo

参数说明：

双数据库实例异地灾备模式下，设置主数据库实例发生异常发生时到已归档到OBS的恢复点所允许的time_to_target_rpo秒。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~3600（秒）

双数据库实例异地灾备模式下，主数据库实例日志将被归档到OBS。0是指不开启日志流控，1~3600是指设置主数据库实例发生异常发生时到已归档到OBS的恢复点所允许的time_to_target_rpo秒，保证主数据库实例因灾难崩溃时，最多可能丢失的数据的时长在允许范围内。time_to_target_rpo设置时间过小会影响主机的性能，设置过大会失去流控效果。

默认值：0

14.3.7 双机复制

14.3.7.1 发送端服务器

max_wal_senders

参数说明：指定事务日志发送进程的并发连接最大数量。不可大于等于**max_connections**。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

wal_level必须设置为archive、hot_standby或者logical以允许备机的连接。

取值范围：整型，0 ~ 1024（建议取值范围：8 ~ 100）

说明

只有当使用单DN实例无主备场景下才可以设置0。

默认值：20

wal_keep_segments

参数说明：Xlog日志文件段数量。设置“pg_xlog”目录下保留事务日志文件的最小数目，备机通过获取主机的日志进行流复制。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，2 ~ INT_MAX

默认值：128

设置建议：

- 当服务器开启日志归档或者从检查点恢复时，保留的日志文件数量可能大于wal_keep_segments设定的值。
- 如果此参数设置过小，则在备机请求事务日志时，此事务日志可能已经被产生的新事务日志覆盖，导致请求失败，主备关系断开。
- 当双机为异步传输时，以COPY方式连续导入4G以上数据需要增大wal_keep_segments配置。以T6000单板为例，如果导入数据量为50G，建议调整参数为1000。您可以在导入完成并且日志同步正常后，动态恢复此参数设置。
- 若synchronous_commit级别小于LOCAL_FLUSH，重建备机时，建议调大该参数为1000，避免重建过程中，主机日志回收导致重建失败。

wal_sender_timeout

参数说明：设置本端等待事务日志接收端接收日志的最大等待时间。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 如果主机数据较大，重建备机数据库时需要增大此参数的值，主机数据在500G时，此参数的参考值为600s。
- 此值不能大于wal_receiver_timeout或数据库重建时的超时参数。

取值范围：整型，0 ~ INT_MAX，单位为毫秒（ms）。

默认值：6s

max_replication_slots

参数说明：设置主机端的日志复制slot个数。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~1024（建议取值范围：8~100）

默认值：20

设置建议：

当使用双机复制、备份恢复、逻辑解码时，该参数值建议设为：当前物理流复制槽数+备份槽数+所需的逻辑复制槽数。如果实际设置值比上述建议值要小，那么可能造成这些功能不可用或异常。

- 物理流复制槽提供了一种自动化的方法来确保主节点在所有备节点或从备节点收到xlog之前，xlog不会被移除。也就是说物理流复制槽用于支撑主备HA。数据库所要的物理流复制槽数为备节点加从备的和与主节点之间的比例。例如，假设数据库高可用方案为1主、1备、1从备，则所需物理流复制槽数为2。假设数据库的高可用方案为1主3备，则所需物理流复制槽数为3。
- 备份槽：记录备份执行过程中的一些复制信息，全量备份和增量备份各自对应单独的备份槽，共2个。
- 目前默认不支持主备从部署方式。
- 关于逻辑复制槽数，请按如下规则考虑：
  - 一个逻辑复制槽只能解码一个数据库的修改，如果需要解码多个数据库，则需要创建多个逻辑复制槽。
  - 如果需要多路逻辑复制同步给多个目标数据库，在源端数据库需要创建多个逻辑复制槽，每个逻辑复制槽对应一条逻辑复制链路。

enable_slot_log

参数说明：是否开启复制槽主备同步特性，目前仅涉及归档槽。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启复制槽主备同步特性。
- off表示不开启复制槽主备同步特性。

默认值：on

max_changes_in_memory

参数说明：逻辑解码时单条事务在内存中缓存的DML语句数量上限。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1~2147483647

默认值：4096

max_cached_tuplebufs

参数说明：逻辑解码时总元组信息在内存中缓存的数量上限。建议设置为**max_changes_in_memory**的两倍以上。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1~2147483647

默认值：8192

logical_decode_options_default

参数说明：指定逻辑解码启动时未指定解码选项的全局默认值。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

当前支持指定的逻辑解码选项包括：parallel-decode-num, parallel-queue-size, max-txn-in-memory, max-reorderbuffer-in-memory, exclude-users。选项的意义请参考《开发指南》的“应用程序开发教程 > 基于JDBC开发 > 示例：逻辑复制代码示例”章节。

取值范围：通过逗号分隔的key=value字符串，例如：'parallel-decode-num=4,parallel-queue-size=128,exclude-users=userA'。其中空字符串表示采用程序硬编码的默认值。

默认值：""

须知

- 该参数SIGHUP生效并不会影响已经启动的逻辑解码流程；后续逻辑解码启动将使用该参数设置的选项作为其默认配置，并优先使用启动命令中指定选项的设置。
- 这里exclude-users选项和逻辑解码启动选项存在差异，不允许指定多个黑名单用户。

logical_sender_timeout

参数说明：设置本端等待逻辑日志接收端接收日志的最大等待时间。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0 ~ 2147483647，单位为毫秒（ms）。

默认值：30s

enable_wal_shipping_compression

参数说明：在流式容灾模式下设置启动跨数据库实例日志压缩功能。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

该参数仅作用于流式容灾中跨数据库实例传输的一对walsender与walreceiver中，在主数据库实例上配置。

取值范围：布尔型

- true表示打开流式容灾跨数据库实例日志压缩
- false表示关闭流式容灾跨数据库实例日志压缩

默认值：false

repl_auth_mode

参数说明：设置主备复制和备机重建的验证模式。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 如果主机上开启了UUID验证功能、且配置了非空字符串的repl_uuid验证码，那么备机也需要开启UUID验证功能、且配置相同的repl_uuid验证码，否则主备日志复制和备机重建请求将被主机拒绝。
- 该参数支持SIGHUP动态加载新值。修改之后，不影响已建连的主备连接，对后续主备复制请求和主备重建请求生效。
- 支持Quorum、DCF协议下的备机重建验证；支持Quorum协议下的主备复制验证；不支持DCF协议下的主备复制验证。
- 不支持跨数据库实例主、备之间的认证，包括Dorado主备实例和容灾主备实例。
- UUID验证功能主要为了防止主、备误连导致的数据串扰和污染，不是用于安全目的。
- 该参数不支持主、备间自动同步。

取值范围：枚举类型

- off 表示关闭UUID验证功能。
- default 表示关闭UUID验证功能。
- uuid 表示开启UUID验证功能。

默认值：default

repl_uuid

参数说明：设置用于主备UUID验证的UUID码。

该参数属于SIGHUP类型参数，请参考表14-1中对应设置方法进行设置。

须知

- 如果主机上开启了UUID验证功能、且配置了非空字符串的repl_uuid验证码，那么备机也需要开启UUID验证功能、且配置相同的repl_uuid验证码，否则主备日志复制和备机重建请求将被主机拒绝。
- 该参数支持SIGHUP动态加载新值。修改之后，不影响已建连的主备连接，对后续主备复制请求和主备重建请求生效。
- 支持Quorum、DCF协议下的备机重建验证；支持Quorum协议下的主备复制验证；不支持DCF协议下的主备复制验证。
- 不支持跨数据库实例主、备之间的认证，包括Dorado主备实例和容灾主备实例。
- UUID验证功能主要为了防止主、备误连导致的数据串扰和污染，不是用于安全目的。
- 该参数不支持主、备间自动同步。

取值范围：字符串类型。长度0 – 63个字符，字母和数字的组合，大小写不敏感，内部统一转换为小写存储。空字符串表示不启用UUID验证功能。

默认值：空字符串

replconninfo1

参数说明：设置本端侦听和鉴权的第一个节点信息。

该参数属于SIGHUP类型参数，请参考表14-1中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第一个节点信息。

默认值：空字符串

replconninfo2

参数说明：设置本端侦听和鉴权的第二个节点信息。

该参数属于SIGHUP类型参数，请参考表14-1中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第二个节点信息。

默认值：空字符串

replconninfo3

参数说明：设置本端侦听和鉴权的第三个节点信息。

该参数属于SIGHUP类型参数，请参考表14-1中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第三个节点信息。

默认值：空字符串

replconninfo4

参数说明：设置本端侦听和鉴权的第四个节点信息。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第四个节点信息。

默认值：空字符串

replconninfo5

参数说明：设置本端侦听和鉴权的第五个节点信息。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第五个节点信息。

默认值：空字符串

replconninfo6

参数说明：设置本端侦听和鉴权的第六个节点信息。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第六个节点信息。

默认值：空字符串

replconninfo7

参数说明：设置本端侦听和鉴权的第七个节点信息。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第七个节点信息。

默认值：空字符串

replconninfo8

参数说明：设置本端侦听和鉴权的第八个节点信息。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第八个节点信息。

默认值：空字符串

cross_cluster_replconninfo1

参数说明：设置跨数据库实例的本端侦听和鉴权的第一个节点信息。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第一个节点信息。

默认值：空字符串

cross_cluster_replconninfo2

参数说明：设置跨数据库实例的本端侦听和鉴权的第二个节点信息。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第二个节点信息。

默认值：空字符串

cross_cluster_replconninfo3

参数说明：设置跨数据库实例的本端侦听和鉴权的第三个节点信息。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第三个节点信息。

默认值：空字符串

cross_cluster_replconninfo4

参数说明：设置跨数据库实例的本端侦听和鉴权的第四个节点信息。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第四个节点信息。

默认值：空字符串

cross_cluster_replconninfo5

参数说明：设置跨数据库实例的本端侦听和鉴权的第五个节点信息。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第五个节点信息。

默认值：空字符串

cross_cluster_replconninfo6

参数说明：设置跨数据库实例的本端侦听和鉴权的第六个节点信息。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第六个节点信息。

默认值：空字符串

cross_cluster_replconninfo7

参数说明：设置跨数据库实例的本端侦听和鉴权的第七个节点信息。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第七个节点信息。

默认值：空字符串

cross_cluster_replconninfo8

参数说明：设置跨数据库实例的本端侦听和鉴权的第八个节点信息。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第八个节点信息。

默认值：空字符串

available_zone

参数说明：设置本端节点所在区域信息。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置节点信息。

默认值：空字符串

enable_availablezone

参数说明：设置本端级联备节点能否连接跨available_zone的备机。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- true表示级联备只能连接相同available_zone中的备机。
- false表示级联备可以连接不同available_zone中的备机。

默认值：false

max_keep_log_seg

参数说明：流控参数，逻辑复制在DN本地会解析物理日志转换成逻辑日志，当未被解析的物理日志文件数量大于该参数时会触发限流。此参数为0表示关闭限流功能。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0 ~ 2147483647。

默认值：0

enable_time_report

参数说明：设定是否记录每条redo时间开销。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示记录redo 记录生成时的时间。
- off表示不记录。

默认值：off

thread_top_level

参数说明：提高 WALWRITERAUXILIARY || WALWRITER || STARTUP ||WALRECEIVER || WAL_NORMAL_SENDER || PGSTAT线程的优先级到最高。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示上述线程优先级提高到最高。
- off表示不提高上述线程优先级。

默认值：off

wal_flush_size

参数说明：每次wal log下刷entry的阈值。

若每条entry调大会减少下刷频率，但是每次下刷的时延变大。

默认-1，表示不进行均匀下刷，每次尽可能多的下刷。设定>0的阈值表示均匀按照阈值下刷。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：最大值16777216 ，最小值-1，单位为Byte。

默认值：-1

page_work_queue_size

参数说明：每一个redo worker的阻塞队列长度。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：最大值100000 ，最小值1。

默认值：4096

14.3.7.2 主服务器

synchronous_standby_names

参数说明：潜在同步复制的备机名称列表，每个名称用逗号分隔。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 当前连接的同步备机是列表中的第一个名称。如果当前同步备机失去连接，则它会立即更换下一个优先级更高的备机，并将此备机的名称放入列表中。
- 备机名称可以通过设置环境变量PGAPPNAME指定。

取值范围：字符串。当取值为*，表示匹配任意提供同步复制的备机名称。支持按如下格式配置：

- ANY *num_sync* (*standby_name* [, ...]) [, ANY *num_sync* (*standby_name* [, ...])]
- [FIRST] *num_sync* (*standby_name* [, ...])
- *standby_name* [, ...]

说明

- 其中*num_sync*是事务需要等待其回复的同步复制的备机的数量，*standby_name*是备机的名称，FIRST以及ANY指定从所列服务器中选取同步复制的备机的策略。
- ANY N (dn_instanceId1, dn_instanceId2,...)表示在括号内任选N个主机名称作为同步复制的备机名称列表。例如，ANY 1(dn_instanceId1, dn_instanceId2)表示在dn_instanceId1和dn_instanceId2中任选一个作为同步复制的备机名称。
- FIRST N (dn_instanceId1, dn_instanceId2,...)表示在括号内按出现顺序的先后作为优先级选择前N个主机名称作为同步复制的备机名称列表。例如，FIRST 1 (dn_instanceId1, dn_instanceId2)表示选择dn_instanceId1作为同步复制的备机名称。
- dn_instanceId1, dn_instanceId2,...和FIRST 1 (dn_instanceId1, dn_instanceId2,...)具有的含义相同。

若使用gs_guc工具设置该参数，需要如下设置：

```
gs_guc reload -Z datanode -N @NODE_NAME@ -D @DN_PATH@ -c "synchronous_standby_names='ANY NODE 1(dn_instanceId1, dn_instanceId2)'";
```

或者：

```
gs_guc reload -Z datanode -N @NODE_NAME@ -D @DN_PATH@ -c "synchronous_standby_names='ANY 1(AZ1, AZ2)'";
```

默认值： *

most_available_sync

参数说明：在有同步备机故障时，主机事务不因同步备机故障而被阻塞。比如有两个同步备机，一个故障，另一个正常，这个时候主机事务只会等好的这个同步备，而不被故障的同步备所阻塞；

再比如执行quorum协议时，一主三同步备，配置ANY 2(node1,node2,node3)，当node1、node3故障，node2正常时，主机业务同样不被阻塞。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示在所有同步备机故障时，不阻塞主机。
- off表示在所有同步备机故障时，阻塞主机。

默认值：off

keep_sync_window

参数说明：延迟进入最大可用模式的时间

- 当最大可用模式most_available_sync配置为on，在主备场景下，当存在同步备发生故障, 导致不满足当前所配置的同步备数量（详细可参考synchonous_standby_name的含义）时, 如果配置了keep_sync_window参数，则在keep_sync_window设置的时间窗口内，继续保持最大保护模式，即阻塞主机的事务提交，延缓进入最大可用模式的时间。
- 若在keep_sync_window超时窗口内，同步备机故障恢复，且满足当前所配置的同步备数量, 则不阻塞事务, 恢复到正常状态.
- 如果设置keep_sync_window，推荐最小配置为5s，以避免监控系统监控到网络不稳定的误报。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，范围0~INT_MAX，单位为秒

- 0表示不设置keep_sync_window超时时间窗口，即直接进入最大可用模式。
- 其余表示keep_sync_window超时时间窗口的大小。

默认值：0

说明

配置该参数可能会对RPO造成影响，若主机在所配置的超时时间窗口内发生故障，则从开始阻塞到主机故障这段时间窗口内的数据可能丢失。

enable_stream_replication

参数说明：控制主备是否进行数据和日志同步。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 此参数属于性能测试参数，用于测试带有备机和不带备机的性能参数。关闭参数后，不能进行切换、故障等异常场景测试，否则会出现主备不一致的情况。
- 此参数属于受控参数，不建议正常业务场景下关闭此参数。
- 当前版本默认不支持主备从部署模式。

取值范围：布尔型

- on表示打开主备同步。
- off表示关闭主备同步。

默认值：on

enable_mix_replication

参数说明：控制主备之间WAL日志及数据复制的方式。

该参数属于INTERNAL类型参数，默认值为off，不允许外部修改。

须知

- 此参数目前不允许正常业务场景下改变其值，即关闭WAL日志、数据页混合复制模式。
- 当前版本默认不支持主备从部署模式。

取值范围：布尔型

- on表示打开WAL日志、数据页混合复制模式。
- off表示关闭WAL日志、数据页混合复制模式。

默认值：off

vacuum_defer_cleanup_age

参数说明：指定VACUUM使用的事务数，VACUUM会延迟清除无效的行存表记录，延迟的事务个数通过vacuum_defer_cleanup_age进行设置。即VACUUM和VACUUM FULL操作不会立即清理刚刚被删除元组。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0～1000000，值为0表示不延迟。

默认值：0

data_replicate_buffer_size

参数说明：发送端与接收端传递数据页时，队列占用内存的大小。此参数会影响主备之间复制的缓冲大小。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，4096~1072693248，单位为KB。

默认值：

128MB（196核CPU/1536G内存，128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，64核CPU/512G内存，60核CPU/480G内存，32核CPU/256G内存，16核CPU/128G内存，8核CPU/64G内存，4核CPU/32G内存）；4MB（4核CPU/16G内存）

walsender_max_send_size

参数说明：设置主机端日志或数据发送缓冲区的大小。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，8~INT_MAX，单位为KB。

默认值：8M（即8192KB）

enable_data_replicate

参数说明：当数据库在数据导入行存表时，主机与备机的数据同步方式可以进行选择。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示导入数据行存表时主备数据采用数据页的方式进行同步。当replication_type参数为1时，不允许设置为on，如果此时用guc工具设置成on，会强制改为off。
- off表示导入数据行存表时主备数据采用日志（Xlog）方式进行同步。

默认值：off

ha_module_debug

参数说明：用于查看数据复制时具体数据块的复制状态日志。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示日志中将打印数据复制时每个数据块的状态。
- off表示日志中不打印数据复制时每个数据块的状态。

默认值：off

enable_incremental_catchup

参数说明：控制主备之间数据追赶（catchup）的方式，目前默认不支持主备从部署模式。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示备机catchup时用增量catchup方式，即从从备本地数据文件扫描获得主备差异数据文件列表，进行主备之间的catchup。
- off表示备机catchup时用全量catchup方式，即从主机本地所有数据文件扫描获得主备差异数据文件列表，进行主备之间的catchup。

默认值：on

wait_dummy_time

参数说明：同时控制增量数据追赶（catchup）时，数据库主备按顺序启动时等待从机启动的最长时间以及等待从机发回扫描列表的最长时间。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，范围1~INT_MAX，单位为秒。

默认值：300

说明

- 单位只能设置为秒。
- 当前版本默认不支持主备从部署模式。

catchup2normal_wait_time

参数说明：打开最大可用模式most_available_sync，主备场景下，控制备机数据追赶（catchup）阻塞主机的最长时间。该时间为估算值，实际结果可能与参数值有偏差。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，范围-1~10000，单位为毫秒。

- -1表示主机阻塞直到备机数据追赶完成。
- 0表示备机数据追赶时始终不阻塞主机。
- 其余值表示备机数据追赶时阻塞主机的最长时间。例如，取值5000，表示当备机数据追赶完成时间还剩5s时，阻塞主机等待其完成。

默认值：-1

check_sync_standby

参数说明：打开备机检查开关，主备场景下配置了正确的synchronous_standby_names参数后，当同步备故障时，主机写业务直接报错写失败。该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：on/off

- on表示开启备机检查。
- off表示关闭备机检查。

默认值：off

说明

- 该参数不支持在job work和自治事务中同步，有可能导致检查不生效。
- 若指定用户或session中未设置备机检查，开启强同步提交模式下备机故障，执行一个表的写操作会导致另一个用户或session中的同一个表的查询hang，此时需要备机恢复或者手动terminate hang住的客户端。
- 不支持非写操作中触发写日志的场景中（vacuum analyze等）开启备机检查开关。若备机不满足同步备配置，则该场景会导致业务hang，需要手动terminate。

sync_config_strategy

参数说明：主机和备机、备机和级联备之间配置文件的同步策略。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：枚举类型

- all_node: 主机配置为all_node时，表示允许主机向所有备机主动同步配置文件；备机配置为all_node时，表示允许当前备机向其主机发送同步请求，允许当前备机向其所有级联备主动同步配置文件；级联备配置为all_node时，表示允许当前级联备向其备机发送同步请求。
- only_sync_node: 主机配置为only_sync_node时，表示仅允许主机向所有同步备机主动同步配置文件；备机配置为only_sync_node时，表示允许当前备机向其主机发送同步请求，不允许当前备机向其所有级联备主动同步配置文件；级联备配置为only_sync_node时，表示允许当前级联备向其备机发送同步请求。
- none_node: 主机配置为none_node时，表示不允许主机向任何备机主动同步配置文件；备机配置为none_node时，表示不允许当前备机向其主机发送同步请求，不允许当前备机向其所有级联备主动同步配置文件；级联备配置为none_node时，表示不允许当前级联备向其备机发送同步请求。

默认值：all_node

hadr_recovery_time_target

参数说明：在流式容灾模式下设置hadr_recovery_time_target能够让备数据库实例完成日志写入和回放。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~3600（秒）

0是指不开启日志流控，1~3600是指备机能够在hadr_recovery_time_target时间内完成日志的写入和回放，可以保证主数据库实例与备数据库实例切换时能够在

hadr_recovery_time_target秒完成日志写入和回放，保证备数据库实例能够快速升主。hadr_recovery_time_target设置时间过小会影响主机的性能，设置过大会失去流控效果。

默认值：0

hadr_recovery_point_target

参数说明：在流式容灾模式下设置hadr_recovery_point_target能够让备数据库实例完成日志刷盘的rpo时间。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~3600（秒）

0是指不开启日志流控，1~3600是指备机能够在hadr_recovery_point_target时间内完成日志的刷盘，可以保证主数据库实例与备数据库实例切换时日志差距能够在hadr_recovery_point_target秒内，保障备数据库实例升主日志量。hadr_recovery_point_target设置时间过小会影响主机的性能，设置过大会失去流控效果。

默认值：0

hadr_super_user_record_path

参数说明：该参数为流式异地容灾参数，表示备数据库实例中hadr_disaster用户的加密文件存放路径。该参数属于SIGHUP类型参数，请

参考**表14-1**中方式对应设置方法进行设置。

修改建议：由流式容灾密码传递工具自动设置，不需要用户手动添加。

取值范围：字符串

默认值：NULL

须知

- 在一个包含了主机、备机的数据库实例中，主机相对于备机是发送端，备机相对于主机是接收端。
- 发送端主动向接收端同步配置文件、接收端请求发送端同步配置文件是两个独立的事件，均会使得配置文件同步。若不希望配置文件同步，则需要在接收端配置为none_node，发送端若为备机只能配置为none_node，发送端若为主机，配置为none_node时主机与所有备机都不同步，为only_sync_node时仅与同步备同步，不与异步备同步。
- 配置参数同步的具体表现为，发送端发送配置文件，对接收端配置文件中的对应参数直接覆盖。若设置了配置文件需要同步的策略，则修改接收端配置参数后，发送端会立刻覆盖接收端的配置参数，使得接收端修改不生效。
- 即使设置了配置文件需要同步的策略，仍有部分配置参数不会被同步。包括："application_name", "archive_command", "audit_directory", "available_zone", "comm_control_port", "comm_sctp_port", "listen_addresses", "log_directory", "port", "replconninfo1", "replconninfo2", "replconninfo3", "replconninfo4", "replconninfo5", "replconninfo6", "replconninfo7", "replconninfo8", "replconninfo9", "replconninfo10", "replconninfo11", "replconninfo12", "replconninfo13", "replconninfo14", "replconninfo15", "replconninfo16", "replconninfo17", "replconninfo18", "ssl", "ssl_ca_file", "ssl_cert_file", "ssl_ciphers", "ssl_crl_file", "ssl_key_file", "ssl_renegotiation_limit", "ssl_cert_notify_time", "synchronous_standby_names", "local_bind_address", "perf_directory", "query_log_directory", "asp_log_directory", "streaming_router_port", "enable_upsert_to_merge", "archive_dest", "recovery_min_apply_delay", "sync_config_strategy"。

14.3.7.3 备服务器

hot_standby

参数说明：设置是否允许备机在恢复过程中连接和查询。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 如果此参数设置为on，**wal_level**级别必须设置为hot_standby或以上，否则将导致数据库无法启动。
- 在双机环境中，因为会对双机其他一些功能产生影响，hot_standby参数不能设置成off。
- 如果hot_standby参数曾经被关闭，且wal_level参数曾被设置低于hot_standby等级，那么，再次打开hot_standby参数之前，为了确保主备环境下备机上待回放的日志都可以支持备机查询功能，需要进行如下操作：
  1. 将主、备的wal_level参数调整到hot_standby等级或以上，并重启实例生效。
  2. 在主机上执行checkpoint操作，并通过查询pg_stat_get_wal_senders()系统函数，确认各个备机的receiver_replay_location追上主机当前的sender_flush_location，保证wal_level的调整同步到备机并生效，且备机不需要再回放之前低等级的日志。
  3. 将主、备的hot_standby参数打开（设为on），并重启实例生效。

取值范围：布尔型

- on表示允许备机在恢复过程中连接和查询。
- off表示不允许备机在恢复过程中连接和查询。

默认值：on

max_standby_archive_delay

参数说明：当开启双机热备模式时，如果备机正处理归档WAL日志数据，这时进行查询就会产生冲突，此参数就是设置备机取消查询之前所等待的时间。当前版本设置暂不生效，统一由参数max_standby_streaming_delay控制。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

-1表示允许备机一直等待冲突的查询完成。

取值范围：整型，范围：-1~INT_MAX，单位为毫秒。

默认值：3s（即3000ms）

max_standby_streaming_delay

参数说明：当开启双机热备模式时，如果备机正通过流复制接收WAL日志数据，这时进行查询就会产生冲突，这个参数就是设置备机取消查询之前所等待的时间。当参数值较大，或业务压力大时，概率出现等待事务回放落盘的报错。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

-1表示允许备机一直等待冲突的查询完成。

取值范围：整型（毫秒），范围：-1~INT_MAX

默认值：3s（即3000ms）

wal_receiver_status_interval

参数说明：设置WAL日志接收进程的状态通知给主机的最大时间间隔。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0 ~ INT_MAX ，单位为毫秒。

默认值：5s（即5000ms）

须知

当该参数设置为0时，表示关闭备机向主机反馈日志接收位置等信息，可能会导致主机事务提交阻塞、switchover操作失败等异常现象。正常业务场景，不建议将该参数设置为0。

hot_standby_feedback

参数说明：设置是否允许将备机上执行查询的结果反馈给主机，这可以避免查询冲突。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许将备机上执行查询的最小事务号反馈给主机。
- off表示不允许将备机上执行查询的最小事务号反馈给主机。

默认值：off

须知

当该参数为on时，主机的旧版本数据的清理会受限于备机正在读的事务，即主机只允许清理小于备机反馈回来的事务所作的更改。

所以，若该参数开启时，会影响主机的性能。若备机回放与查询冲突，出现查询报错，建议适当调大max_standby_streaming_delay。

wal_receiver_timeout

参数说明：设置从主机接收数据的最大等待时间。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型， 0 ~ INT_MAX，单位为毫秒。

默认值：6s（即6000ms）

wal_receiver_connect_timeout

参数说明：设置连接主机的最大等待超时时间。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型， 0 ~ INT_MAX / 1000，单位为秒。

默认值：2s

wal_receiver_connect_retries

参数说明：设置连接主机的最大尝试次数。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型， 1~ INT_MAX。

默认值：1

wal_receiver_buffer_size

参数说明：备机与从备接收Xlog存放到内存缓冲区的大小，目前默认不支持主备从部署模式。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，4096~1047552，单位为KB。

默认值：64MB（即65536KB）

primary_slotname

参数说明：设置备机对应主机的slot name，用于主备校验，与wal日志删除机制。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符型

默认值：空字符串

max_logical_replication_workers

参数说明：订阅端apply worker线程的最大数量。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~262143

默认值：4

max_standby_base_page_size

参数说明：开启极致RTO功能后，备机上最大允许的base page类型文件的存储空间大小。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：长整型，1048576~562949953421311，单位为KB。

默认值：268435456（256 GB）

max_standby_lsn_info_size

参数说明：开启极致RTO功能后，备机上最大允许的lsn info类型文件的存储空间大小。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：长整型，1048576~562949953421311，单位为KB。

默认值：268435456（256 GB）

base_page_saved_interval

参数说明：开启极致RTO功能后，备机上生成base page的间隔数。对同一个页面来说，每回放该参数值的次数，生成一次base page。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，5~2000。

默认值：400

standby_force_recycle_ratio

参数说明：开启极致RTO功能后，备机读相关文件的触发强制回收的比例。当base page文件总大小超过max_standby_base_page_size* standby_force_recycle_ratio，或者lsn info文件总大小超过max_standby_lsn_info_size* standby_force_recycle_ratio时，触发强制回收，会有查询被取消。当standby_force_recycle_ratio = 0时，不会启动强制回收，max_standby_base_page_size和max_standby_lsn_info_size也不会生效。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：双精度浮点类型，0.0～1.0。

默认值：0.8

standby_recycle_interval

参数说明：开启极致RTO功能后，备机读相关文件回收的时间间隔。备机读的资源回收线程，每间隔该参数值的时间，尝试清理一次备机读相关文件。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~86400，单位是秒（s）。

默认值：10

standby_max_query_time

参数说明：开启极致RTO功能后，支持的备机上查询的最大时间，超过该时间会被取消。注：何时取消查询受回收线程的时间间隔参数**standby_recycle_interval**和查询取快照的时间影响，因此备机上查询的实际执行时间要大于该参数。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~86400，单位是秒（s）。

默认值：600

exrto_standby_read_opt

参数说明：支持极致RTO备机读优化，默认开启。主机和备机间不同步该参数。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型，on表示开启优化，off表示关闭优化。

默认值：on

14.3.8 查询规划

介绍查询优化器方法配置、开销常量、规划算法以及一些配置参数。

说明

优化器中涉及的两个参数：

- INT_MAX数据类型INT的最大值，其值为2147483647。
- DBL_MAX数据类型FLOAT的最大值。

全局设置查询规划相关参数除了客户业务外也会对数据库自身运维和监控业务造成影响，如WDR报告生成、扩容、重分布、数据导入导出等。

14.3.8.1 优化器方法配置

这些配置参数提供了影响查询优化器选择查询规划的原始方法。如果优化器为特定的查询选择的缺省规划并不是最优的，可以通过使用这些配置参数强制优化器选择一个不同的规划来临时解决这个问题。更好的方法包括调节优化器开销常量、手动运行ANALYZE、增加配置参数default_statistics_target的值、增加使用ALTER TABLE SET STATISTICS为指定列增加收集的统计信息。

enable_broadcast

参数说明：控制优化器对stream代价估算时对broadcast分布方式的使用。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

说明

该参数在当前版本不生效。

enable_bitmapscan

参数说明：控制优化器对位图扫描规划类型的使用。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

force_bitmapand

参数说明：控制优化器强制使用bitmapand规划类型的使用。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：off

enable_hashagg

参数说明：控制优化器对Hash聚集规划类型的使用。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

enable_hashjoin

参数说明：控制优化器对Hash连接规划类型的使用。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

enable_indexscan

参数说明：控制优化器对索引扫描规划类型的使用。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

enable_indexonlyscan

参数说明：控制优化器对仅索引扫描规划类型的使用。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

enable_material

参数说明：控制优化器对实体化的使用。消除整个实体化是不可能的，但是可以关闭这个变量以防止优化器插入实体节点。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

enable_mergejoin

参数说明：控制优化器对融合连接规划类型的使用。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：off

enable_nestloop

参数说明：控制优化器对内表全表扫描嵌套循环连接规划类型的使用。完全消除嵌套循环连接是不可能的，但是关闭这个变量就会让优化器在存在其他方法的时候优先选择其他方法。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：off

enable_index_nestloop

参数说明：控制优化器对内表参数化索引扫描嵌套循环连接规划类型的使用。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

enable_seqscan

参数说明：控制优化器对顺序扫描规划类型的使用。完全消除顺序扫描是不可能的，但是关闭这个变量会让优化器在存在其他方法的时候优先选择其他方法。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

enable_sort

参数说明：控制优化器使用的排序步骤。完全消除明确的排序是不可能的，但是关闭这个变量可以让优化器在存在其他方法的时候优先选择其他方法。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

enable_tidscan

参数说明：控制优化器对TID扫描规划类型的使用。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

enable_kill_query

参数说明：CASCADE模式删除用户时，会删除此用户拥有的所有对象。此参数标识是否允许在删除用户的时候，取消锁定此用户所属对象的query。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许取消锁定。
- off表示不允许取消锁定。

默认值：off

enforce_a_behavior

参数说明：控制正则表达式的规则匹配模式。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示正则表达式采用A格式的匹配规则。
- off表示正则表达式采用POSIX格式的匹配规则。

默认值：on

max_recursive_times

参数说明：控制with recursive的最大迭代次数。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0～INT_MAX。

默认值：200

enable_vector_engine

参数说明：控制优化器对向量化执行引擎的使用。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

enable_change_hjcost

参数说明：控制优化器在Hash Join代价估算路径选择时，是否使用将内表运行时代价排除在Hash Join节点运行时代价外的估算方式。如果使用，则有利于选择条数少，但运行代价大的表做内表。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：off

enable_absolute_tablespace

参数说明：控制表空间是否可以使用绝对路径。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示可以使用绝对路径。
- off表示不可以使用绝对路径。

默认值：on

enable_valuepartition_pruning

参数说明：是否对DFS分区表进行静态/动态优化。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示对DFS分区表进行静态/动态优化。
- off表示不对DFS分区表进行静态/动态优化。

默认值：on

qrw_inlist2join_optmode

参数说明：控制是否使用inlist-to-join查询重写。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

- disable：关闭inlist2join查询重写。
- cost_base：基于代价的inlist2join查询重写。
- rule_base：基于规则的inlist2join查询重写，即强制使用inlist2join查询重写。
- 任意正整数：inlist2join查询重写阈值，即list内元素个数大于该阈值，进行inlist2join查询重写。

默认值：cost_base

skew_option

参数说明：控制是否使用优化策略。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

- off：关闭策略。
- normal：采用激进策略。对于不确定是否出现倾斜的场景，认为存在倾斜，并进行相应优化。
- lazy：采用保守策略。对于不确定是否出现倾斜场景，认为不存在倾斜，不进行优化。

默认值：normal

default_limit_rows

参数说明：设置生成genericplan的缺省limit估算行数。此参数设置为正数时意为直接将设置的值作为估算limit的行数，为负数时代表使用百分比的形式设置默认的估算值，负数转换为默认百分比，即-5代表5%。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：浮点型，-100~DBL_MAX。

默认值：-10

check_implicit_conversions

参数说明：控制是否对查询中有隐式类型转换的索引列是否会生成候选索引路径进行检查。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示对查询中有隐式类型转换的索引列是否会生成候选索引路径进行检查。
- off表示不进行相关检查。

默认值：off

cost_weight_index

参数说明：设置index_scan的代价权重。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：浮点型，1e-10~1e+10。

默认值：1

enable_default_index_deduplication

参数说明：设置btree索引默认情况下是否对键值重复的元组进行去重压缩。去重压缩功能对主键索引和唯一索引不生效。在重复键值的索引较多时，去重压缩功能可以有效降低索引占用空间。非唯一索引且索引键值重复度很低或者唯一的场景，去重压缩功能会使索引插入性能小幅度劣化。若创建索引时带有with (deduplication=on/off)语法时，优先根据deduplication参数决定该索引是否使用去重压缩功能。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- off：为默认取值，表示btree索引默认关闭索引去重压缩功能。
- on：表示btree索引默认开启索引去重压缩功能。

默认值：off

enable_expr_fusion

参数说明：控制SRF、表达式展平、取消集中式Seq Scan投影、共享聚合函数的转移状态和Step步数优化特性的开关。

该参数属于USERSET类型参数，请参考表14-1中对应设置方法进行设置。

取值范围：布尔型

- off：为默认取值，表示关闭本功能。
- on：表示同时启用SRF、表达式展平、取消集中式Seq Scan投影、共享聚合函数的转移状态和Step步数优化特性。

默认值：off

说明

只支持query_dop=1的场景。

14.3.8.2 优化器开销常量

介绍优化器开销常量。这里描述的开销可以按照任意标准度量。只关心其相对值，因此以相同的系数缩放它们将不会对优化器的选择产生任何影响。缺省时，它们以抓取顺序页的开销为基本单位。也就是说将seq_page_cost设为1.0，同时其他开销参数以它为基准设置。也可以使用其他基准，比如以毫秒计的实际执行时间。

seq_page_cost

参数说明：设置优化器计算一次顺序磁盘页面抓取的开销。

该参数属于USERSET类型参数，请参考表14-1中对应设置方法进行设置。

取值范围：浮点型，0～DBL_MAX。

默认值：1

random_page_cost

参数说明：设置优化器计算一次非顺序抓取磁盘页面的开销。

该参数属于USERSET类型参数，请参考表14-1中对应设置方法进行设置。

须知

虽然服务器允许将random_page_cost设置的比seq_page_cost小，但是物理上实际不受影响。如果所有数据库都位于随机访问内存中时，两者设置为相等很合理。因为在此种情况下，非顺序抓取页并没有副作用。同样，在缓冲率很高的数据库上，应该相对于CPU参数同时降低这两个值，因为获取内存中的页要比通常情况下开销小很多。

取值范围：浮点型，0～DBL_MAX。

默认值：4

说明

- 对于特别表空间中的表和索引，可以通过设置同名的表空间的参数来覆盖这个值。
- 相对于seq_page_cost，减少这个值将导致系统更倾向于使用索引扫描，而增加这个值使得索引扫描开销比较高。可以通过同时增加或减少这两个值来调整磁盘I/O相对于CPU的开销。

cpu_tuple_cost

参数说明：设置优化器计算在一次查询中处理每一行数据的开销。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：浮点型，0～DBL_MAX。

默认值：0.01

cpu_index_tuple_cost

参数说明：设置优化器计算在一次索引扫描中处理每条索引的开销。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：浮点型，0～DBL_MAX。

默认值：0.005

cpu_operator_cost

参数说明：设置优化器计算一次查询中执行一个操作符或函数的开销。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：浮点型，0～DBL_MAX。

默认值：0.0025

effective_cache_size

参数说明：设置优化器在一次单一的查询中可用的磁盘缓冲区的有效大小。

设置这个参数，还要考虑GaussDB的共享缓冲区以及内核的磁盘缓冲区。另外，还要考虑预计的在不同表之间的并发查询数目，因为它们将共享可用的空间。

这个参数对GaussDB分配的共享内存大小没有影响，它也不会使用内核磁盘缓冲，它只用于估算。数值是用磁盘页来计算的，通常每个页面是8192字节。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1～2147483647，单位为8KB。

比默认值高的数值可能会导致使用索引扫描，更低的数值可能会导致选择顺序扫描。

默认值：

280GB（196核CPU/1536G内存）；180GB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存）；135GB（96核CPU/768G内存）；100GB（80核CPU/640G内存）；90GB（64核CPU/512G内存）；80GB（60核CPU/480G内存）；40GB（32核CPU/256G内存）；18GB（16核CPU/128G内存）；8GB（8核CPU/64G内存）；4GB（4核CPU/32G内存）；2GB（4核CPU/16G内存）

allocate_mem_cost

参数说明：设置优化器计算Hash Join创建Hash表开辟内存空间所需的开销，供Hash join估算不准时调优使用。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：浮点型，0～DBL_MAX。

默认值：0

14.3.8.3 基因查询优化器

介绍基因查询优化器相关的参数。基因查询优化器（GEQO）是一种启发式的查询规划算法。这个算法减少了对复杂查询规划的时间，而且生成规划的开销有时也小于正常的详尽的查询算法。

geqo

参数说明：控制基因查询优化的使用。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

通常情况下在执行过程中不要关闭，geqo_threshold变量提供了更精细的控制GEQO的方法。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

geqo_threshold

参数说明：如果执行语句的数量超过设计的FROM的项数，则会使用基因查询优化来执行查询。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 对于简单的查询，通常用详尽搜索方法，当涉及多个表的查询的时候，用GEQO可以更好地管理查询。
- 一个FULL OUTER JOIN构造仅作为一个FROM项。

取值范围：整型，2～INT_MAX。

默认值：12

geqo_effort

参数说明：控制GEQO在规划时间和规划质量之间的平衡。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

geqo_effort实际上并没有直接做任何事情，只是用于计算其他影响GEQO的变量的默认值。如果愿意，可以手工设置其他参数。

取值范围：整型，1～10。

须知

比默认值大的数值增加了查询规划的时间，但是也增加了选中有效查询的几率。

默认值：5

geqo_pool_size

参数说明：控制GEQO使用池的大小，也就是基因全体中的个体数量。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0～INT_MAX。

须知

至少是2，且有用的值一般在100到1000之间。设置为0，表示使用系统自适应方式，GaussDB会基于geqo_effort和表的个数选取合适的值。

默认值：0

geqo_generations

参数说明：控制GEQO使用的算法的迭代次数。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0～INT_MAX。

须知

必须至少是1，且有用的值介于100和1000之间。如果设置为0，则基于geqo_pool_size选取合适的值。

默认值：0

geqo_selection_bias

参数说明：控制GEQO的选择性偏好，即就是一个种群中的选择性压力。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：浮点型，1.5～2.0。

默认值：2

geqo_seed

参数说明：控制GEQO使用的随机数生产器的初始化值，用来从顺序连接在一起的查询空间中查找随机路径。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：浮点型，0.0～1.0。

须知

不同的值会改变搜索的连接路径，从而影响了所找路径的优劣。

默认值：0

14.3.8.4 其他优化器选项

cost_model_version

参数说明：此参数用来指定优化器代价模型的版本。可以视作一个保护参数，用来禁用最新的优化器代价模型，保持和旧版本计划一致。改变此参数，可能会导致很多SQL计划的改变。因此修改前请谨慎评估。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：0、1、2、3

- 0：表示使用最新的cost估算模型。当前版本等价于3。
- 1：表示使用原始的cost估算模型。
- 2：表示在1的基础上，使用增强的coalesce表达式估算、hash join代价估算、semi/anti join代价估算。
- 3：表示在2的基础上，使用边界矫正估计器估算NDV, indexscan的hint可以作用于indexonlyscan。

默认值：0

explain_dna_file

参数说明：指定**explain_perf_mode**为run，导出的csv信息的目标文件。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

这个参数的取值必须是绝对路径加上.csv格式的文件名。

取值范围：字符串

默认值：空

explain_perf_mode

参数说明：此参数用来指定explain的显示格式。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：normal、pretty、summary、run

- normal：代表使用默认的打印格式。
- pretty：代表使用GaussDB改进后的新显示格式。新的格式层次清晰，计划包含了plan node id，性能分析简单直接。
- summary：是在pretty的基础上增加了对打印信息的分析。
- run：在summary的基础上，将统计的信息输出到csv格式的文件中，以便于进一步分析。

须知

explain的显示格式取不同值时，展示顺序可能存在较大的差异。normal格式和pretty格式显示样例说明如下：

normal格式：

```
                                                  QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------
 Sort  (cost=21.23..21.23 rows=1 width=306)
   Sort Key: supplier.s_suppkey
   CTE revenue
     ->  HashAggregate  (cost=12.88..12.88 rows=1 width=76)
           Group By Key: lineitem.l_suppkey
           ->  Partition Iterator  (cost=0.00..12.87 rows=1 width=44)
                 Iterations: 7
                 ->  Partitioned Seq Scan on lineitem  (cost=0.00..12.87 rows=1 width=44)
                       Filter: ((l_shipdate >= '1996-01-01 00:00:00'::timestamp(0) without time zone) AND
(l_shipdate < '1996-04-01 00:00:00'::timestamp without time zone))
                       Selected Partitions:  1..7
   InitPlan 2 (returns $3)
     ->  Aggregate  (cost=0.02..0.03 rows=1 width=64)
           ->  CTE Scan on revenue  (cost=0.00..0.02 rows=1 width=32)
   ->  Nested Loop  (cost=0.00..8.30 rows=1 width=306)
         ->  CTE Scan on revenue  (cost=0.00..0.02 rows=1 width=40)
               Filter: (total_revenue = $3)
         ->  Partition Iterator  (cost=0.00..8.27 rows=1 width=274)
               Iterations: 7
               ->  Partitioned Index Scan using supplier_s_suppkey_idx on supplier  (cost=0.00..8.27 rows=1
width=274)
                     Index Cond: (s_suppkey = revenue.supplier_no)
                     Selected Partitions:  1..7
(21 rows)
```

pretty格式：

```
 id |                          operation                          | E-rows | E-width |   E-costs
----+-----------------------------------------------------------------------------+--------+---------+----------------
  1 | ->  Sort                                                   |      1 |     306 | 21.230..21.235
  2 |    ->  Nested Loop (3,9)                                   |      1 |     306 | 0.000..8.303
  3 |       ->  CTE Scan on revenue                              |      1 |      40 | 0.000..0.022
  4 |    ->  HashAggregate  [3, CTE revenue]                     |      1 |      76 | 12.875..12.885
  5 |       ->  Partition Iterator                               |      1 |      44 | 0.000..12.865
  6 |          ->  Partitioned Seq Scan on lineitem              |      1 |      44 | 0.000..12.865
  7 |    ->  Aggregate  [4, InitPlan 2 (returns $3)]             |      1 |      64 | 0.022..0.033
  8 |       ->  CTE Scan on revenue                              |      1 |      32 | 0.000..0.020
  9 |       ->  Partition Iterator                               |      1 |     274 | 0.000..8.270
 10 |          ->  Partitioned Index Scan using supplier_s_suppkey_idx on supplier |      1 |     274 | 0.000..8.270
(10 rows)

                              Predicate Information (identified by plan id)
------------------------------------------------------------------------------------------------------------------------------
--------------------------------
   5 --Partition Iterator
         Iterations: 7
   6 --Partitioned Seq Scan on lineitem
         Filter: ((l_shipdate >= '1996-01-01 00:00:00'::timestamp(0) without time zone) AND (l_shipdate <
'1996-04-01 00:00:00'::timestamp without time zone))
         Selected Partitions:  1..7
   3 --CTE Scan on revenue
         Filter: (total_revenue = $3)
   9 --Partition Iterator
         Iterations: 7
  10 --Partitioned Index Scan using supplier_s_suppkey_idx on supplier
         Index Cond: (s_suppkey = revenue.supplier_no)
         Selected Partitions:  1..7
(12 rows)
```

说明：上述两种格式的计划块是同一计划的不同展示形式，在pretty格式下，加粗部分是CET和InitPlan计划块，它们是可能穿插在Join连接块中间的，阅读Join连接块时候遇到CTE、InitPlan块时候跳过即可找到对应的Join块的内表。

默认值：pretty

analysis_options

参数说明：通过开启对应选项中所对应的功能选项使用相应的定位功能，包括数据校验、性能统计等，参见取值范围中的选项说明。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

- HASH_CONFLICT表示在数据库节点进程的pg_log目录中的log日志中显示hash表的统计信息，包括hash表大小、hash链长、hash冲突情况。
- STREAM_DATA_CHECK表示对网络传输前后的数据进行CRC校验。

默认值：ALL,on(),off(HASH_CONFLICT,STREAM_DATA_CHECK)，不开启任何定位功能。

cost_param

参数说明：该参数用于控制在特定的客户场景中，使用不同的估算方法使得估算值与真实值更接近。此参数可以同时控制多种方法，与某一方法对应的位做与操作，不为0表示该方法被选择。

当cost_param & 1 不为0，表示对于求不等值连接选择率时选择一种改良机制，此方法在自连接（两个相同的表之间连接）的估算中更加准确。目前，已弃用cost_param & 1 不为0时的路径，默认选择更优的估算公式；

当cost_param & 2 不为0，表示求多个过滤条件（Filter）的选择率时，选择最小的作为总的选择率，而非两者乘积，此方法在过滤条件的列之间关联性较强时估算更加准确；

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0～INT_MAX

默认值：0

var_eq_const_selectivity

参数说明：整型const选择率是否使用新型选择率模型进行估算。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用新型选择率模型计算整型const的选择率：
  - 若整型不落入MCV且不为NULL值，但落入直方图，则利用直方图左右边界情况进行估算；不落入直方图，则使用表的行数进行估算。
  - 若整型为NULL值或者MCV值，使用原逻辑计算选择率。
- off表示使用原有的选择率计算模型。

默认值：off

enable_partitionwise

参数说明：分区表连接操作是否选择智能算法。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示选择智能算法。
- off表示不选择智能算法。

默认值：off

partition_page_estimation

参数说明：分区表页面是否通过剪枝结果进行页面估算优化。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用剪枝结果进行页面估算优化。
- off表示不使用剪枝结果进行页面估算优化。

默认值：off

partition_iterator_elimination

参数说明：分区表在分区剪枝结果为一个分区时，是否消除分区迭代算子来提升执行效率。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示消除分区迭代算子。
- off表示不消除分区迭代算子。

默认值：off

enable_functional_dependency

参数说明：ANALYZE生成的多列统计信息是否包含函数依赖统计信息，是否应用函数依赖统计信息计算选择率。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on包含两个功能：1. 执行ANALYZE生成的多列统计信息包含函数依赖统计信息。2. 计算选择率会使用函数依赖统计信息。
- off包含两个功能：1. 执行ANALYZE生成的多列统计信息不包含函数依赖统计信息。2. 计算选择率不会使用函数依赖统计信息。

默认值：off

rewrite_rule

参数说明：标识开启的可选查询重写规则。有部分查询重写规则是可选的，开启它们并不能总是对查询效率有提升效果。在特定的客户场景中，通过此GUC参数对查询重写规则进行设置，使得查询效率最优。

此参数可以控制查询重写规则的组合，比如有多个重写规则：rule1、rule2、rule3、rule4。可以设置：

```
set rewrite_rule=rule1;          --启用查询重写规则rule1
set rewrite_rule=rule2,rule3;    --启用查询重写规则rule2和rule3
set rewrite_rule=none;           --关闭所有可选查询重写规则
```

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

- none：不使用任何可选查询重写规则。
- lazyagg：使用Lazy Agg查询重写规则（消除子查询中的聚集运算）。
- magicset：使用Magic Set查询重写规则（将带有聚集算子的子查询提前和主查询进行关联，减少子链接的重复扫描）。
- uniquecheck：使用Unique Check查询重写规则（提升目标列中无agg的子查询语句，在执行时检查返回行数是否为1行）。
- intargetlist：使用In Target List查询重写规则（提升目标列中的子查询）。
- predpushnormal：使用Predicate Push查询重写规则（下推谓词条件到子查询中）。
- predpushforce：使用Predicate Push查询重写规则（下推谓词条件到子查询中，尽可能的利用索引加速）。
- predpush：在predpushnormal和predpushforce中根据代价选择最优计划。
- disable_pullup_expr_sublink：禁止优化器将expr_sublink类型的子连接提升，关于sublink的分类和提升原理详见《开发指南》的“SQL调优指南 > 典型SQL调优点 > 子查询调优”章节。
- enable_sublink_pullup_enhanced：使用增强后的sublink查询重写规则，包括where、having子句的非相关子链接提升和winmagic重写优化。
- disable_pullup_not_in_sublink：禁止优化器对not in相关的子链接进行提升，关于sublink的分类和提升原理详见《开发指南》中“SQL调优指南 > 典型SQL调优点 > 子查询调优”章节。

默认值：magicset

说明

可以设置partialpush、disablerep参数，但实际不生效。

enable_pbe_optimization

参数说明：设置优化器是否对以PBE（Parse Bind Execute）形式执行的语句进行查询计划的优化。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型。

- on表示优化器将优化PBE语句的查询计划。
- off表示不使用优化。

默认值：on

enable_global_plancache

参数说明：设置是否对PBE查询和存储过程中语句的执行计划进行缓存共享，开启该功能可以节省高并发下数据库节点的内存使用。

在打开enable_global_plancache的情况下，为保证GPC生效，默认local_syscache_threshold不小于16MB。即如当前local_syscache_threshold小于16MB，则设置为16MB，如大于16MB，则不改变。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型。

- on表示对PBE查询和存储过程中语句的执行计划进行缓存共享。
- off表示不共享。

默认值：off

gpc_clean_timeout

参数说明：开启enable_global_plancache的情况下，如果共享计划列表里的计划超过gpc_clean_timeout的时间没有被使用，则会被清理掉。本参数用于控制没有使用的共享计划的保留时间。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，300~86400

- 单位为秒

默认值：1800，即30min

enable_global_stats

该参数当前版本已废弃，请勿设置。

enable_opfusion

参数说明：控制是否对简单增删改查进行优化。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

简单查询限制如下：

- 只支持indexscan和indexonlyscan，且全部WHERE语句的过滤条件都在索引上。
- 只支持单表增删改查，不支持join、using。
- 只支持行存表，不支持分区表，表不支持有触发器。
- 不支持active sql、QPS等信息统计特性。
- 不支持正在扩容和缩容的表。

- 不支持查询或者修改系统列。
- 只支持简单SELECT语句，例如：
  ```
  SELECT c3 FROM t1 WHERE c1 = ? and c2 =10;
  ```
  仅可以查询目标表的列，c1和c2列为索引列，后边可以是常量或者参数，可以使用 for update。
- 只支持简单INSERT语句，例如：
  ```
  INSERT INTO t1 VALUES (?,10,?);
  ```
  仅支持一个VALUES，VALUES里面的类型可以是常量和参数，不支持returning。
- 只支持简单DELETE语句，例如：
  ```
  DELETE FROM t1 WHERE c1 = ? and c2 = 10;
  ```
  c1和c2列为索引列，后边可以是常量或者参数。
- 只支持简单UPDATE语句，例如：
  ```
  UPDATE t1 SET c3 = c3+? WHERE c1 = ? and c2 = 10;
  ```
  c3列修改的值可以是常量和参数，也可以是一个简单的表达式，c1和c2列为索引列，后边可以是常量或者参数。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

enable_partition_opfusion

参数说明：在enable_opfusion参数打开的状态下，如果开启该参数，可以对分区表的简单查询进行查询优化，提升SQL执行性能。在开启enable_global_plancache参数时，此参数设置on时将不生效。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：off

sql_beta_feature

参数说明：标识开启的可选SQL引擎Beta特性，其中包括对行数估算、查询等价估算等优化。

开启它们可以对特定的场景进行优化，但也可能会导致部分没有被测试覆盖的场景发生性能劣化。在特定的客户场景中，通过此GUC参数对查询重写规则进行设置，使得查询效率最优。

此参数可以控制SQL引擎Beta特性的组合，比如有多个Beta特性：feature1、feature2、feature3、feature4。可以设置：

```
--启用SQL引擎Beta特性feature1。
set sql_beta_feature=feature1;
--启用SQL引擎Beta特性feature2和feature3。
set sql_beta_feature=feature2,feature3;
--关闭所有可选SQL引擎Beta特性。
set sql_beta_feature=none;
```


该参数属于USERSET类型参数，请参考表14-1中对应设置方法进行设置。

取值范围：字符串

- none：不使用任何Beta优化器特性。
- sel_semi_poisson：使用泊松分布对等值的半连接和反连接选择率进行校准。
- sel_expr_instr：使用字符串匹配的行数估算方法对instr(col, 'const') > 0, = 0, = 1进行更准确的估算。
- param_path_gen：生成更多可能的参数化路径。
- rand_cost_opt：对小数据量表的随机读取代价进行优化。
- param_path_opt：利用表的膨胀系数优化索引analyze信息。
- page_est_opt：优化对表索引analyze信息的relpages估算。
- no_unique_index_first：关闭主键索引扫描路径优先的优化。
- join_sel_with_cast_func：估算join行数的时候支持类型转换函数。
- canonical_pathkey：正则化pathkey生成置前。（pathkey：标记数据有序性键值的集合）

  警告

  该参数打开之后，可能会导致带 order by 等语句，在有外连接的情况下，输出数据语义和标准不一样。请联系华为工程师再确定是否打开该参数。

- index_cost_with_leaf_pages_only：估算索引代价时考虑索引叶子节点。
- partition_opfusion：开启分区表优化。
- a_style_coerce：开启Decode类型转换规则兼容O，详见《开发指南》的“SQL参考 > 类型转换 > UNION，CASE和相关构造”章节中的“对于case，在ORA兼容模式下的处理”部分内容。
- partition_fdw_on：支持基于分区表创建postgres foreign table下的相关SQL。
- predpush_same_level：开启predpush hint控制同层参数化路径的功能。
- enable_plsql_smp：开启存储过程中的查询支持并行执行的功能。目前在同一时刻仅支持一条query使用并行执行，且cursor相关操作、自治事务、exception中的查询不会生成并行执行计划。
- disable_bitmap_cost_with_lossy_pages：关闭bitmap路径代价中对lossy pages代价的计算。
- enable_upsert_execute_gplan：pbe场景下on duplicate key update语句中update子句带有参数时设置enable_upsert_execute_gplan允许通过gplan执行。
- disable_merge_append_partition：对于分区表，禁止生成Merge Append路径。

默认值：
"sel_semi_poisson,sel_expr_instr,rand_cost_opt,param_path_opt,page_est_opt"

ngram_gram_size

参数说明：ngram解析器分词的长度。

该参数属于USERSET类型参数，请参考表14-1中对应设置方法进行设置。

取值范围：整型，1～4

默认值：2

ngram_grapsymbol_ignore

参数说明：ngram解析器是否忽略图形化字符。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示忽略图形化字符。
- off表示不忽略图形化字符。

默认值：off

ngram_punctuation_ignore

参数说明：ngram解析器是否忽略标点符号。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示忽略标点符号。
- off表示不忽略标点符号。

默认值：on

default_statistics_target

参数说明：为没有用ALTER TABLE SET STATISTICS设置字段目标的表设置缺省统计目标。此参数设置为正数是代表统计信息的样本数量，为负数时，代表使用百分比的形式设置统计目标，负数转换为对应的百分比，即-5代表5%。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，-100～10000。

须知

- 比默认值大的正数数值增加了ANALYZE所需的时间，但是可能会改善优化器的估计质量。
- 调整此参数可能存在性能劣化的风险，如果某个查询劣化，可以考虑
  1. 恢复默认的统计信息。
  2. 使用plan hint来调整到之前的查询计划。详细参见《开发指南》的“SQL调优指南 > 使用Plan Hint进行调优”章节。

默认值：100

auto_statistic_ext_columns

参数说明：表示会根据数据表上的组合索引的前K列，收集多列统计信息。此GUC参数表示K。例如：某组合索引为(a,b,c,d,e)，此GUC参数设为3，则会在组合列（a,b)、

（a,b,c）上产生多列统计信息。多列统计信息可以在组合条件查询时，让优化器估计基数估计得更准。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 系统表不会生效。
- 组合列中的所有列的类型，都支持比较函数'='和'<'，统计信息才会生效。
- 索引中的系统伪列，如tableoid、ctid不会被收集。
- 默认会收集distinct值、不带NULL的高频值、带NULL的高频值。如果开启了智能基数估计参数**enable_ai_stats**，则不会收集高频值，而是收集智能基数估计的模型。
- 如果创建某多列统计信息的索引被删除，并且没有其它索引包含该多列组合，在下一次 analyze 的时候，该多列统计信息会被删除。
- 如果该参数由大变小，新的索引会依据该参数产生多列统计信息，而已经产生的超过该参数长度的多列统计信息，不会被删除。
- 如果用户希望禁用某一特定组合的多列统计信息，又希望使用其它多列统计信息，可以不修改该参数，而使用DDL命令'ALTER TABLE tablename disable statistics ((column list))'的方式，禁用特定的多列组合。

取值范围：整型，1~4。1就是表示不会自动收集多列统计信息。

默认值：1

constraint_exclusion

参数说明：控制查询优化器使用表约束查询的优化。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：枚举类型

- on表示检查所有表的约束。
- off表示不检查约束。
- partition表示只检查继承的子表和UNION ALL子查询。

须知

当constraint_exclusion为on，优化器用查询条件和表的CHECK约束比较，并且在查询条件和约束冲突的时候忽略对表的扫描。

默认值：partition

说明

目前， constraint_exclusion缺省被打开，通常用来实现表分区。为所有的表打开它时，对于简单的查询强加了额外的规划，并且对简单查询没有什么好处。如果不用分区表，可以关掉它。

cursor_tuple_fraction

参数说明：优化器估计游标获取行数在总行数中的占比。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：浮点型，0.0～1.0。

须知

比默认值小的值与使用 “fast start” 为游标规划的值相偏离，从而使得前几行恢复的很快而抓取全部的行需要很长的时间。比默认值大的值加大了总的估计的时间。在最大的值1.0处，像正常的查询一样规划游标，只考虑总的估计时间和传送第一行的时间。

默认值：0.1

from_collapse_limit

参数说明：根据生成的FROM列表的项数来判断优化器是否将把子查询合并到上层查询，如果FROM列表项个数小于等于该参数值，优化器会将子查询合并到上层查询。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1～INT_MAX。

须知

比默认值小的数值将降低规划时间，但是可能生成差的执行计划。

默认值：8

join_collapse_limit

参数说明：根据得出的列表项数来判断优化器是否执行把除FULL JOINS之外的JOIN构造重写到FROM列表中。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1～INT_MAX。

须知

- 设置为1会避免任何JOIN重排。这样就使得查询中指定的连接顺序就是实际的连接顺序。查询优化器并不是总能选取最优的连接顺序，高级用户可以选择暂时把这个变量设置为1，然后指定它们需要的连接顺序。
- 比默认值小的数值减少规划时间但也降低了执行计划的质量。

默认值：8

plan_mode_seed

参数说明：该参数为调测参数，目前仅支持OPTIMIZE_PLAN和RANDOM_PLAN两种。其中：OPTIMIZE_PLAN表示通过动态规划算法进行代价估算的最优plan，参数值设置为0；RANDOM_PLAN表示随机生成的plan；如果设置为-1，表示用户不指定随机数的种子标识符seed值，由优化器随机生成[1, 2147483647]范围整型值的随机数，并根据随机数生成随机的执行计划；如果用户指定guc参数值为[1, 2147483647]范围的整型值，表示指定的生成随机数的种子标识符seed，优化器需要根据seed值生成随机的执行计划。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，-1~ 2147483647

默认值：0

须知

- 当该参数设置为随机执行计划模式时，优化器会生成不同的随机执行计划，该执行计划可能不是最优计划。因此在随机计划模式下，会对查询性能产生影响，所以建议在升级、扩容、缩容等正常业务操作或运维过程中将该参数保持为默认值0。
- 当该参数不为0时，查询指定的plan hint不会生效。

hashagg_table_size

参数说明：用于设置执行HASH JOIN操作时HASH表的大小。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0～INT_MAX/2。

默认值：0

enable_bloom_filter

参数说明: 标识是否允许使用BloomFilter优化。该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围： 布尔型

- on表示允许使用BloomFilter优化。
- off表示不允许使用BloomFilter优化。

默认值： on

enable_extrapolation_stats

参数说明: 标识对于日期类型是否允许基于历史统计信息使用推理估算的逻辑。使用该逻辑对于未及时收集统计信息的表可以增大估算准确的可能性，但也存在错误推理导致估算过大的可能性，需要对于日期类型数据定期插入的场景开启此开关。该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围： 布尔型

- on表示允许基于历史统计信息使用推理估算的逻辑。
- off表示不允许基于历史统计信息使用推理估算的逻辑。

默认值：off

autoanalyze

参数说明：标识是否允许在生成计划的时候，对于没有统计信息的表进行统计信息自动收集。对于外表和临时表，不支持autoanalyze，如果需要收集统计信息，用户需手动执行analyze操作。如果在auto analyze某个表的过程中数据库发生异常，当数据库正常运行之后再执行语句有可能仍提示需要收集此表的统计信息。此时需要用户对该表手动执行一次analyze操作，以同步统计信息数据。该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许自动进行统计信息收集。
- off表示不允许自动进行统计信息收集。

默认值：off

说明

集中式下该参数不生效。

enable_analyze_check

参数说明: 标识是否允许在生成计划的时候，对于在pg_class中显示reltuples和relpages均为0的表，检查该表是否曾进行过统计信息收集。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许检查。
- off表示不允许检查。

默认值：off

enable_sonic_hashagg

参数说明：标识是否依据规则约束使用基于面向列的hash表设计的Hash Agg算子。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示在满足约束条件时使用基于面向列的hash表设计的Hash Agg算子。
- off表示不使用面向列的hash表设计的Hash Agg算子。

说明

- 在开启enable_sonic_hashagg，且查询达到约束条件使用基于面向列的hash表设计的Hash Agg算子时，查询对应的Hash Agg算子内存使用通常可获得精简。但对于代码生成技术可获得显著性能提升的场景，对应的算子查询性能可能会出现劣化。
- 开启enable_sonic_hashagg，且查询达到约束条件使用基于面向列的hash表设计的Hash Agg算子时，在Explain Analyze/Performance的执行计划和执行信息中，算子显示为“Sonic Hash Aggregation”，而未达到该约束条件时，算子名称将显示为“Hash Aggregation”，Explain详解请参见《开发指南》的“SQL调优指南 > SQL执行计划介绍 > 详解”章节。

默认值：on

enable_sonic_hashjoin

参数说明：标识是否依据规则约束使用基于面向列的hash表设计的Hash Join算子。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示在满足约束条件时使用基于面向列的hash表设计的Hash Join算子。
- off表示不使用面向列的hash表设计的Hash Join算子。

说明

- 当前开关仅适用于Inner Join的场景。
- 在开启enable_sonic_hashjoin，查询对应的Hash Inner算子内存使用通常可获得精简。但对于代码生成技术可获得显著性能提升的场景，对应的算子查询性能可能会出现劣化。
- 开启enable_sonic_hashjoin，且查询达到约束条件使用基于面向列的hash表设计的Hash Join算子时，在Explain Analyze/Performance的执行计划和执行信息中，算子显示为“Sonic Hash Join”，而未达到该约束条件时，算子名称将显示为“Hash Join”，Explain详解请参见《开发指南》的“SQL调优指南 > SQL执行计划介绍 > 详解”章节。

默认值：on

enable_sonic_optspill

参数说明：标识是否对面向列的hash表设计的Hash Join算子进行下盘文件数优化。该参数打开时，在Hash Join算子下盘文件较多的时候，下盘文件数不会显著增加。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示优化面向列的hash表设计的Hash Join算子的下盘文件数。
- off表示不优化面向列的hash表设计的Hash Join算子的下盘文件数。

默认值：on

plan_cache_mode

参数说明：标识在prepare语句中，选择生成执行计划的策略。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：枚举类型

- auto表示按照默认的方式选择custom plan或者generic plan。
- force_generic_plan表示强制走generic plan（软解析）。generic plan是指对于prepare语句生成计划，该计划策略会在执行execute语句的时候把参数bind到plan中，然后执行计划。这种方案的优点是每次执行可以省去重复的优化器开销；缺点是当bind参数字段上数据存在倾斜时该计划可能不是最优的，部分bind参数场景下执行性能较差。
- force_custom_plan表示强制走custom plan（硬解析）。custom plan是指对于prepare语句，在执行execute的时候，把execute语句中的参数嵌套到语句之后生成的计划。custom plan会根据execute语句中具体的参数生成计划，这种方案的优点是每次都按照具体的参数生成优选计划，执行性能比较好；缺点是每次执行前都需要重新生成计划，存在大量的重复的优化器开销。

说明

此参数只对prepare语句生效，一般用在prepare语句中参数化字段存在比较严重的数据倾斜的场景下。

默认值：auto

enable_hypo_index

参数说明：控制优化器执行EXPLAIN命令时是否考虑虚拟索引。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：off

enable_auto_explain

参数说明：控制是否开启自动打印执行计划。该参数是用来定位慢存储过程或慢查询，只对当前连接的数据库主节点有效。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型，true表示开启，false表示关闭。

默认值：false

auto_explain_level

参数说明：控制自动打印执行计划的日志等级。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：枚举型，LOG或NOTICE，LOG表示在日志中打印执行计划，NOTICE表示以提示知的形式打印出计划。

默认值：LOG

auto_explain_log_min_duration

参数说明：控制自动打印执行计划的耗时阈值，整体耗时大于auto_explain_log_min_duration的执行计划才会被打印。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~2147483647，单位为毫秒。

- 设置为0，所有执行过的执行计划都会输出。
- 设置为3000，单次语句执行耗时超过3000毫秒后所有执行的执行计划会输出。

默认值：0

query_dop

参数说明: 用户自定义的查询并行度。该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围: 整型，1~64。打开固定SMP功能，系统会使用固定并行度。

说明

在开启并行查询后，请保证系统CPU、内存、网络等资源充足，以达到最佳效果。

默认值: 1

enable_startwith_debug

参数说明：该参数为start with/connect by用于debug的参数，打开参数可以显示start with特性所有涉及的尾列相关信息。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型，true表示开启，false表示关闭。

默认值：false。

enable_inner_unique_opt

参数说明：对嵌套循环连接、哈希连接、排序归并连接进行Inner Unique优化，即在连接条件中内表对应的属性满足唯一性约束的情况下，减少匹配次数。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on：表示使用。
- off：表示不使用。

默认值：on。

enable_indexscan_optimization

参数说明：控制是否对astore存储引擎下的btree索引扫描（IndexScan和IndexOnlyScan）进行优化。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on：表示使用。
- off：表示不使用。

默认值：off。

immediate_analyze_threshold

参数说明：插入数据后自动做analyze的阈值。当新增数据量达到原有数据量的immediate_analyze_threshold倍，且总行数超过一百时，会自动触发一次analyze。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~1000。当取值为零时，该功能关闭。

默认值：0。

说明

1. 该功能只支持永久表和非日志表，不支持临时表。
2. 同一表不会在10s内两次被自动触发analyze。

enable_dynamic_samplesize

参数说明：是否动态调整采样行数。对于超过一百万行的大表，收集统计信息时动态调整采样行数，提高统计信息准确性。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on：表示该功能打开。
- off：表示该功能关闭。

默认值：on

说明

动态调整采样行数的功能仅支持绝对值采样。

14.3.9 错误报告和日志

14.3.9.1 记录日志的位置

log_destination

参数说明：GaussDB支持多种方法记录服务器日志，log_destination的取值为一个逗号分隔开的列表（如log_destination="stderr,csvlog"）。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

有效值为stderr、csvlog、syslog。

- 取值为stderr，表示日志打印到屏幕。
- 取值为csvlog，表示日志的输出格式为“逗号分隔值”即CSV（Comma Separated Value）格式。使用csvlog记录日志的前提是将**logging_collector**设置为on，请参见**使用CSV格式写日志**。
- 取值为syslog，表示通过操作系统的syslog记录日志。GaussDB使用syslog的LOCAL0 ~ LOCAL7记录日志，请参见**syslog_facility**。使用syslog记录日志需在操作系统后台服务配置文件中添加代码：
  local0.* */var/log/*omm

默认值：stderr

logging_collector

参数说明：控制开启后端日志收集进程logger进行日志收集。该进程捕获发送到stderr或csvlog的日志消息并写入日志文件。

这种记录日志的方法比将日志记录到syslog更加有效，因为某些类型的消息在syslog的输出中无法显示。例如动态链接库加载失败消息和脚本（例如archive_command）产生的错误消息。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

将服务器日志发送到stderr时可以不使用logging_collector参数，此时日志消息会被发送到服务器的stderr指向的空间。这种方法的缺点是日志回滚困难，只适用于较小的日志容量。

取值范围：布尔型

- on表示开启日志收集功能。
- off表示关闭日志收集功能。

默认值：on

log_directory

参数说明：logging_collector设置为on时，log_directory决定存放服务器日志文件的目录。它可以是绝对路径，或者是相对路径（相对于数据目录的路径）。log_directory支持动态修改，可以通过gs_guc reload实现，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 当配置文件中log_directory的值为非法路径时，会导致数据库无法重新启动。
- 通过gs_guc reload动态修改log_directory时，当指定路径为合法路径时，日志输出到新的路径下。当指定路径为非法路径时，日志输出到上一次合法的日志输出路径下而不影响数据库正常运行。此时即使指定的log_directory的值非法，也会写入到配置文件中。
- 在沙箱环境，路径中不可以包含/var/chroot，例如log的绝对路径是/var/chroot/var/lib/log/Ruby/pg_log/cn_log，则只需要设置为/var/lib/log/Ruby/pg_log/cn_log。

说明

- 合法路径：用户对此路径有读写权限。
- 非法路径：用户对此路径无读写权限。

取值范围：字符串

默认值：安装时指定。

log_filename

参数说明：logging_collector设置为on时，log_filename决定服务器运行日志文件的名称。通常日志文件名是按照strftime模式生成，因此可以用系统时间定义日志文件名，用%转义字符实现，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 建议使用%转义字符定义日志文件名称，否则难以对日志文件进行有效的管理。
- 当log_destination设为csvlog时，系统会生成附加了时间戳的日志文件名，文件格式为csv格式，例如“server_log.1093827753.csv”。

取值范围：字符串

默认值：postgresql-%Y-%m-%d_%H%M%S.log

log_file_mode

参数说明：logging_collector设置为on时，log_file_mode设置服务器日志文件的权限。通常log_file_mode的取值是能够被chmod和umask系统调用接受的数字。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 使用此选项前请设置log_directory，将日志存储到数据目录之外的地方。
- 因日志文件可能含有敏感数据，故不能将其设为对外可读。

取值范围：整型，0000～0777（8进制计数，转化为十进制 0 ~ 511）。

说明

- 0600表示只允许服务器管理员读写日志文件。
- 0640表示允许管理员所在用户组成员只能读日志文件。

默认值：0600

log_truncate_on_rotation

参数说明：logging_collector设置为on时，log_truncate_on_rotation设置日志消息的写入方式。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

示例如下：

假设日志需要保留7天，每天生成一个日志文件，日志文件名设置为server_log.Mon、server_log.Tue等。第二周的周二生成的日志消息会覆盖写入到server_log.Tue。设置方法：将log_filename设置为server_log.%a ，log_truncate_on_rotation设置为on，log_rotation_age设置为1440，即日志有效时间为1天。

取值范围： 布尔型

- on表示GaussDB以覆盖写入的方式写服务器日志消息。
- off表示GaussDB将日志消息附加到同名的现有日志文件上。

默认值：off

log_rotation_age

参数说明：logging_collector设置为on时，log_rotation_age决定创建一个新日志文件的时间间隔。当现在的时间减去上次创建一个服务器日志的时间超过了log_rotation_age的值时，将生成一个新的日志文件。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0 ~ 35791394，单位为min。其中0表示关闭基于时间的新日志文件的创建。

默认值：1440(min)

log_rotation_size

参数说明：logging_collector设置为on时，log_rotation_size决定服务器日志文件的最大容量。当日志消息的总量超过日志文件容量时，服务器将生成一个新的日志文件。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0 ~ INT_MAX / 1024，单位为kB。

0表示关闭基于容量的新日志文件的创建。

建议该值大小设置级别至少为MB级,利于日志文件的及时划分。

默认值：20MB

syslog_facility

参数说明：log_destination设置为syslog时，syslog_facility配置使用syslog记录日志的“设备”。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：枚举类型，有效值有local0、local1、local2、local3、local4、local5、local6、local7。

默认值：local0

syslog_ident

参数说明：**log_destination**设置为syslog时，syslog_ident设置在syslog日志中GaussDB日志消息的标识。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

默认值：postgres

event_source

参数说明：该参数仅在windows环境下生效, GaussDB暂不支持。log_destination设置为eventlog时，event_source设置在日志中GaussDB日志消息的标识。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

默认值：PostgreSQL

14.3.9.2 记录日志的时间

client_min_messages

参数说明：控制发送到客户端的消息级别。每个级别都包含排在它后面的所有级别中的信息。级别越低，发送给客户端的消息就越少。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

当client_min_messages和**log_min_messages**取相同值时，其值所代表的级别不同。

取值范围：枚举类型，有效值有debug、debug5、debug4、debug3、debug2、debug1、info、log、notice、warning、error、fatal、panic。参数的详细信息请参见**表14-6**。在实际设置过程中，如果设置的级别大于error，为fatal或panic，系统会默认将级别转为error。

默认值：notice

log_min_messages

参数说明：控制写到服务器日志文件中的消息级别。每个级别都包含排在它后面的所有级别中的信息。级别越低，服务器运行日志中记录的消息就越少。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

当**client_min_messages**和log_min_messages取相同值log时所代表的消息级别不同。部分日志信息的打印需要同时配置该参数与logging_module，即设置该参数打开后可能还需要设置logging_module打开对应模块的日志打印开关。

取值范围：枚举类型，有效值有debug、debug5、debug4、debug3、debug2、debug1、info、log、notice、warning、error、fatal、panic。参数的详细信息请参见**表14-6**。

默认值：warning

log_min_error_statement

参数说明：控制在服务器日志中记录错误的SQL语句。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：枚举类型，有效值有debug、debug5、debug4、debug3、debug2、debug1、info、log、notice、warning、error、fatal、panic。参数的详细信息请参见**表14-6**。

说明

- 设置为error，表示导致错误、日志消息、致命错误、panic的语句都将被记录。
- 设置为panic，表示关闭此特性。

默认值：error

log_min_duration_statement

参数说明：当某条语句的持续时间大于或者等于特定的毫秒数时，log_min_duration_statement参数用于控制记录每条完成语句的持续时间。

设置log_min_duration_statement可以很方便地跟踪需要优化的查询语句。对于使用扩展查询协议的客户端，语法分析、绑定、执行每一步所花时间被独立记录。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

当此选项与**log_statement**同时使用时，已经被log_statement记录的语句文本不会被重复记录。在没有使用syslog情况下，推荐使用**log_line_prefix**记录PID或会话ID，方便将当前语句消息连接到最后的持续时间消息。

取值范围：整型，-1~ 2147483647，单位为毫秒。

- 设置为250，所有运行时间不短于250ms的SQL语句都会被记录。
- 设置为0，输出所有语句的持续时间。
- 设置为-1，关闭此功能。

默认值：3s（即3000ms）

backtrace_min_messages

参数说明：控制当产生该设置参数级别相等或更高级别的信息时，会打印函数的堆栈信息到服务器日志文件中。

该参数属于SUSET类型参数，请参考表14-1中对应设置方法进行设置。

须知

该参数作为客户现场问题定位手段使用，且由于频繁的打印函数栈会对系统的开销及稳定性有一定的影响，因此如果需要进行问题定位时，建议避免将backtrace_min_messages的值设置为fatal及panic以外的级别。

取值范围：枚举类型

有效值有debug、debug5、debug4、debug3、debug2、debug1、info、log、notice、warning、error、fatal、panic。参数的详细信息请参见表14-6。

默认值：panic

表14-6解释GaussDB中使用的消息安全级别。当日志输出到syslog或者eventlog(仅windows环境下，GaussDB版本不涉及该参数)时，GaussDB进行如表中的转换。

表 14-6 信息严重程度分类

信息严重程度类型	详细说明	系统日志	事件日志
debug[1-5]	报告详细调试信息。	DEBUG	INFORMATION
log	报告对数据库管理员有用的信息，比如检查点操作统计信息。	INFO	INFORMATION
info	报告用户可能需求的信息，比如在VACUUM VERBOSE过程中的信息。	INFO	INFORMATION
notice	报告可能对用户有帮助的信息，比如， 长标识符的截断，作为主键一部分创建的索引等。	NOTICE	INFORMATION
warning	报告警告信息， 比如在事务块范围之外的COMMIT。	NOTICE	WARNING
error	报告导致当前命令退出的错误。	WARNING	ERROR

信息严重程度类型	详细说明	系统日志	事件日志
fatal	报告导致当前会话终止的原因。	ERR	ERROR
panic	报告导致整个数据库被关闭的原因。	CRIT	ERROR

14.3.9.3 记录日志的内容

debug_print_parse

参数说明：用于控制打印解析树结果。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启打印结果的功能。
- off表示关闭打印结果的功能。

默认值：off

debug_print_rewritten

参数说明：用于控制打印查询重写结果。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启打印结果的功能。
- off表示关闭打印结果的功能。

默认值：off

debug_print_plan

参数说明：用于设置是否将查询的执行计划打印到日志中。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启打印结果的功能。
- off表示关闭打印结果的功能。

默认值：off

须知

- 只有当日志的级别为log及以上时，debug_print_parse、debug_print_rewritten和debug_print_plan的调试信息才会输出。当这些选项打开时，调试信息只会记录在服务器的日志中，而不会输出到客户端的日志中。通过设置**client_min_messages**和**log_min_messages**参数可以改变日志级别。
- 在打开debug_print_plan开关的情况下需尽量避免调用gs_encrypt_aes128及gs_decrypt_aes128函数，避免敏感参数信息在日志中泄露的风险。同时建议用户在打开debug_print_plan开关生成的日志中对gs_encrypt_aes128及gs_decrypt_aes128函数的参数信息进行过滤后再提供给外部维护人员定位，日志使用完成后请及时删除。

debug_pretty_print

参数说明：设置此选项对debug_print_parse、debug_print_rewritten和debug_print_plan产生的日志进行缩进，会生成易读但比设置为off时更长的输出格式。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示进行缩进。
- off表示不进行缩进。

默认值：on

log_checkpoints

参数说明：控制在服务器日志中记录检查点和重启点的信息。打开此参数时，服务器日志消息包含涉及检查点和重启点的统计量，其中包含需要写的缓存区的数量及写入所花费的时间等。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示打开此参数时，服务器日志消息包含涉及检查点和重启点的统计量。
- off表示关闭此参数时，服务器日志消息包含不涉及检查点和重启点的统计量。

默认值：off

log_connections

参数说明：控制记录客户端的连接请求信息。

该参数属于BACKEND类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

有些客户端程序（例如gsql），在判断是否需要口令的时候会尝试连接两次，因此日志消息中重复的“connection receive”（收到连接请求）并不意味着一定是问题。

取值范围：布尔型

- on表示记录信息。
- off表示不记录信息。

默认值：off

log_disconnections

参数说明：控制记录客户端结束连接信息。

该参数属于BACKEND类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示记录信息。
- off表示不记录信息。

默认值：off

log_duration

参数说明：控制记录每个已完成SQL语句的执行时间。对使用扩展查询协议的客户端、会记录语法分析、绑定和执行每一步所花费的时间。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- 设置为off，该选项与**log_min_duration_statement**的不同之处在于log_min_duration_statement强制记录查询文本。
- 设置为on并且log_min_duration_statement大于零，记录所有持续时间，但是仅记录超过阈值的语句。这可用于在高负载情况下搜集统计信息。

默认值：off

log_error_verbosity

参数说明：控制服务器日志中每条记录的消息写入的详细度。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：枚举类型

- terse输出不包括DETAIL、HINT、QUERY及CONTEXT错误信息的记录。
- verbose输出包括SQLSTATE错误代码 、源代码文件名、函数名及产生错误所在的行号。
- default输出包括DETAIL、HINT、QUERY及CONTEXT错误信息的记录，不包括SQLSTATE错误代码 、源代码文件名、函数名及产生错误所在的行号。

默认值：default

log_hostname

参数说明：选项关闭状态下，连接消息日志只显示正在连接主机的IP地址。打开此选项同时可以记录主机名。由于解析主机名可能需要一定的时间，可能影响数据库的性能。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示可以同时记录主机名。
- off表示不可以同时记录主机名。

默认值：off

log_line_prefix

参数说明：控制每条日志信息的前缀格式。日志前缀类似于printf风格的字符串，在日志的每行开头输出。用以%为开头的“转义字符”代替**表14-7**中的状态信息。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

表 14-7 转义字符表

转义字符	效果
%a	应用程序名称。
%u	用户名。
%d	数据库名。
%r	远端主机名或者IP地址以及远端端口，在不启动log_hostname时显示IP地址及远端端口。
%h	远端主机名或者IP地址，在不启动log_hostname时只显示IP地址。
%p	线程ID。
%t	时间戳（没有毫秒）。
%m	带毫秒的时间戳。
%n	表示指定错误从哪个节点上报的。
%i	命令标签：会话当前执行的命令类型。
%e	SQLSTATE错误码。
%c	会话ID，详见说明。
%l	每个会话或线程的日志编号，从1开始。
%s	进程启动时间。
%v	虚拟事务ID（backendID/ localXID）
%x	事务ID（0表示没有分配事务ID）。
%q	不产生任何输出。如果当前线程是后端线程，忽略这个转义序列，继续处理后面的转义序列；如果当前线程不是后端线程，忽略这个转义序列和它后面的所有转义序列。
%S	会话ID。

转义字符	效果
%T	Trace ID。
%%	字符%。

说明

转义字符%c打印一个会话ID，由两个4字节的十六进制数组成，通过字符“.”分开。这两个十六进制数分别表示进程的启动时间及进程编号，所以%c也可以看作是保存打印这些名目的途径的空间。比如，从pg_stat_activity中产生会话ID，可以用下面的查询：

```
SELECT to_hex(EXTRACT(EPOCH FROM backend_start)::integer) || '.' ||
       to_hex(pid)
FROM pg_stat_activity;
```

- 当log_line_prefix设置为非空值时，请将其最后一个字符作为一个独立的段，以此来直观地与后续的日志进行区分，也可以使用一个标点符号。
- Syslog生成自己的时间戳及进程ID信息，所以当登录日志时，不需要包含这些转义字符。

取值范围：字符串

默认值：%m %n %u %d %h %p %S %x %a

说明

%m %n %u %d %h %p %S %x %a 表示会话开始时间戳、错误上报节点、用户名、数据库名、远程主机名或IP、线程ID、会话ID、事务ID、应用名。

log_lock_waits

参数说明：当一个会话的等待获得一个锁的时间超过**deadlock_timeout**的值时，此选项控制在数据库日志中记录此消息。这对于决定锁等待是否会产生一个坏的行为是非常有用的。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示记录此信息。
- off表示不记录此信息。

默认值：off

log_statement

参数说明：控制记录SQL语句。对于使用扩展查询协议的客户端，记录接收到执行消息的事件和绑定参数的值（内置单引号要双写）。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- 即使log_statement设置为all，包含简单语法错误的语句也不会被记录，因为仅在完成基本的语法分析并确定了语句类型之后才记录日志。在使用扩展查询协议的情况下，在执行阶段之前（语法分析或规划阶段）同样不会记录。将log_min_error_statement设为ERROR或更低才能记录这些语句。
- 设置该参数为非none时，可视为开启相关语句审计功能，数据库DBA可以访问服务端日志查看SQL执行记录。

取值范围：枚举类型

- none表示不记录语句。
- ddl表示记录所有的数据定义语句，比如CREATE、ALTER和DROP语句。
- mod表示记录所有DDL语句，还包括数据修改语句INSERT、UPDATE、DELETE、TRUNCATE和COPY FROM 。
- all表示记录所有语句，PREPARE、EXECUTE和EXPLAIN ANALYZE语句也同样被记录。

默认值：none

log_temp_files

参数说明：控制记录临时文件的删除信息。临时文件可以用来排序、哈希及临时查询结果。当一个临时文件被删除时，将会产生一条日志消息。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，最小值为-1，最大值2147483647，单位KB。

- 正整数表示只记录比log_temp_files设定值大的临时文件的删除信息。
- 值0 表示记录所有的临时文件的删除信息。
- 值-1 表示不记录任何临时文件的删除信息。

默认值：-1

log_timezone

参数说明：设置服务器写日志文件时使用的时区。与**TimeZone**不同，这个值是数据库范围的，针对所有连接到本数据库的会话生效。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，可查询视图PG_TIMEZONE_NAMES（详见《开发指南》的“系统表和系统视图 > 系统视图 > PG_TIMEZONE_NAMES”章节）获得。

默认值：根据OS时区设置

说明

gs_initdb进行相应系统环境设置时会对默认值进行修改。

logging_module

参数说明：用于设置或者显示模块日志在服务端的可输出性。该参数属于会话级参数，不建议通过gs_guc工具来设置。

该参数属于USERSET类型参数，设置请参考**表14-1**中对应设置的方法进行设置。

取值范围：字符串

默认值：LOCK模块日志在服务端输出，其他模块日志在服务端不输出，可由SHOW logging_module查看：

```
ALL,on(LOCK),off(COMMAND,DFS,GUC,GSCLEAN,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,OBS,INDEX,EXECUTOR,OPFUSION,GPC,GSC,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_CHOICE,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,OPT_PLANNER,OPT_STAT_EXT,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PLANHINT,PARQUET,PGSTAT,CARBONDATA,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,WDR_SNAPSHOT,WDR_REPORT,INCRE_CKPT,INCRE_BG_WRITER,DBL_WRT,RTO_RPO,HEARTBEAT,COMM_IPC,COMM_PARAM,TIMESERIES,SCHEMA,SEGMENT_PAGE,LIGHTPROXY,HOTKEY,THREAD_POOL,OPT_AI,WALRECEIVER,USTORE,UPAGE,UBTREE,UNDO,TIMECAPSULE,GEN_COL,DCF,DB4AI,PLDEBUGGER,ADVISOR,SEC,SEC_FE,SEC_LEGER,SEC_POLICY,SEC_SDD,SEC_TDE,COMM_FRAMEWORK,COMM_PROXY,COMM_POOLER,VACUUM,JOB,SPI,NEST_COMPILE,RESOWNER,GSSTACK,LOGICAL_DECODE,GPRC,DISASTER_READ,STANDBY_READ,REPSYNC,SQLPATCH,PARTITION,UBT_NEWPAGE,GPI,GS_DEPENDENCY,LWLOCK)
```

注意

当前版本CN_RETRY不生效。

设置方法：首先，可以通过SHOW logging_module来查看哪些模块是支持可控制的。例如，查询输出结果为：

```
gaussdb=# show logging_module;
logging_module
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
ALL,on(LOCK),off(COMMAND,DFS,GUC,GSCLEAN,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,OBS,INDEX,EXECUTOR,OPFUSION,GPC,GSC,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_CHOICE,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,OPT_PLANNER,OPT_STAT_EXT,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PLANHINT,PARQUET,PGSTAT,CARBONDATA,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,WDR_SNAPSHOT,WDR_REPORT,INCRE_CKPT,INCRE_BG_WRITER,DBL_WRT,RTO_RPO,HEARTBEAT,COMM_IPC,COMM_PARAM,TIMESERIES,SCHEMA,SEGMENT_PAGE,LIGHTPROXY,HOTKEY,THREAD_POOL,OPT_AI,WALRECEIVER,USTORE,UPAGE,UBTREE,UNDO,TIMECAPSULE,GEN_COL,DCF,DB4AI,PLDEBUGGER,ADVISOR,SEC,SEC_FE,SEC_LEGER,SEC_POLICY,SEC_SDD,SEC_TDE,COMM_FRAMEWORK,COMM_PROXY,COMM_POOLER,VACUUM,JOB,SPI,NEST_COMPILE,RESOWNER,GSSTACK,LOGICAL_DECODE,GPRC,DISASTER_READ,STANDBY_READ,REPSYNC,SQLPATCH,PARTITION,UBT_NEWPAGE,GPI,GS_DEPENDENCY,LWLOCK)
(1 row)
```

支持可控制的模块使用大写来标识，特殊标识ALL用于对所有模块日志进行设置。可以使用on/off来控制模块日志的输出。设置SSL模块日志为可输出，使用如下命令：

```
gaussdb=# set logging_module='on(SSL)';
SET
gaussdb=# show
logging_module;
       logging_module
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
ALL,on(SSL,LOCK),off(COMMAND,DFS,GUC,GSCLEAN,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,GDS,TBLSPC,WLM,OBS,INDEX,EXECUTOR,OPFUSION,GPC,GSC,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_CHOICE,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,OPT_PLANNER,OPT_STAT_EXT,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PLANHINT,PARQUET,PGSTAT,CARBONDATA,SNAPSHOT,XACT,HANDL
```

```
E,CLOG,EC,REMOTE,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,WDR_SNAPSHOT,WDR_REPOR
T,INCRE_CKPT,INCRE_BG_WRITER,DBL_WRT,RTO_RPO,HEARTBEAT,COMM_IPC,COMM_PARAM,TIMESERIES,SC
HEMA,SEGMENT_PAGE,LIGHTPROXY,HOTKEY,THREAD_POOL,OPT_AI,WALRECEIVER,USTORE,UPAGE,UBTREE,
UNDO,TIMECAPSULE,GEN_COL,DCF,DB4AI,PLDEBUGGER,ADVISOR,SEC,SEC_FE,SEC_LEGER,SEC_POLICY,SEC_S
DD,SEC_TDE,COMM_FRAMEWORK,COMM_PROXY,COMM_POOLER,VACUUM,JOB,SPI,NEST_COMPILE,RESO
WNER,GSSTACK,LOGICAL_DECODE,GPRC,DISASTER_READ,STANDBY_READ,REPSYNC,SQLPATCH,PARTITION,
UBT_NEWPAGE,GPI,GS_DEPENDENCY,LWLOCK)
(1 row)
```

可以看到模块SSL的日志输出被打开。

ALL标识是相当于一个快捷操作，即对所有模块的日志可输出进行开启或关闭。

```
gaussdb=# set logging_module='off(ALL)';
SET
gaussdb=# show
logging_module;
          logging_module
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
----------------------------------------------------
ALL,on(),off(COMMAND,DFS,GUC,GSCLEAN,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBL
SPC,WLM,OBS,INDEX,EXECUTOR,OPFUSION,GPC,GSC,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT
_JOIN,OPT_AGG,OPT_CHOICE,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,OPT_PLANNER,OPT_STAT_EXT,UDF,CO
OP_ANALYZE,WLMCP,ACCELERATE,PLANHINT,PARQUET,PGSTAT,CARBONDATA,SNAPSHOT,XACT,HANDLE,CL
OG,EC,REMOTE,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,WDR_SNAPSHOT,WDR_REPORT,IN
CRE_CKPT,INCRE_BG_WRITER,DBL_WRT,RTO_RPO,HEARTBEAT,COMM_IPC,COMM_PARAM,TIMESERIES,SCHE
MA,SEGMENT_PAGE,LIGHTPROXY,HOTKEY,THREAD_POOL,OPT_AI,WALRECEIVER,USTORE,UPAGE,UBTREE,U
NDO,TIMECAPSULE,GEN_COL,DCF,DB4AI,PLDEBUGGER,ADVISOR,SEC,SEC_FE,SEC_LEGER,SEC_POLICY,SEC_SD
D,SEC_TDE,COMM_FRAMEWORK,COMM_PROXY,COMM_POOLER,VACUUM,JOB,SPI,NEST_COMPILE,RESOW
NER,GSSTACK,LOGICAL_DECODE,GPRC,DISASTER_READ,STANDBY_READ,REPSYNC,SQLPATCH,PARTITION,UB
T_NEWPAGE,GPI,GS_DEPENDENCY,LWLOCK,LOCK)
(1 row)

gaussdb=# set logging_module='on(ALL)';
SET
gaussdb=# show
logging_module;
         logging_module
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
----------------------------------------------------
ALL,on(COMMAND,DFS,GUC,GSCLEAN,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBLSPC,
WLM,OBS,INDEX,EXECUTOR,OPFUSION,GPC,GSC,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOI
N,OPT_AGG,OPT_CHOICE,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,OPT_PLANNER,OPT_STAT_EXT,UDF,COOP_A
NALYZE,WLMCP,ACCELERATE,PLANHINT,PARQUET,PGSTAT,CARBONDATA,SNAPSHOT,XACT,HANDLE,CLOG,EC
,REMOTE,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,WDR_SNAPSHOT,WDR_REPORT,INCRE_C
KPT,INCRE_BG_WRITER,DBL_WRT,RTO_RPO,HEARTBEAT,COMM_IPC,COMM_PARAM,TIMESERIES,SCHEMA,SE
GMENT_PAGE,LIGHTPROXY,HOTKEY,THREAD_POOL,OPT_AI,WALRECEIVER,USTORE,UPAGE,UBTREE,UNDO,TI
MECAPSULE,GEN_COL,DCF,DB4AI,PLDEBUGGER,ADVISOR,SEC,SEC_FE,SEC_LEGER,SEC_POLICY,SEC_SDD,SEC_
TDE,COMM_FRAMEWORK,COMM_PROXY,COMM_POOLER,VACUUM,JOB,SPI,NEST_COMPILE,RESOWNER,GS
STACK,LOGICAL_DECODE,GPRC,DISASTER_READ,STANDBY_READ,REPSYNC,SQLPATCH,PARTITION,UBT_NEW
PAGE,GPI,GS_DEPENDENCY,LWLOCK,LOCK),off()
(1 row)
```

依赖关系：该参数依赖于log_min_messages参数的设置。

opfusion_debug_mode

参数说明：用于调试简单查询是否进行查询优化。设置成log级别可以在数据库节点的执行计划中看到没有查询优化的具体原因。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：枚举类型

- off表示不打开该功能。

- log表示打开该功能，可以在数据库节点的执行计划中看到没有查询优化的具体原因。

须知

提供在log中显示语句没有查询优化的具体原因，需要将参数设置成log级别，log_min_messages设置成debug4级别，logging_module设置'on(OPFUSION)'，注意log内容可能会比较多，尽可能在调优期间执行少量作业使用。

默认值：off

enable_debug_vacuum

参数说明：允许输出一些与VACUUM相关的日志，便于定位VACUUM相关问题。开发人员专用，不建议普通用户使用。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on/true表示开启此日志开关。
- off/false表示关闭此日志开关。

默认值：off

14.3.9.4 使用 CSV 格式写日志

前提条件

- **log_destination**的值设置为csvlog。
- **logging_collector**的值设置为on。

csvlog 定义

以“逗号分隔值” 即CSV（Comma Separated Value）的形式发出日志。

以下是简单的用来存储CSV形式日志输出的表定义：

```
CREATE TABLE gaussdb_log
(
log_time timestamp(3) with time zone,
node_name text,
user_name text,
database_name text,
process_id bigint,
connection_from text,
"session_id" text,
session_line_num bigint,
command_tag text,
session_start_time timestamp with time zone,
virtual_transaction_id text,
transaction_id bigint,
query_id bigint,
module text,
error_severity text,
sql_state_code text,
message text,
detail text,
```



```
hint text,
internal_query text,
internal_query_pos integer,
context text,
query text,
query_pos integer,
location text,
application_name text
);
```

详细说明请参见**表14-8**。

表 14-8 csvlog 字段含义表

字段名	字段含义	字段名	字段含义
log_time	毫秒级的时间戳	module	日志所属模块
node_name	节点名称	error_severity	ERRORSTATE代码
user_name	用户名	sql_state_code	SQLSTATE代码
database_name	数据库名	message	错误消息
process_id	进程ID	detail	详细错误消息
connection_from	客户主机：端口号	hint	提示
session_id	会话ID	internal_query	内部查询（查询那些导致错误的信息，如果有的话）
session_line_num	每个会话的行数	internal_query_pos	内部查询指针
command_tag	命令标签	context	环境
session_start_time	会话开始时间	query	错误发生位置的字符统计
virtual_transaction_id	常规事务	query_pos	错误发生位置指针
transaction_id	事务ID	location	在GaussDB源代码中错误的位置（如果log_error_verbosity的值设为verbose ）
query_id	查询ID	application_name	应用名称

使用COPY FROM命令将日志文件导入这个表：

```
COPY gaussdb_log FROM '/opt/data/pg_log/logfile.csv' WITH csv;
```

说明

此处的日志名“logfile.csv”要换成实际生成的日志的名称。

简化输入

简化输入到CSV日志文件，可以通过如下操作：

- 设置**log_filename**和**log_rotation_age**，为日志文件提供一个一致的、可预测的命名方案。通过日志文件名，预测一个独立的日志文件完成并进入准备导入状态的时间。
- 将**log_rotation_size**设为0来终止基于尺寸的日志回滚，因为基于尺寸的日志回滚让预测日志文件名变得非常的困难。
- 将**log_truncate_on_rotation**设为on以便区分在同一日志文件中旧的日志数据和新的日志数据。

14.3.10 告警检测

在数据库运行的过程中，会对数据库中的错误场景进行检测，便于用户及早感知到数据库的错误。告警写入的system_alarm日志可以在$GAUSSLOG/cm路径下查看。

enable_alarm

参数说明：允许打开告警检测线程，检测数据库中可能的错误场景。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许打开告警检测线程。
- off表示不允许打开告警检测线程。

默认值：on

说明

该参数生效范围仅为DN节点。

connection_alarm_rate

参数说明：允许和数据库连接的最大并发连接数的比率限制。数据库连接的最大并发连接数为**max_connections*** connection_alarm_rate。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：浮点型，0.0~1.0

默认值：0.9

alarm_report_interval

参数说明：指定告警上报的时间间隔。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，单位为秒。

默认值：10

alarm_component

参数说明：在对告警做上报时，会进行告警抑制，即同一个实例的同一个告警项在alarm_report_interval（默认值为10s）内不做重复上报。在这种情况下设置用于处理告警内容的告警组件的位置，仅sysadmin用户可以访问。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串。

- 若前置脚本gs_preinstall中的--alarm-type参数设置为5时，表示未对接第三方组件，告警写入system_alarm日志，此时GUC参数alarm_component的取值为：/opt/huawei/snas/bin/snas_cm_cmd。
- 若前置脚本gs_preinstall中的--alarm-type参数设置为1时，表示对接第三方组件，此时GUC参数alarm_component的值为第三方组件的可执行程序的绝对路径。

默认值：/opt/huawei/snas/bin/snas_cm_cmd

table_skewness_warning_threshold

参数说明：设置用于表倾斜告警的阈值。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：浮点型，0～1

默认值：1

table_skewness_warning_rows

参数说明：设置用于表倾斜告警的行数。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0～INT_MAX

默认值：100000

14.3.11 运行时统计

14.3.11.1 查询和索引统计收集器

查询和索引统计收集器负责收集数据库系统运行中的统计数据，如在一个表和索引上进行了多少次插入与更新操作、磁盘块的数量和元组的数量、每个表上最近一次执行清理和分析操作的时间等。可以通过查询系统视图pg_stats和pg_statistic查看统计数据。下面的参数设置服务器范围内的统计收集特性。

track_activities

参数说明：控制收集每个会话中当前正在执行命令的统计数据。对于存储过程，打开该参数后，可以通过pg_stat_activity视图看到存储过程内正在执行的perform语句、调用存储过程语句、存储过程内的SQL语句、OPEN CURSOR语句。

该参数属于SUSET类型参数，请参考表14-1中对应设置方法进行设置。

取值范围：布尔型

- on表示开启收集功能。
- off表示关闭收集功能。

默认值：on

track_counts

参数说明：控制收集数据库活动的统计数据。

该参数属于SUSET类型参数，请参考表14-1中对应设置方法进行设置。

取值范围：布尔型

- on表示开启收集功能。
- off表示关闭收集功能。

说明

在AutoVacuum自动清理线程中选择清理的数据库时，需要数据库的统计数据，故默认值设为on。

默认值：on

track_io_timing

参数说明：控制收集数据库I/O调用时序的统计数据。I/O时序统计数据可以在pg_stat_database中查询。

该参数属于SUSET类型参数，请参考表14-1中对应设置方法进行设置。

取值范围：布尔型

- on表示开启收集功能，开启时，收集器会在重复地去查询当前时间的操作系统，这可能会引起某些平台的重大开销，故默认值设置为off。
- off表示关闭收集功能。

默认值：off

track_functions

参数说明：控制收集函数的调用次数和调用耗时的统计数据。

该参数属于SUSET类型参数，请参考表14-1中对应设置方法进行设置。

须知

当SQL语言函数设置为调用查询的“内联”函数时，不管是否设置此选项，这些SQL语言函数无法被追踪到。

取值范围：枚举类型

- pl表示只追踪过程语言函数。
- all表示追踪SQL语言函数。
- none表示关闭函数追踪功能。

默认值：none

track_activity_query_size

参数说明：设置用于跟踪每一个活动会话的当前正在执行命令的字节数。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，100～102400，单位为byte

默认值：1024

update_process_title

参数说明：控制收集因每次服务器接收到一个新的SQL语句时而产生的进程名称更新的统计数据。

进程名称可以通过ps命令进行查看。

该参数属于INTERNAL类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启收集功能。
- off表示关闭收集功能。

默认值：off

stats_temp_directory

参数说明：设置存储临时统计数据的目录，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

将其设置为一个基于RAM的文件系统目录会减少实际的I/O开销并可以提升其性能。

取值范围：字符串

默认值：pg_stat_tmp

track_thread_wait_status_interval

参数说明：用来定期收集thread状态信息的时间间隔。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：0～1天，单位为min。

默认值：30min

enable_save_datachanged_timestamp

参数说明：确定是否收集insert/update/delete, exchange/truncate/drop partition操作对表数据改动的时间。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许收集相关操作对表数据改动的时间。
- off表示禁止收集相关操作对表数据改动的时间。

默认值：on

enable_plan_trace

参数说明：该参数是控制数据库是否开启plan trace特性的开关，该参数不能使用gs_guc命令进行全局设置，只能在连接的session中使用set命令进行设置。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启plan trace特性。
- off表示关闭plan trace特性。

默认值：off

plan_collect_thresh

参数说明：控制收集每个会话中当前正在执行计划的统计数据。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，-1~2147483647

- -1表示不收集运行态计划。
- 0表计划执行之前收集一次运行态计划。
- >0表示当计划中所有算子增量返回tuple数量之和大于等于该值时收集一次运行态计划。

默认值：0

track_sql_count

参数说明：控制对每个会话中当前正在执行的SELECT、INSERT、UPDATE、DELETE、MERGE INTO语句进行计数的统计数据。

在x86架构集中式部署下，硬件配置规格为32核CPU/256GB内存，使用Benchmark SQL 5.0工具测试性能，开关此参数性能影响约0.8%。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启计数功能。
- off表示关闭计数功能。

默认值：on

说明

- track_sql_count参数受track_activities约束：
  - track_activities开启而track_sql_count关闭时，如果查询了gs_sql_count视图，日志中将会有WARNING提示track_sql_count是关闭的；
  - track_activities和track_sql_count同时关闭，那么此时日志中将会有两条WARNING，分别提示track_activities是关闭的和track_sql_count是关闭的；
  - track_activities关闭而track_sql_count开启，此时日志中将仅有WARNING提示track_activities是关闭。
- 当参数关闭时，查询视图的结果为0行。

14.3.11.2 性能统计

在数据库运行过程中，会涉及到锁的访问、磁盘IO操作、无效消息的处理，这些操作都可能是数据库的性能瓶颈，通过GaussDB提供的性能统计方法，可以方便定位性能问题。

输出性能统计日志

参数说明：对每条查询，以下4个选项控制在服务器日志里记录相应模块的性能统计数据，具体含义如下：

- log_parser_stats控制在服务器日志里记录解析器的性能统计数据。
- log_planner_stats控制在服务器日志里记录查询优化器的性能统计数据。
- log_executor_stats控制在服务器日志里记录执行器的性能统计数据。
- log_statement_stats控制在服务器日志里记录整个语句的性能统计数据。

这些参数只能辅助管理员进行粗略分析，类似Linux中的操作系统工具getrusage()。

这些参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

- log_statement_stats记录总的语句统计数据，而其他的只记录针对每个模块的统计数据。
- log_statement_stats不能和其他任何针对每个模块统计的选项一起打开。

取值范围：布尔型

- on表示开启记录性能统计数据的功能。
- off表示关闭记录性能统计数据的功能。

默认值：off

14.3.12 自动清理

系统自动清理线程（autovacuum）自动执行VACUUM和ANALYZE命令，回收被标识为删除状态的记录空间，并更新表的统计数据。

autovacuum

参数说明：控制数据库自动清理线程（autovacuum）的启动。自动清理线程运行的前提是将**track_counts**设置为on。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

说明

- 如果希望系统在故障恢复后，具备自动清理两阶段事务的功能，请将autovacuum设置为on;
- 当设置autovacuum为on，**autovacuum_max_workers**为0时，表示系统不会自动进行autovacuum，只会在故障恢复后，自动清理两阶段事务；
- 当设置autovacuum为on，**autovacuum_max_workers**大于0时，表示系统不仅在故障恢复后，自动清理两阶段事务，并且还可以自动清理线程。

取值范围：布尔型

- on表示开启数据库自动清理线程。
- off表示关闭数据库自动清理线程。

默认值：on

autovacuum_mode

参数说明：该参数仅在autovacuum设置为on的场景下生效，它控制autoanalyze或autovacuum的打开情况。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：枚举类型

- analyze表示只做autoanalyze。
- vacuum表示只做autovacuum。
- mix表示autoanalyze和autovacuum都做。
- none表示二者都不做。

默认值：mix

autoanalyze_timeout

参数说明：设置autoanalyze的超时时间。在对某张表做autoanalyze时，如果该表的analyze时长超过了autoanalyze_timeout，则自动取消该表此次analyze。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，单位是s，0~2147483。

默认值：5min（即300s）

autovacuum_io_limits

参数说明：控制autovacuum进程每秒触发IO的上限。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0～1073741823和-1。其中-1表示不控制，而是使用系统默认控制组。

默认值：-1

log_autovacuum_min_duration

参数说明：当自动清理的执行时间大于或者等于某个特定的值时，向服务器日志中记录自动清理执行的每一步操作。设置此选项有助于追踪自动清理的行为。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

举例如下：

将log_autovacuum_min_duration设置为250ms，记录所有运行大于或者等于250ms的自动清理命令的相关信息。

取值范围：整型，最小值为-1，最大值为2147483647，单位为毫秒。

- 当参数设置为0时，表示所有的自动清理操作都记录到日志中。
- 当参数设置为-1时，表示所有的自动清理操作都不记录到日志中。
- 当参数设置为非-1时，当由于锁冲突的存在导致一个自动清理操作被跳过，记录一条消息。

默认值：-1

autovacuum_max_workers

参数说明：设置能同时运行的自动清理线程的最大数量，该参数的取值上限与GUC参数max_connections和job_queue_processes大小有关。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，最小值为0（表示不会自动进行autovacuum），理论最大值为262143，实际最大值为动态值，计算公式为“262143 - max_inner_tool_connections - max_connections - job_queue_processes - max_concurrent_autonomous_transactions - 辅助线程数 – autovacuum的launcher线程数 - 1”，其中辅助线程数和autovacuum的launcher线程数由两个宏来指定，当前版本的默认值分别为20和2。

默认值：3

autovacuum_naptime

参数说明：设置两次自动清理操作的时间间隔。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，单位为s，最小值为1，最大值为2147483。

默认值：10min（即600s）

autovacuum_vacuum_threshold

参数说明：设置触发VACUUM的阈值。当表上被删除或更新的记录数超过设定的阈值时才会对这个表执行VACUUM操作。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，最小值为0，最大值为2147483647。

默认值：50

autovacuum_analyze_threshold

参数说明：设置触发ANALYZE操作的阈值。当表上被删除、插入或更新的记录数超过设定的阈值时才会对这个表执行ANALYZE操作。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，最小值为0，最大值为2147483647。

默认值：50

autovacuum_vacuum_scale_factor

参数说明：设置触发一个VACUUM时增加到autovacuum_vacuum_threshold的表大小的缩放系数。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：浮点型，0.0～100.0

默认值：0.2

autovacuum_analyze_scale_factor

参数说明：设置触发一个ANALYZE时增加到autovacuum_analyze_threshold的表大小的缩放系数。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：浮点型，0.0～100.0

默认值：0.1

autovacuum_freeze_max_age

参数说明：设置事务内的最大时间，使得表的pg_class.relfrozenxid字段在VACUUM操作执行之前被写入。

- VACUUM也可以删除pg_clog/子目录中的旧文件。
- 即使自动清理线程被禁止，系统也会调用自动清理线程来防止循环重复。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：长整型，100 000～576 460 752 303 423 487

默认值：4000000000

autovacuum_vacuum_cost_delay

参数说明：设置在自动VACUUM操作里使用的开销延迟数值。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，-1～100，单位为毫秒（ms）。其中-1表示使用常规的vacuum_cost_delay。

默认值：20ms

autovacuum_vacuum_cost_limit

参数说明：设置在自动VACUUM操作里使用的开销限制数值。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，-1～10000。其中-1表示使用常规的vacuum_cost_limit。

默认值：-1

defer_csn_cleanup_time

参数说明：用来指定本地回收时间间隔。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~2147483647，单位为毫秒（ms）。

默认值：5s（即5000ms）

14.3.13 客户端连接缺省设置

14.3.13.1 语句行为

介绍SQL语句执行过程的相关默认参数。

search_path

参数说明：当一个被引用对象没有指定模式时，此参数设置模式搜索顺序。它的值由一个或多个模式名构成，不同的模式名用逗号隔开。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

- 当前会话如果存放临时表的模式时，可以使用别名pg_temp将它列在搜索路径中，如'pg_temp，public'。存放临时表的模式始终会作为第一个被搜索的对象，排在pg_catalog和search_path中所有模式的前面，即具有第一搜索优先级。建议用户不要在search_path中显示设置pg_temp。如果在search_path中指定了pg_temp，但不是在最前面，系统会提示设置无效，pg_temp仍被优先搜索。通过使用别名pg_temp，系统只会在存放临时表的模式中搜索表、视图和数据类型这样的数据库对象，不会在里面搜索函数或运算符这样的数据库对象。
- 系统表所在的模式pg_catalog，总是排在search_path中指定的所有模式前面被搜索，即具有第二搜索优先级（pg_temp具有第一搜索优先级）。建议用户不要在search_path中显式设置pg_catalog。如果在search_path中指定了pg_catalog，但不是在最前面，系统会提示设置无效，pg_catalog仍被第二优先搜索。
- 当没有指定一个特定模式而创建一个对象时，它们被放置到以search_path为命名的第一个有效模式中。当搜索路径为空时，会报错误。
- 通过SQL函数current_schema可以检测当前搜索路径的有效值。这和检测search_path的值不尽相同，因为current_schema显示search_path中首位有效的模式名称。

取值范围：字符串

说明

- 设置为"$user"，public时，支持共享数据库（没有用户具有私有模式和所有共享使用public），用户私有模式和这些功能的组合使用。可以通过改变默认搜索路径来获得其他效果，无论是全局化的还是私有化的。
- 设置为空串（''）的时候，系统会自动转换成一对双引号。
- 设置的内容中包含双引号，系统会认为是不安全字符，会将每个双引号转换成一对双引号。

默认值： "$user",public

说明

$user表示与当前会话用户名同名的模式名，如果这样的模式不存在，$user将被忽略。

current_schema

参数说明： 此参数设置当前的模式。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围： 字符串

默认值： "$user",public

说明

$user表示与当前会话用户名同名的模式名，如果这样的模式不存在，$user将被忽略。

default_tablespace

参数说明： 当CREATE命令没有明确声明表空间时，所创建对象（表和索引等）的缺省表空间。

- 值是一个表空间的名称或者一个表示使用当前数据库缺省表空间的空字符串。若指定的是一个非默认表空间，用户必须具有它的CREATE权限，否则尝试创建会失败。
- 临时表不使用此参数，可以用**temp_tablespaces**代替。
- 创建数据库时不使用此参数。默认情况下，一个新的数据库从模板数据库继承表空间配置。

  该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围： 字符串，其中空表示使用默认表空间。

默认值： 空

temp_tablespaces

参数说明： 当一个CREATE命令没有明确指定一个表空间时，temp_tablespaces指定了创建临时对象（临时表和临时表的索引）所在的表空间。在这些表空间中创建临时文件用来做大型数据的排序工作。

其值是一系列表空间名的列表。如果列表中有多个表空间时，每次临时对象的创建，GaussDB会在列表中随机选择一个表空间；如果在事务中，连续创建的临时对象被放置在列表里连续的表空间中。如果选择的列表中的元素是一个空串，GaussDB将自动将当前的数据库设为默认的表空间。

该参数属于USERSET类型参数，请参考表14-1中对应设置方法进行设置。

取值范围：字符串。空字符串表示所有的临时对象仅在当前数据库默认的表空间中创建，请参见default_tablespace。

默认值：空

check_function_bodies

参数说明：设置是否在CREATE FUNCTION执行过程中进行函数体字符串的合法性验证。为了避免产生问题（比如避免从转储中恢复函数定义时向前引用的问题），偶尔会禁用验证。开启后主要验证存储过程中PLSQL的词语法问题，包括数据类型、语句和表达式等，对于其中出现的SQL则在Create阶段不做检查而采用了运行时检查的方式。

该参数属于USERSET类型参数，请参考表14-1中对应设置方法进行设置。

取值范围：布尔型

- on表示在CREATE FUNCTION执行过程中进行函数体字符串的合法性验证。
- off表示在CREATE FUNCTION执行过程中不进行函数体字符串的合法性验证。

默认值：on

default_transaction_isolation

参数说明：设置默认的事务隔离级别。

该参数属于USERSET类型参数，请参考表14-1中对应设置方法进行设置。

说明

当前版本暂不支持设置默认的事务隔离级别，默认为read committed，请勿自行修改。

取值范围：枚举类型

- read committed表示事务读已提交。
- repeatable read表示事务可重复读。
- serializable，GaussDB目前功能上不支持此隔离级别，等价于repeatable read。

默认值：read committed

default_transaction_read_only

参数说明：设置每个新创建事务是否是只读状态。

该参数属于USERSET类型参数，请参考表14-1中对应设置方法进行设置。

注意

该参数设为on后只读，无法执行dml和写事务。

取值范围：布尔型

- on表示只读状态。

- off表示非只读状态。

默认值：off

default_transaction_deferrable

参数说明：控制每个新事务的默认延迟状态。只读事务或者那些比序列化更加低的隔离级别的事务除外。

GaussDB不支持可串行化的隔离级别，因此，该参数无实际意义。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示默认延迟。
- off表示默认不延迟。

默认值：off

session_replication_role

参数说明：控制当前会话与复制相关的触发器和规则的行为。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

设置此参数会丢弃之前任何缓存的执行计划。

取值范围：枚举类型

- origin表示从当前会话中复制插入、删除、更新等操作。
- replica表示从其他地方复制插入、删除、更新等操作到当前会话。
- local表示函数执行复制时会检测当前登录数据库的角色并采取相应的操作。

默认值：origin

statement_timeout

参数说明：当语句执行时间超过该参数设置的时间（从服务器收到命令时开始计时）时，该语句将会报错并退出执行。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。默认值0代表该参数不生效。

取值范围：整型，0 ~ 2147483647，单位为毫秒。

默认值：0

vacuum_freeze_min_age

参数说明：指定VACUUM在扫描一个表时用于判断是否用FrozenXID替换记录的xmin字段（在同一个事务中）。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0 ~ 576 460 752 303 423 487

说明

尽管随时可以将此参数设为上述取值范围之间的任意值，但是，VACUUM将默认其有效值范围限制在**autovacuum_freeze_max_age**的50%以内。

默认值：2000000000

vacuum_freeze_table_age

参数说明：指定VACUUM对全表的扫描冻结元组的时间。如果当前事务号与表pg_class.relfrozenxid64字段的差值已经大于参数指定的时间时，VACUUM对全表进行扫描。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0～576 460 752 303 423 487

说明

尽管随时可以将此参数设为上述取值范围之间的值，但是，VACUUM将默认其有效值范围限制在**autovacuum_freeze_max_age**的95%以内。定期的手动VACUUM可以在对此表的反重叠自动清理启动之前运行。

默认值：4000000000

bytea_output

参数说明：设置bytea类型值的输出格式。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：枚举类型

- hex：将二进制数据编码为每字节2位十六进制数字。
- escape：传统化的PostgreSQL格式。采用以ASCII字符序列表示二进制串的方法，同时将那些无法表示成ASCII字符的二进制串转换成特殊的转义序列。

默认值：hex

xmlbinary

参数说明：设置二进制值是如何在XML中进行编码的。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：枚举类型

- base64
- hex

默认值：base64

xmloption

参数说明：当XML和字符串值之间进行转换时，设置document或content是否是隐含的。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：枚举类型

- document：表示HTML格式的文档。
- content：普通的字符串。

默认值：content

max_compile_functions

参数说明：设置服务器存储的函数编译结果的最大数量。存储过多的函数和存储过程的编译结果可能占用很大内存。将此参数设置为一个合理的值，有助于减少内存占用，提升系统性能。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1 ~ 2147483647。

默认值：

1000（196核CPU/1536G内存，128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，64核CPU/512G内存，60核CPU/480G内存，32核CPU/256G内存，16核CPU/128G内存，8核CPU/64G内存，4核CPU/32G内存）；10（4核CPU/16G内存）

gin_pending_list_limit

参数说明：设置当GIN索引启用fastupdate时，pending list容量的最大值。当pending list的容量大于设置值时，会把pending list中数据批量移动到GIN索引数据结构中以进行清理。单个GIN索引可通过更改索引存储参数覆盖此设置值。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，最小值为64，最大值为INT_MAX，单位为KB。

默认值：4MB

14.3.13.2 区域和格式化

介绍时间格式设置的相关参数。

DateStyle

参数说明：设置日期和时间值的显示格式，以及有歧义的输入值的解析规则。

这个变量包含两个独立的加载部分：输出格式声明（ISO、Postgres、SQL、German）和输入输出的年/月/日顺序（DMY、MDY、YMD）。这两个可以独立设置或者一起设置。关键字Euro和European等价于DMY；关键字US、NonEuro、NonEuropean等价于MDY。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

默认值：ISO, MDY

说明

gs_initdb会将这个参数初始化成与**lc_time**一致的值。

设置建议：优先推荐使用ISO格式。Postgres、SQL和German均采用字母缩写的形式来表示时区，例如“EST、WST、CST”等。这些缩写可同时指代不同的时区，比如CST可同时代表美国中部时间(Central Standard Time (USA) UT-6:00)、澳大利亚中部时间(Central Standard Time (Australia) UT+9:30)、中国标准时间(China Standard Time UT+8:00)。这种情况下在时区转化时可能会得不到正确的结果，从而引发其他问题。

IntervalStyle

参数说明：设置区间值的显示格式。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：枚举类型

- sql_standard表示产生与SQL标准规定匹配的输出。
- postgres表示产生与PostgreSQL 8.4版本相匹配的输出，当**DateStyle**参数被设为ISO时。
- postgres_verbose表示产生与PostgreSQL 8.4版本相匹配的输出，当**DateStyle**参数被设为non_ISO时。
- iso_8601表示产生与在ISO 8601中定义的“格式与代号”相匹配的输出。
- a表示与numtodsinterval函数相匹配的输出结果，详细请参考《开发指南》的“SQL参考 > 函数和操作符 > 时间和日期处理函数和操作符”章节中的numtodsinterval内容。

须知

IntervalStyle参数也会影响不明确的间隔输入的说明。

默认值：postgres

TimeZone

参数说明：设置显示和解释时间类型数值时使用的时区。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，可查询视图PG_TIMEZONE_NAMES（详见《开发指南》的“系统表和系统视图 > 系统视图 > PG_TIMEZONE_NAMES”章节）获得。

默认值：PRC

说明

gs_initdb将设置一个与其系统环境一致的时区值。

timezone_abbreviations

参数说明：设置服务器接受的时区缩写值。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，可查询视图pg_timezone_names获得。

默认值：Default

说明

Default表示通用时区的缩写, 适合绝大部分情况。但也可设置其他诸如 'Australia' 和 'India' 等用来定义特定的安装。而设置除此之外的时区缩写, 需要在建数据库之前通过相应的配置文件进行设置。

extra_float_digits

参数说明：这个参数为浮点数值调整显示的数据位数，浮点类型包括float4、float8 以及几何数据类型。参数值加在标准的数据位数上（FLT_DIG或DBL_DIG中合适的）。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，-15～3

说明

- 设置为3，表示包括部分关键的数据位。这个功能对转储那些需要精确恢复的浮点数据特别有用。
- 设置为负数，表示消除不需要的数据位。

默认值：0

client_encoding

参数说明：设置客户端的字符编码类型。

请根据前端业务的情况确定。尽量客户端编码和服务器端编码一致，提高效率。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：兼容PostgreSQL所有的字符编码类型。其中UTF8表示使用数据库的字符编码类型。

说明

- 使用命令locale -a查看当前系统支持的区域和相应的编码格式，并可以选择进行设置。
- 默认情况下，gs_initdb会根据当前的系统环境初始化此参数，通过locale命令可以查看当前的配置环境。
- 参数建议保持默认值，不建议通过gs_guc工具或其他方式直接在postgresql.conf文件中设置client_encoding参数，即使设置也不会生效，以保证数据库内部通信编码格式一致。

默认值：UTF8

推荐值：SQL_ASCII/UTF8

lc_messages

参数说明：设置信息显示的语言。

- 可接受的值是与系统相关的。
- 在一些系统上，这个区域范畴并不存在，不过仍然允许设置这个变量，只是不会有任何效果。同样，也有可能是所期望的语言的翻译信息不存在。在这种情况下，用户仍然能看到英文信息。

  该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

说明

- 使用命令locale -a查看当前系统支持的区域和相应的编码格式，并可以选择进行设置。
- 默认情况下，gs_initdb会根据当前的系统环境初始化此参数，通过locale命令可以查看当前的配置环境。

默认值：C

lc_monetary

参数说明：设置货币值的显示格式，影响to_char之类的函数的输出。可接受的值是系统相关的。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

说明

- 使用命令locale -a查看当前系统支持的区域和相应的编码格式，并可以选择进行设置。
- 默认情况下，gs_initdb会根据当前的系统环境初始化此参数，通过locale命令可以查看当前的配置环境。

默认值：C

lc_numeric

参数说明：设置数值的显示格式，影响to_char之类的函数的输出。可接受的值是系统相关的。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

说明

- 使用命令locale -a查看当前系统支持的区域和相应的编码格式，并可以选择进行设置。
- 默认情况下，gs_initdb会根据当前的系统环境初始化此参数，通过locale命令可以查看当前的配置环境。

默认值：C

lc_time

参数说明：设置时间和区域的显示格式，影响to_char之类的函数的输出。可接受的值是系统相关的。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

说明

- 使用命令locale -a查看当前系统支持的区域和相应的编码格式，并可以选择进行设置。
- 默认情况下，gs_initdb会根据当前的系统环境初始化此参数，通过locale命令可以查看当前的配置环境。

默认值：C

default_text_search_config

参数说明：设置全文检索的配置信息。

如果设置为不存在的文本搜索配置时将会报错。如果default_text_search_config对应的文本搜索配置被删除，需要重新设置default_text_search_config，否则会报设置错误。

- 其被文本搜索函数使用，这些函数并没有一个明确指定的配置。
- 当与环境相匹配的配置文件确定时，gs_initdb会选择一个与环境相对应的设置来初始化配置文件。

  该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

说明

GaussDB支持pg_catalog.english、pg_catalog.simple两种配置。

默认值：pg_catalog.english

14.3.13.3 其他缺省

主要介绍数据库系统默认的库加载参数。

dynamic_library_path

参数说明：设置数据查找动态加载的共享库文件的路径。当需要打开一个可以动态装载的模块并且在CREATE FUNCTION或LOAD命令里面声明的名称没有目录部分时，系统将搜索这个目录以查找声明的文件，仅sysadmin用户可以访问。

用于dynamic_library_path的数值必须是一个冒号分隔的绝对路径列表。当一个路径名称以特殊变量$libdir为开头时，会替换为GaussDB发布提供的模块安装路径。例如：

```
dynamic_library_path = '/usr/local/lib/gaussdb:/opt/testgs/lib:$libdir'
```

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

说明

设置为空字符串，表示关闭自动路径搜索。

默认值：$libdir

gin_fuzzy_search_limit

参数说明：设置GIN索引返回的集合大小的上限。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~2147483647

默认值：0

local_preload_libraries

参数说明：指定一个或多个共享库，它们在开始连接前预先加载。多个加载库之间用逗号分隔，除了双引号，所有的库名都转换为小写。

- 并非只有系统管理员才能更改此选项，因此只能加载安装的标准库目录下plugins子目录中的库文件，数据库管理员有责任确保该目录中的库都是安全的。local_preload_libraries中指定的项可以明确含有该目录，例如$libdir/plugins/mylib；也可以仅指定库的名称，例如mylib（等价于$libdir/plugins/mylib）。
- 与shared_preload_libraries不同，在会话开始之前加载模块与在会话中使用到该模块的时候临时加载相比并不具有性能优势。相反，这个特性的目的是为了调试或者测量在特定会话中不明确使用LOAD加载的库。例如针对某个用户将该参数设为ALTER USER SET来进行调试。
- 当指定的库未找到时，连接会失败。
- 每一个支持GaussDB的库都有一个“magic block”用于确保兼容性，因此不支持GaussDB的库不能通过这个方法加载。

该参数属于BACKEND类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

默认值：空

14.3.14 锁管理

在GaussDB中，并发执行的事务由于竞争资源会导致死锁。本节介绍的参数主要管理事务锁的机制。

deadlock_timeout

参数说明：设置死锁超时检测时间，以毫秒为单位。当申请的锁超过设定值时，系统会检查是否产生了死锁。该参数仅针对常规锁生效。

- 死锁的检查代价是比较高的，服务器不会在每次等待锁的时候都运行这个过程。在系统运行过程中死锁是不经常出现的，因此在检查死锁前只需等待一个相对较短的时间。增加这个值就减少了无用的死锁检查浪费的时间，但是会减慢真正的死锁错误报告的速度。在一个负载过重的服务器上，用户可能需要增大它。这个值的设置应该超过事务持续时间，这样就可以减少在锁释放之前就开始死锁检查的问题。
- 当设置**log_lock_waits**为on时，deadlock_timeout决定一个等待时间来将查询执行过程中的锁等待耗时信息写入日志。如果要研究锁延时情况，可以设置deadlock_timeout的值比正常情况小。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1~2147483647，单位为毫秒（ms）。

默认值：1s

lockwait_timeout

参数说明：控制单个锁的最长等待时间。当申请的锁等待时间超过设定值时，系统会报错。该参数仅针对常规锁生效。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0 ~ INT_MAX，单位为毫秒（ms）。

默认值：20min

update_lockwait_timeout

参数说明：允许并发更新参数开启情况下，该参数控制并发更新同一行时单个锁的最长等待时间。当申请的锁等待时间超过设定值时，系统会报错。该参数仅针对常规锁生效。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0 ~ 2147483647，单位为毫秒（ms）。

默认值：2min（120000ms）

max_locks_per_transaction

参数说明：控制每个事务能够得到的平均的对象锁的数量。

- 共享的锁表的大小是以假设任意时刻最多只有max_locks_per_transaction*(max_connections+max_prepared_transactions) 个独立的对象需要被锁住为基础进行计算的。不超过设定数量的多个对象可以在任一时刻同时被锁定。当在一个事务里面修改很多不同的表时，可能需要提高这个默认数值。只能在数据库启动的时候设置。
- 增大这个参数可能导致GaussDB请求更多的System V共享内存，有可能超过操作系统的缺省配置。
- 当运行备机时，请将此参数设置不小于主机上的值，否则，在备机上查询操作不会被允许。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，10 ~ INT_MAX

默认值：

256（196核CPU/1536G内存，128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，64核CPU/512G内存，60核CPU/480G内存，32核CPU/256G内存，16核CPU/128G内存，8核CPU/64G内存，4核CPU/32G内存）；64（4核CPU/16G内存）

max_pred_locks_per_transaction

参数说明：控制每个事务允许断定锁的最大数量，是一个平均值。

- 共享的断定锁表的大小是以假设任意时刻最多只有max_pred_locks_per_transaction*(max_connections +max_prepared_transactions) 个独立的对象需要被锁住为基础进行计算的。不超

过设定数量的多个对象可以在任一时刻同时被锁定。当在一个事务里面修改很多不同的表时，可能需要提高这个默认数值。只能在服务器启动的时候设置。

- 增大这个参数可能导致GaussDB请求更多的System V共享内存，有可能超过操作系统的缺省配置。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，10 ~ INT_MAX

默认值：64

gs_clean_timeout

参数说明：控制主节点周期性清理临时表的时间，是一个平均值。

- 数据库连接异常终止时，通常会有临时表残留，此时需要对数据库中的临时表进行清理。
- 增大这个参数可能导致GaussDB临时表清理时间延长。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0 ~ 2147483，单位为秒（s）。

默认值：1min

partition_lock_upgrade_timeout

参数说明：在执行某些查询语句的过程中，会需要将分区表上的锁级别由允许读的ExclusiveLock级别升级到读写阻塞的AccessExclusiveLock级别。如果此时已经存在并发的读事务，那么该锁升级操作将阻塞等待。partition_lock_upgrade_timeout为尝试锁升级的等待超时时间。

- 在分区表上进行MERGE PARTITION和CLUSTER PARTITION操作时，都利用了临时表进行数据重排和文件交换，为了最大程度提高分区上的操作并发度，在数据重排阶段给相关分区加锁ExclusiveLock，在文件交换阶段加锁AccessExclusiveLock。
- 常规加锁方式是等待加锁，直到加锁成功，或者等待时间超过**lockwait_timeout**发生超时失败。
- 在分区表上进行MERGE PARTITION或CLUSTER PARTITION操作时，进入文件交换阶段需要申请加锁AccessExclusiveLock，加锁方式是尝试性加锁，加锁成功了则立即返回，不成功则等待50ms后继续下次尝试，加锁超时时间使用会话级设置参数partition_lock_upgrade_timeout。
- 特殊值：若partition_lock_upgrade_timeout取值-1，表示无限等待，即不停的尝试锁升级，直到加锁成功。

  该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，最小值-1，最大值3000，单位为秒（s）。

默认值：1800

fault_mon_timeout

参数说明：轻量级死锁检测周期。该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，最小值0，最大值1440，单位为分钟（min）

默认值：5min

enable_online_ddl_waitlock

参数说明：控制DDL是否会阻塞等待pg_advisory_lock等数据库锁。主要用于OM在线操作场景，不建议用户设置。

该参数属于SIGHUP类型参数，参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启。
- off表示关闭。

默认值：off

xloginsert_locks

参数说明：控制用于并发写预写式日志锁的个数。主要用于提高写预写式日志的效率。

该参数属于POSTMASTER类型参数，参考**表14-1**中对应设置方法进行设置。

取值范围：整型，最小值1，最大值1000。若CPU为NUMA架构，数值必须为NUMA节点数量的整数倍。

默认值：16

num_internal_lock_partitions

参数说明：控制内部轻量级锁分区的个数。主要用于各类场景的性能调优。内容以关键字和数字的KV方式组织，各个不同类型锁之间以逗号隔开。先后顺序对设置结果不影响，例如“CLOG_PART=256,CSNLOG_PART=512”等同于“CSNLOG_PART=512,CLOG_PART=256”。重复设置同一关键字时，以最后一次设置为准，例如“CLOG_PART=256,CLOG_PART=2”，设置的结果为CLOG_PART=2。当没有设置关键字时，则为默认值，各类锁的使用描述和最大、最小、默认值如下。

- CLOG_PART：CLOG文件控制器的个数，增大该值可以提高CLOG日志写入效率，提升事务提交性能，但是会增大内存使用；减小该值会减少相应内存使用，但可能使得CLOG日志写入冲突变大，影响性能。最小值为1，最大值为256。
- CSNLOG_PART：CSNLOG文件控制器的个数，增大该值可以提高CSNLOG日志写入效率，提升事务提交性能，但是会增大内存使用；减小该值会减少相应内存使用，但可能使得CSNLOG日志写入冲突变大，影响性能。最小值为1，最大值为512。
- LOG2_LOCKTABLE_PART：常规锁表锁分区个数的2对数，增大该值可以提升正常流程常规锁获取锁的并行度，但是可能增加锁转移和锁消除时的耗时，对于等待事件在LockMgrLock时，可以调大该锁增加性能。最小值4，即锁分区数为16；最大值为16，即锁分区数为65536。
- TWOPHASE_PART：两阶段事务锁的分区数，调大该值可以提高两阶段事务提交的并发数。最小值为1，最大值为64。
- FASTPATH_PART：每个线程可以不通过主锁表拿锁的最大锁个数，对于分区表读取、更新、插入、删除操作且等待事件在LockMgrLock时，可以通过调大该值避

免获取LockMgrLock提升性能，建议调整数量大于等于分区数*（1+本地索引数量）+全局索引数量+10，调大该值会额外增加内存。最小值为20，最大值为10000。

该参数属于POSTMASTER类型参数，参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

默认值：

'CLOG_PART=256,CSNLOG_PART=512,LOG2_LOCKTABLE_PART=4,TWOPHASE_PART=1,FASTPATH_PART=20'（196核CPU/1536G内存，128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，64核CPU/512G内存，60核CPU/480G内存，32核CPU/256G内存，16核CPU/128G内存，8核CPU/64G内存，4核CPU/32G内存）；
'CLOG_PART=8,CSNLOG_PART=16,LOG2_LOCKTABLE_PART=4,TWOPHASE_PART=1,FASTPATH_PART=20'（4核CPU/16G内存）

enable_wait_exclusive_lock

参数说明：控制ProcArrayLock的排他锁hang死检测与治愈功能的开关。

该参数属于SIGHUP类型参数，参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on：表示开启。
- off：表示关闭。

默认值：on

14.3.15 版本和平台兼容性

14.3.15.1 历史版本兼容性

GaussDB介绍数据库的向下兼容性和对外兼容性特性的参数控制。数据库系统的向后兼容性能够为旧版本的数据库应用提供支持。本节介绍的参数主要控制数据库的向后兼容性。

array_nulls

参数说明：控制数组输入解析器是否将未用引用的NULL识别为数组的一个NULL元素。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许向数组中输入空元素。
- off表示向下兼容旧式模式。仍然能够创建包含NULL值的数组。

默认值：on

backslash_quote

参数说明：控制字符串文本中的单引号是否能够用\'表示。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

在字符串文本符合SQL标准的情况下，\没有任何其他含义。这个参数影响的是如何处理不符合标准的字符串文本，包括明确的字符串转义语法是（E'...'）。

取值范围：枚举类型

- on表示一直允许使用\'表示。
- off表示拒绝使用\'表示。
- safe_encoding表示仅在客户端字符集编码不会在多字节字符末尾包含\的ASCII值时允许。

默认值：safe_encoding

escape_string_warning

参数说明：警告在普通字符串中直接使用反斜杠转义。

- 如果需要使用反斜杠作为转义，可以调整为使用转义字符串语法(E'...')来做转义，因为在每个SQL标准中，普通字符串的默认行为现在将反斜杠作为一个普通字符。
- 这个变量可以帮助定位需要改变的代码。
- 使用E转义会导致部分场景下日志记录不全。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

默认值：on

lo_compat_privileges

参数说明：控制是否启动对大对象权限检查的向后兼容模式。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

on表示当读取或修改大对象时禁用权限检查，与PostgreSQL 9.0以前的版本兼容。

默认值：off

quote_all_identifiers

参数说明：当数据库生成SQL时，此选项强制引用所有的标识符（包括非关键字）。这将影响到EXPLAIN的输出及函数的结果，例如pg_get_viewdef。详细说明请参见gs_dump的--quote-all-identifiers选项。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示打开强制引用。
- off表示关闭强制引用。

默认值：off

sql_inheritance

参数说明：控制继承语义。用来控制继承表的访问策略，off表示各种命令不能访问子表，即默认使用ONLY关键字。这是为了兼容旧版本而设置的。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示可以访问子表。
- off表示不访问子表。

默认值：on

standard_conforming_strings

参数说明：控制普通字符串文本（'...'）中是否按照SQL标准把反斜扛当普通文本。

- 应用程序通过检查这个参数可以判断字符串文本的处理方式。
- 建议明确使用转义字符串语法（E'...'）来转义字符。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示打开控制功能。
- off表示关闭控制功能。

默认值：on

synchronize_seqscans

参数说明：控制启动同步的顺序扫描。在大约相同的时间内并行扫描读取相同的数据块，共享I/O负载。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示扫描可能从表的中间开始，然后选择"环绕"方式来覆盖所有的行，为了与已经在进行中的扫描活动同步。这可能会造成没有用ORDER BY子句的查询得到行排序造成不可预测的后果。
- off表示确保顺序扫描是从表头开始的。

默认值：on

enable_beta_features

参数说明：控制开启某些非正式发布的特性，仅用于POC验证。这些特性属于延伸特性，建议客户谨慎开启，在某些功能场景下可能存在问题。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启这些功能受限的特性，保持前向兼容。但某些场景可能存在功能上的问题。
- off表示禁止使用这些特性。

默认值：off

default_with_oids

参数说明：在没有声明WITH OIDS和WITHOUT OIDS的情况下，这个选项控制在新创建的表中CREATE TABLE和CREATE TABLE AS是否包含一个OID字段。它还决定SELECT INTO创建的表里面是否包含OID 。

不推荐在用户表中使用OID，故默认设置为off。需要带有OID字段的表应该在创建时声明WITH OIDS 。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示在新创建的表中CREATE TABLE和CREATE TABLE AS可以包含一个OID字段。
- off表示在新创建的表中CREATE TABLE和CREATE TABLE AS不可以包含一个OID字段。

默认值：off

14.3.15.2 平台和客户端兼容性

很多平台都使用数据库系统，数据库系统的对外兼容性给平台提供了很大的方便。

convert_string_to_digit

参数说明：设置隐式转换优先级，是否优先将字符串转为数字。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示优先将字符串转为数字。
- off表示不优先将字符串转为数字。

默认值：on

须知

该参数调整会修改内部数据类型转换规则，导致不可预期的行为，请谨慎操作。

nls_timestamp_format

参数说明：设置时间戳默认格式。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

默认值：DD-Mon-YYYY HH:MI:SS.FF AM

nls_timestamp_tz_format

参数说明：设置带时区时间戳默认格式。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，支持格式同nls_timestamp_format。

默认值：DD-Mon-YYYY HH:MI:SS.FF AM

说明

此参数与在参数a_format_version值为10c和a_format_dev_version值为s1的情况下有效

group_concat_max_len

参数说明：搭配函数GROUP_CONCAT使用，限制其返回值长度，超长截断。

该参数属于USERSET类型参数，请参考表**表14-1**中对应设置方法进行设置。

取值范围：0-9223372036854775807

默认值：1024

说明

目前能返回的最大长度是1073741823，超出此长度后会有out of memory的报错。

max_function_args

参数说明：函数参数最大个数。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：整型

默认值：8192

transform_null_equals

参数说明：控制表达式expr = NULL（或NULL = expr）当做expr IS NULL处理。如果expr得出NULL值则返回真，否则返回假。

- 正确的SQL标准兼容的expr = NULL总是返回NULL（未知）。
- Microsoft Access里的过滤表单生成的查询使用expr = NULL来测试空值。打开这个选项，可以使用该接口来访问数据库。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示控制表达式expr = NULL（或NULL = expr）当做expr IS NULL处理。
- off表示不控制，即expr = NULL总是返回NULL（未知）。

默认值：off

说明

新用户经常在涉及NULL的表达式上语义混淆，故默认值设为off。

support_extended_features

参数说明：控制是否支持数据库的扩展特性。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示支持数据库的扩展特性。
- off表示不支持数据库的扩展特性。

默认值：off

enable_extension

参数说明：控制是否支持创建数据库扩展插件。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示支持创建数据库扩展插件。
- off表示不支持创建数据库扩展插件。

默认值：off

须知

扩展功能为内部使用功能，不建议用户使用。

sql_compatibility

参数说明：控制数据库的SQL语法和语句行为同哪一个主流数据库兼容。该参数属于INTERNAL类型参数，用户无法修改，只能查看。

取值范围：枚举型

- A表示同O数据库兼容。
- B表示同MY数据库兼容。
- C表示同TD数据库兼容。
- PG表示同POSTGRES数据库兼容。

默认值：A

须知

- 该参数只能在执行CREATE DATABASE命令（详见《开发指南》的“SQL参考 > SQL语法 > CREATE DATABASE”章节）创建数据库的时候设置。
- 在数据库中，该参数只能是确定的一个值，要么始终设置为A，要么始终设置为B，请勿任意改动，否则会导致数据库行为不一致。

b_format_behavior_compat_options

参数说明：数据库B模式兼容性行为配置项，该参数的值由若干个配置项用逗号隔开构成。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

默认值：""

说明

- 当前只支持**表1 兼容性B模式配置项**。
- 配置多个兼容性配置项时，相邻配置项用逗号隔开，例如：set b_format_behavior_compat_options='enable_set_variables,set_session_transaction';

表 14-9 兼容性 B 模式配置项

兼容性配置项	兼容性行为控制
enable_set_variables	set语法增强控制开关。 • 不设置此配置时，不支持set自定义变量、set [global | session]语法。 • 设置此配置时，支持B兼容模式下使用上述语法，比如 *set @v1 = 1;* 。
set_session_transaction	set session transaction控制开关。 • 不设置此配置时，set session transaction等效于set local transaction。 • 设置此配置时，支持B兼容模式下使用上述语法，修改当前会话事务特性。
enable_modify_column	ALTER TABLE MODIFY语义控制开关。 • 不设置此配置时，“ALTER TABLE table_name MODIFY column_name data_type;”只修改列的数据类型。 • 设置此配置时，“ALTER TABLE table_name MODIFY column_name data_type;”修改整个列定义。

兼容性配置项	兼容性行为控制
default_collation	默认字符序前向兼容开关。 • 不设置此配置时，在未显式指定字符类型字段的字符集或字符序且表级字符序也为空时，字段为default字符序。 • 设置此配置时，字符类型字段的字符序当表级字符序不为空时继承表级字符序，为空时设置为数据库编码对应的默认字符序。

behavior_compat_options

参数说明：数据库兼容性行为配置项，该参数的值由若干个配置项用逗号隔开构成。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

默认值：""

说明

- 当前只支持**表14-10**。
- 配置多个兼容性配置项时，相邻配置项用逗号隔开，例如：set behavior_compat_options='end_month_calculate,display_leading_zero';

表 14-10 兼容性配置项

兼容性配置项	兼容性行为控制
display_leading_zero	浮点数显示配置项。控制数值类型中char、character、nchar、varchar、character varying、varchar2、nvarchar2、text、clob等所有字符串类型和float4、float8、numeric等任意精度类型的小数点前零显示。并且length计算数字长度同步显示。 • 不设置此配置项时，对于-1~0和0~1之间的小数，不显示小数点前的0。比如： gaussdb=# select 0.1231243 as a, 0.1231243::numeric as b,0.1231243::integer(10,3) as c, length(0.1242343) as d; a | b | c | d ----------+----------+------+--- .1231243 | .1231243 | .123 | 8 (1 row) • 设置此配置项时，对于-1~0和0~1之间的小数，显示小数点前的0。比如： gaussdb=# select 0.1231243 as a, 0.1231243::numeric as b,0.1231243::integer(10,3) as c, length(0.1242343) as d; a | b | c | d ----------+----------+-------+--- 0.1231243 | 0.1231243 | 0.123 | 9 (1 row)

兼容性配置项	兼容性行为控制

end_month_calculate	add_months函数计算逻辑配置项。
	假定函数add_months的两个参数分别为param1和param2，param1的月份和param2的和为月份result。
	● 不设置此配置项时，如果param1的日期（Day字段）为月末，并且param1的日期（Day字段）比result月份的月末日期小，计算结果中的日期字段（Day字段）和param1的日期字段保持一致。比如，
	`gaussdb=# select add_months('2018-02-28',3) from sys_dummy;`
	`add_months`
	`---------------------`
	`2018-05-28 00:00:00`
	`(1 row)`
	● 设置此配置项时，如果param1的日期（Day字段）为月末，并且param1的日期（Day字段）比result月份的月末日期比小，计算结果中的日期字段（Day字段）和result的月末日期保持一致。比如，
	`gaussdb=# select add_months('2018-02-28',3) from sys_dummy;`
	`add_months`
	`---------------------`
	`2018-05-31 00:00:00`
	`(1 row)`

compat_analyze_sample	analyze采样行为配置项。 设置此配置项时，会优化analyze的采样行为，主要体现在analyze时全局采样会更精确的控制在3万条左右，更好地控制analyze时DBnode端的内存消耗，保证analyze性能的稳定性。
bind_schema_tablespace	绑定模式与同名表空间配置项。 如果存在与模式名sche_name相同的表空间名，那么如果设置search_path为sche_name， default_tablespace也会同步切换到sche_name。
bind_procedure_searchpath	未指定模式名的存储过程中的数据库对象的搜索路径配置项。 在存储过程中如果不显示指定模式名，会优先在存储过程所属的模式下搜索。 如果找不到，则有两种情况： ● 若不设置此参数，报错退出。 ● 若设置此参数，按照search_path中指定的顺序继续搜索。如果还是找不到，报错退出。

correct_to_number	控制to_number()结果兼容性的配置项。
	若不设置此配置项，则to_number()函数结果默认与A数据库保持一致。
	`gaussdb=# select '' AS to_number_14, to_number('34,50','999,99');`
	`ERROR:  invalid data.`
	`CONTEXT:  referenced column: to_number`
	若设置此配置项，则to_number()函数结果与pg11保持一致。
	`gaussdb=# select '' AS to_number_14, to_number('34,50','999,99');`
	` to_number_14 | to_number`
	`---------------+-----------`
	`               |      3450`
	`(1 row)`

兼容性配置项	兼容性行为控制
unbind_divide_bound	控制对整数除法的结果进行范围校验。 若不设置此配置项，则会对除法结果做范围校验，例如，INT_MIN/(-1)会因为输出结果大于INT_MAX而报越界错误。 gaussdb=# select (-2147483648)::int4 / (-1)::int4; ERROR: integer out of range 若设置此配置项，则不需要对除法结果做范围校验，例如，INT_MIN/(-1)可以得到输出结果为INT_MAX+1。 gaussdb=# select (-2147483648)::int4 / (-1)::int4; ?column? ------------ 2147483648 (1 row)

convert_string_digit_to_numeric	控制是否将表中以字符串形式表示的numeric常量和数字类型做比较时统一都转换为numeric类型再进行比较。
	gaussdb=# create table test1 (c1 int, c2 varchar);
	gaussdb=# insert into test1 values (2, '1.1');
	gaussdb=# set behavior_compat_options='';
	gaussdb=# select * from test1 where c2 > 1;
	ERROR: invalid input syntax for type bigint: "1.1"
	
	gaussdb=# set behavior_compat_options='convert_string_digit_to_numeric';
	gaussdb=# select * from test1 where c2 > 1;
	 c1 | c2
	----+-----
	  2 | 1.1
	(1 row)

return_null_string	控制函数lpad()和rpad()结果为空字符串''的显示配置项。 ● 不设置此配置项时，空字符串显示为NULL。 gaussdb=# select length(lpad('123',0,'*')) from sys_dummy; length -------- (1 row) ● 设置此配置项时，空字符串显示为''。 gaussdb=# select length(lpad('123',0,'*')) from sys_dummy; length -------- 0 (1 row)

兼容性配置项	兼容性行为控制

compat_concat_variadic	控制函数concat()和concat_ws()对variadic类型结果兼容性的配置项。由于B数据库无variadic类型，所以该选项对B数据库无影响。
	若不设置此配置项，当concat函数参数为variadic类型时，默认A数据库和C数据库兼容模式下结果相同，且与A数据库保持一致。
	`gaussdb=# select concat(variadic NULL::int[]) is NULL;`
	` ?column?`
	`----------`
	` t`
	`(1 row)`
	若设置此配置项，当concat函数参数为variadic类型时，保留A数据库和C数据库兼容模式下不同的结果形式。
	`--A数据库下：`
	`gaussdb=# select concat(variadic NULL::int[]) is NULL;`
	` ?column?`
	`----------`
	` t`
	`(1 row)`
	`--C数据库下：`
	`gaussdb=# select concat(variadic NULL::int[]) is NULL;`
	` ?column?`
	`----------`
	` f`
	`(1 row)`

merge_update_multi	控制在使用MERGE INTO ... WHEN MATCHED THEN UPDATE（参考《开发指南》的“SQL参考 > SQL语法 > MERGE INTO”章节）和INSERT ... ON DUPLICATE KEY UPDATE（参考《开发指南》的“SQL参考 > SQL语法 > INSERT”章节）时，当目标表中一条目标数据与多条源数据冲突时UPDATE行为。 若设置此配置项，当存在上述场景时，该冲突行将会多次执行UPDATE；否则（默认）报错，即MERGE或INSERT操作失败。
plstmt_implicit_savepoint	控制存储过程中更新语句的执行是否拥有独立的子事务。 若设置此配置项，存储过程中每条更新语句前开启隐式保存点，EXCEPTION块中默认回退到最近的保存点，从而保证只回退失败语句的修改。该选项是为了兼容O数据库的EXCEPTION行为。

兼容性配置项	兼容性行为控制

hide_tailing_zero	numeric显示配置项。不设置此项时，numeric按照指定精度显示。设置此项时，所有输出numeric的场景均隐藏小数点后的末尾0，包括显示指定format精度情况。
	gaussdb=# set behavior_compat_options='hide_tailing_zero';
	gaussdb=# select cast(123.123 as numeric(15,10)) as a, to_char(cast(123.123 as numeric(15,10)), '999D999999');
	  a | to_char
	---------+----------
	 123.123 | 123.123
	(1 row)
	gaussdb=# set behavior_compat_options='';
	gaussdb=# select cast(123.123 as numeric(15,10)) as a, to_char(cast(123.123 as numeric(15,10)), '999D999999');
	  a | to_char
	----------------+-------------
	 123.1230000000 | 123.123000
	(1 row)

rownum_type_compat	控制ROWNUM的类型，ROWNUM默认类型为BIGINT，设置此参数后，ROWNUM类型变更为NUMERIC类型。
	gaussdb=# set behavior_compat_options='';
	gaussdb=# create table tb_test(c1 int,c2 varchar2,c3 varchar2);
	gaussdb=# insert into tb_test values(1,'a','b');
	gaussdb=# create or replace view v_test as select rownum from tb_test;
	gaussdb=# \d+ v_test
	  View "public.v_test"
	 Column | Type | Modifiers | Storage | Description
	--------+---------+-----------+---------+-------------
	 rownum | bigint | | plain |
	View definition:
	 SELECT ROWNUM AS "rownum"
	  FROM tb_test;
	
	gaussdb=# set behavior_compat_options = 'rownum_type_compat';
	gaussdb=# create or replace view v_test1 as select rownum from tb_test;
	gaussdb=# \d+ v_test1
	  View "public.v_test1"
	 Column | Type | Modifiers | Storage | Description
	--------+---------+-----------+---------+-------------
	 rownum | numeric | | main |
	View definition:
	 SELECT ROWNUM AS "rownum"
	  FROM tb_test;

兼容性配置项	兼容性行为控制
aformat_null_test	控制rowtype类型判空逻辑。 设置此项时，对于rowtype is not null的判断逻辑为当一行数据有一列不为空的时候返回true；不设置此项时，对于rowtype is not null的判断逻辑为当一行数据所有列不为空的时候返回true。该参数不影响rowtype is null的判断。 gaussdb=# set behavior_compat_options='aformat_null_test'; gaussdb=# select r, r is null as isnull, r is not null as isnotnull from (values (1,row(1,2)), (1,row(null,null)), (1,null), (null,row(1,2)), (null,row(null,null)), (null,null)) r(a,b); r | isnull | isnotnull ----------+--------+----------- (1,"(1,2)") | f | t (1,"(,)") | f | t (1,) | f | t (,"(1,2)") | f | t (,"(,)") | f | t (,) | t | f (6 rows) gaussdb=# set behavior_compat_options=''; gaussdb=# select r, r is null as isnull, r is not null as isnotnull from (values (1,row(1,2)), (1,row(null,null)), (1,null), (null,row(1,2)), (null,row(null,null)), (null,null)) r(a,b); r | isnull | isnotnull ----------+--------+----------- (1,"(1,2)") | f | t (1,"(,)") | f | t (1,) | f | f (,"(1,2)") | f | f (,"(,)") | f | f (,) | t | f (6 rows)
aformat_regexp_match	控制正则表达式函数的匹配行为。 设置此项，且sql_compatibility参数的值为A或B时，正则表达式的 flags 参数支持的选项含义有变更： 1. 默认不能匹配 '\n' 字符。 2. flags 中包含n选项时， . 能够匹配 '\n' 字符。 3. regexp_replace(source, pattern replacement) 函数替换所有匹配的子串。 4. regexp_replace(source, pattern, replacement, flags) 在flags值为'' 或者null时，返回值为null。 否则，正则表达式的 flags 参数支持的选项含义： 1. 默认能匹配 '\n' 字符。 2. flags 中的 n 选项表示按照多行模式匹配。 3. regexp_replace(source, pattern replacement) 函数仅替换第一个匹配到的子串。 4. regexp_replace(source, pattern, replacement, flags) 在flags值为'' 或者null时，返回值为替换后的字符串。
compat_cursor	控制隐式游标状态兼容行为。设置此项，且兼容O，隐式游标状态（SQL%FOUND、SQL%NOTFOUND、SQL%ISOPNE、SQL%ROWCOUNT）由原先的仅在当前执行的函数有效，拓展到包括本函数调用的子函数有效。

兼容性配置项	兼容性行为控制
proc_outparam_override	控制存储过程出参的重载行为，打开该参数后，对于存储过程只有out出参部分不同的情况下，也可以正常创建和调用。目前只有gsql与jdbc连接数据库时可以使用该参数，对于其他工具打开该参数连接数据库时无法正常调用带有out的存储过程。 支持带有out出参的函数返回record类型，且out出参正常赋值。
proc_implicit_for_loop_variable	控制存储过程中FOR_LOOP查询语句行为。设置此项时，在FOR rec IN query LOOP语句中，若rec已经定义，不会复用已经定义的rec变量，而且重新建立一个新的变量。否则，会复用已经定义的rec变量，不会建立新的变量。
allow_procedure_compile_check	控制存储过程中select语句和open cursor语句的编译检查。设置此项时，在存储过程中执行select语句、open cursor for语句、cursor%rowtype语句、for rec in语句时，若查询的表不存在，则无法创建存储过程，不支持trigger函数的编译检查，若查询的表存在，则成功创建存储过程。 注意：创建密态函数时，需要将allow_procedure_compile_check关闭。

char_coerce_compat	控制char(n)类型向其它变长字符串类型转换时的行为。不设置该参数时，char(n)类型转换其它变长字符串类型时会省略尾部的空格，设置该参数时，转换时不再省略尾部的空格，并且在转换时如果char(n)类型的长度超过其它变长字符串类型时将会报错。该参数仅在sql_compatibility参数的值为A时生效，并且开启该参数后无论是隐式转换、显式转换还是通过调用text(bpchar)函数转换类型都不再省略尾部空格。
	gaussdb=# set behavior_compat_options='';
	gaussdb=# create table tab_1(col1 varchar(3));
	gaussdb=# create table tab_2(col2 char(3));
	gaussdb=# insert into tab_2 values(' ');
	gaussdb=# insert into tab_1 select col2 from tab_2;
	gaussdb=# select * from tab_1 where col1 is null;
	col1
	------ 	
	
	(1 row)
	gaussdb=# select * from tab_1 where col1=' ';
	col1
	------ 	
	(0 rows)
	gaussdb=# delete from tab_1;
	gaussdb=# set behavior_compat_options = 'char_coerce_compat';
	gaussdb=# insert into tab_1 select col2 from tab_2;
	gaussdb=# select * from tab_1 where col1 is null;
	col1
	------ 	
	(0 rows)
	gaussdb=# select * from tab_1 where col1=' ';
	col1
	------ 	
	
	(1 row)

兼容性配置项	兼容性行为控制
truncate_numeric_tail_zero	numeric显示配置项。不设置此项时，numeric按照默认精度显示。设置此项时，除去to_char(numeric, format)这种显示设置精度的情况，所有输出numeric的场景均会隐藏小数点后的末尾0。例如： gaussdb=# set behavior_compat_options='truncate_numeric_tail_zero'; gaussdb=# select cast(123.123 as numeric(15,10)) as a, to_char(cast(123.123 as numeric(15,10)), '999D999999'); a | to_char ---------+------------- 123.123 | 123.123000 (1 row) gaussdb=# set behavior_compat_options=''; gaussdb=# select cast(123.123 as numeric(15,10)) as a, to_char(cast(123.123 as numeric(15,10)), '999D999999'); a | to_char ----------------+------------- 123.1230000000 | 123.123000 (1 row)
plsql_security_definer	开启此参数后，创建存储过程时默认为定义者权限。
plpgsql_dependency	开启此参数后，创建函数，存储过程，包支持未定义的对象。可以新建成功。可以在GS_DEPENDENCIES和GS_DEPENDENCIES_OBJ查询对应的依赖关系。 开启此参数后，创建PLSQL对象时，会主动维护依赖于该PLSQL对象的OID，不再需要用户手动更新。 以下情况支持建立依赖： 1. 函数头中依赖类型和参数默认值位置出现函数。 2. 包，函数中类型、变量依赖类型。 3. 变量声明、变量赋值依赖其他package中变量。 4. 函数体中，函数调用、赋值语句右值表达式中调用函数A，函数A出入参包含函数B时，函数B不建立依赖。例如functionA(functionB())。仅建立和函数A的依赖。 5. 视图中依赖函数。 以下情况不支持不建立依赖： 1. Schema中类型依赖其他类型。 2. SQL语句中对函数，变量，表，视图的依赖。例如 select id into var1 from table1 join view1 on table1.id = pkg1.var1; table1,view1,pkg1均不记录依赖。 3. 视图中对函数，变量，表，视图的依赖。 注意: 1. 在并发创建PLSQL对象时，如果需要维护的对象间存在竞争关系，可能会造成死锁。 2. 当修改对象在gs_dependencies和gs_dependencies_obj中存在时，不允许进行rename操作。 3. 当函数，存储过程，包依赖同义词时，该同义词必须提前建立，不支持主动维护oid。

兼容性配置项	兼容性行为控制
disable_rewrite_nesttable	开启此参数后，针对pg_type表中tableof类型相关字段的重写会被禁用。即读取pg_type表时，会显示tableof类型实际存储的值。
skip_insert_gs_source	开启此参数后，创建PLSQL对象时不再插入dbe_pldeveloper.gs_source表中。
disable_emptystr2null	开启此参数后，关闭所有字符类型默认将空串转换为null功能。包括text、clob、blob、raw、bytea、varchar、nvarchar2、bpchar、char、name、byteawithoutorderwithqualcol、byteawithoutordercol类型。该参数为逃生参数，非必要用户不要自行设置。
select_into_return_null	此参数仅在PG兼容模式下生效，开启此参数后，存储过程语句SELECT *select_expressions* INTO [STRICT] *target* FROM ... 允许在不指定STRICT并且查询结果为空时给变量赋NULL值。

兼容性配置项	兼容性行为控制
proc_uncheck_default_param	函数调用时不检查默认参数省略情况配置项。 ● 不设置此配置项时，调用带有默认参数的函数时，入参从左往右排入函数，如果有非默认参数的入参缺失则会报错。比如， gaussdb=# create or replace function test(f1 int, f2 int default 20, f3 int, f4 int default 40, f5 int default 50) return int gaussdb-# as gaussdb$# begin gaussdb$# raise info 'f1:%',f1; gaussdb$# raise info 'f2:%',f2; gaussdb$# raise info 'f3:%',f3; gaussdb$# raise info 'f4:%',f4; gaussdb$# raise info 'f5:%',f5; gaussdb$# return 1; gaussdb$# end; gaussdb$# / CREATE FUNCTION gaussdb=# select test(1,2); ERROR: function test(integer, integer) does not exist LINE 1: select test(1,2); ^ HINT: No function matches the given name and argument types. You might need to add explicit type casts. CONTEXT: referenced column: test ● 设置此配置项时，调用带有默认参数的函数时，入参从左往右排入函数，允许缺省默认参数个入参，如果有非默认参数的入参缺失，则会用错位的默认值填充该参数。比如， gaussdb=# create or replace function test(f1 int, f2 int default 20, f3 int, f4 int default 40, f5 int default 50) return int gaussdb-# as gaussdb$# begin gaussdb$# raise info 'f1:%',f1; gaussdb$# raise info 'f2:%',f2; gaussdb$# raise info 'f3:%',f3; gaussdb$# raise info 'f4:%',f4; gaussdb$# raise info 'f5:%',f5; gaussdb$# return 1; gaussdb$# end; gaussdb$# / CREATE FUNCTION gaussdb=# select test(1,2); INFO: f1:1 CONTEXT: referenced column: test INFO: f2:2 CONTEXT: referenced column: test INFO: f3:20 CONTEXT: referenced column: test INFO: f4:40 CONTEXT: referenced column: test INFO: f5:50 CONTEXT: referenced column: test test ------ 1 (1 row) 如上，f3被错误的默认值填充。 **警告** 该场景下，非默认参数会被错位的默认值填充。

兼容性配置项	兼容性行为控制
dynamic_sql_compat	开启此参数后，动态语句不会将模板SQL中的同名模板参数视为同一个变量，而是按照顺序依次匹配using子句中的变量值。 **注意** • 动态语句执行匿名块语句时调用存储过程的场景，只针对IN参数进行矫正，如果需要对OUT参数进行检查需要设置proc_outparam_override选项。 • 动态语句执行匿名块语句时调用存储过程的场景，开启参数后不会对存储过程中参数的IN/OUT属性和using子句中的IN/OUT属性进行检查。
dynamic_sql_check	开启此参数后，动态语句模板SQL中的不同模板参数个数与using子句中的变量个数不同时将会在动态语句执行期间报错。 **注意** • 不推荐同时使用dynamic_sql_check选项和dynamic_sql_compat选项，开启dynamic_sql_compat选项后dynamic_sql_check选项将不再生效。 • 动态语句执行匿名块语句时调用存储过程的场景，只针对IN参数进行检查，如果需要对OUT参数进行检查需要设置proc_outparam_override选项。 • 动态语句执行匿名块语句时调用存储过程的场景，开启参数后不会对存储过程中参数的IN/OUT属性和using子句中的IN/OUT属性进行检查。

兼容性配置项	兼容性行为控制

enable_funcname_with_argsname	开启参数后，使用select调用函数时投影别名显示完整函数。
	● 不设置此配置项时，使用select调用函数时投影别名仅显示函数名。比如：
	gaussdb=# SELECT power(2,3);
	power
	------- 	
	8
	(1 row)
	
	gaussdb=# SELECT count(*) FROM db_ind_columns;
	count
	------- 	
	611
	(1 row)
	
	gaussdb=# SELECT count(index_name) FROM db_ind_columns;
	count
	------- 	
	611
	(1 row)
	
	gaussdb=# SELECT left('abcde', 2);
	left
	------ 	
	ab
	(1 row)
	
	gaussdb=# SELECT pg_client_encoding();
	pg_client_encoding
	-------------------- 	
	UTF8
	(1 row)
	● 设置此配置项时，使用select调用函数时投影别名显示完整函数。比如：
	gaussdb=# SET behavior_compat_options =
	'enable_funcname_with_argsname';
	SET
	gaussdb=# SELECT power(2,3);
	power(2,3)
	------------ 	
	8
	(1 row)
	
	gaussdb=# SELECT count(*) FROM db_ind_columns;
	count(*)
	---------- 	
	611
	(1 row)
	
	gaussdb=# SELECT count(index_name) FROM db_ind_columns;
	count(index_name)
	------------------- 	
	611
	(1 row)
	
	gaussdb=# SELECT left('abcde', 2);
	left('abcde',2)
	----------------- 	
	ab
	(1 row)
	
	gaussdb=# SELECT pg_client_encoding();
	pg_client_encoding()

兼容性配置项	兼容性行为控制
	---------- UTF8 (1 row) **注意** • 目前仅支持func_name(args_list)、func_name()、func_name(*)三种形式投影别名显示完整函数，且参数args类型仅支持字符类型、数值类型、列名、函数。函数名支持带有schema、包名。不支持参数带有其他子句（如order by子句）、不支持参数为表达式、仅支持参数带有DISTINCT关键字，不支持带有其他关键字时显示完整函数。 • 一些特殊函数不支持投影别名显示完整函数，包括函数：COLLATION FOR、CURRENT_DATE、CURRENT_TIME、CURRENT_TIMESTAMP、DBTIMEZONE、LOCALTIME、LOCALTIMESTAMP、SYSDATE、SESSIONTIMEZONE、ROWNUM、CURRENT_ROLE、CURRENT_USER、SESSION_USER、USER、CURRENT_CATALOG、CURRENT_SCHEMA、CAST、EXTRACT、TIMESTAMPDIFF、OVERLAY、POSITION、SUBSTRING、TREAT、TRIM、NULLIF、NVL、NVL2、COALESCE、GREATEST、LEAST、LNNVL、REGEXP_LIKE以及XML函数。 • 一些安全加解密函数、脱敏函数，投影别名显示完整函数可能存在安全问题，在这里就还是仅显示函数名。包括函数：gs_encrypt_aes128,gs_decrypt_aes128,gs_encrypt,gs_decrypt,aes_encrypt,aes_decrypt,pg_create_physical_replication_slot_extern,dblink_connect,creditcardmasking",basicemailmasking",fullemailmasking,alldigitsmasking,shufflemasking,randommasking,regexpmasking,gs_digest。 • 不支持=>传参方式调用函数时投影别名显示完整函数，不支持投影别名显示""，如：select "power"(2,3)。 • 为了让投影别名显示完整函数，本功能不受去除末尾0等参数影响。
allow_rownum_alias	开启此参数后，将允许ROWNUM在SQL语句中通过AS语法用作列的别名，ROWNUM将作为普通标识符使用，不再能够作为伪列使用。 **警告** 不建议在执行业务期间变更该参数的状态。开启参数的状态下，在数据库中用ROWNUM作为名称创建的数据库对象（如表名，列名，数据库名等）只能在开启参数的情况下使用，否则将会产生歧义，行为不可预期。关闭参数的状态下，在数据库中ROWNUM作为伪列使用的行为将在开启参数后失效，行为不可预期。
current_sysdate	开启此参数后，执行sysdate时，会获取当前操作系统时间。 gaussdb=# set behavior_compat_options='current_sysdate'; SET gaussdb=# select sysdate; current_sysdate --------------------- 2023-06-20 20:15:27 (1 row)

plsql_compile_check_options

参数说明：数据库兼容性行为配置项，该参数的值由若干个配置项用逗号隔开构成。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

默认值：""

说明

- 当前只支持**表14-10**。
- 配置多个兼容性配置项时，相邻配置项用逗号隔开，例如：set plsql_compile_check_options='for_loop,outparam';

表 14-11 兼容性配置项

兼容性配置项	兼容性行为控制
for_loop	控制存储过程中FOR_LOOP查询语句行为设置此项时，在FOR rec IN query LOOP语句中，若rec已经定义，不会复用已经定义的rec变量，而且重新建立一个新的变量。否则，会复用已经定义的rec变量，不会建立新的变量。(与proc_implicit_for_loop_variable相同，后续进行收编)
outparam	out重载条件下，有重载函数；将对out出参常量进行检查，禁止out出参为常量报错。
plsql_expression_check	编译检查模式开启下，打开当前参数，会额外检查function out出参类型，当out出参无法赋值时，将会报warning提升。

a_format_version

参数说明：数据库平台兼容性行为配置项，该参数的值为字符串枚举值。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

默认值：""

说明

- 当前只支持**表14-10**。
- 兼容性配置项时设置字符串，例如：set a_format_version='10c';

表 14-12 兼容性配置项

兼容性配置项	兼容性行为控制
10c	A平台兼容版本

a_format_dev_version

参数说明：数据库平台迭代小版本兼容性行为配置项，该参数的值为字符串枚举值。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

默认值：""

说明

- 当前只支持**表14-13**。
- 兼容性配置项时设置字符串，例如：set a_format_dev_version='s1';

表 14-13 兼容性配置项

兼容性配置项	兼容性行为控制
s1	• A平台兼容迭代小版本，影响函数（TRUNC(date, fmt),ROUND(date, fmt),NVL2,LPAD,RPAD,ADD_MONTHS,MONTHS_BETWEEN,REGEXP_REPLACE,REGEXP_COUNT,TREAT,EMPTY_CLOB,INSTRB,trunc(number),greatest,least,mod,round(number),cast,to_date,to_timestamp,chr,rtrim,translate,to_char,to_number，to_timestamp_tz）。 • 数据类型转换：小数字符串转换成整数类型（int1/int2/int4/int8/int16）时进行四舍五入。 • 数据类型转换：支持timestamp with time zone到timestamp without time zone的隐式转换。
s2	• A平台兼容迭代小版本，影响函数（dump, to_single_byte, to_multi_byte, nls_upper, nls_lower, initcap, ascii2, asciistr, unistr, vsize, cosh, remainder, sinh, tanh, nanvl, current_date, current_timestamp, dbtimezone, numtodsinterval, numtoyminterval, new_time, sessiontimezone, sys_extract_utc, tz_offset, to_binary_double, to_binary_float, to_dsinterval, to_yminterval, lnnvl, ora_hash, rawtohex2, bit2coding, bit4coding）。 • 兼容配置项为s1时的所有行为。

plpgsql.variable_conflict

参数说明：设置同名的存储过程变量和表的列的使用优先级。

该参数属于USERSET类型参数，仅支持**表14-1**中对应设置方法3进行设置。

取值范围：字符串

- error表示遇到存储过程变量和表的列名同名则编译报错。
- use_variable表示存储过程变量和表的列名同名则优先使用变量。
- use_column表示存储过程变量和表的列名同名则优先使用列名。

默认值：error

td_compatible_truncation

参数说明：控制是否开启与Teradata数据库相应兼容的特征。该参数在用户连接上与TD兼容的数据库时，可以将参数设置成为on（即超长字符串自动截断功能启用），该功能启用后，在后续的insert语句中，对目标表中char和varchar类型的列插入超长字符串时，会按照目标表中相应列定义的最大长度对超长字符串进行自动截断。保证数据都能插入目标表中，而不是报错。

说明

超长字符串自动截断功能不适用于insert语句包含外表的场景。

如果向字符集为字节类型编码（SQL_ASCII、LATIN1等）的数据库中插入多字节字符数据（如汉字等），且字符数据跨越截断位置，这种情况下，按照字节长度自动截断，自动截断后会在尾部产生非预期结果。如果用户有对于截断结果正确性的要求，建议用户采用UTF8等能够按照字符截断的输入字符集作为数据库的编码集。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示启动超长字符串自动截断功能。
- off表示停止超长字符串自动截断功能。

默认值：off

uppercase_attribute_name

参数说明：设置列名以大写形式返回给客户端。该参数仅限于ORA兼容模式和集中式环境下使用。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启列名以大写形式返回给客户端。
- off表示关闭列名以大写形式返回给客户端。

默认值：off

lastval_supported

参数说明：控制是否可以使用lastval函数。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示支持lastval函数，同时nextval函数不支持下推。
- off表示不支持lastval函数，同时nextval函数可以下推。

默认值：off

a_format_copy_version

参数说明：数据库平台迭代小版本兼容性行为配置项，该参数的值为字符串枚举值。

该参数属于USERSET类型参数，请参考表14-1中对应设置方法进行设置。使用gs_loader导入工具新特性时，需要设置对应的值。

取值范围：字符串

默认值：''

说明

- 当前只支持表14-14.。
- 兼容性配置项时设置字符串，具体参见《工具参考》的“客户端工具 > gs_loader”章节，使用方式为通过guc_param设置a_format_copy_version参数。

表 14-14 兼容性配置项

兼容性配置项	兼容性行为控制
s1	• 支持gs_loader指定数据类型（CHAR[(length)]/INTEGER external[(length)]/FLOAT external[(length)]/DECIMAL external[(length)]/TIMESTAMP/DATE/DATE EXTERNAL/INTEGER/SMALLINT/RAW[(length)]）导入数据。 • 支持gs_loader列表达式使用场景扩展。

14.3.16 容错性

当数据库系统发生错误时，以下参数控制服务器处理错误的方式。

exit_on_error

参数说明：打开该开关，ERROR级别报错会升级为PANIC报错，从而可以产生core堆栈。主要用于问题定位和业务测试。

该参数属于USERSET类型参数，请参考表14-2中对应设置方法进行设置。

取值范围：布尔型

- on表示ERROR级别报错会升级为PANIC报错。
- off表示不会对ERROR级别报错进行升级。

默认值：off

restart_after_crash

参数说明：设置为on，后端进程崩溃时，GaussDB将自动重新初始化此后端进程。

该参数属于SIGHUP类型参数，请参考表14-2中对应设置方法进行设置。

取值范围：布尔型

- on表示能够最大限度地提高数据库的可用性。
  在某些情况（比如当采用管理工具（例如xCAT）管理GaussDB时），能够最大限度地提高数据库的可用性。
- off表示能够使得管理工具在后端进程崩溃时获取控制权并采取适当的措施进行处理。

默认值：on

omit_encoding_error

参数说明：设置为on，数据库的客户端字符集编码为UTF-8时，出现的字符编码转换错误将打印在日志中，有转换错误的被转换字符会被忽略，以"?"代替。

该参数属于USERSET类型参数，请参考**表14-2**中对应设置方法进行设置。

取值范围：布尔型

- on表示有转换错误的字符将被忽略，以"?"代替，打印错误信息到日志中。
- off表示有转换错误的字符不能被转换，打印错误信息到终端。

默认值：off

cn_send_buffer_size

参数说明：指定数据库主节点发送数据缓存区的大小。

该参数属于POSTMASTER类型参数，请参考**表14-2**中对应设置方法进行设置。

取值范围：整型，8~128， 单位为KB。

默认值：8KB

data_sync_retry

参数说明：控制当fsync到磁盘失败后是否继续运行数据库。由于在某些操作系统的场景下，fsync失败后重试阶段即使再次fsync失败也不会报错，从而导致数据丢失。

该参数属于POSTMASTER类型参数，请参考**表14-2**中对应设置方法进行设置。

取值范围：布尔型

- on表示当fsync同步到磁盘失败后采取重试机制，数据库继续运行。
- off表示当fsync同步到磁盘失败后直接报panic，停止数据库。

默认值：off

remote_read_mode

参数说明：远程读功能开关。读取主机上的页面失败时可以从备机上读取对应的页面。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：枚举类型

- off表示关闭远程读功能。
- non_authentication表示开启远程读功能，但不进行证书认证。
- authentication表示开启远程读功能，但要进行证书认证。

默认值：authentication

14.3.17 连接池参数

当使用连接池访问数据库时，在系统运行过程中，数据库连接是被当作对象存储在内存中的，当用户需要访问数据库时，并非建立一个新的连接，而是从连接池中取出一个已建立的空闲连接来使用。用户使用完毕后，数据库并非将连接关闭，而是将连接放回连接池中，以供下一个请求访问使用。

cache_connection

参数说明：是否回收连接池的连接。

该参数属于SIGHUP类型参数，请参考**表14-2**中对应设置方法进行设置。

取值范围：布尔型

- on表示回收连接池的连接。
- off表示不回收连接池的连接。

默认值：on

14.3.18 事务

介绍数据库事务隔离、事务只读、最大prepared事务数、维护模式目的参数设置及取值范围等内容。

transaction_isolation

参数说明：设置当前事务的隔离级别。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，只识别以下字符串，大小写空格敏感：

- serializable：GaussDB中等价于REPEATABLE READ。
- read committed：只能读取已提交的事务的数据（缺省），不能读取到未提交的数据。
- repeatable read：仅能读取事务开始之前提交的数据，不能读取未提交的数据以及在事务执行期间由其它并发事务提交的修改。
- default：设置为default_transaction_isolation所设隔离级别。

默认值：read committed

transaction_read_only

参数说明：设置当前事务是只读事务。

该参数在数据库恢复过程中或者在备机里，固定为on；否则，固定为default_transaction_read_only的值。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示设置当前事务为只读事务。
- off表示该事务可以是非只读事务。

默认值：off

xc_maintenance_mode

参数说明：设置系统进入维护模式。

该参数属于SUSET类型参数，仅支持**表14-1**中的方式三进行设置。

取值范围：布尔型

- on表示该功能启用。
- off表示该功能被禁用。

须知

谨慎打开这个开关，避免引起数据库数据不一致。

默认值：off

allow_concurrent_tuple_update

参数说明：设置是否允许并发更新。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示该功能启用。
- off表示该功能被禁用。

默认值：on

transaction_deferrable

参数说明：指定是否允许一个只读串行事务延迟执行，使其不会执行失败。该参数设置为on时，当一个只读事务发现读取的元组正在被其他事务修改，则延迟该只读事务直到其他事务修改完成。该参数为预留参数，该版本不生效。与该参数类似的还有一个**default_transaction_deferrable**，设置它来指定一个事务是否允许延迟。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许执行。
- off表示不允许执行。

默认值：off

enable_show_any_tuples

参数说明：该参数只有在只读事务中可用，用于分析。当这个参数被置为on/true时，表中元组的所有版本都会可见。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on/true表示表中元组的所有版本都会可见。
- off/false表示表中元组的所有版本都不可见。

默认值：off

replication_type

参数说明：标记当前HA模式是单主机模式、主备从模式还是一主多备模式。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

该参数用户不能自己去设置参数值。

取值范围：0~2

- 2 表示单主机模式，此模式无法扩展备机。
- 1 表示使用一主多备模式，全场景覆盖，推荐使用。
- 0 表示主备从模式，目前此模式暂不支持。

默认值：1

pgxc_node_name

参数说明：指定节点名称。

该参数属于POSTMASTER类型参数，请参考**表14-2**进行设置。

在备机请求主机进行日志复制时，如果application_name参数没有被设置，那么pgxc_node_name参数会被用来作为备机在主机上的流复制槽名字。该流复制槽的命名方式为 "该参数值_备机ip_备机port"。其中，备机ip和备机port取自replconninfo参数中指定的备机ip和端口号。该流复制槽最大长度为61个字符，如果拼接后的字符串超过该长度，则会使用截断后的pgxc_node_name进行拼接，以保证流复制槽名字长度小于等于61个字符。

⚠ 注意

此参数修改后会导致连接数据库实例失败，不建议进行修改。

取值范围：字符串

默认值：当前节点名称

enable_defer_calculate_snapshot

参数说明：延迟计算快照的xmin和oldestxmin，执行1000个事务或者间隔1s才触发计算，设置为on时可以在高负载场景下减少计算快照的开销，但是会导致oldestxmin推进较慢，影响垃圾元组回收，设置为off时xmin和oldestxmin可以实时推进，但是会增加计算快照时的开销。

该参数属于SIGHUP类型参数，改请参考**表14-2**进行设置

取值范围：布尔型。

- on表示延迟计算快照xmin和oldestxmin。
- off表示实时计算快照xmin和oldestxmin。

默认值：on。

14.3.19 双数据库实例复制参数

RepOriginId

参数说明：该参数是一个会话级别的GUC参数，在双向逻辑复制的场景下，为避免数据循环复制，需要设置为一个非0的值。

该参数属于USERSET类型参数，请参考**表14-1**中方式三对应设置方法进行设置。

取值范围：整型，0~2147483647

默认值：0

stream_cluster_run_mode

参数说明：流式容灾双实例容灾场景，标识DN节点属于主实例还是备实例。单实例使用默认值主实例。

该参数属于POSTMASTER类型参数，请参考表**表14-1**中对应设置方法进行设置。

取值范围：枚举类型

- cluster_primary表示节点是主实例的节点。
- cluster_standby表示节点是备实例的节点。

默认值：cluster_primary

enable_roach_standby_cluster

参数说明：设置双数据库实例中备数据库实例的各个实例为只读模式，仅sysadmin用户可以访问。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示备数据库实例开启只读模式。
- off表示备数据库实例关闭只读模式。此情况下，备数据库实例可读可写。

默认值：off

hadr_process_type

参数说明：基于流式复制异地容灾解决方案或者同城双中心高可用方案的流程标识。

该参数属于SIGHUP类型参数，改请参考**表14-2**进行设置。

取值范围：枚举类型。

- none表示当前无流程。

- failover表示当前处于灾备数据库实例升主流程。
- switchover_promote表示主备数据库实例倒换流程中灾备数据库实例升主流程。
- switchover_demote表示主备数据库实例倒换流程中主数据库实例降为灾备数据库实例流程。
- dorado_failover表示dorado灾备数据库实例升主流程。
- dorado_switchover_demote表示dorado主备数据库实例倒换流程中主数据库实例降为灾备数据库实例流程。
- dorado_failover_abnormal表示dorado主数据库实例共享盘故障时，灾备数据库实例升主流程。

默认值：none

14.3.20 开发人员选项

allow_system_table_mods

参数说明：设置是否允许修改系统表的结构或系统自带模式名称。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许修改系统表的结构或系统自带模式名称。
- off表示不允许修改系统表的结构或系统自带模式名称。

默认值：off

注意

不建议修改该参数默认值，若设置为on，可能导致系统表损坏，甚至数据库无法启动。

allow_create_sysobject

参数说明：设置是否允许在系统模式下创建或修改函数、存储过程、同义词、聚合函数、操作符等对象。此处的系统模式指数据库初始后自带的模式，但不包含public模式。系统模式的oid通常小于16384。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许初始用户和系统管理员在系统模式下创建或修改函数、存储过程、同义词、聚合函数等对象，并允许初始用户在系统模式下创建操作符。sysadmin用户具有默认create or replace/alter/grant/revoke系统对象的权限。其他用户是否允许创建这些对象请参考对应模式的权限要求。
- off表示禁止所有用户在系统模式下创建或修改函数、存储过程、同义词、聚合函数、操作符等对象。sysadmin用户不具有默认create or replace/alter/grant/revoke系统对象的权限。

默认值：on

debug_assertions

参数说明：控制打开各种断言检查。能够协助调试，当遇到奇怪的问题或者崩溃，请把此参数打开，因为它能暴露编程的错误。要使用这个参数，必须在编译GaussDB的时候定义宏USE_ASSERT_CHECKING（通过configure选项 --enable-cassert完成）。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示打开断言检查。
- off表示不打开断言检查。

说明

当启用断言选项编译GaussDB时，debug_assertions缺省值为on 。

默认值：off

ignore_checksum_failure

参数说明： 设置读取数据时是否忽略校验信息检查失败（但仍然会告警），继续执行可能导致崩溃，传播或隐藏损坏数据，无法从远程节点恢复数据及其他严重问题。不建议用户修改设置。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示忽略数据校验错误。
- off表示数据校验错误正常报错。

默认值：off

ignore_system_indexes

参数说明：读取系统表时忽略系统索引（但是修改系统表时依然同时修改索引）。

该参数属于BACKEND类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

这个参数在从系统索引被破坏的表中恢复数据的时候非常有用。

取值范围：布尔型

- on表示忽略系统索引。
- off表示不忽略系统索引。

默认值：off

post_auth_delay

参数说明：在认证成功后，延迟指定时间，启动服务器连接。允许调试器附加到启动进程上。

该参数属于BACKEND类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，最小值为0，最大值为2147，单位为秒。

默认值：0

说明

此参数只用于调试和问题定位，为避免影响正常业务运行，生产环境下请确保参数值为默认值0。参数设置为非0时可能会因认证延迟时间过长导致数据库实例状态异常。

pre_auth_delay

参数说明：启动服务器连接后，延迟指定时间，进行认证。允许调试器附加到认证过程上。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，最小值为0～60，单位为秒。

默认值：0

说明

此参数只用于调试和问题定位，为避免影响正常业务运行，生产环境下请确保参数值为默认值0。参数设置为非0时可能会因认证延迟时间过长导致数据库实例状态异常。

trace_notify

参数说明：为LISTEN和NOTIFY命令生成大量调试输出。**client_min_messages**或**log_min_messages**级别必须是DEBUG1或者更低时，才能把这些输出分别发送到客户端或者服务器日志。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示打开输出功能。
- off表示关闭输出功能。

默认值：off

trace_recovery_messages

参数说明：启用恢复相关调试输出的日志录，否则将不会被记录。该参数允许覆盖正常设置的**log_min_messages**，但是仅限于特定的消息，这是为了在调试备机中使用。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：枚举类型，有效值有debug5、debug4、debug3、debug2、debug1、log，取值的详细信息请参见**log_min_messages**。

默认值：log

说明

- 默认值log表示不影响记录决策。
- 除默认值外，其他值会导致优先级更高的恢复相关调试信息被记录，因为它们有log优先权。对于常见的log_min_messages设置，这会导致无条件地将它们记录到服务器日志上。

trace_sort

参数说明：控制是否在日志中打印排序操作中的资源使用相关信息。这个选项只有在编译GaussDB的时候定义了TRACE_SORT宏的时候才可用，不过目前TRACE_SORT是由缺省定义的。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示打开控制功能。
- off表示关闭控制功能。

默认值：off

zero_damaged_pages

参数说明：控制检测导致GaussDB报告错误的损坏的页头，终止当前事务。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

设置为on时，会导致系统报告一个警告，把损坏的页面填充为零然后继续处理。这种行为会破坏数据，也就是所有在已经损坏页面上的行记录。但是它允许绕开坏页面然后从表中尚存的未损坏页面上继续检索数据行。因此它在因为硬件或者软件错误导致的崩溃中进行恢复是很有用的。通常不应该把它设置为on，除非不需要从崩溃的页面中恢复数据。

默认值：off

remotetype

参数说明：设置远程连接类型。

该参数属于BACKEND类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：枚举类型，有效值有application、datanode、internaltool。

默认值：application

max_user_defined_exception

参数说明：异常最大个数，默认值不可更改。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，当前只能取固定值1000

默认值：1000

enable_fast_numeric

参数说明：标识是否开启Numeric类型数据运算优化。Numeric数据运算是较为耗时的操作之一，通过将Numeric转化为int64/int128类型，提高Numeric运算的性能。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on/true表示开启Numeric优化。
- off/false表示关闭Numeric优化。

默认值：on

enable_compress_spill

参数说明：标识是否开启下盘压缩功能。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on/true表示开启下盘优化。
- off/false表示关闭下盘优化。

默认值：on

resource_track_log

参数说明：控制自诊断的日志级别。目前仅对多列统计信息进行控制。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

- summary：显示简略的诊断信息。
- detail：显示详细的诊断信息。

目前这两个参数值只在显示多列统计信息未收集的告警的情况下有差别，summary不显示未收集多列统计信息的告警，detail会显示这类告警。

默认值：summary

show_acce_estimate_detail

参数说明：评估信息一般用于运维人员在维护工作中使用，因此该参数默认关闭，此外为了避免这些信息干扰正常的explain信息显示，只有在explain命令的verbose选项打开的情况下才显示评估信息（由于规格变更，当前版本已经不再支持本特性，请不要使用）。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示可以在explain命令的输出中显示评估信息。
- off表示不在explain命令的输出中显示评估信息。

默认值：off

说明

当前版本不支持加速数据库实例，因此该参数设置后不生效。

support_batch_bind

参数说明：控制是否允许通过JDBC、ODBC、Libpq等接口批量绑定和执行PBE形式的语句。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用批量绑定和执行。
- off表示不使用批量绑定和执行。

默认值：on

numa_distribute_mode

参数说明：用于控制部分共享数据和线程在NUMA节点间分布的属性。用于大型多NUMA节点的ARM服务器性能调优，一般不用设置。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，当前有效取值为'none', 'all'。

- none： 表示不启用本特性。
- all：表示将部分共享数据和线程分布到不同的NUMA节点下，减少远端访存次数，提高性能。目前仅适用于拥有多个NUMA节点的ARM服务器，并且要求全部NUMA节点都可用于数据库进程，不支持仅选择一部分NUMA节点。

说明

当前版本x86架构下不支持numa_distribute_mode设置为all。

默认值：'none'

log_pagewriter

参数说明：设置用于增量检查点打开后，显示线程的刷页信息以及增量检查点的详细信息，信息比较多，不建议设置为true。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

默认值：on

advance_xlog_file_num

参数说明：用于控制在后台周期性地提前初始化xlog文件的数目。该参数是为了避免事务提交时执行xlog文件初始化影响性能，但仅在超重负载时才可能出现，因此一般不用配置。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~1000000（0表示不提前初始化）。例如，取值10，表示后台线程会周期性地根据当前xlog写入位置提前初始化10个xlog文件。

默认值：0

enable_beta_opfusion

参数说明：在enable_opfusion参数打开的状态下，如果开启该参数，可以支持TPCC中出现的聚集函数，排序两类SQL语句的加速执行，提升SQL执行性能。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启。
- off表示不开启。

默认值：off

enable_csqual_pushdown

参数说明：进行查询时，是否要将过滤条件下推，进行Rough Check。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示进行查询时，要将过滤条件下推，进行Rough Check。
- off表示进行查询时，不要将过滤条件下推，进行Rough Check。

默认值：on

string_hash_compatible

参数说明：该参数用来说明char类型和varchar/text类型的hash值计算方式是否相同，以此来判断进行分布列从char类型到相同值的varchar/text类型转换，数据分布变化时，是否需要进行重分布。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示计算方式相同，不需要进行重分布。
- off表示计算方式不同，需要进行重分布。

说明

计算方式的不同主要体现在字符串计算hash值时传入的字节长度上。（如果为char，则会忽略字符串后面空格的长度，如果为text或varchar，则会保留字符串后面空格的长度。）hash值的计算会影响到查询的计算结果，因此此参数一旦设置后，在整个数据库使用过程中不能再对其进行修改，以避免查询错误。

默认值：off

pldebugger_timeout

参数说明：该参数用来控制pldebugger server端等待debug端响应的超时时间。

该参数属于USERSET类型参数，请参考表14-1中对应设置方法进行设置。

取值范围：整型，1 ~ 86400，单位为秒。

默认值：15min

plsql_show_all_error

参数说明：该参数用来控制编译PLPGSQL对象时是否支持跳过报错继续编译，具体影响请参考《开发指南》的“Schema > DBE_PLDEVELOPER”章节内的说明。

该参数属于USERSET类型参数，请参考表14-1中对应设置方法进行设置。

取值范围：布尔型

默认值：off

ustore_attr

参数说明：该参数主要用来控制USTORE存储引擎表的信息统计，回滚类型，重点模块(包括数据、索引、回滚段、回放等)运行时数据的校验，主要用于协助研发问题定位。

该参数属于USERSET类型参数，请参考表14-1中对应设置方法进行设置。

取值范围：字符串，该参数值的设置方式采用key-value模式，key和value取值对应关系和说明如下。如果是多个key-value组合设置，中间使用";"作为分隔符，例如：

ustore_attr='ustore_verify_level=FAST;ustore_verify_module=UPAGE:UBTREE:UNDO:REDO'。

说明

ustore_attr设置参数值时，key和value之间的"="前后不要有空格或者其他字符，例如ustore_attr='ustore_verify_level = FAST;，内核代码校验会发现参数不合法，导致参数设置失败。

- ustore_verify_level：控制校验的级别。

  取值范围：字符串，取值不区分大小写，见下述表格详细描述。

  表 14-15 ustore_verify_level 取值含义说明

参数取值	含义
NONE	NONE表示关闭校验，性能测试场景下推荐开启。
FAST	FAST表示快速校验，校验内容少，性能影响最小；兼容旧版本取值：NORMAL。
COMPLETE	COMPLETE表示全量校验，校验内容最多，性能影响比较大；兼容旧版本取值：SLOW。

  默认值：FAST

- ustore_verify_module：控制校验的模块。

  取值范围：字符串，设置值UPAGE，UBTREE，UNDO, REDO，ROACH中的一个或者多个，或者单独设置ALL或者NULL(不区分大小写)。当设置

UPAGE，UBTREE，UNDO，REDO，ROACH中的多个值时，使用":"作为连接符。例如ustore_verify_module=UPAGE:UBTREE:UNDO:REDO。

当用户打开ROACH模块，在ROACH备份过程中将无视ustore_verify_level参数，默认最高级别校验，对性能影响极大，建议谨慎使用。

表 14-16 ustore_verify_module 取值含义说明

参数取值	含义
UPAGE	表示开启数据页面校验。
UBTREE	表示开启UBTREE索引校验。
UNDO	表示开启回滚段数据校验。
REDO	表示开启REDO流程的数据页面校验。
ROACH	表示开启ROACH备份的数据页面校验。
ALL	表示同时开启UPAGE，UBTREE，UNDO，REDO，ROACH模块数据的校验。
NULL	表示同时关闭UPAGE，UBTREE，UNDO，REDO，ROACH模块数据的校验。

默认值：UPAGE:UBTREE:UNDO

- index_trace_level：控制开启索引追踪并控制打印级别，开启后在索引扫描的过程中，会根据不同的打印级别对符合条件的索引元组的信息进行打印。

  取值范围：字符串，取值下表格描述。

  默认值：NO

表 14-17 index_trace_level 取值含义说明

参数取值	含义
NO	不打印任何额外信息。
NORMAL	打印**可见索引元组**相关信息，包括： ● 当前索引元组所在索引页面号以及偏移。 ● 当前元组状态。 ● 当前元组对应的TID以及partOid。 ● 当前元组对应的xmin和xmax信息。 ● 当前元组内容（如果开启enable_log_tuple）。
VISIBILITY	在NORMAL的基础上，额外打印没有通过可见性检查的索引元组的信息，并标明是否可见。
SHOWHIKEY	在VISIBILITY的基础上，尝试打印页面上HIKEY元组的信息。
ALL	打印扫描的索引页面上所有元组的相关信息。

- enable_log_tuple：打印日志级提示信息时，是否允许同时将相关元组的内容打印出来，以便进行问题排查和定位。

  取值范围：on或者off（不区分大小写）

  默认值：off

  备注：该参数已弃用
- enable_ustore_sync_rollback：控制USTORE表是否开启同步回滚。

  取值范围：布尔值

  默认值：true
- enable_ustore_async_rollback：控制USTORE表是否开启异步回滚。

  取值范围：布尔值

  默认值：true
- enable_ustore_page_rollback：控制USTORE表是否开启页面回滚。

  取值范围：布尔值

  默认值：true
- enable_ustore_partial_seqscan：是否允许USTORE表开启部分扫描。

  取值范围：布尔值

  默认值：false
- enable_candidate_buf_usage_count：是否开启缓存区使用计数统计。

  取值范围：布尔值

  默认值：false
- ustats_tracker_naptime：控制USTORE表统计信息周期。

  取值范围：1~INT_MAX/1000

  默认值：20, 单位(秒)
- umax_search_length_for_prune：控制USTORE表prune操作搜索的最大深度。

  取值范围：1~INT_MAX/1000

  默认值：10，单次(次)
- ustore_unit_test：研发白盒测试指定测试参数。

  取值范围：字符串

  默认值：空

默认值：空字符串

注意

ustore_attr参数设置请慎重，建议在工程师协助下修改。

14.3.21 审计

14.3.21.1 审计开关

audit_enabled

参数说明：控制审计进程的开启和关闭。审计进程开启后，将从管道读取后台进程写入的审计信息，并写入审计文件。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示启动审计功能。
- off表示关闭审计功能。

默认值：on

audit_directory

参数说明：审计文件的存储目录。可以是相对于数据目录data的相对路径，也可以是绝对路径，自行指定，仅sysadmin用户可以访问。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

默认值：pg_audit。如果使用om工具部署数据库，则审计日志路径为“$GAUSSLOG/pg_audit/实例名称”。

须知

- 不同的DN实例需要设置不同的审计文件存储目录，否则会导致审计日志异常。
- 当配置文件中audit_directory的值为非法路径时，会导致审计功能无法使用。

说明

- 合法路径：用户对此路径有读写权限。
- 非法路径：用户对此路径无读写权限。

audit_data_format

参数说明：审计日志文件的格式。当前仅支持二进制格式，仅sysadmin用户可以访问。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

默认值：binary

audit_rotation_interval

参数说明：指定创建一个新审计日志文件的时间间隔。当现在的时间减去上次创建一个审计日志的时间超过了此参数值时，服务器将生成一个新的审计日志文件。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1~INT_MAX/60，单位为min。

默认值：1d

须知

请不要随意调整此参数，否则可能会导致audit_resource_policy无法生效，如果需要控制审计日志的存储空间和时间，请使用**audit_resource_policy**、**audit_space_limit**和**audit_file_remain_time**参数进行控制。

audit_rotation_size

参数说明：指定审计日志文件的最大容量。当审计日志消息的总量超过此参数值时，服务器将生成一个新的审计日志文件。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1024~1048576，单位为KB。

默认值：10MB

须知

- 请不要随意调整此参数，否则可能会导致audit_resource_policy无法生效，如果需要控制审计日志的存储空间和时间，请使用audit_resource_policy、audit_space_limit和audit_file_remain_time参数进行控制。
- 审计日志文件中记录的单条日志占用空间大小超过此参数值时会被作为无效日志文件。

audit_resource_policy

参数说明：控制审计日志的保存策略，以空间还是时间限制为优先策略。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示采用空间优先策略，最多存储**audit_space_limit**大小的日志。
- off表示采用时间优先策略，最少存储**audit_file_remain_time**长度时间的日志。

默认值：on

audit_file_remain_time

参数说明：表示需记录审计日志的最短时间要求，该参数在**audit_resource_policy**为off时生效。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~730，单位为day，0表示无时间限制。

默认值：90

audit_space_limit

参数说明：审计文件占用的磁盘空间总量。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1024KB~1024GB，单位为KB。

默认值：1GB

须知

多审计线程场景下，审计文件占用的磁盘空间最小值是audit_thread_num与audit_rotation_size的乘积，如果此参数值设置过小则可能会超过设置的参数值。

audit_file_remain_threshold

参数说明：审计目录下审计文件个数的最大值。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，100~1048576

默认值：

1048576（196核CPU/1536G内存，128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，64核CPU/512G内存，60核CPU/480G内存，32核CPU/256G内存，16核CPU/128G内存，8核CPU/64G内存，4核CPU/32G内存）；1024（4核CPU/16G内存）

须知

- 请尽量保证此参数为1048576，并不要随意调整此参数，否则可能会导致audit_resource_policy无法生效，如果需要控制审计日志的存储空间和时间，请使用audit_resource_policy、audit_space_limit和audit_file_remain_time参数进行控制。
- 多审计线程场景下不建议调整此参数，请保证此参数不小于审计线程个数audit_thread_num，否则会导致审计功能无法正常使用与数据库异常。

audit_thread_num

参数说明：审计线程的个数。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1~48

默认值：1

须知

- 当audit_dml_state开关打开且在高性能场景下，建议增大此参数保证审计消息可以被及时处理和记录。
- 请保证此参数不大于审计目录下审计文件个数的最大值audit_file_remain_threshold，否则会导致审计功能无法正常使用与数据库异常。

14.3.21.2 用户和权限审计

audit_login_logout

参数说明：这个参数决定是否审计用户的登录（包括登录成功和登录失败）、注销。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~7。

- 0表示关闭用户登录、注销审计功能。
- 1表示只审计用户登录成功。
- 2表示只审计用户登录失败。
- 3表示只审计用户登录成功和失败。
- 4表示只审计用户注销。
- 5表示只审计用户注销和登录成功。
- 6表示只审计用户注销和登录失败。
- 7表示审计用户登录成功、失败和注销。

默认值：7

audit_database_process

参数说明：该参数决定是否对数据库的启动、停止、切换和恢复进行审计。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0、1。

- 0表示关闭数据库启动、停止、切换和恢复的审计功能。
- 1表示开启数据库启动、停止、切换和恢复的审计功能。

默认值：1

须知

数据库启动时DN走备升主流程，因此DN启动时审计日志中类型为system_switch。

audit_user_locked

参数说明：该参数决定是否审计用户的锁定和解锁。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0、1。

- 0表示关闭用户锁定和解锁审计功能。
- 1表示开启审计用户锁定和解锁功能。

默认值：1

audit_user_violation

参数说明：该参数决定是否审计用户的越权访问操作。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0、1。

- 0表示关闭用户越权操作审计功能。
- 1表示开启用户越权操作审计功能。

默认值：0

audit_grant_revoke

参数说明：该参数决定是否审计用户权限授予和回收的操作。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0、1。

- 0表示关闭审计用户权限授予和回收功能。
- 1表示开启审计用户权限授予和回收功能。

默认值：1

full_audit_users

参数说明：该参数用于配置全量审计用户列表，对列表中的用户执行的所有可被审计的操作记录审计日志。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，多个用户名需使用逗号分隔。

默认值：空字符串

no_audit_client

参数说明：该参数用于配置不需要审计的客户端名称及IP地址列表。参数格式为：客户端名称@IP，同pg_query_audit函数中的client_conninfo字段，例如“cm_agent@127.0.0.1, gs_clean@127.0.0.1”。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，多个配置项需使用逗号分隔。

默认值：空字符串

须知

- 当执行的SQL语句同时满足full_audit_users和no_audit_client参数配置时，以no_audit_client配置优先，不记录审计日志。
- 数据库服务端内部工具或节点之间通信也会产生审计日志，针对这些风险较低的审计场景的可以通过配置no_audit_client参数不记录审计，以节约审计日志占用空间，提升审计日志查询性能。

14.3.21.3 操作审计

audit_system_object

参数说明：该参数决定是否对数据库对象的CREATE、DROP、ALTER操作进行审计。数据库对象包括DATABASE、USER、schema、TABLE等。通过修改该配置参数的值，可以只审计需要的数据库对象的操作。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0～536870911

- 0代表关闭数据库对象的CREATE、DROP、ALTER操作审计功能。
- 非0代表只审计某类或者某些数据库对象的CREATE、DROP、ALTER操作。

取值说明：

该参数的值由29个二进制位的组合求出，这29个二进制位分别代表29类数据库对象。如果对应的二进制位取值为0，表示不审计对应的数据库对象的CREATE、DROP、ALTER操作；取值为1，表示审计对应的数据库对象的CREATE、DROP、ALTER操作。这29个二进制位代表的具体审计内容请参见**表14-18**。

用于记录SQL PATCH的参数存在特殊性，如果对该对象进行审计且audit_dml_state_select也开启时，对于一条SQL PATCH操作的审计日志会作为DML和DDL被记录两次。

默认值：67121159

表 14-18 audit_system_object 取值含义说明

二进制位	含义	取值说明
第0位	是否审计DATABASE对象的CREATE、DROP、ALTER操作。	• 0表示不审计该对象的CREATE、DROP、ALTER操作。 • 1表示审计该对象的CREATE、DROP、ALTER操作。
第1位	是否审计SCHEMA对象的CREATE、DROP、ALTER操作。	• 0表示不审计该对象的CREATE、DROP、ALTER操作。 • 1表示审计该对象的CREATE、DROP、ALTER操作。

二进制位	含义	取值说明
第2位	是否审计USER对象的CREATE、DROP、ALTER操作。	● 0表示不审计该对象的CREATE、DROP、ALTER操作。 ● 1表示审计该对象的CREATE、DROP、ALTER操作。
第3位	是否审计TABLE对象的CREATE、DROP、ALTER、TRUNCATE操作。	● 0表示不审计该对象的CREATE、DROP、ALTER、TRUNCATE操作。 ● 1表示审计该对象的CREATE、DROP、ALTER、TRUNCATE操作。
第4位	是否审计INDEX对象的CREATE、DROP、ALTER操作。	● 0表示不审计该对象的CREATE、DROP、ALTER操作。 ● 1表示审计该对象的CREATE、DROP、ALTER操作。
第5位	是否审计VIEW/MATVIEW对象的CREATE、DROP操作。	● 0表示不审计该对象的CREATE、DROP操作。 ● 1表示审计该对象的CREATE、DROP操作。
第6位	是否审计TRIGGER对象的CREATE、DROP、ALTER操作。	● 0表示不审计该对象的CREATE、DROP、ALTER操作。 ● 1表示审计该对象的CREATE、DROP、ALTER操作。
第7位	是否审计PROCEDURE/FUNCTION对象的CREATE、DROP、ALTER操作。	● 0表示不审计该对象的CREATE、DROP、ALTER操作。 ● 1表示审计该对象的CREATE、DROP、ALTER操作。
第8位	是否审计TABLESPACE对象的CREATE、DROP、ALTER操作。	● 0表示不审计该对象的CREATE、DROP、ALTER操作。 ● 1表示审计该对象的CREATE、DROP、ALTER操作。
第9位	是否审计RESOURCE POOL对象的CREATE、DROP、ALTER操作。	● 0表示不审计该对象的CREATE、DROP、ALTER操作。 ● 1表示审计该对象的CREATE、DROP、ALTER操作
第10位	是否审计WORKLOAD对象的CREATE、DROP、ALTER操作。	● 0表示不审计该对象的CREATE、DROP、ALTER操作。 ● 1表示审计该对象的CREATE、DROP、ALTER操作
第11位	保留	-

二进制位	含义	取值说明
第12位	是否审计DATA SOURCE对象的CRAETE、DROP、ALTER操作。	• 0表示不审计该对象的CREATE、DROP、ALTER操作。 • 1表示审计该对象的CREATE、DROP、ALTER操作。
第13位	保留	-
第14位	是否审计ROW LEVEL SECURITY对象的CREATE、DROP、ALTER操作。	• 0表示不审计该对象的CREATE、DROP、ALTER操作。 • 1表示审计该对象的CREATE、DROP、ALTER操作。
第15位	是否审计TYPE对象的CREATE、DROP、ALTER操作。	• 0表示不审计TYPE对象的CREATE、DROP、ALTER操作。 • 1表示审计TYPE对象的CREATE、DROP、ALTER操作。
第16位	是否审计TEXT SEARCH对象（CONFIGURATION和DICTIONARY）的CREATE、DROP、ALTER操作。	• 0表示不审计TEXT SEARCH对象的CREATE、DROP、ALTER操作。 • 1表示审计TEXT SEARCH对象的CREATE、DROP、ALTER操作。
第17位	是否审计DIRECTORY对象的CREATE、DROP、ALTER操作。	• 0表示不审计DIRECTORY对象的CREATE、DROP、ALTER操作。 • 1表示审计DIRECTORY对象的CREATE、DROP、ALTER操作。
第18位	是否审计SYNONYM对象的CREATE、DROP、ALTER操作。	• 0表示不审计SYNONYM对象的CREATE、DROP、ALTER操作。 • 1表示审计SYNONYM对象的CREATE、DROP、ALTER操作。
第19位	是否审计SEQUENCE对象的CREATE、DROP、ALTER操作。	• 0表示不审计SEQUENCE对象的CREATE、DROP、ALTER操作。 • 1表示审计SEQUENCE对象的CREATE、DROP、ALTER操作。
第20位	是否审计CMK、CEK对象的CREATE、DROP操作。	• 0表示不审计CMK、CEK对象的CREATE、ALTER、DROP操作。 • 1表示审计CMK、CEK对象的CREATE、ALTER、DROP操作。
第21位	是否审计PACKAGE对象的CREATE、DROP、ALTER操作。	• 0表示不审计PACKAGE对象的CREATE、DROP、ALTER操作。 • 1表示审计PACKAGE对象的CREATE、DROP、ALTER操作。

二进制位	含义	取值说明
第22位	是否审计MODEL对象的CREATE、DROP操作。	• 0表示不审计MODEL对象的CREATE、ALTER操作。 • 1表示审计MODEL对象的CREATE、DROP操作。
第23位	是否审计PUBLICATION和SUBSCRIPTION对象的CREATE、DROP、ALTER操作。	• 0表示不审计PUBLICATION和SUBSCRIPTION对象的CREATE、DROP、ALTER操作。 • 1表示审计PUBLICATION和SUBSCRIPTION对象的CREATE、DROP、ALTER操作。
第24位	是否审计对gs_global_config全局对象的ALTER、DROP操作。	• 0表示不审计对系统表gs_global_config全局对象的ALTER、DROP操作。 • 1表示审计对系统表gs_global_config全局对象的ALTER、DROP操作。
第25位	是否审计FOREIGN DATA WRAPPER对象的CREATE、DROP、ALTER操作。	• 0表示不审计FOREIGN DATA WRAPPER对象的CREATE、DROP、ALTER操作。 • 1表示审计FOREIGN DATA WRAPPER对象的CREATE、DROP、ALTER操作。
第26位	是否审计SQL PATCH对象的CREATE、ENABLE、DISABLE、DROP操作。	• 0表示不审计SQL PATCH对象的CREATE、ENABLE、DISABLE、DROP操作。 • 1表示审计SQL PATCH对象的CREATE、ENABLE、DISABLE、DROP操作。
第27位	是否审计EVENT对象的CREATE、ALTER、DROP操作。	• 0表示不审计EVENT对象的CREATE、ENABLE、DISABLE、DROP操作。 • 1表示审计EVENT对象的CREATE、ENABLE、DISABLE、DROP操作。
第28位	是否审计DBLINK对象的CREATE、ALTER、DROP操作。目前DATABASE LINK功能暂不支持。	• 0表示不审计DBLINK对象的CREATE、ALTER、DROP操作。 • 1表示审计DBLINK对象的CREATE、ALTER、DROP操作。

audit_dml_state

参数说明：这个参数决定是否对具体表的INSERT、UPDATE、DELETE操作进行审计。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0、1。

- 0表示关闭具体表的DML操作（SELECT除外）审计功能。
- 1表示开启具体表的DML操作（SELECT除外）审计功能。

默认值：0

audit_dml_state_select

参数说明：这个参数决定是否对SELECT操作进行审计。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0、1。

- 0表示关闭SELECT操作审计功能。
- 1表示开启SELECT审计操作功能。

默认值：0

audit_function_exec

参数说明：这个参数决定在执行存储过程、匿名块或自定义函数（不包括系统自带函数）时是否记录审计信息。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0、1。

- 0表示关闭对存储过程或函数执行的审计功能。
- 1表示开启对存储过程或函数执行的审计功能。

默认值：0

audit_system_function_exec

参数说明：这个参数决定在执行白名单内的系统函数时是否记录审计日志。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0、1。

- 0表示关闭对系统函数执行的审计功能。
- 1表示开启对系统函数执行的审计功能。

默认值：0

支持记录审计的系统函数白名单如下表所示：

set_working_grand_version_num_manually	set_config	pg_cancel_backend	pg_cancel_session	pg_reload_conf	pg_rotate_logfile

pg_terminate_session	pg_terminate_backend	pg_create_restore_point	pg_start_backup	pg_stop_backup	pg_switch_xlog
pg_cbm_rotate_file	pg_cbm_get_merged_file	pg_cbm_recycle_file	pg_enable_delay_ddl_recycle	pg_disable_delay_ddl_recycle	gs_roach_stop_backup
gs_roach_enable_delay_ddl_recycle	gs_roach_disable_delay_ddl_recycle	gs_roach_switch_xlog	pg_last_xlog_receive_location	pg_xlog_replay_pause	pg_xlog_replay_resume
gs_pitr_clean_history_global_barriers	gs_pitr_archive_slot_force_advance	pg_create_physical_replication_slot_extern	gs_set_obs_delete_location	gs_hadr_do_switchover	gs_set_obs_delete_location_with_slotname
gs_streaming_dr_in_switchover	gs_upload_obs_file	gs_download_obs_file	gs_set_obs_file_context	gs_get_hadr_key_cn	pg_advisory_lock
pg_advisory_lock_shared	pg_advisory_unlock	pg_advisory_unlock_shared	pg_advisory_unlock_all	pg_advisory_xact_lock	pg_advisory_xact_lock_shared
pg_try_advisory_lock	pg_try_advisory_lock_shared	pg_try_advisory_xact_lock	pg_try_advisory_xact_lock_shared	pg_create_logical_replication_slot	pg_drop_replication_slot
pg_logical_slot_peek_changes	pg_logical_slot_get_changes	pg_logical_slot_get_binary_changes	pg_replication_slot_advance	pg_replication_origin_create	pg_replication_origin_drop
pg_replication_origin_session_setup	pg_replication_origin_session_reset	pg_replication_origin_session_progress	pg_replication_origin_xact_setup	pg_replication_origin_xact_reset	pg_replication_origin_advance
local_space_shrink	gs_space_shrink	pg_free_remain_segment	gs_fault_inject	gs_repair_file	local_clear_bad_block_info
gs_repair_page	-	-	-	-	-

audit_copy_exec

参数说明：这个参数决定是否对COPY操作进行审计。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0、1。

- 0表示关闭COPY审计功能。
- 1表示开启COPY审计功能。

默认值：1

audit_set_parameter

参数说明：这个参数决定是否对SET操作进行审计。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0、1。

- 0表示关闭SET审计功能。
- 1表示开启SET审计功能。

默认值：0

audit_xid_info

参数说明：这个参数决定是否在审计日志字段detail_info中记录SQL语句的事务ID。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0、1。

- 0表示关闭审计日志记录事务ID功能。
- 1表示开启审计日志记录事务ID功能。

默认值：0

须知

如果开启此开关，审计日志中detail_info信息则以xid开始，例如：

```
detail_info: xid=14619 , create table t1(id int);
```

对于不存在事务ID的审计行为，记录xid=NA。

enableSeparationOfDuty

参数说明：是否开启三权分立选项。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启三权分立。
- off表示不开启三权分立。

默认值：off

enable_nonsysadmin_execute_direct

参数说明：是否允许非系统管理员和非监控管理员执行EXECUTE DIRECT ON语句。

该参数属于POSTMASTER类型参数，请参考表14-1中对应设置方法进行设置。

取值范围：布尔型

- on表示允许任意用户执行EXECUTE DIRECT ON语句。
- off表示只允许系统管理员和监控管理员执行EXECUTE DIRECT ON语句。

默认值：off

enable_access_server_directory

参数说明：是否开启非初始用户创建、修改和删除DIRECTORY的权限。

该参数属于SIGHUP类型参数，请参考表14-1中对应设置方法进行设置。

取值范围：布尔型

- on表示开启非初始用户创建、修改和删除DIRECTORY的权限。
- off表示不开启非初始用户创建、修改和删除DIRECTORY的权限。

默认值：off

须知

- 出于安全考虑，默认情况下，只有初始用户才能够创建、修改和删除DIRECTORY对象。
- 如果开启了enable_access_server_directory，具有SYSADMIN权限的用户和继承了内置角色gs_role_directory_create权限的用户可以创建directory对象；具有SYSADMIN权限的用户、directory对象的属主、被授予了该directory的DROP权限的用户或者继承了内置角色gs_role_directory_drop权限的用户可以删除directory对象；具有SYSADMIN权限的用户和directory对象的属主可以修改directory对象的所有者，且要求该用户是新属主的成员。

14.3.22 CM 相关参数

CM相关参数的修改对GaussDB的运行机制有影响，建议由GaussDB的工程师协助修改。修改CM相关参数的方法，请参考表14-2中方式一进行设置。

14.3.22.1 cm_agent 参数

log_dir

参数说明：log_dir决定存放cm_agent日志文件的目录。可以是绝对路径，或者是相对路径（相对于cm_agent数据目录的路径）。

取值范围：字符串。修改后需要重启cm_agent才能生效。参数修改请参考表14-2进行设置。

默认值：“log”，表示在cm_agent数据目录下生成cm_agent日志。

log_file_size

参数说明：控制日志文件的大小。当日志文件达到指定大小时，则重新创建一个日志文件记录日志信息。

取值范围：整型，取值范围0~2047，单位为MB。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：16MB

log_min_messages

参数说明：控制写到cm_agent日志文件中的消息级别。每个级别都包含排在它后面的所有级别中的信息。级别越低，服务器运行日志中记录的消息就越少。

取值范围：枚举类型，有效值有debug5、debug1、warning、error、log、fatal。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：warning

incremental_build

参数说明：控制重建备DN模式是否为增量。打开这个开关，则增量重建备DN；否则，全量重建备DN。

取值范围：布尔型，有效值有on、off。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：on

alarm_component

参数说明：设置用于处理告警内容的告警组件的位置。

取值范围：字符串。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

- 若前置脚本gs_preinstall中的--alarm-type参数设置为5时，表示未对接第三方组件，告警写入system_alarm日志，此时GUC参数alarm_component的取值为：/opt/huawei/snas/bin/snas_cm_cmd。
- 若前置脚本gs_preinstall中的--alarm-type参数设置为1时，表示对接第三方组件，此时GUC参数alarm_component的值为第三方组件的可执行程序的绝对路径。

默认值：/opt/huawei/snas/bin/snas_cm_cmd

alarm_report_interval

参数说明：指定告警上报的时间间隔。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

取值范围：非负整型，单位为秒。

默认值：1

alarm_report_max_count

参数说明：指定告警上报的最大次数。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

取值范围：非负整型。

默认值：1

agent_report_interval

参数说明：cm_agent上报实例状态的时间间隔。

取值范围：整型，单位为秒。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：1

agent_phony_dead_check_interval

参数说明：cm_agent检测DN进程是否僵死的时间间隔。

取值范围：整型，单位为秒。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：10

agent_check_interval

参数说明：cm_agent查询DN等实例状态的时间间隔。

取值范围：整型，单位为秒。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：2

agent_heartbeat_timeout

参数说明：cm_agent连接cm_server心跳超时时间。

取值范围：整型，2~2^{31} - 1，单位为秒。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：8

agent_connect_timeout

参数说明：cm_agent连接cm_server超时时间。

取值范围：整型，单位为秒。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：1

agent_connect_retries

参数说明：cm_agent连接cm_server尝试次数。

取值范围：整型。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：15

agent_kill_instance_timeout

参数说明：当cm_agent在无法连接cm_server主节点后，发起一次终止本节点上所有实例的操作之前，所需等待的时间间隔。

取值范围：整型。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：0，不发起终止本节点上所有实例的操作。

security_mode

参数说明：控制是否以安全模式启动DN。打开这个开关，则以安全模式启动DN；否则，以非安全模式启动DN。

取值范围：布尔型，有效值有on、off。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：off

upgrade_from

参数说明：就地升级过程中使用，用于标示升级前数据库的内部版本号，此参数禁止手动修改。

取值范围：非负整型。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：0

process_cpu_affinity

参数说明：控制是否以绑核优化模式启动主DN进程。配置该参数为0，则不进行绑核优化；否则，进行绑核优化，且物理CPU片数为2^n个。仅支持ARM。

取值范围：整型，0~2。修改后需要重启数据库、cm_agent才能生效，参数修改请参考**表14-2**进行设置。

默认值：0

log_threshold_check_interval

参数说明：日志压缩和清除的时间间隔，每1800秒压缩和清理一次。

取值范围：整型，0~2147483647，单位为秒。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：1800

dilatation_shard_count_for_disk_capacity_alarm

参数说明：扩容场景下，设置新增的扩容分片数，用于上报磁盘容量告警时的阈值计算。

> **说明**
>
> 该分片数请与实际扩容分片数设置为一致。

取值范围：整型，0~2^{32} - 1，单位为个。该参数设置为0，表示关闭磁盘扩容告警上报；该参数设置为大于0，表示开启磁盘扩容告警上报，且告警上报的阈值根据此参数设置的分片数量进行计算。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：1

log_max_size

参数说明：控制日志最大存储值。

取值范围：整型，0~2147483647，单位为MB。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：10240

log_max_count

参数说明：硬盘上可存储的最多日志数量。

取值范围：整型，0~10000，单位为个。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：10000

log_saved_days

参数说明：日志保存的天数。

取值范围：整型，0~1000，单位为天。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：90

enable_log_compress

参数说明：控制压缩日志功能。

取值范围：布尔型。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

- on表示允许压缩日志。
- off表示不允许压缩日志。

默认值：on

agent_backup_open

参数说明：灾备数据库实例设置，开启后CM按照灾备数据库实例模式运行。

取值范围：整型，0~1。修改后需要重启cm_agent才能生效。参数修改请参考**表14-1**进行设置。

- 0表示关闭。

- 1表示开启。

默认值：0

enable_xc_maintenance_mode

参数说明：在数据库实例为只读模式下，控制是否可以修改pgxc_node系统表。

取值范围：布尔型。修改后需要重启cm_agent才能生效。参数修改请参考**表14-1**进行设置。

- on表示开启可以修改pgxc_node系统表功能。
- off表示关闭可以修改pgxc_node系统表功能。

默认值：on

unix_socket_directory

参数说明：unix套接字的目录位置。

取值范围：字符串。修改后可以reload生效，参数修改请参考**表14-1**进行设置。

默认值：''

enable_dcf

参数说明：DCF模式开关。

取值范围：布尔型。修改后可以reload生效，参数修改请参考**表14-1**进行设置。

- 0表示关闭。
- 1表示开启。

默认值：off

disaster_recovery_type

参数说明：主备数据库实例灾备关系的类型。

取值范围：整型，0~2。修改后可以reload生效，参数修改请参考**表14-1**进行设置。

- 0表示未搭建灾备关系。
- 1表示搭建了obs灾备关系。
- 2表示搭建了流式灾备关系

默认值：0

14.3.22.2 cm_server 参数

log_dir

参数说明：log_dir决定存放cm_server日志文件的目录。它可以是绝对路径，或者是相对路径（相对于cm_server数据目录的路径）。

取值范围：字符串。修改后需要重启cm_server才能生效。参数修改请参考**表14-2**进行设置。

默认值：“log”，表示在cm_server数据目录下生成cm_server日志。

log_file_size

参数说明：控制日志文件的大小。当日志文件达到指定大小时，则重新创建一个日志文件记录日志信息。

取值范围：整型，取值范围0~2047，单位为MB。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：16MB

log_min_messages

参数说明：控制写到cm_server日志文件中的消息级别。每个级别都包含排在它后面的所有级别中的信息。级别越低，服务器运行日志中记录的消息就越少。

取值范围：枚举类型，有效值有debug5、debug1、log、warning、error、fatal。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：warning

thread_count

参数说明：cm_server线程池的线程数。

取值范围：整型，2~1000。修改后需要重启cm_server才能生效。参数修改请参考**表14-2**进行设置。

默认值：1000

alarm_component

参数说明：设置用于处理告警内容的告警组件的位置。

取值范围：字符串。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

- 若前置脚本gs_preinstall中的--alarm-type参数设置为5时，表示未对接第三方组件，告警写入system_alarm日志，此时GUC参数alarm_component的取值为：/opt/huawei/snas/bin/snas_cm_cmd。
- 若前置脚本gs_preinstall中的--alarm-type参数设置为1时，表示对接第三方组件，此时GUC参数alarm_component的值为第三方组件的可执行程序的绝对路径。

默认值：/opt/huawei/snas/bin/snas_cm_cmd

instance_failover_delay_timeout

参数说明：cm_server检测到主机宕机，failover备机的延迟时间。

取值范围：整型，单位为秒。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：0

instance_heartbeat_timeout

参数说明：实例心跳超时时间。

取值范围：整型，单位为秒。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：6

cmserver_ha_connect_timeout

参数说明：cm_server主备连接超时时间。

取值范围：整型，单位为秒。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：2

cmserver_ha_heartbeat_timeout

参数说明：cm_server主备心跳超时时间。

取值范围：整型，单位为秒。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：6

phony_dead_effective_time

参数说明：用于DN进程的僵死检测，当检测到的僵死次数大于该参数值，认为进程僵死，将进程重启。

取值范围：整型，单位为次数。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：5

enable_transaction_read_only

参数说明：控制数据库是否为只读模式开关。

取值范围：布尔型，有效值有on，off，true，false，yes，no，1，0。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：on

datastorage_threshold_check_interval

参数说明：检测磁盘占用的时间间隔。间隔用户指定时间，检测一次磁盘占用。

取值范围：整型，单位为秒。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：10

datastorage_threshold_value_check

参数说明：设置数据库只读模式的磁盘占用阈值，当数据目录所在磁盘占用超过这个阈值，自动将数据库设置为只读模式。调整该参数时，建议同步调整dn的max_size_for_xlog_retention参数，避免因备份操作触发实例只读阈值。

取值范围：整型，1 ~ 99，表示百分比。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：85

max_datastorage_threshold_check

参数说明：设置磁盘使用率的最大检测间隔时间。当用户手动修改只读模式参数后，会自动在指定间隔时间后开启磁盘满检测操作。

取值范围：整型，单位为秒。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：43200

cmserver_ha_status_interval

参数说明：cm_server主备同步状态信息间隔时间。

取值范围：整型，单位为秒。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：1

cmserver_self_vote_timeout

参数说明：cm_server自仲裁超时时间。

取值范围：整型，单位为秒。修改后可以reload生效，参数修改请参考**表14-2**行设置。

默认值：6

alarm_report_interval

参数说明：指定告警上报的时间间隔。

取值范围：非负整型，单位为秒。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：3

alarm_report_max_count

参数说明：指定告警上报的最大次数。

取值范围：非负整型。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：1

enable_az_auto_switchover

参数说明：AZ自动切换开关，若打开，则表示允许cm_server自动切换AZ。否则当发生dn故障等情况时，即使当前AZ已经不再可用，也不会自动切换到其它AZ上，除非手动执行切换命令。

取值范围：非负整型，0或1，0表示开关关闭，1表示开关打开。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：1

instance_keep_heartbeat_timeout

参数说明：cm_agent会定期检测实例状态并上报给cm_server，若实例状态长时间无法成功检测，累积次数超出该数值，则cm_server将下发命令给agent重启该实例。

取值范围：整型，单位为秒。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：40

az_switchover_threshold

参数说明：若一个AZ内DN分片的故障率（故障的dn分片数 / 总dn分片数 * 100%）超过该数值，则会触发AZ自动切换。

取值范围：整型，0~100。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：100

az_check_and_arbitrate_interval

参数说明：当某个AZ状态不正常时，会触发AZ自动切换，该参数是检测AZ状态的时间间隔。

取值范围：整型，单位为秒。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：2

az_connect_check_interval

参数说明：定时检测AZ间的网络连接，该参数表示连续两次检测之间的间隔时间。

取值范围：整型，单位为秒。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：60

az_connect_check_delay_time

参数说明：每次检测AZ间的网络连接时有多次重试，该参数表示两次重试之间的延迟时间。

取值范围：整型，单位为秒。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：150

cmserver_demote_delay_on_etcd_fault

参数说明：因为etcd不健康而导致cm_server从主降为备的时间间隔。

取值范围：整型，单位为秒。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：8

instance_phony_dead_restart_interval

参数说明：当dn实例僵死时，会被cm_agent重启，相同的实例连续因僵死被杀时，其间隔时间不能小于该参数数值，否则cm_agent不会下发命令。

取值范围：整型，单位为秒。最小生效值为1800，如果设置小于此值实际生效值为1800。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：21600

cm_auth_method

参数说明：CM模块端口认证方式，trust表示未配置端口认证，gss表示采用kerberos端口认证。必须注意的是：只有当kerberos服务端和客户端成功安装后才能修改为gss，否则CM模块无法正常通信，将影响数据库状态。

取值范围：枚举类型，有效值有trust, gss。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：trust

cm_krb_server_keyfile

参数说明：kerberos服务端key文件所在位置，需要配置为绝对路径。该文件通常为${GAUSSHOME}/kerberos路径下，以keytab格式结尾，文件名与数据库运行所在用户名相同。与上述cm_auth_method参数是配对的，当cm_auth_method参数修改为gss时，该参数也必须配置为正确路径，否则将影响数据库状态

取值范围：字符串类型，修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：${GAUSSHOME}/kerberos/{UserName}.keytab，默认值无法生效，仅作为提示

cm_server_arbitrate_delay_base_time_out

参数说明：cm_server仲裁延迟基础时长。cm_server主断连后，仲裁启动计时开始，经过仲裁延迟时长后，将选出新的cm_server主。其中仲裁延迟时长由仲裁延迟基础时长、节点index（server ID序号）和增量时长共同决定。公式为：仲裁延迟时长=仲裁延迟基础时长+节点index*仲裁延迟增量时长参数

取值范围：整型，index>0，单位为秒。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：10

cm_server_arbitrate_delay_incrememtal_time_out

参数说明：cm_server仲裁延迟增量时长。cm_server主断连后，仲裁启动计时开始，经过仲裁延迟时长后，将选出新的cm_server主。其中仲裁延迟时长由仲裁延迟基础时长、节点index（server ID序号）和增量时长共同决定。公式为：仲裁延迟时长=仲裁延迟基础时长+节点index*仲裁延迟增量时长参数

取值范围：整型，index>0，单位为秒。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：3

force_promote

参数说明：cm_server是否打开强切逻辑（指数据库状态为Unknown的时候以丢失部分数据为代价保证数据库基本功能可用）的开关。0代表功能关闭，1代表功能开启。该参数同时适用于dn。

取值范围：整型，0~1。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：0

switch_rto

参数说明：cm_server强切逻辑等待时延。在force_promote被置为1时，当数据库的某一分片处于无主状态开始计时，等待该延迟时间后开始执行强切逻辑。

取值范围：整型，60~2147483647，单位为秒。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：600

enable_finishredo_retrieve

参数说明：cm_server强切后是否对redo切除的xlog进行数据找回的功能开关。在enable_finishredo_retrieve被置为on时，发生强切后进行数据自动找回。

取值范围：布尔型，修改后可以reload生效，参数修改请参考**表14-2**进行设置。

- off表示关闭。
- on表示开启。

默认值：off

backup_open

参数说明：灾备数据库实例设置，开启后CM按照灾备数据库实例模式运行

取值范围：整型，0~1。修改后需要重启cm_server才能生效。非灾备数据库实例不能开启该参数。参数修改请参考**表14-2**进行设置。

- 0表示关闭。
- 1表示开启。

默认值：0

enable_dcf

参数说明：DCF模式开关。

取值范围：布尔型。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

- 0表示关闭。
- 1表示开启。

默认值：off

install_type

参数说明：容灾数据库实例相关的设置，用来区别是否是基于dorado的数据库实例。

取值范围：整型，0~2。修改后需要重启cm_server才能生效。非灾备数据库实例不能开启该参数。参数修改请参考**表14-2**进行设置。

- 0表示未搭建容灾关系的数据库实例。
- 1表示基于dorado的数据库实例。
- 2表示基于流式的数据库实例。

默认值：0

enable_ssl

参数说明：ssl证书开关。

取值范围：布尔型。打开后使用ssl证书加密通信。修改后需要重启cm_server才能生效。参数修改请参考**表14-2**进行设置。

- on表示启用ssl。
- off表示不启用ssl。
- **默认值**：off

须知

出于安全性考虑，建议不要关闭该配置。关闭后cm将**不使用**加密通信，所有信息明文传播，可能带来窃听、篡改、冒充等安全风险。

ssl_cert_expire_alert_threshold

参数说明：ssl证书过期告警时间。

取值范围：整型，单位为天。证书过期时间少于该时间时，上报证书即将过期告警。修改后需要重启cm_server才能生效。参数修改请参考**表14-2**进行设置。

默认值：90

ssl_cert_expire_check_interval

参数说明：ssl证书过期检测周期。

取值范围：整型，单位为秒。修改后需要重启cm_server才能生效。参数修改请参考**表14-2**进行设置。

默认值：86400

delay_arbitrate_timeout

参数说明：设置等待跟主DN同AZ节点redo回放后升主的时间。

取值范围：整型，[0, 21474836]，单位：秒。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：0

ddb_type

参数说明：etcd，dcc模式切换开关。

取值范围：整型。0：etcd；1：dcc。修改后需要重启cm_server才能生效。参数修改请参考**表14-2**进行设置。

默认值：0

ddb_log_level

参数说明：设置ddb日志级别。

关闭日志：“NONE”，NONE表示关闭日志打印，不能与以下日志级别混合使用。

开启日志：“RUN_ERR|RUN_WAR|RUN_INF|DEBUG_ERR|DEBUG_WAR|DEBUG_INF|TRACE|PROFILE|OPER”日志级别可以从上述字符串中选取字符串并使用竖线组合使用，不能配置空串。

取值范围：字符串，RUN_ERR|RUN_WAR|RUN_INF|DEBUG_ERR|DEBUG_WAR|DEBUG_INF|TRACE|PROFILE|OPER。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：RUN_ERR|RUN_WAR|DEBUG_ERR|OPER|RUN_INF|PROFILE

ddb_log_backup_file_count

参数说明：最大保存日志文件个数。

取值范围：整型，[1, 100]。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：10

ddb_max_log_file_size

参数说明：单条日志最大字节数。

取值范围：字符串，[1M, 1000M]。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：10M

ddb_log_suppress_enable

参数说明：是否开启日志抑制功能。

取值范围：整型，0：关闭；1：开启。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：1

ddb_election_timeout

参数说明：dcc选举超时时间。

取值范围：整型，[1, 600],单位：秒。修改后可以reload生效，参数修改请参考**表14-2**进行设置。

默认值：3

enable_synclist_single_inst

参数说明：开启降副本降至一主零备功能。

取值范围：布尔型，开启后降副本会降至一主零备，参数设置错误，按默认值处理。修改后可以reload生效。参数修改请参考《工具参考》的“统一数据库管理工具 > cm_ctl工具介绍”章节的表“set cm参数”进行设置。

- off：表示关闭降副本降至一主零备功能。
- on：表示开启降副本降至一主零备功能。

默认值：off

14.3.23 升级参数

IsInplaceUpgrade

参数说明：是否在升级的过程中。该参数用户无法修改，仅sysadmin用户可以访问。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示在升级过程中。
- off表示不在升级过程中。

默认值：off

inplace_upgrade_next_system_object_oids

参数说明：就地升级过程中，新增系统对象的OID。该参数用户无法修改。

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

默认值：空

upgrade_mode

参数说明：升级模式。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：整数，0~INT_MAX

- 0表示不在升级过程中。
- 1表示在就地升级过程中。
- 2表示在灰度升级过程中。

默认值：0

说明

特殊情况：在使用灰度升级的情况下，若选择策略为大版本升级，即需要执行升级脚本和替换二进制包，会将upgrade_mode设置为2，选择策略为小版本升级，只替换二进制包，则不会设置upgrade_mode设置为2。

14.3.24 其它选项

enable_default_ustore_table

参数说明：指定是否开启默认支持Ustore存储引擎，该参数为on时，创建的表类型都为Ustore表。

该参数属于USERSET类型，请参考**表14-1**中对应设置方法进行设置。特别需要注意，使用Ustore表，必须要开启track_counts和track_activities参数，否则会引起空间膨胀。

取值范围：[off,on]

默认值：on

enable_ustore

参数说明：指定是否开启Ustore存储引擎，该参数为on时，支持创建Ustore表。特别需要注意，使用Ustore表，必须要开启track_counts和track_activities参数，否则会引起空间膨胀。

该参数属于POSTMASTER类型，请参考**表14-1**中对应设置方法进行设置。

取值范围：[off,on]

默认值：on

reserve_space_for_nullable_atts

参数说明：指定是否为Ustore表的可空属性预留空间。该参数为on时默认为ustore表的可空属性预留空间。

该参数属于USERSET类型，请参考**表14-1**中对应设置方法进行设置。

取值范围：[off,on]

默认值：on

server_version

参数说明：报告服务器版本号(字符串形式)。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。该参数继承自PostgreSQL内核，表示当前数据库内核兼容PostgreSQL对应的server_version版本，无实际含义，为保持北向对外工具接口的生态兼容性(工具连接时查询)，保留该参数。该参数不推荐使用，如想查询服务器版本号，可通过函数opengauss_version()获取。

取值范围：字符串

默认值：9.2.4

server_version_num

参数说明：报告服务器版本号(整数形式)。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。该参数继承自PostgreSQL内核，表示当前数据库内核兼容PostgreSQL对应的server_version_num版本，无实际含义，为保持北向对外工具接口的生态兼容性(工具连接时查询)，保留该参数。

取值范围：整数

默认值：90204

block_size

参数说明：报告当前数据库所使用的块大小。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：8192

默认值：8192

segment_size

参数说明：报告当前数据库所使用的段文件大小。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

单位: 8KB

默认值：131072, 即1GB

max_index_keys

参数说明：报告当前数据库能够支持的索引键值的最大数目。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

默认值：32

integer_datetimes

参数说明：报告是否支持64位整数形式的日期和时间格式。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：布尔型

- on表示支持。
- off表示不支持。

默认值：on

lc_collate

参数说明：报告当前数据库的字符串排序区域设置。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

默认值：依赖于数据库安装部署时的配置

lc_ctype

参数说明：报告当前数据库的字母类别区域设置。如：哪些字符属于字母，它对应的大写形式是什么。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

默认值：依赖于数据库安装部署时的配置

max_identifier_length

参数说明：报告当前系统允许的标识符最大长度。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：整型

默认值：63

server_encoding

参数说明：报告当前数据库的服务端编码字符集。

默认情况下，gs_initdb会根据当前的系统环境初始化此参数，通过locale命令可以查看当前的配置环境。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

默认值：在创建数据库的时候由当前系统环境决定的。

enable_upgrade_merge_lock_mode

参数说明：当该参数设置为on时，通过提升deltamerge内部实现的锁级别，避免和update/delete并发操作时的报错。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围： 布尔型

- on，提升deltamerge内部实现的锁级别，并发执行deltamerge和update/delete操作时，一个操作先执行，另一个操作被阻塞，在前一个操作完成后，后一个操作再执行。

- off，在对表的delta table的同一行并发执行deltamerge和update/delete操作时，后一个对同一行数据更新的操作会报错退出。

默认值：off

transparent_encrypt_kms_region

参数说明：该参数已废弃。它存储的是整个数据库的部署区域，内容要求不可出现RFC3986标准外的字符，最大长度2047字节。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

默认值：空

basebackup_timeout

参数说明：备份传输完成后连接无读写的超时时间。

通过gs_basebackup工具作传输时，如果指定较高压缩率时，可能在传输表空间完成后超时（客户端需要压缩传输数据）。

取值范围：整型，0 ~ INT_MAX，单位为秒。其中0表示禁用该功能。

默认值：600s

datanode_heartbeat_interval

参数说明：设置心跳线程间心跳消息发送时间间隔，建议值不超过wal_receiver_timeout / 2。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1000～60000（毫秒）

默认值：1s

max_concurrent_autonomous_transactions

参数说明：自治事务最大连接数，同一时间自治事务执行的最大并发数。当设置为0时，将无法执行自治事务。

该参数属于POSTMASTER类型参数，请参考表**表14-1**中对应设置方法进行设置。

取值范围：0-10000，理论最大值为10000，实际最大值为动态值，计算公式为“262143 - job_queue_processes - autovacuum_max_workers - max_inner_tool_connections - max_connections - AUXILIARY_BACKENDS - AV_LAUNCHER_PROCS”，**job_queue_processes**、**autovacuum_max_workers**、**max_inner_tool_connections**和**max_connections**的值取决于对应GUC参数的设置，AUXILIARY_BACKENDS为预留辅助线程数固定为20，AV_LAUNCHER_PROCS为预留autovacuum的launcher线程数固定为2。

默认值：

200（196核CPU/1536G内存，128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存）；150（96核CPU/768G内存）；120（80核CPU/640G内存）；

100（64核CPU/512G内存）；80（60核CPU/480G内存）；40（32核CPU/256G内存）；20（16核CPU/128G内存）；10（8核CPU/64G内存，4核CPU/32G内存，4核CPU/16G内存）

设置建议：根据实际业务需要和硬件配置设置此参数，建议不超过max_connections的1/10。若仅调大此参数，未同比例调整内存参数，业务压力大时，容易出现内存不足，报错提示“memory is temporarily unavailable”。

说明

若升级过程中涉及此参数范围变更，并且在commit前修改了此参数，则如果执行升级回滚，需要将此参数调整至升级前允许的范围，否则可能导致数据库无法启动。

enable_seqscan_fusion

参数说明：控制是否打开SeqScan优化

该参数属于SUSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示打开SeqScan优化
- off表示关闭SeqScan优化

默认值：off

说明

该参数只能优化EXPLAIN ANALYZE语句的seqscan算子的执行时间

cluster_run_mode

参数说明：双数据库实例容灾场景标识DN节点属于主数据库实例还是备数据库实例。单数据库实例使用默认值主数据库实例。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：枚举类型

- cluster_primary表示节点是主数据库实例的节点。
- cluster_standby表示节点是备数据库实例的节点。

默认值：cluster_primary

acceleration_with_compute_pool

参数说明：在查询包含OBS时，通过该参数决定查询是否通过计算资源池进行加速。（由于规格变更，当前版本已经不再支持本特性，请不要使用）

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示包含有OBS或的查询在计算资源池可用时，会根据代价评估决定是否通过计算资源池对查询加速。
- off表示任何查询都不会通过计算资源池进行加速。

默认值：off

dfs_partition_directory_length

参数说明：在HDFS文件系统上，构造HDFS VALUE分区表的分区目录时，目录名长度的上限值。

该参数属于USERSET类型参数，请参考表14-1中对应设置方法进行设置。

取值范围：92-7999

默认值：512

max_resource_package

参数说明：云上环境中，加速数据库实例每个DN可同时运行任务的线程数的上限。

该参数属于POSTMASTER类型参数，请参考表14-1中对应设置方法进行设置。

取值范围：0~2147483647

默认值：0

enable_gpi_auto_update

参数说明：控制在分区DDL命令中是否默认更新Global索引。

该参数属于USERSET类型参数，请参考表14-1中对应设置方法进行设置。

取值范围：布尔型

- on表示默认更新Global索引，此时分区DDL无论带不带UPDATE GLOBAL INDEX子句，都会更新Global索引。
- off表示默认不更新Global索引，此时只有当分区DDL带UPDATE GLOBAL INDEX子句，才会更新Global索引。

默认值：off

14.3.25 等待事件

enable_instr_track_wait

参数说明：是否开启等待事件信息实时收集功能。

在x86架构集中式部署下，硬件配置规格为32核CPU/256GB内存，使用Benchmark SQL 5.0工具测试性能，开关此参数性能影响约1.4%。

该参数属于SIGHUP类型参数，请参考表14-1中对应设置方法进行设置。

取值范围：布尔型

- on：表示打开等待事件信息收集功能。
- off：表示关闭等待事件信息收集功能。

默认值：on

14.3.26 Query

instr_unique_sql_count

参数说明：控制系统中unique sql信息实时收集功能。配置为0表示不启用unique sql信息收集功能。

该值由大变小将会清空系统中原有的数据重新统计（备机不支持此能力）；从小变大不受影响。

当系统中产生的unique sql信息(由dbe_perf.statement/dbe_perf.summary_statement统计)大于instr_unique_sql_count时，系统产生的unique sql信息不被统计。

在x86架构集中式部署下，硬件配置规格为32核CPU/256GB内存，使用Benchmark SQL 5.0工具测试性能，开关此参数性能影响约3%。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~2147483647

默认值：200000

instr_unique_sql_track_type

参数说明：unique sql记录SQL方式。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：枚举类型

- top：只记录顶层SQL。
- all：记录所有SQL。

默认值：top

enable_instr_rt_percentile

参数说明：是否开启计算系统中80%和95%的SQL响应时间的功能

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on：表示打开sql响应时间信息计算功能。
- off：表示关闭sql响应时间信息计算功能。

默认值：on

percentile

参数说明：sql响应时间百分比信息，后台计算线程根据设置的值计算相应的百分比信息。

该参数属于INTERNAL类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串。

默认值：80,95

instr_rt_percentile_interval

参数说明：sql响应时间信息计算间隔，sql响应时间信息计算功能打开后，后台计算线程每隔设置的时间进行一次计算。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0～3600（秒）。

默认值：10s

enable_instr_cpu_timer

参数说明：是否捕获sql执行的cpu时间消耗。

在x86架构集中式部署下，硬件配置规格为32核CPU/256GB内存，使用Benchmark SQL 5.0工具测试性能，开关此参数性能影响约3.5%。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on：表示捕获sql执行的cpu时间消耗。
- off：表示不捕获sql执行的cpu时间消耗。

默认值：on

enable_stmt_track

参数说明：控制是否启用Full /Slow SQL特性。

在x86架构集中式部署下，硬件配置规格为32核CPU/256GB内存，使用Benchmark SQL 5.0工具测试性能，开关此参数性能影响约1.2%。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on：表示开启Full /Slow SQL捕获
- off：表示关闭Full /Slow SQL捕获

默认值：on

track_stmt_parameter

参数说明：开启track_stmt_parameter后，在statement_history中记录的执行语句不再进行归一化操作，可以显示完整SQL语句信息，辅助DBA进行问题定位。其中对于简单查询，显示完整语句信息；对于PBE语句，显示完整语句信息的同时，追加每个变量数值信息，格式为“query string；parameters:$1=value1,$2=value2,...”。该参数提供目的是为用户呈现完整SQL信息，不受track_activity_query_size参数控制。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on：表示开启显示完整SQL语句信息的功能
- off：表示关闭显示完整SQL语句信息的功能

默认值：off

track_stmt_session_slot

参数说明：设置一个session缓存的最大的全量/慢SQL的数量，超过这个数量，新的语句执行将不会被跟踪，直到落盘线程将缓存语句落盘，留出空闲的空间。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0 ~ 2147483647

默认值：1000

track_stmt_details_size

参数说明：设置单语句可以收集的最大的执行事件的大小。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0 ~ 100000000，单位为byte。

默认值：4096

track_stmt_retention_time

参数说明：组合参数，控制全量/慢SQL记录的保留时间。以60秒为周期读取该参数，并执行清理超过保留时间的记录，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符型，该参数分为两部分，形式为'full sql retention time, slow sql retention time'

- full sql retention time为全量SQL保留时间，取值范围为0 ~ 86400，单位为秒。
- slow sql retention time为慢SQL的保留时间，取值范围为0 ~ 604800，单位为秒。

默认值：3600,604800

track_stmt_stat_level

参数说明：控制语句执行跟踪的级别。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置，不区分英文字母大小写。

取值范围：字符型

该参数分为两部分，形式为'full sql stat level, slow sql stat level'

第一部分为全量SQL跟踪级别，取值范围为OFF、L0、L1、L2

第二部分为慢SQL的跟踪级别，取值范围为OFF、L0、L1、L2

说明

若全量SQL跟踪级别值为非OFF时，当前SQL跟踪级别值为全量SQL和慢SQL的较高级别（L2 > L1 >L0），级别说明请参见《开发指南》的“系统表和系统视图 > 系统表 > STATEMENT_HISTORY”中的“STATEMENT_HISTORY字段”表格。

默认值：OFF,L0

enable_auto_clean_unique_sql

参数说明：当系统中产生的unique sql条目数量大于等于instr_unique_sql_count时，是否启用unique sql自动淘汰功能。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

默认值：off

⚠ 注意

由于快照有部分信息是来源于unique sql，所以开启自动淘汰的情况下，在生成wdr报告时，如果选择的起始快照和终止快照跨过了淘汰发生的时间，会导致无法生成wdr报告。

asp_log_directory

参数说明：asp_flush_mode设置为all或者file时，asp_log_directory决定存放服务器asp日志文件的目录。它可以是绝对路径，或者是相对路径（相对于数据目录的路径），仅sysadmin用户可以访问。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

当配置文件中asp_log_directory的值为非法路径时，会导致数据库实例无法重新启动。

说明

- 合法路径：用户对此路径有读写权限。
- 非法路径：用户对此路径无读写权限。

取值范围：字符串

默认值：安装时指定。

enable_slow_query_log（废弃）

参数说明：是否将慢查询信息写到日志文件中，在该版本中已废弃。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on：表示需要将慢查询信息写到日志文件中。
- off：表示不需要将慢查询信息写到日志文件中。

默认值：on

query_log_file（废弃）

参数说明：GUC参数enable_slow_query_log设置为ON，表示需要将慢查询记录写进日志文件中，query_log_file决定服务器慢查询日志文件的名称，仅sysadmin用户可以访问。通常日志文件名是按照strftime模式生成，因此可以用系统时间定义日志文件名，用%转义字符实现，在该版本中已废弃。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

建议使用%转义字符定义日志文件名称，否则难以对日志文件进行有效的管理。

取值范围：字符串

默认值：slow_query_log-%Y-%m-%d_%H%M%S.log

query_log_directory（废弃）

参数说明：enable_slow_query_log设置为on时，query_log_directory决定存放服务器慢查询日志文件的目录，仅sysadmin用户可以访问。它可以是绝对路径，或者是相对路径（相对于数据目录的路径），在该版本中已废弃。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

须知

当配置文件中query_log_directory的值为非法路径时，会导致数据库实例无法重新启动。

说明

合法路径：用户对此路径有读写权限
非法路径：用户对此路径无读写权限

取值范围：字符串

默认值：安装时指定。

perf_directory

参数说明：perf_directory决定性能视图打点任务输出文件的目录，仅sysadmin用户可以访问。它可以是绝对路径，或者是相对路径（相对于数据目录的路径）。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

说明

- 合法路径：用户对此路径有读写权限。
- 非法路径：用户对此路径无读写权限。

取值范围：字符串

默认值：安装时指定。

unique_sql_retention_time

参数说明：清理unique sql哈希表的间隔，默认为30分钟

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1~3650，单位为分钟。

默认值：30min

track_stmt_standby_chain_size

参数说明：组合参数，控制备机快/慢SQL记录的最大占用内存与磁盘空间。仅SysAdmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符型

该参数分为四部分，形式为'Full SQL memory size, Full SQL disk size, Slow SQL memory size, Slow SQL disk size' 。

Full SQL与Slow SQL分开存放于不同位置，因此额外使用了四个值进行控制。

- Full SQL memory size 为保留的快SQL的最大内存占用空间，取值范围为 [16, 1024]，单位为MB。
- Full SQL disk size 为保留的快SQL的最大磁盘占用空间，取值范围为 [512, 1048576]，单位为MB。
- Slow SQL memory size 为保留的慢SQL的最大内存占用空间，取值范围为 [16, 1024]，单位为MB。
- Slow SQL disk size 为保留的慢SQL的最大磁盘占用空间，取值范围为 [512, 1048576]，单位为MB。

其中内存值不可大于磁盘值。

默认值：32, 1024, 16, 512

14.3.27 系统性能快照

enable_wdr_snapshot

参数说明：是否开启数据库监控快照功能。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on：打开数据库监控快照功能。
- off：关闭数据库监控快照功能。

默认值：on

enable_wdr_snapshot_standby

参数说明：是否开启备机支持数据库监控快照功能。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on：打开备机支持数据库监控快照功能。
- off：关闭备机支持数据库监控快照功能。

默认值：off

enable_show_standby_name

参数说明：是否开启显示备机的名字，该名字区分主备节点。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on：打开视图区分主备机名字功能。
- off：关闭视图区分主备机名字功能。

默认值：off

wdr_snapshot_retention_days

参数说明：系统中数据库监控快照数据的保留天数。当数据库运行过程期间所生成的快照量数超过保留天数内允许生成的快照数量的最大值时，系统将每隔wdr_snapshot_interval时间间隔，清理snapshot_id最小的快照数据。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1～30。

默认值：8

wdr_snapshot_query_timeout

参数说明：系统执行数据库监控快照操作时，设置快照操作相关的sql语句的执行超时时间。如果语句超过设置的时间没有执行完并返回结果，则本次快照操作失败。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置，0代表该参数不生效。

取值范围：整型，0～INT_MAX（秒）。

默认值：100s

wdr_snapshot_interval

参数说明：后台线程Snapshot自动对数据库监控数据执行快照操作的时间间隔。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，10～60（分钟）。

默认值：1h

asp_flush_mode

参数说明：ASP刷新到磁盘上的方式分为写文件和写系统表，当为‘file’时，默认写文件，为‘table’时写系统表，为‘all’时，即写文件也写系统表，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，‘table’、‘file’、‘all’。

默认值：‘table’

asp_flush_rate

参数说明：当内存中样本个数达到asp_sample_num时，会按一定比例把内存中样本刷新到磁盘上，asp_flush_rate为刷新比例。该参数为10时表示按10：1进行刷新。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1~10。

默认值：10

asp_log_filename

参数说明：当ASP写文件时，该参数设置文件名的格式，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串。

默认值："asp-%Y-%m-%d_%H%M%S.log"

asp_retention_days

参数说明：当ASP样本写到系统表时，该参数表示保留的最大天数。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1~7。

默认值：2

asp_sample_interval

参数说明：每次采样的间隔。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1~10(秒)。

默认值：1s

asp_sample_num

参数说明：LOCAL_ACTIVE_SESSION视图最大的样本个数，仅sysadmin用户可以访问。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，10～100000。

默认值：

100000（196核CPU/1536G内存，128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，64核CPU/512G内存，60核CPU/480G内存，32核CPU/256G内存，16核CPU/128G内存，8核CPU/64G内存，4核CPU/32G内存）；36000（4核CPU/16G内存）

enable_asp

参数说明：是否开启活跃会话信息active session profile。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on：打开active session profile功能。
- off：关闭active session profile功能。

默认值：on

14.3.28 安全配置

enable_security_policy

参数说明：安全策略开关，控制统一审计和数据动态脱敏策略是否生效。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型。

on：安全策略开关打开。

off：安全策略开关关闭。

默认值：off

use_elastic_search

参数说明：使能统一审计发送日志至Elastic Search系统, enable_security_policy打开且本参数打开后，统一审计日志会通过http(https)传递至Elastic Search系统（默认使用https安全协议）。此参数打开后需要保证elastic_search_ip_addr对应的es服务可正常连通，否则进程启动失败。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型。

on：使能统一审计日志发送至Elastic Search。

off：关闭统一审计日志发送至Elastic Search。

默认值：off

elastic_search_ip_addr

参数说明：Elastic Search系统IP地址 ，使用https协议格式为：https://ip:port:username；使用http协议格式为：http://ip:port。其中，ip为Elastic Search服

务器的IP，port为Elastic Search HTTP通信的侦听端口，范围为9200 - 9299，username为用户在Elastic Search注册账号所使用的用户名，初始用户为elastic使用https协议需要配置相关证书，详见《安全加固指南》中“统一审计“章节。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串。

默认值：''

is_sysadmin

参数说明：表示当前用户是否是初始用户。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：布尔型。

on表示是初始用户。

off表示不是初始用户。

默认值：off

block_encryption_mode

参数说明：aes_encrypt和aes_decrypt函数进行加解密时使用的块加密模式。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：枚举类型，有效值为aes-128-cbc, aes-192-cbc，aes-256-cbc，aes-128-cfb1，aes-192-cfb1，aes-256-cfb1，aes-128-cfb8，aes-192-cfb8，aes-256-cfb8，aes-128-cfb128，aes-192-cfb128，aes-256-cfb128，aes-128-ofb，aes-192-ofb，aes-256-ofb。其中aes表示加/解密算法，128/192/256表示密钥长度（单位：bit），cbc/cfb1/cfb8/cfb128/ofb表示块加/解密模式。

默认值：aes-128-cbc

14.3.29 全局临时表

max_active_global_temporary_table

参数说明：全局临时表功能开关，控制是否可以创建全局临时表，当前Ustore存储引擎不支持全局临时表。该参数的取值决定了共享缓存中预留给全局临时表所需的哈希表的内存使用，并不会强制限制所有会话中的活跃全局临时表总数。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0 ~ 1000000

- 0：全局临时表功能关闭。
- > 0：全局临时表功能打开。

默认值：1000

vacuum_gtt_defer_check_age

参数说明：vacuum执行后检查全局临时表relfrozenxid与普通表的差异。如果全局临时表relfrozenxid落后超过指定参数值，就产生WARNING。一般不用修改。

该参数USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0 ~ 1000000

默认值：10000

enable_gtt_concurrent_truncate

参数说明：是否支持全局临时表truncate table和DML的并发执行，以及全局临时表truncate table和truncate table的并发执行。

该参数SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on/true表示支持上述操作并发。
- off/false表示不支持上述操作并发。

默认值：on

14.3.30 HyperLogLog

hll_default_log2m

参数说明：该参数可以指定hll数据结构桶的个数。桶的个数会影响hll计算distinct值的精度，桶的个数越多，误差越小。误差范围为：$[-1.04/2^{log2m*1/2},+1.04/2^{log2m*1/2}]$。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，10~16。

默认值：14

hll_default_log2explicit

参数说明：该参数可以用来设置从Explicit模式到Sparse模式的默认阈值大小。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~12。0表示跳过Explicit模式，取1-12表示在基数到达$2^{hll_default_log2explicit}$时切换模式。

默认值：10

hll_default_log2sparse

参数说明：该参数可以用来设置从Sparse模式到Full模式的默认阈值大小。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~14。0表示跳过Explicit模式，取1-14表示在基数到达$2^{hll_default_log2sparse}$时切换模式。

默认值：12

hll_duplicate_check

参数说明：该参数可以用来指定是否默认开启duplicatecheck。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：0，1。0表示默认关闭，1表示默认开启

默认值：0

hll_default_regwidth（废弃）

参数说明：该参数可以指定hll数据结构每个桶的位数，该值越大，hll所占内存越高。hll_default_regwidth和hll_default_log2m可以决定当前hll能够计算的最大distinct value。当前regwidth设为固定值，该参数不再使用。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1~5。

默认值：5

hll_default_expthresh（废弃）

参数说明：该参数可以用来设置从Explicit模式到Sparse模式的默认阈值大小。当前已经使用参数hll_default_log2explicit替代类似功能。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，-1~7。-1表示自动模式，0表示跳过Explicit模式，取1-7表示在基数到达$2^{hll_default_expthresh}$时切换模式。

默认值：-1

hll_default_sparseon（废弃）

参数说明：该参数可用来指定是否默认开启Sparse模式。当前已经使用参数hll_default_log2sparse替代类似功能，hll_default_log2sparse设置为0时关闭Sparse模式。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：0，1。0表示默认关闭，1表示默认开启。

默认值：1

hll_max_sparse（废弃）

参数说明：该参数可以用来指定max_sparse的大小。当前已经使用参数hll_default_log2sparse替代类似功能。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，-1~2147483647

默认值：-1

enable_compress_hll（废弃）

参数说明：该参数可以用来指定是否对hll开启内存优化模式。目前hll内存已经进行了优化设计，该参数不再使用。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on/true表示对hll开启内存优化模式。
- off/false表示不开启内存优化模式。

默认值：off

14.3.31 用户自定义函数

udf_memory_limit

参数说明：控制每个数据库节点执行UDF时可用的最大物理内存量。本参数当前版本不生效，请使用FencedUDFMemoryLimit和UDFWorkerMemHardLimit参数控制fenced udf worker虚存。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，200*1024～max_process_memory，单位为KB。

默认值：200MB

FencedUDFMemoryLimit

参数说明：控制每个fenced udf worker进程使用的虚拟内存。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整数，0 ~ 2147483647，单位为KB，设置可带单位（KB，MB，GB）。其中0表示不做内存控制。

默认值：0

UDFWorkerMemHardLimit

参数说明：控制fencedUDFMemoryLimit的最大值。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整数，0 ~ 2147483647，单位为KB，设置时可带单位（KB，MB，GB）。

默认值：1GB

14.3.32 定时任务

job_queue_processes

参数说明：表示系统可以并发执行的job数目。该参数为postmaster级别，通过gs_guc设置，需要重启gaussdb才能生效。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：0～1000

功能：

- 当job_queue_processes设置为0值，表示不启用定时任务功能，任何job都不会被执行（因为开启定时任务的功能会对系统的性能有影响，有些局点可能不需要定时任务的功能，可以通过设置为0不启用定时任务功能）。
- 当job_queue_processes为大于0时，表示启用定时任务功能且系统能够并发处理的最大任务数。

启用定时任务功能后，job_scheduler线程会在定时时间间隔轮询pg_job系统表，系统设置定时任务检查周期默认为1s。

由于并行运行的任务数太多会消耗更多的系统资源，因此需要设置系统并发处理的任务数，当前并发的任务数达到job_queue_processes时，且此时又有任务到期，那么这些任务本次得不到执行而延期到下一轮询周期。因此，建议用户需要根据每个任务的执行时长合理地设置任务的时间间隔（即submit接口中的interval参数），来避免由于任务执行时间太长而导致下个轮询周期无法正常执行。

注：如果同一时间内并行的job数很多，过小的参数值会导致job等待。而过大的参数值则消耗更多的系统资源，建议设置此参数为100，用户可以根据系统资源情况合理调整。

默认值：10

enable_prevent_job_task_startup

参数说明：控制是否启动job线程。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示不能启动job线程。
- off表示可以启动job线程。

默认值：off

14.3.33 线程池

线程池技术的整体设计思想是线程资源池化、并且在不同连接之间复用。系统在启动之后会根据当前核数或者用户配置启动固定一批数量的工作线程，一个工作线程会服务一到多个连接session，这样把session和thread进行了解耦。

enable_thread_pool

参数说明：控制是否使用线程池功能。该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启线程池功能。
- off表示不开启线程池功能。

默认值：on

thread_pool_attr

参数说明：用于控制线程池功能的详细属性，该参数仅在enable_thread_pool打开后生效，仅sysadmin用户可以访问。该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，长度大于0

该参数分为3个部分，'thread_num, group_num, cpubind_info'，这3个部分的具体含义如下：

- thread_num：线程池中的线程总数，取值范围是0~4096。其中0的含义是数据库根据系统CPU core的数量来自动配置线程池的线程数，如果参数值大于0，线程池中的线程数等于thread_num。线程池大小推荐根据硬件配置设置，计算公式如下：thread_num = CPU核数*3~5，thread_num最大值为4096。
- group_num：线程池中的线程分组个数，取值范围是0~64。其中0的含义是数据库根据系统NUMA组的个数来自动配置线程池的线程分组个数，如果参数值大于0，线程池中的线程组个数等于group_num。
- cpubind_info：线程池是否绑核的配置参数。可选择的配置方式有几种：1. '(nobind)' ，线程不做绑核；2. '(allbind)'，利用当前系统所有能查询到的CPU core做线程绑核；3. '(nodebind: 1, 2)'，利用NUMA组1，2中的CPU core进行绑核；4. '(cpubind: 0-30)'，利用0-30号CPU core进行绑核；5. '(numabind: 0-30)'，在NUMA组内利用0-30号CPU core进行绑核。该参数不区分大小写。当开启资源多租模式时，该参数不生效。

默认值：

'4096,2,(nobind)'（196核CPU/1536G内存，128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存）；'2048,2,(nobind)'（96核CPU/768G内存，80核CPU/640G内存）；'1024,2,(nobind)'（64核CPU/512G内存，60核CPU/480G内存，32核CPU/256G内存）；'512,2,(nobind)'（16核CPU/128G内存）；'256,2,(nobind)'（8核CPU/64G内存）；'128,2,(nobind)'（4核CPU/32G内存）；'32,1,(nobind)'（4核CPU/16G内存）

thread_pool_stream_attr

参数说明：用于控制stream线程池功能的详细属性，stream线程只在DN生效，该参数仅在enable_thread_pool打开后生效，仅sysadmin用户可以访问。该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，长度大于0

该参数分为4个部分，'stream_thread_num,
stream_proc_ratio ,group_num ,cpubind_info'，这4个部分的具体含义如下：

- stream_thread_num：stream线程池中的线程总数，取值范围是0~4096。其中0的含义是数据库根据系统CPU core的数量来自动配置线程池的线程数，如果参数值大于0，线程池中的线程数等于stream_thread_num。线程池大小推荐根据硬件配置设置，计算公式如下：stream_thread_num = CPU核数*3~5，stream_thread_num最大值为4096。
- stream_proc_ratio：预留给stream线程的proc数量比例，浮点类型，默认为0.2，预留proc计算方式为：stream_proc_ratio * stream_thread_num。
- group_num：线程池中的线程分组个数，取值范围是0~64。其中0的含义是数据库根据系统NUMA组的个数来自动配置线程池的线程分组个数，如果参数值大于0，线程池中的线程组个数等于group_num。thread_pool_stream_attr的

group_num需与thread_pool_attr的group_num配置和使用保持一致，若设置为不同值，以thread_pool_attr的group_num为准。

- cpubind_info：线程池是否绑核的配置参数。可选择的配置方式有几种：1. '(nobind)'，线程不做绑核；2. '(allbind)'，利用当前系统所有能查询到的CPU core做线程绑核；3. '(nodebind: 1, 2)'，利用NUMA组1,2中的CPU core进行绑核；4. '(cpubind: 0-30)'，利用0-30号CPU core进行绑核；5. '(numabind: 0-30)'，在NUMA组内利用0-30号CPU core进行绑核。该参数不区分大小写。thread_pool_stream_attr的cpubind_info需与thread_pool_attr的cpubind_info配置和使用保持一致，若设置为不同值，以thread_pool_attr的cpubind_info为准。

默认值：

stream_thread_num：16

stream_proc_ratio：0.2

group_num、cpubind_info：参见**thread_pool_attr**。

resilience_threadpool_reject_cond

参数说明：用于控制线程池过载逃生的线程池使用率比例。该参数仅在GUC参数use_workload_manager和enable_thread_pool打开时生效。该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，长度大于0

该参数分为recover_threadpool_percent、overload_threadpool_percent 2部分，这2个部分的具体含义如下：

- recover_threadpool_percent：线程池恢复正常状态时的线程池使用率，当线程池使用率小于该值时，停止过载逃生并放开新连接接入，取值为0~INT_MAX，设置为多少表示百分之多少。
- overload_threadpool_percent：线程池过载时的线程池使用率，当线程池使用率大于该值时，表示当前线程池已经过载，触发过载逃生kill会话并禁止新连接接入，取值为0~INT_MAX，设置为多少表示百分之多少。

默认值：'0,0'，表示关闭线程池逃生功能。

示例：

```
resilience_threadpool_reject_cond = '50,90'
```

表示线程池使用率超过90%后禁止新连接接入并kill堆积的会话，kill会话过程中线程池使用率下降到50%时停止kill会话并允许新连接接入。

须知

- 线程池使用率可以通过DBE_PERF.local_threadpool_status视图查询获得；线程池设置的初试线程池线程数目可以通过查询thread_pool_attr参数获得。
- 该参数如果设置的百分比过小，则会频繁触发线程池过载逃生流程，会使正在执行的会话被强制退出，新连接短时间接入失败，需要根据实际线程池使用情况慎重设置。
- recover_threadpool_percent和overload_threadpool_percent的值可以同时为0，除此之外，recover_threadpool_percent的值必须要小于overload_threadpool_percent，否则会设置不生效。

14.3.34 备份恢复

operation_mode

参数说明：表示系统进入备份恢复模式。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示在备份恢复过程中。
- off表示不在备份恢复过程中。

默认值：off

enable_cbm_tracking

参数说明：当使用roach执行数据库实例的全量和增量备份时需要开启此参数，如果关闭会导致备份失败。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示追踪功能开启。
- off表示追踪功能关闭。

默认值：off

max_size_for_xlog_retention

参数说明：用于控制何时触发备份复制槽的强制推进，以避免由于备份操作过程中日志无法回收导致磁盘满、实例只读等影响。该参数实际设置值建议比cm_server组件的datastorage_threshold_value_check略小一点，以避免实例进入只读状态。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：-100 ~ 2147483647

- 0表示关闭该功能。
- 负值表示按磁盘阈值触发，如-80，表示当磁盘阈值超过80%，且日志回收是由于备份操作被阻塞，那么会触发备份复制槽的强制推进。
- 正值表示按日志积压大小触发，如32，表示当备份复制槽落后当前检查点redo位置超过32段日志大小（每段日志大小为16MB），且日志回收是由于备份操作被阻塞，那么会触发备份复制槽的强制推进。

默认值：0

max_cbm_retention_time

参数说明：用于控制何时触发备份CBM文件的强制回收，以避免由于备份操作过程中CBM文件无法回收导致磁盘满、实例只读等影响。该参数实际设置值建议按照全备的时间间隔设置。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：86400 ~ 2147483647

- 时间单位为秒。
- 最小值为1天。
- 默认值为2周。

默认值：1209600

14.3.35 Undo

undo_space_limit_size

参数说明：用于控制undo强制回收阈值，达到阈值的80%启动强制回收。建议设置undo_space_limit_size参数不小于undo_limit_size_per_transaction参数。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，102400~2147483647，单位为8KB。

默认值：256GB

undo_limit_size_per_transaction

参数说明：用于控制单事务undo分配空间阈值，达到阈值时事务报错回滚。建议设置undo_limit_size_per_transaction参数不大于undo_space_limit_size参数。若设置的undo_limit_size_per_transaction参数大于undo_space_limit_size参数，用户调用show undo_limit_size_per_transaction命令查询参数值时，显示出来的值和用户设置的值仍保持一致，只是在使用时会取undo_space_limit_size和undo_limit_size_per_transaction两者的较小值，作为实际的单事务undo分配空间阈值。如果设置 undo_limit_size_per_transaction 超过 1TB，可能会影响系统的性能和稳定性。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，256~2147483647，单位为8KB。

默认值：32GB

14.3.36 DCF 参数设置

enable_dcf

参数说明：是否开启DCF模式。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型，on/off。on表示当前日志复制模式为DCF模式，off表示当前日志复制模式为非DCF模式。

默认值：off

dcf_ssl

参数说明：此参数不再使用，DCF复用GUC参数**ssl**。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型，on/off。on表示使用SSL，off表示不使用SSL。

默认值：on

dcf_config

参数说明：用户安装时自定义配置信息。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

默认值：字符串，安装时用户自定义配置

dcf_data_path

参数说明：DCF数据路径。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

默认值：字符串，DN数据目录下的dcf_data目录

dcf_log_path

参数说明：DCF日志路径。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

默认值：字符串，DN数据目录下的dcf_log目录

dcf_node_id

参数说明：DCF所在DN节点ID，用户安装和模式切换时自定义。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

默认值：整型，安装时用户自定义配置

dcf_max_workers

参数说明：DCF回调函数线程个数。如果节点数量超过7个，需要增加这个参数的数值（比如增加到40），否则可能会出现主节点一直处于promoting状态，主备节点日志不推进的状态。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，10~262143

默认值：20

dcf_truncate_threshold

参数说明：DN对DCF日志进行truncate的门限阈值。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1~2147483647

默认值：100000

dcf_election_timeout

参数说明：DCF leader和follower选举超时时间。选举超时时间数值依赖于当前DN之间的网络状况，在超时时间较小且网络极差的情形下，会有超时选举发生，待网络恢复选举恢复正常。建议根据当前网络状态合理设置超时时间。对DCF节点时钟的约束：DCF节点间最大时钟差异小于选举超时时间的一半。在DCF手动选举模式下，为了不影响CM及时仲裁选举，禁止对该参数配置修改，按默认选举超时时间设置即可。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，单位s，1~600

默认值：3

dcf_enable_auto_election_priority

参数说明：DCF优先级选主是否允许内部自动调整优先级值。0表示不允许，1表示允许内部自动调整。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~1

默认值：1

dcf_election_switch_threshold

参数说明：DCF防频繁切主门限。推荐根据用户业务可接受的最大故障时间配置。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，单位s，0~2147483647

默认值：0

dcf_run_mode

参数说明：DCF选举模式，0表示自动选举模式，1表示手动选举模式，2表示去使能选举模式。目前去使能选举模式只限定少数派恢复场景使用，修改会导致数据库实例不可用。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

须知：实例在正常状态下进行工作模式切换才能保证切换后工作正常。GUC参数设置和cm_ctl设置的DCF工作模式需要保持一致，即两者需要同步设置为DCF手动或自动模式。

例如，设置DCF手动模式如下：

```
cm_ctl set --param --server -k dn_arbitrate_mode=quorum
cm_ctl reload --param --server
gs_guc reload -Z datanode -I all -N all  -c "dcf_run_mode=1"
```

设置DCF自动模式如下：

```
cm_ctl set --param --server -k dn_arbitrate_mode=paxos
cm_ctl reload --param --server
gs_guc reload -Z datanode -I all -N all  -c "dcf_run_mode=0"
```

取值范围：枚举类型，0、1、2

默认值：1

dcf_log_level

参数说明：DCF日志级别。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

- **关闭日志**：“NONE”，NONE表示关闭日志打印，不能与以下日志级别混合使用。
- **开启日志：**“RUN_ERR|RUN_WAR|RUN_INF|DEBUG_ERR|DEBUG_WAR|DEBUG_INF|TRACE|PROFILE|OPER”

  日志级别可以从上述字符串中选取字符串并使用竖线组合使用，不能配置空串。

默认值：“RUN_ERR|RUN_WAR|DEBUG_ERR|OPER|RUN_INF|PROFILE”

dcf_log_backup_file_count

参数说明：DCF运行日志备份保留个数。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1~1000

默认值：100

dcf_max_log_file_size

参数说明：DCF运行日志单个文件最大大小。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，单位MB，1~1000

默认值：10

dcf_socket_timeout

参数说明：DCF通信模块连接socket超时时间，参数重启生效。对于网络环境比较差的环境，若配置很小的超时时间，可能会导致建链不成功，此时需要适当增大此值。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，单位ms，10~600000

默认值：5000

dcf_connect_timeout

参数说明：DCF通信模块建立连接超时时间，参数重启生效。对于网络环境比较差的环境，若配置很小的超时时间，可能会导致建链不成功，此时需要适当增大此值。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，单位ms，10~600000

默认值：60000

dcf_mec_fragment_size

参数说明：DCF通信模块fragment大小，参数重启生效。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，单位KB，32~10240

默认值：64

dcf_stg_pool_max_size

参数说明：DCF存储模内存池最大值，参数重启生效。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，单位MB，32~2147483647

默认值：2048

dcf_stg_pool_init_size

参数说明：DCF存储模块内存池最小值，参数重启生效。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，单位MB，32~2147483647

默认值：32

dcf_mec_pool_max_size

参数说明：DCF通信模块内存池最大值，参数重启生效。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，单位MB，32~2147483647

默认值：200

dcf_flow_control_disk_rawait_threshold

参数说明：DCF流控功能的磁盘等待阈值。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，单位us，0~2147483647

默认值：100000

dcf_flow_control_net_queue_message_num_threshold

参数说明：DCF流控功能的网络队列消息数阈值。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~2147483647

默认值：1024

dcf_flow_control_cpu_threshold

参数说明：DCF CPU流控阈值。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，单位百分比，0~2147483647

默认值：100

dcf_mec_batch_size

参数说明：DCF通信批量消息数，数值为0时，DCF会根据网络以及写入数据量自适应调整，参数重启生效。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~1024

默认值：0

dcf_mem_pool_max_size

参数说明：DCF内存最大值，参数重启生效。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，单位MB，32~2147483647

默认值：2048

dcf_mem_pool_init_size

参数说明：DCF内存初始化大小，参数重启生效。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，单位MB，32~2147483647

默认值：32

dcf_compress_algorithm

参数说明：DCF运行日志传输压缩算法，参数重启生效。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型

- 0表示不压缩
- 1表示ZSTD压缩算法
- 2表示LZ4压缩算法

默认值：0

dcf_compress_level

参数说明：DCF日志传输压缩级别，参数重启生效，此参数生效前提必须配置有效的压缩算法，即设置合法的dcf_compress_algorithm参数。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1~22

若不开启压缩，配置的压缩级别将不生效。

默认值：1

dcf_mec_channel_num

参数说明：DCF通信通道数量，参数重启生效。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1~64

默认值：1

dcf_rep_append_thread_num

参数说明：DCF日志复制线程数量，参数重启生效。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1~1000

默认值：2

dcf_mec_agent_thread_num

参数说明：DCF通信工作线程数量，参数重启生效。dcf_mec_agent_thread_num值建议不少于2*节点数*dcf_mec_channel_num。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1~1000

默认值：10

dcf_mec_reactor_thread_num

参数说明：DCF使用reactor线程数量，参数重启生效。dcf_mec_reactor_thread_num与dcf_mec_agent_thread_num比例建议1：40。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1~100

默认值：1

dcf_log_file_permission

参数说明：DCF运行日志文件属性，参数重启生效，参数安装阶段配置，后续不支持修改。若用户需要支持同组的其他用户访问日志，首先需要所有的父目录都支持同组的其他用户也能访问。即若参数dcf_log_path_permission配置为750，dcf_log_file_permission只能为600或者640。若参数dcf_log_path_permission配置为700，dcf_log_file_permission只能为600。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：枚举型，600、640

默认值：600

dcf_log_path_permission

参数说明：DCF运行日志目录属性，参数重启生效，参数安装阶段配置，后续不支持修改。若用户需要支持同组的其他用户访问日志路径，需选择参数750，否则选择700。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：枚举型，700、750

默认值：700

dcf_majority_groups

参数说明：DCF策略化多数派功能设置。对于需要配置此参数的group，该group内至少有一台备机收到日志。即该group内存在一台同步备机。若对DCF实例内做了增删节点或者对实例内节点group值进行了调整修改，需同步修改此配置。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串

- 关闭：“”，空字符串表示策略化多数派功能关闭。
- 开启： 配置有效的group值，使用逗号分隔，group值需在dcf_config中存在。例如将group值分别为1和2，加入DCF的策略化多数派配置时，可以设置为"1,2"；若配置了dcf_config中不存在的group值或者其他字符，DCF将认为该配置的group无效。

默认值：空字符串

⚠ 注意

若配置了参数后某一group内所有节点均故障，在对其中某个节点做涉及节点build相关操作（节点修复、不换ip的节点替换）时，需要将该group从此参数列表中移除，待节点恢复正常后可将该group再次配置到此参数。

dcf_node_id_map

参数说明：DN备机名称与DCF node_id映射字典，参数重启生效，参数安装阶段配置，后续不支持修改。在DCF集群安装、升级、节点替换场景会涉及使用此参数。GUC参数synchronous_standby_names中配置的standby_name需包含在此字典内。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串。配置格式例如:
'stanby_name1:dcf_node_id1,standy_name2:dcf_node_id2'，DN备机名称对应的DCF node_id数值，使用逗号分隔。

默认值：空字符串

14.3.37 闪回相关参数

本章节介绍闪回功能相关参数。本版本只支持Ustore引擎闪回功能，不再支持Astore引擎闪回功能。

enable_recyclebin

参数说明：用来控制回收站的实时打开和关闭。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

默认值：off

recyclebin_retention_time

参数说明：设置回收站对象保留时间，超过该时间的回收站对象将被自动清理。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，单位为s，最小值为1，最大值为2147483647。

默认值：15min（即900s）

version_retention_age

参数说明：设置旧版本保留的事务数，超过该事务数的旧版本将被回收清理。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0～576460752303423487，值为0表示不延迟。

默认值：0

注意

该参数已弃用。

vacuum_defer_cleanup_age

参数说明：指定VACUUM使用的事务数，VACUUM会延迟清除无效的行存表记录，延迟的事务个数通过vacuum_defer_cleanup_age进行设置。即VACUUM和VACUUM FULL操作不会立即清理刚刚被删除元组。也可以通过设置该参数，配置闪回功能旧版本保留期限。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0～1000000，值为0表示不延迟。取值范围需要扩展到1亿。

默认值：0

> **⚠ 注意**
>
> 在进行Ustore闪回时，无需关注该参数。其服务于之前版本的astore闪回功能，同时具有其他用途。本版本闪回功能已不使用。

undo_retention_time

参数说明：设置undo旧版本保留时间。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，单位为s，最小值为0，最大值为259200。

默认值：0

> **⚠ 注意**
>
> 1. 在进行Ustore闪回查询时，如果中途设置该参数为0，则会清理闪回点快照信息，之前的任何版本不允许再做闪回查询。执行闪回查询会报错："cannot find the restore point"。
> 2. 如果想要保留的undo记录旧版本时间为time1，闪回查询执行的SQL时间为time2，需要设置参数undo_retention_time大于两者之和。即设置undo_retention_time > time1 + time2 + 3s。建议设置 undo_retention_time = time1 + 1.5 * time2。例如：想要保留3h的旧版本，闪回查询执行时间为1h，则undo_retention_time = 3h + 1.5 * 1h = 4.5h。

14.3.38 回滚相关参数

max_undo_workers

参数说明：异步回滚调用的undoworker线程数量，参数重启生效。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1~100

默认值：5

14.3.39 预留参数

> **说明**
>
> 下列参数为预留参数，该版本不生效。

acce_min_datasize_per_thread

cstore_insert_mode

enable_constraint_optimization

enable_hadoop_env

enable_hdfs_predicate_pushdown

enable_orc_cache

schedule_splits_threshold

backend_version

undo_zone_count

version_retention_age

max_que_size

废弃参数

max_query_retry_times

14.3.40 AI 特性

enable_hypo_index

参数说明：该参数控制数据库的优化器进行EXPLAIN时是否考虑创建的虚拟索引。通过对特定的查询语句执行explain，用户可根据优化器给出的执行计划评估该索引是否能够提升该查询语句的执行效率。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示在进行EXPLAIN时创建虚拟索引。
- off表示在进行EXPLAIN时不创建虚拟索引。

默认值：off

db4ai_snapshot_mode

参数说明：snapshot有2种模式：MSS（物化模式，存储数据实体）和CSS（计算模式，存储增量信息）。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，MSS/CSS

- MSS表示物化模式，db4ai在创建快照的时候存储数据实体。
- CSS表示计算模式，db4ai在创建快照的时候存储增量信息。

默认值：MSS

db4ai_snapshot_version_delimiter

参数说明：该参数为数据表快照版本分隔符。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，长度等于1，且'!'是非法字符。

默认值：@

db4ai_snapshot_version_separator

参数说明：该参数用于指定数据表快照子版本分隔符。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，长度等于1，且'?'是非法字符。

默认值：.

enable_ai_stats

参数说明：该参数用于指定是否创建或者使用智能统计信息。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

默认值：on

multi_stats_type

参数说明：该参数用于指定在enable_ai_stats为on状态下创建的统计信息类别。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：枚举类型，有效值为"BAYESNET"、"MCV"、"ALL"。

- "BAYESNET"：只创建智能统计信息。
- "MCV"：只创建传统统计信息。
- "ALL"：同时创建传统统计信息和智能统计信息。

默认值："BAYESNET"

ai_stats_cache_limit

参数说明：该参数用于指定在enable_ai_stats为on状态下最多缓存的模型数量。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，30~1000

默认值：100

enable_operator_prefer

参数说明：该参数用于指定是否开启算子倾向性规则，在估计代价相近的情况下，倾向于选择参数化路径执行表连接。注意：此参数生效有两个必要的前置条件：1、参数化路径被生成；2、参数化路估计的代价和其他索引扫描算子类似。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

默认值：off

enable_cachedplan_mgr

参数说明：该参数用于指定是否开启自适应计划选择功能。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

默认值：on

max_stmt_aplan_num

参数说明：该参数用于控制自适应计划选择每个查询的候选计划个数上限。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~20

默认值：5

recommend_session_aplan_memory

参数说明：该参数用于控制自适应计划选择每个session中的候选计划内存上限，大于或者等于这个值之后将不再在内存中插入新的候选计划，单位KB

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，1024~102400

默认值：5120

repick_plan_min_duration

参数说明：该参数用于控制探测到的计划的可用下限，在策略探测时，如果被探测策略的执行时间不小于cplan的repick_plan_min_duration倍，将直接报错。注意设置为0的时候表示关闭报错机制。

该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，0~INT_MAX

默认值：0

unix_socket_directory

参数说明：用于指定unix_socket通信方式中，文件存放的路径。此参数只能在配置文件postgresql.conf中指定。在启动fenced模式前需要设定该GUC参数。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：字符串，长度大于等于0

默认值：''

enable_ai_watchdog

参数说明：开启或关闭AI Watchdog功能。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on：表示开启。

- off：表示关闭。

默认值：on

enable_ai_watchdog_forcible_oom_detection

参数说明：强制开启或关闭AI Watchdog的OOM探测功能，若关闭该参数，则会自动根据当前数据库的规格判断是否需要启动OOM探测功能。自动判断模式下，对于max_process_memory 设置为64GB及以上的场景，才会启动OOM探测功能。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on：表示开启。
- off：表示关闭。

默认值：off

enable_ai_watchdog_healing

参数说明：开启或关闭AI Watchdog的自愈功能。16U及以下规格，高负载下容易造成假死，不建议开启。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on：表示开启。
- off：表示关闭。

默认值：on

ai_watchdog_max_cpu_usage

参数说明：预期的数据库CPU使用率上限，该值会根据多核情况进行归一化。该参数值设置为0时，表示不判断CPU使用率情况。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：浮点型，大于等于0，小于等于1。

默认值：0.8

ai_watchdog_oom_dynamic_used_threshold

参数说明：预期的数据库动态内存使用率上限。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：浮点型，大于等于0，小于等于1。

默认值：0.95

ai_watchdog_oom_growth_confidence

参数说明：OOM检测算法置信度。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：浮点型，大于等于0.1，小于等于1。

默认值：0.95

ai_watchdog_oom_malloc_failures

参数说明：容忍的最大连续内存分配失败数量，超过该数量可能会触发OOM探测功能。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，大于等于1，小于等于32000。

默认值：50

ai_watchdog_oom_other_used_memory_threshold

参数说明：预期的数据库的其他部分内存使用上限，单位是MB。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，大于等于1，小于等于32000。

默认值：20480

ai_watchdog_oom_process_threshold

参数说明：预期的数据库进程使用占max_process_memory的使用比例，到达该阈值时，会触发内存泄漏判断；该值可以超过1。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：浮点型，大于等于0，小于等于10。

默认值：1.1

ai_watchdog_oom_shared_threshold

参数说明：预期的数据库共享内存使用比例上限。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：浮点型，大于等于0，小于等于1。

默认值：0.4

ai_watchdog_rto_restriction_time

参数说明：AI Watchdog自愈功能的RTO限制，超过该RTO阈值，则不进行自愈操作，单位是秒。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，大于等于0，小于等于36000。

默认值：600

ai_watchdog_tolerance_times

参数说明：AI Watchdog启动自愈前最多能容忍多少次连续异常事件，通过该参数可以避免错误操作。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，大于等于0，小于等于100。

默认值：4

ai_watchdog_tps_threshold

参数说明：数据库实例的预期TPS使用下限，低于该值，会触发异常判断逻辑。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，大于等于0，小于等于32000。

默认值：2

ai_watchdog_wait_time

参数说明：为了避免数据库频繁进行自愈操作，会在数据库启动后一段时间进行等待，该值即用来调整等待时间，单位是秒。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，大于等于0，小于等于36000。

默认值：1800

ai_watchdog_warning_retention

参数说明：AI Watchdog在dbe_perf.ai_watchdog_detection_warnings视图中保留的告警记录数上限。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：整型，大于等于0，小于等于32000。

默认值：20

14.3.41 Global SysCache 参数

enable_global_syscache

参数说明：控制是否使用全局系统缓存功能。该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启全局系统缓存功能。
- off表示不开启全局系统缓存功能。

默认值：on

推荐结合线程池参数使用。打开该参数后，如果需要访问备机，建议设置备机wal_level级别为hot_standby以上。

global_syscache_threshold

参数说明：全局系统缓存内存最大占用大小。

该参数属于SIGHUP类型参数，请参考**表14-1**中对应设置方法进行设置。

需要打开enable_global_syscache参数。

取值范围：整型，16384~1073741824，单位为kB。

默认值：

163840（196核CPU/1536G内存，128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，64核CPU/512G内存，60核CPU/480G内存，32核CPU/256G内存，16核CPU/128G内存，8核CPU/64G内存，4核CPU/32G内存）；65536（4核CPU/16G内存）

推荐计算公式：热点DB个数和线程个数的最小值乘以每个DB分配的内存大小

即global_syscache_threshold = min(count(hot dbs)，count(threads)) * memofdb

热点DB数即访问较为频繁的数据库，线程数在线程池模式下取线程池线程个数和后台线程个数之和，非线程池模式不需要计算这个值，直接使用热点DB数。

memofdb即平均每个db应该分配的内存，每个DB的底噪内存是2M，平均每增加一个表或者索引，增加11k内存。

如果设置的值过小，会导致内存频繁淘汰，内存存在大量碎片无法回收，导致内存控制失效。

14.3.42 高效数据压缩算法相关参数

pca_shared_buffers

参数说明：类似于shared_buffers，用于设置页面压缩块地址映射管理buffer的大小。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：最小值64K，最大值16G。

说明

- 如果设置值小于64K，设置报错。
- 如果设置值大于16G，参数可以设置成功，但实际运行时候，自动内存运行设置为16G。
- 如果设置参数不带单位，默认是8K(一个页面的大小是8K)乘以设置的参数大小。

默认值：64K

14.3.43 备机数据修复

standby_page_repair

参数说明：控制备机回放时页面修复开关。该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示备机回放时会自动检测修复页面。

- off表示备机回放时不会自动检测修复页面。

默认值：off

14.3.44 分隔符

delimiter_name

参数说明：保存一个delimiter分隔符名称。

gsql客户端识别到分隔符的时候，会立即将输入的一条或多条SQL语句发送到服务端执行。该用法可以用在输入语句较多，并且语句中存在分号时，指定一个特殊的符号作为结束符。该参数属于USERSET类型参数，请参考**表14-1**中对应设置方法进行设置。

此参数只提供在gsql客户端设置，可以通过DELIMITER命令配置。

取值范围：字符串，长度大于0

默认值：";"

14.3.45 Global PLsql Cache 特性参数

enable_global_plsqlcache

参数说明：设置是否对package、存储过程、函数的编译产物进行全局缓存，执行产物进行session级缓存，开启该功能可以节省高并发下数据库节点的内存使用。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示对编译产物进行全局缓存。
- off表示不进行全局缓存。

默认值：off

max_execute_functions

参数说明：需要在enable_global_plsqlcache = on时进行设置，否则设置无效，用于定义session内存储过程、函数的执行产物个数。

个数大于max_execute_functions时将对执行产物进行清理，保留最近调用的max_execute_functions个执行产物。

该参数属于POSTMASTER类型参数，请参考**表14-1**中对应设置方法进行设置。

取值范围：最大值2147483647 ，最小值1。

默认值：1000

