云数据库 GaussDB

开发指南（分布式_V2.0-2.x）

文档版本 01

发布日期 2024-10-31

华为云计算技术有限公司

商标声明


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

注意

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

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

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

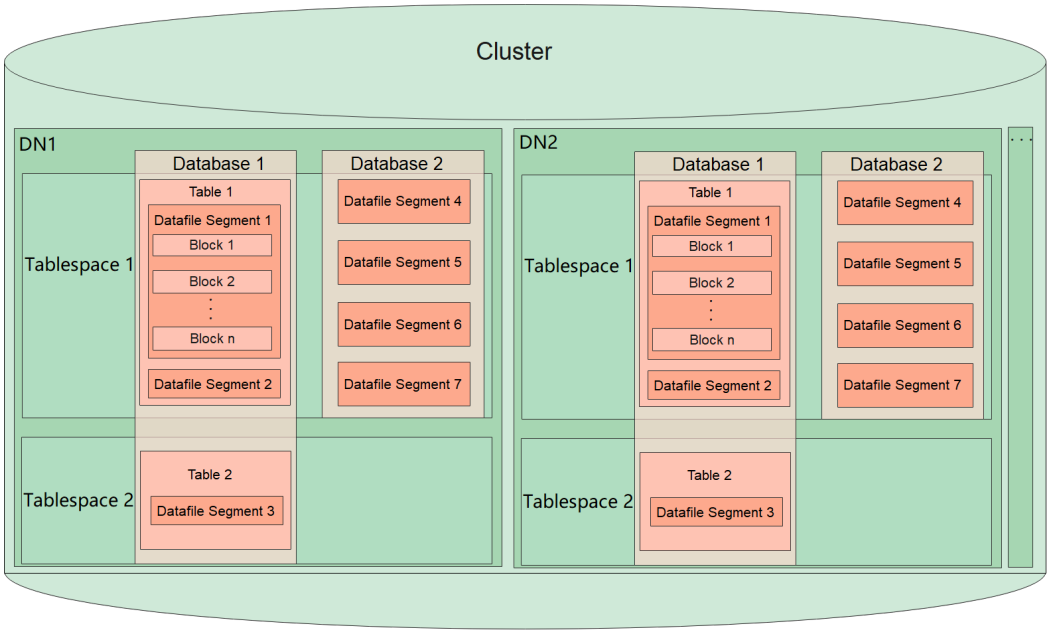
目 录

1 数据库系统概述

1.1 数据库逻辑结构图

集群的每个DN负责存储数据，其存储介质也是磁盘，本节主要从逻辑视角介绍每个DN上都有哪些对象，以及这些对象之间的关系。另外介绍数据在不同节点的分布方式。数据库逻辑结构如图1-1所示。

图 1-1 数据库逻辑结构图

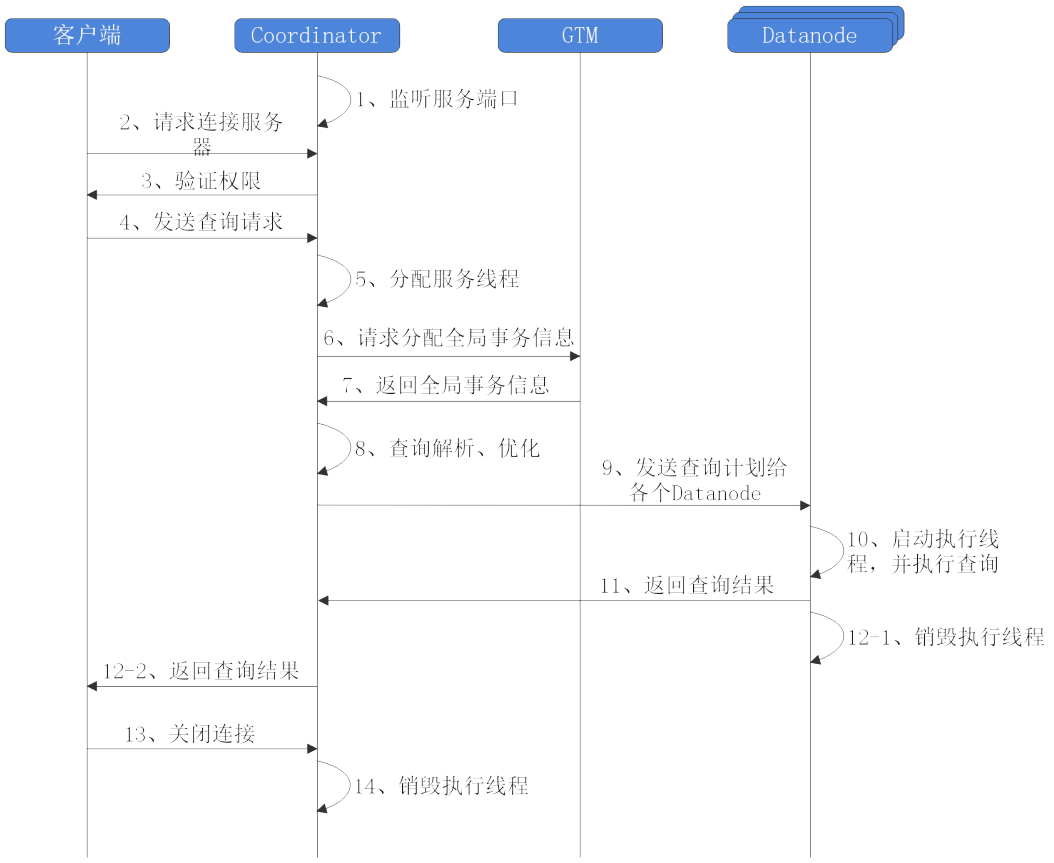


说明

- Tablespace，即表空间，是一个目录，集群中可以存在多个表空间，里面存储的是它所包含的数据库的各种物理文件。每个表空间可以对应多个Database。
- Database，即数据库，用于管理各类数据对象，各数据库间相互隔离。数据库管理的对象可分布在多个Tablespace上。
- Datafile Segment，即数据文件，通常每张表只对应一个数据文件。如果某张表的数据大于1GB，则会分为多个数据文件存储。
- Table，即表，每张表只能属于一个数据库，也只能对应到一个Tablespace。每张表对应的数据文件必须在同一个Tablespace中。
- Block，即数据块，是数据库管理的基本单位，默认大小为8KB。
- 数据在不同的DN上有四种分布方式，可以在建表的时候指定：REPLICATION、HASH、RANGE、LIST。

1.2 数据查询请求处理过程

图 1-2 GaussDB 服务响应流程



1.3 管理事务

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

UNCOMMITTED、REPEATABLE READ和SERIALIZABLE，不推荐使用READ UNCOMMITTED，SERIALIZABLE等价于REPEATABLE READ。

事务控制

以下是数据库支持的事务命令：

- 启动事务

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

- 设置事务

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

- 提交事务

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

- 回滚事务

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

事务隔离级别

事务隔离级别，它决定多个事务并发操作同一个对象时的处理方式。

说明

在事务中第一个数据修改语句（SELECT，INSERT，DELETE，UPDATE，FETCH，COPY）执行之后，事务隔离级别就不能再次设置。

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

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

  因为在读已提交模式里，每个新的命令都是从一个新的快照开始的，而这个快照包含所有到该时刻为止已提交的事务，因此同一事务中后面的命令将看到任何已提交的其它事务的效果。这里关心的问题是在单个命令里是否看到数据库里完全一致的视图。

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

- **READ UNCOMMITTED**：读未提交隔离级别。不推荐使用，可能产生数据不一致现象。在协调节点故障无法恢复时或应急时可考虑使用，越过GTM与CN/DN不一致时的阻塞，但建议写事务不要使用，以免产生数据不一致现象，读事务可应急使用。

- **REPEATABLE READ**：事务可重复读隔离级别，事务只能读到事务开始之前已提交的数据，不能读到未提交的数据以及事务执行期间其它并发事务提交的修改（但是，查询能查看到自身所在事务中先前修改的执行结果，即使先前修改尚未提交）。这个级别和读已提交是不一样的，因为可重复读事务中的查询看到的是事务开始时的快照，不是该事务内部当前查询开始时的快照，就是说，单个事务

内部的select命令总是查看到同样的数据，查看不到自身事务开始之后其他并发事务修改后提交的数据。使用该级别的应用必须准备好重试事务，因为可能会发生串行化失败。

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

1.4 相关概念

数据库

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

表空间

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

模式

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

用户和角色

GaussDB使用用户和角色来控制对数据库的访问。根据角色自身的设置不同，一个角色可以看做是一个数据库用户，或者一组数据库用户。在GaussDB中角色和用户之间的区别只在于角色默认是没有LOGIN权限的。在GaussDB中一个用户唯一对应一个角色，不过可以使用角色叠加来更灵活地进行管理。

事务管理

在事务管理上，GaussDB采取了MVCC（多版本并发控制）结合两阶段锁的方式，其特点是读写之间不阻塞。GaussDB的MVCC没有将历史版本数据统一存放，而是和当前元组的版本放在了一起。GaussDB没有回滚段的概念，但是为了定期清除历史版本数据，GaussDB引入了VACUUM线程。一般情况下，除非用户要做性能调优，否则不用特别关注VACUUM线程。此外，GaussDB对于单语句查询（没有使用BEGIN等语句显示启动事务块）是自动提交事务的。

2 管理数据库安全

2.1 用户及权限

2.1.1 默认权限机制

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

为使其他用户能够使用对象，必须向用户或包含该用户的角色授予必要的权限。

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

要撤销已经授予的权限，可以使用**REVOKE**。对象所有者的权限（例如ALTER、DROP、COMMENT、INDEX、VACUUM、GRANT和REVOKE）是隐式拥有的，即只要拥有对象就可以执行对象所有者的这些隐式权限。对象所有者可以撤消自己的普通权限，例如，使表对自己以及其他人只读。

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

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

不建议用户修改系统表和系统视图的权限。

2.1.2 管理员

初始用户

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

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

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

系统管理员

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

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

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

或者

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

ALTER USER时，要求用户已存在。

安全管理员

安全管理员是指具有CREATEROLE属性的账户，具有创建、修改、删除用户或角色的权限，和赋予或者撤销任何非系统管理员、内置角色、永久用户和运维管理员的权限。

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

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

或者

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

ALTER USER时，要求用户已存在。

审计管理员

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

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

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

或者

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

ALTER USER时，要求用户已存在。

监控管理员

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

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

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

或者

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

ALTER USER时，要求用户已存在。

运维管理员

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

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

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

或者

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

ALTER USER时，要求用户已存在。

安全策略管理员

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

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

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

或者

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

ALTER USER时，要求用户已存在。

2.1.3 三权分立

默认权限机制和**管理员**两节的描述基于的是集群创建之初的默认情况。从前面的介绍可以看出，默认情况下拥有SYSADMIN属性的系统管理员，具备系统最高权限。

在实际业务管理中，为了避免系统管理员拥有过度集中的权利带来高风险，可以设置三权分立，将系统管理员的用户管理和审计管理的权限分别分给安全管理员和审计管理员。

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

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

三权分立的设置办法为：将GUC参数enableSeparationOfDuty设置为on。

⚠ 警告

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

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

表 2-1 默认的用户权限

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

表 2-2 三权分立较非三权分立权限变化说明

对象名称	初始用户（id为10）	系统管理员	安全管理员	审计管理员	普通用户
表空间	无变化。 依然具有所有的权限。	无变化	无变化		
模式		权限缩小。 对自己的模式有所有的权限，对其他用户的非系统模式无权限。	无变化		

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

须知

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

2.1.4 用户

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

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

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

创建、修改和删除用户

- 要创建用户，请使用SQL语句**CREATE USER**。

  例如：创建用户joe，并设置用户拥有CREATEDB属性。

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

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

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

- 要查看用户属性，请查询系统表**PG_AUTHID**：

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

永久用户

GaussDB提供永久用户方案：创建具有PERSISTENCE属性的永久用户。


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

只允许初始用户创建、修改和删除具有PERSISTENCE属性的永久用户。

2.1.5 角色

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

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

要查看所有角色，请查询系统表PG_ROLES：

```
SELECT * FROM PG_ROLES;
```

创建、修改和删除角色

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

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

内置角色

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

表 2-3 内置角色权限描述

角色	权限描述
gs_role_copy_files	具有执行copy … to/from filename 的权限，但需要先打开GUC参数enable_copy_server_files。
gs_role_signal_backend	具有调用函数pg_cancel_backend、pg_terminate_backend和pg_terminate_session来取消或终止其他会话的权限，但不能操作属于初始用户和PERSISTENCE用户的会话。
gs_role_tablespace	具有创建表空间（tablespace）的权限。

角色	权限描述
gs_role_replication	具有调用逻辑复制相关函数的权限，例如kill_snapshot、pg_create_logical_replication_slot、pg_create_physical_replication_slot、pg_drop_replication_slot、pg_replication_slot_advance、pg_create_physical_replication_slot_extern、pg_logical_slot_get_changes、pg_logical_slot_peek_changes、pg_logical_slot_get_binary_changes、pg_logical_slot_peek_binary_changes。
gs_role_account_lock	具有加解锁用户的权限，但不能加解锁初始用户和PERSISTENCE用户。
gs_role_pldebugger	具有执行dbe_pldebugger下调试函数的权限。
gs_role_directory_create	具有执行创建directory对象的权限，但需要先打开GUC参数enable_access_server_directory。
gs_role_directory_drop	具有执行删除directory对象的权限，但需要先打开GUC参数enable_access_server_directory。

关于内置角色的管理有如下约束：

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

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

2.1.6 Schema

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

每个数据库包含一个或多个Schema。数据库中的每个Schema包含表和其他类型的对象。数据库创建初始，默认具有一个名为public的公共Schema，且所有用户都拥有此Schema的usage权限。此外，每个数据库都包含一个pg_catalog Schema，它包含系统表和所有内置数据类型、函数、操作符。只有系统管理员和初始化用户可以在public和pg_catalog Schema下创建函数、存储过程和同义词对象，其他用户即使赋予public和pg_catalog模式的create权限后也不可以创建上述三种对象。可以通过Schema分组数据库对象。Schema类似于操作系统目录，但Schema不能嵌套。默认只有初始化用户可以在pg_catalog模式下创建对象。

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

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

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

创建、修改和删除 Schema

- 要创建Schema，请使用**CREATE SCHEMA**。默认初始用户和系统管理员可以创建Schema，其他用户需要具备数据库的CREATE权限才可以在该数据库中创建Schema，赋权方式请参考**GRANT**中将数据库的访问权限赋予指定的用户或角色中的语法。
- 要更改Schema名称或者所有者，请使用**ALTER SCHEMA**。Schema所有者可以更改Schema。
- 要删除Schema及其对象，请使用**DROP SCHEMA**。Schema所有者可以删除Schema。
- 要在Schema内创建表，请以schema_name.table_name格式创建表。不指定schema_name时，对象默认创建到**搜索路径**中的第一个Schema内。
- 要查看Schema所有者，请对系统表PG_NAMESPACE和PG_USER执行如下关联查询。语句中的schema_name请替换为实际要查找的Schema名称。

  ```
  openGauss=# SELECT s.nspname,u.usename AS nspowner FROM pg_namespace s, pg_user u WHERE nspname='schema_name' AND s.nspowner = u.usesysid;
  ```

- 要查看所有Schema的列表，请查询PG_NAMESPACE系统表。

  ```
  openGauss=# SELECT * FROM pg_namespace;
  ```

- 要查看属于某Schema下的表列表，请查询系统视图PG_TABLES。例如，以下查询会返回Schema PG_CATALOG中的表列表。

  ```
  openGauss=# SELECT distinct(tablename),schemaname from pg_tables where schemaname = 'pg_catalog';
  ```

搜索路径

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

- 要查看当前搜索路径，请使用**SHOW**。

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

  search_path参数的默认值为："$user",public。$user表示与当前会话用户名同名的Schema名，如果这样的模式不存在，$user将被忽略。所以默认情况下，用户连接数据库后，如果数据库下存在同名Schema，则对象会添加到同名Schema下，否则对象被添加到Public Schema下。

- 要更改当前会话的默认Schema，请使用SET命令。

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


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

2.1.7 用户权限设置

- 给用户直接授予某对象的权限，请使用**GRANT**。

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

  例如，下面示例将Schema tpcds的权限赋给用户joe后，将表tpcds.web_returns的select权限赋给用户joe。

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

- 给用户指定角色，使用户继承角色所拥有的对象权限。

  a. 创建角色。

  新建一个角色lily，同时给角色指定系统权限CREATEDB：

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

  b. 给角色赋予对象权限，请使用**GRANT**。

  例如，将模式tpcds的权限赋给角色lily后，将表tpcds.web_returns的select权限赋给角色lily。

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

  c. 将角色的权限赋予用户。

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

  说明

  当将角色的权限赋予用户时，角色的属性并不会传递到用户。

- 回收用户权限，请使用**REVOKE**。

2.1.8 行级访问控制

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

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

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

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

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

--创建表all_data，包含不同用户数据信息
openGauss=# CREATE TABLE all_data(id int, role varchar(100), data varchar(100));
```



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

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

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

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

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

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

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

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

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


须知

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

3 数据库使用入门

3.1 连接数据库

连接数据库的客户端工具包括gsql、应用程序接口（如ODBC和JDBC）。

- gsql是GaussDB自带的客户端工具。**使用gsql连接**数据库，可以交互式地输入、编辑、执行SQL语句。
- 用户可以使用标准的数据库**应用程序接口**（如ODBC和JDBC），开发基于GaussDB的应用程序。

3.1.1 使用 gsql 连接

gsql是GaussDB提供的在命令行下运行的数据库连接工具。此工具除了具备操作数据库的基本功能，还提供了若干高级特性，便于用户使用。本节只介绍如何使用gsql连接数据库，关于gsql使用方法的更多信息请参考《工具参考》中“客户端工具 > gsql”。

注意事项

缺省情况下，客户端连接数据库后处于空闲状态时会根据GUC参数session_timeout的默认值自动断开连接。如果要关闭超时设置，设置GUC参数session_timeout为0即可。

前提条件

已联系管理员获取连接信息。

远程连接数据库

步骤1 完成远程连接配置，请联系管理员处理。

步骤2 在客户端机器上，上传客户端工具包并配置gsql的执行环境变量。

1. 登录客户端机器。
2. 创建“/tmp/tools”目录。
   mkdir */tmp/tools*

3. 获取软件安装包中的“GaussDB-Kernel_VxxxRxxxCxx-xxxxx-64bit-gsql.tar.gz”上传到“/tmp/tools”路径下。

   说明

   - 软件包相对位置为安装时所放位置，根据实际情况填写。
   - 不同的操作系统，工具包文件名称会有差异。请根据实际的操作系统类型选择对应的工具包。

4. 解压文件。

   ```
   cd /tmp/tools
   tar -zxvf GaussDB-Kernel_VxxxRxxxCxx-XXXXX-64bit-gsql.tar.gz
   ```

5. 设置环境变量。

   打开“~/.bashrc”文件。

   ```
   vi ~/.bashrc
   ```

   在其中输入如下内容后，使用“:wq!”命令保存并退出。

   ```
   export PATH=/tmp/tools/bin:$PATH
   export LD_LIBRARY_PATH=/tmp/tools/lib:$LD_LIBRARY_PATH
   ```

6. 使环境变量配置生效。

   ```
   source ~/.bashrc
   ```

步骤3 连接数据库。

数据库安装完成后，会默认生成名称为postgres的数据库。

```
gsql -d postgres -h 10.10.0.11 -U jack -p 8000
Password for user jack:
```

postgres为需要连接的数据库名称，10.10.0.11为CN所在的服务器IP地址，jack为登录数据库的用户名，8000为CN的端口号，上述示例中，连接的服务器的IP和端口也可以替换成DN服务器IP和端口。

说明

- 上述示例中，jack用户为数据库初始化用户创建的普通用户。
- 默认禁止使用数据库初始化用户进行远程连接数据库，开启集群内部kerberos认证时，允许初始化用户在集群内部进行远程连接。

----结束

3.1.2 应用程序接口

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

支持的应用程序接口

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

表 3-1 数据库应用程序接口

API	下载地址
ODBC	• Linux下： 驱动程序：GaussDB-Kernel_VxxxRxxxCxx-xxxxx-64bit-Odbc.tar.gz unixODBC源码包：**http://sourceforge.net/projects/unixodbc/files/unixODBC/2.3.0/unixODBC-2.3.0.tar.gz/download** • Windows下： 驱动程序：GaussDB-Kernel_VxxxRxxxCxx-Windows-Odbc.tar.gz
JDBC	• 驱动程序：GaussDB-Kernel_VxxxRxxxCxx-xxxxx-64bit-Jdbc.tar.gz • 驱动类：org.postgresql.Driver

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

更多支持的应用程序接口详细信息请参考**应用程序开发教程**。

3.2 操作数据库

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

3.2.1 创建数据库用户

默认只有集群安装时创建的管理员用户可以访问初始数据库，您还可以创建其他数据库用户帐号。

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

当结果显示为如下信息，则表示创建成功。

```
CREATE ROLE
```

如上创建了一个用户名为joe，密码为********的用户。如下命令为设置joe用户为系统管理员。

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

使用GRANT命令进行相关权限设置，具体操作请参考**GRANT**。

说明

关于数据库用户的更多信息请参考**用户及权限**。

3.2.2 创建和管理数据库

前提条件

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

背景信息

- 初始时，GaussDB包含两个模板数据库template0、template1，以及一个默认的用户数据库postgres。
- CREATE DATABASE实际上通过拷贝模板数据库来创建新数据库。只支持拷贝template0。请避免使用客户端或其他手段连接及操作两个模板数据库。

  说明

  - 模板数据库中没有用户表，可通过系统表PG_DATABASE查看模板数据库属性。
  - 模板template0不允许用户连接；模板template1只允许数据库初始用户和系统管理员连接，普通用户无法连接。
- 数据库系统中会有多个数据库，但是同一时刻客户端程序只能连接一个数据库。当前，不支持在不同的数据库之间进行相互查询（跨库查询或跨库事务）。
- 当一个数据库集群中存在多个数据库时，可以通过客户端工具的-d参数指定目标数据库进行登录，也可以在客户端程序登录数据库以后通过\c命令进行数据库切换。

注意事项

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

针对这种情况，请遵循以下条件：

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

操作步骤

步骤1 使用如下命令创建一个新的数据库db_tpcds。

```
openGauss=# CREATE DATABASE db_tpcds;
CREATE DATABASE
```


说明

- 数据库名称遵循SQL标识符的一般规则。当前角色自动成为此新数据库的所有者。
- 如果一个数据库系统用于承载相互独立的用户和项目，建议把它们放在不同的数据库里。
- 如果项目或者用户是相互关联的，并且可以相互使用对方的资源，则应该把它们放在同一个数据库里，但可以规划在不同的模式中。模式只是一个纯粹的逻辑结构，某个模式的访问权限由权限系统模块控制。
- 创建数据库时，若数据库名称长度超过63字节，server端会对数据库名称进行截断，保留前63个字节，因此建议数据库名称长度不要超过63个字节。
- 数据库默认创建在pg_default表空间下。若要指定表空间，可以使用如下语句：
  ```
  openGauss=# CREATE DATABASE db_tpcds WITH TABLESPACE = hr_local;
  CREATE DATABASE
  ```
  其中hr_local为表空间名称，关于如何创建表空间，请参考**创建和管理表空间**。
- 创建完db_tpcds数据库后，可以选择继续在默认的postgres数据库进行其他操作，也可以按如下方法退出postgres数据库，使用新用户连接到此数据库执行创建表等操作。
  ```
  openGauss=# \q
  gsql -d db_tpcds -p 8000 -U joe
  Password for user joe:
  gsql((GaussDB Kernel 503.0.XXX build f521c606) compiled at 2021-09-16 14:55:22 commit 2935 last mr 6385 release)
  Non-SSL connection (SSL connection is recommended when requiring high-security)
  Type "help" for help.

  db_tpcds=>
  ```

步骤2 查看数据库

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

步骤3 修改数据库。

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

- 使用如下命令为数据库重新命名。
  ```
  openGauss=# ALTER DATABASE db_tpcds RENAME TO human_tpcds;
  ALTER DATABASE
  ```

说明

执行完参数设置后，需要手动执行CLEAN CONNECTION清理旧连接，否则可能存在节点间参数值不一致。

步骤4 删除数据库

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

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

----结束

3.2.3 创建和管理表空间

背景信息

通过使用表空间，管理员可以控制一个数据库安装的磁盘布局。这样有以下优点：

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

  CM的磁盘自动检查功能默认是开启的，使用如下方式开启CM的磁盘自动检查功能：

  ```
  gs_guc set -Z cmserver -N all -I all -c " enable_transaction_read_only = on "
  ```

  重启数据库使参数设置生效。
- 表空间对应于一个文件系统目录，采用如下命令创建一个对应/pg_location/mount1/path1的表空间，并指定最大可使用空间为500GB。

  ```
  --创建表空间。
  openGauss=# CREATE TABLESPACE ds_location1 LOCATION '/pg_location/mount1/path1' MAXSIZE '500G';
  ```

  通过MAXSIZE进行表空间配额管理对并发插入性能可能会有30%左右的影响，MAXSIZE指定每个DN的配额大小，每个DN实际的表空间容量和配额误差在500MB以内。请根据实际的情况确认是否需要设置表空间的最大值。

GaussDB自带了两个表空间：pg_default和pg_global。

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

注意事项：

- 一般不建议用户使用自定义的表空间。

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

操作步骤

- 创建表空间

a. 执行如下命令创建用户jack，密码为 ********。
```
openGauss=# CREATE USER jack IDENTIFIED BY '********';
```
当结果显示为如下信息，则表示创建成功。
```
CREATE ROLE
```

b. 执行如下命令创建表空间。
```
openGauss=# CREATE TABLESPACE fastspace RELATIVE LOCATION 'my_tablespace/tablespace1';
```
当结果显示为如下信息，则表示创建成功。
```
CREATE TABLESPACE
```
其中“fastspace”为新创建的表空间，“CN和DN数据目录/*pg_location/my_tablespace/tablespace1*”是用户拥有读写权限的空目录。

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

- 在表空间中创建对象

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

  以创建表为例。

  - 方式1：执行如下命令在指定表空间创建表。
    ```
    openGauss=# CREATE TABLE foo(i int) TABLESPACE fastspace;
    ```
    当结果显示为如下信息，则表示创建成功。
    ```
    NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'i' as the distribution column by default.
    HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
    CREATE TABLE
    ```
  - 方式2：先使用set default_tablespace设置默认表空间，再创建表。
    ```
    openGauss=# SET default_tablespace = 'fastspace';
    SET
    openGauss=# CREATE TABLE foo2(i int);
    NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'i' as the distribution column by default.
    HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
    CREATE TABLE
    ```
    假设设置“fastspace”为默认表空间，然后创建表foo2。

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

- 查询表空间使用率

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

b. 计算表空间使用率。

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

- 修改表空间

  执行如下命令对表空间fastspace重命名为fspace。

```
openGauss=# ALTER TABLESPACE fastspace RENAME TO fspace;
ALTER TABLESPACE
```

- 删除表空间
  - 执行如下命令删除用户jack。

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

  - 执行如下命令删除表foo和foo2。

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

    当结果显示为如下信息，则表示删除成功。

```
DROP TABLE
```

  - 执行如下命令删除表空间fspace。

```
openGauss=# DROP TABLESPACE fspace;
DROP TABLESPACE
```

说明

用户必须是表空间的owner或者系统管理员才能删除表空间。

3.2.4 创建和管理表

3.2.4.1 创建表

背景信息

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

如何为业务设计最佳的表，请参考**表设计最佳实践**。

创建表

执行如下命令创建表。

```
openGauss=# CREATE TABLE customer_t1
(
    c_customer_sk             integer,
    c_customer_id             char(5),
    c_first_name              char(6),
    c_last_name               char(8)
)
distribute by hash (c_last_name);
```

当结果显示为如下信息，则表示创建成功。

```
CREATE TABLE
```

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

说明

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

3.2.4.2 向表中插入数据

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

背景信息

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

表 3-2 客户端和服务端设置字符集的输出结果对比

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

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

说明

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

执行如下命令建立示例中需要使用的表table1、table2。

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

表 3-3 示例

编号	服务端字符集	客户端字符集	是否启用自动截断	示例	结果	说明
1	SQL_ASCII	UTF8	是	openGauss=# **INSERT INTO** *table1* **VALUES**(*1,reverse('123A A 78'),reverse('123A A 78'),reverse('123A A 78')*);	id |a|b|c ----+------+------+------ 1 | 87| 87| 87	字符串在服务端翻转后，并进行截断，由于服务端和客户端的字符集不一致，字符A在客户端由多个字节表示，结果产生异常。
2	SQL_ASCII	UTF8	是	openGauss=# **INSERT INTO** *table1* **VALUES**(*2,reverse('123A 78'),reverse('123A 78'),reverse('123A 78')*);	id |a|b|c ----+------+------+------ 2 | 873| 873| 873	字符串翻转后，又进行了自动截断，所以产生了非预期的效果。
3	SQL_ASCII	UTF8	是	openGauss=# **INSERT INTO** *table1* **VALUES**(*3,'87A 123','87A 123','87A 123'*);	id | a | b | c ----+-------+-------+------- 3 | 87A1 | 87A1 | 87A1	字符串类型的字段长度是客户端字符编码长度的整数倍，所以截断后产生结果正常。
4	SQL_ASCII	UTF8	否	openGauss=# **INSERT INTO** *table2* **VALUES**(*1,reverse('123A A 78'),reverse('123A A 78'),reverse('123A A 78')*); openGauss=# **INSERT INTO** *table2* **VALUES**(*2,reverse('123A 78'),reverse('123A 78'),reverse('123A 78')*);	id |a|b|c ---- +---------- --+-------- +-------- 1 | 87 321| 87 321 | 87 321 2 | 87321| 87321| 87321	与示例1类似，多字节字符翻转之后不再表示原来的字符。

操作步骤

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

- 向表customer_t1中插入一行：

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

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

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

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

如果用户不知道所有字段的数值，可以忽略其中的一些。如果INSERT语句中，用户没有显示指定目标表的字段名字，那么VALUES子句中待插入的多个值，将按照列号和目标表的字段一一对应，即VALUES子句的第一个值对应目标表的第一列，VALUES子句的第二个值对应目标表的第二列，依次类推，没有VALUES数值对应的列自动填充缺省值或NULL。例如：

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

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

用户也可以对独立的字段或者整个行明确缺省值：

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

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

- 如果需要在表中插入多行，请使用以下命令：

```
openGauss=# INSERT INTO customer_t1 (c_customer_sk, c_customer_id, c_first_name) VALUES
    (6885, 'maps', 'Joes'),
    (4321, 'tpcds', 'Lily'),
    (9527, 'world', 'James');
```

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

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

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

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

说明

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

- 删除备份表

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

说明

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

3.2.4.3 更新表中数据

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

使用UPDATE命令更新现有行，需要提供以下三种信息：

- 表的名称和要更新的字段名
- 字段的新值
- 要更新哪些行

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

建立表和插入数据的步骤请参考**创建表**和**向表中插入数据**。

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

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

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

比如，把所有c_customer_sk的值增加100：

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

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

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

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

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

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

3.2.4.4 查看数据

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

3.2.4.5 删除表中数据

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

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

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

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

如果执行如下命令之一，会删除表中所有的行。

```
openGauss=# DELETE FROM customer_t1;
或：
openGauss=# TRUNCATE TABLE customer_t1;
```

说明

全表删除的场景下，建议使用truncate，不建议使用delete。

删除创建的表：

```
openGauss=# DROP TABLE customer_t1;
```

3.2.5 查看系统表

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

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

GaussDB提供了以下类型的系统表和视图：

- 兼容PostgreSQL的系统表和视图
  这类系统表和视图具有PG或PGXC前缀。
- GaussDB内核新增的系统表和视图
  这类系统表和视图具有GS前缀。
- 实现Oracle兼容的系统表和视图
  这类系统表和视图具有ALL、DBA、USER或PV前缀。

查看数据库中包含的表

在public Schema下新建五张表：

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

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

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

结果类似如下这样：


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

查看数据库用户

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

```
SELECT * FROM pg_user;
 usename | usesysid | usecreatedb | usesuper | usecatupd | userepl |  passwd  | valbegin | valuntil |
respool    | parent | spacelimit | useconfig | nodegroup | tempspacelimit | spillspacelimit
---------+----------+-------------+----------+-----------+---------+----------+----------+----------+-------------
+------
-------+------------+-----------+-----------+----------------+-----------------
 roach   |       10 | t          | t        | t         | t       | ******** |          |          | default_pool |
0      |          |          |          |             |
(1 row)
```

查看和停止正在运行的查询语句

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

步骤1 设置参数track_activities为on。

```
SET track_activities = on;
```

当此参数为on时，数据库系统才会收集当前活动查询的运行信息。

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

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

如果state字段显示为idle，则表明此连接处于空闲，等待用户输入命令。

如果仅需要查看非空闲的查询语句，则使用如下命令查看：

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

步骤3 若需要取消运行时间过长的查询，通过PG_TERMINATE_BACKEND函数，根据线程ID（即**步骤2**中查询结果的pid字段）结束会话。

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

显示类似如下信息，表示结束会话成功。

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

显示类似如下信息，表示用户执行了结束当前会话的操作。

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


说明

1. gsql客户端使用PG_TERMINATE_BACKEND函数结束当前正在执行会话的后台线程时，如果当前的用户是初始用户，客户端不会退出而是自动重连，即还会返回“The connection to the server was lost. Attempting reset: Succeeded.”；否则客户端会重连失败，即返回“The connection to the server was lost. Attempting reset: Failed.”。这是因为只有初始用户可以免密登录，普遍用户不能免密登录，从而重连失败。

2. 对于使用PG_TERMINATE_BACKEND函数结束非活跃的后台线程时。如果打开了线程池，此时空闲的会话没有线程ID，无法结束会话。非线程池模式下，结束的会话不会自动重连。

----结束

3.2.6 其他操作

3.2.6.1 创建和管理 schema

背景信息

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

注意事项

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

搜索到数据库内置对象。如果用户需要使用和系统内置对象重名的自定义对象时，可以在操作自定义对象时带上自己的模式。

操作步骤

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

  在特定schema下创建对象或者访问特定schema下的对象，需要使用有schema修饰的对象名。该名称包含schema名以及对象名，schema名和对象名之间用“.”号分开。
  - 执行如下命令在myschema下创建mytable表。
    ```
    openGauss=# CREATE TABLE myschema.mytable(id int, name varchar(20));
    CREATE TABLE
    ```
    如果在数据库中指定对象的位置，就需要使用有schema修饰的对象名称。
  - 执行如下命令查询myschema下mytable表的所有数据。
    ```
    openGauss=# SELECT * FROM myschema.mytable;
    id | name
    ----+------
    (0 rows)
    ```
- schema的搜索路径

  可以设置search_path配置参数指定寻找对象可用schema的顺序。在搜索路径列出的第一个schema会变成默认的schema。如果在创建对象时不指定schema，则会创建在默认的schema中。
  - 执行如下命令查看搜索路径。
    ```
    openGauss=# SHOW SEARCH_PATH;
     search_path
    ----------------
     "$user",public
    (1 row)
    ```
  - 执行如下命令将搜索路径设置为myschema, public，首先搜索myschema，然后搜索public。
    ```
    openGauss=# SET SEARCH_PATH TO myschema, public;
    SET
    ```
- schema的权限控制

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

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

  - 撤销PUBLIC在public模式下创建对象的权限，下面语句中第一个“public”是模式，第二个“PUBLIC”指的是所有角色。
    ```
    openGauss=# REVOKE CREATE ON SCHEMA public FROM PUBLIC;
    REVOKE
    ```
  - 使用以下命令查看现有的schema：
    ```
    openGauss=# SELECT current_schema();
     current_schema
    ----------------
     myschema
    (1 row)
    ```
  - 执行如下命令创建用户jack，并将myschema的usage权限赋给用户jack。
    ```
    openGauss=# CREATE USER jack IDENTIFIED BY '********';
    CREATE ROLE
    openGauss=# GRANT USAGE ON schema myschema TO jack;
    GRANT
    ```
  - 将用户jack对于myschema的usage权限收回。
    ```
    openGauss=# REVOKE USAGE ON schema myschema FROM jack;
    REVOKE
    ```
- 删除schema
  - 当schema为空时，即该schema下没有数据库对象，使用DROP SCHEMA命令进行删除。例如删除名为nullschema的空schema。
    ```
    openGauss=# DROP SCHEMA IF EXISTS nullschema;
    DROP SCHEMA
    ```
  - 当schema非空时，如果要删除一个schema及其包含的所有对象，需要使用CASCADE关键字。例如删除myschema及该schema下的所有对象。
    ```
    openGauss=# DROP SCHEMA myschema CASCADE;
    DROP SCHEMA
    ```
  - 执行如下命令删除用户jack。
    ```
    openGauss=# DROP USER jack;
    DROP ROLE
    ```

3.2.6.2 创建和管理分区表

背景信息

GaussDB数据库支持的分区表为范围分区表。

范围分区表：将数据基于范围映射到每一个分区，这个范围是由创建分区表时指定的分区键决定的。这种分区方式是最为常用的，并且分区键经常采用日期，例如将销售数据按照月份进行分区。

分区表和普通表相比具有以下优点：

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

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

操作步骤

示例一：使用默认表空间

- 创建分区表（假设用户已创建tpcds schema）


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

当结果显示为如下信息，则表示创建成功。

```
CREATE TABLE
```

- 插入数据

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

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

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



```
openGauss=# INSERT INTO tpcds.web_returns_p2 SELECT * FROM tpcds.customer_address;
INSERT 0 0
```

- 修改分区表行迁移属性

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

- 删除分区

  删除分区P8。

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

- 增加分区

  增加分区P8，范围为 40000<= P8<=MAXVALUE。

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

- 重命名分区
  - 重命名分区P8为P_9。

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

  - 重命名分区P_9为P8。

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

- 查询分区

  查询分区P6。

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

- 删除分区表和表空间

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

示例二：使用用户自定义表空间

按照以下方式对范围分区表进行操作（示例中的tpcds命名空间需提前创建）。

- 创建表空间

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

  当结果显示为如下信息，则表示创建成功。

```
CREATE TABLESPACE
```

- 创建分区表

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

```
TABLESPACE example1
DISTRIBUTE BY HASH (ca_address_sk)
PARTITION BY RANGE (ca_address_sk)
(
    PARTITION P1 VALUES LESS THAN(5000),
    PARTITION P2 VALUES LESS THAN(10000),
    PARTITION P3 VALUES LESS THAN(15000),
    PARTITION P4 VALUES LESS THAN(20000),
    PARTITION P5 VALUES LESS THAN(25000),
    PARTITION P6 VALUES LESS THAN(30000),
    PARTITION P7 VALUES LESS THAN(40000),
    PARTITION P8 VALUES LESS THAN(MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;
```

当结果显示为如下信息，则表示创建成功。

```
CREATE TABLE
```

- 插入数据

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

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

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

- 修改分区表行迁移属性

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

- 删除分区

  删除分区P8。

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

- 增加分区

  增加分区P8，范围为 40000<= P8<=MAXVALUE。


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

- 重命名分区
  - 重命名分区P8为P_9。
    ```
    openGauss=# ALTER TABLE tpcds.web_returns_p2 RENAME PARTITION P8 TO P_9;
    ALTER TABLE
    ```
  - 重命名分区P_9为P8。
    ```
    openGauss=# ALTER TABLE tpcds.web_returns_p2 RENAME PARTITION FOR (40000) TO P8;
    ALTER TABLE
    ```
- 修改分区的表空间
  - 修改分区P6的表空间为example3。
    ```
    openGauss=# ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P6 TABLESPACE
    example3;
    ALTER TABLE
    ```
  - 修改分区P4的表空间为example4。
    ```
    openGauss=# ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P4 TABLESPACE
    example4;
    ALTER TABLE
    ```
- 查询分区

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

3.2.6.3 创建和管理索引

背景信息

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

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

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

- 为经常出现在关键字ORDER BY、GROUP BY、DISTINCT后面的字段建立索引。

  说明

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

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

操作步骤

创建分区表的步骤请参考**创建和管理分区表**。

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

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

    当结果显示为如下信息，则表示创建成功。

    ```
    CREATE INDEX
    ```

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

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

    当结果显示为如下信息，则表示创建成功。

    ```
    CREATE INDEX
    ```

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

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

    当结果显示为如下信息，则表示修改成功。

    ```
    ALTER INDEX
    ```

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

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

    当结果显示为如下信息，则表示修改成功。

    ```
    ALTER INDEX
    ```

- 重命名索引分区

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

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

  当结果显示为如下信息，则表示重命名成功。


```
ALTER INDEX
```

- 查询索引
  - 执行如下命令查询系统和用户定义的所有索引。
    ```
    openGauss=# SELECT RELNAME FROM PG_CLASS WHERE RELKIND='i';
    ```
  - 执行如下命令查询指定索引的信息。
    ```
    openGauss=# \di+ tpcds.tpcds_web_returns_p2_index2
    ```
- 删除索引
  ```
  openGauss=# DROP INDEX tpcds.tpcds_web_returns_p2_index1;
  openGauss=# DROP INDEX tpcds.tpcds_web_returns_p2_index2;
  ```
  当结果显示为如下信息，则表示删除成功。
  ```
  DROP INDEX
  ```

GaussDB支持4种创建索引的方式请参见表3-4。

说明

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

表 3-4 索引方式

索引方式	描述
唯一索引	可用于约束索引属性值的唯一性，或者属性组合值的唯一性。如果一个表声明了唯一约束或者主键，则GaussDB自动在组成主键或唯一约束的字段上创建唯一索引（可能是多字段索引），以实现这些约束。目前，GaussDB只有B-Tree及UBTree可以创建唯一索引。
多字段索引	一个索引可以定义在表中的多个属性上。目前，GaussDB中的B-Tree支持多字段索引，且最多可在32个字段上创建索引。
部分索引	建立在一个表的子集上的索引，这种索引方式只包含满足条件表达式的元组。
表达式索引	索引建立在一个函数或者从表中一个或多个属性计算出来的表达式上。表达式索引只有在查询时使用与创建时相同的表达式才会起作用。

- 创建一个普通表。
  ```
  openGauss=# CREATE TABLE tpcds.customer_address_bak AS TABLE tpcds.customer_address;
  INSERT 0 0
  ```
- 创建普通索引

  如果对于tpcds.customer_address_bak表，需要经常进行以下查询。
  ```
  openGauss=# SELECT ca_address_sk FROM tpcds.customer_address_bak WHERE
  ca_address_sk=14888;
  ```
  通常，数据库系统需要逐行扫描整个tpcds.customer_address_bak表以寻找所有匹配的元组。如果表tpcds.customer_address_bak的规模很大，但满足WHERE条件的只有少数几个（可能是零个或一个），则这种顺序扫描的性能就比较差。如果让数据库系统在ca_address_sk属性上维护一个索引，用于快速定位匹配的元组，则数据库系统只需要在搜索树上查找少数的几层就可以找到匹配的元组，这

将会大大提高数据查询的性能。同样，在数据库中进行更新和删除操作时，索引也可以提升这些操作的性能。

使用以下命令创建索引。

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

- 创建唯一索引

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

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

- 创建多字段索引

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

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

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

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

- 创建部分索引

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

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

- 创建表达式索引

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

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

  可以为上面的查询创建表达式索引：

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

- 删除tpcds.customer_address_bak表。

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

3.2.6.4 创建和管理视图

背景信息

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

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

管理视图

- 创建视图

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


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

说明

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

- 查询视图

  执行如下命令查询MyView视图。

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

- 查看当前用户下的视图

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

- 查看所有视图

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

- 查看某视图的具体信息

  执行如下命令查询MyView视图的详细信息。

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

- 删除视图

  执行如下命令删除MyView视图。

  ```
  openGauss=# DROP VIEW MyView;
  DROP VIEW
  ```

3.2.6.5 创建和管理序列

背景信息

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

通过序列使某字段成为唯一标识符的方法有两种：

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

操作步骤

方法一： 声明字段类型为序列整型来定义标识符字段。例如：

```
openGauss=# CREATE TABLE T1
(
    id    serial,
    name  text
);
```

当结果显示为如下信息，则表示创建成功。

```
CREATE TABLE
```


方法二： 创建序列，并通过nextval('sequence_name')函数指定为某一字段的默认值。这种方式更灵活，可以为序列定义cache，一次预申请多个序列值，减少与GTM的交互次数，来提高性能。

1. 创建序列

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

   当结果显示为如下信息，则表示创建成功。

   ```
   CREATE SEQUENCE
   ```

2. 指定为某一字段的默认值，使该字段具有唯一标识属性。

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

   当结果显示为如下信息，则表示默认值指定成功。

   ```
   CREATE TABLE
   ```

3. 指定序列与列的归属关系。

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

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

   当结果显示为如下信息，则表示指定成功。

   ```
   ALTER SEQUENCE
   ```

说明

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

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

注意事项

新序列值的产生是靠GTM维护的，默认情况下，每申请一个序列值都要向GTM发送一次申请，GTM在当前值的基础上加上步长值作为产生的新值返回给调用者。GTM作为全局唯一的节点，势必成为性能的瓶颈，所以对于需要大量频繁产生序列号的操作，如使用Bulkload工具进行数据导入场景，是非常不推荐产生默认序列值的。比如，在下面所示的场景中， INSERT INTO SELECT FROM语句的性能会非常慢。

```
openGauss=# CREATE SEQUENCE newSeq1;
openGauss=# CREATE TABLE newT1
        (
          id   int not null default nextval('newSeq1'),
          name text
        );
openGauss=# INSERT INTO newT1(name) SELECT name FROM T1;
```

可以提高性能的写法是（假设T1表导入newT1表中的数据为10000行）：

```
openGauss=# INSERT INTO newT1(id, name) SELECT id,name FROM T1;
openGauss=# SELECT SETVAL('newSeq1',10000);
```

说明

序列操作函数nextval()，setval() 等均不支持回滚。另外setval设置的新值，会对当前会话的nextval立即生效，但对其他会话，如果定义了cache，不会立即生效，在用尽所有缓存的值后，其变动才被其他会话感知。所以为了避免产生重复值，要谨慎使用setval，设置的新值不能是已经产生的值或者在缓存中的值。

如果必须要在bulkload场景下产生默认序列值，则一定要为newSeq1定义足够大的cache，并且不要定义Maxvalue或者Minvalue。数据库会试图将nextval('sequence_name')的调用下推到Data Node，以提高性能。 目前GTM对并发的连接请求是有限制的，当Data Node很多时，将产生大量并发连接， 这时一定要控制bulkload的并发数目，避免耗尽GTM的连接资源。如果目标表为复制表(DISTRIBUTE BY REPLICATION)时下推将不能进行。当数据量较大时，这对数据库将是个灾难。除了性能问题之外，空间也可能会剧烈膨胀，在导入结束后，需要用vacuum full来恢复。推荐采用如上建议，不要在bulkload的场景中产生默认序列值。

另外，序列创建后，在每个节点上都维护了一张单行表，存储序列的定义及当前值，但此当前值并非GTM上的当前值，只是保存本节点与GTM交互后的状态。如果其他节点也向GTM申请了新值，或者调用了Setval修改了序列的状态，不会刷新本节点的单行表，但因每次申请序列值是向GTM申请，所以对序列正确性没有影响。

3.2.6.6 创建和管理定时任务

背景信息

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

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

定时任务管理

步骤1 创建测试表：

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

当结果显示为如下信息，则表示创建成功。

```
CREATE TABLE
```

步骤2 创建自定义存储过程：

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

当结果显示为如下信息，则表示创建成功。

```
CREATE PROCEDURE
```

步骤3 创建任务：

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

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


- 指定job_id创建任务，其中job_id可用范围为1~32767。

```
openGauss=# call dbe_task.id_submit(2,'call public.prc_job_1(); ', sysdate, 'interval ''1 minute''');
isubmit
---------

(1 row)
```

步骤4 通过视图查看当前用户已创建的任务信息。

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

步骤5 停止任务。

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

(1 row)
```

步骤6 启动任务。

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

(1 row)
```

步骤7 修改任务属性：

- 修改JOB的Next_date参数信息。

  --修改Job1的Next_date为1小时以后开始执行。

```
openGauss=# call dbe_task.next_time(1, sysdate+1.0/24);
next_date
-----------

(1 row)
```

- 修改JOB的Interval参数信息。

  --修改Job1的Interval为每隔1小时执行一次。

```
openGauss=# call dbe_task.interval(1,'sysdate + 1.0/24');
interval
----------

(1 row)
```

- 修改JOB的What参数信息。

  --修改Job1的What为执行SQL语句“insert into public.test values(333, sysdate +5);”。

```
openGauss=# call dbe_task.content(1,'insert into public.test values(333, sysdate+5);');
what
------

(1 row)
```

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

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

(1 row)
```


步骤8 删除JOB。

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

(1 row)
```

步骤9 查看JOB执行情况。

当JOB自动执行时，如果JOB执行失败（即job_status状态值为'f'）时，用户可以通过查看当前JOB所属CN的数据目录的pg_log子目录下对应时间点的运行日志来查看JOB的失败信息。

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

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

步骤10 JOB的权限控制：

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

步骤11 JOB的并发控制管理。

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

- 当job_queue_processes设置为0值，表示不启用定时任务功能，任何job都不会被执行。
- 当job_queue_processes为大于0时，表示启用定时任务功能且系统能够并发处理的最大任务数。

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

注：对于不使用JOB的集群中，用户可以通过在集群安装初始化完成后，通过设置job_queue_processes为0来关闭JOB功能，减少系统资源的消耗。

----结束

4 开发设计建议

4.1 开发设计建议概述

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

本开发设计建议中所陈述的“建议”和“关注”含义如下：

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

4.2 数据库对象命名

数据库对象命名需要满足约束：

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

  说明

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

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

4.3 数据库对象设计

4.3.1 Database 和 Schema 设计

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

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

Database 设计建议

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

Schema 设计建议

- 【建议】实际用户环境中Schema数量不建议超过100个。当数据库中存在大量Schema时，会导致gs_dump等依赖Schema数量的操作性能变慢。

- 【关注】如果该用户不具有sysadmin权限或者不是该Schema的owner，要访问Schema下的对象，需要同时给用户赋予Schema的usage权限和对象的相应权限。
- 【关注】如果要在Schema下创建对象，需要授予操作用户该Schema的CREATE权限。
- 【关注】Schema的owner默认拥有该Schema下对象的所有权限，包括删除权限。删除权限影响较大，请谨慎使用。

4.3.2 表设计

GaussDB是分布式架构。数据分布在各个DN上。总体上讲，良好的表设计需要遵循以下原则：

- 【关注】将表数据均匀分布在各个DN上。数据均匀分布，可以防止数据在部分DN上集中分布，从而导致因存储倾斜造成集群有效容量下降。通过选择合适的分布列，可以避免数据倾斜。
- 【关注】将表的扫描压力均匀分散在各个DN上。避免扫描压力集中在部分DN上，而导致性能瓶颈。例如，在事实表上使用等值过滤条件时，将会导致扫描压力不均匀。
- 【关注】减少需要扫描的数据量。通过分区表的剪枝机制可以大幅减少数据的扫描量。
- 【关注】尽量减少随机I/O。通过聚簇/局部聚簇可以实现热数据的连续存储，将随机I/O转换为连续I/O，从而减少扫描的I/O代价。
- 【关注】尽量避免数据shuffle。shuffle，是指在物理上，数据从一个节点，传输到另一个节点。shuffle占用了大量宝贵的网络资源，减少不必要的数据shuffle，可以减少网络压力，使数据的处理本地化，提高集群的性能和可支持的并发度。通过对关联条件和分组条件的仔细设计，能够尽可能的减少不必要的数据shuffle。

选择存储方案

【建议】表的存储类型是表定义设计的第一步，客户业务类型是决定表的存储类型的主要因素，表存储类型的选择依据请参考**表4-1**。

表 4-1 表的存储类型及场景

存储类型	适用场景
行存	• 点查询(返回记录少，基于索引的简单查询)。 • 增、删、改操作较多的场景。

选择分布方案

【建议】表的分布方式的选择一般遵循以下原则：

表 4-2 表的分布方式及使用场景

分布方式	描述	适用场景
Hash	表数据通过Hash方式散列到集群中的所有DN上。	数据量较大的事实表。
Replication	集群中每一个DN都有一份全量表数据。	维度表、数据量较小的事实表。
Range	表数据对指定列按照范围进行映射，分布到对应DN。	用户需要自定义分布规则的场景。
List	表数据对指定列按照具体值进行映射，分布到对应DN。	用户需要自定义分布规则的场景。

说明

当指定Hash、Range或List分布时，创建主键和唯一索引必须包含分布列。

典型的分布表定义如下：

```
--定义一个表，表中每行存在所有DN中。
CREATE TABLE warehouse_d1
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
)DISTRIBUTE BY REPLICATION;

--定义一个表，使用HASH分布。
CREATE TABLE warehouse_d2
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2),
    CONSTRAINT W_CONSTR_KEY3 UNIQUE(W_WAREHOUSE_SK)
)DISTRIBUTE BY HASH(W_WAREHOUSE_SK);

--定义一个表，使用RANGE分布
CREATE TABLE warehouse_d3
(
```



```
W_WAREHOUSE_SK            INTEGER               NOT NULL,
W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
W_WAREHOUSE_NAME            VARCHAR(20)                 ,
W_WAREHOUSE_SQ_FT         INTEGER                     ,
W_STREET_NUMBER           CHAR(10)                    ,
W_STREET_NAME             VARCHAR(60)                 ,
W_STREET_TYPE           CHAR(15)                    ,
W_SUITE_NUMBER            CHAR(10)                    ,
W_CITY                  VARCHAR(60)                 ,
W_COUNTY                  VARCHAR(30)                 ,
W_STATE                 CHAR(2)                     ,
W_ZIP                   CHAR(10)                    ,
W_COUNTRY                 VARCHAR(20)                 ,
W_GMT_OFFSET              DECIMAL(5,2)
)DISTRIBUTE BY RANGE(W_WAREHOUSE_ID)
(
   SLICE s1 VALUES LESS THAN (10) DATANODE dn1,
   SLICE s2 VALUES LESS THAN (20) DATANODE dn2,
   SLICE s3 VALUES LESS THAN (30) DATANODE dn3,
   SLICE s4 VALUES LESS THAN (MAXVALUE) DATANODE dn4
);

--定义一个表，使用LIST分布
CREATE TABLE warehouse_d4
(
W_WAREHOUSE_SK            INTEGER               NOT NULL,
W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
W_WAREHOUSE_NAME            VARCHAR(20)                 ,
W_WAREHOUSE_SQ_FT         INTEGER                     ,
W_STREET_NUMBER           CHAR(10)                    ,
W_STREET_NAME             VARCHAR(60)                 ,
W_STREET_TYPE           CHAR(15)                    ,
W_SUITE_NUMBER            CHAR(10)                    ,
W_CITY                  VARCHAR(60)                 ,
W_COUNTY                  VARCHAR(30)                 ,
W_STATE                 CHAR(2)                     ,
W_ZIP                   CHAR(10)                    ,
W_COUNTRY                 VARCHAR(20)                 ,
W_GMT_OFFSET              DECIMAL(5,2)
)DISTRIBUTE BY LIST(W_COUNTRY)
(
   SLICE s1 VALUES ('USA') DATANODE dn1,
   SLICE s2 VALUES ('CANADA') DATANODE dn2,
   SLICE s3 VALUES ('UK') DATANODE dn3,
   SLICE s4 VALUES (DEFAULT) DATANODE dn4
);
```

更多的表分布语法信息参见**CREATE TABLE**。

选择分布键

分布表的分布键选取至关重要，如果分布键选择不当，可能会导致数据倾斜，从而导致查询时，I/O负载集中在部分DN上，影响整体查询性能。因此，在确定分布表的分布策略之后，需要对表数据进行倾斜性检查，以确保数据的均匀分布。分布键的选择一般需要遵循以下原则：

- 【建议】选作分布键的字段取值应该比较离散，以便数据能在各个DN上均匀分布。当单个字段无法满足离散条件时，可以考虑使用多个字段一起作为分布键。一般情况下，可以考虑选择表的主键作为分布键。例如，在人员信息表中选择证件号码作为分布键。
- 【建议】在满足第一条原则的情况下，尽量不要选取在查询中存在常量过滤条件的字段作为分布键。例如，在表dwcjk相关的查询中，字段zqdh存在常量过滤条件“zqdh='000001'”，那么就应当尽量不选择zqdh字段作为分布键。

- 【建议】在满足前两条原则的情况，尽量选择查询中的关联条件为分布键。当关联条件作为分布键时，join任务的相关数据都分布在DN本地，将极大减少DN之间的数据流动代价。

选择分区方案

当表中的数据量很大时，应当对表进行分区，一般需要遵循以下原则：

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

典型的分区表定义如下：

```
CREATE TABLE staffS_p1
(
  staff_ID       NUMBER(6) not null,
  FIRST_NAME     VARCHAR2(20),
  LAST_NAME      VARCHAR2(25),
  EMAIL          VARCHAR2(25),
  PHONE_NUMBER   VARCHAR2(20),
  HIRE_DATE      DATE,
  employment_ID  VARCHAR2(10),
  SALARY         NUMBER(8,2),
  COMMISSION_PCT NUMBER(4,2),
  MANAGER_ID     NUMBER(6),
  section_ID     NUMBER(4)
)
PARTITION BY RANGE (HIRE_DATE)
(
   PARTITION HIRE_19950501 VALUES LESS THAN ('1995-05-01 00:00:00'),
   PARTITION HIRE_19950502 VALUES LESS THAN ('1995-05-02 00:00:00'),
   PARTITION HIRE_maxvalue VALUES LESS THAN (MAXVALUE)
);
```

4.3.3 字段设计

选择数据类型

在字段设计时，基于查询效率的考虑，一般遵循以下原则：

- 【建议】尽量使用高效数据类型。

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

  关于字符串类型的详细说明，请参见**常用字符串类型介绍**。

常用字符串类型介绍

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

4.3.4 约束设计

DEFAULT 和 NULL 约束

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

唯一约束

- 【关注】行存表支持唯一约束。
- 【建议】从命名上明确标识唯一约束，例如，命名为“UNI+构成字段”。

主键约束

- 【关注】行存表支持主键约束。
- 【建议】从命名上明确标识主键约束，例如，将主键约束命名为 “PK+字段名”。

检查约束

- 【关注】行存表支持检查约束。
- 【建议】从命名上明确标识检查约束，例如，将检查约束命名为 “CK+字段名”。

4.3.5 视图和关联表设计

视图设计

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

关联表设计

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

4.4 工具对接

4.4.1 JDBC 配置

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

连接参数

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

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

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

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

  【建议】通过JDBC连接数据库时，会设置extra_float_digits=3，gsql中设置为extra_float_digits=0，可能会使同一条数据在JDBC显示和gsql显示的精度不同。

  【建议】对于精度敏感的场景，建议使用numeric类型。

- 【建议】通过JDBC连接数据库时，应该保证下面三个时区设置一致：
  - JDBC客户端所在主机的时区。
  - GaussDB集群所在主机的时区。
  - GaussDB集群配置过程中时区。

    说明

    时区设置相关的操作请联系管理员处理。

fetchsize

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

autocommit

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

释放连接

【建议】推荐使用连接池限制应用程序的连接数。每执行一条SQL就连接一次数据库，是一种不好的SQL编写习惯。

【建议】在应用程序完成作业任务之后，应当及时断开和GaussDB的连接，释放资源。建议在任务中设置session超时时间参数。

【建议】使用JDBC连接池，在将连接释放给连接池前，需要执行以下操作，重置会话环境。否则，可能会因为历史会话信息导致的对象冲突。

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

CopyManager

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

4.5 SQL 编写

DDL

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

数据加载和卸载

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

类型转换

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

查询操作

- 【建议】除ETL程序外，应该尽量避免向客户端返回大量结果集的操作。如果结果集过大，应考虑业务设计是否合理。

- 【建议】使用事务方式执行DDL和DML操作。例如，TRUNCATE TABLE、UPDATE TABLE、DELETE TABLE、DROP TABLE等操作，一旦执行提交就无法恢复。对于这类操作，建议使用事务进行封装，必要时可以进行回滚。
- 【建议】在查询编写时，建议明确列出查询涉及的所有字段，不建议使用“SELECT *”这种写法。一方面基于性能考虑，尽量减少查询输出列；另一方面避免增删字段对前端业务兼容性的影响。
- 【建议】在访问表对象时带上Schema前缀，可以避免因Schema切换导致访问到非预期的表。
- 【建议】超过3张表或视图进行关联（特别是FULL JOIN）时，执行代价难以估算。建议使用WITH TABLE AS语句创建中间临时表的方式增加SQL语句的可读性。
- 【建议】尽量避免使用笛卡尔积和FULL JOIN。这些操作会造成结果集的急剧膨胀，同时其执行性能也很低。
- 【关注】NULL值的比较只能使用IS NULL或者IS NOT NULL的方式判断，其他任何形式的逻辑判断都返回NULL。例如：NULL<>NULL、NULL=NULL和NULL<>1返回结果都是NULL，而不是期望的布尔值。
- 【关注】需要统计表中所有记录数时，不要使用count(col)来替代count(*)。count(*)会统计NULL值（真实行数），而count(col)不会统计。
- 【关注】在执行count(col)时，将“值为NULL”的记录行计数为0。在执行sum(col)时，当所有记录都为NULL时，最终将返回NULL；当不全为NULL时，“值为NULL”的记录行将被计数为0。
- 【关注】count(多个字段)时，多个字段名必须用圆括号括起来。例如，count((col1,col2,col3))。注意：通过多字段统计行数时，即使所选字段都为NULL，该行也被计数，效果与count(*)一致。
- 【关注】count(distinct col)用来计算该列不重复的非NULL的数量，NULL将不被计数。
- 【关注】count(distinct (col1,col2,...))用来统计多列的唯一值数量，当所有统计字段都为NULL时，也会被计数，同时这些记录被认为是相同的。
- 【建议】使用连接操作符“||”替换concat函数进行字符串连接。因为concat函数的输出跟data type有关，生成执行计划时不能提前计算结果值，导致查询性能严重劣化。
- 【建议】使用下面时间相关的宏替换now函数来获取当前时间。因为now函数生成的执行计划无法下推，导致查询性能严重劣化。

表 4-3 时间相关的宏

宏名称	描述	示例
CURRENT_DATE	获取当前日期，不包含时分秒。	openGauss=# SELECT CURRENT_DATE; date ------------ 2018-02-02 (1 row)
CURRENT_TIME	获取当前时间，不包含年月日。	openGauss=# SELECT CURRENT_TIME; timetz -------------------- 00:39:34.633938+08 (1 row)

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

- 【建议】尽量避免标量子查询语句的出现。标量子查询是出现在select语句输出列表中的子查询，在下面例子中，下划线部分即为一个标量子查询语句：
  ```
  SELECT id, (SELECT COUNT(*) FROM films f WHERE f.did = s.id) FROM staffs_p1 s;
  ```
  标量子查询往往会导致查询性能急剧劣化，在应用开发过程中，应当根据业务逻辑，对标量子查询进行等价转换，将其写为表关联。
- 【建议】在WHERE子句中，应当对过滤条件进行排序，把选择读较小（筛选出的记录数较少）的条件排在前面。
- 【建议】WHERE子句中的过滤条件，尽量符合单边规则。即把字段名放在比较条件的一边，优化器在某些场景下会自动进行剪枝优化。形如col op expression，其中col为表的一个列，op为‘=’、‘>’的等比较操作符，expression为不含列名的表达式。例如，
  ```
  SELECT id, from_image_id, from_person_id, from_video_id FROM face_data WHERE
  current_timestamp(6) - time < '1 days'::interval;
  ```
  改写为：
  ```
  SELECT id, from_image_id, from_person_id, from_video_id FROM face_data where time >
  current_timestamp(6) - '1 days'::interval;
  ```
- 【建议】尽量避免不必要的排序操作。排序需要耗费大量的内存及CPU，如果业务逻辑许可，可以组合使用ORDER BY和LIMIT，减小资源开销。GaussDB默认按照ASC & NULL LAST进行排序。
- 【建议】使用ORDER BY子句进行排序时，显式指定排序方式（ASC/DESC），NULL的排序方式（NULL FIRST/NULL LAST）。
- 【建议】不要单独依赖LIMIT子句返回特定顺序的结果集。如果部分特定结果集，可以将ORDER BY子句与LIMIT子句组合使用，必要时也可以使用OFFSET跳过特定结果。
- 【建议】在保障业务逻辑准确的情况下，建议尽量使用UNION ALL来代替UNION。
- 【建议】如果过滤条件只有OR表达式，可以将OR表达式转化为UNION ALL以提升性能。使用OR的SQL语句经常无法优化，导致执行速度慢。例如，将下面语句
  ```
  SELECT * FROM scdc.pub_menu
  WHERE (cdp= 300 AND inline=301) OR (cdp= 301 AND inline=302) OR (cdp= 302 ANDinline=301);
  ```
  转换为：
  ```
  SELECT * FROM scdc.pub_menu
  WHERE (cdp= 300 AND inline=301)
  union all
  SELECT * FROM scdc.pub_menu
  WHERE (cdp= 301 AND inline=302)
  union all
  SELECT * FROM tablename
  WHERE (cdp= 302 AND inline=301)
  ```
- 【建议】当IN(val1, val2, val3…)表达式中字段较多时，建议使用IN (VALUES (val1), (val2),(val3)…)语句进行替换。优化器会自动把IN约束转换为非关联子查询，从而提升查询性能。

- 【建议】在关联字段不存在NULL值的情况下，使用(NOT) EXIST代替(NOT) IN。例如，在下面查询语句中，当T1.C1列不存在NULL值时，可以先为T1.C1字段添加NOT NULL约束，再进行如下改写。

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

  可以改写为：

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

  说明

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

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

5 应用程序开发教程

5.1 开发规范

如果用户在APP的开发中，使用了连接池机制，那么需要遵循如下规范：

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

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

应用程序开发驱动兼容性说明如表5-1所示：

表 5-1 兼容性说明

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

在多线程环境下使用驱动：

JDBC驱动程序不是线程安全的，不保证连接上的方法是同步的。由调用者来同步对驱动程序的调用。

5.2 驱动包获取

获取驱动包

单击此处获取GaussDB驱动包“GaussDB_driver.zip”。

单击此处获取GaussDB驱动包校验包“GaussDB_driver.zip.sha256”。

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

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

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

   如果回显OK，则校验通过。

   ```
   GaussDB_driver.zip: OK
   ```

5.3 基于 JDBC 开发

JDBC（Java Database Connectivity，java数据库连接）是一种用于执行SQL语句的Java API，可以为多种关系数据库提供统一访问接口，应用程序可基于它操作数据。GaussDB库提供了对JDBC 4.0特性的支持，需要使用JDK1.8版本编译程序代码，不支持JDBC桥接ODBC方式。

5.3.1 JDBC 包、驱动类和环境类

JDBC 包

从发布包中获取。包名为GaussDB-Kernel_VxxxRxxxCxx-操作系统版本号-64bit-Jdbc.tar.gz。

解压后JDBC的驱动jar包：

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

须知

- 各驱动包只是驱动类加载路径和url前缀不同，接口功能上相同。
- jdbc发布件jar包按照架构分类，gscejdbc.jar包必须与对应的部署环境一致才能使用，其他jar包无需与部署环境一致。
- gsjdbc200.jar：该驱动包适用于从Gauss200迁移业务的场景，驱动类和加载路径与迁移前保持一致，但接口支持情况不完全一致，未支持的接口需要业务侧进行调整。
- 不能使用gsjdbc4的驱动包操作PostgreSQL数据库，虽然部分版本能够建连成功，但部分接口行为与PostgreSQL JDBC不同，可能导致未知错误。
- 不能使用PostgreSQL的驱动包操作GaussDB数据库，虽然部分版本能够建连成功，但部分接口行为与GaussDB JDBC不同，可能导致未知错误。

驱动类

在创建数据库连接之前，需要加载数据库驱动类“org.postgresql.Driver”（对应包gsjdbc4.jar）。

说明

1. 由于GaussDB在JDBC的使用上与PG的使用方法保持兼容，所以同时在同一进程内使用两个JDBC的驱动的时候，可能会类名冲突。
2. 本版本JDBC不再支持IAM认证功能。
3. 相比于PG驱动，GaussDB JDBC驱动主要做了以下特性的增强：
   1. 支持SHA256加密方式登录。
   2. 支持对接实现sf4j接口的第三方日志框架。
   3. 支持连接级别的分布式负载均衡。
   4. 支持容灾切换。

环境类

客户端需配置JDK1.8。JDK是跨平台的，支持Windows、Linux等多种平台，下面以Windows为例，介绍JDK配置流程：

步骤1 DOS窗口（windows下的命令提示符）输入“java -version”，查看JDK版本，确认为JDK1.8版本。如果未安装JDK，请下载安装包并安装。

步骤2 在windows操作系统桌面中“此电脑”图标上单击右键，选择“属性”。

步骤3 在弹出的“系统”窗口中，单击左侧导航栏中“高级系统设置”。

步骤4 在弹出的“系统属性”窗口中，单击右下角的“环境变量”。

步骤5 在弹出的“环境变量”窗口中的“系统变量”区域框中设置如下变量名和变量值。

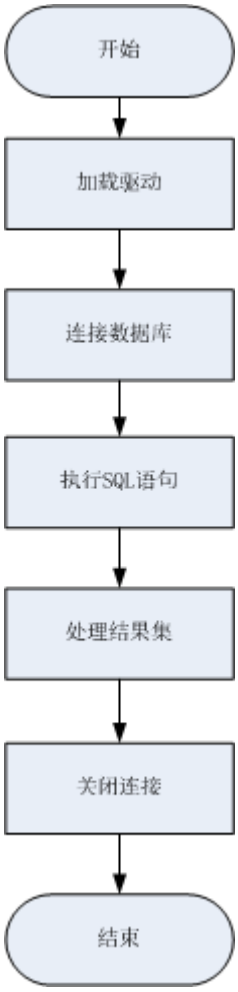
变量名	操作	变量值
JAVA_HOME	• 若存在，则单击“编辑”。 • 若不存在，则单击“新建”。	JAVA的安装目录。 例如：C:\Program Files\Java\jdk1.8.0_131
Path	编辑	• 若配置了JAVA_HOME，则在变量值的最前面加上：%JAVA_HOME%\bin; • 若未配置JAVA_HOME，则在变量值的最前面加上 JAVA安装的全路径： C:\Program Files\Java\jdk1.8.0_131\bin;
CLASSPATH	新建	.;%JAVA_HOME%\lib;%JAVA_HOME%\lib\tools.jar;

步骤6 单击“确定”，并依次关闭各窗口。

----结束

5.3.2 开发流程

图 5-1 采用 JDBC 开发应用程序的流程



5.3.3 加载驱动

在创建数据库连接之前，需要先加载数据库驱动程序。

加载驱动有两种方法：

- 在代码中创建连接之前任意位置隐含装载：
  Class.forName("org.postgresql.Driver");
- 在JVM启动时参数传递：java -Djdbc.drivers=org.postgresql.Driver jdbctest

  说明

  - 上述jdbctest为测试用例程序的名称。
  - 当使用opengaussjdbc.jar时，上面的Driver类名相应修改为“com.huawei.opengauss.jdbc.Driver”。

5.3.4 连接数据库

在创建数据库连接之后，才能使用它来执行SQL语句操作数据。

函数原型

JDBC提供了三个方法，用于创建数据库连接。

- DriverManager.getConnection(String url);
- DriverManager.getConnection(String url, Properties info);
- DriverManager.getConnection(String url, String user, String password);

参数

表 5-2 数据库连接参数

参数	描述
url	gsjdbc4.jar数据库连接描述符。格式如下： ● jdbc:postgresql:database ● jdbc:postgresql://host/database(端口值缺省会使用默认端口） ● jdbc:postgresql://host:port/database ● jdbc:postgresql://host:port/database?param1=value1&param2=value2 ● jdbc:postgresql://host1:port1,host2:port2/database?param1=value1&param2=value2 **说明** 使用gsjdbc200.jar时，将“jdbc:postgresql”修改为“jdbc:gaussdb” ● database为要连接的数据库名称。 ● host为数据库服务器名称或IP地址。 由于安全原因，数据库CN禁止集群内部其他节点无认证接入。如果要在集群内部访问CN，请将JDBC程序部署在CN所在机器，host使用"127.0.0.1"。否则可能会出现“FATAL: Forbid remote connection with trust method!”错误。 建议业务系统单独部署在集群外部，否则可能会影响数据库运行性能。 缺省情况下，连接服务器为localhost。 ● port为数据库服务器端口。 缺省情况下，会尝试连接到5431端口的database。 ● param为参数名称，即数据库连接属性。 参数可以配置在URL中，以"?"开始配置，以"="给参数赋值，以"&"作为不同参数的间隔。也可以采用info对象的属性方式进行配置，详细示例会在本节给出。 ● value为参数值，即数据库连接属性值。 ● 连接时需配置connectTimeout和socketTimeout，如果未配置，默认为0，即不会超时。在DN与客户端出现网络故障时，客户端一直未收到DN侧ACK确认报文，会启动超时重传机制，不断地进行重传。当重传次数达到默认的15次后才会报超时错误，会导致RTO时间很高。 ● 分布式环境下，连接串建议配置autoBalance参数进行负载均衡，同时配置至少两个CN节点，避免因节点故障无法正常建连。

参数	描述
info	数据库连接属性（所有属性大小写敏感）。常用的属性如下： • PGDBNAME：String类型。表示数据库名称（URL中无需配置该参数，自动从URL中解析）。 • PGHOST：String类型。主机IP地址。若配置多个CN，它们的IP和端口用“:”分隔，并作为整体以逗号分隔其他CN（URL中无需配置该参数，自动从URL中解析）。 • PGPORT：Integer类型。主机端口号。若配置多个CN，它们的端口号和IP用":"分隔，并作为整体以逗号分隔其他CN（URL中无需配置该参数，自动从URL中解析）。 • user：String类型。表示创建连接的数据库用户。 • password：String类型。表示数据库用户的密码。 • loggerLevel：String类型。缺省值为NULL，与设置为INFO等效。目前支持4种级别：OFF、INFO、DEBUG、TRACE。设置为OFF关闭日志。设置为INFO、DEBUG和TRACE记录的日志信息详细程度不同。 • loggerFile：String类型。用于指定日志输出路径（目录和文件名）。若只指定文件名，未指定目录则日志生成在客户端运行程序目录；若不配置或配置的路径不存在，则日志会默认通过流输出。此参数已废弃，不再生效，如需使用可通过 java.util.logging 属性文件或系统属性进行配置。 • logger：String类型。表示JDBC Driver要使用的日志输出框架。JDBC Driver支持对接用户应用程序使用的日志输出框架。目前支持的第三方日志输出框架只有基于Slf4j-API的日志框架。具体使用方式，见**6.2.9日志管理。** 1. 如果缺省或设置为JDK LOGGER，则JDBC Driver使用JDK LOGGER。 2. 否则必须设置采用基于slf4j-API 第三方日志框架。 • allowEncodingChanges：Boolean类型，缺省值为false。设置该参数值为“true”进行字符集类型更改，配合参数characterEncoding=CHARSET设置字符集，二者使用"&"分隔；characterEncoding取值范围{UTF8、GBK、LATIN1、GB18030}。 • currentSchema：String类型。在search-path中指定要设置的schema。如果Schema名包含除字母、数字、下划线之外的特殊字符，建议在Schema名上加引号，注意加引号后Schema名大小写敏感。如需配置多个Schema，要用逗号（,）进行分隔，包含特殊字符的Schema也需要加引号处理。 例如：currentSchema=schema_a,"schema-b","schema/c"。 • loadBalanceHosts：Boolean类型。在默认模式下（禁用），顺序连接URL中指定的多个主机。如果启用，则使用洗牌算法从候选主机中随机选择一个主机建立连接。 • autoBalance：String类型。 1. 设置为true或balance或roundrobin表示开启JDBC负载均衡功能，将应用程序的多个连接均衡到数据库集群中的各个可用CN。 例如：jdbc:postgresql://host1:port1,host2:port2/database?autoBalance=true JDBC将定期获取（周期刷新可使用参数refreshCNIpListTime配置，默认为10s）整个集群可用CN列表（注意CN列表中获取的host是数据IP），

参数	描述
	比如获取到的列表为： host1:port1,host2:port2,host3:port3,host4:port4。 host1和host2在autoBalance启用时，仅在首次连接做高可用用途，后续Driver将从host1，host2，host3，host4中依次选择可用的CN刷新可用CN列表，后续用户新建的connection将使用RoundRobin算法从host1，host2，host3，host4选取CN主机进行连接。 2. 设置为priorityn表示开启JDBC优先级负载均衡功能，将应用程序的多个连接首先均衡到url上配置的前n个中可用的CN数据库节点，当url上配置前n个节点全部不可用时，连接会随机分配到数据库集群中其他可用CN数据库节点。n为数字，不小于0，且小于url上配置的CN数量。 例如：jdbc:postgresql:// host1:port1,host2:port2,host3:port3,host4:port4/database?autoBalance=priority2 JDBC将定期获取（周期按refreshCNIpListTime定义）整个集群可用CN列表，比如获取到的列表为： host1:port1,host2:port2,host3:port3,host4:port4,host5:port5,host6:port6，其中host1和host2处于AZ1，host3和host4处于AZ2。 Driver将从优先从host1,host2中做负载均衡，host1和host2全部不可用才从host3, host4, host5, host6中随机选择CN主机连接。 3. 设置为shuffle表示开启JDBC随机负载均衡功能，将应用程序的多个连接随机均衡到数据库集群中的各个可用CN。 例如：jdbc:postgresql://host1:port1,host2:port2,host3:port3/database?autoBalance=shuffle JDBC将定期获取(周期刷新可使用参数refreshCNIpListTime配置，默认为10S)整个集群的可用CN列表，比如获取到的列表为： host1:port1,host2:port2,host3:port3,host4:port4。 host1:port1,host2:port2,host3:port3,仅在首次连接做高可用，后续连接将在刷新后的CN列表中，使用shuffle算法随机选用一个CN节点进行连接。 4. 设置为false，不开启JDBC负载均衡功能和优先级负载均衡功能。默认为false。 **注意** 1. 负载均衡是基于连接级别，不是基于事务级别。如果连接是长连接，并且连接上的负载不均衡，无法保证CN主机上的负载是均衡的。 2. 负载均衡仅能在分布式场景下使用。 3. 在开启负载均衡时，URL中可以配置浮动IP或数据IP，如果配置为浮动IP，系统会根据浮动IP获取对应的数据IP，通过获取的数据IP做负载均衡。因此URL中配置浮动IP或数据IP时，都需要确保数据IP网络连接正常，否则负载均衡功能异常。 ● refreshCNIpListTime：Integer类型。获取CN列表的缓存有效时间。JDBC在建连时会检测数据库集群中CN状态，在refreshCNIpListTime时间内可信。超过则状态失效，下次建连时再次获取可用CN的IP列表。默认为10秒。 ● hostRecheckSeconds：Integer类型。JDBC尝试连接主机后会保存主机状态：连接成功或连接失败。在hostRecheckSeconds时间内保持可信，超过则状态失效。缺省值是10秒。

参数	描述
	• ssl：Boolean类型。以SSL方式连接。 ssl=true可支持NonValidatingFactory通道和使用证书的方式： 1、NonValidatingFactory通道需要配置用户名和密码，同时将SSL设置为true。 2、配置客户端证书、密钥、根证书，将SSL设置为true。 • sslmode：String类型。SSL认证方式。取值范围为：disable、allow、prefer、require、verify-ca、verify-full。 – disable：不使用SSL安全连接。 – allow：如果数据库服务器要求使用，则可以使用SSL安全加密连接，但不验证数据库服务器的真实性。 – prefer：如果数据库支持，那么首选使用SSL连接，但不验证数据库服务器的真实性。 – require只尝试SSL连接，不会检查服务器证书是否由受信任的CA签发，且不会检查服务器主机名与证书中的主机名是否一致。 – verify-ca只尝试SSL连接，并且验证服务器是否具有由可信任的证书机构签发的证书。 – verify-full只尝试SSL连接，并且验证服务器是否具有由可信任的证书机构签发的证书，以及验证服务器主机名是否与证书中的一致。 • sslcert：String类型。提供证书文件的完整路径。客户端和服务端证书的类型为End Entity。 • sslkey：String类型。提供密钥文件的完整路径。如果客户端证书不是DER格式，使用时将客户端证书转换为DER格式，生成方式参考**连接数据库（以SSL方式）**章节。 • sslrootcert：String类型。SSL根证书的文件名。根证书的类型为CA。 • sslpassword：String类型。提供给ConsoleCallbackHandler使用。 • sslpasswordcallback：String类型。SSL密码提供者的类名。缺省值：org.postgresql.ssl.jdbc4.LibPQFactory.ConsoleCallbackHandler。 • sslfactory：String类型。提供的值是SSLSocketFactory在建立SSL连接时用的类名。 • sslprivatekeyfactory: String类型。提供的值是实现私钥解密方法的接口org.postgresql.ssl.PrivateKeyFactory的实现类的完整限定类名。如果不提供，首先尝试默认的jdk私钥解密算法，如果无法解密，则使用org.postgresql.ssl.BouncyCastlePrivateKeyFactory，用户需要自己提供bcpkix-jdk15on.jar包，版本建议：1.65以上。 • sslfactoryarg：String类型。此值是上面提供的sslfactory类的构造函数的可选参数（不推荐使用本参数）。 • sslhostnameverifier：String类型。主机名验证程序的类名。接口实现javax.net.ssl.HostnameVerifier，默认使用org.postgresql.ssl.PGjdbcHostnameVerifier。 • loginTimeout：Integer类型。指建立数据库连接的等待时间。超时时间单位为秒。当url配置多IP时，若获取连接花费的时间超过此值，则连接失败，不再尝试后续IP。

参数	描述
	• connectTimeout：Integer类型。用于连接服务器操作的超时值。如果连接到服务器花费的时间超过此值，则连接断开。超时时间单位为秒，值为0时表示已禁用，timeout不发生。当url配置多IP时，表示连接单个IP的超时时间。 • socketTimeout：Integer类型。用于socket读取操作的超时值。如果从服务器读取所花费的时间超过此值，则连接关闭。超时时间单位为秒，值为0时表示已禁用，timeout不发生。 当JDBC侧触发超时且连接关闭时，其下发给数据库侧正在运行的业务会被强制终止。该能力受GUC参数check_disconnect_query控制，设置为on表示支持该能力，设置为off表示不支持该能力。 • socketTimeoutInConnecting：Integer类型。用于控制建立连接阶段socket读取操作的超时值。如果建立连接时从服务器读取所花费的时间超过此值，则查找下一个节点建立连接。超时时间单位为秒，默认为5s。 • driverInfoMode：String类型。用于控制驱动描述信息的输出模式。取值范围为postgresql、gaussdb，默认缺省值为postgresql，输出postgresql相关的驱动描述信息；设置为gaussdb时输出gaussdb相关的驱动描述信息。 • cancelSignalTimeout：Integer类型。发送取消消息本身可能会阻塞，用于控制取消命令的“connect超时”和“socket超时”。如果取消命令超过指定时间未响应，会中断这个连接，减少占用客户端资源。超时时间单位为秒，默认值为10秒。 • tcpKeepAlive：Boolean类型。启用或禁用TCP保活探测功能。默认为false。 • logUnclosedConnections：Boolean类型，缺省值为false。客户端可能由于未调用Connection对象的close()方法而泄漏Connection对象。最终这些对象将被垃圾回收，并且调用finalize()方法。设置为true之后，如果调用者自己忽略了此操作，该方法将关闭Connection。 • assumeMinServerVersion（废弃）：String类型。该参数设置要连接的服务器版本。 • ApplicationName：String类型。设置正在使用连接的应用程序名称。通过在CN上查询pgxc_stat_activity表可以看到正在连接的客户端信息，显示在application_name列。缺省值为PostgreSQL JDBC Driver。 • connectionExtraInfo：Boolean类型。表示驱动是否上报当前驱动的部署路径、进程属主用户、url连接配置信息到数据库。 取值范围：true或false，默认值为false。设置connectionExtraInfo为true，JDBC驱动会将当前驱动的部署路径、进程属主用户、url连接配置信息上报到数据库中，记录在connection_info参数里；同时可以在PG_STAT_ACTIVITY和PGXC_STAT_ACTIVITY中查询到。 • autosave：String类型。共有3种："always", "never", "conservative"。如果查询失败，指定驱动程序应该执行的操作。在autosave=always模式下，JDBC驱动程序在每次查询之前设置一个保存点，并在失败时回滚到该保存点。在autosave=never模式（默认）下，无保存点。在autosave=conservative模式下，每次查询都会设置保存点，但是只会在“statement XXX无效”等情况下回滚并重试。 • protocolVersion：Integer类型。连接协议版本号，目前仅支持1和3。注意：设置1时仅代表连接的是V1服务端。设置3时将采用md5加密方式，需

参数	描述
	要同步修改数据库的加密方式，将GUC参数password_encryption_type设置为1，重启集群生效后需要创建用md5方式加密口令的用户。同时修改pg_hba.conf，将客户端连接方式修改为md5。用新建用户进行登录（因为设置这个值后，只能使用低等级的加密方式（md5），降低安全性，所以此值不推荐设置）。 **说明** MD5加密算法安全性低，存在安全风险，建议使用更安全的加密算法。 ● prepareThreshold：Integer类型。该值决定着PreparedStatement对象在执行多少次以后使用服务端已经准备好的statement。默认值是5，意味着在执行同一个PreparedStatement对象时，在第五次以及以上执行时不再向服务端发送parse消息对statement进行解析，而使用之前在服务端已经解析好的statement。 ● preparedStatementCacheQueries：Integer类型。该参数确定了每个连接的cache缓存Statement对象生成query的最大个数。默认值为256，若Statement对象生成query个大于256则会将最近最少使用的query从缓存中丢弃。0表示禁用缓存。 ● preparedStatementCacheSizeMiB：Integer类型，该参数确定了每个连接的cache缓存Statement对象所生成query的最大值（以兆字节为单位），默认情况下是5。若缓存了超过5MB的query，则最近最少使用的查询缓存将被丢弃。0表示禁用缓存。 ● databaseMetadataCacheFields：Integer类型。默认值是65536。指定每个连接可缓存的最大字段的个数。“0”表示禁用缓存。 ● databaseMetadataCacheFieldsMiB：Integer类型。默认值是5。指定每个连接可缓存的字段的最大值，单位是MB。“0”表示禁用缓存。 ● stringtype：String类型，可选字段为："unspecified", "varchar"。设置通过setString()方法使用的PreparedStatement参数的类型，如果stringtype设置为VARCHAR（默认值），则这些参数将作为varchar参数发送给服务器。若stringtype设置为unspecified，则参数将作为untyped值发送到服务器，服务器将尝试推断适当的类型。 ● batchMode：String类型。用于确定是否使用batch模式连接。默认值为on，表示开启batch模式。设置batchMode=on执行成功的返回结果为[count, 0, 0...0]，数组第一个元素为批量影响的总条数，设置batchMode=off执行成功的返回结果为[1, 1, 1...1]，数组各元素对应单次修改的影响条数。 ● fetchsize：Integer类型。用于设置数据库连接所创建statement的默认fetchsize。默认值为0，表示一次获取所有结果。与defaultRowFetchSize等价，如果同时设置，以fetchsize为准。 ● reWriteBatchedInserts：Boolean类型。批量导入时，该参数设置为true，可将N条插入语句合并为一条：insert into TABLE_NAME values(values1, ..., valuesN), ..., (values1, ..., valuesN);使用该参数时，需设置batchMode=off。 ● unknownLength：Integer类型，默认为Integer.MAX_VALUE。某些PostgreSQL类型（例如TEXT）没有明确定义的长度，当通过ResultSetMetaData.getColumnDisplaySize和ResultSetMetaData.getPrecision等函数返回关于这些类型的数据时，此参数指定未知长度类型的长度。

参数	描述
	• defaultRowFetchSize：Integer类型。确定一次fetch在ResultSet中读取的行数。限制每次访问数据库时读取的行数可以避免不必要的内存消耗，从而避免OutOfMemoryException。缺省值是0，这意味着ResultSet中将一次获取所有行。本参数不允许设置为负值。 • binaryTransfer：Boolean类型。使用二进制格式发送和接收数据，默认值为“false”。 • binaryTransferEnable：String类型。启用二进制传输的类型列表，以逗号分隔。OID编号和名称二选一，例如binaryTransferEnable=INT4_ARRAY,INT8_ARRAY。 比如：OID名称为BLOB，编号为88，可以如下配置： binaryTransferEnable=BLOB 或 binaryTransferEnable=88 • binaryTransferDisable：String类型。禁用二进制传输的类型列表，以逗号分隔。OID编号和名称二选一。覆盖binaryTransferEnable的设置。 • blobMode：String类型。用于设置setBinaryStream方法绑定参数的数据类型，当该值为on时表示setBinaryStream绑定的数据类型为blob类型，为off时表示绑定的数据类型为bytea类型，默认为on。建议从Oracle、Mysql迁移来的系统将该值设定为on，从PostgreSQL迁移来的系统设定为off。 • socketFactory：String类型。用于创建与服务器socket连接的类的名称。该类必须实现接口“javax.net.SocketFactory”，并定义无参或单String参数的构造函数。 • socketFactoryArg：String类型。此值是上面提供的socketFactory类的构造函数的可选参数，不推荐使用。 • receiveBufferSize：Integer类型。该值用于设置连接流上的SO_RCVBUF。 • sendBufferSize：Integer类型。该值用于设置连接流上的SO_SNDBUF。 • preferQueryMode：String类型。共有4种："extended", "extendedForPrepared", "extendedCacheEverything", "simple"。用于指定执行查询的模式，默认值为extended。simple模式只发送Q消息，仅支持文本模式，不支持parse与bind；extended模式会使用parse、bind和execute消息；extendedForPrepared模式下只有Prepared Statement对象使用扩展查询，Statement对象只使用简单查询；extendedCacheEverything模式会缓存每个Statement对象所生成的query。 • ApplicationType：String类型。共有2种："not_perfect_sharding_type"，"perfect_sharding_type"。用于设置是否开启分布式写入和查询，默认值为"not_perfect_sharding_type"。not_perfect_sharding_type模式下开启分布式写入和查询；perfect_sharding_type模式下默认禁止分布式写入和查询，只有在sql文中加入/* multinode */ 才能执行分布式写入和查询。该项设置只有数据库处于gtm free场景的情况下才会有效。 • priorityServers：Integer类型。此值用于指定url上配置的前n个节点作为主集群被优先连接。默认值为null。该值为数字，大于0，且小于url上配置的CN数量。用于流式容灾场景。 例如：jdbc:postgresql://host1:port1,host2:port2,host3:port3,host4:port4,/database?priorityServers=2。即表示host1与host2为主集群节点，host3与host4为容灾集群节点。

参数	描述
	• usingEip：Boolean类型。此值用于控制是否使用弹性公网IP做负载均衡。默认值为true，表示使用弹性公网IP做负载均衡；false表示使用数据IP做负载均衡。 • traceInterfaceClass：String类型。默认值为null，用于获取traceId的实现类。值是实现获取traceId方法的接口org.postgresql.log.Tracer的实现类的完整限定类名。 • use_boolean：Boolean类型。用于设置extended模式下setBoolean方法绑定的oid类型，默认为false，绑定int2类型；设置为true则绑定bool类型。 • allowReadOnly：Boolean类型。用于设置是否允许只读模式，默认为true，允许设置只读模式；设置为false则禁用只读模式。 • TLSCiphersSupperted：String类型。用于设置支持的TLS加密套件，默认为TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384。 • stripTrailingZeros：Boolean类型。默认值为false，设置为true则去除numeric类型后尾随的0，仅对ResultSet.getObject(int columnIndex)生效。 • enableTimeZone：Boolean类型。默认值为true，用于指定是否启用服务端时区设置，true表示获取JVM时区指定数据库时区，false表示使用数据库时区。 • socketTimeoutInConnecting：Integer类型。默认值5s，用于建立连接时socket读取操作的超时值。如果在建连过程中，从服务器读取所花费的时间超过此值，则连接关闭。超时时间单位为秒，值为0时表示已禁用，timeout不发生。 • driverInfoMode：String类型。用于控制驱动描述信息的输出模式。取值范围为postgresql、gaussdb，默认缺省值为postgresql，输出PostgreSQL相关的驱动描述信息，设置为gaussdb时输出gaussdb相关的驱动描述信息。
user	数据库用户。
password	数据库用户的密码。

示例

```
//以下用例以gsjdbc4.jar为例。
//以下代码将获取数据库连接操作封装为一个接口，可通过给定用户名和密码来连接数据库。
public static Connection getConnect(String username, String passwd)
  {
    //驱动类。
    String driver = "org.postgresql.Driver";
    //数据库连接描述符。
```



```
    String sourceURL = "jdbc:postgresql://$ip:$port/postgres";
    Connection conn = null;

    try
    {
        //加载驱动。
        Class.forName(driver);
    }
    catch( Exception e )
    {
        e.printStackTrace();
        return null;
    }

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

    return conn;
  }
// 以下代码将使用Properties对象作为参数建立连接
public static Connection getConnectUseProp(String username, String passwd)
  {
    //驱动类。
    String driver = "org.postgresql.Driver";
    //数据库连接描述符。
    String sourceURL = "jdbc:postgresql://$ip:$port/postgres?autoBalance=true";
    Connection conn = null;
    Properties info = new Properties();

    try
    {
        //加载驱动。
        Class.forName(driver);
    }
    catch( Exception e )
    {
        e.printStackTrace();
        return null;
    }

    try
    {
         info.setProperty("user", userName);
         info.setProperty("password", password);
         //创建连接。
         conn = DriverManager.getConnection(sourceURL, info);
         System.out.println("Connection succeed!");
    }
    catch(Exception e)
    {
        e.printStackTrace();
        return null;
    }

    return conn;
  }
```


5.3.5 连接数据库（以 SSL 方式）

用户通过JDBC连接GaussDB服务器时，可以通过开启SSL加密客户端和服务器之间的通讯，为敏感数据在Internet上的传输提供了一种安全保障手段。

本小节主要介绍应用程序通过JDBC如何采用SSL的方式对客户端进行配置（服务端配置请联系管理员处理）。

在使用本小节所描述的方法前，默认用户已经获取了服务端和客户端所需要的证书和私钥文件，关于证书等文件的获取请参考OpenSSL相关文档和命令 。

客户端配置

上传证书文件，将在服务端配置章节生成的文件client.key.pk8、client.crt、cacert.pem放置在客户端，具体路径在代码中指定。

示例

注：示例1和示例2选择其一。

```
// 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放(密码应密文存放，使用时解密)，确保安全；
// 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量(环境变量名称请根据自身情况进行设置)EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
public class SSL{
    public static void main(String[] args) {
        Properties urlProps = new Properties();
        String urls = "jdbc:postgresql://$ip:$port/postgres";
        String userName = System.getenv("EXAMPLE_USERNAME_ENV");
        String password = System.getenv("EXAMPLE_PASSWORD_ENV");

        /**
         * ================== 示例1 使用NonValidatingFactory通道
         */
        urlProps.setProperty("sslfactory","org.postgresql.ssl.NonValidatingFactory");
        urlProps.setProperty("user", userName);
        urlProps.setProperty("password", password);
        urlProps.setProperty("ssl", "true");
        /**
         * ================== 示例2 使用证书
         */
        urlProps.setProperty("sslcert", "client.crt");
        urlProps.setProperty("sslkey", "client.key.pk8");
        urlProps.setProperty("sslrootcert", "cacert.pem");
        urlProps.setProperty("user", userName);
        urlProps.setProperty("ssl", "true");
        /* sslmode可配置为：require、verify-ca、verify-full，以下三个示例选择其一*/
        /* ================== 示例2.1 设置sslmode为require，使用证书 */
        urlProps.setProperty("sslmode", "require");
        /* ================== 示例2.2 设置sslmode为verify-ca，使用证书 */
        urlProps.setProperty("sslmode", "verify-ca");
        /* ================== 示例2.3 设置sslmode为verify-full，使用证书（Linux下验证） */
        urls = "jdbc:postgresql://world:8000/postgres";
        urlProps.setProperty("sslmode", "verify-full");

        try {
            Class.forName("org.postgresql.Driver").newInstance();
        } catch (Exception e) {
            e.printStackTrace();
        }
        try {
            Connection conn;
            conn = DriverManager.getConnection(urls,urlProps);
            conn.close();
        } catch (Exception e) {
```



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

5.3.6 连接数据库（以 UDS 方式）

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

需要引用的jar包有junixsocket-core-XXX.jar、junixsocket-common-XXX.jar、junixsocket-native-common-XXX.jar。同时需要在URL连接串中添加：socketFactory=org.newsclub.net.unix.AFUNIXSocketFactory$FactoryArg&socketFactoryArg=*[path-to-the-unix-socket]*。

示例：

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

public class Test {
    public static void main(String[] args) {
        String driver = "org.postgresql.Driver";
        String userName = System.getenv("EXAMPLE_USERNAME_ENV");
        String password = System.getenv("EXAMPLE_PASSWORD_ENV");
        Connection conn;
        try {
            Class.forName(driver).newInstance();
            Properties properties = new Properties();
            properties.setProperty("user", userName);
            properties.setProperty("password", password);
            conn = DriverManager.getConnection("jdbc:postgresql://$ip:$port/postgres?socketFactory=org.newsclub" +
                            ".net.unix" +
                            ".AFUNIXSocketFactory$FactoryArg&socketFactoryArg=/data/tmp/.s.PGSQL.8000",
                    properties);
            System.out.println("Connection Successful!");
            Statement statement = conn.createStatement();
            statement.executeQuery("select 1");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```


须知

- socketFactoryArg参数配置根据真实路径进行配置，与GUC参数unix_socket_directory的值保持一致。
- 连接主机名必须设置为“localhost”。

5.3.7 执行 SQL 语句

执行普通 SQL 语句

应用程序通过执行SQL语句来操作数据库的数据（不用传递参数的语句），需要按以下步骤执行：

步骤1 调用Connection的createStatement方法创建语句对象。

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

步骤2 调用Statement的executeUpdate方法执行SQL语句。

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

说明

- 数据库中收到的一次执行请求（不在事务块中），如果含有多条语句，将会被打包成一个事务，事务块中不支持vacuum操作。如果其中有一个语句失败，那么整个请求都将会被回滚。
- 使用Statement执行多语句时应以“;”作为各语句间的分隔符，存储过程、函数、匿名块不支持多语句执行。当preferQueryMode=simple，语句执行不走解析逻辑，此场景下无法使用";"作为多语句间的分隔符。
- “/”可用作创建单个存储过程、函数、匿名块、包体的结束符。当preferQueryMode=simple，语句执行不走解析逻辑，此场景下无法使用"/"作为结束符。
- 在prepareThreshold=1时，因为preferQueryMode默认模式不对statement进行缓存淘汰，所以statement执行的每条SQL都会缓存语句，可能导致内存膨胀。需要调整preferQueryMode=extendedCacheEverything，对statement进行缓存淘汰。

步骤3 关闭语句对象。

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

----结束

执行预编译 SQL 语句

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

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

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

步骤2 调用PreparedStatement的setShort设置参数。

```
pstmt.setShort(1, (short)2);
```

> **⚠ 注意**
>
> PrepareStatement设置绑定参数后，最终会构建成一个B报文（或U报文）在下一步执行SQL语句时发给服务端。但是B/U报文有最大长度限制（不能超过1023MB），如果一次绑定数据过大，可能因报文过长导致异常。因此在这一步需要注意评估和控制绑定数据的大小，避免超出报文上限。

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

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

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

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

----结束

调用存储过程

GaussDB支持通过JDBC直接调用事先创建的存储过程，步骤如下：

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

```
// 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放(密码应密文存放，使用时解密)，确保安全；
// 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量(环境变量名称请根据自身情况进行设置)EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
String userName = System.getenv("EXAMPLE_USERNAME_ENV");
String password = System.getenv("EXAMPLE_PASSWORD_ENV");
Connection myConn = DriverManager.getConnection("url",userName,password);
CallableStatement cstmt = myConn.prepareCall("{? = CALL TESTPROC(?,?,?)}");
```

步骤2 调用CallableStatement的setInt方法设置参数。

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

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

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

步骤4 调用CallableStatement的execute执行方法调用。

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

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

```
int out = cstmt.getInt(4);  //获取out参数
```

示例：

```
//在数据库中已创建了如下存储过程，它带有out参数。
create or replace procedure testproc
(
    psv_in1 in integer,
    psv_in2 in integer,
    psv_inout inout integer
)
as
begin
    psv_inout := psv_in1 + psv_in2 + psv_inout;
end;
/
```

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

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

说明

- 很多的数据库类如Connection、Statement和ResultSet都有close()方法，在使用完对象后应把它们关闭。要注意的是，Connection的关闭将间接关闭所有与它关联的Statement，Statement的关闭间接关闭了ResultSet。
- 一些JDBC驱动程序还提供命名参数的方法来设置参数。命名参数的方法允许根据名称而不是顺序来设置参数，若参数有默认值，则可以不用指定参数值就可以使用此参数的默认值。即使存储过程中参数的顺序发生了变更，也不必修改应用程序。目前GaussDB数据库的JDBC驱动程序不支持此方法。
- GaussDB数据库不支持带有输出参数的函数，也不支持存储过程和函数参数默认值。
- myConn.prepareCall("{? = CALL TESTPROC(?,?,?)}")，执行存储过程绑定参数时，可以按照占位符的顺序绑定参数，注册第一个参数为出参，也可以按照存储过程中的参数顺序绑定参数，注册第四个参数为出参，上述用例为此场景，注册第四个参数为出参。

须知

- 当游标作为存储过程的返回值时，如果使用JDBC调用该存储过程，返回的游标将不可用。
- 存储过程不能和普通SQL在同一条语句中执行。
- 存储过程中inout类型参数必须注册出参。

----结束

执行批处理

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

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

```
// 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放(密码应密文存放，使用时解密)，确保安全；
// 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量(环境变量名称请根据自身情况进行设置)EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
String userName = System.getenv("EXAMPLE_USERNAME_ENV");
String password = System.getenv("EXAMPLE_PASSWORD_ENV");
Connection conn = DriverManager.getConnection("url",userName,password);
PreparedStatement pstmt = conn.prepareStatement("INSERT INTO customer_t1 VALUES (?)");
```

步骤2 针对每条数据都要调用setShort设置参数，以及调用addBatch确认该条设置完毕。

```
pstmt.setShort(1, (short)2);
pstmt.addBatch();
```

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

```
int[] rowcount = pstmt.executeBatch();
```

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

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


说明

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

----结束

在语句中添加单分片执行语法

步骤1 步骤一设置nodeName参数，通过调用Connection对象的setClientInfo("nodeName","dnx")。

```
Connection conn = getConnection();
conn.setClientInfo("nodeName","datanode1");
```

步骤2 执行SQL语句，其中包括使用Statement对象的executeQuery(String sql)和execute(String sql)以及PreparedStatement对象的executeQuery()和execute()方法

```
PreparedStatement pstm = conn.prepareStatement("select * from test");
pstm.execute();
pstm.executeQuery();
Statement stmt=conn.createStatement();
stmt.execute("select * from test");
stmt.executeQuery("select * from test");
```

步骤3 关闭参数，将参数值设置为空串

```
conn.setClientInfo("nodeName","");
```

须知

1. 该功能基于内核单分片执行功能进行的适配，所以使用前请确认使用的数据库内核是否支持单分片执行。
2. 参数开启后一定要手动关闭参数，否则会对其他查询语句的执行产生影响。
3. 参数一旦开启，当前连接所有的语句执行都会受到影响，到指定的dn上面去执行。
4. 3.参数开启后PreparedStatement对象的缓存机制会受到影响，已经缓存的语句会被清空，之后执行的带单分片查询的语句都不在缓存，直到参数关闭后缓存功能恢复。
5. 参数为连接级参数，所以在同一时间只有一个参数会生效，无法通过此接口做到同一时间两条语句到不同的分片上去执行。

----结束

5.3.8 处理结果集

设置结果集类型

不同类型的结果集有各自的应用场景，应用程序需要根据实际情况选择相应的结果集类型。在执行SQL语句过程中，都需要先创建相应的语句对象，而部分创建语句对象的方法提供了设置结果集类型的功能。具体的参数设置如表5-3所示。涉及的Connection的方法如下：

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

//创建一个PreparedStatement对象，该对象将生成具有给定类型和并发性的ResultSet对象。
```



```
prepareStatement(String sql, int resultSetType, int resultSetConcurrency);

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

表 5-3 结果集类型

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

在结果集中定位

ResultSet对象具有指向其当前数据行的光标。最初，光标被置于第一行之前。next方法将光标移动到下一行；因为该方法在ResultSet对象没有下一行时返回false，所以可以在while循环中使用它来迭代结果集。但对于可滚动的结果集，JDBC驱动程序提供更多的定位方法，使ResultSet指向特定的行。定位方法如表5-4所示。

表 5-4 在结果集中定位的方法

方法	描述
next()	把ResultSet向下移动一行。
previous()	把ResultSet向上移动一行。
beforeFirst()	把ResultSet定位到第一行之前。
afterLast()	把ResultSet定位到最后一行之后。
first()	把ResultSet定位到第一行。
last()	把ResultSet定位到最后一行。
absolute(int)	把ResultSet移动到参数指定的行数。

方法	描述
relative(int)	通过设置为1向前（设置为1，相当于next()）或者向后（设置为-1，相当于previous()）移动参数指定的行。

获取结果集中光标的位置

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

表 5-5 获取结果集光标的位置

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

获取结果集中的数据

ResultSet对象提供了丰富的方法，以获取结果集中的数据。获取数据常用的方法如表5-6所示，其他方法请参考JDK官方文档。

表 5-6 ResultSet 对象的常用方法

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

5.3.9 关闭数据库连接

在使用数据库连接完成相应的数据操作后，需要关闭数据库连接。

关闭数据库连接可以直接调用其close方法即可。

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

5.3.10 日志管理

GaussDB JDBC驱动程序支持使用日志记录来帮助解决在应用程序中使用GaussDB JDBC驱动程序时的问题。GaussDB JDBC支持如下两种日志管理方式：

1. 对接应用程序使用的SLF4J日志框架。
2. 对接应用程序使用的JdkLogger日志框架。

SLF4J和JdkLogger是业界Java应用程序日志管理的主流框架，描述应用程序如何使用这些框架超出了本文范围，用户请参考对应的官方文档（SLF4J：**http://www.slf4j.org/manual.html**，JdkLogger：**https://docs.oracle.com/javase/8/docs/technotes/guides/logging/overview.html**）。

方式一：对接应用程序的SLF4J日志框架。

在建立连接时，url配置logger=Slf4JLogger。

可采用Log4j或Log4j2来实现SLF4J。当采用Log4j实现SLF4J，需要添加如下jar包：log4j-*.jar、slf4j-api-*.jar、slf4j-log4*-*.jar，（*区分版本），和配置文件：log4j.properties。若采用Log4j2实现SLF4J，需要添加如下jar包：log4j-api-*.jar、log4j-core-*.jar、log4j-slf4j18-impl-*.jar、slf4j-api-*-alpha1.jar（*区分版本），和配置文件：log4j2.xml。

此方式支持日志管控。SLF4J可通过文件中的相关配置实现强大的日志管控功能，建议使用此方式进行日志管理。

示例：

```
public static Connection GetConnection(String username, String passwd){

    String sourceURL = "jdbc:postgresql://$ip:$port/postgres?logger=Slf4JLogger";
    Connection conn = null;

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

log4j.properties示例：

```
log4j.logger.org.postgresql=ALL, log_gsjdbc

# 默认文件输出配置
log4j.appender.log_gsjdbc=org.apache.log4j.RollingFileAppender
log4j.appender.log_gsjdbc.Append=true
log4j.appender.log_gsjdbc.File=gsjdbc.log
log4j.appender.log_gsjdbc.Threshold=TRACE
log4j.appender.log_gsjdbc.MaxFileSize=10MB
log4j.appender.log_gsjdbc.MaxBackupIndex=5
```



```
log4j.appender.log_gsjdbc.layout=org.apache.log4j.PatternLayout
log4j.appender.log_gsjdbc.layout.ConversionPattern=%d %p %t %c - %m%n
log4j.appender.log_gsjdbc.File.Encoding = UTF-8
```

log4j2.xml示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration status="OFF">
   <appenders>
      <Console name="Console" target="SYSTEM_OUT">
         <PatternLayout pattern="%d %p %t %c - %m%n"/>
      </Console>
      <File name="FileTest" fileName="test.log">
         <PatternLayout pattern="%d %p %t %c - %m%n"/>
      </File>
      <!--JDBC Driver日志文件输出配置，支持日志回卷，设定日志大小超过10MB时，创建新的文件，新文件的
命名格式为：年-月-日-文件编号-->
      <RollingFile name="RollingFileJdbc" fileName="gsjdbc.log" filePattern="%d{yyyy-MM-dd}-%i.log">
         <PatternLayout pattern="%d %p %t %c - %m%n"/>
         <Policies>
            <SizeBasedTriggeringPolicy size="10 MB"/>
         </Policies>
      </RollingFile>
   </appenders>
   <loggers>
      <root level="all">
         <appender-ref ref="Console"/>
         <appender-ref ref="FileTest"/>
      </root>
      <!--指定JDBC Driver日志，级别为：all，可查看所有日志，输出到gsjdbc.log文件中-->
      <!--当使用opengaussjdbc.jar时，将"org.postgresql"替换为"com.huawei.opengauss.jdbc.Driver"-->
      <logger name="org.postgresql" level="all" additivity="false">
         <appender-ref ref="RollingFileJdbc"/>
      </logger>
   </loggers>
</configuration>
```

须知

- 在配置文件中，可以指定JDBC与上层业务的配置路径，将JDBC日志与业务日志分别存储到不同文件。
- 同一个项目中配置多个GaussDB数据源时，如果存在部分业务配置了logger=Slf4JLogger，部分业务未配置，会互相影响日志配置。建议这种情况下连接串统一配置。

方式二：对接应用程序使用的JdkLogger日志框架。

默认的Java日志记录框架将其配置存储在名为logging.properties的文件中。Java会在Java安装目录的文件夹中安装全局配置文件。logging.properties文件也可以创建并与单个项目一起存储。

logging.properties配置示例：

```
# 指定处理程序为文件。
handlers= java.util.logging.FileHandler

# 指定默认全局日志级别
.level= ALL

# 指定日志输出管控标准
java.util.logging.FileHandler.level=ALL
java.util.logging.FileHandler.pattern = gsjdbc.log
java.util.logging.FileHandler.limit = 500000
java.util.logging.FileHandler.count = 30
```



```
java.util.logging.FileHandler.formatter = java.util.logging.SimpleFormatter
java.util.logging.FileHandler.append=false
```

代码中使用示例：

```
System.setProperty("java.util.logging.FileHandler.pattern","jdbc.log");
FileHandler fileHandler = new FileHandler(System.getProperty("java.util.logging.FileHandler.pattern"));
Formatter formatter = new SimpleFormatter();
fileHandler.setFormatter(formatter);
Logger logger = Logger.getLogger("org.postgresql");
logger.addHandler(fileHandler);
logger.setLevel(Level.ALL);
logger.setUseParentHandlers(false);
```

链路跟踪功能

GaussDB JDBC驱动程序提供了应用到数据库的链路跟踪功能，用于将数据库端离散的SQL和应用程序的请求关联起来。该功能需要应用开发者实现org.postgresql.log.Tracer接口类，并在url中指定接口实现类的全限定名。

url示例：

```
String URL = "jdbc:postgresql://$ip:$port/postgres?traceInterfaceClass=xxx.xxx.xxx.OpenGaussTraceImpl";
```

org.postgresql.log.Tracer接口类定义如下：

```
public interface Tracer {
// Retrieves the value of traceId.
String getTraceId();
}
```

org.postgresql.log.Tracer接口实现类示例：

```
import org.postgresql.log.Tracer;

public class OpenGaussTraceImpl implements Tracer {
    private static MDC mdc = new MDC();

    private final String TRACE_ID_KEY = "traceId";

    public void set(String traceId) {
        mdc.put(TRACE_ID_KEY, traceId);
    }

    public void reset() {
        mdc.clear();
    }

    @Override
    public String getTraceId() {
        return mdc.get(TRACE_ID_KEY);
    }
}
```

上下文映射示例，用于存放不同请求的生成的traceId。

```
import java.util.HashMap;

public class MDC {
    static final private ThreadLocal<HashMap<String, String>> threadLocal = new ThreadLocal<>();

    public void put(String key, String val) {
        if (key == null || val == null) {
            throw new IllegalArgumentException("key or val cannot be null");
        } else {
            if (threadLocal.get() == null) {
                threadLocal.set(new HashMap<>());
            }
            threadLocal.get().put(key, val);
        }
    }
```



```
    public String get(String key) {
        if (key == null) {
            throw new IllegalArgumentException("key cannot be null");
        } else if (threadLocal.get() == null) {
            return null;
        } else {
            return threadLocal.get().get(key);
        }
    }

    public void clear() {
        if (threadLocal.get() == null) {
            return;
        } else {
            threadLocal.get().clear();
        }
    }
}
```

业务使用traceId示例。

```
String traceId = UUID.randomUUID().toString().replaceAll("-", "");
openGaussTrace.set(traceId);
pstm = con.prepareStatement("select * from test_trace_id where id = ?");
pstm.setInt(1, 1);
pstm.execute();
pstm = con.prepareStatement("insert into test_trace_id values(?,?)");
pstm.setInt(1, 2);
pstm.setString(2, "test");
pstm.execute();
openGaussTrace.reset();
```

说明

- 使用链路跟踪功能时，应用层链路功能由业务保证。
- 应用必须向JDBC暴露获取traceId的接口，并将该接口实现类配置到JDBC连接串中。
- 同一请求的不同SQL使用的traceId须相同。
- 应用传入的traceId不能超过32字节，否则多余字节将被截断。

5.3.11 示例：常用操作

示例 1 创建数据库连接、创建表、插入数据

此示例将演示如何基于GaussDB提供的JDBC接口开发应用程序。执行示例前，需要加载驱动，驱动的获取和加载方法请参考**JDBC包、驱动类和环境类**。

```
//DBTest.java
/*以下用例以gsjdbc4.jar为例*/
//演示基于JDBC开发的主要步骤，会涉及创建数据库、创建表、插入数据等。
// 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放(密码应密文存放，使用时解密)，确保安全。
// 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量(环境变量名称请根据自身情况进行设置)EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
// $ip、$port、database需要用户自行修改。

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

public class DBTest {
```

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

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

  return conn;
};

//执行普通SQL语句，创建customer_t1表。
public static void CreateTable(Connection conn) {
  Statement stmt = null;
  try {
    stmt = conn.createStatement();

    //执行普通SQL语句。
    int rc = stmt
        .executeUpdate("CREATE TABLE customer_t1(c_customer_sk INTEGER, c_customer_name
VARCHAR(32));");

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

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

  try {
    //生成预处理语句。
    pst = conn.prepareStatement("INSERT INTO customer_t1 VALUES (?,?)");
    for (int i = 0; i < 3; i++) {
      //添加参数。
      pst.setInt(1, i);
      pst.setString(2, "data " + i);
      pst.addBatch();
    }
    //执行批处理。
    pst.executeBatch();
    pst.close();
  } catch (SQLException e) {
    if (pst != null) {
      try {
        pst.close();
```



```
      } catch (SQLException e1) {
      e1.printStackTrace();
      }
    }
    e.printStackTrace();
  }
}

//执行预编译语句，更新数据。
public static void ExecPreparedSQL(Connection conn) {
  PreparedStatement pstmt = null;
  try {
    pstmt = conn
        .prepareStatement("UPDATE customer_t1 SET c_customer_name = ? WHERE c_customer_sk = 1");
    pstmt.setString(1, "new Data");
    int rowcount = pstmt.executeUpdate();
    pstmt.close();
  } catch (SQLException e) {
    if (pstmt != null) {
      try {
        pstmt.close();
      } catch (SQLException e1) {
        e1.printStackTrace();
      }
    }
    e.printStackTrace();
  }
}


//执行存储过程。
public static void ExecCallableSQL(Connection conn) {
  CallableStatement cstmt = null;
  try {
    // 存储过程TESTPROC需提前创建。
    cstmt=conn.prepareCall("{? = CALL TESTPROC(?,?,?)}");
    cstmt.setInt(2, 50);
    cstmt.setInt(1, 20);
    cstmt.setInt(3, 90);
    cstmt.registerOutParameter(4, Types.INTEGER);  //注册out类型的参数，类型为整型。
    cstmt.execute();
    int out = cstmt.getInt(4);  //获取out参数
    System.out.println("The CallableStatment TESTPROC returns:"+out);
    cstmt.close();
  } catch (SQLException e) {
    if (cstmt != null) {
      try {
        cstmt.close();
      } catch (SQLException e1) {
        e1.printStackTrace();
      }
    }
    e.printStackTrace();
  }
}


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

  //创建表。
  CreateTable(conn);
```



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

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

    //执行存储过程。
    ExecCallableSQL(conn);

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

  }

}
```

示例 2 客户端内存占用过多解决

此示例主要使用setFetchSize来调整客户端内存使用，它的原理是通过数据库游标来分批获取服务器端数据，但它会加大网络交互，可能会损失部分性能。

由于游标事务内有效，故需要先关闭自动提交，最后需要执行手动提交。

```
// 关闭掉自动提交
conn.setAutoCommit(false);
Statement st = conn.createStatement();

// 打开游标，每次获取50行数据
st.setFetchSize(50);
ResultSet rs = st.executeQuery("SELECT * FROM mytable");
while (rs.next())
{
    System.out.print("a row was returned.");
}
conn.commit();
rs.close();

// 关闭服务器游标。
st.setFetchSize(0);
rs = st.executeQuery("SELECT * FROM mytable");
while (rs.next())
{
    System.out.print("many rows were returned.");
}
conn.commit();
rs.close();

// Close the statement.
st.close();
conn.close();
```

执行完毕后可使用如下命令恢复自动提交：

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

示例 3 常用数据类型使用示例

```
//bit类型使用示例，注意此处bit类型取值范围[0,1]
Statement st = conn.createStatement();
String createsql = "create table if not exists t_bit(col_bit bit)";
String sqlstr = "create or replace function fun_1()\n" +
        "returns bit AS $$\n" +
        "select col_bit from t_bit limit 1;\n" +
```



```
        "$$\n" +
        "LANGUAGE SQL;";
st.execute(createsql);
st.execute(sqlstr);
CallableStatement c = conn.prepareCall("{ ? = call fun_1() }");
//注册输出类型，位串类型
c.registerOutParameter(1, Types.BIT);
c.execute();
//使用Boolean类型获取结果
System.out.println(c.getBoolean(1));

// money类型使用示例
// 表结构中包含money类型列的使用示例。
st.execute("create table t_money(id int,col1 money)");
PreparedStatement pstm = conn.prepareStatement("insert into t_money values(1,?)");
// 使用PGobject赋值，取值范围[-92233720368547758.08，92233720368547758.07]
PGobject minMoney = new PGobject();
minMoney.setType("money");
minMoney.setValue("-92233720368547758.08");
pstm.setObject(1, minMoney);
pstm.execute();
// 使用PGMoney赋值,取值范围[-9999999.99,9999999.99]
pstm.setObject(1,new PGmoney(9999999.99));
pstm.execute();
```

⚠ 注意

当前JDBC不支持调用返回数据类型为money的存储过程和函数。

示例 4 获取驱动版本

```
Driver.getGSVersion();
```

5.3.12 示例：重新执行应用 SQL

当主DN故障且10s未恢复时，GaussDB会自动将对应的备DN升主，使集群正常运行。备升主期间正在运行的作业会失败；备升主后启动的作业不会再受影响。如果要做到DN主备切换过程中，上层业务不感知，可参考此示例构建业务层SQL重试机制。执行示例前，需要加载驱动，驱动的获取和加载方法请参考**JDBC包、驱动类和环境类**。

```
//以下用例以gsjdbc4.jar为例。
// 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放(密码应密文存放，使用时解密)，确保安全。
// 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量(环境变量名称请根据自身情况进行设置)EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
// $ip、$port、database需要用户自行修改。

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

class ExitHandler extends Thread {
    private Statement cancel_stmt = null;

    public ExitHandler(Statement stmt) {
        super("Exit Handler");
        this.cancel_stmt = stmt;
    }
    public void run() {
        System.out.println("exit handle");
        try {
```



```
            this.cancel_stmt.cancel();
        } catch (SQLException e) {
            System.out.println("cancel query failed.");
            e.printStackTrace();
        }
    }
}

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

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

    return conn;
}

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


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

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

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

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

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

    try {
      //生成预处理语句。
```

```
    pst = conn.prepareStatement("INSERT INTO jdbc_test1 VALUES (?,?)");
    for (int i = 0; i < 100; i++) {
     //添加参数。
      pst.setInt(1, i);
      pst.setString(2, "data " + i);
      pst.addBatch();
    }
    //执行批处理。
    pst.executeBatch();
    pst.close();
  } catch (SQLException e) {
    if (pst != null) {
      try {
        pst.close();
      } catch (SQLException e1) {
      e1.printStackTrace();
      }
    }
    e.printStackTrace();
  }
}

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

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

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

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

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

//查询语句，执行失败重试，重试次数可配置。
public static void ExecPreparedSQL(Connection conn) throws InterruptedException {
      int maxRetryTime = 50;
      int time = 0;
      String result = null;
      do {
          time++;
          try {
System.out.println("time:" + time);
boolean ret = QueryRedo(conn);
if(ret == false){
 System.out.println("retry, time:" + time);
 Thread.sleep(10000);
```



```
 QueryRedo(conn);
}
          } catch (Exception e) {
              e.printStackTrace();
          }
       } while (null == result && time < maxRetryTime);

 }

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

   //创建表。
   CreateTable(conn);

   //批插数据。
   BatchInsertData(conn);

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

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

 }

}
```

5.3.13 示例：通过本地文件导入导出数据

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

样例程序如下，执行时需要加载GaussDB jdbc驱动，驱动的获取和加载方法请参考**JDBC包、驱动类和环境类**。

前置条件：在数据库中创建表migration_table和migration_table_1，并在migration_table表中插入一些数据。

```
//以下用例以gsjdbc4.jar为例。
// 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放(密码应密文存放，使用时解密)，确保安全。
// 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量(环境变量名称请根据自身情况进行设置)EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
// $ip、$port、database需要用户自行修改。

import java.sql.Connection;
import java.sql.DriverManager;
import java.io.IOException;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.sql.SQLException;
import org.postgresql.copy.CopyManager;
import org.postgresql.core.BaseConnection;
```



```
public class Copy{

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

     try {
          Class.forName(driver);
          conn = DriverManager.getConnection(urls, username, password);
        } catch (ClassNotFoundException e) {
            e.printStackTrace(System.out);
        } catch (SQLException e) {
            e.printStackTrace(System.out);
        }

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

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

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

 e.printStackTrace();
}

     // 将migration_table_1中的数据导出到本地文件d:/data1.txt
     try {
     copyToFile(conn, "d:/data1.txt", tablename1);
  } catch (SQLException e) {

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


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

    FileInputStream fileInputStream = null;

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



```
            e.printStackTrace();
        }
      }
    }
}

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

    FileOutputStream fileOutputStream = null;

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

5.3.14 示例：从 MYSQL 进行数据迁移

下面示例演示如何通过CopyManager从mysql进行数据迁移的过程。执行示例前，需要加载驱动，驱动的获取和加载方法请参考**JDBC包、驱动类和环境类**。

```
//以下用例以gsjdbc4.jar为例
// 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放(密码应密文存放，使用时解密)，确保安全。
// 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量(环境变量名称请根据自身情况进行设置)EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
// $ip、$port、database需要用户自行修改。

import java.io.StringReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import org.postgresql.copy.CopyManager;
import org.postgresql.core.BaseConnection;

public class Migration{

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

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

         //遍历结果集，逐行获取记录
         //将每条记录中各字段值，按指定分隔符分割，由换行符结束，拼成一个字符串
         //把拼成的字符串，添加到缓存buffer
```



```
            while (rs.next()) {
                buffer.append(rs.getString(1) + delimiter
                        + rs.getString(2) + delimiter
                        + rs.getString(3) + delimiter
                        + rs.getString(4)
                        + "\n");
            }
            rs.close();

            try {
                //建立目标数据库连接
                Class.forName(driver);
                Connection conn = DriverManager.getConnection(url, user, pass);
                BaseConnection baseConn = (BaseConnection) conn;
                baseConn.setAutoCommit(false);

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

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

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

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

5.3.15 示例：逻辑复制代码示例

下面示例演示如何通过JDBC接口使用逻辑复制功能的过程。执行示例前，需要加载驱动，驱动的获取和加载方法请参考**JDBC包、驱动类和环境类**。

针对逻辑复制的配置选项，除了在逻辑解码章节中的配置选项外，还有专门给JDBC等流式解码工具增加的配置项，如下所示。

1. 解码线程并行度

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

2. 解码格式

   通过配置选项decode-style，指定解码格式。其取值为char型的字符'j'、't'或'b'，分别代表json格式、text格式及二进制格式。默认值为'b'即二进制格式解码。该选项仅允许并行解码时设置，且二进制格式解码仅在并行解码场景下支持。与二进制格式对应的json和text格式，在批量发送的解码结果中，每条解码语句的前4字节组成的uint32代表该条语句总字节数（不包含该uint32类型占用的4字节，0代表本批次解码结束），8字节uint64代表相应lsn（begin对应first_lsn，commit对应end_lsn，其他场景对应该条语句的lsn）。

说明

二进制格式编码规则如下所示：

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

3. 限制仅备机解码

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

4. 批量发送

   通过配置选项sending-batch，指定是否批量发送。其取值范围为0或1的int型，取0表示逐条发送解码结果，取1表示解码结果累积到达1MB则批量发送解码结果。默认值为0。该选项仅允许并行解码时设置。开启批量发送的场景中，当解码

格式为'j'或't'时，在原来的每条解码语句之前会附加一个uint32类型，表示本条解码结果长度（长度不包含当前的uint32类型），以及一个uint64类型，表示当前解码结果对应的lsn。

5. 并行解码队列长度

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

6. 逻辑解码内存阈值

   通过配置选项max-txn-in-memory指定单个事务解码中间结果缓存的内存阈值；单位为MB，范围【0，100】，默认值为0，表示不管控内存使用。通过配置选项max-reorderbuffer-in-memory指定所有事务解码中间结果缓存的内存阈值；单位为GB，范围【0，100】，默认值为0，表示不管控内存使用。当超过内存阈值时，解码过程将出现解码中间结果写临时文件的现象，影响逻辑解码的性能。

7. 逻辑解码发送超时阈值

   通过配置选项sender-timeout指定内核与客户端的心跳超时阈值。当该时间段内没有收到客户端任何消息，逻辑解码将主动停止，并断开和客户端的连接。单位为毫秒（ms），范围【0, 2147483647】，默认值取决于GUC参数logical_sender_timeout配置。

8. 逻辑解码用户黑名单选项

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

   a. exclude-userids: 指定黑名单用户的OID，多个OID通过逗号分隔，不校验用户OID是否存在。注意：同一个业务用户在不同DN的OID不一定相同。当前分布式直连DN逻辑解码需要传入业务用户在各DN上相应的OID，否则可能出现某些DN逻辑解码结果进行了过滤而某些DN节点未过滤。

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

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

9. 事务逻辑日志输出选项

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

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

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

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

在并行解码的标准场景下（16核CPU、内存128G、网络带宽 > 200MBps、表的列数为10~100、单行数据量0.1KB~1KB、DML操作以insert为主、不涉及落盘事务即单个事务中语句数量小于4096、parallel-decode-num为8、解码格式为'b'且开启批量发送

功能），解码性能（这里以xLog消耗量为标准）不低于100MBps。为保证解码性能达标以及尽量降低对业务的影响，一台备机上应尽量仅建立一个并行解码连接，保证CPU、内存、带宽资源充足。

⚠ 注意

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

```
//以下以gsjdbc4.jar为例
// 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放(密码应密文存放，使用时解密)，确保安全。
// 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量(环境变量名称请根据自身情况进行设置)EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
//逻辑复制功能示例：文件名，LogicalReplicationDemo.java
//前提条件：添加JDBC用户机器IP到数据库白名单里，在pg_hba.conf添加以下内容，然后重启数据库即可：
//假设JDBC用户IP为10.10.10.10
//host    all             all             10.10.10.10/32        sha256
//host    replication     all             10.10.10.10/32        sha256

import org.postgresql.PGProperty;
import org.postgresql.jdbc.PgConnection;
import org.postgresql.replication.LogSequenceNumber;
import org.postgresql.replication.PGReplicationStream;

import java.nio.ByteBuffer;
import java.sql.DriverManager;
import java.util.Properties;
import java.util.concurrent.TimeUnit;

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

        //默认逻辑复制槽的名称是：replication_slot
        //测试模式：创建逻辑复制槽
        int TEST_MODE_CREATE_SLOT = 1;
        //测试模式：开启逻辑复制（前提条件是逻辑复制槽已经存在）
        int TEST_MODE_START_REPL = 2;
        //测试模式：删除逻辑复制槽
        int TEST_MODE_DROP_SLOT = 3;
        //开启不同的测试模式，实际使用时先将testMode设置为TEST_MODE_CREATE_SLOT创建复制槽，
        //testMode设置为TEST_MODE_START_REPL即可开启解码，解码结束后设置为TEST_MODE_DROP_SLOT以删除当前复制槽
        int testMode = TEST_MODE_START_REPL;

        try {
            Class.forName(driver);
        } catch (Exception e) {
            e.printStackTrace();
            return;
        }

        try {
            Properties properties = new Properties();
            PGProperty.USER.set(properties, System.getenv("EXAMPLE_USERNAME_ENV")); //指定解码用户名
            PGProperty.PASSWORD.set(properties, System.getenv("EXAMPLE_PASSWORD_ENV")); //制定对应密码
            //对于逻辑复制，以下三个属性是必须配置项，如下配置即可
            PGProperty.ASSUME_MIN_SERVER_VERSION.set(properties, "9.4"); //指定数据库版本
            PGProperty.REPLICATION.set(properties, "database"); //表示该连接用于执行逻辑复制
            PGProperty.PREFER_QUERY_MODE.set(properties, "simple"); //指定协议模式
            conn = (PgConnection) DriverManager.getConnection(sourceURL, properties);
```

```
            System.out.println("connection success!");

            if(testMode == TEST_MODE_CREATE_SLOT){
                conn.getReplicationAPI()
                        .createReplicationSlot()
                        .logical()
                        .withSlotName("replication_slot") //创建复制槽的名称，这里字符串如包含大写字母则会自动转化为小写字母
                        .withOutputPlugin("mppdb_decoding") //使用mppdb_decoding插件
                        .make();
            }else if(testMode == TEST_MODE_START_REPL) {
                //开启此模式前需要创建复制槽
                //LogSequenceNumber waitLSN = LogSequenceNumber.valueOf("0/2808340"); //解码的过滤点，在此LSN之前提交的事务将不会被输出
                PGReplicationStream stream = conn
                        .getReplicationAPI()
                        .replicationStream()
                        .logical()
                        .withSlotName("replication_slot") //指定解码的复制槽名称
                        .withSlotOption("include-xids", false) //解码结果中是否包含事务号
                        .withSlotOption("skip-empty-xacts", true) //是否跳过空事务
                        //.withStartPosition(waitLSN) //设置解码的过滤点，在此LSN之前提交的事务将不会被输出
                        .withSlotOption("parallel-decode-num", 10) //解码线程并行度
                        //.withSlotOption("white-table-list", "public.t1,public.t2") //白名单列表
                        //.withSlotOption("standby-connection", true) //强制备机解码
                        .withSlotOption("decode-style", "t") //解码格式
                        //.withSlotOption("sending-batch", 1) //批量发送解码结果
                        .withSlotOption("max-txn-in-memory", 100) //单个解码事务落盘内存阈值为100MB
                        .withSlotOption("max-reorderbuffer-in-memory", 50) //正在处理的解码事务落盘内存阈值为50GB
                        .withSlotOption("exclude-users", "userA") //不返回用户userA执行事务的逻辑日志
                        .withSlotOption("include-user", true) //事务BEGIN逻辑日志携带用户名字
                        .start();
                while (true) {
                    ByteBuffer byteBuffer = stream.readPending();

                    if (byteBuffer == null) {
                        TimeUnit.MILLISECONDS.sleep(10L);
                        continue;
                    }

                    int offset = byteBuffer.arrayOffset();
                    byte[] source = byteBuffer.array();
                    int length = source.length - offset;
                    System.out.println(new String(source, offset, length));

                    //如果需要flush lsn，根据业务实际情况调用以下接口
//                    LogSequenceNumber lastRecv = stream.getLastReceiveLSN();
//                    stream.setFlushedLSN(lastRecv);
//                    stream.forceUpdateStatus();

                }
            }else if(testMode == TEST_MODE_DROP_SLOT){
                conn.getReplicationAPI()
                        .dropReplicationSlot("replication_slot");
            }
        } catch (Exception e) {
            e.printStackTrace();
            return;
        }
    }
}
```

解码结果示例如下：

```
BEGIN CSN: 2014 first_lsn: 0/2816A28
table public t1 INSERT: a[integer]:1 b[integer]:2 c[text]:'hello'
commit xid: 15504
BEGIN CSN: 2015 first_lsn: 0/2816C20
table public t1 UPDATE: old-key: a[integer]:1 new-tuple: a[integer]:1 b[integer]:5 c[text]:'hello'
commit xid: 15505
```



```
BEGIN CSN: 2016 first_lsn: 0/2816D60
table public t1 DELETE: a[integer]:1
commit xid: 15506
```

5.3.16 示例：不同场景下连接数据库参数配置

说明

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

容灾场景

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

则url可参考如下进行配置：

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

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

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

负载均衡场景

某客户存在一套数据库集群，包含如下节点{node1,node2,node3,node4,node5,node6,node7,node8,node9,node10,node11,node12}。

1. 客户在应用程序A中建立了120个长连接，并期望应用程序A上的连接可以均匀分布在当前集群各节点上，则url可参考如下配置。

   ```
   jdbc:postgresql://node1,node2,node3/database?autoBalance=true
   ```

2. 客户新开发了两个应用程序B、C，希望当前这三个应用程序均匀分布在指定节点，如应用程序A的连接分布在{node1,node2,node3,node4}；应用程序B的连接分布在{node5,node6,node7,node8}；应用程序C的连接分布在{node9,node10,node11,node12}；则url可参考如下配置。

   应用程序A：jdbc:postgresql://node1,node2,node3,node4,node5/database?autoBalance=priority4

   应用程序B：jdbc:postgresql://node5,node6,node7,node8,node9/database?autoBalance=priority4

   应用程序C：jdbc:postgresql://node9,node10,node11,node12,node1/database?autoBalance=priority4

3. 客户又开发了一些应用程序，并且使用相同的连接配置串，同时期望各应用连接能较均匀的分布在集群各节点上，则url可参考如下配置。

   ```
   jdbc:postgresql://node1,node2,node3,node4/database?autoBalance=shuffle
   ```

4. 客户不想要使用负载均衡功能，则url可参考如下配置。

   ```
   jdbc:postgresql://node1/database
   ```

   或

   ```
   jdbc:postgresql://node1/database?autoBalance=false
   ```


说明

在开启autoBalance参数时，JDBC刷新可用CN列表的周期默认为10S，可使用refreshCNIpListTime进行设置，如下：

```
jdbc:postgresql://node1,node2,node3,node4/database?autoBalance=true&refreshCNIpListTime=3
```

日志诊断场景

某客户在使用为定位问题，可通过开启trace日志进行诊断，url可参考如下进行配置。

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

高性能场景

某客户对于相同sql可能多次执行，仅是传参不同，为了提升执行效率，可开启prepareThreshold参数，避免重复生成执行计划，url可参考如下配置。

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

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

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

某客户需要批量插入1000万数据，为提升效率，可使用batchMode，url可参考如下配置。

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

5.3.17 JDBC 接口参考

JDBC接口是一套提供给用户的API方法，本节将对部分常用接口做具体描述，若涉及其他接口可参考JDK1.6（软件包）/JDBC4.0中相关内容。

5.3.17.1 java.sql.Connection

java.sql.Connection是数据库连接接口。

表 5-7 对 java.sql.Connection 接口的支持情况

方法名	返回值类型	支持JDBC 4
abort(Executor executor)	void	Yes
clearWarnings()	void	Yes
close()	void	Yes
commit()	void	Yes
createArrayOf(String typeName, Object[] elements)	Array	Yes
createBlob()	Blob	Yes
createClob()	Clob	Yes

方法名	返回值类型	支持JDBC 4
createSQLXML()	SQLXML	Yes
createStatement()	Statement	Yes
createStatement(int resultSetType, int resultSetConcurrency)	Statement	Yes
createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)	Statement	Yes
getAutoCommit()	Boolean	Yes
getCatalog()	String	Yes
getClientInfo()	Properties	Yes
getClientInfo(String name)	String	Yes
getHoldability()	int	Yes
getMetaData()	DatabaseMetaData	Yes
getNetworkTimeout()	int	Yes
getSchema()	String	Yes
getTransactionIsolation()	int	Yes
getTypeMap()	Map<String,Class<?>>	Yes
getWarnings()	SQLWarning	Yes
isClosed()	Boolean	Yes
isReadOnly()	Boolean	Yes
isValid(int timeout)	boolean	Yes
nativeSQL(String sql)	String	Yes
prepareCall(String sql)	CallableStatement	Yes
prepareCall(String sql, int resultSetType, int resultSetConcurrency)	CallableStatement	Yes
prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	CallableStatement	Yes
prepareStatement(String sql)	PreparedStatement	Yes

方法名	返回值类型	支持JDBC 4
prepareStatement(String sql, int autoGeneratedKeys)	PreparedStatement	Yes
prepareStatement(String sql, int[] columnIndexes)	PreparedStatement	Yes
prepareStatement(String sql, int resultSetType, int resultSetConcurrency)	PreparedStatement	Yes
prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	PreparedStatement	Yes
prepareStatement(String sql, String[] columnNames)	PreparedStatement	Yes
releaseSavepoint(Savepoint savepoint)	void	Yes
rollback()	void	Yes
rollback(Savepoint savepoint)	void	Yes
setAutoCommit(boolean autoCommit)	void	Yes
setClientInfo(Properties properties)	void	Yes
setClientInfo(String name,String value)	void	Yes
setHoldability(int holdability)	void	Yes
setNetworkTimeout (Executor executor, int milliseconds)	void	Yes
setReadOnly(boolean readOnly)	void	Yes
setSavepoint()	Savepoint	Yes
setSavepoint(String name)	Savepoint	Yes
setSchema(String schema)	void	Yes
setTransactionIsolation(int level)	void	Yes
setTypeMap(Map<String, Class<?>> map)	void	Yes

须知

接口内部默认使用自动提交模式，若通过setAutoCommit(false)关闭自动提交，将会导致后面执行的语句都受到显式事务包裹，数据库中不支持事务中执行的语句不能在此模式下执行。

5.3.17.2 java.sql.CallableStatement

java.sql.CallableStatement是存储过程执行接口。

表 5-8 对 java.sql.CallableStatement 的支持情况

方法名	返回值类型	支持JDBC 4
getArray(int parameterIndex)	Array	Yes
getBigDecimal(int parameterIndex)	BigDecimal	Yes
getBlob(int parameterIndex)	Blob	Yes
getBoolean(int parameterIndex)	boolean	Yes
getByte(int parameterIndex)	byte	Yes
getBytes(int parameterIndex)	byte[]	Yes
getClob(int parameterIndex)	Clob	Yes
getDate(int parameterIndex)	Date	Yes
getDate(int parameterIndex, Calendar cal)	Date	Yes
getDouble(int parameterIndex)	double	Yes
getFloat(int parameterIndex)	float	Yes
getInt(int parameterIndex)	int	Yes
getLong(int parameterIndex)	long	Yes
getObject(int parameterIndex)	Object	Yes
getObject(int parameterIndex, Class<T> type)	Object	Yes
getShort(int parameterIndex)	short	Yes

方法名	返回值类型	支持JDBC 4
getSQLXML(int parameterIndex)	SQLXML	Yes
getString(int parameterIndex)	String	Yes
getNString(int parameterIndex)	String	Yes
getTime(int parameterIndex)	Time	Yes
getTime(int parameterIndex, Calendar cal)	Time	Yes
getTimestamp(int parameterIndex)	Timestamp	Yes
getTimestamp(int parameterIndex, Calendar cal)	Timestamp	Yes
registerOutParameter(int parameterIndex, int type)	void	Yes
registerOutParameter(int parameterIndex, int sqlType, int type)	void	Yes
wasNull()	Boolean	Yes

说明

- 不允许含有OUT参数的语句执行批量操作。
- 以下方法是从java.sql.Statement继承而来：close，execute，executeQuery，executeUpdate，getConnection，getResultSet，getUpdateCount，isClosed，setMaxRows , setFetchSize。
- 以下方法是从java.sql.PreparedStatement继承而来：addBatch，clearParameters，execute，executeQuery，executeUpdate，getMetaData，setBigDecimal，setBoolean，setByte，setBytes，setDate，setDouble，setFloat，setInt，setLong，setNull，setObject，setString，setTime，setTimestamp 。
- registerOutParameter(int parameterIndex, int sqlType, int type)方法仅用于注册复合数据类型，其它类型不支持。

5.3.17.3 java.sql.DatabaseMetaData

java.sql.DatabaseMetaData是数据库对象定义接口。

表 5-9 对 java.sql.DatabaseMetaData 的支持情况

方法名	返回值类型	支持JDBC 4
allProceduresAreCallable()	boolean	Yes
allTablesAreSelectable()	boolean	Yes
autoCommitFailureClosesAllResultSets()	boolean	Yes
dataDefinitionCausesTransactionCommit()	boolean	Yes
dataDefinitionIgnoredInTransactions()	boolean	Yes
deletesAreDetected(int type)	boolean	Yes
doesMaxRowSizeIncludeBlobs()	boolean	Yes
generatedKeyAlwaysReturned()	boolean	Yes
getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)	ResultSet	Yes
getCatalogs()	ResultSet	Yes
getCatalogSeparator()	String	Yes
getCatalogTerm()	String	Yes
getClientInfoProperties()	ResultSet	Yes
getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)	ResultSet	Yes
getConnection()	Connection	Yes
getCrossReference(String parentCatalog, String parentSchema, String parentTable, String foreignCatalog, String foreignSchema, String foreignTable)	ResultSet	Yes
getDefaultTransactionIsolation()	int	Yes

方法名	返回值类型	支持JDBC 4
getExportedKeys(String catalog, String schema, String table)	ResultSet	Yes
getExtraNameCharacters()	String	Yes
getFunctionColumns(String catalog, String schemaPattern, String functionNamePattern, String columnNamePattern)	ResultSet	Yes
getFunctions(String catalog, String schemaPattern, String functionNamePattern)	ResultSet	Yes
getIdentifierQuoteString()	String	Yes
getImportedKeys(String catalog, String schema, String table)	ResultSet	Yes
getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)	ResultSet	Yes
getMaxBinaryLiteralLength()	int	Yes
getMaxCatalogNameLength()	int	Yes
getMaxCharLiteralLength()	int	Yes
getMaxColumnNameLength()	int	Yes
getMaxColumnsInGroupBy()	int	Yes
getMaxColumnsInIndex()	int	Yes
getMaxColumnsInOrderBy()	int	Yes
getMaxColumnsInSelect()	int	Yes
getMaxColumnsInTable()	int	Yes
getMaxConnections()	int	Yes
getMaxCursorNameLength()	int	Yes
getMaxIndexLength()	int	Yes
getMaxLogicalLobSize()	default long	Yes
getMaxProcedureNameLength()	int	Yes

方法名	返回值类型	支持JDBC 4
getMaxRowSize()	int	Yes
getMaxSchemaNameLength()	int	Yes
getMaxStatementLength()	int	Yes
getMaxStatements()	int	Yes
getMaxTableNameLength()	int	Yes
getMaxTablesInSelect()	int	Yes
getMaxUserNameLength()	int	Yes
getNumericFunctions()	String	Yes
getPrimaryKeys(String catalog, String schema, String table)	ResultSet	Yes
getPartitionTablePrimaryKeys(String catalog, String schema, String table)	ResultSet	Yes
getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)	ResultSet	Yes
getProcedures(String catalog, String schemaPattern, String procedureNamePattern)	ResultSet	Yes
getProcedureTerm()	String	Yes
getSchemas()	ResultSet	Yes
getSchemas(String catalog, String schemaPattern)	ResultSet	Yes
getSchemaTerm()	String	Yes
getSearchStringEscape()	String	Yes
getSQLKeywords()	String	Yes
getSQLStateType()	int	Yes
getStringFunctions()	String	Yes
getSystemFunctions()	String	Yes

方法名	返回值类型	支持JDBC 4
getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)	ResultSet	Yes
getTimeDateFunctions()	String	Yes
getTypeInfo()	ResultSet	Yes
getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)	ResultSet	Yes
getURL()	String	Yes
getVersionColumns(String catalog, String schema, String table)	ResultSet	Yes
insertsAreDetected(int type)	boolean	Yes
locatorsUpdateCopy()	boolean	Yes
othersDeletesAreVisible(int type)	boolean	Yes
othersInsertsAreVisible(int type)	boolean	Yes
othersUpdatesAreVisible(int type)	boolean	Yes
ownDeletesAreVisible(int type)	boolean	Yes
ownInsertsAreVisible(int type)	boolean	Yes
ownUpdatesAreVisible(int type)	boolean	Yes
storesLowerCaseIdentifiers()	boolean	Yes
storesMixedCaseIdentifiers()	boolean	Yes
storesUpperCaseIdentifiers()	boolean	Yes
supportsBatchUpdates()	boolean	Yes
supportsCatalogsInDataManipulation()	boolean	Yes
supportsCatalogsInIndexDefinitions()	boolean	Yes

方法名	返回值类型	支持JDBC 4
supportsCatalogsInPrivilegeDefinitions()	boolean	Yes
supportsCatalogsInProcedureCalls()	boolean	Yes
supportsCatalogsInTableDefinitions()	boolean	Yes
supportsCorrelatedSubqueries()	boolean	Yes
supportsDataDefinitionAndDataManipulationTransactions()	boolean	Yes
supportsDataManipulationTransactionsOnly()	boolean	Yes
supportsGetGeneratedKeys()	boolean	Yes
supportsMixedCaseIdentifiers()	boolean	Yes
supportsMultipleOpenResults()	boolean	Yes
supportsNamedParameters()	boolean	Yes
supportsOpenCursorsAcrossCommit()	boolean	Yes
supportsOpenCursorsAcrossRollback()	boolean	Yes
supportsOpenStatementsAcrossCommit()	boolean	Yes
supportsOpenStatementsAcrossRollback()	boolean	Yes
supportsPositionedDelete()	boolean	Yes
supportsPositionedUpdate()	boolean	Yes
supportsRefCursors()	boolean	Yes
supportsResultSetConcurrency(int type, int concurrency)	boolean	Yes
supportsResultSetType(int type)	boolean	Yes
supportsSchemasInIndexDefinitions()	boolean	Yes

方法名	返回值类型	支持JDBC 4
supportsSchemasInPrivilegeDefinitions()	boolean	Yes
supportsSchemasInProcedureCalls()	boolean	Yes
supportsSchemasInTableDefinitions()	boolean	Yes
supportsSelectForUpdate()	boolean	Yes
supportsStatementPooling()	boolean	Yes
supportsStoredFunctionsUsingCallSyntax()	boolean	Yes
supportsStoredProcedures()	boolean	Yes
supportsSubqueriesInComparisons()	boolean	Yes
supportsSubqueriesInExists()	boolean	Yes
supportsSubqueriesInIns()	boolean	Yes
supportsSubqueriesInQuantifieds()	boolean	Yes
supportsTransactionIsolationLevel(int level)	boolean	Yes
supportsTransactions()	boolean	Yes
supportsUnion()	boolean	Yes
supportsUnionAll()	boolean	Yes
updatesAreDetected(int type)	boolean	Yes
getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)	ResultSet	Yes
getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)	ResultSet	Yes
getTableTypes()	ResultSet	Yes
getUserName()	String	Yes
isReadOnly()	boolean	Yes
nullsAreSortedHigh()	boolean	Yes

方法名	返回值类型	支持JDBC 4
nullsAreSortedLow()	boolean	Yes
nullsAreSortedAtStart()	boolean	Yes
nullsAreSortedAtEnd()	boolean	Yes
getDatabaseProductName()	String	Yes
getDatabaseProductVersion()	String	Yes
getDriverName()	String	Yes
getDriverVersion()	String	Yes
getDriverMajorVersion()	int	Yes
getDriverMinorVersion()	int	Yes
usesLocalFiles()	boolean	Yes
usesLocalFilePerTable()	boolean	Yes
supportsMixedCaseIdentifiers()	boolean	Yes
storesUpperCaseIdentifiers()	boolean	Yes
storesLowerCaseIdentifiers()	boolean	Yes
supportsMixedCaseQuotedIdentifiers()	boolean	Yes
storesUpperCaseQuotedIdentifiers()	boolean	Yes
storesLowerCaseQuotedIdentifiers()	boolean	Yes
storesMixedCaseQuotedIdentifiers()	boolean	Yes
supportsAlterTableWithAddColumn()	boolean	Yes
supportsAlterTableWithDropColumn()	boolean	Yes
supportsColumnAliasing()	boolean	Yes
nullPlusNonNullIsNull()	boolean	Yes
supportsConvert()	boolean	Yes
supportsConvert(int fromType, int toType)	boolean	Yes

方法名	返回值类型	支持JDBC 4
supportsTableCorrelationNames()	boolean	Yes
supportsDifferentTableCorrelationNames()	boolean	Yes
supportsExpressionsInOrderBy()	boolean	Yes
supportsOrderByUnrelated()	boolean	Yes
supportsGroupBy()	boolean	Yes
supportsGroupByUnrelated()	boolean	Yes
supportsGroupByBeyondSelect()	boolean	Yes
supportsLikeEscapeClause()	boolean	Yes
supportsMultipleResultSets()	boolean	Yes
supportsMultipleTransactions()	boolean	Yes
supportsNonNullableColumns()	boolean	Yes
supportsMinimumSQLGrammar()	boolean	Yes
supportsCoreSQLGrammar()	boolean	Yes
supportsExtendedSQLGrammar()	boolean	Yes
supportsANSI92EntryLevelSQL()	boolean	Yes
supportsANSI92IntermediateSQL()	boolean	Yes
supportsANSI92FullSQL()	boolean	Yes
supportsIntegrityEnhancementFacility()	boolean	Yes
supportsOuterJoins()	boolean	Yes
supportsFullOuterJoins()	boolean	Yes
supportsLimitedOuterJoins()	boolean	Yes
isCatalogAtStart()	boolean	Yes
supportsSchemasInDataManipulation()	boolean	Yes

方法名	返回值类型	支持JDBC 4
supportsSavepoints()	boolean	Yes
supportsResultSetHoldability(int holdability)	boolean	Yes
getResultSetHoldability()	int	Yes
getDatabaseMajorVersion()	int	Yes
getDatabaseMinorVersion()	int	Yes
getJDBCMajorVersion()	int	Yes
getJDBCMinorVersion()	int	Yes

说明

getPartitionTablePrimaryKeys(String catalog, String schema, String table)接口用于获取分区表含全局索引的主键列，使用示例如下：

```
PgDatabaseMetaData dbmd = (PgDatabaseMetaData)conn.getMetaData();
dbmd.getPartitionTablePrimaryKeys("catalogName", "schemaName", "tableName");
```

5.3.17.4 java.sql.Driver

java.sql.Driver是数据库驱动接口。

表 5-10 对 java.sql.Driver 的支持情况

方法名	返回值类型	支持JDBC 4
acceptsURL(String url)	Boolean	Yes
connect(String url, Properties info)	Connection	Yes
jdbcCompliant()	Boolean	Yes
getMajorVersion()	int	Yes
getMinorVersion()	int	Yes
getParentLogger()	Logger	Yes
getPropertyInfo(String url, Properties info)	DriverPropertyInfo[]	Yes

5.3.17.5 java.sql.PreparedStatement

java.sql.PreparedStatement是预处理语句接口。

表 5-11 对 java.sql.PreparedStatement 的支持情况

方法名	返回值类型	支持JDBC 4
clearParameters()	void	Yes
execute()	Boolean	Yes
executeQuery()	ResultSet	Yes
excuteUpdate()	int	Yes
executeLargeUpdate()	long	No
getMetaData()	ResultSetMetaData	Yes
getParameterMetaData()	ParameterMetaData	Yes
setArray(int parameterIndex, Array x)	void	Yes
setAsciiStream(int parameterIndex, InputStream x, int length)	void	Yes
setBinaryStream(int parameterIndex, InputStream x)	void	Yes
setBinaryStream(int parameterIndex, InputStream x, int length)	void	Yes
setBinaryStream(int parameterIndex, InputStream x, long length)	void	Yes
setBlob(int parameterIndex, InputStream inputStream)	void	Yes
setBlob(int parameterIndex, InputStream inputStream, long length)	void	Yes
setBlob(int parameterIndex, Blob x)	void	Yes
setCharacterStream(int parameterIndex, Reader reader)	void	Yes

方法名	返回值类型	支持JDBC 4
setCharacterStream(int parameterIndex, Reader reader, int length)	void	Yes
setClob(int parameterIndex, Reader reader)	void	Yes
setClob(int parameterIndex, Reader reader, long length)	void	Yes
setClob(int parameterIndex, Clob x)	void	Yes
setDate(int parameterIndex, Date x, Calendar cal)	void	Yes
setNull(int parameterIndex, int sqlType)	void	Yes
setNull(int parameterIndex, int sqlType, String typeName)	void	Yes
setObject(int parameterIndex, Object x)	void	Yes
setObject(int parameterIndex, Object x, int targetSqlType)	void	Yes
setObject(int parameterIndex, Object x, int targetSqlType, int scaleOrLength)	void	Yes
setSQLXML(int parameterIndex, SQLXML xmlObject)	void	Yes
setTime(int parameterIndex, Time x)	void	Yes
setTime(int parameterIndex, Time x, Calendar cal)	void	Yes

方法名	返回值类型	支持JDBC 4
setTimestamp(int parameterIndex, Timestamp x)	void	Yes
setTimestamp(int parameterIndex, Timestamp x, Calendar cal)	void	Yes
setUnicodeStream(int parameterIndex, InputStream x, int length)	void	Yes
setURL(int parameterIndex, URL x)	void	Yes
setBoolean(int parameterIndex, boolean x)	void	Yes
setBigDecimal(int parameterIndex, BigDecimal x)	void	Yes
setByte(int parameterIndex, byte x)	void	Yes
setBytes(int parameterIndex, byte[] x)	void	Yes
setDate(int parameterIndex, Date x)	void	Yes
setDouble(int parameterIndex, double x)	void	Yes
setFloat(int parameterIndex, float x)	void	Yes
setInt(int parameterIndex, int x)	void	Yes
setLong(int parameterIndex, long x)	void	Yes
setShort(int parameterIndex, short x)	void	Yes
setString(int parameterIndex, String x)	void	Yes

方法名	返回值类型	支持JDBC 4
setNString(int parameterIndex, String x)	void	Yes
addBatch()	void	Yes
executeBatch()	int[]	Yes

说明

- addBatch()、execute()必须在clearBatch()之后才能执行。
- 调用executeBatch()方法并不会清除batch。用户必须显式使用clearBatch()清除 。
- 在添加了一个batch的绑定变量后，用户若想重用这些值（再次添加一个batch），无需再次使用set*()方法 。
- 以下方法是从java.sql.Statement继承而来：close，execute，executeQuery，executeUpdate，getConnection，getResultSet，getUpdateCount，isClosed，setMaxRows,setFetchSize。
- executeLargeUpdate()方法必须在JDBC4.2及以上使用。

5.3.17.6 java.sql.ResultSet

java.sql.ResultSet是执行结果集接口。

表 5-12 对 java.sql.ResultSet 的支持情况

方法名	返回值类型	支持JDBC 4
absolute(int row)	Boolean	Yes
afterLast()	void	Yes
beforeFirst()	void	Yes
cancelRowUpdates()	void	Yes
clearWarnings()	void	Yes
close()	void	Yes
deleteRow()	void	Yes
findColumn(String columnLabel)	int	Yes
first()	Boolean	Yes
getArray(int columnIndex)	Array	Yes
getArray(String columnLabel)	Array	Yes

方法名	返回值类型	支持JDBC 4
getAsciiStream(int columnIndex)	InputStream	Yes
getAsciiStream(String columnLabel)	InputStream	Yes
getBigDecimal(int columnIndex)	BigDecimal	Yes
getBigDecimal(String columnLabel)	BigDecimal	Yes
getBinaryStream(int columnIndex)	InputStream	Yes
getBinaryStream(String columnLabel)	InputStream	Yes
getBlob(int columnIndex)	Blob	Yes
getBlob(String columnLabel)	Blob	Yes
getBoolean(int columnIndex)	Boolean	Yes
getBoolean(String columnLabel)	Boolean	Yes
getByte(int columnIndex)	byte	Yes
getBytes(int columnIndex)	byte[]	Yes
getByte(String columnLabel)	byte	Yes
getBytes(String columnLabel)	byte[]	Yes
getCharacterStream(int columnIndex)	Reader	Yes
getCharacterStream (String columnLabel)	Reader	Yes
getClob(int columnIndex)	Clob	Yes
getClob(String columnLabel)	Clob	Yes
getConcurrency()	int	Yes
getCursorName()	String	Yes
getDate(int columnIndex)	Date	Yes

方法名	返回值类型	支持JDBC 4
getDate(int columnIndex, Calendar cal)	Date	Yes
getDate(String columnLabel)	Date	Yes
getDate(String columnLabel, Calendar cal)	Date	Yes
getDouble(int columnIndex)	double	Yes
getDouble(String columnLabel)	double	Yes
getFetchDirection()	int	Yes
getFetchSize()	int	Yes
getFloat(int columnIndex)	float	Yes
getFloat(String columnLabel)	float	Yes
getInt(int columnIndex)	int	Yes
getInt(String columnLabel)	int	Yes
getLong(int columnIndex)	long	Yes
getLong(String columnLabel)	long	Yes
getMetaData()	ResultSetMetaData	Yes
getObject(int columnIndex)	Object	Yes
getObject(int columnIndex, Class<T> type)	<T> T	Yes
getObject(int columnIndex, Map<String, Class<?>> map)	Object	Yes
getObject(String columnLabel)	Object	Yes
getObject(String columnLabel, Class<T> type)	<T> T	Yes

方法名	返回值类型	支持JDBC 4
getObject(String columnLabel, Map<String, Class<?>> map)	Object	Yes
getRow()	int	Yes
getShort(int columnIndex)	short	Yes
getShort(String columnLabel)	short	Yes
getSQLXML(int columnIndex)	SQLXML	Yes
getSQLXML(String columnLabel)	SQLXML	Yes
getStatement()	Statement	Yes
getString(int columnIndex)	String	Yes
getString(String columnLabel)	String	Yes
getNString(int columnIndex)	String	Yes
getNString(String columnLabel)	String	Yes
getTime(int columnIndex)	Time	Yes
getTime(int columnIndex, Calendar cal)	Time	Yes
getTime(String columnLabel)	Time	Yes
getTime(String columnLabel, Calendar cal)	Time	Yes
getTimestamp(int columnIndex)	Timestamp	Yes
getTimestamp(int columnIndex, Calendar cal)	Timestamp	Yes
getTimestamp(String columnLabel)	Timestamp	Yes

方法名	返回值类型	支持JDBC 4
getTimestamp(String columnLabel, Calendar cal)	Timestamp	Yes
getType()	int	Yes
getWarnings()	SQLWarning	Yes
insertRow()	void	Yes
isAfterLast()	Boolean	Yes
isBeforeFirst()	Boolean	Yes
isClosed()	Boolean	Yes
isFirst()	Boolean	Yes
isLast()	Boolean	Yes
last()	Boolean	Yes
moveToCurrentRow()	void	Yes
moveToInsertRow()	void	Yes
next()	Boolean	Yes
previous()	Boolean	Yes
refreshRow()	void	Yes
relative(int rows)	Boolean	Yes
rowDeleted()	Boolean	Yes
rowInserted()	Boolean	Yes
rowUpdated()	Boolean	Yes
setFetchDirection(int direction)	void	Yes
setFetchSize(int rows)	void	Yes
updateArray(int columnIndex, Array x)	void	Yes
updateArray(String columnLabel, Array x)	void	Yes
updateAsciiStream(int columnIndex, InputStream x, int length)	void	Yes
updateAsciiStream(String columnLabel, InputStream x, int length)	void	Yes

方法名	返回值类型	支持JDBC 4
updateBigDecimal(int columnIndex, BigDecimal x)	void	Yes
updateBigDecimal(String columnLabel, BigDecimal x)	void	Yes
updateBinaryStream(int columnIndex, InputStream x, int length)	void	Yes
updateBinaryStream (String columnLabel, InputStream x, int length)	void	Yes
updateBoolean(int columnIndex, boolean x)	void	Yes
updateBoolean(String columnLabel, boolean x)	void	Yes
updateByte(int columnIndex, byte x)	void	Yes
updateByte(String columnLabel, byte x)	void	Yes
updateBytes(int columnIndex, byte[] x)	void	Yes
updateBytes(String columnLabel, byte[] x)	void	Yes
updateCharacterStream (int columnIndex, Reader x, int length)	void	Yes
updateCharacterStream (String columnLabel, Reader reader, int length)	void	Yes
updateDate(int columnIndex, Date x)	void	Yes
updateDate(String columnLabel, Date x)	void	Yes
updateDouble(int columnIndex, double x)	void	Yes
updateDouble(String columnLabel, double x)	void	Yes

方法名	返回值类型	支持JDBC 4
updateFloat(int columnIndex, float x)	void	Yes
updateFloat(String columnLabel, float x)	void	Yes
updateInt(int columnIndex, int x)	void	Yes
updateInt(String columnLabel, int x)	void	Yes
updateLong(int columnIndex, long x)	void	Yes
updateLong(String columnLabel, long x)	void	Yes
updateNull(int columnIndex)	void	Yes
updateNull(String columnLabel)	void	Yes
updateObject(int columnIndex, Object x)	void	Yes
updateObject(int columnIndex, Object x, int scaleOrLength)	void	Yes
updateObject(String columnLabel, Object x)	void	Yes
updateObject(String columnLabel, Object x, int scaleOrLength)	void	Yes
updateRow()	void	Yes
updateShort(int columnIndex, short x)	void	Yes
updateShort(String columnLabel, short x)	void	Yes
updateSQLXML(int columnIndex, SQLXML xmlObject)	void	Yes
updateSQLXML(String columnLabel, SQLXML xmlObject)	void	Yes
updateString(int columnIndex, String x)	void	Yes

方法名	返回值类型	支持JDBC 4
updateString(String columnLabel, String x)	void	Yes
updateTime(int columnIndex, Time x)	void	Yes
updateTime(String columnLabel, Time x)	void	Yes
updateTimestamp(int columnIndex, Timestamp x)	void	Yes
updateTimestamp(String columnLabel, Timestamp x)	void	Yes
wasNull()	Boolean	Yes

说明

- 一个Statement不能有多个处于“open”状态的ResultSet。
- 用于遍历结果集（ResultSet）的游标（Cursor）在被提交后不能保持“open”的状态。

5.3.17.7 java.sql.ResultSetMetaData

java.sql.ResultSetMetaData是对ResultSet对象相关信息的具体描述。

表 5-13 对 java.sql.ResultSetMetaData 的支持情况

方法名	返回值类型	支持JDBC 4
getCatalogName(int column)	String	Yes
getColumnClassName(int column)	String	Yes
getColumnCount()	int	Yes
getColumnDisplaySize(int column)	int	Yes
getColumnLabel(int column)	String	Yes
getColumnName(int column)	String	Yes
getColumnType(int column)	int	Yes
getColumnTypeName(int column)	String	Yes

方法名	返回值类型	支持JDBC 4
getPrecision(int column)	int	Yes
getScale(int column)	int	Yes
getSchemaName(int column)	String	Yes
getTableName(int column)	String	Yes
isAutoIncrement(int column)	boolean	Yes
isCaseSensitive(int column)	boolean	Yes
isCurrency(int column)	boolean	Yes
isDefinitelyWritable(int column)	boolean	Yes
isNullable(int column)	int	Yes
isReadOnly(int column)	boolean	Yes
isSearchable(int column)	boolean	Yes
isSigned(int column)	boolean	Yes
isWritable(int column)	boolean	Yes

5.3.17.8 java.sql.Statement

java.sql.Statement是SQL语句接口。

表 5-14 对 java.sql.Statement 的支持情况

方法名	返回值类型	支持JDBC 4
addBatch(String sql)	void	Yes
clearBatch()	void	Yes
clearWarnings()	void	Yes
close()	void	Yes
closeOnCompletion()	void	Yes
execute(String sql)	Boolean	Yes
execute(String sql, int autoGeneratedKeys)	Boolean	Yes
execute(String sql, int[] columnIndexes)	Boolean	Yes

方法名	返回值类型	支持JDBC 4
execute(String sql, String[] columnNames)	Boolean	Yes
executeBatch()	Boolean	Yes
executeQuery(String sql)	ResultSet	Yes
executeUpdate(String sql)	int	Yes
executeUpdate(String sql, int autoGeneratedKeys)	int	Yes
executeUpdate(String sql, int[] columnIndexes)	int	Yes
executeUpdate(String sql, String[] columnNames)	int	Yes
getConnection()	Connection	Yes
getFetchDirection()	int	Yes
getFetchSize()	int	Yes
getGeneratedKeys()	ResultSet	Yes
getMaxFieldSize()	int	Yes
getMaxRows()	int	Yes
getMoreResults()	boolean	Yes
getMoreResults(int current)	boolean	Yes
getResultSet()	ResultSet	Yes
getResultSetConcurrency()	int	Yes
getResultSetHoldability()	int	Yes
getResultSetType()	int	Yes
getQueryTimeout()	int	Yes
getUpdateCount()	int	Yes
getWarnings()	SQLWarning	Yes
isClosed()	Boolean	Yes

方法名	返回值类型	支持JDBC 4
isCloseOnCompletion()	Boolean	Yes
isPoolable()	Boolean	Yes
setCursorName(String name)	void	Yes
setEscapeProcessing (boolean enable)	void	Yes
setFetchDirection(int direction)	void	Yes
setMaxFieldSize(int max)	void	Yes
setMaxRows(int max)	void	Yes
setPoolable(boolean poolable)	void	Yes
setQueryTimeout(int seconds)	void	Yes
setFetchSize(int rows)	void	Yes
cancel()	void	Yes
executeLargeUpdate(String sql)	long	No
getLargeUpdateCount()	long	No
executeLargeBatch()	long	No
executeLargeUpdate(String sql, int autoGeneratedKeys)	long	No
executeLargeUpdate(String sql, int[] columnIndexes)	long	No
executeLargeUpdate(String sql, String[] columnNames)	long	No

说明

- 通过setFetchSize可以减少结果集在客户端的内存占用情况。它的原理是通过将结果集打包成游标，然后分段处理，所以会加大数据库与客户端的通信量，会有性能损耗。
- 由于数据库游标是事务内有效，所以，在设置setFetchSize的同时，需要将连接设置为非自动提交模式，setAutoCommit(false)。同时在业务数据需要持久化到数据库中时，在连接上执行提交操作。
- LargeUpdate相关方法必须在JDBC4.2及以上使用。

5.3.17.9 javax.sql.ConnectionPoolDataSource

javax.sql.ConnectionPoolDataSource是数据源连接池接口。

表 5-15 对 javax.sql.ConnectionPoolDataSource 的支持情况

方法名	返回值类型	支持JDBC 4
getPooledConnection()	PooledConnection	Yes
getPooledConnection(String user,String password)	PooledConnection	Yes

5.3.17.10 javax.sql.DataSource

javax.sql.DataSource是数据源接口。

表 5-16 对 javax.sql.DataSource 接口的支持情况

方法名	返回值类型	支持JDBC 4
getConneciton()	Connection	Yes
getConnection(String username,String password)	Connection	Yes
getLoginTimeout()	int	Yes
getLogWriter()	PrintWriter	Yes
setLoginTimeout(int seconds)	void	Yes
setLogWriter(PrintWriter out)	void	Yes

5.3.17.11 javax.sql.PooledConnection

javax.sql.PooledConnection是由连接池创建的连接接口。

表 5-17 对 javax.sql.PooledConnection 的支持情况

方法名	返回值类型	支持JDBC 4
addConnectionEventListener (ConnectionEventListener listener)	void	Yes
close()	void	Yes
getConnection()	Connection	Yes
removeConnectionEventListener (ConnectionEventListener listener)	void	Yes

5.3.17.12 javax.naming.Context

javax.naming.Context是连接配置的上下文接口。

表 5-18 对 javax.naming.Context 的支持情况

方法名	返回值类型	支持JDBC 4
bind(Name name, Object obj)	void	Yes
bind(String name, Object obj)	void	Yes
lookup(Name name)	Object	Yes
lookup(String name)	Object	Yes
rebind(Name name, Object obj)	void	Yes
rebind(String name, Object obj)	void	Yes
rename(Name oldName, Name newName)	void	Yes
rename(String oldName, String newName)	void	Yes
unbind(Name name)	void	Yes
unbind(String name)	void	Yes

5.3.17.13 javax.naming.spi.InitialContextFactory

javax.naming.spi.InitialContextFactory是初始连接上下文工厂接口。

表 5-19 对 javax.naming.spi.InitialContextFactory 的支持情况

方法名	返回值类型	支持JDBC 4
getInitialContext(Hashtable<?,?> environment)	Context	Yes

5.3.17.14 CopyManager

CopyManager是GaussDB JDBC驱动中提供的一个API接口类，用于批量向GaussDB集群中导入数据。

CopyManager 的继承关系

CopyManager类位于org.postgresql.copy Package中，继承自java.lang.Object类，该类的声明如下：

```
public class CopyManager
extends Object
```

构造方法

public CopyManager(BaseConnection connection)

throws SQLException

常用方法

表 5-20 CopyManager 常用方法

返回值	方法	描述	throws
CopyIn	copyIn(String sql)	-	SQLException
long	copyIn(String sql, InputStream from)	使用COPY FROM STDIN从InputStream中快速向数据库中的表加载数据。	SQLException,IOException
long	copyIn(String sql, InputStream from, int bufferSize)	使用COPY FROM STDIN从InputStream中快速向数据库中的表加载指定长度的数据。	SQLException,IOException
long	copyIn(String sql, Reader from)	使用COPY FROM STDIN从Reader中快速向数据库中的表加载数据。	SQLException,IOException

返回值	方法	描述	throws
long	copyIn(String sql, Reader from, int bufferSize)	使用COPY FROM STDIN从Reader中快速向数据库中的表加载指定长度的数据。	SQLException,IOException
CopyOut	copyOut(String sql)	-	SQLException
long	copyOut(String sql, OutputStream to)	将一个COPY TO STDOUT的结果集从数据库发送到OutputStream类中。	SQLException,IOException
long	copyOut(String sql, Writer to)	将一个COPY TO STDOUT的结果集从数据库发送到Writer类中。	SQLException,IOException

5.3.17.15 PGReplicationConnection

PGReplicationConnection是GaussDB JDBC驱动中提供的一个API接口类，用于执行逻辑复制相关的功能。

PGReplicationConnection 的继承关系

PGReplicationConnection是逻辑复制的接口，实现类是PGReplicationConnectionImpl，该类位于org.postgresql.replication Package中，该类的声明如下：

```
public class PGReplicationConnection implements PGReplicationConnection
```

构造方法

public PGReplicationConnection(BaseConnection connection)

常用方法

表 5-21 PGReplicationConnection 常用方法

返回值	方法	描述	throws
ChainedCreateReplicationSlotBuilder	createReplicationSlot()	用于创建逻辑复制槽	-
void	dropReplicationSlot(String slotName)	用于删除逻辑复制槽	SQLException,IOException

返回值	方法	描述	throws
ChainedStreamBuilder	replicationStream()	用户开启逻辑复制	-

5.3.17.16 PGReplicationStream

PGReplicationStream是GaussDB JDBC驱动中提供的一个API接口类，用于操作逻辑复制流。

PGReplicationStream 的继承关系

PGReplicationStream是逻辑复制的接口，实现类是V3PGReplicationStream，该类位于org.postgresql.core.v3.replication Package中，该类的声明如下：

```
public class V3PGReplicationStream implements PGReplicationStream
```

构造方法

public V3PGReplicationStream(CopyDual copyDual, LogSequenceNumber startLSN, long updateIntervalMs, ReplicationType replicationType)

常用方法

表 5-22 PGReplicationConnection 常用方法

返回值	方法	描述	throws
void	close()	结束逻辑复制，并释放资源。	SQLException
void	forceUpdateStatus()	强制将上次接收、刷新和应用的 LSN 状态发送到后端。	SQLException
LogSequenceNumber	getLastAppliedLSN()	获取上次主机日志回放的LSN。	-
LogSequenceNumber	getLastFlushedLSN()	获取上次主机刷新的LSN，即当前逻辑解码推进的LSN。	-
LogSequenceNumber	getLastReceiveLSN()	获取上次接收的LSN。	-
boolean	isClosed()	复制流是否关闭。	-
ByteBuffer	read()	从后端读取下一条WAL记录。如果读取不到，该方法阻塞I/O读。	SQLException

返回值	方法	描述	throws
ByteBuffer	readPending()	从后端读取下一条WAL记录。如果读取不到，该方法不阻塞I/O读。	SQLException
void	setAppliedLSN(LogSequenceNumber applied)	设置应用的LSN。	-
void	setFlushedLSN(LogSequenceNumber flushed)	设置刷新的LSN，在下次更新时发送至后端，用于推进服务端LSN。	-

5.3.17.17 ChainedStreamBuilder

ChainedStreamBuilder是GaussDB JDBC驱动中提供的一个API接口类，用于构建复制流。

ChainedStreamBuilder 的继承关系

ChainedStreamBuilder是逻辑复制的接口，实现类是ReplicationStreamBuilder，该类位于org.postgresql.replication.fluent Package中，该类的声明如下：

```
public class ReplicationStreamBuilder implements ChainedStreamBuilder
```

构造方法

public ReplicationStreamBuilder(final BaseConnection connection)

常用方法

表 5-23 ReplicationStreamBuilder 常用方法

返回值	方法	描述	throws
ChainedLogicalStreamBuilder	logical()	创建逻辑复制流	-
ChainedPhysicalStreamBuilder	physical()	创建物理复制流	-

5.3.17.18 ChainedCommonStreamBuilder

ChainedCommonStreamBuilder是GaussDB JDBC驱动中提供的一个API接口类，用于为逻辑和物理复制指定通用参数。

ChainedCommonStreamBuilder 的继承关系

ChainedCommonStreamBuilder是逻辑复制的接口，实现抽象类是AbstractCreateSlotBuilder，该类的继承类是LogicalCreateSlotBuilder，位于org.postgresql.replication.fluent.logical Package中，该类的声明如下：

```
public class LogicalCreateSlotBuilder
    extends AbstractCreateSlotBuilder<ChainedLogicalCreateSlotBuilder>
    implements ChainedLogicalCreateSlotBuilder
```

构造方法

public LogicalCreateSlotBuilder(BaseConnection connection)

常用方法

表 5-24 LogicalCreateSlotBuilder 常用方法

返回值	方法	描述	throws
T	withSlotName(String slotName)	指定复制槽名。	-
ChainedLogicalCreateSlotBuilder	withOutputPlugin(String outputPlugin)	插件名称，当前支持mppdb_decoding。	-
void	make()	在数据库中创建具有指定参数的插槽。	SQLException
ChainedLogicalCreateSlotBuilder	self()	-	-

5.3.18 JDBC 数据类型映射关系

数据类型、JAVA变量类型以及JDBC类型索引关系如下：

兼容模式	GaussDB数据类型	JAVA变量类型	JDBC类型索引
ORA/MYSQL	oid	java.lang.Long	java.sql.Types.BIGINT
ORA/MYSQL	numeric	java.math.BigDecimal	java.sql.Types.NUMERIC
ORA/MYSQL	tinyint	java.lang.Integer	java.sql.Types.TINYINT
ORA/MYSQL	smallint	java.lang.Integer	java.sql.Types.SMALLINT

兼容模式	GaussDB数据类型	JAVA变量类型	JDBC类型索引
ORA/MYSQL	bigint	java.lang.Long	java.sql.Types.BIGINT
ORA/MYSQL	float4	java.lang.Float	java.sql.Types.REAL
ORA/MYSQL	float8	java.lang.Double	java.sql.Types.DOUBLE
ORA/MYSQL	char	java.lang.String	java.sql.Types.CHAR
ORA/MYSQL	character	java.lang.String	java.sql.Types.CHAR
ORA/MYSQL	bpchar	java.lang.String	java.sql.Types.CHAR
ORA/MYSQL	character varying	java.lang.String	java.sql.Types.VARCHAR
ORA/MYSQL	varchar	java.lang.String	java.sql.Types.VARCHAR
ORA/MYSQL	text	java.lang.String	java.sql.Types.VARCHAR
ORA/MYSQL	name	java.lang.String	java.sql.Types.VARCHAR
ORA/MYSQL	bytea	byte[]	java.sql.Types.BINARY
ORA/MYSQL	blob	java.sql.Blob	java.sql.Types.BLOB
ORA/MYSQL	clob	java.sql.Clob	java.sql.Types.CLOB
ORA/MYSQL	bool	java.lang.Boolean	java.sql.Types.BIT
MYSQL	date	java.sql.Date	java.sql.Types.DATE
ORA/MYSQL	time	java.sql.Time	java.sql.Types.TIME
ORA/MYSQL	timetz	java.sql.Time	java.sql.Types.TIME
ORA/MYSQL	timestamp	java.sql.Timestamp	java.sql.Types.TIMESTAMP
ORA/MYSQL	smalldatetime	java.sql.Timestamp	java.sql.Types.TIMESTAMP

兼容模式	GaussDB数据类型	JAVA变量类型	JDBC类型索引
ORA/MYSQL	timestamptz	java.sql.Timestamp	java.sql.Types.TIMESTAMP
ORA/MYSQL	refcursor	java.sql.ResultSet	java.sql.Types.REF_CURSOR java.sql.Types.OTHER

5.3.19 常见问题处理

5.3.19.1 batchMode 设置错误

问题现象

设置url参数batchMode=on且reWriteBatchedInserts=true，使用JDBC批量插入数据，抛出异常，提示绑定参数数量与语句需要的参数数量不一致：

```
bind message supplies * parameters, but prepared statement "" requires *
```

示例1：

```
// conn是已经创建的Connection对象，创建该connection的url参数包含
&batchMode=on&reWriteBatchedInserts=true
// 批量绑定参数后执行，绑定参数数量会与改写后的insert语句的栏位数不匹配，抛出异常
// java.sql.BatchUpdateException: bind message supplies 3 parameters, but prepared statement "" requires 6
PreparedStatement stmt = conn.prepareStatement("insert into test_tbl values (?, ?, ?)");

stmt.setInt(1, 1);
stmt.setString(2, "aaa");
stmt.setString(3, "bbbb");
stmt.addBatch();

stmt.setInt(1, 1);
stmt.setString(2, "aaa");
stmt.setString(3, "bbbb");
stmt.addBatch();

stmt.setInt(1, 1);
stmt.setString(2, "aaa");
stmt.setString(3, "bbbb");
stmt.addBatch();

stmt.executeBatch();
```

原因分析

将参数reWriteBatchedInserts设置为true时，批量语句会将多条SQL合并为一条，导致语句中预留参数栏位数发生变化，如果batchMode=on，会按照合并前的SQL绑定参数，导致绑定参数数量与语句需要的参数数量不一致。

处理方法

reWriteBatchedInserts设置为true时，将batchMode设置为off。

5.3.19.2 使用 SSL 方式建连报错或阻塞

问题现象

JDBC使用SSL方式建立连接时，会在客户端获取强随机数，建立连接过程中可能出现以下的报错信息：

场景1：如下报错：

```
"Thread-0" #18 prio=5 os_prio=0 tid=0x00007f2ad0385000 nid=0x5429 runnable [0x00007f2aa069b000]
   java.lang.Thread.State: RUNNABLE
        at java.io.FileInputStream.readBytes(Native Method)
        at java.io.FileInputStream.read(FileInputStream.java:255)
        at sun.security.provider.NativePRNG$RandomIO.readFully(NativePRNG.java:424)
        at sun.security.provider.NativePRNG$RandomIO.ensureBufferValid(NativePRNG.java:526)
        at sun.security.provider.NativePRNG$RandomIO.implNextBytes(NativePRNG.java:545)
        - locked <0x000000067273a950> (a java.lang.Object)
        at sun.security.provider.NativePRNG$RandomIO.access$400(NativePRNG.java:331)
        at sun.security.provider.NativePRNG$Blocking.engineNextBytes(NativePRNG.java:268)
        at java.security.SecureRandom.nextBytes(SecureRandom.java:468)
        at java.security.SecureRandom.next(SecureRandom.java:491)
        at java.util.Random.nextInt(Random.java:329)
        at sun.security.ssl.SSLContextImpl.engineInit(SSLContextImpl.java:106)
        at javax.net.ssl.SSLContext.init(SSLContext.java:282)
        at org.postgresql.ssl.LibPQFactory.<init>(LibPQFactory.java:175)
        at org.postgresql.core.SocketFactoryFactory.getSslSocketFactory(SocketFactoryFactory.java:62)
        at org.postgresql.ssl.MakeSSL.convert(MakeSSL.java:33)
        at org.postgresql.core.v3.ConnectionFactoryImpl.enableSSL(ConnectionFactoryImpl.java:723)
        at org.postgresql.core.v3.ConnectionFactoryImpl.tryConnect(ConnectionFactoryImpl.java:203)
        at org.postgresql.core.v3.ConnectionFactoryImpl.openConnectionImpl(ConnectionFactoryImpl.java:330)
        at org.postgresql.core.ConnectionFactory.openConnection(ConnectionFactory.java:58)
        at org.postgresql.jdbc.PgConnection.<init>(PgConnection.java:357)
```

场景2：建连阻塞。如果连接串中配置了loginTimeout后，会报Connection attempt timed out，如果不配置该参数，会一直阻塞。

原因分析

客户端环境随机数产生的速度太慢，无法满足产品要求，熵源不足，导致服务启动失败。当前已知在一些Linux环境中存在此问题。

处理方法

方法1：启动客户端环境中的haveged服务，增加系统熵池熵值以提高读取随机数的速度。启动命令为：

```
systemctl start haveged
```

方法2：调整客户端jdk配置

打开$JAVA_PATH/jre/lib/security/java.security文件，修改以下两个配置项：

```
securerandom.source=file:/dev/./urandom
securerandom.strongAlgorithms=NativePRNGNonBlocking:SUN
```

须知

方法2的本质是在获取强随机数时，使用伪随机数代替，减少需要消耗的熵值。会影响客户端所有使用该jdk的应用，在获取强随机数时会使用伪随机数代替。

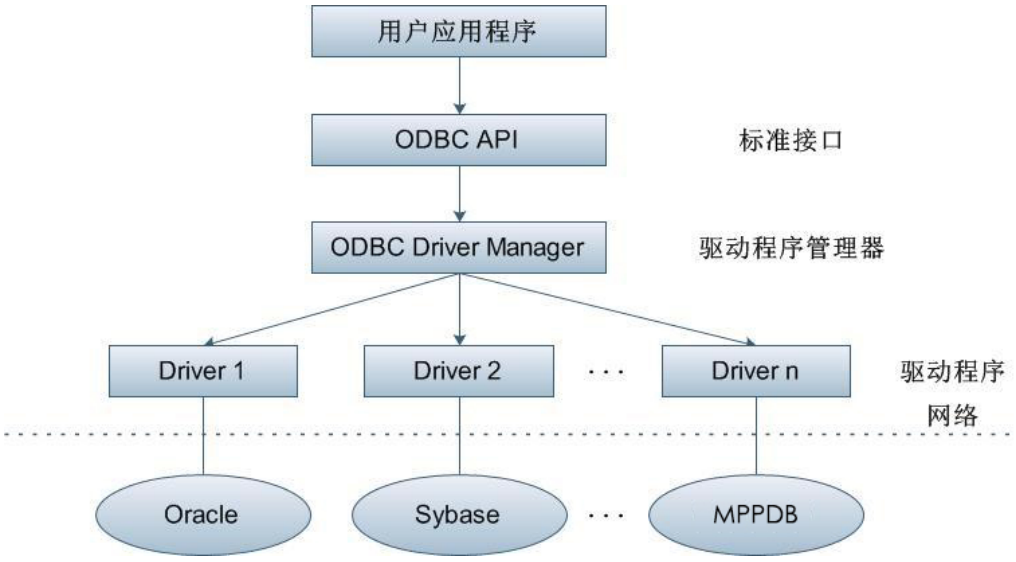
5.4 基于 ODBC 开发

ODBC（Open Database Connectivity，开放数据库互连）是由Microsoft公司基于X/OPEN CLI提出的用于访问数据库的应用程序编程接口。应用程序通过ODBC提供的API

与数据库进行交互，在避免了应用程序直接操作数据库系统的同时，增强了应用程序的可移植性、扩展性和可维护性。

ODBC的系统结构参见图5-2。

图 5-2 ODBC 系统结构



GaussDB目前在以下环境中提供对ODBC3.5的支持。

表 5-25 ODBC 支持平台

操作系统	平台
EulerOS 2.5	x86_64位
EulerOS 2.8	ARM64位
Windows 7	x86_32位
Windows 7	x86_64位
Windows Server 2008	x86_32位
Windows Server 2008	x86_64位
Kylin V10	x86_64位
Kylin V10	ARM64位

UNIX/Linux系统下的驱动程序管理器主要有unixODBC和iODBC，在这选择驱动管理器unixODBC-2.3.0作为连接数据库的组件。

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

说明

当前数据库ODBC驱动基于开源版本，对于华为自研的数据类型，tinyint、smalldatetime、nvarchar2在获取数据类型的时候，可能会出现不兼容。

ODBC 相关约束说明

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

5.4.1 ODBC 包及依赖的库和头文件

Linux 下的 ODBC 包

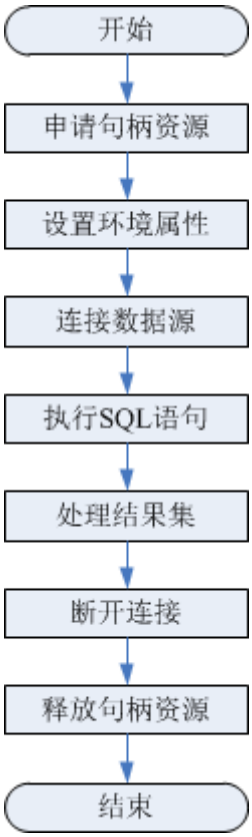
从发布包中获取，包名为GaussDB-Kernel_VxxxRxxxCxx-xxxxx-64bit-Odbc.tar.gz。Linux环境下，开发应用程序要用到unixODBC提供的头文件（sql.h、sqlext.h等）和库libodbc.so。这些头文件和库可从unixODBC-2.3.7的安装包中获得。

Windows 下的 ODBC 包

从发布包中获取，包名为GaussDB-Kernel_VxxxRxxxCxx-Windows-Odbc-X86.tar.gz。Windows环境下，开发应用程序用到的相关头文件和库文件都由系统自带。

5.4.2 开发流程

图 5-3 ODBC 开发应用程序的流程



开发流程中涉及的 API

表 5-26 相关 API 说明

功能	API
申请句柄资源	SQLAllocHandle：申请句柄资源，可替代如下函数： • SQLAllocEnv：申请环境句柄 • SQLAllocConnect：申请连接句柄 • SQLAllocStmt：申请语句句柄
设置环境属性	SQLSetEnvAttr
设置连接属性	SQLSetConnectAttr
设置语句属性	SQLSetStmtAttr
连接数据源	SQLConnect
绑定缓冲区到结果集的列中	SQLBindCol
绑定SQL语句的参数标志和缓冲区	SQLBindParameter
查看最近一次操作错误信息	SQLGetDiagRec
为执行SQL语句做准备	SQLPrepare
执行一条准备好的SQL语句	SQLExecute
直接执行SQL语句	SQLExecDirect
结果集中取行集	SQLFetch
返回结果集中某一列的数据	SQLGetData
获取结果集中列的描述信息	SQLColAttribute
断开与数据源的连接	SQLDisconnect
释放句柄资源	SQLFreeHandle：释放句柄资源，可替代如下函数： • SQLFreeEnv：释放环境句柄 • SQLFreeConnect：释放连接句柄 • SQLFreeStmt：释放语句句柄

说明

数据库中收到的一次执行请求（不在事务块中），如果含有多条语句，将会被打包成一个事务，同时如果其中有一个语句失败，那么整个请求都将会被回滚。

警告

ODBC为应用程序与数据库的中心层，负责把应用程序发出的SQL指令传到数据库当中，自身并不解析SQL语法。故在应用程序中写入带有保密信息的SQL语句时（如明文密码），保密信息会被暴露在驱动日志中。

5.4.3 Linux 下配置数据源

将GaussDB提供的ODBC DRIVER（psqlodbcw.so）配置到数据源中便可使用。配置数据源需要配置“odbc.ini”和“odbcinst.ini”两个文件（在编译安装unixODBC过程中生成且默认放在“/usr/local/etc”目录下），并在服务器端进行配置。

操作步骤

步骤1 获取unixODBC源码包。

获取参考地址：**https://www.unixodbc.org/unixODBC-2.3.7.tar.gz**。

下载后请先按照社区提供的完整性校验算法进行完整性校验。

步骤2 安装unixODBC。如果机器上已经安装了其他版本的unixODBC，可以直接覆盖安装。下载**https://www.unixodbc.org/unixODBC-2.3.7.tar.gz.md5**，查看MD5值，对比MD5值是否与源码包一致。

以unixODBC-2.3.7版本为例，在客户端执行如下命令安装unixODBC。

```
tar zxvf unixODBC-2.3.7.tar.gz
cd unixODBC-2.3.7

./configure --enable-gui=no #如果要在ARM服务器上编译，请追加一个configure参数： --build=aarch64-unknown-linux-gnu
make
#安装可能需要root权限
make install
```

说明

- 目前不支持unixODBC-2.2.1版本。
- 默认安装到“/usr/local”目录下，生成数据源文件到 “/usr/local/etc”目录下，库文件生成在“/usr/local/lib”目录。
- 通过编译带有--enable-fastvalidate=yes选项的unixODBC来获得更高性能。但此选项可能会导致向ODBC API传递无效句柄的应用程序发生故障，而不是返回SQL_INVALID_HANDLE错误。

步骤3 替换客户端GaussDB驱动程序。

将GaussDB-Kernel_VxxxRxxxCxx-xxxxx-64bit-Odbc.tar.gz解压。解压后会得到两个文件夹：lib与odbc，在odbc文件夹中还会有一个lib文件夹。将解压后得到的/lib文件夹与/odbc/lib文件夹中的所有动态库都拷贝到“/usr/local/lib”目录下。

步骤4 配置数据源。

1. 配置ODBC驱动文件。

   在“/usr/local/etc/odbcinst.ini”文件中追加以下内容。

   ```
   [GaussMPP]
   Driver64=/usr/local/lib/psqlodbcw.so
   setup=/usr/local/lib/psqlodbcw.so
   ```

   odbcinst.ini文件中的配置参数说明如**表5-27**所示。

表 5-27 odbcinst.ini 文件配置参数

参数	描述	示例
[DriverName]	驱动器名称，对应数据源DSN中的驱动名。	[DRIVER_N]
Driver64	驱动动态库的路径。	Driver64=/usr/local/lib/psqlodbcw.so
setup	驱动安装路径，与Driver64中动态库的路径一致。	setup=/usr/local/lib/psqlodbcw.so

2. 配置数据源文件。

   在“/usr/local/etc/odbc.ini ”文件中追加以下内容。

```
[gaussdb]
Driver=GaussMPP
Servername=127.0.0.1（数据库Server IP）
Database=postgres （数据库名）
Username=omm （数据库用户名）
Password= （数据库用户密码）
Port=8000（数据库侦听端口）
Sslmode=allow
```

   odbc.ini文件配置参数说明如表5-28所示。

表 5-28 odbc.ini 文件配置参数

参数	描述	示例
[DSN]	数据源的名称。	[gaussdb]
Driver	驱动名，对应odbcinst.ini中的DriverName。	Driver=DRIVER_N
Servername	服务器的IP地址。可配置多个IP地址。	Servername=127.0.0.1
Database	要连接的数据库的名称。	Database=postgres
Username	数据库用户名称。	Username=omm

参数	描述	示例
Password	数据库用户密码。	Password= **说明** ODBC驱动本身已经对内存密码进行过清理，以保证用户密码在连接后不会再在内存中保留。 但是如果配置了此参数，由于UnixODBC对数据源文件等进行缓存，可能导致密码长期保留在内存中。 推荐在应用程序连接时，将密码传递给相应API，而非写在数据源配置文件中。同时连接成功后，应当及时清理保存密码的内存段。 **注意** 配置文件中填写密码时，需要遵循http规则： 1. 字符应当采用URL编码规范，如"!"应写作"%21"，"%"应写作"%25"，因此应当注意特殊处理%。 2. "+"会被替换为空格" "。
Port	服务器的端口号。当开启负载均衡时，可配置多个端口号，且需与配置的多IP一一对应。如果开启负载均衡配置多个IP时，仍只配置一个端口号，则默认所有IP共用同一个端口号，即为配置的端口号。	Port=8000
Sslmode	开启SSL模式	Sslmode=allow
Debug	设置为1时，将会打印psqlodbc驱动的mylog，日志生成目录为/tmp/。设置为0时则不会生成。	Debug=1
UseServerSidePrepare	是否开启数据库端扩展查询协议。 可选值0或1，默认为1，表示打开扩展查询协议。	UseServerSidePrepare=1

参数	描述	示例
UseBatchProtocol	是否开启批量查询协议（打开可提高DML性能）；可选值0或者1，默认为1。 当此值为0时，不使用批量查询协议（主要用于与早期数据库版本通信兼容）。 当此值为1，并且数据库support_batch_bind参数存在且为on时，将打开批量查询协议。	UseBatchProtocol=1
ForExtensionConnector	这个开关控制着savepoint是否发送，savepoint相关问题可以注意这个开关，默认值为1。取值为0，发送savepoint，取值为1，不发送savepoint。	ForExtensionConnector=1
ConnectionExtraInfo	GUC参数connection_info中显示驱动部署路径和进程属主用户的开关。	ConnectionExtraInfo=1 **说明** 默认值为0。当设置为1时，ODBC驱动会将当前驱动的部署路径、进程属主用户上报到数据库中，记录在connection_info参数里；同时可以在PG_STAT_ACTIVITY和PGXC_STAT_ACTIVITY中查询到。
BoolAsChar	设置为Yes，Bools值将会映射为SQL_CHAR。如不设置将会映射为SQL_BIT。默认值为Yes。	BoolsAsChar = Yes
RowVersioning	当尝试更新一行数据时，设置为Yes会允许应用检测数据有没有被其他用户进行修改。默认值为No。	RowVersioning=Yes
ShowSystemTables	设置为Yes，驱动将默认系统表格视为普通SQL表格。默认值为No。	ShowSystemTables=Yes
AutoBalance	ODBC控制负载均衡的开关，默认值为0，0为关闭，1为开启。即为除1以外均不生效。	AutoBalance=1

参数	描述	示例
RefreshCNListTime	开启负载均衡时可配置该参数，该值为刷新CN列表的时间，默认值为10，整数型，单位秒。	RefreshCNListTime=5
Priority	开启负载均衡时可配置该参数，默认值为0，0为关闭，1为开启。即除1以外均不生效。当Priority开启时，应用程序发起的所有连接优先发送到配置文件中配置的CN上，当配置的CN全部不可用时，连接才会发送到剩余的CN上。	Priority=1
UsingEip	开启负载均衡时可配置该参数，默认值为0，0为关闭，1为开启。即除1以外均不生效。此值用于控制是否使用弹性公网IP做负载均衡。当UsingEip开启时，表示使用弹性公网IP做负载均衡；关闭表示使用数据IP做负载均衡。	UsingEip=1
TcpUserTimeout	在支持TCP_USER_TIMEOUT套接字选项的操作系统上，指定传输的数据在TCP连接被强制关闭之前可以保持未确认状态的最大时长。0值表示使用系统缺省。通过UNIX域套接字做的链接忽略这个参数。单位为毫秒，默认为0。	TcpUserTimeout=5000

其中关于Sslmode的选项的允许值，具体信息见下表：

表 5-29 sslmode 的可选项及其描述

sslmode	是否会启用SSL加密	描述
disable	否	不使用SSL安全连接。
allow	可能	如果数据库服务器要求使用，则可以使用SSL安全加密连接，但不验证数据库服务器的真实性。

sslmode	是否会启用SSL加密	描述
prefer	可能	如果数据库支持，那么首选使用SSL安全加密连接，但不验证数据库服务器的真实性。
require	是	必须使用SSL安全连接，但是只做了数据加密，而并不验证数据库服务器的真实性。
verify-ca	是	必须使用SSL安全连接，并且验证数据库是否具有可信证书机构签发的证书。
verify-full	是	必须使用SSL安全连接，在verify-ca的验证范围之外，同时验证数据库所在主机的主机名是否与证书内容一致。如果不一致，需要使用root用户修改/etc/hosts文件，将连接的数据库节点的IP地址和主机名加入。 **说明** 此模式不支持产品默认证书，生成证书请联系管理员处理。

步骤5 SSL模式。具体操作请联系管理员处理。

步骤6 配置数据库服务器。具体操作请联系管理员处理。

步骤7 配置环境变量。

```
vim ~/.bashrc
```

在配置文件中追加以下内容。

```
export LD_LIBRARY_PATH=/usr/local/lib/:$LD_LIBRARY_PATH
export ODBCSYSINI=/usr/local/etc
export ODBCINI=/usr/local/etc/odbc.ini
```

步骤8 执行如下命令使设置生效。

```
source ~/.bashrc
```

----结束

测试数据源配置

安装后/usr/bin下面会存放生成的二进制，可执行isql -v gaussdb(数据源名称)命令。

- 如果显示如下信息，表明配置正确，连接成功。

```
+---------------------------------------+
| Connected!                            |
|                                       |
| sql-statement                         |
| help [tablename]                      |
| quit                                  |
|                                       |
+---------------------------------------+
```

- 若显示ERROR信息，则表明配置错误。请检查上述配置是否正确。
- 若是集群环境，需要在所有机器上都拷贝配置一份unixODBC。

说明

目前通过ODBC连接数据库时，会如下设置内核参数：

```
SET extra_float_digits = 2;
SET DateStyle = 'ISO';
```

这些参数可能会导致ODBC客户端的行为与gsql客户端的行为不一致，例如，Date数据显示方式、浮点数精度表示。如果实际期望和这些配置不符，建议在ODBC应用代码中显式设定这些参数。

常见问题处理

- [UnixODBC][Driver Manager]Can't open lib 'xxx/xxx/psqlodbcw.so' : file not found.

  此问题的可能原因：

  - odbcinst.ini文件中配置的路径不正确

    确认的方法：使用ls命令查看错误信息中的路径，以确保该psqlodbcw.so文件存在，同时具有执行权限。

  - psqlodbcw.so的依赖库不存在，或者不在系统环境变量中

    确认的办法：使用ldd查看错误信息中的路径，如果是缺少libodbc.so.1等UnixODBC的库，那么按照“操作步骤”中的方法重新配置UnixODBC，并确保它的安装路径下的lib目录添加到了LD_LIBRARY_PATH中；如果重装仍无法解决，可以手动将数据库安装包下的unixodbc/lib下的内容拷贝到UnixODBC的安装路径下的lib目录；如果是缺少其他库，请将ODBC驱动包中的lib目录添加到LD_LIBRARY_PATH中。如果缺少其他标准库，请自行安装。

- [UnixODBC]connect to server failed: no such file or directory

  此问题可能的原因：

  - 配置了错误的/不可达的数据库地址，或者端口

    请检查数据源配置中的Servername及Port配置项。

  - 服务器侦听不正确

    如果确认Servername及Port配置正确，请根据“操作步骤”中数据库服务器的相关配置，确保数据库侦听了合适的网卡及端口。

  - 防火墙及网闸设备

    请确认防火墙设置，将数据库的通信端口添加到可信端口中。

    如果有网闸设备，请确认相关的设置。

- [unixODBC]The password-stored method is not supported.

  此问题可能原因：

  数据源中未配置sslmode配置项。

  解决办法：

  请配置该选项至allow或以上选项。此配置的更多信息，见**表5-29**。

- Server common name "xxxx" does not match host name "xxxxx"

  此问题的原因：

  使用了SSL加密的“verify-full”选项，驱动程序会验证证书中的主机名与实际部署数据库的主机名是否一致。

  解决办法：

  碰到此问题可以使用“verify-ca”选项，不再校验主机名；或者重新生成一套与数据库所在主机名相同的服务端证书。

- Driver's SQLAllocHandle on SQL_HANDLE_DBC failed

  此问题的可能原因：

  可执行文件（比如UnixODBC的isql，以下都以isql为例）与数据库驱动（psqlodbcw.so）依赖于不同的odbc的库版本：libodbc.so.1或者libodbc.so.2。此问题可以通过如下方式确认：

  ```
  ldd `which isql` | grep odbc
  ldd psqlodbcw.so | grep odbc
  ```

  这时，如果输出的libodbc.so最后的后缀数字不同或者指向不同的磁盘物理文件，那么基本就可以断定是此问题。isql与psqlodbcw.so都会要求加载libodbc.so，这时如果它们加载的是不同的物理文件，便会导致两套完全同名的函数列表，同时出现在同一个可见域里（UnixODBC的libodbc.so.*的函数导出列表完全一致），产生冲突，无法加载数据库驱动。

  解决办法：

  确定一个要使用的UnixODBC，然后卸载另外一个（比如卸载库版本号为.so.2的UnixODBC），然后将剩下的.so.1的库，新建一个同名但是后缀为.so.2的软链接，便可解决此问题。

- FATAL: Forbid remote connection with trust method!

  由于安全原因，数据库CN禁止集群内部其他节点无认证接入。

  如果要在集群内部访问CN，请将ODBC程序部署在CN所在机器，服务器地址使用"127.0.0.1"。建议业务系统单独部署在集群外部，否则可能会影响数据库运行性能。

- [unixODBC][Driver Manager]Invalid attribute value

  在使用SQL on other GaussDB功能时碰到此问题，有可能是unixODBC的版本并非推荐版本，建议通过“odbcinst --version”命令排查环境中的unixODBC版本。

- authentication method 10 not supported.

  使用开源客户端碰到此问题，可能原因：

  数据库中存储的口令校验只存储了SHA256格式哈希，而开源客户端只识别MD5校验，双方校验方法不匹配报错。

  说明

  - 数据库并不存储用户口令，只存储用户口令的哈希码。
  - 数据库当用户更新用户口令或者新建用户时，会同时存储两种格式的哈希码，这时将兼容开源的认证协议。
  - 但是当老版本升级到新版本时，由于哈希的不可逆性，所以数据库无法还原用户口令，进而生成新格式的哈希，所以仍然只保留了SHA256格式的哈希，导致仍然无法使用MD5做口令认证。
  - MD5加密算法安全性低，存在安全风险，建议使用更安全的加密算法。

  要解决该问题，可以更新用户口令（参见**ALTER USER**）；或者新建一个用户（参见**CREATE USER**），赋予同等权限，使用新用户连接数据库。

- unsupported frontend protocol 3.51: server supports 1.0 to 3.0

  目标数据库版本过低，或者目标数据库为开源数据库。请使用对应版本的数据库驱动连接目标数据库。

- FATAL: GSS authentication method is not allowed because XXXX user password is not disabled.

  目标CN的pg_hba.conf里配置了当前客户端IP使用"gss"方式来做认证，该认证算法不支持用作客户端的身份认证，请修改到"sha256"后再试。配置方法见**步骤6**。

- isql：error while loading shared libraries:xxx
  环境缺少该动态库，需要自行安装对应的库。

5.4.4 Windows 下配置数据源

Windows操作系统自带ODBC数据源管理器，无需用户手动安装管理器便可直接进行配置。

操作步骤

步骤1 替换客户端GaussDB驱动程序。

将GaussDB-Kernel_VxxxRxxxCxx-Windows-Odbc.tar.gz解压后，根据需要，单击psqlodbc.exe（32位）进行驱动安装。

步骤2 打开驱动管理器。

在配置数据源时，请使用**32**位的ODBC驱动管理器（目前仅支持**32**位的ODBC驱动管理器，假设操作系统安装盘符为C盘，如果是其他盘符，请对路径做相应修改）：

- **64**位操作系统请使用：C:\Windows**SysWOW64**\odbcad32.exe，请勿直接使用“控制面板 > 管理工具 > 数据源(ODBC)”。

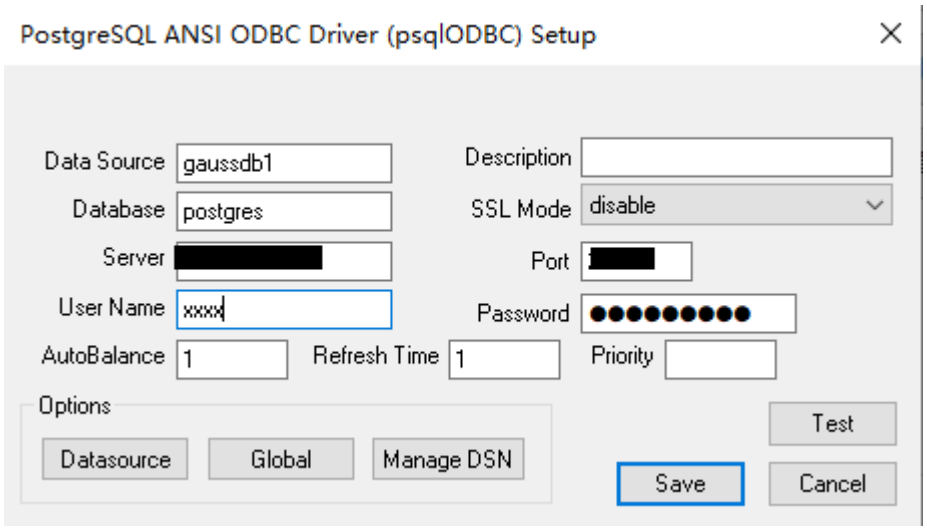
  说明

  WOW64的全称是"Windows 32-bit on Windows 64-bit"，C:\Windows\SysWOW64\存放的是64位系统上的32位运行环境。而C:\Windows\System32\存放的是与操作系统一致的运行环境，具体的技术信息请查阅Windows的相关技术文档。

- **32**位操作系统请使用：C:\Windows\System32\odbcad32.exe，或者单击“计算机 > 控制面板 > 管理工具 > 数据源(ODBC)”打开驱动管理器。

步骤3 配置数据源。

在打开的驱动管理器上，选择“用户DSN > 添加 > PostgreSQL Unicode”，然后进行配置：



须知

此界面上配置的用户名及密码信息，将会被记录在Windows注册表中，再次连接数据库时就不再需要输入认证信息。但是出于安全考虑，建议在单击"Save"按钮保存配置信息前，清空相关敏感信息；在使用ODBC的连接API时，再传入所需的用户名、密码信息。

步骤4 SSL模式。

将**步骤3**中的设置窗口的“SSL Mode”选项调整至“require”。

表 5-30 sslmode 的可选项及其描述

sslmode	是否会启用SSL加密	描述
disable	否	不使用SSL安全连接。
allow	可能	如果数据库服务器要求使用，则可以使用SSL安全加密连接，但不验证数据库服务器的真实性。
prefer	可能	如果数据库支持，那么首选使用SSL安全加密连接，但不验证数据库服务器的真实性。
require	是	必须使用SSL安全连接，但是只做了数据加密，而并不验证数据库服务器的真实性。
verify-ca	是	必须使用SSL安全连接，并且验证数据库是否具有可信证书机构签发的证书。当前windows ODBC不支持cert方式认证。
verify-full	是	必须使用SSL安全连接，在verify-ca的验证范围之外，同时验证数据库所在主机的主机名是否与证书内容一致。当前windows odbc不支持cert方式认证。

步骤5 配置GaussDB服务器。具体操作请联系管理员处理。

步骤6 执行如下命令重启集群。

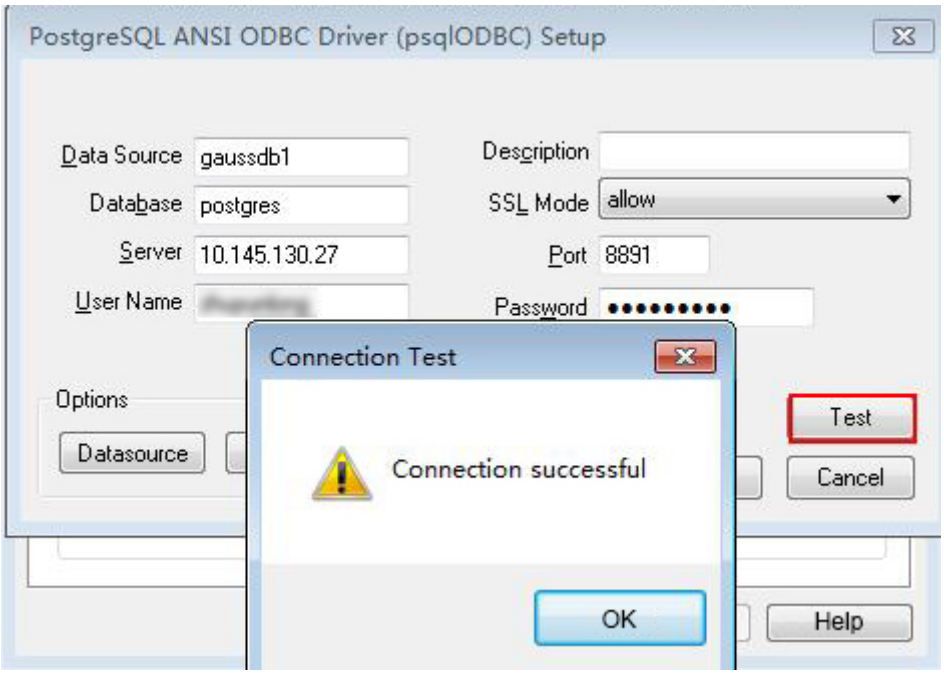
```
gs_om -t stop
gs_om -t start
```

----结束

测试数据源配置

单击Test进行测试。

- 如果显示如下，则表明配置正确，连接成功。



- 若显示ERROR信息，则表明配置错误。请检查上述配置是否正确。

说明

- 目前通过ODBC连接数据库时，会设置以下内核参数：

```
SET extra_float_digits = 2;
SET DateStyle = 'ISO';
```

这些参数可能会导致ODBC客户端的行为与gsql客户端的行为不一致，例如，Date数据显示方式、浮点数精度表示。如果实际期望和这些配置不符，建议在ODBC应用代码中显式设定这些参数。

常见问题处理

- connect to server failed: no such file or directory

  此问题可能的原因：

  - 配置了错误的/不可达的数据库地址，或者端口

    请检查数据源配置中的Server及Port配置项。

  - 服务器侦听不正确

    如果确认Server及Port配置正确，请根据“操作步骤”中数据库服务器的相关配置，确保数据库侦听了合适的网卡及端口。

  - 防火墙及网闸设备

    请确认防火墙设置，将数据库的通信端口添加到可信端口中。

    如果有网闸设备，请确认相关的设置。

- The password-stored method is not supported.

  此问题可能原因：

  数据源中未配置sslmode配置项，请调整此项至allow或以上级别，允许SSL连接，此选项的更多说明，请见**表5-30**。

- authentication method 10 not supported.

  使用开源客户端碰到此问题，可能原因：

数据库中存储的口令校验只存储了SHA256格式哈希，而开源客户端只识别MD5校验，双方校验方法不匹配报错。

说明

- 数据库并不存储用户口令，只存储用户口令的哈希码。
- 数据库当用户更新用户口令或者新建用户时，会同时存储两种格式的哈希码，这时将兼容开源的认证协议。
- 但是当老版本升级到新版本时，由于哈希的不可逆性，所以数据库无法还原用户口令，进而生成新格式的哈希，所以仍然只保留了SHA256格式的哈希，导致仍然无法使用MD5做口令认证。
- MD5加密算法安全性低，存在安全风险，建议使用更安全的加密算法。

要解决该问题，可以更新用户口令（参见**ALTER USER**）；或者新建一个用户（参见**CREATE USER**），赋予同等权限，使用新用户连接数据库。

- unsupported frontend protocol 3.51: server supports 1.0 to 3.0

  目标数据库版本过低，或者目标数据库为开源数据库。请使用对应版本的数据库驱动连接目标数据库。

- FATAL: GSS authentication method is not allowed because XXXX user password is not disabled.

  目标CN的pg_hba.conf里配置了当前客户端IP使用"gss"方式来做认证，该认证算法不支持用作客户端的身份认证，请修改到"sha256"后再试。配置方法见**步骤5**。

5.4.5 示例：常用功能和批量绑定

常用功能示例代码

```
// 此示例演示如何通过ODBC方式获取GaussDB中的数据。
// DBtest.c (compile with: libodbc.so)
#include <stdlib.h>
#include <stdio.h>
#include <sqlext.h>
#ifdef WIN32
#include <windows.h>
#endif
SQLHENV       V_OD_Env;        // Handle ODBC environment
SQLHSTMT      V_OD_hstmt;      // Handle statement
SQLHDBC       V_OD_hdbc;       // Handle connection
char          typename[100];
SQLINTEGER    value = 100;
SQLINTEGER    V_OD_erg,V_OD_buffer,V_OD_err,V_OD_id;
int main(int argc,char *argv[])
{
    // 1. 申请环境句柄
    V_OD_erg = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&V_OD_Env);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
    {
        printf("Error AllocHandle\n");
        exit(0);
    }
    // 2. 设置环境属性（版本信息）
    SQLSetEnvAttr(V_OD_Env, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0);
    // 3. 申请连接句柄
    V_OD_erg = SQLAllocHandle(SQL_HANDLE_DBC, V_OD_Env, &V_OD_hdbc);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
    {
        SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
        exit(0);
    }
    // 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量(环境变量名称请根据自身情况进行设置)EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
```



```
    char *userName;
    userName = getenv("EXAMPLE_USERNAME_ENV");
    char *password;
    password = getenv("EXAMPLE_PASSWORD_ENV");
    // 4. 设置连接属性
    SQLSetConnectAttr(V_OD_hdbc, SQL_ATTR_AUTOCOMMIT, SQL_AUTOCOMMIT_ON, 0);
    // 5. 连接数据源，这里的userName与password分别表示连接数据库的用户名和用户密码。
    // 如果odbc.ini文件中已经配置了用户名密码，那么这里可以留空（""）；但是不建议这么做，因为一旦
odbc.ini权限管理不善，将导致数据库用户密码泄露。
    V_OD_erg = SQLConnect(V_OD_hdbc, (SQLCHAR*) "gaussdb", SQL_NTS,
                          (SQLCHAR*) userName, SQL_NTS,  (SQLCHAR*) password, SQL_NTS);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
    {
        printf("Error SQLConnect %d\n",V_OD_erg);
        SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
        exit(0);
    }
    printf("Connected !\n");
    // 6. 设置语句属性
    SQLSetStmtAttr(V_OD_hstmt,SQL_ATTR_QUERY_TIMEOUT,(SQLPOINTER *)3,0);
    // 7. 申请语句句柄
    SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
    // 8. 直接执行SQL语句。
    SQLExecDirect(V_OD_hstmt,"drop table IF EXISTS customer_t1",SQL_NTS);
    SQLExecDirect(V_OD_hstmt,"CREATE TABLE customer_t1(c_customer_sk INTEGER, c_customer_name
VARCHAR(32));",SQL_NTS);
    SQLExecDirect(V_OD_hstmt,"insert into customer_t1 values(25,'li')",SQL_NTS);
    // 9. 准备执行
    SQLPrepare(V_OD_hstmt,"insert into customer_t1 values(?)",SQL_NTS);
    // 10. 绑定参数
    SQLBindParameter(V_OD_hstmt,1,SQL_PARAM_INPUT,SQL_C_SLONG,SQL_INTEGER,0,0,
                     &value,0,NULL);
    // 11. 执行准备好的语句
    SQLExecute(V_OD_hstmt);
    SQLExecDirect(V_OD_hstmt,"select c_customer_sk from customer_t1",SQL_NTS);
    // 12. 获取结果集某一列的属性
    SQLColAttribute(V_OD_hstmt,1,SQL_DESC_TYPE,typename,100,NULL,NULL);
    printf("SQLColAtrribute %s\n",typename);
    // 13. 绑定结果集
    SQLBindCol(V_OD_hstmt,1,SQL_C_SLONG, (SQLPOINTER)&V_OD_buffer,150,
            (SQLLEN *)&V_OD_err);
    // 14. 通过SQLFetch取结果集中数据
    V_OD_erg=SQLFetch(V_OD_hstmt);
    // 15. 通过SQLGetData获取并返回数据。
    while(V_OD_erg != SQL_NO_DATA)
    {
        SQLGetData(V_OD_hstmt,1,SQL_C_SLONG,(SQLPOINTER)&V_OD_id,0,NULL);
        printf("SQLGetData ----ID = %d\n",V_OD_id);
        V_OD_erg=SQLFetch(V_OD_hstmt);
    };
    printf("Done !\n");
    // 16. 断开数据源连接并释放句柄资源
    SQLFreeHandle(SQL_HANDLE_STMT,V_OD_hstmt);
    SQLDisconnect(V_OD_hdbc);
    SQLFreeHandle(SQL_HANDLE_DBC,V_OD_hdbc);
    SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
    return(0);
}
```

批量绑定示例代码

```
/*************************************************************************
* 请在数据源中打开UseBatchProtocol，同时指定数据库中参数support_batch_bind
* 为on
* CHECK_ERROR的作用是检查并打印错误信息。
* 此示例将与用户交互式获取DSN、模拟的数据量，忽略的数据量，并将最终数据入库到test_odbc_batch_insert
中
*************************************************************************/
#ifdef WIN32
```



```
#include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
#include <string.h>

void Exec(SQLHDBC hdbc, SQLCHAR* sql)
{
    SQLRETURN retcode;                  // Return status
    SQLHSTMT hstmt = SQL_NULL_HSTMT;    // Statement handle
    SQLCHAR     loginfo[2048];

    // Allocate Statement Handle
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLAllocHandle(SQL_HANDLE_STMT) failed");
        return;
    }

    // Prepare Statement
    retcode = SQLPrepare(hstmt, (SQLCHAR*) sql, SQL_NTS);
    sprintf((char*)loginfo, "SQLPrepare log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLPrepare(hstmt, (SQLCHAR*) sql, SQL_NTS) failed");
        return;
    }

    // Execute Statement
    retcode = SQLExecute(hstmt);
    sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLExecute(hstmt) failed");
        return;
    }
    // Free Handle
    retcode = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
    sprintf((char*)loginfo, "SQLFreeHandle stmt log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLFreeHandle(SQL_HANDLE_STMT, hstmt) failed");
        return;
    }
}

int main ()
{
    SQLHENV  henv  = SQL_NULL_HENV;
    SQLHDBC  hdbc  = SQL_NULL_HDBC;
    int      batchCount = 1000; // 批量绑定的数据量
    SQLLEN   rowsCount = 0;
    int      ignoreCount = 0; // 批量绑定的数据中，不要入库的数据量
    int      i = 0;

    SQLRETURN   retcode;
    SQLCHAR     dsn[1024] = {'\0'};
    SQLCHAR     loginfo[2048];

    do
    {
        if (ignoreCount > batchCount)
        {
            printf("ignoreCount(%d) should be less than batchCount(%d)\n", ignoreCount, batchCount);
        }
    }while(ignoreCount > batchCount);
```



```
retcode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLAllocHandle failed");
    goto exit;
}

// Set ODBC Verion
retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
                                (SQLPOINTER*)SQL_OV_ODBC3, 0);

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLSetEnvAttr failed");
    goto exit;
}

// Allocate Connection
retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLAllocHandle failed");
    goto exit;
}

// Set Login Timeout
retcode = SQLSetConnectAttr(hdbc, SQL_LOGIN_TIMEOUT, (SQLPOINTER)5, 0);

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLSetConnectAttr failed");
    goto exit;
}

// Set Auto Commit
retcode = SQLSetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT,
                                (SQLPOINTER)(1), 0);

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLSetConnectAttr failed");
    goto exit;
}

// Connect to DSN
// gaussdb替换成用户所使用的数据源名称
sprintf(loginfo, "SQLConnect(DSN:%s)", dsn);
retcode = SQLConnect(hdbc, (SQLCHAR*) "gaussdb", SQL_NTS,
                        (SQLCHAR*) NULL, 0, NULL, 0);

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLConnect failed");
    goto exit;
}

// init table info.
Exec(hdbc, "drop table if exists test_odbc_batch_insert");
Exec(hdbc, "create table test_odbc_batch_insert(id int primary key, col varchar2(50))");

// 下面的代码根据用户输入的数据量，构造出将要入库的数据:
{
    SQLRETURN retcode;
    SQLHSTMT hstmtinesrt = SQL_NULL_HSTMT;
    SQLCHAR      *sql = NULL;
    SQLINTEGER   *ids  = NULL;
    SQLCHAR      *cols = NULL;
    SQLLEN       *bufLenIds = NULL;
    SQLLEN       *bufLenCols = NULL;
    SQLUSMALLINT *operptr = NULL;
    SQLUSMALLINT *statusptr = NULL;
    SQLULEN      process = 0;
```



```
    // 这里是按列构造，每个字段的内存连续存放在一起。
    ids = (SQLINTEGER*)malloc(sizeof(ids[0]) * batchCount);
    cols = (SQLCHAR*)malloc(sizeof(cols[0]) * batchCount * 50);
    // 这里是每个字段中，每一行数据的内存长度。
    bufLenIds = (SQLLEN*)malloc(sizeof(bufLenIds[0]) * batchCount);
    bufLenCols = (SQLLEN*)malloc(sizeof(bufLenCols[0]) * batchCount);
    // 该行是否需要被处理，SQL_PARAM_IGNORE 或 SQL_PARAM_PROCEED
    operptr = (SQLUSMALLINT*)malloc(sizeof(operptr[0]) * batchCount);
    memset(operptr, 0, sizeof(operptr[0]) * batchCount);
    // 该行的处理结果。
    // 注：由于数据库中处理方式是同一语句隶属同一事务中，所以如果出错，那么待处理数据都将是出错的，并不会部分入库。
    statusptr = (SQLUSMALLINT*)malloc(sizeof(statusptr[0]) * batchCount);
    memset(statusptr, 88, sizeof(statusptr[0]) * batchCount);

    if (NULL == ids || NULL == cols || NULL == bufLenCols || NULL == bufLenIds)
    {
        fprintf(stderr, "FAILED:\tmalloc data memory failed\n");
        goto exit;
    }

    for (i = 0; i < batchCount; i++)
    {
        ids[i] = i;
        sprintf(cols + 50 * i, "column test value %d", i);
        bufLenIds[i] = sizeof(ids[i]);
        bufLenCols[i] = strlen(cols + 50 * i);
        operptr[i] = (i < ignoreCount) ? SQL_PARAM_IGNORE : SQL_PARAM_PROCEED;
    }

    // Allocate Statement Handle
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmtinesrt);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLAllocHandle failed");
        goto exit;
    }

    // Prepare Statement
    sql = (SQLCHAR*)"insert into test_odbc_batch_insert values(?, ?)";
    retcode = SQLPrepare(hstmtinesrt, (SQLCHAR*) sql, SQL_NTS);
    sprintf((char*)loginfo, "SQLPrepare log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLPrepare failed");
        goto exit;
    }

    retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAMSET_SIZE, (SQLPOINTER)batchCount, sizeof(batchCount));

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLSetStmtAttr failed");
        goto exit;
    }

    retcode = SQLBindParameter(hstmtinesrt, 1, SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER, sizeof(ids[0]), 0,&(ids[0]), 0, bufLenIds);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLBindParameter failed");
        goto exit;
    }

    retcode = SQLBindParameter(hstmtinesrt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 50, 50, cols, 50, bufLenCols);

    if (!SQL_SUCCEEDED(retcode)) {
```



```
        printf("SQLBindParameter failed");
        goto exit;
    }

    retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAMS_PROCESSED_PTR, (SQLPOINTER)&process,
sizeof(process));

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLSetStmtAttr failed");
        goto exit;
    }

    retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAM_STATUS_PTR, (SQLPOINTER)statusptr,
sizeof(statusptr[0]) * batchCount);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLSetStmtAttr failed");
        goto exit;
    }

    retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAM_OPERATION_PTR, (SQLPOINTER)operptr,
sizeof(operptr[0]) * batchCount);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLSetStmtAttr failed");
        goto exit;
    }

    retcode = SQLExecute(hstmtinesrt);
    sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLExecute(hstmtinesrt) failed");
        goto exit;

    retcode = SQLRowCount(hstmtinesrt, &rowsCount);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLRowCount failed");
        goto exit;
    }

    if (rowsCount != (batchCount - ignoreCount))
    {
        sprintf(loginfo, "(batchCount - ignoreCount)(%d) != rowsCount(%d)", (batchCount - ignoreCount),
rowsCount);

        if (!SQL_SUCCEEDED(retcode)) {
            printf("SQLExecute failed");
            goto exit;
        }
    }
    else
    {
        sprintf(loginfo, "(batchCount - ignoreCount)(%d) == rowsCount(%d)", (batchCount - ignoreCount),
rowsCount);

        if (!SQL_SUCCEEDED(retcode)) {
            printf("SQLExecute failed");
            goto exit;
        }
    }

    // check row number returned
    if (rowsCount != process)
    {
        sprintf(loginfo, "process(%d) != rowsCount(%d)", process, rowsCount);

        if (!SQL_SUCCEEDED(retcode)) {
```



```
            printf("SQLExecute failed");
            goto exit;
        }
    }
    else
    {
        sprintf(loginfo, "process(%d) == rowsCount(%d)", process, rowsCount);

        if (!SQL_SUCCEEDED(retcode)) {
            printf("SQLExecute failed");
            goto exit;
        }
    }

    for (i = 0; i < batchCount; i++)
    {
        if (i < ignoreCount)
        {
            if (statusptr[i] != SQL_PARAM_UNUSED)
            {
                sprintf(loginfo, "statusptr[%d](%d) != SQL_PARAM_UNUSED", i, statusptr[i]);

                if (!SQL_SUCCEEDED(retcode)) {
                    printf("SQLExecute failed");
                    goto exit;
                }
            }
        }
        else if (statusptr[i] != SQL_PARAM_SUCCESS)
        {
            sprintf(loginfo, "statusptr[%d](%d) != SQL_PARAM_SUCCESS", i, statusptr[i]);

            if (!SQL_SUCCEEDED(retcode)) {
                printf("SQLExecute failed");
                goto exit;
            }
        }
    }

    retcode = SQLFreeHandle(SQL_HANDLE_STMT, hstmtinesrt);
    sprintf((char*)loginfo, "SQLFreeHandle hstmtinesrt");

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLFreeHandle failed");
        goto exit;
    }
  }


exit:
    (void) printf ("\nComplete.\n");

    // Connection
    if (hdbc != SQL_NULL_HDBC) {
        SQLDisconnect(hdbc);
        SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
    }

    // Environment
    if (henv != SQL_NULL_HENV)
        SQLFreeHandle(SQL_HANDLE_ENV, henv);

    return 0;
}
```


5.4.6 典型应用场景配置

日志诊断场景

ODBC日志分为unixODBC驱动管理器日志和psqlODBC驱动端日志。前者可以用于追溯应用程序API的执行成功与否，后者是底层实现过程中的一些DFX日志，用来帮助定位问题。

unixODBC日志需要在odbcinst.ini文件中配置：

```
[ODBC]
Trace=Yes
TraceFile=/path/to/odbctrace.log

[GaussMPP]
Driver64=/usr/local/lib/psqlodbcw.so
setup=/usr/local/lib/psqlodbcw.so
```

psqlODBC日志只需要在odbc.ini加上：

```
[gaussdb]
Driver=GaussMPP
Servername=10.10.0.13（数据库Server IP）
...
Debug=1（打开驱动端debug日志）
```

说明

unixODBC日志将会生成在TraceFile配置的路径下，psqlODBC会在系统/tmp/下生成mylog_xxx.log。

高性能场景

进行大量数据插入时，建议如下：

- 需要设置批量绑定：odbc.ini配置文件中设置UseBatchProtocol=1、数据库设置support_batch_bind=on。
- ODBC程序绑定类型要和数据库中类型一致。
- 客户端字符集和数据库字符集一致。
- 事务改成手动提交。

odbc.ini配置文件：

```
[gaussdb]
Driver=GaussMPP
Servername=10.10.0.13（数据库Server IP）
...
UseBatchProtocol=1 （默认打开）
ConnSettings=set client_encoding=UTF8 （设置客户端字符编码，保证和server端一致）
```

绑定类型用例：

```
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
#include <string.h>
#include <sys/time.h>

#define MESSAGE_BUFFER_LEN  128
SQLHANDLE   h_env = NULL;
SQLHANDLE   h_conn = NULL;
SQLHANDLE   h_stmt = NULL;
void print_error()
{
    SQLCHAR     Sqlstate[SQL_SQLSTATE_SIZE+1];
```



```
    SQLINTEGER  NativeError;
    SQLCHAR     MessageText[MESSAGE_BUFFER_LEN];
    SQLSMALLINT TextLength;
    SQLRETURN   ret = SQL_ERROR;

    ret = SQLGetDiagRec(SQL_HANDLE_STMT, h_stmt, 1, Sqlstate, &NativeError, MessageText,
MESSAGE_BUFFER_LEN, &TextLength);
    if ( SQL_SUCCESS == ret)
    {
        printf("\n STMT ERROR-%05d %s", NativeError, MessageText);
        return;
    }

    ret = SQLGetDiagRec(SQL_HANDLE_DBC, h_conn, 1, Sqlstate, &NativeError, MessageText,
MESSAGE_BUFFER_LEN, &TextLength);
    if ( SQL_SUCCESS == ret)
    {
        printf("\n CONN ERROR-%05d %s", NativeError, MessageText);
        return;
    }

    ret = SQLGetDiagRec(SQL_HANDLE_ENV, h_env, 1, Sqlstate, &NativeError, MessageText,
MESSAGE_BUFFER_LEN, &TextLength);
    if ( SQL_SUCCESS == ret)
    {
        printf("\n ENV ERROR-%05d %s", NativeError, MessageText);
        return;
    }

    return;
}

/* 期盼函数返回SQL_SUCCESS */
#define RETURN_IF_NOT_SUCCESS(func) \
{\
    SQLRETURN ret_value = (func);\
    if (SQL_SUCCESS != ret_value)\
    {\
        print_error();\
        printf("\n failed line = %u: expect SQL_SUCCESS, but ret = %d", __LINE__, ret_value);\
        return SQL_ERROR; \
    }\
}

/* 期盼函数返回SQL_SUCCESS */
#define RETURN_IF_NOT_SUCCESS_I(i, func) \
{\
    SQLRETURN ret_value = (func);\
    if (SQL_SUCCESS != ret_value)\
    {\
        print_error();\
        printf("\n failed line = %u (i=%d): : expect SQL_SUCCESS, but ret = %d", __LINE__, (i), ret_value);\
        return SQL_ERROR; \
    }\
}

/* 期盼函数返回SQL_SUCCESS_WITH_INFO */
#define RETURN_IF_NOT_SUCCESS_INFO(func) \
{\
    SQLRETURN ret_value = (func);\
    if (SQL_SUCCESS_WITH_INFO != ret_value)\
    {\
        print_error();\
        printf("\n failed line = %u: expect SQL_SUCCESS_WITH_INFO, but ret = %d", __LINE__, ret_value);\
        return SQL_ERROR; \
    }\
}

/* 期盼数值相等 */
```



```
#define RETURN_IF_NOT(expect, value) \
if ((expect) != (value))\
{\
    printf("\n failed line = %u: expect = %u, but value = %u", __LINE__, (expect), (value)); \
    return SQL_ERROR;\
}

/* 期盼字符串相同 */
#define RETURN_IF_NOT_STRCMP_I(i, expect, value) \
if (( NULL == (expect) ) || (NULL == (value)))\
{\
    printf("\n failed line = %u (i=%u): input NULL pointer !", __LINE__, (i)); \
    return SQL_ERROR; \
}\
else if (0 != strcmp((expect), (value)))\
{\
    printf("\n failed line = %u (i=%u): expect = %s, but value = %s", __LINE__, (i), (expect), (value)); \
    return SQL_ERROR;\
}


// prepare + execute SQL语句
int execute_cmd(SQLCHAR *sql)
{
    if ( NULL == sql )
    {
        return SQL_ERROR;
    }

    if ( SQL_SUCCESS != SQLPrepare(h_stmt, sql, SQL_NTS))
    {
        return SQL_ERROR;
    }

    if ( SQL_SUCCESS != SQLExecute(h_stmt))
    {
        return SQL_ERROR;
    }

    return SQL_SUCCESS;
}
// execute + commit 句柄
int commit_exec()
{
    if ( SQL_SUCCESS != SQLExecute(h_stmt))
    {
        return SQL_ERROR;
    }

    // 手动提交
    if ( SQL_SUCCESS != SQLEndTran(SQL_HANDLE_DBC, h_conn, SQL_COMMIT))
    {
        return SQL_ERROR;
    }

    return SQL_SUCCESS;
}

int begin_unit_test()
{
    SQLINTEGER    ret;

    /* 申请环境句柄 */
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &h_env);
    if ((SQL_SUCCESS != ret) && (SQL_SUCCESS_WITH_INFO != ret))
    {
        printf("\n begin_unit_test::SQLAllocHandle SQL_HANDLE_ENV failed ! ret = %d", ret);
        return SQL_ERROR;
    }
```



```
    /* 进行连接前必须要先设置版本号 */
    if (SQL_SUCCESS != SQLSetEnvAttr(h_env, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)SQL_OV_ODBC3,
0))
    {
        print_error();
        printf("\n begin_unit_test::SQLSetEnvAttr SQL_ATTR_ODBC_VERSION failed ! ret = %d", ret);
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
        return SQL_ERROR;
    }

    /* 申请连接句柄 */
    ret = SQLAllocHandle(SQL_HANDLE_DBC, h_env, &h_conn);
    if (SQL_SUCCESS != ret)
    {
        print_error();
        printf("\n begin_unit_test::SQLAllocHandle SQL_HANDLE_DBC failed ! ret = %d", ret);
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
        return SQL_ERROR;
    }

    /* 建立连接 */
 ret = SQLConnect(h_conn, (SQLCHAR*) "gaussdb", SQL_NTS,
                          (SQLCHAR*) NULL, 0, NULL, 0);
    if (SQL_SUCCESS != ret)
    {
        print_error();
        printf("\n begin_unit_test::SQLConnect  failed ! ret = %d", ret);
        SQLFreeHandle(SQL_HANDLE_DBC, h_conn);
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
        return SQL_ERROR;
    }

    /* 申请语句句柄 */
    ret = SQLAllocHandle(SQL_HANDLE_STMT, h_conn, &h_stmt);
    if (SQL_SUCCESS != ret)
    {
        print_error();
        printf("\n begin_unit_test::SQLAllocHandle SQL_HANDLE_STMT failed ! ret = %d", ret);
        SQLFreeHandle(SQL_HANDLE_DBC, h_conn);
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
        return SQL_ERROR;
    }

    return SQL_SUCCESS;
}

void end_unit_test()
{
    /* 释放语句句柄 */
    if (NULL != h_stmt)
    {
        SQLFreeHandle(SQL_HANDLE_STMT, h_stmt);
    }

    /* 释放连接句柄 */
    if (NULL != h_conn)
    {
        SQLDisconnect(h_conn);
        SQLFreeHandle(SQL_HANDLE_DBC, h_conn);
    }

    /* 释放环境句柄 */
    if (NULL != h_env)
    {
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
    }

    return;
```



```
}

int main()
{
 // begin test
 if (begin_unit_test() != SQL_SUCCESS)
    {
        printf("\n begin_test_unit failed.");
        return SQL_ERROR;
    }
    // 句柄配置同前面用例
    int         i = 0;
    SQLCHAR*    sql_drop   = "drop table if exists test_bindnumber_001";
    SQLCHAR*    sql_create = "create table test_bindnumber_001("
                             "f4 number, f5 number(10, 2)"
                             ")";
    SQLCHAR*    sql_insert = "insert into test_bindnumber_001 values(?, ?)";
    SQLCHAR*    sql_select = "select * from test_bindnumber_001";
    SQLLEN      RowCount;
    SQL_NUMERIC_STRUCT st_number;
    SQLCHAR     getValue[2][MESSAGE_BUFFER_LEN];

    /* step 1. 建表 */
    RETURN_IF_NOT_SUCCESS(execute_cmd(sql_drop));
    RETURN_IF_NOT_SUCCESS(execute_cmd(sql_create));

    /* step 2.1 通过SQL_NUMERIC_STRUCT结构绑定参数 */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));

    //第一行: 1234.5678
    memset(st_number.val, 0, SQL_MAX_NUMERIC_LEN);
    st_number.precision = 8;
    st_number.scale = 4;
    st_number.sign = 1;
    st_number.val[0] = 0x4E;
    st_number.val[1] = 0x61;
    st_number.val[2] = 0xBC;

    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0, NULL));

    // 关闭自动提交
    SQLSetConnectAttr(h_conn, SQL_ATTR_AUTOCOMMIT, (SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0);

    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    //第二行: 12345678
    memset(st_number.val, 0, SQL_MAX_NUMERIC_LEN);
    st_number.precision = 8;
    st_number.scale = 0;
    st_number.sign = 1;
    st_number.val[0] = 0x4E;
    st_number.val[1] = 0x61;
    st_number.val[2] = 0xBC;

    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 0, &st_number, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 0, &st_number, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    //第三行: 12345678
    memset(st_number.val, 0, SQL_MAX_NUMERIC_LEN);
```



```
    st_number.precision = 0;
    st_number.scale = 4;
    st_number.sign = 1;
    st_number.val[0] = 0x4E;
    st_number.val[1] = 0x61;
    st_number.val[2] = 0xBC;

    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);


    /* step 2.2 第四行通过SQL_C_CHAR字符串绑定参数 */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    SQLCHAR*    szNumber = "1234.5678";
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_NUMERIC, strlen(szNumber), 0, szNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_NUMERIC, strlen(szNumber), 0, szNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* step 2.3 第五行通过SQL_C_FLOAT绑定参数 */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    SQLREAL    fNumber = 1234.5678;
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_FLOAT,
SQL_NUMERIC, sizeof(fNumber), 4, &fNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_FLOAT,
SQL_NUMERIC, sizeof(fNumber), 4, &fNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* step 2.4 第六行通过SQL_C_DOUBLE绑定参数 */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    SQLDOUBLE   dNumber = 1234.5678;
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_DOUBLE,
SQL_NUMERIC, sizeof(dNumber), 4, &dNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_DOUBLE,
SQL_NUMERIC, sizeof(dNumber), 4, &dNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    SQLBIGINT   bNumber1 = 0xFFFFFFFFFFFFFFFF;
    SQLBIGINT   bNumber2 = 12345;

    /* step 2.5 第七行通过SQL_C_SBIGINT绑定参数 */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_SBIGINT,
SQL_NUMERIC, sizeof(bNumber1), 4, &bNumber1, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_SBIGINT,
SQL_NUMERIC, sizeof(bNumber2), 4, &bNumber2, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* step 2.6 第八行通过SQL_C_UBIGINT绑定参数 */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_UBIGINT,
SQL_NUMERIC, sizeof(bNumber1), 4, &bNumber1, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_UBIGINT,
SQL_NUMERIC, sizeof(bNumber2), 4, &bNumber2, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
```



```
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    SQLLEN  lNumber1 = 0xFFFFFFFFFFFFFFFF;
    SQLLEN  lNumber2 = 12345;

    /* step 2.7 第九行通过SQL_C_LONG绑定参数 */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_NUMERIC, sizeof(lNumber1), 0, &lNumber1, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_NUMERIC, sizeof(lNumber2), 0, &lNumber2, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* step 2.8 第十行通过SQL_C_ULONG绑定参数 */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_ULONG,
SQL_NUMERIC, sizeof(lNumber1), 0, &lNumber1, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_ULONG,
SQL_NUMERIC, sizeof(lNumber2), 0, &lNumber2, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    SQLSMALLINT sNumber = 0xFFFF;

    /* step 2.9 第十一行通过SQL_C_SHORT绑定参数 */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_SHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_SHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* step 2.10 第十二行通过SQL_C_USHORT绑定参数 */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_USHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_USHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    SQLCHAR cNumber = 0xFF;

    /* step 2.11 第十三行通过SQL_C_TINYINT绑定参数 */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_TINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_TINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* step 2.12 第十四行通过SQL_C_UTINYINT绑定参数 */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_UTINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_UTINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);
```



```
    /* 用字符串类型统一进行期盼 */
    SQLCHAR*    expectValue[14][2] = {{"1234.5678",            "1234.57"},
                                      {"12345678",             "12345678"},
                                      {"0",                    "0"},
                                      {"1234.5678",            "1234.57"},
                                      {"1234.5677",            "1234.57"},
                                      {"1234.5678",            "1234.57"},
                                      {"-1",                   "12345"},
                                      {"18446744073709551615",  "12345"},
                                      {"-1",                   "12345"},
                                      {"4294967295",           "12345"},
                                      {"-1",                   "-1"},
                                      {"65535",                "65535"},
                                      {"-1",                   "-1"},
                                      {"255",                  "255"},
                                      };

    RETURN_IF_NOT_SUCCESS(execute_cmd(sql_select));
    while ( SQL_NO_DATA != SQLFetch(h_stmt))
    {
        RETURN_IF_NOT_SUCCESS_I(i, SQLGetData(h_stmt, 1, SQL_C_CHAR, &getValue[0],
MESSAGE_BUFFER_LEN, NULL));
        RETURN_IF_NOT_SUCCESS_I(i, SQLGetData(h_stmt, 2, SQL_C_CHAR, &getValue[1],
MESSAGE_BUFFER_LEN, NULL));

        //RETURN_IF_NOT_STRCMP_I(i, expectValue[i][0], getValue[0]);
        //RETURN_IF_NOT_STRCMP_I(i, expectValue[i][1], getValue[1]);
        i++;
    }

    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(i, RowCount);
 SQLCloseCursor(h_stmt);
    /* step final. 删除表还原环境 */
    RETURN_IF_NOT_SUCCESS(execute_cmd(sql_drop));

    end_unit_test();
}
```

说明

比如这个用例中定义了number列，调用SQLBindParameter接口时，绑定SQL_NUMERIC会比SQL_LONG性能高一些。因为如果是char，在数据库服务端插入数据时需要进行数据类型转换，从而引发性能瓶颈。

负载均衡场景

当应用程序有大并发场景时可开启负载均衡：

- 负载均衡即为将并发连接随机分发到所有CN上，避免单个CN负载过大，达到高性能的目的。
- 配置参数AutoBalance=1，开启负载均衡功能。
- 参数RefreshCNListTime=5可以选择性配置，默认刷新时间为10秒。
- 参数Priority=1可以选择性配置，意为并发连接优先发送到配置文件中配置的CN上，当配置的CN全部不可连接时，连接才会被分发到剩余CN上。

示例场景：

集群环境有6个CN，CN1，CN2，CN3，CN4，CN5和CN6；配置文件配置4个CN，为CN1，CN2，CN3和CN4。

配置文件示例：


```
[gaussdb]
Driver=GaussMPP
Servername=10.145.130.26,10.145.130.27,10.145.130.28,10.145.130.29（数据库Server IP）
Database=postgres （数据库名）
Username=omm （数据库用户名）
Password= （数据库用户密码）
Port=8000（数据库侦听端口）
Sslmode=allow
AutoBalance=1
RefreshCNListTime=3
Priority=1
```

当配置文件和集群环境如示例时，并发连接会随机、平均发送到CN1，CN2，CN3和CN4上，连接数均衡。当CN1，CN2，CN3和CN4全部不可用时，并发连接会随机、平均发送到CN5和CN6上。如果此时CN1，CN2，CN3和CN4中有CN重新可用时，连接则不会再发送到CN5和CN6上，而重新发送到重新可用的CN上。

5.4.7 ODBC 接口参考

ODBC接口是一套提供给用户的API函数，本节将对部分常用接口做具体描述，若涉及其他接口可参考msdn（网址：https://msdn.microsoft.com/en-us/library/windows/desktop/ms714177(v=vs.85).aspx）中ODBC Programmer's Reference项的相关内容。

5.4.7.1 SQLAllocEnv

在ODBC 3.x版本中，ODBC 2.x的函数SQLAllocEnv已被SQLAllocHandle代替。有关详细信息请参阅**SQLAllocHandle**。

5.4.7.2 SQLAllocConnect

在ODBC 3.x版本中，ODBC 2.x的函数SQLAllocConnect已被SQLAllocHandle代替。有关详细信息请参阅**SQLAllocHandle**。

5.4.7.3 SQLAllocHandle

功能描述

分配环境、连接、语句或描述符的句柄，它替代了ODBC 2.x函数SQLAllocEnv、SQLAllocConnect及SQLAllocStmt。

原型

```
SQLRETURN SQLAllocHandle(SQLSMALLINT   HandleType,
                         SQLHANDLE     InputHandle,
                         SQLHANDLE     *OutputHandlePtr);
```


参数

表 5-31 SQLAllocHandle 参数

关键字	参数说明
HandleType	由SQLAllocHandle分配的句柄类型。必须为下列值之一： • SQL_HANDLE_ENV（环境句柄） • SQL_HANDLE_DBC（连接句柄） • SQL_HANDLE_STMT（语句句柄） • SQL_HANDLE_DESC（描述句柄） 申请句柄顺序为，先申请环境句柄，再申请连接句柄，最后申请语句句柄，后申请的句柄都要依赖它前面申请的句柄。
InputHandle	将要分配的新句柄的类型。 • 如果HandleType为SQL_HANDLE_ENV，则这个值为SQL_NULL_HANDLE。 • 如果HandleType为SQL_HANDLE_DBC，则这一定是一个环境句柄。 • 如果HandleType为SQL_HANDLE_STMT或SQL_HANDLE_DESC，则它一定是一个连接句柄。
OutputHandlePtr	**输出参数**：一个缓冲区的指针，此缓冲区以新分配的数据结构存放返回的句柄。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。

注意事项

当分配的句柄并非环境句柄时，如果SQLAllocHandle返回的值为SQL_ERROR，则它会将OutputHandlePtr的值设置为SQL_NULL_HDBC、SQL_NULL_HSTMT或SQL_NULL_HDESC。之后，通过调用带有适当参数的**SQLGetDiagRec**，其中HandleType和Handle被设置为IntputHandle的值，可得到相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

参见：**示例**

5.4.7.4 SQLAllocStmt

在ODBC 3.x版本中，ODBC 2.x的函数SQLAllocStmt已被SQLAllocHandle代替。有关详细信息请参阅**SQLAllocHandle**。

5.4.7.5 SQLBindCol

功能描述

将应用程序数据缓冲区绑定到结果集的列中。

原型

```
SQLRETURN SQLBindCol(SQLHSTMT       StatementHandle,
                     SQLUSMALLINT   ColumnNumber,
                     SQLSMALLINT    TargetType,
                     SQLPOINTER     TargetValuePtr,
                     SQLINTEGER     BufferLength,
                     SQLINTEGER     *StrLen_or_IndPtr);
```

参数

表 5-32 SQLBindCol 参数

关键字	参数说明
StatementHandle	语句句柄。
ColumnNumber	要绑定结果集的列号。起始列号为0，以递增的顺序计算列号，第0列是书签列。若未设置书签页，则起始列号为1。
TargetType	缓冲区中C数据类型的标识符。
TargetValuePtr	**输出参数**：指向与列绑定的数据缓冲区的指针。SQLFetch函数返回这个缓冲区中的数据。如果此参数为一个空指针，则StrLen_or_IndPtr是一个有效值。
BufferLength	TargetValuePtr指向缓冲区的长度，以字节为单位。
StrLen_or_IndPtr	**输出参数**：缓冲区的长度或指示器指针。若为空值，则未使用任何长度或指示器值。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。

注意事项

当SQLBindCol返回SQL_ERROR或SQL_SUCCESS_WITH_INFO时，通过调用**SQLGetDiagRec**函数，并将HandleType和Handle参数设置为SQL_HANDLE_STMT和StatementHandle，可得到一个相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

参见：**示例**

5.4.7.6 SQLBindParameter

功能描述

将一条SQL语句中的一个参数标志和一个缓冲区绑定起来。

原型

```
SQLRETURN SQLBindParameter(SQLHSTMT       StatementHandle,
                           SQLUSMALLINT   ParameterNumber,
                           SQLSMALLINT    InputOutputType,
                           SQLSMALLINT    ValuetType,
                           SQLSMALLINT    ParameterType,
                           SQLSMALLINT    ColumnSize,
                           SQLSMALLINT    DecimalDigits,
                           SQLPOINTER     ParameterValuePtr,
                           SQLINTEGER     BufferLength,
                           SQLINTEGER     *StrLen_or_IndPtr);
```

参数

表 5-33 SQLBindParameter

关键词	参数说明
StatementHandle	语句句柄。
ParameterNumber	参数序号，起始为1，依次递增。
InputOutputType	输入输出参数类型。
ValueType	参数的C数据类型。
ParameterType	参数的SQL数据类型。
ColumnSize	列的大小或相应参数标记的表达式。
DecimalDigits	列的十进制数字或相应参数标记的表达式。
ParameterValuePtr	指向存储参数数据缓冲区的指针。
BufferLength	ParameterValuePtr指向缓冲区的长度，以字节为单位。
StrLen_or_IndPtr	缓冲区的长度或指示器指针。若为空值，则未使用任何长度或指示器值。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。

- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。

注意事项

当SQLBindParameter返回SQL_ERROR或SQL_SUCCESS_WITH_INFO时，通过调用SQLGetDiagRec函数，并将HandleType和Handle参数设置为SQL_HANDLE_STMT和StatementHandle，可得到一个相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

参见：示例

5.4.7.7 SQLColAttribute

功能描述

返回结果集中一列的描述符信息。

原型

```
SQLRETURN SQLColAttibute(SQLHSTMT        StatementHandle,
                         SQLUSMALLINT    ColumnNumber,
                         SQLUSMALLINT    FieldIdentifier,
                         SQLPOINTER      CharacterAtrributePtr,
                         SQLSMALLINT     BufferLength,
                         SQLSMALLINT     *StringLengthPtr,
                         SQLPOINTE       NumericAttributePtr);
```

参数

表 5-34 SQLColAttribute 参数

关键字	参数说明
StatementHandle	语句句柄。
ColumnNumber	要检索字段的列号，起始为1，依次递增。
FieldIdentifier	IRD中ColumnNumber行的字段。
CharacterAttributePtr	**输出参数**：一个缓冲区指针，返回FieldIdentifier字段值。
BufferLength	• 如果FieldIdentifier是一个ODBC定义的字段，而且CharacterAttributePtr指向一个字符串或二进制缓冲区，则此参数为该缓冲区的长度。 • 如果FieldIdentifier是一个ODBC定义的字段，而且CharacterAttributePtr指向一个整数，则会忽略该字段。
StringLengthPtr	**输出参数**：缓冲区指针，存放*CharacterAttributePtr中字符类型数据的字节总数，对于非字符类型，忽略BufferLength的值。

关键字	参数说明
NumericAttributePtr	**输出参数**：指向一个整型缓冲区的指针，返回IRD中ColumnNumber行FieldIdentifier字段的值。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。

注意事项

当SQLColAttribute返回SQL_ERROR或SQL_SUCCESS_WITH_INFO时，通过调用**SQLGetDiagRec**函数，并将HandleType和Handle参数设置为SQL_HANDLE_STMT和StatementHandle，可得到一个相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

参见：**示例**

5.4.7.8 SQLConnect

功能描述

在驱动程序和数据源之间建立连接。连接上数据源之后，可以通过连接句柄访问到所有有关连接数据源的信息，包括程序运行状态、事务处理状态和错误信息。

原型

```
SQLRETURN  SQLConnect(SQLHDBC       ConnectionHandle,
                      SQLCHAR       *ServerName,
                      SQLSMALLINT   NameLength1,
                      SQLCHAR       *UserName,
                      SQLSMALLINT   NameLength2,
                      SQLCHAR       *Authentication,
                      SQLSMALLINT   NameLength3);
```

参数

表 5-35 SQLConnect 参数

关键字	参数说明
ConnectionHandle	连接句柄，通过SQLAllocHandle获得。
ServerName	要连接数据源的名称。

关键字	参数说明
NameLength1	ServerName的长度。
UserName	数据源中数据库用户名。
NameLength2	UserName的长度。
Authentication	数据源中数据库用户密码。
NameLength3	Authentication的长度。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。
- SQL_STILL_EXECUTING：表示语句正在执行。

注意事项

当调用SQLConnect函数返回SQL_ERROR或SQL_SUCCESS_WITH_INFO时，通过调用**SQLGetDiagRec**函数，并将HandleType和Handle参数设置为SQL_HANDLE_DBC和ConnectionHandle，可得到一个相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

参见：**示例**

5.4.7.9 SQLDisconnect

功能描述

关闭一个与特定连接句柄相关的连接。

原型

```
SQLRETURN SQLDisconnect(SQLHDBC    ConnectionHandle);
```

参数

表 5-36 SQLDisconnect 参数

关键字	参数说明
ConnectionHandle	连接句柄，通过SQLAllocHandle获得。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。

注意事项

当调用SQLDisconnect函数返回SQL_ERROR或SQL_SUCCESS_WITH_INFO时，通过调用**SQLGetDiagRec**函数，并将HandleType和Handle参数设置为SQL_HANDLE_DBC和ConnectionHandle，可得到一个相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

参见：**示例**

5.4.7.10 SQLExecDirect

功能描述

使用参数的当前值，执行一条准备好的语句。对于一次只执行一条SQL语句，SQLExecDirect是最快的执行方式。

原型

```
SQLRETURN SQLExecDirect(SQLHSTMT         StatementHandle,
                        SQLCHAR          *StatementText,
                        SQLINTEGER       TextLength);
```

参数

表 5-37 SQLExecDirect 参数

关键字	参数说明
StatementHandle	语句句柄，通过SQLAllocHandle获得。
StatementText	要执行的SQL语句，不支持一次执行多条语句。
TextLength	StatementText的长度。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_NEED_DATA：在执行SQL语句前没有提供足够的参数。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。

- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。
- SQL_STILL_EXECUTING：表示语句正在执行。
- SQL_NO_DATA：表示SQL语句不返回结果集。

注意事项

当调用SQLExecDirect函数返回SQL_ERROR或SQL_SUCCESS_WITH_INFO时，通过调用**SQLGetDiagRec**函数，并将HandleType和Handle参数设置为SQL_HANDLE_STMT和StatementHandle，可得到一个相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

参见：**示例**

5.4.7.11 SQLExecute

功能描述

如果语句中存在参数标记的话，SQLExecute函数使用参数标记参数的当前值，执行一条准备好的SQL语句。

原型

```
SQLRETURN SQLExecute(SQLHSTMT    StatementHandle);
```

参数

表 5-38 SQLExecute 参数

关键字	参数说明
StatementHandle	要执行语句的语句句柄。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_NEED_DATA：表示在执行SQL语句前没有提供足够的参数。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_NO_DATA：表示SQL语句不返回结果集。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。
- SQL_STILL_EXECUTING：表示语句正在执行。

注意事项

当SQLExecute函数返回SQL_ERROR或SQL_SUCCESS_WITH_INFO时，可通过调用**SQLGetDiagRec**函数，并将HandleType和Handle参数设置为SQL_HANDLE_STMT和

StatementHandle，可得到一个相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

参见：**示例**

5.4.7.12 SQLFetch

功能描述

从结果集中取下一个行集的数据，并返回所有被绑定列的数据。

原型

```
SQLRETURN SQLFetch(SQLHSTMT    StatementHandle);
```

参数

表 5-39 SQLFetch 参数

关键字	参数说明
StatementHandle	语句句柄，通过SQLAllocHandle获得。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_NO_DATA：表示SQL语句不返回结果集。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。
- SQL_STILL_EXECUTING：表示语句正在执行。

注意事项

当调用SQLFetch函数返回SQL_ERROR或SQL_SUCCESS_WITH_INFO时，通过调用**SQLGetDiagRec**函数，并将HandleType和Handle参数设置为SQL_HANDLE_STMT和StatementHandle，可得到一个相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

参见：**示例**

5.4.7.13 SQLFreeStmt

在ODBC 3.x版本中，ODBC 2.x的函数SQLFreeStmt已被SQLFreeHandle代替。有关详细信息请参阅**SQLFreeHandle**。

5.4.7.14 SQLFreeConnect

在ODBC 3.x版本中，ODBC 2.x的函数SQLFreeConnect已被SQLFreeHandle代替。有关详细信息请参阅**SQLFreeHandle**。

5.4.7.15 SQLFreeHandle

功能描述

释放与指定环境、连接、语句或描述符相关联的资源，它替代了ODBC 2.x函数SQLFreeEnv、SQLFreeConnect及SQLFreeStmt。

原型

```
SQLRETURN SQLFreeHandle(SQLSMALLINT   HandleType,
                        SQLHANDLE     Handle);
```

参数

表 5-40 SQLFreeHandle 参数

关键字	参数说明
HandleType	SQLFreeHandle要释放的句柄类型。必须为下列值之一： • SQL_HANDLE_ENV • SQL_HANDLE_DBC • SQL_HANDLE_STMT • SQL_HANDLE_DESC 如果HandleType不是这些值之一，SQLFreeHandle返回SQL_INVALID_HANDLE。
Handle	要释放的句柄。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。

注意事项

如果SQLFreeHandle返回SQL_ERROR，句柄仍然有效。

示例

参见：**示例**

5.4.7.16 SQLFreeEnv

在ODBC 3.x版本中，ODBC 2.x的函数SQLFreeEnv已被SQLFreeHandle代替。有关详细信息请参阅**SQLFreeHandle**。

5.4.7.17 SQLPrepare

功能描述

准备一个将要进行的SQL语句。

需要注意的是，ODBC发送准备好的语句不支持内核复用计划，会导致每次执行都需要重新生成计划，导致CPU占用率高。如果业务对计划复用有需求建议优先使用JDBC作为客户端。

原型

```
SQLRETURN SQLPrepare(SQLHSTMT      StatementHandle,
                     SQLCHAR       *StatementText,
                     SQLINTEGER    TextLength);
```

参数

表 5-41 SQLPrepare 参数

关键字	参数说明
StatementHandle	语句句柄。
StatementText	SQL文本串。
TextLength	StatementText的长度。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。
- SQL_STILL_EXECUTING：表示语句正在执行。

注意事项

当SQLPrepare返回的值为SQL_ERROR或SQL_SUCCESS_WITH_INFO时，通过调用**SQLGetDiagRec**函数，并将HandleType和Handle参数分别设置为SQL_HANDLE_STMT和StatementHandle，可得到一个相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

参见：**示例**

5.4.7.18 SQLGetData

功能描述

SQLGetData返回结果集中某一列的数据。可以多次调用它来部分地检索不定长度的数据。

原型

```
SQLRETURN SQLGetData(SQLHSTMT        StatementHandle,
                     SQLUSMALLINT    Col_or_Param_Num,
                     SQLSMALLINT     TargetType,
                     SQLPOINTER      TargetValuePtr,
                     SQLLEN          BufferLength,
                     SQLLEN          *StrLen_or_IndPtr);
```

参数

表 5-42 SQLGetData 参数

关键字	参数说明
StatementHandle	语句句柄，通过SQLAllocHandle获得。
Col_or_Param_Num	要返回数据的列号。结果集的列按增序从1开始编号。书签列的列号为0。
TargetType	TargetValuePtr缓冲中的C数据类型的类型标识符。若TargetType为SQL_ARD_TYPE，驱动使用ARD中SQL_DESC_CONCISE_TYPE字段的类型标识符。若为SQL_C_DEFAULT，驱动根据源的SQL数据类型选择缺省的数据类型。
TargetValuePtr	**输出参数**：指向返回数据所在缓冲区的指针。
BufferLength	TargetValuePtr所指向缓冲区的长度。
StrLen_or_IndPtr	**输出参数**：指向缓冲区的指针，在此缓冲区中返回长度或标识符的值。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_NO_DATA：表示SQL语句不返回结果集。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。
- SQL_STILL_EXECUTING：表示语句正在执行。

注意事项

当调用SQLGetData函数返回SQL_ERROR或SQL_SUCCESS_WITH_INFO时，通过调用**SQLGetDiagRec**函数，并将HandleType和Handle参数分别设置为

SQL_HANDLE_STMT和StatementHandle，可得到一个相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

参见：**示例**

5.4.7.19 SQLGetDiagRec

功能描述

返回诊断记录的多个字段的当前值，其中诊断记录包含错误、警告及状态信息。

原型

```
SQLRETURN  SQLGetDiagRec(SQLSMALLINT    HandleType
                         SQLHANDLE      Handle,
                         SQLSMALLINT    RecNumber,
                         SQLCHAR        *SQLState,
                         SQLINTEGER     *NativeErrorPtr,
                         SQLCHAR        *MessageText,
                         SQLSMALLINT    BufferLength
                         SQLSMALLINT    *TextLengthPtr);
```

参数

表 5-43 SQLGetDiagRec 参数

关键字	参数说明
HandleType	句柄类型标识符，它说明诊断所要求的句柄类型。必须为下列值之一： • SQL_HANDLE_ENV • SQL_HANDLE_DBC • SQL_HANDLE_STMT • SQL_HANDLE_DESC
Handle	诊断数据结构的句柄，其类型由HandleType来指出。如果HandleType是SQL_HANDLE_ENV，Handle可以是共享的或非共享的环境句柄。
RecNumber	指出应用从查找信息的状态记录。状态记录从1开始编号。
SQLState	**输出参数**：指向缓冲区的指针，该缓冲区存储着有关RecNumber的五字符的SQLSTATE码。
NativeErrorPtr	**输出参数**：指向缓冲区的指针，该缓冲区存储着本地的错误码。
MessageText	指向缓冲区的指针，该缓冲区存储着诊断信息文本串。
BufferLength	MessageText的长度。

关键字	参数说明
TextLengthPtr	**输出参数**：指向缓冲区的指针，返回MessageText中的字节总数。如果返回字节数大于BufferLength，则MessageText中的诊断信息文本被截断成BufferLength减去NULL结尾字符的长度。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。

注意事项

SQLGetDiagRec不发布自己的诊断记录。它用下列返回值来报告它自己的执行结果：

- SQL_SUCCESS：函数成功返回诊断信息。
- SQL_SUCCESS_WITH_INFO：*MessageText太小以致不能容纳所请求的诊断信息。没有诊断记录生成。
- SQL_INVALID_HANDLE：由HandType和Handle所指出的句柄是不合法句柄。
- SQL_ERROR：RecNumber小于等于0或BufferLength小于0。

如果调用ODBC函数返回SQL_ERROR或SQL_SUCCESS_WITH_INFO，可调用SQLGetDiagRec返回诊断信息值SQLSTATE，SQLSTATE值的如下表。

表 5-44 SQLSTATE 值

SQLSATATE	错误	描述
HY000	一般错误	未定义特定的SQLSTATE所产生的一个错误。
HY001	内存分配错误	驱动程序不能分配所需要的内存来支持函数的执行或完成。
HY008	取消操作	调用SQLCancel取消执行语句后，依然在StatementHandle上调用函数。
HY010	函数系列错误	在为执行中的所有数据参数或列发送数据前就调用了执行函数。
HY013	内存管理错误	不能处理函数调用，可能由当前内存条件差引起。
HYT01	连接超时	数据源响应请求之前，连接超时。
IM001	驱动程序不支持此函数	调用了StatementHandle相关的驱动程序不支持的函数

示例

参见：示例

5.4.7.20 SQLSetConnectAttr

功能描述

设置控制连接各方面的属性。

原型

```
SQLRETURN SQLSetConnectAttr(SQLHDBC      ConnectionHandle
                        SQLINTEGER   Attribute,
                        SQLPOINTER   ValuePtr,
                        SQLINTEGER   StringLength);
```

参数

表 5-45 SQLSetConnectAttr 参数

关键字	参数说明
ConnectionHandle	连接句柄。
Attribute	设置属性。
ValuePtr	指向对应Attribute的值。依赖于Attribute的值，ValuePtr是32位无符号整型值或指向以空结束的字符串。注意，如果ValuePtr参数是驱动程序指定值。ValuePtr可能是有符号的整数。
StringLength	如果ValuePtr指向字符串或二进制缓冲区，这个参数是*ValuePtr长度，如果ValuePtr指向整型，忽略StringLength。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。

注意事项

当SQLSetConnectAttr的返回值为SQL_ERROR或SQL_SUCCESS_WITH_INFO时，通过借助SQL_HANDLE_DBC的HandleType和ConnectionHandle的Handle，调用SQLGetDiagRec可得到相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

参见：示例

5.4.7.21 SQLSetEnvAttr

功能描述

设置控制环境各方面的属性。

原型

```
SQLRETURN SQLSetEnvAttr(SQLHENV      EnvironmentHandle
                        SQLINTEGER   Attribute,
                        SQLPOINTER   ValuePtr,
                        SQLINTEGER   StringLength);
```

参数

表 5-46 SQLSetEnvAttr 参数

关键字	参数说明
EnvironmentHandle	环境句柄。
Attribute	需设置的环境属性，可为如下值： • SQL_ATTR_ODBC_VERSION：指定ODBC版本。 • SQL_CONNECTION_POOLING：连接池属性。 • SQL_OUTPUT_NTS：指明驱动器返回字符串的形式。
ValuePtr	指向对应Attribute的值。依赖于Attribute的值，ValuePtr可能是32位整型值，或为以空结束的字符串。
StringLength	如果ValuePtr指向字符串或二进制缓冲区，这个参数是*ValuePtr长度，如果ValuePtr指向整型，忽略StringLength。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。

注意事项

当SQLSetEnvAttr的返回值为SQL_ERROR或SQL_SUCCESS_WITH_INFO时，通过借助SQL_HANDLE_ENV的HandleType和EnvironmentHandle的Handle，调用**SQLGetDiagRec**可得到相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

参见：**示例**

5.4.7.22 SQLSetStmtAttr

功能描述

设置相关语句的属性。

原型

```
SQLRETURN SQLSetStmtAttr(SQLHSTMT      StatementHandle
                         SQLINTEGER    Attribute,
                         SQLPOINTER    ValuePtr,
                         SQLINTEGER    StringLength);
```

参数

表 5-47 SQLSetStmtAttr 参数

关键字	参数说明
StatementHandle	语句句柄。
Attribute	需设置的属性。
ValuePtr	指向对应Attribute的值。依赖于Attribute的值，ValuePtr可能是32位无符号整型值，或指向以空结束的字符串，二进制缓冲区，或者驱动定义值。注意，如果ValuePtr参数是驱动程序指定值。ValuePtr可能是有符号的整数。
StringLength	如果ValuePtr指向字符串或二进制缓冲区，这个参数是*ValuePtr长度，如果ValuePtr指向整型，忽略StringLength。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他API的返回值同理。

注意事项

当SQLSetStmtAttr的返回值为SQL_ERROR或SQL_SUCCESS_WITH_INFO时，通过借助SQL_HANDLE_STMT的HandleType和StatementHandle的Handle，调用**SQLGetDiagRec**可得到相关的SQLSTATE值，通过SQLSTATE值可以查出调用此函数的具体信息。

示例

参见：**示例**

5.5 基于 libpq 开发

libpq是GaussDB C应用程序接口。libpq是一套允许客户程序向GaussDB服务器服务进程发送查询并且获得查询返回的库函数。同时也是其他几个GaussDB应用接口下面的引擎，如ODBC等依赖的库文件。本章给出了两个示例显示如何利用libpq编写代码。

5.5.1 libpq 包及依赖的库和头文件

从发布包中获取，包名为GaussDB-Kernel-VxxxRxxxCxx-xxxxx-64bit-Libpq.tar.gz。使用libpq的前端程序必须包括头文件libpq-fe.h并且必须与libpq库连接。

5.5.2 开发流程

编译并且连接一个libpq的源程序，需要做下面的一些事情：

- 解压GaussDB-Kernel-VxxxRxxxCxx-xxxxx-64bit-Libpq.tar.gz文件，其中include文件夹下的头文件为所需的头文件，lib文件夹中为所需的libpq库文件。

  说明

  除libpq-fe.h外，include文件夹下默认还存在头文件postgres_ext.h，gs_thread.h，gs_threadlocal.h，这三个头文件是libpq-fe.h的依赖文件。

- 包含libpq-fe.h头文件：
  ```
  #include <libpq-fe.h>
  ```
- 通过-I *directory*选项，提供头文件的安装位置（有些时候编译器会查找缺省的目录，因此可以忽略这些选项）。如：
  ```
  gcc -I (头文件所在目录) -L (libpq库所在目录) testprog.c -lpq
  ```
- 如果要使用制作文件(makefile)，向CPPFLAGS、LDFLAGS、LIBS变量中增加如下选项：
  ```
  CPPFLAGS += -I (头文件所在目录)
  LDFLAGS += -L (libpq库所在目录)
  LIBS += -lpq
  ```

5.5.3 示例

常用功能示例代码

示例1：

```
/*
 * testlibpq.c
 */
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>
#include <string.h>

static void
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

int
main(int argc, char **argv)
{
```

```
    /* 此处user、passwd等变量应从环境变量或配置文件读取，环境变量需用户自己按需配置；非环境变量情况下
可直接赋值字符串 */
    const char *conninfo;
    PGconn     *conn;
    PGresult   *res;
    int         nFields;
    int         i,j;
    char        *passwd = getenv("EXAMPLE_PASSWD_ENV");
    char        *port = getenv("EXAMPLE_PORT_ENV");
    char        *host = getenv("EXAMPLE_HOST_ENV");
    char        *username = getenv("EXAMPLE_USERNAME_ENV");
    char        *dbname = getenv("EXAMPLE_DBNAME_ENV");

    /*
     * 用户在命令行上提供了conninfo字符串的值时使用该值；
     * 否则环境变量或者所有其它连接参数
     * 都使用缺省值。
     */
    if (argc > 1)
        conninfo = argv[1];
    else
        sprintf(conninfo,
            "dbname=%s port=%s host=%s application=test connect_timeout=5 sslmode=allow user=%s
password=%s",
            dbname, port, host, username, password);

    /* 连接数据库 */
    conn = PQconnectdb(conninfo);

    /* 检查后端连接成功建立 */
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s",
                PQerrorMessage(conn));
        exit_nicely(conn);
    }

    /*
     * 测试实例涉及游标的使用时候必须使用事务块。
     *把全部放在一个  "select * from pg_database"
     * PQexec() 里，过于简单，不推荐使用。
     */

    /* 开始一个事务块 */
    res = PQexec(conn, "BEGIN");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "BEGIN command failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    /*
     * 在结果不需要的时候PQclear PGresult，以避免内存泄漏
     */
    PQclear(res);

    /*
     * 从系统表 pg_database（数据库的系统目录）里抓取数据
     */
    res = PQexec(conn, "DECLARE myportal CURSOR FOR select * from pg_database");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "DECLARE CURSOR failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }
    PQclear(res);
```



```
    res = PQexec(conn, "FETCH ALL in myportal");
    if (PQresultStatus(res) != PGRES_TUPLES_OK)
    {
        fprintf(stderr, "FETCH ALL failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    /* 打印属性名称 */
    nFields = PQnfields(res);
    for (i = 0; i < nFields; i++)
        printf("%-15s", PQfname(res, i));
    printf("\n\n");

    /* 打印行 */
    for (i = 0; i < PQntuples(res); i++)
    {
        for (j = 0; j < nFields; j++)
            printf("%-15s", PQgetvalue(res, i, j));
        printf("\n");
    }

    PQclear(res);

    /* 关闭入口 ... 不用检查错误 ... */
    res = PQexec(conn, "CLOSE myportal");
    PQclear(res);

    /* 结束事务 */
    res = PQexec(conn, "END");
    PQclear(res);

    /* 关闭数据库连接并清理 */
    PQfinish(conn);

    return 0;
}
```

示例2：

```
/*
 * testlibpq3.c 测试PQprepare
 * /
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>
#include <string.h>
int main(int argc, char * argv[])
{
    /* 此处user、passwd等变量应从环境变量或配置文件读取，环境变量需用户自己按需配置；非环境变量情况下
可直接赋值字符串 */
    PGconn *conn;
    PGresult * res;
    ConnStatusType pgstatus;
    char connstr[1024];
    char cmd_sql[2048];
    int nParams = 0;
    int paramLengths[5];
    int paramFormats[5];
    Oid paramTypes[5];
    char * paramValues[5];
    int i, cnt;
    char cid[32];
    int k;
    char *passwd = getenv("EXAMPLE_PASSWD_ENV");
    char *port = getenv("EXAMPLE_PORT_ENV");
    char *hostaddr = getenv("EXAMPLE_HOST_ENV");
    char *username = getenv("EXAMPLE_USERNAME_ENV");
    char *dbname = getenv("EXAMPLE_DBNAME_ENV");
    sprintf(connstr,
```



```
            "hostaddr=%s dbname=%s port=%s user=%s password=%s",
             hostaddr, dbname, port, username, paswswd);
    conn = PQconnectdb(connstr);
    pgstatus = PQstatus(conn);
    if (pgstatus == CONNECTION_OK)
    {
        printf("Connect database success!\n");
    }
    else
    {
        printf("Connect database fail:%s\n",PQerrorMessage(conn));
        return -1;
    }
    sprintf(cmd_sql, "SELECT b FROM t01 WHERE a = $1");
    paramTypes[0] = 23;
    res = PQprepare(conn,
                    "pre_name",
                    cmd_sql,
                    1,
                    paramTypes);
   if( PQresultStatus(res) != PGRES_COMMAND_OK )
    {
        printf("Failed to prepare SQL : %s\n: %s\n",cmd_sql, PQerrorMessage(conn));
        PQfinish(conn);
        return -1;
    }
    PQclear(res);
    paramValues[0] = cid;
    for (k=0; k<2; k++)
    {
        sprintf(cid, "%d", 1);
        paramLengths[0] = 6;
        paramFormats[0] = 0;
        res = PQexecPrepared(conn,
                             "pre_name",
                             1,
                             paramValues,
                             paramLengths,
                             paramFormats,
                             0);
        if( (PQresultStatus(res) != PGRES_COMMAND_OK ) && (PQresultStatus(res) != PGRES_TUPLES_OK))
        {
            printf("%s\n",PQerrorMessage(conn));
            PQclear(res);
            PQfinish(conn);
            return -1;
        }
        cnt = PQntuples(res);
        printf("return %d rows\n", cnt);
        for (i=0; i<cnt; i++)
        {
            printf("row %d: %s\n", i, PQgetvalue(res, i, 0));
        }
        PQclear(res);
    }
    PQfinish(conn);
    return 0;
}
```

示例3：

```
/*
 * testlibpq3.c
 *      测试外联参数和二进制I/O。
 *
 * 在运行这个例子之前，用下面的命令填充一个数据库
 *
 *
 * CREATE TABLE test1 (i int4, t text);
 *
```



```
 * INSERT INTO test1 values (2, 'ho there');
 *
 * 期望的输出是：
 *
 *
 * tuple 0: got
 *  i = (4 bytes) 2
 *  t = (8 bytes) 'ho there'
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <libpq-fe.h>

/* for ntohl/htonl */
#include <netinet/in.h>
#include <arpa/inet.h>

static void
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

/*
 * 这个函数打印查询结果，这些结果是二进制格式，从上面的
 * 注释里面创建的表中抓取出来的。
 */
static void
show_binary_results(PGresult *res)
{
    int         i;
    int         i_fnum,
                t_fnum;

    /* 使用 PQfnumber 来避免对结果中的字段顺序进行假设 */
    i_fnum = PQfnumber(res, "i");
    t_fnum = PQfnumber(res, "t");

    for (i = 0; i < PQntuples(res); i++)
    {
        char       *iptr;
        char       *tptr;
        int         ival;

        /* 获取字段值（忽略可能为空的可能） */
        iptr = PQgetvalue(res, i, i_fnum);
        tptr = PQgetvalue(res, i, t_fnum);

        /*
         * INT4 的二进制表现形式是网络字节序，
         * 建议转换成本地字节序。
         */
        ival = ntohl(*((uint32_t *) iptr));

        /*
         * TEXT 的二进制表现形式是文本，因此libpq能够给它附加一个字节零，
         * 把它看做 C 字符串。
         *
         */

        printf("tuple %d: got\n", i);
        printf(" i = (%d bytes) %d\n",
               PQgetlength(res, i, i_fnum), ival);
        printf(" t = (%d bytes) '%s'\n",
               PQgetlength(res, i, t_fnum), tptr);
```



```
        printf("\n\n");
    }
}

int
main(int argc, char **argv)
{
    /* 此处user、passwd等变量应从环境变量或配置文件读取，环境变量需用户自己按需配置；非环境变量情况下
可直接赋值字符串 */
    const char *conninfo;
    PGconn     *conn;
    PGresult   *res;
    const char *paramValues[1];
    int         paramLengths[1];
    int         paramFormats[1];
    uint32_t    binaryIntVal;
    char        *passwd = getenv("EXAMPLE_PASSWD_ENV");
    char        *port = getenv("EXAMPLE_PORT_ENV");
    char        *hostaddr = getenv("EXAMPLE_HOST_ENV");
    char        *username = getenv("EXAMPLE_USERNAME_ENV");
    char        *dbname = getenv("EXAMPLE_DBNAME_ENV");

    /*
     * 如果用户在命令行上提供了参数，
     * 那么使用该值为conninfo 字符串；否则
     * 使用环境变量或者缺省值。
     */
    if (argc > 1)
        conninfo = argv[1];
    else
        sprintf(conninfo,
            "dbname=%s port=%s host=%s application=test connect_timeout=5 sslmode=allow user=%s
password=%s",
            dbname, port, hostaddr, username, password);

    /* 和数据库建立连接 */
    conn = PQconnectdb(conninfo);

    /* 检查与服务器的连接是否成功建立 */
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s",
                PQerrorMessage(conn));
        exit_nicely(conn);
    }

    /* 把整数值 "2" 转换成网络字节序 */
    binaryIntVal = htonl((uint32_t) 2);

    /* 为 PQexecParams 设置参数数组 */
    paramValues[0] = (char *) &binaryIntVal;
    paramLengths[0] = sizeof(binaryIntVal);
    paramFormats[0] = 1;        /* 二进制 */

    res = PQexecParams(conn,
                       "SELECT * FROM test1 WHERE i = $1::int4",
                       1,       /* 一个参数 */
                       NULL,    /* 让后端推导参数类型 */
                       paramValues,
                       paramLengths,
                       paramFormats,
                       1);      /* 要求二进制结果 */

    if (PQresultStatus(res) != PGRES_TUPLES_OK)
    {
        fprintf(stderr, "SELECT failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }
```



```
    show_binary_results(res);

    PQclear(res);

    /* 关闭与数据库的连接并清理 */
    PQfinish(conn);

    return 0;
}
```

5.5.4 libpq 接口参考

5.5.4.1 数据库连接控制函数

数据库连接控制函数控制与GaussDB服务器的连接。一个应用程序一次可以与多个服务器建立连接，如一个客户端连接多个数据库的场景。每个连接都是用一个从函数PQconnectdb、PQconnectdbParams或PQsetdbLogin获得的PGconn对象表示。注意，这些函数总是返回一个非空的对象指针，除非内存分配失败，会返回一个空的指针。连接建立的接口保存在PGconn对象中，可以调用PQstatus函数来检查返回值看看连接是否成功。

5.5.4.1.1 PQconnectdbParams

功能描述

与数据库服务器建立一个新的连接。

原型

```
PGconn *PQconnectdbParams(const char * const *keywords,
                          const char * const *values,
                          int expand_dbname);
```

参数

表 5-48 PQconnectdbParams 参数

关键字	参数说明
keywords	定义为一个字符串的数组，每个都成为一个关键字。
values	给每个关键字一个值。
expand_dbname	当expand_dbname是非零的时，允许将dbname的关键字值看做一个连接字符串。只有第一个出现的dbname是这样展开的，任何随后的dbname值作为纯数据库名处理。

返回值

PGconn *：指向包含链接的对象指针，内存在函数内部申请。

注意事项

该函数用从两个NULL结束的数组中来的参数打开一个新的数据库连接。与PQsetdbLogin不同的是，可以不必更换函数签名（名字）就可以扩展参数集，所以建议应用程序中使用这个函数（或者它的类似的非阻塞变种PQconnectStartParams和PQconnectPoll）。

示例

请参见**示例**章节。

5.5.4.1.2 PQconnectdb

功能描述

与数据库服务器建立一个新的连接。

原型

```
PGconn *PQconnectdb(const char *conninfo);
```

参数

表 5-49 PQconnectdb 参数

关键字	参数说明
conninfo	连接字符串，字符串中的字段见**5.6.4.5章节描述**章节。

返回值

PGconn *：指向包含链接的对象指针，内存在函数内部申请。

注意事项

- 这个函数用从一个字符串conninfo来的参数与数据库打开一个新的连接。
- 传入的参数可以为空，表明使用所有缺省的参数，或者可以包含一个或更多个用空白间隔的参数设置，或者它可以包含一个URL。

示例

请参见**示例**章节。

5.5.4.1.3 PQbackendPID

补充解释

PQbackendPID函数返回值在GaussDB中表示后台线程的槽位ID (SlotID)，而并非后台线程的BackendPid。由于存在上述差异，不建议按照PostgreSQL同名函数的语义执行。若希望获取该连接的后台PID，可以通过执行系统函数pg_backend_pid获取。同时，依赖libpq的其他驱动程序的同名接口（如Python连接驱动psycopg2的get_backend_pid函数）也遵循上述规则。

5.5.4.1.4 PQsetdbLogin

功能描述

与数据库服务器建立一个新的连接。

原型

```
PGconn *PQsetdbLogin(const char *pghost,
                     const char *pgport,
                     const char *pgoptions,
                     const char *pgtty,
                     const char *dbName,
                     const char *login,
                     const char *pwd);
```

参数

表 5-50 PQsetdbLogin 参数

关键字	参数说明
pghost	要连接的主机名，详见**连接参数**章节描述的host字段。
pgport	主机服务器的端口号，详见**连接参数**描述的port字段。
pgoptions	添加命令行选项以在运行时发送到服务器，详见**连接参数**描述的options字段。
pgtty	忽略（以前，这个选项声明服务器日志的输出方向）
dbName	要连接的数据库名，详见**连接参数**描述的dbname字段。
login	要连接的用户名，详见**连接参数**章节描述的user字段。
pwd	如果服务器要求口令认证，所用的口令，详见**连接参数**描述的password字段。

返回值

PGconn *：指向包含链接的对象指针，内存在函数内部申请。

注意事项

- 该函数为PQconnectdb前身，参数个数固定，未定义参数被调用时使用缺省值，若需要给固定参数设置缺省值，则可赋值NULL或者空字符串。
- 若dbName中包含“=”或链接URL的有效前缀，则该dbName被看做一个conninfo字符串并传递至PQconnectdb中，其余参数与PQconnectdbParams保持一致。

示例

请参见**示例**章节。

5.5.4.1.5 PQfinish

功能描述

关闭与服务器的连接，同时释放被PGconn对象使用的存储器。

原型

```
void PQfinish(PGconn *conn);
```

参数

表 5-51 PQfinish 参数

关键字	参数说明
conn	指向包含链接的对象指针。

注意事项

若PQstatus判断服务器连接尝试失败，应用程序调用PQfinish释放被PGconn对象使用的存储器，PQfinish调用后PGconn指针不可再次使用。

示例

请参见示例章节。

5.5.4.1.6 PQreset

功能描述

重置与服务器的通讯端口。

原型

```
void PQreset(PGconn *conn);
```

参数

表 5-52 PQreset 参数

关键字	参数说明
conn	指向包含链接的对象指针。

注意事项

此函数将关闭与服务器的连接并且试图与同一个服务器重建新的连接，并使用所有前面使用过的参数。该函数在连接异常后进行故障恢复时很有用。

示例

请参见示例章节。

5.5.4.1.7 PQstatus

功能描述

返回链接的状态。

原型

```
ConnStatusType PQstatus(const PGconn *conn);
```

参数

表 5-53 PQstatus 参数

关键字	参数说明
conn	指向包含链接的对象指针。

返回值

ConnStatusType：链接状态的枚举，包括：

```
CONNECTION_STARTED
等待进行连接。

CONNECTION_MADE
连接成功；等待发送。

CONNECTION_AWAITING_RESPONSE
等待来自服务器的响应。

CONNECTION_AUTH_OK
已收到认证；等待后端启动结束。

CONNECTION_SSL_STARTUP
协商SSL加密。

CONNECTION_SETENV
协商环境驱动的参数设置。

CONNECTION_OK
链接正常。

CONNECTION_BAD
链接故障。
```

注意事项

状态可以是多个值之一。但是，在异步连接过程之外只能看到其中两个：CONNECTION_OK和CONNECTION_BAD。与数据库的良好连接状态为CONNECTION_OK。状态表示连接尝试失败CONNECTION_BAD。通常，“正常”状态将一直保持到PQfinish，但通信失败可能会导致状态过早变为CONNECTION_BAD。在这种情况下，应用程序可以尝试通过调用进行恢复PQreset。

示例

请参见示例章节。

5.5.4.2 数据库执行语句函数

与数据库服务器的连接成功建立，便可以使用这里描述的函数执行SQL查询和命令。

5.5.4.2.1 PQexec

功能描述

向服务器提交一条命令并等待结果。

原型

```
PGresult *PQexec(PGconn *conn, const char *command);
```

参数

表 5-54 PQexec 参数

关键字	参数说明
conn	指向包含链接的对象指针。
command	需要执行的查询字符串。

返回值

PGresult：包含查询结果的对象指针。

注意事项

应该调用PQresultStatus函数来检查任何错误的返回值（包括空指针的值，在这种情况下它将返回PGRES_FATAL_ERROR）。使用PQerrorMessage获取有关错误的更多信息。

须知

命令字符串可以包括多个SQL命令（用分号分隔）。在一个PQexec调用中发送的多个查询是在一个事务里处理的，除非在查询字符串里有明确的BEGIN/COMMIT命令把整个字符串分隔成多个事务。请注意，返回的PGresult结构只描述字符串里执行的最后一条命令的结果，如果有一个命令失败，那么字符串处理的过程就会停止，并且返回的PGresult会描述错误条件。

示例

请参见示例章节。

5.5.4.2.2 PQprepare

功能描述

用给定的参数提交请求，创建一个预备语句，然后等待结束。

原型

```
PGresult *PQprepare(PGconn *conn,
                    const char *stmtName,
                    const char *query,
                    int nParams,
                    const Oid *paramTypes);
```

参数

表 5-55 PQprepare 参数

关键字	参数说明
conn	指向包含链接的对象指针。
stmtName	需要执行的prepare语句。
query	需要执行的查询字符串。
nParams	参数个数。
paramTypes	声明参数类型的数组。

返回值

PGresult：包含查询结果的对象指针。

注意事项

- PQprepare创建一个为PQexecPrepared执行用的预备语句，本特性支持命令的重复执行，不需要每次都进行解析和规划。PQprepare仅在协议3.0及以后的连接中支持，使用协议2.0时，PQprepare将失败。
- 该函数从查询字符串创建一个名为stmtName的预备语句，该查询字符串必须包含一个SQL命令。stmtName可以是""来创建一个未命名的语句，在这种情况下，任何预先存在的未命名的语句都将被自动替换；否则，如果在当前会话中已经定义了语句名称，那么这就是一个错误。如果使用了任何参数，那么在查询中将它们称为$1,$2等。nParams是在paramTypes[]数组中预先指定类型的参数的数量。（当nParams为0时，数组指针可以为NULL） paramTypes[]通过OID指定要分配给参数符号的数据类型。如果paramTypes为NULL，或者数组中的任何特定元素为零，服务器将按照对非类型化字面字符串的相同方式为参数符号分配数据类型。另外，查询可以使用数字高于nParams的参数符号；还将推断这些符号的数据类型。

须知

通过执行SQLPREPARE语句，还可以创建与PQexecPrepared一起使用的预备语句。此外，虽然没有用于删除预备语句的libpq函数，但是SQL DEALLOCATE语句可用于此目的。

示例

请参见示例章节。

5.5.4.2.3 PQresultStatus

功能描述

返回命令的结果状态。

原型

```
ExecStatusType PQresultStatus(const PGresult *res);
```

参数

表 5-56 PQresultStatus 参数

关键字	参数说明
res	包含查询结果的对象指针。

返回值

PQresultStatus：命令执行结果的枚举，包括：

```
PQresultStatus可以返回下面数值之一：
PGRES_EMPTY_QUERY
发送给服务器的字串是空的。

PGRES_COMMAND_OK
成功完成一个不返回数据的命令。

PGRES_TUPLES_OK
成功执行一个返回数据的查询（比如SELECT或者SHOW）。

PGRES_COPY_OUT
（从服务器）Copy Out（拷贝出）数据传输开始。

PGRES_COPY_IN
Copy In（拷贝入）（到服务器）数据传输开始。

PGRES_BAD_RESPONSE
服务器的响应无法理解。

PGRES_NONFATAL_ERROR
发生了一个非致命错误（通知或者警告）。

PGRES_FATAL_ERROR
发生了一个致命错误。
```

```
PGRES_COPY_BOTH
拷贝入/出（到和从服务器）数据传输开始。这个特性当前只用于流复制， 所以这个状态不会在普通应用中发生。

PGRES_SINGLE_TUPLE
PGresult包含一个来自当前命令的结果元组。 这个状态只在查询选择了单行模式时发生
```

注意事项

- 请注意，恰好检索到零行的SELECT命令仍然显示PGRES_TUPLES_OK。PGRES_COMMAND_OK用于永远不能返回行的命令（插入或更新，不带返回子句等）。PGRES_EMPTY_QUERY响应可能表明客户端软件存在bug。
- 状态为PGRES_NONFATAL_ERROR的结果永远不会由PQexec或其他查询执行函数直接返回，此类结果将传递给通知处理程序。

示例

请参见示例章节。

5.5.4.2.4 PQclear

功能描述

释放与PGresult相关联的存储空间，任何不再需要的查询结果都应该用PQclear释放掉。

原型

```
void PQclear(PGresult *res);
```

参数

表 5-57 PQclear 参数

关键字	参数说明
res	包含查询结果的对象指针。

注意事项

PGresult不会自动释放，当提交新的查询时它并不消失，甚至断开连接后也不会。要删除它，必须调用PQclear，否则则会有内存泄漏。

示例

请参见示例章节。

5.5.4.3 异步命令处理

PQexec函数对普通的同步应用里提交命令已经足够使用。但是它却有几个缺陷，而这些缺陷可能对某些用户很重要：

- PQexec等待命令结束，而应用可能还有其它的工作要做（比如维护用户界面等），此时并不希望PQexec阻塞应用。
- 因为客户端应用在等待结果的时候是处于挂起状态的，所以应用很难判断它是否该尝试结束正在进行的命令。
- PQexec只能返回一个PGresult结构。如果提交的命令字符串包含多个SQL命令，除了最后一个PGresult以外都会被PQexec丢弃。
- PQexec总是收集命令的整个结果，将其缓存在一个PGresult中。虽然这为应用简化了错误处理逻辑，但是对于包含多行的结果是不切实际的。

不想受到这些限制的应用可以改用下面的函数，这些函数也是构造PQexec的函数：PQsendQuery和PQgetResult。PQsendQueryParams，PQsendPrepare，PQsendQueryPrepared也可以和PQgetResult一起使用。

5.5.4.3.1 PQsendQuery

功能描述

向服务器提交一个命令而不等待结果。如果查询成功发送则返回1，否则返回0。

原型

```
int PQsendQuery(PGconn *conn, const char *command);
```

参数

表 5-58 PQsendQuery 参数

关键字	参数说明
conn	指向包含链接的对象指针。
command	需要执行的查询字符串.

返回值

int：执行结果为1表示成功，0表示失败，失败原因存到conn->errorMessage中。

注意事项

在成功调用PQsendQuery后，调用PQgetResult一次或者多次获取结果。PQgetResult返回空指针表示命令已执行完成，否则不能再次调用PQsendQuery（在同一连接上）。

示例

请参见示例章节。

5.5.4.3.2 PQsendQueryParams

功能描述

给服务器提交一个命令和分隔的参数，而不等待结果。

原型

```
int PQsendQueryParams(PGconn *conn,
                      const char *command,
                      int nParams,
                      const Oid *paramTypes,
                      const char * const *paramValues,
                      const int *paramLengths,
                      const int *paramFormats,
                      int resultFormat);
```

参数

表 5-59 PQsendQueryParams 参数

关键字	参数说明
conn	指向包含链接的对象指针。
command	需要执行的查询字符串。
nParams	参数个数。
paramTypes	参数类型。
paramValues	参数值。
paramLengths	参数长度。
paramFormats	参数格式。
resultFormat	结果的格式。

返回值

int：执行结果为1表示成功，0表示失败，失败原因存到conn->errorMessage中。

注意事项

该函数等效于PQsendQuery，只是查询参数可以和查询字符串分开声明。函数的参数处理和PQexecParams类似，它不能在2.0版本的协议连接上工作，并且它只允许在查询字符串里出现一条命令。

示例

请参见**示例**章节。

5.5.4.3.3 PQsendPrepare

功能描述

发送一个请求，创建一个给定参数的预备语句，而不等待结束。

原型

```
int PQsendPrepare(PGconn *conn,
                  const char *stmtName,
                  const char *query,
                  int nParams,
                  const Oid *paramTypes);
```

参数

表 5-60 PQsendPrepare 参数

关键字	参数说明
conn	指向包含链接的对象指针。
stmtName	需要执行的prepare名称。
query	需要执行的查询字符串。
nParams	参数个数。
paramTypes	声明参数类型的数组。

返回值

int：执行结果为1表示成功，0表示失败，失败原因存到conn->errorMessage中。

注意事项

该函数为PQprepare的异步版本：如果能够分派请求，则返回1，否则返回0。调用成功后，调用PQgetResult判断服务端是否成功创建了preparedStatement。函数的参数与PQprepare一样处理。与PQprepare一样，它也不能在2.0协议的连接上工作。

示例

请参见示例章节。

5.5.4.3.4 PQsendQueryPrepared

功能描述

发送一个请求执行带有给出参数的预备语句，不等待结果。

原型

```
int PQsendQueryPrepared(PGconn *conn,
                        const char *stmtName,
```



```
                int nParams,
                const char * const *paramValues,
                const int *paramLengths,
                const int *paramFormats,
                int resultFormat);
```

参数

表 5-61 PQsendQueryPrepared 参数

关键字	参数说明
conn	指向包含链接信息的对象指针。
stmtName	需要执行的prepare名称。
nParams	参数类型。
paramValues	参数值。
paramLengths	参数长度。
paramFormats	参数格式。
resultFormat	结果的格式。

返回值

int：执行结果为1表示成功，0表示失败，失败原因存到conn->error_message中。

注意事项

该函数类似于PQsendQueryParams，但是要执行的命令是通过命名一个预先准备的语句来指定的，而不是提供一个查询字符串。该函数的参数与PQexecPrepared一样处理。和PQexecPrepared一样，它也不能在2.0协议的连接上工作。

示例

请参见示例章节。

5.5.4.3.5 PQflush

功能描述

尝试将任何排队的输出数据刷新到服务器。

原型

```
int PQflush(PGconn *conn);
```

参数

表 5-62 PQflush 参数

关键字	参数说明
conn	指向包含链接信息的对象指针

返回值

int：如果成功（或者如果发送队列为空），则返回0；如果由于某种原因失败，则返回-1；如果发送队列中的所有数据都发送失败，则返回1。（此情况只有在连接为非阻塞时才能发生），失败原因存到conn->error_message中。

注意事项

在非阻塞连接上发送任何命令或数据之后，调用PQflush。如果返回1，则等待套接字变为读或写就绪。如果为写就绪状态，则再次调用PQflush。如果已经读到，调用PQconsumeInput，然后再次调用PQflush。重复，直到PQflush返回0。（必要检查读就绪并使用 PQconsumeInput 耗尽输入，因为服务器可能会阻止尝试向客户端发送数据（例如 NOTICE 消息），并且在客户端读取它的数据之前不会读取客户端的数据。）一旦PQflush返回0，就等待套接字准备好，然后按照上面描述读取响应。

示例

请参见**示例**章节。

5.5.4.4 取消正在处理的查询

客户端应用可以使用本节描述的函数，要求取消一个仍在被服务器处理的命令。

5.5.4.4.1 PQgetCancel

功能描述

创建一个数据结构，其中包含取消通过特定数据库连接发出的命令所需的信息。

原型

```
PGcancel *PQgetCancel(PGconn *conn);
```

参数

表 5-63 PQgetCancel 参数

关键字	参数说明
conn	指向包含链接信息的对象指针。

返回值

PGcancel：指向包含cancel信息对象的指针。

注意事项

PQgetCancel创建一个给定PGconn连接对象的PGcancel对象。如果给定的conn是NULL或无效连接，它将返回NULL。PGcancel对象是一个不透明的结构，应用程序不能直接访问它；它只能传递给PQcancel或PQfreeCancel。

示例

请参见**示例**章节。

5.5.4.4.2 PQfreeCancel

功能描述

释放PQgetCancel创建的数据结构。

原型

```
void PQfreeCancel(PGcancel *cancel);
```

参数

表 5-64 PQfreeCancel 参数

关键字	参数说明
cancel	指向包含cancel信息的对象指针。

注意事项

PQfreeCancel释放一个由前面的PQgetCancel创建的数据对象。

示例

请参见**示例**章节。

5.5.4.4.3 PQcancel

功能描述

要求服务器放弃处理当前命令。

原型

```
int PQcancel(PGcancel *cancel, char *errbuf, int errbufsize);
```


参数

表 5-65 PQcancel 参数

关键字	参数说明
cancel	指向包含cancel信息的对象指针。
errbuf	出错时保存错误信息的buffer。
errbufsize	保存错误信息的buffer大小。

返回值

int：执行结果为1表示成功，0表示失败，失败原因存到errbuf中。

注意事项

- 成功发送并不保证请求将产生任何效果。如果取消有效，当前命令将提前终止并返回错误结果。如果取消失败（例如，因为服务器已经处理完命令），无返回结果。
- 如果errbuf是信号处理程序中的局部变量，则可以安全地从信号处理程序中调用PQcancel。就PQcancel而言，PGcancel对象是只读的，因此它也可以从一个线程中调用，这个线程与操作PGconn对象线程是分离的。

示例

请参见示例章节。

5.5.5 连接参数

表 5-66 连接参数

参数	描述
host	要连接的主机名。如果主机名以斜杠开头，则它声明使用UNIX域套接字通讯而不是TCP/IP通讯；该值就是套接字文件所存储的目录。如果没有声明host，那么默认是与位于/tmp目录（或者安装GaussDB的时候声明的套接字目录）里面的UNIX域套接字连接。在没有UNIX域套接字的机器上，默认与localhost连接。

参数	描述
hostaddr	与之连接的主机的IP地址，是标准的IPv4地址格式，比如，172.28.40.9。如果声明了一个非空的字符串，那么使用TCP/IP通讯机制。 使用hostaddr取代host可以让应用避免一次主机名查找，这一点对于那些有时间约束的应用来说可能是非常重要的。不过，GSSAPI或SSPI认证方法要求主机名（host）。因此，应用下面的规则： 1. 如果声明了不带hostaddr的host那么就强制进行主机名查找。 2. 如果声明中没有host，hostaddr的值给出服务器网络地址；如果认证方法要求主机名，那么连接尝试将失败。 3. 如果同时声明了host和hostaddr，那么hostaddr的值作为服务器网络地址。host的值将被忽略，除非认证方法需要它，在这种情况下它将被用作主机名。 **须知** • 要注意如果host不是网络地址hostaddr处的服务器名，那么认证很有可能失败。 • 如果主机名（host）和主机地址都没有，那么libpq将使用一个本地的UNIX域套接字进行连接；或者是在没有UNIX域套接字的机器上，它将尝试与localhost连接。
port	主机服务器的端口号，或者在UNIX域套接字连接时的套接字扩展文件名。
user	要连接的用户名，缺省是与运行该应用的用户操作系统名同名的用户。
dbname	数据库名，缺省和用户名相同。
password	如果服务器要求口令认证，所用的口令。
connect_timeout	连接的最大等待时间，以秒计（用十进制整数字符串书写），0或者不声明表示无穷。不建议把连接超时的值设置得小于2秒。
client_encoding	为这个连接设置client_encoding配置参数。除了对应的服务器选项接受的值，可以使用auto从客户端中的当前环境中确定正确的编码（UNIX系统上是LC_CTYPE环境变量）。
tty	忽略（以前，该参数指定了发送服务器调试输出的位置）。
options	添加命令行选项以在运行时发送到服务器。
application_name	为application_name配置参数指定一个值，表明当前用户身份。
fallback_application_name	为application_name配置参数指定一个后补值。如果通过一个连接参数或PGAPPNAME环境变量没有为application_name给定一个值，将使用这个值。在希望设置一个默认应用名但不希望它被用户覆盖的一般工具程序中指定一个后补值很有用。
keepalives	控制客户端侧的TCP保持激活是否使用。缺省值是1，意思为打开，但是如果不想要保持激活，可以更改为0，意思为关闭。通过UNIX域套接字做的连接忽略这个参数。

参数	描述
keepalives_idle	在TCP应该发送一个保持激活的信息给服务器之后，控制不活动的秒数。0值表示使用系统缺省。通过UNIX域套接字做的连接或者如果禁用了保持激活则忽略这个参数。
keepalives_interval	在TCP保持激活信息没有被应该传播的服务器承认之后，控制秒数。0值表示使用系统缺省。通过UNIX域套接字做的连接或者如果禁用了保持激活则忽略这个参数。
keepalives_count	控制TCP发送保持激活信息的次数。0值表示使用系统缺省。通过UNIX域套接字做的连接或者如果禁用了保持激活则忽略这个参数。
tcp_user_timeout	在支持TCP_USER_TIMEOUT套接字选项的操作系统上，指定传输的数据在TCP连接被强制关闭之前可以保持未确认状态的最大时长。0值表示使用系统缺省。通过UNIX域套接字做的连接忽略这个参数。
rw_timeout	设置客户端连接读写超时时间。 当libpq侧触发超时且连接关闭时，其下发给数据库侧正在运行的业务会被强制终止。该能力受GUC参数check_disconnect_query控制，设置为on表示支持该能力，设置为off表示不支持该能力。
sslmode	启用SSL加密的方式： ● disable：不使用SSL安全连接。 ● allow：如果数据库服务器要求使用，则可以使用SSL安全加密连接，但不验证数据库服务器的真实性。 ● prefer：如果数据库支持，那么首选使用SSL安全加密连接，但不验证数据库服务器的真实性。 ● require：必须使用SSL安全连接，但是只做了数据加密，而并不验证数据库服务器的真实性。 ● verify-ca：必须使用SSL安全连接，当前windows odbc不支持cert方式认证。 ● verify-full：必须使用SSL安全连接，当前windows odbc不支持cert方式认证。
sslcompression	如果设置为1（默认），SSL连接之上传送的数据将被压缩（这要求OpenSSL版本为0.9.8或更高）。如果设置为0，压缩将被禁用（这要求OpenSSL版本为1.0.0或更高）。如果建立的是一个没有SSL的连接，这个参数会被忽略。如果使用的OpenSSL版本不支持该参数，它也会被忽略。压缩会占用CPU时间，但是当瓶颈为网络时可以提高吞吐量。如果CPU性能是限制因素，禁用压缩能够改进响应时间和吞吐量。
sslcert	这个参数指定客户端SSL证书的文件名。如果没有建立SSL连接，这个参数会被忽略。
sslkey	这个参数指定用于客户端证书的密钥位置。它能够指定一个从外部“引擎”（引擎是OpenSSL的可载入模块）得到的密钥。一个外部引擎说明应该由一个冒号分隔的引擎名称以及一个引擎相关的关键标识符组成。如果没有建立SSL连接，这个参数会被忽略。
sslrootcert	这个参数指定一个包含SSL证书机构（CA）证书的文件名称。如果该文件存在，服务器的证书将被验证是由这些机构之一签发。

参数	描述
sslcrl	这个参数指定SSL证书撤销列表（CRL）的文件名。列在这个文件中的证书如果存在，在尝试认证该服务器证书时会被拒绝。
requirepeer	这个参数指定服务器的操作系统用户，例如requirepeer=postgres。当建立一个UNIX域套接字连接时，如果设置了这个参数，客户端在连接开始时检查服务器进程是否运行在指定的用户名之下。如果发现不是，该连接会被一个错误中断。这个参数能被用来提供与TCP/IP连接上SSL证书相似的服务器认证（注意，如果UNIX域套接字在/tmp或另一个公共可写的位置，任何用户能启动一个在那里侦听的服务器。使用这个参数来保证你连接的是一个由可信用户运行的服务器）。这个选项只在实现了peer认证方法的平台上受支持。
krbsrvname	当用GSSAPI认证时，要使用的Kerberos服务名。为了让Kerberos认证成功，这必须匹配在服务器配置中指定的服务名。
gsslib	用于GSSAPI认证的GSS库。只用在Windows上。设置为gssapi可强制libpq用GSSAPI库来代替默认的SSPI进行认证。
service	用于附加参数的服务名。它指定保持附加连接参数的pg_service.conf中的一个服务名。这允许应用只指定一个服务名，这样连接参数能被集中维护。
authtype	不再使用“authtype”，因此将其标记为“不显示”。将其保留在数组中，以免拒绝旧应用程序中的conninfo字符串，这些应用程序可能仍在尝试设置它。
remote_nodename	指定连接本地节点的远端节点名称。
localhost	指定在一个连接通道中的本地地址。
localport	指定在一个连接通道中的本地端口。
fencedUdfRPCMode	控制fenced UDF RPC协议是使用UNIX域套接字或特殊套接字文件名。缺省值是0，意思为关闭。使用UNIX domain socket模式，文件类型为“.s.PGSQL.%d”；但是要使用fenced udf ，文件类型为.s.fencedMaster_unixdomain，可以更改为1，意思为开启。
replication	这个选项决定是否该连接应该使用复制协议而不是普通协议。这是PostgreSQL的复制连接以及pg_basebackup之类的工具在内部使用的协议，但也可以被第三方应用使用。支持下列值，大小写无关： ● true、on、yes、1 连接进入到物理复制模式。 ● database 连接进入到逻辑复制模式，连接到dbname参数中指定的数据库。 ● false、off、no、0 该连接是一个常规连接，这是默认行为。 在物理或者逻辑复制模式中，仅能使用简单查询协议。
backend_version	传递到远端的后端版本号。

参数	描述
prototype	设置当前协议级别，默认：PROTO_TCP。
connection_info	Connection_info是一个包含driver_name、driver_version、driver_path和os_user的json字符串。 如果不为NULL，使用connection_info 忽略connectionExtraInf。 如果为NULL，生成与libpq相关的连接信息字符串，当connectionExtraInf为false时connection_info只有driver_name和driver_version。
connectionExtraInf	设置connection_info是否存在扩展信息，默认值为0，如果包含其他信息，则需要设置为1。

5.6 基于 Psycopg 开发

Psycopg是一种用于执行SQL语句的PythonAPI，可以为GaussDB数据库提供统一访问接口，应用程序可基于它进行数据操作。Psycopg2是对libpq的封装，主要使用C语言实现，既高效又安全。它具有客户端游标和服务器端游标、异步通信和通知、支持“COPY TO/COPY FROM”功能。支持多种类型Python开箱即用，适配GaussDB数据类型；通过灵活的对象适配系统，可以扩展和定制适配。Psycopg2兼容Unicode。

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

表 5-67 Psycopg 支持平台

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

须知

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

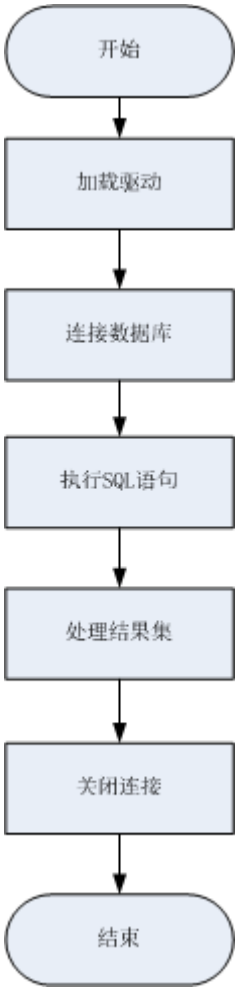
例如，在执行某个调用psycopg2的应用软件client.py时，将环境变量显性赋予该应用软件：

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

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

5.6.1 开发流程

图 5-4 采用 Psycopg2 开发应用程序的流程



5.6.2 Psycopg 包

步骤1 准备相关驱动和依赖库。可以从发布包中获取，包名为GaussDB-Kernel_数据库版本号_操作系统版本号_64bit_Python.tar.gz。

解压后有两个文件夹：

- psycopg2：psycopg2库文件。
- lib：lib库文件。

步骤2 加载驱动。

- 在使用驱动之前，需要做如下操作：
  a. 先解压版本对应驱动包。
  ```
  tar zxvf xxxx-Python.tar.gz
  ```
  b. 使用root用户将psycopg2复制到python安装目录下的site-packages文件夹下。
  ```
  su root
  cp psycopg2 $(python3 -c 'import site; print(site.getsitepackages()[0])') -r
  ```
  c. 修改psycopg2目录权限为755。
  ```
  chmod 755 $(python3 -c 'import site; print(site.getsitepackages()[0])')/psycopg2 -R
  ```
  d. 将psycopg2目录添加到环境变量$PYTHONPATH，并使之生效。
  ```
  export PYTHONPATH=$(python3 -c 'import site; print(site.getsitepackages()[0])'):$PYTHONPATH
  ```
  e. 对于非数据库用户，需要将解压后的lib目录，配置在LD_LIBRARY_PATH中。
  ```
  export LD_LIBRARY_PATH=path/to/lib:$LD_LIBRARY_PATH
  ```
- 在创建数据库连接之前，需要先加载如下数据库驱动程序：
  ```
  import  psycopg2
  ```

步骤3 连接数据库。

非SSL方式连接数据库：

1. 使用psycopg2.connect函数获得connection对象。
2. 使用connection对象创建cursor对象。

SSL方式连接数据库：

用户通过psycopy2连接GaussDB服务器时，可以通过开启SSL加密客户端和服务器之间的通讯。在使用SSL时，默认用户已经获取了服务端和客户端所需要的证书和私钥文件，关于证书等文件的获取请参见Openssl相关文档和命令。

1. 使用*.ini文件（python的configparser包可以解析这种类型的配置文件）保存数据库连接的配置信息。
2. 在连接选项中添加SSL连接相关参数：sslmode、sslcert、sslkey、sslrootcert。
   a. sslmode：可选项见**表5-68**。
   b. sslcert：客户端证书路径。
   c. sslkey：客户端密钥路径。
   d. sslrootcert：根证书路径。
3. 使用psycopg2.connect函数获得connection对象。
4. 使用connection对象创建cursor对象。

⚠ 注意

使用SSL安全连接数据库，需保证所使用的python解释器为生成动态链接库（.so）文件的方式编译，可通过如下步骤确认python解释器的连接方式。

1. 在python解释器命令行中输入import ssl，导入SSL。
2. 执行ps ux查询python解释器运行的pid（假设pid为******）。
3. 在shell命令行中执行pmap -p ****** | grep ssl，查看返回结果中是否包含libssl.so的相关路径。如果有，则python解释器为动态链接方式编译。

表 5-68 sslmode 的可选项及其描述

sslmode	是否会启用SSL加密	描述
disable	否	不使用SSL安全连接。
allow	可能	如果数据库服务器要求使用，则可以使用SSL安全加密连接，但不验证数据库服务器的真实性。
prefer	可能	如果数据库支持，那么首选使用SSL连接，但不验证数据库服务器的真实性。
require	是	必须使用SSL安全连接，但是仅进行数据加密，而并不验证数据库服务器的真实性。
verify-ca	是	必须使用SSL安全连接，并且校验服务端CA有效性。
verify-full	是	必须使用SSL安全连接，目前GaussDB暂不支持。

步骤4 执行SQL语句。

1. 构造操作语句，使用%s作为占位符，执行时psycopg2会用参数值智能替换掉占位符。可以添加RETURNING子句，来得到自动生成的字段值。
2. 使用cursor.execute方法来操作一行SQL语句，使用cursor.executemany方法来操作多行SQL语句。

步骤5 处理结果集。

1. cursor.fetchone()：这种方法提取的查询结果集的下一行，返回一个序列，没有数据可用时则返回空。
2. cursor.fetchall()：这种方法获取所有查询结果（剩余）行，返回一个列表。空行时则返回空列表。

说明

对于数据库特有数据类型，如tinyint类型，查询结果中相应字段为字符串形式。

步骤6 关闭连接。

在使用数据库连接完成相应的数据操作后，需要关闭数据库连接。关闭数据库连接可以直接调用其close方法，如connection.close()。

⚠ 注意

此方法关闭数据库连接，并不自动调用commit()。如果只是关闭数据库连接而不调用commit()方法，那么所有更改将会丢失。

----结束

5.6.3 示例：常用操作

```
import psycopg2
import os

# 从环境变量中获取用户名和密码
user = os.getenv('user')
password = os.getenv('password')

# 创建连接对象
conn=psycopg2.connect(database="database", user=user, password=password, host="localhost", port=port)
cur=conn.cursor() #创建指针对象

# 创建连接对象（SSl连接）
conn = psycopg2.connect(dbname="database", user=user, password=password, host="localhost", port=port,
        sslmode="verify-ca", sslcert="client.crt",sslkey="client.key",sslrootcert="cacert.pem")
注意： 如果sslcert、sslkey、sslrootcert没有填写，默认取当前用户.postgresql目录下对应的client.crt、
client.key、root.crt


# 创建表
cur.execute("CREATE TABLE student(id integer,name varchar,sex varchar);")

# 插入数据
cur.execute("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",(1,'Aspirin','M'))
cur.execute("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",(2,'Taxol','F'))
cur.execute("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",(3,'Dixheral','M'))

# 批量插入数据
stus = ((4,'John','M'),(5,'Alice','F'),(6,'Peter','M'))
cur.executemany("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",stus)

# 获取结果
cur.execute('SELECT * FROM student')
results=cur.fetchall()
print (results)

# 提交操作
conn.commit()

# 插入一条数据
cur.execute("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",(7,'Lucy','F'))

# 回退操作
conn.rollback()

# 关闭连接
cur.close()
conn.close()

psycopg2常用连接方式
1. conn = psycopg2.connect(dbname="dbname", user=user, password=password, host="localhost",
port=port)
2. conn = psycopg2.connect(f"dbname=dbname user={user} password={password}  host=localhost
port=port")
3. 使用日志
import logging
import psycopg2
from psycopg2.extras import LoggingConnection
```



```
import os

# 从环境变量中获取用户名和密码
user = os.getenv('user')
password = os.getenv('password')

logging.basicConfig(level=logging.DEBUG) # 日志级别
logger = logging.getLogger(__name__)

db_settings = {
    "user": user,
    "password": password,
    "host": "localhost",
    "database": "dbname",
    "port": port
}

# LoggingConnection默认记录所有SQL，可自行实现filter过滤不需要的或敏感的SQL，下面给出了简单的过滤password相关SQL的示例
class SelfLoggingConnection(LoggingConnection):

    def filter(self, msg, curs):
        if db_settings['password'] in msg.decode():
            return b'queries containing the password will not be recorded'
        return msg

conn = psycopg2.connect(connection_factory=SelfLoggingConnection, **db_settings)
conn.initialize(logger)
```

⚠ 注意

- LoggingConnection默认记录所有SQL信息，且不会对敏感信息进行脱敏，可通过filter函数自行定义输出的SQL内容。
- 日志功能是psycopg2为了方便开发者显性调试全量SQL而提供的额外功能，默认情况下不需要使用。该功能会在pyscopg2执行SQL语句前打印SQL语句，但是，需要在debug日志级别下才会输出。该功能不是默认功能，只是在有特殊需要的时候才使用，没有特别需求，不建议使用。详情参考：https://www.psycopg.org/docs/extras.html?highlight=loggingconnection

5.6.4 Psycopg 接口参考

Psycopg接口是一套提供给用户的API方法，本节将对部分常用接口做具体描述。

5.6.4.1 psycopg2.connect()

功能描述

此方法创建新的数据库会话并返回新的connection对象。

原型

```
import os
conn=psycopg2.connect(dbname="test",user=os.getenv('user'),password=os.getenv('password'),host="127.0.0.1",port="5432")
```

参数

表 5-69 psycopg2.connect 参数

关键字	参数说明
dbname	数据库名称
user	用户名
password	密码
host	数据库IP地址，默认为UNIX socket类型。
port	连接端口号，默认为5432。
sslmode	ssl模式，ssl连接时用。
sslcert	客户端证书路径，ssl连接时用。
sslkey	客户端密钥路径，ssl连接时用。
sslrootcert	根证书路径，ssl连接时用。
hostaddr	数据库IP地址。
connect_timeout	客户端连接超时时间。
client_encoding	客户端编码格式。
application_name	application_name的参数值。
fallback_application_name	application_name参数的回退值。
keepalives	控制是否客户端tcp保持连接，默认为1，表示打开；值为0，表示关闭。若UNIX域套接字连接，则忽略。
options	连接开始时发送给服务器的命令行选项。
keepalives_idle	控制向服务器发送keepalive消息之前不活动的描述，若keepalive被禁用，则忽略此参数。
keepalives_interval	控制未得到服务器确认的keepalive消息应重新传输的描述，若keepalive被禁用，则忽略此参数。
keepalives_count	控制客户端与服务端连接断开之前可能丢失的tcp保持连接的数量。
replication	确认连接使用的是复制协议而不是普通协议。
requiressl	支持sslmode设置。
sslcompression	ssl压缩。设置为1，则通过ssl连接发送的数据将被压缩；设置为0，则禁用压缩。若没有建立ssl的连接，则忽略此参数。
sslcrl	证书吊销列表文件路径，验证ssl服务端证书是否可用。
requirepeer	指定服务器的操作系统用户名。

返回值

connection对象（连接数据库实例的对象）。

示例

请参见**示例：常用操作**。

5.6.4.2 connection.cursor()

功能描述

此方法用于返回新的cursor对象。

原型

```
cursor(name=None, cursor_factory=None, scrollable=None, withhold=False)
```

参数

表 5-70 connection.cursor 参数

关键字	参数说明
name	cursor名称，默认为None。
cursor_factory	用于创造非标准cursor，默认为None。
scrollable	设置SCROLL选项，默认为None。
withhold	设置HOLD选项，默认为False。

返回值

cursor对象（用于整个数据库使用Python编程的cursor）。

示例

请参见**示例：常用操作**。

5.6.4.3 cursor.execute(query,vars_list)

功能描述

此方法执行被参数化的SQL语句（即占位符，而不是SQL文字）。psycopg2模块支持用%s标志的占位符。

原型

```
cursor.execute(query,vars_list)
```


参数

表 5-71 cursor.execute 参数

关键字	参数说明
query	待执行的sql语句。
vars_list	变量列表，匹配query中%s占位符。

返回值

无

示例

请参见**示例：常用操作**。

5.6.4.4 cursor.executemany(query,vars_list)

功能描述

此方法执行SQL命令所有参数序列或序列中的SQL映射。

原型

```
cursor.executemany(query,vars_list)
```

参数

表 5-72 cursor.executemany 参数

关键字	参数说明
query	待执行的SQL语句。
vars_list	变量列表，匹配query中%s占位符。

返回值

无

示例

请参见**示例：常用操作**。

5.6.4.5 connection.commit()

功能描述

此方法将当前挂起的事务提交到数据库。

> **注意**
>
> 默认情况下，Psycopg在执行第一个命令之前打开一个事务：如果不调用commit()，任何数据操作的效果都将丢失。

原型

```
connection.commit()
```

参数

无

返回值

无

示例

请参见**示例：常用操作**。

5.6.4.6 connection.rollback()

功能描述

此方法回滚当前挂起事务。

> **注意**
>
> 执行关闭连接“close()”而不先提交更改“commit()”将导致执行隐式回滚。

原型

```
connection.rollback()
```

参数

无

返回值

无

示例

请参见**示例：常用操作**。

5.6.4.7 cursor.fetchone()

功能描述

此方法提取查询结果集的下一行，并返回一个元组。

原型

```
cursor.fetchone()
```

参数

无

返回值

单个元组，为结果集的第一条结果，当没有更多数据可用时，返回为“None”。

示例

请参见**示例：常用操作**。

5.6.4.8 cursor.fetchall()

功能描述

此方法获取查询结果的所有（剩余）行，并将它们作为元组列表返回。

原型

```
cursor.fetchall()
```

参数

无

返回值

元组列表，为结果集的所有结果。空行时则返回空列表。

示例

请参见**示例：常用操作**。

5.6.4.9 cursor.close()

功能描述

此方法关闭当前连接的游标。

原型

```
cursor.close()
```

参数

无

返回值

无

示例

请参见**示例：常用操作**。

5.6.4.10 connection.close()

功能描述

此方法关闭数据库连接。

> **注意**
>
> 此方法关闭数据库连接，并不自动调用commit()。如果只是关闭数据库连接而不调用commit()方法，那么所有更改将会丢失。

原型

```
connection.close()
```

参数

无

返回值

无

示例

请参见**示例：常用操作**。

5.7 调试

用户可以根据自己的需要，通过修改实例数据目录下的postgresql.conf文件中特定的配置参数来控制日志的输出，从而更好的了解数据库的运行状态。

可调整的配置参数请参见**表5-73**。

表 5-73 配置参数

参数名称	描述	取值范围	备注
client_min_messages	配置发送到客户端信息的级别。	• DEBUG5 • DEBUG4 • DEBUG3 • DEBUG2 • DEBUG1 • LOG • NOTICE • WARNING • ERROR • FATAL • PANIC 默认值：NOTICE 。	设置级别后，发送到客户端的信息包含所设级别及以下所有低级别会发送的信息。级别越低，发送的信息越少。
log_min_messages	配置写到服务器日志里信息的级别。	• DEBUG5 • DEBUG4 • DEBUG3 • DEBUG2 • DEBUG1 • INFO • NOTICE • WARNING • ERROR • LOG • FATAL • PANIC 默认值：WARNING。	指定某一级别后，写到日志的信息包含所有更高级别会输出的信息。级别越高，服务器日志的信息越少。

参数名称	描述	取值范围	备注
log_min_error_statement	配置写到服务器日志中错误SQL语句的级别。	• DEBUG5 • DEBUG4 • DEBUG3 • DEBUG2 • DEBUG1 • INFO • NOTICE • WARNING • ERROR • FATAL • PANIC 缺省值：ERROR。	所有导致一个特定级别（或者更高级别）错误的SQL语句都将记录在服务器日志中。 只有系统管理员可以修改该参数。
log_min_duration_statement	配置语句执行持续的最短时间。如果某个语句的持续时间大于或者等于设置的毫秒数，则会在日志中记录该语句及其持续时间。打开这个选项可以方便地跟踪需要优化的查询。	INT类型。 默认值：-1。 单位：毫秒。	设置为-1表示关闭这个功能。 只有系统管理员可以修改该参数。
log_connections/ log_disconnections	配置是否在每次会话连接或结束时向服务器日志里打印一条信息。	• on：每次会话连接或结束时向日志里打印一条信息。 • off：每次会话连接或结束时不向日志里打印信息。 默认值：off。	-
log_duration	配置是否记录每个已完成语句的持续时间。	• on：记录每个已完成语句的持续时间。 • off：不记录已完成语句的持续时间。 默认值：off。	只有系统管理员可以修改该参数。

参数名称	描述	取值范围	备注
log_statement	配置日志中记录哪些SQL语句。	• none：不记录任何SQL语句。 • ddl：记录数据定义语句。 • mod：记录数据定义语句和数据操作语句。 • all ：记录所有语句。 默认值： none。	只有系统管理员可以修改该参数。
log_hostname	配置是否记录主机名。	• on：记录主机名。 • off：不记录主机名。 默认值：off。	缺省时，连接日志只记录所连接主机的IP地址。打开这个选项会同时记录主机名。 该参数同时影响查看审计结果、PG_STAT_ACTIVITY和log_line_prefix参数。

上表有关参数级别的说明请参见表5-74。

表 5-74 日志级别参数说明

级别	说明
DEBUG[1-5]	提供开发人员使用的信息。5级为最高级别，依次类推，1级为最低级别。
INFO	提供用户隐含要求的信息。如在VACUUM VERBOSE过程中的信息。
NOTICE	提供可能对用户有用的信息。如长标识符的截断，作为主键一部分创建的索引。
WARNING	提供给用户的警告。如在事务块范围之外的COMMIT。
ERROR	报告导致当前命令退出的错误。
LOG	报告一些管理员感兴趣的信息。如检查点活跃性。
FATAL	报告导致当前会话终止的原因。
PANIC	报告导致所有会话退出的原因。

6 SQL 调优指南

SQL调优的唯一目的是“资源利用最大化”，即CPU、内存、磁盘I/O、网络I/O四种资源利用最大化。所有调优手段都是围绕资源使用开展的。所谓资源利用最大化是指SQL语句尽量高效，节省资源开销，以最小的代价实现最大的效益。比如做典型点查询的时候，可以用seqscan+filter(即读取每一条元组和点查询条件进行匹配)实现，也可以通过indexscan实现，显然indexscan可以以更小的代价实现相同的效果。

根据硬件资源和客户的业务特征确定合理的集群部署方案和表定义是数据库在多数情况下满足性能要求的基础。下文的调优说明假设您已根据“软件安装”指引在安装过程中按照合理的集群方案完成了安装，且已经根据“开发设计建议”的指引进行了数据库设计。

6.1 Query 执行流程

SQL引擎从接受SQL语句到执行SQL语句需要经历的步骤如**图6-1**和**表6-1**所示。其中，红色字体部分为DBA可以介入实施调优的环节。

图 6-1 SQL 引擎执行查询类 SQL 语句的流程

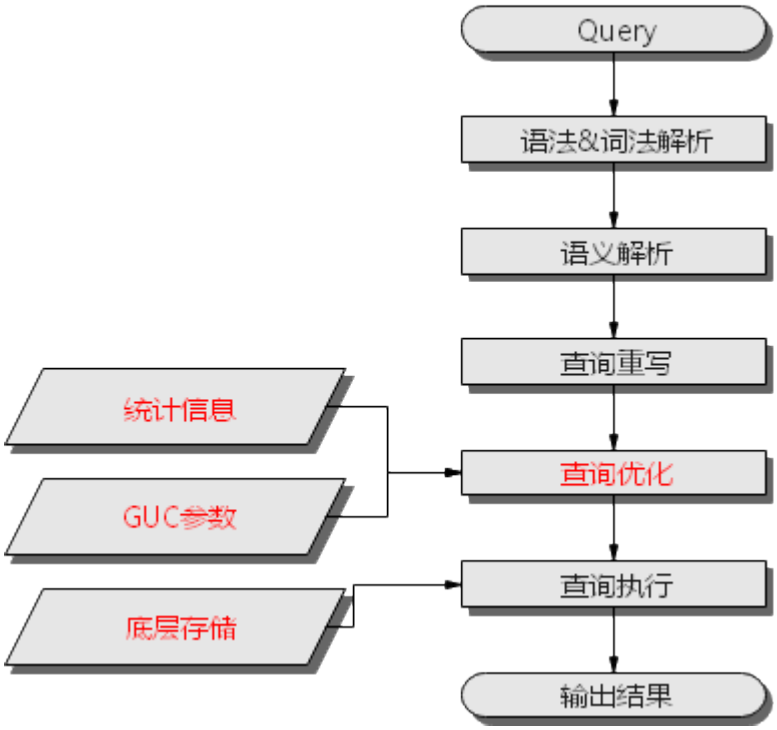


表 6-1 SQL 引擎执行查询类 SQL 语句的步骤说明

步骤	说明
1、语法&词法解析	按照约定的SQL语句规则，把输入的SQL语句从字符串转化为格式化结构(Stmt)。
2、语义解析	将“语法&词法解析”输出的格式化结构转化为数据库可以识别的对象。
3、查询重写	根据规则把“语义解析”的输出等价转化为执行上更为优化的结构。
4、查询优化	根据“查询重写”的输出和数据库内部的统计信息规划SQL语句具体的执行方式，也就是执行计划。统计信息和GUC参数对查询优化（执行计划）的影响，请参见**调优手段之统计信息**和**调优手段之GUC参数**。
5、查询执行	根据“查询优化”规划的执行路径执行SQL查询语句。底层存储方式的选择合理性，将影响查询执行效率。

调优手段之统计信息

GaussDB优化器是典型的基于代价的优化 (Cost-Based Optimization，简称CBO)。在这种优化器模型下，数据库根据表的元组数、字段宽度、NULL记录比率、distinct值、MCV值、HB值等表的特征值，以及一定的代价计算模型，计算出每一个执行步骤的不同执行方式的输出元组数和执行代价(cost)，进而选出整体执行代价最小/首元组返回代价最小的执行方式进行执行。这些特征值就是统计信息。从上面描述可以看出统计信息是查询优化的核心输入，准确的统计信息将帮助规划器选择最合适的查询规

划，一般来说通过analyze语法收集整个表或者表的若干个字段的统计信息，周期性地运行ANALYZE，或者在对表的大部分内容做了更改之后马上运行它是个好习惯。

注意，DDL可能会导致统计信息发生变化，进而导致计划跳变。当表上做了DDL操作后，应注意统计信息是否需要重新收集。

调优手段之 GUC 参数

查询优化的主要目的是为查询语句选择高效的执行方式。

如下SQL语句:

```
select count(1)
from customer inner join store_sales on (ss_customer_sk = c_customer_sk);
```

在执行customer inner join store_sales的时候，GaussDB支持Nested Loop、Merge Join和Hash Join三种不同的Join方式。优化器会根据表customer和表store_sales的统计信息估算结果集的大小以及每种Join方式的执行代价，然后对比选出执行代价最小的执行计划。

正如前面所说，执行代价计算都是基于一定的模型和统计信息进行估算，当因为某些原因代价估算不能反映真实的cost的时候，就需要通过GUC参数设置的方式让执行计划倾向更优规划。例如：random_page_cost参数表示优化器计算一次非顺序抓取磁盘页面的开销，该参数默认值为4。当机器磁盘随机读取的速度较快时，比如SSD设备，可以将该参数的值适当调小，更改后，索引扫描的代价降低，生成计划时更倾向于选择索引扫描的方式。

调优手段之 SQL 重写

除了上述干预SQL引擎所生成执行计划的执行性能外，根据数据库的SQL执行机制以及大量的实践发现，有些场景下，在保证客户业务SQL逻辑的前提下，通过一定规则由DBA重写SQL语句，可以大幅度地提升SQL语句的性能。

这种调优场景对DBA的要求比较高，需要对客户业务有足够的了解，同时也需要扎实的SQL语句基本功，后续会介绍几个常见的SQL改写场景。

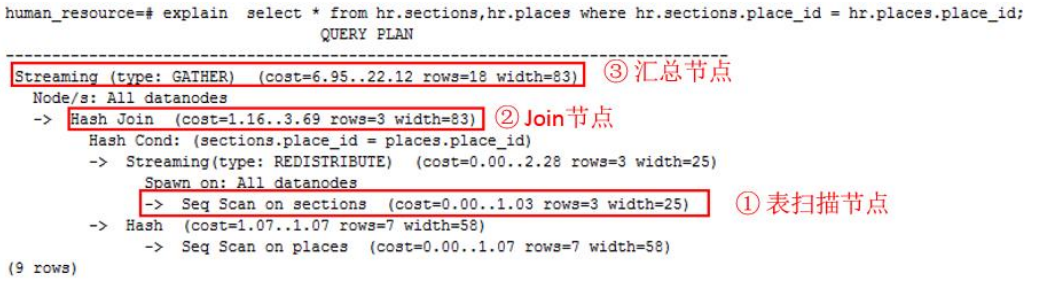
6.2 SQL 执行计划介绍

6.2.1 SQL 执行计划概述

SQL执行计划是一个节点树，显示GaussDB执行一条SQL语句时执行的详细步骤。每一个步骤为一个数据库运算符。

使用EXPLAIN命令可以查看优化器为每个查询生成的具体执行计划。EXPLAIN给每个执行节点都输出一行，显示基本的节点类型和优化器为执行这个节点预计的开销值。如图6-2所示。

图 6-2 SQL 执行计划示例



- 最底层节点是表扫描节点，它扫描表并返回原始数据行。不同的表访问模式有不同的扫描节点类型：顺序扫描、索引扫描等。最底层节点的扫描对象也可能是非表行数据（不是直接从表中读取的数据），如VALUES子句和返回行集的函数，它们有自己的扫描节点类型。
- 如果查询需要连接、聚集、排序、或者对原始行做其它操作，那么就会在扫描节点之上添加其它节点。 并且这些操作通常都有多种方法，因此在这些位置也有可能出现不同的执行节点类型。
- 第一行(最上层节点)是执行计划总执行开销的预计。这个数值就是优化器试图最小化的数值。

执行计划显示格式

GaussDB对执行计划提供了normal、pretty、summary、run四种显示格式：

- normal：代表使用默认的打印格式。图6-2中即为此显示格式。
- pretty：代表使用GaussDB改进后的新显示格式。新的格式层次清晰，计划包含了plan node id，性能分析简单直接。如图6-3。
- summary：是在pretty的基础上增加了对打印信息的分析。
- run：在summary的基础上，将统计的信息输出到csv格式的文件中，以便于进一步分析。

图 6-3 pretty 格式执行计划示例

```
openGauss=# explain select cjxh, count(1) from dwcjk group by cjxh;
 id |               operation                | E-rows | E-memory | E-width | E-costs
----+----------------------------------------+--------+----------+---------+---------
  1 | ->  Row Adapter                        |      1 |          |      52 | 58.42
  2 |    ->  Vector Streaming (type: GATHER) |      1 |          |      52 | 58.42
  3 |       ->  Vector Hash Aggregate        |      1 | 16MB     |      52 | 58.02
  4 |          ->  CStore Scan on dwcjk      |      1 | 1MB      |      44 | 58.00
(4 rows)
```

通过设置GUC参数explain_perf_mode，可以显示不同格式的执行计划。下文的用例默认显示pretty格式。

执行计划显示信息

除了设置不同的执行计划显示格式外，还可以通过不同的EXPLAIN用法，显示不同详细程度的执行计划信息。常见有如下几种，关于更多用法请参见**EXPLAIN**语法说明。

- EXPLAIN *statement*： 只生成执行计划，不实际执行。其中statement代表SQL语句。

- EXPLAIN ANALYZE *statement*：生成执行计划，进行执行，并显示执行的概要信息。显示中加入了实际的运行时间统计，包括在每个规划节点内部花掉的总时间(以毫秒计)和它实际返回的行数。
- EXPLAIN PERFORMANCE *statement*：生成执行计划，进行执行，并显示执行期间的全部信息。

为了测量运行时在执行计划中每个节点的开销，EXPLAIN ANALYZE或EXPLAIN PERFORMANCE会在当前查询执行上增加性能分析的开销。在一个查询上运行EXPLAIN ANALYZE或EXPLAIN PERFORMANCE有时会比普通查询明显地花费更多的时间。超支的数量依赖于查询的本质和使用的平台。

因此，当定位SQL运行慢问题时，如果SQL长时间运行未结束，建议通过EXPLAIN命令查看执行计划，进行初步定位。如果SQL可以运行出来，则推荐使用EXPLAIN ANALYZE或EXPLAIN PERFORMANCE查看执行计划及其实际的运行信息，以便更精准地定位问题原因。

6.2.2 详解

如**SQL执行计划概述**节中所说，EXPLAIN会显示执行计划，但并不会实际执行SQL语句。EXPLAIN ANALYZE和EXPLAIN PERFORMANCE两者都会实际执行SQL语句并返回执行信息。在这一节将详细解释执行计划及执行信息。

执行计划

以如下SQL语句为例：

```
select
    cjxh,
    count(1)
from dwcjk
group by cjxh;
```

执行EXPLAIN的输出为：

```
openGauss=# EXPLAIN SELECT * FROM t1,t2 WHERE t1.c1 = t2.c2;
                           QUERY PLAN
-----------------------------------------------------------------
 Hash Join  (cost=23.73..341.30 rows=16217 width=180)
   Hash Cond: (t1.c1 = t2.c2)
   ->  Seq Scan on t1  (cost=0.00..122.17 rows=5317 width=76)
   ->  Hash  (cost=16.10..16.10 rows=610 width=104)
         ->  Seq Scan on t2  (cost=0.00..16.10 rows=610 width=104)
(5 rows)
```

执行计划字段解读（横向）：

- id：执行算子节点编号。
- operation：具体的执行节点算子名称。

  Streaming是一个特殊的算子，它实现了分布式架构的核心数据shuffle功能，Streaming共有三种形态，分别对应了分布式结构下不同的数据shuffle功能：
  - Streaming (type: GATHER)：作用是coordinator从DN收集数据。
  - Streaming(type: REDISTRIBUTE)：作用是DN根据选定的列把数据重分布到所有的DN。
  - Streaming(type: BROADCAST)：作用是把当前DN的数据广播给其他所有的DN
- E-rows：每个算子估算的输出行数。

- E-memory：DN上每个算子估算的内存使用量，只有DN上执行的算子会显示。某些场景会在估算的内存使用量后使用括号显示该算子在内存资源充足下可以自动扩展的内存上限。
- E-width：每个算子输出元组的估算宽度。
- E-costs：每个算子估算的执行代价。
  - E-costs是优化器根据成本参数定义的单位来衡量的，习惯上以磁盘页面抓取为1个单位， 其它开销参数将参照它来设置。
  - 每个节点的开销（E-costs值）包括它的所有子节点的开销。
  - 开销只反映了优化器关心的东西，并没有把结果行传递给客户端的时间考虑进去。虽然这个时间可能在实际的总时间里占据相当重要的分量，但是被优化器忽略了，因为它无法通过修改规划来改变。

执行计划层级解读（纵向）：

1. 第一层：Seq Scan on t2

   表扫描算子，用Seq Scan的方式扫描表t2。这一层的作用是把表t2的数据从buffer或者磁盘上读上来输送给上层节点参与计算。

2. 第二层：Hash

   Hash算子，作用是把下层计算输送上来的算子计算hash值，为后续hash join操作做数据准备。

3. 第三层：Seq Scan on t1

   表扫描算子，用Seq Scan的方式扫描表t1。这一层的作用是把表t1的数据从buffer或者磁盘上读上来输送给上层节点参与hash join计算。

4. 第四层：Hash Join

   join算子，主要作用是将t1表和t2表的数据通过hash join的方式连接，并输出结果数据。

执行计划中的主要关键字说明：

1. 表访问方式
   - Seq Scan

     全表顺序扫描。

   - Index Scan

     优化器决定使用两步的规划：最底层的规划节点访问一个索引，找出匹配索引条件的行的位置，然后上层规划节点真实地从表中抓取出那些行。独立地抓取数据行比顺序地读取它们的开销高很多，但是因为并非所有表的页面都被访问了，这么做实际上仍然比一次顺序扫描开销要少。使用两层规划的原因是，上层规划节点在读取索引标识出来的行位置之前，会先将它们按照物理位置排序，这样可以最小化独立抓取的开销。

     如果在WHERE里面使用的好几个字段上都有索引，那么优化器可能会使用索引的AND或OR的组合。但是这么做要求访问两个索引，因此与只使用一个索引，而把另外一个条件只当作过滤器相比，这个方法未必是更优。

     根据索引排序机制的差异，索引扫描可以分为以下几类。

     - Bitmap Index Scan

       使用位图索引抓取数据页。

     - Index Scan using index_name

使用简单索引搜索，该方式按照索引键的顺序在索引表中抓取数据。该方式最常用于在大数据量表中只抓取少量数据的情况，或者通过ORDER BY条件匹配索引顺序的查询，以减少排序时间。

- Index-Only Scan

  当需要的所有信息都包含在索引中时，仅索引扫描便可获取所有数据，不需要引用表。

- Bitmap Heap Scan

  从其他操作创建的位图中读取页面，过滤掉不符合条件的行。位图堆扫描可避免随机I/O，加快读取速度。

- TID Scan

  通过TupleID扫描表。

- Index Ctid Scan

  通过Ctid上的索引对表进行扫描。

- CTE Scan

  CTE对子查询的操作进行评估并将查询结果临时存储，相当于一个临时表。CTE Scan算子对该临时表进行扫描。

- Foreign Scan

  从远程数据源读取数据。

- Function Scan

  获取函数返回的结果集，将它们作为从表中读取的行并返回。

- Sample Scan

  查询并返回采样数据。

- Subquery Scan

  读取子查询的结果。

- Values Scan

  作为VALUES命令的一部分读取常量。

- WorkTable Scan

  工作表扫描。在操作中间阶段读取，通常是使用WITH RECURSIVE声明的递归操作。

- CStore Index Ctid Scan

  按照索引的条件进行扫描，返回满足条件的tid集合。

- CStore Index Heap Scan

  实现tid集合的交、差、并运算，并通过集合的结果获取对应元组。

2. 表连接方式

   - Nested Loop

     嵌套循环，适用于被连接的数据子集较小的查询。在嵌套循环中，外表驱动内表，外表返回的每一行都要在内表中检索找到它匹配的行，因此整个查询返回的结果集不能太大（不能大于10000），要把返回子集较小的表作为外表，而且在内表的连接字段上建议要有索引。

   - (Sonic) Hash Join

     哈希连接，适用于数据量大的表的连接方式。优化器使用两个表中较小的表，利用连接键在内存中建立hash表，然后扫描较大的表并探测散列，找到

与散列匹配的行。Sonic和非Sonic的Hash Join的区别在于所使用hash表结构不同，不影响执行的结果集。

- Merge Join

  归并连接，通常情况下执行性能差于哈希连接。如果源数据已经被排序过，在执行融合连接时，并不需要再排序，此时融合连接的性能优于哈希连接。

3. 运算符
   - sort

     对结果集进行排序。
   - filter

     EXPLAIN输出显示WHERE子句当作一个"filter"条件附属于顺序扫描计划节点。这意味着规划节点为它扫描的每一行检查该条件，并且只输出符合条件的行。预计的输出行数降低了，因为有WHERE子句。不过，扫描仍将必须访问所有 10000 行，因此开销没有降低；实际上它还增加了一些（确切的说，通过10000 * cpu_operator_cost）以反映检查WHERE条件的额外CPU时间。
   - LIMIT

     LIMIT限定了执行结果的输出记录数。如果增加了LIMIT，那么不是所有的行都会被检索到。
   - Append

     合并子操作的结果。
   - Aggregate

     将查询行产生的结果进行组合。可以是GROUPBY、UNION、SELECT DISTINCT子句等函数的组合。
   - BitmapAnd

     位图的AND操作，通过该操作组成匹配更复杂条件的位图。
   - BitmapOr

     位图的OR操作，通过该操作组成匹配更复杂条件的位图。
   - Gather

     将并行线程的数据汇总。
   - Group

     对行进行分组，以进行GROUP BY操作。
   - GroupAggregate

     聚合GROUP BY操作的预排序行。
   - Hash

     对查询行进行散列操作，以供父查询使用。通常用于执行JOIN操作。
   - HashAggregate

     使用哈希表聚合GROUP BY的结果行。
   - Merge Append

     以保留排序顺序的方式对子查询结果进行组合，可用于组合表分区中已排序的行。
   - Recursive Union

     对递归函数的所有步骤进行并集操作。
   - SetOp

     集合运算，如INTERSECT或EXCEPT。

- Unique
  从有序的结果集中删除重复项。
- HashSetOp
  一种用于 INTERSECT 或 EXCEPT 等集合操作的策略，它使用 Append 来避免预排序的输入。
- LockRows
  锁定有问题的行以阻止其他查询写入，但允许读。
- Materialize
  将子查询的结果存储在内存里，以方便父查询快速访问获取。
- Result
  在不进行扫描的情况下返回一个值（比如硬编码的值）。
- WindowAgg
  窗口聚合函数，一般由OVER语句触发。
- Merge
  归并操作。
- StartWith Operator
  层次查询算子，用于执行递归查询操作。
- Index Cond
  索引扫描条件。
- Cstore Index And
  实现tid集合的交运算，和CStore Index Heap Scan搭配使用。
- Cstore Index Or
  实现tid集合的并运算，和CStore Index Heap Scan搭配使用。

4. 其他关键字
   - Partitioned
     对具体分区的操作。
   - Partition Iterator
     分区迭代器，通常代表子查询是对分区的操作。
   - InitPlan
     非相关子计划。
   - Remote Query
     下推到数据节点上的查询语句。
   - Exec Nodes
     具体执行计划的节点。
   - Data Node Scan on
     说明语句已下推给DN执行。

执行信息

在SQL调优过程中经常需要执行EXPLAIN ANALYZE或EXPLAIN PERFORMANCE查看SQL语句实际执行信息，通过对比实际执行与优化器的估算之间的差别来为优化提供依据。EXPLAIN PERFORMANCE相对于EXPLAIN ANALYZE增加了每个DN上的执行信息。

以如下SQL语句为例：

```
select count(1) from tb1;
```

执行EXPLAIN PERFORMANCE输出为：

```
openGauss=#  explain performance select count(1) from tb1;
 id |             operation             |     A-time      | A-rows | E-rows | E-distinct | Peak Memory  | E-memory | A-width | E-width | E-costs
----+-----------------------------------+-----------------+--------+--------+------------+--------------+----------+---------+---------+---------
  1 | ->  Aggregate                     | 19.269          |      1 |      1 |            | 10KB         |          |         |       8 | 2.19
  2 |    ->  Streaming (type: GATHER)   | 19.191          |      4 |      4 |            | 144KB        |          |         |       8 | 2.19
  3 |       ->  Aggregate               | [0.004,0.069]   |      4 |      4 |            | [10KB, 10KB] | 1MB      |         |       8 | 2.03
  4 |          ->  Seq Scan on public.tb1 | [0.001,0.060] |      5 |      5 |            | [12KB, 12KB] | 1MB      |         |       0 | 2.02
(4 rows)


                Memory Information (identified by plan id)
----------------------------------------------------------------------
 Coordinator Query Peak Memory:
        Query Peak Memory: 0MB
 DataNode Query Peak Memory
        datanode1 Query Peak Memory: 0MB
        datanode2 Query Peak Memory: 0MB
        datanode3 Query Peak Memory: 0MB
        datanode4 Query Peak Memory: 0MB
   1 --Aggregate
        Peak Memory: 10KB, Estimate Memory: 64MB
   2 --Streaming (type: GATHER)
        Peak Memory: 144KB, Estimate Memory: 64MB
   3 --Aggregate
        datanode1 Peak Memory: 10KB, Estimate Memory: 1024KB
        datanode2 Peak Memory: 10KB, Estimate Memory: 1024KB
        datanode3 Peak Memory: 10KB, Estimate Memory: 1024KB
        datanode4 Peak Memory: 10KB, Estimate Memory: 1024KB
        datanode1 Stream Send time: 0.000; Data Serialize time: 0.006
        datanode2 Stream Send time: 0.000; Data Serialize time: 0.004
        datanode3 Stream Send time: 0.000; Data Serialize time: 0.005
        datanode4 Stream Send time: 0.000; Data Serialize time: 0.003
   4 --Seq Scan on public.tb1
        datanode1 Peak Memory: 12KB, Estimate Memory: 1024KB
        datanode2 Peak Memory: 12KB, Estimate Memory: 1024KB
        datanode3 Peak Memory: 12KB, Estimate Memory: 1024KB
        datanode4 Peak Memory: 12KB, Estimate Memory: 1024KB
(25 rows)


 Targetlist Information (identified by plan id)
------------------------------------------------
   1 --Aggregate
        Output: count((count(1)))
   2 --Streaming (type: GATHER)
        Output: (count(1))
        Node/s: All datanodes
   3 --Aggregate
        Output: count(1)
   4 --Seq Scan on public.tb1
        Output: a, b
        Distribute Key: a
(10 rows)
```

```
                    Datanode Information (identified by plan id)
--------------------------------------------------------------------------------
  1 --Aggregate
       (actual time=19.269..19.269 rows=1 loops=1)
       (Buffers: 0)
       (CPU: ex c/r=50593, ex row=4, ex cyc=202372, inc cyc=50094480)
  2 --Streaming (type: GATHER)
       (actual time=12.371..19.191 rows=4 loops=1)
       (Buffers: 0)
       (CPU: ex c/r=12473027, ex row=4, ex cyc=49892108, inc cyc=49892108)
  3 --Aggregate
       datanode1 (actual time=0.041..0.041 rows=1 loops=1)
       datanode2 (actual time=0.063..0.063 rows=1 loops=1)
       datanode3 (actual time=0.069..0.069 rows=1 loops=1)
       datanode4 (actual time=0.004..0.004 rows=1 loops=1)
       datanode1 (Buffers: shared hit=1)
       datanode2 (Buffers: shared hit=1)
       datanode3 (Buffers: shared hit=1)
       datanode4 (Buffers: 0)
       datanode1 (CPU: ex c/r=32888, ex row=1, ex cyc=32888, inc cyc=107740)
       datanode2 (CPU: ex c/r=9992, ex row=2, ex cyc=19984, inc cyc=163264)
       datanode3 (CPU: ex c/r=12272, ex row=2, ex cyc=24544, inc cyc=180008)
       datanode4 (CPU: ex c/r=0, ex row=0, ex cyc=8100, inc cyc=10768)
  4 --Seq Scan on public.tb1
       datanode1 (actual time=0.027..0.029 rows=1 loops=1)
       datanode2 (actual time=0.054..0.055 rows=2 loops=1)
       datanode3 (actual time=0.059..0.060 rows=2 loops=1)
       datanode4 (actual time=0.001..0.001 rows=0 loops=1)
       datanode1 (Buffers: shared hit=1)
       datanode2 (Buffers: shared hit=1)
       datanode3 (Buffers: shared hit=1)
       datanode4 (Buffers: 0)
       datanode1 (CPU: ex c/r=74852, ex row=1, ex cyc=74852, inc cyc=74852)
       datanode2 (CPU: ex c/r=71640, ex row=2, ex cyc=143280, inc cyc=143280)
       datanode3 (CPU: ex c/r=77732, ex row=2, ex cyc=155464, inc cyc=155464)
       datanode4 (CPU: ex c/r=0, ex row=0, ex cyc=2668, inc cyc=2668)
(34 rows)

                    User Define Profiling
--------------------------------------------------------------------
 Plan Node id: 2  Track name: coordinator get datanode connection
       coordinator1: (time=0.054 total_calls=1 loops=1)
(2 rows)

                    ====== Query Summary =====
--------------------------------------------------------------------------------
 Datanode executor start time [datanode1, datanode2]: [1.352 ms,2.276 ms]
 Datanode executor end time [datanode4, datanode3]: [0.121 ms,0.187 ms]
 Remote query poll time: 10.992 ms, Deserialze time: 0.000 ms
 System available mem: 9188966KB
 Query Max mem: 9473228KB
 Query estimated mem: 2048KB
 Coordinator executor start time: 0.505 ms
 Coordinator executor run time: 19.277 ms
 Coordinator executor end time: 0.380 ms
 Planner runtime: 2.260 ms
 Query Id: 79375943434081342
 Total runtime: 20.803 ms
(12 rows)
```

图中显示执行信息分为以下7个部分

1. 以表格的形式将计划显示出来，包含有11个字段，分别是：id、operation、A-time、A-rows、E-rows、E-distinct、Peak Memory、E-memory、A-width、E-width和E-costs。其中计划类字段（id、operation以及E开头字段）的含义与执行EXPLAIN时的含义一致，详见**执行计划**小节中的说明。A-time、A-rows、E-distinct、Peak Memory、A-width的含义说明如下：
   - A-time：当前算子执行完成时间，一般DN上执行的算子的A-time是由[]括起来的两个值，分别表示此算子在所有DN上完成的最短时间和最长时间。

- A-rows：表示当前算子的实际输出元组数。
- E-distinct：表示hashjoin算子的distinct估计值。
- Peak Memory：此算子在每个DN上执行时使用的内存峰值。
- A-width：表示当前算子每行元组的实际宽度，仅对于重内存使用算子会显示，包括：(Vec)HashJoin、(Vec)HashAgg、(Vec) HashSetOp、(Vec)Sort、(Vec)Materialize算子等，其中(Vec)HashJoin计算的宽度是其右子树算子的宽度，会显示在其右子树上。

2. Predicate Information (identified by plan id)：

   这一部分主要显示的是静态信息，即在整个计划执行过程中不会变的信息，主要是一些join条件和一些filter信息。

3. Memory Information (identified by plan id)：

   这一部分显示的是整个计划中会将内存的使用情况打印出来的算子的内存使用信息，主要是Hash、Sort算子，包括算子峰值内存（peak memory），控制内存（control memory），估算内存使用（operator memory），执行时实际宽度（width），内存使用自动扩展次数（auto spread num），是否提前下盘（early spilled），以及下盘信息，包括重复下盘次数（spill Time(s)），内外表下盘分区数（inner/outer partition spill num），下盘文件数（temp file num），下盘数据量及最小和最大分区的下盘数据量（written disk IO [min, max] ）。

4. Targetlist Information (identified by plan id)

   这一部分显示的是每一个算子输出的目标列。

5. DataNode Information (identified by plan id)：

   这一部分会将各个算子的执行时间、CPU、buffer的使用情况全部打印出来。

6. User Define Profiling

   这一部分显示的是CN和DN、DN和DN建连的时间，以及存储层的一些执行信息。

7. ====== Query Summary =====：

   这一部分主要打印总的执行时间和网络流量，包括了各个DN上初始化和结束阶段的最大最小执行时间、CN上的初始化、执行、结束阶段的时间，以及当前语句执行时系统可用内存、语句估算内存等信息。

须知

- A-rows和E-rows的差异体现了优化器估算和实际执行的偏差度。一般来说，偏差越大，越可以认为优化器生成的计划越不可信，人工干预调优的必要性越大。
- A-time中的两个值偏差越大，表明此算子的计算偏斜（在不同DN上执行时间差异）越大，人工干预调优的必要性越大。
- Max Query Peak Memory经常用来估算SQL语句耗费内存，也被用来作为SQL语句调优时运行态内存参数设置的重要依据。一般会以EXPLAIN ANALYZE或EXPLAIN PERFORMANCE的输出作为进一步调优的输入。

6.3 调优流程

对慢SQL语句进行分析，通常包括以下步骤：

操作步骤

步骤1 收集SQL中涉及到的所有表的统计信息。在数据库中，统计信息是规划器生成计划的源数据。没有收集统计信息或者统计信息陈旧往往会造成执行计划严重劣化，从而导致性能问题。从经验数据来看，10%左右性能问题是因为没有收集统计信息。具体请参见**更新统计信息**。

步骤2 通过查看执行计划来查找原因。如果SQL长时间运行未结束，通过EXPLAIN命令查看执行计划，进行初步定位。如果SQL可以运行出来，则推荐使用EXPLAIN ANALYZE或EXPLAIN PERFORMANCE查看执行计划及实际运行情况，以便更精准地定位问题原因。有关执行计划的详细介绍请参见**SQL执行计划介绍**。

步骤3 **审视和修改表定义**。

步骤4 针对EXPLAIN或EXPLAIN PERFORMANCE信息，定位SQL慢的具体原因以及改进措施，具体参见**典型SQL调优点**。

步骤5 通常情况下，有些SQL语句可以通过查询重写转换成等价的，或特定场景下等价的语句。重写后的语句比原语句更简单，且可以简化某些执行步骤达到提升性能的目的。查询重写方法在各个数据库中基本是通用的。**经验总结：SQL语句改写规则**介绍了几种常用的通过改写SQL进行调优的方法。

----结束

6.4 更新统计信息

在数据库中，统计信息是规划器生成计划的源数据。没有收集统计信息或者统计信息陈旧往往会造成执行计划严重劣化，从而导致性能问题。

背景信息

ANALYZE语句可收集与数据库中表内容相关的统计信息，统计结果存储在系统表PG_STATISTIC中。查询优化器会使用这些统计数据，以生成最有效的执行计划。

建议在执行了大批量插入/删除操作后，例行对表或全库执行ANALYZE语句更新统计信息。目前默认收集统计信息的采样比例是30000行（即：guc参数default_statistics_target默认设置为100），如果表的总行数超过一定行数（大于1600000），建议设置guc参数default_statistics_target为-2，即按2%收集样本估算统计信息。

对于在批处理脚本或者存储过程中生成的中间表，也需要在完成数据生成之后显式的调用ANALYZE。

操作步骤

使用以下命令更新某个表或者整个database的统计信息。

```
ANALYZE tablename;                    --更新单个表的统计信息
ANALYZE;                              --更新全库的统计信息
```

说明

使用EXPLAIN查看各SQL的执行计划时，如果发现某个表SEQ SCAN的输出中rows=10，rows=10是系统给的默认值，有可能该表没有进行ANALYZE，需要对该表执行ANALYZE。

6.5 审视和修改表定义

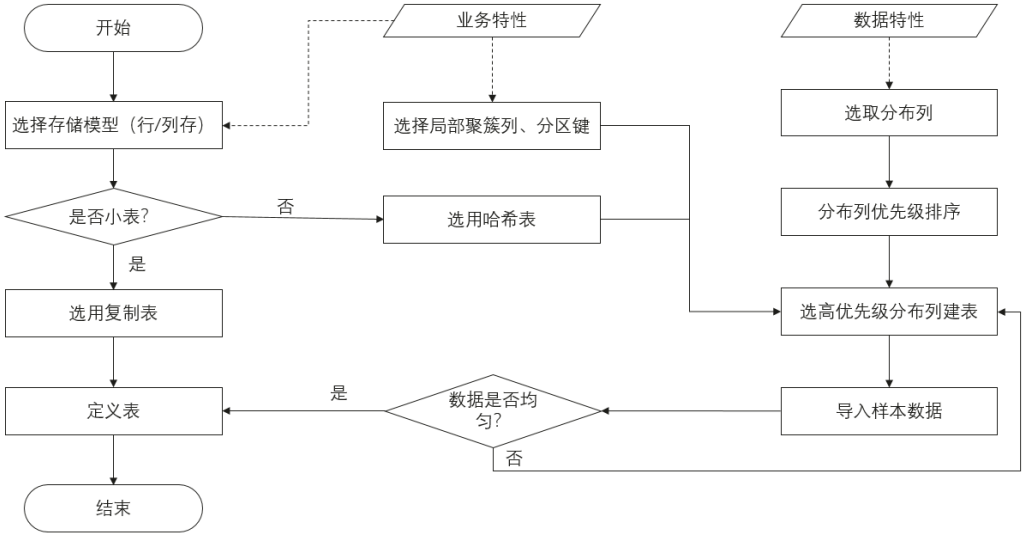
6.5.1 审视和修改表定义概述

在分布式框架下，数据分布在各个DN上。一个或者几个DN的数据存在一块物理存储设备上，好的表定义至少需要达到以下几个目标：

1. **表数据均匀分布在各个DN上**，以防止单个DN对应的存储设备空间不足造成集群有效容量下降。选择合适分布列，避免数据分布倾斜可以实现该点。
2. **表Scan压力均匀分散在各个DN上**，以避免单DN的Scan压力过大，形成Scan的单节点瓶颈。分布列不选择基表上等值filter中的列可以实现该点。
3. **减少扫描数据量**。通过分区的剪枝机制可以实现该点。
4. **尽量减少随机I/O**。通过聚簇/局部聚簇可以实现该点。
5. **尽量避免数据shuffle**，减小网络压力。通过选择join-condition或者group by列为分布列可以最大程度的实现这点。

从上述描述来看表定义中最重要的一点是分布列的选择。创建表定义一般遵循图6-4所示流程。表定义在数据库设计阶段创建，在SQL调优过程中进行审视和修改。

图 6-4 表定义流程



6.5.2 选择存储模型

进行数据库设计时，表设计上的一些关键项将严重影响后续整库的查询性能。表设计对数据存储也有影响：好的表设计能够减少I/O操作及最小化内存使用，进而提升查询性能。

表的存储模型选择是表定义的第一步。客户业务属性是表的存储模型的决定性因素，依据下面表格选择适合当前业务的存储模型。

存储模型	适用场景
行存	点查询(返回记录少，基于索引的简单查询)。 增删改比较多的场景。

6.5.3 选择分布方式

复制表（Replication）方式将表中的全量数据在集群的每一个DN实例上保留一份。主要适用于记录集较小的表。这种存储方式的优点是每个DN上都有该表的全量数据，在join操作中可以避免数据重分布操作，从而减小网络开销，同时减少了plan segment(每个plan segment都会起对应的线程)；缺点是每个DN都保留了表的完整数据，造成数据的冗余。一般情况下只有较小的维度表才会定义为Replication表。

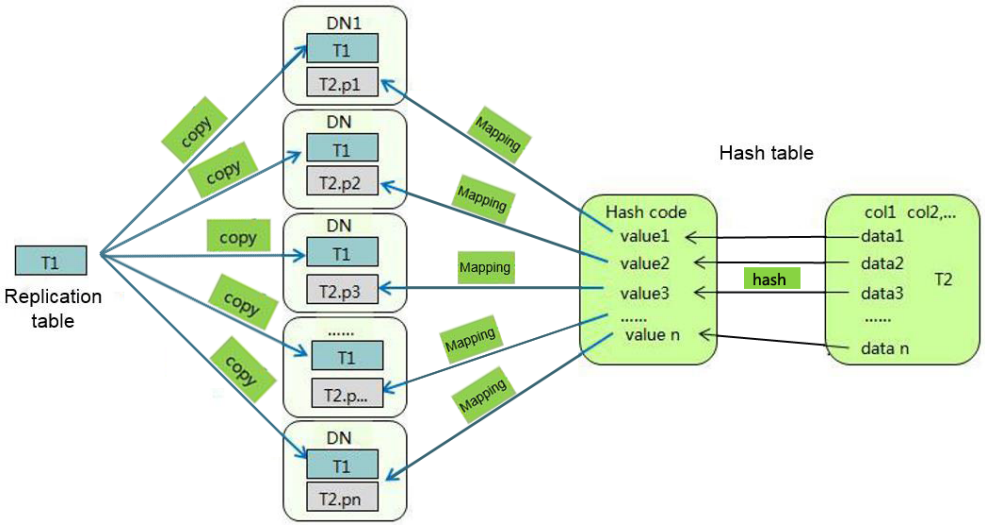
哈希（Hash）表将表中某一个或几个字段进行hash运算后，生成对应的hash值，根据DN实例与哈希值的映射关系获得该元组的目标存储位置。对于Hash分布表，在读/写数据时可以利用各个节点的I/O资源，大大提升表的读/写速度。一般情况下大表定义为Hash表。

范围（Range）和列表（List）分布是由用户自定义的分布策略，根据分布列的取值落入满足一定范围或者具体值的对应目标DN，这两种分布方式便于用户灵活地进行数据管理，但对用户本身的数据抽象能力有一定的要求。

策略	描述	适用场景
Hash	表数据通过hash方式散列到集群中的所有DN实例上。	数据量较大的事实表。
Replication	集群中每一个DN实例上都有一份全量表数据。	小表、维度表。
Range	表数据对指定列按照范围进行映射，分布到对应DN。	用户需要自定义分布规则的场景。
List	表数据对指定列按照具体值进行映射，分布到对应DN。	用户需要自定义分布规则的场景。

如图6-5所示，复制表如图中的表T1，哈希表如图中的表T2。

图 6-5 复制表和哈希表



6.5.4 选择分布列

Hash分布表的分布列选取至关重要，需要满足以下原则：

1. **列值应比较离散，以便数据能够均匀分布到各个DN**。例如，考虑选择表的主键为分布列，如在人员信息表中选择身份证号码为分布列。
2. **在满足第一条原则的情况下尽量不要选取存在常量filter的列**。例如，表dwcjk相关的部分查询中出现dwcjk的列zqdh存在常量的约束(例如zqdh=’000001’)，那么就应当尽量不用zqdh做分布列。
3. **在满足前两条原则的情况，考虑选择查询中的连接条件为分布列**，以便Join任务能够下推到DN中执行，且减少DN之间的通信数据量。

对于Hash分表策略，如果分布列选择不当，可能导致数据倾斜，查询时出现部分DN的I/O短板，从而影响整体查询性能。因此在采用Hash分表策略之后需对表的数据进行数据倾斜性检查，以确保数据在各个DN上是均匀分布的。可以使用以下SQL检查数据倾斜性

```
select
xc_node_id, count(1)
from tablename
group by xc_node_id
order by xc_node_id desc;
```

其中xc_node_id对应DN，一般来说，**不同DN的数据量相差5%以上即可视为倾斜，如果相差10%以上就必须要调整分布列**。

GaussDB支持多分布列特性，可以更好地满足数据分布的均匀性要求。

Range/List分布表的分布列由用户根据实际需要进行选择。除了需要选择合适的分布列，还需要注意分布规则对数据分布的影响。

6.5.5 使用分区表

分区表是把逻辑上的一张表根据某种方案分成几张物理块进行存储。这张逻辑上的表称之为分区表，物理块称之为分区。分区表是一张逻辑表，不存储数据，数据实际是存储在分区上的。分区表和普通表相比具有以下优点：

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

GaussDB支持的分区表为范围分区表。

范围分区表：将数据基于范围映射到每一个分区。这个范围是由创建分区表时指定的分区键决定的。分区键经常采用日期，例如将销售数据按照月份进行分区。

6.5.6 选择数据类型

高效数据类型，主要包括以下三方面：

1. **尽量使用执行效率比较高的数据类型**

   一般来说整型数据运算(包括=、＞、＜、≧、≦、≠等常规的比较运算，以及group by)的效率比字符串、浮点数要高。

2. **尽量使用短字段的数据类型**

   长度较短的数据类型不仅可以减小数据文件的大小，提升I/O性能；同时也可以减小相关计算时的内存消耗，提升计算性能。比如对于整型数据，如果可以用smallint就尽量不用int，如果可以用int就尽量不用bigint。

3. **使用一致的数据类型**

   表关联列尽量使用相同的数据类型。如果表关联列数据类型不同，数据库必须动态地转化为相同的数据类型进行比较，这种转换会带来一定的性能开销。

6.6 典型 SQL 调优点

SQL调优是一个不断分析与尝试的过程：试跑Query，判断性能是否满足要求；如果不满足要求，则通过**查看执行计划**分析原因并进行针对性优化；然后重新试跑和优化，直到满足性能目标。

6.6.1 SQL 自诊断

用户在执行查询或者执行INSERT/DELETE/UPDATE/CREATE TABLE AS语句时，可能会遇到性能问题。

SQL自诊断的告警类型与GUC参数resource_track_level的设置有关系。如果resource_track_level设置为query，则可以诊断多列/单列统计信息未收集和SQL不下推的告警。如果resource_track_level设置为operator，则可以诊断所有的告警场景。

SQL自诊断的诊断范围与GUC参数resource_track_cost的设置有关系。当SQL的代价大于resource_track_cost时，SQL才会被诊断。SQL的代价可以通过explain来确认。

SQL自诊断功能受enable_analyze_check参数影响，使用前应确认该开关已打开。

执行语句较多时，可能会由于内存管控导致部分数据无法收集，可以尝试将instr_unique_sql_count设置值调高。

告警场景

目前支持对以下7种导致性能问题的场景上报告警。

- 单列统计信息未收集

如果存在单列统计信息未收集，则上报相关告警。调优方法可以参考**更新统计信息**和**统计信息调优**。

告警信息示例：

整表的统计信息未收集：

```
Statistic Not Collect:
    schema_test.t1
```

单列统计信息未收集：

```
Statistic Not Collect:
    schema_test.t2(c1,c2)
```

- SQL不下推

  对于不下推的SQL，尽可能详细上报导致不下推的原因。调优方法可以参考案例**语句下推调优**。

  - 对于因函数而导致的不下推，会告警对应的函数名信息。
  - 对于不支持下推的语法，会告警对应语法不支持下推，例如：含有With Recursive，Distinct On，row表达式，返回值为record类型的，会告警相应语法不支持下推等等。

告警信息示例：

```
SQL is not plan-shipping, reason : "With Recursive" can not be shipped"
SQL is not plan-shipping, reason : "Function now() can not be shipped"
SQL is not plan-shipping, reason : "Function string_agg() can not be shipped"
```

- HashJoin中大表做内表

如果在表连接过程中使用了Hashjoin，且连接的内表行数是外表行数的10倍或以上；同时内表在每个DN上的平均行数大于10万行，且发生了下盘，则上报相关告警。调优方法可以参考**使用plan hint调优执行计划**。

告警信息示例：

```
PlanNode[7] Large Table is INNER in HashJoin “Hash Aggregate”
```

- 大表等值连接使用Nestloop

如果在表连接过程中使用了nestloop，并且两个表中较大表的行数平均每个DN上的行数大于10万行、表的连接中存在等值连接，则上报相关告警。调优方法可以参考**使用plan hint调优执行计划**。

告警信息示例：

```
PlanNode[5] Large Table with Equal-Condition use Nestloop"Nested Loop"
```

- 大表Broadcast

如果在Broadcast算子中，平均每DN的行数大于10万行，则告警大表broadcast。调优方法可以参考**使用plan hint调优执行计划**。

告警信息示例：

```
PlanNode[5] Large Table in Broadcast "Streaming(type: BROADCAST dop: 1/2)"
```

- 数据倾斜

某表在各DN上的分布，存在某DN上的行数是另一DN上行数的10倍或以上，且有DN中的行数大于10万行，则上报相关告警。调优方法可以参考案例**选择合适的分布列**和**数据倾斜调优**。

告警信息示例：


```
PlanNode[6] DataSkew:"Seq Scan", min_dn_tuples:0, max_dn_tuples:524288
```

- 估算不准

如果优化器的估算行数比实际行数大于10万行，并且估算行数和实际行数中较大值是较小值的10倍或以上，则上报相关告警。调优方法可以参考**使用plan hint调优执行计划**。

告警信息示例：

```
PlanNode[5] Inaccurate Estimation-Rows: "Hash Join" A-Rows:0, E-Rows:52488
```

规格约束

1. 告警字符串长度上限为2048。如果告警信息超过这个长度（例如存在大量未收集统计信息的超长表名，列名等信息）则不告警，只上报warning：
   ```
   WARNING, "Planner issue report is truncated, the rest of planner issues will be skipped"
   ```
2. 如果query存在limit节点（即查询语句中包含limit），则不会上报limit节点以下的Operator级别的告警。
3. 对于“数据倾斜”和“估算不准”两种类型告警，在某一个plan树结构下，只上报下层节点的告警，上层节点不再重复告警。这主要是因为这两种类型的告警可能是因为底层触发上层的。例如，如果在scan节点已经存在数据倾斜，那么在上层的hashagg等其他算子很可能也出现数据倾斜。

6.6.2 语句下推调优

语句下推介绍

目前，GaussDB优化器在分布式框架下制定语句的执行策略时，有三种执行计划方式：生成下推语句计划、生成分布式执行计划、生成发送语句的分布式执行计划。

- 下推语句计划：指直接将完整的查询语句从CN发送到DN进行执行，然后将执行结果返回给CN。
- 分布式执行计划：指CN对查询语句进行编译和优化，生成计划树，再将计划树发送给DN进行执行，并在执行完毕后返回结果到CN。
- 发送语句的分布式执行计划：上述两种方式都不可行时，将可下推的查询部分组成查询语句（多为基表扫描语句）下推到DN进行执行，获取中间结果到CN，然后在CN执行剩下的部分。

在第3种策略中，要将大量中间结果从DN发送到CN，并且要在CN运行不能下推的部分语句，会导致CN成为性能瓶颈（带宽、存储、计算等）。在进行性能调优的时候，应尽量避免只能选择第3种策略的查询语句。

执行语句不能下推是因为语句中含有**不支持下推的函数**或者**不支持下推的语法**。一般都可以通过等价改写规避执行计划不能下推的问题。

语句下推典型场景

在GaussDB优化器中如果想要支持语句下推需要将GUC参数enable_fast_query_shipping设置为on即可。通常而言explain语句后没有显示具体的执行计划算子，仅存在类似关键字“Data Node Scan on”出现在第一行（不包括计划格式）则说明语句已下推给DN去执行。下面从三个维度场景介绍下语句下推以及其支持的范围。

1 单表查询语句下推

在分布式数据库中对于单表查询而言，当前语句是否可以下推需要判断CN是否要进一步参与计算而不是简单收集数据。如果CN要进一步对DN结果进行计算则语句不可下推。通常带有agg, windows function, limit/offset, sort, distinct等关键字都不可下推。

- 可下推：简单查询，无需在CN进一步计算则可以下推。
```
openGauss=# explain select * from t where c1 > 1;
                              QUERY PLAN
------------------------------------------------------------------------
 Data Node Scan on "__REMOTE_FQS_QUERY__"  (cost=0.00..0.00 rows=0 width=0)
   Node/s: All datanodes
(2 rows)
```
- 不可下推：带有limit子句，对于CN而言不能简单发语句给DN并收集数据，明显与limit语义不符。
```
openGauss=# explain select * from t limit 1;
                                 QUERY PLAN
-----------------------------------------------------------------------------------------
 Limit  (cost=0.00..0.00 rows=1 width=12)
   ->  Data Node Scan on "__REMOTE_LIMIT_QUERY__"  (cost=0.00..0.00 rows=1 width=12)
         Node/s: All datanodes
(3 rows)
```
- 不可下推：带有聚集函数CN不能简单下推语句，而应该对从DN收集结果进一步聚集运算处理。
```
openGauss=# explain select sum(c1), count(*) from t;
                                 QUERY PLAN
-----------------------------------------------------------------------------------------
 Aggregate  (cost=0.10..0.11 rows=1 width=20)
   ->  Data Node Scan on "__REMOTE_GROUP_QUERY__"  (cost=0.00..0.00 rows=20 width=4)
         Node/s: All datanodes
(3 rows)
```
- 删除两个hash分布表。
```
openGauss=# DROP TABLE t;
DROP TABLE
openGauss=# DROP TABLE t1;
DROP TABLE
```

2 多表查询语句下推

多表查询场景下语句能否下推通常与join条件以及分布列有关，即如果join条件与表分布列匹配得上则可下推，否则无法下推。对于复制表来说通常可以下推。

- 创建两个hash分布表。
```
openGauss=# create table t(c1 int, c2 int, c3 int)distribute by hash(c1);
CREATE TABLE
openGauss=# create table t1(c1 int, c2 int, c3 int)distribute by hash(c1);
CREATE TABLE
```
- 可下推：join条件满足两个表hash分布列属性。
```
openGauss=# explain select * from t1 join t on t.c1 = t1.c1;
                              QUERY PLAN
------------------------------------------------------------------------
 Data Node Scan on "__REMOTE_FQS_QUERY__"  (cost=0.00..0.00 rows=0 width=0)
   Node/s: All datanodes
(2 rows)
```
- 不可下推：join条件不满足hash分布列属性，即t1.c2不是t1表的分布列。
```
openGauss=# explain select * from t1 join t on t.c1 = t1.c2;
                                   QUERY PLAN
---------------------------------------------------------------------------------------------
 Hash Join  (cost=0.25..0.53 rows=20 width=24)
   Hash Cond: (t1.c2 = t.c1)
   ->  Data Node Scan on t1 "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00 rows=20 width=12)
         Node/s: All datanodes
   ->  Hash  (cost=0.00..0.00 rows=20 width=12)
         ->  Data Node Scan on t "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00 rows=20 width=12)
```



```
            Node/s: All datanodes
(7 rows)
```

3 特殊场景

对于有一些特殊场景通常无法下推。

查看执行计划是否下推

执行计划是否下推可以依靠如下方法快速判断：

步骤1 将GUC参数enable_fast_query_shipping设置为off，使查询优化器使用分布式框架策略。

```
SET enable_fast_query_shipping = off;
```

步骤2 查看执行计划。

如果执行计划中有Data Node Scan节点，那么此执行计划是发送语句的分布式执行计划，为不可下推的执行计划；如果执行计划中有Streaming节点，那么计划是可以下推的。

例如如下业务SQL：

```
openGauss=# explain select
count(ss.ss_sold_date_sk order by ss.ss_sold_date_sk)c1
from store_sales ss, store_returns sr
where
sr.sr_customer_sk = ss.ss_customer_sk;
```

执行计划如下，可以看出此SQL语句不能下推。

```
                            QUERY PLAN
------------------------------------------------------------------------------
Aggregate
->  Hash Join
Hash Cond: (ss.ss_customer_sk = sr.sr_customer_sk)
->  Data Node Scan on store_sales "_REMOTE_TABLE_QUERY_"
Node/s: All datanodes
->  Hash
->  Data Node Scan on store_returns "_REMOTE_TABLE_QUERY_"
Node/s: All datanodes
(8 rows)
```

----结束

不支持下推的语法

以如下三个表定义说明不支持下推的SQL语法。

```
openGauss=# CREATE TABLE CUSTOMER1
(
    C_CUSTKEY     BIGINT NOT NULL
  , C_NAME        VARCHAR(25) NOT NULL
  , C_ADDRESS     VARCHAR(40) NOT NULL
  , C_NATIONKEY   INT NOT NULL
  , C_PHONE       CHAR(15) NOT NULL
  , C_ACCTBAL     DECIMAL(15,2)   NOT NULL
  , C_MKTSEGMENT  CHAR(10) NOT NULL
  , C_COMMENT     VARCHAR(117) NOT NULL
)
DISTRIBUTE BY hash(C_CUSTKEY);
openGauss=# CREATE TABLE test_stream(a int,b float); --float不支持重分布
openGauss=# CREATE TABLE sal_emp ( c1 integer[] ) DISTRIBUTE BY replication;
```


- 不支持returning语句下推

```
openGauss=# explain update customer1 set C_NAME = 'a' returning c_name;
                            QUERY PLAN
------------------------------------------------------------------
 Update on customer1  (cost=0.00..0.00 rows=30 width=187)
   Node/s: All datanodes
   Node expr: c_custkey
   ->  Data Node Scan on customer1 "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00 rows=30 width=187)
         Node/s: All datanodes
(5 rows)
```

- 不支持聚集函数中使用order by语句的下推

```
openGauss=# explain verbose select count ( c_custkey order by c_custkey) from customer1;

                        QUERY PLAN
------------------------------------------------------------------- Aggregate  (cost=2.50..2.51 rows=1 width=8)
   Output: count(customer1.c_custkey ORDER BY customer1.c_custkey)
   ->  Data Node Scan on customer1 "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00 rows=30 width=8)
         Output: customer1.c_custkey
         Node/s: All datanodes
         Remote query: SELECT c_custkey FROM ONLY public.customer1 WHERE true
(6 rows)
```

- count(distinct expr)中的字段不支持重分布，则不支持下推

```
openGauss=# explain verbose select count(distinct b) from test_stream;
                                  QUERY PLAN
------------------------------------------------------------------- Aggregate  (cost=2.50..2.51 rows=1 width=8)
   Output: count(DISTINCT test_stream.b)
   ->  Data Node Scan on test_stream "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00 rows=30 width=8)
         Output: test_stream.b
         Node/s: All datanodes
         Remote query: SELECT b FROM ONLY public.test_stream WHERE true
(6 rows)
```

- 不支持distinct on用法下推

```
openGauss=# explain verbose select distinct on (c_custkey) c_custkey from customer1 order by
c_custkey;
                                   QUERY PLAN
------------------------------------------------------------------- Unique  (cost=49.83..54.83 rows=30 width=8)
   Output: customer1.c_custkey
   ->  Sort  (cost=49.83..52.33 rows=30 width=8)
         Output: customer1.c_custkey
         Sort Key: customer1.c_custkey
         ->  Data Node Scan on customer1 "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00 rows=30
width=8)
               Output: customer1.c_custkey
               Node/s: All datanodes
               Remote query: SELECT c_custkey FROM ONLY public.customer1 WHERE true
(9 rows)
```

- 不支持数组表达式下推

```
openGauss=# explain verbose select array[c_custkey,1] from customer1 order by c_custkey;

                        QUERY PLAN
------------------------------------------------------------------- Sort  (cost=49.83..52.33 rows=30 width=8)
   Output: (ARRAY[customer1.c_custkey, 1::bigint]), customer1.c_custkey
   Sort Key: customer1.c_custkey
   ->  Data Node Scan on "__REMOTE_SORT_QUERY__"  (cost=0.00..0.00 rows=30 width=8)
         Output: (ARRAY[customer1.c_custkey, 1::bigint]), customer1.c_custkey
         Node/s: All datanodes
         Remote query: SELECT ARRAY[c_custkey, 1::bigint], c_custkey FROM ONLY public.customer1
WHERE true ORDER BY 2
(7 rows)
```

- With Recursive当前版本不支持下推的场景和原因如下：

序号	场景	不下推原因
1	包含外表的查询场景	LOG: SQL can't be shipped, reason: RecursiveUnion contains ForeignScan is not shippable（LOG为CN日志中打印的不下推原因，下同） 外表，当前版本暂不支持下推。
2	多nodegroup场景	LOG: SQL can't be shipped, reason: With-Recursive under multi-nodegroup scenario is not shippable 基表存储nodegroup不相同，或者计算nodegroup与基表不相同，当前版本暂不支持下推。

3	UNION不带ALL，需要去重	LOG: SQL can't be shipped, reason: With-Recursive does not contain "ALL" to bind recursive & none-recursive branches
		例如:
		`WITH recursive t_result AS (`
		`SELECT dm,sj_dm,name,1 as level`
		`FROM test_rec_part`
		`WHERE sj_dm > 10`
		`UNION`
		`SELECT t2.dm,t2.sj_dm,t2.name||' > '||`
		`t1.name,t1.level+1`
		`FROM t_result t1`
		`JOIN test_rec_part t2 ON t2.sj_dm = t1.dm`
		`)`
		`SELECT * FROM t_result t;`

4	基表中有系统表	LOG: SQL can't be shipped, reason: With-Recursive contains system table is not shippable
		例如：
		`WITH RECURSIVE x(id) AS`
		`(`
		`select count(1) from pg_class where oid=1247`
		`UNION ALL`
		`SELECT id+1 FROM x WHERE id < 5`
		`), y(id) AS`
		`(`
		`select count(1) from pg_class where oid=1247`
		`UNION ALL`
		`SELECT id+1 FROM x WHERE id < 10`
		`)`
		`SELECT y.*, x.* FROM y LEFT JOIN x USING (id) ORDER BY 1;`

序号	场景	不下推原因
5	基表扫描只有VALUES子句，仅在CN上即可完成执行。	LOG: SQL can't be shipped, reason: With-Recursive contains only values rte is not shippable 例如： WITH RECURSIVE t(n) AS (VALUES (1) UNION ALL SELECT n+1 FROM t WHERE n < 100) SELECT sum(n) FROM t;
6	相关子查询的关联条件仅在递归部分，非递归部分无关联条件。	LOG: SQL can't be shipped, reason: With-Recursive recursive term correlated only is not shippable 例如： select a.ID,a.Name, (with recursive cte as (select ID, PID, NAME from b where b.ID = 1 union all select parent.ID,parent.PID,parent.NAME from cte as child join b as parent on child.pid=parent.id where child.ID = a.ID) select NAME from cte limit 1) cName from (select id, name, count(*) as cnt from a group by id,name) a order by 1,2;
7	非递归部分带limit为Replicate计划，递归部分为 Hash计划，计划存在冲突。	LOG: SQL can't be shipped, reason: With-Recursive contains conflict distribution in none-recursive(Replicate) recursive(Hash) 例如： WITH recursive t_result AS (select * from(SELECT dm,sj_dm,name,1 as level FROM test_rec_part WHERE sj_dm < 10 order by dm limit 6 offset 2) UNION all SELECT t2.dm,t2.sj_dm,t2.name||' > '|| t1.name,t1.level+1 FROM t_result t1 JOIN test_rec_part t2 ON t2.sj_dm = t1.dm) SELECT * FROM t_result t;

序号	场景	不下推原因
8	多层Recursive嵌套，即recursive的递归部分又嵌套另一个recursive查询。	LOG: SQL can't be shipped, reason: Recursive CTE references recursive CTE "cte" 例如： with recursive cte as (select * from rec_tb4 where id<4 union all select h.id,h.parentID,h.name from (with recursive cte as (select * from rec_tb4 where id<4 union all select h.id,h.parentID,h.name from rec_tb4 h inner join cte c on h.id=c.parentID) SELECT id ,parentID,name from cte order by parentID) h inner join cte c on h.id=c.parentID) SELECT id ,parentID,name from cte order by parentID,1,2,3;

删除表：

```
openGauss=# DROP TABLE CUSTOMER1;
DROP TABLE
openGauss=# DROP TABLE test_stream;
DROP TABLE
openGauss=# DROP TABLE sal_emp;
DROP TABLE
```

不支持下推的函数

首先介绍函数的易变性。在GaussDB中共分三种形态：

- **IMMUTABLE**

  表示该函数在给出同样的参数值时总是返回同样的结果。

- **STABLE**

  表示该函数不能修改数据库，对相同参数值，在同一次表扫描里，该函数的返回值不变，但是返回值可能在不同SQL语句之间变化。

- **VOLATILE**

  表示该函数值可以在一次表扫描内改变，因此不会做任何优化。

函数易变性可以查询pg_proc的provolatile字段获得，i代表IMMUTABLE，s代表STABLE，v代表VOLATILE。另外，在pg_proc中的proshippable字段，取值范围为t/f/NULL，这个字段与provolatile字段一起用于描述函数是否下推。

- 如果函数的provolatile属性为i，则无论proshippable的值是否为t，则函数始终可以下推。
- 如果函数的provolatile属性为s或v，则仅当proshippable的值为t时，函数可以下推。

- random，exec_hadoop_sql，exec_on_extension如果出现CTE中，也不下推。因为这种场景下下推可能出现结果错误。

对于用户自定义函数，可以在创建函数的时候指定provolatile和proshippable属性的值，详细请参考**CREATE FUNCTION**。

对于函数不能下推的场景：

- 如果是系统函数，建议根据业务等价替换这个函数。
- 如果是自定义函数，建议分析客户业务场景，看函数的provolatile和proshippable属性定义是否正确。

实例分析：自定义函数

对于自定义函数，如果对于确定的输入，有确定的输出，则应将函数定义为immutable类型。

利用TPCDS的销售信息举个例子，比如写一个函数，获取商品的打折情况，需要一个计算折扣的函数，可以将这个函数定义为：

```
CREATE FUNCTION func_percent_2 (NUMERIC, NUMERIC) RETURNS NUMERIC
AS 'SELECT $1 / $2 WHERE $2 > 0.01'
LANGUAGE SQL
VOLATILE;
```

执行下列语句：

```
SELECT func_percent_2(ss_sales_price, ss_list_price)
FROM store_sales;
```

其执行计划为：

```
 Data Node Scan on store_sales "_REMOTE_TABLE_QUERY_"
   Output: func_percent_2(store_sales.ss_sales_price, store_sales.ss_list_price)
   Remote query: SELECT ss_sales_price, ss_list_price FROM ONLY store_sales WHERE true
(3 rows)
```

可见，func_percent_2并没有被下推，而是将ss_sales_price和ss_list_price收到CN上，再进行计算，消耗大量CN的资源，而且计算缓慢。

由于该自定义函数对确定的输入有确定的输出，如果将该自定义函数改为：

```
CREATE FUNCTION func_percent_1 (NUMERIC, NUMERIC) RETURNS NUMERIC
AS 'SELECT $1 / $2 WHERE $2 > 0.01'
LANGUAGE SQL
IMMUTABLE;
```

执行语句：

```
SELECT func_percent_1(ss_sales_price, ss_list_price)
FROM store_sales;
```

其执行计划为：

```
 Data Node Scan
   Output: (func_percent_1(store_sales.ss_sales_price, store_sales.ss_list_price))
   Remote query: SELECT func_percent_1(ss_sales_price, ss_list_price) AS func_percent_1 FROM store_sales
(3 rows)
```

可见函数func_percent_1被下推到DN执行。

实例分析 2：使排序下推

请参考**案例：使排序下推**。

6.6.3 子查询调优

子查询背景介绍

应用程序通过SQL语句来操作数据库时会使用大量的子查询，这种写法比直接对两个表做连接操作在结构上和思路上更清晰，尤其是在一些比较复杂的查询语句中，子查询有更完整、更独立的语义，会使SQL对业务逻辑的表达更清晰更容易理解，因此得到了广泛的应用。

GaussDB根据子查询在SQL语句中的位置把子查询分成了子查询、子链接两种形式。

- 子查询SubQuery：对应于查询解析树中的范围表RangeTblEntry，更通俗一些指的是出现在FROM语句后面的独立的SELECT语句。
- 子链接SubLink：对应于查询解析树中的表达式，更通俗一些指的是出现在where/on子句、targetlist里面的语句。

  综上，对于查询解析树而言，SubQuery的本质是范围表、而SubLink的本质是表达式。针对SubLink场景而言，由于SubLink可以出现在约束条件、表达式中，按照GaussDB对sublink的实现，sublink可以分为以下几类：
  - exist_sublink：对应EXIST、NOT EXIST语句
  - any_sublink：对应op ANY(select…)语句，其中OP可以是<,>,=操作符，另外IN/NOT IN (select ...)也属于这一类。
  - all_sublink：对应op ALL(select…)语句，其中OP可以是<,>,=操作符
  - rowcompare_sublink：对应record op (select …)语句
  - expr_sublink：对应(SELECT with single targetlist item ...)语句
  - array_sublink：对应ARRAY(select…)语句
  - cte_sublink：对应with query(…)语句

  其中的sublink为exist_sublink、any_sublink，在GaussDB的优化引擎中对其应用场景做了优化（子链接提升）。另外，expr_sublink 也可以提升，但是由于SQL语句中子查询的使用的灵活性，会带来SQL子查询过于复杂造成性能问题。如果希望关闭 expr_sublink 的提升优化，可以通过 guc 参数 rewrite_rule 来设置。子查询从大类上来看，分为非相关子查询和相关子查询：
  - **非相关子查询None-Correlated SubQuery**

    子查询的执行不依赖于外层父查询的任何属性值。这样子查询具有独立性，可独自求解，形成一个子查询计划先于外层的查询求解。

    例如：

```
openGauss=# explain select t1.c1,t1.c2
from t1
where t1.c1 in (
   select c2
   from t2
   where t2.c2 IN (2,3,4)
);
                          QUERY PLAN
------------------------------------------------------------------
Streaming (type: GATHER)
  Node/s: All datanodes
  ->  Hash Right Semi Join
        Hash Cond: (t2.c2 = t1.c1)
        ->  Streaming(type: REDISTRIBUTE)
              Spawn on: All datanodes
              ->  Seq Scan on t2
                    Filter: (c2 = ANY ('{2,3,4}'::integer[]))
        ->  Hash
```



```
            ->  Seq Scan on t1
(10 rows)
```

- **相关子查询Correlated-SubQuery**

  子查询的执行依赖于外层父查询的一些属性值（如下列示例t2.c1 = t1.c1条件中的t1.c1）作为内层查询的一个与条件。这样的子查询不具备独立性，需要和外层查询按分组进行求解。

  例如：

```
openGauss=# explain select t1.c1,t1.c2
from t1
where t1.c1 in (
   select c2
   from t2
   where t2.c1 = t1.c1 AND t2.c2 in (2,3,4)
);
                              QUERY PLAN
-----------------------------------------------------------------------
Streaming (type: GATHER)
  Node/s: All datanodes
  ->  Seq Scan on t1
        Filter: (SubPlan 1)
        SubPlan 1
          ->  Result
                Filter: (t2.c1 = t1.c1)
                ->  Materialize
                      ->  Streaming(type: BROADCAST)
                          Spawn on: All datanodes
                          ->  Seq Scan on t2
                                Filter: (c2 = ANY ('{2,3,4}'::integer[]))
(12 rows)
```

GaussDB 对 SubLink 的优化

针对SubLink的优化策略主要是让内层的子查询提升(pullup)，能够和外表直接做关联查询，从而避免生成SubPlan+Broadcast内表的执行计划。判断子查询是否存在性能风险，可以通过explain查询语句查看Sublink的部分是否被转换成SubPlan+Broadcast的执行计划。

例如：

```
gaussdb=# EXPLAIN SELECT t1.c1, t1.c2 FROM t1 WHERE t1.c1 IN(SELECT c2 FROM t2 WHERE t2.c1 =
t1.c1);
          QUERY PLAN
--------------------------------
 Seq Scan on t1
   Filter: (SubPlan 1)
   SubPlan 1
     ->  Seq Scan on t2
           Filter: (c1 = t1.c1)
(5 rows)
```

- **目前GaussDB支持的Sublink-Release场景**
  - IN-Sublink无相关条件
    - 不能包含上一层查询的表中的列（可以包含更高层查询表中的列）。
    - 不能包含易变函数。

    例如：

```
gaussdb=# EXPLAIN SELECT t1.c1, t1.c2 FROM t1 WHERE t1.c1 IN(SELECT c2 FROM t2 WHERE
t2.c1 = 1);
             QUERY PLAN
-------------------------------------
```



```
 Hash Join
   Hash Cond: (t1.c1 = t2.c2)
   ->  Seq Scan on t1
   ->  Hash
         ->  HashAggregate
               Group By Key: t2.c2
               ->  Seq Scan on t2
                     Filter: (c1 = 1)
(8 rows)
```

- Exist-Sublink包含相关条件

  Where子句中必须包含上一层查询的表中的列，子查询的其它部分不能含有上层查询的表中的列。其它限制如下。

  - 子查询必须有from子句。
  - 子查询不能含有with子句。
  - 子查询不能含有聚集函数。
  - 子查询里不能包含集合操作、排序、limit、windowagg、having操作。
  - 不能包含易变函数。

  例如下面查询语句：

```
gaussdb=# EXPLAIN (COSTS OFF) SELECT t1.c1, t1.c2 FROM t1 WHERE exists (SELECT c2 FROM
t2 WHERE t2.c1 = t1.c1);
         QUERY PLAN
------------------------------
 Hash Semi Join
   Hash Cond: (t1.c1 = t2.c1)
   ->  Seq Scan on t1
   ->  Hash
         ->  Seq Scan on t2
(5 rows)
```

  如上打印的执行计划应替换成下面的执行计划：

```
           QUERY PLAN
-----------------------------------
 Hash Join
   Hash Cond: (t1.c1 = t2.c1)
   ->  Seq Scan on t1
   ->  Hash
         ->  HashAggregate
               Group By Key: t2.c1
               ->  Seq Scan on t2
(7 rows)
```

- 包含聚集函数的等值相关子查询的提升

  子查询的where条件中必须含有来自上一层的列，而且此列必须和子查询本层涉及表中的列做相等判断，且这些条件必须用and连接。其它地方不能包含上层的列。其它限制条件如下。

  - 子查询中where条件包含的表达式(列名)必须是表中的列。
  - 子查询的Select关键字后，必须有且仅有一个输出列，此输出列必须是聚集函数(如max)，并且聚集函数的参数(t2.c2)不能是来自外层表(t1)中的列。聚集函数不能是count。

    例如，下列示例可以提升。

```
select * from t1 where c1 >(
      select max(t2.c1) from t2 where t2.c1=t1.c1
);
```

    下列示例不能提升，因为子查询没有聚集函数。


```
select * from t1 where c1 >(
      select  t2.c1 from t2 where t2.c1=t1.c1
);
```

下列示例不能提升，因为子查询有两个输出列。

```
select * from t1 where (c1,c2) >(
      select  max(t2.c1),min(t2.c2) from t2 where t2.c1=t1.c1
);
```

- 子查询必须是from子句。
- 子查询中不能有groupby、having、集合操作。
- 子查询只能是inner join。

  例如：下列示例不能提升。

  ```
  select * from t1 where c1 >(
        select max(t2.c1) from t2 full join t3 on (t2.c2=t3.c2) where t2.c1=t1.c1
  );
  ```

- 子查询的targetlist中不能包含返回set的函数。
- 子查询的where条件中必须含有来自上一层的列，而且此列必须和子查询层涉及表中的列做相等判断，且这些条件必须用and连接。其它地方不能包含上层的上层中的列。例如：下列示例中的最内层子链接可以提升。

  ```
  select * from t3 where t3.c1=(
        select t1.c1
        from t1 where c1 >(
              select max(t2.c1) from t2 where t2.c1=t1.c1
  ));
  ```

  基于上面的示例，再加一个条件，则不能提升，因为最内侧子查询引用了上上层中的列。示例如下：

  ```
  select * from t3 where t3.c1=(
        select t1.c1
        from t1 where c1 >(
              select max(t2.c1) from t2 where t2.c1=t1.c1 and t3.c1>t2.c2

  ));
  ```

– 提升OR子句中的SubLink

  当WHERE过滤条件中有OR连接的EXIST相关SubLink，

  例如：

  ```
  select a, c from t1
  where t1.a = (select avg(a) from t3 where t1.b = t3.b) or
  exists (select * from t4 where t1.c = t4.c);
  ```

  将或条件连接的EXIST相关子查询OR子句的提升过程：

  i. 提取where条件中，or子句中的opExpr。为：t1.a = (select avg(a) from t3 where t1.b = t3.b)

  ii. 这个op操作中包含subquery，判断是否可以提升，如果可以提升，重写subquery为：select avg(a), t3.b from t3 group by t3.b，生成not null条件t3.b is not null，并将这个opexpr用这个not null条件替换。此时SQL变为：

  ```
  select a, c
  from t1 left join (select avg(a) avg, t3.b from t3 group by t3.b)  as t3 on (t1.a = avg and t1.b = t3.b)
  where t3.b is not null or exists (select * from t4 where t1.c = t4.c);
  ```

  iii. 再次提取or子句中的exists sublink，exists (select * from t4 where t1.c = t4.c)，判断是否可以提升，如果可以提升，转换subquery为：select

t4.c from t4 group by t4.c生成NotNull条件t4.c is not null提升查询，SQL变为：

```
select t1.a, t1.c from t1 left join (select avg(a) avg, t3.b from t3 group by t3.b) as t3 on (t1.a = avg and t1.b = t3.b) left join (select t5.c from t5 group by t5.c) as t5 on (t1.c = t5.c) where t3.b is not null or t5.c is not null;
```

- **目前GaussDB不支持的Sublink-Release场景**

  除了以上场景之外都不支持Sublink提升，因此关联子查询会被计划成SubPlan +Broadcast的执行计划，当inner表的数据量较大时则会产生性能风险。

  如果相关子查询中跟外层的两张表做join，那么无法提升该子查询，需要通过将父SQL创建成with子句，然后再跟子查询中的表做相关子查询查询。

  例如：

  ```
  select distinct t1.a, t2.a
  from t1 left join t2 on t1.a=t2.a and not exists (select a,b from test1 where test1.a=t1.a and test1.b=t2.a);
  ```

  改写为

  ```
  with temp as
  (
        select * from (select t1.a as a, t2.a as b from t1 left join t2 on t1.a=t2.a)

  )
  select distinct a,b
  from temp
  where not exists (select a,b from test1 where temp.a=test1.a and temp.b=test1.b);
  ```

  - 出现在targetlist里的相关子查询无法提升(不含count)

    例如：

    ```
    openGauss=# explain (costs off)
    select (select c2 from t2 where t1.c1 = t2.c1) ssq, t1.c2
    from t1
    where t1.c2 > 10;
    ```

    执行计划为：

    ```
    openGauss=# explain (costs off)
    select (select c2 from t2 where t1.c1 = t2.c1) ssq, t1.c2
    from t1
    where t1.c2 > 10;
                      QUERY PLAN
    -----------------------------------------------------
     Streaming (type: GATHER)
       Node/s: All datanodes
       ->  Seq Scan on t1
             Filter: (c2 > 10)
             SubPlan 1
               ->  Result
                     Filter: (t1.c1 = t2.c1)
                     ->  Materialize
                           ->  Streaming(type: BROADCAST)
                                 Spawn on: All datanodes
                                 ->  Seq Scan on t2
    (11 rows)
    ```

    由于相关子查询出现在targetlist(查询返回列表)里，对于t1.c1=t2.c1不匹配的场景仍然需要输出值，因此使用right-outerjoin关联t2&t1，以确保t1.c1=t2.c1在不匹配时，子SSQ能够返回不匹配的补空值。

    说明

    SSQ和CSSQ的解释如下：

    - SSQ：ScalarSubQuery一般指返回1行1列scalar值的sublink，简称SSQ。
    - CSSQ：Correlated-ScalarSubQuery和SSQ相同不过是指包含相关条件的SSQ。

    上述SQL语句可以改写为：


```
with ssq as
(
   select * from t1 where t1.c2 > 10
)
select t2.c2,ssq.c2
from t2 right join ssq on ssq.c1 = t2.c1;
```

改写后的执行计划为：

```
            QUERY PLAN
---------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Hash Right Join
         Hash Cond: (t2.c1 = t1.c1)
         ->  Seq Scan on t2
         ->  Hash
               ->  Seq Scan on t1
                     Filter: (c2 > 10)
(8 rows)
```

可以看到出现在SSQ返回列表里的相关子查询SSQ，已经被提升成Right Join，从而避免当内表t2较大时出现SubPlan+Broadcast计划导致性能变差。

- 出现在targetlist里的相关子查询无法提升(带count)

  例如：

```
select (select count(*) from t2 where t2.c1=t1.c1) cnt, t1.c1, t3.c1
from t1,t3
where t1.c1=t3.c1 order by cnt, t1.c1;
```

  执行计划为

```
                          QUERY PLAN
------------------------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Sort
         Sort Key: ((SubPlan 1)), t1.c1
         ->  Hash Join
               Hash Cond: (t1.c1 = t3.c1)
               ->  Seq Scan on t1
               ->  Hash
                     ->  Seq Scan on t3
               SubPlan 1
                 ->  Aggregate
                       ->  Result
                             Filter: (t2.c1 = t1.c1)
                             ->  Materialize
                                   ->  Streaming(type: BROADCAST)
                                         Spawn on: All datanodes
                                         ->  Seq Scan on t2
(17 rows)
```

由于相关子查询出现在targetlist(查询返回列表)里，对于t1.c1=t2.c1不匹配的场景仍然需要输出值，因此使用left-outerjoin关联T1&T2确保t1.c1=t2.c1在不匹配时子SSQ能够返回不匹配的补空值，但是这里带了count语句及时在t1.c1=t2.t1不匹配时需要输出0，因此可以使用一个case-when NULL then 0 else count(*)来代替。

上述SQL语句可以改写为：

```
with ssq as
(
   select count(*) cnt, c1 from t2 group by c1
)
select case when
          ssq.cnt is null then 0
          else ssq.cnt
      end cnt, t1.c1, t3.c1
from t1 left join ssq on ssq.c1 = t1.c1,t3
```



```
where t1.c1 = t3.c1
order by ssq.cnt, t1.c1;
```

改写后的执行计划为

```
 QUERY PLAN
-----------------------------------------------------
Streaming (type: GATHER)
  Node/s: All datanodes
  ->  Sort
        Sort Key: (count(*)), t1.c1
        ->  Hash Join
              Hash Cond: (t1.c1 = t3.c1)
              ->  Hash Left Join
                    Hash Cond: (t1.c1 = t2.c1)
                    ->  Seq Scan on t1
                    ->  Hash
                          ->  HashAggregate
                                Group By Key: t2.c1
                                ->  Seq Scan on t2
              ->  Hash
                    ->  Seq Scan on t3
(15 rows)
```

- 相关条件为不等值场景

  例如：

  ```
  select t1.c1, t1.c2
  from t1
  where t1.c1 = (select agg() from t2.c2 > t1.c2);
  ```

  对于非等值相关条件的SubLink目前无法提升，从语义上可以通过做2次join（一次CorrelationKey，一次rownum自关联）达到提升改写的目的。

  改写方案有两种。

  - 子查询改写方式

    ```
    select t1.c1, t1.c2
    from t1, (
       select t1.rowid, agg() aggref
       from t1,t2
       where t1.c2 > t2.c2 group by t1.rowid
    ) dt /* derived table */
    where t1.rowid = dt.rowid AND t1.c1 = dt.aggref;
    ```

  - CTE改写方式

    ```
    WITH dt as
    (
       select t1.rowid, agg() aggref
       from t1,t2
       where t1.c2 > t2.c2 group by t1.rowid
    )
    select t1.c1, t1.c2
    from t1, derived_table
    where t1.rowid = derived_table.rowid AND
    t1.c1 = derived_table.aggref;
    ```


须知

- 目前GaussDB尚无高效的实现表、中间结果集的全局唯一rowid，因此目前此类场景很难改写，建议通过业务层进行规避，或者可以使用t1.xc_nodeid + t1.ctid进行rowid关联，但是xc_nodeid的重复率较高会导致join关联效率变低，而xc_node_id+ctid类型无法作为hashjoin的关联条件。
- 对于AGG类型为count(*)时需要进行CASE-WHEN对没有match的场景补0处理，非COUNT(*)场景NULL处理。
- CTE改写方式如果有sharescan支持性能上能够更优。

更多优化示例

示例1：修改基表为replication表，并且在过滤列上创建索引。

```
create table master_table (a int);
create table sub_table(a int, b int);
select a from master_table group by a having a in (select a from sub_table);
```

上述事例中存在一个相关性子查询，为了提升查询的性能，建表时，可以将sub_table修改为一个replication表，并且在字段a上创建一个index。

示例2：修改select语句，将子查询修改为和主表的join，或者修改为可以提升的subquery，但是在修改前后需要保证语义的正确性。

```
openGauss=# explain (costs off)select * from master_table as t1 where t1.a in (select t2.a from sub_table as t2 where t1.a = t2.b);
                      QUERY PLAN
------------------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Seq Scan on master_table t1
         Filter: (SubPlan 1)
         SubPlan 1
           ->  Result
                 Filter: (t1.a = t2.b)
                 ->  Materialize
                       ->  Streaming(type: BROADCAST)
                             Spawn on: All datanodes
                             ->  Seq Scan on sub_table t2
(11 rows)
```

上面事例计划中存在一个subPlan，为了消除这个subPlan可以修改语句为：

```
openGauss=# explain(costs off) select * from master_table as t1 where exists (select t2.a from sub_table as t2 where t1.a = t2.b and t1.a = t2.a);
                   QUERY PLAN
-------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Hash Semi Join
         Hash Cond: (t1.a = t2.b)
         ->  Seq Scan on master_table t1
         ->  Hash
               ->  Streaming(type: REDISTRIBUTE)
                     Spawn on: All datanodes
                     ->  Seq Scan on sub_table t2
(9 rows)
```

从计划可以看出，subPlan消除了，计划变成了两个表的semi join，这样会大大提高执行效率。

6.6.4 统计信息调优

统计信息调优介绍

GaussDB是基于代价估算生成的最优执行计划。优化器需要根据analyze收集的统计信息进行行数估算和代价估算，因此统计信息对优化器行数估算和代价估算起着至关重要的作用。通过analyze收集全局统计信息，主要包括：pg_class表中的relpages和reltuples；pg_statistic表中的stadistinct、stanullfrac、stanumbersN、stavaluesN、histogram_bounds等。

实例分析 1：未收集统计信息导致查询性能差

在很多场景下，由于查询中涉及到的表或列没有收集统计信息，会对查询性能有很大的影响。

表结构如下所示：

```
CREATE TABLE LINEITEM
(
L_ORDERKEY        BIGINT        NOT NULL
, L_PARTKEY       BIGINT        NOT NULL
, L_SUPPKEY       BIGINT        NOT NULL
, L_LINENUMBER    BIGINT        NOT NULL
, L_QUANTITY      DECIMAL(15,2) NOT NULL
, L_EXTENDEDPRICE  DECIMAL(15,2) NOT NULL
, L_DISCOUNT      DECIMAL(15,2) NOT NULL
, L_TAX           DECIMAL(15,2) NOT NULL
, L_RETURNFLAG    CHAR(1)       NOT NULL
, L_LINESTATUS    CHAR(1)       NOT NULL
, L_SHIPDATE      DATE          NOT NULL
, L_COMMITDATE    DATE          NOT NULL
, L_RECEIPTDATE   DATE          NOT NULL
, L_SHIPINSTRUCT  CHAR(25)      NOT NULL
, L_SHIPMODE      CHAR(10)      NOT NULL
, L_COMMENT       VARCHAR(44)   NOT NULL
) distribute by hash(L_ORDERKEY);

CREATE TABLE ORDERS
(
O_ORDERKEY       BIGINT        NOT NULL
, O_CUSTKEY      BIGINT        NOT NULL
, O_ORDERSTATUS  CHAR(1)       NOT NULL
, O_TOTALPRICE   DECIMAL(15,2) NOT NULL
, O_ORDERDATE    DATE NOT NULL
, O_ORDERPRIORITY CHAR(15)     NOT NULL
, O_CLERK        CHAR(15)      NOT NULL
, O_SHIPPRIORITY  BIGINT       NOT NULL
, O_COMMENT      VARCHAR(79)   NOT NULL
) distribute by hash(O_ORDERKEY);
```

查询语句如下所示：

```
explain verbose select
count(*) as numwait
from
lineitem l1,
orders
where
o_orderkey = l1.l_orderkey
and o_orderstatus = 'F'
and l1.l_receiptdate > l1.l_commitdate
and not exists (
select
*
from
```



```
lineitem l3
where
l3.l_orderkey = l1.l_orderkey
and l3.l_suppkey <> l1.l_suppkey
and l3.l_receiptdate > l3.l_commitdate
)
order by
numwait desc;
```

当出现该问题时，可以通过如下方法确认查询中涉及到的表或列有没有做过analyze收集统计信息。

1. 通过explain verbose执行query分析执行计划时会提示WARNING信息，如下所示：
   ```
   WARNING:Statistics in some tables or columns(public.lineitem.l_receiptdate,
   public.lineitem.l_commitdate, public.lineitem.l_orderkey, public.lineitem.l_suppkey,
   public.orders.o_orderstatus, public.orders.o_orderkey) are not collected.
   HINT:Do analyze for them in order to generate optimized plan.
   ```
2. 可以通过在pg_log目录下的日志文件中查找以下信息来确认当前执行的query是否由于没有收集统计信息导致查询性能变差。
   ```
   2017-06-14 17:28:30.336 CST 140644024579856 20971684 [BACKEND] LOG:Statistics in some tables
   or columns(public.lineitem.l_receiptdate, public.lineitem.l_commitdate, public.lineitem.l_orderkey,
   public.linei
   tem.l_suppkey, public.orders.o_orderstatus, public.orders.o_orderkey) are not collected.
   2017-06-14 17:28:30.336 CST 140644024579856 20971684 [BACKEND] HINT:Do analyze for them in
   order to generate optimized plan.
   ```

当通过以上方法查看到哪些表或列没有做analyze，可以通过对WARNING或日志中上报的表或列做analyze来解决由于未收集统计信息导致查询变慢的问题。

实例分析 2：设置 cost_param 对查询性能优化

请参考**案例：设置cost_param对查询性能优化**。

实例分析 3：多表 join 的复杂查询存在中间结果不准调优

现象描述：查询与指定人在前后15分钟内、同一网吧登记上网的人员信息：

```
SELECT
C.WBM,
C.DZQH,
C.DZ,
B.ZJHM,
B.SWKSSJ,
B.XWSJ
FROM
b_zyk_wbswxx A,
b_zyk_wbswxx B,
b_zyk_wbcs C
WHERE
A.ZJHM = '522522******3824'
AND A.WBDM = B.WBDM
AND A.WBDM = C.WBDM
AND abs(to_date(A.SWKSSJ,'yyyymmddHH24MISS') - to_date(B.SWKSSJ,'yyyymmddHH24MISS')) <
INTERVAL '15 MINUTES'
ORDER BY
B.SWKSSJ,
B.ZJHM
limit 10 offset 0
;
```

执行计划如**图6-6**所示。该查询实际耗时约12秒。

图 6-6 应用 unlogged table 案例（一）

```
                                   QUERY PLAN
--------------------------------------------------------------------------------------------------------------
 Limit  (cost=221021.41..221021.43 rows=10 width=120)
   ->  Sort  (cost=221021.41..221022.01 rows=240 width=120)
         Sort Key: b.swkssj, b.zjhm
         ->  Streaming (type: GATHER)  (cost=221015.62..221016.22 rows=240 width=120)
               Node/s: All datanodes
               ->  Limit  (cost=9208.98..9209.01 rows=10 width=120)
                     ->  Sort  (cost=9208.98..9211.60 rows=1048 width=120)
                           Sort Key: b.swkssj, b.zjhm
                           ->  Nested Loop  (cost=23.27..9186.34 rows=1048 width=120)
                                 Join Filter: (((a.zjhm)::text <> (b.zjhm)::text) AND ((a.wbdm)::text = (b.wbdm)::text)
                                 AND (abs(((to_date((a.swkssj)::text, 'yyyymmddHH24MISS'::text)
                                    - to_date((b.swkssj)::text, 'yyyymmddHH24MISS'::text)))::numeric) < .0104166666666667))
                                 ->  Streaming(type: BROADCAST)  (cost=0.00..6.33 rows=24 width=135)
                                       Spawn on: All datanodes
                                       ->  Nested Loop  (cost=0.00..106.80 rows=1 width=135)
                                             ->  Streaming(type: BROADCAST)  (cost=0.00..24.75 rows=264 width=48)
                                                   Spawn on: All datanodes
                                                   ->  Partition Iterator  (cost=0.00..48.44 rows=11 width=48)
                                                         Iterations: 25
                                                         ->  Partitioned Index Scan using idx_b_zyk_wbswxx_zjhm on b_zyk_wbswxx a  (cost=0.00..48.44 rows=11 width=48)
                                                               Index Cond: ((zjhm)::text = '522522******3824'::text)
                                                               Selected Partitions:  1..25
                                             ->  Index Scan using idx_b_zyk_wbcs_wbdm on b_zyk_wbcs c  (cost=0.00..2.82 rows=1 width=87)
                                                   Index Cond: ((wbdm)::text = (a.wbdm)::text)
                                 ->  Partition Iterator  (cost=23.27..7306.33 rows=2454 width=63)
                                       Iterations: 25
                                       ->  Partitioned Bitmap Heap Scan on b_zyk_wbswxx b  (cost=23.27..7306.33 rows=2454 width=63)
                                             Recheck Cond: ((wbdm)::text = (c.wbdm)::text)
                                             Filter: ('522522198405243824'::text <> (zjhm)::text)
                                             Selected Partitions:  1..25
                                             ->  Partitioned Bitmap Index Scan on idx_b_zyk_wbswxx_wbdm  (cost=0.00..22.65 rows=2454 width=0)
                                                   Index Cond: ((wbdm)::text = (c.wbdm)::text)
(30 rows)
```

优化分析：分析过程如下：

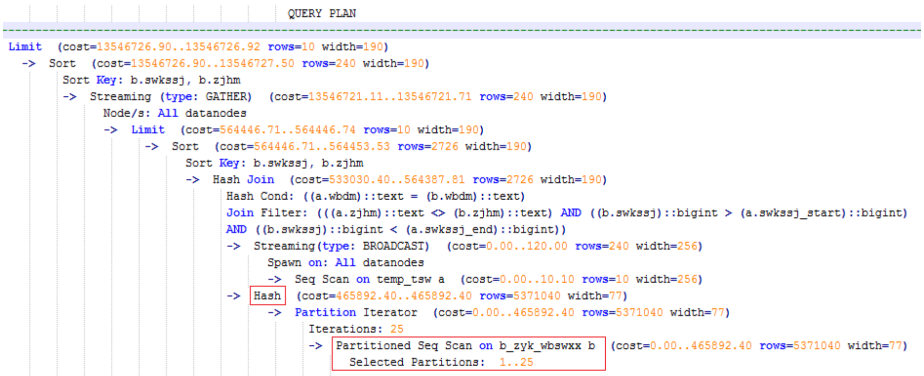
1. 分析该执行计划发现，扫描节点已使用Index Scan，耗时主要在最外层Nest Loop Join的Join Filter计算中，且该计算执行了字符串的加减法和不等值比较。
2. 考虑使用unlogged table保存目标人的上网信息，且在插入时处理上网开始时间和终止时间，以避免后续进行时间加减。

```
//创建临时unlogged table
CREATE UNLOGGED TABLE temp_tsw
(
ZJHM         NVARCHAR2(18),
WBDM          NVARCHAR2(14),
SWKSSJ_START NVARCHAR2(14),
SWKSSJ_END   NVARCHAR2(14),
WBM          NVARCHAR2(70),
DZQH         NVARCHAR2(6),
DZ          NVARCHAR2(70),
IPDZ         NVARCHAR2(39)
)
;
//插入目标人的上网记录，并处理上网开始和结束时间。
INSERT INTO
temp_tsw
SELECT
A.ZJHM,
A.WBDM,
to_char((to_date(A.SWKSSJ,'yyyymmddHH24MISS') - INTERVAL '15
MINUTES'),'yyyymmddHH24MISS'),
to_char((to_date(A.SWKSSJ,'yyyymmddHH24MISS') + INTERVAL '15
MINUTES'),'yyyymmddHH24MISS'),
B.WBM,B.DZQH,B.DZ,B.IPDZ
FROM
b_zyk_wbswxx A,
b_zyk_wbcs B
WHERE
A.ZJHM='522522******3824' AND A.WBDM = B.WBDM
;

//查询和目标人在前后十五分钟内在同一网吧上网的人员信息，比较大小时强制转换为int8。
SELECT
A.WBM,
A.DZQH,
A.DZ,
A.IPDZ,
```

```
B.ZJHM,
B.XM,
to_date(B.SWKSSJ,'yyyymmddHH24MISS') as SWKSSJ,
to_date(B.XWSJ,'yyyymmddHH24MISS') as XWSJ,
B.SWZDH
FROM temp_tsw A,
b_zyk_wbswxx B
WHERE
A.ZJHM <> B.ZJHM
AND A.WBDM = B.WBDM
AND (B.SWKSSJ)::int8 > (A.swkssj_start)::int8
AND (B.SWKSSJ)::int8 < (A.swkssj_end)::int8
order by
B.SWKSSJ,
B.ZJHM
limit 10 offset 0
;
```

上述查询耗时约7秒，执行计划如图6-7所示。

图 6-7 应用 unlogged table 案例（二）



```
QUERY PLAN
-----------------------------------------------------------------------------------------------------
Limit  (cost=13546726.90..13546726.92 rows=10 width=190)
  ->  Sort  (cost=13546726.90..13546727.50 rows=240 width=190)
        Sort Key: b.swkssj, b.zjhm
        ->  Streaming (type: GATHER)  (cost=13546721.11..13546721.71 rows=240 width=190)
              Node/s: All datanodes
              ->  Limit  (cost=564446.71..564446.74 rows=10 width=190)
                    ->  Sort  (cost=564446.71..564453.53 rows=2726 width=190)
                          Sort Key: b.swkssj, b.zjhm
                          ->  Hash Join  (cost=533030.40..564387.81 rows=2726 width=190)
                                Hash Cond: ((a.wbdm)::text = (b.wbdm)::text)
                                Join Filter: (((a.zjhm)::text <> (b.zjhm)::text) AND ((b.swkssj)::bigint > (a.swkssj_start)::bigint)
                                AND ((b.swkssj)::bigint < (a.swkssj_end)::bigint))
                                ->  Streaming(type: BROADCAST)  (cost=0.00..120.00 rows=240 width=256)
                                      Spawn on: All datanodes
                                      ->  Seq Scan on temp_tsw a  (cost=0.00..10.10 rows=10 width=256)
                                ->  Hash  (cost=465892.40..465892.40 rows=5371040 width=77)
                                      ->  Partition Iterator  (cost=0.00..465892.40 rows=5371040 width=77)
                                            Iterations: 25
                                            ->  Partitioned Seq Scan on b_zyk_wbswxx b  (cost=0.00..465892.40 rows=5371040 width=77)
                                                  Selected Partitions:  1..25
```

3. 分析上述执行计划，发现执行了Hash Join，对大表b_zyk_wbswxx（网吧上网信息）建立了Hash Table。由于该表数据量大，创建过程耗时较长。

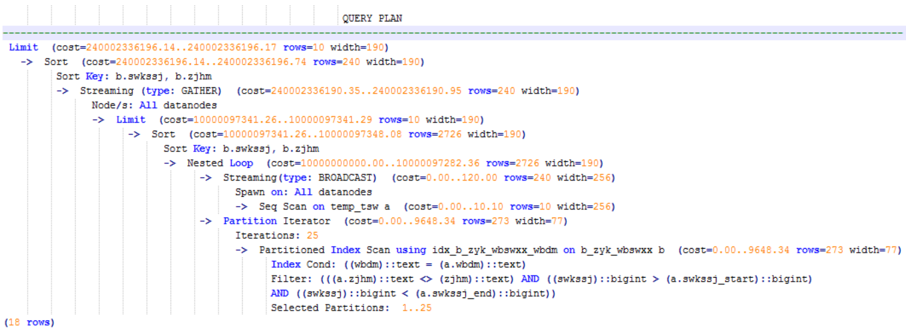
   由于temp_tsw（上网人员信息）中仅包含几百条记录，且temp_tsw和b_zyk_wbswxx（网吧上网信息）均通过wbdm（网吧代码）执行等值连接。因此，如果Join方式改为Nest Loop Join，则扫描节点可以实现Index Scan，性能预计将会提升。

4. 执行如下语句，将Join方式改为Nest Loop Join。

```
SET enable_hashjoin = off;
```

   执行计划如图6-8所示。查询耗时约3秒。

图 6-8 应用 unlogged table 案例（三）



```
QUERY PLAN
-----------------------------------------------------------------------------------------------------
Limit  (cost=240002336196.14..240002336196.17 rows=10 width=190)
  ->  Sort  (cost=240002336196.14..240002336196.74 rows=240 width=190)
        Sort Key: b.swkssj, b.zjhm
        ->  Streaming (type: GATHER)  (cost=240002336190.35..240002336190.95 rows=240 width=190)
              Node/s: All datanodes
              ->  Limit  (cost=10000097341.26..10000097341.29 rows=10 width=190)
                    ->  Sort  (cost=10000097341.26..10000097348.08 rows=2726 width=190)
                          Sort Key: b.swkssj, b.zjhm
                          ->  Nested Loop  (cost=10000000000.00..10000097282.36 rows=2726 width=190)
                                ->  Streaming(type: BROADCAST)  (cost=0.00..120.00 rows=240 width=256)
                                      Spawn on: All datanodes
                                      ->  Seq Scan on temp_tsw a  (cost=0.00..10.10 rows=10 width=256)
                                ->  Partition Iterator  (cost=0.00..9648.34 rows=273 width=77)
                                      Iterations: 25
                                      ->  Partitioned Index Scan using idx_b_zyk_wbswxx_wbdm on b_zyk_wbswxx b  (cost=0.00..9648.34 rows=273 width=77)
                                            Index Cond: ((wbdm)::text = (a.wbdm)::text)
                                            Filter: (((a.zjhm)::text <> (zjhm)::text) AND ((swkssj)::bigint > (a.swkssj_start)::bigint)
                                            AND ((swkssj)::bigint < (a.swkssj_end)::bigint))
                                            Selected Partitions:  1..25
(18 rows)
```

5. 使用unlogged table保存结果集并用于分页显示。

如果需要在上层应用页面实现分页显示，需要修改offset值确定显示目标页的结果集。按此实现，每次翻页时均执行上面查询语句，耗时较长。

为解决上述问题，建议使用unlogged table保存结果集。

```
//创建保存结果集的unlogged table
CREATE UNLOGGED TABLE temp_result
(
WBM      NVARCHAR2(70),
DZQH     NVARCHAR2(6),
DZ       NVARCHAR2(70),
IPDZ     NVARCHAR2(39),
ZJHM     NVARCHAR2(18),
XM       NVARCHAR2(30),
SWKSSJ   date,
XWSJ     date,
SWZDH    NVARCHAR2(32)
);

//将结果集插入unlogged table，插入耗时约3秒。
INSERT INTO
temp_result
SELECT
A.WBM,
A.DZQH,
A.DZ,
A.IPDZ,
B.ZJHM,
B.XM,
to_date(B.SWKSSJ,'yyyymmddHH24MISS') as SWKSSJ,
to_date(B.XWSJ,'yyyymmddHH24MISS') as XWSJ,
B.SWZDH
FROM temp_tsw A,
b_zyk_wbswxx B
WHERE
A.ZJHM <> B.ZJHM
AND A.WBDM = B.WBDM
AND (B.SWKSSJ)::int8 > (A.swkssj_start)::int8
AND (B.SWKSSJ)::int8 < (A.swkssj_end)::int8
;

//查询结果集表进行分页显示，分页查询耗时约10ms。
SELECT
*
FROM
temp_result
ORDER BY
SWKSSJ,
ZJHM
LIMIT 10 OFFSET 0;
```

⚠ 注意

收集更准确的统计信息，通常会改善查询性能，但是也有可能使性能劣化。如果遇到性能劣化，可以考虑：

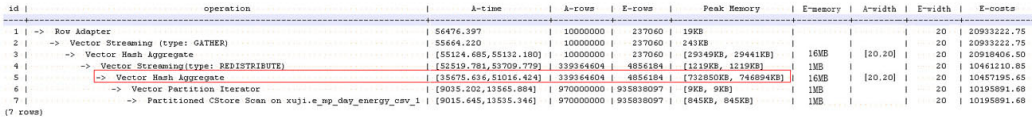
- 恢复默认的统计信息。
- 使用plan hint来调整到之前的查询计划。（详细参见**使用Plan Hint进行调优**）

6.6.5 算子级调优

算子级调优介绍

一个查询语句要经过多个算子步骤才会输出最终的结果。由于个别算子耗时过长导致整体查询性能下降的情况比较常见。这些算子是整个查询的瓶颈算子。通用的优化手段是EXPLAIN ANALYZE/PERFORMANCE命令查看执行过程的瓶颈算子，然后进行针对性优化。

如下面的执行过程信息中，Hashagg算子的执行时间占总时间的：(51016-13535)/56476 ≈66%，此处Hashagg算子就是这个查询的瓶颈算子，在进行性能优化时应当优先考虑此算子的优化。



```
 id |                                operation                                |        A-time         |  A-rows   | E-rows  |      Peak Memory      | E-memory | A-width | E-width |   E-costs
----+-------------------------------------------------------------------------+-----------------------+-----------+---------+-----------------------+----------+---------+---------+-------------
  1 | ->  Row Adapter                                                         | 56476.397             |  10000000 |  237060 | 19KB                  |          |         |      20 | 20933222.75
  2 |    ->  Vector Streaming (type: GATHER)                                  | 55664.220             |  10000000 |  237060 | 243KB                 |          |         |      20 | 20933222.75
  3 |       ->  Vector Hash Aggregate                                         | [55124.685,55132.180] |  10000000 |  237060 | [29349KB, 29441KB]    | 16MB     | [20,20] |      20 | 20918406.50
  4 |          ->  Vector Streaming(type: REDISTRIBUTE)                       | [52519.781,53709.779] | 339364604 | 4856184 | [1219KB, 1219KB]      | 1MB      |         |      20 | 10461210.85
  5 |             ->  Vector Hash Aggregate                                   | [35675.636,51016.424] | 339364604 | 4856184 | [732850KB, 746894KB]  | 16MB     | [20,20] |      20 | 10457195.65
  6 |                ->  Vector Partition Iterator                            | [9035.202,13565.864]  | 970000000 | 935838097 | [9KB, 9KB]            | 1MB      |         |      20 | 10195891.68
  7 |                   ->  Partitioned CStore Scan on xuji.e_mp_day_energy_csv_1 | [9015.645,13535.346] | 970000000 | 935838097 | [845KB, 845KB]        | 1MB      |         |      20 | 10195891.68
(7 rows)
```

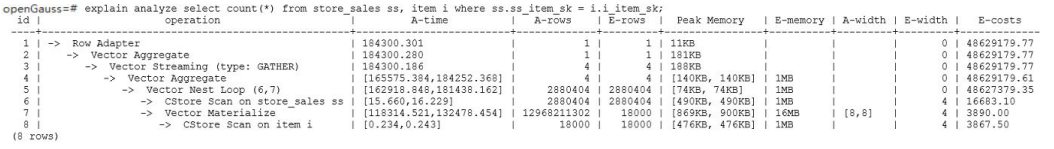
算子级调优示例

示例1：基表扫描时，对于点查或者范围扫描等过滤大量数据的查询，如果使用SeqScan全表扫描会比较耗时，可以在条件列上建立索引选择IndexScan进行索引扫描提升扫描效率。

```
openGauss=# explain (analyze on, costs off) select * from store_sales where ss_sold_date_sk = 2450944;
 id |            operation            |       A-time        | A-rows | Peak Memory  | A-width
----+---------------------------------+---------------------+--------+--------------+---------
  1 | ->  Streaming (type: GATHER)    | 3666.020            |   3360 | 195KB        |
  2 |    ->  Seq Scan on store_sales | [3594.611,3594.611] |   3360 | [34KB, 34KB] |
(2 rows)

 Predicate Information (identified by plan id)
-----------------------------------------------
   2 --Seq Scan on store_sales
         Filter: (ss_sold_date_sk = 2450944)
         Rows Removed by Filter: 4968936
openGauss=# create index idx on store_sales_row(ss_sold_date_sk);
CREATE INDEX
openGauss=# explain (analyze on, costs off) select * from store_sales_row where ss_sold_date_sk = 2450944;
 id |                   operation                    |      A-time      | A-rows | Peak Memory  | A-width
----+------------------------------------------------+------------------+--------+--------------+---------
  1 | ->  Streaming (type: GATHER)                   | 81.524           |   3360 | 195KB        |
  2 |    ->  Index Scan using idx on store_sales_row | [13.352,13.352] |   3360 | [34KB, 34KB] |
(2 rows)
```

上述例子中，全表扫描返回3360条数据，过滤掉大量数据，在ss_sold_date_sk列上建立索引后，使用IndexScan扫描效率显著提高，从3.6秒提升到13毫秒。

示例2：如果从执行计划中看，两表join选择了NestLoop，而实际行数比较大时，NestLoop Join可能执行比较慢。如下的例子中NestLoop耗时181秒，如果设置参数enable_mergejoin=off关掉Merge Join，同时设置参数enable_nestloop=off关掉NestLoop，让优化器选择HashJoin，则Join耗时提升至200多毫秒。



```
openGauss=# explain analyze select count(*) from store_sales ss, item i where ss.ss_item_sk = i.i_item_sk;
 id |                          operation                          |          A-time          |   A-rows    | E-rows  |   Peak Memory   | E-memory | A-width | E-width |   E-costs
----+-------------------------------------------------------------+--------------------------+-------------+---------+-----------------+----------+---------+---------+-------------
  1 | ->  Row Adapter                                             | 184300.301               |           1 |       1 | 11KB            |          |         |       0 | 48629179.77
  2 |    ->  Vector Aggregate                                     | 184300.280               |           1 |       1 | 181KB           |          |         |       0 | 48629179.77
  3 |       ->  Vector Streaming (type: GATHER)                   | 184300.186               |           4 |       4 | 188KB           |          |         |       0 | 48629179.77
  4 |          ->  Vector Aggregate                               | [165575.384,184252.368]  |           4 |       4 | [140KB, 140KB]  | 1MB      |         |       0 | 48629179.61
  5 |             ->  Vector Nest Loop (6,7)                      | [162918.848,181438.162]  |     2880404 | 2880404 | [74KB, 74KB]    | 1MB      |         |       0 | 48627379.35
  6 |                ->  CStore Scan on store_sales ss            | [15.660,16.229]          |     2880404 | 2880404 | [490KB, 490KB]  | 1MB      |         |       4 | 16683.10
  7 |                ->  Vector Materialize                       | [118314.521,132478.454]  | 12968211302 |   18000 | [869KB, 900KB]  | 16MB     | [8,8]   |       4 | 3890.00
  8 |                   ->  CStore Scan on item i                 | [0.234,0.243]            |       18000 |   18000 | [476KB, 476KB]  | 1MB      |         |       4 | 3867.50
(8 rows)
```

```
openGauss=# set enable_nestloop=off;
SET
openGauss=# set enable_mergejoin=off;
SET
openGauss=# explain analyze select count(*) fpostgres=# ales ss, item i where ss.ss_item_sk = i.i_item_sk;
 id |                       operation                       |       A-time        | A-rows  | E-rows  |  Peak Memory   | E-memory | A-width | E-width | E-costs
----+-------------------------------------------------------+---------------------+---------+---------+----------------+----------+---------+---------+----------
  1 | ->  Row Adapter                                       | 291.066             |       1 |       1 | 11KB           |          |         |       0 | 32308.66
  2 |    ->  Vector Aggregate                               | 291.052             |       1 |       1 | 181KB          |          |         |       0 | 32308.66
  3 |       ->  Vector Streaming (type: GATHER)             | 290.973             |       4 |       4 | 188KB          |          |         |       0 | 32308.66
  4 |          ->  Vector Aggregate                         | [220.792,234.532]   |       4 |       4 | [140KB, 140KB] | 1MB      |         |       0 | 32308.50
  5 |             ->  Vector Hash Join (6,7)                | [209.987,223.345]   | 2880404 | 2880404 | [236KB, 241KB] | 16MB     | [8,8]   |       0 | 30508.24
  6 |                ->  CStore Scan on store_sales ss      | [13.132,13.717]     | 2880404 | 2880404 | [490KB, 490KB] | 1MB      |         |       4 | 16683.10
  7 |                ->  CStore Scan on item i              | [0.214,0.246]       |   18000 |   18000 | [477KB, 477KB] | 1MB      |         |       4 | 3867.50
(7 rows)
```

示例3：通常情况下Agg选择HashAgg性能较好，如果大结果集选择了Sort +GroupAgg，则需要设置enable_sort=off，HashAgg耗时明显优于Sort+GroupAgg。

```
openGauss=#  explain analyze select count(*) from store_sales group by ss_item_sk;
 id |                  operation                  |       A-time        | A-rows  | E-rows  |      Peak Memory      | E-memory | A-width | E-width | E-costs
----+---------------------------------------------+---------------------+---------+---------+-----------------------+----------+---------+---------+----------
  1 | ->  Row Adapter                             | 1977.385            |   18000 |   17644 | 20KB                  |          |         |       4 | 92875.24
  2 |    ->  Vector Streaming (type: GATHER)      | 1973.617            |   18000 |   17644 | 1946KB                |          |         |       4 | 92875.24
  3 |       ->  Vector Sort Aggregate             | [1784.800,1883.243] |   18000 |   17644 | [273KB, 273KB]        | 1MB      |         |       4 | 92186.02
  4 |          ->  Vector Sort                    | [1752.270,1848.357] | 2880404 | 2880404 | [128466KB, 135135KB]  | 16MB     | [8,8]   |       4 | 88541.40
  5 |             ->  CStore Scan on store_sales  | [12.483,13.548]     | 2880404 | 2880404 | [490KB, 490KB]        | 1MB      |         |       4 | 16683.10
(5 rows)


openGauss=#  set enable_sort=off;
SET
openGauss=#  explain analyze select count(*) from store_sales group by ss_item_sk;
 id |                  operation                  |       A-time        | A-rows  | E-rows  |      Peak Memory      | E-memory | A-width | E-width | E-costs
----+---------------------------------------------+---------------------+---------+---------+-----------------------+----------+---------+---------+----------
  1 | ->  Row Adapter                             | 838.218             |   18000 |   17644 | 20KB                  |          |         |       4 | 21016.93
  2 |    ->  Vector Streaming (type: GATHER)      | 834.264             |   18000 |   17644 | 228KB                 |          |         |       4 | 21016.93
  3 |       ->  Vector Hash Aggregate             | [585.017,758.204]   |   18000 |   17644 | [262552KB, 262564KB]  | 16MB     | [8,8]   |       4 | 20327.72
  4 |          ->  CStore Scan on store_sales     | [12.540,13.941]     | 2880404 | 2880404 | [490KB, 490KB]        | 1MB      |         |       4 | 16683.10
(4 rows)
```

6.6.6 数据倾斜调优

数据倾斜问题是分布式架构的重要难题，它破坏了MPP架构中各个节点对等的要求，导致单节点（倾斜节点）所存储或者计算的数据量远大于其他节点，所以会造成以下危害：

- 存储上的倾斜会严重限制系统容量，在系统容量不饱和的情况下，由于单节点倾斜的限制，使得整个系统容量无法继续增长。
- 计算上的倾斜会严重影响系统性能，由于倾斜节点所需要运算的数据量远大于其它节点，导致倾斜节点降低系统整体性能。
- 数据倾斜还严重影响了MPP架构的扩展性。由于在存储或者计算时，往往会将相同值的数据放到同一节点，因此当倾斜数据（大量数据的值相同）出现之后，即使增加节点，系统瓶颈仍然受限于倾斜节点的容量或者性能。

GaussDB数据库针对数据倾斜问题给出了完整的解决方案，包括存储倾斜和计算倾斜两大问题，下面分别进行介绍。

存储层数据倾斜

GaussDB数据库中，数据分布存储在各个DN上，通过分布式执行提高查询的效率。但是，如果数据分布存在倾斜，则会导致分布式执行某些DN成为瓶颈，影响查询性能。这种情况通常是由于分布列选择不合理，可以通过调整分布列的方式解决。

例如下例：

```
openGauss=# explain performance select count(*) from inventory;
5 --CStore Scan on lmz.inventory
        dn_6001_6002 (actual time=0.444..83.127 rows=42000000 loops=1)
        dn_6003_6004 (actual time=0.512..63.554 rows=27000000 loops=1)
        dn_6005_6006 (actual time=0.722..99.033 rows=45000000 loops=1)
        dn_6007_6008 (actual time=0.529..100.379 rows=51000000 loops=1)
        dn_6009_6010 (actual time=0.382..71.341 rows=36000000 loops=1)
        dn_6011_6012 (actual time=0.547..100.274 rows=51000000 loops=1)
        dn_6013_6014 (actual time=0.596..118.289 rows=60000000 loops=1)
        dn_6015_6016 (actual time=1.057..132.346 rows=63000000 loops=1)
        dn_6017_6018 (actual time=0.940..110.310 rows=54000000 loops=1)
```

```
        dn_6019_6020 (actual time=0.231..41.198 rows=21000000 loops=1)
        dn_6021_6022 (actual time=0.927..114.538 rows=54000000 loops=1)
        dn_6023_6024 (actual time=0.637..118.385 rows=60000000 loops=1)
        dn_6025_6026 (actual time=0.288..32.240 rows=15000000 loops=1)
        dn_6027_6028 (actual time=0.566..118.096 rows=60000000 loops=1)
        dn_6029_6030 (actual time=0.423..82.913 rows=42000000 loops=1)
        dn_6031_6032 (actual time=0.395..78.103 rows=39000000 loops=1)
        dn_6033_6034 (actual time=0.376..51.052 rows=24000000 loops=1)
        dn_6035_6036 (actual time=0.569..79.463 rows=39000000 loops=1)
```

在performance信息中，可以看到inventory表各DN的scan行数，发现各DN的行数差距较大，最大的为63000000，最小的只有15000000，差了4倍。这个差距对于数据扫描的性能影响还可以接受，但如果上层有join算子，则影响较大。

通常，数据表在各DN上是hash分布的，因此分布列的选择很重要。通过table_skewness()来查看上述inventory表在各DN的数据分布倾斜，查询结果如下：

```
openGauss=# select table_skewness('inventory');
              table_skewness
-------------------------------------------
 ("dn_6015_6016       ",63000000,8.046%)
 ("dn_6013_6014       ",60000000,7.663%)
 ("dn_6023_6024       ",60000000,7.663%)
 ("dn_6027_6028       ",60000000,7.663%)
 ("dn_6017_6018       ",54000000,6.897%)
 ("dn_6021_6022       ",54000000,6.897%)
 ("dn_6007_6008       ",51000000,6.513%)
 ("dn_6011_6012       ",51000000,6.513%)
 ("dn_6005_6006       ",45000000,5.747%)
 ("dn_6001_6002       ",42000000,5.364%)
 ("dn_6029_6030       ",42000000,5.364%)
 ("dn_6031_6032       ",39000000,4.981%)
 ("dn_6035_6036       ",39000000,4.981%)
 ("dn_6009_6010       ",36000000,4.598%)
 ("dn_6003_6004       ",27000000,3.448%)
 ("dn_6033_6034       ",24000000,3.065%)
 ("dn_6019_6020       ",21000000,2.682%)
 ("dn_6025_6026       ",15000000,1.916%)
(18 rows)
```

通过查询建表定义，可以发现，目前该表是以inv_date_sk作为分布列的，导致存在倾斜。通过查看各列的数据分布情况，建表时改为inv_item_sk作为分布列，则倾斜情况分布如下：

```
openGauss=# select table_skewness('inventory');
              table_skewness
-------------------------------------------
 ("dn_6001_6002       ",43934200,5.611%)
 ("dn_6007_6008       ",43829420,5.598%)
 ("dn_6003_6004       ",43781960,5.592%)
 ("dn_6031_6032       ",43773880,5.591%)
 ("dn_6033_6034       ",43763280,5.589%)
 ("dn_6011_6012       ",43683600,5.579%)
 ("dn_6013_6014       ",43551660,5.562%)
 ("dn_6027_6028       ",43546340,5.561%)
 ("dn_6009_6010       ",43508700,5.557%)
 ("dn_6023_6024       ",43484540,5.554%)
 ("dn_6019_6020       ",43466800,5.551%)
 ("dn_6021_6022       ",43458500,5.550%)
 ("dn_6017_6018       ",43448040,5.549%)
 ("dn_6015_6016       ",43247700,5.523%)
 ("dn_6005_6006       ",43200240,5.517%)
 ("dn_6029_6030       ",43181360,5.515%)
 ("dn_6025_6026       ",43179700,5.515%)
 ("dn_6035_6036       ",42960080,5.487%)
(18 rows)
```

数据分布倾斜的问题得到解决。

除了table_skewness()视图外，当前版本还提供了**table_distribution**函数和**PGXC_GET_TABLE_SKEWNESS**视图，可以更加高效的查询各表的数据倾斜情况。

计算层数据倾斜

即使通过修改表的分布键，使得数据存储在各个节点上是均衡的，但是在执行查询的过程中，仍然可能出现数据倾斜的问题。在运算过程中某个算子在DN上输出的结果集出现倾斜，从而导致此算子上层的运算出现计算倾斜。一般来说，这是由于在执行过程中，数据重分布导致的。

在查询执行的过程中，join key、group by key等往往不是表的分布列，因此需要按照join key、group by key上数据的hash值，让数据在各个DN之间进行重新分布，这个过程对应于计划中的Redistribute算子。当重分布列上的数据存在倾斜时，就会导致运行时的数据倾斜，即重分布后部分节点的数据远远大于其他。倾斜节点需要处理更多的数据，导致倾斜节点的计算性能远远低于其他节点。

如下例中，s表和t表join，join条件中的s.x和t.x均不是表的分布列，因此需要重分布（REDISTRIBUTE算子）。其中s.x列上存在倾斜值，t.x上不存在倾斜。id=6的stream算子在datanode2节点输出的结果集是其他DN的3倍，从而导致了计算倾斜。

```
openGauss=# explain select * from skew s,test t where s.x = t.x order by s.a limit 1;
 id |                    operation                    |        A-time
----+-------------------------------------------------+----------------------
  1 | ->  Limit                                       | 52622.382
  2 |    ->  Streaming (type: GATHER)                 | 52622.374
  3 |       ->  Limit                                 | [30138.494,52598.994]
  4 |          ->  Sort                               | [30138.486,52598.986]
  5 |             ->  Hash Join (6,8)                 | [30127.013,41483.275]
  6 |                ->  Streaming(type: REDISTRIBUTE)    | [11365.110,22024.845]
  7 |                   ->  Seq Scan on public.skew s     | [2019.168,2175.369]
  8 |                ->  Hash                         | [2460.108,2499.850]
  9 |                   ->  Streaming(type: REDISTRIBUTE) | [1056.214,1121.887]
 10 |                      ->  Seq Scan on public.test t  | [310.848,325.569]
(10 rows)
 6 --Streaming(type: REDISTRIBUTE)
        datanode1 (rows=5050368)
        datanode2 (rows=15276032)
        datanode3 (rows=5174272)
        datanode4 (rows=5219328)
```

和存储倾斜相比，计算倾斜更难以提前识别，因此GaussDB提出了RLBT(Runtime Load Balance Technology)方案，用以解决运行时的计算倾斜问题，该特性由GUC参数skew_option控制。RLBT方案主要分为两个层面，第一步是计算倾斜识别，第二步是计算倾斜解决。下面分别进行介绍。

1. 倾斜识别

   计算倾斜的识别，即预先识别计算过程中的重分布列是否存在倾斜数据。RLBT方案中给出了三个解决手段，统计信息识别，hint方式指定以及规则识别：

   - 统计信息识别

     需要用户先执行analyze收集各表的统计信息，然后优化器能够自动利用统计信息对重分布键上的倾斜数据进行提前识别，对于存在倾斜的查询，生成相应的优化计划。在重分布键有多列的情况，只有所有列都属于同一个基表才能利用统计信息进行识别。

     统计信息只能给出基表的倾斜情况，当基表某一列存在倾斜，其他列上带有过滤条件，或者经过和其他表的join之后，无法准确判断倾斜列上倾斜数据是否依旧存在。当skew_option为normal时，这里认为倾斜数据依旧存在，仍然会对基表中识别到的倾斜进行优化；当skew_option为lazy时，这里认为倾斜数据已经不再存在，也就不会进行相应的优化。

- hint方式指定

  统计信息有着一定的局限性，对于较为复杂的查询，其中间结果难以通过统计信息进行估算和识别倾斜数据。对于这种情况，设计了hint手段，通过用户手动指定的方式，给定倾斜信息。优化器根据用户给定的倾斜信息，来对查询进行优化。详细hint使用语法参见**运行倾斜的hint**。

- 规则识别

  现在BI系统往往会产生大量带有outer join（left join、right join、full join）的SQL，outer join在匹配失败的情况下会补空产生大量NULL值，如果接下来在补空列上进行join或者group by操作，就会导致NULL值倾斜。当前RLBT技术会自动识别这种场景，并生成相应的NULL值倾斜优化计划。

2. 计算倾斜解决

   在解决倾斜时，目前针对最常见的join和agg算子进行了优化。

   - join优化

   基本思路是将倾斜数据和非倾斜数据进行隔离处理。主要分为以下三种情况：

   a. join两侧都需要做重分布：

      对倾斜侧做PART_REDISTRIBUTE_PART_ROUNDROBIN，其中对倾斜数据做roundrobin，非倾斜数据做redistribute；

      对非倾斜侧做PART_REDISTRIBUTE_PART_BROADCAST，其中对倾斜数据做broadcast，非倾斜数据做redistribute；

   b. join一侧需要重分布，另一侧不需要重分布：

      对需要重分布的一侧做PART_REDISTRIBUTE_PART_ROUNDROBIN；

      对不需要重分布的一侧做PART_LOCAL_PART_BROADCAST，其中对等于倾斜值的部分做broadcast，其余数据保留在本地。

   c. 对于有补NULL值的表：

      对该表做PART_REDISTERIBUTE_PART_LOCAL，其中将NULL值保留在本地，其余数据做redistribute。

   以前面的查询为例，s.x列上存在倾斜数据，倾斜数据的值为0。优化器通过统计信息，识别到了该倾斜数据，生成了倾斜优化计划如下：

```
 id |                                operation                                 |        A-time
----+--------------------------------------------------------------------------+----------------------
  1 | ->  Limit                                                                | 23642.049
  2 |    ->  Streaming (type: GATHER)                                          | 23642.041
  3 |       ->  Limit                                                          | [23310.768,23618.021]
  4 |          ->  Sort                                                        | [23310.761,23618.012]
  5 |             ->  Hash Join (6,8)                                          | [20898.341,21115.272]
  6 |                ->  Streaming(type: PART REDISTRIBUTE PART ROUNDROBIN)    |
[7125.834,7472.111]
  7 |                   ->  Seq Scan on public.skew s                          | [1837.079,1911.025]
  8 |                ->  Hash                                                  | [2612.484,2640.572]
  9 |                   ->  Streaming(type: PART REDISTRIBUTE PART BROADCAST)  | [1193.548,1297.894]
 10 |                      ->  Seq Scan on public.test t                       | [314.343,328.707]
(10 rows)
   5 --Hash Join (6,8)
       Hash Cond: s.x = t.x
       Skew Join Optimizated by Statistic
   6 --Streaming(type: PART REDISTRIBUTE PART ROUNDROBIN)
       datanode1 (rows=7635968)
       datanode2 (rows=7517184)
       datanode3 (rows=7748608)
       datanode4 (rows=7818240)
```

上述执行计划中，可以看到Skew Join Optimizated by Statistic的字样，代表该计划为倾斜优化计划，其中Statistic关键字代表该倾斜优化来自于统计信息，除此之外还有Hint和Rule，分别代表倾斜优化来自于hint语句和规则。对比前面的计划可

以看到，这里对于非倾斜数据和倾斜数据做了分别处理。对于s表中的非倾斜数据，依旧按照原有的方案，根据数据的hash值进行重分布；而对于倾斜数据（即等于0的数据），则通过轮询发送的方式，均衡地发送到所有节点。通过这样的方式，解决了倾斜数据分布不均衡的问题。

同时，为了保证结果的正确性，需要对t表做相应的处理。对于t表中等于0（s.x表中的倾斜值）的数据做广播，对于其他数据，依旧根据数据的hash值进行重分布。

通过这样的方式，就解决了join操作中，数据倾斜的问题。从上面的结果来看，id=6的stream算子各个DN的输出结果已经非常均衡，同时查询端到端性能提升了1倍。

- agg优化

对于agg操作，解决倾斜的思路与join操作不同，这里是通过首先在本DN内按照group by key对重分布键进行去重操作，然后再进行重分布。因为经过DN内部去重之后，重分布键的值每个DN最多只有一个，所以从全局来看，每个重分布键值的数量都不会超过DN数，因此不会出现严重的数据倾斜问题。以如下query为例：

```
select c1, c2, c3, c4, c5, c6, c7, c8, c9, count(*) from t group by c1, c2, c3, c4, c5, c6, c7, c8, c9 limit 10;
```

原执行结果如下：

```
 id |                operation                 |         A-time         | A-rows
----+------------------------------------------+------------------------+----------
  1 | ->  Streaming (type: GATHER)             | 130621.783             |       12
  2 |    ->  GroupAggregate                    | [85499.711,130432.341] |       12
  3 |       ->  Sort                           | [85499.509,103145.632] | 36679237
  4 |          ->  Streaming(type: REDISTRIBUTE) | [25668.897,85499.050]  | 36679237
  5 |             ->  Seq Scan on public.t     | [9835.069,10416.388]   | 36679237
(5 rows)

  4 --Streaming(type: REDISTRIBUTE)
        datanode1 (rows=36678837)
        datanode2 (rows=100)
        datanode3 (rows=100)
        datanode4 (rows=200)
```

其中存在大量倾斜数据，导致数据按照group by key进行重分布之后，datanode1的数据量是其他节点的数十万倍。在倾斜优化之后，首先在本DN进行一次group by操作，达到数据去重的效果，然后再进行重分布，可以发现基本没有数据倾斜的问题出现。

```
 id |                operation                 |        A-time
----+------------------------------------------+-----------------------
  1 | ->  Streaming (type: GATHER)             | 10961.337
  2 |    ->  HashAggregate                     | [10953.014,10953.705]
  3 |       ->  HashAggregate                  | [10952.957,10953.632]
  4 |          ->  Streaming(type: REDISTRIBUTE) | [10952.859,10953.502]
  5 |             ->  HashAggregate            | [10084.280,10947.139]
  6 |                ->  Seq Scan on public.t  | [4757.031,5201.168]
(6 rows)

 Predicate Information (identified by plan id)
-----------------------------------------------
  3 --HashAggregate
        Skew Agg Optimized by Statistic
(2 rows)

  4 --Streaming(type: REDISTRIBUTE)
        datanode1 (rows=17)
        datanode2 (rows=8)
        datanode3 (rows=8)
        datanode4 (rows=14)
```

适用范围

- join算子
  - 支持nest loop，merge join，hash join等join方式；
  - 当倾斜数据处于join的left侧时，支持inner join，left join，semi join，anti join；当倾斜属于位于join的right侧时，支持inner join，right join，right semi join，right anti join。
  - 通过统计信息得到的倾斜优化计划，优化器会根据代价判断该计划是否为最优计划。通过hint和规则会强制生成倾斜优化计划。
- agg算子
  - array_agg、string_agg、subplan in agg qual这几种场景不支持优化；
  - 通过统计信息识别到的倾斜优化计划会受到代价、plan_mode_seed参数、best_agg_plan参数影响，而hint、规则识别到的不会。

6.7 经验总结：SQL 语句改写规则

根据数据库的SQL执行机制以及大量的实践，总结发现：通过一定的规则调整SQL语句，在保证结果正确的基础上，能够提高SQL执行效率。如果遵守这些规则，常常能够大幅度提升业务查询效率。

- **使用union all代替union**

  union在合并两个集合时会执行去重操作，而union all则直接将两个结果集合并、不执行去重。执行去重会消耗大量的时间，因此，在一些实际应用场景中，如果通过业务逻辑已确认两个集合不存在重叠，可用union all替代union以便提升性能。

- **join列增加非空过滤条件**

  若join列上的NULL值较多，则可以加上is not null过滤条件，以实现数据的提前过滤，提高join效率。

- **not in转not exists**

  not in语句需要使用nestloop anti join来实现，而not exists则可以通过hash anti join来实现。在join列不存在null值的情况下，not exists和not in等价。因此在确保没有null值时，可以通过将not in转换为not exists，通过生成hash join来提升查询效率。

  如下所示，如果t2.d2字段中没有null值(t2.d2字段在表定义中not null)查询可以修改为

  ```
  SELECT * FROM t1 WHERE  NOT EXISTS (SELECT * FROM t2 WHERE t1.c1=t2.d2);
  ```

  产生的计划如下：

图 6-9 not exists 执行计划

```
 id |                 operation
----+-------------------------------------------
  1 | ->  Streaming (type: GATHER)
  2 |    ->  Hash Anti Join (3, 4)
  3 |       ->  Seq Scan on t1
  4 |       ->  Hash
  5 |          ->  Streaming(type: REDISTRIBUTE)
  6 |             ->  Seq Scan on t2
(6 rows)

 Predicate Information (identified by plan id)
-----------------------------------------------
   2 --Hash Anti Join (3, 4)
         Hash Cond: (t1.c1 = t2.d2)
(2 rows)
```

- **选择hashagg。**

  查询中GROUP BY语句如果生成了groupagg+sort的plan性能会比较差，可以通过加大work_mem的方法生成hashagg的plan，因为不用排序而提高性能。

- **尝试将函数替换为case语句。**

  GaussDB函数调用性能较低，如果出现过多的函数调用导致性能下降很多，可以根据情况把可下推函数的函数改成CASE表达式。

- **避免对索引使用函数或表达式运算。**

  对索引使用函数或表达式运算会停止使用索引转而执行全表扫描。

- **尽量避免在where子句中使用!=或<>操作符、null值判断、or连接、参数隐式转换。**

- 如果where条件中出现了 >= 和 <= 同一个值，由于当前不支持范围等价类推导，尽量把条件改为 = 查询。

  如：SELECT * FROM t1 WHERE c1 >= 1 AND c1 <= 1 修改为SELECT * FROM t1 WHERE c1 = 1。

  对于范围查询，优化器在计算选择率时误差相对等值查询较大，所以尽可能把范围查询改为等值查询。

- **对复杂SQL语句进行拆分。**

  对于过于复杂并且不易通过以上方法调整性能的SQL可以考虑拆分的方法，把SQL中某一部分拆分成独立的SQL并把执行结果存入临时表，拆分常见的场景包括但不限于：

  - 作业中多个SQL有同样的子查询，并且子查询数据量较大。
  - Plan cost计算不准，导致子查询hash bucket太小，比如实际数据1000W行，hash bucket只有1000。
  - 函数（如substr,to_number）导致大数据量子查询选择度计算不准。
  - 多DN环境下对大表做broadcast的子查询。

6.8 SQL 调优关键参数调整

本节将介绍影响GaussDB SQL调优性能的关键CN配置参数。

表 6-2 CN 配置参数

参数/参考值	描述
enable_nestloop=on	控制查询优化器对嵌套循环连接（Nest Loop Join）类型的使用。当设置为“on”后，优化器优先使用Nest Loop Join；当设置为“off”后，优化器在存在其他方法时将优先选择其他方法。 **说明** 如果只需要在当前数据库连接（即当前Session）中临时更改该参数值，则只需要在SQL语句中执行如下命令： `SET enable_nestloop to off;` 实际调优中应根据情况选择是否关闭。一般情况下，在三种join方式（Nested Loop、Merge Join和Hash Join）里，Nested Loop适合小数据量或者有索引的场景，Hash Join适合大数据分析场景。
enable_bitmapscan=on	控制查询优化器对位图扫描规划类型的使用。设置为“on”，表示使用；设置为“off”，表示不使用。 **说明** 如果只需要在当前数据库连接（即当前Session）中临时更改该参数值，则只需要在SQL语句中执行命令如下命令： `SET enable_bitmapscan to off;` bitmapscan扫描方式适用于“where a > 1 and b > 1”且a列和b列都有索引这种查询条件，但有时其性能不如indexscan。因此，现场调优如发现查询性能较差且计划中有bitmapscan算子，可以关闭bitmapscan，看性能是否有提升。
enable_fast_query_shipping=on	控制查询优化器是否使用分布式框架，执行快速执行计划。设置为“on”，表示执行计划在CN和DN上各自生成；设置为“off”，表示使用分布式框架，即执行计划在CN上生成，然后发送到DN中执行。 **说明** 如果只需要在当前数据库连接（即当前Session）中临时更改该参数值，则只需要在SQL语句中执行如下命令： `SET enable_fast_query_shipping to off;`
enable_hashagg=on	控制优化器对Hash聚集规划类型的使用。
enable_hashjoin=on	控制优化器对Hash连接规划类型的使用。
enable_mergejoin=on	控制优化器对融合连接规划类型的使用。
enable_indexscan=on	控制优化器对索引扫描规划类型的使用。
enable_indexonlyscan=on	控制优化器对仅索引扫描规划类型的使用。
enable_seqscan=on	控制优化器对顺序扫描规划类型的使用。完全消除顺序扫描是不可能的，但是关闭这个变量会让优化器在存在其他方法的时候优先选择其他方法。

参数/参考值	描述
enable_sort=on	控制优化器使用的排序步骤。该设置不可能完全消除明确的排序，但是关闭这个变量可以让优化器在存在其他方法的时候优先选择其他方法。
enable_broadcast=on	控制查询优化器对于broadcast广播模式数据传输的使用。此方式网络传输数据量较大，因此当网络传输节点（Stream）实际数据量较大而估算不准时，可以将该参数设置为off，看性能是否有提升。
rewrite_rule	控制优化器是否启用LAZYAGG\MAGICSET\PARTIALPUSH\UNIQUECHECK\DISABLEREP\INTARGETLIST\PREDPUSHNORMAL\PREDPUSHFORCE\PREDPUSH\DISABLE_PULLUP_EXPR_SUBLINK\DISABLE_PULLUP_NOT_IN_SUBLINK\ENABLE_SUBLINK_PULLUP_ROWNUM重写规则。
sql_beta_feature	控制优化器是否启用SEL_SEMI_POISSON/NO_UNIQUE_INDEX_FIRST/JOIN_SEL_WITH_CAST_FUNC/SEL_EXPR_INSTR/PARAM_PATH_GEN/RAND_COST_OPT/PARAM_PATH_OPT/PAGE_EST_OPT/CANONICAL_PATHKEY/INDEX_COST_WITH_INDEX_COST_WITH_LEAF_PAGES_ONLY/PREDPUSH_SAME_LEVEL/PARTITION_FDW_ON/DISABLE_BITMAP_COST_WITH_LOSSY_PAGES测试功能。

6.9 使用 Plan Hint 进行调优

6.9.1 Plan Hint 调优概述

Plan Hint为用户提供了直接影响执行计划生成的手段，用户可以通过指定join顺序，join、stream、scan方法，指定结果行数，指定重分布过程中的倾斜信息等多个手段来进行执行计划的调优，以提升查询的性能。

GaussDB还提供了SQL PATCH功能，在不修改业务语句的前提下通过创建SQL PATCH的方式使得Hint生效。

功能描述

Plan Hint支持在SELECT、INSERT、UPDATE、DELETE、MERGE等关键字后通过如下形式指定：

```
/*+ <plan hint>*/
```

可以同时指定多个hint，之间使用空格分隔。hint只能hint当前层的计划，对于子查询计划的hint，需要在子查询的select关键字后指定hint。

例如：

```
select /*+ <plan_hint1> <plan_hint2> */ * from t1, (select /*+ <plan_hint3> */ * from t2) where 1=1;
```

其中<plan_hint1>，<plan_hint2>为外层查询的hint，<plan_hint3>为内层子查询的hint。

检查Plan Hint调优的效果可以借助explain语法进行分析。通过explain可以查看使用Plan Hint后目标SQL的计划，对比计划是否符合要求以验证Plan Hint的效果。explain有多种计划展示的模式，通过explain_perf_mode进行控制。本节的示例一般通过设置explain_perf_mode为pretty模式来展示计划，展示较全的计划相关信息。部分示例设置explain_perf_mode为normal模式以精简输出信息。

须知

如果在视图定义（CREATE VIEW）时指定hint，则在该视图每次被应用时会使用该hint。

当使用random plan功能（参数plan_mode_seed不为0）时，查询指定的plan hint不会被使用。

支持范围

当前版本Plan Hint支持的范围如下，后续版本会进行增强。

- 指定Join顺序的hint - leading hint。
- 指定Join方式的hint，仅支持除semi/anti join，unique plan之外的常用hint。
- 指定结果集行数的hint。
- 指定Stream方式的hint。
- 指定Scan方式的hint，仅支持常用的tablescan，indexscan和indexonlyscan的hint。
- 指定子链接块名的hint。
- 指定倾斜信息的hint，仅支持Join与HashAgg的重分布过程倾斜。
- 指定本query内生效的guc参数的hint（在视图内使用不生效）。
- 指定使用custom plan或generic plan的hint（只对PBE执行的查询语句生效）。
- 指定子查询不展开的hint。
- 指定当前查询语句不进入全局计划缓存（enable_global_plancache打开且当前语句为PBE执行时生效）。

注意事项

- 不支持Agg、Sort、Setop和Subplan的hint。
- 不支持SMP和Node Group场景下的Hint。

示例

本章节使用同一个语句进行示例，便于Plan Hint支持的各方法作对比，示例语句及不带hint的原计划如下所示：

```
explain
select i_product_name product_name
,i_item_sk item_sk
,s_store_name store_name
,s_zip store_zip
,ad2.ca_street_number c_street_number
,ad2.ca_street_name c_street_name
,ad2.ca_city c_city
,ad2.ca_zip c_zip
```



```
,count(*) cnt
,sum(ss_wholesale_cost) s1
,sum(ss_list_price) s2
,sum(ss_coupon_amt) s3
FROM   store_sales
,store_returns
,store
,customer
,promotion
,customer_address ad2
,item
WHERE  ss_store_sk = s_store_sk AND
ss_customer_sk = c_customer_sk AND
ss_item_sk = i_item_sk and
ss_item_sk = sr_item_sk and
ss_ticket_number = sr_ticket_number and
c_current_addr_sk = ad2.ca_address_sk and
ss_promo_sk = p_promo_sk and
i_color in ('maroon','burnished','dim','steel','navajo','chocolate') and
i_current_price between 35 and 35 + 10 and
i_current_price between 35 + 1 and 35 + 15
group by i_product_name
,i_item_sk
,s_store_name
,s_zip
,ad2.ca_street_number
,ad2.ca_street_name
,ad2.ca_city
,ad2.ca_zip
;
 HashAggregate  (cost=53.53..53.76 rows=1 width=880)
   Group By Key: item.i_product_name, item.i_item_sk, store.s_store_name, store.s_zip, ad2.ca_street_number,
ad2.ca_street_name, ad2.ca_city, ad2.ca_zip
   ->  Streaming (type: GATHER)  (cost=53.53..53.76 rows=2 width=880)
         Node/s: All datanodes
         ->  HashAggregate  (cost=53.10..53.11 rows=2 width=880)
               Group By Key: item.i_product_name, item.i_item_sk, store.s_store_name, store.s_zip,
ad2.ca_street_number, ad2.ca_street_name, ad2.ca_city, ad2.ca_zip
               ->  Nested Loop  (cost=0.00..53.07 rows=2 width=776)
                     ->  Streaming(type: REDISTRIBUTE)  (cost=0.00..46.36 rows=2 width=416)
                           Spawn on: All datanodes
                           ->  Nested Loop  (cost=0.00..45.99 rows=2 width=416)
                                 ->  Streaming(type: REDISTRIBUTE)  (cost=0.00..39.27 rows=2 width=258)
                                       Spawn on: All datanodes
                                       ->  Nested Loop  (cost=0.00..38.99 rows=2 width=258)
                                             ->  Streaming(type: REDISTRIBUTE)  (cost=0.00..32.28 rows=2 width=262)
                                                   Spawn on: All datanodes
                                                   ->  Nested Loop  (cost=0.00..32.00 rows=2 width=262)
                                                         ->  Streaming(type: REDISTRIBUTE)  (cost=0.00..25.28 rows=2
width=262)
                                                               Spawn on: All datanodes
                                                               ->  Nested Loop  (cost=0.00..25.00 rows=2 width=262)
                                                                     ->  Nested Loop  (cost=0.00..21.64 rows=2 width=270)
                                                                           ->  Seq Scan on item  (cost=0.00..13.36 rows=1
width=208)
                                                                                 Filter: ((i_current_price >= 35::numeric) AND
(i_current_price <= 45::numeric) AND (i_current_price >= 36::numeric) AND
 (i_current_price <= 50::numeric) AND (i_color = ANY
('{maroon,burnished,dim,steel,navajo,chocolate}'::bpchar[])))
                                                                           ->  Index Scan using store_sales_pkey on store_sales
(cost=0.00..8.27 rows=1 width=62)
                                                                                 Index Cond: (ss_item_sk = item.i_item_sk)
                                                                     ->  Index Only Scan using store_returns_pkey on
store_returns  (cost=0.00..3.35 rows=1 width=8)
                                                                           Index Cond: ((sr_item_sk = store_sales.ss_item_sk) AND
(sr_ticket_number = store_sales.ss_ticket_number))
                                                         ->  Index Scan using customer_pkey on customer  (cost=0.00..3.35
rows=1 width=8)
                                                               Index Cond: (c_customer_sk = store_sales.ss_customer_sk)
```



```
                                        ->  Index Only Scan using promotion_pkey on promotion  (cost=0.00..3.35
rows=1 width=4)
                                              Index Cond: (p_promo_sk = store_sales.ss_promo_sk)
                                  ->  Index Scan using store_pkey on store  (cost=0.00..3.35 rows=1 width=166)
                                        Index Cond: (s_store_sk = store_sales.ss_store_sk)
                            ->  Index Scan using customer_address_pkey on customer_address ad2  (cost=0.00..3.35
rows=1 width=368)
                                  Index Cond: (ca_address_sk = customer.c_current_addr_sk)
(34 rows)
```

6.9.2 Join 顺序的 Hint

功能描述

指明join的顺序，包括内外表顺序。

语法格式

- 仅指定join顺序，不指定内外表顺序。

```
leading(join_table_list)
```

- 同时指定join顺序和内外表顺序，内外表顺序仅在最外层生效。

```
leading((join_table_list))
```

参数说明

join_table_list为表示表join顺序的hint字符串，可以包含当前层的任意个表（别名），或对于子查询提升的场景，也可以包含子查询的hint别名，同时任意表可以使用括号指定优先级，表之间使用空格分隔。

须知

表只能用单个字符串表示，不能带schema。

表如果存在别名，需要优先使用别名来表示该表。

join table list中指定的表需要满足以下要求，否则会报语义错误。

- list中的表必须在当前层或提升的子查询中存在。
- list中的表在当前层或提升的子查询中必须是唯一的。如果不唯一，需要使用不同的别名进行区分。
- 同一个表只能在list里出现一次。
- 如果表存在别名，则list中的表需要使用别名。

例如：

leading(t1 t2 t3 t4 t5)表示：t1, t2, t3, t4, t5先join，五表join顺序及内外表不限。

leading((t1 t2 t3 t4 t5))表示：t1和t2先join，t2做内表；再和t3 join，t3做内表；再和t4 join，t4做内表；再和t5 join，t5做内表。

leading(t1 (t2 t3 t4) t5)表示：t2, t3, t4先join，内外表不限；再和t1, t5 join，内外表不限。

leading((t1 (t2 t3 t4) t5))表示：t2, t3, t4先join，内外表不限；在最外层，t1再和t2, t3, t4的join表join，t1为外表，再和t5 join，t5为内表。

leading((t1 (t2 t3) t4 t5)) leading((t3 t2))表示：t2，t3先做join，t2做内表；然后再和t1做join，t2，t3的join表做内表；然后再跟t4做join，t4做内表，最后和t5做join，t5做内表。

示例

对**示例**中原语句使用如下hint:

```
explain
select /*+ leading((((((store_sales store) promotion) item) customer) ad2) store_returns) leading((store store_sales))*/ i_product_name product_name ...
```

该hint表示：表之间的join关系是：store_sales和store先join， store_sales做内表，然后依次跟promotion, item, customer, ad2, store_returns做join。生成计划如下所示：

```
WARNING:  Duplicated or conflict hint: Leading(store_sales store), will be discarded.
 id |                                  operation                                  |   E-rows   | E-memory | E-width |   E-costs
----+-----------------------------------------------------------------------------+------------+----------+---------+-------------
  1 | ->  Row Adapter                                                             |          6 |          |     273 | 16308094.34
  2 |    ->  Vector Streaming (type: GATHER)                                      |          6 |          |     273 | 16308094.34
  3 |       ->  Vector Hash Aggregate                                             |          6 | 16MB     |     273 | 16308092.67
  4 |          ->  Vector Hash Join (5,20)                                        |          6 | 585MB    |     169 | 16308092.63
  5 |             ->  Vector Streaming(type: REDISTRIBUTE)                        |    1320811 | 1MB      |     181 | 16069870.93
  6 |                ->  Vector Hash Join (7,19)                                  |    1320811 | 43MB     |     181 | 16061891.00
  7 |                   ->  Vector Streaming(type: REDISTRIBUTE)                  |    1320811 | 1MB      |     131 | 16056566.78
  8 |                      ->  Vector Hash Join (9,18)                            |    1320811 | 27MB     |     131 | 16048586.85
  9 |                         ->  Vector Streaming(type: REDISTRIBUTE)            |    1383248 | 1MB      |     131 | 16038321.62
 10 |                            ->  Vector Hash Join (11,17)                     |    1383248 | 16MB     |     131 | 16029664.50
 11 |                               ->  Vector Hash Join (12,16)                  | 2626366951 | 16MB     |      73 | 15751384.88
 12 |                                  ->  Vector Hash Join (13,14)               | 2750085660 | 2156MB   |      77 | 14226077.19
 13 |                                     ->  CStore Scan on store                |      24048 | 1MB      |      19 | 2264.00
 14 |                                     ->  Vector Partition Iterator           | 2879987999 | 1MB      |      66 | 2756066.50
 15 |                                        ->  Partitioned CStore Scan on store_sales | 2879987999 | 1MB      |      66 | 2756066.50
 16 |                                  ->  CStore Scan on promotion               |      36000 | 1MB      |       4 | 1268.50
 17 |                               ->  CStore Scan on item                       |        158 | 1MB      |      58 | 4051.25
 18 |                         ->  CStore Scan on customer                         |   12000000 | 1MB      |       8 | 12923.00
 19 |                      ->  CStore Scan on customer_address ad2                |    6000000 | 1MB      |      58 | 5770.00
 20 |                ->  Vector Partition Iterator                                | 287999764 | 1MB      |      12 | 227383.99
 21 |                   ->  Partitioned CStore Scan on store_returns              | 287999764 | 1MB      |      12 | 227383.99
(21 rows)
```

图中计划顶端warning的提示详见**Hint的错误、冲突及告警**的说明。

6.9.3 Join 方式的 Hint

功能描述

指明Join使用的方法，可以为Nested Loop，Hash Join和Merge Join。

语法格式

```
[no] nestloop|hashjoin|mergejoin(table_list)
```

参数说明

- **no**表示hint的join方式不使用。
- **table_list**为表示hint表集合的字符串，该字符串中的表与**join_table_list**相同，只是中间不允许出现括号指定join的优先级。

例如：

no nestloop(t1 t2 t3)表示：生成t1，t2，t3三表连接计划时，不使用nestloop。三表连接计划可能是t2 t3先join，再跟t1 join，或t1 t2先join，再跟t3 join。此hint只hint最后一次join的join方式，对于两表连接的方法不hint。如果需要，可以单独指定，例如：任意表均不允许nestloop连接，且希望t2 t3先join，则增加hint：no nestloop(t2 t3)。

示例

对**示例**中原语句使用如下hint:

```
explain
select /*+ nestloop(store_sales store_returns item) */ i_product_name product_name ...
```

该hint表示：生成store_sales，store_returns和item三表的结果集时，最后的两表关联使用nestloop。生成计划如下所示：

```
 id |                                  operation                                  |   E-rows   | E-memory | E-width |     E-costs
----+-----------------------------------------------------------------------------+------------+----------+---------+-----------------
  1 | ->  Row Adapter                                                             |          6 |          |     273 | 100061693161.06
  2 |    ->  Vector Streaming (type: GATHER)                                      |          6 |          |     273 | 100061693161.06
  3 |       ->  Vector Hash Aggregate                                             |          6 | 16MB     |     273 | 100061693159.40
  4 |          ->  Vector Streaming(type: REDISTRIBUTE)                           |          6 | 1MB      |     169 | 100061693159.36
  5 |             ->  Vector Hash Join (6,22)                                     |          6 | 43MB     |     169 | 100061693158.99
  6 |                ->  Vector Streaming(type: REDISTRIBUTE)                     |          6 | 1MB      |     119 | 100061688591.48
  7 |                   ->  Vector Hash Join (8,21)                               |          6 | 16MB     |     119 | 100061688591.30
  8 |                      ->  Vector Hash Join (9,20)                            |          7 | 27MB     |     123 | 100061687304.04
  9 |                         ->  Vector Streaming(type: REDISTRIBUTE)            |          7 | 1MB      |     123 | 100061677823.27
 10 |                            ->  Vector Hash Join (11,19)                     |          7 | 16MB     |     123 | 100061677823.12
 11 |                               ->  Vector Nest Loop (12,17)                  |          7 | 1MB      |     112 | 100061675546.57
 12 |                                  ->  Vector Hash Join (13,15)               |      13670 | 585MB    |      62 | 6163443.54
 13 |                                     ->  Vector Partition Iterator           | 2879987999 | 1MB      |      66 | 2756066.50
 14 |                                        ->  Partitioned CStore Scan on store_sales | 2879987999 | 1MB      |      66 | 2756066.50
 15 |                                     ->  Vector Partition Iterator           |  287999764 | 1MB      |      12 | 227383.99
 16 |                                        ->  Partitioned CStore Scan on store_returns |  287999764 | 1MB      |      12 | 227383.99
 17 |                                  ->  Vector Materialize                     |        158 | 16MB     |      58 | 4051.28
 18 |                                     ->  CStore Scan on item                 |        158 | 1MB      |      58 | 4051.25
 19 |                               ->  CStore Scan on store                      |      24048 | 1MB      |      19 | 2264.00
 20 |                            ->  CStore Scan on customer                      |   12000000 | 1MB      |       8 | 12923.00
 21 |                         ->  CStore Scan on promotion                        |      36000 | 1MB      |       4 | 1268.50
 22 |                   ->  CStore Scan on customer_address ad2                   |    6000000 | 1MB      |      58 | 5770.00
(22 rows)
```

6.9.4 行数的 Hint

功能描述

指明中间结果集的大小，支持绝对值和相对值的hint。

语法格式

```
rows(table_list #|+|-|* const)
```

参数说明

- **#,+,-,***，进行行数估算hint的四种操作符号。#表示直接使用后面的行数进行hint。+,-,*表示对原来估算的行数进行加、减、乘操作，运算后的行数最小值为1行。table_list为hint对应的单表或多表join结果集，与**Join方式的Hint**中**table_list**相同。
- **const**可以是任意非负数，支持科学计数法。

例如：

rows(t1 #5)表示：指定t1表的结果集为5行。

rows(t1 t2 t3 *1000)表示：指定t1, t2, t3 join完的结果集的行数乘以1000。

建议

- 推荐使用两个表*的hint。对于两个表的采用*操作符的hint，只要两个表出现在join的两端，都会触发hint。例如：设置hint为rows(t1 t2 * 3)，对于(t1 t3 t4)和(t2 t5 t6)join时，由于t1和t2出现在join的两端，所以其join的结果集也会应用该hint规则乘以3。

- rows hint支持在单表、多表、function table及subquery scan table的结果集上指定hint。

示例

对**示例**中原语句使用如下hint:

```
explain
select /*+ rows(store_sales store_returns *50) */ i_product_name product_name ...
```

该hint表示：store_sales，store_returns关联的结果集估算行数在原估算行数基础上乘以50。生成计划如下所示：

```
 id |                                operation                                 |   E-rows   | E-memory | E-width |  E-costs
----+--------------------------------------------------------------------------+------------+----------+---------+------------
  1 | ->  Row Adapter                                                          |        312 |          |     273 | 3401656.58
  2 |    ->  Vector Streaming (type: GATHER)                                   |        312 |          |     273 | 3401656.58
  3 |       ->  Vector Hash Aggregate                                          |        312 | 16MB     |     273 | 3401634.91
  4 |          ->  Vector Streaming(type: REDISTRIBUTE)                        |        313 | 1MB      |     169 | 3401634.39
  5 |             ->  Vector Hash Join (6,21)                                  |        313 | 43MB     |     169 | 3401633.06
  6 |                ->  Vector Streaming(type: REDISTRIBUTE)                  |        313 | 1MB      |     119 | 3397065.38
  7 |                   ->  Vector Hash Join (8,20)                            |        313 | 27MB     |     119 | 3397064.31
  8 |                      ->  Vector Streaming(type: REDISTRIBUTE)            |        328 | 1MB      |     119 | 3387583.37
  9 |                         ->  Vector Hash Join (10,19)                     |        328 | 16MB     |     119 | 3387582.18
 10 |                            ->  Vector Hash Join (11,18)                  |        344 | 16MB     |     123 | 3386294.74
 11 |                               ->  Vector Hash Join (12,14)               |        360 | 19MB     |     112 | 3384018.02
 12 |                                  ->  Vector Partition Iterator           |  287999764 | 1MB      |      12 | 227383.99
 13 |                                     ->  Partitioned CStore Scan on store_returns | 287999764 | 1MB |      12 | 227383.99
 14 |                                  ->  Vector Hash Join (15,17)            |    1516824 | 16MB     |     124 | 3065686.08
 15 |                                     ->  Vector Partition Iterator        | 2879987999 | 1MB      |      66 | 2756066.50
 16 |                                        ->  Partitioned CStore Scan on store_sales | 2879987999 | 1MB |      66 | 2756066.50
 17 |                                     ->  CStore Scan on item              |        158 | 1MB      |      58 | 4051.25
 18 |                                  ->  CStore Scan on store                |      24048 | 1MB      |      19 | 2264.00
 19 |                               ->  CStore Scan on promotion               |      36000 | 1MB      |       4 | 1268.50
 20 |                            ->  CStore Scan on customer                   |   12000000 | 1MB      |       8 | 12923.00
 21 |                      ->  CStore Scan on customer_address ad2             |    6000000 | 1MB      |      58 | 5770.00
(21 rows)
```

第11行算子的估算行数修正为360行，原估算行数为7行（四舍五入后取值）。

6.9.5 Stream 方式的 Hint

功能描述

指明stream使用的方法，可以为broadcast和redistribute，或者直接指定生成gather计划。

语法格式

```
[no] broadcast|redistribute(table_list)
gather(REL|JOIN|ALL)
```

参数说明

- broadcast和redistribute
  - no表示hint的stream方式不使用。
  - table_list为进行stream操作的单表或多表join结果集，见**参数说明**。
- gather

  gather hint可以指定三种计划生成方式：
  - REL：只生成基于基表的gather路径，然后再在CN上执行剩余计划。
  - JOIN：尽可能生成基于join的gather路径，在能下推的join子计划上面（join下面不包含重分布节点）添加gather路径，剩余计划在CN上执行。对于需要重分布节点的join计划则生成不出这种基于join的gather路径，会回退生成基于基表的gather路径。

> **注意**
>
> 在指定Hint(JOIN)后，对于分布表和复制表做连接的情况会导致生成不出来Hint(JOIN)期望的计划，因为优化器已经寻找更优的计划进行替代。

- ALL：基于最优方式选择Gather Rel或Gather Join路径。

示例

对示例中原语句使用如下hint:

```
explain
select /*+ no redistribute(store_sales store_returns item store) leading(((store_sales store_returns item
store) customer)) */ i_product_name product_name ...
```

原计划中，(store_sales store_returns item store)和customer做join时，前者做了重分布，此hint表示禁止前者混合表做重分布，但仍然保持join顺序，则生成计划如下所示：

```
 id |                                operation                                 |   E-rows   | E-memory | E-width |  E-costs
----+--------------------------------------------------------------------------+------------+----------+---------+------------
  1 | ->  Row Adapter                                                          |          6 |          |     273 | 5718448.94
  2 |    ->  Vector Streaming (type: GATHER)                                   |          6 |          |     273 | 5718448.94
  3 |       ->  Vector Hash Aggregate                                          |          6 | 16MB     |     273 | 5718447.27
  4 |          ->  Vector Streaming(type: REDISTRIBUTE)                        |          6 | 1MB      |     169 | 5718447.23
  5 |             ->  Vector Hash Join (6,21)                                  |          6 | 16MB     |     169 | 5718446.86
  6 |                ->  Vector Hash Join (7,20)                               |          7 | 43MB     |     173 | 5717159.60
  7 |                   ->  Vector Streaming(type: REDISTRIBUTE)               |          7 | 1MB      |     123 | 5712592.09
  8 |                      ->  Vector Hash Join (9,18)                         |          7 | 585MB    |     123 | 5712591.93
  9 |                         ->  Vector Hash Join (10,17)                     |          7 | 16MB     |     123 | 3386294.56
 10 |                            ->  Vector Hash Join (11,13)                  |          7 | 19MB     |     112 | 3384018.02
 11 |                               ->  Vector Partition Iterator              |  287999764 | 1MB      |      12 | 227383.99
 12 |                                  ->  Partitioned CStore Scan on store_returns | 287999764 | 1MB |      12 | 227383.99
 13 |                               ->  Vector Hash Join (14,16)               |    1516824 | 16MB     |     124 | 3065686.08
 14 |                                  ->  Vector Partition Iterator           | 2879987999 | 1MB      |      66 | 2756066.50
 15 |                                     ->  Partitioned CStore Scan on store_sales | 2879987999 | 1MB |      66 | 2756066.50
 16 |                                  ->  CStore Scan on item                 |        158 | 1MB      |      58 | 4051.25
 17 |                            ->  CStore Scan on store                      |      24048 | 1MB      |      19 | 2264.00
 18 |                         ->  Vector Streaming(type: BROADCAST)            |  288000000 | 1MB      |       8 | 2176297.36
 19 |                            ->  CStore Scan on customer                   |   12000000 | 1MB      |       8 | 12923.00
 20 |                      ->  CStore Scan on customer_address ad2             |    6000000 | 1MB      |      58 | 5770.00
 21 |                   ->  CStore Scan on promotion                           |      36000 | 1MB      |       4 | 1268.50
(21 rows)
```

对语句进行Gather Hint指定：

1. 生成基表Gather计划 /* +GATHER(REL)*/。

```
openGauss=# explain select /*+ GATHER(REL)*/* from t1, t2, t3 where t1.c2 = t2.c2 and t2.c2 = t3.c2;
 id |              operation               | E-rows | E-width | E-costs
----+--------------------------------------+--------+---------+---------
  1 | ->  Hash Join (2,8)                  |     20 |      36 | 44.10
  2 |    ->  Hash Join (3,5)               |     20 |      24 | 29.22
  3 |       ->  Streaming (type: GATHER)   |     20 |      12 | 14.35
  4 |          ->  Seq Scan on t1          |     20 |      12 | 13.13
  5 |       ->  Hash                       |     20 |      12 | 14.35
  6 |          ->  Streaming (type: GATHER) |     20 |      12 | 14.35
  7 |             ->  Seq Scan on t2       |     20 |      12 | 13.13
  8 |    ->  Hash                          |     20 |      12 | 14.35
  9 |       ->  Streaming (type: GATHER)   |     20 |      12 | 14.35
 10 |          ->  Seq Scan on t3          |     20 |      12 | 13.13
(10 rows)

 Predicate Information (identified by plan id)
-----------------------------------------------
   1 --Hash Join (2,8)
         Hash Cond: (t1.c2 = t3.c2)
   2 --Hash Join (3,5)
         Hash Cond: (t1.c2 = t2.c2)
(4 rows)
```

2. 生成可下推计划的Join Gather计划 /*+ GATHER(REL)*/。

```
openGauss=# explain select /*+ GATHER(JOIN)*/* from t1, t2, t3 where t1.c1 = t2.c1 and t2.c2 = t3.c2;
 id |            operation             | E-rows | E-width | E-costs
----+----------------------------------+--------+---------+---------
  1 | ->  Hash Join (2,7)              |     20 |      36 | 42.37
  2 |    ->  Streaming (type: GATHER)  |     20 |      24 | 27.49
  3 |       ->  Hash Join (4,5)        |     20 |      24 | 26.56
  4 |          ->  Seq Scan on t1      |     20 |      12 | 13.13
  5 |          ->  Hash                |     21 |      12 | 13.13
  6 |             ->  Seq Scan on t2   |     20 |      12 | 13.13
  7 |    ->  Hash                      |     20 |      12 | 14.35
  8 |       ->  Streaming (type: GATHER) |     20 |      12 | 14.35
  9 |          ->  Seq Scan on t3      |     20 |      12 | 13.13
(9 rows)

 Predicate Information (identified by plan id)
-----------------------------------------------
   1 --Hash Join (2,7)
         Hash Cond: (t2.c2 = t3.c2)
   3 --Hash Join (4,5)
         Hash Cond: (t1.c1 = t2.c1)
(4 rows)
```

3. 生成最优方式的Gather计划 /*+ GATHER(ALL)*/。

   会基于最优方式及规则选择GATHER(REL)或者GATHER(JOIN)路径。

```
openGauss=# explain select /*+ GATHER(ALL)*/* from t1, t2, t3 where t1.c1 = t2.c1 and t2.c2 = t3.c2;
 id |            operation             | E-rows | E-width | E-costs
----+----------------------------------+--------+---------+---------
  1 | ->  Hash Join (2,7)              |     20 |      36 | 42.37
  2 |    ->  Streaming (type: GATHER)  |     20 |      24 | 27.49
  3 |       ->  Hash Join (4,5)        |     20 |      24 | 26.56
  4 |          ->  Seq Scan on t1      |     20 |      12 | 13.13
  5 |          ->  Hash                |     21 |      12 | 13.13
  6 |             ->  Seq Scan on t2   |     20 |      12 | 13.13
  7 |    ->  Hash                      |     20 |      12 | 14.35
  8 |       ->  Streaming (type: GATHER) |     20 |      12 | 14.35
  9 |          ->  Seq Scan on t3      |     20 |      12 | 13.13
(9 rows)

 Predicate Information (identified by plan id)
-----------------------------------------------
   1 --Hash Join (2,7)
         Hash Cond: (t2.c2 = t3.c2)
   3 --Hash Join (4,5)
         Hash Cond: (t1.c1 = t2.c1)
(4 rows)
```

6.9.6 Scan 方式的 Hint

功能描述

指明scan使用的方法，可以是tablescan、indexscan和indexonlyscan。

语法格式

```
[no] tablescan|indexscan|indexonlyscan(table [index])
```

参数说明

- **no**表示hint的scan方式不使用。
- **table**表示hint指定的表，只能指定一个表，如果表存在别名应优先使用别名进行hint。
- **index**表示使用indexscan或indexonlyscan的hint时，指定的索引名称，当前只能指定一个。

说明

对于indexscan或indexonlyscan，只有hint的索引属于hint的表时，才能使用该hint。

scan hint支持在行存表、子查询表上指定。

示例

为了hint使用索引扫描，需要首先在表item的i_item_sk列上创建索引，名称为i。

```
create index i on item(i_item_sk);
```

对示例中原语句使用如下hint:

```
explain
select /*+ indexscan(item i) */ i_product_name product_name ...
```

该hint表示：item表使用索引i进行扫描。生成计划如下所示：

```
 id |                                operation                                 |   E-rows   | E-memory | E-width |     E-costs
----+--------------------------------------------------------------------------+------------+----------+---------+-----------------
  1 | ->  Row Adapter                                                          |          6 |          |     273 | 100061674938.26
  2 |    ->  Vector Streaming (type: GATHER)                                   |          6 |          |     273 | 100061674938.26
  3 |       ->  Vector Hash Aggregate                                          |          6 | 16MB     |     273 | 100061674936.59
  4 |          ->  Vector Streaming(type: REDISTRIBUTE)                        |          6 | 1MB      |     169 | 100061674936.55
  5 |             ->  Vector Hash Join (6,21)                                  |          6 | 43MB     |     169 | 100061674936.19
  6 |                ->  Vector Streaming(type: REDISTRIBUTE)                  |          6 | 1MB      |     119 | 100061670368.67
  7 |                   ->  Vector Hash Join (8,20)                            |          6 | 16MB     |     119 | 100061670368.50
  8 |                      ->  Vector Hash Join (9,19)                         |          7 | 27MB     |     123 | 100061669081.23
  9 |                         ->  Vector Streaming(type: REDISTRIBUTE)         |          7 | 1MB      |     123 | 100061659600.47
 10 |                            ->  Vector Hash Join (11,18)                  |          7 | 16MB     |     123 | 100061659600.31
 11 |                               ->  Vector Nest Loop (12,17)               |          7 | 1MB      |     112 | 100061657323.77
 12 |                                  ->  Vector Hash Join (13,15)            |      13670 | 585MB    |      62 | 6163443.54
 13 |                                     ->  Vector Partition Iterator        | 2879987999 | 1MB      |      66 | 2756066.50
 14 |                                        ->  Partitioned CStore Scan on store_sales | 2879987999 | 1MB |      66 | 2756066.50
 15 |                                     ->  Vector Partition Iterator        |  287999764 | 1MB      |      12 | 227383.99
 16 |                                        ->  Partitioned CStore Scan on store_returns |  287999764 | 1MB |      12 | 227383.99
 17 |                                  ->  CStore Index Scan using i on item   |          1 | 1MB      |      58 | 4.01
 18 |                               ->  CStore Scan on store                   |      24048 | 1MB      |      19 | 2264.00
 19 |                            ->  CStore Scan on customer                   |   12000000 | 1MB      |       8 | 12923.00
 20 |                         ->  CStore Scan on promotion                     |      36000 | 1MB      |       4 | 1268.50
 21 |                   ->  CStore Scan on customer_address ad2                |    6000000 | 1MB      |      58 | 5770.00
(21 rows)
```

6.9.7 子链接块名的 hint

功能描述

指明子链接块的名称。

语法格式

```
blockname (table)
```

参数说明

- **table**表示为该子链接块hint的别名的名称。

说明

- blockname hint仅在对应的子链接块没有提升时才会被上层查询使用。目前支持的子链接提升包括IN子链接提升、EXISTS子链接提升和包含Agg等值相关子链接提升。该hint通常会和前面章节提到的hint联合使用。
- 对于FROM关键字后的子查询，则需要使用子查询的别名进行hint，blockname hint不会被用到。
- 如果子链接中含有多个表，则提升后这些表可与外层表以任意优化顺序连接，hint也不会被用到。

示例

```
explain select /*+nestloop(store_sales tt) */ * from store_sales where ss_item_sk in (select /*
+blockname(tt)*/ i_item_sk from item group by 1);
```

该hint表示：子链接的别名为tt，提升后与上层的store_sales表关联时使用nestloop。生成计划如下所示：

```
 id |                       operation                        |   E-rows   | E-memory | E-width |     E-costs
----+--------------------------------------------------------+------------+----------+---------+-----------------
  1 | ->  Row Adapter                                        | 1439994000 |          |     216 | 325105765847.91
  2 |    ->  Vector Streaming (type: GATHER)                 | 1439994000 |          |     216 | 325105765847.91
  3 |       ->  Vector Nest Loop Semi Join (4, 6)            | 1439994000 | 1MB      |     216 | 325026664615.00
  4 |          ->  Vector Partition Iterator                 | 2879987999 | 1MB      |     216 | 2756066.50
  5 |             ->  Partitioned CStore Scan on store_sales | 2879987999 | 1MB      |     216 | 2756066.50
  6 |          ->  Vector Materialize                        |     300000 | 16MB     |       4 | 4176.25
  7 |             ->  Vector Hash Aggregate                  |     300000 | 16MB     |       4 | 3988.75
  8 |                ->  CStore Scan on item                 |     300000 | 1MB      |       4 | 3832.50
(8 rows)
```

6.9.8 运行倾斜的 hint

功能描述

指明查询运行时重分布过程中存在倾斜的重分布键和倾斜值，针对Join和HashAgg运算中的重分布进行优化。

语法格式

- 指定单表倾斜：
  ```
  skew(table (column) [(value)])
  ```
- 指定中间结果倾斜：
  ```
  skew((join_rel) (column) [(value)])
  ```

参数说明

- **table**表示存在倾斜的单个表名。
- **join_rel**表示参与join的两个或多个表，如（t1 t2）表示t1和t2join后的结果存在倾斜。
- **column**表示倾斜表中存在倾斜的一个或多个列。
- **value**表示倾斜的列中存在倾斜的一个或多个值。

说明

- skew hint仅在需要重分布且指定的倾斜信息与查询执行过程中的重分布信息相匹配时才会被使用。
- skew hint受GUC参数skew_option限制，如果参数处于关闭状态，则无法进行skew hint倾斜调优。
- skew hint目前仅处理普通表和子查询类型的表关系，支持基表hint、子查询hint、with as子句hint。对于子查询，无论提升与否都支持在skew hint中使用，这点与其它hint不一样。
- 对于倾斜表，如果定义了别名，则在hint中必须使用别名。
- 对于倾斜列，在不产生歧义的情况下，可以使用原名也可以使用别名。skew hint的column不支持表达式，如果需要指定采用分布键为表达式的重分布存在倾斜，需要将重分布键指定为新的列，以新的列进行hint。
- 对于倾斜值，个数需为列数的整数倍并按列的顺序进行组合，组合的个数不能超过10个。如果各倾斜列的倾斜值的个数不一样，为了满足按列组合，值可以重复指定。如，表t1的c1和c2存在倾斜，c1列的倾斜值只有a1，而c2列的倾斜有b1和b2，则skew hint如下：skew(t1 (c1 c2) ((a1 b1)(a1 b2)))。例中(a1 b1)为一个值组合，NULL可以作为倾斜值出现，每个hint中的值组合不超过十个， 且需为列的整数倍。
- 在Join的重分布优化中，skew hint中的value不可缺省，在HashAgg中可以缺省。
- 对于表、列、值中若指定多个，则同类间需以空格分离。
- 对于倾斜值，不支持在hint中进行类型强转；对于string类型，需要使用单引号。

例如：

- 指定单表倾斜

  每一个skew hint用来表示一个表关系存在的倾斜信息，如果想要指定在查询中的多个表关系存在的倾斜信息，则通过指定多个skew hint实现。

  在指定skew时，包括以下四个场景的用法：

  - 单列单值： skew(t (c1) (v1))

    说明：表关系t的c1列中的v1值在查询执行中存在倾斜。
  - 单列多值：skew(t (c1) (v1 v2 v3 ...))

    说明：表关系t的c1列中的v1、v2、v3…等值在查询执行中存在倾斜。
  - 多列单值：skew(t (c1 c2) (v1 v2))

    说明：表关系t的c1列的v1值和c2列的v2值在查询执行中存在倾斜。
  - 多列多值：skew(t (c1 c2) ((v1 v2) (v3 v4) (v5 v6) ...))

    说明：表关系t的c1列的v1、v3、v5…值和c2列的v2、v4、v6…值在查询执行中存在倾斜。

  须知

  多列多值时，各组倾斜值间也可以不使用括号，如：skew(t (c1 c2) (v1 v2 v3 v4 v5 v6 ...))。是否使用括号必须统一，不可混合，

  如：skew(t (c1 c2) (v1 v2 v3 v4 （ v5 v6 ） ...)) 将会产生语法报错。

- 指定中间结果倾斜

  如果基表不存在倾斜，而是查询执行中的中间结果出现倾斜，则需要通过指定中间结果倾斜的skew hint来进行倾斜的调优。skew((t1 t2) (c1) (v1))

  说明：表关系t1和t2 Join后的结果存在倾斜，倾斜的是t1表的c1列，c1列的倾斜值是v1。

为了避免产生歧义，“c1”只能存在于join_rel的一个表关系中，如果存在同名列则通过别名进行规避。

建议

- 如果查询具有多层，则哪一层出现倾斜，则将hint写在哪一层中。
- 对于提升的子查询，skew hint支持直接使用子查询名进行hint。如果明确子查询提升后的哪一个基表存在倾斜，则直接使用基表进行hint的可用性更高。
- 无论对于表或列，若存在别名，则优先使用别名进行hint。

示例

指定单表倾斜

- 原query中进行hint。

  采用如下查询进行skew hint倾斜调优的举例，查询语句及不带hint的原计划如下所示：

```
explain
with customer_total_return as
(select sr_customer_sk as ctr_customer_sk
,sr_store_sk as ctr_store_sk
,sum(SR_FEE) as ctr_total_return
from store_returns
,date_dim
where sr_returned_date_sk = d_date_sk
and d_year =2000
group by sr_customer_sk
,sr_store_sk)
 select  c_customer_id
from customer_total_return ctr1
,store
,customer
where ctr1.ctr_total_return > (select avg(ctr_total_return)*1.2
from customer_total_return ctr2
where ctr1.ctr_store_sk = ctr2.ctr_store_sk)
and s_store_sk = ctr1.ctr_store_sk
and s_state = 'NM'
and ctr1.ctr_customer_sk = c_customer_sk
order by c_customer_id
limit 100;
```

```
 id |                                  operation                                  |  E-rows   |    E-memory    | E-width |  E-costs
----+-----------------------------------------------------------------------------+-----------+----------------+---------+-----------
  1 | ->  Row Adapter                                                             |       100 |                |      20 | 911254.47
  2 |    ->  Vector Limit                                                         |       100 |                |      20 | 911254.47
  3 |       ->  Vector Streaming (type: GATHER)                                   |      2400 |                |      20 | 911325.75
  4 |          ->  Vector Limit                                                   |      2400 | 1MB            |      20 | 911247.62
  5 |             ->  Vector Sort                                                 |   3684816 | 16MB           |      20 | 911631.21
  6 |                ->  Vector Hash Join (7,29)                                  |   3684817 | 41MB(12374MB)  |      20 | 905379.41
  7 |                   ->  Vector Streaming(type: REDISTRIBUTE)                  |   3684817 | 384KB          |       4 | 883010.31
  8 |                      ->  Vector Hash Join (9,19)                            |   3684817 | 16MB           |       4 | 861302.05
  9 |                         ->  Vector Hash Join (10,18)                        |  11054450 | 16MB           |      44 | 427109.71
 10 |                            ->  Vector Hash Aggregate                        |  50247501 | 397MB(12671MB) |      54 | 395302.57
 11 |                               ->  Vector Streaming(type: REDISTRIBUTE)      |  50247501 | 384KB          |      22 | 358663.76
 12 |                                  ->  Vector Hash Join (13,15)               |  50247501 | 16MB           |      22 | 294300.51
 13 |                                     ->  Vector Partition Iterator           | 287999764 | 1MB            |      26 | 227383.99
 14 |                                        ->  Partitioned CStore Scan on store_returns | 287999764 | 1MB |      26 | 227383.99
 15 |                                     ->  Vector Streaming(type: BROADCAST)   |      8712 | 384KB          |       4 | 975.56
 16 |                                        ->  Vector Partition Iterator        |       363 | 1MB            |       4 | 910.65
 17 |                                           ->  Partitioned CStore Scan on date_dim |       363 | 1MB |       4 | 910.65
 18 |                            ->  CStore Scan on store                         |        44 | 1MB            |       4 | 1006.39
 19 |                         ->  Vector Hash Aggregate                           |       192 | 16MB           |      68 | 426707.38
 20 |                            ->  Vector Subquery Scan on ctr2                 |  50247501 | 1MB            |      36 | 416239.03
 21 |                               ->  Vector Hash Aggregate                     |  50247501 | 397MB(12671MB) |      54 | 395302.57
 22 |                                  ->  Vector Streaming(type: REDISTRIBUTE)   |  50247501 | 384KB          |      22 | 358663.76
 23 |                                     ->  Vector Hash Join (24,26)            |  50247501 | 16MB           |      22 | 294300.51
 24 |                                        ->  Vector Partition Iterator        | 287999764 | 1MB            |      26 | 227383.99
 25 |                                           ->  Partitioned CStore Scan on store_returns | 287999764 | 1MB |      26 | 227383.99
 26 |                                        ->  Vector Streaming(type: BROADCAST)|      8712 | 384KB          |       4 | 975.56
 27 |                                           ->  Vector Partition Iterator     |       363 | 1MB            |       4 | 910.65
 28 |                                              ->  Partitioned CStore Scan on date_dim |       363 | 1MB |       4 | 910.65
 29 |                   ->  CStore Scan on customer                               |  12000000 | 1MB            |      24 | 12923.00
(29 rows)
```

对内层with子句中的HashAgg和外层的Hash Join进行hint指定，带hint的查询如下：

```
explain
with customer_total_return as
```



```
(select /*+ skew(store_returns(sr_store_sk sr_customer_sk)) */sr_customer_sk as ctr_customer_sk
,sr_store_sk as ctr_store_sk
,sum(SR_FEE) as ctr_total_return
from store_returns
,date_dim
where sr_returned_date_sk = d_date_sk
and d_year =2000
group by sr_customer_sk
,sr_store_sk)
 select  /*+ skew(ctr1(ctr_customer_sk)(11))*/  c_customer_id
from customer_total_return ctr1
,store
,customer
where ctr1.ctr_total_return > (select avg(ctr_total_return)*1.2
from customer_total_return ctr2
where ctr1.ctr_store_sk = ctr2.ctr_store_sk)
and s_store_sk = ctr1.ctr_store_sk
and s_state = 'NM'
and ctr1.ctr_customer_sk = c_customer_sk
order by c_customer_id
limit 100;
```

该hint表示：内层with子句中的group by在做HashAgg中进行重分布时存在倾斜，对应原计划的10和21号Hash Agg算子；外层ctr1表的ctr_customer_sk列在做Hash Join中进行重分布时存在倾斜，对应原计划的6号算子。生成计划如下所示：

```
 id |                               operation                                   |  E-rows   |   E-memory      | E-width |  E-costs
----+---------------------------------------------------------------------------+-----------+-----------------+---------+------------
  1 | ->  Row Adapter                                                           |       100 |                 |      20 | 1061778.14
  2 |   ->  Vector Limit                                                        |       100 |                 |      20 | 1061778.14
  3 |     ->  Vector Streaming (type: GATHER)                                   |      2400 |                 |      20 | 1061849.41
  4 |       ->  Vector Limit                                                    |      2400 | 1MB             |      20 | 1061771.29
  5 |         ->  Vector Sort                                                   |   3684816 | 16MB            |      20 | 1062154.87
  6 |           ->  Vector Hash Join (7,31)                                     |   3684817 | 41MB(12344MB)   |      20 | 1055903.08
  7 |             ->  Vector Streaming(type: PART REDISTRIBUTE PART ROUNDROBIN) |   3684817 | 384KB           |       4 | 1013056.49
  8 |               ->  Vector Hash Join (9,20)                                 |   3684817 | 16MB            |       4 | 1000006.10
  9 |                 ->  Vector Hash Join (10,19)                              |  11054450 | 16MB            |      44 | 496461.73
 10 |                   ->  Vector Hash Aggregate                               |  50247501 | 397MB(12010MB)  |      54 | 464654.59
 11 |                     ->  Vector Streaming(type: REDISTRIBUTE)              |  50247501 | 384KB           |      54 | 428015.79
 12 |                       ->  Vector Hash Aggregate                           |  50247501 | 397MB(12010MB)  |      54 | 330939.31
 13 |                         ->  Vector Hash Join (14,16)                      |  50247501 | 16MB            |      22 | 294300.51
 14 |                           ->  Vector Partition Iterator                   | 287999764 | 1MB             |      26 | 227383.99
 15 |                             ->  Partitioned CStore Scan on store_returns  | 287999764 | 1MB             |      26 | 227383.99
 16 |                           ->  Vector Streaming(type: BROADCAST)           |      8712 | 384KB           |       4 | 975.56
 17 |                             ->  Vector Partition Iterator                 |       363 | 1MB             |       4 | 910.65
 18 |                               ->  Partitioned CStore Scan on date_dim     |       363 | 1MB             |       4 | 910.65
 19 |                   ->  CStore Scan on store                                |        44 | 1MB             |       4 | 1006.39
 20 |                 ->  Vector Hash Aggregate                                 |       192 | 16MB            |      68 | 496059.40
 21 |                   ->  Vector Subquery Scan on ctr2                        |  50247501 | 1MB             |      36 | 485591.05
 22 |                     ->  Vector Hash Aggregate                             |  50247501 | 397MB(12010MB)  |      54 | 464654.59
 23 |                       ->  Vector Streaming(type: REDISTRIBUTE)            |  50247501 | 384KB           |      54 | 428015.79
 24 |                         ->  Vector Hash Aggregate                         |  50247501 | 397MB(12010MB)  |      54 | 330939.31
 25 |                           ->  Vector Hash Join (26,28)                    |  50247501 | 16MB            |      22 | 294300.51
 26 |                             ->  Vector Partition Iterator                 | 287999764 | 1MB             |      26 | 227383.99
 27 |                               ->  Partitioned CStore Scan on store_returns| 287999764 | 1MB             |      26 | 227383.99
 28 |                             ->  Vector Streaming(type: BROADCAST)         |      8712 | 384KB           |       4 | 975.56
 29 |                               ->  Vector Partition Iterator               |       363 | 1MB             |       4 | 910.65
 30 |                                 ->  Partitioned CStore Scan on date_dim   |       363 | 1MB             |       4 | 910.65
 31 |             ->  Vector Streaming(type: PART LOCAL PART BROADCAST)         |  12000000 | 384KB           |      24 | 34485.50
 32 |               ->  CStore Scan on customer                                 |  12000000 | 1MB             |      24 | 12923.00
(32 rows)
```

从优化后的计划可以看出：①对于Hash Agg，由于其重分布存在倾斜，所以优化为双层Agg；②对于Hash Join，同样由于其重分布存在倾斜，所以优化为采用新的重分布算子。

- 需要改写query后进行hint

  不带hint的查询和计划如下：

```
explain select count(*) from store_sales_1 group by round(ss_list_price);
```

```
----+-------------------------------------------------------------+---------+----------+---------+----------
  1 | ->  Row Adapter                                             |   16672 |          |      14 | 62261.28
  2 |   ->  Vector Streaming (type: GATHER)                       |   16672 |          |      14 | 62261.28
  3 |     ->  Vector Streaming(type: LOCAL GATHER dop: 1/2)       |   16672 | 32KB     |      14 | 61479.78
  4 |       ->  Vector Hash Aggregate                             |   16672 | 16MB     |      14 | 61452.00
  5 |         ->  Vector Streaming(type: SPLIT REDISTRIBUTE dop: 2/2) | 3112836 | 128KB |       6 | 57498.43
  6 |           ->  CStore Scan on store_sales_1                  | 3112836 | 1MB      |       6 | 21810.25
(6 rows)
```

由于hint中列不支持表达式，在进行倾斜优化时需要借助subquery改写查询，改写后的查询和计划如下：

```
explain
select count(*)
from (select round(ss_list_price),ss_hdemo_sk
from store_sales_1)tmp(a,ss_hdemo_sk)
group by a;
```

```
----+------------------------------------------------------------------+---------+----------+---------+----------
  1 | ->  Row Adapter                                                  |   16672 |          |      14 | 62261.28
  2 |    ->  Vector Streaming (type: GATHER)                           |   16672 |          |      14 | 62261.28
  3 |       ->  Vector Streaming(type: LOCAL GATHER dop: 1/2)          |   16672 | 32KB     |      14 | 61479.78
  4 |          ->  Vector Hash Aggregate                               |   16672 | 16MB     |      14 | 61452.00
  5 |             ->  Vector Streaming(type: SPLIT REDISTRIBUTE dop: 2/2) | 3112836 | 128KB |       6 | 57498.43
  6 |                ->  CStore Scan on store_sales_1                  | 3112836 | 1MB      |       6 | 21810.25
(6 rows)
```

改写注意不要影响到业务逻辑。

采用改写后的查询进行hint，带hint的查询和计划如下：

```
explain
select /*+ skew(tmp(a)) */ count(*)
from (select round(ss_list_price),ss_hdemo_sk
from store_sales_1)tmp(a,ss_hdemo_sk)
group by a;
```

```
 id |                            operation                              | E-rows  | E-memory | E-width | E-costs
----+-------------------------------------------------------------------+---------+----------+---------+----------
  1 | ->  Row Adapter                                                   |   16672 |          |      14 | 27771.82
  2 |    ->  Vector Streaming (type: GATHER)                            |   16672 |          |      14 | 27771.82
  3 |       ->  Vector Streaming(type: LOCAL GATHER dop: 1/2)           |   16672 | 32KB     |      14 | 26990.32
  4 |          ->  Vector Hash Aggregate                                |   16671 | 16MB     |      14 | 26962.54
  5 |             ->  Vector Streaming(type: SPLIT REDISTRIBUTE dop: 2/2) |  66216 | 128KB   |      14 | 26838.09
  6 |                ->  Vector Hash Aggregate                          |   66216 | 16MB     |      14 | 25949.61
  7 |                   ->  CStore Scan on store_sales_1                | 3112836 | 1MB      |       6 | 21810.25
(7 rows)
```

从计划可以看出，对Hash Agg进行倾斜优化后，采用了双层agg实现，大大过滤了进行重分布时的数据量，减少了重分布时间。

此外，需要说明的是，对于子查询，支持使用查询内部的列进行hint，如：

```
explain
select /*+ skew(tmp(b)) */ count(*)
from (select round(ss_list_price) b,ss_hdemo_sk
from store_sales_1)tmp(a,ss_hdemo_sk)
group by a;
```

6.9.9 参数化路径的 Hint

功能描述

指明参数化路径，条件谓词下推方式。

语法格式

```
predpush(src1 src2)
predpush(src, dest)
```

参数说明

- src, src1, src2表示predpush下推candidates一侧表集合。
- dest表示predpush下推所指定的dest表也就是目标表。
- predpush如果没有逗号表示所有表都是candidates表, 如果有逗号就说明同时指定了candidates表和dest表。

说明

使用predpush hint将过滤表达式尽可能移至靠近数据源的位置以达到查询优化的目的。

- 使用predpush hint需要确保rewrite_rule GUC参数包含PREDPUSH|REDPUSHFORCE|PREDPUSHNORMAL选项。
- subquery_block也可以是视图/物化视图。

示例

灵活使用predpush hint可以大幅提高语句的执行效率。示例如下：

```
set rewrite_rule = 'predpushnormal';
explain (costs off) SELECT /*+PREDPUSH(t2, st3)*/ *
FROM t2,
    (SELECT sum(t3.b), t3.a FROM t3, t4 where t3.a = t4.a GROUP BY t3.a) st3
WHERE st3.a = t2.a;
 id |                     operation
----+-----------------------------------------------------
  1 | ->  Streaming (type: GATHER)
  2 |    ->  Nested Loop (3,4)
  3 |       ->  Seq Scan on t2
  4 |       ->  GroupAggregate
  5 |          ->  Nested Loop (6,7)
  6 |             ->  Index Only Scan using t4_a_idx on t4
  7 |             ->  Materialize
  8 |                ->  Index Scan using t3_a_idx on t3
(8 rows)

 Predicate Information (identified by plan id)
-----------------------------------------------
   6 --Index Only Scan using t4_a_idx on t4
        Index Cond: (a = t2.a)
   8 --Index Scan using t3_a_idx on t3
        Index Cond: (a = t2.a)
(4 rows)
```

在未使用predpush hint的情况下，子查询中t3，t4在做join之前没有经过任何来自query block外的处理，所以返回的结果集较大，造成性能浪费。

然而，如上面计划所示，在使用了predpush hint后，t3，t4在做join之前先基于t2表进行了一次条件过滤，join后返回的结果集较小，可以有效提升性能。

6.9.10 Hint 的错误、冲突及告警

Plan Hint的结果会体现在计划的变化上，可以通过explain来查看变化。

Hint中的错误不会影响语句的执行，只是不能生效，该错误会根据语句类型以不同方式提示用户。对于explain语句，hint的错误会以warning形式显示在界面上，对于非explain语句，会以debug1级别日志显示在日志中，关键字为PLANHINT。

hint的错误分为以下类型：

- 语法错误

  语法规则树归约失败，会报错，指出出错的位置。

  例如：hint关键字错误，leading hint或join hint指定2个表以下，其它hint未指定表等。一旦发现语法错误，则立即终止hint的解析，所以此时只有错误前面的解析完的hint有效。

  例如：

  ```
  leading((t1 t2)) nestloop(t1) rows(t1 t2 #10)
  ```

  nestloop(t1)存在语法错误，则终止解析，可用hint只有之前解析的leading((t1 t2))。

- 语义错误
  - 表不存在，存在多个，或在leading或join中出现多次，均会报语义错误。
  - scanhint中的index不存在，会报语义错误。
  - 另外，如果子查询提升后，同一层出现多个名称相同的表，且其中某个表需要被hint，hint会存在歧义，无法使用，需要为相同表增加别名规避。
- hint重复或冲突

  如果存在hint重复或冲突，只有第一个hint生效，其它hint均会失效，会给出提示。

  - hint重复是指，hint的方法及表名均相同。例如：nestloop(t1 t2) nestloop(t1 t2)。
  - hint冲突是指，table list一样的hint，存在不一样的hint，hint的冲突仅对于每一类hint方法检测冲突。

    例如：nestloop (t1 t2) hashjoin (t1 t2)，则后面与前面冲突，此时hashjoin的hint失效。注意：nestloop(t1 t2)和no mergejoin(t1 t2)不冲突。

    须知

    leading hint中的多个表会进行拆解。例如：leading ((t1 t2 t3))会拆解成：leading((t1 t2)) leading(((t1 t2) t3))，此时如果存在leading((t2 t1))，则两者冲突，后面的会被丢弃。（例外：指定内外表的hint若与不指定内外表的hint重复，则始终丢弃不指定内外表的hint。）

- 子链接提升后hint失效

  子链接提升后的hint失效，会给出提示。通常出现在子链接中存在多个表连接的场景。提升后，子链接中的多个表不再作为一个整体出现在join中。
- 列类型不支持重分布
  - 对于skew hint来说，目的是为了进行重分布时的调优，所以当hint列的类型不支持重分布时，hint将无效。
- hint未被使用
  - 非等值join使用hashjoin hint或mergejoin hint
  - 不包含索引的表使用indexscan hint或indexonlyscan hint
  - 通常只有在索引列上使用过滤条件才会生成相应的索引路径，全表扫描将不会使用索引，因此使用indexscan hint或indexonlyscan hint将不会使用
  - indexonlyscan只有输出列仅包含索引列才会使用，否则指定时hint不会被使用
  - 多个表存在等值连接时，仅尝试有等值连接条件的表的连接，此时没有关联条件的表之间的路径将不会生成，所以指定相应的leading，join，rows hint将不使用，例如：t1 t2 t3表join，t1和t2, t2和t3有等值连接条件，则t1和t3不会优先连接，leading(t1 t3)不会被使用。
  - 生成stream计划时，如果表的分布列与join列相同，则不会生成redistribute的计划；如果不同，且另一表分布列与join列相同，只能生成redistribute的计划，不会生成broadcast的计划，指定相应的hint则不会被使用。
  - 如果子链接未被提升，则blockname hint不会被使用。
  - 对于skew hint，hint未被使用可能由于：
    - 计划中不需要进行重分布。
    - hint指定的列为包含分布键。
    - hint指定倾斜信息有误或不完整，如对于join优化未指定值。
    - 倾斜优化的GUC参数处于关闭状态。

6.9.11 Plan Hint 实际调优案例

本节以TPC-DS标准测试的Q24的部分语句为例，在1000X，24DN环境上，说明使用plan hint进行实际调优的过程。示例如下：


```
select avg(netpaid) from
(select c_last_name
,c_first_name
,s_store_name
,ca_state
,s_state
,i_color
,i_current_price
,i_manager_id
,i_units
,i_size
,sum(ss_sales_price) netpaid
from store_sales
,store_returns
,store
,item
,customer
,customer_address
where ss_ticket_number = sr_ticket_number
and ss_item_sk = sr_item_sk
and ss_customer_sk = c_customer_sk
and ss_item_sk = i_item_sk
and ss_store_sk = s_store_sk
and c_birth_country = upper(ca_country)
and s_zip = ca_zip
and s_market_id=7
group by c_last_name
,c_first_name
,s_store_name
,ca_state
,s_state
,i_color
,i_current_price
,i_manager_id
,i_units
,i_size);
```

1. 该语句的初始计划如下，运行时间110s：

```
 id |                                   operation                                    |        A-time         |  A-rows   |  E-rows   |
----+--------------------------------------------------------------------------------+-----------------------+-----------+-----------+
  1 | ->  Row Adapter                                                                | 110324.107            |         1 |         1 |
  2 |    ->  Vector Aggregate                                                        | 110324.093            |         1 |         1 |
  3 |       ->  Vector Streaming (type: GATHER)                                      | 110323.958            |        24 |        24 |
  4 |          ->  Vector Aggregate                                                  | [110179.302,110309.653] |        24 |        24 |
  5 |             ->  Vector Hash Aggregate                                          | [110178.388,110308.515] |    647824 |     16656 |
  6 |                ->  Vector Streaming(type: REDISTRIBUTE)                        | [77616.177,96478.771]   |  666834733 |     16664 |
  7 |                   ->  Vector Hash Join (8,22)                                  | [81727.257,84728.519]   |  666834733 |     16664 |
  8 |                      ->  Vector Streaming(type: REDISTRIBUTE)                  | [78770.520,82021.087]   |  666834733 |     16664 |
  9 |                         ->  Vector Hash Join (10,21)                           | [88066.755,90701.860]   |  666834733 |     16664 |
 10 |                            ->  Vector Streaming(type: BROADCAST)               | [7940.962,21430.725]    |  591882336 |     51360 |
 11 |                               ->  Vector Hash Join (12,20)                     | [2419.995,5319.606]     |   24661764 |      2140 |
 12 |                                  ->  Vector Streaming(type: REDISTRIBUTE)      | [1750.448,4659.581]     |   25258268 |      2241 |
 13 |                                     ->  Vector Hash Join (14,18)               | [15240.666,17159.616]   |   25258268 |      2241 |
 14 |                                        ->  Vector Hash Join (15,17)            | [12112.913,13563.366]   |  252564412 |  472070592 |
 15 |                                           ->  Vector Partition Iterator        | [11148.731,12473.230]   | 2879987999 | 2879987999 |
 16 |                                              ->  Partitioned CStore Scan on public.store_sales | [11097.921,12412.596]   | 2879987999 | 2879987999 |
 17 |                                           ->  CStore Scan on public.store      | [0.447,0.689]           |      2064 |      2064 |
 18 |                                        ->  Vector Partition Iterator           | [296.805,319.014]       |  287999764 |  287999764 |
 19 |                                           ->  Partitioned CStore Scan on public.store_returns | [292.938,314.787]       |  287999764 |  287999764 |
 20 |                                  ->  CStore Scan on public.customer            | [114.358,144.462]       |   12000000 |   12000000 |
 21 |                            ->  CStore Scan on public.customer_address          | [38.426,56.753]         |    6000000 |    6000000 |
 22 |                      ->  CStore Scan on public.item                            | [3.160,5.026]           |     300000 |     300000 |
(22 rows)
```

该计划中，第10层算子使用broadcast性能较差，由于第11层算子估算行数为2140，比实际行数严重低估。错误行数估算主要来源于第13层算子的行数低估，根因是第13层hashjoin中，使用store_sales的(ss_ticket_number, ss_item_sk)列和store_returns的(sr_ticket_number, sr_item_sk)列进行关联，由于缺少多列相关性的估算导致行数严重低估。

2. 使用如下的rows hint进行调优后，计划如下，运行时间318s：

```
select avg(netpaid) from
(select /*+rows(store_sales store_returns * 11270)*/ c_last_name ...
```



```
 id |                                operation                                |          A-time          |  A-rows   |  E-rows
----+-------------------------------------------------------------------------+--------------------------+-----------+-----------+
  1 | ->  Row Adapter                                                         | 318585.246               |         1 |         1 |
  2 |    ->  Vector Aggregate                                                 | 318585.232               |         1 |         1 |
  3 |       ->  Vector Streaming (type: GATHER)                               | 318585.082               |        24 |        24 |
  4 |          ->  Vector Aggregate                                           | [318323.324,318499.290]  |        24 |        24 |
  5 |             ->  Vector Hash Aggregate                                   | [318320.813,318497.054]  |    647824 | 187770504 |
  6 |                ->  Vector Streaming(type: REDISTRIBUTE)                 | [288074.860,305601.698]  |  666834733 | 187770507 |
  7 |                   ->  Vector Hash Join (8,22)                           | [253642.468,315808.664]  |  666834733 | 187770507 |
  8 |                      ->  Vector Hash Join (9,18)                        | [250904.317,315684.018]  |  666834733 | 187770507 |
  9 |                         ->  Vector Streaming(type: REDISTRIBUTE)        | [4552.500,310602.307]    |  275042158 | 147106999 |
 10 |                            ->  Vector Hash Join (11,17)                 | [7658.951,14053.823]     |  275042158 | 147106999 |
 11 |                               ->  Vector Streaming(type: REDISTRIBUTE)  | [3953.255,10264.943]     |  287999764 | 154060900 |
 12 |                                  ->  Vector Hash Join (13,15)           | [28196.188,32838.794]    |  287999764 | 154060900 |
 13 |                                     ->  Vector Partition Iterator       | [11477.673,12324.583]    | 2879987999 | 2879987999 |
 14 |                                        ->  Partitioned CStore Scan on public.store_sales | [11411.382,12250.209] | 2879987999 | 2879987999 |
 15 |                                     ->  Vector Partition Iterator       | [304.188,403.205]        |  287999764 | 287999764 |
 16 |                                        ->  Partitioned CStore Scan on public.store_returns | [299.838,398.255] |  287999764 | 287999764 |
 17 |                                  ->  CStore Scan on public.customer     | [122.246,170.128]        |   12000000 |  12000000 |
 18 |                         ->  Vector Streaming(type: REDISTRIBUTE)        | [57.558,117.461]         |     492915 |    146467 |
 19 |                            ->  Vector Hash Join (20,21)                 | [45.554,96.238]          |     492915 |    146467 |
 20 |                               ->  CStore Scan on public.customer_address | [39.738,89.412]       |    6000000 |   6000000 |
 21 |                               ->  CStore Scan on public.store           | [0.361,1.095]         |       2064 |      2064 |
 22 |                      ->  Vector Streaming(type: BROADCAST)              | [48.986,91.170]          |    7200000 |   7200000 |
 23 |                         ->  CStore Scan on public.item                  | [4.506,6.602]            |     300000 |    300000 |
(23 rows)
```

时间反而劣化了，原因是第8层hashjoin过慢引起第9层redistribute时间过慢导致，其中第9层redistribute并没有数据倾斜，hashjoin慢的原因是由于第18层redistribute后数据倾斜导致。

3. 经过实际数据查证，customer_address的两个join列的不同值数目较少，使用其进行join容易出现数据倾斜，故把customer_address放到最后进行join。使用如下的hint进行调优后，计划如下，运行时间116s：

```
select avg(netpaid) from
(select /*+rows(store_sales store_returns *11270)
leading((store_sales store_returns store item customer) customer_address)*/
c_last_name ...
```

```
 id |                                operation                                |          A-time          |  A-rows   |  E-rows
----+-------------------------------------------------------------------------+--------------------------+-----------+-----------+
  1 | ->  Row Adapter                                                         | 116326.597               |         1 |         1 |
  2 |    ->  Vector Aggregate                                                 | 116326.590               |         1 |         1 |
  3 |       ->  Vector Streaming (type: GATHER)                               | 116326.473               |        24 |        24 |
  4 |          ->  Vector Aggregate                                           | [116157.161,116236.494]  |        24 |        24 |
  5 |             ->  Vector Hash Aggregate                                   | [116155.328,116233.946]  |    647824 | 187770504 |
  6 |                ->  Vector Streaming(type: REDISTRIBUTE)                 | [84103.951,102052.326]   |  666834733 | 187770507 |
  7 |                   ->  Vector Hash Join (8,10)                           | [23229.469,47484.697]    |  666834733 | 187770507 |
  8 |                      ->  Vector Streaming(type: REDISTRIBUTE)           | [38.367,74.930]          |    6000000 |   6000000 |
  9 |                         ->  CStore Scan on public.customer_address      | [69.877,121.460]         |    6000000 |   6000000 |
 10 |                      ->  Vector Streaming(type: REDISTRIBUTE)           | [17404.744,17567.550]    |   24661764 |  24112909 |
 11 |                         ->  Vector Hash Join (12,22)                    | [16123.627,16397.246]    |   24661764 |  24112909 |
 12 |                            ->  Vector Streaming(type: REDISTRIBUTE)     | [15320.663,15741.646]    |   25258268 |  25252751 |
 13 |                               ->  Vector Hash Join (14,21)              | [14962.342,16375.458]    |   25258268 |  25252751 |
 14 |                                  ->  Vector Hash Join (15,19)           | [14449.031,15825.949]    |   25258268 |  25252751 |
 15 |                                     ->  Vector Hash Join (16,18)        | [11439.959,12510.065]    |  252564412 | 472070592 |
 16 |                                        ->  Vector Partition Iterator    | [10531.986,11536.213]    | 2879987999 | 2879987999 |
 17 |                                           ->  Partitioned CStore Scan on public.store_sales | [10483.634,11474.944] | 2879987999 | 2879987999 |
 18 |                                        ->  CStore Scan on public.store  | [0.347,0.463]            |       2064 |      2064 |
 19 |                                     ->  Vector Partition Iterator       | [293.977,365.021]        |  287999764 | 287999764 |
 20 |                                        ->  Partitioned CStore Scan on public.store_returns | [289.936,360.808] |  287999764 | 287999764 |
 21 |                                  ->  CStore Scan on public.item         | [3.109,5.245]            |     300000 |    300000 |
 22 |                               ->  CStore Scan on public.customer        | [113.871,141.791]        |   12000000 |  12000000 |
(22 rows)
```

发现时间基本花在了第6层redistribute算子上，需要进一步优化。

4. 由于最后一层redistribute包含倾斜，所以时间较长。为了避免倾斜，需要将item表放在最后join，由于item表的join并不能使行数减少。修改hint如下并执行，计划如下，运行时间120s：

```
select avg(netpaid) from
(select /*+rows(store_sales store_returns *11270)
leading((customer_address (store_sales store_returns store customer) item))
c_last_name ...
```

```
 id |                                   operation                                    |          A-time           |   A-rows   |   E-rows   |
----+--------------------------------------------------------------------------------+---------------------------+------------+------------+
  1 | ->  Row Adapter                                                                | 120377.258                |          1 |          1 |
  2 |    ->  Vector Aggregate                                                        | 120377.245                |          1 |          1 |
  3 |       ->  Vector Streaming (type: GATHER)                                      | 120377.091                |         24 |         24 |
  4 |          ->  Vector Aggregate                                                  | [120184.884,120301.704]   |         24 |         24 |
  5 |             ->  Vector Hash Aggregate                                          | [120183.119,120297.845]   |     647824 |  187770504 |
  6 |                ->  Vector Streaming(type: REDISTRIBUTE)                        | [87775.682,106070.878]    |  666834733 |  187770507 |
  7 |                   ->  Vector Hash Join (8,22)                                  | [22323.764,49878.523]     |  666834733 |  187770507 |
  8 |                      ->  Vector Hash Join (9,11)                               | [21129.236,45208.255]     |  666834733 |  187770507 |
  9 |                         ->  Vector Streaming(type: REDISTRIBUTE)               | [37.859,75.412]           |    6000000 |    6000000 |
 10 |                            ->  CStore Scan on public.customer_address          | [74.798,114.449]          |    6000000 |    6000000 |
 11 |                         ->  Vector Streaming(type: REDISTRIBUTE)               | [15714.458,15824.928]     |   24661764 |   24112909 |
 12 |                            ->  Vector Hash Join (13,21)                        | [14637.516,14955.464]     |   24661764 |   24112909 |
 13 |                               ->  Vector Streaming(type: REDISTRIBUTE)         | [13898.593,14333.200]     |   25258268 |   25252751 |
 14 |                                  ->  Vector Hash Join (15,19)                  | [14166.917,15378.244]     |   25258268 |   25252751 |
 15 |                                     ->  Vector Hash Join (16,18)              | [11272.239,12052.532]     |  252564412 |  472070592 |
 16 |                                        ->  Vector Partition Iterator          | [10409.566,11127.981]     | 2879987999 | 2879987999 |
 17 |                                           ->  Partitioned CStore Scan on public.store_sales | [10365.838,11077.601] | 2879987999 | 2879987999 |
 18 |                                        ->  CStore Scan on public.store         | [0.431,0.609]             |       2064 |       2064 |
 19 |                                     ->  Vector Partition Iterator             | [343.780,408.254]         |  287999764 |  287999764 |
 20 |                                        ->  Partitioned CStore Scan on public.store_returns | [339.844,403.923] |  287999764 |  287999764 |
 21 |                               ->  CStore Scan on public.customer               | [117.234,163.598]         |   12000000 |   12000000 |
 22 |                      ->  Vector Streaming(type: BROADCAST)                     | [44.571,130.129]          |    7200000 |    7200000 |
 23 |                         ->  CStore Scan on public.item                         | [4.169,6.347]             |     300000 |     300000 |
(23 rows)
```

该计划中的redistribute问题并没有解决，因为第22层item表做了broadcast，导致与customer_address表join后的倾斜并没有被消除掉。

5. 增加如下禁止item表做broadcast的hint，使与customer_address join的表做redistribute（也可以进行join表redistribute的hint），计划如下，运行时间105s：

```
select avg(netpaid) from
(select /*+rows(store_sales store_returns *11270)
leading((customer_address (store_sales store_returns store customer) item))
no broadcast(item)*/
c_last_name ...
```

```
 id |                                   operation                                    |          A-time           |   A-rows   |   E-rows   |
----+--------------------------------------------------------------------------------+---------------------------+------------+------------+
  1 | ->  Row Adapter                                                                | 105854.957                |          1 |          1 |
  2 |    ->  Vector Aggregate                                                        | 105854.948                |          1 |          1 |
  3 |       ->  Vector Streaming (type: GATHER)                                      | 105854.825                |         24 |         24 |
  4 |          ->  Vector Aggregate                                                  | [105706.709,105776.135]   |         24 |         24 |
  5 |             ->  Vector Hash Aggregate                                          | [105705.061,105773.013]   |     647824 |  187770504 |
  6 |                ->  Vector Streaming(type: REDISTRIBUTE)                        | [70701.966,89973.672]     |  666834733 |  187770507 |
  7 |                   ->  Vector Hash Join (8,23)                                  | [71759.500,79018.433]     |  666834733 |  187770507 |
  8 |                      ->  Vector Streaming(type: REDISTRIBUTE)                  | [69794.307,77269.178]     |  666834733 |  187770507 |
  9 |                         ->  Vector Hash Join (10,12)                           | [21443.307,46714.378]     |  666834733 |  187770507 |
 10 |                            ->  Vector Streaming(type: REDISTRIBUTE)            | [41.295,83.419]           |    6000000 |    6000000 |
 11 |                               ->  CStore Scan on public.customer_address       | [70.405,166.072]          |    6000000 |    6000000 |
 12 |                            ->  Vector Streaming(type: REDISTRIBUTE)            | [15689.053,15788.475]     |   24661764 |   24112909 |
 13 |                               ->  Vector Hash Join (14,22)                     | [14517.847,14712.929]     |   24661764 |   24112909 |
 14 |                                  ->  Vector Streaming(type: REDISTRIBUTE)      | [13806.733,14089.770]     |   25258268 |   25252751 |
 15 |                                     ->  Vector Hash Join (16,20)              | [13709.384,15095.449]     |   25258268 |   25252751 |
 16 |                                        ->  Vector Hash Join (17,19)           | [10944.796,11827.285]     |  252564412 |  472070592 |
 17 |                                           ->  Vector Partition Iterator       | [10070.316,10884.728]     | 2879987999 | 2879987999 |
 18 |                                              ->  Partitioned CStore Scan on public.store_sales | [10018.966,10828.990] | 2879987999 | 2879987999 |
 19 |                                           ->  CStore Scan on public.store     | [0.447,0.568]             |       2064 |       2064 |
 20 |                                        ->  Vector Partition Iterator          | [293.042,329.056]         |  287999764 |  287999764 |
 21 |                                           ->  Partitioned CStore Scan on public.store_returns | [288.631,324.782] |  287999764 |  287999764 |
 22 |                                  ->  CStore Scan on public.customer            | [113.735,138.235]         |   12000000 |   12000000 |
 23 |                      ->  CStore Scan on public.item                            | [3.127,5.357]             |     300000 |     300000 |
(23 rows)
```

6. 发现最后一层使用单层Agg，但行数缩减较多。使用相同的hint，同时结合参数best_agg_plan=3进行双层Agg调优，最终计划如下图所示，运行时间94s，完成调优。

```
 id |                                                                operation                                                                |       A-time        |  A-rows   |  E-rows   |
----+-----------------------------------------------------------------------------------------------------------------------------------------+---------------------+-----------+-----------+
  1 | ->  Row Adapter                                                                                                                         | 94004.670           |         1 |         1 |
  2 |    ->  Vector Aggregate                                                                                                                 | 94004.655           |         1 |         1 |
  3 |       ->  Vector Streaming (type: GATHER)                                                                                               | 94004.504           |        24 |        24 |
  4 |          ->  Vector Aggregate                                                                                                           | [93833.832,93928.052] |        24 |        24 |
  5 |             ->  Vector Hash Aggregate                                                                                                   | [93832.460,93926.412] |    647824 | 187770507 |
  6 |                ->  Vector Streaming(type: REDISTRIBUTE)                                                                                 | [93640.866,93787.939] |    647824 | 183912384 |
  7 |                   ->  Vector Hash Aggregate                                                                                             | [93687.544,93791.242] |    647824 | 183912384 |
  8 |                      ->  Vector Hash Join (9,24)                                                                                        | [70025.469,72773.161] |  666834733 | 187770507 |
  9 |                         ->  Vector Streaming(type: REDISTRIBUTE)                                                                        | [68242.223,71275.972] |  666834733 | 187770507 |
 10 |                            ->  Vector Hash Join (11,13)                                                                                 | [21421.136,44830.306] |  666834733 | 187770507 |
 11 |                               ->  Vector Streaming(type: REDISTRIBUTE)                                                                  | [35.444,71.328]       |    6000000 |    6000000 |
 12 |                                  ->  CStore Scan on public.customer_address                                                             | [67.246,119.224]      |    6000000 |    6000000 |
 13 |                               ->  Vector Streaming(type: REDISTRIBUTE)                                                                  | [16089.853,16212.570] |   24661764 |   24112909 |
 14 |                                  ->  Vector Hash Join (15,23)                                                                           | [14822.972,15188.942] |   24661764 |   24112909 |
 15 |                                     ->  Vector Streaming(type: REDISTRIBUTE)                                                            | [14061.867,14604.162] |   25258268 |   25252751 |
 16 |                                        ->  Vector Hash Join (17,21)                                                                     | [13949.756,15492.311] |   25258268 |   25252751 |
 17 |                                           ->  Vector Hash Join (18,20)                                                                  | [10935.742,12160.719] |  252564412 |  472070592 |
 18 |                                              ->  Vector Partition Iterator                                                              | [10052.958,11194.962] | 2879987999 | 2879987999 |
 19 |                                                 ->  Partitioned CStore Scan on public.store_sales                                       | [10008.415,11143.984] | 2879987999 | 2879987999 |
 20 |                                              ->  CStore Scan on public.store                                                            | [0.452,0.839]         |       2064 |       2064 |
 21 |                                           ->  Vector Partition Iterator                                                                 | [298.235,332.736]     |  287999764 |  287999764 |
 22 |                                              ->  Partitioned CStore Scan on public.store_returns                                        | [294.067,327.629]     |  287999764 |  287999764 |
 23 |                                        ->  CStore Scan on public.customer                                                               | [114.377,145.156]     |   12000000 |   12000000 |
 24 |                                  ->  CStore Scan on public.item                                                                         | [3.150,3.530]         |     300000 |     300000 |
(24 rows)
```

如果有统计信息变更引起的查询劣化，可以考虑用plan hint来调整到之前的查询计划。这里以TPCH-Q17为例，在收集default_statistics_target设置为–2的统计信息之后，计划相比于默认统计信息发生劣化。

1. 默认统计信息（default_statistics_target设置为100）的计划如下：

```
 id |                                                                operation                                                                |        A-time
----+-----------------------------------------------------------------------------------------------------------------------------------------+---------------------
  1 | ->  Row Adapter                                                                                                                         | 265006.779
  2 |    ->  Vector Aggregate                                                                                                                 | 265006.764
  3 |       ->  Vector Streaming (type: GATHER)                                                                                               | 265006.071
  4 |          ->  Vector Aggregate                                                                                                           | [263699.512,264503.084]
  5 |             ->  Vector Hash Join (6,17)                                                                                                 | [263676.665,264477.932]
  6 |                ->  Vector Streaming(type: LOCAL GATHER dop: 1/4)                                                                        | [1.998,7.594]
  7 |                   ->  Vector Hash Aggregate                                                                                             | [201775.393,202432.672]
  8 |                      ->  Vector Streaming(type: SPLIT REDISTRIBUTE dop: 4/4)                                                            | [201567.130,202231.524]
  9 |                         ->  Vector Hash Join (10,12)                                                                                    | [170675.231,199908.410]
 10 |                            ->  Vector Partition Iterator                                                                                | [34847.797,51968.266]
 11 |                               ->  Partitioned CStore Scan on tpch10wx_col.lineitem                                                      | [33805.013,51137.657]
 12 |                            ->  Vector Hash Aggregate                                                                                    | [23283.387,25359.493]
 13 |                               ->  Vector Streaming(type: SPLIT BROADCAST dop: 4/4)                                                      | [12850.624,14608.515]
 14 |                                  ->  Vector Hash Aggregate                                                                              | [2690.439,3616.623]
 15 |                                     ->  Vector Partition Iterator                                                                       | [2659.700,3579.390]
 16 |                                        ->  Partitioned CStore Scan on tpch10wx_col.part                                                 | [2642.213,3559.093]
 17 |                ->  Vector Streaming(type: REDISTRIBUTE dop: 1/4)                                                                        | [262300.732,262961.078]
 18 |                   ->  Vector Hash Join (19,21)                                                                                          | [225749.727,260990.322]
 19 |                      ->  Vector Partition Iterator                                                                                      | [40046.078,56220.694]
 20 |                         ->  Partitioned CStore Scan on tpch10wx_col.lineitem                                                            | [39204.414,55328.448]
 21 |                      ->  Vector Streaming(type: SPLIT BROADCAST dop: 4/4)                                                               | [55748.177,61987.136]
 22 |                         ->  Vector Partition Iterator                                                                                   | [3042.864,3873.942]
 23 |                            ->  Partitioned CStore Scan on tpch10wx_col.part                                                             | [3027.023,3848.159]
(23 rows)
```

2. 统计信息变更（default_statistics_target设置为–2）的计划如下：

```
 id |                                                                operation                                                                |        A-time
----+-----------------------------------------------------------------------------------------------------------------------------------------+---------------------
  1 | ->  Row Adapter                                                                                                                         | 1440492.994
  2 |    ->  Vector Aggregate                                                                                                                 | 1440492.982
  3 |       ->  Vector Streaming (type: GATHER)                                                                                               | 1440491.021
  4 |          ->  Vector Streaming(type: LOCAL GATHER dop: 1/6)                                                                              | [1439737.284,1440008.568]
  5 |             ->  Vector Aggregate                                                                                                        | [1439008.369,1439854.148]
  6 |                ->  Vector Hash Join (7,18)                                                                                              | [1439006.016,1439851.619]
  7 |                   ->  Vector Streaming(type: LOCAL BROADCAST dop: 6/6)                                                                  | [2.932,139.405]
  8 |                      ->  Vector Hash Aggregate                                                                                          | [190452.312,195910.748]
  9 |                         ->  Vector Streaming(type: SPLIT REDISTRIBUTE dop: 6/6)                                                         | [190171.929,195653.119]
 10 |                            ->  Vector Hash Join (11,13)                                                                                 | [161076.195,178831.123]
 11 |                               ->  Vector Partition Iterator                                                                             | [27306.318,45564.565]
 12 |                                  ->  Partitioned CStore Scan on tpch10wx_col.lineitem                                                   | [26752.444,44912.020]
 13 |                               ->  Vector Hash Aggregate                                                                                 | [35601.624,39812.058]
 14 |                                  ->  Vector Streaming(type: SPLIT BROADCAST dop: 6/6)                                                   | [23096.460,27057.137]
 15 |                                     ->  Vector Hash Aggregate                                                                           | [2372.587,3052.445]
 16 |                                        ->  Vector Partition Iterator                                                                    | [2345.381,3012.732]
 17 |                                           ->  Partitioned CStore Scan on tpch10wx_col.part                                              | [2329.874,2989.393]
 18 |                   ->  Vector Hash Join (19,22)                                                                                          | [1437388.414,1438470.781]
 19 |                      ->  Vector Streaming(type: SPLIT REDISTRIBUTE dop: 6/6)                                                            | [1392693.529,1408571.859]
 20 |                         ->  Vector Partition Iterator                                                                                   | [29065.204,41264.514]
 21 |                            ->  Partitioned CStore Scan on tpch10wx_col.lineitem                                                         | [28212.219,40133.491]
 22 |                      ->  Vector Streaming(type: LOCAL REDISTRIBUTE dop: 6/6)                                                            | [2570.841,3438.567]
 23 |                         ->  Vector Partition Iterator                                                                                   | [2447.569,3276.369]
 24 |                            ->  Partitioned CStore Scan on tpch10wx_col.part                                                             | [2432.124,3263.641]
(24 rows)
```

3. 经过对比，劣化的原因主要为lineitem和part表join时stream类型由BroadCast变更为Redistribute导致。可以对语句进行stream方式的hint来调整到之前的计划，例如：

```
select /*+ no redistribute(part lineitem) */
    sum(l_extendedprice) / 7.0 as avg_yearly
from
    lineitem,
    part
where
    p_partkey = l_partkey
    and p_brand = 'Brand#23'
    and p_container = 'MED BOX'
    and l_quantity < (
        select
            0.2 * avg(l_quantity)
        from
            lineitem
        where
            l_partkey = p_partkey
    );
```

6.9.12 优化器 GUC 参数的 Hint

功能描述

设置本次查询执行内生效的查询优化相关GUC参数。hint的推荐使用场景可以参考各guc参数的说明，此处不作赘述。

语法格式

```
set(param value)
```

参数说明

- **param**：表示参数名。
- **value**：表示参数的取值。
- 目前支持使用Hint设置生效的参数有：
  - 布尔类：

    enable_bitmapscan, enable_hashagg, enable_hashjoin, enable_indexscan, enable_indexonlyscan, enable_material, enable_mergejoin, enable_nestloop, enable_index_nestloop, enable_seqscan, enable_sort, enable_tidscan, enable_stream_operator, enable_stream_recursive, enable_broadcast, enable_fast_query_shipping, enable_trigger_shipping, enable_remotejoin, enable_remotegroup, enable_remotelimit, enable_remotesort
  - 整型类：

    best_agg_plan, query_dop
  - 浮点类：

    cost_weight_index, default_limit_rows, seq_page_cost, random_page_cost, cpu_tuple_cost, cpu_index_tuple_cost, cpu_operator_cost, effective_cache_size
  - 字符串类：

    node_name

通过设置node_name可以指定当前的sql下发到node_name对应的dn上去执行。

示例：

```
select /*+ set(node_name datanode1) */ from table_name;
```

其中，datanode1是从 pgxc_node 系统表里查询出来的数据节点的名称（不用加引号），table_name 是表名。该查询表示直接去datanode1上执行查询。

须知

- node_name只支持在select语句里设置，如果在其他语句里设置将会不生效。
- node_name只支持设置data node名字，不支持设置coodninator名字。
- node_name不支持通过SET语句进行修改，只能用在plan hint里。
- node_name不支持通过gs_guc进行修改。
- node_name仅支持简单查询语句，不支持带union，union all, 子查询，多表关联等复杂查询语句。
- 支持普通用户执行。
- 不支持与行级访问控制同时使用，同时使用会报错。

说明

- 设置不在白名单中的参数，参数取值不合法，或hint语法错误时，不会影响查询执行的正确性。使用explain(verbose on)执行可以看到hint解析错误的报错提示。
- GUC参数的hint只在最外层查询生效，子查询内的GUC参数hint不生效。
- 视图定义内的GUC参数hint不生效。
- CREATE TABLE ... AS ... 查询最外层的GUC参数hint可以生效。

6.9.13 Custom Plan 和 Generic Plan 选择的 Hint

功能描述

对于以PBE方式执行的查询语句和DML语句，优化器会基于规则、代价、参数等因素选择生成Custom Plan或Generic Plan执行。用户可以通过use_cplan/use_gplan的hint指定使用哪种计划执行方式。

语法格式

- 指定使用Custom Plan：
  ```
  use_cplan
  ```
- 指定使用Generic Plan：
  ```
  use_gplan
  ```

  说明

  - 对于非PBE方式执行的SQL语句，设置本hint不会影响执行方式。
  - 本Hint的优先级仅高于基于代价的选择和plan_cache_mode参数，即plan_cache_mode无法强制选择执行方式的语句本hint也无法生效。

示例

强制使用Custom Plan

```
set enable_fast_query_shipping = off;
create table t (a int, b int, c int);
prepare p as select /*+ use_cplan */ * from t where a = $1;
explain execute p(1);
```

计划如下。可以看到过滤条件为入参的实际值，即此计划为Custom Plan。

```
                              QUERY PLAN
-----------------------------------------------------------------
 Streaming (type: GATHER)  (cost=0.06..13.26 rows=1 width=12)
   Node/s: datanode1
   ->  Seq Scan on t  (cost=0.00..13.16 rows=1 width=12)
         Filter: (a = 1)
(4 rows)
```

强制使用Generic Plan

```
deallocate p;
prepare p as select /*+ use_gplan */ * from t where a = $1;
explain execute p(1);
```

计划如下。可以看到过滤条件为待填充的入参，即此计划为Generic Plan。

```
                              QUERY PLAN
-----------------------------------------------------------------
 Streaming (type: GATHER)  (cost=0.06..13.26 rows=1 width=12)
   Node/s: All datanodes
   ->  Seq Scan on t  (cost=0.00..13.16 rows=1 width=12)
         Filter: (a = $1)
(4 rows)
```

6.9.14 指定子查询不展开的 Hint

功能描述

数据库在对查询进行逻辑优化时通常会将可以提升的子查询提升到上层来避免嵌套执行，但对于某些本身选择率较低且可以使用索引过滤访问页面的子查询，嵌套执行不会导致性能下降过多，而提升之后扩大了查询路径的搜索范围，可能导致性能变差。对于此类情况，可以使用no_expand Hint进行调试。大多数情况下不建议使用此hint。

语法格式

```
no_expand
```

示例

正常的查询执行

```
explain select * from t1 where t1.a in (select t2.a from t2);
```

计划


```
                                  QUERY PLAN
------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=0.06..2.13 rows=2 width=12)
   Node/s: All datanodes
   ->  Nested Loop Semi Join  (cost=0.00..2.03 rows=2 width=12)
         Join Filter: (t1.a = t2.a)
         ->  Seq Scan on t1  (cost=0.00..1.01 rows=1 width=12)
         ->  Seq Scan on t2  (cost=0.00..1.01 rows=1 width=4)
(6 rows)
```

加入no_expand

```
explain select * from t1 where t1.a in (select /*+ no_expand*/ t2.a from t2);
```

计划

```
                                  QUERY PLAN
------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=1.09..2.13 rows=1 width=12)
   Node/s: All datanodes
   ->  Seq Scan on t1  (cost=1.02..2.04 rows=1 width=12)
         Filter: (hashed SubPlan 1)
         SubPlan 1
           ->  Materialize  (cost=0.00..1.02 rows=4 width=4)
                 ->  Streaming(type: BROADCAST)  (cost=0.00..1.01 rows=2 width=4)
                       Spawn on: All datanodes
                       ->  Seq Scan on t2  (cost=0.00..1.01 rows=1 width=4)
(9 rows)
```

6.9.15 指定不使用全局计划缓存的 Hint

功能描述

全局计划缓存打开时，可以通过no_gpc Hint来强制单个查询语句不在全局共享计划缓存，只保留会话生命周期的计划缓存。

语法格式

```
no_gpc
```

说明

本参数仅在enable_global_plancache=on时对PBE执行的语句生效。

示例

```
openGauss=# deallocate all;
DEALLOCATE ALL
openGauss=# prepare insert_nogpc as insert /*+ no_gpc */ into t1 select c1, c2 from t2 where c1 = $1;
PREPARE
openGauss=# execute insert_nogpc(1);
INSERT 0 1
openGauss=# select * from dbe_perf.global_plancache_status where schema_name = 'schema_hint_iud' order by 1,2;
 nodename | query | refcount | valid | databaseid | schema_name | params_num | func_id
----------+-------+----------+-------+------------+-------------+------------+---------
(0 rows)
```

dbe_perf.global_plancache_status视图中无结果即没有计划被全局缓存。

6.9.16 同层参数化路径的 Hint

功能描述

通过predpush_same_level Hint来指定同层表或物化视图之间参数化路径生成。

跨层参数化路径hint请参考**参数化路径的Hint**。

语法格式

```
predpush_same_level(src, dest)
predpush_same_level(src1 src2 ..., dest)
```

说明

本参数仅在rewrite_rule中的predpushforce选项打开时生效。

示例

查看下面的计划示例需要设置以下参数：

```
set enable_fast_query_shipping = off;
set enable_stream_operator = on;
```

准备参数和表及索引：

```
openGauss=# set rewrite_rule = 'predpushforce';
SET
openGauss=# create table t1(a int, b int) distribute by hash(a);
CREATE TABLE
openGauss=# create table t2(a int, b int) distribute by hash(a);
CREATE TABLE
openGauss=# create index idx1 on t1(a);
CREATE INDEX
openGauss=# create index idx2 on t2(a);
CREATE INDEX
```

执行语句查看计划：

```
openGauss=# explain select * from t1, t2 where t1.a = t2.a;
                            QUERY PLAN
------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=18.25..77.00 rows=1000 width=16)
   Node/s: All datanodes
   ->  Hash Join  (cost=14.25..30.12 rows=1000 width=16)
         Hash Cond: (t1.a = t2.a)
         ->  Seq Scan on t1  (cost=0.00..9.00 rows=1000 width=8)
         ->  Hash  (cost=8.00..8.00 rows=1000 width=8)
               ->  Seq Scan on t2  (cost=0.00..8.00 rows=1000 width=8)
(7 rows)
```

可以看到t1.a = t2.a条件过滤在Join上面，此时可以通过predpush_same_level(t1, t2)将条件下推至t2的扫描算子上：

```
openGauss=# explain select /*+predpush_same_level(t1, t2)*/ * from t1, t2 where t1.a = t2.a;
                              QUERY PLAN
------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=4.00..263.88 rows=1000 width=16)
   Node/s: All datanodes
   ->  Nested Loop  (cost=0.00..217.00 rows=1000 width=16)
         ->  Seq Scan on t1  (cost=0.00..9.00 rows=1000 width=8)
         ->  Index Scan using idx2 on t2  (cost=0.00..0.41 rows=1 width=8)
               Index Cond: (a = t1.a)
(6 rows)
```


须知

- predpush_same_level可以指定多个src，但是所有的src必须在同一个条件中。
- 如果指定的src和dest条件不存在，或该条件不符合参数化路径要求，则本hint不生效。
- 如果dest扫描算子上存在stream算子，则本hint不生效。

6.10 检查隐式转换的性能问题

在某些场景下，数据类型的隐式转换可能会导致潜在的性能问题。请看如下的场景：

```
SET enable_fast_query_shipping = off;
CREATE TABLE t1(c1 VARCHAR, c2 VARCHAR);
CREATE INDEX on t1(c1);
EXPLAIN verbose SELECT * FROM t1 WHERE c1 = 10;
```

上述查询的执行计划如下：

```
                            QUERY PLAN
------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=0.06..13.29 rows=1 width=64)
   Output: c1, c2
   Node/s: All datanodes
   ->  Seq Scan on public.t1  (cost=0.00..13.20 rows=1 width=64)
         Output: c1, c2
         Distribute Key: c1
         Filter: ((t1.c1)::bigint = 10)
(7 rows)
```

c1的数据类型是varchar，当查询的过滤条件为c1 = 10时，优化器默认将c1隐式转换为bigint类型，导致两个后果：

- 不能进行DN裁剪，计划下发到所有DN上执行。
- 计划中不能使用Index Scan方式扫描数据。

这会引起潜在的性能问题。

当知道了问题原因后，可以做针对性的SQL改写。对于上面的场景，只要将过滤条件中的常量显示转换为varchar类型，结果如下：

```
EXPLAIN verbose SELECT * FROM t1 WHERE c1 = 10::varchar;
```

```
                                  QUERY PLAN
------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=0.06..8.36 rows=1 width=64)
   Output: c1, c2
   Node/s: datanode2
   ->  Index Scan using t1_c1_idx on public.t1  (cost=0.00..8.27 rows=1 width=64)
         Output: c1, c2
         Distribute Key: c1
         Index Cond: ((t1.c1)::text = '10'::text)
(7 rows)
```

为了提前识别隐式类型转换可能带来的性能影响，GaussDB提供了一个guc option：check_implicit_conversions。打开该参数后，对于查询中出现的隐式类型转换的索引

列，在路径生成阶段进行检查，如果发现索引列没有生成候选的索引扫描路径，则会通过报错的形式提示给用户。举例如下：

```
SET check_implicit_conversions = on;
SELECT * FROM t1 WHERE c1 = 10;
ERROR:  There is no optional index path for index column: "t1"."c1".
Please check for potential performance problem.
```

说明

- 参数check_implicit_conversions只用于检查隐式类型转换引起的潜在性能问题，在正式生产环境中请关闭该参数（该参数默认关闭）。
- 在将check_implicit_conversions打开时，必须同时关闭enable_fast_query_shipping参数，否则由于后一个参数的作用，无法查看对隐式类型转换修复的结果。
- 一个表的候选路径可能包括seq scan和index scan等多个可能的数据扫描方式，最终执行计划使用的表扫描方式是由执行计划的代价来决定的，因此即使生成了索引扫描的候选路径，也可能生成的最终执行计划中使用其它扫描方式。

6.11 实际调优案例

6.11.1 案例：选择合适的分布列

现象描述

表定义如下：

```
CREATE TABLE t1 (a int, b int);
CREATE TABLE t2 (a int, b int);
```

执行如下查询：

```
SELECT * FROM t1, t2 WHERE t1.a = t2.b;
```

优化分析

如果将a作为t1和t2的分布列：

```
CREATE TABLE t1 (a int, b int) DISTRIBUTE BY HASH (a);
CREATE TABLE t2 (a int, b int) DISTRIBUTE BY HASH (a);
```

则执行计划将存在“Streaming”，导致DN之间存在较大通信数据量，如图6-10所示。

图 6-10 选择合适的分布列案例（一）

```
openGauss => explain select * from t1, t2 where t1.a = t2.b;
                                  QUERY PLAN
------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=245.40..582.15 rows=240 width=16)
   Node/s: All datanodes
   ->  Hash Join  (cost=10.22..24.26 rows=10 width=16)
         Hash Cond: (t1.a = t2.b)
         ->  Seq Scan on t1  (cost=0.00..10.10 rows=10 width=8)
         ->  Hash  (cost=3.79..3.79 rows=10 width=8)
               ->  Streaming(type: REDISTRIBUTE)  (cost=0.00..3.79 rows=10 width=8)
                     Spawn on: All datanodes
                     ->  Seq Scan on t2  (cost=0.00..10.10 rows=10 width=8)
(9 rows)
```

如果将a作为t1的分布列，将b作为t2的分布列：


```
CREATE TABLE t1 (a int, b int) DISTRIBUTE BY HASH (a);
CREATE TABLE t2 (a int, b int) DISTRIBUTE BY HASH (b);
```

则执行计划将不包含“Streaming”，减少DN之间存在的通信数据量，从而提升查询性能，如图6-11所示。

图 6-11 选择合适的分布列案例（二）

```
openGauss=> explain select * from t1, t2 where t1.a = t2.b;
                                QUERY PLAN
---------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=245.40..491.10 rows=240 width=16)
   Node/s: All datanodes
   ->  Hash Join  (cost=10.22..20.46 rows=10 width=16)
         Hash Cond: (t1.a = t2.b)
         ->  Seq Scan on t1  (cost=0.00..10.10 rows=10 width=8)
         ->  Hash  (cost=10.10..10.10 rows=10 width=8)
               ->  Seq Scan on t2  (cost=0.00..10.10 rows=10 width=8)
(7 rows)
```

6.11.2 案例：建立合适的索引

现象描述

查询与销售部所有员工的信息：

```
--建表
CREATE TABLE staffs (staff_id NUMBER(6) NOT NULL, first_name VARCHAR2(20), last_name
VARCHAR2(25), employment_id VARCHAR2(10), section_id NUMBER(4), state_name VARCHAR2(10), city
VARCHAR2(10));
CREATE TABLE sections(section_id NUMBER(4), place_id NUMBER(4), section_name VARCHAR2(20));
CREATE TABLE states(state_id NUMBER(4));
CREATE TABLE places(place_id NUMBER(4), state_id  NUMBER(4));
--优化前查询
EXPLAIN SELECT staff_id,first_name,last_name,employment_id,state_name,city
FROM staffs,sections,states,places
WHERE sections.section_name='Sales'
AND staffs.section_id = sections.section_id
AND sections.place_id = places.place_id
AND places.state_id = states.state_id
ORDER BY staff_id;
--创建索引
CREATE INDEX loc_id_pk ON places(place_id);
CREATE INDEX state_c_id_pk ON states(state_id);
--优化后查询
EXPLAIN SELECT staff_id,first_name,last_name,employment_id,state_name,city
FROM staffs,sections,states,places
WHERE sections.section_name='Sales'
AND staffs.section_id = sections.section_id
AND sections.place_id = places.place_id
AND places.state_id = states.state_id
ORDER BY staff_id;
```

优化分析

在优化前，没有创建places.place_id和states.state_id索引，执行计划如下：

```
 id |                      operation                       | E-rows | E-width | E-costs
----+------------------------------------------------------+--------+---------+---------
  1 | ->  Streaming (type: GATHER)                         |      2 |     254 | 54.08
  2 |    ->  Sort                                          |      2 |     254 | 53.90
  3 |       ->  Nested Loop (4,5)                          |      2 |     254 | 53.88
```



```
 4 |         ->  Seq Scan on staffs                          |     20 |     266 | 13.13
 5 |         ->  Materialize                                 |      4 |      12 | 40.37
 6 |           ->  Streaming(type: BROADCAST)                |      4 |      12 | 40.36
 7 |             ->  Nested Loop (8,9)                       |      2 |      12 | 40.20
 8 |               ->  Seq Scan on states                    |     20 |      12 | 13.13
 9 |               ->  Materialize                           |      2 |      24 | 26.69
10 |                 ->  Streaming(type: REDISTRIBUTE)       |      2 |      24 | 26.68
11 |                   ->  Nested Loop (12,14)               |      2 |      24 | 26.57
12 |                     ->  Streaming(type: REDISTRIBUTE)   |      1 |      24 | 13.28
13 |                       ->  Seq Scan on sections          |      1 |      24 | 13.16
14 |                     ->  Seq Scan on places              |     20 |      24 | 13.13
(14 rows)

        Predicate Information (identified by plan id)
-------------------------------------------------------------
  3 --Nested Loop (4,5)
        Join Filter: (sections.section_id = staffs.section_id)
  7 --Nested Loop (8,9)
        Join Filter: (places.state_id = states.state_id)
 11 --Nested Loop (12,14)
        Join Filter: (sections.place_id = places.place_id)
 13 --Seq Scan on sections
        Filter: ((section_name)::text = 'Sales'::text)
(8 rows)
```

建议在places.place_id和states.state_id列上建立2个索引（参考**现象描述**），执行计划如下：

```
 id |                          operation                          | E-rows | E-width | E-costs
----+-------------------------------------------------------------+--------+---------+---------
  1 | ->  Streaming (type: GATHER)                                |      2 |     254 | 42.26
  2 |   ->  Sort                                                  |      2 |     254 | 42.08
  3 |     ->  Nested Loop (4,5)                                   |      2 |     254 | 42.06
  4 |       ->  Seq Scan on staffs                                |     20 |     266 | 13.13
  5 |       ->  Materialize                                       |      4 |      12 | 28.55
  6 |         ->  Streaming(type: BROADCAST)                      |      4 |      12 | 28.54
  7 |           ->  Nested Loop (8,13)                            |      2 |      12 | 28.38
  8 |             ->  Streaming(type: REDISTRIBUTE)               |      2 |      24 | 21.66
  9 |               ->  Nested Loop (10,12)                       |      2 |      24 | 21.56
 10 |                 ->  Streaming(type: REDISTRIBUTE)           |      1 |      24 | 13.28
 11 |                   ->  Seq Scan on sections                  |      1 |      24 | 13.16
 12 |                 ->  Index Scan using loc_id_pk on places    |      1 |      24 | 8.27
 13 |               ->  Index Only Scan using state_c_id_pk on states |      1 |      12 | 3.35
(13 rows)

        Predicate Information (identified by plan id)
-------------------------------------------------------------
  3 --Nested Loop (4,5)
        Join Filter: (sections.section_id = staffs.section_id)
 11 --Seq Scan on sections
        Filter: ((section_name)::text = 'Sales'::text)
 12 --Index Scan using loc_id_pk on places
        Index Cond: (place_id = sections.place_id)
 13 --Index Only Scan using state_c_id_pk on states
        Index Cond: (state_id = places.state_id)
(8 rows)
```

6.11.3 案例：增加 JOIN 列非空条件

现象描述

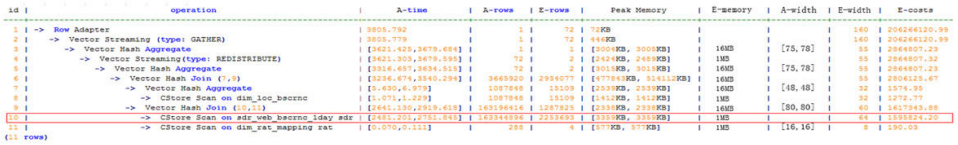
```
SELECT
 *
FROM
( ( SELECT
  STARTTIME STTIME,
  SUM(NVL(PAGE_DELAY_MSEL,0)) PAGE_DELAY_MSEL,
```



```
 SUM(NVL(PAGE_SUCCEED_TIMES,0)) PAGE_SUCCEED_TIMES,
 SUM(NVL(FST_PAGE_REQ_NUM,0)) FST_PAGE_REQ_NUM,
 SUM(NVL(PAGE_AVG_SIZE,0)) PAGE_AVG_SIZE,
 SUM(NVL(FST_PAGE_ACK_NUM,0)) FST_PAGE_ACK_NUM,
 SUM(NVL(DATATRANS_DW_DURATION,0)) DATATRANS_DW_DURATION,
 SUM(NVL(PAGE_SR_DELAY_MSEL,0)) PAGE_SR_DELAY_MSEL
FROM
 PS.SDR_WEB_BSCRNC_1DAY SDR
 INNER JOIN (SELECT
     BSCRNC_ID,
     BSCRNC_NAME,
     ACCESS_TYPE,
     ACCESS_TYPE_ID
    FROM
     nethouse.DIM_LOC_BSCRNC
    GROUP BY
     BSCRNC_ID,
     BSCRNC_NAME,
     ACCESS_TYPE,
     ACCESS_TYPE_ID) DIM
 ON SDR.BSCRNC_ID = DIM.BSCRNC_ID
 AND DIM.ACCESS_TYPE_ID IN (0,1,2)
 INNER JOIN nethouse.DIM_RAT_MAPPING RAT
 ON (RAT.RAT = SDR.RAT)
WHERE
 ( (STARTTIME >= 1461340800
 AND STARTTIME < 1461427200) )
 AND RAT.ACCESS_TYPE_ID IN (0,1,2)
 --and SDR.BSCRNC_ID is not null
GROUP BY
 STTIME ) ) ;
```

执行计划如图6-12所示。

图 6-12 增加 JOIN 列非空条件（一）



优化分析

1. 分析执行计划图6-12可知，在顺序扫描阶段耗时较多。
2. 多表JOIN中，由于表PS.SDR_WEB_BSCRNC_1DAY的JOIN列“BSCRNC_ID”存在大量空值，JOIN性能差。

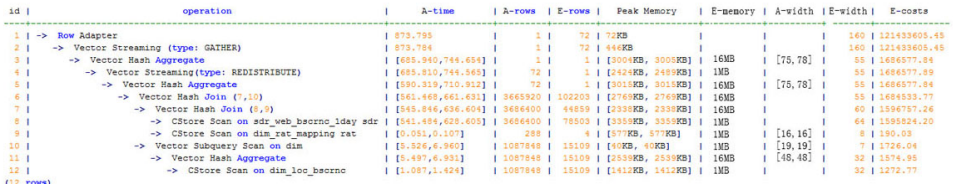
   建议在语句中手动添加JOIN列的非空判断，修改后的语句如下所示。

```
SELECT
 *
FROM
( ( SELECT
  STARTTIME STTIME,
  SUM(NVL(PAGE_DELAY_MSEL,0)) PAGE_DELAY_MSEL,
  SUM(NVL(PAGE_SUCCEED_TIMES,0)) PAGE_SUCCEED_TIMES,
  SUM(NVL(FST_PAGE_REQ_NUM,0)) FST_PAGE_REQ_NUM,
  SUM(NVL(PAGE_AVG_SIZE,0)) PAGE_AVG_SIZE,
  SUM(NVL(FST_PAGE_ACK_NUM,0)) FST_PAGE_ACK_NUM,
  SUM(NVL(DATATRANS_DW_DURATION,0)) DATATRANS_DW_DURATION,
  SUM(NVL(PAGE_SR_DELAY_MSEL,0)) PAGE_SR_DELAY_MSEL
 FROM
  PS.SDR_WEB_BSCRNC_1DAY SDR
  INNER JOIN (SELECT
      BSCRNC_ID,
      BSCRNC_NAME,
```

```
    ACCESS_TYPE,
    ACCESS_TYPE_ID
   FROM
    nethouse.DIM_LOC_BSCRNC
   GROUP BY
    BSCRNC_ID,
    BSCRNC_NAME,
    ACCESS_TYPE,
    ACCESS_TYPE_ID) DIM
 ON SDR.BSCRNC_ID = DIM.BSCRNC_ID
 AND DIM.ACCESS_TYPE_ID IN (0,1,2)
 INNER JOIN nethouse.DIM_RAT_MAPPING RAT
 ON (RAT.RAT = SDR.RAT)
WHERE
 ( (STARTTIME >= 1461340800
 AND STARTTIME < 1461427200) )
 AND RAT.ACCESS_TYPE_ID IN (0,1,2)
 and SDR.BSCRNC_ID is not null
GROUP BY
 STTIME ) ) A;
```

执行计划如图6-13所示。

图 6-13 增加 JOIN 列非空条件（二）



6.11.4 案例：使排序下推

现象描述

在做场景性能测试时，发现某场景大部分时间是CN端在做window agg，占到总执行时间95%以上，系统资源不能充分利用。研究发现该场景的特点是：将两列分别求sum作为一个子查询，外层对两列的和再求和后做trunc，然后排序。

表结构如下所示：

```
CREATE TABLE public.test(imsi int,L4_DW_THROUGHPUT int,L4_UL_THROUGHPUT int)
with (orientation = column) DISTRIBUTE BY hash(imsi);
```

查询语句如下所示：

```
SELECT COUNT(1) over() AS DATACNT,
IMSI AS IMSI_IMSI,
CAST(TRUNC(((SUM(L4_UL_THROUGHPUT) + SUM(L4_DW_THROUGHPUT))), 0) AS
DECIMAL(20)) AS TOTAL_VOLOME_KPIID
FROM public.test AS test
GROUP BY IMSI
order by TOTAL_VOLOME_KPIID DESC;
```

执行计划如下：（注：向量化执行算子不建议开发者使用）。

```
Row Adapter  (cost=10.70..10.70 rows=10 width=12)
  ->  Vector Sort  (cost=10.68..10.70 rows=10 width=12)
        Sort Key: ((trunc((((sum(l4_ul_throughput)) + (sum(l4_dw_throughput))))::numeric,
0))::numeric(20,0))
        ->  Vector WindowAgg  (cost=10.09..10.51 rows=10 width=12)
              ->  Vector Streaming (type: GATHER)  (cost=242.04..246.84 rows=240 width=12)
                    Node/s: All datanodes
                    ->  Vector Hash Aggregate  (cost=10.09..10.29 rows=10 width=12)
```

```
                  Group By Key: imsi
                  ->  CStore Scan on test  (cost=0.00..10.01 rows=10 width=12)
```

可以看到window agg和sort全部在CN端执行，耗时非常严重。

优化分析

尝试将语句改写为子查询。

```
SELECT COUNT(1) over() AS DATACNT, IMSI_IMSI, TOTAL_VOLOME_KPIID
FROM (SELECT IMSI AS IMSI_IMSI,
CAST(TRUNC(((SUM(L4_UL_THROUGHPUT) + SUM(L4_DW_THROUGHPUT))),
0) AS DECIMAL(20)) AS TOTAL_VOLOME_KPIID
FROM public.test AS test
GROUP BY IMSI
ORDER BY TOTAL_VOLOME_KPIID DESC);
```

将trunc两列的和作为一个子查询，然后在子查询的外面做window agg，这样排序就可以下推了，执行计划如下：

注：向量化执行算子不建议开发者使用。

```
Row Adapter  (cost=10.70..10.70 rows=10 width=24)
  ->  Vector WindowAgg  (cost=10.45..10.70 rows=10 width=24)
       ->  Vector Streaming (type: GATHER)  (cost=250.83..253.83 rows=240 width=24)
            Node/s: All datanodes
            ->  Vector Sort  (cost=10.45..10.48 rows=10 width=12)
                 Sort Key: ((trunc(((sum(test.l4_ul_throughput) + sum(test.l4_dw_throughput)))::numeric,
0))::numeric(20,0))
                 ->  Vector Hash Aggregate  (cost=10.09..10.29 rows=10 width=12)
                      Group By Key: test.imsi
                      ->  CStore Scan on test  (cost=0.00..10.01 rows=10 width=12)
```

经过SQL改写，性能由120s提升到7s，优化效果明显。

6.11.5 案例：设置 cost_param 对查询性能优化

现象描述 1

cost_param的bit0(set cost_param=1)值为1时，表示对于求由不等式（!=）条件连接的选择率时选择一种改良机制，此方法在自连接（两个相同的表之间连接）的估算中更加准确。下面查询的例子是cost_param的bit0为1时的优化场景。当前版本已弃用cost_param & 1不为0时的路径，默认选择已优化的估算公式。

注：选择率是两表join时，满足join条件的行数在join结果集中所占的比率。

表结构如下所示：

```
CREATE TABLE LINEITEM
(
L_ORDERKEY BIGINT NOT NULL
, L_PARTKEY BIGINT NOT NULL
, L_SUPPKEY BIGINT NOT NULL
, L_LINENUMBER BIGINT NOT NULL
, L_QUANTITY DECIMAL(15,2) NOT NULL
, L_EXTENDEDPRICE DECIMAL(15,2) NOT NULL
, L_DISCOUNT DECIMAL(15,2) NOT NULL
, L_TAX DECIMAL(15,2) NOT NULL
, L_RETURNFLAG CHAR(1) NOT NULL
, L_LINESTATUS CHAR(1) NOT NULL
, L_SHIPDATE DATE NOT NULL
, L_COMMITDATE DATE NOT NULL
, L_RECEIPTDATE DATE NOT NULL
, L_SHIPINSTRUCT CHAR(25) NOT NULL
, L_SHIPMODE CHAR(10) NOT NULL
```



```
, L_COMMENT VARCHAR(44) NOT NULL
)  distribute by hash(L_ORDERKEY);

CREATE TABLE ORDERS
(
O_ORDERKEY BIGINT NOT NULL
, O_CUSTKEY BIGINT NOT NULL
, O_ORDERSTATUS CHAR(1) NOT NULL
, O_TOTALPRICE DECIMAL(15,2) NOT NULL
, O_ORDERDATE DATE NOT NULL
, O_ORDERPRIORITY CHAR(15) NOT NULL
, O_CLERK CHAR(15) NOT NULL
, O_SHIPPRIORITY BIGINT NOT NULL
, O_COMMENT VARCHAR(79) NOT NULL
) distribute by hash(O_ORDERKEY);
```

查询语句如下所示：

```
explain verbose select
count(*) as numwait
from
lineitem l1,
orders
where
o_orderkey = l1.l_orderkey
and o_orderstatus = 'F'
and l1.l_receiptdate > l1.l_commitdate
and not exists (
select
*
from
lineitem l3
where
l3.l_orderkey = l1.l_orderkey
and l3.l_suppkey <> l1.l_suppkey
and l3.l_receiptdate > l3.l_commitdate
)
order by
numwait desc;
```

执行计划如下图所示：（verbose条件下，新增distinct列，受cost off/on控制，hashjoin行显示内外表的distinct估值，其他行为空）

```
 id |                        operation                        | E-rows | E-distinct | E-width | E-costs
----+---------------------------------------------------------+--------+------------+---------+---------
  1 | ->  Row Adapter                                         |      1 |            |       8 | 39.36
  2 |    ->  Vector Sort                                      |      1 |            |       8 | 39.36
  3 |       ->  Vector Aggregate                              |      1 |            |       8 | 39.34
  4 |          ->  Vector Streaming (type: GATHER)            |      2 |            |       8 | 39.34
  5 |             ->  Vector Aggregate                        |      2 |            |       8 | 39.25
  6 |                ->  Vector Hash Anti Join (7, 10)        |      2 | 4, 5       |       0 | 39.24
  7 |                   ->  Vector Hash Join (8,9)            |      2 | 200, 1     |      16 | 26.12
  8 |                      ->  CStore Scan on public.lineitem l1 |      7 |            |      16 | 13.05
  9 |                      ->  CStore Scan on public.orders   |      1 |            |       8 | 13.05
 10 |                   ->  CStore Scan on public.lineitem l3 |      7 |            |      16 | 13.05
(10 rows)
```

优化分析 1

以上查询为lineitem表自连接的Anti Join，当使用cost_param的bit0为0时，估算Anti Join的行数与实际行数相差很大，导致查询性能下降。可以通过设置cost_param的bit0为1时，使Anti Join的行数估算更准确，从而提高查询性能。优化后的执行计划如下：


```
 id |                    operation                    |  E-rows   | E-memory | E-width |   E-costs
----+-------------------------------------------------+-----------+----------+---------+--------------
  1 |   -> Row Adapter                                |         1 |          |       0 | 9104892.37 9
  2 |     -> Vector Sort                              |         1 |          |       0 | 9104892.37 9
  3 |       -> Vector Aggregate                       |         1 |          |       0 | 9104892.35 8
  4 |         -> Vector Streaming  (type: GATHER)     |        48 |          |       0 | 9104892.35 8
  5 |           -> Vector Aggregate                   |        48 | 1MB      |       0 | 9104890.82 5
  6 |             -> Vector Hash Join (7.12)          | 2526630903 | 929MB    |       0 | 8973295.45 4
  7 |               -> Vector Hash Anti Join (8. 10)  | 1999996587 | 3178MB   |       8 | 7198231.14
  8 |                 -> Vector Partition Iterator    | 1999996587 | 1MB      |      16 | 3000158.25
  9 |                   -> Partitioned CStore Scan on public.lineitem 11 | 1999996587 | 1MB | 16 | 3000158.25 1
 10 |                 -> Vector Partition Iterator    | 1999996587 | 1MB      |      16 | 3000158.25
 11 |                   -> Partitioned CStore Scan on public.lineitem 13 | 1999996587 | 1MB | 16 | 3000158.25
 12 |               -> Vector Partition Iterator      |  730839014 | 1MB      |       8 | 589611.00
 13 |                 -> Partitioned CStore Scan on public.orders | 730839014 | 1MB | 8 | 589611.00
(13 rows)
```

现象描述 2

当cost_param的bit1(set cost_param=2)为1时，表示求多个过滤条件（Filter）的选择率时，选择最小的作为总的选择率，而非两者乘积，此方法在过滤条件的列之间关联性较强时估算更加准确。下面查询的例子是cost_param的bit1为1时的优化场景。

表结构如下所示：

```
CREATE TABLE NATION
(
  N_NATIONKEY INT NOT NULL
, N_NAME      CHAR(25) NOT NULL
, N_REGIONKEY INT NOT NULL
, N_COMMENT   VARCHAR(152)
) distribute by replication;
CREATE TABLE SUPPLIER
(
  S_SUPPKEY   BIGINT NOT NULL
, S_NAME      CHAR(25) NOT NULL
, S_ADDRESS   VARCHAR(40) NOT NULL
, S_NATIONKEY INT NOT NULL
, S_PHONE     CHAR(15) NOT NULL
, S_ACCTBAL   DECIMAL(15,2) NOT NULL
, S_COMMENT   VARCHAR(101) NOT NULL
) distribute by hash(S_SUPPKEY);
CREATE TABLE PARTSUPP
(
  PS_PARTKEY    BIGINT NOT NULL
, PS_SUPPKEY    BIGINT NOT NULL
, PS_AVAILQTY   BIGINT NOT NULL
, PS_SUPPLYCOST DECIMAL(15,2)NOT NULL
, PS_COMMENT    VARCHAR(199) NOT NULL
)distribute by hash(PS_PARTKEY);
```

查询语句如下所示：

```
set cost_param=2;
explain verbose select
nation,
sum(amount) as sum_profit
from
(
select
n_name as nation,
l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
from
supplier,
lineitem,
partsupp,
nation
```



```
where
s_suppkey = l_suppkey
and ps_suppkey = l_suppkey
and ps_partkey = l_partkey
and s_nationkey = n_nationkey
) as profit
group by nation
order by nation;
```

当cost_param的bit1为0时，执行计划如下图所示：

```
 id |                           operation                           | E-rows | E-distinct | E-width | E-costs
----+---------------------------------------------------------------+--------+------------+---------+---------
  1 | ->  Sort                                                      |      1 |            |     208 | 61.52
  2 |    ->  HashAggregate                                          |      1 |            |     208 | 61.51
  3 |       ->  Streaming (type: GATHER)                            |      2 |            |     208 | 61.51
  4 |          ->  HashAggregate                                    |      2 |            |     208 | 61.36
  5 |             ->  Hash Join (6,7)                               |      2 | 20, 15     |     176 | 61.33
  6 |                ->  Seq Scan on public.nation                  |     40 |            |     108 | 20.20
  7 |                ->  Hash                                       |      2 |            |      76 | 41.04
  8 |                   ->  Hash Join (9,16)                        |      2 | 10, 13     |      76 | 41.04
  9 |                      ->  Streaming(type: REDISTRIBUTE)        |      2 |            |      88 | 27.73
 10 |                         ->  Hash Join (11,14)                 |      2 | 10, 13     |      88 | 27.62
 11 |                            ->  Streaming(type: REDISTRIBUTE)  |     20 |            |      70 | 14.19
 12 |                               ->  Row Adapter                 |     21 |            |      70 | 13.01
 13 |                                  ->  CStore Scan on public.lineitem |     20 |            |      70 | 13.01
 14 |                            ->  Hash                           |     21 |            |      34 | 13.13
 15 |                               ->  Seq Scan on public.partsupp |     20 |            |      34 | 13.13
 16 |                      ->  Hash                                 |     21 |            |      12 | 13.13
 17 |                         ->  Seq Scan on public.supplier       |     20 |            |      12 | 13.13
(17 rows)
```

优化分析 2

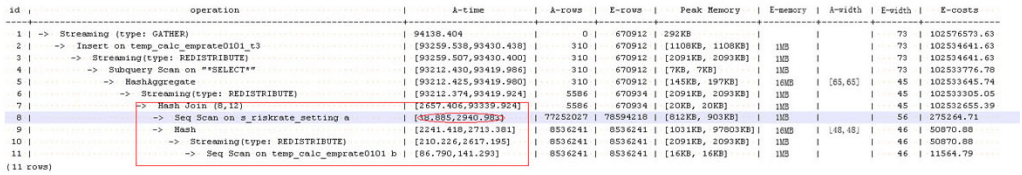
在以上查询中，supplier、lineitem、partsupp三表做hashjoin的条件为(lineitem.l_suppkey = supplier.s_suppkey) AND (lineitem.l_partkey = partsupp.ps_partkey)，此hashjoin条件中存在两个过滤条件，这前一个过滤条件中的lineitem.l_suppkey和后一个过滤条件中的lineitem.l_partkey同为lineitem表的两列，这两列存在强相关的关联关系。在这种情况，估算hashjoin条件的选择率时，如果使用cost_param的bit1为0时，实际是将AND的两个过滤条件分别计算的2个选择率的值相乘来得到hashjoin条件的选择率，导致行数估算不准确，查询性能较差。所以需要将cost_param的bit1为1时，选择最小的选择率作为总的选择率估算行数比较准确，查询性能较好，优化后的计划如下图所示：

```
 id |                           operation                           | E-rows | E-distinct | E-width | E-costs
----+---------------------------------------------------------------+--------+------------+---------+---------
  1 | ->  Sort                                                      |     10 |            |     208 | 64.42
  2 |    ->  HashAggregate                                          |     10 |            |     208 | 64.23
  3 |       ->  Streaming (type: GATHER)                            |     20 |            |     208 | 64.23
  4 |          ->  HashAggregate                                    |     20 |            |     208 | 62.71
  5 |             ->  Hash Join (6,7)                               |     20 | 20, 10     |     176 | 62.46
  6 |                ->  Seq Scan on public.nation                  |     40 |            |     108 | 20.20
  7 |                ->  Hash                                       |     20 |            |      76 | 41.97
  8 |                   ->  Hash Join (9,16)                        |     20 | 10, 13     |      76 | 41.97
  9 |                      ->  Streaming(type: REDISTRIBUTE)        |     20 |            |      82 | 28.54
 10 |                         ->  Hash Join (11,14)                 |     20 | 10, 13     |      82 | 27.63
 11 |                            ->  Streaming(type: REDISTRIBUTE)  |     20 |            |      70 | 14.19
 12 |                               ->  Row Adapter                 |     21 |            |      70 | 13.01
 13 |                                  ->  CStore Scan on public.lineitem |     20 |            |      70 | 13.01
 14 |                            ->  Hash                           |     21 |            |      12 | 13.13
 15 |                               ->  Seq Scan on public.supplier |     20 |            |      12 | 13.13
 16 |                      ->  Hash                                 |     21 |            |      34 | 13.13
 17 |                         ->  Seq Scan on public.partsupp       |     20 |            |      34 | 13.13
(17 rows)
```

6.11.6 案例：调整分布键

现象描述

某局点测试过程中EXPLAIN ANALYZE后有如下情况：



id	operation	A-time	A-rows	E-rows	Peak Memory	E-memory	A-width	E-width	E-costs
1	-> Streaming (type: GATHER)	94138.404	0	670912	292KB			73	102576573.63
2	-> Insert on temp_calc_emprate0101_t3	[93259.538,93430.438]	310	670912	[1108KB, 1108KB]	1MB		73	102534641.63
3	-> Streaming(type: REDISTRIBUTE)	[93259.507,93430.400]	310	670912	[2091KB, 2093KB]	1MB		73	102534641.63
4	-> Subquery Scan on "*SELECT*"	[93212.430,93419.986]	310	670912	[7KB, 7KB]	1MB		73	102533776.78
5	-> HashAggregate	[93212.425,93419.980]	310	670912	[145KB, 197KB]	16MB	[65,65]	45	102533645.74
6	-> Streaming(type: REDISTRIBUTE)	[93212.374,93419.924]	5586	670934	[2091KB, 2093KB]	1MB		45	102533305.05
7	-> Hash Join (8,12)	[2657.406,93339.924]	5586	670934	[20KB, 20KB]	1MB		45	102532655.39
8	-> Seq Scan on s_riskrate_setting a	[38.885,2940.983]	77252027	78594218	[812KB, 903KB]	1MB		56	275264.71
9	-> Hash	[2241.418,2713.381]	8536241	8536241	[1031KB, 97803KB]	16MB	[48,48]	46	50870.88
10	-> Streaming(type: REDISTRIBUTE)	[210.226,2617.195]	8536241	8536241	[2091KB, 2093KB]	1MB		46	50870.88
11	-> Seq Scan on temp_calc_emprate0101 b	[86.790,141.293]	8536241	8536241	[16KB, 16KB]	1MB		46	11564.79

(11 rows)

从执行信息上比较明确的可以看出HashJoin是整个计划的性能瓶颈点，并且从HashJoin的执行时间信息[2657.406,93339.924](数值的具体含义请参见**SQL执行计划详解**)，上可以看出HashJoin在不同的DN上存在严重的计算偏斜。

同时在Memory Information(如下图)中可以看出各个节点的内存资源消耗也存在极为严重的偏斜。

```
                Memory Information (identified by plan id)
--------------------------------------------------------------------------------
Coordinator:
   Query Peak Memory: 4MB
Datanode:
   Max Query Peak Memory: 118MB
   Min Query Peak Memory: 24MB
  12 --Hash
         Max Buckets: 131072   Max Batches: 1   Max Memory Usage: 91857kB
         Min Buckets: 131072   Min Batches: 1   Min Memory Usage: 0kB
(8 rows)
```

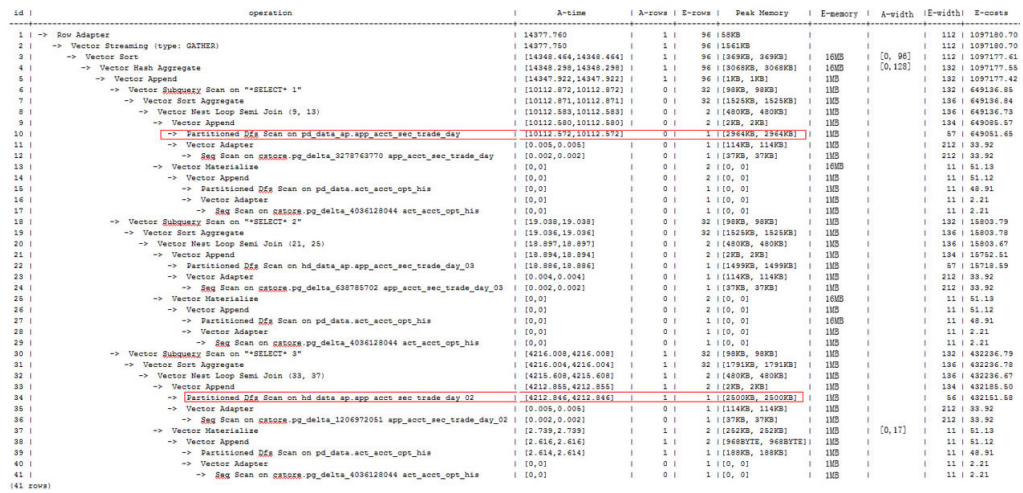
优化分析

上述两个特征表明了此SQL语句存在极为严重的计算倾斜。进一步向HashJoin算子的下层分析发现Seq Scan on s_riskrate_setting也存在极为严重的计算倾斜[38.885,2940.983]。根据Scan的含义推测此计划性能问题的根源在于表s_riskrate_setting数据的分布倾斜。实际分析之后确实发现表s_riskrate_setting存在严重的数据倾斜。整改之后性能从94s提升为50s。

6.11.7 案例：调整局部聚簇键

现象描述

某局点EXPLAIN PERFORMANCE信息如下。分析发现如图红框标识的两个性能瓶颈点均为表Scan动作。



id	operation	A-time	A-rows	E-rows	Peak Memory	E-memory	A-width	E-width	E-costs
1	-> Row Adapter	14377.760	1	96	58KB			112	1097180.70
2	-> Vector Streaming (type: GATHER)	14377.750	1	96	1561KB			112	1097180.70
3	-> Vector Sort	[14348.464,14348.464]	1	96	[369KB, 369KB]	16MB	[0, 96]	112	1097177.61
4	-> Vector Hash Aggregate	[14348.298,14348.298]	1	96	[3068KB, 3068KB]	16MB	[0,128]	132	1097177.55
5	-> Vector Append	[14347.922,14347.922]	1	96	[1KB, 1KB]	1MB		132	1097177.42
6	-> Vector Subquery Scan on "*SELECT* 1"	[10112.872,10112.872]	0	32	[98KB, 98KB]	1MB		192	649136.85
7	-> Vector Sort Aggregate	[10112.871,10112.871]	0	32	[1525KB, 1525KB]	1MB		136	649136.84
8	-> Vector Nest Loop Semi Join (9, 13)	[10112.583,10112.583]	0	2	[480KB, 480KB]	1MB		136	649136.73
9	-> Vector Append	[10112.580,10112.580]	0	2	[2KB, 2KB]	1MB		134	649085.57
10	-> Partitioned Dfs Scan on pd_data_ap.app_acct_sec_trade_day	[10112.572,10112.572]	0	1	[2564KB, 2564KB]	1MB		57	649061.65
11	-> Vector Adapter	[0.005,0.005]	0	1	[114KB, 114KB]	1MB		212	33.92
12	-> Seq Scan on cstore.pg_delta_3278763770 app_acct_sec_trade_day	[0.002,0.002]	0	1	[37KB, 37KB]	1MB		212	33.92
13	-> Vector Materialize	[0,0]	0	2	[0, 0]	16MB		11	51.13
14	-> Vector Append	[0,0]	0	2	[0, 0]	1MB		11	51.12
15	-> Partitioned Dfs Scan on pd_data.act_acct_opt_his	[0,0]	0	1	[0, 0]	1MB		11	48.91
16	-> Vector Adapter	[0,0]	0	1	[0, 0]	1MB		11	2.21
17	-> Seq Scan on cstore.pg_delta_4036128044 act_acct_opt_his	[0,0]	0	1	[0, 0]	1MB		11	2.21
18	-> Vector Subquery Scan on "*SELECT* 2"	[19.038,19.038]	0	32	[98KB, 98KB]	1MB		132	15803.79
19	-> Vector Sort Aggregate	[19.036,19.036]	0	32	[1525KB, 1525KB]	1MB		136	15803.78
20	-> Vector Nest Loop Semi Join (21, 25)	[18.897,18.897]	0	2	[480KB, 480KB]	1MB		136	15803.67
21	-> Vector Append	[18.894,18.894]	0	2	[2KB, 2KB]	1MB		134	15752.51
22	-> Partitioned Dfs Scan on hd_data_ap.app_acct_sec_trade_day_03	[18.886,18.886]	0	1	[1499KB, 1499KB]	1MB		57	15718.59
23	-> Vector Adapter	[0.004,0.004]	0	1	[114KB, 114KB]	1MB		212	33.92
24	-> Seq Scan on cstore.pg_delta_638785702 app_acct_sec_trade_day_03	[0.002,0.002]	0	1	[37KB, 37KB]	1MB		212	33.92
25	-> Vector Materialize	[0,0]	0	2	[0, 0]	16MB		11	51.13
26	-> Vector Append	[0,0]	0	2	[0, 0]	1MB		11	51.12
27	-> Partitioned Dfs Scan on pd_data.act_acct_opt_his	[0,0]	0	1	[0, 0]	16MB		11	48.91
28	-> Vector Adapter	[0,0]	0	1	[0, 0]	1MB		11	2.21
29	-> Seq Scan on cstore.pg_delta_4036128044 act_acct_opt_his	[0,0]	0	1	[0, 0]	1MB		11	2.21
30	-> Vector Subquery Scan on "*SELECT* 3"	[4216.008,4216.008]	1	32	[98KB, 98KB]	1MB		132	432236.79
31	-> Vector Sort Aggregate	[4216.004,4216.004]	1	32	[1791KB, 1791KB]	1MB		136	432236.78
32	-> Vector Nest Loop Semi Join (33, 37)	[4215.608,4215.608]	1	2	[480KB, 480KB]	1MB		136	432236.67
33	-> Vector Append	[4212.855,4212.855]	1	2	[2KB, 2KB]	1MB		134	432185.50
34	-> Partitioned Dfs Scan on hd_data_ap.app_acct_sec_trade_day_02	[4212.846,4212.846]	1	1	[2500KB, 2500KB]	1MB		56	432161.58
35	-> Vector Adapter	[0.005,0.005]	0	1	[114KB, 114KB]	1MB		212	33.92
36	-> Seq Scan on cstore.pg_delta_1206972051 app_acct_sec_trade_day_02	[0.002,0.002]	0	1	[37KB, 37KB]	1MB		212	33.92
37	-> Vector Materialize	[2.739,2.739]	1	2	[252KB, 252KB]	1MB	[0,17]	11	51.13
38	-> Vector Append	[2.616,2.616]	1	2	[968BYTE, 968BYTE]	1MB		11	51.12
39	-> Partitioned Dfs Scan on pd_data.act_acct_opt_his	[2.614,2.614]	1	1	[188KB, 188KB]	1MB		11	48.91
40	-> Vector Adapter	[0,0]	0	1	[0, 0]	1MB		11	2.21
41	-> Seq Scan on cstore.pg_delta_4036128044 act_acct_opt_his	[0,0]	0	1	[0, 0]	1MB		11	2.21

(41 rows)

优化分析

进一步分析表Scan的filter条件发现两个表存在acct_id = 'A012709548'::bpchar这样的filter条件。

```
10 --Partitioned Dfs Scan on pd_data_ap.app_acct_sec_trade_day
     Filter: ((pd_data_ap.app_acct_sec_trade_day.sec_code ~~ '58%'::text) AND (((CASE WHEN (pd_data_ap.app_acct_sec_trade_day.sec_code ~~ ANY ('{201%,202%,204%}'::text[])) THEN ((pd_data_ap.app_acct_sec_trade_day.buy_vol * pd_
     Pushdown Predicate Filter: (pd_data_ap.app_acct_sec_trade_day.acct_id = 'A012709548'::bpchar)
12 --Seq Scan on cstore.pg_delta_3278763770 app_acct_sec_trade_day
     Filter: ((cstore.app_acct_sec_trade_day.sec_code ~~ '58%'::text) AND (cstore.app_acct_sec_trade_day.acct_id = 'A012709548'::bpchar) AND (((CASE WHEN (cstore.app_acct_sec_trade_day.sec_code ~~ ANY ('{201%,202%,204%}'::text
15 --Partitioned Dfs Scan on pd_data.act_acct_opt_his
     Pushdown Predicate Filter: (pd_data.act_acct_opt_his.acct_id = 'A012709548'::bpchar)
17 --Seq Scan on cstore.pg_delta_4036128044 act_acct_opt_his
     Filter: (cstore.act_acct_opt_his.acct_id = 'A012709548'::bpchar)
22 --Partitioned Dfs Scan on hd_data_ap.app_acct_sec_trade_day_03
     Filter: (hd_data_ap.app_acct_sec_trade_day_03.sec_code ~~ '58%'::text) AND (((CASE WHEN (hd_data_ap.app_acct_sec_trade_day_03.sec_code ~~ ANY ('{201%,202%,204%}'::text[])) THEN ((hd_data_ap.app_acct_sec_trade_day_03.buy_
     Pushdown Predicate Filter: (hd_data_ap.app_acct_sec_trade_day_03.acct_id = 'A012709548'::bpchar)
24 --Seq Scan on cstore.pg_delta_638785702 app_acct_sec_trade_day_03
     Filter: ((cstore.app_acct_sec_trade_day_03.sec_code ~~ '58%'::text) AND (cstore.app_acct_sec_trade_day_03.acct_id = 'A012709548'::bpchar) AND (((CASE WHEN (cstore.app_acct_sec_trade_day_03.sec_code ~~ ANY ('{201%,202%,20
27 --Partitioned Dfs Scan on pd_data.act_acct_opt_his
     Pushdown Predicate Filter: (pd_data.act_acct_opt_his.acct_id = 'A012709548'::bpchar)
29 --Seq Scan on cstore.pg_delta_4036128044 act_acct_opt_his
     Filter: (cstore.act_acct_opt_his.acct_id = 'A012709548'::bpchar)
34 --Partitioned Dfs Scan on hd_data_ap.app_acct_sec_trade_day_02
     Filter: ((hd_data_ap.app_acct_sec_trade_day_02.sec_code ~~ '58%'::text) AND (((CASE WHEN (hd_data_ap.app_acct_sec_trade_day_02.sec_code ~~ ANY ('{201%,202%,204%}'::text[])) THEN ((hd_data_ap.app_acct_sec_trade_day_02.buy_
     Pushdown Predicate Filter: (hd_data_ap.app_acct_sec_trade_day_02.acct_id = 'A012709548'::bpchar)
36 --Seq Scan on cstore.pg_delta_1206972051 app_acct_sec_trade_day_02
     Filter: ((cstore.app_acct_sec_trade_day_02.sec_code ~~ '58%'::text) AND (cstore.app_acct_sec_trade_day_02.acct_id = 'A012709548'::bpchar) AND (((CASE WHEN (cstore.app_acct_sec_trade_day_02.sec_code ~~ ANY ('{201%,202%,20
39 --Partitioned Dfs Scan on pd_data.act_acct_opt_his
     Pushdown Predicate Filter: (pd_data.act_acct_opt_his.acct_id = 'A012709548'::bpchar)
41 --Seq Scan on cstore.pg_delta_4036128044 act_acct_opt_his
     Filter: (cstore.act_acct_opt_his.acct_id = 'A012709548'::bpchar)
```

试着给两个表的acct_id列增加局部聚簇键，然后对两张表执行VACUUM FULL，使局部聚簇生效。调整后性能得到提升。

6.11.8 案例：改建分区表

现象描述

如下简单SQL语句查询， 性能瓶颈点在dwcjk的Scan上。

```
openGauss=#   explain performance select zqdh, count(1) from dwcjk where cjrq = '2015-05-02 00:00:00' group by zqdh;
 id |                  operation                  |       A-time        | A-rows  | E-rows  |    Peak Memory    | E-memory |  A-width  | E-width | E-costs
----+---------------------------------------------+---------------------+---------+---------+-------------------+----------+-----------+---------+-----------
  1 | ->  Row Adapter                             | 1599.794            |      58 |      12 | 19KB              |          |           |       7 | 771106.83
  2 |    ->  Vector Streaming (type: GATHER)      | 1599.781            |      58 |      12 | 210KB             |          |           |       7 | 771106.83
  3 |       ->  Vector Hash Aggregate             | [1445.092,1446.332] |      58 |       2 | [2315KB, 2315KB]  |   16MB   |  [16,16]  |       7 | 128517.80
  4 |          ->  Vector Streaming(type: REDISTRIBUTE) | [1444.996,1446.259] |     340 |      12 | [247KB, 247KB]    |   1MB    |           |       7 | 128518.09
  5 |             ->  Vector Hash Aggregate       | [573.150,1261.354]  |     340 |      12 | [2297KB, 2297KB]  |   16MB   |  [16,16]  |       7 | 128517.80
  6 |                ->  CStore Scan on public.dwcjk | [330.178,1021.695]  | 10000000 | 1623137 | [786KB, 786KB]    |   1MB    |           |       7 | 120402.00
(6 rows)
```

优化分析

从业务层确认表数据(在cjrq字段上)有明显的日期特征，符合分区表的特征。重新规划dwcjk表的表定义：字段cjrq为分区键、天为间隔单位定义分区表dwcjk_part。修改后结果如下，性能提升近1倍。

```
openGauss=# explain performance select zqdh, count(1) from dwcjk_part where cjrq = '2015-05-02 00:00:00' group by zqdh;
 id |                        operation                        |       A-time        | A-rows  | E-rows  |    Peak Memory    | E-memory | A-width | E-width | E-costs
----+---------------------------------------------------------+---------------------+---------+---------+-------------------+----------+---------+---------+-----------
  1 | ->  Row Adapter                                         | 977.457             |      58 |      14 | 19KB              |          |         |       7 | 773142.84
  2 |    ->  Vector Streaming (type: GATHER)                  | 977.437             |      58 |      14 | 210KB             |          |         |       7 | 773142.84
  3 |       ->  Vector Hash Aggregate                         | [651.238,734.931]   |      58 |       2 | [2316KB, 2316KB]  |   16MB   | [16,16] |       7 | 128857.14
  4 |          ->  Vector Streaming(type: REDISTRIBUTE)       | [651.137,734.834]   |     340 |      14 | [247KB, 247KB]    |   1MB    |         |       7 | 128857.47
  5 |             ->  Vector Hash Aggregate                   | [402.145,515.752]   |     340 |      14 | [2297KB, 2297KB]  |   16MB   | [16,16] |       7 | 128857.14
  6 |                ->  Vector Partition Iterator            | [162.630,275.990]   | 10000000 | 1691000 | [312BYTE, 312BYTE] |   1MB    |         |       7 | 120402.00
  7 |                   ->  Partitioned CStore Scan on public.dwcjk_part | [161.746,275.207] | 10000000 | 1691000 | [795KB, 795KB]    |   1MB    |         |       7 | 120402.00
(7 rows)
```

6.11.9 案例：调整 GUC 参数 best_agg_plan

现象描述

t1的表定义为：

```
create table t1(a int, b int, c int) distribute by hash(a);
```

假设agg下层算子所输出结果集的分布列为setA，agg操作的group by列为setB，则在Stream框架下，Agg操作可以分为两个场景。

1. setA是setB的一个子集。

   对于这种场景，直接对下层结果集进行汇聚的结果就是正确的汇聚结果，上层算子直接使用即可。如下图所示：

```
openGauss=# explain select a, count(1) from t1 group by a;
 id |          operation           | E-rows | E-width | E-costs
----+------------------------------+--------+---------+---------
  1 | ->  Streaming (type: GATHER) |     30 |       4 | 15.56
  2 |    ->  HashAggregate         |     30 |       4 | 14.31
  3 |       ->  Seq Scan on t1     |     30 |       4 | 14.14
(3 rows)
```

2. setA不是setB的一个子集。

   对于这种场景，Stream执行框架分为如下三种计划形态：

   hashagg+gather(redistribute)+hashagg

   redistribute+hashagg(+gather)

   hashagg+redistribute+hashagg(+gather)

   GaussDB提供了guc参数best_agg_plan来干预执行计划，强制其生成上述对应的执行计划，此参数取值范围为0，1，2，3

   - 取值为1时，强制生成第一种计划。
   - 取值为2时，如果group by列可以重分布，强制生成第二种计划，否则生成第一种计划。
   - 取值为3时，如果group by列可以重分布，强制生成第三种计划，否则生成第一种计划。
   - 取值为0时，优化器会根据以上三种计划的估算代价选择最优的一种计划生成。

   具体影响请看下述图片

```
openGauss=# set best_agg_plan to 1;
SET
openGauss=# explain select b,count(1) from t1 group by b;
 id |           operation            | E-rows | E-width | E-costs
----+--------------------------------+--------+---------+---------
  1 | ->  HashAggregate              |      8 |       4 | 15.83
  2 |    ->  Streaming (type: GATHER) |     25 |       4 | 15.83
  3 |       ->  HashAggregate        |     25 |       4 | 14.33
  4 |         ->  Seq Scan on t1     |     30 |       4 | 14.14
(4 rows)
openGauss=# set best_agg_plan to 2;
SET
openGauss=# explain select b,count(1) from t1 group by b;
 id |               operation                | E-rows | E-width | E-costs
----+----------------------------------------+--------+---------+---------
  1 | ->  Streaming (type: GATHER)           |     30 |       4 | 15.85
  2 |    ->  HashAggregate                   |     30 |       4 | 14.60
  3 |       ->  Streaming(type: REDISTRIBUTE) |     30 |       4 | 14.45
  4 |         ->  Seq Scan on t1             |     30 |       4 | 14.14
(4 rows)
openGauss=# set best_agg_plan to 3;
SET
openGauss=# explain select b,count(1) from t1 group by b;
 id |               operation                | E-rows | E-width | E-costs
----+----------------------------------------+--------+---------+---------
  1 | ->  Streaming (type: GATHER)           |     30 |       4 | 15.84
  2 |    ->  HashAggregate                   |     30 |       4 | 14.59
  3 |       ->  Streaming(type: REDISTRIBUTE) |     25 |       4 | 14.59
  4 |         ->  HashAggregate              |     25 |       4 | 14.33
  5 |           ->  Seq Scan on t1           |     30 |       4 | 14.14
(5 rows)
```

优化说明

通常优化器总会选择最优的执行计划，但是众所周知代价估算，尤其是中间结果集的代价估算一般会有比较大的偏差，这种比较大的偏差就可能会导致agg的计算方式出现比较大的偏差，这时候就需要通过best_agg_plan进行agg计算模型的干预。

一般来说，当agg汇聚的收敛度很小时，即结果集的个数在agg之后并没有明显变少时（经验上以5倍为临界点），选择redistribute+hashagg执行方式，否则选择hashagg+redistribute+hashagg执行方式。

6.11.10 案例：改写 SQL 消除子查询

现象描述

```
select
   1,
   (select count(*) from customer_address_001 a4 where a4.ca_address_sk = a.ca_address_sk) as GZCS
from customer_address_001 a;
```

此SQL性能较差，查看发现执行计划中存在SubPlan，具体如下：

```
openGauss=# explain select 1,(select count(*)
openGauss(#           from customer_address_001 a4
openGauss(#          where a4.ca_address_sk = a.ca_address_sk
openGauss(#         ) as GZCS from customer_address_001 a;
 id |                          operation                          | E-rows | E-width | E-costs
----+-------------------------------------------------------------+--------+---------+---------
  1 | ->  Streaming (type: GATHER)                                |    320 |       4 | 4529.27
  2 |    ->  Seq Scan on customer_address_001 a                   |    320 |       4 | 4496.27
  3 |       ->  Aggregate  [2, SubPlan 1]                         |     32 |       4 | 139.50
  4 |          ->  Result                                         |  10240 |       4 | 138.69
  5 |             ->  Materialize                                 |  10240 |       4 | 138.69
  6 |                ->  Streaming(type: BROADCAST)               |  10240 |       4 | 137.09
  7 |                   ->  Seq Scan on customer_address_001 a4   |    320 |       4 | 32.32
(7 rows)
```

优化说明

此优化的核心就是消除子查询。分析业务场景发现*a.ca_address_sk*不为null，那么从SQL语义出发，可以等价改写SQL为：

```
select
count(*)
from customer_address_001 a4, customer_address_001 a
where a4.ca_address_sk = a.ca_address_sk
group by  a.ca_address_sk;
```

说明

为了保证改写的等效性，在*customer_address_001. ca_address_sk*加了*not null*约束。

6.11.11 案例：改写 SQL 排除剪枝干扰

现象描述

某局点测试中：ddw_f10_op_cust_asset_mon为分区表，分区键为year_mth，此字段是由年月两个值拼接而成的字符串。

测试SQL如下：

```
select
   count(1)
from t_ddw_f10_op_cust_asset_mon b1
where b1.year_mth between to_char(add_months(to_date(''20170222'','yyyymmdd'), -11),'yyyymm') and
substr(''20170222'',1 ,6 );
```

测试结果显示此SQL的表Scan耗时长达135s。初步猜测可能是性能瓶颈点。

说明

add_months为本地适配函数:

```
CREATE OR REPLACE FUNCTION ADD_MONTHS(date, integer) RETURNS date
   AS $$
   SELECT
   CASE
   WHEN (EXTRACT(day FROM $1) = EXTRACT(day FROM (date_trunc('month', $1) + INTERVAL '1
month - 1 day'))) THEN
       date_trunc('month', $1) + CAST($2 + 1 || ' month - 1 day' as interval)
   ELSE
       $1 + CAST($2 || ' month' as interval)
   END
   $$
   LANGUAGE SQL
   IMMUTABLE;
```

优化说明

分析语句的执行计划，发现执行计划中显示的基表filter如下：

```
Filter: (((year_mth)::text <= '201702'::text) AND ((year_mth)::text >=
to_char(add_months(to_date('20170222'::text, 'YYYYMMDD'::text), (-11)), 'YYYYMM'::text)))
```

Filter条件中存在表达式to_char(add_months(to_date(''20170222'','yyyymmdd'), -11),'yyyymm')，这种**非常量的表达式是不能用来剪枝的**，因而会导致查询语句扫描分区表所有数据。

查询pg_proc发现此处的to_date和to_char均为stable类型的函数，根据数据库对函数行为的约定，此类函数不能在预处理阶段转化为Const值，这也是不能导致分区剪枝的根本原因。

根据以上分析，优化表达式使其可以进行分区剪枝是性能优化的关键。根据语意将原SQL等价改写为：

```
select
    count(1)
from t_ddw_f10_op_cust_asset_mon b1
where b1.year_mth between(substr(ADD_MONTHS('20170222'::date, -11), 1, 4)||
substr(ADD_MONTHS('20170222'::date, -11), 6, 2)) and substr(''20170222'',1 ,6 );
```

改写之后，SQL执行时间从135s提升至18s。

6.11.12 案例：改写 SQL 消除 in-clause

现象描述

in-clause/any-clause是常见的SQL语句约束条件，有时in或any后面的clause都是常量，类似于：

```
select
count(1)
from calc_empfyc_c1_result_tmp_t1
where ls_pid_cusr1 in (‘20120405’, ‘20130405’);
```

或者

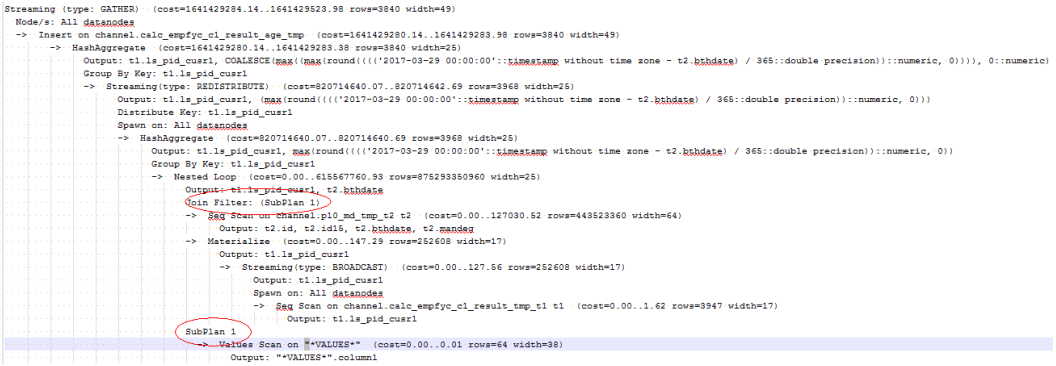
```
select
count(1)
from calc_empfyc_c1_result_tmp_t1
where ls_pid_cusr1 in any(‘20120405’, ‘20130405’);
```

但是也有一些如下的特殊用法：


```
SELECT
ls_pid_cusr1,COALESCE(max(round((current_date-bthdate)/365)),0)
FROM calc_empfyc_c1_result_tmp_t1 t1,p10_md_tmp_t2 t2
WHERE t1.ls_pid_cusr1 = any(values(id),(id15))
GROUP BY ls_pid_cusr1;
```

其中，id、id15为p10_md_tmp_t2中的两列，“t1.ls_pid_cusr1 = any(values(id),(id15))”等价于“t1.ls_pid_cusr1 = id or t1.ls_pid_cusr1 = id15”。

因此join-condition实质上是一个不等式，这种不等值的join操作必须走nestloop，对应执行计划如下：

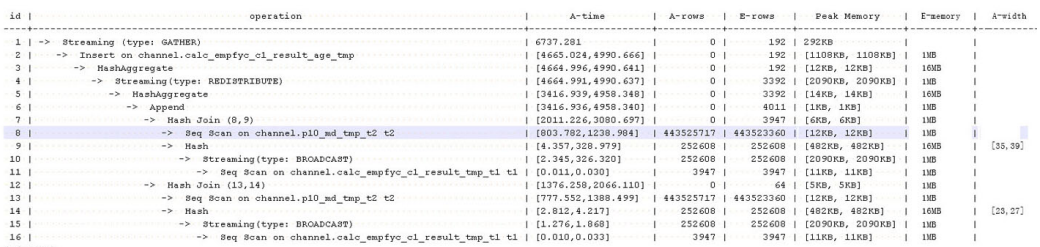


优化说明

测试发现由于两表结果集过大，导致nestloop耗时过长，超过一小时未返回结果，因此性能优化的关键是消除nestloop，让join走更高效的hashjoin。从语义等价的角度消除any-clause，SQL改写如下：

```
select
ls_pid_cusr1,COALESCE(max(round(ym/365)),0)
from
(
	(
		SELECT
			ls_pid_cusr1,(current_date-bthdate) as ym
		FROM calc_empfyc_c1_result_tmp_t1 t1,p10_md_tmp_t2 t2
		WHERE t1.ls_pid_cusr1 = t2.id and t1.ls_pid_cusr1 != t2.id15
	)
	union all
	(
		SELECT
			ls_pid_cusr1,(current_date-bthdate) as ym
		FROM calc_empfyc_c1_result_tmp_t1 t1,p10_md_tmp_t2 t2
		WHERE t1.ls_pid_cusr1 = id15
	)
)
GROUP BY ls_pid_cusr1;
```

优化后的SQL查询由两个等值join的子查询构成，而每个子查询都可以走更适合此场景的hashjoin。优化后的执行计划如下



优化后，从超过1个小时未返回结果优化到7s返回结果。

6.11.13 案例：调整查询重写 GUC 参数 rewrite_rule

rewrite_rule包含了多个查询重写规则：magicset、partialpush、uniquecheck、disablerep、intargetlist、predpush。下面简要说明其中重要的几个规则的使用场景：

案例环境准备

为了便于规则的使用场景演示，需准备建表语句如下：

```
--清理环境
DROP SCHEMA IF EXISTS rewrite_rule_guc_test CASCADE;
CREATE SCHEMA rewrite_rule_guc_test;
SET current_schema=rewrite_rule_guc_test;
--创建测试表
CREATE TABLE t(c1 INT, c2 INT, c3 INT, c4 INT);
CREATE TABLE t1(c1 INT, c2 INT, c3 INT, c4 INT);
CREATE TABLE t2(c1 INT, c2 INT, c3 INT, c4 INT);
```

部分下推参数 partialpush 的使用

查询下推到DN分布式执行，可以大大加速查询。如果查询语句中有一个不能下推的因素，整个语句就不能下推，无法生成Stream计划在DN分布式执行，性能通常较差。

举例如下查询：

```
yshen=# set rewrite_rule='none';
SET
yshen=# explain (verbose on, costs off)  select two_sum(tt.c1, tt.c2) from (select t1.c1,t2.c2 from t1,t2
where t1.c1=t2.c2) tt(c1,c2);
                           QUERY PLAN
--------------------------------------------------------------------------------
 Hash Join
   Output: two_sum(t1.c1, t2.c2)
   Hash Cond: (t1.c1 = t2.c2)
   ->  Data Node Scan on t1 "_REMOTE_TABLE_QUERY_"
         Output: t1.c1
         Node/s: All datanodes
         Remote query: SELECT c1 FROM ONLY public.t1 WHERE true
   ->  Hash
         Output: t2.c2
         ->  Data Node Scan on t2 "_REMOTE_TABLE_QUERY_"
               Output: t2.c2
               Node/s: All datanodes
               Remote query: SELECT c2 FROM ONLY public.t2 WHERE true
(13 rows)
```

其中two_sum()函数无法下推，导致走到RemoteQuery的计划：

1. 首先下发select c1 from t1 where true语句到DN读取全部t1表的数据。
2. 然后下发select c2 from t2 where true语句到DN读取全部t2表的数据。
3. 获取需要的数据之后，在CN上做HASH JOIN。
4. 最后结果参与two_sum运算并返回最终结果。

该计划很慢，原因是网络传输了大量数据，然后在CN上执行HASH JOIN，不能充分利用集群资源。

通过增加partialpush查询重写参数，可以把1,2,3下推到DN分布式执行，极大提升语句的性能：

```
yshen=# set rewrite_rule='partialpush';
SET
```



```
yshen=# explain (verbose on, costs off) select two_sum(tt.c1, tt.c2) from (select t1.c1,t2.c2 from t1,t2 where t1.c1=t2.c2) tt(c1,c2);
                    QUERY PLAN
-----------------------------------------------------------
 Subquery Scan on tt
   Output: two_sum(tt.c1, tt.c2)
   ->  Streaming (type: GATHER)  --Gather以下计划在DN分布式执行
         Output: t1.c1, t2.c2
         Node/s: All datanodes
         ->  Nested Loop
               Output: t1.c1, t2.c2
               Join Filter: (t1.c1 = t2.c2)
               ->  Seq Scan on public.t1
                     Output: t1.c1, t1.c2, t1.c3
                     Distribute Key: t1.c1
               ->  Materialize
                     Output: t2.c2
                     ->  Streaming(type: REDISTRIBUTE)
                           Output: t2.c2
                           Distribute Key: t2.c2
                           Spawn on: All datanodes
                           Consumer Nodes: All datanodes
                           ->  Seq Scan on public.t2
                                 Output: t2.c2
                                 Distribute Key: t2.c1
(21 rows)
```

目标列子查询提升参数 intargetlist

通过将目标列中子查询提升，转为JOIN，往往可以极大提升查询性能。举例如下查询：

```
yshen=# set rewrite_rule='none';
SET
yshen=# explain (verbose on, costs off) select c1,(select avg(c2) from t2 where t2.c2=t1.c2) from t1 where t1.c1<100 order by t1.c2;
                          QUERY PLAN
--------------------------------------------------------------------------
 Streaming (type: GATHER)
   Output: t1.c1, ((SubPlan 1)), t1.c2
   Merge Sort Key: t1.c2
   Node/s: All datanodes
   ->  Sort
         Output: t1.c1, ((SubPlan 1)), t1.c2
         Sort Key: t1.c2
         ->  Seq Scan on public.t1
               Output: t1.c1, (SubPlan 1), t1.c2
               Distribute Key: t1.c1
               Filter: (t1.c1 < 100)
               SubPlan 1
                 ->  Aggregate
                       Output: avg(t2.c2)
                       ->  Result
                             Output: t2.c2
                             Filter: (t2.c2 = t1.c2)
                             ->  Materialize
                                   Output: t2.c2
                                   ->  Streaming(type: BROADCAST)
                                         Output: t2.c2
                                         Spawn on: All datanodes
                                         Consumer Nodes: All datanodes
                                         ->  Seq Scan on public.t2
                                               Output: t2.c2
                                               Distribute Key: t2.c1
(26 rows)
```


由于目标列中的相关子查询(select avg(c2) from t2 where t2.c2=t1.c2)无法提升的缘故，导致每扫描t1的一行数据，就会触发子查询的一次执行，效率低下。如果打开intargetlist参数会把子查询提升转为JOIN，来提升查询的性能：

```
yshen=# set rewrite_rule='intargetlist';
SET
yshen=# explain (verbose on, costs off) select c1,(select avg(c2) from t2 where t2.c2=t1.c2) from t1 where
t1.c1<100 order by t1.c2;
                        QUERY PLAN
------------------------------------------------------------------
 Streaming (type: GATHER)
   Output: t1.c1, (avg(t2.c2)), t1.c2
   Merge Sort Key: t1.c2
   Node/s: All datanodes
   ->  Sort
         Output: t1.c1, (avg(t2.c2)), t1.c2
         Sort Key: t1.c2
         ->  Hash Right Join
               Output: t1.c1, (avg(t2.c2)), t1.c2
               Hash Cond: (t2.c2 = t1.c2)
               ->  Streaming(type: BROADCAST)
                     Output: (avg(t2.c2)), t2.c2
                     Spawn on: All datanodes
                     Consumer Nodes: All datanodes
                     ->  HashAggregate
                           Output: avg(t2.c2), t2.c2
                           Group By Key: t2.c2
                           ->  Streaming(type: REDISTRIBUTE)
                                 Output: t2.c2
                                 Distribute Key: t2.c2
                                 Spawn on: All datanodes
                                 Consumer Nodes: All datanodes
                                 ->  Seq Scan on public.t2
                                       Output: t2.c2
                                       Distribute Key: t2.c1
               ->  Hash
                     Output: t1.c1, t1.c2
                     ->  Seq Scan on public.t1
                           Output: t1.c1, t1.c2
                           Distribute Key: t1.c1
                           Filter: (t1.c1 < 100)
(31 rows)
```

提升无 agg 的子查询 uniquecheck

子链接提升需要保证对于每个条件只有一行输出，对于有agg的子查询可以自动提升，对于无agg的子查询如：

select t1.c1 from t1 where t1.c1 = (select t2.c1 from t2 where t1.c1=t2.c2) ;

重写为：

select t1.c1 from t1 join (select t2.c1 from t2 where t2.c1 is not null group by t2.c1(unique check)) tt(c1) on tt.c1=t1.c1;

为了保证语义等价，子查询tt必须保证对于每个group by t2.c1只能有一行输出。打开uniquecheck查询重写参数保证可以提升并且等价，如果在运行时输出了多于一行的数据，就会报错。

```
yshen=# set rewrite_rule='uniquecheck';
SET
yshen=# explain verbose select t1.c1 from t1 where t1.c1 = (select t2.c1 from t2 where t1.c1=t2.c1) ;
                           QUERY PLAN
--------------------------------------------------------------------------------
 Streaming (type: GATHER)
   Output: t1.c1
```



```
  Node/s: All datanodes
  ->  Nested Loop
        Output: t1.c1
        Join Filter: (t1.c1 = subquery."?column?")
        ->  Seq Scan on public.t1
              Output: t1.c1, t1.c2, t1.c3
              Distribute Key: t1.c1
        ->  Materialize
              Output: subquery."?column?", subquery.c1
              ->  Subquery Scan on subquery
                    Output: subquery."?column?", subquery.c1
                    ->  HashAggregate
                          Output: t2.c1, t2.c1
                          Group By Key: t2.c1
                          Filter: (t2.c1 IS NOT NULL)
                          Unique Check Required    --如果在运行时输出了多于一行的数据，就会报错。
                          ->  Index Only Scan using t2idx on public.t2
                                Output: t2.c1
                                Distribute Key: t2.c1
(21 rows)
```

注意：因为分组group by t2.c1 unique check发生在过滤条件tt.c1=t1.c1之前，可能导致原来不报错的查询重写之后报错。举例：

有t1,t2表，其中的数据为：

```
yshen=# select * from t1 order by c2;
 c1 | c2 | c3
----+----+----
  1 |  1 |  1
  2 |  2 |  2
  3 |  3 |  3
  4 |  4 |  4
  5 |  5 |  5
  6 |  6 |  6
  7 |  7 |  7
  8 |  8 |  8
  9 |  9 |  9
 10 | 10 | 10
(10 rows)

yshen=# select * from t2 order by c1;
 c1 | c2 | c3
----+----+----
  1 |  1 |  1
  2 |  2 |  2
  3 |  3 |  3
  4 |  4 |  4
  5 |  5 |  5
  6 |  6 |  6
  7 |  7 |  7
  8 |  8 |  8
  9 |  9 |  9
 10 | 10 | 10
 11 | 11 | 11
 11 | 11 | 11
 12 | 12 | 12
 12 | 12 | 12
 13 | 13 | 13
 13 | 13 | 13
 14 | 14 | 14
 14 | 14 | 14
 15 | 15 | 15
 15 | 15 | 15
 16 | 16 | 16
 16 | 16 | 16
 17 | 17 | 17
 17 | 17 | 17
 18 | 18 | 18
 18 | 18 | 18
```



```
 19 | 19 | 19
 19 | 19 | 19
 20 | 20 | 20
 20 | 20 | 20
(30 rows)
```

分别关闭和打开uniquecheck参数对比，打开之后报错。

```
yshen=#  select t1.c1 from t1 where t1.c1 = (select t2.c1 from t2 where t1.c1=t2.c2) ;
 c1
----
  6
  7
  3
  1
  2
  4
  5
  8
  9
 10
(10 rows)

yshen=# set rewrite_rule='uniquecheck';
SET
yshen=#  select t1.c1 from t1 where t1.c1 = (select t2.c1 from t2 where t1.c1=t2.c2) ;
ERROR:  more than one row returned by a subquery used as an expression
```

将条件下推到子查询中 predpush、predpushnormal、predpushforce

通常优化器以查询块为单位进行优化，不同查询块独立优化，如果有涉及到跨查询块的谓词条件，难以从全局角度考虑谓词应用的位置。predpush可以将谓词下推到子查询块中，在父查询块中的数据量较小或子查询中可以利用索引的场景下能够提升性能。涉及到predpush的rewrite_rule规则有3个，分别是：

- predpushnormal：尝试下推谓词到子查询中，需要利用STREAM算子，如BROADCAST来实现分布式计划。
- predpushforce：尝试下推谓词到子查询中，尽量利用参数化路径的索引扫描。
- predpush：利用代价在predpushnormal和predpushforce中选择一个最优的分布式计划，但是会增加优化时间。

以下是关闭和开启该查询重写规则的计划示例：

```
openGauss=# set enable_fast_query_shipping=off; -- 关闭fqs优化
SET
openGauss=# show rewrite_rule;
 rewrite_rule
--------------
 magicset
(1 row)

openGauss=# explain (costs off) select * from t1, (select sum(c2), c1 from t2 group by c1) st2 where st2.c1
= t1.c1;
             QUERY PLAN
-------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Nested Loop
         Join Filter: (t1.c1 = t2.c1)
         ->  HashAggregate
               Group By Key: t2.c1
               ->  Seq Scan on t2
         ->  Seq Scan on t1
(8 rows)
```



```
openGauss=# set rewrite_rule='predpushnormal';
SET
openGauss=# explain (costs off) select * from t1, (select sum(c2), c1 from t2 group by c1) st2 where st2.c1
= t1.c1;
                QUERY PLAN
-------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Nested Loop
         ->  Seq Scan on t1
         ->  HashAggregate
               Group By Key: t2.c1
               ->  Result
                     Filter: (t1.c1 = t2.c1)
                     ->  Seq Scan on t2
(9 rows)

--可以看到过滤条件被推到子查询中执行。

openGauss=# set rewrite_rule='predpushforce';
SET

openGauss=# explain (costs off) select /*+predpush(t1 st2)*/ * from t1, (select  sum(c2), c1 from t2 group
by c1) st2 where st2.c1 = t1.c1;
                    QUERY PLAN
---------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Nested Loop
         ->  Seq Scan on t1
         ->  HashAggregate
               Group By Key: t2.c1
               ->  Index Scan using t2_c1_idx on t2
                     Index Cond: (t1.c1 = c1)
(8 rows)

--结合predpush hint一起使用，可以看到使用了参数化路径。

openGauss=# set rewrite_rule = 'predpush';
SET
openGauss=# explain (costs off) select * from t1, (select sum(c2), c1 from t2 group by c1) st2 where st2.c1
= t1.c1;
                    QUERY PLAN
---------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Nested Loop
         ->  Seq Scan on t1
         ->  HashAggregate
               Group By Key: t2.c1
               ->  Index Scan using t2_c1_idx on t2
                     Index Cond: (t1.c1 = c1)
(8 rows)
```


7 SQL 参考

7.1 GaussDB SQL

什么是 SQL

SQL是用于访问和处理数据库的标准计算机语言。

SQL提供了各种任务的语句，包括：

- 查询数据。
- 在表中插入、更新和删除行。
- 创建、替换、更改和删除对象。
- 控制对数据库及其对象的访问。
- 保证数据库的一致性和完整性。

SQL语言由用于处理数据库和数据库对象的命令和函数组成。该语言还会强制实施有关数据类型、表达式和文本使用的规则。因此在**SQL参考**章节，除了SQL语法参考外，还会看到有关数据类型、表达式、函数和操作符等信息。

SQL 发展简史

SQL发展简史如下：

- 1986年，ANSI X3.135-1986，ISO/IEC 9075:1986，SQL-86
- 1989年，ANSI X3.135-1989，ISO/IEC 9075:1989，SQL-89
- 1992年，ANSI X3.135-1992，ISO/IEC 9075:1992，SQL-92（SQL2）
- 1999年，ISO/IEC 9075:1999，SQL:1999（SQL3）
- 2003年，ISO/IEC 9075:2003，SQL:2003（SQL4）
- 2011年，ISO/IEC 9075:200N，SQL:2011（SQL5）

GaussDB 支持的 SQL 标准

GaussDB默认支持SQL2、SQL3和SQL4的主要特性。

7.2 关键字

SQL里有保留关键字和非保留关键字之分。根据标准，保留关键字绝不能用做其他标识符。非保留关键字只是在特定的环境里有特殊的含义，而在其他环境里是可以用做标识符的。

须知

1. 目前“非保留”关键字在作为数据库对象的标识符时存在如下限制：
   1. 不支持直接作为列别名使用，即类似SELECT 1 ABORT的用法会导致错误。
   2. 对于ENTITYESCAPING、NOENTITYESCAPING以及WELLFORMED关键字，无论是否带有双引号，均不支持作为表名、列名、表别名以及列别名的标识符。此外，不带双引号时不支持作为函数名。
   3. 不带双引号的RAW关键字不支持作为表名和函数名的标识符。
   4. 不带双引号的SET关键字不支持作为表别名的标识符，即类似SELECT * FROM T1 SET的用法均会导致错误。
   5. 不带双引号的BEGIN、BY、CLOSE、CURSOR、DECLARE、DELETE、EXECUTE、FUNCTION、IF、IMMEDIATE、INSERT、LOOP、MOVE、OF、REF、RELEASE、RETURN、SAVEPOINT、STRICT、TYPE以及UPDATE等关键字不支持作为变量名使用。
   6. 用SYS_REFCURSOR关键字作为数据库对象的标识符时，如果不附带双引号，则创建名为REFCURSOR的数据库对象，如果附带了双引号，则创建名为SYS_REFCURSOR的数据库对象。
2. 与“非保留”关键字类似，“非保留（不能是函数或类型）”关键字不支持直接作为列别名使用。
3. 对于未带有双引号的“保留”关键字CURRENT_TIMESTAMP而言，不允许作为函数名。

标识符命名规范

标识符的命名需要遵守如下规范：

- 标识符需要为字母（a-z）、下划线（_）、数字（0-9）或美元符号（$）。
- 标识符必须以字母（a-z）或下划线（_）开头。

  说明

  - 此命名规范为建议项，非强制项。
  - 特殊情况下可以使用双引号规避特殊字符报错。

SQL 关键字

表 7-1 SQL 关键字

关键字	GaussDB	SQL:1999	SQL-92
ABORT	非保留	-	-
ABS	-	非保留	-
ABSOLUTE	非保留	保留	保留
ACCESS	非保留	-	-
ACCOUNT	非保留	-	-
ACTION	非保留	保留	保留
ADA	-	非保留	非保留
ADD	非保留	保留	保留
ADMIN	非保留	保留	-
AFTER	非保留	保留	-
AGGREGATE	非保留	保留	-
ALGORITHM	非保留	-	-
ALIAS	-	保留	-
ALL	保留	保留	保留
ALLOCATE	-	保留	保留
ALSO	非保留	-	-
ALTER	非保留	保留	保留
ALWAYS	非保留	-	-
ANALYSE	保留	-	-
ANALYZE	保留	-	-
AND	保留	保留	保留
ANY	保留	保留	保留
APP	非保留	-	-
APPEND	非保留	-	-
ARCHIVE	非保留	-	-
ARE	-	保留	保留
ARRAY	保留	保留	-
AS	保留	保留	保留

关键字	GaussDB	SQL:1999	SQL-92
ASC	保留	保留	保留
ASENSITIVE	-	非保留	-
ASSERTION	非保留	保留	保留
ASSIGNMENT	非保留	非保留	-
ASYMMETRIC	保留	非保留	-
AT	非保留	保留	保留
ATOMIC	-	非保留	-
ATTRIBUTE	非保留	-	-
AUDIT	非保留	-	-
AUTHID	保留	-	-
AUTHORIZATION	保留(可以是函数或类型)	保留	保留
AUTOEXTEND	非保留	-	-
AUTOMAPPED	非保留	-	-
AVG	-	非保留	保留
BACKWARD	非保留	-	-
BARRIER	非保留	-	-
BEFORE	非保留	保留	-
BEGIN	非保留	保留	保留
BEGIN_NON_ANOYBLOCK	非保留	-	-
BETWEEN	非保留(不能是函数或类型)	非保留	保留
BIGINT	非保留(不能是函数或类型)	-	-
BINARY	保留(可以是函数或类型)	保留	-
BINARY_DOUBLE	非保留(不能是函数或类型)	-	-
BINARY_INTEGER	非保留(不能是函数或类型)	-	-
BIT	非保留(不能是函数或类型)	保留	保留

关键字	GaussDB	SQL:1999	SQL-92
BIT_LENGTH	-	非保留	保留
BITVAR	-	非保留	-
BLANKS	非保留	-	-
BLOB	非保留	保留	-
BLOCKCHAIN	非保留	-	-
BODY	非保留	-	-
BOOLEAN	非保留(不能是函数或类型)	保留	-
BOTH	保留	保留	保留
BREADTH	-	保留	-
BUCKETCNT	非保留(不能是函数或类型)	-	-
BUCKETS	保留	-	-
BY	非保留	保留	保留
BYTEAWITHOUTORDER	非保留(不能是函数或类型)	-	-
BYTEAWITHOUTORDERWITHEQUAL	非保留(不能是函数或类型)	-	-
C	-	非保留	非保留
CACHE	非保留	-	-
CALL	非保留	保留	-
CALLED	非保留	非保留	-
CANCELABLE	非保留	-	-
CARDINALITY	-	非保留	-
CASCADE	非保留	保留	保留
CASCADED	非保留	保留	保留
CASE	保留	保留	保留
CAST	保留	保留	保留
CATALOG	非保留	保留	保留
CATALOG_NAME	-	非保留	非保留
CHAIN	非保留	非保留	-

关键字	GaussDB	SQL:1999	SQL-92
CHAR	非保留(不能是函数或类型)	保留	保留
CHAR_LENGTH	-	非保留	保留
CHARACTER	非保留(不能是函数或类型)	保留	保留
CHARACTER_LENGTH	-	非保留	保留
CHARACTER_SET_CATALOG	-	非保留	非保留
CHARACTER_SET_NAME	-	非保留	非保留
CHARACTER_SET_SCHEMA	-	非保留	非保留
CHARACTERISTICS	非保留	-	-
CHARACTERSET	非保留	-	-
CHECK	保留	保留	保留
CHECKED	-	非保留	-
CHECKPOINT	非保留	-	-
CLASS	非保留	保留	-
CLASS_ORIGIN	-	非保留	非保留
CLEAN	非保留	-	-
CLIENT	非保留	-	-
CLIENT_MASTER_KEY	非保留	-	-
CLIENT_MASTER_KEYS	非保留	-	-
CLOB	非保留	保留	-
CLOSE	非保留	保留	保留
CLUSTER	非保留	-	-
COALESCE	非保留(不能是函数或类型)	非保留	保留
COBOL	-	非保留	非保留
COLLATE	保留	保留	保留
COLLATION	保留(可以是函数或类型)	保留	保留
COLLATION_CATALOG	-	非保留	非保留

关键字	GaussDB	SQL:1999	SQL-92
COLLATION_NAME	-	非保留	非保留
COLLATION_SCHEMA	-	非保留	非保留
COLUMN	保留	保留	保留
COLUMN_ENCRYPTION_KEY	非保留	-	-
COLUMN_ENCRYPTION_KEYS	非保留	-	-
COLUMN_NAME	-	非保留	非保留
COMMAND_FUNCTION	-	非保留	非保留
COMMAND_FUNCTION_CODE	-	非保留	-
COMMENT	非保留	-	-
COMMENTS	非保留	-	-
COMMIT	非保留	保留	保留
COMMITTED	非保留	非保留	非保留
COMPACT	保留(可以是函数或类型)	-	-
COMPATIBLE_ILLEGAL_CHARS	非保留	-	-
COMPLETE	非保留	-	-
COMPLETION	-	保留	-
CONCURRENTLY	保留(可以是函数或类型)	-	-
CONDITION	非保留	-	-
CONDITION_NUMBER	-	非保留	非保留
CONFIGURATION	非保留	-	-
CONNECT	非保留	保留	保留
CONNECTION	非保留	保留	保留
CONNECTION_NAME	-	非保留	非保留
CONSTANT	非保留	-	-
CONSTRAINT	保留	保留	保留
CONSTRAINT_CATALOG	-	非保留	非保留

关键字	GaussDB	SQL:1999	SQL-92
CONSTRAINT_NAME	-	非保留	非保留
CONSTRAINT_SCHEMA	-	非保留	非保留
CONSTRAINTS	非保留	保留	保留
CONSTRUCTOR	-	保留	-
CONTAINS	-	非保留	-
CONTENT	非保留	-	-
CONTINUE	非保留	保留	保留
CONTVIEW	非保留	-	-
CONVERSION	非保留	-	-
CONVERT	-	非保留	保留
COORDINATOR	非保留	-	-
COORDINATORS	非保留	-	-
COPY	非保留	-	-
CORRESPONDING	-	保留	保留
COST	非保留	-	-
COUNT	-	非保留	保留
CREATE	保留	保留	保留
CROSS	保留(可以是函数或类型)	保留	保留
CSN	保留(可以是函数或类型)	-	-
CSV	非保留	-	-
CUBE	非保留	保留	-
CURRENT	非保留	保留	保留
CURRENT_CATALOG	保留	-	-
CURRENT_DATE	保留	保留	保留
CURRENT_PATH	-	保留	-
CURRENT_ROLE	保留	保留	-
CURRENT_SCHEMA	保留(可以是函数或类型)	-	-
CURRENT_TIME	保留	保留	保留

关键字	GaussDB	SQL:1999	SQL-92
CURRENT_TIMESTAMP	保留	保留	保留
CURRENT_USER	保留	保留	保留
CURSOR	非保留	保留	保留
CURSOR_NAME	-	非保留	非保留
CYCLE	非保留	保留	-
DATA	非保留	保留	非保留
DATABASE	非保留	-	-
DATAFILE	非保留	-	-
DATANODE	非保留	-	-
DATANODES	非保留	-	-
DATATYPE_CL	非保留	-	-
DATE	非保留(不能是函数或类型)	保留	保留
DATE_FORMAT	非保留	-	-
DATETIME_INTERVAL_CODE	-	非保留	非保留
DATETIME_INTERVAL_PRECISION	-	非保留	非保留
DAY	非保留	保留	保留
DBCOMPATIBILITY	非保留	-	-
DEALLOCATE	非保留	保留	保留
DEC	非保留(不能是函数或类型)	保留	保留
DECIMAL	非保留(不能是函数或类型)	保留	保留
DECLARE	非保留	保留	保留
DECODE	非保留(不能是函数或类型)	-	-
DEFAULT	保留	保留	保留
DEFAULTS	非保留	-	-
DEFERRABLE	保留	保留	保留
DEFERRED	非保留	保留	保留

关键字	GaussDB	SQL:1999	SQL-92
DEFINED	-	非保留	-
DEFINER	非保留	非保留	-
DELETE	非保留	保留	保留
DELIMITER	非保留	-	-
DELIMITERS	非保留	-	-
DELTA	非保留	-	-
DELTAMERGE	保留(可以是函数或类型)	-	-
DEPTH	-	保留	-
DEREF	-	保留	-
DESC	保留	保留	保留
DESCRIBE	-	保留	保留
DESCRIPTOR	-	保留	保留
DESTROY	-	保留	-
DESTRUCTOR	-	保留	-
DETERMINISTIC	非保留	保留	-
DIAGNOSTICS	-	保留	保留
DICTIONARY	非保留	保留	-
DIRECT	非保留	-	-
DIRECTORY	非保留	-	-
DISABLE	非保留	-	-
DISCARD	非保留	-	-
DISCONNECT	非保留	保留	保留
DISPATCH	-	非保留	-
DISTINCT	保留	保留	保留
DISTRIBUTE	非保留	-	-
DISTRIBUTION	非保留	-	-
DO	保留	-	-
DOCUMENT	非保留	-	-
DOMAIN	非保留	保留	保留

关键字	GaussDB	SQL:1999	SQL-92
DOUBLE	非保留	保留	保留
DROP	非保留	保留	保留
DUPLICATE	非保留	-	-
DYNAMIC	-	保留	-
DYNAMIC_FUNCTION	-	非保留	非保留
DYNAMIC_FUNCTION_CODE	-	非保留	-
EACH	非保留	保留	-
ELASTIC	非保留	-	-
ELSE	保留	保留	保留
ENABLE	非保留	-	-
ENCLOSED	非保留	-	-
ENCODING	非保留	-	-
ENCRYPTED	非保留	-	-
ENCRYPTED_VALUE	非保留	-	-
ENCRYPTION	非保留	-	-
ENCRYPTION_TYPE	非保留	-	-
END	保留	保留	保留
END-EXEC	-	保留	保留
ENFORCED	非保留	-	-
ENUM	非保留	-	-
EOL	非保留	-	-
EQUALS	-	保留	-
ERRORS	非保留	-	-
ESCAPE	非保留	保留	保留
ESCAPING	非保留	-	-
EVERY	非保留	保留	-
EXCEPT	保留	保留	保留
EXCEPTION	-	保留	保留
EXCHANGE	非保留	-	-

关键字	GaussDB	SQL:1999	SQL-92
EXCLUDE	非保留	-	-
EXCLUDED	保留	-	-
EXCLUDING	非保留	-	-
EXCLUSIVE	非保留	-	-
EXEC	-	保留	保留
EXECUTE	非保留	保留	保留
EXISTING	-	非保留	-
EXISTS	非保留(不能是函数或类型)	非保留	保留
EXPIRED	非保留	-	-
EXPLAIN	非保留	-	-
EXTENSION	非保留	-	-
EXTERNAL	非保留	保留	保留
EXTRACT	非保留(不能是函数或类型)	非保留	保留
FALSE	-	保留	保留
FAMILY	非保留	-	-
FAST	非保留	-	-
FEATURES	非保留	-	-
FENCED	保留	-	-
FETCH	保留	保留	保留
FIELDS	非保留	-	-
FILEHEADER	非保留	-	-
FILL_MISSING_FIELDS	非保留	-	-
FILLER	非保留	-	-
FILTER	非保留	-	保留
FINAL	-	非保留	-
FIRST	非保留	保留	保留
FIXED	非保留	-	保留
FLOAT	非保留(不能是函数或类型)	保留	保留

关键字	GaussDB	SQL:1999	SQL-92
FOLLOWING	非保留	-	-
FOR	保留	保留	保留
FORCE	非保留	-	-
FOREIGN	保留	保留	保留
FORMATTER	非保留	-	-
FORTRAN	-	非保留	非保留
FORWARD	非保留	-	-
FOUND	-	保留	保留
FREE	-	保留	-
FREEZE	保留(可以是函数或类型)	-	-
FROM	保留	保留	保留
FULL	保留(可以是函数或类型)	保留	保留
FUNCTION	非保留	保留	-
FUNCTIONS	非保留	-	-
G	-	非保留	-
GENERAL	-	保留	-
GENERATED	非保留	非保留	-
GET	-	保留	保留
GLOBAL	非保留	保留	保留
GO	-	保留	保留
GOTO	-	保留	保留
GRANT	保留	保留	保留
GRANTED	非保留	非保留	-
GREATEST	非保留(不能是函数或类型)	-	-
GROUP	保留	保留	保留
GROUPING	非保留(不能是函数或类型)	保留	-
GROUPPARENT	保留	-	-
HANDLER	非保留	-	-

关键字	GaussDB	SQL:1999	SQL-92
HAVING	保留	保留	保留
HDFSDIRECTORY	保留(可以是函数或类型)	-	-
HEADER	非保留	-	-
HIERARCHY	-	非保留	-
HOLD	非保留	非保留	-
HOST	-	保留	-
HOUR	非保留	保留	保留
IDENTIFIED	非保留	-	-
IDENTITY	非保留	保留	保留
IF	非保留	-	-
IGNORE	-	保留	-
IGNORE_EXTRA_DATA	非保留	-	-
ILIKE	保留(可以是函数或类型)	-	-
IMMEDIATE	非保留	保留	保留
IMMUTABLE	非保留	-	-
IMPLEMENTATION	-	非保留	-
IMPLICIT	非保留	-	-
IN	保留	保留	保留
INCLUDE	非保留	-	-
INCLUDING	非保留	-	-
INCREMENT	非保留	-	-
INCREMENTAL	非保留	-	-
INDEX	非保留	-	-
INDEXES	非保留	-	-
INDICATOR	-	保留	保留
INFILE	非保留	-	-
INFIX	-	非保留	-
INHERIT	非保留	-	-
INHERITS	非保留	-	-

关键字	GaussDB	SQL:1999	SQL-92
INITIAL	非保留	-	-
INITIALIZE	-	保留	-
INITIALLY	保留	保留	保留
INITRANS	非保留	-	-
INLINE	非保留	-	-
INNER	保留(可以是函数或类型)	保留	保留
INOUT	非保留(不能是函数或类型)	保留	-
INPUT	非保留	保留	保留
INSENSITIVE	非保留	非保留	保留
INSERT	非保留	保留	保留
INSTANCE	-	非保留	-
INSTANTIABLE	-	非保留	-
INSTEAD	非保留	-	-
INT	非保留(不能是函数或类型)	保留	保留
INTEGER	非保留(不能是函数或类型)	保留	保留
INTERNAL	非保留	-	-
INTERSECT	保留	保留	保留
INTERVAL	非保留(不能是函数或类型)	保留	保留
INTO	保留	保留	保留
INVOKER	非保留	非保留	-
IP	非保留	-	-
IS	保留	保留	保留
ISNULL	非保留	-	-
ISOLATION	非保留	保留	保留
ITERATE	-	保留	-
JOIN	保留(可以是函数或类型)	保留	保留

关键字	GaussDB	SQL:1999	SQL-92
K	-	非保留	-
KEY	非保留	保留	保留
KEY_MEMBER	-	非保留	-
KEY_PATH	非保留	-	-
KEY_STORE	非保留	-	-
KEY_TYPE	-	非保留	-
KILL	非保留	-	-
LABEL	非保留	-	-
LANGUAGE	非保留	保留	保留
LARGE	非保留	保留	-
LAST	非保留	保留	保留
LATERAL	-	保留	-
LC_COLLATE	非保留	-	-
LC_CTYPE	非保留	-	-
LEADING	保留	保留	保留
LEAKPROOF	非保留	-	-
LEAST	非保留(不能是函数或类型)	-	-
LEFT	保留(可以是函数或类型)	保留	保留
LENGTH	-	非保留	非保留
LESS	保留	保留	-
LEVEL	非保留	保留	保留
LIKE	保留(可以是函数或类型)	保留	保留
LIMIT	保留	保留	-
LIST	非保留	-	-
LISTEN	非保留	-	-
LOAD	非保留	-	-
LOCAL	非保留	保留	保留
LOCALTIME	保留	保留	-

关键字	GaussDB	SQL:1999	SQL-92
LOCALTIMESTAMP	保留	保留	-
LOCATION	非保留	-	-
LOCATOR	-	保留	-
LOCK	非保留	-	-
LOG	非保留	-	-
LOGGING	非保留	-	-
LOGIN_ANY	非保留	-	-
LOGIN_FAILURE	非保留	-	-
LOGIN_SUCCESS	非保留	-	-
LOGOUT	非保留	-	-
LOOP	非保留	-	-
LOWER	-	非保留	保留
M	-	非保留	-
MAP	-	保留	-
MAPPING	非保留	-	-
MASKING	非保留	-	-
MASTER	非保留	-	-
MATCH	非保留	保留	保留
MATCHED	非保留	-	-
MATERIALIZED	非保留	-	-
MAX	-	非保留	保留
MAXEXTENTS	非保留	-	-
MAXSIZE	非保留	-	-
MAXTRANS	非保留	-	-
MAXVALUE	保留	-	-
MERGE	非保留	-	-
MESSAGE_LENGTH	-	非保留	非保留
MESSAGE_OCTET_LENGTH	-	非保留	非保留
MESSAGE_TEXT	-	非保留	非保留

关键字	GaussDB	SQL:1999	SQL-92
METHOD	-	非保留	-
MIN	-	非保留	保留
MINEXTENTS	非保留	-	-
MINUS	保留	-	-
MINUTE	非保留	保留	保留
MINVALUE	非保留	-	-
MOD	-	非保留	-
MODE	非保留	-	-
MODEL	非保留	-	-
MODIFIES	-	保留	-
MODIFY	保留	保留	-
MODULE	-	保留	保留
MONTH	非保留	保留	保留
MORE	-	非保留	非保留
MOVE	非保留	-	-
MOVEMENT	非保留	-	-
MUMPS	-	非保留	非保留
NAME	非保留	非保留	非保留
NAMES	非保留	保留	保留
NATIONAL	非保留(不能是函数或类型)	保留	保留
NATURAL	保留(可以是函数或类型)	保留	保留
NCHAR	非保留(不能是函数或类型)	保留	保留
NCLOB	-	保留	-
NEW	-	保留	-
NEXT	非保留	保留	保留
NO	非保留	保留	保留
NOCYCLE	保留	-	-
NODE	非保留	-	-

关键字	GaussDB	SQL:1999	SQL-92
NOLOGGING	非保留	-	-
NOMAXVALUE	非保留	-	-
NOMINVALUE	非保留	-	-
NONE	非保留(不能是函数或类型)	保留	-
NOT	保留	保留	保留
NOTHING	非保留	-	-
NOTIFY	非保留	-	-
NOTNULL	保留(可以是函数或类型)	-	-
NOWAIT	非保留	-	-
NULL	保留	保留	保留
NULLABLE	-	非保留	非保留
NULLCOLS	非保留	-	-
NULLIF	非保留(不能是函数或类型)	非保留	保留
NULLS	非保留	-	-
NUMBER	非保留(不能是函数或类型)	非保留	非保留
NUMERIC	非保留(不能是函数或类型)	保留	保留
NUMSTR	非保留	-	-
NVARCHAR2	非保留(不能是函数或类型)	-	-
NVL	非保留(不能是函数或类型)	-	-
OBJECT	非保留	保留	-
OCTET_LENGTH	-	非保留	保留
OF	非保留	保留	保留
OFF	非保留	保留	-
OFFSET	保留	-	-
OIDS	非保留	-	-
OLD	-	保留	-

关键字	GaussDB	SQL:1999	SQL-92
ON	保留	保留	保留
ONLY	保留	保留	保留
OPEN	-	保留	保留
OPERATION	-	保留	-
OPERATOR	非保留	-	-
OPTIMIZATION	非保留	-	-
OPTION	非保留	保留	保留
OPTIONALLY	非保留	-	-
OPTIONS	非保留	非保留	-
OR	保留	保留	保留
ORDER	保留	保留	保留
ORDINALITY	-	保留	-
OUT	非保留(不能是函数或类型)	保留	-
OUTER	保留(可以是函数或类型)	保留	保留
OUTPUT	-	保留	保留
OVER	非保留	-	-
OVERLAPS	保留(可以是函数或类型)	非保留	保留
OVERLAY	非保留(不能是函数或类型)	非保留	-
OVERRIDING	-	非保留	-
OWNED	非保留	-	-
OWNER	非保留	-	-
PACKAGE	非保留	-	-
PACKAGES	非保留	-	-
PAD	-	保留	保留
PARAMETER	-	保留	-
PARAMETER_MODE	-	非保留	-
PARAMETER_NAME	-	非保留	-

关键字	GaussDB	SQL:1999	SQL-92
PARAMETER_ORDINAL_POSITION	-	非保留	-
PARAMETER_SPECIFIC_CATALOG	-	非保留	-
PARAMETER_SPECIFIC_NAME	-	非保留	-
PARAMETER_SPECIFIC_SCHEMA	-	非保留	-
PARAMETERS	-	保留	-
PARSER	非保留	-	-
PARTIAL	非保留	保留	保留
PARTITION	非保留	-	-
PARTITIONS	非保留	-	-
PASCAL	-	非保留	非保留
PASSING	非保留	-	-
PASSWORD	非保留	-	-
PATH	-	保留	-
PCTFREE	非保留	-	-
PER	非保留	-	-
PERCENT	非保留	-	-
PERFORMANCE	保留	-	-
PERM	非保留	-	-
PLACING	保留	-	-
PLAN	非保留	-	-
PLANS	非保留	-	-
PLI	-	非保留	非保留
POLICY	非保留	-	-
POOL	非保留	-	-
POSITION	非保留(不能是函数或类型)	非保留	保留
POSTFIX	-	保留	-
PRECEDING	非保留	-	-

关键字	GaussDB	SQL:1999	SQL-92
PRECISION	非保留(不能是函数或类型)	保留	保留
PREDICT	非保留	-	-
PREFERRED	非保留	-	-
PREFIX	非保留	保留	-
PREORDER	-	保留	-
PREPARE	非保留	保留	保留
PREPARED	非保留	-	-
PRESERVE	非保留	保留	保留
PRIMARY	保留	保留	保留
PRIOR	非保留	保留	保留
PRIORER	保留	-	-
PRIVATE	非保留	-	-
PRIVILEGE	非保留	-	-
PRIVILEGES	非保留	保留	保留
PROCEDURAL	非保留	-	-
PROCEDURE	保留	保留	保留
PROFILE	非保留	-	-
PUBLIC	-	保留	保留
PUBLICATION	非保留	-	-
PUBLISH	非保留	-	-
PURGE	非保留	-	-
QUERY	非保留	-	-
QUOTE	非保留	-	-
RANDOMIZED	非保留	-	-
RANGE	非保留	-	-
RATIO	非保留	-	-
RAW	非保留	-	-
READ	非保留	保留	保留
READS	-	保留	-

关键字	GaussDB	SQL:1999	SQL-92
REAL	非保留(不能是函数或类型)	保留	保留
REASSIGN	非保留	-	-
REBUILD	非保留	-	-
RECHECK	非保留	-	-
RECURSIVE	非保留	保留	-
RECYCLEBIN	保留(可以是函数或类型)	-	-
REDISANYVALUE	非保留	-	-
REF	非保留	保留	-
REFERENCES	保留	保留	保留
REFERENCING	-	保留	-
REFRESH	非保留	-	-
REINDEX	非保留	-	-
REJECT	保留	-	-
RELATIVE	非保留	保留	保留
RELEASE	非保留	-	-
RELOPTIONS	非保留	-	-
REMOTE	非保留	-	-
REMOVE	非保留	-	-
RENAME	非保留	-	-
REPEATABLE	非保留	非保留	非保留
REPLACE	非保留	-	-
REPLICA	非保留	-	-
RESET	非保留	-	-
RESIZE	非保留	-	-
RESOURCE	非保留	-	-
RESTART	非保留	-	-
RESTRICT	非保留	保留	保留
RESULT	-	保留	-
RETURN	非保留	保留	-

关键字	GaussDB	SQL:1999	SQL-92
RETURNED_LENGTH	-	非保留	非保留
RETURNED_OCTET_LENGTH	-	非保留	非保留
RETURNED_SQLSTATE	-	非保留	非保留
RETURNING	保留	-	-
RETURNS	非保留	保留	-
REUSE	非保留	-	-
REVOKE	非保留	保留	保留
RIGHT	保留(可以是函数或类型)	保留	保留
ROLE	非保留	保留	-
ROLES	非保留	-	-
ROLLBACK	非保留	保留	保留
ROLLUP	非保留	保留	-
ROTATION	非保留	-	-
ROUTINE	-	保留	-
ROUTINE_CATALOG	-	非保留	-
ROUTINE_NAME	-	非保留	-
ROUTINE_SCHEMA	-	非保留	-
ROW	非保留(不能是函数或类型)	保留	-
ROW_COUNT	-	非保留	非保留
ROWNUM	保留	-	-
ROWS	非保留	保留	保留
ROWTYPE	非保留	-	-
RULE	非保留	-	-
SAMPLE	非保留	-	-
SAVEPOINT	非保留	保留	-
SCALE	-	非保留	非保留
SCHEMA	非保留	保留	保留
SCHEMA_NAME	-	非保留	非保留

关键字	GaussDB	SQL:1999	SQL-92
SCOPE	-	保留	-
SCROLL	非保留	保留	保留
SEARCH	非保留	保留	-
SECOND	非保留	保留	保留
SECTION	-	保留	保留
SECURITY	非保留	非保留	-
SELECT	保留	保留	保留
SELF	-	非保留	-
SENSITIVE	-	非保留	-
SEQUENCE	非保留	保留	-
SEQUENCES	非保留	-	-
SERIALIZABLE	非保留	非保留	非保留
SERVER	非保留	-	-
SERVER_NAME	-	非保留	非保留
SESSION	非保留	保留	保留
SESSION_USER	保留	保留	保留
SET	非保留	保留	保留
SETOF	非保留(不能是函数或类型)	-	-
SETS	非保留	保留	-
SHARE	非保留	-	-
SHIPPABLE	非保留	-	-
SHOW	非保留	-	-
SHUTDOWN	非保留	-	-
SIBLINGS	非保留	-	-
SIMILAR	保留(可以是函数或类型)	非保留	-
SIMPLE	非保留	非保留	-
SIZE	非保留	保留	保留
SKIP	非保留	-	-
SLICE	非保留	-	-

关键字	GaussDB	SQL:1999	SQL-92
SMALLDATETIME	非保留(不能是函数或类型)	-	-
SMALLDATETIME_FORMAT	非保留	-	-
SMALLINT	非保留(不能是函数或类型)	保留	保留
SNAPSHOT	非保留	-	-
SOME	保留	保留	保留
SOURCE	非保留	非保留	-
SPACE	非保留	保留	保留
SPECIFIC	-	保留	-
SPECIFIC_NAME	-	非保留	-
SPECIFICTYPE	-	保留	-
SPILL	非保留	-	-
SPLIT	非保留	-	-
SQL	-	保留	保留
SQLCODE	-	-	保留
SQLERROR	-	-	保留
SQLEXCEPTION	-	保留	-
SQLSTATE	-	保留	保留
SQLWARNING	-	保留	-
STABLE	非保留	-	-
STANDALONE	非保留	-	-
START	非保留	保留	-
STATE	-	保留	-
STATEMENT	非保留	保留	-
STATEMENT_ID	非保留	-	-
STATIC	-	保留	-
STATISTICS	非保留	-	-
STDIN	非保留	-	-
STDOUT	非保留	-	-

关键字	GaussDB	SQL:1999	SQL-92
STORAGE	非保留	-	-
STORE	非保留	-	-
STORED	非保留	-	-
STRATIFY	非保留	-	-
STREAM	非保留	-	-
STRICT	非保留	-	-
STRIP	非保留	-	-
STRUCTURE	-	保留	-
STYLE	-	非保留	-
SUBCLASS_ORIGIN	-	非保留	非保留
SUBLIST	-	非保留	-
SUBPARTITION	非保留	-	-
SUBSCRIPTION	非保留	-	-
SUBSTRING	非保留(不能是函数或类型)	非保留	保留
SUM	-	非保留	保留
SYMMETRIC	保留	非保留	-
SYNONYM	非保留	-	-
SYS_REFCURSOR	非保留	-	-
SYSDATE	保留	-	-
SYSID	非保留	-	-
SYSTEM	非保留	非保留	-
SYSTEM_USER	-	保留	保留
TABLE	保留	保留	保留
TABLE_NAME	-	非保留	非保留
TABLES	非保留	-	-
TABLESAMPLE	保留(可以是函数或类型)	-	-
TABLESPACE	非保留	-	-
TARGET	非保留	-	-
TEMP	非保留	-	-

关键字	GaussDB	SQL:1999	SQL-92
TEMPLATE	非保留	-	-
TEMPORARY	非保留	保留	保留
TERMINATE	-	保留	-
TERMINATED	非保留	-	-
TEXT	非保留	-	-
THAN	非保留	保留	-
THEN	保留	保留	保留
TIME	非保留(不能是函数或类型)	保留	保留
TIME_FORMAT	非保留	-	-
TIMECAPSULE	保留(可以是函数或类型)	-	-
TIMESTAMP	非保留(不能是函数或类型)	保留	保留
TIMESTAMP_FORMAT	非保留	-	-
TIMESTAMPDIFF	非保留(不能是函数或类型)	-	-
TIMEZONE_HOUR	-	保留	保留
TIMEZONE_MINUTE	-	保留	保留
TINYINT	非保留(不能是函数或类型)	-	-
TO	保留	保留	保留
TRAILING	保留	保留	保留
TRANSACTION	非保留	保留	保留
TRANSACTION_ACTIVE	-	非保留	-
TRANSACTIONS_COMMITTED	-	非保留	-
TRANSACTIONS_ROLLED_BACK	-	非保留	-
TRANSFORM	非保留	非保留	-
TRANSFORMS	-	非保留	-
TRANSLATE	-	非保留	保留
TRANSLATION	-	保留	保留

关键字	GaussDB	SQL:1999	SQL-92
TREAT	非保留(不能是函数或类型)	保留	-
TRIGGER	非保留	保留	-
TRIGGER_CATALOG	-	非保留	-
TRIGGER_NAME	-	非保留	-
TRIGGER_SCHEMA	-	非保留	-
TRIM	非保留(不能是函数或类型)	非保留	保留
TRUE	-	保留	保留
TRUNCATE	非保留	-	-
TRUSTED	非保留	-	-
TSFIELD	非保留	-	-
TSTAG	非保留	-	-
TSTIME	非保留	-	-
TYPE	非保留	非保留	非保留
TYPES	非保留	-	-
UNBOUNDED	非保留	-	-
UNCOMMITTED	非保留	非保留	非保留
UNDER	-	保留	-
UNENCRYPTED	非保留	-	-
UNION	保留	保留	保留
UNIQUE	保留	保留	保留
UNKNOWN	非保留	保留	保留
UNLIMITED	非保留	-	-
UNLISTEN	非保留	-	-
UNLOCK	非保留	-	-
UNLOGGED	非保留	-	-
UNNAMED	-	非保留	非保留
UNNEST	-	保留	-
UNTIL	非保留	-	-
UNUSABLE	非保留	-	-

关键字	GaussDB	SQL:1999	SQL-92
UPDATE	非保留	保留	保留
UPPER	-	非保留	保留
USAGE	-	保留	保留
USEEOF	非保留	-	-
USER	保留	保留	保留
USER_DEFINED_TYPE_CATALOG	-	非保留	-
USER_DEFINED_TYPE_NAME	-	非保留	-
USER_DEFINED_TYPE_SCHEMA	-	非保留	-
USING	保留	保留	保留
VACUUM	非保留	-	-
VALID	非保留	-	-
VALIDATE	非保留	-	-
VALIDATION	非保留	-	-
VALIDATOR	非保留	-	-
VALUE	非保留	保留	保留
VALUES	非保留(不能是函数或类型)	保留	保留
VARCHAR	非保留(不能是函数或类型)	保留	保留
VARCHAR2	非保留(不能是函数或类型)	-	-
VARIABLE	-	保留	-
VARIABLES	非保留	-	-
VARIADIC	保留	-	-
VARYING	非保留	保留	保留
VCGROUP	非保留	-	-
VERBOSE	保留(可以是函数或类型)	-	-
VERIFY	保留	-	-
VERSION	非保留	-	-

关键字	GaussDB	SQL:1999	SQL-92
VIEW	非保留	保留	保留
VOLATILE	非保留	-	-
WAIT	非保留	-	-
WEAK	非保留	-	-
WHEN	保留	保留	保留
WHENEVER	-	保留	保留
WHERE	保留	保留	保留
WHITESPACE	非保留	-	-
WINDOW	保留	-	-
WITH	保留	保留	保留
WITHIN	非保留	-	-
WITHOUT	非保留	保留	-
WORK	非保留	保留	保留
WORKLOAD	非保留	-	-
WRAPPER	非保留	-	-
WRITE	非保留	保留	保留
XML	非保留	-	-
XMLATTRIBUTES	非保留(不能是函数或类型)	-	-
XMLCONCAT	非保留(不能是函数或类型)	-	-
XMLELEMENT	非保留(不能是函数或类型)	-	-
XMLEXISTS	非保留(不能是函数或类型)	-	-
XMLFOREST	非保留(不能是函数或类型)	-	-
XMLPARSE	非保留(不能是函数或类型)	-	-
XMLPI	非保留(不能是函数或类型)	-	-
XMLROOT	非保留(不能是函数或类型)	-	-

关键字	GaussDB	SQL:1999	SQL-92
XMLSERIALIZE	非保留(不能是函数或类型)	-	-
YEAR	非保留	保留	保留
YES	非保留	-	-
ZONE	非保留	保留	保留

下表所示字段在建表时禁止作为列名。

CTID	XMIN	CMIN	XMAX	CMAX
TABLEOID	XC_NODE_ID	TID	GS_TUPLE_UID	TABLEBUCKETID

7.3 数据类型

数据类型是数据的一个基本属性，用于区分不同类型的数据。不同的数据类型所占的存储空间不同，能够进行的操作也不相同。数据库中的数据存储在数据表中。数据表中的每一列都定义了数据类型，用户存储数据时，须遵从这些数据类型的属性，否则可能会出错。

GaussDB支持某些数据类型间的隐式转换，具体转换关系请参见**PG_CAST**。

7.3.1 数值类型

表7-2列出了所有的可用类型。数字操作符和相关的内置函数请参见**数字操作函数和操作符**。

表 7-2 整数类型

名称	描述	存储空间	范围
TINYINT	微整数，别名为INT1。	1字节	0 ~ 255
SMALLINT	小范围整数，别名为INT2。	2字节	-32,768 ~ +32,767
INTEGER	常用的整数，别名为INT4。	4字节	-2,147,483,648 ~ +2,147,483,647
BINARY_INTEGER	常用的整数，INTEGER的别名，为兼容ORA数据库类型。	4字节	-2,147,483,648 ~ +2,147,483,647

名称	描述	存储空间	范围
BIGINT	大范围的整数，别名为INT8。	8字节	-9,223,372,036,854,775,808 ~ +9,223,372,036,854,775,807
int16	十六字节的大范围整数，目前不支持用户用于建表等使用。	16字节	-170,141,183,460,469,231,731,687,303,715,884,105,728 ~ +170,141,183,460,469,231,731,687,303,715,884,105,727

示例：

```
--创建具有TINYINT类型数据的表。
openGauss=# CREATE TABLE int_type_t1
        (
         IT_COL1 TINYINT
        );

--插入数据。
openGauss=# INSERT INTO int_type_t1 VALUES(10);

--查看数据。
openGauss=# SELECT * FROM int_type_t1;
 it_col1
---------
 10
(1 row)

--删除表。
openGauss=# DROP TABLE int_type_t1;
--创建具有TINYINT,INTEGER,BIGINT类型数据的表。
openGauss=# CREATE TABLE int_type_t2
(
    a TINYINT,
    b TINYINT,
    c INTEGER,
    d BIGINT
);

--插入数据。
openGauss=# INSERT INTO int_type_t2 VALUES(100, 10, 1000, 10000);

--查看数据。
openGauss=# SELECT * FROM int_type_t2;
  a  | b  |  c   |   d
-----+----+------+-------
 100 | 10 | 1000 | 10000
(1 row)

--删除表。
openGauss=# DROP TABLE int_type_t2;
```

说明

- TINYINT、SMALLINT、INTEGER、BIGINT和INT16类型存储各种范围的数字，也就是整数。试图存储超出范围以外的数值将会导致错误。
- 常用的类型是INTEGER，因为它提供了在范围、存储空间、性能之间的最佳平衡。一般只有取值范围确定不超过SMALLINT的情况下，才会使用SMALLINT类型。而只有在INTEGER的范围不够的时候才使用BIGINT，因为前者相对快得多。

表 7-3 任意精度类型

名称	描述	存储空间	范围
NUMERIC[(p[,s])], DECIMAL[(p[,s])]	精度p取值范围为[1,1000]，标度s取值范围为[0,p]。 **说明** p为总位数，s为小数位数。	用户声明精度。每四位（十进制位）占用两个字节，然后在整个数据上加上八个字节的额外开销。	未指定精度的情况下，小数点前最大131,072位，小数点后最大16,383位。
NUMBER[(p[,s])]	NUMERIC类型的别名。	用户声明精度。每四位（十进制位）占用两个字节，然后在整个数据上加上八个字节的额外开销。	未指定精度的情况下，小数点前最大131,072位，小数点后最大16,383位。

示例：

```
--创建表。
openGauss=# CREATE TABLE decimal_type_t1
(
    DT_COL1 DECIMAL(10,4)
);

--插入数据。
openGauss=# INSERT INTO decimal_type_t1 VALUES(123456.122331);

--查询表中的数据。
openGauss=# SELECT * FROM decimal_type_t1;
   dt_col1
-------------
 123456.1223
(1 row)

--删除表。
openGauss=# DROP TABLE decimal_type_t1;
--创建表。
openGauss=# CREATE TABLE numeric_type_t1
(
    NT_COL1 NUMERIC(10,4)
);

--插入数据。
openGauss=# INSERT INTO numeric_type_t1 VALUES(123456.12354);

--查询表中的数据。
openGauss=# SELECT * FROM numeric_type_t1;
   nt_col1
-------------
 123456.1235
(1 row)

--删除表。
openGauss=# DROP TABLE numeric_type_t1;
```


说明

- 与整数类型相比，任意精度类型需要更大的存储空间，其存储效率、运算效率以及压缩比效果都要差一些。在进行数值类型定义时，优先选择整数类型。当且仅当数值超出整数可表示最大范围时，再选用任意精度类型。
- 使用NUMERIC/DECIMAL进行列定义时，建议指定该列的精度p以及标度s。

表 7-4 序列整型

名称	描述	存储空间	范围
SMALLSERIAL	二字节序列整型。	2字节	-32,768 ~ +32,767
SERIAL	四字节序列整型。	4字节	-2,147,483,648 ~ +2,147,483,647
BIGSERIAL	八字节序列整型。	8字节	-9,223,372,036,854,775,808 ~ +9,223,372,036,854,775,807

示例：

```
--创建表。
openGauss=# CREATE TABLE smallserial_type_tab(a SMALLSERIAL);

--插入数据。
openGauss=# INSERT INTO smallserial_type_tab VALUES(default);

--再次插入数据。
openGauss=# INSERT INTO smallserial_type_tab VALUES(default);

--查看数据。
openGauss=# SELECT * FROM smallserial_type_tab;
 a
---
 1
 2
(2 rows)

--创建表。
openGauss=# CREATE TABLE serial_type_tab(b SERIAL);

--插入数据。
openGauss=# INSERT INTO serial_type_tab VALUES(default);

--再次插入数据。
openGauss=# INSERT INTO serial_type_tab VALUES(default);

--查看数据。
openGauss=# SELECT * FROM serial_type_tab;
 b
---
 1
 2
(2 rows)

--创建表。
openGauss=# CREATE TABLE bigserial_type_tab(c BIGSERIAL);

--插入数据。
openGauss=# INSERT INTO bigserial_type_tab VALUES(default);

--再次插入数据。
openGauss=# INSERT INTO bigserial_type_tab VALUES(default);
```



```
--查看数据。
openGauss=# SELECT * FROM bigserial_type_tab;
 c
---
 1
 2
(2 rows)

--删除表。
openGauss=# DROP TABLE smallserial_type_tab;

openGauss=# DROP TABLE serial_type_tab;

openGauss=# DROP TABLE bigserial_type_tab;
```

说明

SMALLSERIAL，SERIAL和BIGSERIAL类型不是真正的类型，只是为在表中设置唯一标识做的概念上的便利。因此，创建一个整数字段，并且把它的缺省数值安排为从一个序列发生器读取。应用了一个NOT NULL约束以确保NULL不会被插入。在大多数情况下用户可能还希望附加一个UNIQUE或PRIMARY KEY约束避免意外地插入重复的数值，但这个不是自动的。最后，将序列发生器将从属于那个字段，这样当该字段或表被删除的时候也一并删除它。目前只支持在创建表时候指定SERIAL列，不可以在已有的表中，增加SERIAL列。另外临时表也不支持创建SERIAL列。因为SERIAL不是真正的类型，所以也不可以将表中存在的列类型转化为SERIAL。

表 7-5 浮点类型

名称	描述	存储空间	范围
REAL, FLOAT4	单精度浮点数，不精准。	4字节。	-3.402E+38~+3.402E+38，6位十进制数字精度。
DOUBLE PRECISION, FLOAT8	双精度浮点数，不精准。	8字节。	-1.79E+308~+1.79E+308，15位十进制数字精度。
FLOAT[(p)]	浮点数，不精准。精度p取值范围为[1,53]。 **说明** p为精度，表示二进制总位数。	4字节或8字节。	根据精度p不同选择REAL或DOUBLE PRECISION作为内部表示。如不指定精度，内部用DOUBLE PRECISION表示。
BINARY_DOUBLE	是DOUBLE PRECISION的别名。	8字节。	-1.79E+308~+1.79E+308，15位十进制数字精度。

名称	描述	存储空间	范围
DEC[(p[,s])]	精度p取值范围为[1,1000]，标度s取值范围为[0,p]。 在满足说明中的场景且未指定精度和标度的情况下，默认精度p为10，标度s为0。 该类型映射为NUMERIC，使用场景参考NUMERIC。	用户声明精度。每四位（十进制位）占用两个字节，然后在整个数据上加上八个字节的额外开销。	在精度和标度指定最大的情况下，小数点前最大131,072位，小数点后最大16,383位。
INTEGER[(p[,s])]	精度p取值范围为[1,1000]，标度s取值范围为[0,p]。 在未指定精度和标度的情况下，默认精度p为10，标度s为0。 未指定精度和标度的情况下，该类型映射为INTEGER。指定精度和标度的情况下，该类型映射为NUMERIC。	用户声明精度。每四位（十进制位）占用两个字节，然后在整个数据上加上八个字节的额外开销。	在精度和标度指定最大的情况下，小数点前最大131,072位，小数点后最大16,383位。 未指定精度和标度的情况下，范围是-2,147,483,648 ~ +2,147,483,647。

说明

- 二进制浮点数据类型REAL、FLOAT4、DOUBLE、DOUBLE PRECISION、FLOAT8、FLOAT[(p)]和BINARY_DOUBLE为不精确的数值类型，其内部存储为近似值，因此存储和检索时可能会显示轻微的差异。当用户在使用二进制浮点数据类型时需要注意以下几点：
  - 精确存储和计算：如果需要精确存储和计算（例如货币金额），请改用精确的数据类型（例如numeric）。
  - 复杂计算：若使用不精确的数据类型执行复杂计算以获得重要数据，需要仔细评估其结果。
  - 浮点数比较：比较两个浮点数是否相等的结果可能与预期存在差异。
  - 下溢错误：如果一个浮点数太过接近零，反而无法准确表示，会导致下溢错误。
- 关于浮点类型的精度，目前只能保证直接读取时的精度位数。涉及分布式计算时，由于计算执行在各个DN节点上，并且最终汇聚到一个CN节点，因此误差可能会随计算节点数量增加而被放大。

示例：

```
--创建表。
openGauss=# CREATE TABLE float_type_t2
(
    FT_COL1 INTEGER,
    FT_COL2 FLOAT4,
    FT_COL3 FLOAT8,
```



```
    FT_COL4 FLOAT(3),
    FT_COL5 BINARY_DOUBLE,
    FT_COL6 DECIMAL(10,4),
    FT_COL7 INTEGER(6,3)
)DISTRIBUTE BY HASH ( ft_col1);

--插入数据。
openGauss=# INSERT INTO float_type_t2 VALUES(10,10.365456,123456.1234,10.3214, 321.321, 123.123654,
123.123654);

--查看数据。
openGauss=# SELECT * FROM float_type_t2 ;
 ft_col1 | ft_col2 |    ft_col3    | ft_col4 | ft_col5 | ft_col6  | ft_col7
---------+---------+---------------+---------+---------+----------+---------
      10 | 10.3655 | 123456.1234 | 10.3214 | 321.321 | 123.1237 | 123.124
(1 row)

--删除表。
openGauss=# DROP TABLE float_type_t2;
```

7.3.2 货币类型

货币类型存储带有固定小数精度的货币金额。

表7-6中显示的范围假设有两位小数。可以以任意格式输入，包括整型、浮点型或者典型的货币格式（如“$1,000.00”）。根据区域字符集，输出一般是最后一种形式。

表 7-6 货币类型

名称	描述	存储容量	范围
money	货币金额	8 字节	-92233720368547758.08 到 +92233720368547758.07

numeric，int和bigint类型的值可以转换为money类型。如果从real和double precision类型转换到money类型，可以先转换为numeric类型，再转换为money类型，例如：

```
openGauss=# SELECT '12.34'::float8::numeric::money;
 money
--------
 $12.34
(1 row)
```

这种用法是不推荐使用的。浮点数不应该用来处理货币类型，因为小数点的位数可能会导致错误。

money类型的值可以转换为numeric类型而不丢失精度。转换为其他类型可能丢失精度，并且必须通过以下两步来完成：

```
openGauss=# SELECT '52093.89'::money::numeric::float8;
  float8
-----------
 52093.89
(1 row)
```

当一个money类型的值除以另一个money类型的值时，结果是double precision（也就是，一个纯数字，而不是money类型）；在运算过程中货币单位相互抵消。

7.3.3 布尔类型

表 7-7 布尔类型

名称	描述	存储空间	取值
BOOLEAN	布尔类型	1字节。	• true：真 • false：假 • null：未知（unknown）

“真”值的有效文本值是：

TRUE、't'、'true'、'y'、'yes'、'1' 、'TRUE'、true、on、以及所有非0整数。

“假”值的有效文本值是：

FALSE、'f'、'false'、'n'、'no'、'0'、0、'FALSE'、false、off。

使用TRUE和FALSE是比较规范的用法（也是SQL兼容的用法）。

示例

显示用字母t和f输出Boolean值。

```
--创建表。
openGauss=# CREATE TABLE bool_type_t1
(
    BT_COL1 BOOLEAN,
    BT_COL2 TEXT
)DISTRIBUTE BY HASH(BT_COL2);

--插入数据。
openGauss=# INSERT INTO bool_type_t1 VALUES (TRUE, 'sic est');

openGauss=# INSERT INTO bool_type_t1 VALUES (FALSE, 'non est');

--查看数据。
openGauss=# SELECT * FROM bool_type_t1;
 bt_col1 | bt_col2
---------+---------
 t       | sic est
 f       | non est
(2 rows)

openGauss=# SELECT * FROM bool_type_t1 WHERE bt_col1 = 't';
 bt_col1 | bt_col2
---------+---------
 t       | sic est
(1 row)

--删除表。
openGauss=# DROP TABLE bool_type_t1;
```

7.3.4 字符类型

GaussDB支持的字符类型请参见**表7-8**。字符串操作符和相关的内置函数请参见**字符处理函数和操作符**。

表 7-8 字符类型

名称	描述	存储空间
CHAR(n) CHARACTER(n) NCHAR(n)	定长字符串，不足补空格。n是指字节长度，如不带精度n，默认精度为1。	n最大为10485760（即10MB）。
VARCHAR(n) CHARACTER VARYING(n)	变长字符串。PG兼容模式下，n是字符长度。其他兼容模式下，n是指字节长度。	n最大为10485760（即10MB）。 不带n最大存储长度为1GB-85-前n列长度，比如(a int, b varchar)最大长度为1GB-85-4=1,073,741,735。
VARCHAR2(n)	变长字符串。是VARCHAR(n)类型的别名，为兼容Oracle类型。n是指字节长度。	n最大为10485760（即10MB）。 不带n最大存储长度为1GB-85-前n列长度，比如(a int, b varchar)最大长度为1GB-85-4=1,073,741,735。
NVARCHAR2(n)	变长字符串。n是指字符长度。	n最大为10485760（即10MB）。 不带n最大存储长度为1GB-85-前n列长度，比如(a int, b varchar)最大长度为1GB-85-4=1,073,741,735。
CLOB	文本大对象。兼容Oracle类型。	最大为32TB-1，但还需要考虑到列描述头信息的大小，以及列所在元组的大小限制（也小于32TB-1），因此CLOB类型最大可能小于32TB-1。
TEXT	变长字符串。	最大存储长度为1GB-85-前n列长度，比如(a int, b varchar)最大长度为1GB-85-4=1,073,741,735。

说明

1. 除了每列的大小限制以外，每个元组的总大小也不可超过1GB-1字节，主要受列的控制头信息、元组控制头信息以及元组中是否存在NULL字段等影响。
2. NCHAR为bpchar类型的别名，VARCHAR2(n)为VARCHAR(n)类型的别名。
3. 超过1GB的clob只有dbe_lob相关高级包支持，系统函数不支持大于1GB clob。

在GaussDB里另外还有两种定长字符类型。在表7-9里显示。name类型只用在内部系统表中，作为存储标识符，不建议普通用户使用。该类型长度当前定为64字节（63可用字符加结束符）。类型“char”只用了一个字节的存储空间，在系统内部主要用于系统表，主要作为简单化的枚举类型使用。

表 7-9 特殊字符类型

名称	描述	存储空间
name	用于对象名的内部类型。	64字节。
"char"	单字节内部类型。	1字节。

示例

```
--创建表。
openGauss=# CREATE TABLE char_type_t1
(
    CT_COL1 CHARACTER(4)
)DISTRIBUTE BY HASH (CT_COL1);

--插入数据。
openGauss=# INSERT INTO char_type_t1 VALUES ('ok');

--查询表中的数据。
openGauss=# SELECT ct_col1, char_length(ct_col1) FROM char_type_t1;
 ct_col1 | char_length
---------+-------------
 ok      |           4
(1 row)

--删除表。
openGauss=# DROP TABLE char_type_t1;
--创建表。
openGauss=# CREATE TABLE char_type_t2
(
    CT_COL1 VARCHAR(5)
)DISTRIBUTE BY HASH (CT_COL1);

--插入数据。
openGauss=# INSERT INTO char_type_t2 VALUES ('ok');

openGauss=# INSERT INTO char_type_t2 VALUES ('good');

--插入的数据长度超过类型规定的长度报错。
openGauss=# INSERT INTO char_type_t2 VALUES ('too long');
ERROR:  value too long for type character varying(5)
CONTEXT:  referenced column: ct_col1

--明确类型的长度，超过数据类型长度后会自动截断。
openGauss=# INSERT INTO char_type_t2 VALUES ('too long'::varchar(5));

--查询数据。
openGauss=# SELECT ct_col1, char_length(ct_col1) FROM char_type_t2;
 ct_col1 | char_length
---------+-------------
 ok      |           2
 good    |           4
 too l   |           5
(3 rows)

--删除数据。
openGauss=# DROP TABLE char_type_t2;
```

7.3.5 二进制类型

GaussDB支持的二进制类型请参见**表7-10**。

表 7-10 二进制类型

名称	描述	存储空间
BLOB	二进制大对象 目前BLOB支持的外部存取接口仅为： • DBE_LOB.GET_LENGTH • DBE_LOB.READ • DBE_LOB.WRITE • DBE_LOB.WRITE_APPEND • DBE_LOB.COPY • DBE_LOB.ERASE 这些接口详细说明请参见**DBE_LOB**。	最大为32TB（即35184372088832字节）。
RAW	变长的十六进制类型。	4字节加上实际的十六进制字符串。最大为1GB-8203字节（即1073733621字节）。
BYTEA	变长的二进制字符串。	4字节加上实际的二进制字符串。最大为1GB-8203字节（即1073733621字节）。

说明

除了每列的大小限制以外，每个元组的总大小也不可超过1GB-8203字节（即1073733621字节）。

示例:

```
--创建表。
openGauss=# CREATE TABLE blob_type_t1
(
    BT_COL1 INTEGER,
    BT_COL2 BLOB,
    BT_COL3 RAW,
    BT_COL4 BYTEA
) DISTRIBUTE BY REPLICATION;

--插入数据。
openGauss=# INSERT INTO blob_type_t1 VALUES(10,empty_blob(),
HEXTORAW('DEADBEEF'),E'\\xDEADBEEF');

--查询表中的数据。
openGauss=# SELECT * FROM blob_type_t1;
 bt_col1 | bt_col2 | bt_col3  |  bt_col4
---------+---------+----------+------------
      10 |         | DEADBEEF | \xdeadbeef
(1 row)

--删除表。
openGauss=# DROP TABLE blob_type_t1;
```


7.3.6 日期/时间类型

GaussDB支持的日期/时间类型请参见**表7-11**。该类型的操作符和内置函数请参见**时间和日期处理函数和操作符**。

说明

如果其他的数据库时间格式和GaussDB的时间格式不一致，可通过修改配置参数DateStyle的值来保持一致。

表 7-11 日期/时间类型

名称	描述	存储空间
DATE	日期。 最小值：公元前4713年，4713-01-01BC。最大值：公元5874897年，5874897-12-31AD。 **说明** ORA兼容性下，数据库将空字符串作为NULL处理，数据类型DATE会被替换为TIMESTAMP(0) WITHOUT TIME ZONE。	4字节（兼容模式ORA下存储空间大小为8字节）
TIME [(p)] [WITHOUT TIME ZONE]	只用于一日内时间，不带时区。 p表示小数点后的精度，取值范围为0~6。 最小值：00:00:00。最大值：24:00:00。	8字节
TIME [(p)] [WITH TIME ZONE]	只用于一日内时间，带时区。 p表示小数点后的精度，取值范围为0~6。 最小值：00:00:00+1559。最大值：24:00:00。	12字节
TIMESTAMP[(p)] [WITHOUT TIME ZONE]	日期和时间，不带时区。 p表示小数点后的精度，取值范围为0~6。 最小值：公元前4713年，4713-11-24BC 00:00:00.000000。最大值：公元294277年，294277-01-09AD 00:00:00.000000。	8字节

名称	描述	存储空间
TIMESTAMP[(p)] [WITH TIME ZONE]	日期和时间，带时区。TIMESTAMP的别名为TIMESTAMPTZ。 p表示小数点后的精度，取值范围为0~6。 最小值：公元前4713年，4713-11-24BC 00:00:00.000000。最大值：公元294277年，294277-01-09AD 00:00:00.000000。 时区更新：部分国家或地区因为政治、经济、战争等因素经常会更新时区信息， 数据库系统也因此常常需要同步修改时区文件以确保时间内容的正确性。 GaussDB时区类型目前只涉及timestamp with timezone，当新的时区文件生效时，不会对已有的老数据进行变更，新数据会随时区文件信息进行同步调整。在这点上和ORA数据库的同类型数据能力有差异。	8字节
SMALLDATETIME	日期和时间，不带时区。 精确到分钟，秒位大于等于30秒进一位。 最小值：公元前4713年，4713-11-24BC 00:00:00.000000。最大值：公元294277年，294277-01-09AD 00:00:00.000000。	8字节
INTERVAL DAY (l) TO SECOND (p)	时间间隔，X天X小时X分X秒。 • l：天数的精度，取值范围为0~6。因兼容性原因，未实现具体功能。 • p：秒数的精度，取值范围为0~6。小数末尾的零不显示。	16字节

名称	描述	存储空间
INTERVAL [FIELDS] [(p)]	时间间隔。 • fields：可以是YEAR，MONTH，DAY，HOUR，MINUTE，SECOND，DAY TO HOUR，DAY TO MINUTE，DAY TO SECOND，HOUR TO MINUTE，HOUR TO SECOND，MINUTE TO SECOND。 • p：秒数的精度，取值范围为0~6，且fields为SECOND，DAY TO SECOND，HOUR TO SECOND或MINUTE TO SECOND时，参数p才有效。小数末尾的零不显示。	12字节
reltime	相对时间间隔。 • 格式为：X years X mons X days XX:XX:XX。 • 采用儒略历计时，规定一年为365.25天，一个月为30天，计算输入值对应的相对时间间隔。	4字节
abstime	日期和时间。 • 格式为：YYYY-MM-DD hh:mm:ss +timezone • 取值范围为1901-12-13 20:45:53 GMT~2038-01-18 23:59:59 GMT，精度为秒。	4字节

说明

1. 时间类型的数据在显示的时候会自动忽略末尾的所有零。
2. 精度p默认取值为6。
3. 对于INTERVAL类型，日期和时间在系统内部分别用int32和double类型存储，所以两者的取值范围和对应数据类型的取值范围一致。
4. 插入时间超出范围的时候，系统可能不报错，但不保证行为正常。

示例：

```
--创建表。
openGauss=# CREATE TABLE date_type_tab(coll date);

--插入数据。
openGauss=# INSERT INTO date_type_tab VALUES (date '12-10-2010');

--查看数据。
openGauss=# SELECT * FROM date_type_tab;
    coll
------------
 2010-12-10
(1 row)
```



```
--删除表。
openGauss=# DROP TABLE date_type_tab;

--创建表。
openGauss=# CREATE TABLE time_type_tab (da time without time zone ,dai time with time zone,dfgh
timestamp without time zone,dfga timestamp with time zone, vbg smalldatetime);

--插入数据。
openGauss=# INSERT INTO time_type_tab VALUES ('21:21:21','21:21:21 pst','2010-12-12','2013-12-11
pst','2003-04-12 04:05:06');

--查看数据。
openGauss=# SELECT * FROM time_type_tab;
    da    |     dai     |        dfgh         |          dfga          |         vbg
----------+-------------+---------------------+------------------------+---------------------
 21:21:21 | 21:21:21-08 | 2010-12-12 00:00:00 | 2013-12-11 16:00:00+08 | 2003-04-12 04:05:00
(1 row)

--删除表。
openGauss=# DROP TABLE time_type_tab;

--创建表。
openGauss=# CREATE TABLE day_type_tab (a int,b INTERVAL DAY(3) TO SECOND (4));

--插入数据。
openGauss=# INSERT INTO day_type_tab VALUES (1, INTERVAL '3' DAY);

--查看数据。
openGauss=# SELECT * FROM day_type_tab;
 a |   b
---+--------
 1 | 3 days
(1 row)

--删除表。
openGauss=# DROP TABLE day_type_tab;

--创建表。
openGauss=# CREATE TABLE year_type_tab(a int, b interval year (6));

--插入数据。
openGauss=# INSERT INTO year_type_tab VALUES(1,interval '2' year);

--查看数据。
openGauss=# SELECT * FROM year_type_tab;
 a |    b
---+---------
 1 | 2 years
(1 row)

--删除表。
openGauss=# DROP TABLE year_type_tab;
```

日期输入

日期和时间的输入几乎可以是任何合理的格式，包括ISO-8601格式、SQL-兼容格式、传统POSTGRES格式或者其它的格式。系统支持按照日、月、年的顺序自定义日期输入。如果把DateStyle参数设置为MDY就按照“月-日-年”解析，设置为DMY就按照“日-月-年”解析，设置为YMD就按照“年-月-日”解析。

日期的文本输入需要加单引号包围，语法如下：

type [(p)] 'value'

可选的精度声明中的p是一个整数，表示在秒域中小数部分的位数。**表7-12**显示了date类型的输入格式。

表 7-12 日期输入方式

例子	描述
1999-01-08	ISO 8601格式（建议格式），任何方式下都是1999年1月8日。
January 8, 1999	在任何datestyle输入模式下都无歧义。
1/8/1999	有歧义，在MDY模式下是一月八日，在DMY模式下是八月一日。
1/18/1999	MDY模式下是一月十八日，其它模式下被拒绝。
01/02/03	• MDY模式下的2003年1月2日。 • DMY模式下的2003年2月1日。 • YMD模式下的2001年2月3日。
1999-Jan-08	任何模式下都是1月8日。
Jan-08-1999	任何模式下都是1月8日。
08-Jan-1999	任何模式下都是1月8日。
99-Jan-08	YMD模式下是1月8日，否则错误。
08-Jan-99	一月八日，除了在YMD模式下是错误的之外。
Jan-08-99	一月八日，除了在YMD模式下是错误的之外。
19990108	ISO 8601格式；任何模式下都是1999年1月8日。
990108	ISO 8601格式；任何模式下都是1999年1月8日。
1999.008	年和年里的第几天。
J2451187	儒略日。
January 8, 99 BC	公元前99年。

示例：

```
--创建表。
openGauss=# CREATE TABLE date_type_tab(coll date);

--插入数据。
openGauss=# INSERT INTO date_type_tab VALUES (date '12-10-2010');

--查看数据。
openGauss=# SELECT * FROM date_type_tab;
        coll
---------------------
 2010-12-10 00:00:00
(1 row)

--查看日期格式。
openGauss=# SHOW datestyle;
 DateStyle
-----------
 ISO, MDY
(1 row)
```



```
--设置日期格式。
openGauss=# SET datestyle='YMD';
SET

--插入数据。
openGauss=# INSERT INTO date_type_tab VALUES(date '2010-12-11');

--查看数据。
openGauss=# SELECT * FROM date_type_tab;
        coll
---------------------
 2010-12-10 00:00:00
 2010-12-11 00:00:00
(2 rows)

--删除表。
openGauss=# DROP TABLE date_type_tab;
```

时间

时间类型包括time [(p)] without time zone和time [(p)] with time zone。如果只写time等效于time without time zone。

如果在time without time zone类型的输入中声明了时区，则会忽略这个时区。

时间输入类型的详细信息请参见表7-13，时区输入类型的详细信息请参见表7-14。

表 7-13 时间输入

例子	描述
05:06.8	ISO 8601
4:05:06	ISO 8601
4:05	ISO 8601
040506	ISO 8601
4:05 AM	与04:05一样，输入小时数必须<= 12
4:05 PM	与16:05一样，输入小时数必须<= 12
04:05:06.789-8	ISO 8601
04:05:06-08:00	ISO 8601
04:05-08:00	ISO 8601
040506-08	ISO 8601
04:05:06 PST	缩写的时区
2003-04-12 04:05:06 America/New_York	用名称声明的时区

表 7-14 时区输入

例子	描述
PST	太平洋标准时间（Pacific Standard Time）
America/New_York	完整时区名称
-8:00	ISO 8601与PST的偏移
-800	ISO 8601与PST的偏移
-8	ISO 8601与PST的偏移

示例：

```
openGauss=# SELECT time '04:05:06';
   time
----------
 04:05:06
(1 row)

openGauss=# SELECT time '04:05:06 PST';
   time
----------
 04:05:06
(1 row)

openGauss=# SELECT time with time zone '04:05:06 PST';
   timetz
-------------
 04:05:06-08
(1 row)
```

特殊值

GaussDB支持几个特殊值，在读取的时候将被转换成普通的日期/时间值，请参考**表 7-15**。

表 7-15 特殊值

输入字符串	适用类型	描述
epoch	date，timestamp	1970-01-01 00:00:00+00 （UNIX系统零时）
infinity	timestamp	无穷大，比任何其他时间戳都晚。
-infinity	timestamp	无穷小，比任何其他时间戳都早。
now	date，time，timestamp	当前事务的开始时间。
today	date，timestamp	今日午夜零时。
tomorrow	date，timestamp	明日午夜零时。
yesterday	date，timestamp	昨日午夜零时。

输入字符串	适用类型	描述
allballs	time	00:00:00.00 UTC

示例：

```
--创建表。
openGauss=# CREATE TABLE realtime_type_special(col1 varchar(20), col2 date, col3 timestamp, col4 time);

--插入数据。
openGauss=# INSERT INTO realtime_type_special VALUES('epoch', 'epoch', 'epoch', NULL);
openGauss=# INSERT INTO realtime_type_special VALUES('now', 'now', 'now', 'now');
openGauss=# INSERT INTO realtime_type_special VALUES('today', 'today', 'today', NULL);
openGauss=# INSERT INTO realtime_type_special VALUES('tomorrow', 'tomorrow', 'tomorrow', NULL);
openGauss=# INSERT INTO realtime_type_special VALUES('yesterday', 'yesterday', 'yesterday', NULL);

--查看数据。
openGauss=# SELECT * FROM realtime_type_special;
   col1    |        col2         |            col3             |      col4
-----------+---------------------+-----------------------------+-----------------
 epoch     | 1970-01-01 00:00:00 | 1970-01-01 00:00:00         |
 now       | 2023-02-27 11:38:13 | 2023-02-27 11:38:13.032815 | 11:38:13.032815
 today     | 2023-02-27 00:00:00 | 2023-02-27 00:00:00         |
 tomorrow  | 2023-02-28 00:00:00 | 2023-02-28 00:00:00         |
 yesterday | 2023-02-26 00:00:00 | 2023-02-26 00:00:00         |
(5 rows)

openGauss=# SELECT * FROM realtime_type_special WHERE col3 < 'infinity';
   col1    |        col2         |            col3             |      col4
-----------+---------------------+-----------------------------+-----------------
 epoch     | 1970-01-01 00:00:00 | 1970-01-01 00:00:00         |
 now       | 2023-02-27 11:38:13 | 2023-02-27 11:38:13.032815 | 11:38:13.032815
 today     | 2023-02-27 00:00:00 | 2023-02-27 00:00:00         |
 tomorrow  | 2023-02-28 00:00:00 | 2023-02-28 00:00:00         |
 yesterday | 2023-02-26 00:00:00 | 2023-02-26 00:00:00         |
(5 rows)

openGauss=# SELECT * FROM realtime_type_special WHERE col3 > '-infinity';
   col1    |        col2         |            col3             |      col4
-----------+---------------------+-----------------------------+-----------------
 epoch     | 1970-01-01 00:00:00 | 1970-01-01 00:00:00         |
 now       | 2023-02-27 11:38:13 | 2023-02-27 11:38:13.032815 | 11:38:13.032815
 today     | 2023-02-27 00:00:00 | 2023-02-27 00:00:00         |
 tomorrow  | 2023-02-28 00:00:00 | 2023-02-28 00:00:00         |
 yesterday | 2023-02-26 00:00:00 | 2023-02-26 00:00:00         |
(5 rows)

openGauss=# SELECT * FROM realtime_type_special WHERE col3 > 'now';
   col1   |        col2         |        col3         | col4
----------+---------------------+---------------------+------
 tomorrow | 2023-02-28 00:00:00 | 2023-02-28 00:00:00 |
(1 row)

openGauss=# SELECT * FROM realtime_type_special WHERE col3 = 'today';
 col1  |        col2         |        col3         | col4
-------+---------------------+---------------------+------
 today | 2023-02-27 00:00:00 | 2023-02-27 00:00:00 |
(1 row)

openGauss=# SELECT * FROM realtime_type_special WHERE col3 = 'tomorrow';
   col1   |        col2         |        col3         | col4
----------+---------------------+---------------------+------
 tomorrow | 2023-02-28 00:00:00 | 2023-02-28 00:00:00 |
(1 row)

openGauss=# SELECT * FROM realtime_type_special WHERE col3 > 'yesterday';
```



```
  col1   |        col2         |            col3            |      col4
---------+---------------------+----------------------------+-----------------
 now      | 2023-02-27 11:38:13 | 2023-02-27 11:38:13.032815 | 11:38:13.032815
 today    | 2023-02-27 00:00:00 | 2023-02-27 00:00:00        |
 tomorrow | 2023-02-28 00:00:00 | 2023-02-28 00:00:00        |
(3 rows)

--删除表。
openGauss=# DROP TABLE realtime_type_special;
```

时间段输入

reltime的输入方式可以采用任何合法的时间段文本格式，包括数字形式（含负数和小数）及时间形式，其中时间形式的输入支持SQL标准格式、ISO-8601格式、POSTGRES格式等。另外，文本输入需要加单引号。

时间段输入的详细信息请参考**表7-16**。

表 7-16 时间段输入

输入示例	输出结果	描述
60	2 mons	采用数字表示时间段，默认单位是day，可以是小数或负数。特别的，负数时间段，在语义上，可以理解为“早于多久”。
31.25	1 mons 1 days 06:00:00	
-365	-12 mons -5 days	
1 years 1 mons 8 days 12:00:00	1 years 1 mons 8 days 12:00:00	采用POSTGRES格式表示时间段，可以正负混用，不区分大小写，输出结果为将输入时间段计算并转换得到的简化POSTGRES格式时间段。
-13 months -10 hours	-1 years -25 days -04:00:00	
-2 YEARS +5 MONTHS 10 DAYS	-1 years -6 mons -25 days -06:00:00	
P-1.1Y10M	-3 mons -5 days -06:00:00	采用ISO-8601格式表示时间段，可以正负混用，不区分大小写，输出结果为将输入时间段计算并转换得到的简化POSTGRES格式时间段。
-12H	-12:00:00	

示例：

```
--创建表。
openGauss=# CREATE TABLE reltime_type_tab(col1 character(30), col2 reltime);

--插入数据。
openGauss=# INSERT INTO reltime_type_tab VALUES ('90', '90');
openGauss=# INSERT INTO reltime_type_tab VALUES ('-366', '-366');
openGauss=# INSERT INTO reltime_type_tab VALUES ('1975.25', '1975.25');
openGauss=# INSERT INTO reltime_type_tab VALUES ('-2 YEARS +5 MONTHS 10 DAYS', '-2 YEARS +5
MONTHS 10 DAYS');
openGauss=# INSERT INTO reltime_type_tab VALUES ('30 DAYS 12:00:00', '30 DAYS 12:00:00');
openGauss=# INSERT INTO reltime_type_tab VALUES ('P-1.1Y10M', 'P-1.1Y10M');

--查看数据。
openGauss=# SELECT * FROM reltime_type_tab;
```



```
             col1              |               col2
-------------------------------+-----------------------------------
 90                            | 3 mons
 -366                          | -1 years -18:00:00
 1975.25                       | 5 years 4 mons 29 days
 -2 YEARS +5 MONTHS 10 DAYS    | -1 years -6 mons -25 days -06:00:00
 30 DAYS 12:00:00              | 1 mon 12:00:00
 P-1.1Y10M                     | -3 mons -5 days -06:00:00
(6 rows)

--删除表。
openGauss=# DROP TABLE reltime_type_tab;
```

7.3.7 几何类型

GaussDB支持的几何类型请参见**表7-17**。最基本的类型：点，是其它类型的基础。

表 7-17 几何类型

名称	存储空间	说明	表现形式
point	16字节	平面中的点	(x,y)
lseg	32字节	（有限）线段	((x1,y1),(x2,y2))
box	32字节	矩形	((x1,y1),(x2,y2))
path	16+16*n字节	闭合路径（与多边形相似）	((x1,y1),...)
path	16+16*n字节	开放路径	[(x1,y1),...]
polygon	40+16*n字节	多边形（与闭合路径相似）	((x1,y1),...)
circle	24 字节	圆	<(x,y),r>（圆心和半径）

GaussDB提供了一系列的函数和操作符用来进行各种几何计算，如拉伸、转换、旋转、计算相交等。详细信息请参考**几何函数和操作符**。

点

点是几何类型的基本二维构造单位。用下面语法描述point的数值：

```
( x , y )
x , y
```

x和y是用浮点数表示的点的坐标，点的数值类型为float8类型。

点输出使用第一种语法。

示例：

```
openGauss=# select point(1.1, 2.2);
   point
-----------
 (1.1,2.2)
(1 row)
```


线段

线段（lseg）是用一对点来代表的。用下面的语法描述lseg的数值：

```
[ ( x1 , y1 ) , ( x2 , y2 ) ]
( ( x1 , y1 ) , ( x2 , y2 ) )
( x1 , y1 ) , ( x2 , y2 )
x1 , y1   ,   x2 , y2
```

(x1,y1)和(x2,y2)表示线段的端点，点的数值类型为float8类型。

线段输出使用第一种语法。

示例：

```
openGauss=# select lseg(point(1.1, 2.2), point(3.3, 4.4));
        lseg
-----------------------
 [(1.1,2.2),(3.3,4.4)]
(1 row)
```

矩形

矩形是用一对对角点来表示的。用下面的语法描述box的值：

```
( ( x1 , y1 ) , ( x2 , y2 ) )
( x1 , y1 ) , ( x2 , y2 )
x1 , y1   ,   x2 , y2
```

(x1,y1)和(x2,y2)表示矩形的一对对角点，点的数值类型为float8类型。

矩形的输出使用第二种语法。

任何两个对角都可以出现在输入中，但按照那样的顺序，右上角和左下角的值会被重新排序以存储。

示例：

```
openGauss=# select box(point(1.1, 2.2), point(3.3, 4.4));
        box
----------------------
 (3.3,4.4),(1.1,2.2)
(1 row)
```

路径

路径由一系列连接的点组成。路径可能是开放的，也就是认为列表中第一个点和最后一个点没有连接，也可能是闭合的，这时认为第一个和最后一个点连接起来。

用下面的语法描述path的数值：

```
[ ( x1 , y1 ) , ... , ( xn , yn ) ]
( ( x1 , y1 ) , ... , ( xn , yn ) )
( x1 , y1 ) , ... , ( xn , yn )
( x1 , y1   , ... ,   xn , yn )
x1 , y1   , ... ,   xn , yn
```

点表示组成路径的线段的端点，点的数值类型为float8类型。方括号（[]）表明一个开放的路径，圆括号（()）表明一个闭合的路径。当最外层的括号被省略，如在第三至第五语法，会假定一个封闭的路径。

路径的输出使用第一种或第二种语法输出。

示例：


```
openGauss=# select path(polygon '((0,0),(1,1),(2,0))');
       path
--------------------
 ((0,0),(1,1),(2,0))
(1 row)
```

多边形

多边形由一系列点代表（多边形的顶点）。多边形可以认为与闭合路径一样，但是存储方式不一样而且有自己的一套支持函数。

用下面的语法描述polygon的数值：

```
( ( x1 , y1 ) , ... , ( xn , yn ) )
( x1 , y1 ) , ... , ( xn , yn )
( x1 , y1   , ... ,   xn , yn )
x1 , y1   , ... ,   xn , yn
```

点表示多边形的顶点，点的数值类型为float8类型。

多边形输出使用第一种语法。

示例：

```
openGauss=# select polygon(box '((0,0),(1,1))');
         polygon
---------------------------
 ((0,0),(0,1),(1,1),(1,0))
(1 row)
```

圆

圆由一个圆心和半径标识。用下面的格式描述circle的数值：

```
< ( x , y ) , r >
( ( x , y ) , r )
( x , y ) , r
x , y   , r
```

(x,y)表示圆心，r表示半径，点的数值类型为float8类型。

圆的输出用第一种格式。

示例：

```
openGauss=# select circle(point(0,0),1);
  circle
----------
 <(0,0),1>
(1 row)
```

7.3.8 网络地址类型

GaussDB提供用于存储IPv4、MAC地址的数据类型。

用这些数据类型存储网络地址比用纯文本类型好，因为这些类型提供输入错误检查和特殊的操作和功能（请参见**网络地址函数和操作符**）。

表 7-18 网络地址类型

名称	存储空间	描述
cidr	7字节	IPv4网络

名称	存储空间	描述
inet	7字节	IPv4主机和网络
macaddr	6字节	MAC地址

cidr

cidr（无类别域间路由，Classless Inter-Domain Routing）类型，保存一个IPv4网络地址。声明网络格式为address/y，address表示IPv4地址，y表示子网掩码的二进制位数。如果省略y，则掩码部分使用已有类别的网络编号系统进行计算，但要求输入的数据已经包括了确定掩码所需的所有字节。

表 7-19 cidr 类型输入举例

cidr输入	cidr输出	abbrev（cidr）
192.168.100.128/25	192.168.100.128/25	192.168.100.128/25
192.168/24	192.168.0.0/24	192.168.0/24
192.168/25	192.168.0.0/25	192.168.0.0/25
192.168.1	192.168.1.0/24	192.168.1/24
192.168	192.168.0.0/24	192.168.0/24
10.1.2	10.1.2.0/24	10.1.2/24
10.1	10.1.0.0/16	10.1/16
10	10.0.0.0/8	10/8
10.1.2.3/32	10.1.2.3/32	10.1.2.3/32

示例：

```
openGauss=# CREATE TABLE cidr_test(id int, c cidr);
CREATE TABLE
openGauss=# INSERT INTO cidr_test VALUES (1, '192.168.100.128/25');
INSERT 0 1
openGauss=# INSERT INTO cidr_test VALUES (2, '192.168/24');
INSERT 0 1
openGauss=# INSERT INTO cidr_test VALUES (3, '192.168/25');
INSERT 0 1
openGauss=# INSERT INTO cidr_test VALUES (4, '192.168.1');
INSERT 0 1
openGauss=# INSERT INTO cidr_test VALUES (5, '192.168');
INSERT 0 1
openGauss=# INSERT INTO cidr_test VALUES (6, '10.1.2');
INSERT 0 1
openGauss=# INSERT INTO cidr_test VALUES (7, '10.1');
INSERT 0 1
openGauss=# INSERT INTO cidr_test VALUES (8, '10');
INSERT 0 1
openGauss=# INSERT INTO cidr_test VALUES (9, '2001:4f8:3:ba::/64');
INSERT 0 1
openGauss=# INSERT INTO cidr_test VALUES (10, '2001:4f8:3:ba:2e0:81ff:fe22:d1f1/128');
```



```
INSERT 0 1
openGauss=# INSERT INTO cidr_test VALUES (11, '::ffff:127.0.0.0/120');
INSERT 0 1
openGauss=# INSERT INTO cidr_test VALUES (12, '::ffff:127.0.0.0/128');
INSERT 0 1
openGauss=# SELECT * FROM cidr_test ORDER BY id;
 id |                  c
----+---------------------------------------
  1 | 192.168.100.128/25
  2 | 192.168.0.0/24
  3 | 192.168.0.0/25
  4 | 192.168.1.0/24
  5 | 192.168.0.0/24
  6 | 10.1.2.0/24
  7 | 10.1.0.0/16
  8 | 10.0.0.0/8
  9 | 2001:4f8:3:ba::/64
 10 | 2001:4f8:3:ba:2e0:81ff:fe22:d1f1/128
 11 | ::ffff:127.0.0.0/120
 12 | ::ffff:127.0.0.0/128
(12 rows)

openGauss=# DROP TABLE cidr_test;
DROP TABLE
```

inet

inet类型在一个数据区域内保存主机的IPv4地址，以及一个可选子网。主机地址中网络地址的位数表示子网（“子网掩码”）。如果子网掩码是32并且地址是IPv4，则这个值不表示任何子网，只表示一台主机。

该类型的输入格式是address/y，address表示IPv4地址，y是子网掩码的二进制位数。如果省略/y，则子网掩码对IPv4是32，所以该值表示只有一台主机。如果该值表示只有一台主机，/y将不会显示。

inet和cidr类型之间的基本区别是inet接受子网掩码，而cidr不接受。

示例：

```
openGauss=# CREATE TABLE inet_test(id int, i inet);
CREATE TABLE
openGauss=# INSERT INTO inet_test VALUES (1, '192.168.100.128/25');
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (2, '192.168.100.128');
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (3, '192.168.1.0/24');
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (4, '192.168.1.0/25');
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (5, '192.168.1.255/24');
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (6, '192.168.1.255/25');
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (7, '10.1.2.3/8');
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (8, '11.1.2.3/16');
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (9, '12.1.2.3/24');
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (10, '13.1.2.3/32');
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (11, '2001:4f8:3:ba::/64');
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (12, '2001:4f8:3:ba:2e0:81ff:fe22:d1f1/128');
INSERT 0 1
openGauss=# INSERT INTO inet_test VALUES (13, '::ffff:127.0.0.0/120');
INSERT 0 1
```



```
openGauss=# INSERT INTO inet_test VALUES (14, '::ffff:127.0.0.0/128');
INSERT 0 1
openGauss=# SELECT * FROM inet_test ORDER BY id;
 id |                 i
----+-----------------------------------
  1 | 192.168.100.128/25
  2 | 192.168.100.128
  3 | 192.168.1.0/24
  4 | 192.168.1.0/25
  5 | 192.168.1.255/24
  6 | 192.168.1.255/25
  7 | 10.1.2.3/8
  8 | 11.1.2.3/16
  9 | 12.1.2.3/24
 10 | 13.1.2.3
 11 | 2001:4f8:3:ba::/64
 12 | 2001:4f8:3:ba:2e0:81ff:fe22:d1f1
 13 | ::ffff:127.0.0.0/120
 14 | ::ffff:127.0.0.0
(14 rows)

openGauss=# DROP TABLE inet_test;
DROP TABLE
```

macaddr

macaddr类型存储MAC地址，也就是以太网卡硬件地址（尽管MAC地址还用于其它用途）。可以接受下列格式：

```
'08:00:2b:01:02:03'
'08-00-2b-01-02-03'
'08002b:010203'
'08002b-010203'
'0800.2b01.0203'
'08002b010203'
```

这些示例都表示同一个地址。对于数据位a到f，大小写都行。输出时都是以第一种形式展示。

示例：

```
openGauss=# CREATE TABLE macaddr_test(id int, m macaddr);
CREATE TABLE
openGauss=# INSERT INTO macaddr_test VALUES (1, '08:00:2b:01:02:03');
INSERT 0 1
openGauss=# INSERT INTO macaddr_test VALUES (2, '08-00-2b-01-02-03');
INSERT 0 1
openGauss=# INSERT INTO macaddr_test VALUES (3, '08002b:010203');
INSERT 0 1
openGauss=# INSERT INTO macaddr_test VALUES (4, '08002b-010203');
INSERT 0 1
openGauss=# INSERT INTO macaddr_test VALUES (5, '0800.2b01.0203');
INSERT 0 1
openGauss=# INSERT INTO macaddr_test VALUES (6, '08002b010203');
INSERT 0 1
openGauss=# SELECT * FROM macaddr_test ORDER BY id;
 id |         m
----+-------------------
  1 | 08:00:2b:01:02:03
  2 | 08:00:2b:01:02:03
  3 | 08:00:2b:01:02:03
  4 | 08:00:2b:01:02:03
  5 | 08:00:2b:01:02:03
  6 | 08:00:2b:01:02:03
(6 rows)

openGauss=# DROP TABLE macaddr_test;
DROP TABLE
```


7.3.9 位串类型

位串就是一串1和0的字符串。它们可以用于存储位掩码。

GaussDB支持两种位串类型：bit(n)和bit varying(n)，这里的n是一个正整数，n最大取值为83886080，相当于10M的容量。

bit类型的数据必须准确匹配长度n，如果存储短或者长的数据都会报错。bit varying类型的数据是最长为n的变长类型，长度超过n时会被拒绝。一个没有长度的bit等效于bit(1)，没有长度的bit varying表示没有长度限制。

说明

- 如果用户明确地把一个位串值转换成bit(n)，则此位串右边的内容将被截断或者在右边补齐零，直到刚好n位，而不会抛出任何错误。
- 如果用户明确地把一个位串数值转换成bit varying(n)，如果它超过了n位，则它的右边将被截断。
- 使用ADMS平台8.1.3-200驱动版本及之前版本时，写入bit类型需要用::bit varying进行类型转换，否则可能出现异常报错。

```
--创建表。
openGauss=# CREATE TABLE bit_type_t1
(
    BT_COL1 INTEGER,
    BT_COL2 BIT(3),
    BT_COL3 BIT VARYING(5)
) DISTRIBUTE BY REPLICATION;

--插入数据。
openGauss=# INSERT INTO bit_type_t1 VALUES(1, B'101', B'00');

--插入数据的长度不符合类型的标准会报错。
openGauss=# INSERT INTO bit_type_t1 VALUES(2, B'10', B'101');
ERROR:  bit string length 2 does not match type bit(3)
CONTEXT:  referenced column: bt_col2

--将不符合类型长度的数据进行转换。
openGauss=# INSERT INTO bit_type_t1 VALUES(2, B'10'::bit(3), B'101');

--查看数据。
openGauss=# SELECT * FROM bit_type_t1;
 bt_col1 | bt_col2 | bt_col3
---------+---------+---------
       1 | 101     | 00
       2 | 100     | 101
(2 rows)

--删除表。
openGauss=# DROP TABLE bit_type_t1;
```

7.3.10 UUID 类型

UUID数据类型用来存储RFC 4122，ISO/IEF 9834-8:2005以及相关标准定义的通用唯一标识符（UUID）。这个标识符是一个由算法产生的128位标识符，确保它不可能使用相同算法在已知的模块中产生相同的标识符。

因此，对分布式系统而言，这种标识符比序列能更好的保证唯一性，因为序列只能在单一数据库中保证是唯一。

UUID是一个小写十六进制数字的序列，由连字符分成几组，一组8位数字+三组4位数字+一组12位数字，总共32个数字代表128位，标准的UUID示例如下：

```
a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
```


GaussDB同样支持以其他方式输入：大写字母和数字、由花括号包围的标准格式、省略部分或所有连字符、在任意一组四位数字之后加一个连字符。示例：

```
A0EEBC99-9C0B-4EF8-BB6D-6BB9BD380A11
{a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11}
a0eebc999c0b4ef8bb6d6bb9bd380a11
a0ee-bc99-9c0b-4ef8-bb6d-6bb9-bd38-0a11
```

一般是以标准格式输出。

示例

```
-- 创建表
openGauss=# CREATE TABLE uuid_test(id int, test uuid) DISTRIBUTE BY HASH(test);

-- 插入数据，使用示例格式插入数据
openGauss=# INSERT INTO uuid_test VALUES(1, 'A0EEBC99-9C0B-4EF8-BB6D-6BB9BD380A11');
openGauss=# INSERT INTO uuid_test VALUES(2, '{a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11}');
openGauss=# INSERT INTO uuid_test VALUES(3, 'a0eebc999c0b4ef8bb6d6bb9bd380a11');
openGauss=# INSERT INTO uuid_test VALUES(4, 'a0ee-bc99-9c0b-4ef8-bb6d-6bb9-bd38-0a11');

-- 查看数据，输出时以标准格式输出
openGauss=# SELECT * FROM uuid_test;
 id |                 test
----+--------------------------------------
  1 | a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
  2 | a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
  3 | a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
  4 | a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
(4 rows)
```

7.3.11 JSON/JSONB 类型

JSON(JavaScript Object Notation)数据，可以是单独的一个标量，也可以是一个数组，也可以是一个键值对象，其中数组和对象可以统称容器(container)：

- 标量(scalar)：单一的数字、bool、string、null都可以叫做标量。
- 数组(array)：[]结构，里面存放的元素可以是任意类型的JSON，并且不要求数组内所有元素都是同一类型。
- 对象(object)：{}结构，存储key:value的键值对，其键只能是用""包裹起来的字符串，值可以是任意类型的JSON，对于重复的键，按最后一个键值为准。

GaussDB内存在两种数据类型JSON和JSONB，可以用来存储JSON数据。其中JSON是对输入的字符串的完整拷贝，使用时再去解析，所以它会保留输入的空格、重复键以及顺序等；JSONB数据以解析的二进制格式存储，它在解析时会删除语义无关的细节和重复的键，对键值也会进行排序，使用时不用再次解析。

因此可以发现，两者其实都是JSON，它们接受相同的字符串作为输入。它们实际的主要差别是效率。JSON数据类型存储输入文本的精确拷贝，处理函数必须在每个执行上重新解析； 而JSONB数据以解析的二进制格式存储， 由于添加了解析机制，因此在输入上稍微慢些，但是在处理上明显更快， 因为不需要重新解析。同时由于JSONB类型存在解析后的格式归一化等操作，同等的语义下只会有一种格式，因此可以更好更强大的支持很多其他额外的操作，比如按照一定的规则进行大小比较等。JSONB也支持索引，这也是一个明显的优势。

输入格式

输入必须是一个符合JSON数据格式的字符串，此字符串用单引号''声明。

null (null-json)：仅null，全小写。


```
openGauss=# SELECT 'null'::json;   -- suc
 json
------
 null
(1 row)

openGauss=# SELECT 'NULL'::jsonb;  -- err
ERROR:  invalid input syntax for type json
```

数字 (num-json)：正负整数、小数、0，支持科学计数法。

```
openGauss=# SELECT '1'::json;
json
------
 1
(1 row)

openGauss=# SELECT '-1.5'::json;
 json
------
 -1.5
(1 row)

openGauss=# SELECT '-1.5e-5'::jsonb, '-1.5e+2'::jsonb;
   jsonb   | jsonb
-----------+-------
 -0.000015 | -150
(1 row)

openGauss=# SELECT '001'::json, '+15'::json, 'NaN'::json;  -- 不支持多余的前导0，正数的+号，以及NaN和
infinity。
ERROR:  invalid input syntax for type json
```

布尔(bool-json)：仅true、false，全小写。

```
openGauss=# SELECT 'true'::json;
 json
------
 true
(1 row)

openGauss=# SELECT 'false'::jsonb;
 jsonb
-------
 false
(1 row)
```

字符串(str-json)：必须是加双引号的字符串。

```
openGauss=# SELECT '"a"'::json;
 json
------
 "a"
(1 row)

openGauss=# SELECT '"abc"'::jsonb;
 jsonb
-------
 "abc"
(1 row)
```

数组(array-json)：使用中括号[]包裹，满足数组书写条件。数组内元素类型可以是任意合法的JSON，且不要求类型一致。

```
openGauss=# SELECT '[1, 2, "foo", null]'::json;
        json
--------------------
 [1, 2, "foo", null]
(1 row)
```



```
openGauss=# SELECT '[]'::json;
 json
------
 []
(1 row)

openGauss=# SELECT '[1, 2, "foo", null, [[]], {}]'::jsonb;
            jsonb
--------------------------------
 [1, 2, "foo", null, [[]], {}]
(1 row)
```

对象(object-json)：使用大括号{}包裹，键必须是满足JSON字符串规则的字符串，值可以是任意合法的JSON。

```
openGauss=# SELECT '{}'::json;
 json
------
 {}
(1 row)

openGauss=# SELECT '{"a": 1, "b": {"a": 2,  "b": null}}'::json;
                json
------------------------------------
 {"a": 1, "b": {"a": 2,  "b": null}}
(1 row)

openGauss=# SELECT '{"foo": [true, "bar"], "tags": {"a": 1, "b": null}}'::jsonb;
                        jsonb
------------------------------------------------------
 {"foo": [true, "bar"], "tags": {"a": 1, "b": null}}
(1 row)
```

⚠ 注意

- 区分 'null'::json 和 null::json 是两个不同的概念，类似于字符串 str="" 和 str=null。
- 对于数字，当使用科学计数法的时候，jsonb类型会将其展开，而json会精准拷贝输入。

JSONB 高级特性

- 注意事项
  - 不支持作为分区键。
  - 不支持外表、mot。

JSON和JSONB的主要差异在于存储方式上的不同，JSONB存储的是解析后的二进制，能够体现JSON的层次结构，更方便直接访问等，因此JSONB会有很多JSON所不具有的高级特性。

- 格式归一化
  - 对于输入的object-json字符串，解析成jsonb二进制后，会天然的丢弃语义上无关紧要的细节，比如空格：

    ```
    openGauss=# select '   [1, " a ", {"a"   :1    }]  '::jsonb;
           jsonb
    ----------------------
     [1, " a ", {"a": 1}]
    (1 row)
    ```

  - 对于object-json，会删除重复的键值，只保留最后一个出现的，如：


```
openGauss=# select '{"a" : 1, "a" : 2}'::jsonb;
  jsonb
----------
 {"a": 2}
(1 row)
```

  - 对于object-json，键值会重新进行排序，排序规则：长度长的在后、长度相等则ascii码大的在后，如：

```
openGauss=# select '{"aa" : 1, "b" : 2, "a" : 3}'::jsonb;
           jsonb
----------------------------
{"a": 3, "b": 2, "aa": 1}
(1 row)
```

- 大小比较

  由于经过了格式归一化，保证了同一种语义下的jsonb只会有一种存在形式，因此按照制定的规则，可以比较大小。

  - 首先比较类型：object-jsonb > array-jsonb > bool-jsonb > num-jsonb > str-jsonb > null-jsonb
  - 同类型则比较内容：
    - str-jsonb类型：依据text比较的方法，使用数据库默认排序规则进行比较，返回值正数代表大于，负数代表小于，0表示相等。
    - num-jsonb类型：数值比较。
    - bool-jsonb类型：true > false。
    - array-jsonb类型：长度长的 > 长度短的，长度相等则依次比较每个元素。
    - object-jsonb类型：长度长的 > 长度短的，长度相等则依次比较每个键值对，先比较键，然后比较值。

  ⚠ 注意

  object-jsonb类型内比较时，使用的是格式整理后的最终结果进行比较，因此相对于直接的输入未必会很直观。

- 创建索引、主键
  - BTREE索引

    jsonb类型支持创建btree索引，支持创建主键。
- 包含存在

  查询一个JSON之中是否包含某些元素，或者某些元素是否存在于某个JSON中是jsonb的一个重要能力。

  相关的操作符请参见**JSON/JSONB函数和操作符**。
- 函数和操作符

  json/jsonb类型相关支持的函数和操作符请参见**JSON/JSONB函数和操作符**。

7.3.12 HLL 数据类型

HLL（HyperLoglog）是统计数据集中唯一值个数的高效近似算法。它有着计算速度快、节省空间的特点，不需要直接存储集合本身，而是存储一种名为HLL的数据结构。

每当有新数据加入进行统计时，只需要把数据经过哈希计算并插入到HLL中，最后根据HLL就可以得到结果。

HLL与其他算法的比较请参见表7-20。

表 7-20 HLL 与其他算法比较

项目	Sort算法	Hash算法	HLL
时间复杂度	O(nlogn)	O(n)	O(n)
空间复杂度	O(n)	O(n)	log(logn)
误差率	0	0	≈0.8%
所需存储空间	原始数据大小	原始数据大小	默认规格下最大16KB

HLL在计算速度和所占存储空间上都占优势。在时间复杂度上，Sort算法需要排序至少O(nlogn)的时间，虽说Hash算法和HLL一样扫描一次全表O(n)的时间就可以得出结果，但是存储空间上，Sort算法和Hash算法都需要先把原始数据存起来再进行统计，会导致存储空间消耗巨大，而对HLL来说不需要存原始数据，只需要维护HLL数据结构，故占用空间有很大的压缩，默认规格下HLL数据结构的最大空间约为16KB。

须知

- 当前默认规格下可计算最大distinct值的数量约为1.1e+15个，误差率为0.8%。用户应注意如果计算结果超过当前规格下distinct最大值会导致计算结果误差率变大，或导致计算结果失败并报错。
- 用户在首次使用该特性时，应该对业务的distinct value做评估，选取适当的配置参数并做验证，以确保精度符合要求：
  - 当前默认参数下，可以计算的distinct值为1.1e+15，如果计算得到的distinct值为NaN，需要调整log2m，或者采用其他算法计算distinct值。
  - 虽然hash算法存在极低的hash collision概率，但是建议用户在首次使用时，选取2-3个hash seed验证，如果得到的distinct value相差不大，则可以从该组seed中任选一个作为hash seed。

HLL中主要的数据结构，请参见表7-21。

表 7-21 HyperLogLog 中主要数据结构

数据类型	功能描述
hll	hll头部为27字节长度字段，默认规格下数据段长度0~16KB，可直接计算得到distinct值。

创建HLL数据类型时，可以支持0~4个参数入参，具体的参数含义与参数规格同函数hll_empty一致。第一个参数为log2m，表示分桶数的对数值，取值范围10~16；第二

个参数为log2explicit，表示Explicit模式的阈值大小，取值范围0~12；第三个参数为log2sparse，表示Sparse模式的阈值大小，取值范围0~14；第四个参数为duplicatecheck，表示是否启用duplicatecheck，取值范围为0~1。当入参输入值为-1时，会采用默认值设定HLL的参数。可以通过\d或\d+查看HLL类型的参数。

说明

创建HLL数据类型时，根据入参的行为不同，结果不同：

- 创建HLL类型时对应入参不输入或输入-1，采用默认值设定对应的HLL参数。
- 输入合法范围的入参，对应HLL参数采用输入值。
- 输入不合法范围的入参，创建HLL类型报错。

```
-- 创建hll类型的表，不指定入参。
openGauss=# CREATE TABLE t1 (id integer, set hll);
openGauss=# \d t1
      Table "public.t1"
 Column |  Type   | Modifiers
--------+---------+-----------
 id     | integer |
 set    | hll     |

-- 创建hll类型的表，指定前两个入参，后两个采用默认值。
openGauss=# CREATE TABLE t2 (id integer, set hll(12,4));
openGauss=# \d t2
          Table "public.t2"
 Column |      Type      | Modifiers
--------+----------------+-----------
 id     | integer        |
 set    | hll(12,4,12,0) |

--创建hll类型的表，指定第三个入参，其余采用默认值。
openGauss=# CREATE TABLE t3(id int, set hll(-1,-1,8,-1));
openGauss=# \d t3
          Table "public.t3"
 Column |      Type      | Modifiers
--------+----------------+-----------
 id     | integer        |
 set    | hll(14,10,8,0) |

--创建hll类型的表，指定入参不合法报错。
openGauss=# CREATE TABLE t4(id int, set hll(5,-1));
ERROR:  log2m = 5 is out of range, it should be in range 10 to 16, or set -1 as default

--删除已创建的hll类型的表。
openGauss=# DROP TABLE t1,t2,t3;
DROP TABLE
```

说明

对含有HLL类型的表插入HLL对象时，HLL类型的设定参数须同插入对象的设定参数一致，否则报错。

```
-- 创建带有hll类型的表。
openGauss=# CREATE TABLE t1(id integer, set hll(14));

-- 向表中插入hll对象,参数一致，成功。
openGauss=# INSERT INTO t1 VALUES (1, hll_empty(14,-1));

-- 向表中插入hll对象，参数不一致，失败。
openGauss=# INSERT INTO  t1(id, set) VALUES (1, hll_empty(14,5));
ERROR:  log2explicit does not match: source is 5 and dest is 10

-- 删除表。
openGauss=# DROP TABLE t1;
```

HLL的应用场景。

- 场景1：“Hello World”

  通过下面的示例说明如何使用hll数据类型：

```
-- 创建带有hll类型的表。
openGauss=# CREATE TABLE helloworld (id integer, set hll);

-- 向表中插入空的hll。
openGauss=# INSERT INTO  helloworld(id, set) VALUES (1, hll_empty());

-- 把整数经过哈希计算加入到hll中。
openGauss=# UPDATE helloworld SET set = hll_add(set, hll_hash_integer(12345)) WHERE id = 1;

-- 把字符串经过哈希计算加入到hll中。
openGauss=# UPDATE helloworld SET set = hll_add(set, hll_hash_text('hello world')) WHERE id = 1;

-- 得到hll中的distinct值。
openGauss=# SELECT hll_cardinality(set) FROM helloworld WHERE id = 1;
 hll_cardinality
-----------------
               2
(1 row)

-- 删除表。
openGauss=# DROP TABLE helloworld;
```

- 场景2：“网站访客数量统计”

  通过下面的示例说明hll如何统计在一段时间内访问网站的不同用户数量：

```
-- 创建原始数据表，表示某个用户在某个时间访问过网站。
openGauss=# CREATE TABLE facts (
        date            date,
        user_id         integer
);

-- 构造数据，表示一天中有哪些用户访问过网站。
openGauss=# INSERT INTO  facts VALUES ('2019-02-20', generate_series(1,100));
openGauss=# INSERT INTO  facts VALUES ('2019-02-21', generate_series(1,200));
openGauss=# INSERT INTO  facts VALUES ('2019-02-22', generate_series(1,300));
openGauss=# INSERT INTO  facts VALUES ('2019-02-23', generate_series(1,400));
openGauss=# INSERT INTO  facts VALUES ('2019-02-24', generate_series(1,500));
openGauss=# INSERT INTO  facts VALUES ('2019-02-25', generate_series(1,600));
openGauss=# INSERT INTO  facts VALUES ('2019-02-26', generate_series(1,700));
openGauss=# INSERT INTO  facts VALUES ('2019-02-27', generate_series(1,800));

-- 创建表并指定列为hll。
openGauss=# CREATE TABLE daily_uniques (
    date            date UNIQUE,
    users           hll
);

-- 根据日期把数据分组，并把数据插入到hll中。
openGauss=# INSERT INTO  daily_uniques(date, users)
    SELECT date, hll_add_agg(hll_hash_integer(user_id))
    FROM facts
    GROUP BY 1;

-- 计算每一天访问网站不同用户数量。
openGauss=# SELECT date, hll_cardinality(users) FROM daily_uniques ORDER BY date;
    date    | hll_cardinality
------------+------------------
 2019-02-20 |              100
 2019-02-21 | 200.217913059312
 2019-02-22 |  301.76494508014
 2019-02-23 | 400.862858326446
 2019-02-24 | 502.626933349694
 2019-02-25 | 601.922606454213
 2019-02-26 | 696.602316769498
 2019-02-27 | 798.111731634412
(8 rows)
```



```
-- 计算在2019.02.20到2019.02.26一周中有多少不同用户访问过网站。
openGauss=# SELECT hll_cardinality(hll_union_agg(users)) FROM daily_uniques WHERE date >=
'2019-02-20'::date AND date <= '2019-02-26'::date;
 hll_cardinality
------------------
 696.602316769498
(1 row)

-- 计算昨天访问过网站而今天没访问网站的用户数量。
openGauss=# SELECT date, (#hll_union_agg(users) OVER two_days) - #users AS lost_uniques FROM
daily_uniques WINDOW two_days AS (ORDER BY date ASC ROWS 1
PRECEDING);
    date    | lost_uniques
------------+--------------
 2019-02-20 |            0
 2019-02-21 |            0
 2019-02-22 |            0
 2019-02-23 |            0
 2019-02-24 |            0
 2019-02-25 |            0
 2019-02-26 |            0
 2019-02-27 |            0
(8 rows)

-- 删除表。
openGauss=# DROP TABLE facts;
openGauss=# DROP TABLE daily_uniques;
```

- 场景3：“插入数据不满足hll数据结构要求”

  当用户给hll类型的字段插入数据的时候，必须保证插入的数据满足hll数据结构要求，如果解析后不满足就会报错。如下示例中： 插入数据'E\\1234'时，该数据不满足hll数据结构要求，不能解析成功因此失败报错。

```
openGauss=# CREATE TABLE test(id integer, set hll);
openGauss=# INSERT INTO  test VALUES(1, 'E\\1234');
ERROR:  not a hll type, size=6 is not enough

openGauss=# DROP TABLE test;
```

7.3.13 范围类型

范围类型是表达某种元素类型（称为范围的subtype）的一个值的范围的数据类型。例如，timestamp的范围可以被用来表达一个会议室被保留的时间范围。在这种情况下，数据类型是tsrange（“timestamp range”的简写），而timestamp是其subtype。subtype必须具有一种总体的顺序，这样对于元素值是在一个范围值之内、之前还是之后就已经是明确的了。

范围类型非常有用，因为它们可以表达一种单一范围值中的多个元素值，并且可以很清晰地表达诸如范围重叠等概念，如用于时间安排的时间和日期范围，以及价格范围、仪器的量程等场景都是可以表示的。

内建范围类型

有下列内建范围类型：

- int4range — integer的范围
- int8range — bigint的范围
- numrange — numeric的范围
- tsrange — 不带时区的 timestamp的范围
- tstzrange — 带时区的 timestamp的范围

- daterange — date的范围

此外，用户可以定义自己的范围类型，详见**CREATE TYPE**。

例子

```
--创建表格并插入数据。
openGauss=# CREATE TABLE reservation (room int, during tsrange);
openGauss=# INSERT INTO reservation VALUES (1108, '[2010-01-01 14:30, 2010-01-01 15:30)');
-- 包含。
openGauss=# SELECT int4range(10, 20) @> 3;
 ?column?
----------
 f
(1 row)

--判断是否重叠。
openGauss=# SELECT numrange(11.1, 22.2) && numrange(20.0, 30.0);
 ?column?
----------
 t
(1 row)

--抽取上界。
openGauss=# SELECT upper(int8range(15, 25));
 upper
-------
    25
(1 row)
--计算交集。
openGauss=# SELECT int4range(10, 20) * int4range(15, 25);
 ?column?
----------
 [15,20)
(1 row)
--判断范围是否为空。
openGauss=# SELECT isempty(numrange(1, 5));
 isempty
---------
 f
(1 row)

--删除表格。
openGauss=# DROP TABLE reservation;
```

范围类型上的操作符和函数的完整列表可见**范围函数和操作符**。

包含和排除边界

每一个非空范围都有两个界限，下界和上界。上下界之间的所有值都被包括在范围内。一个包含界限意味着边界点本身也被包括在范围内，而一个排除边界意味着边界点不被包括在范围内。

在一个范围的文本形式中，一个包含下界被表达为“[”而一个排除下界被表达为“(”。同样，一个包含上界被表达为“]”而一个排除上界被表达为“)”（详见**范围输入/输出**）。

函数lower_inc和upper_inc分别测试一个范围值的上下界。

无限（无界）范围

一个范围的下界可以被忽略，意味着所有小于上界的值都被包括在范围中。 同样，如果范围的上界被忽略，那么所有比下界大的值都被包括在范围中。如果上下界都被忽略，该元素类型的所有值都被认为在该范围中。 规定缺失的包括界限自动转换为排

除。 用户可以认为这些缺失值为 +/- 无穷大，但它们是特殊范围类型值，并且被视为超出任何范围元素类型的 +/- 无穷大值。

具有“infinity”概念的元素类型可以用它们作为显式边界值。 例如，在时间戳范围，[today,infinity) 不包括特殊的 timestamp 值 infinity，尽管 [today,infinity] 包括它，就好比 [today,) 和 [today,]。

函数lower_inf和upper_inf分别测试一个范围的无限上下界。

范围输入/输出

一个范围值的输入必须遵循下列模式之一：

```
(lower-bound,upper-bound)
(lower-bound,upper-bound]
[lower-bound,upper-bound)
[lower-bound,upper-bound]
empty
```

一个范围值的输出必须遵循下列模式之一：

```
[lower-bound,upper-bound)
empty
```

圆括号或方括号指示上下界是否为排除的或者包含的。注意最后一个模式是empty，它表示一个空范围（一个不包含点的范围）。

lower-bound可以是作为 subtype 的合法输入的一个字符串，或者是空表示没有下界。同样，upper-bound可以是作为 subtype 的合法输入的一个字符串，或者是空表示没有上界。

每个界限值可以使用"（双引号）字符引用。如果界限值包含圆括号、方括号、逗号、双引号或反斜线时，这样做是必须的，否则那些字符会被认作范围语法的一部分。要把一个双引号或反斜线放在一个被引用的界限值中，就在它前面放一个反斜线（还有，在一个双引号引用的界限值中的一对双引号表示一个双引号字符，这与 SQL 字符串中的单引号规则类似）。此外，用户可以避免引用或者使用反斜线转义来保护所有数据字符，否则它们会被当做范围语法的一部分。还有，要写一个是空字符串的界限值，则可以写成""，因为什么都不写表示一个无限界限。

范围值前后允许有空格，但是圆括号或方括号之间的任何空格会被当做上下界值的一部分（取决于元素类型，它可能是也可能不是有意义的）。

例子：

```
--包括3，不包括7之间的所有点。
openGauss=# SELECT '[3,7)'::int4range;
 int4range
-----------
 [3,7)
(1 row)
--既不包括3也不包括7之间的所有点。
openGauss=# SELECT '(3,7)'::int4range;
 int4range
-----------
 [4,7)
(1 row)
--只包括单独一个点4。
openGauss=# SELECT '[4,4]'::int4range;
 int4range
-----------
 [4,5)
(1 row)
--不包括点（并且将被标准化为 '空'）。
openGauss=# SELECT '[4,4)'::int4range;
```



```
 int4range
-----------
 empty
(1 row)
```

构造范围

每一种范围类型都有一个与其同名的构造器函数。使用构造器函数常常比写一个范围文字常数更方便，因为它避免了对界限值的额外引用。构造器函数接受两个或三个参数。两个参数的形式以标准的形式构造一个范围（下界是包含的，上界是排除的），而三个参数的形式按照第三个参数指定的界限形式构造一个范围。第三个参数必须是下列字符串之一：“()”、“(]”、“[)”或者“[]”。例如：

```
--完整形式是：下界、上界以及指示界限包含性/排除性的文本参数。
openGauss=# SELECT numrange(1.0, 14.0, '(]');
  numrange
------------
 (1.0,14.0]
(1 row)
--如果第三个参数被忽略，则假定为 '[)'。
openGauss=# SELECT numrange(1.0, 14.0);
  numrange
------------
 [1.0,14.0)
(1 row)
--尽管这里指定了 '(]'，显示时该值将被转换成标准形式，因为int8range是一种离散范围类型。
openGauss=# SELECT int8range(1, 14, '(]');
 int8range
-----------
 [2,15)
(1 row)
--为一个界限使用NULL导致范围在那一边是无界的。
openGauss=# SELECT numrange(NULL, 2.2);
 numrange
----------
 (,2.2)
(1 row)
```

离散范围类型

一种范围的元素类型具有一个良定义的“步长”，例如integer或date。在这些类型中，如果两个元素之间没有合法值，它们可以被说成是相邻。这与连续范围相反，连续范围中总是（或者几乎总是）可以在两个给定值之间标识其他元素值。例如，numeric类型之上的一个范围就是连续的，timestamp上的范围也是（尽管timestamp具有有限的精度，并且在理论上可以被当做离散的，但是可以认为它是连续的，因为通常并不关心它的步长）。

另一种考虑离散范围类型的方法是对每一个元素值都有一个清晰的“下一个”或“上一个”值。了解了这种思想之后，通过选择原来给定的下一个或上一个元素值来取代它，就可以在一个范围界限的包含和排除表达之间转换。例如，在一个整数范围类型中，[4,8]和(3,9)表示相同的值集合，但是对于 numeric 上的范围就不是这样。

一个离散范围类型应该具有一个正规化函数，它知道元素类型期望的步长。正规化函数负责把范围类型的相等值转换成具有相同的表达，特别是与包含或者排除界限一致。如果没有指定一个正规化函数，那么具有不同格式的范围将总是会被当作不等，即使它们实际上是表达相同的一组值。

内建的范围类型int4range、int8range和daterange都使用一种正规的形式，该形式包括下界并且排除上界，也就是[)。不过，用户定义的范围类型可以使用其他形式。

索引

B-树和哈希索引可以在范围类型的表列上创建。对于这些索引类型，基本上唯一有用的范围操作就是等值。使用相应的< 和 >操作符，对于范围值定义有一种 B-树排序顺序，但是该顺序相当任意并且在真实世界中通常不怎么有用。范围类型的 B-树和哈希支持主要是为了允许在查询内部进行排序和哈希，而不是创建真正的索引。

7.3.14 对象标识符类型

GaussDB在内部使用对象标识符（OID）作为各种系统表的主键。系统不会给用户创建的表增加一个OID系统字段，OID类型代表一个对象标识符。

目前OID类型用一个四字节的无符号整数实现。因此不建议在创建的表中使用OID字段做主键。

表 7-22 对象标识符类型

名称	引用	描述	示例
OID	-	数字化的对象标识符。	564182
CID	-	命令标识符。它是系统字段cmin和cmax的数据类型。命令标识符是32位的量。	-
XID	-	事务标识符。它是系统字段xmin和xmax的数据类型。事务标识符也是64位的量。	-
TID	-	行标识符。它是系统表字段ctid的数据类型。行ID是一对数值（块号，块内的行索引），它标识该行在其所在表内的物理位置。	-
REGCONFIG	pg_ts_config	文本搜索配置。	english
REGDICTIONARY	pg_ts_dict	文本搜索字典。	simple
REGOPER	pg_operator	操作符名。	-
REGOPERATOR	pg_operator	带参数类型的操作符。	*(integer,integer)或-(NONE,integer)
REGPROC	pg_proc	函数名称。	sum
REGPROCEDURE	pg_proc	带参数类型的函数。	sum(int4)
REGCLASS	pg_class	关系名。	pg_type
REGTYPE	pg_type	数据类型名。	integer

OID类型：主要作为数据库系统表中字段使用。

示例：

```
openGauss=# SELECT oid FROM pg_class WHERE relname = 'pg_type';
 oid
------
 1247
(1 row)
```

OID别名类型REGCLASS：主要用于对象OID值的简化查找。

示例：

```
openGauss=# SELECT attrelid,attname,atttypid,attstattarget FROM pg_attribute WHERE attrelid =
'pg_type'::REGCLASS;
attrelid |  attname   | atttypid | attstattarget
----------+------------+----------+---------------
    1247 | xc_node_id     |       23 |             0
    1247 | tableoid       |       26 |             0
    1247 | cmax           |       29 |             0
    1247 | xmax           |       28 |             0
    1247 | cmin           |       29 |             0
    1247 | xmin           |       28 |             0
    1247 | oid            |       26 |             0
    1247 | ctid           |       27 |             0
    1247 | typname        |       19 |            -1
    1247 | typnamespace   |       26 |            -1
    1247 | typowner       |       26 |            -1
    1247 | typlen         |       21 |            -1
    1247 | typbyval       |       16 |            -1
    1247 | typtype        |       18 |            -1
    1247 | typcategory    |       18 |            -1
    1247 | typispreferred |       16 |            -1
    1247 | typisdefined   |       16 |            -1
    1247 | typdelim       |       18 |            -1
    1247 | typrelid       |       26 |            -1
    1247 | typelem        |       26 |            -1
    1247 | typarray       |       26 |            -1
    1247 | typinput       |       24 |            -1
    1247 | typoutput      |       24 |            -1
    1247 | typreceive     |       24 |            -1
    1247 | typsend        |       24 |            -1
    1247 | typmodin       |       24 |            -1
    1247 | typmodout      |       24 |            -1
    1247 | typanalyze     |       24 |            -1
    1247 | typalign       |       18 |            -1
    1247 | typstorage     |       18 |            -1
    1247 | typnotnull     |       16 |            -1
    1247 | typbasetype    |       26 |            -1
    1247 | typtypmod      |       23 |            -1
    1247 | typndims       |       23 |            -1
    1247 | typcollation   |       26 |            -1
    1247 | typdefaultbin  |      194 |            -1
    1247 | typdefault     |       25 |            -1
    1247 | typacl         |     1034 |            -1
(38 rows)
```

7.3.15 伪类型

GaussDB数据类型中包含一系列特殊用途的类型，这些类型按照类别被称为伪类型。伪类型不能作为字段的数据类型，但是可以用于声明函数的参数或者结果类型。

当一个函数不仅是简单地接受并返回某种SQL数据类型的情况下伪类型是很有用的。**表7-23**列出了所有的伪类型。

表 7-23 伪类型

名称	描述
any	表示函数接受任何输入数据类型。
anyelement	表示函数接受任何数据类型。
anyarray	表示函数接受任意数组数据类型。
anynonarray	表示函数接受任意非数组数据类型。
anyenum	表示函数接受任意枚举数据类型。
anyrange	表示函数接受任意范围数据类型。
cstring	表示函数接受或者返回一个空结尾的C字符串。
internal	表示函数接受或者返回一种服务器内部的数据类型。
language_handler	声明一个过程语言调用句柄返回language_handler。
fdw_handler	声明一个外部数据封装器返回fdw_handler。
record	标识函数返回一个未声明的行类型。
trigger	声明一个触发器函数返回trigger。
void	表示函数不返回数值。
opaque	一个已经过时的类型，以前用于所有上面这些用途。

声明用C编写的函数（不管是内置的还是动态装载的）都可以接受或者返回任何这样的伪数据类型。当伪类型作为参数类型使用时，用户需要保证函数的正常运行。

用过程语言编写的函数只能使用实现语言允许的伪类型。目前，过程语言都不允许使用作为参数类型的伪类型，并且只允许使用void和record作为结果类型。一些多态的函数还支持使用anyelement、anyarray、anynonarray、anyenum和anyrange类型。

伪类型internal用于声明只能在数据库系统内部调用的函数，这些函数不能直接在SQL查询里调用。如果函数至少有一个internal类型的参数，则不能从SQL里调用。建议不要创建任何声明返回internal的函数，除非至少有一个internal类型的参数。

示例：

```
--创建表。
openGauss=# create table t1 (a int);

--插入两条数据。
openGauss=# insert into t1 values(1),(2);

--创建函数showall()。
openGauss=# CREATE OR REPLACE FUNCTION showall() RETURNS SETOF record
AS $$ SELECT count(*) from t1; $$
LANGUAGE SQL;

--调用函数showall()。
openGauss=# SELECT showall();
 showall
---------
 (2)
```



```
(1 row)

--删除函数。
openGauss=# DROP FUNCTION showall();

--删除表。
openGauss=# drop table t1;
```

7.3.16 aclitem 类型

aclitem数据类型是用来存储对象权限信息的，它的内部实现是int类型，支持的格式为“*user1=privs|user2*”。

aclitem[]数据类型为aclitem组成的数组，支持的格式为“{*user1=privs1|user3*，*user2=privs2|user3*}”。

其中user1，user2和user3为数据库中已存在的用户/角色名，privs为数据库中支持的权限（参见**表12-30**）。

示例：

```
--创建相应用户。
openGauss=# CREATE USER user1 WITH PASSWORD 'Aa123456789';
openGauss=# CREATE USER user2 WITH PASSWORD 'Aa123456789';
openGauss=# CREATE USER omm WITH PASSWORD 'Aa123456789';

--新建一张数据表table_acl，有三个字段，类型分别为int、aclitem、aclitem[]。
openGauss=# CREATE TABLE table_acl (id int,priv aclitem,privs aclitem[]);
--向数据表table_acl插入一条内容为(1,'user1=arw/omm','{omm=d/user2,omm=w/omm}')的数据。
openGauss=# INSERT INTO table_acl VALUES (1,'user1=arw/omm','{omm=d/user2,omm=w/omm}');
--向数据表table_acl再插入一条内容为(2,'user1=aw/omm','{omm=d/user2}')的数据。
openGauss=# INSERT INTO table_acl VALUES (2,'user1=aw/omm','{omm=d/user2}');
openGauss=# SELECT * FROM  table_acl;
 id |     priv      |          privs
----+---------------+-------------------------
  1 | user1=arw/omm | {omm=d/user2,omm=w/omm}
  2 | user1=aw/omm  | {omm=d/user2}
(2 rows)

--删除表和用户。
openGauss=# DROP USER user1;
openGauss=# DROP USER user2;
openGauss=# DROP USER omm;
openGauss=# DROP TABLE table_acl;
```

7.4 常量与宏

GaussDB支持的常量和宏请参见**表7-24**。

表 7-24 常量和宏

参数	描述	示例
CURRENT_CATALOG	当前数据库	testdb=# SELECT CURRENT_CATALOG; current_database ---------- testdb (1 row)

参数	描述	示例
CURRENT_ROLE	当前用户	openGauss=# SELECT CURRENT_ROLE; current_user -------------- omm (1 row)
CURRENT_SCHEMA	当前数据库模式	openGauss=# SELECT CURRENT_SCHEMA; current_schema ---------------- public (1 row)
CURRENT_USER	当前用户	openGauss=# SELECT CURRENT_USER; current_user -------------- omm (1 row)
LOCALTIMESTAMP	当前会话时间（无时区）	openGauss=# SELECT LOCALTIMESTAMP; timestamp ---------------------------- 2015-10-10 15:37:30.968538 (1 row)
NULL	空值	-
SESSION_USER	当前系统用户	openGauss=# SELECT SESSION_USER; session_user -------------- omm (1 row)
SYSDATE	当前系统日期	openGauss=# SELECT SYSDATE; sysdate --------------------- 2015-10-10 15:48:53 (1 row)
USER	当前用户，此用户为CURRENT_USER的别名。	openGauss=# SELECT USER; current_user -------------- omm (1 row)

7.5 函数和操作符

操作符可以对一个或多个操作数进行处理，位置上可能处于操作数之前、之后，或两个操作数中间。完成处理之后，返回处理结果。

函数是对一些业务逻辑的封装，以完成特定的功能。函数可以有参数，也可以没有参数。函数是有返回类型的，执行完成后，会返回执行结果。

对于系统函数，用户可以进行修改，但是修改之后系统函数的语义可能会发生改变，从而导致系统控制紊乱。正常情况下不允许用户手工修改系统函数。

7.5.1 逻辑操作符

常用的逻辑操作符有AND、OR和NOT，运算结果有三个值，分别为TRUE、FALSE和NULL，其中NULL代表未知。运算优先级顺序为：NOT>AND>OR。

运算规则请参见**表7-25**，表中的a和b代表逻辑表达式。

表 7-25 运算规则表

a	b	a AND b的结果	a OR b的结果	NOT a的结果
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
TRUE	NULL	NULL	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE
FALSE	NULL	FALSE	NULL	TRUE
NULL	NULL	NULL	NULL	NULL

说明

操作符AND和OR具有交换性，即交换左右两个操作数，不影响其结果。

7.5.2 比较操作符

大部分数据类型都可用比较操作符进行比较，并返回一个布尔类型的值。

比较操作符均为双目操作符，被比较的两个数据类型必须是相同的数据类型或者是可以进行隐式转换的类型。

GaussDB提供的比较操作符请参见表7-26。

表 7-26 比较操作符

操作符	描述
<	小于
>	大于
<=	小于或等于
>=	大于或等于
=	等于
<> 或 !=或^=	不等于

- 比较操作符可以用于所有相关的数据类型。所有比较操作符都是双目操作符，返回布尔类型数值。不等号的计算优先级高于等号。当输入的数据类型不同且无法隐式转换时, 比较操作将会失败。例如像1<2<3这样的表达式是非法的，因为布尔值和3之间无法用小于号（<）比较。
- 另外，上述每种操作符在pg_proc系统表中都有对应的函数，如果其对应的函数的属性proleakproof值为f，表示该函数不是防数据泄露的。如果用户只拥有视图权限而不拥有该视图对应表的权限，在查询该视图的时候，可能存在查询计划不是最优的问题。

7.5.3 字符处理函数和操作符

GaussDB提供的字符处理函数和操作符主要用于字符串与字符串、字符串与非字符串之间的连接，以及字符串的模式匹配操作。注意：字符串处理函数除了length相关函数，其他函数和操作符不支持大于1GB clob作为参数。

- bit_length(string)

  描述：字符串的位数。

  返回值类型：int

  示例：

```
openGauss=# SELECT bit_length('world');
 bit_length
------------
         40
(1 row)
```

- btrim(string text [, characters text])

  描述：从string开头和结尾删除只包含characters中字符（缺省是空白）的最长字符串。

  返回值类型：text

  示例：

```
openGauss=# SELECT btrim('sring' , 'ing');
 btrim
-------
 sr
(1 row)
```

- char_length(string)或character_length(string)

  描述：字符串中的字符个数。

  返回值类型：int

  示例：

```
openGauss=# SELECT char_length('hello');
 char_length
-------------
           5
(1 row)
```

- instr(text,text,int,int)

  描述：instr(string1,string2,int1,int2)返回在string1中从int1位置开始匹配到第int2次string2的位置，第一个int表示开始匹配起始位置，第二个int表示匹配的次数。

  返回值类型：int

  示例：

```
openGauss=# SELECT instr( 'abcdabcdabcd', 'bcd', 2, 2 );
 instr
-------
     6
(1 row)
```

- lengthb(text/bpchar)

  描述：获取指定字符串的字节数。

  返回值类型：int

  示例：

```
openGauss=# SELECT lengthb('hello');
 lengthb
```



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

- left(str text, n int)

  描述：返回字符串的前n个字符。当n是负数时，返回除最后|n|个字符以外的所有字符。

  返回值类型：text

  示例：

```
openGauss=# SELECT left('abcde', 2);
 left
------
 ab
(1 row)
```

- length(string bytea, encoding name)

  描述：指定encoding编码格式的string的字符数。在这个编码格式中，string必须是有效的。

  返回值类型：int

  示例：

```
openGauss=# SELECT length('jose', 'UTF8');
 length
--------
      4
(1 row)
```

  说明

  如果是查询bytea类型的长度，指定utf8编码时，最大长度只能为536870888。

- lpad(string text, length int [, fill text])

  描述：通过填充字符fill（缺省时为空白），把string填充为length长度。如果string已经比length长则将其尾部截断。

  返回值类型：text

  示例：

```
openGauss=# SELECT lpad('hi', 5, 'xyza');
 lpad
-------
 xyzhi
(1 row)
```

- notlike(x bytea name text, y bytea text)

  描述：比较x和y是否不一致。

  返回值类型：Boolean

  示例：

```
openGauss=# SELECT notlike(1,2);
   notlike
--------------
          t
(1 row)
openGauss=# SELECT notlike(1,1);
   notlike
--------------
          f
(1 row)
```

- octet_length(string)

  描述：字符串中的字节数。

返回值类型：int

示例：

```
openGauss=# SELECT octet_length('jose');
 octet_length
--------------
            4
(1 row)
```

- overlay(string placing string FROM int [for int])

  描述：替换子字符串。FROM int表示从第一个string的第几个字符开始替换，for int表示替换第一个string的字符数目。

  返回值类型：text

  示例：

```
openGauss=# SELECT overlay('hello' placing 'world' from 2 for 3 );
 overlay
---------
 hworldo
(1 row)
```

- position(substring in string)

  描述：指定子字符串的位置。字符串区分大小写。

  返回值类型：int，字符串不存在时返回0。

  示例：

```
openGauss=# SELECT position('ing' in 'string');
 position
----------
        4
(1 row)
```

- pg_client_encoding()

  描述：当前客户端编码名称。

  返回值类型：name

  示例：

```
openGauss=# SELECT pg_client_encoding();
 pg_client_encoding
--------------------
 UTF8
(1 row)
```

- quote_ident(string text)

  描述：返回适用于SQL语句的标识符形式（使用适当的引号进行界定）。只有在必要的时候才会添加引号（字符串包含非标识符字符或者会转换大小写的字符）。返回值中嵌入的引号都写了两次。

  返回值类型：text

  示例：

```
openGauss=# SELECT quote_ident('hello world');
 quote_ident
--------------
 "hello world"
(1 row)
```

- quote_literal(string text)

  描述：返回适用于SQL语句的标识符形式（使用适当的引号进行界定）。

  返回值类型：text

  示例：


```
openGauss=# SELECT quote_literal('hello');
 quote_literal
---------------
 'hello'
(1 row)
```

如果出现如下写法，text文本将进行转义。

```
openGauss=# SELECT quote_literal(E'O\'hello');
 quote_literal
---------------
 'O''hello'
(1 row)
```

如果出现如下写法，反斜杠会写入两次。

```
openGauss=# SELECT quote_literal('O\hello');
 quote_literal
---------------
 E'O\\hello'
(1 row)
```

如果参数为NULL，返回空。如果参数可能为null，通常使用函数quote_nullable更适用。

```
openGauss=# SELECT quote_literal(NULL);
 quote_literal
---------------

(1 row)
```

- quote_literal(value anyelement)

  描述：将给定的值强制转换为text，加上引号作为文本。

  返回值类型：text

  示例：

  ```
  openGauss=# SELECT quote_literal(42.5);
   quote_literal
  ---------------
   '42.5'
  (1 row)
  ```

  如果出现如下写法，定值将进行转义。

  ```
  openGauss=# SELECT quote_literal(E'O\'42.5');
   quote_literal
  ---------------
   '0''42.5'
  (1 row)
  ```

  如果出现如下写法，反斜杠会写入两次。

  ```
  openGauss=# SELECT quote_literal('O\42.5');
   quote_literal
  ---------------
   E'O\\42.5'
  (1 row)
  ```

- quote_nullable(string text)

  描述：返回适用于在SQL语句里当作字符串使用的形式（使用适当的引号进行界定）。

  返回值类型：text

  示例：

  ```
  openGauss=# SELECT quote_nullable('hello');
   quote_nullable
  ----------------
   'hello'
  (1 row)
  ```

  如果出现如下写法，text文本将进行转义。


```
openGauss=# SELECT quote_nullable(E'O\'hello');
 quote_nullable
----------------
 'O''hello'
(1 row)
```

如果出现如下写法，反斜杠会写入两次。

```
openGauss=# SELECT quote_nullable('O\hello');
 quote_nullable
----------------
 E'O\\hello'
(1 row)
```

如果参数为NULL，返回NULL。

```
openGauss=# SELECT quote_nullable(NULL);
 quote_nullable
----------------
 NULL
(1 row)
```

- quote_nullable(value anyelement)

  描述：将给定的参数值转换为text，加上引号作为文本。

  返回值类型：text

  示例：

```
openGauss=# SELECT quote_nullable(42.5);
 quote_nullable
----------------
 '42.5'
(1 row)
```

如果出现如下写法，定值将进行转义。

```
openGauss=# SELECT quote_nullable(E'O\'42.5');
 quote_nullable
----------------
 'O''42.5'
(1 row)
```

如果出现如下写法，反斜杠会写入两次。

```
openGauss=# SELECT quote_nullable('O\42.5');
 quote_nullable
----------------
 E'O\\42.5'
(1 row)
```

如果参数为NULL，返回NULL。

```
openGauss=# SELECT quote_nullable(NULL);
 quote_nullable
----------------
 NULL
(1 row)
```

- substring_inner(string [from int] [for int])

  描述：截取子字符串，from int表示从第几个字符开始截取，for int表示截取几个字节。

  返回值类型：text

  示例：

```
openGauss=# select substring_inner('adcde', 2,3);
 substring_inner
-----------------
 dcd
(1 row)
```

- substring(string [from int] [for int])

描述：截取子字符串，from int表示从第几个字符开始截取，for int表示截取几个字节。

返回值类型：text

示例：

```
openGauss=# SELECT substring('Thomas' from 2 for 3);
 substring
-----------
 hom
(1 row)
```

- substring(string from *pattern*)

  描述：截取匹配POSIX正则表达式的子字符串。如果没有匹配它返回空值，否则返回文本中匹配模式的那部分。

  返回值类型：text

  示例：

```
openGauss=# SELECT substring('Thomas' from '...$');
 substring
-----------
 mas
(1 row)
openGauss=# SELECT substring('foobar' from 'o(.)b');
 result
--------
 o
(1 row)
openGauss=# SELECT substring('foobar' from '(o(.)b)');
 result
--------
 oob
(1 row)
```

  说明

  如果POSIX正则表达式模式包含任何圆括号，那么将返回匹配第一对子表达式（对应第一个左圆括号的）的文本。如果你想在表达式里使用圆括号而又不想导致这个例外，那么你可以在整个表达式外边放上一对圆括号。

- substring(string from *pattern* for *escape*)

  描述：截取匹配SQL正则表达式的子字符串。声明的模式必须匹配整个数据串，否则函数失败并返回空值。为了标识在成功的时候应该返回的模式部分，模式必须包含两个逃逸字符引用的部分，并且逃逸字符后面要添加双引号（"）。匹配这两个标记之间的模式的文本将被返回。

  返回值类型：text

  示例：

```
openGauss=# SELECT substring('Thomas' from '%#"o_a#"_' for '#');
 substring
-----------
 oma
(1 row)
```

- rawcat(raw,raw)

  描述：字符串拼接函数。

  返回值类型：raw

  示例：

```
openGauss=# SELECT rawcat('ab','cd');
 rawcat
--------
 ABCD
(1 row)
```


- regexp_like(text,text,text)

  描述：正则表达式的模式匹配函数。

  返回值类型：bool

  示例：

  ```
  openGauss=# SELECT regexp_like('str','[ac]');
   regexp_like
  -------------
   f
  (1 row)
  ```

- regexp_substr(string text, pattern text [, position int [, occurrence int [, flags text]]])

  描述：正则表达式的抽取子串函数。与substr功能相似，正则表达式出现多个并列的括号时，也全部处理。

  参数说明：

  - string：用于匹配的源字符串。
  - pattern：用于匹配的正则表达式模式串。
  - position：可选参数，表示从源字符串的第几个字符开始匹配，默认值为1。
  - occurrence：可选参数，表示抽取第几个满足匹配的子串，默认值为1。
  - flags：可选参数，包含零个或多个改变函数匹配行为的单字母标记。flags 支持的选项值及含义描述如表7-27所示：

    表 7-27 flags 支持的选项值

选项	描述
'b'	按照无扩展的 BRE 规则匹配。
'c'	大小写敏感匹配。
'e'	按照扩展的ERE规则匹配。
'i'	大小写不敏感匹配。
'm'	多行模式匹配。flags中包含'm'时，按照多行模式匹配，否则按照单行模式匹配。
'n'	n选项的含义和GUC参数**behavior_compat_options**及数据库当前的兼容模式有关： ● 数据库SQL语法兼容模式为A或B，且GUC参数**behavior_compat_options**值包含aformat_regexp_match时，'n'选项表示“.”能够匹配换行符('\n')；flags未指定'n'选项时，“.”不会匹配换行符。 ● 其他情况下，'n'选项和'm'选项的含义一样。
'p'	部分新行敏感的匹配，影响'.'和方括号表达式，和新行敏感的匹配('m'或'n')一样，但是不影响^和$。
'q'	普通字符匹配。
's'	单行模式匹配，含义与'm', 'n'相反。
't'	紧凑模式匹配，空白符匹配自身。

选项	描述
'w'	逆部分新行匹配，与'p'含义相反。
'x'	宽松模式匹配，忽略空白符。

返回值类型：text

示例：

```
openGauss=# SELECT regexp_substr('str','[ac]');
 regexp_substr
---------------


(1 row)

openGauss=# SELECT regexp_substr('foobarbaz', 'b(..)', 3, 2) AS RESULT;
 result
--------
 baz
(1 row)
```

- regexp_count(string text, pattern text [, position int [, flags text]])

  描述：获取满足匹配的子串个数。

  参数说明：

  - string：用于匹配的源字符串。
  - pattern：用于匹配的正则表达式模式串。
  - position：表示从源字符串的第几个字符开始匹配，为可选参数，默认值为1。
  - flags：可选参数，包含零个或多个改变函数匹配行为的单字母标记。flags 支持的选项值及含义描述如**表7-27**所示。

  返回值类型：int

  示例：

  ```
  openGauss=# SELECT regexp_count('foobarbaz','b(..)', 5) AS RESULT;
  result
  --------
  1
  (1 row)
  ```

- regexp_instr(string text, pattern text [, position int [, occurrence int [, return_opt int [, flags text]]]])

  描述：获取满足匹配条件的子串位置（从1开始）。如果没有匹配的子串，则返回0。

  参数说明：

  - string：用于匹配的源字符串。
  - pattern：用于匹配的正则表达式模式串。
  - position：可选参数，表示从源字符串的第几个字符开始匹配，默认值为1。
  - occurrence：可选参数，表示获取第occurrence个匹配子串的位置，默认值为1。
  - return_opt：可选参数，用于控制返回匹配子串的首字符位置还是尾字符位置。取值为0时，返回匹配子串的第一个字符的位置（从1开始计算），取值为大于0的值时，返回匹配子串的尾字符的下一个字符的位置。默认值为0。

– flags：可选参数，包含零个或多个改变函数匹配行为的单字母标记。flags 支持的选项值及含义描述如**表7-27**所示。

返回值类型：int

示例：

```
openGauss=# SELECT regexp_instr('foobarbaz','b(..)', 1, 1, 0) AS RESULT;
result
--------
4
(1 row)

openGauss=# SELECT regexp_instr('foobarbaz','b(..)', 1, 2, 0) AS RESULT;
result
--------
7
(1 row)
```

- regexp_matches(string text, pattern text [, flags text])

  描述：返回string中所有匹配POSIX正则表达式的子字符串。如果pattern不匹配，该函数不返回行。如果模式不包含圆括号子表达式，则每一个被返回的行都是一个单一元素的文本数组，其中包括匹配整个模式的子串。如果模式包含圆括号子表达式，该函数返回一个文本数组，它的第n个元素是匹配模式的第n个圆括号子表达式的子串。

  flags参数为可选参数，包含零个或多个改变函数行为的单字母标记。i表示进行大小写无关的匹配，g表示替换每一个匹配的子字符串而不仅仅是第一个。

  须知

  如果提供了最后一个参数，但参数值是空字符串（''），且数据库SQL兼容模式设置为ORA的情况下，会导致返回结果为空集。这是因为ORA兼容模式将''作为NULL处理，避免此类行为的方式有如下几种：

  - 将数据库SQL兼容模式改为TD。
  - 不提供最后一个参数，或最后一个参数不为空字符串。

  返回值类型：setof text[]

  示例：

```
openGauss=# SELECT regexp_matches('foobarbequebaz', '(bar)(beque)');
 regexp_matches
----------------
 {bar,beque}
(1 row)
openGauss=# SELECT regexp_matches('foobarbequebaz', 'barbeque');
 regexp_matches
----------------
 {barbeque}
(1 row)
openGauss=# SELECT regexp_matches('foobarbequebazilbarfbonk', '(b[^b]+)(b[^b]+)', 'g');
 regexp_matches
---------------
 {bar,beque}
 {bazil,barf}
(2 rows)
```

- regexp_split_to_array(string text, pattern text [, flags text])

  描述：用POSIX正则表达式作为分隔符，分隔string。和regexp_split_to_table相同，不过regexp_split_to_array会把它的结果以一个text数组的形式返回。

  返回值类型：text[]

示例：

```
openGauss=# SELECT regexp_split_to_array('hello world', E'\\s+');
 regexp_split_to_array
-----------------------
 {hello,world}
(1 row)
```

- regexp_split_to_table(string text, pattern text [, flags text])

  描述：用POSIX正则表达式作为分隔符，分隔string。如果没有与pattern的匹配，该函数返回string。如果至少有一个匹配，对每一个匹配，它都返回从上一个匹配的末尾（或者串的开头）到这次匹配开头之间的文本。当没有更多匹配时，它返回从上一次匹配的末尾到串末尾之间的文本。

  flags参数包含零个或多个改变函数行为的单字母标记。i表示进行大小写无关的匹配，不设参数默认表示替换每一个匹配的子字符串，而不仅仅是替换第一个子字符串。

  返回值类型：setof text

  示例：

```
openGauss=# SELECT regexp_split_to_table('hello world', E'\\s+');
 regexp_split_to_table
-----------------------
 hello
 world
(2 rows)
```

- repeat(string text, number int)

  描述：将string重复number次。

  返回值类型：text

  示例：

```
openGauss=# SELECT repeat('Pg', 4);
  repeat
----------
 PgPgPgPg
(1 row)
```

  说明

  由于数据库内存分配机制限制单次内存分配不可超过1GB，因此number最大值不应超过(1G-x)/lengthb(string) - 1。x为头信息长度，通常大于4字节，其具体值在不同的场景下存在差异。

- replace(string text, from text, to text)

  描述：把字符串string里出现的所有子字符串from的内容替换成子字符串to的内容。

  返回值类型：text

  示例：

```
openGauss=# SELECT replace('abcdefabcdef', 'cd', 'XXX');
    replace
----------------
 abXXXefabXXXef
(1 row)
```

- replace(string, substring)

  描述：删除字符串string里出现的所有子字符串substring的内容。

  string类型：text

  substring类型：text

  返回值类型：text

示例：

```
openGauss=# SELECT replace('abcdefabcdef', 'cd');
    replace
----------------
 abefabef
(1 row)
```

- reverse(str)

  描述：返回颠倒的字符串。

  返回值类型：text

  示例：

  ```
  openGauss=# SELECT reverse('abcde');
   reverse
  ---------
   edcba
  (1 row)
  ```

- right(str text, n int)

  描述：返回字符串中的后n个字符。当n是负值时，返回除前|n|个字符以外的所有字符。

  返回值类型：text

  示例：

  ```
  openGauss=# SELECT right('abcde', 2);
   right
  -------
   de
  (1 row)

  openGauss=# SELECT right('abcde', -2);
   right
  -------
   cde
  (1 row)
  ```

- rpad(string text, length int [, fill text])

  描述：使用填充字符fill（缺省时为空白），把string填充到length长度。如果string已经比length长则将其从尾部截断。

  返回值类型：text

  示例：

  ```
  openGauss=# SELECT rpad('hi', 5, 'xy');
   rpad
  -------
   hixyx
  (1 row)
  ```

- rtrim(string text [, characters text])

  描述：从字符串string的结尾删除只包含characters中字符（缺省是个空白）的最长的字符串。

  返回值类型：text

  示例：

  ```
  openGauss=# SELECT rtrim('trimxxxx', 'x');
   rtrim
  -------
   trim
  (1 row)
  ```

- substrb(text,int,int)

  描述：提取子字符串，第一个int表示提取的起始位置，第二个表示提取几位字符。

返回值类型：text

示例：

```
openGauss=# SELECT substrb('string',2,3);
 substrb
---------
 tri
(1 row)
```

- substrb(text,int)

  描述：提取子字符串，int表示提取的起始位置。

  返回值类型：text

  示例：

  ```
  openGauss=# SELECT substrb('string',2);
   substrb
  ---------
   tring
  (1 row)
  ```

- substr(bytea,from,count)

  描述：从参数bytea中抽取子字符串。from表示抽取的起始位置，count表示抽取的子字符串长度。

  返回值类型：text

  示例：

  ```
  openGauss=# SELECT substr('string',2,3);
   substr
  --------
   tri
  (1 row)
  ```

- string || string

  描述：连接字符串。

  返回值类型：text

  示例：

  ```
  openGauss=# SELECT 'MPP'||'DB' AS RESULT;
   result
  --------
   MPPDB
  (1 row)
  ```

- string || non-string或non-string || string

  描述：连接字符串和非字符串。

  返回值类型：text

  示例：

  ```
  openGauss=# SELECT 'Value: '||42 AS RESULT;
    result
  -----------
   Value: 42
  (1 row)
  ```

- split_part(string text, delimiter text, field int)

  描述：根据delimiter分隔string返回生成的第field个子字符串（从出现第一个delimiter的text为基础）。

  返回值类型：text

  示例：

  ```
  openGauss=# SELECT split_part('abc~@~def~@~ghi', '~@~', 2);
   split_part
  ```



```
------------
 def
(1 row)
```

- strpos(string, substring)

  描述：指定的子字符串的位置。和position(substring in string)一样，不过参数顺序相反。

  返回值类型：int

  示例：

```
openGauss=# SELECT strpos('source', 'rc');
 strpos
--------
      4
(1 row)
```

- to_hex(number int or bigint)

  描述：把number转换成十六进制表现形式。

  返回值类型：text

  示例：

```
openGauss=# SELECT to_hex(2147483647);
  to_hex
----------
 7fffffff
(1 row)
```

- translate(string text, from text, to text)

  描述：把在string中包含的任何匹配from中的字符转换为对应的在to中的字符。如果from比to长，删掉在from中出现的额外的字符。

  返回值类型：text

  示例：

```
openGauss=# SELECT translate('12345', '143', 'ax');
 translate
-----------
 a2x5
(1 row)
```

- length(string)

  描述：获取参数string中字符的数目。

  返回值类型：integer

  示例：

```
openGauss=# SELECT length('abcd');
 length
--------
      4
(1 row)
```

- lengthb(string)

  描述：获取参数string中字节的数目。与字符集有关，同样的中文字符，在GBK与UTF8中，返回的字节数不同。

  返回值类型：integer

  示例：

```
openGauss=# SELECT lengthb('Chinese');
 lengthb
---------
       7
(1 row)
```


- substr(string,from)

  描述：

  从参数string中抽取子字符串。

  from表示抽取的起始位置。

  - from为0时，按1处理。
  - from为正数时，抽取从from到末尾的所有字符。
  - from为负数时，抽取字符串的后n个字符，n为from的绝对值。

  返回值类型：text

  示例：

  from为正数时：

  ```
  openGauss=# SELECT substr('ABCDEF',2);
   substr
  --------
   BCDEF
  (1 row)
  ```

  from为负数时：

  ```
  openGauss=# SELECT substr('ABCDEF',-2);
   substr
  --------
   EF
  (1 row)
  ```

- substr(string,from,count)

  描述：

  从参数string中抽取子字符串。

  from表示抽取的起始位置。

  count表示抽取的子字符串长度。

  - from为0时，按1处理。
  - from为正数时，抽取从from开始的count个字符。
  - from为负数时，抽取从倒数第n个开始的count个字符，n为from的绝对值。
  - count小于1时，返回null。

  返回值类型：text

  示例：

  from为正数时：

  ```
  openGauss=# SELECT substr('ABCDEF',2,2);
   substr
  --------
   BC
  (1 row)
  ```

  from为负数时：

  ```
  openGauss=# SELECT substr('ABCDEF',-3,2);
   substr
  --------
   DE
  (1 row)
  ```

- substrb(string,from)

  描述：该函数和SUBSTR(string,from)函数功能一致，但是计算单位为字节。

  返回值类型：text

  示例：


```
openGauss=# SELECT substrb('ABCDEF',-2);
 substrb
---------
 EF
(1 row)
```

- substrb(string,from,count)

  描述：该函数和SUBSTR(string,from,count)函数功能一致，但是计算单位为字节。

  返回值类型：text

  示例：

```
openGauss=# SELECT substrb('ABCDEF',2,2);
 substrb
---------
 BC
(1 row)
```

- trim([leading |trailing |both] [characters] from string)

  描述：从字符串string的开头、结尾或两边删除只包含characters中字符（缺省是一个空白）的最长的字符串。

  返回值类型：text

  示例：

```
openGauss=# SELECT trim(BOTH 'x' FROM 'xTomxx');
 btrim
-------
 Tom
(1 row)
openGauss=# SELECT trim(LEADING 'x' FROM 'xTomxx');
 ltrim
-------
 Tomxx
(1 row)
openGauss=# SELECT trim(TRAILING 'x' FROM 'xTomxx');
 rtrim
-------
 xTom
(1 row)
```

- rtrim(string [, characters])

  描述：从字符串string的结尾删除只包含characters中字符（缺省是个空白）的最长的字符串。

  返回值类型：text

  示例：

```
openGauss=# SELECT rtrim('TRIMxxxx','x');
 rtrim
-------
 TRIM
(1 row)
```

- ltrim(string [, characters])

  描述：从字符串string的开头删除只包含characters中字符（缺省是一个空白）的最长的字符串。

  返回值类型：text

  示例：

```
openGauss=# SELECT ltrim('xxxxTRIM','x');
 ltrim
-------
 TRIM
(1 row)
```


- upper(string)

  描述：把字符串转换为大写。

  返回值类型：text

  示例：

```
openGauss=# SELECT upper('tom');
 upper
-------
 TOM
(1 row)
```

- lower(string)

  描述：把字符串转换为小写。

  返回值类型：text

  示例：

```
openGauss=# SELECT lower('TOM');
 lower
-------
 tom
(1 row)
```

- rpad(string varchar, length int [, fill varchar])

  描述：使用填充字符fill（缺省时为空白），把string填充到length长度。如果string已经比length长则将其从尾部截断。

  length参数在GaussDB中表示字符长度。一个汉字长度计算为一个字符。

  返回值类型：text

  示例：

```
openGauss=# SELECT rpad('hi',5,'xyza');
 rpad
-------
 hixyx
(1 row)
openGauss=# SELECT rpad('hi',5,'abcdefg');
 rpad
-------
 hiabc
(1 row)
```

- instr(string,substring[,position,occurrence])

  描述：从字符串string的position（缺省时为1）所指的位置开始查找并返回第occurrence（缺省时为1）次出现子串substring的位置的值。

  - 当position为0时，返回0。
  - 当position为负数时，从字符串倒数第n个字符往前逆向搜索。n为position的绝对值。

  本函数以字符为计算单位，如一个汉字为一个字符。

  返回值类型：integer

  示例：

```
openGauss=# SELECT instr('corporate floor','or', 3);
 instr
-------
     5
(1 row)
openGauss=# SELECT instr('corporate floor','or',-3,2);
 instr
-------
     2
(1 row)
```


- initcap(string)

  描述：将字符串中的每个单词的首字母转换为大写，其他字母转换为小写。

  返回值类型：text

  示例：

```
openGauss=# SELECT initcap('hi THOMAS');
  initcap
-----------
 Hi Thomas
(1 row)
```

- ascii(string)

  描述：参数string的第一个字符的ASCII码。

  返回值类型：integer

  示例：

```
openGauss=# SELECT ascii('xyz');
 ascii
-------
   120
(1 row)
```

- replace(string varchar, search_string varchar, replacement_string varchar)

  描述：把字符串string中所有子字符串search_string替换成子字符串replacement_string。

  返回值类型：text

  示例：

```
openGauss=# SELECT replace('jack and jue','j','bl');
    replace
----------------
 black and blue
(1 row)
```

- lpad(string varchar, length int[, repeat_string varchar])

  描述：在string的左侧添上一系列的repeat_string（缺省为空白）来组成一个总长度为n的新字符串。

  如果string本身的长度比指定的长度length长，则本函数将把string截断并把前面长度为length的字符串内容返回。

  返回值类型：varchar

  示例：

```
openGauss=# SELECT lpad('PAGE 1',15,'*.');
      lpad
----------------
 *.*.*.*.*PAGE 1
(1 row)
openGauss=# SELECT lpad('hello world',5,'abcd');
 lpad
-------
 hello
(1 row)
```

- concat(str1,str2)

  描述：将字符串str1和str2连接并返回，若str1或str2为NULL时，返回NULL。注意，concat会调用data type的输出函数，返回值不确定，导致优化器在生成计划的时候不能提前计算结果。如果对性能有要求，建议用 || 替代。

说明

- 在sql_compatibility = 'MYSQL'的情况下，参数str1或str2为NULL会导致返回结果为NULL。
- concat函数返回值类型为变长类型，和表中数据比较时，会因为拼接结果丢失字符串长度，导致比较结果不相等。

返回值类型：text

示例：

```
openGauss=# SELECT concat('Hello', ' World!');
   concat
--------------
 Hello World!
(1 row)
openGauss=# SELECT concat('Hello', NULL);
 concat
--------

(1 row)
```

- chr(integer)

  描述：给出ASCII码的字符。

  返回值类型：varchar

  示例：

```
openGauss=# SELECT chr(65);
 chr
-----
 A
(1 row)
```

- regexp_replace(string, pattern, replacement [,flags])

  描述：替换匹配POSIX正则表达式的子字符串。 如果没有匹配pattern，那么返回不加修改的string串。 如果有匹配，则返回的string串里面的匹配子串将被replacement串替换掉。

  replacement串可以包含\n， 其中\n是1 到9， 表明string串里匹配模式里第n个圆括号子表达式的子串应该被插入， 并且它可以包含\&表示应该插入匹配整个模式的子串。

  可选的flags参数包含零个或多个改变函数行为的单字母标记。i表示进行大小写无关的匹配，g表示替换每一个匹配的子字符串而不仅仅是第一个。

  返回值类型：text

  示例：

```
openGauss=# SELECT regexp_replace('Thomas', '.[mN]a.', 'M');
 regexp_replace
----------------
 ThM
(1 row)
openGauss=# SELECT regexp_replace('foobarbaz','b(..)', E'X\\1Y', 'g') AS
RESULT;
   result
-------------
 fooXarYXazY
(1 row)
```

- concat_ws(sep text, str"any" [, str"any" [, ...]])

  描述：以第一个参数为分隔符，连接第二个以后的所有参数。NULL参数被忽略。

须知

- 如果第一个参数值是NULL，会导致返回结果为NULL。
- 如果第一个参数值是空字符串（''），且数据库SQL兼容模式设置为ORA的情况下，会导致返回结果为NULL。这是因为A兼容模式将''作为NULL处理，避免此类行为，可以将数据库SQL兼容模式改为MYSQL、TD或者Postgres。

返回值类型：text

示例：

```
openGauss=# SELECT concat_ws(',', 'ABCDE', 2, NULL, 22);
 concat_ws
------------
 ABCDE,2,22
(1 row)
```

- nlssort(string text, sort_method text)

  描述：以sort_method指定的排序方式返回字符串在该排序模式下的编码值，该编码值可用于排序，其决定了string在这种排序模式下的先后位置。目前支持的sort_method为'nls_sort=schinese_pinyin_m'和'nls_sort=generic_m_ci'。其中，'nls_sort=generic_m_ci'仅支持纯英文不区分大小写排序。

  string类型：text

  sort_method类型：text

  返回值类型：text

  示例：

```
openGauss=# create table test(a text);

openGauss=# insert into test(a) values ('abC 不');

openGauss=# insert into test(a) values ('abC 啊');

openGauss=# insert into test(a) values ('abc 啊');

openGauss=# select * from test order by nlssort(a,'nls_sort=schinese_pinyin_m');
   a
--------
 abc 啊
 abC 啊
 abC 不
(3 rows)

openGauss=# select * from test order by nlssort(a, 'nls_sort=generic_m_ci');
   a
--------
 abC 啊
 abc 啊
 abC 不
(3 rows)

openGauss=# DROP TABLE test;
```

- convert(string bytea, src_encoding name, dest_encoding name)

  描述：以dest_encoding指定的目标编码方式转换字符串string。src_encoding指定源编码方式，在该编码下，string必须是合法的。

  返回值类型：bytea

  示例：

```
openGauss=# SELECT convert('text_in_utf8', 'UTF8', 'GBK');
        convert
```



```
----------------------------
 \x746578745f696e5f75746638
(1 row)
```

说明

如果源编码格式到目标编码格式的转换规则不存在，则字符串不进行任何转换直接返回，如GBK和LATIN1之间的转换规则是不存在的，具体转换规则可以通过查看系统表pg_conversion获得。

示例：

```
openGauss=# show server_encoding;
 server_encoding
-----------------
 LATIN1
(1 row)

openGauss=# SELECT convert_from('some text', 'GBK');
 convert_from
--------------
 some text
(1 row)

db_latin1=# SELECT convert_to('some text', 'GBK');
     convert_to
----------------------
 \x736f6d652074657874
(1 row)

db_latin1=# SELECT convert('some text', 'GBK', 'LATIN1');
      convert
----------------------
 \x736f6d652074657874
(1 row)
```

- convert_from(string bytea, src_encoding name)

  描述：以数据库的编码方式转换字符串string。

  src_encoding指定源编码方式，在该编码下，string必须是合法的。

  返回值类型：text

  示例：

  ```
  openGauss=# SELECT convert_from('text_in_utf8', 'UTF8');
   convert_from
  --------------
   text_in_utf8
  (1 row)
  ```

- convert_to(string text, dest_encoding name)

  描述：将字符串转换为dest_encoding的编码格式。

  返回值类型：bytea

  示例：

  ```
  openGauss=# SELECT convert_to('some text', 'UTF8');
      convert_to
  ----------------------
   \x736f6d652074657874
  (1 row)
  ```

- string [NOT] LIKE pattern [ESCAPE escape-character]

  描述：模式匹配函数。

  如果pattern不包含百分号或者下划线，该模式只代表它本身，这时候LIKE的行为就像等号操作符。在pattern里的下划线（_）匹配任何单个字符；而一个百分号（%）匹配零或多个任何字符。

要匹配下划线或者百分号本身，在pattern里相应的字符必须前导逃逸字符。缺省的逃逸字符是反斜杠，但是用户可以用ESCAPE子句指定一个。要匹配逃逸字符本身，写两个逃逸字符。

返回值类型：Boolean

示例：

```
openGauss=# SELECT 'AA_BBCC' LIKE '%A@_B%' ESCAPE '@' AS RESULT;
 result
--------
 t
(1 row)
openGauss=# SELECT 'AA_BBCC' LIKE '%A@_B%' AS RESULT;
 result
--------
 f
(1 row)
openGauss=# SELECT 'AA@_BBCC' LIKE '%A@_B%' AS RESULT;
 result
--------
 t
(1 row)
```

- REGEXP_LIKE(source_string, pattern [, match_parameter])

  描述：正则表达式的模式匹配函数。

  source_string为源字符串，pattern为正则表达式匹配模式。 match_parameter为匹配选项，可取值为：

  - 'i'：大小写不敏感。
  - 'c'：大小写敏感。
  - 'n'：允许正则表达式元字符“.”匹配换行符。
  - 'm'：将source_string视为多行。

  若忽略match_parameter选项，默认为大小写敏感，“.”不匹配换行符，source_string视为单行。

  返回值类型：Boolean

  示例：

```
openGauss=# SELECT regexp_like('ABC', '[A-Z]');
 regexp_like
-------------
 t
(1 row)
openGauss=# SELECT regexp_like('ABC', '[D-Z]');
 regexp_like
-------------
 f
(1 row)
openGauss=# SELECT regexp_like('ABC', '[A-Z]','i');
 regexp_like
-------------
 t
(1 row)
openGauss=# SELECT regexp_like('ABC', '[A-Z]');
 regexp_like
-------------
 t
(1 row)
```

- format(formatstr text [, str"any" [, ...]])

  描述：格式化字符串。

  返回值类型：text

  示例：


```
openGauss=# SELECT format('Hello %s, %1$s', 'World');
      format
-------------------
 Hello World, World
(1 row)
```

- md5(string)

  描述：将string使用MD5加密，并以16进制数作为返回值。

  说明

  MD5加密算法安全性低，存在安全风险，建议使用更安全的加密算法。

  返回值类型：text

  示例：

```
openGauss=# SELECT md5('ABC');
               md5
----------------------------------
 902fbdd2b1df0c4f70b4a5d23525e932
(1 row)
```

- decode(string text, format text)

  描述：将二进制数据从文本数据中解码。

  返回值类型：bytea

  示例：

```
openGauss=# SELECT decode('MTIzAAE=', 'base64');
   decode
--------------
 \x3132330001
(1 row)
```

- similar_escape(pat text, esc text)

  描述：将一个SQL:2008风格的正则表达式转换为POSIX风格。

  返回值类型：text

  示例：

```
openGauss=# select similar_escape('\s+ab','2');
 similar_escape
----------------
 ^(?:\\s+ab)$
(1 row)
```

- encode(data bytea, format text)

  描述：将二进制数据编码为文本数据。

  返回值类型：text

  示例：

```
openGauss=# SELECT encode(E'123\\000\\001', 'base64');
  encode
----------
 MTIzAAE=
(1 row)
```


说明

- 若字符串中存在换行符，如字符串由一个换行符和一个空格组成，在GaussDB中LENGTH和LENGTHB的值为2。
- 对于CHAR(n) 类型，GaussDB中n是指字符个数。因此，对于多字节编码的字符集，LENGTHB函数返回的长度可能大于n。
- GaussDB支持多种类型的数据库，目前有4种，分别是ORA类型，MYSQL类型，TD类型以及Postgres类型。在不指定数据库类型时，数据库默认是ORA类型，ORA的词法分析器与另外三种不一样，在ORA中空字符串会被当作是NULL。所以，当使用ORA类型的数据库时，假如上述字符操作函数中有空字符串作为参数，会出现没有输出的情况。例如：

```
openGauss=# SELECT translate('12345','123','');
translate
-----------


(1 row)
```

这是因为内核在调用相应的函数进行处理前，会判断所输入的参数中是否含有NULL，假如有，则不会调用相应的函数，因此会没有输出。而在Postgres模式下，字符串的处理方式与Postgres保持一致，因此不会有上述问题产生。

7.5.4 二进制字符串函数和操作符

字符串操作符

SQL定义了一些字符串函数，在这些函数里使用关键字而不是逗号来分隔参数。

- octet_length(string)

  描述：二进制字符串中的字节数。

  返回值类型：int

  示例：

```
openGauss=# SELECT octet_length(E'jo\\000se'::bytea) AS RESULT;
 result
--------
      5
(1 row)
```

- overlay(string placing string from int [for int])

  描述：替换子串。

  返回值类型：bytea

  示例：

```
openGauss=# SELECT overlay(E'Th\\000omas'::bytea placing E'\\002\\003'::bytea from 2 for 3) AS
RESULT;
     result
----------------
 \x5402036d6173
(1 row)
```

- position(substring in string)

  描述：特定子字符串的位置。

  返回值类型：int

  示例：

```
openGauss=# SELECT position(E'\\000om'::bytea in E'Th\\000omas'::bytea) AS RESULT;
 result
--------
      3
(1 row)
```


- substring(string [from int] [for int])
  描述：截取子串。
  返回值类型：bytea
  示例：

```
openGauss=# SELECT substring(E'Th\\000omas'::bytea from 2 for 3) AS RESULT;
 result
----------
 \x68006f
(1 row)
```

- substr(bytea [from int] [for int])
  描述：截取子串。
  返回值类型：bytea
  示例：

```
openGauss=# select substr(E'Th\\000omas'::bytea,2, 3) as result;
 result
----------
 \x68006f
(1 row)
```

- trim([both] bytes from string)
  描述：从string的开头和结尾删除只包含bytes中字节的最长字符串。
  返回值类型：bytea
  示例：

```
openGauss=# SELECT trim(E'\\000'::bytea from E'\\000Tom\\000'::bytea) AS RESULT;
 result
----------
 \x546f6d
(1 row)
```

二进制字符串函数

GaussDB也提供了函数调用所使用的常用语法。

- btrim(string bytea,bytes bytea)
  描述：从string的开头和结尾删除只包含bytes中字节的最长的字符串。
  返回值类型：bytea
  示例：

```
openGauss=# SELECT btrim(E'\\000trim\\000'::bytea, E'\\000'::bytea) AS RESULT;
   result
------------
 \x7472696d
(1 row)
```

- get_bit(string, offset)
  描述：从字符串中抽取位。
  返回值类型：int
  示例：

```
openGauss=# SELECT get_bit(E'Th\\000omas'::bytea, 45) AS RESULT;
 result
--------
      1
(1 row)
```

- get_byte(string, offset)
  描述：从字符串中抽取字节。

返回值类型：int

示例：

```
openGauss=# SELECT get_byte(E'Th\\000omas'::bytea, 4) AS RESULT;
 result
--------
    109
(1 row)
```

- set_bit(string,offset, newvalue)

  描述：设置字符串中的位。

  返回值类型：bytea

  示例：

  ```
  openGauss=# SELECT set_bit(E'Th\\000omas'::bytea, 45, 0) AS RESULT;
       result
  ------------------
   \x5468006f6d4173
  (1 row)
  ```

- set_byte(string,offset, newvalue)

  描述：设置字符串中的字节。

  返回值类型：bytea

  示例：

  ```
  openGauss=# SELECT set_byte(E'Th\\000omas'::bytea, 4, 64) AS RESULT;
       result
  ------------------
   \x5468006f406173
  (1 row)
  ```

- rawcmp

  描述：raw数据类型比较函数。

  参数：raw, raw

  返回值类型：integer

- raweq

  描述：raw数据类型比较函数。

  参数：raw, raw

  返回值类型：boolean

- rawge

  描述：raw数据类型比较函数。

  参数：raw, raw

  返回值类型：boolean

- rawgt

  描述：raw数据类型比较函数。

  参数：raw, raw

  返回值类型：boolean

- rawin

  描述：raw数据类型解析函数。

  参数：cstring

  返回值类型：bytea

- rawle
  描述：raw数据类型解析函数。
  参数：raw, raw
  返回值类型：boolean
- rawlike
  描述：raw数据类型解析函数。
  参数：raw, raw
  返回值类型：boolean
- rawlt
  描述：raw数据类型解析函数。
  参数：raw, raw
  返回值类型：boolean
- rawne
  描述：比较raw类型是否一样。
  参数：raw, raw
  返回值类型：boolean
- rawnlike
  描述：比较raw类型与模式是否不匹配。
  参数：raw, raw
  返回值类型：boolean
- rawout
  描述：RAW类型的输出接口。
  参数：bytea
  返回值类型：cstring
- rawsend
  描述：转换bytea为二进制类型。
  参数：raw
  返回值类型：bytea
- rawtohex
  描述：raw格式转换为十六进制。
  参数：text
  返回值类型：text

7.5.5 位串函数和操作符

位串操作符

除了常用的比较操作符之外，还可以使用以下的操作符。&，|和#的位串操作数必须等长。在位移的时候，保留原始的位串长度（并以0填充）。

- ||
  描述：位串之间进行连接。
  示例：


```
openGauss=# SELECT B'10001' || B'011' AS RESULT;
 result
----------
 10001011
(1 row)
```

说明

- 单字段内部连续连接操作不能超过180次，若超过需拆分为多个连续连接的字符串，它们之间再做连接操作。

  例如：str1||str2||str3||str4 拆分为 (str1||str2)||(str3||str4)。
- ORA兼容模式下位串中包含了空字符串，会忽略空字符串连接其他字符串，其他兼容模式下则会返回空串。

  例如：str1||NULL||str2，ORA兼容模式下返回str1str2，其他兼容模式下返回NULL。

- &

  描述：位串之间进行“与”操作。

  示例：

```
openGauss=# SELECT B'10001' & B'01101' AS RESULT;
 result
--------
 00001
(1 row)
```

- |

  描述：位串之间进行“或”操作。

  示例：

```
openGauss=# SELECT B'10001' | B'01101' AS RESULT;
 result
--------
 11101
(1 row)
```

- #

  描述：位串之间如果不一致进行“或”操作。如果两个位串中对应位置都为1或者0，则该位置返回为0。

  示例：

```
openGauss=# SELECT B'10001' # B'01101' AS RESULT;
 result
--------
 11100
(1 row)
```

- ~

  描述：位串之间进行“非”操作。

  示例：

```
openGauss=# SELECT ~B'10001'AS RESULT;
 result
----------
 01110
(1 row)
```

- <<

  描述：位串进行左移操作。

  示例：

```
openGauss=# SELECT B'10001' << 3 AS RESULT;
 result
----------
 01000
(1 row)
```


- >>

  描述：位串进行右移操作。

  示例：

```
openGauss=# SELECT B'10001' >> 2 AS RESULT;
 result
----------
 00100
(1 row)
```

下面的SQL标准函数除了可以用于字符串之外，也可以用于位串：length，bit_length，octet_length，position，substring，overlay。

下面的函数用于位串和二进制字符串：get_bit，set_bit。当用于位串时，这些函数位数从字符串的第一位（最左边）作为0位。

另外，可以在整数和bit之间来回转换。示例：

```
openGauss=# SELECT 44::bit(10) AS RESULT;
   result
------------
 0000101100
(1 row)

openGauss=# SELECT 44::bit(3) AS RESULT;
 result
--------
 100
(1 row)

openGauss=# SELECT cast(-44 as bit(12)) AS RESULT;
    result
--------------
 111111010100
(1 row)

openGauss=# SELECT '1110'::bit(4)::integer AS RESULT;
 result
--------
     14
(1 row)

openGauss=# select substring('10101111'::bit(8), 2);
 substring
-----------
 0101111
(1 row)
```

说明

只是转换为“bit”的意思是转换成bit(1)，因此只会转换成整数的最低位。

7.5.6 模式匹配操作符

数据库提供了三种独立的实现模式匹配的方法：SQL LIKE操作符、SIMILAR TO操作符和POSIX-风格的正则表达式。除了这些基本的操作符外，还有一些函数可用于提取或替换匹配子串并在匹配位置分离一个串。

- LIKE

  描述：判断字符串是否能匹配上LIKE后的模式字符串。如果字符串与提供的模式匹配，则LIKE表达式返回为真（NOT LIKE表达式返回假），否则返回为假（NOT LIKE表达式返回真）。

  匹配规则：

a. 此操作符只有在它的模式匹配整个串的时候才能成功。如果要匹配在串内任何位置的序列，该模式必须以百分号开头和结尾。

b. 下划线（_）代表（匹配）任何单个字符； 百分号（%）代表任意串的通配符。

c. 要匹配文本里的下划线或者百分号，在提供的模式里相应字符必须前导逃逸字符。逃逸字符的作用是禁用元字符的特殊含义，缺省的逃逸字符是反斜线，也可以用ESCAPE子句指定一个不同的逃逸字符。

d. 要匹配逃逸字符本身，写两个逃逸字符。例如要写一个包含反斜线的模式常量，那你就要在SQL语句里写两个反斜线。

> **说明**
>
> 参数standard_conforming_strings设置为off时，在文串常量中写的任何反斜线都需要被双写。因此，写一个匹配单个反斜线的模式实际上要在语句里写四个反斜线。（你可以通过用ESCAPE选择一个不同的逃逸字符来避免这种情况，这样反斜线就不再是LIKE的特殊字符了。但仍然是字符文本分析器的特殊字符，所以你还是需要两个反斜线。）也可以通过写“ESCAPE”的方式不选择逃逸字符，这样可以有效地禁用逃逸机制，但是没有办法关闭下划线和百分号在模式中的特殊含义。

e. 关键字ILIKE可以用于替换LIKE，区别是LIKE大小写敏感，ILIKE大小写不敏感。

f. 操作符~~等效于LIKE，操作符~~*等效于ILIKE。

示例：

```
openGauss=# SELECT 'abc' LIKE 'abc' AS RESULT;
 result
------------
 t
(1 row)
openGauss=# SELECT 'abc' LIKE 'a%' AS RESULT;
 result
------------
 t
(1 row)
openGauss=# SELECT 'abc' LIKE '_b_' AS RESULT;
 result
------------
 t
(1 row)
openGauss=# SELECT 'abc' LIKE 'c' AS RESULT;
 result
------------
 f
(1 row)
```

- SIMILAR TO

  描述：SIMILAR TO操作符根据自己的模式是否匹配给定串而返回真或者假。他和LIKE非常类似，只不过他使用SQL标准定义的正则表达式理解模式。

  匹配规则：

  a. 和LIKE一样，此操作符只有在它的模式匹配整个串的时候才能成功。如果要匹配在串内任何位置的序列，该模式必须以百分号开头和结尾。

  b. 下划线（_）代表（匹配）任何单个字符； 百分号（%）代表任意串的通配符。

  c. SIMILAR TO也支持下面这些从POSIX正则表达式借用的模式匹配元字符。

元字符	含义
|	表示选择（两个候选之一）

元字符	含义
*	表示重复前面的项零次或更多次
+	表示重复前面的项一次或更多次
?	表示重复前面的项零次或一次
{m}	表示重复前面的项刚好m次
{m,}	表示重复前面的项m次或更多次
{m,n}	表示重复前面的项至少m次并且不超过n次
()	把多个项组合成一个逻辑项
[...]	声明一个字符类，就像POSIX正则表达式一样

d. 前导逃逸字符可以禁止所有这些元字符的特殊含义。逃逸字符的使用规则和LIKE一样。

正则表达式函数：

支持使用函数**substring(string from pattern for escape)**截取匹配SQL正则表达式的子字符串。

示例：

```
openGauss=# SELECT 'abc' SIMILAR TO 'abc' AS RESULT;
 result
------------
 t
(1 row)
openGauss=# SELECT 'abc' SIMILAR TO 'a' AS RESULT;
 result
------------
 f
(1 row)
openGauss=# SELECT 'abc' SIMILAR TO '%(b|d)%' AS RESULT;
 result
------------
 t
(1 row)
openGauss=# SELECT 'abc' SIMILAR TO '(b|c)%'  AS RESULT;
 result
------------
 f
(1 row)
```

- POSIX正则表达式

  描述：正则表达式是一个字符序列，它是定义一个串集合（一个正则集）的缩写。 如果一个串是正则表达式描述的正则集中的一员时， 就说这个串匹配该正则表达式。 POSIX正则表达式提供了比LIKE和SIMILAR TO操作符更强大的含义。**表1 正则表达式匹配操作符**列出了所有可用于POSIX正则表达式模式匹配的操作符。

表 7-28 正则表达式匹配操作符

操作符	描述	例子
~	匹配正则表达式，大小写敏感	'thomas' ~ '.*thomas.*'
~*	匹配正则表达式，大小写不敏感	'thomas' ~* '.*Thomas.*'
！ ~	不匹配正则表达式，大小写敏感	'thomas' !~ '.*Thomas.*'
！ ~*	不匹配正则表达式，大小写不敏感	'thomas' !~* '.*vadim.*'

匹配规则：

a. 与LIKE不同，正则表达式允许匹配串里的任何位置，除非该正则表达式显式地挂接在串的开头或者结尾。

b. 除了上文提到的元字符外， POSIX正则表达式还支持下列模式匹配元字符。

元字符	含义
^	表示串开头的匹配
$	表示串末尾的匹配
.	匹配任意单个字符

正则表达式函数：

POSIX正则表达式支持下面函数。

- **substring(string from pattern)**函数提供了抽取一个匹配POSIX正则表达式模式的子串的方法。
- **regexp_count(string text, pattern text [, position int [, flags text]])**函数提供了获取匹配POSIX正则表达式模式的子串数量的功能。
- **regexp_instr(string text, pattern text [, position int [, occurrence int [, return_opt int [, flags text]]]])**函数提供了获取匹配POSIX正则表达式模式子串位置的功能。
- **regexp_substr(string text, pattern text [, position int [, occurrence int [, flags text]]])**函数提供了抽取一个匹配POSIX正则表达式模式的子串的方法。
- **regexp_replace(string, pattern, replacement [,flags])**函数提供了将匹配POSIX正则表达式模式的子串替换为新文本的功能。
- **regexp_matches(string text, pattern text [, flags text])**函数返回一个文本数组，该数组由匹配一个POSIX正则表达式模式得到的所有被捕获子串构成。
- **regexp_split_to_table(string text, pattern text [, flags text])**函数把一个POSIX正则表达式模式当作一个定界符来分离一个串。

- **regexp_split_to_array(string text, pattern text [, flags text])**和regexp_split_to_table类似，是一个正则表达式分离函数，不过它的结果以一个text数组的形式返回。

  说明

  正则表达式分离函数会忽略零长度的匹配，这种匹配发生在串的开头或结尾或者正好发生在前一个匹配之后。这和正则表达式匹配的严格定义是相悖的，后者由regexp_matches实现，但是通常前者是实际中最常用的行为。

示例：

```
openGauss=# SELECT 'abc' ~ 'Abc' AS RESULT;
result
--------
 f
(1 row)
openGauss=# SELECT 'abc' ~* 'Abc' AS RESULT;
 result
--------
 t
(1 row)
openGauss=# SELECT 'abc' !~ 'Abc' AS RESULT;
 result
--------
 t
(1 row)
openGauss=# SELECT 'abc'!~* 'Abc' AS RESULT;
 result
--------
 f
(1 row)
openGauss=# SELECT 'abc' ~ '^a' AS RESULT;
 result
--------
 t
(1 row)
openGauss=# SELECT 'abc' ~ '(b|d)'AS RESULT;
 result
--------
 t
(1 row)
openGauss=# SELECT 'abc' ~ '^(b|c)'AS RESULT;
 result
--------
 f
(1 row)
```

虽然大部分的正则表达式搜索都能很快地执行，但是正则表达式仍可能被人为地处理成需要任意长的时间和任意量的内存。不建议从非安全模式来源接受正则表达式搜索模式，如果必须这样做，建议加上语句超时限制。使用SIMILAR TO模式的搜索具有同样的安全性危险， 因为SIMILAR TO提供了很多和POSIX-风格正则表达式相同的能力。LIKE搜索比其他两种选项简单得多，因此在接受非安全模式来源搜索时要更安全些。

7.5.7 数字操作函数和操作符

数字操作符

- +

  描述：加

  示例：

```
openGauss=# SELECT 2+3 AS RESULT;
 result
```



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

- -

  描述：减

  示例：

```
openGauss=# SELECT 2-3 AS RESULT;
 result
--------
    -1
(1 row)
```

- *

  描述：乘

  示例：

```
openGauss=# SELECT 2*3 AS RESULT;
 result
--------
     6
(1 row)
```

- /

  描述：除（除法操作符不会取整）

  示例：

```
openGauss=# SELECT 4/2 AS RESULT;
 result
--------
     2
(1 row)
openGauss=# SELECT 4/3 AS RESULT;
      result
------------------
 1.33333333333333
(1 row)
```

- +/-

  描述：正/负

  示例：

```
openGauss=# SELECT -2 AS RESULT;
 result
--------
    -2
(1 row)
```

- %

  描述：模（求余）

  示例：

```
openGauss=# SELECT 5%4 AS RESULT;
 result
--------
     1
(1 row)
```

- @

  描述：绝对值

  示例：

```
openGauss=# SELECT @ -5.0 AS RESULT;
 result
--------
```



```
   5.0
(1 row)
```

- ^

  描述：幂（指数运算）

  示例：

```
openGauss=# SELECT 2.0^3.0 AS RESULT;
       result
--------------------
 8.0000000000000000
(1 row)
```

- |/

  描述：平方根

  示例：

```
openGauss=# SELECT |/ 25.0 AS RESULT;
 result
--------
      5
(1 row)
```

- ||/

  描述：立方根

  示例：

```
openGauss=# SELECT ||/ 27.0 AS RESULT;
 result
--------
      3
(1 row)
```

- !

  描述：阶乘

  示例：

```
openGauss=# SELECT 5! AS RESULT;
 result
--------
    120
(1 row)
```

- !!

  描述：阶乘（前缀操作符）

  示例：

```
openGauss=# SELECT !!5 AS RESULT;
 result
--------
    120
(1 row)
```

- &

  描述：二进制AND

  示例：

```
openGauss=# SELECT 91&15  AS RESULT;
 result
--------
     11
(1 row)
```

- |

  描述：二进制OR

示例：

```
openGauss=# SELECT 32|3  AS RESULT;
 result
--------
     35
(1 row)
```

- #

  描述：二进制XOR

  示例：

```
openGauss=# SELECT 17#5  AS RESULT;
 result
--------
     20
(1 row)
```

- ~

  描述：二进制NOT

  示例：

```
openGauss=# SELECT ~1 AS RESULT;
 result
--------
     -2
(1 row)
```

- <<

  描述：二进制左移

  示例：

```
openGauss=# SELECT 1<<4 AS RESULT;
 result
--------
     16
(1 row)
```

- >>

  描述：二进制右移

  示例：

```
openGauss=# SELECT 8>>2 AS RESULT;
 result
--------
      2
(1 row)
```

数字操作函数

- abs(x)

  描述：绝对值。

  返回值类型：和输入相同。

  示例：

```
openGauss=# SELECT abs(-17.4);
 abs
------
 17.4
(1 row)
```

- acos(x)

  描述：反余弦。

  返回值类型：double precision

示例：

```
openGauss=# SELECT acos(-1);
       acos
------------------
 3.14159265358979
(1 row)
```

- asin(x)

  描述：反正弦。

  返回值类型：double precision

  示例：

  ```
  openGauss=# SELECT asin(0.5);
         asin
  ------------------
   .523598775598299
  (1 row)
  ```

- atan(x)

  描述：反正切。

  返回值类型：double precision

  示例：

  ```
  openGauss=# SELECT atan(1);
         atan
  ------------------
   .785398163397448
  (1 row)
  ```

- atan2(y, x)

  描述：y/x的反正切。

  返回值类型：double precision

  示例：

  ```
  openGauss=# SELECT atan2(2, 1);
       atan2
  ------------------
   1.10714871779409
  (1 row)
  ```

- bitand(integer, integer)

  描述：计算两个数字与运算(&)的结果。

  返回值类型：bigint类型数字。

  示例：

  ```
  openGauss=# SELECT bitand(127, 63);
   bitand
  --------
      63
  (1 row)
  ```

- cbrt(dp)

  描述：立方根。

  返回值类型：double precision

  示例：

  ```
  openGauss=# SELECT cbrt(27.0);
   cbrt
  ------
     3
  (1 row)
  ```


- ceil(x)
  描述：不小于参数的最小的整数。
  返回值类型：整数。
  示例：

```
openGauss=# SELECT ceil(-42.8);
 ceil
------
  -42
(1 row)
```

- ceiling(dp or numeric)
  描述：不小于参数的最小整数（ceil的别名）。
  返回值类型：与输入相同。
  示例：

```
openGauss=# SELECT ceiling(-95.3);
 ceiling
---------
     -95
(1 row)
```

- cos(x)
  描述：余弦。
  返回值类型：double precision
  示例：

```
openGauss=# SELECT cos(-3.1415927);
        cos
-------------------
 -.999999999999999
(1 row)
```

- cot(x)
  描述：余切。
  返回值类型：double precision
  示例：

```
openGauss=# SELECT cot(1);
       cot
------------------
 .642092615934331
(1 row)
```

- degrees(dp)
  描述：把弧度转为角度。
  返回值类型：double precision
  示例：

```
openGauss=# SELECT degrees(0.5);
     degrees
------------------
 28.6478897565412
(1 row)
```

- div(y numeric, x numeric)
  描述：y除以x的商的整数部分。
  返回值类型：numeric
  示例：

```
openGauss=# SELECT div(9,4);
 div
```



```
-----
   2
(1 row)
```

- exp(x)

  描述：自然指数。

  返回值类型：与输入相同。

  示例：

```
openGauss=# SELECT exp(1.0);
        exp
--------------------
 2.7182818284590452
(1 row)
```

- floor(x)

  描述：不大于参数的最大整数。

  返回值类型：与输入相同。

  示例：

```
openGauss=# SELECT floor(-42.8);
 floor
-------
   -43
(1 row)
```

- int1(in)

  描述：将传入的text参数转换为int1类型值并返回。

  返回值类型：int1

  示例：

```
openGauss=# select int1('123');
 int1
------
 123
(1 row)
openGauss=# select int1('a');
 int1
------
   0
(1 row)
```

- int2(in)

  描述：将传入参数转换为int2类型值并返回。支持的入参类型包括：bigint，float4，float8，int16，integer，numeric，real，text。

  返回值类型：int2

  示例：

```
openGauss=# select int2('1234');
 int2
------
 1234
(1 row)
openGauss=# select int2(25.3);
 int2
------
   25
(1 row)
```

- int4(in)

  描述：将传入参数转换为int4类型值并返回。支持的入参类型包括：bit，boolean，char，double precision，int16，numeric，real，smallint，text。

  返回值类型：int4

示例：

```
openGauss=# select int4('789');
 int4
------
 789
(1 row)
openGauss=# select int4(99.9);
 int4
------
   99
(1 row)
```

- int8(in)

  描述：将传入参数转换为int8类型值并返回。支持的入参类型包括：bit，double precision，int16，integer，numeric，oid，real，smallint，text。

  返回值类型：int8

  示例：

```
openGauss=# select int8('789');
 int8
------
 789
(1 row)
openGauss=# select int8(99.9);
 int8
------
   99
(1 row)
```

- float4(in)

  描述：将传入参数转换为float4类型值并返回。支持的入参类型包括：bigint，double precision，int16, integer, numeric，smallint，text。

  返回值类型：float4

  示例：

```
openGauss=# select float4('789');
 float4
--------
    789
(1 row)

openGauss=# select float4(99.9);
 float4
--------
   99.9
(1 row)
```

- float8(in)

  描述：将传入参数转换为float8类型值并返回。支持的入参类型包括：bigint，int16, integer, numeric，real，smallint，text。

  返回值类型：float8

  示例：

```
openGauss=# select float8('789');
 float8
--------
    789
(1 row)

openGauss=# select float8(99.9);
 float8
--------
   99.9
(1 row)
```


- int16(in)

  描述：将传入参数转换为int16类型值并返回。支持的入参类型包括：bigint，boolean, double precision, integer，numeric，oid，real，smallint，tinyint。

  返回值类型：int16

  示例：

```
openGauss=# select int16('789');
 int16
--------
   789
(1 row)

openGauss=# select int16(99.9);
 int16
--------
  99
(1 row)
```

- numeric(in)

  描述：将传入参数转换为numeric类型值并返回。支持的入参类型包括：bigint，boolean, double precision, int16，integer，money，real，smallint。

  返回值类型：numeric

  示例：

```
openGauss=# select "numeric"('789');
 numeric
---------
    789
(1 row)

openGauss=# select "numeric"(99.9);
 numeric
---------
   99.9
(1 row)
```

- oid(in)

  描述：将传入参数转换为oid类型值并返回。支持的入参类型包括：bigint，int16。

  返回值类型：oid

- radians(dp)

  描述：把角度转为弧度。

  返回值类型：double precision

  示例：

```
openGauss=# SELECT radians(45.0);
     radians
------------------
 .785398163397448
(1 row)
```

- random()

  描述：0.0到1.0之间的随机数。

  返回值类型：double precision

  示例：

```
openGauss=# SELECT random();
      random
------------------
 .824823560658842
(1 row)
```


- multiply(x double precision or text, y double precision or text)
  描述：x和y的乘积。
  返回值类型：double precision
  示例：

```
openGauss=# SELECT multiply(9.0, '3.0');
    multiply
-------------------
                27
(1 row)
openGauss=# SELECT multiply('9.0', 3.0);
    multiply
-------------------
                27
(1 row)
```

- ln(x)
  描述：自然对数。
  返回值类型：与输入相同。
  示例：

```
openGauss=# SELECT ln(2.0);
         ln
--------------------
 .6931471805599453
(1 row)
```

- log(x)
  描述：以10为底的对数。
  返回值类型：与输入相同。
  示例：

```
openGauss=# SELECT log(100.0);
        log
--------------------
 2.0000000000000000
(1 row)
```

- log(b numeric, x numeric)
  描述：以b为底的对数。
  返回值类型：numeric
  示例：

```
openGauss=# SELECT log(2.0, 64.0);
        log
--------------------
 6.0000000000000000
(1 row)
```

- mod(x,y)
  描述：
  x/y的余数（模）
  如果x是0，则返回y。
  返回值类型：与参数类型相同。
  示例：

```
openGauss=# SELECT mod(9,4);
 mod
-----
   1
(1 row)
```



```
openGauss=# SELECT mod(9,0);
 mod
-----
   9
(1 row)
```

- pi()

  描述：“π”常量。

  返回值类型：double precision

  示例：

```
openGauss=# SELECT pi();
        pi
------------------
 3.14159265358979
(1 row)
```

- power(a double precision, b double precision)

  描述：a的b次幂。

  返回值类型：double precision

  示例：

```
openGauss=# SELECT power(9.0, 3.0);
        power
----------------------
 729.0000000000000000
(1 row)
```

- round(x)

  描述：离输入参数最近的整数。

  返回值类型：与输入相同。

  示例：

```
openGauss=# SELECT round(42.4);
 round
-------
    42
(1 row)

openGauss=# SELECT round(42.6);
 round
-------
    43
(1 row)
```

- round(v numeric, s int)

  描述：保留小数点后s位，s后一位进行四舍五入。

  返回值类型：numeric

  示例：

```
openGauss=# SELECT round(42.4382, 2);
 round
-------
 42.44
(1 row)
```

- setseed(dp)

  描述：为随后的random()调用设置种子(-1.0到1.0之间，包含)。

  返回值类型：void

  示例：

```
openGauss=# SELECT setseed(0.54823);
 setseed
```



```
---------

(1 row)
```

- sign(x)

  描述：输出此参数的符号。

  返回值类型：-1表示负数，0表示0，1表示正数。

  示例：

  ```
  openGauss=# SELECT sign(-8.4);
   sign
  ------
    -1
  (1 row)
  ```

- sin(x)

  描述：正弦。

  返回值类型：double precision

  示例：

  ```
  openGauss=# SELECT sin(1.57079);
         sin
  -------------------
   .999999999979986
  (1 row)
  ```

- sqrt(x)

  描述：平方根。

  返回值类型：与输入相同。

  示例：

  ```
  openGauss=# SELECT sqrt(2.0);
         sqrt
  --------------------
   1.414213562373095
  (1 row)
  ```

- tan(x)

  描述：正切。

  返回值类型：double precision

  示例：

  ```
  openGauss=# SELECT tan(20);
         tan
  -------------------
   2.23716094422474
  (1 row)
  ```

- trunc(x)

  描述：截断（取整数部分）。

  返回值类型：与输入相同。

  示例：

  ```
  openGauss=# SELECT trunc(42.8);
   trunc
  -------
     42
  (1 row)
  ```

- trunc(v numeric, s int)

  描述：截断为s位小数。

  返回值类型：numeric

示例：

```
openGauss=# SELECT trunc(42.4382, 2);
 trunc
-------
 42.43
(1 row)
```

- width_bucket(op numeric, b1 numeric, b2 numeric, count int)

  描述：返回一个桶，这个桶是在一个有count个桶，上界为b1下界为b2的等深柱图中operand将被赋予的那个桶。

  返回值类型：int

  示例：

  ```
  openGauss=# SELECT width_bucket(5.35, 0.024, 10.06, 5);
   width_bucket
  --------------
             3
  (1 row)
  ```

- width_bucket(op dp, b1 dp, b2 dp, count int)

  描述：返回一个桶，这个桶是在一个有count个桶，上界为b1下界为b2的等深柱图中operand将被赋予的那个桶。

  返回值类型：int

  示例：

  ```
  openGauss=# SELECT width_bucket(5.35, 0.024, 10.06, 5);
   width_bucket
  --------------
             3
  (1 row)
  ```

- int1abs

  描述：返回uint8类型数据的绝对值。

  参数：tinyint

  返回值类型：tinyint

- int1and

  描述：返回两个uint8类型数据按位与的结果。

  参数：tinyint, tinyint

  返回值类型：tinyint

- int1cmp

  描述：返回两个uint8类型数据比较的结果，若第一个参数大，则返回1；若第二个参数大，则返回-1；若相等，则返回0。

  参数：tinyint, tinyint

  返回值类型：integer

- int1div

  描述：返回两个uint8类型数据相除的结果，结果为float8类型。

  参数：tinyint, tinyint

  返回值类型：tinyint

- int1eq

  描述：比较两个uint8类型数据是否相等。

  参数：tinyint, tinyint

  返回值类型：boolean

- int1ge
  描述：判断两个uint8类型数据是否第一个参数大于等于第二个参数。
  参数：tinyint, tinyint
  返回值类型：boolean
- int1gt
  描述：无符号1字节整数做大于运算。
  参数：tinyint, tinyint
  返回值类型：boolean
- int1larger
  描述：返回无符号一字节整数中较大值。
  参数：tinyint, tinyint
  返回值类型：tinyint
- int1le
  描述：判断无符号一字节整数是否小于等于。
  参数：tinyint, tinyint
  返回值类型：boolean
- int1lt
  描述：判断无符号一字节整数是否小于。
  参数：tinyint, tinyint
  返回值类型：boolean
- int1smaller
  描述：返回两个无符号一字节整数中较小的数。
  参数：tinyint, tinyint
  返回值类型：tinyint
- int1inc
  描述：无符号一字节整数加一。
  参数：tinyint
  返回值类型：tinyint
- int1mi
  描述：无符号一字节整数做差运算。
  参数：tinyint, tinyint
  返回值类型：tinyint
- int1mod
  描述：无符号一字节整数做取余运算。
  参数：tinyint, tinyint
  返回值类型：tinyint
- int1mul
  描述：无符号一字节整数做乘法运算。
  参数：tinyint, tinyint
  返回值类型：tinyint

- int1ne
  描述：无符号一字节整数不等于运算。
  参数：tinyint, tinyint
  返回值类型：boolean
- int1pl
  描述：无符号一字节整数加法。
  参数：tinyint, tinyint
  返回值类型：tinyint
- int1um
  描述：无符号一字节数取相反数并返回有符号二字节整数。
  参数：tinyint
  返回值类型：smallint
- int1xor
  描述：无符号一字节整数异或操作。
  参数：tinyint, tinyint
  返回值类型：tinyint
- cash_div_int1
  描述：对money类型进行除法运算。
  参数：money, tinyint
  返回值类型：money
- cash_mul_int1
  描述：对money类型进行乘法运算。
  参数：money, tinyint
  返回值类型：money
- int1not
  描述：无符号一字节整数二进制位翻转。
  参数：tinyint
  返回值类型：tinyint
- int1or
  描述：无符号一字节整数或运算。
  参数：tinyint, tinyint
  返回值类型：tinyint
- int1shl
  描述：无符号一字节整数左移指定位数。
  参数：tinyint, integer
  返回值类型：tinyint
- int1shr
  描述：无符号一字节整数右移指定位数。
  参数：tinyint, integer
  返回值类型：tinyint

7.5.8 时间和日期处理函数和操作符

时间日期操作符

警告

用户在使用时间和日期操作符时，对应的操作数请使用明确的类型前缀修饰，以确保数据库在解析操作数的时候能够与用户预期一致，不会产生用户非预期的结果。

比如下面示例没有明确数据类型就会出现异常错误。

```
openGauss=# SELECT date '2001-10-01' - '7' AS RESULT;
ERROR:
GAUSS-10416: invalid input syntax for type timestamp: "7"
SQLSTATE: 22007
LINE 1: SELECT date '2001-10-01' - '7' AS RESULT;
                                   ^
CONTEXT:  referenced column: result
```

表 7-29 时间和日期操作符

操作符	示例
+	openGauss=# SELECT date '2001-9-28' + integer '7' AS RESULT; result ------------ 2001-10-05 (1 row)
	openGauss=# SELECT date '2001-09-28' + interval '1 hour' AS RESULT; result --------------------- 2001-09-28 01:00:00 (1 row)
	openGauss=# SELECT date '2001-09-28' + time '03:00' AS RESULT; result --------------------- 2001-09-28 03:00:00 (1 row)
	openGauss=# SELECT interval '1 day' + interval '1 hour' AS RESULT; result ---------------- 1 day 01:00:00 (1 row)
	openGauss=# SELECT timestamp '2001-09-28 01:00' + interval '23 hours' AS RESULT; result --------------------- 2001-09-29 00:00:00 (1 row)
	openGauss=# SELECT time '01:00' + interval '3 hours' AS RESULT; result ---------- 04:00:00 (1 row)

操作符	示例
-	openGauss=# SELECT date '2001-10-01' - date '2001-09-28' AS RESULT; result -------- 3 (1 row)
	openGauss=# SELECT date '2001-10-01' - integer '7' AS RESULT; result --------------------- 2001-09-24 00:00:00 (1 row)
	openGauss=# SELECT date '2001-09-28' - interval '1 hour' AS RESULT; result --------------------- 2001-09-27 23:00:00 (1 row)
	openGauss=# SELECT time '05:00' - time '03:00' AS RESULT; result ---------- 02:00:00 (1 row)
	openGauss=# SELECT time '05:00' - interval '2 hours' AS RESULT; result ---------- 03:00:00 (1 row)
	openGauss=# SELECT timestamp '2001-09-28 23:00' - interval '23 hours' AS RESULT; result --------------------- 2001-09-28 00:00:00 (1 row)
	openGauss=# SELECT interval '1 day' - interval '1 hour' AS RESULT; result ---------- 23:00:00 (1 row)
	openGauss=# SELECT timestamp '2001-09-29 03:00' - timestamp '2001-09-27 12:00' AS RESULT; result ---------------- 1 day 15:00:00 (1 row)
*	openGauss=# SELECT 900 * interval '1 second' AS RESULT; result ---------- 00:15:00 (1 row)
	openGauss=# SELECT 21 * interval '1 day' AS RESULT; result --------- 21 days (1 row)
	openGauss=# SELECT double precision '3.5' * interval '1 hour' AS RESULT; result ---------- 03:30:00 (1 row)

操作符	示例
/	openGauss=# SELECT interval '1 hour' / double precision '1.5' AS RESULT; result ---------- 00:40:00 (1 row)

时间/日期函数

- age(timestamp, timestamp)

  描述：将两个参数相减，并以年、月、日作为返回值。若相减值为负，则函数返回亦为负，入参可以都带timezone或都不带timezone。

  返回值类型：interval

  示例：

  ```
  openGauss=# SELECT age(timestamp '2001-04-10', timestamp '1957-06-13');
            age
  --------------------------
   43 years 9 mons 27 days
  (1 row)
  ```

- age(timestamp)

  描述：当前时间和参数相减，入参可以带或者不带timezone。

  返回值类型：interval

  示例：

  ```
  openGauss=# SELECT age(timestamp '1957-06-13');
            age
  --------------------------
   60 years 2 mons 18 days
  (1 row)
  ```

- clock_timestamp()

  描述：实时时钟的当前时间戳。volatile函数，每次扫描都会取最新的时间戳，因此在一次查询中每次调用结果不相同。

  返回值类型：timestamp with time zone

  示例：

  ```
  openGauss=# SELECT clock_timestamp();
         clock_timestamp
  -------------------------------
   2017-09-01 16:57:36.636205+08
  (1 row)
  ```

- current_date

  描述：当前日期。

  返回值类型：date

  示例：

  ```
  openGauss=# SELECT current_date;
     date
  ------------
   2017-09-01
  (1 row)
  ```

- current_time

  描述：当前时间。

返回值类型：time with time zone

示例：

```
openGauss=# SELECT current_time;
      timetz
-------------------
 16:58:07.086215+08
(1 row)
```

- current_timestamp

  描述：当前日期及时间。语句级别时间，同一个语句内返回结果不变。

  返回值类型：timestamp with time zone

  示例：

```
openGauss=# SELECT current_timestamp;
      pg_systimestamp
------------------------------
 2017-09-01 16:58:19.22173+08
(1 row)
```

- pg_systimestamp()

  描述：当前日期和时间(当前语句的开始)。

  返回值类型：timestamp with time zone

  示例：

```
openGauss=# SELECT pg_systimestamp();
       pg_systimestamp
-------------------------------
 2015-10-14 11:21:28.317367+08
(1 row)
```

- date_part(text, timestamp)

  描述：

  获取日期或者时间值中子域的值，例如年或者小时的值。

  等效于extract(field from timestamp)。

  timestamp类型：abstime、date、interval、reltime、time with time zone、time without time zone、timestamp with time zone、timestamp without time zone。

  返回值类型：double precision

  示例：

```
openGauss=# SELECT date_part('hour', timestamp '2001-02-16 20:38:40');
 date_part
-----------
        20
(1 row)
```

- timestamp_diff(text, timestamp, timestamp)

  描述：计算两个日期时间之间的差值，截取到参数text指定的精度。在兼容MY数据库模式下，该函数功能与**TIMESTAMPDIFF(unit , timestamp_expr1, timestamp_expr2)**相同。

  返回值类型：int64

  示例：

```
openGauss=# SELECT timestamp_diff('year','2018-01-01','2020-04-01');
 timestamp_diff
----------------
              2
(1 row)
openGauss=# SELECT timestamp_diff('month','2018-01-01','2020-04-01');
 timestamp_diff
```



```
----------------
           27
(1 row)
openGauss=# SELECT timestamp_diff('quarter','2018-01-01','2020-04-01');
 timestamp_diff
----------------
            9
(1 row)
openGauss=# SELECT timestamp_diff('week','2018-01-01','2020-04-01');
 timestamp_diff
----------------
          117
(1 row)
openGauss=# SELECT timestamp_diff('day','2018-01-01','2020-04-01');
 timestamp_diff
----------------
          821
(1 row)
openGauss=# SELECT timestamp_diff('hour','2018-01-01 10:10:10','2018-01-01 12:12:12');
 timestamp_diff
----------------
            2
(1 row)
openGauss=# SELECT timestamp_diff('minute','2018-01-01 10:10:10','2018-01-01 12:12:12');
 timestamp_diff
----------------
          122
(1 row)
openGauss=# SELECT timestamp_diff('second','2018-01-01 10:10:10','2018-01-01 10:12:12');
 timestamp_diff
----------------
          122
(1 row)
openGauss=# SELECT timestamp_diff('microsecond','2018-01-01 10:10:10','2018-01-01 10:12:12');
 timestamp_diff
----------------
     122000000
(1 row)
```

- date_part(text, interval)

  描述：获取月份的值。如果大于12，则取与12的模。等效于extract(field from timestamp)。

  返回值类型：double precision

  示例：

  ```
  openGauss=# SELECT date_part('month', interval '2 years 3 months');
   date_part
  -----------
           3
  (1 row)
  ```

- date_trunc(text, timestamp)

  描述：截取到参数text指定的精度。

  返回值类型：interval、timestamp with time zone、timestamp without time zone

  示例：

  ```
  openGauss=# SELECT date_trunc('hour', timestamp  '2001-02-16 20:38:40');
      date_trunc
  ---------------------
   2001-02-16 20:00:00
  (1 row)
  ```

- trunc(timestamp)

  描述：默认按天截取。

  示例：


```
openGauss=# SELECT trunc(timestamp  '2001-02-16
20:38:40');
                      trunc
---------------------
2001-02-16 00:00:00
(1 row)
```

- trunc(arg1, arg2)

  描述：截取到arg2指定的精度。

  arg1类型：interval、timestamp with time zone、timestamp without time zone

  arg2类型：text

  返回值类型：interval、timestamp with time zone、timestamp without time zone

  示例：

```
openGauss=# SELECT trunc(timestamp  '2001-02-16 20:38:40',
'hour');
                    trunc
---------------------
2001-02-16 20:00:00
(1 row)
```

- daterange(arg1, arg2)

  描述：获取时间边界信息。

  arg1类型：date

  arg2类型：date

  返回值类型：daterange

  示例：

```
openGauss=# select daterange('2000-05-06','2000-08-08');
      daterange
-------------------------
 [2000-05-06,2000-08-08)
(1 row)
```

- daterange(arg1, arg2, text)

  描述：获取时间边界信息。

  arg1类型：date

  arg2类型：date

  text类型：text

  返回值类型：daterange

  示例：

```
openGauss=# select daterange('2000-05-06','2000-08-08','[]');
      daterange
-------------------------
 [2000-05-06,2000-08-09)
(1 row)
```

- extract(field from timestamp)

  描述：获取小时的值。

  返回值类型：double precision

  示例：

```
openGauss=# SELECT extract(hour from timestamp '2001-02-16 20:38:40');
 date_part
-----------
```



```
      20
(1 row)
```

- extract(field from interval)

  描述：获取月份的值。如果大于12，则取与12的模。

  返回值类型：double precision

  示例：

```
openGauss=# SELECT extract(month from interval '2 years 3 months');
 date_part
-----------
         3
(1 row)
```

- isfinite(date)

  描述：测试是否为有效日期。

  返回值类型：Boolean

  示例：

```
openGauss=# SELECT isfinite(date '2001-02-16');
 isfinite
----------
 t
(1 row)
```

- isfinite(timestamp)

  描述：测试判断是否为有效时间。

  返回值类型：Boolean

  示例：

```
openGauss=# SELECT isfinite(timestamp '2001-02-16 21:28:30');
 isfinite
----------
 t
(1 row)
```

- isfinite(interval)

  描述：测试是否为有效区间。

  返回值类型：Boolean

  示例：

```
openGauss=# SELECT isfinite(interval '4 hours');
 isfinite
----------
 t
(1 row)
```

- justify_days(interval)

  描述：将时间间隔以月（30天为一月）为单位。

  返回值类型：interval

  示例：

```
openGauss=# SELECT justify_days(interval '35 days');
 justify_days
--------------
 1 mon 5 days
(1 row)
```

- justify_hours(interval)

  描述：将时间间隔以天（24小时为一天）为单位。

  返回值类型：interval

示例：

```
openGauss=# SELECT JUSTIFY_HOURS(INTERVAL '27 HOURS');
 justify_hours
---------------
 1 day 03:00:00
(1 row)
```

- justify_interval(interval)

  描述：结合justify_days和justify_hours，调整interval。

  返回值类型：interval

  示例：

  ```
  openGauss=# SELECT JUSTIFY_INTERVAL(INTERVAL '1 MON -1 HOUR');
   justify_interval
  ------------------
   29 days 23:00:00
  (1 row)
  ```

- localtime

  描述：当前时间。

  返回值类型：time

  示例：

  ```
  openGauss=# SELECT localtime AS RESULT;
      result
  ----------------
   16:05:55.664681
  (1 row)
  ```

- localtimestamp

  描述：当前日期及时间。

  返回值类型：timestamp

  示例：

  ```
  openGauss=# SELECT localtimestamp;
           timestamp
  ----------------------------
   2017-09-01 17:03:30.781902
  (1 row)
  ```

- now()

  描述：当前日期及时间。事务级别时间，同一个事务内返回结果相同。

  返回值类型：timestamp with time zone

  示例：

  ```
  openGauss=# SELECT now();
                now
  -------------------------------
   2017-09-01 17:03:42.549426+08
  (1 row)
  ```

- timenow()

  描述：当前日期及时间。

  返回值类型：timestamp with time zone

  示例：

  ```
  openGauss=# select timenow();
         timenow
  ------------------------
   2020-06-23 20:36:56+08
  (1 row)
  ```


- numtodsinterval(num, interval_unit)

  描述：将数字转换为interval类型。num为numeric类型数字，interval_unit为固定格式字符串（'DAY' | 'HOUR' | 'MINUTE' | 'SECOND'）。

  可以通过设置GUC参数IntervalStyle为oracle，兼容该函数在Oracle中的interval输出格式。

  示例：

  ```
  openGauss=# SELECT numtodsinterval(100, 'HOUR');
   numtodsinterval
  -----------------
   100:00:00
  (1 row)

  openGauss=# SET intervalstyle = oracle;
  SET
  openGauss=# SELECT numtodsinterval(100, 'HOUR');
         numtodsinterval
  -------------------------------
   +000000004 04:00:00.000000000
  (1 row)
  ```

- pg_sleep(seconds)

  描述：服务器线程延迟时间，单位为秒。注意，当数据库调用该函数时，会获取相应的事务快照，相当于一个长事务，如果入参时间过长可能导致数据库oldestxmin无法推进，影响表的回收和查询性能。

  返回值类型：void

  示例：

  ```
  openGauss=# SELECT pg_sleep(10);
   pg_sleep
  ----------

  (1 row)
  ```

- statement_timestamp()

  描述：当前日期和时间(当前语句的开始)。

  返回值类型：timestamp with time zone

  示例：

  ```
  openGauss=# SELECT statement_timestamp();
       statement_timestamp
  -------------------------------
   2017-09-01 17:04:39.119267+08
  (1 row)
  ```

- sysdate

  描述：当前日期及时间。

  返回值类型：timestamp

  示例：

  ```
  openGauss=# SELECT sysdate;
        sysdate
  ---------------------
   2017-09-01 17:04:49
  (1 row)
  ```

- timeofday()

  描述：当前日期及时间（像clock_timestamp，但是返回时为text）。

  返回值类型：text

  示例：


```
openGauss=# SELECT timeofday();
              timeofday
-------------------------------------
 Fri Sep 01 17:05:01.167506 2017 CST
(1 row)
```

- transaction_timestamp()

  描述：当前日期及时间，与current_timestamp等效。

  返回值类型：timestamp with time zone

  示例：

```
openGauss=# SELECT transaction_timestamp();
    transaction_timestamp
-------------------------------
 2017-09-01 17:05:13.534454+08
(1 row)
```

- add_months(d,n)

  描述：用于计算时间点d再加上n个月的时间。

  d：timestamp类型的值，以及可以隐式转换为timestamp类型的值。

  n：INTEGER类型的值，以及可以隐式转换为INTEGER类型的值。

  返回值类型：timestamp

  示例：

```
openGauss=# SELECT add_months(to_date('2017-5-29', 'yyyy-mm-dd'), 11) FROM sys_dummy;
    add_months
---------------------
 2018-04-29 00:00:00
(1 row)
```

- last_day(d)

  描述：用于计算时间点d当月最后一天的时间。

  返回值类型：timestamp

  示例：

```
openGauss=# select last_day(to_date('2017-01-01', 'YYYY-MM-DD')) AS cal_result;
    cal_result
---------------------
 2017-01-31 00:00:00
(1 row)
```

- next_day(x,y)

  描述：用于计算时间点x开始的下一个星期几（y）的时间。

  返回值类型：timestamp

  示例：

```
openGauss=# select next_day(timestamp '2017-05-25 00:00:00','Sunday')AS cal_result;
    cal_result
---------------------
 2017-05-28 00:00:00
(1 row)
```

- tinterval(abstime, abstime)

  描述：用两个绝对时间创建时间间隔。

  返回值类型：tinterval

  示例：

```
openGauss=# call tinterval(abstime 'May 10, 1947 23:59:12', abstime 'Mon May  1 00:30:30 1995');
                    tinterval
----------------------------------------------------------
 ["1947-05-10 23:59:12+08" "1995-05-01 00:30:30+08"]
(1 row)
```


- tintervalend(tinterval)
  描述：返回tinterval的结束时间。
  返回值类型：abstime
  示例：

```
openGauss=# select tintervalend('["Sep 4, 1983 23:59:12" "Oct4, 1983 23:59:12"]');
     tintervalend
------------------------
 1983-10-04 23:59:12+08
(1 row)
```

- tintervalrel(tinterval)
  描述：计算并返回tinterval的相对时间。
  返回值类型：reltime
  示例：

```
openGauss=# select tintervalrel('["Sep 4, 1983 23:59:12" "Oct4, 1983 23:59:12"]');
 tintervalrel
--------------
 1 mon
(1 row)
```

- smalldatetime_ge
  描述：判断是否第一个参数大于第二个参数。
  参数：smalldatetime, smalldatetime
  返回值类型：boolean
- smalldatetime_cmp
  描述：对比smalldatetime是否相等。
  参数：smalldatetime, smalldatetime
  返回值类型：integer
- smalldatetime_eq
  描述：对比smalldatetime是否相等。
  参数：smalldatetime, smalldatetime
  返回值类型：boolean
- smalldatetime_gt
  描述：判断是否第一个参数小于第二个参数。
  参数：smalldatetime, smalldatetime
  返回值类型：boolean
- smalldatetime_hash
  描述：计算timestamp对应的哈希值。
  参数：smalldatetime
  返回值类型：integer
- smalldatetime_in
  描述：输入timestamp。
  参数：cstring, oid, integer
  返回值类型：smalldatetime

- smalldatetime_larger
  描述：返回较大的timestamp。
  参数：smalldatetime, smalldatetime
  返回值类型：smalldatetime
- smalldatetime_le
  描述：判断是否第一个参数小于第二个参数。
  参数：smalldatetime, smalldatetime
  返回值类型：boolean
- smalldatetime_lt
  描述：判断是否第一个参数大于第二个参数。
  参数：smalldatetime, smalldatetime
  返回值类型：boolean
- smalldatetime_ne
  描述：比较两个timestamp是否不相等。
  参数：smalldatetime, smalldatetime
  返回值类型：boolean
- smalldatetime_out
  描述：timestamp转换为外部形式。
  参数：smalldatetime
  返回值类型：cstring
- smalldatetime_send
  描述：timestamp转换为二进制格式。
  参数：smalldatetime
  返回值类型：bytea
- smalldatetime_smaller
  描述：返回较小的一个smalldatetime。
  参数：smalldatetime, smalldatetime
  返回值类型：smalldatetime
- smalldatetime_to_abstime
  描述：smalldatetime转换为abstime。
  参数：smalldatetime
  返回值类型：abstime
- smalldatetime_to_time
  描述：smalldatetime转换为time。
  参数：smalldatetime
  返回值类型：time without time zone
- smalldatetime_to_timestamp
  描述：smalldatetime转换为timestamp。

参数：smalldatetime

返回值类型：timestamp without time zone

- smalldatetime_to_timestamptz

  描述：smalldatetime转换为timestamptz。

  参数：smalldatetime

  返回值类型：timestamp with time zone

- smalldatetime_to_varchar2

  描述：smalldatetime转换为varchar2。

  参数：smalldatetime

  返回值类型：character varying

  说明

  获取当前时间有多种方式，请根据实际业务从场景选择合适的接口：

  1. 以下接口按照当前事务的开始时刻返回值：

     ```
     CURRENT_DATE
     CURRENT_TIME
     CURRENT_TIME(precision)
     CURRENT_TIMESTAMP(precision)
     LOCALTIME
     LOCALTIMESTAMP
     LOCALTIME(precision)
     LOCALTIMESTAMP(precision)
     transaction_timestamp()
     now()
     ```

     其中CURRENT_TIME和CURRENT_TIMESTAMP(precision)传递带有时区的值；LOCALTIME和LOCALTIMESTAMP传递的值不带时区。CURRENT_TIME、LOCALTIME和LOCALTIMESTAMP可以指定精度参数，这会导致结果在秒字段中四舍五入到小数位数。如果没有精度参数，结果将被给予所能得到的全部精度。

     因为这些函数全部都按照当前事务的开始时刻返回结果，所以它们的值在事务运行的整个期间内都不改变。可以认为这是一个特性：目的是为了允许一个事务在“当前”时间上有一致的概念，这样在同一个事务里的多个修改可以保持同样的时间戳。

     其中transaction_timestamp()等价于CURRENT_TIMESTAMP(precision)，表示当前语句所在事务的开启时间。now()等效于transaction_timestamp()。

  2. 以下接口返回当前语句开始时间：

     ```
     statement_timestamp()
     ```

     statement_timestamp()返回当前语句的开始时刻（更准确的说是收到客户端最后一条命令的时间）。statement_timestamp()和transaction_timestamp()在一个事务的第一条命令期间返回值相同，但是在随后的命令中却不一定相同。

  3. 以下接口返回函数被调用时的真实当前时间：

     ```
     clock_timestamp()
     timeofday()
     ```

     clock_timestamp()返回真正的当前时间，因此它的值甚至在同一条SQL 命令中都会变化。timeofday()和clock_timestamp()相似，timeofday()也返回真实的当前时间，但是它的结果是一个格式化的text串，而不是timestamp with time zone值。

表7-30显示了可以用于截断日期和时间值的模板。

表 7-30 用于日期/时间截断的模式

类别	模式	描述
微秒	MICROSECON	截断日期/时间，精确到微秒（000000 - 999999）
	US	
	USEC	
	USECOND	
毫秒	MILLISECON	截断日期/时间，精确到毫秒（000 - 999）
	MS	
	MSEC	
	MSECOND	
秒	S	截断日期/时间，精确到秒（00 - 59）
	SEC	
	SECOND	
分钟	M	截断日期/时间，精确到分钟（00 - 59）
	MI	
	MIN	
	MINUTE	
小时	H	截断日期/时间，精确到小时（00 - 23）
	HH	
	HOUR	
	HR	
天	D	截断日期/时间，精确到天（01-01 - 12-31）
	DAY	
	DD	
	DDD	
	J	
周	W	截断日期/时间，精确到周（本周的第一天）
	WEEK	
月	MM	截断日期/时间，精确到月（本月的第一天）
	MON	
	MONTH	

类别	模式	描述
季度	Q	截断日期/时间，精确到季度（本季度的第一天）
	QTR	
	QUARTER	
年	Y	截断日期/时间，精确到年（本年的第一天）
	YEAR	
	YR	
	YYYY	
十年	DEC	截断日期/时间，精确到十年（本十年的第一天）
	DECADE	
世纪	C	截断日期/时间，精确到世纪（本世纪的第一天）
	CC	
	CENT	
	CENTURY	
千年	MIL	截断日期/时间，精确到千年（本千年的第一天）
	MILLENNIA	
	MILLENNIUM	

TIMESTAMPDIFF

- **TIMESTAMPDIFF(***unit , timestamp_expr1, timestamp_expr2***)**

  timestampdiff函数是计算两个日期时间之间(timestamp_expr2-timestamp_expr1)的差值，并以unit形式范围结果。timestamp_expr1，timestamp_expr2必须是一个timestamp、timestamptz、date类型的值表达式。unit表示的是两个日期差的单位。

  说明

  该函数仅在GaussDB兼容MySQL类型时（即dbcompatibility = 'MYSQL'）有效，其他类型不支持该函数。等效于**timestamp_diff(text, timestamp, timestamp)**。

- year

  年份。

  ```
  openGauss=# SELECT TIMESTAMPDIFF(YEAR, '2018-01-01', '2020-01-01');
   timestamp_diff
  ----------------
                2
  (1 row)
  ```

- quarter

  季度。

  ```
  openGauss=# SELECT TIMESTAMPDIFF(QUARTER, '2018-01-01', '2020-01-01');
   timestamp_diff
  ```



```
----------------
               8
(1 row)
```

- month

  月份。

  ```
  openGauss=# SELECT TIMESTAMPDIFF(MONTH, '2018-01-01', '2020-01-01');
   timestamp_diff
  ----------------
                24
  (1 row)
  ```

- week

  星期。

  ```
  openGauss=# SELECT TIMESTAMPDIFF(WEEK, '2018-01-01', '2020-01-01');
   timestamp_diff
  ----------------
               104
  (1 row)
  ```

- day

  天。

  ```
  openGauss=# SELECT TIMESTAMPDIFF(DAY, '2018-01-01', '2020-01-01');
   timestamp_diff
  ----------------
               730
  (1 row)
  ```

- hour

  小时。

  ```
  openGauss=# SELECT TIMESTAMPDIFF(HOUR, '2020-01-01 10:10:10', '2020-01-01 11:11:11');
   timestamp_diff
  ----------------
                 1
  (1 row)
  ```

- minute

  分钟。

  ```
  openGauss=# SELECT TIMESTAMPDIFF(MINUTE, '2020-01-01 10:10:10', '2020-01-01 11:11:11');
   timestamp_diff
  ----------------
                61
  (1 row)
  ```

- second

  秒。

  ```
  openGauss=# SELECT TIMESTAMPDIFF(SECOND, '2020-01-01 10:10:10', '2020-01-01 11:11:11');
   timestamp_diff
  ----------------
              3661
  (1 row)
  ```

- microseconds

  秒域（包括小数部分）乘以1,000,000。

  ```
  openGauss=# SELECT TIMESTAMPDIFF(MICROSECOND, '2020-01-01 10:10:10.000000', '2020-01-01
  10:10:10.111111');
   timestamp_diff
  ----------------
              111111
  (1 row)
  ```

- timestamp_expr含有时区


```
openGauss=# SELECT TIMESTAMPDIFF(HOUR,'2020-05-01 10:10:10-01','2020-05-01 10:10:10-03');
 timestamp_diff
----------------
              2
(1 row)
```

EXTRACT

- **EXTRACT(*field* FROM *source*)**

  extract函数从日期或时间的数值里抽取子域，比如年、小时等。source必须是一个timestamp、time或interval类型的值表达式（类型为date的表达式转换为timestamp，因此也可以用）。field是一个标识符或者字符串，它指定从源数据中抽取的域。extract函数返回类型为double precision的数值。field的取值范围如下所示。

- century

  世纪。

  第一个世纪从0001-01-01 00:00:00 AD开始。这个定义适用于所有使用阳历的国家。没有0世纪，直接从公元前1世纪到公元1世纪。

  示例：

  ```
  openGauss=# SELECT EXTRACT(CENTURY FROM TIMESTAMP '2000-12-16 12:21:13');
   date_part
  -----------
         20
  (1 row)
  ```

- day
  - 如果source为timestamp，表示月份里的日期（1-31）。

    ```
    openGauss=# SELECT EXTRACT(DAY FROM TIMESTAMP '2001-02-16 20:38:40');
     date_part
    -----------
          16
    (1 row)
    ```

  - 如果source为interval，表示天数。

    ```
    openGauss=# SELECT EXTRACT(DAY FROM INTERVAL '40 days 1 minute');
     date_part
    -----------
          40
    (1 row)
    ```

- decade

  年份除以10。

  ```
  openGauss=# SELECT EXTRACT(DECADE FROM TIMESTAMP '2001-02-16 20:38:40');
   date_part
  -----------
        200
  (1 row)
  ```

- dow

  每周的星期几，星期天（0）到星期六（6）。

  ```
  openGauss=# SELECT EXTRACT(DOW FROM TIMESTAMP '2001-02-16 20:38:40');
   date_part
  -----------
          5
  (1 row)
  ```

- doy

  一年的第几天（1~365/366）。

  ```
  openGauss=# SELECT EXTRACT(DOY FROM TIMESTAMP '2001-02-16 20:38:40');
   date_part
  ```



```
-----------
      47
(1 row)
```

- epoch
  - 如果source为timestamp with time zone，表示自1970-01-01 00:00:00-00 UTC以来的秒数（结果可能是负数）；

    如果source为date和timestamp，表示自1970-01-01 00:00:00-00当地时间以来的秒数；

    如果source为interval，表示时间间隔的总秒数。

    ```
    openGauss=# SELECT EXTRACT(EPOCH FROM TIMESTAMP WITH TIME ZONE '2001-02-16 20:38:40.12-08');
      date_part
    --------------
     982384720.12
    (1 row)
    openGauss=# SELECT EXTRACT(EPOCH FROM INTERVAL '5 days 3 hours');
     date_part
    -----------
        442800
    (1 row)
    ```

  - 将epoch值转换为时间戳的方法。

    ```
    openGauss=# SELECT TIMESTAMP WITH TIME ZONE 'epoch' + 982384720.12 * INTERVAL '1 second' AS RESULT;
              result
    ---------------------------
     2001-02-17 12:38:40.12+08
    (1 row)
    ```

- hour

  小时域（0-23）。

  ```
  openGauss=# SELECT EXTRACT(HOUR FROM TIMESTAMP '2001-02-16 20:38:40');
   date_part
  -----------
         20
  (1 row)
  ```

- isodow

  一周的第几天（1-7）。

  星期一为1，星期天为7。

  说明

  除了星期天外，都与dow相同。

  ```
  openGauss=# SELECT EXTRACT(ISODOW FROM TIMESTAMP '2001-02-18 20:38:40');
   date_part
  -----------
          7
  (1 row)
  ```

- isoyear

  日期中的ISO 8601标准年（不适用于间隔）。

  每个带有星期一开始的周中包含1月4日的ISO年，所以在年初的1月或12月下旬的ISO年可能会不同于阳历的年。详细信息请参见后续的week描述。

  ```
  openGauss=# SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-01');
   date_part
  -----------
       2005
  (1 row)
  openGauss=# SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-02');
   date_part
  ```



```
-----------
      2006
(1 row)
```

- microseconds

  秒域（包括小数部分）乘以1,000,000。

```
openGauss=# SELECT EXTRACT(MICROSECONDS FROM TIME '17:12:28.5');
 date_part
-----------
  28500000
(1 row)
```

- millennium

  千年。

  20世纪（19xx年）里面的年份在第二个千年里。第三个千年从2001年1月1日零时开始。

```
openGauss=# SELECT EXTRACT(MILLENNIUM FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
         3
(1 row)
```

- milliseconds

  秒域（包括小数部分）乘以1000。请注意它包括完整的秒。

```
openGauss=# SELECT EXTRACT(MILLISECONDS FROM TIME '17:12:28.5');
 date_part
-----------
     28500
(1 row)
```

- minute

  分钟域（0-59）。

```
openGauss=# SELECT EXTRACT(MINUTE FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
        38
(1 row)
```

- month

  如果source为timestamp，表示一年里的月份数（1-12）。

```
openGauss=# SELECT EXTRACT(MONTH FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
         2
(1 row)
```

  如果source为interval，表示月的数目，然后对12取模（0-11）。

```
openGauss=# SELECT EXTRACT(MONTH FROM INTERVAL '2 years 13 months');
 date_part
-----------
         1
(1 row)
```

- quarter

  该天所在的该年的季度（1-4）。

```
openGauss=# SELECT EXTRACT(QUARTER FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
         1
(1 row)
```

- second

  秒域，包括小数部分（0-59）。


```
openGauss=# SELECT EXTRACT(SECOND FROM TIME '17:12:28.5');
 date_part
-----------
      28.5
(1 row)
```

- timezone

  与UTC的时区偏移量，单位为秒。正数对应UTC东边的时区，负数对应UTC西边的时区。

- timezone_hour

  时区偏移量的小时部分。

- timezone_minute

  时区偏移量的分钟部分。

- week

  该天在所在的年份里是第几周。ISO 8601定义一年的第一周包含该年的一月四日（ISO-8601 的周从星期一开始）。换句话说，一年的第一个星期四在第一周。

  在ISO定义里，一月的头几天可能是前一年的第52或者第53周，十二月的后几天可能是下一年第一周。比如，2005-01-01是2004年的第53周，而2006-01-01是2005年的第52周，2012-12-31是2013年的第一周。建议isoyear字段和week一起使用以得到一致的结果。

```
openGauss=# SELECT EXTRACT(WEEK FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
         7
(1 row)
```

- year

  年份域。

```
openGauss=# SELECT EXTRACT(YEAR FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
      2001
(1 row)
```

date_part

date_part函数是在传统的Ingres函数的基础上制作的（该函数等效于SQL标准函数extract）：

date_part('*field*', *source*)

这里的field参数必须是一个字符串，而不是一个名称。有效的field与extract一样，详细信息请参见**EXTRACT**。

示例：

```
openGauss=# SELECT date_part('day', TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
        16
(1 row)
openGauss=# SELECT date_part('hour', INTERVAL '4 hours 3 minutes');
 date_part
-----------
         4
(1 row)
```

表7-31显示了可以用于格式化日期和时间值的模板。

表 7-31 用于日期/时间格式化的模式

类别	模式	描述
小时	HH	一天的小时数（01-12）
	HH12	一天的小时数（01-12）
	HH24	一天的小时数（00-23）
分钟	MI	分钟（00-59）
秒	SS	秒（00-59）
	FF	微秒（000000-999999）
	FF1	微秒（0-9）
	FF2	微秒（00-99）
	FF3	微秒（000-999）
	FF4	微秒（0000-9999）
	FF5	微秒（00000-99999）
	FF6	微秒（000000-999999）
	SSSSS	午夜后的秒（0-86399）
上、下午	AM或A.M.	上午标识
	PM或P.M.	下午标识
年	Y,YYY	带逗号的年（4和更多位）
	SYYYY	公元前四位年
	YYYY	年（4和更多位）
	YYY	年的后三位
	YY	年的后两位
	Y	年的最后一位
	IYYY	ISO年（4位或更多位）
	IYY	ISO年的最后三位
	IY	ISO年的最后两位
	I	ISO年的最后一位
	RR	年的后两位（可在21世纪存储20世纪的年份）
	RRRR	可接收4位年或两位年。若是两位，则和RR的返回值相同，若是四位，则和YYYY相同。
	• BC或B.C. • AD或A.D.	纪元标识。BC（公元前），AD（公元后）。

类别	模式	描述
月	MONTH	全长大写月份名（空白填充为9字符）
	MON	大写缩写月份名（3字符）
	MM	月份数（01-12）
	RM	罗马数字的月份（I-XII； I=JAN）（大写）
天	DAY	全长大写日期名（空白填充为9字符）
	DY	缩写大写日期名（3字符）
	DDD	一年里的日（001-366）
	DD	一个月里的日（01-31）
	D	一周里的日（1-7； 周日是 1）
周	W	一个月里的周数（1-5）（第一周从该月第一天开始）
	WW	一年里的周数（1-53）（第一周从该年的第一天开始）
	IW	ISO一年里的周数（第一个星期四在第一周里）
世纪	CC	世纪（2位）（21 世纪从 2001-01-01 开始）
儒略日	J	儒略日（自公元前 4712 年 1 月 1 日来的天数）
季度	Q	季度

说明

上表中RR计算年的规则如下：

- 输入的两位年份在00~49之间：
  当前年份的后两位在00~49之间，返回值年份的前两位和当前年份的前两位相同；
  当前年份的后两位在50~99之间，返回值年份的前两位是当前年份的前两位加1。
- 输入的两位年份在50~99之间：
  当前年份的后两位在00~49之间，返回值年份的前两位是当前年份的前两位减1；
  当前年份的后两位在50~99之间，返回值年份的前两位和当前年份的前两位相同。

7.5.9 类型转换函数

类型转换函数

- cash_words(money)
  描述：类型转换函数，将money转换成text。
  示例：

```
openGauss=# SELECT cash_words('1.23');
        cash_words
```

```
-----------------------------------
 One dollar and twenty three cents
(1 row)
```

- cast(x as y)

  描述：类型转换函数，将x转换成y指定的类型。

  示例：

```
openGauss=# SELECT cast('22-oct-1997' as timestamp);
      timestamp
---------------------
 1997-10-22 00:00:00
(1 row)
```

- hextoraw(text)

  描述：将一个十六进制构成的字符串转换为raw类型。

  返回值类型：raw

  示例：

```
openGauss=# SELECT hextoraw('7D');
 hextoraw
----------
 7D
(1 row)
```

- numtoday(numeric)

  描述：将数字类型的值转换为指定格式的时间戳。

  返回值类型：timestamp

  示例：

```
openGauss=# SELECT numtoday(2);
 numtoday
----------
 2 days
(1 row)
```

- rawtohex(string)

  描述：将一个二进制构成的字符串转换为十六进制的字符串。

  结果为输入字符的ACSII码，以十六进制表示。

  返回值类型：varchar

  示例：

```
openGauss=# SELECT rawtohex('1234567');
    rawtohex
----------------
 31323334353637
(1 row)
```

- to_bigint(varchar)

  描述：将字符类型转换为bigint类型。

  返回值类型：bigint

  示例：

```
openGauss=# SELECT to_bigint('123364545554455');
    to_bigint
----------------
 123364545554455
(1 row)
```

- to_char(datetime/interval [, fmt])

  描述：将一个DATE、TIMESTAMP、TIMESTAMP WITH TIME ZONE或者TIMESTAMP WITH LOCAL TIME ZONE类型的DATETIME或者INTERVAL值按照fmt指定的格式转换为VARCHAR类型。

- 可选参数fmt可以为以下几类：日期、时间、星期、季度和世纪。每类都可以有不同的模板，模板之间可以合理组合，常见的模板有：HH、MI、SS、YYYY、MM、DD。
- 模板可以有修饰词，常用的修饰词是FM，可以用来抑制前导的零或尾随的空白。

返回值类型：text

示例：

```
openGauss=# SELECT to_char(current_timestamp,'HH12:MI:SS');
 to_char
----------
 10:19:26
(1 row)
openGauss=# SELECT to_char(current_timestamp,'FMHH12:FMMI:FMSS');
 to_char
----------
 10:19:46
(1 row)
```

- to_char(double precision/real, text)

  描述：将浮点类型的值转换为指定格式的字符串。

  返回值类型：text

  示例：

```
openGauss=# SELECT to_char(125.8::real, '999D99');
 to_char
---------
  125.80
(1 row)
```

- to_char(numeric/smallint/integer/bigint/double precision/real[, fmt])

  描述：将一个整型或者浮点类型的值转换为指定格式的字符串。

  - 可选参数fmt可以为以下几类：十进制字符、“分组”符、正负号和货币符号，每类都可以有不同的模板，模板之间可以合理组合，常见的模板有：9、0、,（千分隔符）、.（小数点）。
  - 模板可以有类似FM的修饰词，但FM不抑制由模板0指定而输出的0。
  - 要将整型类型的值转换成对应16进制值的字符串，使用模板X或x。

  返回值类型：text

  示例：

```
openGauss=# SELECT to_char(1485,'9,999');
 to_char
---------
  1,485
(1 row)
openGauss=# SELECT to_char( 1148.5,'9,999.999');
  to_char
------------
  1,148.500
(1 row)
openGauss=# SELECT to_char(148.5,'990999.909');
   to_char
-------------
   0148.500
(1 row)
openGauss=# SELECT to_char(123,'XXX');
 to_char
---------
  7B
(1 row)
```


- to_char(interval, text)

  描述：将时间间隔类型的值转换为指定格式的字符串。

  返回值类型：text

  示例：

```
openGauss=# SELECT to_char(interval '15h 2m 12s', 'HH24:MI:SS');
 to_char
----------
 15:02:12
(1 row)
```

- to_char(integer, text)

  描述：将整数类型的值转换为指定格式的字符串。

  返回值类型：text

  示例：

```
openGauss=# SELECT to_char(125, '999');
 to_char
---------
  125
(1 row)
```

- to_char(numeric, text)

  描述：将数字类型的值转换为指定格式的字符串。

  返回值类型：text

  示例：

```
openGauss=# SELECT to_char(-125.8, '999D99S');
 to_char
---------
 125.80-
(1 row)
```

- to_char (string)

  描述：将CHAR、VARCHAR、VARCHAR2、CLOB类型转换为VARCHAR类型。

  如使用该函数对CLOB类型进行转换，且待转换CLOB类型的值超出目标类型的范围，则返回错误。

  返回值类型：text

  示例：

```
openGauss=# SELECT to_char('01110');
 to_char
---------
 01110
(1 row)
```

- to_nvarchar2

  描述：转换为nvarchar2类型。

  参数：numeric

  返回值类型：nvarchar2

- to_char(timestamp, text)

  描述：将时间戳类型的值转换为指定格式的字符串。

  返回值类型：text

  示例：

```
openGauss=# SELECT to_char(current_timestamp, 'HH12:MI:SS');
 to_char
----------
```



```
 10:55:59
(1 row)
```

说明

to_char函数对于错误的fmt会原样输出，如fmt为FF10，会匹配到FF1进行格式化输出，然后原样输出0。

- to_clob(char/nchar/varchar/nvarchar/varchar2/nvarchar2/text/raw)

  描述：将RAW类型或者文本字符集类型CHAR、NCHAR、VARCHAR、VARCHAR2、NVARCHAR2、TEXT转成CLOB类型。

  返回值类型：clob

  示例：

```
openGauss=# SELECT to_clob('ABCDEF'::RAW(10));
 to_clob
---------
 ABCDEF
(1 row)
openGauss=# SELECT to_clob('hello111'::CHAR(15));
 to_clob
----------
 hello111
(1 row)
openGauss=# SELECT to_clob('gauss123'::NCHAR(10));
 to_clob
----------
 gauss123
(1 row)
openGauss=# SELECT to_clob('gauss234'::VARCHAR(10));
 to_clob
----------
 gauss234
(1 row)
openGauss=# SELECT to_clob('gauss345'::VARCHAR2(10));
 to_clob
----------
 gauss345
(1 row)
openGauss=# SELECT to_clob('gauss456'::NVARCHAR2(10));
 to_clob
----------
 gauss456
(1 row)
openGauss=# SELECT to_clob('World222!'::TEXT);
  to_clob
-----------
 World222!
(1 row)
```

- to_date(text)

  描述：将文本类型的值转换为指定格式的时间戳。

  - 格式一：无分隔符日期，如20150814，需要包括完整的年月日。
  - 格式二：带分隔符日期，如2014-08-14，分隔符可以是单个任意非数字字符。

  返回值类型：timestamp without time zone

  示例：

```
openGauss=# SELECT to_date('2015-08-14');
      to_date
---------------------
 2015-08-14 00:00:00
(1 row)
```


- to_date(text, text)

  描述：将字符串类型的值转换为指定格式的日期。

  返回值类型：timestamp without time zone

  示例：

```
openGauss=# SELECT to_date('05 Dec 2000', 'DD Mon YYYY');
      to_date
---------------------
 2000-12-05 00:00:00
(1 row)
```

- to_number (expr [, fmt])

  描述：将expr按指定格式转换为一个NUMBER类型的值。

  类型转换格式请参考**表7-32**。

  转换十六进制字符串为十进制数字时，最多支持16个字节的十六进制字符串转换为无符号数。

  转换十六进制字符串为十进制数字时，格式字符串中不允许出现除'x'或'X'以外的其他字符，否则报错。

  返回值类型：number

  示例：

```
openGauss=# SELECT to_number('12,454.8-', '99G999D9S');
 to_number
-----------
  -12454.8
(1 row)
```

- to_number(text, text)

  描述：将字符串类型的值转换为指定格式的数字。

  返回值类型：numeric

  示例：

```
openGauss=# SELECT to_number('12,454.8-', '99G999D9S');
 to_number
-----------
  -12454.8
(1 row)
```

- to_timestamp(double precision)

  描述：把UNIX纪元转换成时间戳。

  返回值类型：timestamp with time zone

  示例：

```
openGauss=# SELECT to_timestamp(1284352323);
      to_timestamp
------------------------
 2010-09-13 12:32:03+08
(1 row)
```

- to_timestamp(string [,fmt])

  描述：将字符串string按fmt指定的格式转换成时间戳类型的值。不指定fmt时，按参数nls_timestamp_format所指定的格式转换。

  GaussDB的to_timestamp中，

  – 如果输入的年份YYYY=0，系统报错。
  – 如果输入的年份YYYY<0，在fmt中指定SYYYY，则正确输出公元前绝对值n的年份。

  fmt中出现的字符必须与日期/时间格式化的模式相匹配，否则报错。

返回值类型：timestamp without time zone

示例：

```
openGauss=# SHOW nls_timestamp_format;
   nls_timestamp_format
----------------------------
 DD-Mon-YYYY HH:MI:SS.FF AM
(1 row)

openGauss=# SELECT to_timestamp('12-sep-2014');
   to_timestamp
---------------------
 2014-09-12 00:00:00
(1 row)
openGauss=# SELECT to_timestamp('12-Sep-10 14:10:10.123000','DD-Mon-YY HH24:MI:SS.FF');
      to_timestamp
--------------------------
 2010-09-12 14:10:10.123
(1 row)
openGauss=# SELECT to_timestamp('-1','SYYYY');
      to_timestamp
-----------------------
 0001-01-01 00:00:00 BC
(1 row)
openGauss=# SELECT to_timestamp('98','RR');
   to_timestamp
---------------------
 1998-01-01 00:00:00
(1 row)
openGauss=# SELECT to_timestamp('01','RR');
   to_timestamp
---------------------
 2001-01-01 00:00:00
(1 row)
```

- to_timestamp(text, text)

  描述：将字符串类型的值转换为指定格式的时间戳。

  返回值类型：timestamp

  示例：

```
openGauss=# SELECT to_timestamp('05 Dec 2000', 'DD Mon YYYY');
   to_timestamp
---------------------
 2000-12-05 00:00:00
(1 row)
```

表 7-32 数值格式化的模板模式

模式	描述
9	带有指定数值位数的值。
0	带前导零的值。
.（句点）	小数点。
,（逗号）	分组（千）分隔符。
PR	尖括号内负值。
S	带符号的数值（使用区域设置）。
L	货币符号（使用区域设置）。

模式	描述
D	小数点（使用区域设置）。
G	分组分隔符（使用区域设置）。
MI	在指明的位置的负号（如果数字 < 0）。
PL	在指明的位置的正号（如果数字 > 0）。
SG	在指明的位置的正/负号。
RN	罗马数字（输入在 1 和 3999 之间）。
TH或th	序数后缀。
V	移动指定位（小数）。
x或X	16进制转换10进制标识符。

- abstime_text
  描述：将abstime类型转为text类型输出。
  参数：abstime
  返回值类型：text
- abstime_to_smalldatetime
  描述：将abstime类型转为smalldatatime类型。
  参数：abstime
  返回值类型：smalldatetime
- bigint_tid
  描述：将bigint转为tid。
  参数：bigint
  返回值类型：tid
- bool_int1
  描述：将bool转为int1。
  参数：boolean
  返回值类型：tinyint
- bool_int2
  描述：将bool转为int2。
  参数：boolean
  返回值类型：smallint
- bool_int8
  描述：将bool转为int8。
  参数：boolean
  返回值类型：bigint
- bpchar_date
  描述：将字符串转为日期。

参数：character
返回值类型：date

- bpchar_float4
  描述：将字符串转为float4。
  参数：character
  返回值类型：real
- bpchar_float8
  描述：将字符串转为float8。
  参数：character
  返回值类型：double precision
- bpchar_int4
  描述：将字符串转为int4。
  参数：character
  返回值类型：integer
- bpchar_int8
  描述：将字符串转为int8。
  参数：character
  返回值类型：bigint
- bpchar_numeric
  描述：将字符串转为numeric。
  参数：character
  返回值类型：numeric
- bpchar_timestamp
  描述：将字符串转为时间戳。
  参数：character
  返回值类型：timestamp without time zone
- bpchar_to_smalldatetime
  描述：将字符串转为smalldatetime。
  参数：character
  返回值类型：smalldatetime
- complex_array_in
  描述：将外部complex_array类型转化为内部anyarray数组类型。
  参数：cstring, oid, int2vector
  返回值类型：anyarray
- date_bpchar
  描述：将date类型转换为bpchar类型。
  参数：date
  返回值类型：character
- date_text
  描述：将date类型转换为text类型。

参数：date
返回值类型：text

- date_varchar
  描述：将date类型转换为varchar类型。
  参数：date
  返回值类型：character varying
- f4toi1
  描述：把float4类型强转为uint8类型。
  参数：real
  返回值类型：tinyint
- f8toi1
  描述：把float8类型强转为uint8类型。
  参数：double precision
  返回值类型：tinyint
- float4_bpchar
  描述：float4转换为bpchar。
  参数：real
  返回值类型：character
- float4_text
  描述：float4转换为text。
  参数：real
  返回值类型：text
- float4_varchar
  描述：float4转换为varchar。
  参数：real
  返回值类型：character varying
- float8_bpchar
  描述：float8转换为bpchar。
  参数：double precision
  返回值类型：character
- float8_interval
  描述：float8转换为interval。
  参数：double precision
  返回值类型：interval
- float8_text
  描述：float8转换为text。
  参数：double precision
  返回值类型：text
- float8_varchar
  描述：float8转换为varchar。

参数：double precision
返回值类型：character varying

- i1tof4
  描述：uint8转换为float4。
  参数：tinyint
  返回值类型：real
- i1tof8
  描述：uint8转换为float8。
  参数：tinyint
  返回值类型：double precision
- i1toi2
  描述：uint8转换为int16。
  参数：tinyint
  返回值类型：smallint
- i1toi4
  描述：uint8转换为int32。
  参数：tinyint
  返回值类型：integer
- i1toi8
  描述：uint8转换为int64。
  参数：tinyint
  返回值类型：bigint
- i2toi1
  描述：int16转换为uint8。
  参数：smallint
  返回值类型：tinyint
- i4toi1
  描述：int32转换为uint8。
  参数：integer
  返回值类型：tinyint
- i8toi1
  描述：int64转换为uint8。
  参数：bigint
  返回值类型：tinyint
- int1_avg_accum
  描述：将第二个uint8类型参数，加入到第一个参数中，一个参数为bigint类型数组。
  参数：bigint[], tinyint
  返回值类型：bigint[]
- int1_bool
  描述：uint8转换为bool。

参数：tinyint
返回值类型：boolean

- int1_bpchar
  描述：uint8转换为bpchar。
  参数：tinyint
  返回值类型：character
- int1_mul_cash
  描述：返回一个int8类型参数和一个cash类型参数的乘积，返回值为cash类型。
  参数：tinyint, money
  返回值类型：money
- int1_numeric
  描述：uint8转换为numeric。
  参数：tinyint
  返回值类型：numeric
- int1_nvarchar2
  描述：uint8转换为nvarchar2。
  参数：tinyint
  返回值类型：nvarchar2
- int1_text
  描述：uint8转换为text。
  参数：tinyint
  返回值类型：text
- int1_varchar
  描述：uint8转换为varchar。
  参数：tinyint
  返回值类型：character varying
- int1in
  描述：字符串转化为无符号一字节整数。
  参数：cstring
  返回值类型：tinyint
- int1out
  描述：无符号一字节整数转化为字符串。
  返回值类型：cstring
- int1up
  描述：输入整数转化为无符号一字节整数。
  参数：tinyint
  返回值类型：tinyint
- int2_bool
  描述：将有符号二字节整数转化为bool型。
  参数：smallint

返回值类型：boolean

- int2_bpchar
  描述：将有符号二字节整数转化为BpChar。
  参数：smallint
  返回值类型：character
- int2_text
  描述：有符号二字节整数转化为text类型。
  参数：smallint
  返回值类型：text
- int2_varchar
  描述：有符号二字节整数转化为varchar类型。
  参数：smallint
  返回值类型：character varying
- int4_bpchar
  描述：有符号四字节整数转化为bpchar。
  参数：integer
  返回值类型：character
- int4_text
  描述：有符号四字节整数转化为text类型。
  参数：integer
  返回值类型：text
- int4_varchar
  描述：有符号四字节整数转化为varchar。
  参数：integer
  返回值类型：character varying
- int8_bool
  描述：有符号八字节整数转化为bool。
  参数：bigint
  返回值类型：boolean
- int8_bpchar
  描述：有符号八字节整数转化为bpchar。
  参数：bigint
  返回值类型：character
- int8_text
  描述：有符号八字节整数转化为text类型。
  参数：bigint
  返回值类型：text
- int8_varchar
  描述：有符号八字节整数转化为varchar。
  参数：bigint

返回值类型：character varying

- intervaltonum
  描述：将内部数据类型日期转化为numeric类型。
  参数：interval
  返回值类型：numeric
- numeric_bpchar
  描述：numeric 转化为bpchar。
  参数：numeric
  返回值类型：character
- numeric_int1
  描述：numeric 转化为有符号1字节整数。
  参数：numeric
  返回值类型：tinyint
- numeric_text
  描述：numeric 转化为text。
  参数：numeric
  返回值类型：text
- numeric_varchar
  描述：numeric 转化为varchar。
  参数：numeric
  返回值类型：character varying
- nvarchar2in
  描述：将c字符串转化为varchar。
  参数：cstring, oid, integer
  返回值类型：nvarchar2
- nvarchar2out
  描述：将text转化为c字符串。
  参数：nvarchar2
  返回值类型：cstring
- nvarchar2send
  描述：将varchar转化为二进制。
  参数：nvarchar2
  返回值类型：bytea
- oidvectorin_extend
  描述：将字符串转化为oidvector。
  参数：cstring
  返回值类型：oidvector_extend
- oidvectorout_extend
  描述：将oidvector转化为字符串。
  参数：oidvector_extend

返回值类型：cstring

- oidvectorsend_extend
  描述：将oidvector转化为字符串。
  参数：oidvector_extend
  返回值类型：bytea
- reltime_text
  描述：reltime转换为text。
  参数：reltime
  返回值类型：text
- text_date
  描述：text类型转换为date类型。
  参数：text
  返回值类型：date
- text_float4
  描述：text类型转换为float4类型。
  参数：text
  返回值类型：real
- text_float8
  描述：text类型转换为float8类型。
  参数：text
  返回值类型：double precision
- text_int1
  描述：text类型转换为int1类型。
  参数：text
  返回值类型：tinyint
- text_int2
  描述：text类型转换为int2类型。
  参数：text
  返回值类型：smallint
- text_int4
  描述：text类型转换为int4类型。
  参数：text
  返回值类型：integer
- text_int8
  描述：text类型转换为int8类型。
  参数：text
  返回值类型：bigint
- text_numeric
  描述：text类型转换为numeric类型。
  参数：text

返回值类型：numeric

- text_timestamp
  描述：text类型转换为timestamp类型。
  参数：text
  返回值类型：timestamp without time zone
- time_text
  描述：time类型转换为text类型。
  参数：time without time zone
  返回值类型：text
- timestamp_text
  描述：timestamp类型转换为text类型。
  参数：timestamp without time zone
  返回值类型：text
- timestamp_to_smalldatetime
  描述：timestamp类型转换为smalldatetime类型。
  参数：timestamp without time zone
  返回值类型：smalldatetime
- timestamp_varchar
  描述：timestamp类型转换为varchar类型。
  参数：timestamp without time zone
  返回值类型：character varying
- timestamptz_to_smalldatetime
  描述：timestamptz类型转换为smalldatetime。
  参数：timestamp with time zone
  返回值类型：smalldatetime
- timestampzone_text
  描述：timestampzone类型转换为text类型。
  参数：timestamp with time zone
  返回值类型：text
- timetz_text
  描述：timetz类型转换为text类型。
  参数：time with time zone
  返回值类型：text
- to_integer
  描述：转换为integer类型。
  参数：character varying
  返回值类型：integer
- to_interval
  描述：转换为interval类型。
  参数：character varying

返回值类型：interval

- to_numeric
  描述：转换为numeric类型。
  参数：character varying
  返回值类型：numeric
- to_text
  描述：转换为text类型。
  参数：smallint
  返回值类型：text
- to_ts
  描述：转换为ts类型。
  参数：character varying
  返回值类型：timestamp without time zone
- to_varchar2
  描述：转换为varchar2类型。
  参数：timestamp without time zone
  返回值类型：character varying
- varchar_date
  描述：varchar类型转换为date。
  参数：character varying
  返回值类型：date
- varchar_float4
  描述：varchar类型转换为float4。
  参数：character varying
  返回值类型：real
- varchar_float8
  描述：varchar类型转换为float8。
  参数：character varying
  返回值类型：double precision
- varchar_int4
  描述：varchar类型转换为int4。
  参数：character varying
  返回值类型：integer
- varchar_int8
  描述：varchar类型转换为int8。
  参数：character varying
  返回值类型：bigint
- varchar_numeric
  描述：varchar类型转换为numeric。
  参数：character varying

返回值类型：numeric

- varchar_timestamp
  描述：varchar类型转换为timestamp。
  参数：character varying
  返回值类型：timestamp without time zone
- varchar2_to_smlldatetime
  描述：varchar2类型转换为smlldatetime。
  参数：character varying
  返回值类型：smalldatetime
- xidout4
  描述：xid输出为4字节数字。
  参数：xid32
  返回值类型：cstring
- xidsend4
  描述：xid转换为二进制格式。
  参数：xid32
  返回值类型：bytea

编码类型转换

- convert_to_nocase(text, text)
  描述：将字符串转换为指定的编码类型。
  返回值类型：bytea
  示例：

```
openGauss=# SELECT convert_to_nocase('12345', 'GBK');
 convert_to_nocase
-------------------
 \x3132333435
(1 row)
```

7.5.10 几何函数和操作符

几何操作符

- +
  描述：平移。
  示例：

```
openGauss=# SELECT box '((0,0),(1,1))' + point '(2.0,0)' AS RESULT;
   result
-------------
 (3,1),(2,0)
(1 row)
```

- -e
  描述：平移。
  示例：

```
openGauss=# SELECT box '((0,0),(1,1))' - point '(2.0,0)' AS RESULT;
    result
---------------
```



```
 (-1,1),(-2,0)
(1 row)
```

- *

  描述：伸展/旋转。

  示例：

```
openGauss=# SELECT box '((0,0),(1,1))' * point '(2.0,0)' AS RESULT;
   result
-------------
 (2,2),(0,0)
(1 row)
```

- /

  描述：收缩/旋转。

  示例：

```
openGauss=# SELECT box '((0,0),(2,2))' / point '(2.0,0)' AS RESULT;
   result
-------------
 (1,1),(0,0)
(1 row)
```

- #

  描述：两个图形交面。

  示例：

```
openGauss=# SELECT box '((1,-1),(-1,1))' # box '((1,1),(-2,-2))' AS RESULT;
 result
---------------
 (1,1),(-1,-1)
(1 row)
```

- #

  描述：图形的路径数目或多边形顶点数。

  示例：

```
openGauss=# SELECT # path'((1,0),(0,1),(-1,0))' AS RESULT;
 result
--------
      3
(1 row)
```

- @-@

  描述：图形的长度或者周长。

  示例：

```
openGauss=# SELECT @-@ path '((0,0),(1,0))' AS RESULT;
 result
--------
      2
(1 row)
```

- @@

  描述：图形的中心。

  示例：

```
openGauss=# SELECT @@ circle '((0,0),10)' AS RESULT;
 result
--------
 (0,0)
(1 row)
```

- <->

  描述：两个图形之间的距离。

示例：

```
openGauss=# SELECT circle '((0,0),1)' <-> circle '((5,0),1)' AS RESULT;
 result
--------
      3
(1 row)
```

- &&

  描述：两个图形是否重叠（有一个共同点就为真）。

  示例：

```
openGauss=# SELECT box '((0,0),(1,1))' && box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- <<

  描述：图形是否全部在另一个图形的左边（没有相同的横坐标）。

  示例：

```
openGauss=# SELECT circle '((0,0),1)' << circle '((5,0),1)' AS RESULT;
 result
--------
 t
(1 row)
```

- >>

  描述：图形是否全部在另一个图形的右边（没有相同的横坐标）。

  示例：

```
openGauss=# SELECT circle '((5,0),1)' >> circle '((0,0),1)' AS RESULT;
 result
--------
 t
(1 row)
```

- &<

  描述：图形的最右边是否不超过在另一个图形的最右边。

  示例：

```
openGauss=# SELECT box '((0,0),(1,1))' &< box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- &>

  描述：图形的最左边是否不超过在另一个图形的最左边。

  示例：

```
openGauss=# SELECT box '((0,0),(3,3))' &> box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- <<|

  描述：图形是否全部在另一个图形的下边（没有相同的纵坐标）。

  示例：

```
openGauss=# SELECT box '((0,0),(3,3))' <<| box '((3,4),(5,5))' AS RESULT;
 result
--------
 t
(1 row)
```


- |>>

  描述：图形是否全部在另一个图形的上边（没有相同的纵坐标）。

  示例：

```
openGauss=# SELECT box '((3,4),(5,5))' |>> box '((0,0),(3,3))' AS RESULT;
 result
--------
 t
(1 row)
```

- &<|

  描述：图形的最上边是否不超过另一个图形的最上边。

  示例：

```
openGauss=# SELECT box '((0,0),(1,1))' &<| box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- |&>

  描述：图形的最下边是否不超过另一个图形的最下边。

  示例：

```
openGauss=# SELECT box '((0,0),(3,3))' |&> box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- <^

  描述：图形是否低于另一个图形（允许两个图形有接触）。

  示例：

```
openGauss=# SELECT box '((0,0),(-3,-3))' <^ box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- >^

  描述：图形是否高于另一个图形（允许两个图形有接触）。

  示例：

```
openGauss=# SELECT box '((0,0),(2,2))' >^ box '((0,0),(-3,-3))'  AS RESULT;
 result
--------
 t
(1 row)
```

- ?#

  描述：两个图形是否相交。

  示例：

```
openGauss=# SELECT lseg '((-1,0),(1,0))' ?# box '((-2,-2),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- ?-

  描述：图形是否处于水平位置。

  示例：


```
openGauss=# SELECT ?- lseg '((-1,0),(1,0))' AS RESULT;
 result
--------
 t
(1 row)
```

- ?-

  描述：图形是否水平对齐。

  示例：

```
openGauss=# SELECT point '(1,0)' ?- point '(0,0)' AS RESULT;
 result
--------
 t
(1 row)
```

- ?|

  描述：图形是否处于竖直位置。

  示例：

```
openGauss=# SELECT ?| lseg '((-1,0),(1,0))' AS RESULT;
 result
--------
 f
(1 row)
```

- ?|

  描述：图形是否竖直对齐。

  示例：

```
openGauss=# SELECT point '(0,1)' ?| point '(0,0)' AS RESULT;
 result
--------
 t
(1 row)
```

- ?-|

  描述：两条线是否垂直。

  示例：

```
openGauss=# SELECT lseg '((0,0),(0,1))' ?-| lseg '((0,0),(1,0))' AS RESULT;
 result
--------
 t
(1 row)
```

- ?||

  描述：两条线是否平行。

  示例：

```
openGauss=# SELECT lseg '((-1,0),(1,0))' ?|| lseg '((-1,2),(1,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- @>

  描述：图形是否包含另一个图形。

  示例：

```
openGauss=# SELECT circle '((0,0),2)' @> point '(1,1)' AS RESULT;
 result
--------
 t
(1 row)
```


- <@
  描述：图形是否被包含于另一个图形。
  示例：

```
openGauss=# SELECT point '(1,1)' <@ circle '((0,0),2)' AS RESULT;
 result
--------
 t
(1 row)
```

- ~=
  描述：两个图形是否相同?
  示例：

```
openGauss=# SELECT polygon '((0,0),(1,1))' ~= polygon '((1,1),(0,0))' AS RESULT;
 result
--------
 t
(1 row)
```

几何函数

- area(object)
  描述：计算图形的面积。
  返回类型：double precision
  示例：

```
openGauss=# SELECT area(box '((0,0),(1,1))') AS RESULT;
 result
--------
      1
(1 row)
```

- center(object)
  描述：计算图形的中心。
  返回类型：point
  示例：

```
openGauss=# SELECT center(box '((0,0),(1,2))') AS RESULT;
 result
---------
 (0.5,1)
(1 row)
```

- diameter(circle)
  描述：计算圆的直径。
  返回类型：double precision
  示例：

```
openGauss=# SELECT diameter(circle '((0,0),2.0)') AS RESULT;
 result
--------
      4
(1 row)
```

- height(box)
  描述：矩形的竖直高度。
  返回类型：double precision
  示例：

```
openGauss=# SELECT height(box '((0,0),(1,1))') AS RESULT;
 result
```



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

- isclosed(path)

  描述：图形是否为闭合路径。

  返回类型：Boolean

  示例：

```
openGauss=# SELECT isclosed(path '((0,0),(1,1),(2,0))') AS RESULT;
 result
--------
 t
(1 row)
```

- isopen(path)

  描述：图形是否为开放路径。

  返回类型：Boolean

  示例：

```
openGauss=# SELECT isopen(path '[(0,0),(1,1),(2,0)]') AS RESULT;
 result
--------
 t
(1 row)
```

- length(object)

  描述：计算图形的长度。

  返回类型：double precision

  示例：

```
openGauss=# SELECT length(path '((-1,0),(1,0))') AS RESULT;
 result
--------
     4
(1 row)
```

- npoints(path)

  描述：计算路径的顶点数。

  返回类型：int

  示例：

```
openGauss=# SELECT npoints(path '[(0,0),(1,1),(2,0)]') AS RESULT;
 result
--------
     3
(1 row)
```

- npoints(polygon)

  描述：计算多边形的顶点数。

  返回类型：int

  示例：

```
openGauss=# SELECT npoints(polygon '((1,1),(0,0))') AS RESULT;
 result
--------
     2
(1 row)
```

- pclose(path)

  描述：把路径转换为闭合路径。

  返回类型：path

示例：

```
openGauss=# SELECT pclose(path '[(0,0),(1,1),(2,0)]') AS RESULT;
      result
--------------------
 ((0,0),(1,1),(2,0))
(1 row)
```

- popen(path)

  描述：把路径转换为开放路径。

  返回类型：path

  示例：

```
openGauss=# SELECT popen(path '((0,0),(1,1),(2,0))') AS RESULT;
      result
--------------------
 [(0,0),(1,1),(2,0)]
(1 row)
```

- radius(circle)

  描述：计算圆的半径。

  返回类型：double precision

  示例：

```
openGauss=# SELECT radius(circle '((0,0),2.0)') AS RESULT;
 result
--------
      2
(1 row)
```

- width(box)

  描述：计算矩形的水平尺寸。

  返回类型：double precision

  示例：

```
openGauss=# SELECT width(box '((0,0),(1,1))') AS RESULT;
 result
--------
      1
(1 row)
```

几何类型转换函数

- box(circle)

  描述：将圆转换成矩形

  返回类型：box

  示例：

```
openGauss=# SELECT box(circle '((0,0),2.0)') AS RESULT;
                                result
-----------------------------------------------------------------------
 (1.41421356237309,1.41421356237309),(-1.41421356237309,-1.41421356237309)
(1 row)
```

- box(point, point)

  描述：将点转换成矩形

  返回类型：box

  示例：

```
openGauss=# SELECT box(point '(0,0)', point '(1,1)') AS RESULT;
  result
-------------
```



```
 (1,1),(0,0)
(1 row)
```

- box(polygon)

  描述：将多边形转换成矩形

  返回类型：box

  示例：

```
openGauss=# SELECT box(polygon '((0,0),(1,1),(2,0))') AS RESULT;
  result
-------------
 (2,1),(0,0)
(1 row)
```

- circle(box)

  描述：矩形转换成圆

  返回类型：circle

  示例：

```
openGauss=# SELECT circle(box '((0,0),(1,1))') AS RESULT;
            result
-------------------------------
 <(0.5,0.5),0.707106781186548>
(1 row)
```

- circle(point, double precision)

  描述：将圆心和半径转换成圆

  返回类型：circle

  示例：

```
openGauss=# SELECT circle(point '(0,0)', 2.0) AS RESULT;
 result
-----------
 <(0,0),2>
(1 row)
```

- circle(polygon)

  描述：将多边形转换成圆

  返回类型：circle

  示例：

```
openGauss=# SELECT circle(polygon '((0,0),(1,1),(2,0))') AS RESULT;
                  result
-------------------------------------------
 <(1,0.333333333333333),0.924950591148529>
(1 row)
```

- lseg(box)

  描述：矩形对角线转化成线段

  返回类型：lseg

  示例：

```
openGauss=# SELECT lseg(box '((-1,0),(1,0))') AS RESULT;
    result
---------------
 [(1,0),(-1,0)]
(1 row)
```

- lseg(point, point)

  描述：点转换成线段

  返回类型：lseg

示例：

```
openGauss=# SELECT lseg(point '(-1,0)', point '(1,0)') AS RESULT;
    result
---------------
 [(-1,0),(1,0)]
(1 row)
```

- slope(point, point)

  描述：计算两个点构成直线的斜率。

  返回类型：double

  示例：

  ```
  openGauss=# SELECT slope(point '(1,1)', point '(0,0)') AS RESULT;
   result
  --------
       1
  (1 row)
  ```

- path(polygon)

  描述：多边形转换成路径

  返回类型：path

  示例：

  ```
  openGauss=# SELECT path(polygon '((0,0),(1,1),(2,0))') AS RESULT;
        result
  ---------------------
   ((0,0),(1,1),(2,0))
  (1 row)
  ```

- point(double precision, double precision)

  描述：节点

  返回类型：point

  示例：

  ```
  openGauss=# SELECT point(23.4, -44.5) AS RESULT;
     result
  --------------
   (23.4,-44.5)
  (1 row)
  ```

- point(box)

  描述：矩形的中心

  返回类型：point

  示例：

  ```
  openGauss=# SELECT point(box '((-1,0),(1,0))') AS RESULT;
   result
  --------
   (0,0)
  (1 row)
  ```

- point(circle)

  描述：圆心

  返回类型：point

  示例：

  ```
  openGauss=# SELECT point(circle '((0,0),2.0)') AS RESULT;
   result
  --------
   (0,0)
  (1 row)
  ```


- point(lseg)

  描述：线段的中心

  返回类型：point

  示例：

```
openGauss=# SELECT point(lseg '((-1,0),(1,0))') AS RESULT;
 result
--------
 (0,0)
(1 row)
```

- point(polygon)

  描述：多边形的中心

  返回类型：point

  示例：

```
openGauss=# SELECT point(polygon '((0,0),(1,1),(2,0))') AS RESULT;
        result
----------------------
 (1,0.333333333333333)
(1 row)
```

- polygon(box)

  描述：矩形转换成4点多边形

  返回类型：polygon

  示例：

```
openGauss=# SELECT polygon(box '((0,0),(1,1))') AS RESULT;
          result
---------------------------
 ((0,0),(0,1),(1,1),(1,0))
(1 row)
```

- polygon(circle)

  描述：圆转换成12点多边形

  返回类型：polygon

  示例：

```
openGauss=# SELECT polygon(circle '((0,0),2.0)') AS RESULT;

result

------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------
 ((-2,0),(-1.73205080756888,1),(-1,1.73205080756888),(-1.22464679914735e-16,2),
(1,1.73205080756888),(1.73205080756888,1),(2,2.44929359829471e-16),
(1.73205080756888,-0.999999999999999),(1,-1.73205080756888),(3.67394039744206e-16,-2),
(-0.999999999999999,-1.73205080756888),(-1.73205080756888,-1))
(1 row)
```

- polygon(npts, circle)

  描述：圆转换成npts点多边形

  返回类型：polygon

  示例：

```
openGauss=# SELECT polygon(12, circle '((0,0),2.0)') AS RESULT;

result

------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
```



```
------------------------------------------------------------
 ((-2,0),(-1.73205080756888,1),(-1,1.73205080756888),(-1.22464679914735e-16,2),
(1,1.73205080756888),(1.73205080756888,1),(2,2.44929359829471e-16),
(1.73205080756888,-0.999999999999999),(1,-1.73205080756888),(3.67394039744206e-16,-2),
(-0.999999999999999,-1.73205080756888),(-1.73205080756888,-1))
(1 row)
```

- polygon(path)

  描述：路径转换成多边形

  返回类型：polygon

  示例：

```
openGauss=# SELECT polygon(path '((0,0),(1,1),(2,0))') AS RESULT;
      result
---------------------
 ((0,0),(1,1),(2,0))
(1 row)
```

7.5.11 网络地址函数和操作符

cidr 和 inet 操作符

操作符<<，<<=，>>，>>=对子网进行测试。它们只考虑两个地址的网络部分（忽略任何主机部分），然后判断其中一个网络是等于另外一个网络，还是另外一个网络的子网。

- <

  描述：小于

  示例：

```
openGauss=# SELECT inet '192.168.1.5' < inet '192.168.1.6' AS RESULT;
 result
--------
 t
(1 row)
```

- <=

  描述：小于或等于

  示例：

```
openGauss=# SELECT inet '192.168.1.5' <= inet '192.168.1.5' AS RESULT;
 result
--------
 t
(1 row)
```

- =

  描述：等于

  示例：

```
openGauss=# SELECT inet '192.168.1.5' = inet '192.168.1.5' AS RESULT;
 result
--------
 t
(1 row)
```

- >=

  描述：大于或等于

  示例：

```
openGauss=# SELECT inet '192.168.1.5' >= inet '192.168.1.5' AS RESULT;
 result
```



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

- >

  描述：大于

  示例：

```
openGauss=# SELECT inet '192.168.1.5' > inet '192.168.1.4' AS RESULT;
 result
--------
 t
(1 row)
```

- <>

  描述：不等于

  示例：

```
openGauss=# SELECT inet '192.168.1.5' <> inet '192.168.1.4' AS RESULT;
 result
--------
 t
(1 row)
```

- <<

  描述：包含于

  示例：

```
openGauss=# SELECT inet '192.168.1.5' << inet '192.168.1/24' AS RESULT;
 result
--------
 t
(1 row)
```

- <<=

  描述：包含于或等于

  示例：

```
openGauss=# SELECT inet '192.168.1/24' <<= inet '192.168.1/24' AS RESULT;
 result
--------
 t
(1 row)
```

- >>

  描述：包含

  示例：

```
openGauss=# SELECT inet '192.168.1/24' >> inet '192.168.1.5' AS RESULT;
 result
--------
 t
(1 row)
```

- >>=

  描述：包含或等于

  示例：

```
openGauss=# SELECT inet '192.168.1/24' >>= inet '192.168.1/24' AS RESULT;
 result
--------
 t
(1 row)
```

- ~

  描述：位非

示例：

```
openGauss=# SELECT ~ inet '192.168.1.6' AS RESULT;
    result
---------------
 63.87.254.249
(1 row)
```

- &

  描述：两个网络地址的每一位都进行“与”操作。

  示例：

```
openGauss=# SELECT inet '192.168.1.6' & inet '10.0.0.0' AS RESULT;
 result
---------
 0.0.0.0
(1 row)
```

- |

  描述：两个网络地址的每一位都进行“或”操作。

  示例：

```
openGauss=# SELECT inet '192.168.1.6' | inet '10.0.0.0' AS RESULT;
   result
-------------
 202.168.1.6
(1 row)
```

- +

  描述：加

  示例：

```
openGauss=# SELECT inet '192.168.1.6' + 25 AS RESULT;
    result
--------------
 192.168.1.31
(1 row)
```

- -

  描述：减

  示例：

```
openGauss=# SELECT inet '192.168.1.43' - 36 AS RESULT;
   result
-------------
 192.168.1.7
(1 row)
```

- -

  描述：减

  示例：

```
openGauss=# SELECT inet '192.168.1.43' - inet '192.168.1.19' AS RESULT;
 result
--------
     24
(1 row)
```

cidr 和 inet 函数

函数abbrev，host，text主要是为了提供可选的显示格式。

- abbrev(inet)

  描述：缩写显示格式文本。

返回类型：text

示例：

```
openGauss=# SELECT abbrev(inet '10.1.0.0/16') AS RESULT;
  result
-------------
 10.1.0.0/16
(1 row)
```

- abbrev(cidr)

  描述：缩写显示格式文本。

  返回类型：text

  示例：

  ```
  openGauss=# SELECT abbrev(cidr '10.1.0.0/16') AS RESULT;
   result
  ---------
   10.1/16
  (1 row)
  ```

- broadcast(inet)

  描述：网络广播地址。

  返回类型：inet

  示例：

  ```
  openGauss=# SELECT broadcast('192.168.1.5/24') AS RESULT;
       result
  ------------------
   192.168.1.255/24
  (1 row)
  ```

- family(inet)

  描述：抽取地址族，4为IPv4。

  返回类型：int

  示例：

  ```
  openGauss=# SELECT family('127.0.0.1') AS RESULT;
   result
  --------
       4
  (1 row)
  ```

- host(inet)

  描述：将主机地址类型抽出为文本。

  返回类型：text

  示例：

  ```
  openGauss=# SELECT host('192.168.1.5/24') AS RESULT;
    result
  -------------
   192.168.1.5
  (1 row)
  ```

- hostmask(inet)

  描述：为网络构造主机掩码。

  返回类型：inet

  示例：

  ```
  openGauss=# SELECT hostmask('192.168.23.20/30') AS RESULT;
   result
  ---------
   0.0.0.3
  (1 row)
  ```


- masklen(inet)
  描述：抽取子网掩码长度。
  返回类型：int
  示例：

```
openGauss=# SELECT masklen('192.168.1.5/24') AS RESULT;
 result
--------
     24
(1 row)
```

- netmask(inet)
  描述：为网络构造子网掩码。
  返回类型：inet
  示例：

```
openGauss=# SELECT netmask('192.168.1.5/24') AS RESULT;
    result
---------------
 255.255.255.0
(1 row)
```

- network(inet)
  描述：抽取地址的网络部分。
  返回类型：cidr
  示例：

```
openGauss=# SELECT network('192.168.1.5/24') AS RESULT;
     result
----------------
 192.168.1.0/24
(1 row)
```

- set_masklen(inet, int)
  描述：为inet数值设置子网掩码长度。
  返回类型：inet
  示例：

```
openGauss=# SELECT set_masklen('192.168.1.5/24', 16) AS RESULT;
     result
----------------
 192.168.1.5/16
(1 row)
```

- set_masklen(cidr, int)
  描述：为cidr数值设置子网掩码长度。
  返回类型：cidr
  示例：

```
openGauss=# SELECT set_masklen('192.168.1.0/24'::cidr, 16) AS RESULT;
     result
----------------
 192.168.0.0/16
(1 row)
```

- text(inet)
  描述：把IP地址和掩码长度抽取为文本。
  返回类型：text
  示例：

```
openGauss=# SELECT text(inet '192.168.1.5') AS RESULT;
     result
```



```
----------------
 192.168.1.5/32
(1 row)
```

任何cidr值都能以显式或者隐式的方式转换为inet值，因此上述能够操作inet值的函数也同样能够操作cidr值。inet值也可以转换为cidr值，此时inet子网掩码右侧的所有位都将转换为零，以创建一个有效的cidr值。另外，用户还可以使用常规的类型转换语法将一个文本字符串转换为inet或cidr值。例如：inet(expression)或colname::cidr。

macaddr 函数

函数trunc(macaddr)返回一个MAC地址，该地址的最后三个字节设置为零。

trunc(macaddr)

描述：把后三个字节置为零。

返回类型：macaddr

示例：

```
openGauss=# SELECT trunc(macaddr '12:34:56:78:90:ab') AS RESULT;
      result
-------------------
 12:34:56:00:00:00
(1 row)
```

macaddr类型还支持标准关系操作符（>，<=等）用于词法排序，和按位运算符（~，&和|）非，与和或。

7.5.12 JSON/JSONB 函数和操作符

JSON/JSONB数据类型参考**JSON/JSONB类型**。

表 7-33 JSON/JSONB 通用操作符

操作符	左操作数类型	右操作数类型	返回类型	描述	示例
->	Array-json(b)	int	jsonb	获得array-json元素。下标不存在返回空。	SELECT '[{"a":"foo"},{"b":"bar"},{"c":"baz"}]'::json->2; ?column? ---------- {"c":"baz"} (1 row)
->	object-json(b)	text	jsonb	通过键获得值。不存在则返回空。	SELECT '{"a": {"b":"foo"}}'::json->'a'; ?column? ---------- {"b":"foo"} (1 row)
->>	Array-json(b)	int	text	获得 JSON 数组元素。下标不存在返回空。	SELECT '[1,2,3]'::json->>2; ?column? ---------- 3 (1 row)

操作符	左操作数类型	右操作数类型	返回类型	描述	示例
->>	object-json(b)	text	text	通过键获得值。不存在则返回空。	SELECT '{"a":1,"b":2}'::json->>'b'; ?column? ---------- 2 (1 row)
#>	container-json (b)	text[]	json(b)	获取在指定路径的 JSON 对象，路径不存在则返回空。	SELECT '{"a": {"b":{"c": "foo"}}}'::json #>'{a,b}'; ?column? -------------- {"c": "foo"} (1 row)
#>>	container-json (b)	text[]	text	获取在指定路径的 JSON 对象，路径不存在则返回空。	SELECT '{"a":[1,2,3],"b":[4,5,6]}'::json #>>'{a,2}'; ?column? ---------- 3 (1 row)

⚠ 注意

对于 #> 和 #>> 操作符，当给出的路径无法查找到数据时，不会报错，会返回空。

表 7-34 JSONB 额外支持操作符

操作符	右操作数类型	描述	例子
@>	jsonb	左边的 JSON的顶层是否包含右边JSON的顶层所有项。	'{"a":1, "b":2}'::jsonb @> '{"b":2}'::jsonb
<@	jsonb	左边的 JSON的所有项是否全部存在于右边JSON的顶层。	'{"b":2}'::jsonb <@ '{"a":1, "b":2}'::jsonb
?	text	键/元素的字符串是否存在于 JSON 值的顶层。	'{"a":1, "b":2}'::jsonb ? 'b'
?|	text[]	这些数组字符串中的任何一个是否作为顶层键存在。	'{"a":1, "b":2, "c":3}'::jsonb ?| array['b', 'c']
?&	text[]	是否所有这些数组字符串都作为顶层键存在。	'["a", "b"]'::jsonb ?& array['a', 'b']

操作符	右操作数类型	描述	例子
=	jsonb	判断两个jsonb的大小关系，同函数jsonb_eq。	/
<>	jsonb	判断两个jsonb的大小关系，同函数jsonb_ne。	/
<	jsonb	判断两个jsonb的大小关系，同函数jsonb_lt。	/
>	jsonb	判断两个jsonb的大小关系，同函数jsonb_gt。	/
<=	jsonb	判断两个jsonb的大小关系，同函数jsonb_le。	/
>=	jsonb	判断两个jsonb的大小关系，同函数jsonb_ge。	/

JSON/JSONB 支持的函数

- array_to_json(anyarray [, pretty_bool])

  描述：返回JSON类型的数组。将一个多维数组组成一个JSON数组。如果pretty_bool为true，将在一维元素之间添加换行符。

  返回类型：json

  示例：

```
openGauss=# SELECT array_to_json('{{1,5},{99,100}}'::int[]);
  array_to_json
  ------------------
  [[1,5],[99,100]]
  (1 row)
```

- row_to_json(record [, pretty_bool])

  描述：返回JSON类型的行。如果pretty_bool为true，将在第一级元素之间添加换行符。

  返回类型：json

  示例：

```
openGauss=# SELECT row_to_json(row(1,'foo'));
       row_to_json
  ---------------------
   {"f1":1,"f2":"foo"}   (1 row)
```

- json_array_element(array-json, integer)、jsonb_array_element(array-jsonb, integer)

  描述：同操作符`->`, 返回数组中指定下标的元素。

  返回类型：json、jsonb

示例：

```
openGauss=# SELECT json_array_element('[1,true,[1,[2,3
]],null]',2);
 json_array_element
 --------------------
 [1,[2,3]]
 (1 row)
```

- json_array_element_text(array-json, integer)、jsonb_array_element_text(array-jsonb, integer)

  描述：同操作符`->>`, 返回数组中指定下标的元素。

  返回类型：text、text

  示例：

  ```
  openGauss=# SELECT json_array_element_text('[1,true,[1,[2,3]],null]',2);
  json_array_element_text
  ------------------------
   [1,[2,3]]
   (1 row)
  ```

- json_object_field(object-json, text)、jsonb_object_field(object-jsonb, text)

  描述：同操作符`->`, 返回对象中指定键对应的值。

  返回类型：json、json

  示例：

  ```
  openGauss=# SELECT json_object_field('{"a": {"b":"foo"}}','a');
   json_object_field
   -------------------
   {"b":"foo"}
   (1 row)
  ```

- json_object_field_text(object-json, text)、jsonb_object_field_text(object-jsonb, text)

  描述：同操作符`->>`, 返回对象中指定键对应的值。

  返回类型：text、text

  示例：

  ```
  openGauss=# SELECT json_object_field_text('{"a": {"b":"foo"}}','a');
   json_object_field_text
   -----------------------
   {"b":"foo"}
   (1 row)
  ```

- json_extract_path(json, VARIADIC text[])、jsonb_extract_path((jsonb, VARIADIC text[])

  描述：等价于操作符`#>`。根据$2所指的路径，查找json，并返回。

  返回类型：json、jsonb

  示例：

  ```
  openGauss=# SELECT json_extract_path('{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}', 'f4','f6');
    json_extract_path
   -------------------
    "stringy"
   (1 row)
  ```

- json_extract_path_op(json, text[])、jsonb_extract_path_op(jsonb, text[])

  描述：同操作符`#>`。根据$2所指的路径，查找json，并返回。

  返回类型：json、jsonb

  示例：

  ```
   openGauss=# SELECT json_extract_path_op('{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}',
  ARRAY['f4','f6']);
    json_extract_path_op
  ```



```
---------------------
 "stringy"
(1 row)
```

- json_extract_path_text(json, VARIADIC text[])、jsonb_extract_path_text((jsonb, VARIADIC text[])

  描述：等价于操作符`#>>`。根据$2所指的路径，查找json，并返回。

  返回类型：text、text

  示例：

```
openGauss=# SELECT json_extract_path_text('{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}', 'f4','f6');
 json_extract_path_text
------------------------
 stringy
(1 row)
```

- json_extract_path_text_op(json, text[])、jsonb_extract_path_text_op(jsonb, text[])

  描述：同操作符`#>>`。根据$2所指的路径，查找json，并返回。

  返回类型：text、text

  示例：

```
openGauss=# SELECT json_extract_path_text_op('{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}',
ARRAY['f4','f6']);
  json_extract_path_text_op
---------------------------
 stringy
(1 row)
```

- json_array_elements(array-json)、jsonb_array_elements(array-jsonb)

  描述：拆分数组，每一个元素返回一行。

  返回类型：json、jsonb

  示例：

```
openGauss=# SELECT json_array_elements('[1,true,[1,[2,3]],null]');
 json_array_elements
---------------------
 1
 true
 [1,[2,3]]
 null
(4 rows)
```

- json_array_elements_text(array-json)、jsonb_array_elements_text(array-jsonb)

  描述：拆分数组，每一个元素返回一行。

  返回类型：text、text

  示例：

```
openGauss=# SELECT * FROM  json_array_elements_text('[1,true,[1,[2,3]],null]');
    value
-----------
 1
 true
 [1,[2,3]]

(4 rows)
```

- json_array_length(array-json)、jsonb_array_length(array-jsonb)

  描述：返回数组长度。

  返回类型：integer

  示例：

```
 openGauss=# SELECT json_array_length('[1,2,3,{"f1":1,"f2":[5,6]},4,null]');
  json_array_length
```



```
-------------------
                 6
(1 row)
```

- json_each(object-json)、jsonb_each(object-jsonb)

  描述：将对象的每个键值对拆分转换成一行两列。

  返回类型：setof(key text, value json)、setof(key text, value jsonb)

  示例：

```
openGauss=# SELECT * FROM  json_each('{"f1":[1,2,3],"f2":{"f3":1},"f4":null}');
 key |  value
-----+----------
 f1  | [1,2,3]
 f2  | {"f3":1}
 f4  | null
(3 rows)
```

- json_each_text(object-json)、jsonb_each_text(object-jsonb)

  描述：将对象的每个键值对拆分转换成一行两列。

  返回类型：setof(key text, value text)、setof(key text, value text)

  示例：

```
openGauss=# SELECT * FROM  json_each_text('{"f1":[1,2,3],"f2":{"f3":1},"f4":null}');
 key |  value
-----+----------
 f1  | [1,2,3]
 f2  | {"f3":1}
 f4  |
(3 rows)
```

- json_object_keys(object-json)、jsonb_object_keys(object-jsonb)

  描述：返回对象中顶层的所有键。

  返回类型：SETOF text

  示例：

```
openGauss=# SELECT json_object_keys('{"f1":"abc","f2":{"f3":"a", "f4":"b"}, "f1":"abcd"}');
 json_object_keys
------------------
 f1
 f2
 f1
(3 rows)
```

- jsonb中会有去重操作

```
openGauss=# SELECT jsonb_object_keys('{"f1":"abc","f2":{"f3":"a", "f4":"b"}, "f1":"abcd"}');
 jsonb_object_keys
-------------------
 f1
 f2
(2 rows)
```

- json_populate_record(anyelement, object-json [, bool])、jsonb_populate_record(anyelement, object-jsonb [, bool])

  描述：$1必须是一个复合类型的参数。将会把object-json里的每个对键值进行拆分，以键当做列名，与$1中的列名进行匹配查找，并填充到$1的格式中。

  返回类型：anyelement、anyelement

  示例：

```
openGauss=# CREATE TYPE jpop as (a text, b int, c bool);
CREATE TYPE
openGauss=# SELECT * FROM json_populate_record(null::jpop,'{"a":"blurfl","x":43.2}');
   a    | b | c
--------+---+---
 blurfl |   |
(1 row)
```



```
openGauss=# SELECT * FROM json_populate_record((1,1,null)::jpop,'{"a":"blurfl","x":43.2}');
    a    | b | c
 --------+---+---
  blurfl | 1 |
 (1 row)
```

- json_populate_record_set(anyelement, array-json [, bool])、jsonb_populate_record_set(anyelement, array-jsonb [, bool])

  描述：参考上述函数json_populate_record、jsonb_populate_record，对$2数组的每一个元素进行上述参数函数的操作，因此这也要求$2数组的每个元素都是object-json类型的。

  返回类型：setof anyelement、setof anyelement

  示例：

```
openGauss=# CREATE TYPE jpop as (a text, b int, c bool);
 CREATE TYPE
openGauss=# SELECT * FROM json_populate_recordset(null::jpop, '[{"a":1,"b":2},{"a":3,"b":4}]');
 a | b | c
---+---+---
 1 | 2 |
 3 | 4 |
(2 rows)
```

- json_typeof(json)、jsonb_typeof(jsonb)

  描述：检测json类型

  返回类型：text、text

  示例：

```
openGauss=# SELECT value, json_typeof(value) FROM (values (json '123.4'), (json '"foo"'), (json
'true'), (json 'null'), (json '[1, 2, 3]'), (json '{"x":"foo", "y":123}'), (NULL::json))  as data(value);
         value          | json_typeof
------------------------+-------------
 123.4                  | number
 "foo"                  | string
 true                   | boolean
 null                   | null
 [1, 2, 3]              | array
 {"x":"foo", "y":123} | object
                        |
(7 rows)
```

- json_build_array([VARIADIC "any"])

  描述：从一个可变参数列表构造出一个JSON数组。

  返回类型：array-json

  示例：

```
openGauss=# SELECT json_build_array('a',1,'b',1.2,'c',true,'d',null,'e',json '{"x": 3, "y": [1,2,3]}','');
                          json_build_array
-------------------------------------------------------------------------
 ["a", 1, "b", 1.2, "c", true, "d", null, "e", {"x": 3, "y": [1,2,3]}, ""]
(1 row)
```

- json_build_object([VARIADIC "any"])

  描述：从一个可变参数列表构造出一个JSON对象，其入参必须为偶数个，两两一组组成键值对。注意键不可为null。

  返回类型：object-json

  示例：

```
openGauss=# SELECT json_build_object(1,2);
 json_build_object
-------------------
 {"1" : 2}
(1 row)
```

- json_to_record(object-json, bool)

描述：正如所有返回record 的函数一样，调用者必须用一个AS子句显式地定义记录的结构。会将object-json的键值对进行拆分重组，把键当做列名，去匹配填充as显示指定的记录的结构。

返回类型：record

示例：

```
openGauss=# SELECT * FROM json_to_record('{"a":1,"b":"foo","c":"bar"}',true) as x(a int, b text, d
text);
 a |  b  | d
---+-----+---
 1 | foo |
(1 row)
```

- json_to_recordset(array-json, bool)

  描述：参考函数json_to_record，对数组内每个元素，执行上述函数的操作，因此这要求数组内的每个元素都得是object-json。

  返回类型：setof record

  示例：

```
openGauss=# SELECT * FROM json_to_recordset(
openGauss(#    '[{"a":1,"b":"foo","d":false},{"a":2,"b":"bar","c":true}]',
openGauss(#    false
openGauss(# ) as x(a int, b text, c boolean);
 a |  b  | c
---+-----+---
 1 | foo |
 2 | bar | t
(2 rows)
```

- json_object(text[])、json_object(text[], text[])

  描述：从一个文本数组构造一个object-json。这是个重载函数，当入参为一个文本数组的时候，其数组长度必须为偶数，成员被当做交替出现的键值对。两个文本数组的时候，第一个数组认为是键，第二个认为是值，两个数组长度必须相等。键不可为null。

  返回类型：object-json

  示例：

```
openGauss=# SELECT json_object('{a,1,b,2,3,NULL,"d e f","a b c"}');
                       json_object
-------------------------------------------------------
 {"a" : "1", "b" : "2", "3" : null, "d e f" : "a b c"}
(1 row)
openGauss=# SELECT json_object('{a,b,"a b c"}', '{a,1,1}');
              json_object
---------------------------------------
 {"a" : "a", "b" : "1", "a b c" : "1"}
(1 row)
```

- json_agg(any)

  描述：将值聚集为json数组。

  返回类型：array-json

  示例：

```
openGauss=# SELECT * FROM classes;
 name | score
------+-------
A    |     2
A    |     3
D    |     5
D    |
(4 rows)
openGauss=# SELECT name, json_agg(score) score FROM classes group by name order by name;
 name |      score
------+------------------
```



```
A    | [2, 3]
D    | [5, null]
     | [null]
(3 rows)
```

- json_object_agg(any, any)

  描述：将值聚集为json对象。

  返回类型：object-json

  示例：

```
openGauss=# SELECT * FROM classes;
name | score
-----+-------
A    |     2
A    |     3
D    |     5
D    |
(4 rows)
```

```
openGauss=# SELECT json_object_agg(name, score) FROM classes group by name order by name;
     json_object_agg
-------------------------
 { "A" : 2, "A" : 3 }
 { "D" : 5, "D" : null }
(2 rows)
```

- jsonb_contained(jsonb, jsonb)

  描述：同操作符 `<@`, 判断$1中的所有元素是否在$2的顶层存在。

  返回类型：bool

  示例：

```
openGauss=# SELECT jsonb_contained('[1,2,3]', '[1,2,3,4]');
 jsonb_contained
-----------------
 t
(1 row)
```

- jsonb_contains(jsonb, jsonb)

  描述：同操作符 `@>`, 判断$1中的顶层所有元素是否包含在$2的所有元素。

  返回类型：bool

  示例：

```
openGauss=# SELECT jsonb_contains('[1,2,3,4]', '[1,2,3]');
 jsonb_contains
----------------
 t
(1 row)
```

- jsonb_exists(jsonb, text)

  描述：同操作符 `?`, 字符串$2是否存在$1的顶层以key\elem\scalar的形式存在。

  返回类型：bool

  示例：

```
openGauss=# SELECT jsonb_exists('["1",2,3]', '1');
 jsonb_exists
--------------
 t
(1 row)
```

- jsonb_exists_all(jsonb, text[])

  描述：同操作符 `?&`, 字符串数组$2里面，是否所有的元素，都在$1的顶层以key\elem\scalar的形式存在。

  返回类型：bool

  示例：


```
openGauss=# SELECT jsonb_exists_all('["1","2",3]', '{1, 2}');
 jsonb_exists_all
------------------
 t
(1 row)
```

- jsonb_exists_any(jsonb, text[])

  描述：同操作符 `?|`, 字符串数组$2里面，是否存在的元素，在$1的顶层以key\elem\scalar的形式存在。

  返回类型：bool

  示例：

```
openGauss=# SELECT jsonb_exists_any('["1","2",3]', '{1, 2, 4}');
 jsonb_exists_any
------------------
 t
(1 row)
```

- jsonb_cmp(jsonb, jsonb)

  描述：比较大小，正数代表大于，负数代表小于，0表示相等。

  返回类型：integer

  示例：

```
openGauss=# SELECT jsonb_cmp('["a", "b"]', '{"a":1, "b":2}');
 jsonb_cmp
-----------
        -1
(1 row)
```

- jsonb_eq(jsonb, jsonb)

  描述：同操作符 `=`, 比较两个值的大小。

  返回类型：bool

  示例：

```
openGauss=# SELECT jsonb_eq('["a", "b"]', '{"a":1, "b":2}');
 jsonb_eq
----------
 f
(1 row)
```

- jsonb_ne(jsonb, jsonb)

  描述：同操作符 `<>`, 比较两个值的大小。

  返回类型：bool

  示例：

```
openGauss=# SELECT jsonb_ne('["a", "b"]', '{"a":1, "b":2}');
 jsonb_ne
----------
 t
(1 row)
```

- jsonb_gt(jsonb, jsonb)

  描述：同操作符 `>`, 比较两个值的大小。

  返回类型：bool

  示例：

```
openGauss=# SELECT jsonb_gt('["a", "b"]', '{"a":1, "b":2}');
 jsonb_gt
----------
 f
(1 row)
```

- jsonb_ge(jsonb, jsonb)

  描述：同操作符 `>=`, 比较两个值的大小。

返回类型：bool

示例：

```
openGauss=# SELECT jsonb_ge('["a", "b"]', '{"a":1, "b":2}');
 jsonb_ge
----------
 f
(1 row)
```

- jsonb_lt(jsonb, jsonb)

  描述：同操作符 `<`, 比较两个值的大小。

  返回类型：bool

  示例：

```
openGauss=# SELECT jsonb_lt('["a", "b"]', '{"a":1, "b":2}');
 jsonb_lt
----------
 t
(1 row)
```

- jsonb_le(jsonb, jsonb)

  描述：同操作符 `<=`, 比较两个值的大小。

  返回类型：bool

  示例：

```
openGauss=# SELECT jsonb_le('["a", "b"]', '{"a":1, "b":2}');
 jsonb_le
----------
 t
(1 row)
```

- to_json(anyelement)

  描述：把参数转换为`json`

  返回类型：json

  示例：

```
openGauss=# SELECT to_json('{1,5}'::text[]);
   to_json
-----------
 ["1","5"]
(1 row)
```

- jsonb_hash(jsonb)

  描述：对jsonb进行hash运算

  返回类型：integer

  示例：

```
openGauss=# SELECT jsonb_hash('[1,2,3]');
  josnb_hash
--------------
 -559968547
(1 row)
```

- 其他函数

  描述：json\jsonb聚集函数所用到的内部函数，功能不过多赘述。

```
json_agg_transfn
json_agg_finalfn
json_object_agg_transfn
json_object_agg_finalfn
```


7.5.13 HLL 函数和操作符

哈希函数

- hll_hash_boolean(bool)

  描述：对bool类型数据计算哈希值。

  返回值类型：hll_hashval

  示例：

```
openGauss=# SELECT hll_hash_boolean(FALSE);
 hll_hash_boolean
--------------------
 -5451962507482445012
(1 row)
```

- hll_hash_boolean(bool, int32)

  描述：设置hash seed（即改变哈希策略）并对bool类型数据计算哈希值。

  返回值类型：hll_hashval

  示例：

```
openGauss=# SELECT hll_hash_boolean(FALSE, 10);
 hll_hash_boolean
--------------------
 -1169037589280886076
(1 row)
```

- hll_hash_smallint(smallint)

  描述：对smallint类型数据计算哈希值。

  返回值类型：hll_hashval

  示例：

```
openGauss=# SELECT hll_hash_smallint(100::smallint);
 hll_hash_smallint
---------------------
 962727970174027904
(1 row)
```

  说明

  数值大小相同的参数使用不同数据类型的哈希函数计算，最后结果会不一样，因为不同类型哈希函数会选取不同的哈希计算策略。

- hll_hash_smallint(smallint, int32)

  描述：设置hash seed（即改变哈希策略）同时对smallint类型数据计算哈希值。

  返回值类型：hll_hashval

  示例：

```
openGauss=# SELECT hll_hash_smallint(100::smallint, 10);
 hll_hash_smallint
---------------------
 -9056177146160443041
(1 row)
```

- hll_hash_integer(integer)

  描述：对integer类型数据计算哈希值。

  返回值类型：hll_hashval

  示例：

```
openGauss=# SELECT hll_hash_integer(0);
  hll_hash_integer
```



```
----------------------
 5156626420896634997
(1 row)
```

- hll_hash_integer(integer, int32)

  描述：对integer类型数据计算哈希值，并设置hashseed（即改变哈希策略）。

  返回值类型：hll_hashval

  示例：

```
openGauss=# SELECT hll_hash_integer(0, 10);
  hll_hash_integer
--------------------
 -5035020264353794276
(1 row)
```

- hll_hash_bigint(bigint)

  描述：对bigint类型数据计算哈希值。

  返回值类型：hll_hashval

  示例：

```
openGauss=# SELECT hll_hash_bigint(100::bigint);
   hll_hash_bigint
----------------------
 -2401963681423227794
(1 row)
```

- hll_hash_bigint(bigint, int32)

  描述：对bigint类型数据计算哈希值，并设置hashseed（即改变哈希策略）。

  返回值类型：hll_hashval

  示例：

```
openGauss=# SELECT hll_hash_bigint(100::bigint, 10);
   hll_hash_bigint
----------------------
 -2305749404374433531
(1 row)
```

- hll_hash_bytea(bytea)

  描述：对bytea类型数据计算哈希值。

  返回值类型：hll_hashval

  示例：

```
openGauss=# SELECT hll_hash_bytea(E'\\x');
 hll_hash_bytea
----------------
 0
(1 row)
```

- hll_hash_bytea(bytea, int32)

  描述：对bytea类型数据计算哈希值，并设置hashseed（即改变哈希策略）。

  返回值类型：hll_hashval

  示例：

```
openGauss=# SELECT hll_hash_bytea(E'\\x', 10);
   hll_hash_bytea
----------------------
 7233188113542599437
(1 row)
```

- hll_hash_text(text)

  描述：对text类型数据计算哈希值。

  返回值类型：hll_hashval

示例：

```
openGauss=# SELECT hll_hash_text('AB');
   hll_hash_text
---------------------
 -5666002586880275174
(1 row)
```

- hll_hash_text(text, int32)

  描述：对text类型数据计算哈希值, 并设置hashseed（即改变哈希策略）。

  返回值类型：hll_hashval

  示例：

```
openGauss=# SELECT hll_hash_text('AB', 10);
hll_hash_text
---------------------
-2215507121143724132
(1 row)
```

- hll_hash_any(anytype)

  描述：对任意类型数据计算哈希值。

  返回值类型：hll_hashval

  示例：

```
openGauss=# SELECT hll_hash_any(1);
    hll_hash_any
----------------------
 -1316670585935156930
(1 row)

openGauss=# SELECT hll_hash_any('08:00:2b:01:02:03'::macaddr);
    hll_hash_any
----------------------
 -3719950434455589360
(1 row)
```

- hll_hash_any(anytype, int32)

  描述：对任意类型数据计算哈希值，并设置hashseed（即改变哈希策略）。

  返回值类型：hll_hashval

  示例：

```
openGauss=# SELECT hll_hash_any(1, 10);
    hll_hash_any
----------------------
 7048553517657992351
(1 row)
```

- hll_hashval_eq(hll_hashval, hll_hashval)

  描述：比较两个hll_hashval类型数据是否相等。

  返回值类型：bool

  示例：

```
openGauss=# SELECT hll_hashval_eq(hll_hash_integer(1), hll_hash_integer(1));
 hll_hashval_eq
----------------
 t
(1 row)
```

- hll_hashval_ne(hll_hashval, hll_hashval)

  描述：比较两个hll_hashval类型数据是否不相等。

  返回值类型：bool

  示例：


```
openGauss=# SELECT hll_hashval_ne(hll_hash_integer(1), hll_hash_integer(1));
 hll_hashval_ne
----------------
 f
(1 row)
```

日志函数

hll主要存在三种模式Explicit，Sparse，Full。当数据规模比较小的时候会使用Explicit模式，这种模式下distinct值的计算是没有误差的；随着distinct值越来越多，hll会先后转换为Sparse模式和Full模式，这两种模式在计算结果上没有任何区别，只影响hll函数的计算效率和hll对象的存储空间。下面的函数可以用于查看hll的一些参数。

- hll_print(hll)

  描述：打印hll的一些debug参数信息。

  示例：

```
openGauss=# SELECT hll_print(hll_empty());
                                hll_print
-------------------------------------------------------------------------------
 type=1(HLL_EMPTY), log2m=14, log2explicit=10, log2sparse=12, duplicatecheck=0
(1 row)
```

- hll_type(hll)

  描述：查看当前hll的类型。返回值具体含义如下：返回值0，表示HLL_UNINIT，未初始化的hll对象；返回值1，表示HLL_EMPTY，hll空对象；返回值2，表示HLL_EXPLICIT，Explicit模式的hll对象；返回值3，表示HLL_SPARSE，Sparse模式的hll对象；返回值4，表示HLL_FULL，Full模式的hll对象；返回值5，表示HLL_UNDEFINED，不合法的hll对象。

  示例：

```
openGauss=# SELECT hll_type(hll_empty());
 hll_type
----------
        1
(1 row)
```

- hll_log2m(hll)

  描述：查看当前hll数据结构中的log2m数值，log2m是分桶数的对数值，此值会影响最后hll计算distinct误差率，误差率计算公式为±1.04/√(2 ^ log2m)。当显式指定log2m的取值为10-16之间时，hll会设置分桶数为2^{log2m}。当显示指定log2explicit为-1时，会采用内置默认值。

  示例：

```
openGauss=# SELECT hll_log2m(hll_empty());
 hll_log2m
-----------
        14
(1 row)

openGauss=# SELECT hll_log2m(hll_empty(10));
 hll_log2m
-----------
        10
(1 row)

openGauss=# SELECT hll_log2m(hll_empty(-1));
 hll_log2m
-----------
        14
(1 row)
```


- hll_log2explicit(hll)

  描述：查看当前hll数据结构中的log2explicit数值。hll通常会由Explicit模式到Sparse模式再到Full模式，这个过程称为promotion hierarchy策略。可以通过调整log2explicit值的大小改变策略，比如log2explicit为0的时候就会跳过Explicit模式而直接进入Sparse模式。当显式指定log2explicit的取值为1-12之间时，hll会在数据段长度超过$2^{log2explicit}$时转为Sparse模式。当显示指定log2explicit为-1时，会采用内置默认值。

  示例：

```
openGauss=# SELECT hll_log2explicit(hll_empty());
 hll_log2explicit
------------------
               10
(1 row)

openGauss=# SELECT hll_log2explicit(hll_empty(12, 8));
 hll_log2explicit
------------------
                8
(1 row)

openGauss=# SELECT hll_log2explicit(hll_empty(12, -1));
 hll_log2explicit
------------------
               10
(1 row)
```

- hll_log2sparse(hll)

  描述：查看当前hll数据结构中的log2sparse数值。hll通常会由Explicit模式到Sparse模式再到Full模式，这个过程称为promotion hierarchy策略。可以通过调整log2sparse值的大小改变策略，比如log2sparse为0的时候就会跳过Sparse模式而直接进入Full模式。当显式指定Sparse的取值为1-14之间时，hll会在数据段长度超过$2^{log2sparse}$时转为Full模式。当显示指定log2sparse为-1时，会采用内置默认值。

  示例：

```
openGauss=# SELECT hll_log2sparse(hll_empty());
 hll_log2sparse
----------------
             12
(1 row)

openGauss=# SELECT hll_log2sparse(hll_empty(12, 8, 10));
 hll_log2sparse
----------------
             10
(1 row)

openGauss=# SELECT hll_log2sparse(hll_empty(12, 8, -1));
 hll_log2sparse
----------------
             12
(1 row)
```

- hll_duplicatecheck(hll)

  描述：是否启用duplicatecheck，0是关闭，1是开启。默认关闭，对于有较多重复值出现的情况，可以开启以提高效率。当显示指定duplicatecheck为-1时，会采用内置默认值。

  示例：

```
openGauss=# SELECT hll_duplicatecheck(hll_empty());
 hll_duplicatecheck
--------------------
```



```
             0
(1 row)

openGauss=# SELECT hll_duplicatecheck(hll_empty(12, 8, 10, 1));
 hll_duplicatecheck
--------------------
             1
(1 row)

openGauss=# SELECT hll_duplicatecheck(hll_empty(12, 8, 10, -1));
 hll_duplicatecheck
--------------------
             0
(1 row)
```

功能函数

- hll_empty()

  描述：创建一个空的hll。

  返回值类型：hll

  示例：

```
openGauss=# SELECT hll_empty();
                                                  hll_empty
-------------------------------------------------------------
 \x484c4c00000000002b050000000000000000000000000000000000000000
(1 row)
```

- hll_empty(int32 log2m)

  描述：创建空的hll并指定参数log2m，取值范围是10到16。若输入-1，则采用内置默认值。

  返回值类型：hll

  示例：

```
openGauss=# SELECT hll_empty(10);
                                                  hll_empty
-------------------------------------------------------------
 \x484c4c00000000002b040000000000000000000000000000000000000000
(1 row)

openGauss=# SELECT hll_empty(-1);
                                                  hll_empty
-------------------------------------------------------------
 \x484c4c00000000002b050000000000000000000000000000000000000000
(1 row)
```

- hll_empty(int32 log2m, int32 log2explicit)

  描述：创建空的hll并依次指定参数log2m、log2explicit。log2explicit取值范围是0到12，0表示直接跳过Explicit模式。该参数可以用来设置Explicit模式的阈值大小，在数据段长度达到$2^{log2explicit}$后切换为Sparse模式或者Full模式。若输入-1，则log2explicit采用内置默认值。

  返回值类型: hll

  示例：

```
openGauss=# SELECT hll_empty(10, 4);
                                                  hll_empty
-------------------------------------------------------------
 \x484c4c0000000000130400000000000000000000000000000000000000
(1 row)

openGauss=# SELECT hll_empty(10, -1);
                                                  hll_empty
-------------------------------------------------------------
```



```
 \x484c4c00000000002b040000000000000000000000000000000000000000
(1 row)
```

- hll_empty(int32 log2m, int32 log2explicit, int64 log2sparse)

  描述：创建空的hll并依次指定参数log2m、log2explicit、log2sparse。log2sparse取值范围是0到14，0表示直接跳过Sparse模式。该参数可以用来设置Sparse模式的阈值大小，在数据段长度达到$2^{log2sparse}$后切换为Full模式。若输入-1，则log2sparse采用内置默认值。

  返回值类型：hll

  示例：

```
openGauss=# SELECT hll_empty(10, 4, 8);
                        hll_empty
------------------------------------------------------------
 \x484c4c0000000000120400000000000000000000000000000000000000
(1 row)

openGauss=# SELECT hll_empty(10, 4, -1);
                        hll_empty
------------------------------------------------------------
 \x484c4c0000000000130400000000000000000000000000000000000000
(1 row)
```

- hll_empty(int32 log2m, int32 log2explicit, int64 log2sparse, int32 duplicatecheck)

  描述：创建空的hll并依次指定参数log2m、log2explicit、log2sparse、duplicatecheck。duplicatecheck取0或者1，表示是否开启该模式，默认情况下该模式会关闭。若输入-1，则duplicatecheck采用内置默认值。

  返回值类型：hll

  示例：

```
openGauss=# SELECT hll_empty(10, 4, 8, 0);
                        hll_empty
------------------------------------------------------------
 \x484c4c0000000000120400000000000000000000000000000000000000
(1 row)

openGauss=# SELECT hll_empty(10, 4, 8, -1);
                        hll_empty
------------------------------------------------------------
 \x484c4c0000000000120400000000000000000000000000000000000000
(1 row)
```

- hll_add(hll, hll_hashval)

  描述：把hll_hashval加入到hll中。

  返回值类型：hll

  示例：

```
openGauss=# SELECT hll_add(hll_empty(), hll_hash_integer(1));
                                hll_add
------------------------------------------------------------------------
 \x484c4c08000002002b0900000000000000f03f3e2921ff133fbaed3e2921ff133fbaed00
(1 row)
```

- hll_add_rev(hll_hashval, hll)

  描述：把hll_hashval加入到hll中，和hll_add功能一样，只是参数位置进行了交换。

  返回值类型：hll

  示例：

```
openGauss=# SELECT hll_add_rev(hll_hash_integer(1), hll_empty());
                              hll_add_rev
```



```
--------------------------------------------------------------------------------
 \x484c4c08000002002b0900000000000000f03f3e2921ff133fbaed3e2921ff133fbaed00
(1 row)
```

- hll_eq(hll, hll)

  描述：比较两个hll是否相等。

  返回值类型：bool

  示例：

```
openGauss=# SELECT hll_eq(hll_add(hll_empty(), hll_hash_integer(1)), hll_add(hll_empty(),
hll_hash_integer(2)));
 hll_eq
--------
 f
(1 row)
```

- hll_ne(hll, hll)

  描述：比较两个hll是否不相等。

  返回值类型：bool

  示例：

```
openGauss=# SELECT hll_ne(hll_add(hll_empty(), hll_hash_integer(1)), hll_add(hll_empty(),
hll_hash_integer(2)));
 hll_ne
--------
 t
(1 row)
```

- hll_cardinality(hll)

  描述：计算hll的distinct值。

  返回值类型：int

  示例：

```
openGauss=# SELECT hll_cardinality(hll_empty() || hll_hash_integer(1));
 hll_cardinality
-----------------
               1
(1 row)
```

- hll_union(hll, hll)

  描述：把两个hll数据结构union成一个。

  返回值类型：hll

  示例：

```
openGauss=# SELECT hll_union(hll_add(hll_empty(), hll_hash_integer(1)), hll_add(hll_empty(),
hll_hash_integer(2)));
                                    hll_union
------------------------------------------------------------------------------------------

\x484c4c10002000002b0900000000000000004000000000000000000b3ccc49320cca1ae3e2921ff133fba
ed00
(1 row)
```

聚合函数

- hll_add_agg(hll_hashval)

  描述：把哈希后的数据按照分组放到hll中。

  返回值类型：hll

  示例：

```
--准备数据
openGauss=# CREATE TABLE t_id(id int);
```

```
openGauss=# INSERT INTO t_id VALUES(generate_series(1,500));
openGauss=# CREATE TABLE t_data(a int, c text);
openGauss=# INSERT INTO t_data SELECT mod(id,2), id FROM t_id;

--创建表并指定列为hll
openGauss=# CREATE TABLE t_a_c_hll(a int, c hll);

--根据a列group by对数据分组，把各组数据加到hll中
openGauss=# INSERT INTO t_a_c_hll SELECT a, hll_add_agg(hll_hash_text(c)) FROM t_data GROUP
BY a;

--得到每组数据中hll的Distinct值
openGauss=# SELECT a, #c as cardinality FROM t_a_c_hll ORDER BY a;
 a |   cardinality
---+------------------
 0 | 247.862354346299
 1 | 250.908710610377
(2 rows)
```

- hll_add_agg(hll_hashval, int32 log2m)

  描述：把哈希后的数据按照分组放到hll中， 并指定参数log2m，取值范围是10到16。若输入-1或者NULL，则采用内置默认值。

  返回值类型：hll

  示例：

```
openGauss=# SELECT hll_cardinality(hll_add_agg(hll_hash_text(c), 12)) FROM t_data;
 hll_cardinality
------------------
 497.965240179228
(1 row)
```

- hll_add_agg(hll_hashval, int32 log2m, int32 log2explicit)

  描述：把哈希后的数据按照分组放到hll中，依次指定参数log2m、log2explicit。log2explicit取值范围是0到12，0表示直接跳过Explicit模式。该参数可以用来设置Explicit模式的阈值大小，在数据段长度达到$2^{log2explicit}$后切换为Sparse模式或者Full模式。若输入-1或者NULL，则log2explicit采用内置默认值。

  返回值类型：hll

  示例：

```
openGauss=# SELECT hll_cardinality(hll_add_agg(hll_hash_text(c), NULL, 1)) FROM t_data;
 hll_cardinality
------------------
 498.496062953313
(1 row)
```

- hll_add_agg(hll_hashval, int32 log2m, int32 log2explicit, int64 log2sparse)

  描述：把哈希后的数据按照分组放到hll中， 依次指定参数log2m、log2explicit、log2sparse。，log2sparse取值范围是0到14，0表示直接跳过Sparse模式。该参数可以用来设置Sparse模式的阈值大小，在数据段长度达到$2^{log2sparse}$后切换为Full模式。若输入-1或者NULL，则log2sparse采用内置默认值。

  返回值类型：hll

  示例：

```
openGauss=# SELECT hll_cardinality(hll_add_agg(hll_hash_text(c), NULL, 6, 10)) FROM t_data;
 hll_cardinality
------------------
 498.496062953313
(1 row)
```

- hll_add_agg(hll_hashval, int32 log2m, int32 log2explicit, int64 log2sparse, int32 duplicatecheck)

  描述：把哈希后的数据按照分组放到hll中, 依次指定参数log2m、log2explicit、log2sparse、duplicatecheck，duplicatecheck取值范围是0或者1，表示是否开启

该模式，默认情况下该模式会关闭。若输入-1或者NULL，则duplicatecheck采用内置默认值。

返回值类型：hll

示例：

```
openGauss=# SELECT hll_cardinality(hll_add_agg(hll_hash_text(c), NULL, 6, 10, -1)) FROM t_data;
 hll_cardinality
------------------
 498.496062953313
(1 row)
```

- hll_union_agg(hll)

  描述：将多个hll类型数据union成一个hll。

  返回值类型：hll

  示例：

```
--将各组中的hll数据union成一个hll，并计算distinct值。
openGauss=# SELECT #hll_union_agg(c) as cardinality FROM t_a_c_hll;
   cardinality
------------------
 498.496062953313
(1 row)

--删除表
openGauss=# DROP TABLE t_id;
openGauss=# DROP TABLE t_data;
openGauss=# DROP TABLE t_a_c_hll;
```

  说明

  注意：当两个或者多个hll数据结构做union的时候，必须要保证其中每一个hll里面的精度参数一样，否则将不可以进行union。同样的约束也适用于函数hll_union(hll,hll)。

废弃函数

由于版本升级，HLL（HyperLogLog）有一些旧的函数废弃，用户可以用类似的函数进行替代。

- hll_schema_version(hll)

  描述：查看当前hll中的schema version。旧版本schema version是常值1，用来进行hll字段的头部校验，重构后的hll在头部增加字段“HLL”进行校验，schema version不再使用。

- hll_regwidth(hll)

  描述：查看hll数据结构中桶的位数大小。旧版本桶的位数regwidth取值1~5，会存在较大的误差，也限制了基数估计上限。 重构后regwidth为固定值6，不再使用regwidth变量。

- hll_expthresh(hll)

  描述：得到当前hll中expthresh大小。采用hll_log2explicit(hll)替代类似功能。

- hll_sparseon(hll)

  描述：是否启用Sparse模式。采用hll_log2sparse(hll)替代类似功能，0表示关闭Sparse模式。

内置函数

HLL（HyperLogLog）有一系列内置函数用于内部对数据进行处理，一般情况下用户不需要熟知这些函数的使用。详情见**表7-35**。

表 7-35 内置函数

函数名称	功能描述
hll_in	以string格式接收hll数据。
hll_out	以string格式发送hll数据。
hll_recv	以bytea格式接收hll数据。
hll_send	以bytea格式发送hll数据。
hll_trans_in	以string格式接收hll_trans_type数据。
hll_trans_out	以string格式发送hll_trans_type数据。
hll_trans_recv	以bytea形式接收hll_trans_type数据。
hll_trans_send	以bytea形式发送hll_trans_type数据。
hll_typmod_in	接收typmod类型数据。
hll_typmod_out	发送typmod类型数据。
hll_hashval_in	接收hll_hashval类型数据。
hll_hashval_out	发送hll_hashval类型数据。
hll_add_trans0	类似于hll_add所提供的功能，初始化时无指定入参，通常在分布式聚合运算的第一阶段DN上使用。
hll_add_trans1	类似于hll_add所提供的功能，初始化时指定一个入参，通常在分布式聚合运算的第一阶段DN上使用。
hll_add_trans2	类似于hll_add所提供的功能，初始化时指定两个入参，通常在分布式聚合运算的第一阶段DN上使用。
hll_add_trans3	类似于hll_add所提供的功能，初始化时指定三个入参，通常在分布式聚合运算的第一阶段DN上使用。
hll_add_trans4	类似于hll_add所提供的功能，初始化时指定四个入参，通常在分布式聚合运算的第一阶段DN上使用。
hll_union_trans	类似hll_union所提供的功能，在分布式聚合运算的第一阶段DN上使用。
hll_union_collect	类似于hll_union所提供的功能，在分布式聚合运算第二阶段CN上使用，汇总各个DN上的结果。
hll_pack	在分布式聚合运算第三阶段CN上使用，把自定义hll_trans_type类型最后转换成hll类型。
hll	用于hll类型转换成hll类型，根据输入参数会设定指定参数。
hll_hashval	用于bigint类型转换成hll_hashval类型。
hll_hashval_int4	用于int4类型转换成hll_hashval类型。

操作符

- =

  描述：比较hll或hll_hashval的值是否相等。

  返回值类型：bool

  示例：

```
--hll
openGauss=# SELECT (hll_empty() || hll_hash_integer(1)) = (hll_empty() || hll_hash_integer(1));
column
----------
 t
(1 row)

--hll_hashval
openGauss=# SELECT hll_hash_integer(1) = hll_hash_integer(1);
 ?column?
----------
 t
(1 row)
```

- <> or !=

  描述：比较hll或hll_hashval是否不相等。

  返回值类型：bool

  示例：

```
--hll
openGauss=# SELECT (hll_empty() || hll_hash_integer(1)) <> (hll_empty() || hll_hash_integer(2));
 ?column?
----------
 t
(1 row)

--hll_hashval
openGauss=# SELECT hll_hash_integer(1) <> hll_hash_integer(2);
 ?column?
----------
 t
(1 row)
```

- ||

  描述：可代表hll_add, hll_union, hll_add_rev三个函数的功能。

  返回值类型：hll

  示例：

```
--hll_add
openGauss=# SELECT hll_empty() || hll_hash_integer(1);
                                ?column?
--------------------------------------------------------------------------------
 \x484c4c08000002002b0900000000000000f03f3e2921ff133fbaed3e2921ff133fbaed00
(1 row)

--hll_add_rev
openGauss=# SELECT hll_hash_integer(1) || hll_empty();
                                ?column?
--------------------------------------------------------------------------------
 \x484c4c08000002002b0900000000000000f03f3e2921ff133fbaed3e2921ff133fbaed00
(1 row)

--hll_union
openGauss=# SELECT (hll_empty() || hll_hash_integer(1)) || (hll_empty() || hll_hash_integer(2));
                                      ?column?
---------------------------------------------------------------------------------------------

\x484c4c10002000002b0900000000000000004000000000000000000b3ccc49320cca1ae3e2921ff133fba
```



```
ed00
(1 row)
```

- #

  描述：计算出hll的Dintinct值, 同hll_cardinality函数。

  返回值类型：int

  示例：

```
openGauss=# SELECT #(hll_empty() || hll_hash_integer(1));
 ?column?
----------
        1
(1 row)
```

7.5.14 SEQUENCE 函数

序列函数为用户从序列对象中获取后续的序列值提供了简单的多用户安全的方法。

- nextval(regclass)

  描述：递增序列并返回新值。

  说明

  - 为了避免从同一个序列获取值的并发事务被阻塞， nextval操作不会回滚；也就是说，一旦一个值已经被抓取， 那么就认为它已经被用过了，并且不会再被返回。 即使该操作处于事务中，当事务之后中断，或者如果调用查询结束不使用该值，也是如此。这种情况将在指定值的顺序中留下未使用的"空洞"。 因此，GaussDB序列对象不能用于获得"无间隙"序列。
  - 如果nextval被下推到DN上时，各个DN会自动连接GTM，请求next values值，例如（insert into t1 select xxx，t1某一列需要调用nextval函数），由于GTM上有最大连接数为8192的限制，而这类下推语句会导致消耗过多的GTM连接数，因此对于这类语句的并发数目限制为7000（其它语句需要占用部分连接）/集群DN数目。

  返回类型：numeric

  nextval函数有两种调用方式（其中第二种调用方式兼容Oracle的语法，目前不支持Sequence命名中有特殊字符"."的情况），如下：

```
openGauss=# CREATE SEQUENCE seqDemo;

--示例1：
openGauss=# SELECT nextval('seqDemo');
 nextval
---------
       1
(1 row)

--示例2：
openGauss=# SELECT seqDemo.nextval;
 nextval
---------
       2
(1 row)
openGauss=# DROP SEQUENCE seqDemo;
```

- currval(regclass)

  描述：返回当前会话里最近一次nextval返回的数值。如果当前会话还没有调用过指定的sequence的nextval，那么调用currval将会报错。需要注意的是，这个函数在默认情况下是不支持的，需要通过设置enable_beta_features为true之后，才能使用这个函数。同时在设置enable_beta_features为true之后，nextval()函数将不支持下推。

  返回类型：numeric

currval函数有两种调用方式（其中第二种调用方式兼容Oracle的语法，目前不支持Sequence命名中有特殊字符"."的情况），如下：

```
openGauss=# CREATE SEQUENCE seq1;
openGauss=# SELECT nextval('seq1');
openGauss=# SET enable_beta_features = true;

--示例1：
openGauss=# SELECT currval('seq1');
 currval
---------
       1
(1 row)

--示例2：
openGauss=# SELECT seq1.currval seq1;
 seq1
------
    1
(1 row)
openGauss=# DROP SEQUENCE seq1;
openGauss=# SET enable_beta_features = false;
```

- lastval()

  描述：返回当前会话里最近一次nextval返回的数值。这个函数等效于currval，只是它不用序列名为参数，它抓取当前会话里面最近一次nextval使用的序列。如果当前会话还没有调用过nextval，那么调用lastval将会报错。

  需要注意的是，这个函数在默认情况下是不支持的，需要通过设置enable_beta_features或者lastval_supported为true之后，才能使用这个函数。同时这种情况下，nextval()函数将不支持下推。

  返回类型：numeric

  示例：

  ```
  openGauss=# CREATE SEQUENCE seq1;
  openGauss=# SELECT nextval('seq1');
  openGauss=# SET enable_beta_features = true;
  openGauss=# SELECT lastval();
   lastval
  ---------
         1
  (1 row)
  openGauss=# DROP SEQUENCE seq1;
  openGauss=# SET enable_beta_features = false;
  ```

- setval(regclass, bigint)

  描述：设置序列的当前数值。

  返回类型：numeric

  示例：

  ```
  openGauss=# CREATE SEQUENCE seqDemo;
  openGauss=# SELECT nextval('seqDemo');
  openGauss=# SELECT setval('seqDemo',3);
   setval
  --------
       3
  (1 row)
  openGauss=# DROP SEQUENCE seqDemo;
  ```

- setval(regclass, numeric, Boolean)

  描述：设置序列的当前数值以及is_called标志。

  返回类型：numeric

  示例：

  ```
  openGauss=# CREATE SEQUENCE seqDemo;
  openGauss=# SELECT nextval('seqDemo');
  ```



```
openGauss=# SELECT setval('seqDemo',5,true);
 setval
--------
      5
(1 row)
openGauss=# DROP SEQUENCE seqDemo;
```

说明

Setval后当前会话及GTM上会立刻生效，但如果其他会话有缓存的序列值，只能等到缓存值用尽才能感知Setval的作用。所以为了避免序列值冲突，setval要谨慎使用。

因为序列是非事务的，setval造成的改变不会由于事务的回滚而撤销。

- pg_sequence_last_value(sequence_oid oid, OUT cache_value int16, OUT last_value int16)

  描述：获取指定sequence的参数，包含缓存值，当前值。

  返回类型：int16，int16

7.5.15 数组函数和操作符

数组操作符

- =

  描述：两个数组是否相等

  示例：

```
openGauss=# SELECT ARRAY[1.1,2.1,3.1]::int[] = ARRAY[1,2,3] AS RESULT ;
 result
--------
 t
(1 row)
```

- <>

  描述：两个数组是否不相等

  示例：

```
openGauss=# SELECT ARRAY[1,2,3] <> ARRAY[1,2,4] AS RESULT;
 result
--------
 t
(1 row)
```

- <

  描述：一个数组是否小于另一个数组

  示例：

```
openGauss=# SELECT ARRAY[1,2,3] < ARRAY[1,2,4] AS RESULT;
 result
--------
 t
(1 row)
```

- >

  描述：一个数组是否大于另一个数组

  示例：

```
openGauss=# SELECT ARRAY[1,4,3] > ARRAY[1,2,4] AS RESULT;
 result
--------
 t
(1 row)
```


- <=

  描述：一个数组是否小于或等于另一个数组

  示例：

```
openGauss=# SELECT ARRAY[1,2,3] <= ARRAY[1,2,3] AS RESULT;
 result
--------
 t
(1 row)
```

- >=

  描述：一个数组是否大于或等于另一个数组

  示例：

```
openGauss=# SELECT ARRAY[1,4,3] >= ARRAY[1,4,3] AS RESULT;
 result
--------
 t
(1 row)
```

- @>

  描述：一个数组是否包含另一个数组

  示例：

```
openGauss=# SELECT ARRAY[1,4,3] @> ARRAY[3,1] AS RESULT;
 result
--------
 t
(1 row)
```

- <@

  描述：一个数组是否被包含于另一个数组

  示例：

```
openGauss=# SELECT ARRAY[2,7] <@ ARRAY[1,7,4,2,6] AS RESULT;
 result
--------
 t
(1 row)
```

- &&

  描述：一个数组是否和另一个数组重叠（有共同元素）

  示例：

```
openGauss=# SELECT ARRAY[1,4,3] && ARRAY[2,1] AS RESULT;
 result
--------
 t
(1 row)
```

- ||

  描述：数组与数组进行连接

  示例：

```
openGauss=# SELECT ARRAY[1,2,3] || ARRAY[4,5,6] AS RESULT;
   result
---------------
 {1,2,3,4,5,6}
(1 row)
openGauss=# SELECT ARRAY[1,2,3] || ARRAY[[4,5,6],[7,8,9]] AS RESULT;
         result
-----------------------------
 {{1,2,3},{4,5,6},{7,8,9}}
(1 row)
```


- ||
  描述：元素与数组进行连接
  示例：

```
openGauss=# SELECT 3 || ARRAY[4,5,6] AS RESULT;
  result
-----------
 {3,4,5,6}
(1 row)
```

- ||
  描述：数组与元素进行连接
  示例：

```
openGauss=# SELECT ARRAY[4,5,6] || 7 AS RESULT;
  result
-----------
 {4,5,6,7}
(1 row)
```

数组比较是使用默认的B-tree比较函数对所有元素逐一进行比较的。多维数组的元素按照行顺序进行访问。如果两个数组的内容相同但维数不等，决定排序顺序的首要因素是维数。

数组函数

- array_append(anyarray, anyelement)
  描述：向数组末尾添加元素，只支持一维数组。
  返回类型：anyarray
  示例：

```
openGauss=# SELECT array_append(ARRAY[1,2], 3) AS RESULT;
 result
---------
 {1,2,3}
(1 row)
```

- array_prepend(anyelement, anyarray)
  描述：向数组开头添加元素，只支持一维数组。
  返回类型：anyarray
  示例：

```
openGauss=# SELECT array_prepend(1, ARRAY[2,3]) AS RESULT;
 result
---------
 {1,2,3}
(1 row)
```

- array_cat(anyarray, anyarray)
  描述：连接两个数组，支持多维数组。
  返回类型：anyarray
  示例：

```
openGauss=# SELECT array_cat(ARRAY[1,2,3], ARRAY[4,5]) AS RESULT;
   result
-------------
 {1,2,3,4,5}
(1 row)

openGauss=# SELECT array_cat(ARRAY[[1,2],[4,5]], ARRAY[6,7]) AS RESULT;
      result
---------------------
```



```
 {{1,2},{4,5},{6,7}}
(1 row)
```

- array_union(anyarray, anyarray)

  描述：连接两个数组，只支持一维数组。

  返回类型：anyarray

  示例：

```
openGauss=# SELECT array_union(ARRAY[1,2,3], ARRAY[3,4,5]) AS RESULT;
  result
-------------
 {1,2,3,3,4,5}
(1 row)
```

- array_union_distinct(anyarray, anyarray)

  描述：连接两个数组，并去重，只支持一维数组。

  返回类型：anyarray

  示例：

```
openGauss=# SELECT array_union_distinct(ARRAY[1,2,3], ARRAY[3,4,5]) AS RESULT;
  result
-------------
 {1,2,3,4,5}
(1 row)
```

- array_intersect(anyarray, anyarray)

  描述：两个数组取交集，只支持一维数组。

  返回类型：anyarray

  示例：

```
openGauss=# SELECT array_intersect(ARRAY[1,2,3], ARRAY[3,4,5]) AS RESULT;
  result
-------------
 {3}
(1 row)
```

- array_intersect_distinct(anyarray, anyarray)

  描述：两个数组取交集，并去重，只支持一维数组。

  返回类型：anyarray

  示例：

```
openGauss=# SELECT array_intersect_distinct(ARRAY[1,2,2], ARRAY[2,2,4,5]) AS RESULT;
  result
-------------
 {2}
(1 row)
```

- array_except(anyarray, anyarray)

  描述：两个数组取差，只支持一维数组。

  返回类型：anyarray

  示例：

```
openGauss=# SELECT array_except(ARRAY[1,2,3], ARRAY[3,4,5]) AS RESULT;
  result
-------------
 {1,2}
(1 row)
```

- array_except_distinct(anyarray, anyarray)

  描述：两个数组取差，并去重，只支持一维数组。

  返回类型：anyarray

示例：

```
openGauss=# SELECT array_except_distinct(ARRAY[1,2,2,3], ARRAY[3,4,5]) AS RESULT;
  result
-------------
 {1,2}
(1 row)
```

- array_ndims(anyarray)

  描述：返回数组的维数。

  返回类型：int

  示例：

  ```
  openGauss=# SELECT array_ndims(ARRAY[[1,2,3], [4,5,6]]) AS RESULT;
   result
  --------
       2
  (1 row)
  ```

- array_dims(anyarray)

  描述：返回数组维数的文本表示。

  返回类型：text

  示例：

  ```
  openGauss=# SELECT array_dims(ARRAY[[1,2,3], [4,5,6]]) AS RESULT;
    result
  ------------
   [1:2][1:3]
  (1 row)
  ```

- array_length(anyarray, int)

  描述：返回数组维度的长度。

  返回类型：int

  示例：

  ```
  openGauss=# SELECT array_length(array[1,2,3], 1) AS RESULT;
   result
  --------
       3
  (1 row)
  ```

- array_lower(anyarray, int)

  描述：返回数组维数的下界。

  返回类型：int

  示例：

  ```
  openGauss=# SELECT array_lower('[0:2]={1,2,3}'::int[], 1) AS RESULT;
   result
  --------
       0
  (1 row)
  ```

- array_sort(anyarray)

  描述：返回从小到大排列好的数组。

  返回类型：anyarray

  示例：

  ```
  openGauss=# SELECT array_sort(ARRAY[5,1,3,6,2,7]) AS RESULT;
   result
  -------------
  {1,2,3,5,6,7}
  (1 row)
  ```


- array_upper(anyarray, int)

  描述：返回数组维数的上界。

  返回类型：int

  示例：

```
openGauss=# SELECT array_upper(ARRAY[1,8,3,7], 1) AS RESULT;
 result
--------
      4
(1 row)
```

- array_to_string(anyarray, text [, text])

  描述：使用第一个text作为数组的新分隔符，使用第二个text替换数组值为null的值。

  返回类型：text

  示例：

```
openGauss=# SELECT array_to_string(ARRAY[1, 2, 3, NULL, 5], ',', '*') AS RESULT;
  result
-----------
 1,2,3,*,5
(1 row)
```

- array_delete(anyarray)

  描述：清空数组中的元素并返回一个同类型的空数组。

  返回类型：anyarray

  示例：

```
openGauss=# SELECT array_delete(ARRAY[1,8,3,7]) AS RESULT;
 result
--------
 {}
(1 row)
```

- array_deleteidx(anyarray, int)

  描述：从数组中删除指定下标的元素并返回剩余元素组成的数组。

  返回类型：anyarray

  示例：

```
openGauss=# SELECT array_deleteidx(ARRAY[1,2,3,4,5], 1) AS RESULT;
  result
-----------
 {2,3,4,5}
(1 row)
```

- array_extendnull(anyarray, int)

  描述：往数组尾部添加指定个数空元素。

  返回类型：anyarray

  示例：

```
openGauss=# SELECT array_extend(ARRAY[1,8,3,7],1) AS RESULT;
    result
--------------
{1,8,3,7,null}
(1 row)
```

- array_trim(anyarray, int)

  描述：从数组尾部删除指定个数个元素。

  返回类型：anyarray

  示例：


```
openGauss=# SELECT array_trim(ARRAY[1,8,3,7],1) AS RESULT;
 result
---------
 {1,8,3}
(1 row)
```

- array_exists(anyarray, int)

  描述：检查第二个参数是否是数组的合法下标。

  返回类型：boolean

  示例：

```
openGauss=# SELECT array_exists(ARRAY[1,8,3,7],1) AS RESULT;
 result
--------
 t
(1 row)
```

- array_next(anyarray, int)

  描述：根据第二个入参返回数组中指定下标元素的下一个元素的下标。

  返回类型：int

  示例：

```
openGauss=# SELECT array_next(ARRAY[1,8,3,7],1) AS RESULT;
 result
--------
      2
(1 row)
```

- array_prior(anyarray, int)

  描述：根据第二个入参返回数组中指定下标元素的上一个元素的下标。

  返回类型：int

  示例：

```
openGauss=# SELECT array_prior(ARRAY[1,8,3,7],2) AS RESULT;
 result
--------
  1
(1 row)
```

- string_to_array(text, text [, text])

  描述：使用第二个text指定分隔符，使用第三个可选的text作为NULL值替换模板，如果分隔后的子串与第三个可选的text完全匹配，则将其替换为NULL。

  返回类型：text[]

  示例：

```
openGauss=# SELECT string_to_array('xx~^~yy~^~zz', '~^~', 'yy') AS RESULT;
    result
--------------
 {xx,NULL,zz}
(1 row)
openGauss=# SELECT string_to_array('xx~^~yy~^~zz', '~^~', 'y') AS RESULT;
   result
------------
 {xx,yy,zz}
(1 row)
```

- unnest(anyarray)

  描述：扩大一个数组为一组行。

  返回类型：setof anyelement

  示例：

```
openGauss=# SELECT unnest(ARRAY[1,2]) AS RESULT;
 result
```



```
--------
     1
     2
(2 rows)
```

在string_to_array中，如果分隔符参数是NULL，输入字符串中的每个字符将在结果数组中变成一个独立的元素。如果分隔符是一个空白字符串，则整个输入的字符串将变为一个元素的数组。否则输入字符串将在每个分隔字符串处分开。

在string_to_array中，如果省略null字符串参数或为NULL，将字符串中没有输入内容的子串替换为NULL。

在array_to_string中，如果省略null字符串参数或为NULL，运算中将跳过在数组中的任何null元素，并且不会在输出字符串中出现。

- _pg_keysequal

  描述：判断两个smallint数组是否相同。

  参数：smallint[], smallint[]

  返回值类型：boolean

  说明

  此函数存在于information_schema命名空间。

7.5.16 范围函数和操作符

范围操作符

- =

  描述：等于

  示例：

```
openGauss=# SELECT int4range(1,5) = '[1,4]'::int4range AS RESULT;
 result
--------
 t
(1 row)
```

- <>

  描述：不等于

  示例：

```
openGauss=# SELECT numrange(1.1,2.2) <> numrange(1.1,2.3) AS RESULT;
 result
--------
 t
(1 row)
```

- <

  描述：小于

  示例：

```
openGauss=# SELECT int4range(1,10) < int4range(2,3) AS RESULT;
 result
--------
 t
(1 row)
```

- >

  描述：大于

示例：

```
openGauss=# SELECT int4range(1,10) > int4range(1,5) AS RESULT;
 result
--------
 t
(1 row)
```

- <=

  描述：小于或等于

  示例：

```
openGauss=# SELECT numrange(1.1,2.2) <= numrange(1.1,2.2) AS RESULT;
 result
--------
 t
(1 row)
```

- >=

  描述：大于或等于

  示例：

```
openGauss=# SELECT numrange(1.1,2.2) >= numrange(1.1,2.0) AS RESULT;
 result
--------
 t
(1 row)
```

- @>

  描述：包含范围

  示例：

```
openGauss=# SELECT int4range(2,4) @> int4range(2,3) AS RESULT;
 result
--------
 t
(1 row)
```

- @>

  描述：包含元素

  示例：

```
openGauss=# SELECT '[2011-01-01,2011-03-01)'::tsrange @> '2011-01-10'::timestamp AS RESULT;
 result
--------
 t
(1 row)
```

- <@

  描述：范围包含于

  示例：

```
openGauss=# SELECT int4range(2,4) <@ int4range(1,7) AS RESULT;
 result
--------
 t
(1 row)
```

- <@

  描述：元素包含于

  示例：

```
openGauss=# SELECT 42 <@ int4range(1,7) AS RESULT;
 result
--------
 f
(1 row)
```


- &&

  描述：重叠（有共同点）

  示例：

```
openGauss=# SELECT int8range(3,7) && int8range(4,12) AS RESULT;
 result
--------
 t
(1 row)
```

- <<

  描述：范围值是否比另一个范围值的最小值还小（没有交集）

  示例：

```
openGauss=# SELECT int8range(1,10) << int8range(100,110) AS RESULT;
 result
--------
 t
(1 row)
```

- >>

  描述：范围值是否比另一个范围值的最大值还大（没有交集）

  示例：

```
openGauss=# SELECT int8range(50,60) >> int8range(20,30) AS RESULT;
 result
--------
 t
(1 row)
```

- &<

  描述：范围值的最大值是否不超过另一个范围值的最大值。

  示例：

```
openGauss=# SELECT int8range(1,20) &< int8range(18,20) AS RESULT;
 result
--------
 t
(1 row)
```

- &>

  描述：范围值的最小值是否不小于另一个范围值的最小值。

  示例：

```
openGauss=# SELECT int8range(7,20) &> int8range(5,10) AS RESULT;
 result
--------
 t
(1 row)
```

- -|-

  描述：相邻

  示例：

```
openGauss=# SELECT numrange(1.1,2.2) -|- numrange(2.2,3.3) AS RESULT;
 result
--------
 t
(1 row)
```

- +

  描述：并集

  示例：


```
openGauss=# SELECT numrange(5,15) + numrange(10,20) AS RESULT;
 result
--------
 [5,20)
(1 row)
```

- *

  描述：交集

  示例：

```
openGauss=# SELECT int8range(5,15) * int8range(10,20) AS RESULT;
 result
---------
 [10,15)
(1 row)
```

- -

  描述：差集

  示例：

```
openGauss=# SELECT int8range(5,15) - int8range(10,20) AS RESULT;
 result
--------
 [5,10)
(1 row)
```

简单的比较操作符<，>，<=和>=先比较下界，只有下界相等时才比较上界。

<<、>>和-|-操作符当包含空范围时也会返回false；也就是，不认为空范围在其他范围之前或之后。

并集和差集操作符的执行结果无法包含两个不相交的子范围。

范围函数

如果范围是空或者需要的界限是无穷的，lower和upper函数将返回null。lower_inc、upper_inc、lower_inf和upper_inf函数均对空范围返回false。

- numrange(numeric, numeric, [text])

  描述：表示一个范围。

  返回类型：范围元素类型

  示例：

```
openGauss=# SELECT numrange(1.1,2.2) AS RESULT;
 result
--------
[1.1,2.2)
(1 row)
openGauss=# SELECT numrange(1.1,2.2, '()') AS RESULT;
 result
--------
(1.1,2.2)
(1 row)
```

- lower(anyrange)

  描述：范围的下界。

  返回类型：范围元素类型

  示例：

```
openGauss=# SELECT lower(numrange(1.1,2.2)) AS RESULT;
 result
--------
```



```
   1.1
(1 row)
```

- upper(anyrange)

  描述：范围的上界

  返回类型：范围元素类型

  示例：

```
openGauss=# SELECT upper(numrange(1.1,2.2)) AS RESULT;
 result
--------
   2.2
(1 row)
```

- isempty(anyrange)

  描述：范围是否为空

  返回类型：Boolean

  示例：

```
openGauss=# SELECT isempty(numrange(1.1,2.2)) AS RESULT;
 result
--------
 f
(1 row)
```

- lower_inc(anyrange)

  描述：是否包含下界

  返回类型：Boolean

  示例：

```
openGauss=# SELECT lower_inc(numrange(1.1,2.2)) AS RESULT;
 result
--------
 t
(1 row)
```

- upper_inc(anyrange)

  描述：是否包含上界

  返回类型：Boolean

  示例：

```
openGauss=# SELECT upper_inc(numrange(1.1,2.2)) AS RESULT;
 result
--------
 f
(1 row)
```

- lower_inf(anyrange)

  描述：下界是否为无穷

  返回类型：Boolean

  示例：

```
openGauss=# SELECT lower_inf('(,)'::daterange) AS RESULT;
 result
--------
 t
(1 row)
```

- upper_inf(anyrange)

  描述：上界是否为无穷

  返回类型：Boolean

示例：

```
openGauss=# SELECT upper_inf('(,)'::daterange) AS RESULT;
 result
--------
 t
(1 row)
```

- elem_contained_by_range(anyelement, anyrange)

  描述：判断元素是否在范围内。

  返回类型：Boolean

  示例：

```
openGauss=# SELECT elem_contained_by_range('2', numrange(1.1,2.2));
 elem_contained_by_range
-------------------------
 t
(1 row)
```

7.5.17 聚集函数

聚集函数

- sum(expression)

  描述：所有输入行的expression总和。

  返回类型：

  通常情况下输入数据类型和输出数据类型是相同的，但以下情况会发生类型转换：

  - 对于SMALLINT或INT输入，输出类型为BIGINT。
  - 对于BIGINT输入，输出类型为NUMBER 。
  - 对于浮点数输入，输出类型为DOUBLE PRECISION。

  示例：

```
openGauss=# CREATE TABLE tab(a int);
CREATE TABLE
openGauss=# INSERT INTO tab values(1);
INSERT 0 1
openGauss=# INSERT INTO tab values(2);
INSERT 0 1
openGauss=# SELECT sum(a) FROM tab;
 sum
-----
   3
(1 row)
```

- max(expression)

  描述：所有输入行中expression的最大值。

  参数类型：任意数组、数值、字符串、日期/时间类型。

  返回类型：与参数数据类型相同

  示例：

```
openGauss=# CREATE TABLE max_t1(a int, b int);

openGauss=# INSERT INTO max_t1 VALUES(1,2),(2,3),(3,4),(4,5);

openGauss=# SELECT MAX(a) FROM max_t1;
 max
-----
   4
```



```
(1 row)

openGauss=# DROP TABLE max_t1;
```

- min(expression)

  描述：所有输入行中expression的最小值。

  参数类型：任意数组、数值、字符串、日期/时间类型。

  返回类型：与参数数据类型相同

  示例：

```
openGauss=# CREATE TABLE min_t1(a int, b int);

openGauss=# INSERT INTO min_t1 VALUES(1,2),(2,3),(3,4),(4,5);

openGauss=# SELECT MIN(a) FROM min_t1;
 min
-----
   1
(1 row)

openGauss=# DROP TABLE min_t1;
```

- avg(expression)

  描述：所有输入值的均值（算术平均）。

  返回类型：

  对于任何整数类型输入，结果都是NUMBER类型。

  对于任何浮点输入，结果都是DOUBLE PRECISION类型。

  否则和输入数据类型相同。

  示例：

```
openGauss=# CREATE TABLE avg_t1(a int, b int);

openGauss=# INSERT INTO avg_t1 VALUES(1,2),(2,3),(3,4),(4,5);

openGauss=# SELECT AVG(a) FROM avg_t1;
        avg
--------------------
 2.5000000000000000
(1 row)

openGauss=# DROP TABLE avg_t1;
```

- count(expression)

  描述：返回表中满足expression不为NULL的行数。

  返回类型：BIGINT

  示例：

```
openGauss=# CREATE TABLE count_t1(a int, b int);

openGauss=# INSERT INTO count_t1 VALUES (NULL,1),(1,2),(2,3),(3,4),(4,5);

openGauss=# SELECT COUNT(a) FROM count_t1;
 count
-------
     4
(1 row)

openGauss=# DROP TABLE count_t1;
```

- count(*)

  描述：返回表中的记录行数。

  返回类型：BIGINT

示例：

```
openGauss=# CREATE TABLE count_t1(a int, b int);

openGauss=# INSERT INTO count_t1 VALUES (NULL,1),(1,2),(2,3),(3,4),(4,5);

openGauss=# SELECT COUNT(*) FROM count_t1;
 count
-------
     5
(1 row)

openGauss=# DROP TABLE count_t1;
```

- array_agg(expression)

  描述：将所有输入值（包括空）连接成一个数组。

  返回类型：参数类型的数组

  示例：

```
openGauss=# CREATE TABLE array_agg_t1(a int, b int);

openGauss=# INSERT INTO array_agg_t1 VALUES (NULL,1),(1,2),(2,3),(3,4),(4,5);

openGauss=# SELECT ARRAY_AGG(a) FROM array_agg_t1;
   array_agg
----------------
 {NULL,1,2,3,4}
(1 row)

openGauss=# DROP TABLE array_agg_t1;
```

- string_agg(expression, delimiter)

  描述：将输入值连接成为一个字符串，用分隔符分开。

  返回类型：和参数数据类型相同。

  示例：

```
openGauss=# CREATE TABLE string_agg_t1(a int, b int);

openGauss=# INSERT INTO string_agg_t1 VALUES (NULL,1),(1,2),(2,3),(3,4),(4,5);

openGauss=# SELECT STRING_AGG(a,';') FROM string_agg_t1;
 string_agg
------------
 1;2;3;4
(1 row)

openGauss=# DROP TABLE string_agg_t1;
```

- listagg(expression [, delimiter]) WITHIN GROUP(ORDER BY order-list)

  描述：将聚集列数据按WITHIN GROUP指定的排序方式排列，并用delimiter指定的分隔符拼接成一个字符串。

  - expression：必选。指定聚集列名或基于列的有效表达式，不支持DISTINCT关键字和VARIADIC参数。
  - delimiter：可选。指定分隔符，可以是字符串常数或基于分组列的确定性表达式，缺省时表示分隔符为空。
  - order-list：必选。指定分组内的排序方式。

  返回类型：text

  说明

  listagg是兼容Oracle 11g2的列转行聚集函数，可以指定OVER子句用作窗口函数。为了避免与函数本身WITHIN GROUP子句的ORDER BY造成二义性，listagg用作窗口函数时，OVER子句不支持ORDER BY的窗口排序或窗口框架。

示例：

聚集列是文本字符集类型。

```
openGauss=# CREATE TABLE listagg_t1(a int, b text);

openGauss=# INSERT INTO listagg_t1 VALUES (NULL,'a1'),(1,'b2'),(1,'c3'),(2,'d4'),(2,'e5'),(3,'f6');

openGauss=# SELECT a,LISTAGG(b,';') WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
 a | listagg
---+---------
 1 | b2;c3
 2 | d4;e5
 3 | f6
   | a1
(4 rows)

openGauss=# DROP TABLE listagg_t1;
```

聚集列是整型。

```
openGauss=# CREATE TABLE listagg_t1(a int, b int);

openGauss=# INSERT INTO listagg_t1 VALUES (NULL,1),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT a,LISTAGG(b,';') WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
 a | listagg
---+---------
 1 | 2;3
 2 | 4;5
 3 | 6
   | 1
(4 rows)

openGauss=# DROP TABLE listagg_t1;
```

聚集列是浮点类型。

```
openGauss=# CREATE TABLE listagg_t1(a int, b float);

openGauss=# INSERT INTO listagg_t1 VALUES (NULL,1.111),(1,2.222),(1,3.333),(2,4.444),(2,5.555),
(3,6.666);

openGauss=# SELECT a,LISTAGG(b,';') WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
 a |      listagg
---+-------------------
 1 | 2.222000;3.333000
 2 | 4.444000;5.555000
 3 | 6.666000
   | 1.111000
(4 rows)

openGauss=# DROP TABLE listagg_t1;
```

聚集列是时间类型。

```
openGauss=# CREATE TABLE listagg_t1(a int, b timestamp);

openGauss=# INSERT INTO listagg_t1 VALUES (NULL,'2000-01-01'),(1,'2000-02-02'),(1,'2000-03-03'),
(2,'2000-04-04'),(2,'2000-05-05'),(3,'2000-06-06');

openGauss=# SELECT a,LISTAGG(b,';') WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
 a |                 listagg
---+-----------------------------------------
 1 | 2000-02-02 00:00:00;2000-03-03 00:00:00
 2 | 2000-04-04 00:00:00;2000-05-05 00:00:00
 3 | 2000-06-06 00:00:00
   | 2000-01-01 00:00:00
(4 rows)

openGauss=# DROP TABLE listagg_t1;
```

聚集列是时间间隔类型。


```
openGauss=# CREATE TABLE listagg_t1(a int, b interval);

openGauss=# INSERT INTO listagg_t1 VALUES (NULL,'1 days'),(1,'2 days'),(1,'3 days'),(2,'4 days'),(2,'5
days'),(3,'6 days');

openGauss=# SELECT a,LISTAGG(b,';') WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
 a |    listagg
---+---------------
 1 | 2 days;3 days
 2 | 4 days;5 days
 3 | 6 days
   | 1 day
(4 rows)

openGauss=# DROP TABLE listagg_t1;
```

分隔符缺省时，默认为空。

```
openGauss=# CREATE TABLE listagg_t1(a int, b interval);

openGauss=# INSERT INTO listagg_t1 VALUES (NULL,'1 days'),(1,'2 days'),(1,'3 days'),(2,'4 days'),(2,'5
days'),(3,'6 days');

openGauss=# SELECT a,LISTAGG(b) WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
 a |   listagg
---+---------------
 1 | 2 days3 days
 2 | 4 days5 days
 3 | 6 days
   | 1 day
(4 rows)

openGauss=# DROP TABLE listagg_t1;
```

listagg作为窗口函数时，OVER子句不支持ORDER BY的窗口排序，listagg列为对应分组的有序聚集。

```
openGauss=# CREATE TABLE listagg_t1(a int, b interval);

openGauss=# INSERT INTO listagg_t1 VALUES (NULL,'1 days'),(1,'2 days'),(1,'3 days'),(2,'4 days'),(2,'5
days'),(3,'6 days');

openGauss=# SELECT a,LISTAGG(b) WITHIN GROUP(ORDER BY b) OVER(PARTITION BY a) FROM
listagg_t1;
 a |   listagg
---+---------------
 1 | 2 days3 days
 1 | 2 days3 days
 2 | 4 days5 days
 2 | 4 days5 days
 3 | 6 days
   | 1 day
(6 rows)

openGauss=# DROP TABLE listagg_t1;
```

- covar_pop(Y, X)

  描述：总体协方差。

  返回类型：double precision

  示例：

```
openGauss=# CREATE TABLE covar_pop_t1(a int, b int);

openGauss=# INSERT INTO covar_pop_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);

openGauss=# SELECT COVAR_POP(a,b) FROM covar_pop_t1;
 covar_pop
------------
        100
(1 row)
```



```
openGauss=# DROP TABLE covar_pop_t1;
```

- covar_samp(Y, X)

  描述：样本协方差。

  返回类型：double precision

  示例：

```
openGauss=# CREATE TABLE covar_samp_t1(a int, b int);

openGauss=# INSERT INTO covar_samp_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);

openGauss=# SELECT COVAR_SAMP(a,b) FROM covar_samp_t1;
 covar_samp
------------
         125
(1 row)

openGauss=# DROP TABLE covar_samp_t1;
```

- stddev_pop(expression)

  描述：总体标准差。

  返回类型：对于浮点类型的输入返回double precision，其他输入返回numeric。

  示例：

```
openGauss=# CREATE TABLE stddev_pop_t1(a int, b int);

openGauss=# INSERT INTO stddev_pop_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);

openGauss=# SELECT STDDEV_POP(a) FROM stddev_pop_t1;
     stddev_pop
--------------------
 7.4833147735478828
(1 row)

openGauss=# DROP TABLE stddev_pop_t1;
```

- stddev_samp(expression)

  描述：样本标准差。

  返回类型：对于浮点类型的输入返回double precision，其他输入返回numeric。

  示例：

```
openGauss=# CREATE TABLE stddev_samp_t1(a int, b int);

openGauss=# INSERT INTO stddev_samp_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);

openGauss=# SELECT STDDEV_SAMP(a) FROM stddev_samp_t1;
    stddev_samp
--------------------
 8.3666002653407555
(1 row)

openGauss=# DROP TABLE stddev_samp_t1;
```

- var_pop(expression)

  描述：总体方差（总体标准差的平方）

  返回类型：对于浮点类型的输入返回double precision类型，其他输入返回numeric类型。

  示例：

```
openGauss=# CREATE TABLE var_pop_t1(a int, b int);

openGauss=# INSERT INTO var_pop_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);
```



```
openGauss=# SELECT VAR_POP(a) FROM var_pop_t1;
       var_pop
---------------------
 56.0000000000000000
(1 row)

openGauss=# DROP TABLE var_pop_t1;
```

- var_samp(expression)

  描述：样本方差（样本标准差的平方）

  返回类型：对于浮点类型的输入返回double precision类型，其他输入返回numeric类型。

  示例：

```
openGauss=# CREATE TABLE var_samp_t1(a int, b int);

openGauss=# INSERT INTO var_samp_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);

openGauss=# SELECT VAR_SAMP(a) FROM var_samp_t1;
      var_samp
---------------------
 70.0000000000000000
(1 row)

openGauss=# DROP TABLE var_samp_t1;
```

- bit_and(expression)

  描述：所有非NULL输入值的按位与(AND)，如果全部输入值皆为NULL，那么结果也为NULL 。

  返回类型：和参数数据类型相同。

  示例：

```
openGauss=# CREATE TABLE bit_and_t1(a int, b int);

openGauss=# INSERT INTO bit_and_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT BIT_AND(a) FROM bit_and_t1;
 bit_and
---------
       0
(1 row)

openGauss=# DROP TABLE bit_and_t1;
```

- bit_or(expression)

  描述：所有非NULL输入值的按位或(OR)，如果全部输入值皆为NULL，那么结果也为NULL。

  返回类型：和参数数据类型相同

  示例：

```
openGauss=# CREATE TABLE bit_or_t1(a int, b int);

openGauss=# INSERT INTO bit_or_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT BIT_OR(a) FROM bit_or_t1;
 bit_or
--------
      3
(1 row)

openGauss=# DROP TABLE bit_or_t1;
```

- bool_and(expression)

  描述：如果所有输入值都是真，则为真，否则为假。

返回类型：bool

示例：

```
openGauss=# SELECT bool_and(100 <2500);
 bool_and
----------
 t
(1 row)
```

- bool_or(expression)

  描述：如果所有输入值只要有一个为真，则为真，否则为假。

  返回类型：bool

  示例：

```
openGauss=# SELECT bool_or(100 <2500);
 bool_or
----------
 t
(1 row)
```

- corr(Y, X)

  描述：相关系数

  返回类型：double precision

  示例：

```
openGauss=# CREATE TABLE corr_t1(a int, b int);

openGauss=# INSERT INTO corr_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT CORR(a,b) FROM corr_t1;
       corr
------------------
 .944911182523068
(1 row)

openGauss=# DROP TABLE corr_t1;
```

- every(expression)

  描述：等效于bool_and。

  返回类型：bool

  示例：

```
openGauss=# SELECT every(100 <2500);
 every
-------
 t
(1 row)
```

- rank(expression)

  描述：根据expression对不同组内的元组进行跳跃排序。

  返回类型：BIGINT

  示例：

```
openGauss=# CREATE TABLE rank_t1(a int, b int);

openGauss=# INSERT INTO rank_t1 VALUES(NULL,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

openGauss=# SELECT a,b,RANK() OVER(PARTITION BY a ORDER BY b) FROM rank_t1;
 a | b | rank
---+---+------
 1 | 2 |    1
 1 | 3 |    2
 2 | 4 |    1
 2 | 5 |    2
```



```
 3 | 6 |    1
   | 1 |    1
(6 rows)

openGauss=# DROP TABLE rank_t1;
```

- regr_avgx(Y, X)

  描述：自变量的平均值 (sum(X)/Y)

  返回类型：double precision

  示例：

```
openGauss=# CREATE TABLE regr_t1(a int, b int);

openGauss=# INSERT INTO regr_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT REGR_AVGX(a,b) FROM regr_t1;
 regr_avgx
-----------
         4
(1 row)

openGauss=# DROP TABLE regr_t1;
```

- regr_avgy(Y, X)

  描述：因变量的平均值 (sum(Y)/X)

  返回类型：double precision

  示例：

```
openGauss=# CREATE TABLE regr_avgy_t1(a int, b int);

openGauss=# INSERT INTO regr_avgy_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT REGR_AVGY(a,b) FROM regr_avgy_t1;
 regr_avgy
-----------
       1.8
(1 row)

openGauss=# DROP TABLE regr_avgy_t1;
```

- regr_count(Y, X)

  描述：两个表达式都不为NULL的输入行数。

  返回类型：bigint

  示例：

```
openGauss=# CREATE TABLE regr_count_t1(a int, b int);

openGauss=# INSERT INTO regr_count_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT REGR_COUNT(a,b) FROM regr_count_t1;
 regr_count
------------
          5
(1 row)

openGauss=# DROP TABLE regr_count_t1;
```

- regr_intercept(Y, X)

  描述：根据所有输入的点(X, Y)按照最小二乘法拟合成一个线性方程，然后返回该直线的Y轴截距。

  返回类型：double precision

  示例：

```
openGauss=# CREATE TABLE regr_intercept_t1(a int, b int);
```



```
openGauss=# INSERT INTO regr_intercept_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT REGR_INTERCEPT(b,a) FROM regr_intercept_t1;
  regr_intercept
------------------
 .785714285714286
(1 row)

openGauss=# DROP TABLE regr_intercept_t1;
```

- regr_r2(Y, X)

  描述：相关系数的平方

  返回类型：double precision

  示例：

```
openGauss=# CREATE TABLE regr_r2_t1(a int, b int);

openGauss=# INSERT INTO regr_r2_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT REGR_R2(b,a) FROM regr_r2_t1;
     regr_r2
------------------
 .892857142857143
(1 row)

openGauss=# DROP TABLE regr_r2_t1;
```

- regr_slope(Y, X)

  描述：根据所有输入的点(X, Y)按照最小二乘法拟合成一个线性方程， 然后返回该直线的斜率。

  返回类型：double precision

  示例：

```
openGauss=# CREATE TABLE regr_slope_t1(a int, b int);

openGauss=# INSERT INTO regr_slope_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT REGR_SLOPE(b,a) FROM regr_slope_t1;
    regr_slope
------------------
 1.78571428571429
(1 row)

openGauss=# DROP TABLE regr_slope_t1;
```

- regr_sxx(Y, X)

  描述：sum(Y^2) - sum(X)^2/N （自变量的“平方和”）

  返回类型：double precision

  示例：

```
openGauss=# CREATE TABLE regr_sxx_t1(a int, b int);

openGauss=# INSERT INTO regr_sxx_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT REGR_SXX(b,a) FROM regr_sxx_t1;
 regr_sxx
----------
      2.8
(1 row)

openGauss=# DROP TABLE regr_sxx_t1;
```

- regr_sxy(Y, X)

  描述：sum(X*Y) - sum(X) * sum(Y)/N （自变量和因变量的“乘方积”）

  返回类型：double precision

示例：

```
openGauss=# CREATE TABLE regr_sxy_t1(a int, b int);

openGauss=# INSERT INTO regr_sxy_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT REGR_SXY(b,a) FROM regr_sxy_t1;
 regr_sxy
----------
        5
(1 row)

openGauss=# DROP TABLE regr_sxy_t1;
```

- regr_syy(Y, X)

  描述：sum(Y^2) - sum(X)^2/N（因变量的"平方和"）

  返回类型：double precision

  示例：

```
openGauss=# CREATE TABLE regr_syy_t1(a int, b int);

openGauss=# INSERT INTO regr_syy_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT REGR_SYY(b,a) FROM regr_syy_t1;
 regr_syy
----------
       10
(1 row)

openGauss=# DROP TABLE regr_syy_t1;
```

- stddev(expression)

  描述：stddev_samp的别名。

  返回类型：对于浮点类型的输入返回double precision，其他输入返回numeric。

  示例：

```
openGauss=# CREATE TABLE stddev_t1(a int, b int);

openGauss=# INSERT INTO stddev_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT STDDEV(a) FROM stddev_t1;
        stddev
-----------------------
 .83666002653407554798
(1 row)

openGauss=# DROP TABLE stddev_t1;
```

- variance(expexpression,ression)

  描述：var_samp的别名。

  返回类型：对于浮点类型的输入返回double precision类型，其他输入返回numeric类型。

  示例：

```
openGauss=# CREATE TABLE variance_t1(a int, b int);

openGauss=# INSERT INTO variance_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT VARIANCE(a) FROM variance_t1;
      variance
-----------------------
 .70000000000000000000
(1 row)

openGauss=# DROP TABLE variance_t1;
```


- spread

  描述：该函数用于计算某段时间内最大和最小值差值。

  参数：real

  返回值类型：real

- checksum(expression)

  描述：返回所有输入值的CHECKSUM值。使用该函数可以用来验证GaussDB数据库（不支持GaussDB之外的其他数据库）的备份恢复或者数据迁移操作前后表中的数据是否相同。在备份恢复或者数据迁移操作前后都需要用户通过手工执行SQL命令的方式获取执行结果，通过对比获取的执行结果判断操作前后表中的数据是否相同。

  说明

  - 对于大表，CHECKSUM函数可能会需要很长时间。
  - 如果某两表的CHECKSUM值不同，则表明两表的内容是不同的。由于CHECKSUM函数中使用散列函数不能保证无冲突，因此两个不同内容的表可能会得到相同的CHECKSUM值，存在这种情况的可能性较小。对于列进行的CHECKSUM也存在相同的情况。
  - 对于时间类型timestamp, timestamptz和smalldatetime，计算CHECKSUM值时请确保时区设置一致。

  – 若计算某列的CHECKSUM值，且该列类型可以默认转为TEXT类型，则expression为列名。
  – 若计算某列的CHECKSUM值，且该列类型不能默认转为TEXT类型，则expression为列名::TEXT。
  – 若计算所有列的CHECKSUM值，则expression为表名::TEXT。

  可以默认转换为TEXT类型的类型包括：char, name, int8, int2, int1, int4, raw, pg_node_tree, float4, float8, bpchar, varchar, nvarchar2, date, timestamp, timestamptz, numeric, smalldatetime，其他类型需要强制转换为TEXT。

  返回类型：numeric。

  示例：

  表中可以默认转为TEXT类型的某列的CHECKSUM值。

```
openGauss=# CREATE TABLE checksum_t1(a int, b int);

openGauss=# INSERT INTO checksum_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT CHECKSUM(a) FROM checksum_t1;
  checksum
-------------
 18126842830
(1 row)

openGauss=# DROP TABLE checksum_t1;
```

  表中不能默认转为TEXT类型的某列的CHECKSUM值。注意此时CHECKSUM参数是列名::TEXT。

```
openGauss=# CREATE TABLE checksum_t1(a int, b int);

openGauss=# INSERT INTO checksum_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT CHECKSUM(a::TEXT) FROM checksum_t1;
  checksum
-------------
 18126842830
(1 row)

openGauss=# DROP TABLE checksum_t1;
```


表中所有列的CHECKSUM值。注意此时CHECKSUM参数是表名::TEXT，且表名前不加Schema。

```
openGauss=# CREATE TABLE checksum_t1(a int, b int);

openGauss=# INSERT INTO checksum_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

openGauss=# SELECT CHECKSUM(checksum_t1::TEXT) FROM checksum_t1;
  checksum
-------------
 11160522226
(1 row)

openGauss=# DROP TABLE checksum_t1;
```

7.5.18 窗口函数

窗口函数

窗口函数与OVER语句一起使用。OVER语句用于对数据进行分组，并对组内元素进行排序。窗口函数用于给组内的值生成序号。

说明

窗口函数中的order by后面必须跟字段名，若order by后面跟数字，该数字会被按照常量处理，因此对目标列没有起到排序的作用。

- RANK()

  描述：RANK函数为各组内值生成跳跃排序序号，其中，相同的值具有相同序号。

  返回值类型：BIGINT

  示例：

```
openGauss=# CREATE TABLE rank_t1(a int, b int);

openGauss=# INSERT INTO rank_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

openGauss=# SELECT a,b,RANK() OVER(PARTITION BY a ORDER BY b) FROM rank_t1;
 a | b | rank
---+---+------
 1 | 1 |    1
 1 | 1 |    1
 1 | 2 |    3
 1 | 3 |    4
 2 | 4 |    1
 2 | 5 |    2
 3 | 6 |    1
(7 rows)

openGauss=# DROP TABLE rank_t1;
```

- ROW_NUMBER()

  描述：ROW_NUMBER函数为各组内值生成连续排序序号，其中，相同的值其序号也不相同。

  返回值类型：BIGINT

  示例：

```
openGauss=# CREATE TABLE row_number_t1(a int, b int);

openGauss=# INSERT INTO row_number_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

openGauss=# SELECT a,b,ROW_NUMBER() OVER(PARTITION BY a ORDER BY b) FROM
row_number_t1;
 a | b | row_number
```



```
---+---+------------
 1 | 1 |          1
 1 | 1 |          2
 1 | 2 |          3
 1 | 3 |          4
 2 | 4 |          1
 2 | 5 |          2
 3 | 6 |          1
(7 rows)

openGauss=# DROP TABLE row_number_t1;
```

- DENSE_RANK()

  描述：DENSE_RANK函数为各组内值生成连续排序序号，其中，相同的值具有相同序号。

  返回值类型：BIGINT

  示例：

```
openGauss=# CREATE TABLE dense_rank_t1(a int, b int);

openGauss=# INSERT INTO dense_rank_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

openGauss=# SELECT a,b,DENSE_RANK() OVER(PARTITION BY a ORDER BY b) FROM dense_rank_t1;
 a | b | dense_rank
---+---+------------
 1 | 1 |          1
 1 | 1 |          1
 1 | 2 |          2
 1 | 3 |          3
 2 | 4 |          1
 2 | 5 |          2
 3 | 6 |          1
(7 rows)

openGauss=# DROP TABLE dense_rank_t1;
```

- PERCENT_RANK()

  描述：PERCENT_RANK函数为各组内对应值生成相对序号，即根据公式 (rank - 1) / (totalrows - 1)计算所得的值。其中rank为该值依据RANK函数所生成的对应序号，totalrows为该分组内的总元素个数。

  返回值类型：DOUBLE PRECISION

  示例：

  示例：

```
openGauss=# CREATE TABLE percent_rank_t1(a int, b int);

openGauss=# INSERT INTO percent_rank_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

openGauss=# SELECT a,b,PERCENT_RANK() OVER(PARTITION BY a ORDER BY b) FROM
percent_rank_t1;
 a | b |    percent_rank
---+---+-----------------
 1 | 1 |               0
 1 | 1 |               0
 1 | 2 | .666666666666667
 1 | 3 |               1
 2 | 4 |               0
 2 | 5 |               1
 3 | 6 |               0
(7 rows)

openGauss=# DROP TABLE percent_rank_t1;
```

- CUME_DIST()

  描述：CUME_DIST函数为各组内对应值生成累积分布序号。即根据公式(小于等于当前值的数据行数)/(该分组总行数totalrows)计算所得的相对序号。

返回值类型：DOUBLE PRECISION

示例：

```
openGauss=# CREATE TABLE cume_dist_t1(a int, b int);

openGauss=# INSERT INTO cume_dist_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

openGauss=# SELECT a,b,CUME_DIST() OVER(PARTITION BY a ORDER BY b) FROM cume_dist_t1;
 a | b | cume_dist
---+---+-----------
 1 | 1 |        .5
 1 | 1 |        .5
 1 | 2 |       .75
 1 | 3 |         1
 2 | 4 |        .5
 2 | 5 |         1
 3 | 6 |         1
(7 rows)

openGauss=# DROP TABLE cume_dist_t1;
```

- NTILE(num_buckets integer)

  描述：NTILE函数根据num_buckets integer将有序的数据集合平均分配到num_buckets所指定数量的桶中，并将桶号分配给每一行。分配时应尽量做到平均分配。

  返回值类型：INTEGER

  示例：

```
openGauss=# CREATE TABLE ntile_t1(a int, b int);

openGauss=# INSERT INTO ntile_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

openGauss=# SELECT a,b,NTILE(2) OVER(PARTITION BY a ORDER BY b) FROM ntile_t1;
 a | b | ntile
---+---+-------
 1 | 1 |     1
 1 | 1 |     1
 1 | 2 |     2
 1 | 3 |     2
 2 | 4 |     1
 2 | 5 |     2
 3 | 6 |     1
(7 rows)

openGauss=# DROP TABLE ntile_t1;
```

- LAG(value any [, offset integer [, default any]])

  描述：LAG函数为各组内对应值生成滞后值。即当前值对应的行数往前偏移offset位后所得行的value值作为序号。若经过偏移后行数不存在，则对应结果取为default值。若无指定，在默认情况下，offset取为1，default值取为NULL。default值的类型需要与value值的类型保持一致。

  返回值类型：与参数数据类型相同

  示例：

```
-- 建表并插入数据
openGauss=# CREATE TABLE ta1 (hire_date date, last_name varchar(20), department_id int);
CREATE TABLE
openGauss=# INSERT INTO ta1 values('07-DEC-02', 'Raphaely', 30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('24-JUL-05', 'Tobias',  30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('24-DEC-05', 'Baida',  30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('18-MAY-03', 'Khoo', 30);
INSERT 0 1
```



```
openGauss=# INSERT INTO ta1 values('15-NOV-06', 'Himuro',  30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('10-AUG-07', 'Colmenares',  30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('10-MAY-07', 'yq',  11);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('10-MAY-08', 'zi',  11);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('', 'yq1',  30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values(null, 'yq2',  30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('10-DEC-07', 'yq3',  30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values(null, null,  11);
INSERT 0 1
openGauss=# INSERT INTO ta1 values(null, null,  11);
INSERT 0 1

-- 调用LAG,指定offset=3, default=null
openGauss=# SELECT hire_date, last_name, department_id, lag(hire_date, 3, null)  OVER (PARTITION
BY department_id ORDER BY last_name) AS "NextHired" FROM ta1 ORDER BY department_id;
     hire_date      | last_name  | department_id |      NextHired
--------------------+------------+---------------+---------------------
 2007-05-10 00:00:00 | yq         |            11 |
 2008-05-10 00:00:00 | zi         |            11 |
                     |            |            11 |
                     |            |            11 | 2007-05-10 00:00:00
 2005-12-24 00:00:00 | Baida      |            30 |
 2007-08-10 00:00:00 | Colmenares |            30 |
 2006-11-15 00:00:00 | Himuro     |            30 |
 2003-05-18 00:00:00 | Khoo       |            30 | 2005-12-24 00:00:00
 2002-12-07 00:00:00 | Raphaely   |            30 | 2007-08-10 00:00:00
 2005-07-24 00:00:00 | Tobias     |            30 | 2006-11-15 00:00:00
                     | yq1        |            30 | 2003-05-18 00:00:00
                     | yq2        |            30 | 2002-12-07 00:00:00
 2007-12-10 00:00:00 | yq3        |            30 | 2005-07-24 00:00:00
(13 rows)

openGauss=# DROP TABLE ta1;
```

- LEAD(value any [, offset integer [, default any]])

  描述：LEAD函数为各组内对应值生成提前值。即当前值对应的行数向后偏移offset位后所得行的value值作为序号。若经过向后偏移后行数超过当前组内的总行数，则对应结果取为default值。若无指定，在默认情况下，offset取为1，default值取为NULL。default值的类型需要与value值的类型保持一致。

  返回值类型：与参数数据类型相同。

  示例：

```
openGauss=# CREATE TABLE ta1 (hire_date date, last_name varchar(20), department_id int);
CREATE TABLE
openGauss=# INSERT INTO ta1 values('07-DEC-02', 'Raphaely', 30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('24-JUL-05', 'Tobias',  30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('24-DEC-05', 'Baida',  30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('18-MAY-03', 'Khoo', 30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('15-NOV-06', 'Himuro',  30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('10-AUG-07', 'Colmenares',  30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('10-MAY-07', 'yq',  11);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('10-MAY-08', 'zi',  11);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('', 'yq1',  30);
```



```
INSERT 0 1
openGauss=# INSERT INTO ta1 values(null, 'yq2',  30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values('10-DEC-07', 'yq3',  30);
INSERT 0 1
openGauss=# INSERT INTO ta1 values(null, null,  11);
INSERT 0 1
openGauss=# INSERT INTO ta1 values(null, null,  11);
INSERT 0 1

-- 调用LEAD,指定offset=2
openGauss=# SELECT hire_date, last_name, department_id, lead(hire_date, 2) OVER (PARTITION BY
department_id ORDER BY last_name) AS "NextHired" FROM ta1 ORDER BY department_id;
      hire_date      | last_name  | department_id |      NextHired
---------------------+------------+---------------+---------------------
 2007-05-10 00:00:00 | yq         |            11 |
 2008-05-10 00:00:00 | zi         |            11 |
                     |            |            11 |
                     |            |            11 |
 2005-12-24 00:00:00 | Baida      |            30 | 2006-11-15 00:00:00
 2007-08-10 00:00:00 | Colmenares |            30 | 2003-05-18 00:00:00
 2006-11-15 00:00:00 | Himuro     |            30 | 2002-12-07 00:00:00
 2003-05-18 00:00:00 | Khoo       |            30 | 2005-07-24 00:00:00
 2002-12-07 00:00:00 | Raphaely   |            30 |
 2005-07-24 00:00:00 | Tobias     |            30 |
                     | yq1        |            30 | 2007-12-10 00:00:00
                     | yq2        |            30 |
 2007-12-10 00:00:00 | yq3        |            30 |
(13 rows)

openGauss=# DROP TABLE ta1;
```

- FIRST_VALUE(value any)

  描述：FIRST_VALUE函数取各组内的第一个值作为返回结果。

  返回值类型：与参数数据类型相同。

  示例：

```
openGauss=# CREATE TABLE first_value_t1(a int, b int);

openGauss=# INSERT INTO first_value_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

openGauss=# SELECT a,b,FIRST_VALUE(b) OVER(PARTITION BY a ORDER BY b) FROM first_value_t1;
 a | b | first_value
---+---+-------------
 1 | 1 |           1
 1 | 1 |           1
 1 | 2 |           1
 1 | 3 |           1
 2 | 4 |           4
 2 | 5 |           4
 3 | 6 |           6
(7 rows)

openGauss=# DROP TABLE first_value_t1;
```

- LAST_VALUE(value any)

  描述：LAST_VALUE函数取各组内的最后一个值作为返回结果。

  返回值类型：与参数数据类型相同。

  示例：

```
openGauss=# CREATE TABLE last_value_t1(a int, b int);

openGauss=# INSERT INTO last_value_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

openGauss=# SELECT a,b,LAST_VALUE(b) OVER(PARTITION BY a ORDER BY b) FROM last_value_t1;
 a | b | last_value
---+---+------------
```



```
 1 | 1 |         1
 1 | 1 |         1
 1 | 2 |         2
 1 | 3 |         3
 2 | 4 |         4
 2 | 5 |         5
 3 | 6 |         6
(7 rows)

openGauss=# DROP TABLE last_value_t1;
```

- NTH_VALUE(value any, nth integer)

  描述：NTH_VALUE函数返回该组内的第nth行作为结果。若该行不存在，则默认返回NULL。

  返回值类型：与参数数据类型相同。

  示例：

```
openGauss=# CREATE TABLE nth_value_t1(a int, b int);

openGauss=# INSERT INTO nth_value_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

openGauss=# SELECT a,b,NTH_VALUE(b, 2) OVER(PARTITION BY a order by b) FROM nth_value_t1;
 a | b | nth_value
---+---+-----------
 1 | 1 |         1
 1 | 1 |         1
 1 | 2 |         1
 1 | 3 |         1
 2 | 4 |
 2 | 5 |         5
 3 | 6 |
(7 rows)

openGauss=# DROP TABLE nth_value_t1;
```

- delta

  描述：返回当前行和前一行的差值。

  参数：numeric

  返回值类型：numeric

- spread

  描述：该函数用于计算某段时间内最大和最小值得差值。

  参数：real

  返回值类型：real

7.5.19 安全函数

安全函数

- gs_encrypt_aes128(encryptstr,keystr)

  描述：使用基于keystr派生出的密钥对encryptstr字符串进行加密，返回加密后的字符串。keystr的长度范围为8~16字节，并且至少包含大写字母、小写字母、数字和特殊字符中的三种字符。支持的加密数据类型：目前数据库支持的数值类型，字符类型，二进制类型中的RAW，日期/时间类型中的DATE、TIMESTAMP、SMALLDATETIME。

  返回值类型：text

  返回值长度：至少为92字节，不超过4*[(Len+68)/3]字节，其中Len为加密前数据长度（单位为字节）。

示例：

```
openGauss=# SELECT gs_encrypt_aes128('MPPDB','1234@abc');
                                gs_encrypt_aes128
-----------------------------------------------------------------------------------------
 OF1g3+70oeqFfyKiWlpxfYxPnpeitNc6+7nAe02Ttt37fZF8Q+bbEYhdw/YG+0c9tHKRWM6OcTzlB3HnqvX
+1d8Bflo=
(1 row)
```

说明

由于该函数的执行过程需要传入口令，为了安全起见，gsql工具不会将包含该函数名字样的SQL记录入执行历史。即无法在gsql里通过上下翻页功能找到该函数的执行历史。

- gs_decrypt_aes128(decryptstr,keystr)

  描述：使用基于keystr派生出的密钥对decrypt字符串进行解密，返回解密后的字符串。解密使用的keystr必须保证与加密时使用的keystr一致才能正确解密。keystr不得为空。

  说明

  此函数需要结合gs_encrypt_aes128加密函数共同使用。

  返回值类型：text

  示例：

  ```
  openGauss=# SELECT gs_decrypt_aes128('OF1g3+70oeqFfyKiWlpxfYxPnpeitNc6+7nAe02Ttt37fZF8Q
  +bbEYhdw/YG+0c9tHKRWM6OcTzlB3HnqvX+1d8Bflo=','1234@abc');
   gs_decrypt_aes128
  -------------------
   MPPDB
  (1 row)
  ```

  说明

  由于该函数的执行过程需要传入口令，为了安全起见，gsql工具不会将包含该函数名字样的SQL记录入执行历史；即无法在gsql里通过上下翻页功能找到该函数的执行历史。

- gs_password_deadline

  描述：显示当前账户密码离过期还距离多少天。

  返回值类型：interval

  示例：

  ```
  openGauss=# SELECT gs_password_deadline();
    gs_password_deadline
  -------------------------
   83 days 17:44:32.196094
  (1 row)
  ```

- gs_password_notifytime()

  描述：显示账户密码到期前提醒的天数。

  返回值类型：int32

- login_audit_messages(BOOLEAN)

  描述：查看登录用户的登录信息。

  返回值类型：元组

  示例：

  - 查看上一次登录成功的日期、时间和IP等信息。

    ```
    openGauss=> SELECT * FROM login_audit_messages(true);
     username | database |       logintime        |    mytype     | result | client_conninfo
    ----------+----------+------------------------+---------------+--------+-----------------
     omm      | postgres | 2020-06-29 21:56:40+08 | login_success | ok     | gsql@[local]
    (1 row)
    ```


- 查看自从上一次登录成功以来登录失败的尝试次数、日期和时间。

```
openGauss=>  SELECT * FROM login_audit_messages(false);
 username | database |       logintime        |    mytype    | result |  client_conninfo
----------+----------+------------------------+--------------+--------+-------------------
 omm      | postgres | 2020-06-29 21:57:55+08 | login_failed | failed | [unknown]@[local]
 omm      | postgres | 2020-06-29 21:57:53+08 | login_failed | failed | [unknown]@[local]
(2 rows)
```

- login_audit_messages_pid

  描述：查看登录用户的登录信息。与login_audit_messages的区别在于结果基于当前backendid向前查找。所以不会因为同一用户的后续登录，而影响本次登录的查询结果。也就是查询不到该用户后续登录的信息。

  返回值类型：元组

  说明

  在开启线程池的情况下，由于线程切换，同一session中获取到的backendid可能会发生变化，因此会造成多次调用该函数返回值不一致的情况。不建议用户在开启线程池的情况下调用此函数。

  示例：

  - 查看上一次登录成功的日期、时间和IP等信息。

```
openGauss=> SELECT * FROM login_audit_messages_pid(true);
 username | database |       logintime        |    mytype    | result | client_conninfo |    backendid
----------+----------+------------------------+--------------+--------+-----------------+-----------------
 omm      | postgres | 2020-06-29 21:56:40+08 | login_success | ok     | gsql@[local]    |
139823109633792
(1 row)
```

  - 查看自从上一次登录成功以来登录失败的尝试次数、日期和时间。

```
openGauss=> SELECT * FROM login_audit_messages_pid(false);
 username | database |       logintime        |    mytype    | result |  client_conninfo  |    backendid
----------+----------+------------------------+--------------+--------+-------------------+-----------------
 omm      | postgres | 2020-06-29 21:57:55+08 | login_failed | failed | [unknown]@[local] |
139823109633792
 omm      | postgres | 2020-06-29 21:57:53+08 | login_failed | failed | [unknown]@[local] |
139823109633792
(2 rows)
```

- inet_server_addr

  描述：显示服务器IP信息。

  返回值类型：inet

  示例：

```
openGauss=# SELECT inet_server_addr();
 inet_server_addr
------------------
 10.10.0.13
(1 row)
```

  说明

  - 上面是以客户端在10.10.0.50上，服务器端在10.10.0.13上为例。
  - 如果是通过本地连接，使用此接口显示为空。

- inet_client_addr

  描述：显示客户端IP信息。

  返回值类型：inet

  示例：

```
openGauss=# SELECT inet_client_addr();
 inet_client_addr
------------------
```



```
 10.10.0.50
(1 row)
```

说明

- 上面是以客户端在10.10.0.50上，服务器端在10.10.0.13上为例。
- 如果是通过本地连接，使用此接口显示为空。

- pg_query_audit
  描述：查看当前CN节点审计日志。
  返回值类型：record
  函数返回字段如下：

名称	类型	描述
time	timestamp with time zone	操作时间
type	text	操作类型
result	text	操作结果
userid	oid	用户id
username	text	执行操作的用户名
database	text	数据库名称
client_conninfo	text	客户端连接信息
object_name	text	操作对象名称
detail_info	text	执行操作详细信息
node_name	text	节点名称
thread_id	text	线程id
local_port	text	本地端口
remote_port	text	远端端口

- pgxc_query_audit
  描述：查看所有CN节点审计日志。
  返回值类型：record
  函数返回字段同pg_query_audit函数。
- pg_delete_audit
  描述：删除指定时间段的审计日志。
  返回值类型：void
- alldigitsmasking
  描述：脱敏策略的内部函数，对所有字符进行脱敏。
  参数：col text, letter character default '0'
  返回值类型：text

- creditcardmasking

  描述：脱敏策略的内部函数，对所有信用卡信息进行脱敏。

  参数：col text, letter character default 'x'

  返回值类型：text

- randommasking

  描述：脱敏策略的内部函数，使用随机策略。

  参数：col text,

  返回值类型：text

- fullemailmasking

  描述：脱敏策略的内部函数，对出现最后一个'.'之前的文本（除'@'之外）进行脱敏。

  参数：col text, letter character default 'x'

  返回值类型：text

- basicemailmasking

  描述：脱敏策略的内部函数，对出现第一个'@'之前的文本进行脱敏。

  参数：col text, letter character default 'x'

  返回值类型：text

- shufflemasking

  描述：脱敏策略的内部函数，对字符进行乱序排列。

  参数：col text

  返回值类型：text

- regexpmasking

  描述：脱敏策略的内部函数，对字符进行正则表达式替换。

  参数：col text, reg text, replace_text text, pos INTEGER default 0, reg_len INTEGER default -1

  返回值类型：text

- gs_encrypt(encryptstr,keystr, encrypttype)

  描述：根据encrypttype，以keystr为密钥对encryptstr字符串进行加密，返回加密后的字符串。keystr的长度范围为8~16字节，至少包含3种字符（大写字母、小写字母、数字、特殊字符），encrypttype可以是aes128或sm4。

  返回值类型：text

  示例：

  ```
  openGauss=#  SELECT gs_encrypt('MPPDB','Asdf1234','sm4');
        gs_encrypt
  ------------------------------
  ZBzOmaGA4Bb+coyucJ0B8AkIShqc
  (1 row)
  ```

  说明

  由于该函数的执行过程需要传入解密口令，为了安全起见，gsql工具不会将包含该函数名字样的SQL记录入执行历史。即无法在gsql里通过上下翻页功能找到该函数的执行历史。

- gs_decrypt(decryptstr,keystr,decrypttype)

  描述：根据decrypttype，以keystr为密钥对decrypt字符串进行解密，返回解密后的字符串。解密使用的decrypttype 及keystr必须保证与加密时使用的encrypttype

及keystr一致才能正常解密。keystr不得为空。decrypttype可以是aes128或sm4。

此函数需要结合gs_encrypt加密函数共同使用。

返回值类型：text

示例：

```
openGauss=# SELECT gs_decrypt('ZBzOmaGA4Bb+coyucJ0B8AkIShqc','Asdf1234','sm4');
 gs_decrypt
------------
 MPPDB
(1 row)
```

说明

由于该函数的执行过程需要传入解密口令，为了安全起见，gsql工具不会将包含该函数名字样的SQL记录入执行历史；即无法在gsql里通过上下翻页功能找到该函数的执行历史。

7.5.20 返回集合的函数

序列号生成函数

- generate_series(start, stop)

  描述：生成一个数值序列，从start到stop，步长为1。

  参数类型：int、bigint、numeric

  返回值类型：setof int、setof bigint、setof numeric（与参数类型相同）

- generate_series(start, stop, step)

  描述：生成一个数值序列，从start到stop，步长为step。

  参数类型：int、bigint、numeric

  返回值类型：setof int、setof bigint、setof numeric（与参数类型相同）

- generate_series(start, stop, step interval)

  描述：生成一个数值序列，从start到stop，步长为step。

  参数类型：timestamp或timestamp with time zone

  返回值类型：setof timestamp或setof timestamp with time zone（与参数类型相同）

如果step是正数且start大于stop，则返回零行。相反，如果step是负数且start小于stop，则也返回零行。如果输入是NULL，同样产生零行。如果step为零则是一个错误。

示例：

```
openGauss=# SELECT * FROM generate_series(2,4);
 generate_series
-----------------
               2
               3
               4
(3 rows)

openGauss=# SELECT * FROM generate_series(5,1,-2);
 generate_series
-----------------
               5
               3
               1
(3 rows)
```



```
openGauss=# SELECT * FROM generate_series(4,3);
 generate_series
-----------------
(0 rows)

--这个示例应用于date-plus-integer操作符。
openGauss=# SELECT current_date + s.a AS dates FROM generate_series(0,14,7) AS s(a);
   dates
------------
 2017-06-02
 2017-06-09
 2017-06-16
(3 rows)

openGauss=# SELECT * FROM generate_series('2008-03-01 00:00'::timestamp, '2008-03-04 12:00', '10
hours');
   generate_series
---------------------
 2008-03-01 00:00:00
 2008-03-01 10:00:00
 2008-03-01 20:00:00
 2008-03-02 06:00:00
 2008-03-02 16:00:00
 2008-03-03 02:00:00
 2008-03-03 12:00:00
 2008-03-03 22:00:00
 2008-03-04 08:00:00
(9 rows)
```

下标生成函数

- generate_subscripts(array anyarray, dim int)

  描述：生成一系列包括给定数组的下标。

  返回值类型：setof int

- generate_subscripts(array anyarray, dim int, reverse boolean)

  描述：生成一系列包括给定数组的下标。当reverse为真时，该系列则以相反的顺序返回。

  返回值类型：setof int

generate_subscripts是一个为给定数组中的指定维度生成有效下标集的函数。如果数组中没有所请求的维度或者NULL数组，返回零行（但是会给数组元素为空的返回有效下标）。示例：

```
--基本用法。
openGauss=# SELECT generate_subscripts('{NULL,1,NULL,2}'::int[], 1) AS s;
 s
---
 1
 2
 3
 4
(4 rows)
--unnest一个2D数组。
openGauss=# CREATE OR REPLACE FUNCTION unnest2(anyarray)
RETURNS SETOF anyelement AS $$
SELECT $1[i][j]
   FROM generate_subscripts($1,1) g1(i),
        generate_subscripts($1,2) g2(j);
$$ LANGUAGE sql IMMUTABLE;

openGauss=# SELECT * FROM unnest2(ARRAY[[1,2],[3,4]]);
 unnest2
---------
```



```
     1
     2
     3
     4
(4 rows)

--删除函数。
openGauss=# DROP FUNCTION unnest2;
```

7.5.21 条件表达式函数

条件表达式函数

- coalesce(expr1, expr2, ..., exprn)

  描述：

  返回参数列表中第一个非NULL的参数值。

  COALESCE(expr1, expr2) 等价于CASE WHEN expr1 IS NOT NULL THEN expr1 ELSE expr2 END。

  示例：

  ```
  openGauss=# SELECT coalesce(NULL,'hello');
   coalesce
  ----------
   hello
  (1 row)
  ```

  备注：

  - 如果表达式列表中的所有表达式都等于NULL，则本函数返回NULL。
  - 它常用于在显示数据时用缺省值替换NULL。
  - 和CASE表达式一样，COALESCE不会计算不需要用来判断结果的参数；即在第一个非空参数右边的参数不会被计算。

- decode(base_expr, compare1, value1, Compare2,value2, … default)

  描述：把base_expr与后面的每个compare(n) 进行比较，如果匹配返回相应的value(n)。如果没有发生匹配，则返回default。

  示例：

  ```
  openGauss=# SELECT decode('A','A',1,'B',2,0);
   case
  ------
   1
  (1 row)
  ```

- nullif(expr1, expr2)

  描述：当且仅当expr1和expr2相等时，NULLIF才返回NULL，否则它返回expr1。

  nullif(expr1, expr2) 逻辑上等价于CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END。

  示例：

  ```
  openGauss=# SELECT nullif('hello','world');
   nullif
  --------
   hello
  (1 row)
  ```

  备注：

  如果两个参数的数据类型不同，则：

  - 若两种数据类型之间存在隐式转换，则以其中优先级较高的数据类型为基准将另一个参数隐式转换成该类型，转换成功则进行计算，转换失败则返回错误。如：

```
openGauss=# SELECT nullif('1234'::VARCHAR,123::INT4);
 nullif
--------
   1234
(1 row)
openGauss=# SELECT nullif('1234'::VARCHAR,'2012-12-24'::DATE);
ERROR:  invalid input syntax for type timestamp: "1234"
```

  - 若两种数据类型之间不存在隐式转换，则返回错误 。如：

```
openGauss=# SELECT nullif(TRUE::BOOLEAN,'2012-12-24'::DATE);
ERROR:  operator does not exist: boolean = timestamp without time zone
LINE 1: SELECT nullif(TRUE::BOOLEAN,'2012-12-24'::DATE) FROM sys_dummy;
^
HINT:  No operator matches the given name and argument type(s). You might need to add explicit type casts.
```

- nvl(expr1 , expr2)

  描述：

  - 如果expr1为NULL则返回expr2。
  - 如果expr1非NULL，则返回expr1。

  示例：

```
openGauss=# SELECT nvl('hello','world');
  nvl
-------
 hello
(1 row)
```

  备注：参数expr1和expr2可以为任意类型，当NVL的两个参数不属于同类型时，看第二个参数是否可以向第一个参数进行隐式转换，如果可以则返回第一个参数类型。如果第二个参数不能向第一个参数进行隐式转换而第一个参数可以向第二个参数进行隐式转换，则返回第二个参数的类型。如果两个参数之间不存在隐式类型转换并且也不属于同一类型则报错。

- greatest(expr1 [, ...])

  描述：获取并返回参数列表中值最大的表达式的值。

  返回值类型：

  示例：

```
openGauss=# SELECT greatest(1*2,2-3,4-1);
 greatest
----------
        3
(1 row)
openGauss=# SELECT greatest('HARRY', 'HARRIOT', 'HAROLD');
 greatest
----------
 HARRY
(1 row)
```

- least(expr1 [, ...])

  描述：获取并返回参数列表中值最小的表达式的值。

  示例：

```
openGauss=# SELECT least(1*2,2-3,4-1);
 least
-------
    -1
(1 row)
openGauss=# SELECT least('HARRY','HARRIOT','HAROLD');
 least
```



```
--------
 HAROLD
(1 row)
```

- EMPTY_BLOB()

  描述：使用EMPTY_BLOB在INSERT或UPDATE语句中初始化一个BLOB变量，取值为NULL。

  返回值类型：BLOB

  示例：

  ```
  --新建表
  openGauss=# CREATE TABLE blob_tb(b blob,id int) DISTRIBUTE BY REPLICATION;
  --插入数据
  openGauss=# INSERT INTO blob_tb VALUES (empty_blob(),1);
  --删除表
  openGauss=# DROP TABLE blob_tb;
  ```

  备注：使用DBE_LOB.GET_LENGTH求得的长度为0。

7.5.22 系统信息函数

会话信息函数

- current_catalog

  描述：当前数据库的名称（在标准SQL中称"catalog"）。

  返回值类型：name

  示例：

  ```
  testdb=# SELECT current_catalog;
   current_database
  ------------------
   testdb
  (1 row)
  ```

- current_database()

  描述：当前数据库的名称。

  返回值类型：name

  示例：

  ```
  testdb=# SELECT current_database();
   current_database
  ------------------
   testdb
  (1 row)
  ```

- current_query()

  描述：由客户端提交的当前执行语句（可能包含多个声明）。

  返回值类型：text

  示例：

  ```
  openGauss=# SELECT current_query();
       current_query
  -------------------------
   SELECT current_query();
  (1 row)
  ```

- current_schema[()]

  描述：当前模式的名称。

  返回值类型：name

  示例：


```
openGauss=# SELECT current_schema();
 current_schema
----------------
 public
(1 row)
```

备注：current_schema返回在搜索路径中第一个顺位有效的模式名。（如果搜索路径为空则返回NULL，没有有效的模式名也返回NULL）。如果创建表或者其他命名对象时没有声明目标模式，则将使用这些对象的模式。

- current_schemas(Boolean)

  描述：搜索路径中的模式名称。

  返回值类型：name[]

  示例：

  ```
  openGauss=# SELECT current_schemas(true);
    current_schemas
  ---------------------
  {pg_catalog,public}
  (1 row)
  ```

  备注：

  current_schemas(Boolean)返回搜索路径中所有模式名称的数组。布尔选项决定像pg_catalog这样隐含包含的系统模式是否包含在返回的搜索路径中。

  说明

  搜索路径可以通过运行时设置更改。命令是：

  ```
  SET search_path TO schema [, schema, ...]
  ```

- current_user

  描述：当前执行环境下的用户名。

  返回值类型：name

  示例：

  ```
  openGauss=# SELECT current_user;
   current_user
  --------------
   omm
  (1 row)
  ```

  备注：current_user是用于权限检查的用户标识。通常，他表示会话用户，但是可以通过**SET ROLE**改变他。在函数执行的过程中随着属性SECURITY DEFINER的改变，其值也会改变。

- definer_current_user

  描述：当前执行环境下的用户名。

  返回值类型：name

  示例：

  ```
  openGauss=# SELECT definer_current_user();
   definer_current_user
  ----------------------
   omm
  (1 row)
  ```

  备注：大多数情况下definer_current_user和current_user结果相同，但在存储过程中执行该函数会返回定义当前存储过程的用户名。

- pg_current_sessionid()

  描述：当前执行环境下的会话ID。

  返回值类型：text

示例：

```
openGauss=# SELECT pg_current_sessionid();
    pg_current_sessionid
----------------------------
 1579228402.140190434944768
(1 row)
```

备注：pg_current_sessionid()是用于获取当前执行环境下的会话ID。其组成结构为：时间戳.会话ID，当线程池模式开启（enable_thread_pool=on）时，会话ID为SessionID；而线程池模式关闭时，会话ID实际为线程ID。

- pg_current_sessid

  描述：当前执行环境下的会话ID。

  返回值类型：text

  示例：

  ```
  openGauss=# select pg_current_sessid();
  pg_current_sessid
  -------------------
  140308875015936
  (1 row)
  ```

  备注：在线程池模式下获得当前会话的会话ID，非线程池模式下获得当前会话对应的后台线程ID。

- pg_current_userid

  描述：当前用户ID。

  返回值类型：text

  示例：

  ```
  openGauss=# SELECT pg_current_userid();
  pg_current_userid
  -------------------
  10
  (1 row)
  ```

- tablespace_oid_name(oid)

  描述：根据表空间oid，查找表空间名称。

  返回值类型：text

  示例：

  ```
  openGauss=# select tablespace_oid_name(1663);
   tablespace_oid_name
  ---------------------
   pg_default
  (1 row)
  ```

- inet_client_addr()

  描述：连接的远端地址。inet_client_addr返回当前客户端的IP地址。

  说明

  此函数只有在远程连接模式下有效。

  返回值类型：inet

  示例：

  ```
  openGauss=# SELECT inet_client_addr();
   inet_client_addr
  ------------------
   10.10.0.50
  (1 row)
  ```

- inet_client_port()

描述：连接的远端端口。inet_client_port返回当前客户端的端口号。

说明

此函数只有在远程连接模式下有效。

返回值类型：int

示例：

```
openGauss=# SELECT inet_client_port();
 inet_client_port
------------------
            33143
(1 row)
```

- inet_server_addr()

  描述：连接的本地地址。inet_server_addr返回服务器接收当前连接用的IP地址。

  说明

  此函数只有在远程连接模式下有效。

  返回值类型：inet

  示例：

  ```
  openGauss=# SELECT inet_server_addr();
   inet_server_addr
  ------------------
   10.10.0.13
  (1 row)
  ```

- inet_server_port()

  描述：连接的本地端口。inet_server_port返回接收当前连接的端口号。如果是通过UNIX-domain socket连接的，则所有这些函数都返回NULL。

  说明

  此函数只有在远程连接模式下有效。

  返回值类型：int

  示例：

  ```
  openGauss=# SELECT inet_server_port();
   inet_server_port
  ------------------
              8000
  (1 row)
  ```

- pg_backend_pid()

  描述：当前会话连接的服务进程的进程ID。

  返回值类型：int

  示例：

  ```
  openGauss=# SELECT pg_backend_pid();
   pg_backend_pid
  ----------------
   140229352617744
  (1 row)
  ```

- pg_conf_load_time()

  描述：配置加载时间。pg_conf_load_time返回最后加载服务器配置文件的时间戳。

  返回值类型：timestamp with time zone

  示例：


```
openGauss=# SELECT pg_conf_load_time();
     pg_conf_load_time
-------------------------------
 2017-09-01 16:05:23.89868+08
(1 row)
```

- pg_my_temp_schema()

  描述：会话的临时模式的OID，不存在则为0。

  返回值类型：oid

  示例：

  ```
  openGauss=# SELECT pg_my_temp_schema();
   pg_my_temp_schema
  -------------------
                   0
  (1 row)
  ```

  备注：pg_my_temp_schema返回当前会话中临时模式的OID，如果不存在（没有创建临时表）的话则返回0。如果给定的OID是其它会话中临时模式的OID，pg_is_other_temp_schema则返回true。

- pg_is_other_temp_schema(oid)

  描述：是否为另一个会话的临时模式。

  返回值类型：Boolean

  示例：

  ```
  openGauss=# SELECT pg_is_other_temp_schema(25356);
   pg_is_other_temp_schema
  -------------------------
   f
  (1 row)
  ```

- pg_listening_channels()

  描述：会话正在侦听的信道名称。

  返回值类型：setof text

  示例：

  ```
  openGauss=# SELECT pg_listening_channels();
   pg_listening_channels
  -----------------------
  (0 rows)
  ```

  备注：pg_listening_channels返回当前会话正在侦听的一组信道名称。

- pg_postmaster_start_time()

  描述：服务器启动时间。pg_postmaster_start_time返回服务器启动时的timestamp with time zone。

  返回值类型：timestamp with time zone

  示例：

  ```
  openGauss=# SELECT pg_postmaster_start_time();
    pg_postmaster_start_time
  -------------------------------
   2017-08-30 16:02:54.99854+08
  (1 row)
  ```

- sessionid2pid()

  描述：从sessionid中得到pid信息（如: pv_session_stat中sessid列）。

  返回值类型：int8

  示例：

  ```
  openGauss=# select sessionid2pid(sessid::cstring) from pv_session_stat limit 2;
   sessionid2pid
  ```



```
-----------------
 139973107902208
 139973107902208
(2 rows)
```

- session_context('namespace' , 'parameter')

  描述：获取并返回指定namespace下参数parameter的值。

  返回值类型：VARCHAR

  示例：

  ```
  openGauss=# SELECT session_context('USERENV', 'CURRENT_SCHEMA');
   session_context
  -------------
   public
  (1 row)
  ```

  根据当前所在的实际schema而变化。

  备注：目前仅支持SESSION_CONTEXT('USERENV', 'CURRENT_SCHEMA') 和SESSION_CONTEXT('USERENV', 'CURRENT_USER')两种格式。

- pg_trigger_depth()

  描述：触发器的嵌套层次。

  返回值类型：int

  示例：

  ```
  openGauss=# SELECT pg_trigger_depth();
   pg_trigger_depth
  ------------------
                 0
  (1 row)
  ```

- opengauss_version()

  描述：openGauss版本信息。

  返回值类型：text

  使用示例如下，查询结果中的*x.x.x*请以实际输出为准：

  ```
  openGauss=# SELECT opengauss_version();
   opengauss_version
  -------------------
   x.x.x
  (1 row)
  ```

- gs_deployment()

  描述：当前系统的部署形态信息，对于分布式系统来说返回的是“Distribute”。

  返回值类型：text

  示例：

  ```
  openGauss=# select gs_deployment();
   gs_deployment
  ---------------
   Distribute
  (1 row)
  ```

- session_user

  描述：会话用户名。

  返回值类型：name

  示例：

  ```
  openGauss=# SELECT session_user;
   session_user
  --------------
   omm
  (1 row)
  ```


备注：session_user通常是连接当前数据库的初始用户，不过系统管理员可以用**SET SESSION AUTHORIZATION**修改这个设置。

- user
  描述：等价于current_user。
  返回值类型：name
  示例：
```
openGauss=# SELECT user;
 current_user
--------------
 omm
(1 row)
```
- get_shard_oids_byname
  描述：输入node的名字返回node的oid。
  返回值类型：oid
  示例：
```
openGauss=# select get_shard_oids_byname('datanode1');
 get_shard_oids_byname
-----------------------
 {16385}
(1 row)
```
- getpgusername()
  描述：获取数据库用户名。
  返回值类型：name
  示例：
```
openGauss=# select getpgusername();
 getpgusername
---------------
 GaussDB_userna
(1 row)
```
- getdatabaseencoding()
  描述：获取数据库编码方式。
  返回值类型：name
  示例：
```
openGauss=# select getdatabaseencoding();
 getdatabaseencoding
---------------------
 SQL_ASCII
(1 row)
```
- version()
  描述：版本信息。version返回一个描述服务器版本信息的字符串。
  返回值类型：text
  示例：
```
openGauss=# SELECT version();
                                                       version
---------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------
 openGauss 2.0.0 (GaussDBl VxxxRxxxCxx build f521c606) compiled at 2021-09-16 14:55:22 commit
2935 last mr 6385 release
(1 row)
```
- working_version_num()
  描述：版本序号信息。返回一个系统兼容性有关的版本序号。

返回值类型：int

示例：

```
openGauss=# SELECT working_version_num();
 working_version_num
---------------------
               92231
(1 row)
```

- get_hostname()

  描述：返回当前节点的hostname。

  返回值类型：text

  示例：

  ```
  openGauss=# SELECT get_hostname();
   get_hostname
  --------------
   linux-user
  (1 row)
  ```

- get_nodename()

  描述：返回当前节点的名字。

  返回值类型：text

  示例：

  ```
  openGauss=# SELECT get_nodename();
   get_nodename
  --------------
   coordinator1
  (1 row)
  ```

- get_schema_oid(cstring)

  描述：返回查询schema的oid。

  返回值类型：oid

  示例：

  ```
  openGauss=# SELECT get_schema_oid('public');
   get_schema_oid
  ----------------
             2200
  (1 row)
  ```

- pgxc_parse_clog(OUT xid int8, OUT nodename text, OUT status text)

  描述：返回当前集群中所有事务的状态。

  返回值类型：set of record

  示例：

  ```
  openGauss=# SELECT pgxc_parse_clog();
   pgxc_parse_clog
  -----------------
  (0,dn_6004_6005_6006,INPROGRESS)
  (1,dn_6004_6005_6006,COMMITTED)
  (2,dn_6004_6005_6006,INPROGRESS)
  (3 row)
  ```

- pgxc_prepared_xact( )

  描述：返回集群中处于prepared阶段事务GID列表。

  返回值类型：set of text

  示例：

  ```
  openGauss=# SELECT pgxc_prepared_xact();
   pgxc_prepared_xact
  ```



```
--------------------
(0 row)
```

- pgxc_xacts_iscommitted()

  描述：返回集群中指定事务xid的事务的状态。t代表committed，f代表aborted，null代表others。需要sysadmin或者monadmin权限执行。

  返回值类型：set of record

  示例：

```
openGauss=# SELECT pgxc_xacts_iscommitted(1);
 pgxc_xacts_iscommitted
--------------------
(dn_6004_6005_6006,t)
(cn_5001,t)
(cn_5002,t)
(dn_6001_6002_6003,t)
(4 row)
```

- pgxc_total_memory_detail()

  描述：显示集群内存使用情况。需要sysadmin或者monadmin权限执行。

  说明

  若GUC参数enable_memory_limit=off，该函数不能使用。

  返回值类型：set of pv_total_memory_detail

  示例：

```
openGauss=# SELECT pgxc_total_memory_detail();
pgxc_total_memory_detail
--------------------
(dn_6004_6005_6006,max_process_memory,81920)
(dn_6004_6005_6006,process_used_memory,72747)
(dn_6004_6005_6006,max_dynamic_memory,12096)
(dn_6004_6005_6006,dynamic_used_memory,1530)
(4 row)
```

- pv_total_memory_detail

  描述：统计当前数据库节点使用内存的信息，单位为MB。

  说明

  若GUC参数enable_memory_limit=off，该函数不能使用。

  返回值类型：record

表 7-36 返回值说明

名称	类型	描述
nodename	text	节点名称。

名称	类型	描述
memorytype	text	内存类型，包括以下几种： ● max_process_memory：GaussDB集群实例所占用的内存大小。 ● process_used_memory：GaussDB进程所使用的内存大小。 ● max_dynamic_memory：最大动态内存。 ● dynamic_used_memory：已使用的动态内存。 ● dynamic_peak_memory：内存的动态峰值。 ● dynamic_used_shrctx：最大动态共享内存上下文。 ● dynamic_peak_shrctx：共享内存上下文的动态峰值。 ● max_shared_memory：最大共享内存。 ● shared_used_memory：已使用的共享内存。 ● max_sctpcomm_memory：通信库所允许使用的最大内存。 ● sctpcomm_used_memory：通信库已使用的内存大小。 ● sctpcomm_peak_memory：通信库的内存峰值。 ● other_used_memory：其他已使用的内存大小。
memorymbytes	integer	内存类型分配内存的大小。

- get_client_info()
  描述：返回客户端信息。
  返回值类型：record

访问权限查询函数

DDL类权限ALTER、DROP、COMMENT、INDEX、VACUUM属于所有者固有的权限，隐式拥有。

以下访问权限查询函数仅表示用户是否具有某对象上的某种对象权限，即返回记录在系统表acl字段中的对象权限拥有情况。

- has_any_column_privilege(user, table, privilege)
  描述：指定用户是否有访问表任何列的权限。

表 7-37 参数类型说明

参数名	合法入参类型	描述	取值范围
user	name，oid	用户	用户名字或id。
table	text，oid	表	表名称或表id。
privilege	text	权限	• SELECT：允许对指定表任何列执行SELECT语句。 • INSERT：允许对指定表任何列执行INSERT语句。 • UPDATE：允许对指定表任何列任意字段执行UPDATE语句。 • REFERENCES：允许创建一个外键约束（分布式场景暂不支持）。 • COMMENT：允许对指定表任何列执行COMMENT语句。

返回类型：Boolean

- has_any_column_privilege(table, privilege)

  描述：当前用户是否有访问表任何列的权限，合法参数类型见**表7-37**。

  返回类型：Boolean

  备注：has_any_column_privilege检查用户是否以特定方式访问表的任何列。其参数可能与has_table_privilege类似，除了访问权限类型必须是SELECT、INSERT、UPDATE或REFERENCES的一些组合。

  说明

  拥有表的表级别权限则隐含的拥有该表每列的列级权限，因此如果与has_table_privilege参数相同，has_any_column_privilege总是返回true。但是如果授予至少一列的列级权限也返回成功。

- has_column_privilege(user, table, column, privilege)

  描述：指定用户是否有访问列的权限。

表 7-38 参数类型说明

参数名	合法入参类型	描述	取值范围
user	name，oid	用户	用户名字或者id。
table	text，oid	表名	表的名字或id。
column	text，smallint	列名	列的名字或属性号。

参数名	合法入参类型	描述	取值范围
privilege	text	权限	• SELECT：允许对表的指定列执行SELECT语句。 • INSERT：允许对表的指定列执行INSERT语句。 • UPDATE：允许对表的指定列执行UPDATE语句。 • REFERENCES：允许创建一个外键约束（分布式场景暂不支持）。 • COMMENT：允许对表的指定列执行COMMENT语句。

返回类型：Boolean

- has_column_privilege(table, column, privilege)

  描述：当前用户是否有访问列的权限，合法参数类型见**表7-38**。

  返回类型：Boolean

  备注：has_column_privilege检查用户是否以特定方式访问一列。其参数类似于has_table_privilege，可以通过列名或属性号添加列。想要的访问权限类型必须是SELECT、INSERT、UPDATE或REFERENCES的一些组合。

  说明

  拥有表的表级别权限则隐含的拥有该表每列的列级权限。

- has_database_privilege(user, database, privilege)

  描述：指定用户是否有访问数据库的权限。

表 7-39 参数类型说明

参数名	合法入参类型	描述	取值范围
user	name，oid	用户	用户名字或id。
database	text，oid	数据库	数据库名字或id。

参数名	合法入参类型	描述	取值范围
privilege	text	权限	• CREATE：对于数据库，允许在数据库里创建新的模式。 • TEMPORARY：允许在使用数据库的时候创建临时表。 • TEMP：允许在使用数据库的时候创建临时表。 • CONNECT：允许用户连接到指定的数据库。 • ALTER：允许用户修改指定对象的属性。 • DROP：允许用户删除指定的对象。 • COMMENT：允许用户定义或修改指定对象的注释。

返回类型：Boolean

- has_database_privilege(database, privilege)

  描述：当前用户是否有访问数据库的权限，合法参数类型见表7-39。

  返回类型：Boolean

  备注：has_database_privilege检查用户是否能以在特定方式访问数据库。其参数类似has_table_privilege。访问权限类型必须是CREATE、CONNECT、TEMPORARY或TEMP（等价于TEMPORARY）的一些组合。

- has_directory_privilege(user, directory, privilege)

  表 7-40 参数类型说明

参数名	合法入参类型	描述	取值范围
user	name，oid	用户	用户名字或id。
directory	text，oid	目录	目录名字或者oid。
privilege	text	权限	• READ：允许对该目录进行读操作。 • WRITE:允许对该目录进行写操作。

  描述：指定用户是否有访问directory的权限。

  返回类型：Boolean

- has_directory_privilege(directory, privilege)

  描述：当前用户是否有访问directory的权限，合法参数类型见表7-40。

返回类型：Boolean

- has_foreign_data_wrapper_privilege(user, fdw, privilege)

表 7-41 参数类型说明

参数名	合法入参类型	描述	取值范围
user	name，oid	用户	用户名字或者id。
fdw	text，oid	外部数据封装器	外部数据封装器名字或id。
privilege	text	权限	USAGE：允许访问外部数据封装器。

描述：指定用户是否有访问外部数据封装器的权限。

返回类型：Boolean

- has_foreign_data_wrapper_privilege(fdw, privilege)

  描述：当前用户是否有访问外部数据封装器的权限，合法参数类型见表7-41。

  返回类型：Boolean

  备注：has_foreign_data_wrapper_privilege检查用户是否能以特定方式访问外部数据封装器。其参数类似has_table_privilege。访问权限类型必须是USAGE。

- has_function_privilege(user, function, privilege)

表 7-42 参数类型说明

参数名	合法入参类型	描述	取值范围
user	name，oid	用户	用户名字或者id
function	text，oid	函数	函数名称或id
privilege	text	权限	EXECUTE：允许使用指定的函数，以及利用这些函数实现的操作符。 • ALTER：允许用户修改指定对象的属性。 • DROP：允许用户删除指定的对象。 • COMMENT：允许用户定义或修改指定对象的注释。

描述：指定用户是否有访问函数的权限。

返回类型：Boolean

- has_function_privilege(function, privilege)

  描述：当前用户是否有访问函数的权限。合法参数类型见表7-42。

  返回类型：Boolean

备注：has_function_privilege检查一个用户是否能以指定方式访问一个函数。其参数类似has_table_privilege。使用文本字符而不是OID声明一个函数时，允许输入的类型和regprocedure数据类型一样（请参考**对象标识符类型**）。访问权限类型必须是EXECUTE。

- has_language_privilege(user, language, privilege)

表 7-43 参数类型说明

参数名	合法入参类型	描述	取值范围
user	name，oid	用户	用户名字或id。
language	text，oid	语言	语言名称或id。
privilege	text	权限	USAGE：对于过程语言，允许用户在创建函数的时候指定过程语言。

描述：指定用户是否有访问语言的权限。

返回类型：Boolean

- has_language_privilege(language, privilege)

  描述：当前用户是否有访问语言的权限。合法参数类型见**表7-43**。

  返回类型：Boolean

  备注：has_language_privilege检查用户是否能以特定方式访问一个过程语言。其参数类似has_table_privilege。访问权限类型必须是USAGE。

- has_nodegroup_privilege(user, nodegroup, privilege)

  描述：检查用户是否有集群节点访问权限。

  返回类型：Boolean

表 7-44 参数类型说明

参数名	合法入参类型	描述	取值范围
user	name，oid	用户	已存在用户名称或id。
nodegroup	text，oid	集群节点	已存在的集群节点。

参数名	合法入参类型	描述	取值范围
privilege	text	权限	• USAGE：对于子集群，对包含在指定模式中的对象有访问权限时，USAGE允许访问指定子集群下的表对象。 • CREATE：对于子集群，允许在子集群中创建表对象。 • COMPUTE：针对计算子集群，允许用户在具有compute权限的计算子集群上进行弹性计算。 • ALTER：允许用户修改指定对象的属性。 • DROP：允许用户删除指定的对象。

- has_nodegroup_privilege(nodegroup, privilege)
  描述：检查用户是否有集群节点访问权限。
  返回类型：Boolean
- has_schema_privilege(user, schema, privilege)
  描述：指定用户是否有访问模式的权限。
  返回类型：Boolean
- has_schema_privilege(schema, privilege)
  描述：当前用户是否有访问模式的权限。
  返回类型：Boolean
  备注：has_schema_privilege检查用户是否能以特定方式访问一个模式。其参数类似has_table_privilege。访问权限类型必须是CREATE、USAGE、ALTER、DROP或COMMENT的一些组合。
- has_sequence_privilege(user, sequence, privilege)
  描述：指定用户是否有访问序列的权限。
  返回类型：Boolean

表 7-45 参数类型说明

参数名	合法入参类型	描述	取值范围
user	name，oid	用户	已存在用户名称或id。
sequence	text，oid	序列	已存在序列名称或id。

参数名	合法入参类型	描述	取值范围
privilege	text	权限	• USAGE：对于序列，USAGE允许使用nextval函数。 • SELECT：允许创建序列。 • UPDATE：允许执行UPDATE语句。 • ALTER：允许用户修改指定对象的属性。 • DROP：允许用户删除指定的对象。 • COMMENT：允许用户定义或修改指定对象的注释。

- has_sequence_privilege(sequence, privilege)

  描述：指定当前用户是否有访问序列的权限。

  返回类型：Boolean
- has_server_privilege(user, server, privilege)

  描述：指定用户是否有访问外部服务的权限。

  返回类型：Boolean
- has_server_privilege(server, privilege)

  描述：当前用户是否有访问外部服务的权限。

  返回类型：Boolean

  备注：has_server_privilege检查用户是否能以指定方式访问一个外部服务器。其参数类似has_table_privilege。访问权限类型必须是USAGE、ALTER、DROP或COMMENT之一的值。
- has_table_privilege(user, table, privilege)

  描述：指定用户是否有访问表的权限。

  返回类型：Boolean
- has_table_privilege(table, privilege)

  描述：当前用户是否有访问表的权限。

  返回类型：Boolean

  备注：has_table_privilege检查用户是否以特定方式访问表。用户可以通过名称或OID（pg_authid.oid）来指定，public表示为PUBLIC角色，或如果缺省该参数，则使用current_user。该表可以通过名称或者OID声明。如果用名称声明，则在必要时可以用模式进行修饰。如果使用文本字符串来声明所希望的权限类型，这个文本字符串必须是SELECT、INSERT、UPDATE、DELETE、TRUNCATE、REFERENCESTRIGGER、ALTER、DROP、COMMENT、INDEX或VACUUM之一的值。可以给权限类型添加WITH GRANT OPTION，用来测试权限是否拥有授权选项。也可以用逗号分隔列出的多个权限类型，如果拥有任何所列出的权限，则结果便为true。

  示例：

```
openGauss=# CREATE TABLE tt(a int);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
```



```
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
openGauss=# SELECT has_table_privilege('tt', 'select');
 has_table_privilege
---------------------
 t
(1 row)

# 查看数据库用户名列表
openGauss=# \du;                                                List of roles
  Role name  |                                                Attributes                                                |
Member of
-------------
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
+-----------
 omm         | Sysadmin, Create role, Create DB, Replication, Administer audit, Monitoradmin,
Operatoradmin, Policyadmin, UseFT | {}
 simple_user |                                                                                                          | {}
 test        | Sysadmin                                                                                                  | {}



openGauss=# SELECT has_table_privilege('omm', 'tt', 'select,INSERT WITH GRANT OPTION ');
 has_table_privilege
---------------------
 t
(1 row)
```

- has_tablespace_privilege(user, tablespace, privilege)

  描述：指定用户是否有访问表空间的权限。

  返回类型：Boolean

- has_tablespace_privilege(tablespace, privilege)

  描述：当前用户是否有访问表空间的权限。

  返回类型：Boolean

  备注：has_tablespace_privilege检查用户是否能以特定方式访问一个表空间。其参数类似has_table_privilege。访问权限类型必须是CREATE、ALTER、DROP或COMMENT之一的值。

- pg_has_role(user, role, privilege)

  描述：指定用户是否有角色的权限。

  返回类型：Boolean

- pg_has_role(role, privilege)

  描述：当前用户是否有角色的权限。

  返回类型：Boolean

  备注：pg_has_role检查用户是否能以特定方式访问一个角色。其参数类似has_table_privilege，除了public不能用做用户名。访问权限类型必须是MEMBER或USAGE的一些组合。 MEMBER表示的是角色中的直接或间接成员关系（也就是SET ROLE的权限），而USAGE表示无需通过SET ROLE也直接拥有角色的使用权限。

- has_any_privilege(user, privilege)

  描述：指定用户是否有某项ANY权限，若同时查询多个权限，只要具有其中一个则返回true。

  返回类型：Boolean

表 7-46 参数类型说明

参数名	合法入参类型	描述	取值范围
user	name	用户	已存在的用户名。
privilege	text	ANY权限	可选取值： CREATE ANY TABLE [WITH ADMIN OPTION] ALTER ANY TABLE [WITH ADMIN OPTION] DROP ANY TABLE [WITH ADMIN OPTION] SELECT ANY TABLE [WITH ADMIN OPTION] INSERT ANY TABLE [WITH ADMIN OPTION] UPDATE ANY TABLE [WITH ADMIN OPTION] DELETE ANY TABLE [WITH ADMIN OPTION] CREATE ANY SEQUENCE [WITH ADMIN OPTION] CREATE ANY INDEX [WITH ADMIN OPTION] CREATE ANY FUNCTION [WITH ADMIN OPTION] EXECUTE ANY FUNCTION [WITH ADMIN OPTION] CREATE ANY PACKAGE [WITH ADMIN OPTION] EXECUTE ANY PACKAGE [WITH ADMIN OPTION] CREATE ANY TYPE [WITH ADMIN OPTION]

模式可见性查询函数

每个函数执行检查数据库对象类型的可见性。对于函数和操作符，如果在前面的搜索路径中没有相同的对象名称和参数的数据类型，则此对象是可见的。对于操作符类，则要同时考虑名称和相关索引的访问方法。

所有这些函数都需要使用OID来标识需要检查的对象。如果用户想通过名称测试对象，则使用OID别名类型（regclass、regtype、regprocedure、regoperator、regconfig或regdictionary）将会很方便。

比如，如果一个表所在的模式在搜索路径中，并且在前面的搜索路径中没有同名的表，则这个表是可见的。它等效于表可以不带明确模式修饰进行引用。比如，要列出所有可见表的名称：

```
openGauss=# SELECT relname FROM pg_class WHERE pg_table_is_visible(oid);
```

- pg_collation_is_visible(collation_oid)
  描述：该排序是否在搜索路径中可见。
  返回类型：Boolean
- pg_conversion_is_visible(conversion_oid)
  描述：该转换是否在搜索路径中可见。
  返回类型：Boolean
- pg_function_is_visible(function_oid)
  描述：该函数是否在搜索路径中可见。
  返回类型：Boolean
- pg_opclass_is_visible(opclass_oid)
  描述：该操作符类是否在搜索路径中可见。
  返回类型：Boolean
- pg_operator_is_visible(operator_oid)
  描述：该操作符是否在搜索路径中可见。
  返回类型：Boolean
- pg_opfamily_is_visible(opclass_oid)
  描述：该操作符族是否在搜索路径中可见。
  返回类型：Boolean
- pg_table_is_visible(table_oid)
  描述：该表是否在搜索路径中可见。
  返回类型：Boolean
- pg_ts_config_is_visible(config_oid)
  描述：该文本检索配置是否在搜索路径中可见。
  返回类型：Boolean
- pg_ts_dict_is_visible(dict_oid)
  描述：该文本检索词典是否在搜索路径中可见。
  返回类型：Boolean
- pg_ts_parser_is_visible(parser_oid)
  描述：该文本搜索解析是否在搜索路径中可见。
  返回类型：Boolean
- pg_ts_template_is_visible(template_oid)
  描述：该文本检索模板是否在搜索路径中可见。
  返回类型：Boolean
- pg_type_is_visible(type_oid)
  描述：该类型（或域）是否在搜索路径中可见。
  返回类型：Boolean

系统表信息函数

- format_type(type_oid, typemod)

  描述：获取数据类型的SQL名称

  返回类型：text

  备注：format_type通过某个数据类型的OID以及可能的修饰词，返回其SQL名称。如果不知道具体的修饰词，则在修饰词的位置传入NULL。修饰词一般只对有长度限制的数据类型有意义。format_type所返回的SQL名称中包含数据类型的长度值，其大小是：实际存储长度len - sizeof(int32)，单位字节。原因是数据存储时需要32位的空间来存储用户对数据类型的自定义长度信息，即实际存储长度要比用户定义长度多4个字节。在下例中，format_type返回的SQL名称为“character varying(6)”，6表示varchar类型的长度值是6字节，因此该类型的实际存储长度为10字节。

  ```
  openGauss=# SELECT format_type((SELECT oid FROM pg_type WHERE typname='varchar'), 10);
      format_type
  ----------------------
   character varying(6)
  (1 row)
  ```

- pg_check_authid(role_oid)

  描述：检查是否存在给定oid的角色名

  返回类型：bool

  ```
  openGauss=# select pg_check_authid(1);
  pg_check_authid
  -----------------
  f
  (1 row)
  ```

- pg_describe_object(catalog_id, object_id, object_sub_id)

  描述：获取数据库对象的描述

  返回类型：text

  备注：pg_describe_object返回由目录OID，对象OID和一个（或许0个）子对象ID指定的数据库对象的描述。这有助于确认存储在pg_depend系统表中对象的身份。

- pg_get_constraintdef(constraint_oid)

  描述：获取约束的定义

  返回类型：text

- pg_get_constraintdef(constraint_oid, pretty_bool)

  描述：获取约束的定义

  返回类型：text

  备注：pg_get_constraintdef和pg_get_indexdef分别从约束或索引上使用创建命令进行重构。

- pg_get_expr(pg_node_tree, relation_oid)

  描述：反编译表达式的内部形式，假设其中的任何Vars都引用第二个参数指定的关系。

  返回类型：text

- pg_get_expr(pg_node_tree, relation_oid, pretty_bool)

  描述：反编译表达式的内部形式，假设其中的任何Vars都引用第二个参数指定的关系。

  返回类型：text

备注：pg_get_expr反编译一个独立表达式的内部形式，比如一个字段的缺省值。在检查系统表的内容的时候很有用。如果表达式可能包含关键字，则指定引用相关的OID作为第二个参数；如果没有关键字，零就足够了。

- pg_get_functiondef(func_oid)

  描述：获取函数的定义

  返回类型：text

  示例：

```
openGauss=# select * from pg_get_functiondef(598);
 headerlines |                  definition
-------------+-----------------------------------------------
           4 | CREATE OR REPLACE FUNCTION pg_catalog.abbrev(inet)+
             |  RETURNS text                                 +
             |  LANGUAGE internal                            +
             |  IMMUTABLE STRICT NOT FENCED NOT SHIPPABLE    +
             | AS $function$inet_abbrev$function$            +
             |
(1 row)
```

- pg_get_function_arguments(func_oid)

  描述：获取函数定义的参数列表（带默认值）

  返回类型：text

  备注：pg_get_function_arguments返回一个函数的参数列表，需要在CREATE FUNCTION中使用这种格式。

- pg_get_function_identity_arguments(func_oid)

  描述：获取参数列表来确定一个函数 (不带默认值)

  返回类型：text

  备注：pg_get_function_identity_arguments返回需要的参数列表用来标识函数，这种形式需要在ALTER FUNCTION中使用，并且这种形式省略了默认值。

- pg_get_function_result(func_oid)

  描述：获取函数的RETURNS子句

  返回类型：text

  备注：pg_get_function_result为函数返回适当的RETURNS子句。

- pg_get_indexdef(index_oid)

  描述：获取索引的CREATE INDEX命令。

  返回类型：text

  示例：

```
openGauss=# select * from pg_get_indexdef(16416);
                         pg_get_indexdef
-----------------------------------------------------------------------------
 CREATE INDEX test3_b_idx ON test3 USING btree (b) TABLESPACE pg_default
(1 row)
```

- pg_get_indexdef(index_oid, dump_schema_only)

  描述：获取索引的CREATE INDEX命令，仅用于dump场景。当前版本dump_schema_only参数取值对函数输出结果无影响。

  返回类型：text

  示例：

```
openGauss=# select * from pg_get_indexdef(16416, true);
                         pg_get_indexdef
-----------------------------------------------------------------------------
 CREATE INDEX test3_b_idx ON test3 USING btree (b) TABLESPACE pg_default
(1 row)
```


- pg_get_indexdef(index_oid, column_no, pretty_bool)

  描述：获取索引的CREATE INDEX命令，或者如果column_no不为零，则只获取一个索引字段的定义。

  返回类型：text

  示例：

```
openGauss=# select * from pg_get_indexdef(16416, 0, false);
                            pg_get_indexdef
-------------------------------------------------------------------------------
 CREATE INDEX test3_b_idx ON test3 USING btree (b) TABLESPACE pg_default
(1 row)
openGauss=# select * from pg_get_indexdef(16416, 1, false);
 pg_get_indexdef
-----------------
 b
(1 row)
```

  备注：pg_get_functiondef为函数返回一个完整的CREATE OR REPLACE FUNCTION语句。

- pg_get_keywords()

  描述：获取SQL关键字和类别列表

  返回类型：setof record

  备注：pg_get_keywords返回一组关于描述服务器识别SQL关键字的记录。word列包含关键字。catcode列包含一个分类代码：U表示通用的，C表示列名，T表示类型或函数名，或R表示保留。catdesc列包含了一个可能本地化描述分类的字符串。

- pg_get_ruledef(rule_oid)

  描述：获取规则的CREATE RULE命令

  返回类型：text

- pg_get_ruledef(rule_oid, pretty_bool)

  描述：获取规则的CREATE RULE命令

  返回类型：text

- pg_get_userbyid(role_oid)

  描述：获取给定OID的角色名

  返回类型：name

  备注：pg_get_userbyid通过角色的OID抽取对应的用户名。

- pg_check_authid(role_id)

  描述：通过role_id检查用户是否存在

  返回类型：text

```
openGauss=# select pg_check_authid(20);
pg_check_authid
-----------------
f
(1 row)
```

- pg_get_viewdef(view_name)

  描述：为视图获取底层的SELECT命令

  返回类型：text

- pg_get_viewdef(view_name, pretty_bool)

  描述：为视图获取底层的SELECT命令，如果pretty_bool为true，行字段可以包含80列。

返回类型：text

备注：pg_get_viewdef重构出定义视图的SELECT查询。这些函数大多数都有两种形式，其中带有pretty_bool参数，且参数为true时，是"适合打印"的结果，这种格式更容易读。另一种是缺省的格式，更有可能被将来的不同版本用同样的方法解释。如果是用于转储，那么尽可能避免使用适合打印的格式。给pretty-print参数传递false生成的结果和没有这个参数的变种生成的结果完全一样。

- pg_get_viewdef(view_oid)

  描述：为视图获取底层的SELECT命令

  返回类型：text

- pg_get_viewdef(view_oid, pretty_bool)

  描述：为视图获取底层的SELECT命令，如果pretty_bool为true，行字段可以包含80列。

  返回类型：text

- pg_get_viewdef(view_oid, wrap_column_int)

  描述：为视图获取底层的SELECT命令；行字段被换到指定的列数，打印是隐含的。

  返回类型：text

- pg_get_tabledef(table_oid)

  描述：根据table_oid获取表定义。

  返回类型：text

  示例：

```
openGauss=# select * from pg_get_tabledef(16384);
                  pg_get_tabledef
------------------------------------------------------------
 SET search_path = public;                          +
 CREATE  TABLE t1 (                                +
        c1 bigint DEFAULT nextval('serial'::regclass)+
 )                                                  +
 WITH (orientation=row)              +
 DISTRIBUTE BY HASH(c1)                             +
 TO GROUP group1;
(1 row)
```

- pg_get_tabledef(table_name)

  描述：根据table_name获取表定义。

  返回类型：text

  示例：

```
openGauss=# select * from pg_get_tabledef('t1');
                  pg_get_tabledef
------------------------------------------------------------
 SET search_path = public;                          +
 CREATE  TABLE t1 (                                +
        c1 bigint DEFAULT nextval('serial'::regclass)+
 )                                                  +
 WITH (orientation=row)              +
 DISTRIBUTE BY HASH(c1)                             +
 TO GROUP group1;
(1 row)
```

  备注：pg_get_tabledef重构出表定义的CREATE语句，包含了表定义本身、索引信息、comments信息。对于表对象依赖的group、schema、tablespace、server等信息，需要用户自己去创建，表定义里不会有这些对象的创建语句。

- pg_options_to_table(reloptions)

  描述：获取存储选项名称/值对的集合

返回类型：setof record

备注：pg_options_to_table当通过pg_class.reloptions或pg_attribute.attoptions时返回存储选项名称/值对（option_name/option_value）的集合。

- pg_tablespace_databases(tablespace_oid)

  描述：获取在指定的表空间中有对象的数据库OID集合

  返回类型：setof oid

  备注：pg_tablespace_databases允许检查表空间的状况，返回在该表空间中保存了对象的数据库OID集合。如果这个函数返回数据行，则该表空间就是非空的，因此不能删除。要显示该表空间中的特定对象，用户需要连接pg_tablespace_databases标识的数据库与查询pg_class系统表。

- pg_tablespace_location(tablespace_oid)

  描述：获取表空间所在的文件系统的路径

  返回类型：text

- pg_typeof(any)

  描述：获取任何值的数据类型

  返回类型：regtype

  备注：pg_typeof返回传递给他的值的数据类型OID。这可能有助于故障排除或动态构造SQL查询。声明此函数返回regtype，这是一个OID别名类型（请参考**对象标识符类型**）；这意味着它是一个为了比较而显示类型名称的OID。

  示例：

  ```
  openGauss=# SELECT pg_typeof(33);
   pg_typeof
  -----------
   integer
  (1 row)

  openGauss=# SELECT typlen FROM pg_type WHERE oid = pg_typeof(33);
   typlen
  --------
       4
  (1 row)
  ```

- collation for (any)

  描述：获取参数的排序

  返回类型：text

  备注：表达式collation for返回传递给他的值的排序。示例：

  ```
  openGauss=# SELECT collation for (description) FROM pg_description LIMIT 1;
   pg_collation_for
  ------------------
   "default"
  (1 row)
  ```

  值可能是引号括起来的并且模式限制的。如果没有为参数表达式排序，则返回一个null值。如果参数不是排序的类型，则抛出一个错误。

- getdistributekey(table_name)

  描述：获取一个hash表的分布列。

  返回类型：text

  示例：

  ```
  openGauss=# SELECT getdistributekey('item');
   getdistributekey
  ------------------
  ```



```
 i_item_sk
(1 row)
```

- pg_get_serial_sequence(tablename, colname)

  描述：获取对应表名和列名上的序列。

  返回类型：text

  示例：

```
openGauss=# select * from pg_get_serial_sequence('t1', 'c1');
 pg_get_serial_sequence
------------------------
 public.serial
(1 row)
```

- pg_sequence_parameters(sequence_oid)

  描述：获取指定sequence的参数，包含起始值，最小值和最大值，递增值等。

  返回类型：int16，int16，int16，int16，Boolean

  示例：

```
openGauss=# select * from pg_sequence_parameters(16420);
 start_value | minimum_value |    maximum_value    | increment | cycle_option
-------------+---------------+---------------------+-----------+--------------
         101 |             1 | 9223372036854775807 |         1 | f
(1 row)
```

- pgxc_get_variable_info( )

  描述：获取节点上的变量值，包括nodeName，nextOid，nextXid，oldestXid，xidVacLimit，oldestXidDB，lastExtendCSNLogpage，startExtendCSNLogpage，nextCommitSeqNo，latestCompleteXid,startupMaxXid。

  返回类型：set of pg_variable_info

  示例：

```
openGauss=# select pgxc_get_variable_info( );
                    pgxc_get_variable_info
-----------------------------------------------------------------------------
(dn_6004_6005_6006,25617,141396349,2073,20000002073,15808,138111,0,127154152,141396348,104
433004)
(1 row)
```

- gs_get_index_status(schema_name, index_name)

  描述：获取所有节点上的索引状态信息，包括索引是否可插入以及索引是否可用，主要用于在线创建索引过程中或者构建失败的情况下，确认索引状态。返回值包括节点名称node_name，索引是否可插入状态indisready，索引是否可以用状态indisvalid，只有当所有节点上索引的indisready和indisvalid状态均为true，且索引状态没有被修改为unusable的情况下，当前索引才“可用”。

  返回类型：text，boolean，boolean

  示例：

```
openGauss=# select * from gs_get_index_status('public', 'index1');
   node_name    | indisready | indisvalid
----------------+------------+------------
 datanode1      | t          | t
 datanode2      | t          | t
 coordinator1   | t          | t
(3 row)
```

注释信息函数

- col_description(table_oid, column_number)

  描述：获取一个表字段的注释

返回类型：text

备注：col_description返回一个表中字段的注释，通过表OID和字段号来声明。

- obj_description(object_oid, catalog_name)

  描述：获取一个数据库对象的注释

  返回类型：text

  备注：带有两个参数的obj_description返回一个数据库对象的注释，该对象是通过其OID和其所属的系统表名称声明。比如，obj_description(123456,'pg_class')将返回OID为123456的表的注释。只带一个参数的obj_description只要求对象OID。

  obj_description不能用于表字段，因为字段没有自己的OID。

- obj_description(object_oid)

  描述：获取一个数据库对象的注释

  返回类型：text

- shobj_description(object_oid, catalog_name)

  描述：获取一个共享数据库对象的注释

  返回类型：text

  备注：shobj_description和obj_description差不多，不同之处仅在于前者用于共享对象。一些系统表是通用于集群中所有数据库的全局表，因此这些表的注释也是全局存储的。

事务 ID 和快照

内部事务ID类型（xid）是64位。这些函数使用的数据类型txid_snapshot，存储在特定时刻事务ID可见性的信息。其组件描述在**表7-47**。

表 7-47 快照组件

名称	描述
xmin	最早的事务ID（txid）仍然活动。所有较早事务将是已经提交可见的，或者是直接回滚。
xmax	作为尚未分配的txid。所有大于或等于此txids的都是尚未开始的快照时间，因此不可见。
xip_list	当前快照中活动的txids。这个列表只包含在xmin和xmax之间活动的txids；有可能活动的txids高于xmax。介于大于等于xmin、小于xmax，并且不在这个列表中的txid，在这个时间快照已经完成的，因此按照提交状态查看他是可见还是回滚。这个列表不包含子事务的txids。

txid_snapshot的文本表示为：xmin:xmax:xip_list。

示例：10:20:10,14,15意思为：xmin=10, xmax=20, xip_list=10, 14, 15。

以下的函数在一个输出形式中提供服务器事务信息。这些函数的主要用途是为了确定在两个快照之间有哪个事务提交。

- pgxc_is_committed(transaction_id)

描述：如果提交或忽略给定的XID（gxid）。NULL表示的状态是未知的（运行、准备、冻结等）。

返回类型：bool

- txid_current()

  描述：获取当前事务ID。

  返回类型：bigint

- gs_txid_oldestxmin()

  描述：获取当前最小事务id的值oldesxmin。

  返回类型：bigint

- txid_current_snapshot()

  描述：获取当前快照。

  返回类型：txid_snapshot

- txid_snapshot_xip(txid_snapshot)

  描述：在快照中获取正在进行的事务ID。

  返回类型：setof bigint

- txid_snapshot_xmax(txid_snapshot)

  描述：获取快照的xmax。

  返回类型：bigint

- txid_snapshot_xmin(txid_snapshot)

  描述：获取快照的xmin。

  返回类型：bigint

- txid_visible_in_snapshot(bigint, txid_snapshot)

  描述：在快照中事务ID是否可见（不使用子事务ID）。

  返回类型：Boolean

- get_local_prepared_xact()

  描述：获取当前节点两阶段残留事务信息，包括事务id，两阶段gid名称，prepared的时间，owner的oid，database的oid及当前节点的node_name。

  返回类型：xid, text, timestamptz, oid, oid，text

- get_remote_prepared_xacts()

  描述：获取所有远程节点两阶段残留事务信息，包括事务id，两阶段gid名称，prepared的时间，owner的名称，database的名称及node_name。

  返回类型：xid, text, timestamptz, name, name，text

- global_clean_prepared_xacts(text, text)

  描述：并发清理两阶段残留事务，仅gs_clean工具调用该函数进行清理，其他情况下调用均返回false。

  返回类型：Boolean

- pgxc_stat_get_wal_senders()

  描述：返回集群中所有主DN的发送日志的信息和其对应的备DN的接收日志的信息。仅支持system admin或monitor admin权限用户使用。

  返回值如下：

表 7-48 pgxc_stat_get_wal_senders 返回参数说明

字段名	描述
nodename	实例名。
sender_pid	发送日志的线程pid。
local_role	实例角色。
peer_role	接收端的实例的角色。
peer_state	接收端的实例的状态。
state	实例间同步的状态。
sender_sent_location	发送端发送日志的位置。
sender_write_location	发送端写日志的位置。
sender_flush_location	发送端刷盘日志的位置。
sender_replay_location	当前实例的日志位置。如果是主DN，则该位置和sender_flush_location相同；否则该位置为当前实例日志回放到的位置。
receiver_received_location	接收端日志接收到的位置。
receiver_write_location	接收端日志写的位置。
receiver_flush_location	接收端日志刷盘的位置。
receiver_replay_location	接收端日志回放的位置。

- pgxc_stat_get_wal_senders_status()

  描述：返回所有节点事务日志接收状态。仅支持system admin或monitor admin权限用户使用。

  返回值如下：

  表 7-49 pgxc_stat_get_wal_senders_status 返回参数说明

字段名	描述
nodename	主节点名。
source_ip	主节点IP。
source_port	主节点端口。

字段名	描述
dest_ip	备节点IP。
dest_port	备节点端口。
sender_pid	发送线程PID。
local_role	主节点类型。
peer_role	备节点类型。
peer_state	备节点状态。
state	wal sender状态。
sender_sent_location	主节点发送位置。
sender_write_location	主节点落盘位置。
sender_replay_location	主节点redo位置。
receiver_received_location	备节点接收位置。
receiver_write_location	备节点落盘位置。
receiver_flush_location	备节点flush磁盘位置。
receiver_replay_location	备节点redo位置。

- gs_get_next_xid_csn()
  描述：返回全局所有节点上的next_xid和next_csn值。
  返回值如下：

  表 7-50 gs_get_next_xid_csn 返回参数说明

字段名	描述
nodename	节点名称。
next_xid	当前节点下一个事务id号。
next_csn	当前节点下一个csn号。

- simsearch_lib_load_status()
  描述: 搜索动态库的状态加载成功或者失败。
  返回值类型：SETOF record
- simsearch_gpu_vector_status()
  描述：搜索searchlet的状态是否有向量。
  返回值类型：SETOF record

  说明

  由于规格变更，当前版本已经不再支持该函数，请不要使用。

- pg_control_system()
  描述：返回系统控制文件状态。
  返回类型：SETOF record
- pg_control_checkpoint()
  描述：返回系统检查点状态。
  返回类型：SETOF record
- get_prepared_pending_xid
  描述：当恢复完成时，返回nextxid。
  参数：nan
  返回值类型：text
- pg_clean_region_info
  描述：清理regionmap。
  参数：nan
  返回值类型：character varying
- pg_get_delta_info
  描述：从单个dn获取delta info。
  参数：rel text, schema_name text
  返回值类型：part_name text, live_tuple bigint, data_size bigint, blocknum bigint
- pgxc_get_delta_info
  描述：从全部dn获取delta info。仅sysadmin和monitor admin用户可以访问。
  参数：rel text, schema_name text
  返回值类型：part_name text, live_tuple bigint, data_size bigint, blocknum bigint
- pg_get_replication_slot_name
  描述：获取slot name。
  参数：nan
  返回值类型：text
- pg_get_running_xacts
  描述：获取运行中的xact。
  参数：nan
  返回值类型：handle integer, gxid xid, state tinyint, node text, xmin xid, vacuum boolean, timeline bigint, prepare_xid xid, pid bigint, next_xid xid
- pg_get_variable_info
  描述：获取共享内存变量cache。
  参数：nan
  返回值类型：node_name text, nextOid oid, nextXid xid, oldestXid xid, xidVacLimit xid, oldestXidDB oid, lastExtendCSNLogpage xid, startExtendCSNLogpage xid, nextCommitSeqNo xid, latestCompletedXid xid, startupMaxXid xid
- pg_get_xidlimit
  描述：从共享内存获取事务id信息。

参数：nan

返回值类型：nextXid xid, oldestXid xid, xidVacLimit xid, xidWarnLimit xid, xidStopLimit xid, xidWrapLimit xid, oldestXidDB oid

- pg_stat_file_recursive

  描述：列出路径下所有文件。

  参数：location text

  返回值类型：path text, filename text, size bigint, isdir boolean

- pg_stat_get_activity_for_temptable

  描述：返回临时表相关的后台进程的记录。

  参数：nan

  返回值类型：datid oid, timelineid integer, tempid integer, sessionid bigint

- pg_stat_get_activity_ng

  描述：返回nodegroup相关的后台进程的记录。

  参数：pid bigint

  返回值类型：datid oid, pid bigint, sessionid bigint, node_group text

- pg_stat_get_cgroup_info

  描述：返回cgroup信息。

  参数：nan

  返回值类型：cgroup_name text, percent integer, usage_percent integer, shares bigint, usage bigint, cpuset text, relpath text, valid text, node_group text

- pg_stat_get_realtime_info_internal

  描述：返回实时信息，当前该接口已不可用，返回FailedToGetSessionInfo。

  参数：oid, oid, bigint, cstring, oid

  返回值类型：text

- pg_stat_get_session_wlmstat

  描述：返回当前会话负载信息。

  参数：pid integer

  返回值类型：datid oid, threadid bigint, sessionid bigint, threadpid integer, usesysid oid, appname text, query text, priority bigint, block_time bigint, elapsed_time bigint, total_cpu_time bigint, skew_percent integer, statement_mem integer, active_points integer, dop_value integer, current_cgroup text, current_status text, enqueue_state text, attribute text, is_plana boolean, node_group text, srespool name

- pg_stat_get_wlm_instance_info

  描述：返回当前实例负载信息。

  参数：nan

  返回值类型：instancename text, timestamp timestamp with time zone, used_cpu integer, free_memory integer, used_memory integer, io_await double precision, io_util double precision, disk_read double precision, disk_write double precision, process_read bigint, process_write bigint, logical_read bigint, logical_write bigint, read_counts bigint, write_counts bigint

- pg_stat_get_wlm_instance_info_with_cleanup

  描述：返回当前实例负载信息，并且保存到系统表中。

参数：nan

返回值类型：instancename text, timestamp timestamp with time zone, used_cpu integer, free_memory integer, used_memory integer, io_await double precision, io_util double precision, disk_read double precision, disk_write double precision, process_read bigint, process_write bigint, logical_read bigint, logical_write bigint, read_counts bigint, write_counts bigint

- pg_stat_get_wlm_node_resource_info

  描述：获取当前节点资源信息。

  参数：nan

  返回值类型：min_mem_util integer, max_mem_util integer, min_cpu_util integer, max_cpu_util integer, min_io_util integer, max_io_util integer, used_mem_rate integer

- pg_stat_get_wlm_operator_info

  描述：从内部哈希表中获取执行计划算子信息。

  参数：nan

  返回值类型：queryid bigint, pid bigint, plan_node_id integer, plan_node_name text, start_time timestamp with time zone, duration bigint, query_dop integer, estimated_rows bigint, tuple_processed bigint, min_peak_memory integer, max_peak_memory integer, average_peak_memory integer, memory_skew_percent integer, min_spill_size integer, max_spill_size integer, average_spill_size integer, spill_skew_percent integer, min_cpu_time bigint, max_cpu_time bigint, total_cpu_time bigint, cpu_skew_percent integer, warning text

- pg_stat_get_wlm_realtime_operator_info

  描述：从内部哈希表中获取实时执行计划算子信息。

  参数：nan

  返回值类型：queryid bigint, pid bigint, plan_node_id integer, plan_node_name text, start_time timestamp with time zone, duration bigint, status text, query_dop integer, estimated_rows bigint, tuple_processed bigint, min_peak_memory integer, max_peak_memory integer, average_peak_memory integer, memory_skew_percent integer, min_spill_size integer, max_spill_size integer, average_spill_size integer, spill_skew_percent integer, min_cpu_time bigint, max_cpu_time bigint, total_cpu_time bigint, cpu_skew_percent integer, warning text

- pg_stat_get_wlm_realtime_session_info

  描述：返回实时会话负载信息。

  参数：nan

  返回值类型：nodename text, threadid bigint, block_time bigint, duration bigint, estimate_total_time bigint, estimate_left_time bigint, schemaname text, query_band text, spill_info text, control_group text, estimate_memory integer, min_peak_memory integer, max_peak_memory integer, average_peak_memory integer, memory_skew_percent integer, min_spill_size integer, max_spill_size integer, average_spill_size integer, spill_skew_percent integer, min_dn_time bigint, max_dn_time bigint, average_dn_time bigint, dntime_skew_percent integer, min_cpu_time bigint, max_cpu_time bigint, total_cpu_time bigint, cpu_skew_percent integer, min_peak_iops integer, max_peak_iops integer, average_peak_iops integer, iops_skew_percent integer, warning text, query text, query_plan text, cpu_top1_node_name text,

cpu_top2_node_name text, cpu_top3_node_name text, cpu_top4_node_name text, cpu_top5_node_name text, mem_top1_node_name text, mem_top2_node_name text, mem_top3_node_name text, mem_top4_node_name text, mem_top5_node_name text, cpu_top1_value bigint, cpu_top2_value bigint, cpu_top3_value bigint, cpu_top4_value bigint, cpu_top5_value bigint, mem_top1_value bigint, mem_top2_value bigint, mem_top3_value bigint, mem_top4_value bigint, mem_top5_value bigint, top_mem_dn text, top_cpu_dn text

- pg_stat_get_wlm_session_info_internal

  描述：返回会话负载信息。

  参数：oid, bigint, bigint, oid

  返回值类型：SETOF text

- pg_stat_get_wlm_session_iostat_info

  描述：返回会话负载I/O信息。

  参数：nan

  返回值类型：threadid bigint, maxcurr_iops integer, mincurr_iops integer, maxpeak_iops integer, minpeak_iops integer, iops_limits integer, io_priority integer, curr_io_limits integer

- pg_stat_get_wlm_statistics

  描述：返回会话负载统计数据。

  参数：nan

  返回值类型：statement text, block_time bigint, elapsed_time bigint, total_cpu_time bigint, qualification_time bigint, skew_percent integer, control_group text, status text, action text

- pg_test_err_contain_err

  描述：测试错误类型和返回信息。

  参数：integer

  返回值类型：void

- pv_session_memory_detail_tp

  描述：返回会话的内存使用情况，参考pv_session_memory_detail。

  参数：nan

  返回值类型：sessid text, sesstype text, contextname text, level smallint, parent text, totalsize bigint, freesize bigint, usedsize bigint

- gs_get_table_distribution

  描述：返回表数据在各个数据节点的分布情况。

  参数：table_name text, schema_name text

  返回值类型：text

- pv_builtin_functions

  描述：查看所有内置系统函数信息。

  参数：nan

  返回值类型：proname name, pronamespace oid, proowner oid, prolang oid, procost real, prorows real, provariadic oid, protransform regproc, proisagg boolean, proiswindow boolean, prosecdef boolean, proleakproof boolean, proisstrict boolean, proretset boolean, provolatile "char", pronargs smallint,

pronargdefaults smallint, prorettype oid, proargtypes oidvector, proallargtypes integer[], proargmodes "char"[], proargnames text[], proargdefaults pg_node_tree, prosrc text, probin text, proconfig text[], proacl aclitem[], prodefaultargpos int2vector, fencedmode boolean, proshippable boolean, propackage boolean, oid oid

- pv_thread_memory_detail

  描述：返回各线程的内存信息。

  参数：nan

  返回值类型：threadid text, tid bigint, thrdtype text, contextname text, level smallint, parent text, totalsize bigint, freesize bigint, usedsize bigint

- pg_shared_memory_detail

  描述：返回所有已产生的共享内存上下文的使用信息，各列描述请参考**SHARED_MEMORY_DETAIL**。

  参数：nan

  返回值类型：contextname text, level smallint, parent text, totalsize bigint, freesize bigint, usedsize bigint

- pgxc_get_running_xacts

  描述：返回集群中各个节点运行事务的信息，字段内容和**PGXC_RUNNING_XACTS**相同。只有system admin和monitor admin用户有权限查看。

  参数：nan

  返回值类型：setof record

- pgxc_snapshot_status()

  描述：在GTM模式下，返回GTM中关键内存信息状态，用来支持问题定位，GTM-Free、GTM-Lite不支持本函数。

  参数：nan

  返回值类型：xmin xid, xmax xid, xcnt int, oldestxmin xid, next_xid xid, timeline int, active_thread_num int, max_active_thread_num int, snapshot_num int, snapshot_totalsize bigint

  返回值描述如下：

  表 7-51 get_gtm_lite_status 返回参数说明

字段名	描述
xmin	GTM上当前最小的活跃事务id。
xmax	GTM上当前提交的最大的事务id + 1，大于等于该值的事务id是活跃的。
xcnt	GTM上当前活跃事务个数。
oldestxmin	GTM上最老被访问的事务id号。
next_xid	GTM上下一个分配的事务id号。
timeline	GTM上当前的时间线。
active_thread_num	GTM上当前活跃的工作线程数。

字段名	描述
max_active_thread_num	GTM上1分钟内工作线程数峰值。
snapshot_num	GTM上1分钟内下发的快照个数。
snapshot_totalsize	GTM上1分钟内下发快照总大小。

- get_gtm_lite_status()

  描述：返回GTM上的backupXid和csn号，用来支持问题定位，GTM-FREE模式下不支持使用本系统函数。

  返回值如下：

  表 7-52 get_gtm_lite_status 返回参数说明

字段名	描述
backup_xid	GTM上的备份Xid值。
csn	GTM当前下发的最新的csn号。

7.5.23 系统管理函数

7.5.23.1 配置设置函数

配置设置函数是可以用于查询以及修改运行时配置参数的函数。

- current_setting(setting_name)

  描述：当前的设置值。

  返回值类型：text

  备注：current_setting用于以查询形式获取setting_name的当前值。和SQL语句SHOW是等效的。比如：

  ```
  openGauss=# SELECT current_setting('datestyle');

   current_setting
  -----------------
   ISO, MDY
  (1 row)
  ```

- set_working_grand_version_num_manually(tmp_version)

  描述：通过切换授权版本号来更新和升级高斯数据库的新特性。

  返回值类型：void

- shell_in(type)

  描述：为shell类型输入路由（那些尚未填充的类型）。

  返回值类型：void

- shell_out(type)

  描述：为shell类型输出路由（那些尚未填充的类型）。

  返回值类型：void

- set_config(setting_name, new_value, is_local)

  描述：设置参数并返回新值。

  返回值类型：text

  备注：set_config将参数setting_name设置为new_value，如果is_local为true，则新值将只应用于当前事务。如果希望new_value应用于当前会话，可以使用false，和SQL语句SET是等效的。

  示例：

```
openGauss=# SELECT set_config('log_statement_stats', 'off', false);

 set_config
------------
 off
(1 row)
```

7.5.23.2 通用文件访问函数

通用文件访问函数提供了对数据库服务器上的文件的本地访问接口。只有数据库集群目录和log_directory目录里面的文件可以访问。使用相对路径访问集群目录里面的文件，以及匹配log_directory配置而设置的路径访问日志文件。只有数据库初始化用户才能使用这些函数。

- pg_ls_dir(dirname text)

  描述：列出目录中的文件。

  返回值类型：setof text

  备注：pg_ls_dir返回指定目录里面的除了特殊项“.”和“..”之外所有名称。

  示例：

```
openGauss=# SELECT pg_ls_dir('./');
     pg_ls_dir
----------------------
 .postgresql.conf.swp
 postgresql.conf
 pg_tblspc
 PG_VERSION
 pg_ident.conf
 core
 server.crt
 pg_serial
 pg_twophase
 postgresql.conf.lock
 pg_stat_tmp
 pg_notify
 pg_subtrans
 pg_ctl.lock
 pg_xlog
 pg_clog
 base
 pg_snapshots
 postmaster.opts
 postmaster.pid
 server.key.rand
 server.key.cipher
 pg_multixact
 pg_errorinfo
 server.key
 pg_hba.conf
 pg_replslot
 .pg_hba.conf.swp
 cacert.pem
 pg_hba.conf.lock
 global
```



```
 gaussdb.state
(32 rows)
```

- pg_read_file(filename text, offset bigint, length bigint)

  描述：返回一个文本文件的内容。

  返回值类型：text

  备注：pg_read_file返回一个文本文件的一部分，从offset开始，最多返回length字节（如果先达到文件结尾，则小于这个数值）。如果offset是负数，则它是相对于文件结尾回退的长度。如果省略了offset和length，则返回整个文件。

  示例：

```
openGauss=# SELECT pg_read_file('postmaster.pid',0,100);
            pg_read_file
----------------------------------------
 53078                                  +
 /srv/BigData/testdir/data1/coordinator+
 1500022474                             +
 8000                                   +
 /var/run/FusionInsight                 +
 localhost                              +
  2
(1 row)
```

- pg_read_binary_file(filename text [, offset bigint, length bigint,missing_ok boolean])

  描述：返回一个二进制文件的内容。

  返回值类型：bytea

  备注：pg_read_binary_file的功能与pg_read_file类似，除了结果的返回值为bytea类型不一致，相应地不会执行编码检查。与convert_from函数结合，这个函数可以用来读取用指定编码的一个文件。

```
openGauss=# SELECT convert_from(pg_read_binary_file('filename'), 'UTF8');
```

- pg_stat_file(filename text)

  描述：返回一个文本文件的状态信息。

  返回值类型：record

  备注：pg_stat_file返回一条记录，其中包含：文件大小、最后访问时间戳、最后更改时间戳、最后文件状态修改时间戳以及标识传入参数是否为目录的Boolean值。典型的用法：

```
openGauss=# SELECT * FROM pg_stat_file('filename');
openGauss=# SELECT (pg_stat_file('filename')).modification;
```

  示例：

```
openGauss=# SELECT convert_from(pg_read_binary_file('postmaster.pid'), 'UTF8');
            convert_from
----------------------------------------
 4881                                   +
 /srv/BigData/gaussdb/data1/coordinator+
 1496308688                             +
 25108                                  +
 /opt/huawei/Bigdata/gaussdb/gaussdb_tmp +
 *                                      +
  25108001  43352069                    +

(1 row)
openGauss=# SELECT * FROM pg_stat_file('postmaster.pid');

 size |         access         |      modification      |         change
| creation | isdir
------+------------------------+------------------------+------------------------
+----------+-------
  117 | 2017-06-05 11:06:34+08 | 2017-06-01 17:18:08+08 | 2017-06-01 17:18:08+08
```



```
|          | f
(1 row)
openGauss=# SELECT (pg_stat_file('postmaster.pid')).modification;
      modification
------------------------
 2017-06-01 17:18:08+08
(1 row)
```

7.5.23.3 服务器信号函数

服务器信号函数向其他服务器进程发送控制信号。只有系统管理员有权执行以下函数。

- pg_cancel_backend(pid int)

  描述：取消一个后端线程正在执行的语句。

  返回值类型：Boolean

  备注：pg_cancel_backend向由pid标识的后端进程发送一个查询取消（SIGINT）信号。一个活动的后端进程的PID可以从pg_stat_activity视图的pid字段找到，或者在服务器上用ps列出数据库进程。具有SYSADMIN权限的用户，后端进程所连接的数据库的属主，后端进程的属主或者继承了内置角色gs_role_signal_backend权限的用户有权使用该函数。

- pg_cancel_session(pid bigint, sessionid bigint)

  描述：线程池模式下，取消一个活跃状态会话正在执行的语句。

  返回值类型：Boolean

  备注：pg_cancel_session的入参可以通过pg_stat_activity中的pid字段和sessionid的字段查询，可以用于清理线程池模式下，活跃状态会话正在执行的语句。当入参pid和sessionid相同，且均为线程id时，功能和pg_cancel_backend相同。

- pg_cancel_invalid_query()

  描述：取消一个后端的无效查询。

  返回值类型：Boolean

  备注：只有系统管理员才有权限取消连接到降级的GTM的后端中运行的查询。

- pg_reload_conf()

  描述：导致所有服务器进程重新装载它们的配置文件。

  返回值类型：Boolean

  备注：pg_reload_conf给服务器发送一个SIGHUP信号，导致所有服务器进程重新装载配置文件。

- pg_rotate_logfile()

  描述：滚动服务器的日志文件。

  返回值类型：Boolean

  备注：pg_rotate_logfile给日志文件管理器发送信号，告诉它立即切换到一个新的输出文件。这个函数只有在redirect_stderr用于日志输出的时候才有用，否则根本不存在日志文件管理器子进程。

- pg_terminate_session(pid bigint, sessionid bigint)

  描述：线程池模式下，终止一个后台会话。

  返回值类型：Boolean

  备注：本函数的入参可以通过pg_stat_activity中的pid字段和sessionid的字段查询。具有SYSADMIN权限的用户，会话所连接的数据库的属主，会话的属主或者继承了内置角色gs_role_signal_backend权限的用户有权使用该函数。

须知

当入参pid和sessionid相同，且均为线程id时，该函数可终止非线程池的线程、活跃状态的线程池线程。

当入参pid和sessionid不同时，该函数可终止活跃状态的会话，或关闭非活跃状态会话和客户端的socket连接。

- pg_terminate_active_session_socket(pid int64, sessionid int64)

  描述：关闭一个活跃session和客户端的socket连接。

  返回值类型：Boolean

  备注：如果成功，函数返回true，否则返回false。仅限初始化用户才可使用该函数。

- pg_terminate_backend(pid int)

  描述：终止一个后台线程。

  返回值类型：Boolean

  备注：如果成功，函数返回true，否则返回false。具有SYSADMIN权限的用户，后端线程所连接的数据库的属主，后端线程的属主或者继承了内置角色gs_role_signal_backend权限的用户有权使用该函数。

须知

该函数可终止非线程池的线程、活跃状态的线程池线程，但无法终止非活跃状态的线程池线程。

示例：

```
openGauss=# SELECT pid from pg_stat_activity;
       pid
-----------------
 140657876268816
(1 rows)

openGauss=# SELECT pg_terminate_backend(140657876268816);
 pg_terminate_backend
----------------------
 t
(1 row)
```

7.5.23.4 备份恢复控制函数

备份控制函数

备份控制函数可帮助进行在线备份。

- pg_create_restore_point(name text)

  描述：为执行恢复创建一个命名点。（需要管理员角色）

  返回值类型：text

  备注：pg_create_restore_point创建了一个可以用作恢复目的、有命名的事务日志记录，并返回相应的事务日志位置。在恢复过程中，recovery_target_name可以通过这个名称定位对应的日志恢复点，并从此处开始执行恢复操作。避免使用

相同的名称创建多个恢复点，因为恢复操作将在第一个匹配（恢复目标）的名称上停止。

- pg_current_xlog_location()

  描述：获取当前事务日志的写入位置。

  返回值类型：text

  备注：pg_current_xlog_location使用与前面那些函数相同的格式显示当前事务日志的写入位置。如果是只读操作，不需要系统管理员权限。

- pg_current_xlog_insert_location()

  描述：获取当前事务日志的插入位置。

  返回值类型：text

  备注：pg_current_xlog_insert_location显示当前事务日志的插入位置。插入点是事务日志在某个瞬间的“逻辑终点”，而实际的写入位置则是从服务器内部缓冲区写出时的终点。写入位置是可以从服务器外部检测到的终点，如果要归档部分完成事务日志文件，则该操作即可实现。插入点主要用于服务器调试目的。如果是只读操作，不需要系统管理员权限。

- gs_current_xlog_insert_end_location()

  描述：获取当前事务日志的插入位置。

  返回值类型：text

  备注：gs_current_xlog_insert_end_location显示当前事务日志的实际插入位置。

- pg_start_backup(label text [, fast boolean])

  描述：开始执行在线备份（需要管理员角色、复制的角色或运维管理员角色打开operation_mode）。以gs_roach开头的label串为保留命名串，只能由内部备份工具GaussRoach使用。

  返回值类型：text

  备注：pg_start_backup接受一个用户定义的备份标签（通常这是备份转储文件存放地点的名称）。这个函数向数据库集群的数据目录写入一个备份标签文件，然后以文本方式返回备份的事务日志起始位置。

  ```
  openGauss=# SELECT pg_start_backup('label_goes_here',true);
   pg_start_backup
  -----------------
   0/3000020
  (1 row)
  ```

- pg_stop_backup()

  描述：完成执行在线备份。需要管理员角色、复制的角色执行或运维管理员角色打开operation_mode。

  返回值类型：text

  备注：pg_stop_backup删除pg_start_backup创建的标签文件，并且在事务日志归档区里创建一个备份历史文件。这个历史文件包含给予pg_start_backup的标签、备份的事务日志起始与终止位置、备份的起始和终止时间。返回值是备份的事务日志终止位置。计算出中止位置后，当前事务日志的插入点将自动前进到下一个事务日志文件，这样，结束的事务日志文件可以被立即归档从而完成备份。

- pg_switch_xlog()

  描述：切换到一个新的事务日志文件。需要管理员角色或运维管理员角色打开operation_mode。

  返回值类型：text

  备注：pg_switch_xlog移动到下一个事务日志文件，以允许将当前日志文件归档（假定使用连续归档）。返回值是刚完成的事务日志文件的事务日志结束位置

+1。如果从最后一次事务日志切换以来没有活动的事务日志，则pg_switch_xlog不进行移动操作，直接返回当前事务日志文件的开始位置。

- pg_xlogfile_name(location text)

  描述：将事务日志的位置字符串转换为文件名。

  返回值类型：text

  备注：pg_xlogfile_name仅抽取事务日志文件名称。如果给定的事务日志位置恰好位于事务日志文件的交界上，这两个函数都返回前一个事务日志文件的名称。这对于管理事务日志归档来说是非常有利的，因为前一个文件是当前最后一个需要归档的文件。

- pg_xlogfile_name_offset(location text)

  描述：将事务日志的位置字符串转换为文件名并返回在文件中的字节偏移量。

  返回值类型：text,integer

  备注：可以使用pg_xlogfile_name_offset从前述函数的返回结果中抽取相应的事务日志文件名称和字节偏移量。例如：

  ```
  openGauss=# SELECT * FROM pg_xlogfile_name_offset(pg_stop_backup());
  NOTICE:  pg_stop_backup cleanup done, waiting for required WAL segments to be archived
  NOTICE:  pg_stop_backup complete, all required WAL segments have been archived
         file_name         | file_offset
  ---------------------------+-------------
  000000010000000000000003 |         272
  (1 row)
  ```

- pg_xlog_location_diff(location text, location text)

  描述：计算两个事务日志位置之间在字节上的区别。

  返回值类型：numeric

- pg_cbm_tracked_location()

  描述：用于查询cbm解析到的lsn位置。

  返回值类型：text

- pg_cbm_get_merged_file(startLSNArg text, endLSNArg text)

  描述：用于将指定lsn范围之内的cbm文件合并成一个cbm文件，并返回合并完的cbm文件名。

  返回值类型：text

  备注：必须是系统管理员或运维管理员才能获取cbm合并文件。

- pg_cbm_get_changed_block(startLSNArg text, endLSNArg text)

  描述：用于将指定lsn范围之内的cbm文件合并成一个表，并返回表的各行记录。

  返回值类型：records

  备注：pg_cbm_get_changed_block返回的表字段包含：合并起始的lsn、合并截止的lsn、表空间oid、库oid、表的relfilenode、表的fork number、表是否被删除、表是否被创建、表是否被截断、表被截断后的页面数、有多少页被修改以及被修改的页号的列表。

- pg_cbm_recycle_file(targetLSNArg text)

  描述：删除不再使用的cbm文件，并返回删除后的第一条lsn。

  返回值类型：text

- pg_cbm_force_track(targetLSNArg text,timeOut int)

  描述：强制执行一次cbm追踪到指定的xLog位置，并返回实际追踪结束点的xLog位置。

返回值类型：text

- pg_enable_delay_ddl_recycle()

  描述：开启延迟DDL功能，并返回开启点的xLog位置。需要管理员角色或运维管理员角色打开operation_mode。

  返回值类型：text

- pg_disable_delay_ddl_recycle(barrierLSNArg text, isForce bool)

  描述：关闭延迟DDL功能，并返回本次延迟DDL生效的xLog范围。需要管理员角色或运维管理员角色打开operation_mode。

  返回值类型：records

- pg_enable_delay_xlog_recycle()

  描述：开启延迟xLog回收功能，cn修复使用。需要管理员角色或运维管理员角色打开operation_mode。

  返回值类型：void

- pg_disable_delay_xlog_recycle()

  描述：关闭延迟xLog回收功能，cn修复使用。需要管理员角色或运维管理员角色打开operation_mode。

  返回值类型：void

- pg_cbm_rotate_file(rotate_lsn text)

  描述：等待cbm解析到rotate_lsn之后，强制切换文件，在build期间调用。

  返回值类型：void。

- gs_roach_stop_backup(backupid text)

  描述：停止一个内部备份工具GaussRoach开启的备份。与pg_stop_backup系统函数类似，但更轻量。

  返回值类型：text，内容为当前日志的插入位置。

- gs_roach_enable_delay_ddl_recycle(backupid name)

  描述：开启延迟DDL功能，并返回开启点的日志位置。与pg_enable_delay_ddl_recycle系统函数类似，但更轻量。并且，通过传入不同的backupid，可以支持并发打开延迟DDL。

  返回值类型：text，内容为返回开启点的日志位置。

- gs_roach_disable_delay_ddl_recycle(backupid text)

  描述：关闭延迟DDL功能，并返回本次延迟DDL生效的日志范围。与pg_enable_delay_ddl_recycle系统函数类似，但更轻量。并且，通过传入不同的backupid，可以支持并发关闭延迟DDL功能。

  返回值类型：records，内容为本次延迟DDL生效的日志范围。

- gs_roach_switch_xlog(request_ckpt bool)

  描述：切换当前使用的日志段文件，并且，如果request_ckpt 为true，则触发一个全量检查点。

  返回值类型：text，内容为切段日志的位置。

- gs_block_dw_io(timeout int, identifier text)

  描述：阻塞双写页面刷盘。

  参数说明：

  - timeout

    阻塞时长。

取值范围：[0, 3600]（秒），0为阻塞时长为0。

– identifier

此次操作的标识。

取值范围：字符串，不支持除大小写字母、数字以及下划线(_)以外的字符。

返回值类型：bool

备注：调用该函数的用户需要具有SYSADMIN权限或具有OPRADMIN权限，运维管理员角色须打开operation_mode。

- gs_is_dw_io_blocked()

  描述：查看当前双写页面刷盘是否被阻塞，如果处于阻塞中则返回true。

  返回值类型：bool

  备注：调用该函数的用户需要具有SYSADMIN权限或具有OPRADMIN权限，运维管理员角色须打开operation_mode。

恢复控制函数

恢复信息函数提供了当前备机状态的信息。这些函数可能在恢复期间或正常运行中执行。

- pg_is_in_recovery()

  描述：如果恢复仍然在进行中则返回true。

  返回值类型：bool

- pg_last_xlog_receive_location()

  描述：获取最后接收事务日志的位置并通过流复制将其同步到磁盘。当流复制正在进行时，事务日志将持续递增。如果恢复已完成，则最后一次获取的WAL记录会被静态保持并在恢复过程中同步到磁盘。如果流复制不可用，或还没有开始，这个函数返回NULL。

  返回值类型：text

- pg_last_xlog_replay_location()

  描述：获取最后一个事务日志在恢复时重放的位置。如果恢复仍在进行，事务日志将持续递增。如果已经完成恢复，则将保持在恢复期间最后接收WAL记录的值。如果未进行恢复但服务器正常启动时，则这个函数返回NULL。

  返回值类型：text

- pg_last_xact_replay_timestamp()

  描述：获取最后一个事务在恢复时重放的时间戳。这是为在主节点上生成事务提交或终止WAL记录的时间。如果在恢复时没有事务重放，则这个函数返回NULL。如果恢复仍在进行，则事务日志将持续递增。如果恢复已经完成，则将保持在恢复期间最后接收WAL记录的值。如果服务器无需恢复就已正常启动，则这个函数返回NULL。

  返回值类型：timestamp with time zone

恢复控制函数控制恢复的进程。这些函数可能只在恢复时被执行。

- pg_is_xlog_replay_paused()

  描述：如果恢复暂停则返回true。

  返回值类型：bool

- pg_xlog_replay_pause()

  描述：立即暂停恢复。

返回值类型：void

- pg_xlog_replay_resume()

  描述：如果恢复处于暂停状态，则重新启动。

  返回值类型：void

- gs_get_active_archiving_standby()

  描述：查询同一分片内归档备机的信息。返回备机名，备机归档位置和已归档日志个数。

  返回值类型：text，text，int

- gs_pitr_get_warning_for_xlog_force_recycle()

  描述：查询开启归档后是否因归档槽不推进日志大量堆积导致日志被回收。

  返回值类型：bool

- gs_pitr_clean_history_global_barriers(stop_barrier_timestamp cstring)

  描述：清理指定时间之前所有barrier记录。返回最老的barrier记录。入参为cstring类型，Linux时间戳。需要管理员角色或运维管理员角色执行。

  返回值类型：text

- gs_pitr_archive_slot_force_advance(stop_barrier_timestamp cstring)

  描述：强制推进归档槽，并清理不需要的barrier记录。返回新的归档槽位置。入参为cstring类型，Linux时间戳。需要管理员角色或运维管理员角色执行。

  返回值类型：text

当恢复暂停时，没有发生数据库更改。如果是在热备里，所有新的查询将看到一致的数据库快照，并且不会有进一步的查询冲突产生，直到恢复继续。

如果不能使用流复制，则暂停状态将无限的延续。当流复制正在进行时，将连续接收WAL记录，最终将填满可用磁盘空间，这个进度取决于暂停的持续时间，WAL生成的速度和可用的磁盘空间。

7.5.23.5 双集群容灾控制函数

双集群容灾控制函数可以创建归档槽，归档槽指定了保存物理日志的obs信息。

- pg_create_physical_replication_slot_extern(slotname text, dummy_standby bool, extra_content text, need_recycle_xlog bool)

  描述：创建OBS/NAS归档槽。slotname 为本次灾备的slotname ，主备必须使用同一个slotname 。dummy_standby为false表示一主多备。extra_content包含了归档槽的一些信息。对于OBS归档槽，其格式为"OBS;obs_server_ip;obs_bucket_name;obs_ak;obs_sk;archive_path;is_recovery;is_vote_replicate"，OBS表示归档槽的归档的介质，obs_server_ip为obs的ip，obs_bucket_name为obs的桶名，obs_ak为obs的ak，obs_sk为obs的sk，archive_path为归档的路径i，is_recovery标志是归档槽还是恢复槽，0表示是归档槽，主要是主集群使用；1表示是恢复槽，主要是灾备集群使用。is_vote_replicate标志是否是投票副本优先，0表示同步备机归档优先，1表示投票副本归档优先，当前版本该字段为预留字段，暂未适配。对于NAS归档槽，其格式为"NAS;archive_path;is_recovery;is_vote_replicate"，相比OBS归档槽，缺少了OBS相关的配置信息，其余字段意义相同。

  如果是不指定OBS或NAS介质的话，默认指定的是OBS归档槽，其extra_content格式为"obs_server_ip;obs_bucket_name;obs_ak;obs_sk;archive_path;is_recovery;is_vote_replicate"。

need_recycle_xlog标志创建归档槽时是否回收老的归档日志，true表示回收，false表示不回收。

返回值类型：records包含本次灾备的slotname和xlog_position。

备注：调用该函数的用户需要具有SYSADMIN权限或具有REPLICATION权限或继承了内置角色gs_role_replication的权限。目前不支持创建多归档槽。

例如：

创建OBS归档槽：

```
openGauss=# select * from pg_create_physical_replication_slot_extern('uuid', false, 'OBS;obs.cn-north-7.ulanqab.huawei.com;dyk;19D772JBCACXX3KWS51D;********;openGauss_uuid/dn1;0;0', false);
 slotname | xlog_position
----------+---------------
 uuid     |
(1 row)
```

创建NAS归档槽：

```
openGauss=# select * from pg_create_physical_replication_slot_extern('uuid', false, 'NAS;/data/nas/media/openGauss_uuid/dn1;0;0', false);
 slotname | xlog_position
----------+---------------
 uuid     |
```

- gs_set_obs_delete_location(delete_location text)

  描述：设置obs归档日志可删除的位置。delete_location实际为Log Sequence Number（LSN），该位置之前的日志在灾备集群已经完成回放并且落盘，可以在obs上进行删除。

  返回值类型：xlog_file_name text，表明此次可删除点所在的日志文件名。无论obs删除是否成功，该值都会正常返回。

  ```
  openGauss=# select gs_set_obs_delete_location('0/54000000');
   gs_set_obs_delete_location
  ----------------------------
   000000010000000000000054_00
  (1 row)
  ```

- gs_hadr_do_switchover()

  描述：异地容灾集群中主集群在执行计划内switchover过程中截断业务的接口。

  返回值类型：bool，表明此次业务截断是否成功，是否可以正常进行switchover流程。

- gs_set_obs_delete_location_with_slotname(cstring, cstring)

  描述：设置某个容灾关系上obs归档日志可删除的位置。第一个参数实际为Log Sequence Number（LSN），该位置之前的日志在灾备数据库实例已经完成回放并且落盘，可以在obs上进行删除，第二个参数为归档槽的名称。

  返回值类型：xlog_file_name text，表明此次可删除点所在的日志文件名。无论obs删除是否成功，该值都会正常返回。

- gs_streaming_dr_in_switchover()

  描述：基于流式复制的异地容灾解决方案中主集群在执行计划内switchover过程中截断业务的接口。

  返回值类型：bool，表明此次业务截断是否成功，是否可以正常进行switchover流程。

7.5.23.6 双集群容灾查询函数

- gs_get_global_barrier_status()

  描述：两地三中心跨Region容灾特性开启后，主集群和灾备集群通过obs进行日志同步，通过barrier日志在主集群的落盘，在灾备集群的回放来确定主集群归档日

志进度与灾备集群日志回放进度。gs_get_global_barrier_status用以查询主集群已在obs完成归档的最新global barrier。

返回值类型：text

global_barrier_id：全局最新barrier ID

global_achive_barrier_id：全局最新归档barrier ID

- gs_get_local_barrier_status()

  描述：两地三中心跨Region容灾特性开启后，主集群和灾备集群通过obs进行日志同步，通过barrier日志在主集群的落盘，在灾备集群的回放来确定主集群归档日志进度与灾备集群日志回放进度。gs_get_local_barrier_status用于查询灾备集群每个节点当前的日志回放情况。

  返回值类型：text

  barrier_id：灾备集群某节点当前回放到的最新barrier ID

  barrier_lsn：灾备集群某节点当前回放到的最新barrier ID的Log Sequence Number（LSN）

  archive_lsn：灾备集群某节点当前已获得归档日志的位置，该参数当前未生效。

  flush_lsn：灾备集群某节点当前已完成刷盘日志位置。

- gs_get_global_barriers_status()

  描述：两地三中心跨Region容灾特性-基于OBS的解决方案开启后，主数据库实例和多个灾备数据库实例通过obs进行日志同步，通过barrier日志在主数据库实例的落盘，在灾备数据库实例的回放来确定主数据库实例归档日志进度与灾备数据库实例日志回放进度。gs_get_global_barriers_status用以查询主数据库实例已在obs完成归档的最新global barrier。

  返回值类型：text

  slot_name：容灾使用的槽位名。

  global_barrier_id：全局最新barrier ID。

  global_achive_barrier_id：全局最新归档barrier ID。

- gs_upload_obs_file('slot_name', 'src_file', 'dest_file')

  描述：两地三中心跨Region容灾特性开启后，主集群上传数据到OBS上的函数。

  返回值类型：void

  slot_name：主集群CN创建的复制槽的名字。

  src_file：主集群CN数据目录下的需要上传的文件的文件位置。

  dest_file：上传到OBS上对应的文件的文件位置。

- gs_download_obs_file('slot_name', 'src_file', 'dest_file')

  描述：两地三中心跨Region容灾特性开启后，灾备集群从OBS上下载数据到本地的函数。

  返回值类型：void

  slot_name：灾备集群CN创建的复制槽的名字。

  src_file：OBS需要下载的文件的文件位置。

  dest_file：灾备集群CN数据目录下需要存放下载文件对应的文件位置。

- gs_get_obs_file_context('file_name', 'slot_name')

  描述：两地三中心跨Region容灾特性开启后，查询OBS上对应文件的内容。

  返回值类型：text

  file_name：OBS上文件的文件名。

slot_name：主/灾备集群CN创建的复制槽的名字。

- gs_set_obs_file_context('file_name', 'file_context','slot_name')

  描述：两地三中心跨Region容灾特性开启后，在OBS上创建文件并写入对应的内容。

  返回值类型：text

  file_name：OBS上文件的文件名。

  file_context： 写入文件的内容。

  slot_name：主/灾备集群CN创建的复制槽的名字。

- gs_get_hadr_key_cn()

  描述：两地三中心跨Region容灾特性开启后，在OBS上创建文件并写入对应的内容。

  返回值类型：text

  file_name：OBS上文件的文件名。

  file_context： 写入文件的内容。

  slot_name：主/灾备集群CN创建的复制槽的名字。

- gs_hadr_has_barrier_creator()

  描述：两地三中心跨Region容灾特性开启后，查询当前cn节点是否存在barrier_creator线程，存在返回true（需要系统管理员角色）。

  返回值类型：Boolean

  备注：该函数只有在容灾集群启动计划内switchover时使用。

- gs_hadr_in_recovery()

  描述：两地三中心跨Region容灾特性开启后，查询当前节点是否处于基于目标barrier的日志恢复中，还在恢复中返回true。只有完成日志恢复，才会启动switchover流程中的灾备集群升为生产集群的步骤，需要系统管理员角色执行。

  返回值类型：Boolean

  说明

  该函数只有在容灾集群启动计划内switchover时使用。

- gs_streaming_dr_get_switchover_barrier()

  描述：两地三中心跨Region容灾-基于流式复制的解决方案中，查询灾备集群参与容灾的CN与首备DN实例是否已接收到switchover barrier日志并完成回放，已完成返回true。灾备集群只有在所有DN实例都完成switchover barrier日志回放，才会启动switchover流程中的灾备数据库实例升为生产数据库实例的步骤（需要系统管理员角色）。

  返回值类型：Boolean

  备注：该函数只有在流式容灾解决方案中容灾数据库实例启动计划内switchover时使用。

- gs_streaming_dr_service_truncation_check()

  描述：两地三中心跨Region容灾-基于流式复制的解决方案中，查询主集群参与容灾的CN与主DN实例是否已完成switchover barrier日志发送，已完成返回true。只有完成日志发送，才会启动switchover流程中的生产数据库实例降为灾备数据库实例的步骤（需要系统管理员角色）。

  返回值类型：Boolean

  备注：该函数只有在容灾数据库实例启动计划内switchover时使用。

- gs_hadr_local_rto_and_rpo_stat()

  描述：显示流式容灾的本地节点数据库实例和灾备数据库实例日志流控信息（如果在没有参与流式容灾的节点执行，如备DN或部分CN节点，则可能返回空）。

  返回值类型：record，具体各个字段的类型和含义如下：

参数	类型	描述
hadr_sender_node_name	text	节点的名称，包含主数据库实例和备数据库实例首备。
hadr_receiver_node_name	text	备数据库实例首备名称。
source_ip	text	主数据库实例主DN IP地址。
source_port	int	主数据库实例主DN通信端口。
dest_ip	text	备数据库实例首备DN IP地址。
dest_port	int	备数据库实例首备DN通信端口。
current_rto	int	流控的信息，当前主备数据库实例的日志rto时间（单位：秒）。
target_rto	int	流控的信息，目标主备数据库实例间的rto时间（单位：秒）。
current_rpo	int	流控的信息，当前主备数据库实例的日志rpo时间（单位：秒）。
target_rpo	int	流控的信息，目标主备数据库实例间的rpo时间（单位：秒）。
rto_sleep_time	int	RTO流控信息，为了达到目标rto，预期主机walsender所需要的睡眠时间（单位：微秒）。
rpo_sleep_time	int	RPO流控信息，为了达到目标rpo，预期主机xlog插入（Insert）所需要的睡眠时间（单位：微秒）。

- gs_hadr_remote_rto_and_rpo_stat()

  描述：显示流式容灾的其他所有分片或CN数据库实例和灾备数据库实例日志流控信息（一般在CN节点执行；如果在DN节点执行，可能返回为空）。

  返回值类型：record，具体各个字段的类型和含义如下：

参数	类型	描述
hadr_sender_node_name	text	节点的名称，包含主数据库实例和备数据库实例首备。

参数	类型	描述
hadr_receiver_node_name	text	备数据库实例首备名称。
source_ip	text	主数据库实例主DN IP地址。
source_port	int	主数据库实例主DN通信端口。
dest_ip	text	备数据库实例首备DN IP地址。
dest_port	int	备数据库实例首备DN通信端口。
current_rto	int	流控的信息，当前主备数据库实例的日志rto时间（单位：秒）。
target_rto	int	流控的信息，目标主备数据库实例间的rto时间（单位：秒）。
current_rpo	int	流控的信息，当前主备数据库实例的日志rpo时间（单位：秒）。
target_rpo	int	流控的信息，目标主备数据库实例间的rpo时间（单位：秒）。
rto_sleep_time	int	RTO流控信息，为了达到目标rto，预期主机walsender所需要的睡眠时间（单位：微秒）。
rpo_sleep_time	int	RPO流控信息，为了达到目标rpo，预期主机xlogInsert所需要的睡眠时间（单位：微秒）。

7.5.23.7 快照同步函数

快照同步函数是导出当前快照的标识符。

- pg_export_snapshot()

  描述：保存当前的快照并返回它的标识符。

  返回值类型：text

  备注：函数pg_export_snapshot保存当前的快照并返回一个文本字符串标识此快照。这个字符串必须传递给想要导入快照的客户端。可用在set transaction snapshot snapshot_id时导入snapshot，但是应用的前提是该事务设置了SERIALIZABLE或REPEATABLE READ隔离级别。而GaussDB目前是不支持这两种隔离级别的。该函数的输出不可用做set transaction snapshot的输入。

- pg_export_snapshot_and_csn()

  描述：保存当前的快照并返回它的标识符。比pg_export_snapshot()多返回一列CSN，表示当前快照的CSN。

  返回值类型：text

7.5.23.8 数据库对象函数

数据库对象尺寸函数

数据库对象尺寸函数计算数据库对象使用的实际磁盘空间。

- pg_column_size(any)

  描述：存储一个指定的数值需要的字节数（可能压缩过）。

  返回值类型：int

  备注：pg_column_size显示用于存储某个独立数据值的空间。

  ```
  openGauss=# SELECT pg_column_size(1);
   pg_column_size
  ----------------
               4
  (1 row)
  ```

- pg_database_size(oid)

  描述：指定OID代表的数据库使用的磁盘空间。

  返回值类型：bigint

- pg_database_size(name)

  描述：指定名称的数据库使用的磁盘空间。

  返回值类型：bigint

  备注：pg_database_size接受一个数据库的OID或者名称，然后返回该对象使用的全部磁盘空间。

  示例：

  ```
  openGauss=# SELECT pg_database_size('testdb');
   pg_database_size
  ------------------
          51590112
  (1 row)
  ```

- pg_relation_size(oid)

  描述：pg_relation_size(..., 'main')的简写，指定OID代表的表或者索引所使用的磁盘空间。

  返回值类型：bigint

- pg_relation_size(relation regclass, fork text)

  描述：指定表或索引的指定分叉树（'main'，'fsm'或'vm'）使用的磁盘空间。

  返回值类型：bigint

- pg_relation_size(relation regclass)

  描述：pg_relation_size(..., 'main')的简写。

  返回值类型：bigint

  备注：pg_relation_size接受一个表、索引的OID或者名称，然后返回它们的字节大小。

- pg_partition_size(oid,oid)

  描述：指定OID代表的分区使用的磁盘空间。其中，第一个oid为表的OID，第二个oid为分区的OID。

  返回值类型：bigint

- pg_partition_size(text, text)

描述：指定名称的分区使用的磁盘空间。其中，第一个text为表名，第二个text为分区名。

返回值类型：bigint

- pg_partition_indexes_size(oid,oid)

  描述：指定OID代表的分区的索引使用的磁盘空间。其中，第一个oid为表的OID，第二个oid为分区的OID。

  返回值类型：bigint

- pg_partition_indexes_size(text,text)

  描述：指定名称的分区的索引使用的磁盘空间。其中，第一个text为表名，第二个text为分区名。

  返回值类型：bigint

- pg_indexes_size(regclass)

  描述：附加到指定表的索引使用的总磁盘空间。

  返回值类型：bigint

- pg_size_pretty(bigint)

  描述：将以64位整数表示的字节值转换为具有单位的易读格式。

  返回值类型：text

- pg_size_pretty(numeric)

  描述：将以数值表示的字节值转换为具有单位的易读格式。

  返回值类型：text

  备注：pg_size_pretty用于把其他函数的结果格式化成一种易读的格式，可以根据情况使用kB 、MB 、GB 、TB。

- pg_table_size(regclass)

  描述：指定的表使用的磁盘空间，不计索引（但是包含TOAST，自由空间映射和可见性映射）。

  返回值类型：bigint

- pg_tablespace_size(oid)

  描述：指定OID代表的表空间使用的磁盘空间。

  返回值类型：bigint

- pg_tablespace_size(name)

  描述：指定名称的表空间使用的磁盘空间。

  返回值类型：bigint

  备注：

  pg_tablespace_size接受一个数据库的OID或者名称，然后返回该对象使用的全部磁盘空间。

- pg_total_relation_size(oid)

  描述：指定OID代表的表使用的磁盘空间，包括索引数据。

  返回值类型：bigint

- pg_total_relation_size(regclass)

  描述：指定的表使用的总磁盘空间，包括所有的索引和TOAST数据。

  返回值类型：bigint

- pg_total_relation_size(text)

  描述：指定名称的表所使用的全部磁盘空间，包括索引数据。表名称可以用模式名修饰。

  返回值类型：bigint

  备注：pg_total_relation_size接受一个表的OID或者名称，然后返回以字节计的数据和所有相关的索引的尺寸。

- datalength(any)

  描述：计算一个指定的数据需要的字节数（不考虑数据的管理空间和数据压缩，数据类型转换等情况）。

  返回值类型：int

  备注：datalength用于计算某个独立数据值的空间。

  示例：

  ```
  openGauss=# SELECT datalength(1);
  datalength
  ------------
  4
  (1 row)
  ```

  目前支持的数据类型及计算方式见下表：

数据类型			存储空间
数值类型	整数类型	TINYINT	1
		SMALLINT	2
		INTEGER	4
		BINARY_INTEGER	4
		BIGINT	8
	任意精度型	DECIMAL	每4位十进制数占两个字节，小数点前后数字分别计算
		NUMERIC	每4位十进制数占两个字节，小数点前后数字分别计算
		NUMBER	每4位十进制数占两个字节，小数点前后数字分别计算
	序列整型	SMALLSERIAL	2
		SERIAL	4
		BIGSERIAL	8
	浮点类型	FLOAT4	4
		DOUBLE PRECISION	8
		FLOAT8	8

		BINARY_DOUBLE	8
		FLOAT[(p)]	每4位十进制数占两个字节，小数点前后数字分别计算
		DEC[(p[,s])]	每4位十进制数占两个字节，小数点前后数字分别计算
		INTEGER[(p[,s])]	每4位十进制数占两个字节，小数点前后数字分别计算
布尔类型	布尔类型	BOOLEAN	1
字符类型	字符类型	CHAR	n
		CHAR(n)	n
		CHARACTER(n)	n
		NCHAR(n)	n
		VARCHAR(n)	n
		CHARACTER	字符实际字节数
		VARYING(n)	字符实际字节数
		VARCHAR2(n)	字符实际字节数
		NVARCHAR2(n)	字符实际字节数
		TEXT	字符实际字节数
		CLOB	字符实际字节数
时间类型	时间类型	DATE	8
		TIME	8
		TIMEZ	12
		TIMESTAMP	8
		TIMESTAMPZ	8
		SMALLDATETIME	8
		INTERVAL DAY TO SECOND	16
		INTERVAL	16
		RELTIME	4
		ABSTIME	4
		TINTERVAL	12

数据库对象位置函数

- pg_relation_filenode(relation regclass)

  描述：指定关系的文件节点数。

  返回值类型：oid

  备注：pg_relation_filenode接受一个表、索引、序列的OID或者名称，并且返回当前分配给它的"filenode"数。文件节点是关系使用的文件名称的基本组件。对大多数表来说，结果和pg_class.relfilenode相同，但对确定的系统目录来说，relfilenode为0而且这个函数必须用来获取正确的值。如果传递一个没有存储的关系，比如一个视图，那么这个函数返回NULL。

- pg_relation_filepath(relation regclass)

  描述：指定关系的文件路径名。

  返回值类型：text

  备注：pg_relation_filepath类似于pg_relation_filenode，但是它返回关系的整个文件路径名（相对于数据库集群的数据目录PGDATA）。

- get_large_table_name(relfile_node text, threshold_size_gb int8)

  描述：根据表的文件编码（relfile_node）查询对应的表大小（单位为GB）是否超过阈值（threshold_size_gb），如果超过则返回模式名和表名（形式为schemaname.tablename）， 否则返回字符串’null’。

  返回值类型：text

- pg_filenode_relation(tablespacename, relname)

  描述：获取到对应的tablespace和relfilenode所对应的表名。

  返回类型：regclass

- pg_partition_filenode(partition_oid)

  描述：获取到指定分区表的oid锁对应的filenode。

  返回类型：oid

- pg_partition_filepath(partition_oid)

  描述：指定分区的文件路径名。

  返回值类型：text

回收站对象函数

- gs_is_recycle_object(classid, objid, objname)

  描述：判断是否为回收站对象。分布式不支持该函数。

  返回值类型：bool

7.5.23.9 咨询锁函数

咨询锁函数用于管理咨询锁（Advisory Lock）。

- pg_advisory_lock(key bigint)

  描述：获取会话级别的排他咨询锁。

  返回值类型：void

  备注：pg_advisory_lock锁定应用程序定义的资源，该资源可以用一个64位或两个不重叠的32位键值标识。如果已经有另外的会话锁定了该资源，则该函数将阻

塞到该资源可用为止。这个锁是排他的。多个锁定请求将会被压入栈中，因此，如果同一个资源被锁定了三次，它必须被解锁三次以将资源释放给其他会话使用。

- pg_advisory_lock(key1 int, key2 int)

  描述：获取会话级别的排他咨询锁。

  返回值类型：void

  备注：只允许sysadmin对键值对(65535, 65535)加会话级别的排他咨询锁，普通用户无权限。

- pg_advisory_lock(lock_id int4, lock_id int4, datebase_name Name)

  描述：通过传入锁ID和数据库名字，获取指定数据库的排他咨询锁。

  返回值类型：void

- pg_advisory_lock_shared(key bigint)

  描述：获取会话级别的共享咨询锁。

  返回值类型：void

- pg_advisory_lock_shared(key1 int, key2 int)

  描述：获取会话级别的共享咨询锁。

  返回值类型：void

  备注：pg_advisory_lock_shared类似于pg_advisory_lock，不同之处仅在于共享锁会话可以和其他请求共享锁的会话共享资源，但排他锁除外。

- pg_advisory_unlock(key bigint)

  描述：释放会话级别的排他咨询锁。

  返回值类型：Boolean

- pg_advisory_unlock(key1 int, key2 int)

  描述：释放会话级别的排他咨询锁。

  返回值类型：Boolean

  备注：pg_advisory_unlock释放先前取得的排他咨询锁。如果释放成功则返回true。如果实际上并未持有指定的锁，将返回false并在服务器中产生一条SQL警告信息。

- pg_advisory_unlock(lock_id int4, lock_id int4, datebase_name Name)

  描述：通过传入锁ID和数据库名字，释放指定数据库上的排他咨询锁。

  返回值类型：Boolean

  备注：如果释放成功则返回true；如果未持有锁，则返回false。

- pg_advisory_unlock_shared(key bigint)

  描述：释放会话级别的共享咨询锁。

  返回值类型：Boolean

- pg_advisory_unlock_shared(key1 int, key2 int)

  描述：释放会话级别的共享咨询锁。

  返回值类型：Boolean

  备注：pg_advisory_unlock_shared类似于pg_advisory_unlock，不同之处在于该函数释放的是共享咨询锁。

- pg_advisory_unlock_all()

  描述：释放当前会话持有的所有咨询锁。

返回值类型：void

备注：pg_advisory_unlock_all将会释放当前会话持有的所有咨询锁，该函数在会话结束的时候被隐含调用，即使客户端异常地断开连接也是一样。

- pg_advisory_xact_lock(key bigint)

  描述：获取事务级别的排他咨询锁。

  返回值类型：void

- pg_advisory_xact_lock(key1 int, key2 int)

  描述：获取事务级别的排他咨询锁。

  返回值类型：void

  备注：pg_advisory_xact_lock类似于pg_advisory_lock，不同之处在于锁是自动在当前事务结束时释放，而且不能被显式的释放。只允许sysadmin对键值对(65535, 65535)加事务级别的排他咨询锁，普通用户无权限。

- pg_advisory_xact_lock_shared(key bigint)

  描述：获取事务级别的共享咨询锁。

  返回值类型：void

- pg_advisory_xact_lock_shared(key1 int, key2 int)

  描述：获取事务级别的共享咨询锁。

  返回值类型：void

  备注：pg_advisory_xact_lock_shared类似于pg_advisory_lock_shared，不同之处在于锁是在当前事务结束时自动释放，而且不能被显式的释放。

- pg_try_advisory_lock(key bigint)

  描述：尝试获取会话级排他咨询锁。

  返回值类型：Boolean

  备注：pg_try_advisory_lock类似于pg_advisory_lock，不同之处在于该函数不会阻塞以等待资源的释放。它要么立即获得锁并返回true，要么返回false表示目前不能锁定。

- pg_try_advisory_lock(key1 int, key2 int)

  描述：尝试获取会话级排他咨询锁。

  返回值类型：Boolean

  备注：只允许sysadmin对键值对(65535, 65535)加会话级别的排他咨询锁，普通用户无权限。

- pg_try_advisory_lock_shared(key bigint)

  描述：尝试获取会话级共享咨询锁。

  返回值类型：Boolean

- pg_try_advisory_lock_shared(key1 int, key2 int)

  描述：尝试获取会话级共享咨询锁。

  返回值类型：Boolean

  备注：pg_try_advisory_lock_shared类似于pg_try_advisory_lock，不同之处在于该函数尝试获得共享锁而不是排他锁。

- pg_try_advisory_xact_lock(key bigint)

  描述：尝试获取事务级别的排他咨询锁。

  返回值类型：Boolean

- pg_try_advisory_xact_lock(key1 int, key2 int)

  描述：尝试获取事务级别的排他咨询锁。

  返回值类型：Boolean

  备注：pg_try_advisory_xact_lock类似于pg_try_advisory_lock，不同之处在于如果得到锁，在当前事务的结束时自动释放，而且不能被显式的释放。只允许sysadmin对键值对(65535, 65535)加事务级别的排他咨询锁，普通用户无权限。

- pg_try_advisory_xact_lock_shared(key bigint)

  描述：尝试获取事务级别的共享咨询锁。

  返回值类型：Boolean

- pg_try_advisory_xact_lock_shared(key1 int, key2 int)

  描述：尝试获取事务级别的共享咨询锁。

  返回值类型：Boolean

  备注：pg_try_advisory_xact_lock_shared类似于pg_try_advisory_lock_shared，不同之处在于如果得到锁，在当前事务结束时自动释放，而且不能被显式的释放。

- lock_cluster_ddl()

  描述：尝试对集群内所有存活的CN节点获取会话级别的排他咨询锁。

  返回值类型：Boolean

  备注：只允许sysadmin调用，普通用户无权限。

- unlock_cluster_ddl()

  描述：尝试对CN节点会话级别的排他咨询锁。

  返回值类型：Boolean

7.5.23.10 逻辑复制函数

- pg_create_logical_replication_slot('slot_name', 'plugin_name')

  描述：创建逻辑复制槽。

  参数说明：

  - slot_name

    流复制槽名称。

    取值范围：字符串，仅支持小写字母、数字以及_?-.字符，且不支持“.”或“..”单独作为复制槽名称。

  - plugin_name

    插件名称。

    取值范围：字符串，当前只支持“mppdb_decoding”。

  返回值类型：name, text

  备注：第一个返回值表示slot_name，第二个返回值表示该逻辑复制槽解码的起始LSN位置。调用该函数的用户需要具有SYSADMIN权限或具有REPLICATION权限或继承了内置角色gs_role_replication的权限。

- pg_create_physical_replication_slot('slot_name', 'isDummyStandby')

  描述：创建新的物理复制槽。

  参数说明：

  - slot_name

流复制槽名称。

取值范围：字符串，仅支持小写字母、数字以及_?-.字符，且不支持“.”或“..”单独作为复制槽名称。

- isDummyStandby

返回值类型：name，text

备注：调用该函数的用户需要具有SYSADMIN权限或具有REPLICATION权限或继承了内置角色gs_role_replication的权限。

- pg_drop_replication_slot('slot_name')

  描述：删除流复制槽。

  参数说明：

  - slot_name

    流复制槽名称。

    取值范围：字符串，仅支持小写字母、数字以及_?-.字符，且不支持“.”或“..”单独作为复制槽名称。

  返回值类型：void

  备注：调用该函数的用户需要具有SYSADMIN权限或具有REPLICATION权限或继承了内置角色gs_role_replication的权限。

- pg_logical_slot_peek_changes('slot_name', 'LSN', upto_nchanges, 'options_name', 'options_value')

  描述：解码并不推进流复制槽（下次解码可以再次获取本次解出的数据）。

  参数说明：

  - slot_name

    流复制槽名称。

    取值范围：字符串，仅支持小写字母、数字以及_?-.字符，且不支持“.”或“..”单独作为复制槽名称。

  - LSN

    日志的LSN，表示只解码小于等于此LSN的日志。

    取值范围：字符串（LSN，格式为xlogid/xrecoff），如'1/2AAFC60'。为NULL时表示不对解码截止的日志位置做限制。

  - upto_nchanges

    解码条数（包含begin和commit）。假设一共有三条事务，分别包含3、5、7条记录，如果upto_nchanges为4，那么会解码出前两个事务共8条记录。解码完第二条事务时发现解码条数记录大于等于upto_nchanges，会停止解码。

    取值范围：非负整数。

    说明

    LSN和upto_nchanges中任一参数达到限制，解码都会结束。

  - options：此项为可选参数，由一系列options_name和options_value一一对应组成。

    - include-xids

      解码出的data列是否包含xid信息。

      取值范围：0或1，默认值为1。

- 0：设为0时，解码出的data列不包含xid信息。
- 1：设为1时，解码出的data列包含xid信息。

- skip-empty-xacts

  解码时是否忽略空事务信息。

  取值范围：0或1，默认值为0。
  - 0：设为0时，解码时不忽略空事务信息。
  - 1：设为1时，解码时会忽略空事务信息。

- include-timestamp

  解码信息是否包含commit时间戳。

  取值范围：0或1，默认值为0。
  - 0：设为0时，解码信息不包含commit时间戳。
  - 1：设为1时，解码信息包含commit时间戳。

- only-local

  是否仅解码本地日志。

  取值范围：0或1，默认值为1。
  - 0：设为0时，解码非本地日志和本地日志。
  - 1：设为1时，仅解码本地日志。

- force-binary

  是否以二进制格式输出解码结果

  取值范围：0或1，默认值为0。
  - 0：设为0时，以二进制格式输出解码结果。

- white-table-list

  白名单参数，包含需要进行解码的schema和表名。

  取值范围：包含白名单中表名的字符串，不同的表以','为分隔符进行隔离；使用'*'来模糊匹配所有情况；schema名和表名间以'.'分隔，不允许存在任意空白符。例：

  ```
  select * from pg_logical_slot_peek_changes('slot1', NULL, 4096, 'white-table-list',
  'public.t1,public.t2');
  ```

- max-txn-in-memory

  内存管控参数，单位为MB，单个事务占用内存大于该值即进行落盘。

  取值范围：0~100的整型，默认值为0，即不开启此种管控。

- max-reorderbuffer-in-memory

  内存管控参数，单位为GB，拼接-发送线程中正在拼接的事务总内存（包含缓存）大于该值则对当前解码事务进行落盘。

  取值范围：0~100的整型，默认值为0，即不开启此种管控。

返回值类型：text, xid, text

备注：函数返回解码结果，每一条解码结果包含三列，对应上述返回值类型，分别表示LSN位置、xid和解码内容。调用该函数的用户需要具有SYSADMIN权限或具有REPLICATION权限或继承了内置角色gs_role_replication的权限。

- pg_logical_slot_get_changes('slot_name', 'LSN', upto_nchanges, 'options_name', 'options_value')

  描述：解码并推进流复制槽。

  参数说明： 与pg_logical_slot_peek_changes一致，详细内容请参见**•pg_logical_slot_peek_ch...**。

  备注：调用该函数的用户需要具有SYSADMIN权限或具有REPLICATION权限或继承了内置角色gs_role_replication的权限。

- pg_logical_slot_peek_binary_changes('slot_name', 'LSN', upto_nchanges, 'options_name', 'options_value')

  描述：以二进制格式解码且不推进流复制槽（下次解码可以再次获取本次解出的数据）。

  参数说明：

  – slot_name

    流复制槽名称。

    取值范围：字符串，仅支持小写字母、数字以及_?-.字符，且不支持“.”或“..”单独作为复制槽名称。

  – LSN

    日志的LSN，表示只解码小于等于此LSN的日志。

    取值范围：字符串（LSN，格式为xlogid/xrecoff），如'1/2AAFC60'。为NULL时表示不对解码截止的日志位置做限制。

  – upto_nchanges

    解码条数（包含begin和commit）。假设一共有三条事务，分别包含3、5、7条记录，如果upto_nchanges为4，那么会解码出前两个事务共8条记录。解码完第二条事务时发现解码条数记录大于等于upto_nchanges，会停止解码。

    取值范围：非负整数。

    说明

    LSN和upto_nchanges中任一参数达到限制，解码都会结束。

  – options：此项为可选参数，由一系列options_name和options_value一一对应组成。

    - include-xids

      解码出的data列是否包含xid信息。

      取值范围：0或1，默认值为1。

      - 0：设为0时，解码出的data列不包含xid信息。
      - 1：设为1时，解码出的data列包含xid信息。

    - skip-empty-xacts

      解码时是否忽略空事务信息。

      取值范围：0或1，默认值为0。

      - 0：设为0时，解码时不忽略空事务信息。
      - 1：设为1时，解码时会忽略空事务信息。

    - include-timestamp

      解码信息是否包含commit时间戳。

取值范围：0或1，默认值为0。

- 0：设为0时，解码信息不包含commit时间戳。
- 1：设为1时，解码信息包含commit时间戳。

- only-local

  是否仅解码本地日志。

  取值范围：0或1，默认值为1。

  - 0：设为0时，解码非本地日志和本地日志。
  - 1：设为1时，仅解码本地日志。

- force-binary

  是否以二进制格式输出解码结果。

  取值范围：0或1，默认值为0，均以二进制格式输出结果。

- white-table-list

  白名单参数，包含需要进行解码的schema和表名。

  取值范围：包含白名单中表名的字符串，不同的表以','为分隔符进行隔离；使用'*'来模糊匹配所有情况；schema名和表名间以'.'分隔，不允许存在任意空白符。例：

  ```
  select * from pg_logical_slot_peek_binary_changes('slot1', NULL, 4096, 'white-table-list', 'public.t1,public.t2');
  ```

返回值类型：text, xid, bytea

备注：函数返回解码结果，每一条解码结果包含三列，对应上述返回值类型，分别表示LSN位置、xid和二进制格式的解码内容。调用该函数的用户需要具有SYSADMIN权限或具有REPLICATION权限或继承了内置角色gs_role_replication的权限。

- pg_logical_slot_get_binary_changes('slot_name', 'LSN', upto_nchanges, 'options_name', 'options_value')

  描述：以二进制格式解码并推进流复制槽。

  参数说明：与pg_logical_slot_peek_binary_changes一致，详细内容请参见**•pg_logical_slot_peek_bi...**。

  备注：调用该函数的用户需要具有SYSADMIN权限或具有REPLICATION权限或继承了内置角色gs_role_replication的权限。

- pg_replication_slot_advance ('slot_name', 'LSN')

  描述：直接推进流复制槽到指定LSN，不输出解码结果。

  参数说明：

  – slot_name

    流复制槽名称。

    取值范围：字符串，不支持除小写字母，数字,以及_?-.以外的字符，且不支持'.'或'..'单独作为复制槽名称。

  – LSN

    推进到的日志LSN位置，下次解码时只会输出提交位置比该LSN大的事务结果。如果输入的LSN比当前流复制槽记录的推进位置还要小，则报错；如果输入的LSN比当前最新物理日志LSN还要大，则推进到当前最新物理日志LSN。

    取值范围：字符串（LSN，格式为xlogid/xrecoff）。

返回值类型：name, text

备注：返回值分别对应slot_name和实际推进至的LSN。调用该函数的用户需要具有SYSADMIN权限或具有REPLICATION权限或继承了内置角色gs_role_replication的权限。

- pg_get_replication_slots()

  描述：获取复制槽列表。

  示例：

```
openGauss=# select * from pg_get_replication_slots();
 slot_name |     plugin     | slot_type | datoid | active | xmin | catalog_xmin | restart_lsn |
dummy_standby | confirmed_flush
-----------+----------------+-----------+--------+--------+------+--------------+-------------+---------------
+-----------------
 dn_s1     |                | physical  |      0 | t      |      |              | 0/23DB14E0  | f             |
 slot1     | mppdb_decoding | logical   |  16304 | f      |      |        60966 | 0/1AFA1BB0  | f             |
0/23DA5700
(2 rows)
```

  返回值: text, text, text, oid, boolean, xid, xid, text, boolean, text

- pg_logical_get_area_changes('LSN_start', 'LSN_end', upto_nchanges, 'decoding_plugin', 'xlog_path', 'options_name', 'options_value')

  描述：没有ddl的前提下，指定lsn区间进行解码，或者指定xLog文件进行解码。

  约束条件如下：

  1. 调用接口时，日志级别wal_level=logical，且只有在wal_level=logical期间产生的日志文件才能被解析，如果使用的xLog文件为非logical级别，则解码内容没有对应的值和类型，无其他影响。

  2. xLog文件只能被完全同构的dn的某个副本解析，确保可以找到数据对应的元信息，且没有DDL操作和VACUUM FULL。

  3. 用户可以找到需要解析的xLog。

  4. 用户需要注意一次不要读入过多xLog文件，推荐一次一个，一个xLog文件估测占用内存为xLog文件大小的2~3倍。

  5. 无法解码扩容前的xLog文件。

  备注：调用该函数的用户需要具有SYSADMIN权限或运维模式打开下的运维管理员的权限。

  参数说明：

  - LSN_start

    指定开始解码的lsn。

    取值范围：字符串（LSN，格式为xlogid/xrecoff），如'1/2AAFC60'。为NULL时表示不对解码起始的日志位置做限制。

  - LSN_end

    指定解码结束的lsn。

    取值范围：字符串（LSN，格式为xlogid/xrecoff），如'1/2AAFC60'。为NULL时表示不对解码截止的日志位置做限制。

  - upto_nchanges

    解码条数（包含begin和commit）。假设一共有三条事务，分别包含3、5、7条记录，如果upto_nchanges为4，那么会解码出前两个事务共8条记录。解码完第二条事务时发现解码条数记录大于等于upto_nchanges，会停止解码。

    取值范围：非负整数。

说明

LSN和upto_nchanges中任一参数达到限制，解码都会结束。

– decoding_plugin

解码插件，指定解码内容输出格式的so插件

取值范围：提供mppdb_decoding和sql_decoding两个解码插件。

– xlog_path

解码插件，指定解码文件的xLog绝对路径，文件级别

取值范围：NULL 或者 xLog文件绝对路径的字符串。

– options：此项为可选参数，由一系列options_name和options_value一一对应组成，可以缺省，详见**pg_logical_slot_peek_cha...**。

示例：

```
openGauss=# SELECT pg_current_xlog_location();
 pg_current_xlog_location
--------------------------
 0/E62E238
(1 row)

openGauss=# create table t1 (a int primary key,b int,c int);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "t1_pkey" for table "t1"
CREATE TABLE
openGauss=# insert into t1 values(1,1,1);
INSERT 0 1
openGauss=# insert into t1 values(2,2,2);
INSERT 0 1

openGauss=# select data from pg_logical_get_area_changes('0/
E62E238',NULL,NULL,'sql_decoding',NULL);
 location  |  xid  |                      data
-----------+-------+-------------------------------------------------
 0/E62E8D0 | 27213 | COMMIT (at 2022-01-26 15:08:03.349057+08) 3020226
 0/E6325F0 | 27214 | COMMIT (at 2022-01-26 15:08:07.309869+08) 3020234
.....
```

- gs_get_parallel_decode_status()

  描述：监控各个解码线程的读取日志队列和解码结果队列的长度，以便定位并行解码性能瓶颈。

  返回值类型：text, int, text, text, text, int64, int64

  示例：

```
openGauss=# select * from gs_get_parallel_decode_status();
 slot_name | parallel_decode_num | read_change_queue_length  | decode_change_queue_length |
reader_lsn | working_txn_cnt | working_txn_memory
-----------+---------------------+---------------------------+----------------------------+-----------
+-----------------+--------------------
 slot1     |                   2 | queue0: 1005, queue1: 320 | queue0: 63, queue1: 748    | 0/1DCE2578
|            42 |         192927504
(1 row)
```

  备注：返回值的slot_name代表复制槽名，parallel_decode_num代表该复制槽的并行解码线程数，read_change_queue_length列出了每个解码线程读取日志队列的当前长度，decode_change_queue_length列出了每个解码线程解码结果队列的当前长度，reader_lsn表示当前reader线程读取的日志位置，working_txn_cnt表示当前拼接-发送线程中正在拼接的事务个数，working_txn_memory代表拼接-发送线程中拼接事务占用总内存（单位字节）。

- pg_replication_origin_create (node_name)

  描述：用给定的外部名称创建一个复制源，并且返回分配给它的内部ID。

  备注：调用该函数的用户需要具有SYSADMIN权限。

参数说明：

– node_name

待创建的复制源的名称。

取值范围：字符串，不支持除字母、数字以及（_?-.）以外的字符。

返回值类型：oid

- pg_replication_origin_drop (node_name)

  描述：删除一个以前创建的复制源，包括任何相关的重放进度。

  备注：调用该函数的用户需要具有SYSADMIN权限。

  参数说明：

  – node_name

  待删除的复制源的名称。

  取值范围：字符串，不支持除字母、数字以及（_?-.）以外的字符。

- pg_replication_origin_oid (node_name)

  描述：根据名称查找复制源并返回内部ID。如果没有发现这样的复制源，则抛出错误。

  备注：调用该函数的用户需要具有SYSADMIN权限。

  参数说明：

  – node_name

  要查找的复制源的名称

  取值范围：字符串，不支持除字母、数字以及（_?-.）以外的字符。

  返回值类型：oid

- pg_replication_origin_session_setup (node_name)

  描述：将当前会话标记为从给定的原点回放，从而允许跟踪回放进度。只能在当前没有选择原点时使用。使用pg_replication_origin_session_reset 命令来撤销。

  备注：调用该函数的用户需要具有SYSADMIN权限。

  参数说明：

  – node_name

  复制源名称。

  取值范围：字符串，不支持除字母、数字以及（_?-.）以外的字符。

- pg_replication_origin_session_reset ()

  描述：取消pg_replication_origin_session_setup()的效果。

  备注：调用该函数的用户需要具有SYSADMIN权限。

- pg_replication_origin_session_is_setup ()

  描述：如果在当前会话中选择了复制源则返回真。

  备注：调用该函数的用户需要具有SYSADMIN权限。

  返回值类型：boolean

- pg_replication_origin_session_progress (flush)

  描述：返回当前会话中选择的复制源的重放位置。

  备注：调用该函数的用户需要具有SYSADMIN权限。

  参数说明：

- flush
  决定对应的本地事务是否被确保已经刷入磁盘。
  取值范围：boolean

返回值类型：LSN

- pg_replication_origin_xact_setup (origin_lsn, origin_timestamp)
  描述：将当前事务标记为重放在给定LSN和时间戳上提交的事务。只能在使用pg_replication_origin_session_setup选择复制源时调用。
  备注：调用该函数的用户需要具有SYSADMIN权限。
  参数说明：
  - origin_lsn
    复制源回放位置。
    取值范围：LSN
  - origin_timestamp
    事务提交时间。
    取值范围：timestamp with time zone
- pg_replication_origin_xact_reset ()
  描述：取消pg_replication_origin_xact_setup()的效果。
  备注：调用该函数的用户需要具有SYSADMIN权限。
- pg_replication_origin_advance (node_name, lsn)
  描述：
  将给定节点的复制进度设置为给定的位置。这主要用于设置初始位置，或在配置更改或类似的变更后设置新位置。
  注意：这个函数的使用不当可能会导致不一致的复制数据。
  备注：调用该函数的用户需要具有SYSADMIN权限。
  参数说明：
  - node_name
    已有复制源名称。
    取值范围：字符串，不支持除字母、数字以及（_?-.）以外的字符。
  - lsn
    复制源回放位置。
    取值范围：LSN
- pg_replication_origin_progress (node_name, flush)
  描述：返回给定复制源的重放位置。
  备注：调用该函数的用户需要具有SYSADMIN权限。
  参数说明：
  - node_name
    复制源名称。
    取值范围：字符串，不支持除字母、数字以及（_?-.）以外的字符。
  - flush
    决定对应的本地事务是否被确保已经刷入磁盘。
    取值范围：boolean

- pg_show_replication_origin_status()
  描述：获取复制源的复制状态。
  备注：调用该函数的用户需要具有SYSADMIN权限。
  返回值类型：
  - local_id：oid，复制源id。
  - external_id：text，复制源名称。
  - remote_lsn：LSN，复制源的lsn位置。
  - local_lsn：LSN，本地的lsn位置。

7.5.23.11 段页式存储函数

- local_segment_space_info(tablespacename TEXT, databasename TEXT)
  描述：输出为当前节点，该表空间下所有ExtentGroup的使用信息。
  返回值类型：

node_name	节点名称
extent_size	该ExtentGroup的extent规格，单位是block数。
forknum	Fork号
total_blocks	物理文件总extent数目。
meta_data_blocks	表空间管理的metadata占用的block数，只包括space header，map page等，不包括segment head。
used_data_blocks	存数据占用的extent数目。包括segment head。
utilization	使用的block数占总block数的百分比。即(used_data_blocks +meta_data_block)/total_blocks。
high_water_mark	高水位线，被分配出去的extent，最大的物理页号。超过高水位线的block都没有被使用，可以被直接回收。

  例如：

```
select * from local_segment_space_info('pg_default', 'testdb');
     node_name      | extent_size | forknum | total_blocks | meta_data_blocks | used_data_blocks |
utilization | high_water_mark
--------------------+-------------+---------+--------------+------------------+------------------+-------------
+-----------------
 dn_6001_6002_6003 |           1 |       0 |        16384 |             4157 |                1 |     .253784 |
4158
 dn_6001_6002_6003 |           8 |       0 |        16384 |             4157 |                8 |     .254211 |
4165
(2 rows)
```

- global_segment_space_info(tablespacename TEXT, databasename TEXT)
  描述：效果跟local_segment_space_info类似，返回的是整个集群中所有节点上的使用信息。
- pg_stat_segment_extent_usage(int4 tablespace oid, int4 database oid, int4 extent_type, int4 forknum)
  描述：每次返回一个ExtentGroup中，每个被分配出去的extent的使用情况。extent_type表示ExtentGroup的类型，合理取值为[1,5]的int值。在此范围外的会

报error。forknum 表示fork号，合法取值为[0,4]的int值，目前只有三种值有效，数据文件为0，FSM文件为1，visibility map文件为2。

返回值类型：

名称	描述
start_block	Extent的起始物理页号。
extent_size	Extent的大小。
usage_type	Extent的使用类型，比如segment head，data extent等。
ower_location	有指针指向该extent的对象的位置。比如data extent的owner就是它所属的segment的head位置。
special_data	该extent在它owner中的位置。该字段的数据跟使用类型有关。比如data extent的special data就是它在所属segment中的extent id。

其中，usage_type为枚举类型，每一项的含义为：

- Non-bucket table segment head ： 非hashbucket表的数据段头。
- Non-bucket table fork head：非段页式表的fork段头。
- Bucket table main head：hashbucket表的主表段头。
- Bucket table map block：hashbucket表的MapBlock。
- Bucket segment head：hashbucket表每个bucket的段头。
- Data extent：数据块。

例如：

```
select * from pg_stat_segment_extent_usage((select oid::int4 from pg_tablespace where
spcname='pg_default'), (select oid::int4 from pg_database where datname='testdb'), 1, 0);
start_block | extent_size |       usage_type       | ower_location | special_data
-------------+-------------+------------------------+---------------+--------------
4157 |           1 | Bucket table main head |    4294967295 |            0
4158 |           1 | Bucket table map block |          4157 |            0
4159 |           1 | Bucket table map block |          4157 |            1
4160 |           1 | Bucket table map block |          4157 |            2
4161 |           1 | Bucket table map block |          4157 |            3
4162 |           1 | Bucket table map block |          4157 |            4
4163 |           1 | Bucket table map block |          4157 |            5
4164 |           1 | Bucket table map block |          4157 |            6
4165 |           1 | Bucket table map block |          4157 |            7
4166 |           1 | Bucket table map block |          4157 |            8
```

- local_space_shrink(tablespacename TEXT, databasename TEXT)

  描述：当前节点上对指定段页式空间做物理空间收缩。注意，目前只支持对当前连接的database做shrink。

  返回值：空

- gs_space_shrink(int4 tablespace, int4 database, int4 extent_type, int4 forknum)

  描述：效果跟local_space_shrink类似，对指定段页式空间做物理空间收缩，但参数不同，传入的是tablespace和database的oid，extent_type为[2,5]的int值。注

意：extent_type = 1表示段页式元数据，目前不支持对元数据所在的物理文件做收缩。该函数仅限工具使用，不建议用户直接使用。

返回值：空

- global_space_shrink(tablespacename TEXT, databasename TEXT)

  描述：在cn上执行，对整个集群上所有dn执行段页式存储空间压缩。

  注意：global_space_shrink 锁cluster，在此期间不能执行DDL操作。而local_space_shrink不会锁集群。

- pg_stat_remain_segment_info()

  描述：展示在当前节点上，因为故障等原因，残留的extent。残留extent主要分为两类：分配而未被利用的segment和分配出去而未被利用的extent。两者主要区别在于segment会包含多个extent，回收时，要将segment上的extent一并全部回收。

  返回值类型：

名称	描述
space_id	表空间ID
db_id	数据库ID
block_id	Extent的ID
type	Extent的类型，当前有三种：ALLOC_SEGMENT|DROP_SEGMENT|SHRINK_EXTENT

  其中type的三种类型分别表示：

  - ALLOC_SEGMENT:用户创建一张段页式表，当segment刚被分配，但是建表语句所在事务仍未提交时，节点故障，导致该segment被分配后，没有被使用。
  - DROP_SEGMENT:用户删除段页式表，当该事务成功提交，但是此表的segment页面对应的bit位未被重置，就发生掉电等故障，造成该segment未被使用，也未被释放。
  - SHRINK_EXTENT:用户对段页式表执行shrink操作，在未对空置出的extent进行释放时，发生掉电等故障，造成该extent残留，无法被重新利用。

  例如：

```
select * from pg_stat_remain_segment_info();
space_id | db_id | block_id | type
----------+-------+----------+------
1663     |   16385|        4156| ALLOC_SEGMENT
```

- pg_free_remain_segment(int4 spaceId, int4 dbId, int4 segmentId)

  描述：释放指定的残留extent。参数取值必须为从函数pg_stat_remain_segment_info中查询获取。函数会对传入值校验，如果指定extent不在记录的残留extent中，将返回错误信息。指定的extent如果为单个extent，则只将其独自释放；如果为一个segment，则会将此segment以及此segment上记录的所有extent释放。

  返回值：空

7.5.23.12 其它函数

- pgxc_pool_check()

  描述：检查连接池中缓存的连接数据是否与pgxc_node一致。

  返回值类型：Boolean

- pgxc_pool_reload()

  描述：更新连接池中缓存的连接信息。

  返回值类型：Boolean

- reload_active_coordinator()

  描述：对所有存活的CN，更新连接池中缓存的连接信息。

  返回值类型：void

- pgxc_lock_for_backup()

  描述：为备份操作给集群加锁，这些备份是为在新增节点上做恢复。

  返回值类型：Boolean

  说明

  pgxc_lock_for_backup是在使用gs_dump或gs_dumpall工具备份集群前，用来给集群加锁的。当给集群加锁后，不允许有改变系统结构的操作。该函数不影响DML语句。

- pg_pool_validate(clear bool, node_name text)

  描述：显示CN到节点node_name之间pooler中无效连接，当clear为true时清理无效连接。

  返回值类型：record

- pgxc_pool_connection_status()

  描述：检查pooler连接状态是否正常。

  返回值类型：boolean

- pg_nodes_memory()

  描述：查看所有节点的内存占用。

  返回值类型：record

- table_skewness(text)

  描述：查看表数据在所有节点的占比。

  参数：表示待查询表的表名，为text类型。

  返回值类型：record

- table_skewness(text, text, text)

  描述：查看表数据指定列在所有节点的占比。

  参数：表示待查询表的表名、指定列名、指定的表的记录数（默认值为0，查询所有记录），都为text类型。

  返回值类型：record

  返回值说明：节点编号，指定列的数据行数，当前节点数据量相对总数据量的占比。

  示例：

```
返回't'表'a'字段前5行数据在节点上的分布。
openGauss=# select table_skewness('t', 'a',5);
 table_skewness
----------------
```



```
 (1,3,60.000%)
 (2,2,40.000%)
(2 rows)

返回't'表'a'字段所有数据在节点上的分布。
openGauss=# select table_skewness('t', 'a');
 table_skewness
----------------
 (1,7,70.000%)
 (2,2,20.000%)
 (0,1,10.000%)
(3 rows)
```

- table_skewness_with_schema(text, text)

  描述：查看表数据在所有节点的占比，与table_skewness(text)作用相同。

  参数：表示待查询表的schema名称和表名，为text类型。

  返回值类型：record

- table_data_skewness(colrecord, type)

  描述：查看表数据所在节点。

  参数说明：

  colrecord：表示待查询表的列名记录，为record类型。

  type：hash分布类型

  返回值类型：smallint

  示例：

```
openGauss=# select table_data_skewness(row(index), 'R') from test1;
 table_data_skewness
---------------------
                   4
                   3
                   1
                   2
(4 rows)
```

- table_distribution(schemaname text, tablename text)

  描述：查看指定表在各个节点上占用的存储空间。

  参数：表示待查询表的模式名和表名，均为text类型。

  返回值类型：record

  说明

  - 使用本函数查询指定表存储分布信息，需要具备指定表的SELECT权限。
  - table_distribution性能比table_skewness更优，尤其是在大数据量场景下，请优先考虑使用table_distribution函数。
  - 当使用table_distribution并希望直观的看到空间占比时，可使用dnsize/(sum(dnsize) over ())的方式查看出具体的占比情况。

- table_distribution()

  描述：查看当前库中所有表在各节点的存储空间分布情况。

  返回值类型：record

  说明

  - 使用本函数涉及全库表信息查询，需要具备管理员权限。
  - 当前基于table_distribution()函数，GaussDB提供视图**PGXC_GET_TABLE_SKEWNESS**进行数据倾斜查询，建议在数据库中表数量（小于10000）较少的场景直接使用。

- plan_seed

  描述：获取前一次查询语句的seed值（内部使用）。

  返回值类型：int

- pg_stat_get_env

  描述：获取当前节点的环境变量信息，仅sysadmin和monitor admin可以访问。

  返回值类型：record

  示例：

```
openGauss=# select pg_stat_get_env();
                                                                    pg_stat_get_env
-------------------------------------------------------------------------------------------------------------------------------------------------
 (coordinator1,localhost,144773,49100,/data1/GaussDB_Kernel_TRUNK/install,/data1/
GaussDB_Kernel_TRUNK/install/data/coordinator1,pg_log)
(1 row)
```

- pg_catalog.plancache_clean()

  描述：清理节点上无人使用的全局计划缓存。

  返回值类型：bool

- pg_stat_get_thread

  描述：提供当前节点下线程的状态信息，sysadmin和monitor admin用户可以查看所有线程的信息，普通用户只能查看本用户的线程信息。

  返回值类型：record

- pgxc_get_os_threads

  描述：提供整个集群中所有正常节点下的线程状态信息。

  返回值类型：record

- pg_stat_get_sql_count

  描述：提供当前节点中用户执行的SELECT/UPDATE/INSERT/DELETE/MERGE INTO语句的计数结果，sysadmin和monitor admin用户可以查看所有用户的信息，普通用户只能查看本用户的统计信息。

  返回值类型：record

- pgxc_get_sql_count

  描述：提供整个集群所有节点中所有用户执行的SELECT/UPDATE/INSERT/DELETE/MERGE INTO语句的计数结果。

  返回值类型：record

- pgxc_get_node_env

  描述：提供获取集群中所有节点的环境变量信息。

  返回值类型：record

- pgxc_disaster_read_set(text)

  描述：设置灾备集群的节点信息到ETCD上。仅灾备集群可用，仅初始用户可调用。

  返回值类型：Boolean

- pgxc_disaster_read_init

  描述：初始化灾备可读的资源和状态信息。仅灾备集群可用，仅初始用户可调用。

  返回值类型：Boolean

- pgxc_disaster_read_clear

  描述：清理灾备可读的资源和状态信息。仅灾备集群可用，仅初始用户可调用。

  返回值类型：Boolean

- pgxc_disaster_read_status

  描述：提供灾备集群的节点信息，仅灾备集群可用。

  返回值类型：record

- gs_switch_relfilenode

  描述：交换两个表或分区的元信息（重分布工具内部使用，用户直接使用会有错误信息提示）。

  返回值类型：int

- pg_catalog.plancache_clean()

  描述：清理当前节点上无人使用的全局计划缓存。

  返回值类型：boolean

- DBE_PERF.global_plancache_clean()

  描述：清理所有节点上无人使用的全局计划缓存。

  返回值类型：Boolean

- copy_error_log_create()

  描述：创建COPY FROM容错机制所需要的错误表（public.pgxc_copy_error_log）。

  返回值类型：Boolean

  说明

  - 此函数会尝试创建public.pgxc_copy_error_log表，表的详细信息请参见**表7-53**。
  - 在relname列上创建B-tree索引，并REVOKE ALL on public.pgxc_copy_error_log FROM public对错误表进行权限控制（与COPY语句权限一致）。
  - 由于尝试创建的public.pgxc_copy_error_log定义是一张行存表，因此集群上必须支持行存表的创建才能够正常运行此函数，并使用后续的COPY容错功能。需要特别注意的是，enable_hadoop_env这个GUC参数开启后会禁止在集群内创建行存表（GaussDB默认为off）。
  - 此函数自身权限为Sysadmin及以上（与错误表、COPY权限一致）。
  - 若创建前public.pgxc_copy_error_log表已存在或者copy_error_log_relname_idx索引已存在，则此函数会报错回滚。

表 7-53 错误表 public.pgxc_copy_error_log 信息

列名称	类型	描述
relname	character varying	表名称。以模式名.表名形式显示。
begintime	timestamp with time zone	出现数据格式错误的时间。
filename	character varying	出现数据格式错误的数据源文件名。
lineno	bigint	在数据源文件中，出现数据格式错误的行号。

列名称	类型	描述
rawrecord	text	在数据源文件中，出现数据格式错误的原始记录。
detail	text	详细错误信息。

- pg_stat_get_data_senders()

  描述：提供当前活跃的数据复制发送线程的详细信息。

  返回值类型：record

- textlen()

  描述：提供查询text的逻辑长度的方法。

  返回值类型：int

- threadpool_status()

  描述：显示线程池中工作线程及会话的状态信息。

  返回值类型：record

- get_local_active_session()

  描述：提供当前节点保存在内存中的历史活跃session状态的采样记录，sysadmin和monitor admin权限能查看当前节点所有的历史活跃session记录，普通用户查看本会话的历史活跃session记录。

  返回值类型：record

- dbe_perf.get_global_active_session()

  描述：提供所有节点保存在内存中的历史活跃session状态的采样记录。

  返回值类型：record

- dbe_perf.get_global_gs_asp(timestamp,timestamp)

  描述：提供所有节点保存在系统表gs_asp中的历史活跃session状态的采样记录。

  返回值类型：record

- get_wait_event_info()

  描述：提供wait event事件的具体信息。

  返回值类型：record

- dbe_perf.get_datanode_active_session(text)

  描述：提供从CN查询DN上保存在内存中的历史活跃session状态的采样记录。

  返回值类型：record

  备注：该函数查询目标DN上local_active_session视图中记录并和所有CN上的local_active_session中的记录进行匹配获取query string，所以会占用大量的内存。

- dbe_perf.get_datanode_active_session_hist(text,timestamp,timestamp)

  描述：提供从CN查询DN上保存在系统表gs_asp中的历史活跃session状态的采样记录。

  返回值类型：record

  备注：该函数查询目标DN上指定时间段的gs_asp记录，如果指定时间段过长造成查询的记录过多，会耗费大量时间。

- generate_wdr_report(bigint, bigint, cstring, cstring,cstring)

  描述：基于两个snapshot生成系统诊断报告，默认初始化用户或监控管理员用户可以访问。只可在系统库中查询到结果，用户库中无法查询。

  返回值类型：text

  表 7-54 generate_wdr_report 参数说明

参数	说明	取值范围
begin_snap_id	生成某段时间内性能诊断报告的开始snapshotid。	-
end_snap_id	结束snapshot的id，默认end_snap_id大于begin_snap_id。	-
report_type	指定生成report的类型。	• summary • detail • all，即同时包含summary和detail。
report_scope	指定生成report的范围。	• cluster：数据库级别的信息 • node：节点级别的信息。
node_name	• 在“report_scope”指定为“node”时，需要把该参数指定为对应节点的名称。 • 在“report_scope”为“cluster”时，该参数可以省略，或指定为NULL。	• node：GaussDB中的节点名称。 • cluster：省略/空/NULL。

- create_wdr_snapshot()

  描述：手工生成系统诊断快照，该函数需要sysadmin权限，且只能在CCN上执行。

  返回值类型：text

- kill_snapshot()

  描述：kill后台的WDR snapshot线程，调用该函数的用户需要具有SYSADMIN权限或具有REPLICATION权限或继承了内置角色gs_role_replication的权限。

  返回值类型：void

- capture_view_to_json(text,integer)

  描述：将视图的结果存入GUC: perf_directory所指定的目录，如果is_crossdb为1，则表示对于所有的database都会访问一次view；如果is_crossdb为0，则表示仅对当前database进行一次视图访问。该函数只有sysadmin和monitor admin用户可以执行。

  返回值类型：int

- reset_unique_sql(text,text,bigint)

描述：用来清理CN/DN内存中的Unique SQL（需要sysadmin/monitor admin权限）。

返回值类型：Boolean

表 7-55 reset_unique_sql 参数说明

参数	类型	描述
scope	text	清理范围类型： 'GLOBAL' - 清理所有的CN/DN节点，如果是'GLOBAL'，则只可以为CN节点执行此函数。 'LOCAL' - 清理本节点。
clean_type	text	'BY_USERID' - 按用户ID来进行清理Unique SQL。 'BY_CNID' - 按CN的ID来进行清理Unique SQL。 'ALL' - 全部清理。
clean_value	int8	具体清理type对应的清理值。如果第二个参数为ALL，则第三个参数不起作用，可以取任意值。

- wdr_xdb_query(db_name_str text, query text)

  描述：提供本地跨数据库执行query的能力。例如: 在连接到testdb库时, 访问test库下的表。只有系统管理员才有权限执行。

  ```
  select col1 from wdr_xdb_query('dbname=test','select col1 from t1') as dd(col1 int);
  ```

  返回值类型：record

- pg_wlm_jump_queue(pid int)

  描述：调整任务到CN队列的最前端。

  返回值类型：boolean

  - true：成功。
  - false：失败。

- gs_wlm_switch_cgroup(pid int, cgroup text)

  描述：调整作业的优先级到新控制组。

  返回值类型：boolean

  - true：成功。
  - false：失败。

- pv_session_memctx_detail(threadid tid, MemoryContextName text)

  描述：将线程tid的MemoryContextName内存上下文信息记录到“$GAUSSLOG/pg_log/${node_name}/dumpmem”目录下的“threadid_timestamp.log”文件中。其中threadid可通过查询表PV_SESSION_MEMORY_DETAIL中的sessid字段获得。在正式发布的版本中仅接受MemoryContextName为空串（两个单引号表示输入为空串，即''）的输入，此时会记录所有的内存上下文信息，否则不会有任何操作。该函数需要管理员权限的用户才能执行。

  返回值类型：boolean

  - true：成功。
  - false：失败。

- pg_shared_memctx_detail(MemoryContextName text)

  描述：将MemoryContextName内存上下文信息记录到“$GAUSSLOG/pg_log/${node_name}/dumpmem”目录下的“threadid_timestamp.log”文件中。在正式发布版本中调用该函数不会有任何操作。该函数需要管理员权限的用户才能执行。

  返回值类型：boolean

  - true：成功。
  - false：失败。

- local_bgwriter_stat()

  描述：显示本实例的bgwriter线程刷页信息，候选buffer链中页面个数，buffer淘汰信息。

  返回值类型：record

- local_candidate_stat()

  描述：显示本实例的候选buffer链中页面个数，buffer淘汰信息，包含normal buffer pool和segment buffer pool。

  返回值类型：record

- local_ckpt_stat()

  描述：显示本实例的检查点信息和各类日志刷页情况。

  返回值类型：record

- local_double_write_stat()

  描述：显示本实例的双写文件的情况。

  返回值类型：record

表 7-56 local_double_write_stat 参数说明

参数	类型	描述
node_name	text	实例名称。
curr_dwn	int8	当前双写文件的序列号。
curr_start_page	int8	当前双写文件恢复起始页面。
file_trunc_num	int8	当前双写文件复用的次数。
file_reset_num	int8	当前双写文件写满后发生重置的次数。
total_writes	int8	当前双写文件总的I/O次数。
low_threshold_writes	int8	低效率写双写文件的I/O次数（一次I/O刷页数量少于16页面）。
high_threshold_writes	int8	高效率写双写文件的I/O次数（一次I/O刷页数量多于一批，421个页面）。
total_pages	int8	当前刷页到双写文件区的总的页面个数。

参数	类型	描述
low_threshold_pages	int8	低效率刷页的页面个数。
high_threshold_pages	int8	高效率刷页的页面个数。
file_id	int8	当前双写文件的id号

- local_single_flush_dw_stat()
  描述：显示本实例的单页面淘汰双写文件的情况。
  返回值类型：record
- local_pagewriter_stat()
  描述：显示本实例的刷页信息和检查点信息。
  返回值类型：record
- local_redo_stat()
  描述：显示本实例的备机的当前回放状态。
  返回值类型：record
  备注：返回的回放状态主要包括当前回放位置，回放最小恢复点位置等信息。
- local_recovery_status()
  描述：显示本实例的主机和备机的日志流控信息。
  返回值类型：record
- local_rto_status()
  描述：显示本实例的主机和备机的日志流控信息。
  返回值类型：record
- gs_cgroup_map_ng_conf(group name)
  描述：读取指定Node group的cgroup配置文件。该函数只有sysadmin权限的用户可以执行。
  返回值类型：record
- pgxc_cgroup_map_ng_conf(group name)
  描述：在所有节点上读取指定Node group的cgroup配置文件。该函数只有sysadmin权限的用户可以执行。
  返回值类型：bool
- gs_wlm_switch_cgroup(sess_id int8, cgroup name)
  描述：切换指定会话的控制组。
  返回值类型：record
- comm_client_info()
  描述：用于查询单个节点活跃的客户端连接信息，返回结果解释见**COMM_CLIENT_INFO**。
  返回值类型：setof record
- pg_get_flush_lsn()
  描述：返回当前节点flush的xLog位置。

返回值类型：text

- pg_get_sync_flush_lsn()

  描述：返回当前节点多数派flush的xLog位置。

  返回值类型：text

- pgxc_wlm_rebuild_user_resource_pool()

  描述：重新构建用户及资源池缓存信息。需要系统管理员权限才可以执行该函数。

  返回值类型：boolean

- locktag_decode(locktag text)

  描述：从locktag中解析锁的具体信息。

  示例：

```
openGauss=# select locktag_decode('271b:0:0:0:0:6');
                locktag_decode
-----------------------------------------------
 locktype:transactionid, transactionid:10011
(1 row)
```

  返回值类型：text

- disable_conn(disconn_mode text, host text, port integer)

  描述：CM Agent处理CM Server下发的命令，在DN进行选主时设置该DN拒绝连接所有DN、强制连接某个DN或轮询连接所有DN。只有初始化用户和系统管理员才可以调用该函数。

  返回值类型：void

  表 7-57 disable_conn 参数说明

参数	类型	描述
disconn_mode	text	DN连接模式： ● 'prohibit_connection' - 拒绝连接所有DN。 ● 'specify_connection' - 强制连接某个DN。 ● 'polling_connection' - 轮询连接所有DN。
host	text	DN的IP。
port	integer	DN的端口号。

- dbe_perf.get_global_full_sql_by_timestamp(start_timestamp timestamp with time zone, end_timestamp timestamp with time zone)

  描述：获取集群级的全量SQL(Full SQL)信息。只可在系统库中查询到结果，用户库中无法查询。

  返回值类型：record

表 7-58 dbe_perf.get_global_full_sql_by_timestamp 参数说明

参数	类型	描述
start_timestamp	timestamp with time zone	SQL启动时间范围的开始时间点。
end_timestamp	timestamp with time zone	SQL启动时间范围的结束时间点。

- dbe_perf.get_global_slow_sql_by_timestamp(start_timestamp timestamp with time zone, end_timestamp timestamp with time zone)

  描述：获取集群级的慢SQL(Slow SQL)信息。只可在系统库中查询到结果，用户库中无法查询。

  返回值类型：record

  表 7-59 dbe_perf.get_global_slow_sql_by_timestamp 参数说明

参数	类型	描述
start_timestamp	timestamp with time zone	SQL启动时间范围的开始时间点。
end_timestamp	timestamp with time zone	SQL启动时间范围的结束时间点。

- statement_detail_decode(detail text, format text, pretty boolean)

  描述：解析全量/慢SQL语句中的details字段的信息。只可在系统库中查询到结果，用户库中无法查询。

  返回值类型：text

  表 7-60 statement_detail_decode 参数说明

参数	类型	描述
detail	text	SQL语句产生的事件的集合（不可读）。
format	text	解析输出格式，取值为plaintext。
pretty	boolean	当format为plaintext时，是否以优雅的格式展示： ● true表示通过“\n”分隔事件。 ● false表示通过“，”分隔事件。

- pgxc_get_csn(tid)

  描述：返回给定的事务id对应的事务提交序号（CSN）。

  返回值类型：int8

- get_global_user_transaction()

  描述：返回所有节点上各用户的事务相关信息。

返回值类型：node_name name, usename name, commit_counter bigint, rollback_counter bigint, resp_min bigint, resp_max bigint, resp_avg bigint, resp_total bigint, bg_commit_counter bigint, bg_rollback_counter bigint, bg_resp_min bigint, bg_resp_max bigint, bg_resp_avg bigint, bg_resp_total bigint

- pg_collation_for

  描述：返回入参字符串对应的排序规则

  参数：any（如果是常量必须进行显式类型转换）

  返回值类型：text

- pgxc_unlock_for_sp_database(name Name)

  描述：释放指定数据库锁。

  参数：数据库名

  返回值类型：布尔

- pgxc_lock_for_sp_database(name Name)

  描述：对指定的数据库加锁。

  参数：数据库名

  返回值类型：布尔

- pgxc_unlock_for_transfer(name Name)

  描述：释放用于数据传输（数据重分布）锁。

  参数：数据库名

  返回值类型：布尔

- pgxc_lock_for_transfer(name Name)

  描述：对数据库枷锁，用于数据传输（数据重分布）。

  参数：数据库名

  返回值类型：布尔

- gs_catalog_attribute_records()

  描述：对于指定的系统表oid，返回该系统表对应的各个字段的定义。仅支持oid小于10000的普通系统表（不支持索引、toast表等）。

  参数：系统表oid

  返回值类型：record

- dynamic_func_control(scope text, function_name text, action text, "{params}" text[])

  描述：动态开启内置的功能，当前仅支持动态开启全量SQL。

  返回值类型：record

表 7-61 dynamic_func_control 参数说明

参数	类型	描述
scope	text	动态开启功能的范围，当前仅支持'GLOBAL/LOCAL'。
function_name	text	功能的名称，当前仅支持'STMT'。

参数	类型	描述
action	text	当function_name为'STMT'时，action仅支持TRACK/UNTRACK/LIST/CLEAN： ● TRACK - 开始记录归一化SQL的全量SQL信息。 ● UNTRACK - 取消记录归一化SQL的全量SQL信息。 ● LIST - 列取当前TRACK的归一化SQL的信息。 ● CLEAN - 清理记录当前归一化SQL的信息。
params	text[]	当function_name为'STMT'时，对应不同的action时，对应的params设置如下： ● TRACK - '{"归一化SQLID", "L0/L1/L2"}' ● UNTRACK - '{"归一化SQLID"}' ● LIST - '{}' ● CLEAN - '{}'

- gs_parse_page_bypath(path text, blocknum bigint, relation_type text, read_memory boolean)

  描述：用于解析指定表页面，并返回存放解析内容的路径。

  返回值类型：text

  备注：必须是系统管理员或运维管理员才能执行此函数。

表 7-62 gs_parse_page_bypath 参数说明

参数	类型	描述
path	text	• 对于普通表或段页式的普通表，相对路径为：tablespace name/database oid/表的relfilenode(物理文件名)；例如：base/16603/16394 • 对于段页式的hashbucket表，相对路径为：tablespace name/database oid/Segment Head的逻辑页号_b(bucketid)。例如：base/16603/16394_b1437 • 表文件的相对路径可以通过pg_relation_filepath(table_name text)查找。分区表的路径可以查看pg_partition系统表和调用pg_partition_filepath(partition_oid)。 • 合法的path格式列举： – global/relNode – base/dbNode/relNode – pg_tblspc/spcNode/version_dir/dbNode/relNode 对于hashbucket表，在此格式基础上，路径末尾加上_b段页式的逻辑页号。
blocknum	bigint	• -1：所有block的信息（强制从磁盘解析）。 • 0~MaxBlockNumber：对应block的信息。
relation_type	text	• heap(astore表) • btree(BTree索引) • segment(段页式)
read_memory	boolean	• false，从磁盘文件解析； • true，首先尝试从共享缓冲区中解析该页面；如果共享缓冲区中不存在，则从磁盘文件解析。

- gs_xlogdump_lsn(start_lsn text, end_lsn text)

  描述：用于解析指定lsn范围之内的xLog日志，并返回存放解析内容的路径。可以通过pg_current_xlog_location()获取当前xLog位置。

  参数：LSN起始位置，LSN结束位置

  返回值类型：text

  备注：必须是系统管理员或运维管理员才能执行此函数。

- gs_xlogdump_xid(c_xid xid)

  描述：用于解析指定xid的xLog日志，并返回存放解析内容的路径。可以通过txid_current()获取当前事务ID。

  参数：事务ID

  返回值类型：text

备注：必须是系统管理员或运维管理员才能执行此函数。

- gs_xlogdump_tablepath(path text, blocknum bigint, relation_type text)
  描述：用于解析指定表页面对应的日志，并返回存放解析内容的路径。
  返回值类型：text
  备注：必须是系统管理员或运维管理员才能执行此函数。

  表 7-63 gs_xlogdump_tablepath 参数说明

参数	类型	描述
path	text	● 对于普通表或段页式的普通表，相对路径为：tablespace name/database oid/表的relfilenode(物理文件名)。例如：base/16603/16394。 ● 对于段页式的hashbucket表，相对路径为：tablespace name/database oid/Segment Head的逻辑页号_b(bucketid)。例如：base/16603/16394_b1437 ● 表文件的相对路径可以通过pg_relation_filepath(table_name text)查找。分区表的路径可以查看pg_partition系统表和调用pg_partition_filepath(partition_oid)。 ● 合法的path格式列举： – global/relNode – base/dbNode/relNode – pg_tblspc/spcNode/version_dir/dbNode/relNode 对于hashbucket表，在此格式基础上，路径末尾加上_b段页式的逻辑页号。
blocknum	bigint	● -1：所有block的信息（强制从磁盘解析）。 ● 0~MaxBlockNumber：对应block的信息。
relation_type	text	● heap(astore 表) ● btree(BTree 索引) ● segment(段页式)

- gs_xlogdump_parsepage_tablepath(path text, blocknum bigint, relation_type text, read_memory boolean)
  描述：用于解析指定表页面和表页面对应的日志，并返回存放解析内容的路径。可以看做一次执行gs_parse_page_bypath和gs_xlogdump_tablepath。该函数执行的前置条件是表文件存在。如果想查看已删除的表的相关日志，请直接调用gs_xlogdump_tablepath。
  返回值类型：text
  备注：必须是系统管理员或运维管理员才能执行此函数。

表 7-64 gs_xlogdump_parsepage_tablepath 参数说明

参数	类型	描述
path	text	● 对于普通表或段页式的普通表，相对路径为：tablespace name/database oid/表的relfilenode(物理文件名)；例如：base/16603/16394 ● 对于段页式的hashbucket表，相对路径为：tablespace name/database oid/Segment Head的逻辑页号_b(bucketid)。例如：base/16603/16394_b1437 ● 表文件的相对路径可以通过pg_relation_filepath(table_name text)查找。分区表的路径可以查看pg_partition系统表和调用pg_partition_filepath(partition_oid)。 ● 合法的path格式列举： – global/relNode – base/dbNode/relNode – pg_tblspc/spcNode/version_dir/dbNode/relNode 对于hashbucket表，在此格式基础上，路径末尾加上_b段页式的逻辑页号。
blocknum	bigint	● -1：所有block的信息（强制从磁盘解析）。 ● 0~MaxBlockNumber：对应block的信息。
relation_type	text	● heap(astore 表) ● btree(BTree 索引) ● segment(段页式)
read_memory	boolean	● false，从磁盘文件解析； ● true，首先尝试从共享缓冲区中解析该页面；如果共享缓冲区中不存在，则从磁盘文件解析。

- gs_index_recycle_queue(Oid oid, int type, uint32 blkno)

  描述：用于解析UBtree索引回收队列信息。

  返回值类型：record

  表 7-65 gs_index_recycle_queue 参数说明

参数	类型	描述
oid	Oid	● 索引文件relfilenode,可以通过select relfilenode from pg_class where relname='name'查询，其中name表示对应的索引文件名字

参数	类型	描述
type	int	● 0，表示解析整个待回收队列 ● 1，表示解析整个空页队列 ● 2，表示解析单个页面
blkno	uint32	回收队列页面编号，该参数只有在type=2的时候有效，blkno有效取值范围为1~4294967294。

说明

该函数功能在分布式版本上不支持，有报错提示。

- gs_stat_wal_entrytable(int64 idx)
  描述：用于输出xLog中预写日志插入状态表的内容。
  返回值类型：record

  表 7-66 gs_stat_wal_entrytable 参数说明

参数类型	参数名	类型	描述
输入参数	idx	int64	● -1：查询数组所有元素。 ● 0-最大值：具体某个数组元素内容。
输出参数	idx	uint64	记录对应数组中的下标
输出参数	endlsn	uint64	记录的LSN标签
输出参数	lrc	int32	记录对应的LRC
输出参数	status	uint32	标识当前entry对应的xLog是否已经完全拷贝到wal buffer中 ● 0：非COPIED ● 1: COPIED

- gs_walwriter_flush_position()
  描述：输出预写日志的刷新位置。
  返回值类型：record

表 7-67 gs_walwriter_flush_position 参数说明

参数类型	参数名	类型	描述
输出参数	last_flush_status_entry	int32	xLog flush上一个刷盘的tblEntry下标索引。
输出参数	last_scanned_lrc	int32	xLog flush上一次扫描到的最后一个tblEntry记录的LRC。
输出参数	curr_lrc	int32	WALInsertStatusEntry状态表中LRC最新的使用情况，该LRC表示下一个xLog记录写入时在WALInsertStatusEntry对应的LRC值。
输出参数	curr_byte_pos	uint64	xLog记录写入WAL 文件，最新分配的位置，下一个xLog记录插入点。
输出参数	prev_byte_size	uint32	上一个xLog记录的长度。
输出参数	flush_result	uint64	当前全局xLog刷盘的位置。
输出参数	send_result	uint64	当前主机上xLog发送位置。
输出参数	shm_rqst_write_pos	uint64	共享内存中记录的XLogCtl中LogwrtRqst请求的write位置。
输出参数	shm_rqst_flush_pos	uint64	共享内存中记录的XLogCtl中LogwrtRqst请求的flush位置。
输出参数	shm_result_write_pos	uint64	共享内存中记录的XLogCtl中LogwrtResult的write位置。
输出参数	shm_result_flush_pos	uint64	共享内存中记录的XLogCtl中LogwrtResult的flush位置。
输出参数	curr_time	text	当前时间。

- gs_walwriter_flush_stat(int operation)

  描述：用于统计预写日志write与sync的次数频率与数据量，以及xLog文件的信息。

  返回值类型：record

表 7-68 gs_walwriter_flush_stat 参数说明

参数类型	参数名	类型	描述
输入参数	operation	int	● -1： 关闭统计开关(默认状态为关闭)。 ● 0：打开统计开关。 ● 1：查询统计信息。 ● 2：重置统计信息。
输出参数	write_times	uint64	xLog调用write接口的次数
输出参数	sync_times	uint64	xLog调用sync接口次数
输出参数	total_xlog_sync_bytes	uint64	Backend线程请求写入xLog总量统计值
输出参数	total_actual_xlog_sync_bytes	uint64	调用sync接口实际刷盘的xLog总量统计值
输出参数	avg_write_bytes	uint32	每次调用XLogWrite接口请求写的xLog量
输出参数	avg_actual_write_bytes	uint32	实际每次调用write接口写的xLog量
输出参数	avg_sync_bytes	uint32	平均每次请求sync的xLog量
输出参数	avg_actual_sync_bytes	uint32	实际每次调用sync刷盘xLog量
输出参数	total_write_time	uint64	调用write操作总时间统计(单位：us)
输出参数	total_sync_time	uint64	调用sync操作总时间统计(单位：us)
输出参数	avg_write_time	uint32	每次调用write接口平均时间(单位：us)
输出参数	avg_sync_time	uint32	每次调用sync接口平均时间(单位：us)
输出参数	curr_init_xlog_segno	uint64	当前最新创建的xLog段文件编号
输出参数	curr_open_xlog_segno	uint64	当前正在写的xLog段文件编号
输出参数	last_reset_time	text	上一次重置统计信息的时间

参数类型	参数名	类型	描述
输出参数	curr_time	text	当前时间

- pg_ls_tmpdir()

  描述：返回默认表空间下临时目录（pgsql_tmp）中每个文件的名称、大小和最后修改时间。

  参数：nan

  返回值类型：record

  备注：必须是系统管理员或者监控管理员才能执行此函数。

参数类型	参数名	类型	描述
输出参数	name	text	文件名称
输出参数	size	int8	文件大小（单位：byte）
输出参数	modification	timestamptz	文件最后修改时间

- pg_ls_tmpdir(oid)

  描述：返回指定表空间下临时目录（pgsql_tmp）中每个文件的名称、大小和最后修改时间。

  参数：oid

  返回值类型：record

  备注：必须是系统管理员或者监控管理员才能执行此函数。

参数类型	参数名	类型	描述
输入参数	oid	oid	表空间id
输出参数	name	text	文件名称
输出参数	size	int8	文件大小（单位：byte）
输出参数	modification	timestamptz	文件最后修改时间

- pg_ls_waldir()

  描述：返回预写日志(WAL)目录中每个文件的名称、大小和最后修改时间。

  参数：nan

  返回值类型：record

  备注：必须是系统管理员或者监控管理员才能执行此函数。

参数类型	参数名	类型	描述

输出参数	name	text	文件名称
输出参数	size	int8	文件大小（单位：byte）
输出参数	modification	timestamptz	文件最后修改时间

- gs_undo_dump_xid(undo_xid xid)
  描述：根据xid解析undo记录
  返回值类型：record

  表 7-69 gs_undo_dump_xid 参数说明

参数类型	参数名	类型	描述
输入参数	undo_xid	xid	事务xid
输出参数	undoptr	xid	需要解析的undo记录起始位置
输出参数	xactid	text	事务id
输出参数	cid	text	command id
输出参数	reloid	text	relation oid
输出参数	relfilenode	text	文件的relfinode
输出参数	utype	text	undo记录类型
输出参数	blkprev	text	同一个块前一条undo记录的位置
输出参数	blockno	text	块号
输出参数	uoffset	text	undo记录偏移
输出参数	prevurp	text	前一条undo记录位置
输出参数	payloadlen	text	undo记录数据部分长度
输出参数	oldxactid	text	前一个事务id

参数类型	参数名	类型	描述
输出参数	partitionoid	text	分区oid
输出参数	tablespace	text	表空间
输出参数	alreadyread_bytes	text	读取到的undo记录长度
输出参数	prev_undorec_len	text	前一条undo记录长度
输出参数	td_id	text	Transaction Directory的id
输出参数	reserved	text	是否保存
输出参数	flag	text	标识1
输出参数	flag2	text	标识2
输出参数	t_hoff	text	Undo记录数据头的长度

- gs_write_term_log(void)

  描述：写入一条日志记录DN节点当前的term值。备DN节点返回false，主DN节点写入成功后返回true。

  返回值类型：Boolean

7.5.24 统计信息函数

统计信息函数根据访问对象分为两种类型：针对某个数据库进行访问的函数，以数据库中每个表或索引的OID作为参数，标识需要报告的数据库；针对某个服务器进行访问的函数，以一个服务器进程号为参数，其范围从1到当前活跃服务器的数目。

- pg_stat_get_db_conflict_tablespace(oid)

  描述：由于恢复与数据库中删除的表空间发生冲突而取消的查询数。

  返回值类型：bigint

- pg_control_group_config()

  描述：在当前节点上打印cgroup配置。该函数需要sysadmin权限的用户才能够执行。

  返回值类型：record

- pg_stat_get_db_stat_reset_time(oid)

  描述：上次重置数据库统计信息的时间。首次连接到每个数据库期间初始化为系统时间。当您在数据库上调用pg_stat_reset以及针对其中的任何表或索引执行pg_stat_reset_single_table_counters时，重置时间都会更新。

返回值类型：timestamptz

- pg_stat_get_function_total_time(oid)

  描述：该函数花费的总挂钟时间，以微秒为单位。包括花费在此函数调用其它函数上的时间。

  返回值类型：bigint

- pg_stat_get_xact_tuples_returned(oid)

  描述：当前事务中参数为表时通过顺序扫描读取的行数，或参数为索引时返回的索引条目数。

  返回值类型：bigint

- pg_stat_get_xact_numscans(oid)

  描述：当前事务中参数为表时执行的顺序扫描次数，或参数为索引时执行的索引扫描次数。

  返回值类型：bigint

- pg_stat_get_xact_blocks_fetched(oid)

  描述：当前事务中对表或索引的磁盘块获取请求数。

  返回值类型：bigint

- pg_stat_get_xact_blocks_hit(oid)

  描述：当前事务中对缓存中找到的表或索引的磁盘块获取请求数。

  返回值类型：bigint

- pg_stat_get_xact_function_calls(oid)

  描述：在当前事务中调用该函数的次数。

  返回值类型：bigint

- pg_stat_get_xact_function_self_time(oid)

  描述：在当前事务中仅花费在此函数上的时间，不包括花费在此函数内部调用其它函数上的时间。

  返回值类型：bigint

- pg_stat_get_xact_function_total_time(oid)

  描述：当前事务中该函数所花费的总挂钟时间（以微秒为单位）。包括花费在此函数内部调用其它函数上的时间。

  返回值类型：bigint

- pg_lock_status()

  描述：查询打开事务所持有的锁信息，所有用户均可执行该函数。

  返回值类型：返回字段可参考**PG_LOCKS**视图返回字段，该视图是通过查询本函数得到的结果。

- pg_stat_get_wal_senders()

  描述：在主机端查询walsender信息。

  返回值类型：setofrecord

  返回字段说明如下：

表 7-70 返回字段说明

字段名称	字段类型	字段说明
pid	bigint	walsender的线程号。
sender_pid	integer	walsender的pid相对的轻量级线程号。
local_role	text	主节点类型。
peer_role	text	备节点类型。
peer_state	text	备节点状态。
state	text	walsender状态。
catchup_start	timestamp with time zone	catchup启动时间。
catchup_end	timestamp with time zone	catchup结束时间。
sender_sent_location	text	主节点发送位置。
sender_write_location	text	主节点落盘位置。
sender_flush_location	text	主节点flush磁盘位置。
sender_replay_location	text	主节点redo位置。
receiver_received_location	text	备节点接收位置。
receiver_write_location	text	备节点落盘位置。
receiver_flush_location	text	备节点flush磁盘位置。
receiver_replay_location	text	备节点redo磁盘位置。
sync_percent	text	同步百分比。
sync_state	text	同步状态。
sync_group	text	同步复制的所属分组。
sync_priority	text	同步复制的优先级。
sync_most_available	text	最大可用模式设置。
channel	text	walsender信道信息。

- pgxc_get_senders_catchup_time()

  描述：在CN实例查询集群中是否存在处于日志追赶状态的备DN，以及追赶状态详情。

  返回值类型：setofrecord
- pg_stat_get_stream_replications()

  描述：查询主备复制状态。

返回值类型：setofrecord
返回值说明如下：

表 7-71 返回值说明

返回参数	返回参数类型	返回参数说明
local_role	text	本地角色。
static_connections	integer	连接统计。
db_state	text	数据库状态。
detail_information	text	详细信息。

- pg_stat_get_db_numbackends(oid)
  描述：处理该数据库活跃的服务器进程数目。
  返回值类型：integer
- pg_stat_get_db_xact_commit(oid)
  描述：数据库中已提交事务的数量。
  返回值类型：bigint
- pg_stat_get_db_xact_rollback(oid)
  描述：数据库中回滚事务的数量。
  返回值类型：bigint
- pg_stat_get_db_blocks_fetched(oid)
  描述：数据库中磁盘块抓取请求的总数。
  返回值类型：bigint
- pg_stat_get_db_blocks_hit(oid)
  描述：数据库在缓冲区中找到的磁盘块抓取请求的总数。
  返回值类型：bigint
- pg_stat_get_db_tuples_returned(oid)
  描述：为数据库返回的Tuple数。
  返回值类型：bigint
- pg_stat_get_db_tuples_fetched(oid)
  描述：为数据库中获取的Tuple数。
  返回值类型：bigint
- pg_stat_get_db_tuples_inserted(oid)
  描述：在数据库中插入Tuple数。
  返回值类型：bigint
- pg_stat_get_db_tuples_updated(oid)
  描述：在数据库中更新的Tuple数。
  返回值类型：bigint
- pg_stat_get_db_tuples_deleted(oid)
  描述：数据库中删除Tuple数。

返回值类型：bigint

- pg_stat_get_db_conflict_lock(oid)
  描述：数据库中锁冲突的数量。
  返回值类型：bigint
- pg_stat_get_db_deadlocks(oid)
  描述：数据库中死锁的数量。
  返回值类型：bigint
- pg_stat_get_numscans(oid)
  描述：如果参数是一个表，则顺序扫描读取的行数目。如果参数是一个索引，则返回索引行的数目。
  返回值类型：bigint
- pg_stat_get_role_name(oid)
  描述：根据用户oid获取用户名。仅sysadmin和MONADMIN用户可以访问。
  返回值类型：text
  示例：

```
openGauss=# select pg_stat_get_role_name(10);
 pg_stat_get_role_name
-----------------------
 aabbcc
(1 row)
```

- pg_stat_get_tuples_returned(oid)
  描述：如果参数是一个表，则顺序扫描读取的行数目。如果参数是一个索引，则返回的索引行的数目。
  返回值类型：bigint
- pg_stat_get_tuples_fetched(oid)
  描述：如果参数是一个表，则位图扫描抓取的行数目。如果参数是一个索引，则用简单索引扫描抓取的行数目。
  返回值类型：bigint
- pg_stat_get_tuples_inserted(oid)
  描述：插入表中行的数量。
  返回值类型：bigint
- pg_stat_get_tuples_updated(oid)
  描述：在表中已更新行的数量。
  返回值类型：bigint
- pg_stat_get_tuples_deleted(oid)
  描述：从表中删除行的数量。
  返回值类型：bigint
- pg_stat_get_tuples_changed(oid)
  描述：该表上一次analyze或autoanalyze之后插入、更新、删除行的总数量。
  返回值类型：bigint
- pg_stat_get_tuples_hot_updated(oid)
  描述：表热更新的行数。
  返回值类型：bigint

- pg_stat_get_live_tuples(oid)
  描述：表活行数。
  返回值类型：bigint
- pg_stat_get_dead_tuples(oid)
  描述：表死行数。
  返回值类型：bigint
- pg_stat_get_blocks_fetched(oid)
  描述：表或者索引的磁盘块抓取请求的数量。
  返回值类型：bigint
- pg_stat_get_blocks_hit(oid)
  描述：在缓冲区中找到的表或者索引的磁盘块请求数目。
  返回值类型：bigint
- pg_stat_get_partition_tuples_inserted(oid)
  描述：插入相应表分区中行的数量。
  返回值类型：bigint
- pg_stat_get_partition_tuples_updated(oid)
  描述：在相应表分区中已更新行的数量。
  返回值类型：bigint
- pg_stat_get_partition_tuples_deleted(oid)
  描述：从相应表分区中删除行的数量。
  返回值类型：bigint
- pg_stat_get_partition_tuples_changed(oid)
  描述：该表分区上一次analyze或autoanalyze之后插入、更新、删除行的总数量。
  返回值类型：bigint
- pg_stat_get_partition_live_tuples(oid)
  描述：活行数表分区。
  返回值类型：bigint
- pg_stat_get_partition_dead_tuples(oid)
  描述：死行数表分区。
  返回值类型：bigint
- pg_stat_get_xact_tuples_fetched(oid)
  描述：事务中扫描的tuple行数。
  返回值类型：bigint
- pg_stat_get_xact_tuples_inserted(oid)
  描述：表相关的活跃子事务中插入的tuple数。
  返回值类型：bigint
- pg_stat_get_xact_tuples_deleted(oid)
  描述：表相关的活跃子事务中删除的tuple数。
  返回值类型：bigint

- pg_stat_get_xact_tuples_hot_updated(oid)
  描述：表相关的活跃子事务中热更新的tuple数。
  返回值类型：bigint
- pg_stat_get_xact_tuples_updated(oid)
  描述：表相关的活跃子事务中更新的tuple数。
  返回值类型：bigint
- pg_stat_get_xact_partition_tuples_inserted(oid)
  描述：表分区相关的活跃子事务中插入的tuple数。
  返回值类型：bigint
- pg_stat_get_xact_partition_tuples_deleted(oid)
  描述：表分区相关的活跃子事务中删除的tuple数。
  返回值类型：bigint
- pg_stat_get_xact_partition_tuples_hot_updated(oid)
  描述：表分区相关的活跃子事务中热更新的tuple数。
  返回值类型：bigint
- pg_stat_get_xact_partition_tuples_updated(oid)
  描述：表分区相关的活跃子事务中更新的tuple数。
  返回值类型：bigint
- pg_stat_get_last_vacuum_time(oid)
  描述：用户在该表上最后一次手动启动清理或者autovacuum线程启动清理的时间。
  返回值类型：timestamptz
- pg_stat_get_last_autovacuum_time(oid)
  描述：autovacuum守护进程在该表上最后一次启动清理的时间。
  返回值类型：timestamptz
- pg_stat_get_vacuum_count(oid)
  描述：用户在该表上启动清理的次数。
  返回值类型：bigint
- pg_stat_get_autovacuum_count(oid)
  描述：autovacuum守护进程在该表上启动清理的次数。
  返回值类型：bigint
- pg_stat_get_last_analyze_time(oid)
  描述：用户在该表上最后一次手动启动分析或者autovacuum线程启动分析的时间。
  返回值类型：timestamptz
- pg_stat_get_last_autoanalyze_time(oid)
  描述：autovacuum守护进程在该表上最后一次启动分析的时间。
  返回值类型：timestamptz
- pg_stat_get_analyze_count(oid)
  描述：用户在该表上启动分析的次数。
  返回值类型：bigint

- pg_stat_get_autoanalyze_count(oid)

  描述：autovacuum守护进程在该表上启动分析的次数。

  返回值类型：bigint

- pg_total_autovac_tuples(bool)

  描述：返回total autovac相关的tuple记录，如nodename、nspname、relname以及各类tuple的IUD信息，入参为：是否查询relation信息。

  返回值类型：setofrecord

  返回参数说明如下：

  表 7-72 返回参数说明

返回参数	返回参数类型	返回参数说明
nodename	name	节点名称。
nspname	name	名称空间名称。
relname	name	表、索引、视图等对象名称。
partname	name	分区名称。
n_dead_tuples	bigint	表分区内的死行数。
n_live_tuples	bigint	表分区内的活行数。
changes_since_analyze	bigint	analyze产生改变的数量。

- pg_autovac_status(oid)

  描述：返回和autovac状态相关的参数信息，如nodename,nspname,relname,analyze,vacuum设置，analyze/vacuum阈值,analyze/vacuum tuple数等。仅sysadmin可以使用该函数。

  返回值类型：setofrecord

  返回值参数说明如下：

  表 7-73 返回值参数说明

返回参数	返回参数类型	返回参数说明
nspname	text	名称空间名称。
relname	text	表、索引、视图等对象名称。
nodename	text	节点名称。
doanalyze	Boolean	是否执行analyze。
anltuples	bigint	analyze tuple数量。
anlthresh	bigint	analyze阈值。

返回参数	返回参数类型	返回参数说明
dovacuum	Boolean	是否执行vacuum。
vactuples	bigint	vacuum tuple数量。
vacthresh	bigint	vacuum阈值。

- pg_autovac_timeout(oid)

  描述：返回某个表做autovac连续超时的次数，表信息非法或node信息异常返回NULL。

  返回值类型：bigint

- pg_autovac_coordinator(oid)

  描述：返回对某个表做autovac的coordinator名称，表信息非法或node信息异常返回NULL。

  返回值类型：text

- pg_stat_get_last_data_changed_time(oid)

  描述：insert/update/delete, exchange/truncate/drop partition在该表上最后一次操作的时间，**PG_STAT_ALL_TABLES**视图last_data_changed列的数据是通过该函数求值，在表数量很大的场景中，通过视图获取表数据最后修改时间的性能较差，建议直接使用该函数获取表数据的最后修改时间。

  返回值类型：timestamptz

- pg_stat_set_last_data_changed_time(oid)

  描述：手动设置该表上最后一次insert/update/delete, exchange/truncate/drop partition操作的时间。

  返回值类型：void

- pg_backend_pid()

  描述：当前会话的服务器线程的线程ID。

  返回值类型：integer

- pg_stat_get_activity(integer)

  描述：返回一个关于带有特殊PID的后台进程的记录信息，当参数为NULL时，则返回每个活动的后台进程的记录。返回结果不包含connection_info列。初始用户、系统管理员和MONADMIN可以查看所有的数据，普通用户只能查询自己的结果。

  示例：

```
openGauss=# select * from pg_stat_get_activity(139881386280704);
 datid |       pid       | sessionid | usesysid | application_name | state  |
query                      | waiting |          xact_start           |          query_start          |
    backend_start         |         state_change          | client_addr | client_hostname | client_port | enqueue
|     query_id      |   srespool   | global_sessionid | unique_sql_id | trace_id
-------+-----------------+-----------+----------+------------------+--------
+-----------------------------------------------------+---------+-------------------------------
+-------------------------------+-----
-------------------------+-----------------------------+-------------+-----------------+-------------+---------
+------------------+--------------+------------------+---------------+----------
 16545 | 139881386280704 |        69 |       10 | gsql             | active | select * from
pg_stat_get_activity(139881386280704); | f       | 2022-01-18 19:43:05.167718+08 | 2022-01-18
19:43:05.167718+08 | 2022
-01-18 19:42:33.513507+08 | 2022-01-18 19:43:05.16773+08 |             |               |         -1 |         |
72620543991624410 | default_pool | 1938253334#69#0  |    3751941862 |
(1 row)
```


返回值类型：setofrecord

返回参数说明如下：

表 7-74 返回参数说明

返回参数	返回参数类型	返回参数说明
datid	oid	用户会话在后台连接到的数据库OID。
pid	bigint	后台线程ID。
sessionid	bigint	会话ID。
usesysid	oid	登录该后台的用户OID。
application_name	text	连接到该后台的应用名。
state	text	该后台当前总体状态。
query	text	该后台的最新查询。如果state状态是active（活跃的），此字段显示当前正在执行的查询。所有其他情况表示上一个查询。
waiting	Boolean	如果后台当前正等待锁则为true。
xact_start	timestamp with time zone	启动当前事务的时间，如果没有事务是活跃的，则为null。 如果当前查询是首个事务，则这列等同于query_start列。
query_start	timestamp with time zone	开始当前活跃查询的时间，如果state的值不是active，则这个值是上一个查询的开始时间。如果是存储过程、函数、PACKAGE，则查询的是第一个查询时间，不会随着存储过程内语句运行而改变。
backend_start	timestamp with time zone	该过程开始的时间，即当客户端连接服务器时。
state_change	timestamp with time zone	上次状态改变的时间。

返回参数	返回参数类型	返回参数说明
client_addr	inet	连接到该后台的客户端的IP地址。如果此字段是null，它表明通过服务器机器上UNIX套接字连接客户端或者这是内部进程，如autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过client_addr的反向DNS查找得到。这个字段只有在启动log_hostname且使用IP连接时才非空。
client_port	integer	客户端用于与后台通讯的TCP端口号，如果使用Unix套接字，则为-1。
enqueue	text	该字段暂不支持。
query_id	bigint	查询语句的ID。
srespool	name	资源池名字。
global_sessionid	text	全局会话id。
unique_sql_id	bigint	语句的unique sql id。
trace_id	text	驱动传入的trace id，与应用的一次请求相关联。

- pg_stat_get_activity_with_conninfo(integer)

  描述：返回一个关于带有特殊PID的后台进程的记录信息，当参数为NULL时，则返回每个活动的后台进程的记录。初始用户、系统管理员和MONADMIN可以查看所有的数据，普通用户只能查询自己的结果。

  返回值类型：setofrecord

  返回值说明如下：

  表 7-75 返回值说明

返回值	返回值类型	返回值说明
datid	oid	用户会话在后台连接到的数据库OID。
pid	bigint	后台线程ID。
sessionid	bigint	会话ID。
usesysid	oid	登录该后台的用户OID。

返回值	返回值类型	返回值说明
application_name	text	连接到该后台的应用名。
state	text	改后台当前总体状态。
query	text	该后台的最新查询。如果state状态是active（活跃的），此字段显示当前正在执行的查询。所有其他情况表示上一个查询。
waiting	Boolean	如果后台当前正等待锁则为true。
xact_start	timestamp with time zone	启动当前事务的时间，如果没有事务是活跃的，则为null。如果当前查询是首个事务，则这列等同于query_start列。
query_start	timestamp with time zone	开始当前活跃查询的时间，如果state的值不是active，则这个值是上一个查询的开始时间。如果是存储过程、函数、PACKAGE，则查询的是第一个查询时间，不会随着存储过程内语句运行而改变。
backend_start	timestamp with time zone	该过程开始的时间，即当客户端连接服务器时。
state_change	timestamp with time zone	上次状态改变的时间。
client_addr	inet	连接到该后台的客户端的IP地址。如果此字段是null，它表明通过服务器机器上UNIX套接字连接客户端或者这是内部进程，如autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过client_addr的反向DNS查找得到。这个字段只有在启动log_hostname且使用IP连接时才非空。

返回值	返回值类型	返回值说明
client_port	integer	客户端用于与后台通讯的TCP端口号，如果使用Unix套接字，则为-1。
enqueue	text	该字段暂不支持。
query_id	bigint	查询语句的ID。
connection_info	text	json格式字符串，记录当前连接数据库的驱动类型、驱动版本号、当前驱动的部署路径、进程属主用户等信息。
srespool	name	资源池名字。
global_sessionid	text	全局会话ID。
unique_sql_id	bigint	语句的unique sql id。
trace_id	text	驱动传入的trace id，与应用的一次请求相关联。

- pg_stat_get_activity_ng(integer)

  描述：返回一个关于带有特殊PID的活跃后台线程记录信息，当参数为NULL时，则返回每个活跃的后台线程的记录。系统管理员和MONADMIN可以查看所有的数据，普通用户只能查询自己的结果。

  返回值类型：setofrecord

  函数返回字段说明如下：

名称	类型	描述
datid	oid	数据库oid。
pid	bigint	后端线程的ID。
sessionid	bigint	会话的id。
node_group	text	数据所属用户对应的Node group。

- pg_stat_get_function_calls(oid)

  描述：函数已被调用次数。

  返回值类型：bigint

- pg_stat_get_function_self_time(oid)

  描述：只有在此函数所花费的时间。函数嵌套调用其它函数所花费的时间被排除在外。

返回值类型：bigint

- pg_stat_get_backend_idset()

  描述：设置当前活动的服务器进程数（从1到活动服务器进程的数量）。

  返回值类型：setofinteger

- pg_stat_get_backend_pid(integer)

  描述：指定的服务器线程的线程ID。

  返回值类型：bigint

- pg_stat_get_backend_dbid(integer)

  描述：指定服务器进程的数据库ID。

  返回值类型：oid

- pg_stat_get_backend_userid(integer)

  描述：指定服务器进程的用户ID，本函数仅系统管理员可调用。

  返回值类型：oid

- pg_stat_get_backend_activity(integer)

  描述：指定服务器进程的当前活动查询，仅在调用者是系统管理员或被查询会话的用户，并且打开track_activities的时候才能获得结果。

  返回值类型：text

- pg_stat_get_backend_waiting(integer)

  描述：如果指定服务器进程在等待某个锁，并且调用者是系统管理员或被查询会话的用户，并且打开track_activities的时候才返回真。

  返回值类型：Boolean

- pg_stat_get_backend_activity_start(integer)

  描述：指定服务器进程当前正在执行的查询的起始时间，仅在调用者是系统管理员或被查询会话的用户，并且打开track_activities的时候才能获得结果。

  返回值类型：timestampwithtimezone

- pg_stat_get_backend_xact_start(integer)

  描述：指定服务器进程当前正在执行的事务的开始时间，但只有当前用户是系统管理员或被查询会话的用户，并且打开track_activities的时候才能获得结果。

  返回值类型：timestampwithtimezone

- pg_stat_get_backend_start(integer)

  描述：指定服务器进程启动的时间，如果当前用户不是系统管理员或被查询的后端的用户，则返回NULL。

  返回值类型：timestampwithtimezone

- pg_stat_get_backend_client_addr(integer)

  描述：连接到指定客户端后端的IP地址。如果是通过UNIX域套接字连接的则返回NULL；如果当前用户不是系统管理员或被查询会话的用户，也返回NULL。

  返回值类型：inet

- pg_stat_get_backend_client_port(integer)

  描述：连接到指定客户端后端的TCP端口。如果是通过UNIX域套接字连接的则返回-1；如果当前用户不是系统管理员或被查询会话的用户，也返回NULL。

  返回值类型：integer

- pg_stat_get_bgwriter_timed_checkpoints()
  描述：后台写进程开启定时检查点的时间（因为checkpoint_timeout时间已经过期了）。
  返回值类型：bigint
- pg_stat_get_bgwriter_requested_checkpoints()
  描述：后台写进程开启基于后端请求的检查点的时间，因为已经超过了checkpoint_segments或因为已经执行了CHECKPOINT。
  返回值类型：bigint
- pg_stat_get_bgwriter_buf_written_checkpoints()
  描述：在检查点期间后台写进程写入的缓冲区数目。
  返回值类型：bigint
- pg_stat_get_bgwriter_buf_written_clean()
  描述：为日常清理脏块，后台写进程写入的缓冲区数目。
  返回值类型：bigint
- pg_stat_get_bgwriter_maxwritten_clean()
  描述：后台写进程停止清理扫描的时间，因为已经写入了更多的缓冲区（相比bgwriter_lru_maxpages参数声明的缓冲区数）。
  返回值类型：bigint
- pg_stat_get_buf_written_backend()
  描述：后端进程写入的缓冲区数，因为它们需要分配一个新的缓冲区。
  返回值类型：bigint
- pg_stat_get_buf_alloc()
  描述：分配的总缓冲区数。
  返回值类型：bigint
- pg_stat_clear_snapshot()
  描述：清理当前的统计快照。该函数仅sysadmin和MONADMIN可以执行。
  返回值类型：void
- pg_stat_reset()
  描述：为当前数据库重置统计计数器为0（需要系统管理员权限）。
  返回值类型：void
- gs_stat_reset()
  描述：将各节点上的为当前数据库重置统计计数器为0（需要系统管理员权限）。
  返回值类型：void
- pg_stat_reset_shared(text)
  描述：重置shared cluster每个节点当前数据统计计数器为0（需要系统管理员权限）。
  返回值类型：void
- pg_stat_reset_single_table_counters(oid)
  描述：为当前数据库中的一个表或索引重置统计为0（需要系统管理员权限）。
  返回值类型：void
- pg_stat_reset_single_function_counters(oid)

描述：为当前数据库中的一个函数重置统计为0（需要系统管理员权限）。

返回值类型：void

- pg_stat_get_db_cu_hdd_asyn(oid)

  描述：获取当前节点一个数据库从磁盘异步读取CU次数。

  返回值类型：bigint

- pgxc_fenced_udf_process(integer)

  描述：查看UDF Master和Work进程数，仅sysadmin和MONADMIN用户有权限执行。入参为1时查看master进程数，入参为2时查看worker进程数，入参为3时终止所有worker进程。

  返回值类型：text

- fenced_udf_process(integer)

  描述：查看本地UDF Master和Work进程数。入参为1时查看master线程数，入参为2时查看worker线程数，入参为3时终止所有worker线程。

  返回值类型：text

- total_cpu()

  描述：获取当前节点使用的cpu时间，单位是jiffies。

  返回值类型：bigint

- total_memory()

  描述：获取当前节点使用的虚拟内存大小，单位KB。

  返回值类型：bigint

- pgxc_terminate_all_fenced_udf_process()

  描述：Kill所有的UDF Work进程，仅sysadmin和MONADMIN用户有权限执行。

  返回值类型：bool

- GS_ALL_NODEGROUP_CONTROL_GROUP_INFO(text)

  描述：提供了所有Node group的控制组信息。该函数在调用的时候需要指定要查询Node group的名称。

  例如要查询'installationNode group的控制组信息：

  ```
  SELECT * FROM GS_ALL_NODEGROUP_CONTROL_GROUP_INFO('installation')
  ```

  返回值类型：record

  函数返回字段如下：

名称	类型	描述
name	text	控制组的名称。
type	text	控制组的类型。
gid	bigint	控制组ID。
classgid	bigint	Workload所属Class的控制组ID。
class	text	Class控制组。
workload	text	Workload控制组。
shares	bigint	控制组分配的CPU资源配额。
limits	bigint	控制组分配的CPU资源限额。

名称	类型	描述
wdlevel	bigint	Workload控制组层级。
cpucores	text	控制组使用的CPU核的信息。

- gs_get_nodegroup_tablecount(name)

  描述：得到一个Node group中所有数据库包含的用户表数目。

  返回值类型：integer

- pgxc_max_datanode_size(name)

  描述：得到一个Node group的所有DN节点中数据库文件占用磁盘空间的最大值，单位为字节。

  返回值类型：bigint

- gs_check_logic_cluster_consistency()

  描述：检查当前系统中所有Node group是否存在系统信息不一致的情况，如果返回空记录，表示不存在不一致情况；否则，Node group中CN和DN上的NodeGroup信息存在不一致。该函数应该在非扩缩容重分布时调用。

  返回值类型：record

- gs_check_tables_distribution()

  描述： 检查当前系统中用户表的分布是否存在不一致，如果返回空记录，表示不存在不一致。该函数应该在非扩缩容重分布时调用。

  返回值类型：record

- pg_stat_bad_block(text, int, int, int, int, int, timestamp with time zone, timestamp with time zone)

  描述：获取当前节点自启动后，读取出现Page/CU的损坏信息。

  返回值类型：record

- pgxc_stat_bad_block(text, int, int, int, int, int, timestamp with time zone, timestamp with time zone)

  描述：获取集群所有节点自启动后，读取出现Page/CU的损坏信息。

  返回值类型：record

- pg_stat_bad_block_clear()

  描述：清理节点记录的读取出现的Page/CU损坏信息（需要系统管理员权限）。

  返回值类型：void

- pgxc_stat_bad_block_clear

  描述：清理集群所有节点记录的读取出现的Page/CU损坏信息（需要系统管理员权限）。

  返回值类型：void

- pgxc_log_comm_status(void)

  描述：当使用TCP代理通信时，PGXC系统视图将datanode的通信层状态输出到各个日志文件中。

  返回值类型：void

- gs_respool_exception_info(pool text)

  描述：查看某个资源池关联的查询规则信息。

返回值类型：record

- gs_control_group_info(pool text)

  描述：查看资源池关联的控制组信息。该函数需要sysadmin权限的用户才能够执行。

  返回值类型：record

  返回信息如下：

属性	属性值	描述
name	class_a:workload_a1	class和workload名称。
class	class_a	Class控制组名称。
workload	workload_a1	Workload控制组名称。
type	DEFWD	控制组类型（Top、CLASS、BAKWD、DEFWD、TSWD）。
gid	87	控制组id。
shares	30	占父节点CPU资源的百分比。
limits	0	占父节点CPU核数的百分比。
rate	0	Timeshare中的分配比例。
cpucores	0-3	CPU核心数。

- gs_all_control_group_info()

  描述：查看数据库内所有的控制组信息。函数返回信息具体的字段16.3.48 GS_ALL_CONTROL_GROUP_INFO字段。

  返回值类型：record

- gs_get_control_group_info()

  描述：查看所有的控制组信息。函数返回信息具体的字段16.3.53 GS_GET_CONTROL_GROUP_INFO字段。该函数需要sysadmin权限的用户才能够执行。

  返回值类型：record

- get_instr_workload_info(integer)

  描述：获取当前CN节点上事务量信息，事务时间信息。

  返回值类型：record

属性	属性值	描述
user_oid	10	用户id。
commit_counter	4	前端事务commit数量。
rollback_counter	1	前端事务rollback数量。
resp_min	949	前端事务最小响应时间（单位：微秒）。

属性	属性值	描述
resp_max	201891	前端事务最大响应时间（单位：微秒）。
resp_avg	43564	前端事务平均响应时间（单位：微秒）。
resp_total	217822	前端事务总响应时间（单位：微秒）。
bg_commit_counter	910	后端事务commit数量。
bg_rollback_counter	0	后端事务rollback数量。
bg_resp_min	97	后端事务最小响应时间（单位：微秒）。
bg_resp_max	678080687	后端事务最大响应时间（单位：微秒）。
bg_resp_avg	327847884	后端事务平均响应时间（单位：微秒）。
bg_resp_total	298341575300	后端事务总响应时间（单位：微秒）。

- pv_instance_time()
  描述：获取当前节点上各个关键阶段的时间消耗。
  返回值类型：record

Stat_name属性	属性值	描述
DB_TIME	1062385	所有线程端到端的墙上时间（WALL TIME）消耗总和（单位：微秒）。
CPU_TIME	311777	所有线程CPU时间消耗总和（单位：微秒）。
EXECUTION_TIME	380037	消耗在执行器上的时间总和（单位：微秒）。
PARSE_TIME	6033	消耗在SQL解析上的时间总和（单位：微秒）。
PLAN_TIME	173356	消耗在执行计划生成上的时间总和（单位：微秒）。
REWRITE_TIME	2274	消耗在查询重写上的时间总和（单位：微秒）。
PL_EXECUTION_TIME	0	消耗在plpgsql执行上的时间总和（单位：微秒）。

Stat_name属性	属性值	描述
PL_COMPILATION_TIME	557	消耗在SQL编译上的时间总和（单位：微秒）。
NET_SEND_TIME	1673	消耗在网络发送上的时间总和（单位：微秒）。
DATA_IO_TIME	426622	消耗在数据读写上的时间总和（单位：微秒）。

- DBE_PERF.get_global_instance_time()

  描述：提供整个集群各个关键阶段的时间消耗，仅在CN上支持查询。

  返回值类型：record

- get_instr_unique_sql()

  描述：获取当前节点的执行语句（归一化SQL）信息，查询该函数必须具有sysadmin权限或者MONADMIN权限。

  返回值类型：record

- get_instr_wait_event(integer)

  描述：获取当前节点event等待的统计信息。

  返回值类型：record

- get_instr_user_login()

  描述：获取当前节点的用户登录退出次数信息，查询该函数必须具有sysadmin或者MONADMIN权限。

  返回值类型：record

- get_instr_rt_percentile(integer)

  描述：获取CCN节点SQL 响应时间P80、P95分布信息，集群统一的信息在CCN节点上，其他节点查询为0。

  返回值类型：record

- get_node_stat_reset_time()

  描述：获取当前节点的统计信息重置（重启，主备倒换，数据库删除）时间。

  返回值类型：record

- gs_paxos_stat_replication()

  描述：在主机端查询备机信息。目前分布式不支持。

- get_paxos_replication_info()

  描述：查询主备复制信息。目前分布式不支持。

- gs_total_nodegroup_memory_detail

  描述：返回当前数据库Node group使用内存的信息，单位为MB。

  说明

  若GUC参数enable_memory_limit=off，该函数不能使用。

  返回值类型：setof record

表 7-76 返回值说明

名称	类型	描述
ngname	text	Node group名称。
memorytype	text	内存类型，包括以下几种： ● ng_total_memory：该Node group的总内存大小。 ● ng_used_memory：该Node group的实际使用内存大小。 ● ng_estimate_memory：该Node group的估算使用内存大小。 ● ng_foreignrp_memsize：该Node group的外部资源池的总内存大小。 ● ng_foreignrp_usedsize：该Node group的外部资源池实际使用内存大小。 ● ng_foreignrp_peaksize：该Node group的外部资源池使用内存的峰值。 ● ng_foreignrp_mempct：该Node group的外部资源池占该Node group总内存大小的百分比。 ● ng_foreignrp_estmsize：该Node group的外部资源池估算使用内存大小。
memorymbytes	integer	内存类型分配内存的大小。

- gs_io_wait_status()

  描述：返回当前节点I/O管控的实时统计信息。

  返回值类型：setof record

名称	类型	描述
node_name	text	节点名称。
device_name	text	节点挂载的数据磁盘名称。
read_per_second	float	读完成每秒次数。
write_per_second	float	写完成每秒次数。
write_ratio	float	写磁盘占总的I/O使用的比例。
io_util	float	每秒I/O所占CPU总时间的百分比。
total_io_util	integer	过去三次I/O所占CPU总时间的等级（取值为0~6）。
tick_count	integer	更新磁盘I/O信息的周期，固定为1秒，每次读取数据前都会被清零。

名称	类型	描述
io_wait_list_len	integer	I/O请求线程等待队列的大小，若为0，则表示当前没有I/O被管控。

- gs_get_shared_memctx_detail(text)

  描述：返回指定内存上下文上的内存申请的详细信息，包含每一处内存申请所在的文件、行号和大小（同一文件同一行大小会做累加）。只支持查询通过pg_shared_memory_detail视图查询出来的内存上下文，入参为内存上下文名称（即pg_shared_memory_detail返回结果的contextname列）。查询该函数必须具有sysadmin权限或者MONADMIN权限。

  返回值类型：setof record

名称	类型	描述
file	text	申请内存所在文件的文件名。
line	int8	申请内存所在文件的代码行号。
size	int8	申请的内存大小，同一文件同一行多次申请会做累加。

- gs_get_session_memctx_detail(text)

  描述：返回指定内存上下文上的内存申请的详细信息，包含每一处内存申请所在的文件、行号和大小（同一文件同一行大小会做累加）。仅在线程池模式下生效。且只支持查询通过pv_session_memory_context视图查询出来的内存上下文，入参为内存上下文名称（即pv_session_memory_context返回结果的contextname列）。查询该函数必须具有sysadmin权限或者MONADMIN权限。

  返回值类型：setof record

名称	类型	描述
file	text	申请内存所在文件的文件名。
line	int8	申请内存所在文件的代码行号。
size	int8	申请的内存大小，单位为byte，同一文件同一行多次申请会做累加。

- gs_get_thread_memctx_detail(tid,text)

  描述：返回指定内存上下文上的内存申请的详细信息，包含每一处内存申请所在的文件、行号和大小（同一文件同一行大小会做累加）。只支持查询通过pv_thread_memory_context视图查询出来的内存上下文，第一个入参为线程id（即pv_thread_memory_context返回数据的tid列），第二个参数为内存上下文名称（即pv_thread_memory_context返回数据的contextname列）。查询该函数必须具有sysadmin权限或者MONADMIN权限。

  返回值类型：setof record

名称	类型	描述
file	text	申请内存所在文件的文件名。

名称	类型	描述
line	int8	申请内存所在文件的代码行号。
size	int8	申请的内存大小，单位为byte，同一文件同一行多次申请会做累加。

- gs_get_history_memory_detail(cstring)

  描述：查询历史内存快照信息，入参类型为cstring，取值为NULL或内存快照log文件名称：

  a. 若入参为NULL，则显示当前节点所有的内存快照log文件列表。
  b. 若入参为a查询到的列表中的内存快照log名称，则显示该log文件记录的内存快照详细信息。
  c. 若输入其他入参，则会提示入参错误或打开文件失败。

  查询该函数必须具有sysadmin权限或者MONADMIN权限。

  返回值类型：text

名称	类型	描述
memory_info	text	内存信息，如果函数入参为NULL，该列显示内存快照文件列表信息；入参为内存快照文件名称，则显示该文件的具体内容。

- gs_stat_get_hotkeys_info()

  说明

  若GUC参数enable_hotkeys_collection = off, gs_stat_get_hotkeys_info、global_stat_get_hotkeys_info函数和global_stat_hotkeys_info视图无法正常查询。不影响gs_stat_clean_hotkeys和global_stat_clean_hotkeys清理接口的正常使用。

  描述：获取当前节点上热点key的统计情况。

  返回值类型：record

```
openGauss=# select * from gs_stat_get_hotkeys_info() order by count, hash_value;
 database_name | schema_name |     table_name      | key_value | hash_value | count
---------------+-------------+---------------------+-----------+------------+-------
 regression    | public      | hotkey_single_col | {22}      | 1858004829 |     2
 regression    | public      | hotkey_single_col | {11}      | 2011968649 |     2
(2 rows)
```

表1 返回值说明

名称	类型	描述
database_name	text	热点key所在database名称。
schema_name	text	热点key所在schema名称。
table_name	text	热点key所在table名称。
key_value	text	热点key的value。

名称	类型	描述
hash_value	bigint	热点key在数据库中的哈希值，如果是List/Range分布表，该字段为0。
count	bigint	热点key被访问频次。

- gs_stat_clean_hotkeys()

  说明

  - 热点key检测是针对大并发大流量场景设计的特性，访问几次的场景，查询清理会存在一定误差。
  - 清理接口的设计上，只会清理LRU队列中的统计数据，而不会清理FIFO中的历史数据。因此如果清理完后，再访问一次FIFO中存在的历史键值，仍会被当做热点key处理。global_stat_clean_hotkeys同理。

  描述：清理当前节点上热点key的统计信息。

  返回值：boolean

```
openGauss=# select * from gs_stat_clean_hotkeys();
 gs_stat_clean_hotkeys
-----------------------
 t
(1 row)
```

- global_stat_get_hotkeys_info()

  说明

  执行业务过程中，执行select * from global_stat_hotkeys_info minus select * from global_stat_get_hotkeys_info();因为存在时间差，可能出现不为0的情况。

  描述：获取整个集群中热点key的统计情况。

  返回值类型：record

```
openGauss=# select * from global_stat_get_hotkeys_info() order by count, hash_value;
 database_name | schema_name |    table_name     | key_value | hash_value | count
---------------+-------------+-------------------+-----------+------------+-------
 regression    | public      | hotkey_single_col | {22}      | 1858004829 |     2
 regression    | public      | hotkey_single_col | {11}      | 2011968649 |     2
(2 rows)
```

- global_stat_clean_hotkeys()

  描述：清理整个集群中热点key的统计信息。

  返回值：boolean

```
openGauss=# select * from global_stat_clean_hotkeys();
 global_stat_clean_hotkeys
-----------------------
 t
(1 row)
```

- global_comm_get_recv_stream()

  描述：获取所有DN节点上所有的通信库接收流状态。函数返回信息具体字段参考**PG_COMM_RECV_STREAM**字段。

  返回值类型：record

- global_comm_get_send_stream()

  描述：获取所有DN节点上所有的通信库发送流状态。函数返回信息具体字段参考**PG_COMM_SEND_STREAM**字段。

  返回值类型：record

- global_comm_get_status()

  描述：获取所有DN节点的通信库状态。函数返回信息具体字段参考**PG_COMM_STATUS**字段。

  返回值类型：record

- global_comm_client_info()

  描述：获取全局节点活跃的客户端连接信息。函数返回信息具体字段参考**COMM_CLIENT_INFO**字段。

  返回类型：record

- global_comm_get_client_info()

  描述：获取全局节点客户端连接信息。函数返回信息具体字段参考**COMM_CLIENT_INFO**字段。

  返回类型：record

- pgxc_stat_activity()

  描述：显示当前集群下所有CN的当前用户查询相关的信息，该函数仅具有sysadmin或者MONADMIN权限的用户可以执行，普通用户可查看本用户的相关信息。

  返回值类型：record

名称	类型	描述
coorname	text	当前集群下的CN名称。
datid	oid	用户会话在后台连接到的数据库OID。
datname	text	用户会话在后台连接到的数据库名称。
pid	bigint	后台线程ID。
sessionid	bigint	会话ID。
usesysid	oid	登录该后台的用户OID。
usename	text	登录该后台的用户名。
application_name	text	连接到该后台的应用名。
client_addr	inet	连接到该后台的客户端的IP地址。如果此字段是null，它表明通过服务器机器上UNIX套接字连接客户端或者这是内部进程，如autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过client_addr的反向DNS查找得到。这个字段只有在启动log_hostname且使用IP连接时才非空。
client_port	integer	客户端用于与后台通讯的TCP端口号，如果使用UNIX套接字，则为-1。
backend_start	timestamp with time zone	该过程开始的时间，即当客户端连接服务器的时间。

名称	类型	描述
xact_start	timestamp with time zone	启动当前事务的时间，如果没有事务是活跃的，则为null。如果当前查询是首个事务，则这列等同于query_start列。
query_start	timestamp with time zone	开始当前活跃查询的时间，如果state的值不是active，则这个值是上一个查询的开始时间。
state_change	timestamp with time zone	上次状态改变的时间。
waiting	boolean	如果后台当前正等待锁则为true。
enqueue	text	语句当前排队状态。可能值是： ● waiting in queue：表示语句在排队中。 ● 空：表示语句正在运行。

名称	类型	描述
state	text	该后台当前总体状态。可能值是： ● active：后台正在执行一个查询。 ● idle：后台正在等待一个新的客户端命令。 ● idle in transaction：后台在事务中，但事务中没有语句在执行。 ● idle in transaction (aborted)：后台在事务中，但事务中有语句执行失败。 ● fastpath function call：后台正在执行一个fast-path函数。 ● disabled：如果后台禁用track_activities，则报告这个状态。 **说明** 只有系统管理员能查看到自己账户所对应的会话状态。其他账户的state信息为空。例如以judy用户连接数据库后，在pgxc_stat_activity中查看到的普通用户joe及初始用户omm的state信息为空： SELECT datname, usename, usesysid, state,pid FROM pgxc_stat_activity; datname | usename | usesysid | state | pid ----------+---------+----------+--------+----------------- postgres | omm | 10 | | 139968752121616 postgres | omm | 10 | | 139968903116560 db_tpcds | judy | 16398 | active | 139968391403280 postgres | omm | 10 | | 139968643069712 postgres | omm | 10 | | 139968680818448 postgres | joe | 16390 | | 139968563377936 (6 rows)
resource_pool	name	用户使用的资源池。
query_id	bigint	查询语句的ID。
query	text	该后台的最新查询。如果state状态是active（活跃的），此字段显示当前正在执行的查询。所有其他情况表示上一个查询。
global_sessionid	text	全局会话ID。
unique_sql_id	bigint	语句的unique sql id。
trace_id	text	驱动传入的trace id，与应用的一次请求相关联。

- pgxc_stat_activity_with_conninfo()

  描述：显示当前集群下所有CN的当前用户查询相关的信息，返回值定义参考pgxc_stat_activity视图。该函数仅具有sysadmin或者MONADMIN权限的用户可以执行，普通用户只能查看本用户相关的信息。

  返回值类型：record

- pgxc_stat_all_tables()

  描述：显示各节点数据中每个表（包括TOAST表）的一行的统计信息，该函数仅具有sysadmin或者MONADMIN权限的用户可以执行。

  返回值类型：record

- pgxc_get_thread_wait_status()

  描述：查看集群各个节点上所有SQL语句产生的线程之间的调用层次关系，以及各个线程的阻塞等待状态。

  返回值类型：record

- pv_session_memory

  描述：统计Session级别的内存使用情况，包含执行作业在数据节点上gaussdb线程和Stream线程分配的所有内存。

  说明

  若GUC参数enable_memory_limit=off，该函数不能使用。

  返回值类型：record

  表 7-77 返回值说明

名称	类型	描述
sessid	text	线程启动时间+线程标识。
init_mem	integer	当前正在执行作业进入执行器前已分配的内存，单位MB。
used_mem	integer	当前正在执行作业已分配的内存，单位MB。
peak_mem	integer	当前正在执行作业已分配的内存峰值，单位MB。

- dbe_perf.gs_stat_activity_timeout(int)

  描述：获取当前节点上执行时间超过超时阈值的查询作业信息。需要GUC参数track_activities设置为on才能正确返回结果。超时阈值的取值范围是0~2147483。

  返回值类型：setof record

名称	类型	描述
database	name	用户会话连接的数据库名称。
pid	bigint	后台线程ID。
sessionid	bigint	会话ID。
usesysid	oid	登录该后台的用户OID。

名称	类型	描述
application_name	text	连接到该后台的应用名。
query	text	该后台正在执行的查询。
xact_start	timestamptz	启动当前事务的时间。
query_start	timestamptz	开始当前查询的时间。如果是存储过程、函数、PACKAGE，则查询的是第一个查询时间，不会随着存储过程内语句运行而改变。
query_id	bigint	查询语句ID。

- dbe_perf.global_stat_activity_timeout(int)

  描述：获取当前系统（所有CN）中执行时间超过超时阈值的查询作业信息。需要GUC参数track_activities设置为on才能正确返回结果。超时阈值的取值范围是0~2147483。

  返回值类型：setof record

名称	类型	描述
nodename	text	用户会话连接的coordinate node的名称。
database	name	用户会话连接的数据库名称。
pid	bigint	后台线程ID。
sessionid	bigint	会话ID。
usesysid	oid	登录该后台的用户OID。
application_name	text	连接到该后台的应用名。
query	text	该后台正在执行的查询。
xact_start	timestamptz	启动当前事务的时间。
query_start	timestamptz	开始当前查询的时间。
query_id	bigint	查询语句ID。

- DBE_PERF.get_global_active_session()

  描述：显示所有节点上的ACTIVE SESSION PROFILE内存中的样本的汇总。

  返回值类型：record

- DBE_PERF.get_global_os_runtime()

  描述：显示当前操作系统运行的状态信息，仅在CN上支持查询。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_os_threads()

  描述：提供整个集群中所有正常节点下的线程状态信息，仅在CN上支持查询。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_os_threads()

  描述：提供整个集群中所有正常节点下的线程状态信息，仅在CN上支持查询。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_workload_sql_count()

  描述：提供整个集群中不同负载SELECT，UPDATE，INSERT，DELETE，DDL，DML，DCL计数信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_workload_sql_elapse_time()

  描述：提供整个集群中不同负载SELECT，UPDATE，INSERT，DELETE，响应时间信息（TOTAL、AVG、MIN、MAX）。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_workload_transaction()

  描述：获取集群内所有节点上的事务量信息，事务时间信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_session_stat()

  描述：获取集群内所有节点上的会话状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

  说明

  状态信息有如下17项，commit、rollback、sql、table_scan、blocks_fetched、physical_read_operation、shared_blocks_dirtied、local_blocks_dirtied、shared_blocks_read、local_blocks_read、blocks_read_time、blocks_write_time、sort_imemory、sort_idisk、cu_mem_hit、cu_hdd_sync_read、cu_hdd_asyread。

- DBE_PERF.get_global_session_time()

  描述：提供整个集群各节点各个关键阶段的时间消耗。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_session_memory()

  描述：汇聚各节点的Session级别的内存使用情况，包含执行作业在数据节点上gaussdb线程和Stream线程分配的所有内存，单位为MB。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_session_memory_detail()

  描述：汇聚各节点的线程的内存使用情况，以MemoryContext节点来统计。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_session_stat_activity()

  描述：汇聚集群内各节点上正在运行的线程相关的信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_thread_wait_status()

  描述：汇聚所有节点上工作线程（backend thread）以及辅助线程（auxiliary thread）的阻塞等待情况。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_operator_history_table()

  描述：汇聚当前用户所有CN上执行作业结束后的算子相关记录（持久化）。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_operator_history()

  描述：汇聚当前用户所有CN上执行作业结束后的算子相关记录。集群创建后的默认情况下，查询该函数必须具有MONADMIN和sysadmin权限。

  返回值类型：record

- DBE_PERF.get_global_operator_runtime()

  描述：汇聚当前用户所有CN上执行作业实时的算子相关记录。集群创建后的默认情况下，查询该函数必须具有MONADMIN和sysadmin权限。

  返回值类型：record

- DBE_PERF.get_global_statement_complex_history()

  描述：汇聚当前用户所有CN节点上复杂查询的历史记录。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_statement_complex_history_table()

  描述：汇聚当前用户所有CN节点上复杂查询的历史记录（持久化）。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_statement_complex_runtime()

  描述：汇聚当前用户所有CN节点上复杂查询的实时信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN和sysadmin权限。

  返回值类型：record

- DBE_PERF.get_global_memory_node_detail()

  描述：汇聚某个数据库在所有节点上的内存使用情况。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_shared_memory_detail()

  描述：汇聚所有节点已产生的共享内存上下文的使用信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_comm_delay()

描述：汇聚所有DN节点的通信库时延状态。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

返回值类型：record

- DBE_PERF.get_global_comm_recv_stream()

  描述：汇聚所有DN节点的通信库接收流状态。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_comm_send_stream()

  描述：汇聚所有DN节点的通信库发送流状态。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_comm_status()

  描述：汇聚所有DN节点的通信库状态。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_statio_all_indexes

  描述：汇聚所有节点当前数据库中的索引信息及I/O统计量。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_local_toastname_and_toastindexname()

  描述：提供本地toast表的name和index和其关联表的对应关系。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_statio_all_indexes

  描述：统计所有节点当前数据库中的每个索引行，显示特定索引的I/O的统计。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_statio_all_sequences

  描述：提供命名空间中所有sequences的I/O状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_statio_all_tables

  描述：汇聚各节点的数据库中每个表I/O的统计。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_statio_all_tables

  描述：统计集群内数据库中每个表I/O的统计。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_local_toast_relation()

  描述：提供本地toast表的name和其关联表的对应关系。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_statio_sys_indexes()

  描述：汇聚各节点的命名空间中所有系统表索引的I/O状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_statio_sys_indexes()

  描述：统计各节点的命名空间中所有系统表索引的I/O状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_statio_sys_sequences()

  描述：提供命名空间中所有系统表为sequences的I/O状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_statio_sys_tables()

  描述：提供各节点的命名空间中所有系统表的I/O状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_statio_sys_tables()

  描述：集群内汇聚命名空间中所有系统表的I/O状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_statio_user_indexes()

  描述：各节点的命名空间中所有用户关系表索引的I/O状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_statio_user_indexes()

  描述：集群内汇聚命名空间中所有用户关系表索引的I/O状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_statio_user_sequences()

  描述：显示各节点的命名空间中所有用户的sequences的I/O状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_statio_user_tables()

  描述：显示各节点的命名空间中所有用户关系表的I/O状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_statio_user_tables()

  描述：集群内汇聚命名空间中所有用户关系表的I/O状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_stat_db_cu()

  描述：视图查询集群各个节点，每个数据库的CU命中情况。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

返回值类型：record

- DBE_PERF.get_global_dn_stat_all_tables()

  描述：汇聚DN节点数据库中每个表的统计信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_cn_stat_all_tables()

  描述：汇聚CN节点数据库中每个表的统计信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_dn_stat_all_tables()

  描述：统计DN节点数据库中每个表的统计信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_cn_stat_all_tables()

  描述：统计CN节点数据库中每个表的统计信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_all_indexes()

  描述：汇聚所有节点数据库中每个索引的统计信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_all_indexes()

  描述：统计所有节点数据库中每个索引的统计信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_sys_tables()

  描述：汇聚各节点pg_catalog、information_schema模式的所有命名空间中系统表的统计信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_sys_tables()

  描述：统计各节点pg_catalog、information_schema模式的所有命名空间中系统表的统计信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_sys_indexes()

  描述：汇聚各节点pg_catalog、information_schema模式中所有系统表的索引状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_sys_indexes()

  描述：统计各节点pg_catalog、information_schema模式中所有系统表的索引状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_user_tables()

  描述：汇聚所有命名空间中用户自定义普通表的状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_user_tables()

  描述：统计所有命名空间中用户自定义普通表的状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_user_indexes()

  描述：汇聚所有数据库中用户自定义普通表的索引状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_user_indexes()

  描述：统计所有数据库中用户自定义普通表的索引状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_database()

  描述：汇聚所有节点数据库统计信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_database_conflicts()

  描述：统计所有节点数据库统计信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_xact_all_tables()

  描述：汇聚命名空间中所有普通表和toast表的事务状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_xact_all_tables()

  描述：统计命名空间中所有普通表和toast表的事务状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_xact_sys_tables()

  描述：汇聚所有节点命名空间中系统表的事务状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_xact_sys_tables()

  描述：统计所有节点命名空间中系统表的事务状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_xact_user_tables()

  描述：汇聚所有节点命名空间中用户表的事务状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

返回值类型：record

- DBE_PERF.get_summary_stat_xact_user_tables()

  描述：统计所有节点命名空间中用户表的事务状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_user_functions()

  描述：汇聚所有节点命名空间中用户定义函数的事务状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_xact_user_functions()

  描述：统计所有节点命名空间中用户定义函数的事务状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_stat_bad_block()

  描述：汇聚所有节点表、索引等文件的读取失败信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_file_redo_iostat()

  描述：统计所有节点表、索引等文件的读取失败信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_file_iostat()

  描述：汇聚所有节点数据文件I/O的统计。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_locks()

  描述：汇聚所有节点的锁信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_replication_slots()

  描述：汇聚所有节点上逻辑复制信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_bgwriter_stat()

  描述：汇聚所有节点后端写进程活动的统计信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_replication_stat()

  描述：汇聚各节点日志同步状态信息，如发起端发送日志位置，收端接收日志位置等。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_pooler_status()

描述：汇聚所有CN节点的pooler中的缓存连接状态。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

返回值类型：record

- DBE_PERF.get_global_transactions_running_xacts()

  描述：汇聚各节点运行事务的信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_transactions_running_xacts()

  描述：统计各节点运行事务的信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_transactions_prepared_xacts()

  描述：汇聚各节点当前准备好进行两阶段提交的事务的信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_transactions_prepared_xacts()

  描述：统计各节点当前准备好进行两阶段提交的事务的信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_statement()

  描述：汇聚各节点历史执行语句状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN和sysadmin权限。

  返回值类型：record

- DBE_PERF.get_global_statement_count()

  描述：汇聚各节点SELECT，UPDATE，INSERT，DELETE，响应时间信息（TOTAL、AVG、MIN、MAX）。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_config_settings()

  描述：汇聚各节点GUC参数配置信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_global_wait_events()

  描述：汇聚各节点wait events状态信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_statement_responsetime_percentile()

  描述：获取集群SQL响应时间P80，P95分布信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.get_summary_user_login()

  描述：统计集群各节点用户登录退出次数信息。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

返回值类型：record

- DBE_PERF.get_global_record_reset_time()

  描述：汇聚集群统计信息重置（重启，主备倒换，数据库删除）时间。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

  返回值类型：record

- DBE_PERF.track_memory_context(context_list text)

  描述：设置需要统计内存申请详细信息的内存上下文。入参为内存上下文的名称，使用“，”分隔，如“ThreadTopMemoryContext, SessionCacheMemoryContext”，注意该内存上下文名称是上下文敏感的。此外，单个内存上下文的长度为63，超过的部分会被截断。而且一次能够统计的内存上下文上限为16个，设置超过16个内存上下文会设置失败。每一次调用该函数都会将上次统计的结果清空，当入参指定为“”时，表示取消该统计功能。只有初始用户或者具有MONADMIN权限的用户可以执行该函数。

  返回值类型：boolean

- DBE_PERF.track_memory_context_detail()

  描述：获取DBE_PERF.track_memory_context函数指定的内存上下文的内存申请详细信息。返回值的定义见视图DBE_PERF.track_memory_context_detail。只有初始用户或者具有MONADMIN权限的用户可以执行该函数。

  返回值类型：record

- DBE_PERF.global_io_wait_info()

  描述：查询所有CN和DN节点上的I/O管控的实时统计信息。

  返回值类型：record

- pg_stat_get_mem_mbytes_reserved(tid)

  描述：统计资源管理相关变量值，仅用于定位问题使用。

  参数：线程id。

  返回值类型：text

- pg_stat_get_file_stat()

  描述：通过对数据文件I/O的统计，反映数据的I/O性能，用以发现I/O操作异常等性能问题。

  返回值类型：record

- pg_stat_get_redo_stat()

  描述：用于统计会话线程日志回放情况。

  返回值类型：record

- pg_stat_get_status(int8)

  描述：可以检测当前实例中工作线程（backend thread）以及辅助线程（auxiliary thread）的阻塞等待情况。

  返回值类型：record

- get_local_rel_iostat()

  描述：查询当前节点的数据文件I/O状态累计值。

  返回值类型：record

- DBE_PERF.get_global_rel_iostat()

  描述：汇聚所有节点数据文件I/O的统计。集群创建后的默认情况下，查询该函数必须具有MONADMIN权限。

返回值类型：record

- pg_catalog.plancache_status()

  描述：显示在当前节点上的全局计划缓存的状态信息。函数返回信息和**GLOBAL_PLANCACHE_STATUS**一致。

  返回值类型：record

- DBE_PERF.global_plancache_status()

  描述：显示在所有节点上的全局计划缓存的状态信息。函数返回信息见字段**GLOBAL_PLANCACHE_STATUS**。

  返回值类型：record

- pg_catalog.prepare_statement_status()（废弃）

  描述：显示在当前节点上的prepare statement状态信息。函数返回信息和**GLOBAL_PREPARE_STATEMENT_STATUS（废弃）**一致。

  返回值类型：record

- DBE_PERF.global_prepare_statement_status()（废弃）

  描述：显示在所有节点上的prepare statement的状态信息。函数返回信息见字段**GLOBAL_PREPARE_STATEMENT_STATUS（废弃）**。

  返回值类型：record

- DBE_PERF.global_threadpool_status()

  描述：显示在所有节点上的线程池中工作线程及会话的状态信息。函数返回信息见字段**18.7.14-表 GLOBAL_THREADPOOL_STATUS**。

  返回值类型：record

- comm_check_connection_status

  描述：返回该CN和所有活跃节点（CN和主DN）的连接情况。该函数仅支持在CN上查询，普通用户可使用。

  参数：nan

  返回值类型：node_name text, remote_name text, remote_host text, remote_port integer, is_connected boolean, no_error_occur boolean

- DBE_PERF.global_comm_check_connection_status

  描述：返回所有CN和所有活跃节点（CN和主DN）的连接情况。该函数仅支持在CN上查询，权限控制继承DBE_PERF schema。

  参数：nan

  返回值类型：node_name text, remote_name text, remote_host text, remote_port integer, is_connected boolean, no_error_occur boolean

- remote_candidate_stat()

  描述：显示本实例的候选buffer链中页面个数，buffer淘汰信息，包含normal buffer pool和segment buffer pool。

  返回值类型：record

表 7-78 remote_candidate_stat 参数说明

名称	类型	描述
node_name	text	节点名称。

名称	类型	描述
candidate_slots	integer	当前Normal Buffer Pool候选buffer链中页面个数。
get_buf_from_list	bigint	Normal Buffer Pool，buffer淘汰从候选buffer链中获取页面的次数。
get_buf_clock_sweep	bigint	Normal Buffer Pool，buffer淘汰从原淘汰方案中获取页面的次数。
seg_candidate_slots	integer	当前Segment Buffer Pool候选buffer链中页面个数。
seg_get_buf_from_list	bigint	Segment Buffer Pool，buffer淘汰从候选buffer链中获取页面的次数。
seg_get_buf_clock_sweep	bigint	Segment Buffer Pool，buffer淘汰从原淘汰方案中获取页面的次数。

- remote_ckpt_stat()

  描述：用于显示整个集群所有实例的检查点信息和各类日志刷页情况（本节点除外、DN上不可使用）。

  返回值类型：record

  表 7-79 remote_ckpt_stat 参数说明

参数	类型	描述
node_name	text	实例名称。
ckpt_redo_point	text	当前实例的检查点。
ckpt_clog_flush_num	int8	从启动到当前时间clog刷盘页面数。
ckpt_csnlog_flush_num	int8	从启动到当前时间csnlog刷盘页面数。
ckpt_multixact_flush_num	int8	从启动到当前时间multixact刷盘页面数。
ckpt_predicate_flush_num	int8	从启动到当前时间predicate刷盘页面数。
ckpt_twophase_flush_num	int8	从启动到当前时间twophase刷盘页面数。

- remote_double_write_stat()

  描述：显示整个集群所有实例的双写文件的情况(本节点除外、DN上不可使用)。

  返回值类型：record

表 7-80 remote_double_write_stat 参数说明

参数	类型	描述
node_name	text	实例名称。
curr_dwn	int8	当前双写文件的序列号。
curr_start_page	int8	当前双写文件恢复起始页面。
file_trunc_num	int8	当前双写文件复用的次数。
file_reset_num	int8	当前双写文件写满后发生重置的次数。
total_writes	int8	当前双写文件总的I/O次数。
low_threshold_writes	int8	低效率写双写文件的I/O次数（一次I/O刷页数量少于16页面）。
high_threshold_writes	int8	高效率写双写文件的I/O次数（一次I/O刷页数量多于一批，421个页面）。
total_pages	int8	当前刷页到双写文件区的总的页面个数。
low_threshold_pages	int8	低效率刷页的页面个数。
high_threshold_pages	int8	高效率刷页的页面个数。
file_id	int8	当前双写文件的id号。

- remote_single_flush_dw_stat()

  描述：显示整个集群所有实例的单页面淘汰双写文件的情况（本节点除外、DN上不可使用）。

  返回值类型：record

  表 7-81 remote_single_flush_dw_stat 参数说明

参数	类型	描述
node_name	text	实例名称。
curr_dwn	integer	当前双写文件的序列号。
curr_start_page	integer	当前双写文件start位置。
total_writes	bigint	当前双写文件总计写数据页面个数。
file_trunc_num	bigint	当前双写文件复用的次数。
file_reset_num	bigint	当前双写文件写满后发生重置的次数。

- remote_pagewriter_stat()

  描述：显示整个集群所有实例的刷页信息和检查点信息（本节点除外、DN上不可使用）。

返回值类型：record

表 7-82 remote_pagewriter_stat 参数说明

参数	类型	描述
node_name	text	实例名称。
pgwr_actual_flush_total_num	int8	从启动到当前时间总计刷脏页数量。
pgwr_last_flush_num	int4	上一批刷脏页数量。
remain_dirty_page_num	int8	当前预计还剩余多少脏页。
queue_head_page_rec_lsn	text	当前实例的脏页队列第一个脏页的recovery_lsn。
queue_rec_lsn	text	当前实例的脏页队列的recovery_lsn。
current_xlog_insert_lsn	text	当前实例xLog写入的位置。
ckpt_redo_point	text	当前实例的检查点。

- remote_recovery_status()
  描述：显示关于主机和备机的日志流控信息（本节点除外、DN上不可使用）。
  返回值类型：record

  表 7-83 remote_recovery_status 参数说明

参数	类型	描述
node_name	text	节点的名称，包含主机和备机。
standby_node_name	text	备机名称。
source_ip	text	主机的IP地址。
source_port	int4	主机的端口号。
dest_ip	text	备机的IP地址。
dest_port	int4	备机的端口号。
current_rto	int8	备机当前的日志流控时间，单位秒。
target_rto	int8	备机通过GUC参数设置的预期流控时间，单位秒。
current_sleep_time	int8	为了达到这个预期主机所需要的睡眠时间，单位微秒。

- remote_rto_stat()
  描述：显示关于主机和备机的日志流控信息（本节点除外、DN上不可使用）。
  返回值类型：record

  表 7-84 remote_rto_stat 参数说明

参数	类型	描述
node_name	text	节点的名称，包含主机和备机。
rto_info	text	流控的信息，包含了备机当前的日志流控时间（单位：秒），备机通过GUC参数设置的预期流控时间（单位：秒），为了达到这个预期主机所需要的睡眠时间（单位：微秒）。

- remote_redo_stat()
  描述：显示整个集群所有实例的日志回放情况（本节点除外、DN上不可使用）。
  返回值类型：record

  表 7-85 remote_redo_stat 参数说明

参数	类型	描述
node_name	text	实例名称。
redo_start_ptr	int8	当前实例日志回放的起始点。
redo_start_time	int8	当前实例日志回放的起始UTC时间。
redo_done_time	int8	当前实例日志回放的结束UTC时间。
curr_time	int8	当前实例的当前UTC时间。
min_recovery_point	int8	当前实例日志的最小一致性点位置。
read_ptr	int8	当前实例日志的读取位置。
last_replayed_read_ptr	int8	当前实例的日志回放位置。
recovery_done_ptr	int8	当前实例启动完成时的回放位置。
read_xlog_io_counter	int8	当前实例读取回放日志的I/O次数计数。
read_xlog_io_total_dur	int8	当前实例读取回放日志的I/O总时延。
read_data_io_counter	int8	当前实例回放过程中读取数据页面的I/O次数计数。

参数	类型	描述
read_data_io_total_dur	int8	当前实例回放过程中读取数据页面的I/O总时延。
write_data_io_counter	int8	当前实例回放过程中写数据页面的I/O次数计数。
write_data_io_total_dur	int8	当前实例回放过程中写数据页面的I/O总时延。
process_pending_counter	int8	当前实例回放过程中日志分发线程的同步次数计数。
process_pending_total_dur	int8	当前实例回放过程中日志分发线程的同步总时延。
apply_counter	int8	当前实例回放过程中回放线程的同步次数计数。
apply_total_dur	int8	当前实例回放过程中回放线程的同步总时延。
speed	int8	当前实例日志回放速率。
local_max_ptr	int8	当前实例启动成功后本地收到的回放日志的最大值。
primary_flush_ptr	int8	主机落盘日志的位置。
worker_info	text	当前实例回放线程信息，若没有开并行回放则该值为空。

- pgxc_gtm_snapshot_status()

  描述：用于查看当前GTM上事务信息，仅在GTM模式下支持本系统函数，GTM-LITE和GTM-FREE模式下不支持。

  返回值类型：record

  函数返回字段描述如下：

  表 7-86 pgxc_gtm_snapshot_status 返回参数说明

名称	类型	描述
xmin	xid	仍在运行的最小事务号。
xmax	xid	已完成的所有事务号中最大事务号的下一个事务号。
csn	integer	待提交事务的序列号。
oldestxmin	xid	当前最早的活跃事务在其取快照时，所有运行事务号最小的事务。
xcnt	integer	当前活跃的事务个数。

名称	类型	描述
running_xids	text	当前活跃的事务号。

- pg_stat_get_partition_tuples_hot_updated(oid)

  描述：返回指定分区id的分区热更新元组数的统计。

  参数：oid

  返回值类型：bigint

- pv_os_run_info()

  描述：显示当前操作系统运行的状态信息，具体字段信息参考**PV_OS_RUN_INFO**。

  参数：nan

  返回值类型：setof record

- pv_session_stat()

  描述：以会话线程或AutoVacuum线程为单位，统计会话状态信息，具体字段信息参考**PV_SESSION_STAT**。

  参数：nan

  返回值类型：setof record

- pv_session_time()

  描述：用于统计会话线程的运行时间信息，及各执行阶段所消耗时间，具体字段信息参考**PV_SESSION_TIME**。

  参数：nan

  返回值类型：setof record

- pg_stat_get_db_temp_bytes()

  描述：用于统计通过数据库查询写入临时文件的数据总量。计算所有临时文件，不论为什么创建临时文件，而且不管log_temp_files设置。

  参数：oid

  返回值类型：bigint

- pg_stat_get_db_temp_files()

  描述：通过数据库查询创建的临时文件数量。计算所有临时文件，不论为什么创建临时文件（比如排序或者哈希），而且不管log_temp_files设置。

  参数：oid

  返回值类型：bigint

- local_redo_time_count()

  描述：返回本节点各个回放线程的各个流程的耗时统计（仅在备机上有有效数据）。

  返回值如下：

  表 7-87 local_redo_time_count 返回参数说明

字段名	描述
thread_name	线程名字。

字段名	描述
step1_total	step1的总时间，每个线程对应的流程如下： ● 极致RTO： batch redo：从队列中获取一条日志。 redo manager：从队列中获取一条日志。 redo worker：从队列中获取一条日志。 trxn manager：从队列中读取一条日志。 trxn worker：从队列中读取一条日志。 read worker：从文件中读取一次xLog page（整体）。 read page worker：从队列中获取一个日志。 startup：从队列中获取一个日志。 ● 并行回放： page redo：从队列中获取一条日志。 startup：读取一条日志。
step1_count	step1的统计次数。
step2_total	step2的总时间，每个线程对应的流程如下： ● 极致RTO： batch redo：处理日志（整体）。 redo manager：处理日志（整体）。 redo worker：处理日志（整体）。 trxn manager：处理日志（整体）。 trxn worker：处理日志（整体）。 redo worker：读取xLog page耗时。 read page worker：生成和发送lsn forwarder。 startup：check stop(是否回放到指定位置）。 ● 并行回放： page redo：处理日志（整体）。 startup：check stop（是否回放到指定位置）。
step2_count	step2的统计次数。

字段名	描述
step3_total	step3的总时间，每个线程对应的流程如下： ● 极致RTO： batch redo：更新standbystate。 redo manager：数据日志处理。 redo worker：回放page页日志（整体）。 trxn manager：更新flush lsn。 trxn worker：回放日志处理。 redo worker：推进xLog segment。 read page worker：获取一个新的item。 startup：redo delay（延迟回放特性等待时间）。 ● 并行回放： page redo：更新standbystate。 startup：redo delay（延迟回放特性等待时间）。
step3_count	step3的统计次数。
step4_total	step4的总时间，每个线程对应的流程如下： ● 极致RTO： batch redo：解析xLog。 redo manager：DDL处理。 redo worker：读取数据page页。 trxn manager：同步等待时间。 trxn worker：更新本线程lsn。 read page worker：将日志放入分发线程。 startup：分发（整体）。 ● 并行回放： page redo：undo日志回放。 startup：分发（整体）。
step4_count	step4的统计次数。

字段名	描述
step5_total	step5的总时间，每个线程对应的流程如下： ● 极致RTO： batch redo：分发给redo manager。 redo manager：分发给redo worker。 redo worker：回放数据page页的日志。 trxn manager：分发给trxn worker。 trxn worker：强同步wait时间。 read page worker：更新本线程lsn。 startup：日志decode。 ● 并行回放： page redo：sharetrxn日志回放。 startup：日志回放。
step5_count	step5的统计次数。
step6_total	step6的总时间，每个线程对应的流程如下： ● 极致RTO： redo worker：回放非数据页page日志。 trxn manager：全局lsn更新。 read page worker：日志crc校验。 ● 并行回放： page redo：synctrxn日志回放。 startup：强同步等待。
step6_count	step6的统计次数。
step7_total	step7的总时间，每个线程对应的流程如下： ● 极致RTO： redo manager：创建表空间。 redo worker：fsm更新。 ● 并行回放： page redo：single日志回放。
step7_count	step7的统计次数。
step8_total	step8的总时间，每个线程对应的流程如下： ● 极致RTO： redo worker：强同步等待。 ● 并行回放： page redo：all workers do日志回放。
step8_count	step8的统计次数。

字段名	描述
step9_total	step9的总时间，每个线程对应的流程如下： ● 极致RTO： 无。 ● 并行回放： page redo：muliti workers do日志回放。
step9_count	step9的统计次数。

- local_xlog_redo_statics()

  描述：返回本节点已经回放的各个类型的日志统计信息（仅在备机上有有效数据）。

  返回值如下：

  表 7-88 local_xlog_redo_statics 返回参数说明

字段名	描述
xlog_type	日志类型。
rmid	resource manager id。
info	xLog operation。
num	日志个数。
extra	针对page回放日志和xact日志有有效值。 ● page页回放日志类型：表示从磁盘读取page的个数。 ● xact日志类型：表示删除文件的个数。

- remote_bgwriter_stat()

  描述：显示整个集群所有实例的bgwriter线程刷页信息，候选buffer链中页面个数，buffer淘汰信息（查询结果不包含当前节点的信息，DN上不支持使用）。

  返回值类型：record

  表 7-89 remote_bgwriter_stat 参数说明

参数	类型	描述
node_name	text	实例名称。
bgwr_actual_flush_total_num	bigint	从启动到当前时间bgwriter线程总计刷脏页数量。
bgwr_last_flush_num	integer	bgwriter线程上一批刷脏页数量。
candidate_slots	integer	当前候选buffer链中页面个数。

参数	类型	描述
get_buffer_from_list	bigint	buffer淘汰从候选buffer链中获取页面的次数。
get_buf_clock_sweep	bigint	buffer淘汰从原淘汰方案中获取页面的次数。

示例：

remote_bgwriter_stat函数显示整个集群所有实例的bgwriter线程刷页信息、候选buffer链中页面个数、buffer淘汰信息（查询结果不包含当前节点的信息，DN上不支持使用）。

```
openGauss=# SELECT * FROM remote_bgwriter_stat();
 node_name | bgwr_actual_flush_total_num | bgwr_last_flush_num | candidate_slots |
get_buffer_from_list | get_buf_clock_sweep
-----------+-----------------------------+---------------------+-----------------+----------------------
+----------------------
 datanode2 |                           0 |                   0 |          393208 |                  665 |                    0
 datanode1 |                           0 |                   0 |          393208 |                  659 |                    0
(2 rows)
```

- gs_stack()

  描述：显示线程调用栈。查询该函数需要有sysadmin权限或者MONADMIN权限。

  参数：tid，线程id。tid是可选参数，指定tid参数时，函数返回tid对应线程调用栈；当不指定tid参数时，函数返回所有线程的调用栈。

  返回值：当指定tid时，返回值为text；当不指定tid时，返回值为setof record。

  示例：

  select * from gs_stack(tid)获取指定线程调用栈。

```
openGauss=# select * from gs_stack(139663481165568);
                          gs_stack
-------------------------------------------------------------------
 __poll + 0x2d                                                    +
 WaitLatchOrSocket(Latch volatile*, int, int, long) + 0x29f       +
 WaitLatch(Latch volatile*, int, long) + 0x2e                     +
 JobScheduleMain() + 0x90f                                        +
 int GaussDbThreadMain<(knl_thread_role)9>(knl_thread_arg*) + 0x456+
 InternalThreadFunc(void*) + 0x2d                                 +
 ThreadStarterFunc(void*) + 0xa4                                  +
 start_thread + 0xc5                                              +
 clone + 0x6d                                                     +
(1 row)
```

  select * from gs_stack()获取所有线程的调用栈。

```
openGauss=# select * from gs_stack();
-[ RECORD
1 ]-----------------------------------------------------------------------------------------------------------
tid   | 139670364324352
lwtid | 308
stack | __poll + 0x2d
      | CommWaitPollParam::caller(int (*)(pollfd*, unsigned long, int), unsigned long) + 0x34
      | int comm_socket_call<CommWaitPollParam, int (*)(pollfd*, unsigned long,
int)>(CommWaitPollParam*, int (*)(pollfd*, unsigned long
, int)) + 0x28
      | comm_poll(pollfd*, unsigned long, int) + 0xb1
      | ServerLoop() + 0x72b
      | PostmasterMain(int, char**) + 0x314e
      | main + 0x617
      | __libc_start_main + 0xf5
```



```
     | 0x55d38f8db3a7
[ RECORD 2 ]-----------------------------------------------------------------------------------------------------------
tid   | 139664851859200
lwtid | 520
stack | __poll + 0x2d
      | WaitLatchOrSocket(Latch volatile*, int, int, long) + 0x29f
      | SysLoggerMain(int) + 0xc86
      | int GaussDbThreadMain<(knl_thread_role)17>(knl_thread_arg*) + 0x45d
      | InternalThreadFunc(void*) + 0x2d
      | ThreadStarterFunc(void*) + 0xa4
      | start_thread + 0xc5
      | clone + 0x6d
```

7.5.25 触发器函数

- pg_get_triggerdef(oid)

  描述：获取触发器的定义信息。

  参数：待查触发器的OID。

  返回值类型：text

- pg_get_triggerdef(oid, boolean)

  描述：获取触发器的定义信息。

  参数：待查触发器的OID及是否以pretty方式展示。

  返回值类型：text

7.5.26 HashFunc 函数

- bucketabstime(value，flag)

  描述：对abstime格式的数值value计算hash值并找到对应的hashbucket桶。

  参数：value为需要转换的数值，类型为abstime，flag为int类型表示数据分布方式，0表示hash分布。

  返回值类型：int32

  示例：

```
openGauss=# select bucketabstime('2011-10-01 10:10:10.112',1);
 bucketabstime
---------------
         13954
(1 row)
```

- bucketbool(value，flag)

  描述：对bool格式的数值value计算hash值并找到对应的hashbucket桶。

  参数：value为需要转换的数值，类型为bool，flag为int类型表示数据分布方式，0表示hash分布。

  返回值类型：int32

  示例：

```
openGauss=# select bucketbool(true,1);
 bucketbool
------------
          1
(1 row)
openGauss=# select bucketbool(false,1);
 bucketbool
------------
          0
(1 row)
```


- bucketbpchar(value, flag)

  描述：对bpchar格式的数值value计算hash值并找到对应的hashbucket桶。

  参数：value为需要转换的数值，类型为bpchar，flag为int类型表示数据分布方式，0表示hash分布。

  返回值类型：int32

  示例：

  ```
  openGauss=# select bucketbpchar('test',1);
   bucketbpchar
  --------------
          9761
  (1 row)
  ```

- bucketbytea(value，flag)

  描述：对bytea格式的数值value计算hash值并找到对应的hashbucket桶。

  参数：value为需要转换的数值，类型为bytea，flag为int类型表示数据分布方式，0表示hash分布。

  返回值类型：int32

  示例：

  ```
  openGauss=# select bucketbytea('test',1);
   bucketbytea
  -------------
         9761
  (1 row)
  ```

- bucketcash(value，flag)

  描述：对money格式的数值value计算hash值并找到对应的hashbucket桶。

  参数：value为需要转换的数值，类型为money，flag为int类型表示数据分布方式，0表示hash分布。

  返回值类型：int32

  示例：

  ```
  openGauss=# select bucketcash(10::money,1);
   bucketcash
  ------------
        8468
  (1 row)
  ```

- getbucket(value，flag)

  描述：从分布列获取hashbucket桶。

  value为需要输入的数值，类型：

  “char”,abstime,bigint,boolean,bytea,character varying,character,date,double precision,int2vector,integer,interval,money,name,numeric,nvarchar2,oid,oidvector,raw,real,record,reltime,smalldatetime,smallint,text,time with time zone,time without time zone,timestamp with time zone,timestamp without time zone,tinyint,uuid。

  flag表示数据分布方式，类型：integer

  返回值类型：integer

  示例：

  ```
  openGauss=# select getbucket(10,'H');
   getbucket
  -----------
      14535
  (1 row)
  ```



```
openGauss=# select getbucket(11,'H');
 getbucket
-----------
     13449
(1 row)

openGauss=# select getbucket(11,'R');
 getbucket
-----------
     13449
(1 row)

openGauss=# select getbucket(12,'R');
 getbucket
-----------
      9412
(1 row)
```

- hash_array(anyarray)

  描述：数组哈希，将数组的元素通过哈希函数得到结果，并返回合并结果。

  参数：数据类型为anyarray。

  返回值类型：integer

  示例：

```
openGauss=# select hash_array(ARRAY[[1,2,3],[1,2,3]]);
 hash_array
------------
 -382888479
(1 row)
```

- hash_group(key)

  描述：流引擎（由于规格变更，当前版本已经不再支持本特性，请不要使用）中，该函数可将Group Clause中的各列计算为一个hash值。

  参数：key为Group Clause中各列的值。

  返回值类型：32位hash值

  示例：

```
按照步骤依次执行。
openGauss=# CREATE TABLE tt(a int, b int,c int,d int);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
openGauss=# select * from tt;
 a | b | c | d
---+---+---+---
(0 rows)

openGauss=# insert into tt values(1,2,3,4);
INSERT 0 1
openGauss=# select * from tt;
 a | b | c | d
---+---+---+---
 1 | 2 | 3 | 4
(1 row)

openGauss=# insert into tt values(5,6,7,8);
INSERT 0 1
openGauss=# select * from tt;
 a | b | c | d
---+---+---+---
 1 | 2 | 3 | 4
 5 | 6 | 7 | 8
(2 rows)

openGauss=# select hash_group(a,b) from tt where a=1 and b=2;
```



```
 hash_group
------------
  990882385
(1 row)
```

- hash_numeric(numeric)

  描述：计算Numeric类型的数据的hash值。

  参数：Numeric类型的数据。

  返回值类型：integer

  示例：

```
openGauss=# select hash_numeric(30);
 hash_numeric
--------------
   -282860963
(1 row)
```

- hash_range(anyrange)

  描述：计算range的哈希值。

  参数：anyrange类型的数据。

  返回值类型：integer

  示例：

```
openGauss=# select hash_range(numrange(1.1,2.2));
 hash_range
------------
  683508754
(1 row)
```

- hashbpchar(character)

  描述：计算bpchar的哈希值。

  参数：character类型的数据。

  返回值类型：integer

  示例：

```
openGauss=# select hashbpchar('hello');
 hashbpchar
-------------
 -1870292951
(1 row)
```

- hashchar(char)

  描述：char和布尔数据转换为哈希值。

  参数：char类型的数据或者bool类型的数据。

  返回值类型：integer

  示例：

```
openGauss=# select hashbpchar('hello');
 hashbpchar
-------------
 -1870292951
(1 row)

openGauss=# select hashchar('true');
  hashchar
------------
 1686226652
(1 row)
```

- hashenum(anyenum)

  描述：枚举类型转哈希值。

参数：anyenum类型的数据。

返回值类型：integer

示例：

```
openGauss=# CREATE TYPE b1 AS ENUM('good', 'bad', 'ugly');
CREATE TYPE
openGauss=# call hashenum('good'::b1);
 hashenum
------------
 1821213359
(1 row)
```

- hashfloat4(real)

  描述：float4转哈希值。

  参数：real类型的数据。

  返回值类型：integer

  示例：

  ```
  openGauss=# select hashfloat4(12.1234);
   hashfloat4
  ------------
   1398514061
  (1 row)
  ```

- hashfloat8(double precision)

  描述：float8转哈希值。

  参数：double precision类型的数据。

  返回值类型：integer

  示例：

  ```
  openGauss=# select hashfloat8(123456.1234);
   hashfloat8
  ------------
   1673665593
  (1 row)
  ```

- hashinet(inet)

  描述：支持inet / cidr上的哈希索引的功能。返回传入inet的hash值。

  参数：inet类型的数据。

  返回值类型：integer

  示例：

  ```
  openGauss=# select hashinet('127.0.0.1'::inet);
   hashinet
  -------------
   -1435793109
  (1 row)
  ```

- hashint1(tinyint)

  描述：INT1转哈希值。

  参数：tinyint类型的数据。

  返回值类型：uint32

  示例：

  ```
  openGauss=# select hashint1(20);
   hashint1
  -------------
   -2014641093
  (1 row)
  ```


- hashint2(smallint)
  描述：INT2转哈希值。
  参数：smallint类型的数据。
  返回值类型：uint32
  示例：

```
openGauss=# select hashint2(20000);
 hashint2
------------
 -863179081
(1 row)
```

- bucketchar
  描述：计算入参的哈希值。
  参数：char, integer
  返回值类型：integer
- bucketdate
  描述：计算入参的哈希值。
  参数：date, integer
  返回值类型：integer
- bucketfloat4
  描述：计算入参的哈希值。
  参数：real, integer
  返回值类型：integer
- bucketfloat8
  描述：计算入参的哈希值。
  参数：double precision, integer
  返回值类型：integer
- bucketint1
  描述：计算入参的哈希值。
  参数：tinyint, integer
  返回值类型：integer
- bucketint2
  描述：计算入参的哈希值。
  参数：smallint, integer
  返回值类型：integer
- bucketint2vector
  描述：计算入参的哈希值。
  参数：int2vector, integer
  返回值类型：integer
- bucketint4
  描述：计算入参的哈希值。
  参数：integer, integer
  返回值类型：integer

- bucketint8
  描述：计算入参的哈希值。
  参数：bigint, integer
  返回值类型：integer
- bucketinterval
  描述：计算入参的哈希值。
  参数：interval, integer
  返回值类型：integer
- bucketname
  描述：计算入参的哈希值。
  参数：name, integer
  返回值类型：integer
- bucketnumeric
  描述：计算入参的哈希值。
  参数：numeric, integer
  返回值类型：integer
- bucketnvarchar2
  描述：计算入参的哈希值。
  参数：nvarchar2, integer
  返回值类型：integer
- bucketoid
  描述：计算入参的哈希值。
  参数：oid, integer
  返回值类型：integer
- bucketoidvector
  描述：计算入参的哈希值。
  参数：oidvector, integer
  返回值类型：integer
- bucketraw
  描述：计算入参的哈希值。
  参数：raw, integer
  返回值类型：integer
- bucketreltime
  描述：计算入参的哈希值。
  参数：reltime, integer
  返回值类型：integer
- bucketsmalldatetime
  描述：计算入参的哈希值。
  参数：smalldatetime, integer
  返回值类型：integer

- buckettext
  描述：计算入参的哈希值。
  参数：text, integer
  返回值类型：integer
- buckettime
  描述：计算入参的哈希值。
  参数：time without time zone, integer
  返回值类型：integer
- buckettimestamp
  描述：计算入参的哈希值。
  参数：timestamp without time zone, integer
  返回值类型：integer
- buckettimestamptz
  描述：计算入参的哈希值。
  参数：timestamp with time zone, integer
  返回值类型：integer
- buckettimetz
  描述：计算入参的哈希值。
  参数：time with time zone, integer
  返回值类型：integer
- bucketuuid
  描述：计算入参的哈希值。
  参数：uuid, integer
  返回值类型：integer
- bucketvarchar
  描述：计算入参的哈希值。
  参数：character varying, integer
  返回值类型：integer

7.5.27 提示信息函数

- report_application_error()
  描述：PL执行过程中，可以使用此函数来抛ERROR。
  返回值类型：void

  表 7-90 report_application_error 参数说明

参数	类型	说明	是否必选
log	text	error消息的内容。	是
code	int4	error消息对应的error code，范围为：-20999 ~ -20000。	否

示例

```
openGauss=#  CREATE OR REPLACE FUNCTION GET_RESULT_UNKNOWN(
opengauss(#     IN context_id int, /*context_id*/
opengauss(#     IN col_type text /*col_type*/
opengauss(# )RETURNS INTEGER
opengauss-# AS $$
opengauss$# BEGIN
opengauss$#     if col_type is NULL then
opengauss$#         PG_CATALOG.REPORT_APPLICATION_ERROR('invalid input for the third parameter
col_type should not be null');
opengauss$#     end if;
opengauss$#     PG_CATALOG.REPORT_APPLICATION_ERROR('UnSupport data type for
column_value(context: '||context_id||', '||PG_CATALOG.QUOTE_LITERAL(col_type)||')');
opengauss$#     return -1;
opengauss$# END;
opengauss$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
opengauss=# CALL GET_RESULT_UNKNOWN(NULL, NULL);
ERROR:  invalid input for the third parameter col_type should not be null
CONTEXT:  SQL statement "CALL pg_catalog.report_application_error('invalid input for the third
parameter col_type should not be null')"
PL/pgSQL function get_result_unknown(integer,text) line 4 at PERFORM
```

7.5.28 故障注入系统函数

gs_fault_inject(int64, text, text, text, text, text)

描述：该函数不能调用，调用时会报WARNING信息："unsupported fault injection"，并不会对数据库产生任何影响和改变。

参数：int64注入故障类型（0：CLOG扩展页面，1：读取CLOG页面，2：强制死锁）。

- text第二个入参在第一入参为2的模式下若为“1”则死锁，其余不死锁；第二个入参在第一入参为0，1时，表示CLOG开始扩展或读取的起始页面号。
- text第三个入参在第一入参为0，1时，表示扩展或读取的页面个数。
- text第四到六入参为预留参数。

返回值类型：int64

7.5.29 重分布函数

以下函数为重分布期间gs_redis工具所用的系统函数，用户不要主动调用：

- pg_get_redis_rel_end_ctid(text, name, int, int)
- pg_get_redis_rel_start_ctid(text, name, int, int)
- pg_enable_redis_proc_cancelable()
- pg_disable_redis_proc_cancelable()
- pg_tupleid_get_blocknum(tid)
- pg_tupleid_get_offset(tid)
- pg_tupleid_get_ctid_to_bigint (ctid)

7.5.30 分布列推荐函数

分布列推荐针对的是在分布式数据库下分布列以及分布方式的推荐，目的是在进行业务迁移或业务上线时，减少选择表分布列的人力成本。

- sqladvisor.init(char, boolean, boolean, boolean, int, int)

描述：初始化参数。

返回值类型：bool

表 7-91 init 参数说明

参数	类型	说明	是否必选
kind	char	推荐类型，目前支持分布列推荐 'D'。	是
isUseCost	boolean	是否使用优化器，有数据的情况选择是。	是
isUseCollect	boolean	是否从收集的负载中开始分析，默认值为false。	否
isConstraintPrimaryKey	boolean	是否保持主键约束，默认值为true。	否
sqlCount	int	收集sql数量，默认值为10000，范围：1 ~ 100000。	否
maxMemory	int	分布列推荐最大占用内存，默认值为1024，范围：1 ~ 10240，单位MB。	否

- sqladvisor.set_weight_params(real, real, real)

描述：设置启发式规则不同成分的权重，调用init的时候会设置一个默认参数，分析时可以不调用该函数。

返回值类型：bool

表 7-92 set_weight_params 参数说明。

参数	类型	说明	是否必选
joinWeight	real	join的权重，范围：0 ~ 1000	是
goupbyWeight	real	group by的权重，范围：0 ~ 1000	是
qualWeight	real	predicate的权重，范围：0 ~ 1000	是

说明

此函数非必须调用，在执行init函数的时候，会预置一个默认权重分别是join: 1.0, group by: 0.1, predicate: 0.05。

- sqladvisor.set_cost_params(bigint, boolean, text)

描述：使用WhatIf代价模式可以设置的参数。

返回值类型：bool

表 7-93 set_cost_params 参数说明

参数	类型	说明	是否必选
maxTime	bigint	推荐的最大时间，小于等于0默认为不限时，单位min。	是
isTotalSQL	boolean	是否使用全部SQL参与计算，true：全部SQL，false：会按照百分位过滤掉代价过大或者过小的SQL。	是
compressLevel	text	压缩程度表示推荐算法搜索空间的大小，'low', 'med', 'high'。	是

说明

- 此函数非必须调用，在执行init函数的时候，会预置参数maxTime：-1，isTotalSQL：true, compressLevel :'high'。
- 用户选择的压缩程度越低，时间越长，越有可能推出更好的结果。

- sqladvisor.assign_table_type(text)

描述：指定表为复制表。

参数：表名

返回值类型：bool

说明

指定复制表需要在调用analyze_query和analyze_workload前使用。

- sqladvisor.analyze_query(text, int)

描述：导入需要推荐的SQL语句，并对语句的成分进行分析。

返回值类型：bool

表 7-94 analyze_query 参数说明

参数	类型	说明	是否必选
query	text	SQL语句。	是
frequency	int	该语句在负载中的频率，默认值为1，范围：1 ~ 2147483647。	否

说明

- 如果参数query中存在如单引号（'）等特殊字符，可以使用单引号（'）进行转义。
- 半在线模式不支持该函数。

- sqladvisor.analyze_workload()

描述：分析在线收集的负载信息。

返回值类型：bool

- sqladvisor.get_analyzed_result(text)

描述：获取当前表提取出来的有益成分。

参数：text

返回值类型：record

函数返回字段说明如下。

名称	类型	描述
schema_name	text	模式名
table_name	text	表名
col_name	text	列名
operator	text	算子类型
count	int	统计该操作符的次数

- sqladvisor.run()

描述：根据指定的模式和输入的SQL进行计算分析。

返回值类型：bool

- sqladvisor.get_distribution_key()

描述：获取推荐结果。

说明

分析结果保存在session中，session断连结果丢失。

返回值类型：record

函数返回字段说明如下。

名称	类型	描述
db_name	text	数据库名
schema_name	text	模式名
table_name	text	表名

名称	类型	描述
distribution_type	text	推荐的分布类型
distribution_key	text	推荐分布列
start_time	timestamp	推荐开始时间
end_time	timestamp	推荐结束时间
cost_improve	text	推荐结果对于代价的提升
comment	text	备注

- sqladvisor.clean()

描述：清理session中推荐过程中的全部内存。

返回值类型：bool

- sqladvisor.start_collect_workload(int, int)

描述：开启在线收集负载。

返回值类型：bool

表 7-95 start_collect_workload 参数说明

参数	类型	说明	是否必选
sqlCount	int	在线收集负载最大sql数量，默认值为10000，范围：1 ~ 100000。	是
maxMemory	int	在线收集负载最大占用内存，默认值为1024，范围：1 ~ 10240，单位MB。	是

须知

- 在线收集功能只能由系统管理员调用。
- 同一时间只能收集一个数据库的负载。
- 收集负载目前只支持普通SQL和存储过程中的DML、DQL语句。

- sqladvisor.end_collect_workload()

描述：关闭在线收集负载。

返回值类型：bool

须知

- 关闭在线收集功能只能由系统管理员调用。

- sqladvisor.clean_workload()

描述：清理负载中的内存。

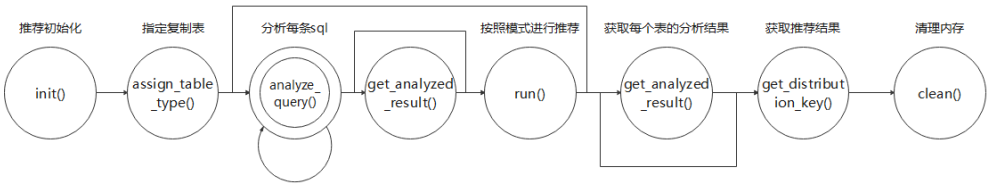
返回值类型：bool

须知

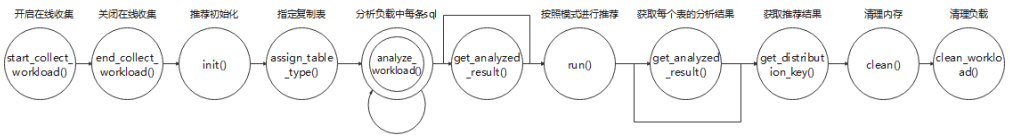
- 清理负载内存功能只能由系统管理员调用。
- 必须手动执行清理函数。

使用建议

- 启发式、WhatIf代价推荐模式调用状态机。



- 半在线推荐模式调用状态机。



7.5.31 其他系统函数

其他系统函数包含两类，兼容PostgreSQL的函数和实现内部功能的函数。这些函数不推荐使用，若需使用，请联系华为技术支持工程师。

兼容 PostgreSQL 的函数和操作符

下述列表为GaussDB的内建函数和操作符兼容PostgreSQL。

_pg_char_max_length	_pg_char_octet_length	_pg_datetime_precision	_pg_expandarray	_pg_index_position	_pg_interval_type	_pg_numeric_precision

_pg_numeric_precision_radix	_pg_numeric_scale	_pg_truetypid	_pg_truetypmod	q	abs	abstime
abstimeeq	abstimege	abstimegt	abstimein	abstimele	abstimelt	abstimene
abstimeout	abstimerecv	abstimesend	aclcontains	acldefault	aclexplode	aclinsert
aclitemeq	aclitemin	aclitemout	aclremove	acos	age	akeys
any_in	any_out	anyarray_in	anyarray_out	anyarray_recv	anyarray_send	anyelement_in
anyelement_out	anyenum_in	anyenum_out	anynonarray_in	anynonarray_out	anyrange_in	anyrange_out
anytextcat	area	areajoinsel	areasel	array_agg	array_agg_finalfn	array_agg_transfn
array_append	array_cat	array_dims	array_eq	array_fill	array_ge	array_gt
array_in	array_larger	array_le	array_length	array_lower	array_lt	array_ndims
array_ne	array_out	array_prepend	array_recv	array_send	array_smaller	array_to_json
array_to_string	array_typanalyze	array_upper	arraycontained	arraycontains	arraycontjoinsel	arraycontsel
arrayoverlap	ascii	asin	atan	atan2	avals	avg
big5_to_euc_tw	big5_to_mic	big5_to_utf8	bit	bit_and	bit_in	bit_length
bit_or	bit_out	bit_recv	bit_send	bitand	bitcat	bitcmp
biteq	bitge	bitgt	bitle	bitlt	bitne	bitnot
bitor	bitshiftleft	bitshiftright	bittypmodin	bittypmodout	bitxor	bool
bool_and	bool_or	booland_statefunc	booleq	boolge	boolgt	boolin

boolle	boollt	boolne	boolor_statefunc	boolout	boolrecv	boolsend
box	box_above	box_above_eq	box_add	box_below	box_below_eq	box_center
box_contain	box_contain_pt	box_contained	box_distance	box_div	box_eq	box_ge
box_gt	box_in	box_intersect	box_le	box_left	box_lt	box_mul
box_out	box_overabove	box_overbelow	box_overlap	box_overleft	box_overright	box_recv
box_right	box_same	box_send	box_sub	bpchar	bpchar_larger	bpchar_pattern_ge
bpchar_pattern_gt	bpchar_pattern_le	bpchar_pattern_lt	bpchar_smaller	bpchar_sortsupport	bpcharcmp	bpchareq
bpcharge	bpchargt	bpcharicliike	bpcharicnlike	bpcharicregexeq	bpcharicregexne	bpcharin
bpcharle	bpcharlike	bpcharlt	bpcharne	bpcharnlike	bpcharout	bpcharrecv
bpcharregexeq	bpcharregexne	bpcharsend	bpchartypmodin	bpchartypmodout	broadcast	btabstimecmp
btarraycmp	btbeginscan	btboolcmp	btbpchar_pattern_cmp	btbuild	btbuildempty	btbulkdelete
btcanreturn	btcharcmp	btcostestimate	btendscan	btfloat48cmp	btfloat4cmp	btfloat4sortsupport
btfloat84cmp	btfloat8cmp	btfloat8sortsupport	btgetbitmap	btgettuple	btinsert	btint24cmp
btint28cmp	btint2cmp	btint2sortsupport	btint42cmp	btint48cmp	btint4cmp	btint4sortsupport
btint82cmp	btint84cmp	btint8cmp	btint8sortsupport	btmarkpos	btnamecmp	btnamesortsupport
btoidcmp	btoidsortsupport	btoidvectorcmp	btoptions	btrecordcmp	btreltimecmp	btrescan

btrestrpos	btrim	bttext_pattern_cmp	bttextcmp	bttextsortsupport	bttidcmp	bttintervalcmp
btvacuumcleanup	bytea_sortsupport	bytea_string_agg_finalfn	bytea_string_agg_transfn	byteacat	byteacmp	byteaeq
byteage	byteagt	byteain	byteale	bytealike	bytealt	byteane
byteanlike	byteaout	bytearecv	byteasend	cash_cmp	cash_div_cash	cash_div_flt4
cash_div_flt8	cash_div_int2	cash_div_int4	cash_div_int8	cash_eq	cash_ge	cash_gt
cash_in	cash_le	cash_lt	cash_mi	cash_mul_flt4	cash_mul_flt8	cash_mul_int2
cash_mul_int4	cash_mul_int8	cash_ne	cash_out	cash_pl	cash_recv	cash_send
cashlarger	cashsmaller	cbrt	ceil	ceiling	center	char
char_length	character_length	chareq	charge	chargt	charin	charle
charlt	charne	charout	charrecv	charsend	chr	cideq
cidin	cidout	cidr	cidr_in	cidr_out	cidr_recv	cidr_send
cidrecv	cidsend	circle	circle_above	circle_add_pt	circle_below	circle_center
circle_contain	circle_contain_pt	circle_contained	circle_distance	circle_div_pt	circle_eq	circle_ge
circle_gt	circle_in	circle_le	circle_left	circle_lt	circle_mul_pt	circle_ne
circle_out	circle_overabove	circle_overbelow	circle_overlap	circle_overleft	circle_overright	circle_recv
circle_right	circle_same	circle_send	circle_sub_pt	clock_timestamp	close_lb	close_ls
close_lseg	close_pb	close_pl	close_ps	close_sb	close_sl	col_description

concat	concat_ws	contjoinsel	contsel	convert	convert_from	convert_to
corr	cos	cot	count	covar_pop	covar_samp	cstring_in
cstring_out	cstring_recv	cstring_send	cume_dist	current_database	current_query	current_schema
xpath_exists	current_setting	current_user	currtid	currtid2	currval	cursor_to_xml
cursor_to_xmlschema	database_to_xml	database_to_xml_and_xmlschema	database_to_xmlschema	date	date_cmp	date_cmp_timestamp
date_cmp_timestamptz	date_eq	date_eq_timestamp	date_eq_timestamptz	date_ge	date_ge_timestamp	date_ge_timestamptz
date_gt	date_gt_timestamp	date_gt_timestamptz	date_in	date_larger	date_le	date_le_timestamp
date_le_timestamptz	date_lt	date_lt_timestamp	date_lt_timestamptz	date_mi	date_mi_interval	date_mii
date_ne	date_ne_timestamp	date_ne_timestamptz	date_out	date_pl_interval	date_pli	date_recv
date_send	date_smaller	date_sortsupport	daterange_canonical	daterange_subdiff	datetime_pl	datetimetz_pl
dcbrt	decode	defined	degrees	delete	dense_rank	dexp
diagonal	diameter	dispell_init	dispell_lexize	dist_cpoly	dist_lb	dist_pb
dist_pc	dist_pl	dist_ppath	dist_ps	dist_sb	dist_sl	div
dlog1	dlog10	domain_in	domain_recv	dpow	dround	dsimple_init
dsimple_lexize	dsnowball_init	dsnowball_lexize	dsqrt	dsynonym_init	dsynonym_lexize	dtrunc
each	enum_ne	enum_out	enum_range	enum_recv	enum_send	enum_smaller

eqjoinsel	eqsel	euc_cn_to_mic	euc_cn_to_utf8	euc_jis_2004_to_shift_jis_2004	euc_jis_2004_to_utf8	euc_jp_to_mic
euc_jp_to_sjis	euc_jp_to_utf8	euc_kr_to_mic	euc_kr_to_utf8	euc_tw_to_big5	euc_tw_to_mic	euc_tw_to_utf8
every	exist	exists_all	exists_any	exp	factorial	family
fdw_handler_in	fdw_handler_out	fetchval	first_value	float4	float4_accum	float48div
float48eq	float48ge	float48gt	float48le	float48lt	float48mi	float48mul
float48ne	float48pl	float4abs	float4div	float4eq	float4ge	float4gt
float4in	float4larger	float4le	float4lt	float4mi	float4mul	float4ne
float4out	float4pl	float4recv	float4send	float4smaller	float4um	float4up
float8	float8_accum	float8_avg	float8_collect	float8_corr	float8_covar_pop	float8_covar_samp
float8_regr_accum	float8_regr_avgx	float8_regr_avgy	float8_regr_collect	float8_regr_intercept	float8_regr_r2	float8_regr_slope
float8_regr_sxx	float8_regr_sxy	float8_regr_syy	float8_stddev_pop	float8_stddev_samp	float8_var_pop	float8_var_samp
float84div	float84eq	float84ge	float84gt	float84le	float84lt	float84mi
float84mul	float84ne	float84pl	float8abs	float8div	float8eq	float8ge
float8gt	float8in	float8larger	float8le	float8lt	float8mi	float8mul
float8ne	float8out	float8pl	float8recv	float8send	float8smaller	float8um
float8up	floor	flt4_mul_cash	flt8_mul_cash	fmgr_c_validator	fmgr_internal_validator	fmgr_sql_validator

format	format_type	gb18030_to_utf8	gbk_to_utf8	generate_series	generate_subscripts	get_bit
get_byte	get_current_ts_config	get_global_gs_asp	get_large_table_name	gin_clean_pending_list	gin_cmp_prefix	gin_cmp_tslexeme
gin_extract_tsquery	gin_extract_tsvector	gin_tsquery_consistent	gin_tsquery_triconsistent	ginarrayconsistent	ginarrayextract	ginarraytriconsistent
ginbeginscan	ginbuild	ginbuildempty	ginbulkdelete	gincostestimate	ginendscan	gingetbitmap
gininsert	ginmarkpos	ginoptions	ginqueryarrayextract	ginrescan	ginrestrpos	ginvacuumcleanup
gist_box_compress	gist_box_consistent	gist_box_decompress	gist_box_penalty	gist_box_picksplit	gist_box_same	gist_box_union
gist_circle_compress	gist_circle_consistent	gist_point_compress	gist_point_consistent	gist_point_distance	gist_poly_compress	gist_poly_consistent
gistbeginscan	gistbuild	gistbuildempty	gistbulkdelete	gistcostestimate	gistendscan	gistgetbitmap
gistgettuple	gistinsert	gistmarkpos	gistoptions	gistrescan	gistrestrpos	gistvacuumcleanup
gtsquery_compress	gtsquery_consistent	gtsquery_decompress	gtsquery_penalty	gtsquery_picksplit	gtsquery_same	gtsquery_union
gtsvector_compress	gtsvector_consistent	gtsvector_decompress	gtsvector_penalty	gtsvector_picksplit	gtsvector_same	gtsvector_union
gtsvectorin	gtsvectorout	has_tablespace_privilege	has_type_privilege	hash_aclitem	hashbeginscan	hashbuild
hashbuildempty	hashbulkdelete	hashcostestimate	hashendscan	hashgetbitmap	hashgettuple	hashinsert
hashint2vector	hashint4	hashint8	hashmacaddr	hashmarkpos	hashname	hashoid

hashoidvector	hashoptions	hashrescan	hashrestrpos	hashtext	hashvacuumcleanup	hashvarlena
host	hostmask	iclikejoinsel	iclikesel	icnlikejoinsel	icnlikesel	icregexeqjoinsel
icregexeqsel	icregexnejoinsel	icregexnesel	inet_client_addr	inet_client_port	inet_in	inet_out
inet_recv	inet_send	inet_server_addr	inet_server_port	inetand	inetmi	inetmi_int8
inetnot	inetor	inetpl	initcap	int2_accum	int2_avg_accum	int2_mul_cash
int2_sum	int24div	int24eq	int24ge	int24gt	int24le	int24lt
int24mi	int24mul	int24ne	int24pl	int28div	int28eq	int28ge
int28gt	int28le	int28lt	int28mi	int28mul	int28ne	int28pl
int2abs	int2and	int2div	int2eq	int2ge	int2gt	int2in
int2larger	int2le	int2lt	int2mi	int2mod	int2mul	int2ne
int2not	int2or	int2out	int2pl	int2recv	int2send	int2shl
int2shr	int2smaller	int2um	int2up	int2vectoreq	int2vectorin	int2vectorout
int2vectorrecv	int2vectorsend	int2xor	int4_accum	int4_avg_accum	int4_mul_cash	int4_sum
int42div	int42eq	int42ge	int42gt	int42le	int42lt	int42mi
int42mul	int42ne	int42pl	int48div	int48eq	int48ge	int48gt
int48le	int48lt	int48mi	int48mul	int48ne	int48pl	int4abs
int4and	int4div	int4eq	int4ge	int4gt	int4in	int4inc
int4larger	int4le	int4lt	int4mi	int4mod	int4mul	int4ne

int4not	int4or	int4out	int4pl	int4range	int4range_canonical	int4range_subdiff
int4recv	int4send	int4shl	int4shr	int4smaller	int4um	int4up
int4xor	int8	int8_avg	int8_avg_accum	int8_avg_collect	int8_mul_cash	int8_sum
int8_sum_to_int8	int8+1635:1668_accum	int82div	int82eq	int82ge	int82gt	int82le
int82lt	int82mi	int82mul	int82ne	int82pl	int84div	int84eq
int84ge	int84gt	int84le	int84lt	int84mi	int84mul	int84ne
int84pl	int8abs	int8and	int8div	int8eq	int8ge	int8gt
int8in	int8inc	int8inc_any	int8inc_float8_float8	int8larger	int8le	int8lt
int8mi	int8mod	int8mul	int8ne	int8not	int8or	int8out
int8pl	int8pl_inet	int8range	int8range_canonical	int8range_subdiff	int8recv	int8send
int8shl	int8shr	int8smaller	int8um	int8up	int8xor	integer_pl_date
inter_lb	inter_sb	inter_sl	internal_in	internal_out	interval	interval_accum
interval_avg	interval_cmp	interval_collect	interval_div	interval_eq	interval_ge	interval_gt
interval_hash	interval_in	interval_larger	interval_le	interval_lt	interval_mi	interval_mul
interval_ne	interval_out	interval_pl	interval_pl_date	interval_pl_time	interval_pl_timestamp	interval_pl_timestamptz
interval_pl_timetz	interval_recv	interval_send	interval_smaller	interval_transform	interval_um	intervaltypmodin
intervaltypmodout	intinterval	isexists	ishorizontal	iso_to_koi8r	iso_to_mic	iso_to_win1251

iso_to_win866	iso8859_1_to_utf8	iso8859_to_utf8	isparallel	isperp	isvertical	johab_to_utf8
jsonb_in	jsonb_out	jsonb_recv	jsonb_send	-	-	-
json_in	json_out	json_recv	json_send	justify_days	justify_hours	justify_interval
koi8r_to_iso	koi8r_to_mic	koi8r_to_utf8	koi8r_to_win1251	koi8r_to_win866	koi8u_to_utf8	language_handler_in
language_handler_out	latin1_to_mic	latin2_to_mic	latin2_to_win1250	latin3_to_mic	latin4_to_mic	like_escape
likejoinsel	likesel	line	line_distance	line_eq	line_horizontal	line_in
line_interpt	line_intersect	line_out	line_parallel	line_perp	line_recv	line_send
line_vertical	ln	lo_close	lo_creat	lo_create	lo_export	lo_import
lo_lseek	lo_open	lo_tell	lo_truncate	lo_unlink	log	loread
lower	lower_inc	lower_inf	lowrite	lpad	lseg	lseg_center
lseg_distance	lseg_eq	lseg_ge	lseg_gt	lseg_horizontal	lseg_in	lseg_interpt
lseg_intersect	lseg_le	lseg_length	lseg_lt	lseg_ne	lseg_out	lseg_parallel
lseg_perp	lseg_recv	lseg_send	lseg_vertical	ltrim	macaddr_and	macaddr_cmp
macaddr_eq	macaddr_ge	macaddr_gt	macaddr_in	macaddr_le	macaddr_lt	macaddr_ne
macaddr_not	macaddr_or	macaddr_out	macaddr_recv	macaddr_send	makeaclitem	masklen
max	md5 MD5加密算法安全性低，存在安全风险，建议使用更安全的加密算法。	mic_to_big5	mic_to_euc_cn	mic_to_euc_jp	mic_to_euc_kr	mic_to_euc_tw

mic_to_iso	mic_to_koi8r	mic_to_latin1	mic_to_latin2	mic_to_latin3	mic_to_latin4	mic_to_sjis
mic_to_win1250	mic_to_win1251	mic_to_win866	min	mktinterval	money	mul_d_interval
name	nameeq	namege	namegt	nameiclike	nameicnlike	nameicregexeq
nameicregexne	namein	namele	namelike	namelt	namene	namenlike
nameout	namerecv	nameregexeq	nameregexne	namesend	neqjoinsel	neqsel
network_cmp	network_eq	network_ge	network_gt	network_le	network_lt	network_ne
network_sub	network_subeq	network_sup	network_supeq	nlikejoinsel	nlikesel	numeric
numeric_abs	numeric_accum	numeric_add	numeric_avg	numeric_avg_accum	numeric_avg_collect	numeric_cmp
numeric_collect	numeric_div	numeric_div_trunc	numeric_eq	numeric_exp	numeric_fac	numeric_ge
numeric_gt	numeric_in	numeric_inc	numeric_larger	numeric_le	numeric_ln	numeric_log
numeric_lt	numeric_mod	numeric_mul	numeric_ne	numeric_out	numeric_power	numeric_recv
numeric_send	numeric_smaller	numeric_sortsupport	numeric_sqrt	numeric_stddev_pop	numeric_stddev_samp	numeric_sub
numeric_transform	numeric_uminus	numeric_uplus	numeric_var_pop	numeric_var_samp	numerictypmodin	numerictypmodout
numrange_subdiff	oid	oideq	oidge	oidgt	oidin	oidlarger
oidle	oidlt	oidne	oidout	oidrecv	oidsend	oidsmaller
oidvectoreq	oidvectorge	oidvectorgt	oidvectorin	oidvectorle	oidvectorlt	oidvectorne

oidvectorout	oidvectorrecv	oidvectorsend	oidvectortypes	on_pb	on_pl	on_ppath
on_ps	on_sb	on_sl	opaque_in	opaque_out	ordered_set_transition	overlaps
overlay	path	path_add	path_add_pt	path_center	path_contain_pt	path_distance
path_div_pt	path_in	path_inter	path_length	path_mul_pt	path_n_eq	path_n_ge
path_n_gt	path_n_le	path_n_lt	path_npoints	path_out	path_recv	path_send
path_sub_pt	percentile_cont	percentile_cont_float8_final	percentile_cont_interval_final	pg_char_to_encoding	pg_cursor	pg_encoding_max_length
pg_encoding_to_char	-	-	-	pg_node_tree_in	pg_node_tree_out	pg_node_tree_recv
pg_node_tree_send	pg_prepared_statement	pg_prepared_xact	pg_notify	pg_stat_get_wal_receiver	pg_show_all_settings	pg_stat_get_bgwriter_stat_reset_time
pg_stat_get_buf_fsync_backend	pg_stat_get_checkpoint_sync_time	pg_stat_get_checkpoint_write_time	pg_stat_get_db_blk_read_time	pg_stat_get_db_blk_write_time	pg_stat_get_db_conflict_all	pg_stat_get_db_conflict_bufferpin
pg_stat_get_db_conflict_snapshot	pg_stat_get_db_conflict_startup_deadlock	pg_switch_xlog	xpath	pg_timezone_abbrevs	pg_timezone_names	pgxc_node_str
plpgsql_call_handler	plpgsql_inline_handler	plpgsql_validator	point_above	point_add	point_below	point_distance
point_div	point_eq	point_horiz	point_in	point_left	point_mul	point_ne
point_out	point_recv	point_right	point_send	point_sub	point_vert	poly_above

poly_below	poly_center	poly_contain	poly_contain_pt	poly_contained	poly_distance	poly_in
poly_left	poly_npoints	poly_out	poly_overabove	poly_overbelow	poly_overlap	poly_overleft
poly_overright	poly_recv	poly_right	poly_same	poly_send	polygon	position
positionjoinsel	positionsel	postgresql_fdw_validator	pow	power	prsd_end	prsd_headline
prsd_lextype	prsd_nexttoken	prsd_start	pt_contained_circle	pt_contained_poly	query_to_xml	query_to_xml_and_xmlschema
query_to_xmlschema	quote_ident	quote_literal	quote_nullable	radians	radius	random
range_adjacent	range_after	range_before	range_cmp	range_contained_by	range_contains	range_contains_elem
range_eq	range_ge	range_gist_compress	range_gist_consistent	range_gist_decompress	range_gist_penalty	range_gist_picksplit
range_gist_same	range_gist_union	range_gt	range_in	range_intersect	range_le	range_lt
range_minus	range_ne	range_out	range_overlaps	range_overleft	range_overright	range_recv
range_send	range_typanalyze	range_union	rank	record_eq	record_ge	record_gt
record_in	record_le	record_lt	record_ne	record_out	record_recv	record_send
regclass	regclassin	regclassout	regclassrecv	regclasssend	regconfigin	regconfigout
regconfigrecv	regconfigsend	regdictionaryin	regdictionaryout	regdictionaryrecv	regdictionarysend	regexeqjoinsel
regexeqsel	regexnejoinsel	regexnesel	regexp_matches	regexp_replace	regexp_split_to_array	regexp_split_to_table

regoperatorin	regoperatorout	regoperatorrecv	regoperatorsend	regoperin	regoperout	regoperrecv
regopersend	regprocedurein	regprocedureout	regprocedurerecv	regproceduresend	regprocin	regprocout
regprocrecv	regprocsend	regr_avgx	regr_avgy	regr_count	regr_intercept	regr_r2
regr_slope	regr_sxx	regr_sxy	regr_syy	regtypein	regtypeout	regtyperecv
regtypesend	reltime	reltimeeq	reltimege	reltimegt	reltimein	reltimele
reltimelt	reltimene	reltimeout	reltimerecv	reltimesend	repeat	replace
reverse	RI_FKey_cascade_del	RI_FKey_cascade_upd	RI_FKey_check_ins	RI_FKey_check_upd	RI_FKey_noaction_del	RI_FKey_noaction_upd
RI_FKey_restrict_del	RI_FKey_restrict_upd	RI_FKey_setdefault_del	RI_FKey_setdefault_upd	RI_FKey_setnull_del	RI_FKey_setnull_upd	right
round	row_number	row_to_json	rpad	rtrim	scalargtjoinsel	scalargtsel
scalarltjoinsel	scalarltsel	schema_to_xml	schema_to_xml_and_xmlschema	schema_to_xmlschema	session_user	set_bit
set_byte	set_config	set_masklen	shift_jis_2004_to_euc_jis_2004	shift_jis_2004_to_utf8	sjis_to_euc_jp	sjis_to_mic
sjis_to_utf8	smgrin	smgrout	spg_kd_choose	spg_kd_config	spg_kd_inner_consistent	spg_kd_picksplit
spg_quad_choose	spg_quad_config	spg_quad_inner_consistent	spg_quad_leaf_consistent	spg_quad_picksplit	spg_text_choose	spg_text_config

spg_text_inner_consistent	spg_text_leaf_consistent	spg_text_picksplit	spgbeginscan	spgbuild	spgbuildempty	spgbulkdelete
spgcanreturn	spgcostestimate	spgendscan	spggetbitmap	spggettuple	spginsert	spgmarkpos
spgoptions	spgrescan	spgrestrpos	spgvacuumcleanup	stddev	stddev_pop	stddev_samp
string_agg	string_agg_finalfn	string_agg_transfn	strip	sum	suppress_redundant_updates_trigger	table_to_xml
table_to_xml_and_xmlschema	table_to_xmlschema	tan	text	text_ge	text_gt	text_larger
text_le	text_lt	text_pattern_ge	text_pattern_gt	text_pattern_le	text_pattern_lt	text_smaller
textanycat	textcat	texteq	texticlike	texticnlike	texticregexeq	texticregexne
textin	textlike	textne	textnlike	textout	textrecv	textregexeq
textregexne	textsend	thesaurus_init	thesaurus_lexize	tideq	tidge	tidgt
tidin	tidlarger	tidle	tidlt	tidne	tidout	tidrecv
tidsend	tidsmaller	time	time_cmp	time_eq	time_ge	time_gt
time_hash	time_in	time_larger	time_le	time_lt	time_mi_interval	time_mi_time
time_ne	time_out	time_pl_interval	time_recv	time_send	time_smaller	time_transform
timedate_pl	timemi	timepl	timestamp	timestamp_cmp	timestamp_cmp_date	timestamp_cmp_timestamptz

timestamp_eq	timestamp_eq_date	timestamp_eq_timestamptz	timestamp_ge	timestamp_ge_date	timestamp_ge_timestamptz	timestamp_gt
timestamp_gt_date	timestamp_gt_timestamptz	timestamp_hash	timestamp_in	timestamp_larger	timestamp_le	timestamp_le_date
timestamp_le_timestamptz	timestamp_lt	timestamp_lt_date	timestamp_lt_timestamptz	timestamp_mi	timestamp_mi_interval	timestamp_ne
timestamp_ne_date	timestamp_ne_timestamptz	timestamp_out	timestamp_pl_interval	timestamp_recv	timestamp_send	timestamp_smaller
timestamp_sortsupport	timestamp_transform	timestamptypmodin	timestamptypmodout	timestamptz	timestamptz_cmp	timestamptz_cmp_date
timestamptz_cmp_timestamp	timestamptz_eq	timestamptz_eq_date	timestamptz_eq_timestamp	timestamptz_ge	timestamptz_ge_date	timestamptz_ge_timestamp
timestamptz_gt	timestamptz_gt_date	timestamptz_gt_timestamp	timestamptz_in	timestamptz_larger	timestamptz_le	timestamptz_le_date
timestamptz_le_timestamp	timestamptz_lt	timestamptz_lt_date	timestamptz_lt_timestamp	timestamptz_mi	timestamptz_mi_interval	timestamptz_ne
timestamptz_ne_date	timestamptz_ne_timestamp	timestamptz_out	timestamptz_pl_interval	timestamptz_recv	timestamptz_send	timestamptz_smaller
timestamptztypmodin	timestamptztypmodout	timetypmodin	timetypmodout	timetz	timetz_cmp	timetz_eq
timetz_ge	timetz_gt	timetz_hash	timetz_in	timetz_larger	timetz_le	timetz_lt
timetz_mi_interval	timetz_ne	timetz_out	timetz_pl_interval	timetz_recv	timetz_send	timetz_smaller

timetzdate_pl	timetztypmodin	timetztypmodout	timezone（2069）	timezone（1159）	timezone（2037）	timezone (2070)
timezone (1026)	timezone (2038)	tintervalct	tintervaleq	tintervalge	tintervalgt	tintervalin
tintervalle	tintervalleneq	tintervallenge	tintervallengt	tintervallenle	tintervallenlt	tintervallenne
tintervallt	tintervalne	tintervalout	tintervalov	tintervalrecv	tintervalsame	tintervalsend
tintervalstart	to_ascii（1845）	to_ascii（1847）	to_ascii（1846）	trigger_in	trigger_out	ts_match_qv
ts_match_tq	ts_match_tt	ts_match_vq	ts_rank	ts_rank_cd	ts_rewrite	ts_stat
ts_token_type	ts_typanalyze	tsmatchjoinsel	tsmatchsel	tsq_mcontained	tsq_mcontains	tsquery_and
tsquery_cmp	tsquery_eq	tsquery_ge	tsquery_gt	tsquery_le	tsquery_lt	tsquery_ne
tsquery_not	tsquery_or	tsqueryin	tsqueryout	tsqueryrecv	tsquerysend	tsrange
tsrange_subdiff	tstzrange	tstzrange_subdiff	tsvector_cmp	tsvector_concat	tsvector_eq	tsvector_ge
tsvector_gt	tsvector_le	tsvector_lt	tsvector_ne	tsvector_update_trigger	tsvector_update_trigger_column	tsvectorin
tsvectorout	tsvectorrecv	tsvectorsend	txid_current	txid_current_snapshot	txid_snapshot_in	txid_snapshot_out
txid_snapshot_recv	txid_snapshot_send	txid_snapshot_xip	txid_snapshot_xmax	txid_snapshot_xmin	txid_visible_in_snapshot	uhc_to_utf8
unique_key_recheck	unknownin	unknownout	unknownrecv	unknownsend	unnest	utf8_to_big5
utf8_to_euc_cn	utf8_to_euc_jis_2004	utf8_to_euc_jp	utf8_to_euc_kr	utf8_to_euc_tw	utf8_to_gb18030	utf8_to_gbk

utf8_to_iso8859	utf8_to_iso8859_1	utf8_to_johab	utf8_to_koi8r	utf8_to_koi8u	utf8_to_shift_jis_2004	utf8_to_sjis
utf8_to_uhc	utf8_to_win	uuid_cmp	uuid_eq	uuid_ge	uuid_gt	uuid_hash
uuid_in	uuid_le	uuid_lt	uuid_ne	uuid_out	uuid_recv	uuid_send
var_pop	var_samp	varbit	varbit_in	varbit_out	varbit_recv	varbit_send
varbit_transform	varbitcmp	varbiteq	varbitge	varbitgt	varbitle	varbitlt
varbitne	varbittypmodin	varbittypmodout	varchar	varchar_transform	varcharin	varcharout
varcharrecv	varcharsend	varchartypmodin	varchartypmodout	variance	void_in	void_out
void_recv	void_send	win_to_utf8	win1250_to_latin2	win1250_to_mic	win1251_to_iso	win1251_to_koi8r
win1251_to_mic	win1251_to_win866	win866_to_iso	win866_to_koi8r	win866_to_mic	win866_to_win1251	xideq
xideqint4	xidin	xidout	xidrecv	xidsend	xml	xml_in
xml_is_well_formed	xml_is_well_formed_content	xml_is_well_formed_document	xml_out	xml_recv	xml_send	xmlagg
xmlcomment	xmlconcat2	xmlexists	xmlvalidate	-	-	-

实现内部功能的函数

下述列表为GaussDB实现系统内部功能所使用的函数，不推荐使用，若需使用，请联系华为技术支持工程师。

- smgreq(a smgr, b smgr)
  描述：比较两个smgr是否一样。
  参数：smgr, smgr
  返回值类型：boolean

- smgrne(a smgr, b smgr)
  描述：判断两个smgr是否不一样。
  参数：smgr, smgr
  返回值类型：boolean
- spread_collect
  描述：该函数用于计算某段时间内最大和最小值得差值，用于聚合函数的数据收集过程。
  参数：s real[], v real[]
  返回值类型：real[]
- spread_final
  描述：该函数用于计算某段时间内最大和最小值得差值，用于聚合函数的数据最终处理过程。
  参数：s real[]
  返回值类型：real
- spread_internal
  描述：该函数用于计算某段时间内最大和最小值得差值，用于聚合函数的数据中间过程。
  参数：s real[], v real
  返回值类型：real[]
- xidin4
  描述：输入4字节的xid。
  参数：cstring
  返回值类型：xid32
- set_hashbucket_info
  描述：设置哈希桶信息。
  参数：text
  返回值类型：boolean
- gap_fill_internal
  描述：返回参数列表中第一个非NULL的参数值。
  参数：s anyelement, v anyelement
  返回值类型：anyelement
- int1send
  描述：将无符号一字节整数打包放入内部数据缓冲流。
  参数：tinyint
  返回值类型：bytea
- is_contain_namespace
  描述：查找表名和namespace分割的位置，如果不存在namespace，返回0。
  参数：relationname name
  返回值类型：integer
- is_oid_in_group_members
  描述：不支持

参数：node_oid oid, group_members oidvector_extend。

返回值类型：boolean

- isubmit_on_nodes_internal

  描述：不支持

  参数：job bigint, node_name name, database name, what text, next_date timestamp without time zone, job_interval text

  返回值类型：integer

- listagg

  描述：list类型agg聚集函数。

  参数：smallint, text

  返回值类型：text

- log_fdw_validator

  描述：验证函数。

  参数：text[], oid

  返回值类型：void

- nvarchar2typmodin

  描述：获取varchar的typmod信息。

  参数：cstring[]

  返回值类型：integer

- nvarchar2typmodout

  描述：获取varchar的typmod信息，并构造字符串返回。

  参数：integer

  返回值类型：cstring

- pg_nodes_memmon

  描述：不支持

  参数：nan

  返回值类型：innernname text, innerusedmem bigint, innertopctxt bigint, nname text, usedmem text, sharedbuffercache text, topcontext text

- read_disable_conn_file

  描述：读取禁止的连接文件。

  参数：nan

  返回值类型：disconn_mode text, disconn_host text, disconn_port text, local_host text, local_port text, redo_finished text

- regex_like_m

  描述：正则匹配，判断字符串是否符合给定的正则表达式。

  参数：text, text

  返回值类型：boolean

- update_pgjob

  描述：更新job。

  参数：bigint, "char", bigint, timestamp without time zone, timestamp without time zone, timestamp without time zone, timestamp without time zone, timestamp without time zone, smallint, text

返回值类型：void

- enum_cmp
  描述：枚举类比较函数，用于判断两个枚举类是否相等，以及相对大小。
  参数：anyenum, anyenum
  返回值类型：integer
- enum_eq
  描述：枚举类比较函数，用于实现=符号。
  参数：anyenum, anyenum
  返回值类型：boolean
- enum_first
  描述：返回枚举类中的第一个元素。
  参数：anyenum
  返回值类型：anyenum
- enum_ge
  描述：枚举类比较函数，用于实现>=符号。
  参数：anyenum, anyenum
  返回值类型：boolean
- enum_gt
  描述：枚举类比较函数，用于实现>符号。
  参数：anyenum, anyenum
  返回值类型：boolean
- enum_in
  描述：枚举类比较函数，用于判断元素是否在枚举类中。
  参数：cstring, oid
  返回值类型：anyenum
- enum_larger
  描述：枚举类比较函数，用于实现>符号。
  参数：anyenum, anyenum
  返回值类型：anyenum
- enum_last
  描述：返回枚举类中的最后一个元素。
  参数：anyenum
  返回值类型：anyenum
- enum_le
  描述：枚举类比较函数，用于实现<=符号。
  参数：anyenum, anyenum
  返回值类型：boolean
- enum_lt
  描述：枚举类比较函数，用于实现<符号。
  参数：anyenum, anyenum

返回值类型：boolean

- enum_smaller
  描述：枚举类比较函数，用于实现<符号。
  参数：anyenum, anyenum
  返回值类型：boolean
- node_oid_name
  描述：不支持
  参数：oid
  返回值类型：cstring
- pg_buffercache_pages
  描述：从共享buffer缓存里读取数据。
  参数：nan
  返回值类型：bufferid integer, relfilenode oid, bucketid smallint, storage_type oid, reltablespace oid, reldatabase oid, relforknumber smallint, relblocknumber bigint, isdirty boolean, usage_count smallint
- pg_check_xidlimit
  描述：判断nextxid是否>= xidwarnlimit。
  参数：nan
  返回值类型：boolean
- pg_comm_delay
  描述：展示单个DN的通信库时延状态。
  参数：nan
  返回值类型：text, text, integer, integer, integer, integer
- pg_comm_recv_stream
  描述：展示单个DN上所有的通信库接收流状态。
  参数：nan
  返回值类型：text, bigint, text, bigint, integer, integer, integer, text, bigint, integer, integer, integer, bigint, bigint, bigint, bigint, bigint
- pg_comm_send_stream
  描述：展示单个DN上所有的通信库发送流状态。
  参数：nan
  返回值类型：text, bigint, text, bigint, integer, integer, integer, text, bigint, integer, integer, integer, bigint, bigint, bigint, bigint, bigint
- pg_comm_status
  描述： 展示单个DN的通信状态。
  参数：nan
  返回值类型：text, integer, integer, bigint, bigint, bigint, bigint, bigint, integer, integer, integer, integer, integer
- pg_log_comm_status
  描述：在dn上打印一些log。
  参数：nan
  返回值类型：boolean

- pg_parse_clog
  描述：解析clog获取xid的status。
  参数：nan
  返回值类型：xid xid, status text
- pg_pool_ping
  描述：设置PoolerPing。
  参数：boolean
  返回值类型：SETOF boolean
- pg_pool_validate
  描述：通过比较pgxc_node系统表中的字段，验证连接是否可用。
  参数：clear boolean, co_node_name cstring
  返回值类型：pid bigint, node_name text
- pg_resume_bkp_flag
  描述：用于备份恢复获取delay xlong标志。
  参数：slot_name name
  返回值类型：start_backup_flag boolean, to_delay boolean, ddl_delay_recycle_ptr text, rewind_time text
- pg_stat_get_pooler_status
  描述：查询pooler中的缓存连接状态。
  参数：nan
  返回值类型：text、 text、 bigint、 text、 bigint、 boolean、 text、 bigint、 bigint、 bigint、 bigint、 bigint

表 7-96 PG_STAT_GET_POOLER_STATUS 字段

名称	类型	描述
database_name	OUT text	数据库名称。
user_name	OUT text	用户名。
tid	OUT bigint	非线程池逻辑下为连接CN的线程id，线程池逻辑下为连接CN的sessionid。
pgoptions	OUT text	数据库连接选项，详见**连接参数**描述的options字段。
node_oid	OUT bigint	连接的实例节点OID。
in_use	OUT boolean	连接是否正被使用。 • t（true）：表示连接正在使用。 • f（false）：表示连接没有使用。
session_params	OUT text	由此连接下发的GUC session参数。
fdsock	OUT bigint	本端socket。

名称	类型	描述
remote_pid	OUT bigint	对端线程号。
used_count	OUT bigint	该连接的复用次数。
idx	OUT bigint	通信对端DN在本CN内的标识编号。
streamid	OUT bigint	通信流在物理连接中的标识编号。

- psortoptions
  描述：返回psort属性。
  参数：text[], boolean
  返回值类型：bytea
- remove_job_class_depend
  描述：移除job依赖。
  参数：oid
  返回值类型：void
- xideq4
  描述：对比两个xid是否相等。
  参数：xid32, xid32
  返回值类型：boolean
- xideqint8
  描述：对比两个xid是否相等。
  参数：xid, bigint
  返回值类型：boolean
- xidlt
  描述：返回xid1 < xid2是否成立。
  参数：xid, xid
  返回值类型：boolean
- xidlt4
  描述：返回xid1 < xid2是否成立。
  参数：xid32, xid32
  返回值类型：boolean
- get_local_cont_query_stat
  描述：获取本机节点的指定持续计算视图统计信息。
  参数：cq_id oid
  返回值类型：cq oid, w_in_rows int8, w_in_bytes int8, w_out_rows int8, w_out_bytes int8, w_pendings int8, w_errors int8, r_in_rows int8, r_in_bytes int8, r_out_rows int8, r_out_bytes int8, r_errors int8, c_in_rows int8, c_in_bytes int8, c_out_rows int8, c_out_bytes int8, c_pendings int8, c_errors int8

- get_local_cont_query_stats

  描述：获取本机节点的所有持续计算视图统计信息。

  参数：nan

  返回值类型：cq oid, w_in_rows int8, w_in_bytes int8, w_out_rows int8, w_out_bytes int8, w_pendings int8, w_errors int8, r_in_rows int8, r_in_bytes int8, r_out_rows int8, r_out_bytes int8, r_errors int8, c_in_rows int8, c_in_bytes int8, c_out_rows int8, c_out_bytes int8, c_pendings int8, c_errors int8

- get_cont_query_stats

  描述：获取各个DN节点的所有持续计算视图统计信息。

  参数：nan

  返回值类型：node name, cq oid, w_in_rows int8, w_in_bytes int8, w_out_rows int8, w_out_bytes int8, w_pendings int8, w_errors int8, r_in_rows int8, r_in_bytes int8, r_out_rows int8, r_out_bytes int8, r_errors int8, c_in_rows int8, c_in_bytes int8, c_out_rows int8, c_out_bytes int8, c_pendings int8, c_errors int8

- reset_local_cont_query_stat

  描述：复位本机节点的指定持续计算视图统计信息。

  参数：cq_id oid

  返回值类型：boolean

- reset_local_cont_query_stats

  描述：复位本机节点的指定持续计算视图的关联统计信息。

  参数：cq_id oid

  返回值类型：boolean

- reset_cont_query_stats

  描述：复位各个DN节点的STREAM对象对应的持续计算视图统计信息。

  参数：stream_id oid

  返回值类型：boolean

- check_cont_query_schema_changed

  描述：判断指定持续计算视图的schema change状态。

  参数：cq_id oid

  返回值类型：boolean

- gs_get_standby_cluster_barrier_status

  描述：查看备cn/dn的barrier日志回放情况，包括已接收到的最新barrier点、已接收到的最新barrier点的LSN，上一次回放的barrier点，回放的目标barrier点。

  参数：nan

  返回值类型：barrier_id text，barrier_lsn text，recovery_id text，target_id text

  备注：调用该函数的用户需要具有SYSADMIN权限或具有OPRADMIN权限，运维管理员角色须打开operate_mode。

- gs_set_standby_cluster_target_barrier_id

  描述：设置回放的目标barrier点。

  参数：barrier_id字符串

  返回值类型：target_id text

备注：调用该函数的用户需要具有SYSADMIN权限或具有OPRADMIN权限，运维管理员角色须打开operate_mode。

- gs_query_standby_cluster_barrier_id_exist

  描述：查询指定barrier点备机是否接收到。

  参数：barrier_id字符串

  返回值类型：boolean

  备注：调用该函数的用户需要具有SYSADMIN权限或具有OPRADMIN权限，运维管理员角色须打开operate_mode。

  以下流函数虽存在但功能尚未支持，不建议使用：

  streaming_int8_avg_gather、streaming_numeric_avg_gather、streaming_float8_avg_gather、streaming_interval_avg_gather、streaming_int8_sum_gather、streaming_int2_int4_sum_gather

7.5.32 内部函数

GaussDB中下列函数使用了内部数据类型，用户无法直接调用，在此章节列出。

- 选择率计算函数

areajoinsel	areasel	arraycontjoinsel	arraycontsel	contjoinsel	contsel	eqjoinsel
eqsel	iclikejoinsel	iclikesel	icnlikejoinsel	icnlikesel	icregexeqjoinsel	icregexeqsel
icregexnejoinsel	icregexnesel	likejoinsel	likesel	neqjoinsel	neqsel	nlikejoinsel
nlikesel	positionjoinsel	positionsel	regexeqjoinsel	regexeqsel	regexnejoinsel	regexnesel
scalargtjoinsel	scalargtsel	scalarltjoinsel	scalarltsel	tsmatchjoinsel	tsmatchsel	-

- 统计信息收集函数

array_typanalyze	range_typanalyze	ts_typanalyze
local_rto_stat	remote_rto_stat	-

- 排序内部功能函数

bpchar_sortsupport	bytea_sortsupport	date_sortsupport	numeric_sortsupport	timestamp_sortsupport

- 内部类型处理函数

abstimerecv	euc_jis_2004_to_utf8	int2recv	line_recv	oidvectorrecv_extend	tidrecv	utf8_to_koi8u

anyarray_recv	euc_jp_to_mic	int2vectorrecv	lseg_recv	path_recv	time_recv	utf8_to_shift_jis_2004
array_recv	euc_jp_to_sjis	int4recv	macaddr_recv	pg_node_tree_recv	time_transform	utf8_to_sjis
ascii_to_mic	euc_jp_to_utf8	int8recv	mic_to_ascii	point_recv	timestamp_recv	utf8_to_uhc
ascii_to_utf8	euc_kr_to_mic	internal_out	mic_to_big5	poly_recv	timestamp_transform	utf8_to_win
big5_to_euc_tw	euc_kr_to_utf8	interval_recv	mic_to_euc_cn	pound_nexttoken	timestamptz_recv	uuid_recv
big5_to_mic	euc_tw_to_big5	interval_transform	mic_to_euc_jp	prsd_nexttoken	timetz_recv	varbit_recv
big5_to_utf8	euc_tw_to_mic	iso_to_koi8r	mic_to_euc_kr	range_recv	tinterval recv	varbit_transform
bit_recv	euc_tw_to_utf8	iso_to_mic	mic_to_euc_tw	rawrecv	tsqueryrecv	varchar_transform
boolrecv	float4recv	iso_to_win1251	mic_to_iso	record_recv	tsvectorrecv	varcharrecv
box_recv	float8recv	iso_to_win866	mic_to_koi8r	regclassrecv	txid_snapshot_recv	void_recv
bpcharrecv	gb18030_to_utf8	iso8859_1_to_utf8	mic_to_latin1	regconfigrecv	uhc_to_utf8	win_to_utf8
btoidsortsupport	gbk_to_utf8	iso8859_to_utf8	mic_to_latin2	regdictionaryrecv	unknownrecv	win1250_to_latin2
byteasend	gin_extract_tsvector	johab_to_utf8	mic_to_latin3	regoperatorrecv	utf8_to_ascii	win1250_to_mic
byteawithoutorderwithequalcolrecv	gtsvector_compress	json_recv	mic_to_latin4	regoperrecv	utf8_to_big5	win1251_to_iso

cash_recv	gtsvector_consistent	koi8r_to_iso	mic_to_sjis	regprocedurerecv	utf8_to_euc_cn	win1251_to_koi8r
charrecv	gtsvector_decompress	koi8r_to_mic	mic_to_win1250	regprocrecv	utf8_to_euc_jis_2004	win1251_to_mic
cidr_recv	gtsvector_penalty	koi8r_to_utf8	mic_to_win1251	regtyperecv	utf8_to_euc_jp	win1251_to_win866
cidrecv	gtsvector_picksplit	koi8r_to_win1251	mic_to_win866	reltimerecv	utf8_to_euc_kr	win866_to_iso
circle_recv	gtsvector_same	koi8r_to_win866	namerecv	shift_jis_2004_to_euc_jis_2004	utf8_to_euc_tw	win866_to_koi8r
cstring_recv	gtsvector_union	koi8u_to_utf8	ngram_nexttoken	shift_jis_2004_to_utf8	utf8_to_gb18030	win866_to_mic
date_recv	hll_recv	latin1_to_mic	numeric_recv	sjis_to_euc_jp	utf8_to_gbk	win866_to_win1251
domain_recv	hll_trans_recv	latin2_to_mic	numeric_transform	sjis_to_mic	utf8_to_iso8859	xidrecv
euc_cn_to_mic	large_seq_upgrade_ntree	latin2_to_win1250	nvarchar2recv	sjis_to_utf8	utf8_to_iso8859_1	xidrecv4
euc_cn_to_utf8	inet_recv	latin3_to_mic	oidrecv	smalldatetime_recv	utf8_to_johab	xml_recv
euc_jis_2004_to_shift_jis_2004	int1recv	latin4_to_mic	oidvectorrecv	textrecv	utf8_to_koi8r	cstore_tid_out
numeric_bool	int2vectorin_extend	int2vectorout_extend	int2vectorrecv_extend	int2vectorsend_extend	int8_accum	large_seq_rollback_ntree
i16toi1	int16eq	int16ge	int16gt	int16in	int16le	int16lt
int16mi	int16mul	int16ne	int16out	int16pl	int16recv	int16send

int16_bool	-	-	-	-	-	-

- 聚合操作内部函数

array_agg_finalfn	array_agg_transfn	bytea_string_agg_finalfn	bytea_string_agg_transfn	date_list_agg_noarg2_transfn	date_list_agg_transfn	float4_list_agg_noarg2_transfn
float4_list_agg_transfn	float8_list_agg_noarg2_transfn	float8_list_agg_transfn	int2_list_agg_noarg2_transfn	int2_list_agg_transfn	int4_list_agg_noarg2_transfn	int4_list_agg_transfn
int8_list_agg_noarg2_transfn	int8_list_agg_transfn	interval_list_agg_noarg2_transfn	interval_list_agg_transfn	list_agg_finalfn	list_agg_noarg2_transfn	list_agg_transfn
median	median_float8_finalfn	median_interval_finalfn	median_transfn	mode_final	numeric_list_agg_noarg2_transfn	numeric_list_agg_transfn
ordered_set_transition	percentile_cont_float8_final	percentile_cont_interval_final	string_agg_finalfn	string_agg_transfn	timestamp_list_agg_noarg2_transfn	timestamp_list_agg_transfn
timestamptz_list_agg_noarg2_transfn	timestamptz_list_agg_transfn	checksumtext_agg_transfn	json_agg_transfn	json_agg_finalfn	json_object_agg_transfn	json_object_agg_finalfn

- 哈希内部功能函数

hashbeginscan	hashbuild	hashbuildempty	hashbulkdelete	hashcostestimate	hashendscan	hashgetbitmap
hashgettuple	hashinsert	hashmarkpos	hashmerge	hashrescan	hashrestrpos	hashvacuumcleanup
hashvarlena	jsonb_hash	-	-	-	-	-

- Btree索引内部功能函数

cbtreebuild	cbtreecanreturn	cbtreecostestimate	cbtreegetbitmap	cbtreegettuple	btbeginscan	btbuild
btbuildempty	btbulkdelete	btcanreturn	btcostestimate	btendscan	btfloat4sortsupport	btfloat8sortsupport
btgetbitmap	btgettuple	btinsert	btint2sortsupport	btint4sortsupport	btint8sortsupport	btmarkpos
btmerge	btnamesortsupport	btrescan	btrestrpos	bttextsortsupport	btvacuumcleanup	cbtreeoptions

- Psort索引内部函数

psortbuild	psortcanreturn	psortcostestimate	psortgetbitmap	psortgettuple

- Ubtree索引内部函数

ubtbeginscan	ubtbuild	ubtbuildempty	ubtbulkdelete	ubtcanreturn
ubtcostestimate	ubtendscan	ubtgetbitmap	ubtgettuple	ubtinsert
ubtmarkpos	ubtmerge	ubtoptions	ubtrescan	ubtrestrpos
ubtvacuumcleanup	-	-	-	-

- plpgsql内部函数

  plpgsql_inline_handler
- 外表相关内部函数

dist_fdw_handler	roach_handler	streaming_fdw_handler	dist_fdw_validator	file_fdw_handler	file_fdw_validator	log_fdw_handler
gc_fdw_handler	gc_fdw_validator	-	-	-	-	-

- 数据倾斜优化相关内部函数

  distributed_count
- 表统计信息相关内部函数

pgxc_get_stat_dirty_tables	pgxc_stat_dirty_tables	get_global_stat_all_tables	get_summary_stat_all_tables

- 远程读取数据函数

  gs_read_block_from_remote 用于读取非段页式表文件的页面。默认只有初始化用户可以查看，其余用户需要赋权后才可以使用。

  gs_read_segment_block_from_remote 用于读取段页式表文件的页面。默认只有初始化用户可以查看，其余用户需要赋权后才可以使用。

- 远程读取文件函数

  gs_read_file_size_from_remote 用于读取指定文件的大小，gs_repair_file函数修复文件时，要先获取远端关于这个文件的大小，用于校验本地文件缺失的文件信息，然后将缺失的文件逐个修复。默认只有初始化用户可以查看，其余用户需要赋权后才可以使用。

  gs_read_file_from_remote 用于读取指定的文件，gs_repair_file利用gs_read_file_size_from_remote函数获取文件大小后，依赖这个函数将远端文件逐段读取。默认只有初始化用户可以查看，其余用户需要赋权后才可以使用。

- 视图相关引用函数

  adm_hist_sqlstat_func

  adm_hist_sqlstat_idlog_func

7.5.33 动态数据脱敏函数

说明

该函数为内部功能调用函数。

- creditcardmasking(col text, letter char default 'x')

  描述：将col字符串后四位之前的数字使用letter替换。

  参数：待替换的字符串、替换字符。

  返回值类型：text

- basicmailmasking(col text, letter char default 'x')

  描述：将col字符串中第一个'@'之前的字符使用letter替换。

  参数：待替换的字符串、替换字符。

  返回值类型：text

- fullmailmasking(col text, letter char default 'x')

  描述：将col字符串中出现最后一个'.'之前的字符(除'@'外)使用letter替换。

  参数：待替换的字符串、替换字符。

  返回值类型：text

- alldigitsmasking(col text, letter char default '0')

  描述：将col字符串中出现的数字使用letter替换。

  参数：待替换的字符串、替换字符。

  返回值类型：text

- shufflemasking(col text)

  描述：将col字符串中的字符乱序排列。

  参数：待替换的字符串、替换字符。

  返回值类型：text

- randommasking(col text)

  描述：将col字符串中的字符随机化。

  参数：待替换的字符串、替换字符。

  返回值类型：text

- regexpmasking(col text, reg text, replace_text text, pos INTEGER default 0, reg_len INTEGER default -1)

  描述：将col字符串使用正则表达式替换。

  参数：待替换的字符串、正则表达式、替换的起始位置、替换长度、。

  返回值类型：text

7.5.34 hotkey 特性函数

- gs_stat_get_hotkeys_info()

  描述：获取本地节点查询的热词信息。

  返回值类型：Tuple

  示例：

```
openGauss=# select * from gs_stat_get_hotkeys_info() order by count, hash_value;
 database_name | schema_name |     table_name     | key_value | hash_value | count
---------------+-------------+--------------------+-----------+------------+-------
 regression    | public      | hotkey_single_col | {22}      | 1858004829 |     2
 regression    | public      | hotkey_single_col | {11}      | 2011968649 |     2
(2 rows)
```

- gs_stat_clean_hotkeys()

  描述：清理hotkey缓存，重置hotkey状态信息。

  返回值类型：bool, 恒为true

  示例：

```
openGauss=# select * from gs_stat_clean_hotkeys();
 gs_stat_clean_hotkeys
-----------------------
 t
(1 row)
```

7.5.35 Global SysCache 特性函数

- gs_gsc_table_detail(database_id default NULL, rel_id default NULL)

  描述：查看数据库里全局系统缓存的表元数据。调用该函数的用户需要具有SYSADMIN权限。

  参数：指定需要查看全局系统缓存的数据库和表，database_id默认值NULL或者-1表示所有的数据库，0表示共享表，其他数字表示指定数据库及共享表，rel_id表示指定表的oid，默认值NULL或者-1表示所有的表，其他值表示指定的表，database_id不存在会报错，rel_id 不存在查询结果为空。

  返回值类型：Tuple

```
select * from gs_gsc_table_detail(-1) limit 1;
database_oid | database_name | reloid |          relname          | relnamespace | reltype | reloftype |
relowner | relam | relfilenode | reltablespace | relhasindex | relisshared | relkind | relnatts | relhasoids |
relhaspkey | parttype | tdhasuids | attnames  | extinfo
--------------+---------------+--------+---------------------------+--------------+---------+-----------+----------
+-------+-------------+---------------+-------------+-------------+---------+----------+------------+------------
+----------+-----------+-----------+---------
0 |               |   2676 | pg_authid_rolname_index |           11 |       0 |         0 |       10 |   403 |          0
```

```
|       1664 | f          | t          | i      |       1 | f         | f         | n       | f         | 'rolname' |
(1 row)
```

- gs_gsc_catalog_detail(database_id default NULL, rel_id default NULL)

  描述：查看数据库里全局系统缓存的系统表行信息。调用该函数的用户需要具有SYSADMIN权限。

  参数：指定需要查看全局系统缓存的数据库和表，database_id 默认值NULL或者-1表示所有的数据库，0表示共享表，其他数字表示指定数据库及共享表，rel_id表示指定表的id，仅包含所有有系统缓存的系统表，默认值NULL或者-1表示所有的表，其他值表示指定的表，database_id不存在会报错，rel_id不存在结果为空。

  返回值类型：Tuple

  示例：

```
openGauss=#
select * from gs_gsc_catalog_detail(16574, 1260);
 database_id | database_name | rel_id | rel_name  | cache_id |  self  |  ctid  | infomask | infomask2 |
hash_value | refcount
-------------+---------------+--------+-----------+----------+--------+--------+----------+-----------+------------
+----------
           0 |               |   1260 | pg_authid |       10 | (0, 9) | (0, 9) |    10507 |        26 |  531311568 |
10
           0 |               |   1260 | pg_authid |       11 | (0, 4) | (0, 4) |     2313 |        26 |  365368336 |        1
           0 |               |   1260 | pg_authid |       11 | (0, 9) | (0, 9) |    10507 |        26 | 3911517328 |
10
           0 |               |   1260 | pg_authid |       11 | (0, 7) | (0, 7) |     2313 |        26 | 1317799983 |
1
           0 |               |   1260 | pg_authid |       11 | (0, 5) | (0, 5) |     2313 |        26 | 3664347448 |
1
           0 |               |   1260 | pg_authid |       11 | (0, 1) | (0, 1) |     2313 |        26 |  276477273 |        1
           0 |               |   1260 | pg_authid |       11 | (0, 3) | (0, 3) |     2313 |        26 | 2465837659 |
1
           0 |               |   1260 | pg_authid |       11 | (0, 8) | (0, 8) |     2313 |        26 | 3205288035 |
1
           0 |               |   1260 | pg_authid |       11 | (0, 6) | (0, 6) |     2313 |        26 |  131811687 |        1
           0 |               |   1260 | pg_authid |       11 | (0, 2) | (0, 2) |     2313 |        26 | 1226484587 |
1
(10 rows)
```

- gs_gsc_clean(database_id default NULL)

  描述：清理global syscache的缓存，需要注意，正在使用中的数据不会被清理。调用该函数的用户需要具有SYSADMIN权限。

  参数：指定需要清理全局系统缓存的数据库，默认值NULL或者-1表示强制清理所有的数据库全局系统缓存，0表示只淘汰共享表的全局系统缓存，其他数字表示淘汰指定数据库以及共享表的全局系统缓存，database_id不存在会报错。

  返回值类型：bool

  示例：

```
openGauss=# select * from gs_gsc_clean();
gs_gsc_clean
--------------
t
(1 row)
```

- gs_gsc_dbstat_info(database_id default NULL)

  描述：获取本地节点的GSC的内存统计信息，包括tuple、relation、partition的缓存查询，命中，加载、失效、占用空间信息，DB级别的淘汰信息，线程引用信息，内存占用信息。可以用于定位性能问题，例如当发现hits/searches数组远小于1时，可能是global_syscache_threshold设置太小，导致查询命中率下降。调用该函数的用户需要具有SYSADMIN权限。

参数：指定需要查看的数据库全局系统缓存统计信息，NULL或者-1表示查看所有的数据库，0表示只查看共享表信息，其他数字表示查看指定的数据库和共享表的信息。不合法的输入值，database_id不存在会报错。

返回值类型：Tuple

示例：

```
openGauss=# select * from gs_gsc_dbstat_info();
database_id | database_name | tup_searches | tup_hits | tup_miss | tup_count | tup_dead | tup_memory
| rel_searches | rel_hits | rel_mis
s | rel_count | rel_dead | rel_memory | part_searches | part_hits | part_miss | part_count | part_dead |
part_memory | total_memory | swa
pout_count | refcount
-------------+---------------+--------------+----------+----------+-----------+----------+------------
+--------------+----------+--------
--+-----------+----------+------------+---------------+-----------+-----------+------------+-----------
+-------------+--------------+----
-----------+----------
          0 |               |          300 |      235 |       31 |        22 |        2 |       9752 |          598 |      108
|       1
8 |        18 |        0 |     77720 |            0 |         0 |         0 |         0 |         0 |          0 |       752912 |
         0 |        0
      16574 | testdb      |         3368 |     2289 |      329 |       273 |        0 |      92593 |         1113 |
524 |       4
8 |        48 |        0 |    340456 |            0 |         0 |         0 |         0 |         0 |          0 |      4124792 |
         0 |       10
(2 rows)
```

7.5.36 数据损坏检测修复函数

- gs_verify_data_file(verify_segment bool)

  描述：校验当前实例当前库是否存在文件丢失的情况。校验只包括数据表主文件是否有中间段丢失的情况。默认参数是false，表示不校验段页式表数据文件。参数设置为true时仅校验段页式表文件。默认只有初始用户、具有sysadmin属性的用户以及在运维模式下具有运维管理员属性的用户可以查看，其余用户需要赋权后才可以使用。

  返回的结果：

  - 非段页式表：rel_oid和rel_name是对应文件的表oid和表名，miss_file_path表示丢失文件的相对路径。
  - 段页式表：因所有表存放在相同文件中，所以rel_oid和rel_name无法显示具体表的信息。对于段页式表，如果第一个文件损坏，不会检查出后面的.1 .2等文件。例如3、3.1、3.2损坏，只能检查出3损坏。当段页式文件不足5个时，使用函数检测时，未生成的文件也会校验出来，例如只有1和2文件，校验段页式时，也会检测出3，4，5文件。以下示例，第一个是校验非段页式表的示例，第二是校验段页式表的示例。

  参数说明：

  - verify_segment

    指定文件校验的范围。false校验非段页式表；true校验段页式表。

    取值范围：true和false，默认是false。

  返回值类型：record。

  示例（仅当发现有异常时才会输出异常行，否则输出0行）：

  校验非段页式表

```
openGauss=# select * from gs_verify_data_file();
node_name         | rel_oid |  rel_name    |  miss_file_path
------------------+---------+--------------+------------------
dn_6001_6002_6003 |   16554 |     test     | base/16552/24745
```


校验段页式表

```
openGauss=# select * from gs_verify_data_file(true);
     node_name      | rel_oid | rel_name | miss_file_path
--------------------+---------+----------+----------------
 dn_6001_6002_6003 |       0 | none     | base/16573/2
```

- gs_repair_file(tableoid Oid, path text, timeout int)

  描述：根据传入的参数修复文件，仅支持有正常主备连接的主DN使用。参数依据gs_verify_data_file函数返回的oid和路径填写。段页式表tableoid赋值为0到4，294，967，295的任意值（内部校验根据文件路径判断是否是段页式表文件，段页式表文件则不使用tableoid）。修复成功返回值为true，修复失败会显示具体失败原因。默认只有在主DN节点上，使用初始用户、具有sysadmin属性的用户以及在运维模式下具有运维管理员属性的用户可以查看，其余用户需要赋权后才可以使用。

  ⚠ 注意

  1. 当DN实例上存在文件损坏时，进行升主会校验出错，报PANIC退出无法升主，为正常现象。
  2. 当文件存在但是大小为0时，此时不会去修复该文件，若想要修复该文件，需要将为0的文件删除后再修复。
  3. 删除文件需要等文件句柄（fd，file descriptor）自动关闭后再修复，人工操作可以执行重启进程，主备切换等命令。

  参数说明：

  – tableoid

    要修复的文件对应的表oid，依据gs_verify_data_file函数返回的列表中rel_oid一列填写。

    取值范围： Oid，0 - 4294967295。注意：输入负值等数值都会被强制转成非负整数类型。

  – path

    需要修复的文件路径，依据gs_verify_data_file函数返回的列表中miss_file_path一列填写。

    取值范围：字符串。

  – timeout

    等待备DN回放的时长，修复文件需要等待备DN回放到当前主DN对应的位置，根据备DN回放所需时长设定。

    取值范围：60s - 3600s。

  返回值类型：bool。

  示例（按照gs_verify_data_file的输出填写tableoid和path）：

```
openGauss=# select * from gs_repair_file(16554,'base/16552/24745',360);
gs_repair_file
----------------
t
```

- local_bad_block_info()

  描述：显示本实例页面损坏的情况。从磁盘读取页面，发现页面CRC校验失败时进行记录。默认只有初始用户、具有sysadmin属性的用户、具有监控管理员属性的用户以及在运维模式下具有运维管理员属性的用户、以及监控用户可以查看，其余用户需要赋权后才可以使用。file_path是损坏文件的相对路径，如果是段页

式表，则显示的是逻辑信息，不是实际的物理文件信息。block_num是该文件损坏的具体页面号，页面号从0开始。check_time表示发现页面损坏的时间。repair_time表示修复页面的时间。

返回值类型：record。

示例（仅当有损坏记录时输出相关条目，否则输出0行。）：

```
openGauss=# select * from local_bad_block_info();
node_name    | spc_node | db_node | rel_node| bucket_node | fork_num | block_num |    file_path     |
check_time            |   repair_time
-----------------+--------+---------+---------+-------------+----------+-----------+-----------------
+---------------------------+-------------------------------
dn_6001_6002_6003|  1663 |  16552 |  24745 |        -1    |    0    | 0        | base/16552/24745 |
2022-01-13 20:19:08.385004+08 | 2022-01-13 20:19:08.407314+08
```

- remote_bad_block_info()

  描述：CN上查询时，查询除本实例以外其他实例页面损坏的情况，记录数据和在其他实例上执行local_bad_block_info函数一致。DN上执行结果为空。默认只有初始用户、具有sysadmin属性的用户、具有监控管理员属性的用户以及在运维模式下具有运维管理员属性的用户、以及监控用户可以查看，其余用户需要赋权后才可以使用。

  返回值类型：record。

- local_clear_bad_block_info()

  描述：清理local_bad_block_info中已修复页面的数据，也就是repair_time不为空的信息。默认只有初始用户、具有sysadmin属性的用户以及在运维模式下具有运维管理员属性的用户、以及监控用户可以查看，其余用户需要赋权后才可以使用。

  返回值类型：bool。

  示例：

  ```
  openGauss=# select * from local_clear_bad_block_info();
  result
  --------
  t
  ```

- remote_clear_bad_block_info()

  描述：CN上执行时，清理除本实例以外其他实例已修复页面的数据，也就是repair_time不为空的信息。DN上执行结果为空。默认只有初始用户、具有sysadmin属性的用户以及在运维模式下具有运维管理员属性的用户、以及监控用户可以查看，其余用户需要赋权后才可以使用。

  返回值类型：record。

- gs_verify_and_tryrepair_page (path text, blocknum Oid, verify_mem bool, is_segment bool)

  描述：校验本实例指定页面的情况。默认只有在主DN节点上，使用初始用户、具有sysadmin属性的用户以及在运维模式下具有运维管理员属性的用户可以查看，其余用户需要赋权后才可以使用。返回的结果信息，disk_page_res表示磁盘上页面的校验结果，mem_page_res表示内存中页面的校验结果，is_repair表示在校验的过程中是否触发修复功能，t表示已修复，f表示未修复。

  注意：

  1. 当DN实例上存在页面损坏时，进行升主会校验出错，报PANIC退出无法升主，为正常现象。不支持hashbucket表页面损坏的修复。

  2. 此函数触发的修复仅支持修复内存中的页面，需要在内存页面落盘后物理页面修复才能正式生效。

  参数说明：

- path

  损坏文件的路径，依据local_bad_block_info中file_path一列填写。如果要对存储类型为USTORE的表进行UNDO页面校验，请直接填写需要校验的UNDO页面路径。

  取值范围：字符串。

- blocknum

  损坏文件的页号，依据local_bad_block_info中block_num一列填写。如果要对存储类型为USTORE的表进行UNDO页面校验，请直接填写需要校验的UNDO页面的块号。

  取值范围：Oid，0 - 4294967295。注意：输入负值等都会被强制转成非负整数类型。

- verify_mem

  指定是否校验内存中的指定页面。设定为false时，只校验磁盘上的页面。设置为true时，校验内存中的页面和磁盘上的页面。如果发现磁盘上页面损坏，会将内存中的页面做一个基本信息校验刷盘，修复磁盘上页面。如果校验内存页面时发现页面不在内存中，会经内存接口读取磁盘上的页面。此过程中如果磁盘页面有问题，则会触发远程读自动修复功能。

  取值范围：bool，true和false。

- is_segment

  是否是段页式表。根据local_bad_block_info中的bucket_node列值决定。如果bucket_node为-1时，表示不是段页式表，将is_segment设置为false；非-1的情况将is_segment设置为true。

  取值范围：bool，true和false。

返回值类型：record。

示例（请依据local_bad_block_info的输出传参，否则报错）：

```
openGauss=# select * from gs_verify_and_tryrepair_page('base/16552/24745',0,false,false);
node_name         |       path      |  blocknum  |         disk_page_res         | mem_page_res | is_repair
------------------+-----------------+------------+-------------------------------+--------------+----------
dn_6001_6002_6003 | base/16552/24745 |     0      | page verification succeeded.|              | f
```

- gs_repair_page(path text, blocknum oid is_segment bool, timeout int)

  描述：修复本实例指定页面，仅支持有正常主备连接的主DN使用。页面修复成功返回true，修复过程中出错会有报错信息提示。默认只有在主DN节点上，使用初始用户、具有sysadmin属性的用户以及在运维模式下具有运维管理员属性的用户可以查看，其余用户需要赋权后才可以使用。

  注意：当DN实例上存在页面损坏时，进行升主会校验出错，报PANIC退出无法升主，为正常现象。不支持hashbucket表页面损坏的修复。

  参数说明：

  - path

    损坏页面的路径。根据local_bad_block_info中file_path一列设置，或者是gs_verify_and_tryrepair_page函数中path一列设置。

    取值范围：字符串。

  - blocknum

    损坏页面的页面号。根据local_bad_block_info中block_num一列设置，或者是gs_verify_and_tryrepair_page函数中blocknum一列设置。

    取值范围：Oid，0 - 4294967295。注意：输入负值等数值都会被强制转成非负整数类型。

- is_segment

  是否是段页式表。根据local_bad_block_info中的bucket_node列值决定，如果bucket_node为-1时，表示不是段页式表，将is_segment设置为false，非-1的情况将is_segment设置为true。

  取值范围：bool，true或者false。

- timeout

  等待备DN回放的时长。修复页面需要等待备DN回放到当前主DN对应的位置，根据备DN回放所需时长设定。

  取值范围：60s - 3600s。

返回值类型：bool。

示例（请根据local_bad_block_info的输出传参，否则报错）：

```
openGauss=# select * from gs_repair_page('base/16552/24745',0,false,60);
result
--------
t
```

7.5.37 废弃函数

GaussDB中下列函数在最新版本中已废弃：

gs_wlm_get_session_info	gs_wlm_get_user_session_info	check_engine_status	encode_plan_node	model_train_opt	gs_stat_get_wlm_plan_operator_info	track_model_train_opt
array_extend	dbe_perf.global_slow_query_info	dbe_perf.global_slow_query_info_bytime	dbe_perf.global_slow_query_history	pg_reload_conf	pg_rotate_logfile	gs_stat_ustore
pgxc_get_searchlet_info()	pgxc_get_searchlet_table_attr_info()	gs_comm_proxy_thread_status()	-	-	-	-

7.6 表达式

7.6.1 简单表达式

逻辑表达式

逻辑表达式的操作符和运算规则，请参见**逻辑操作符**。

比较表达式

常用的比较操作符，请参见**比较操作符**。

除比较操作符外，还可以使用以下句式结构：

- BETWEEN操作符
  a BETWEEN x AND y等效于a >= x AND a <= y
  a NOT BETWEEN x AND y等效于a < x OR a > y
- 检查一个值是不是null，可使用：
  expression IS NULL
  expression IS NOT NULL
  或者与之等价的句式结构，但不是标准的：
  expression ISNULL
  expression NOTNULL

须知

不要写expression=NULL或expression<>(!=)NULL，因为NULL代表一个未知的值，不能通过该表达式判断两个未知值是否相等。

示例

```
openGauss=# SELECT 2 BETWEEN 1 AND 3 AS RESULT;
 result
----------
 t
(1 row)

openGauss=# SELECT 2 >= 1 AND 2 <= 3 AS RESULT;
 result
----------
 t
(1 row)

openGauss=# SELECT 2 NOT BETWEEN 1 AND 3 AS RESULT;
 result
----------
 f
(1 row)

openGauss=# SELECT 2 < 1 OR 2 > 3 AS RESULT;
 result
----------
 f
(1 row)

openGauss=# SELECT 2+2 IS NULL AS RESULT;
 result
----------
 f
(1 row)

openGauss=# SELECT 2+2 IS NOT NULL AS RESULT;
 result
----------
 t
(1 row)

openGauss=# SELECT 2+2 ISNULL AS RESULT;
 result
----------
 f
(1 row)
```



```
openGauss=# SELECT 2+2 NOTNULL AS RESULT;
 result
----------
 t
(1 row)

openGauss=# SELECT 2+2 IS DISTINCT FROM NULL AS RESULT;
 result
----------
 t
(1 row)

openGauss=# SELECT 2+2 IS NOT DISTINCT FROM NULL AS RESULT;
 result
----------
 f
(1 row)
```

7.6.2 条件表达式

在执行SQL语句时，可通过条件表达式筛选出符合条件的数据。

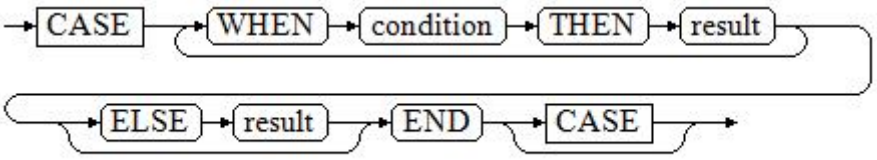
条件表达式主要有以下几种：

- CASE

  CASE表达式是条件表达式，类似于其他编程语言中的CASE语句。

  CASE表达式的语法图请参考图7-1。

  图 7-1 case::=

  CASE子句可以用于合法的表达式中。condition是一个返回BOOLEAN数据类型的表达式：

  - 如果结果为真，CASE表达式的结果就是符合该条件所对应的result。
  - 如果结果为假，则以相同方式处理随后的WHEN或ELSE子句。
  - 如果各WHEN condition都不为真，表达式的结果就是在ELSE子句执行的result。如果省略了ELSE子句且没有匹配的条件，结果为NULL。

  示例：

  ```
  openGauss=# CREATE TABLE case_when_t1(CW_COL1 INT);

  openGauss=# INSERT INTO case_when_t1 VALUES (1), (2), (3);

  openGauss=# SELECT * FROM case_when_t1;
  cw_col1
  ---------
   1
   2
   3
  (3 rows)

  openGauss=# SELECT CW_COL1, CASE WHEN CW_COL1=1 THEN 'one' WHEN CW_COL1=2 THEN
  'two' ELSE 'other' END FROM case_when_t1 ORDER BY 1;
   cw_col1 | case
  ---------+-------
  ```

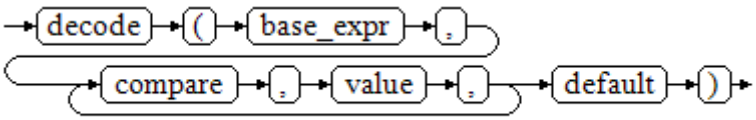


```
     1 | one
     2 | two
     3 | other
(3 rows)

openGauss=# DROP TABLE case_when_t1;
```

- DECODE

  DECODE的语法图请参见**图7-2**。

  图 7-2 decode::=

  将表达式base_expr与后面的每个compare(n) 进行比较，如果匹配返回相应的value(n)。如果没有发生匹配，则返回default。

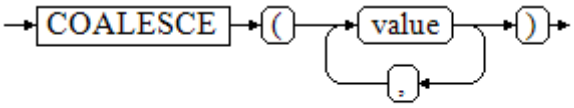
  示例请参见**条件表达式函数**。

```
openGauss=# SELECT DECODE('A','A',1,'B',2,0);
 case
------
    1
(1 row)
```

- COALESCE

  COALESCE的语法图请参见**图7-3**。

  图 7-3 coalesce::=

  COALESCE返回它的第一个非NULL的参数值。如果参数都为NULL，则返回NULL。它常用于在显示数据时用缺省值替换NULL。和CASE表达式一样，COALESCE只计算用来判断结果的参数，即在第一个非空参数右边的参数不会被计算。

  示例

```
openGauss=# CREATE TABLE c_tabl(description varchar(10), short_description varchar(10), last_value
varchar(10)) ;

openGauss=# INSERT INTO c_tabl VALUES('abc', 'efg', '123');
openGauss=# INSERT INTO c_tabl VALUES(NULL, 'efg', '123');

openGauss=# INSERT INTO c_tabl VALUES(NULL, NULL, '123');

openGauss=# SELECT description, short_description, last_value, COALESCE(description,
short_description, last_value) FROM c_tabl ORDER BY 1, 2, 3, 4;
 description | short_description | last_value | coalesce
-------------+-------------------+------------+----------
 abc         | efg               | 123        | abc
             | efg               | 123        | efg
             |                   | 123        | 123
(3 rows)
```



```
openGauss=# DROP TABLE c_tabl;
```

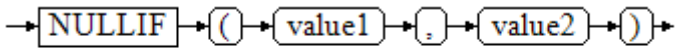
如果description不为NULL，则返回description的值，否则计算下一个参数short_description；如果short_description不为NULL，则返回short_description的值，否则计算下一个参数last_value；如果last_value不为NULL，则返回last_value的值，否则返回（none）。

```
openGauss=# SELECT COALESCE(NULL,'Hello World');
   coalesce
---------------
 Hello World
(1 row)
```

- NULLIF

  NULLIF的语法图请参见图7-4。

  图 7-4 nullif::=

  只有当value1和value2相等时，NULLIF才返回NULL。否则它返回value1。

  示例

```
openGauss=# CREATE TABLE null_if_t1 (
    NI_VALUE1 VARCHAR(10),
    NI_VALUE2 VARCHAR(10)
);

openGauss=# INSERT INTO null_if_t1 VALUES('abc', 'abc');
openGauss=# INSERT INTO null_if_t1 VALUES('abc', 'efg');

openGauss=# SELECT NI_VALUE1, NI_VALUE2, NULLIF(NI_VALUE1, NI_VALUE2) FROM null_if_t1
ORDER BY 1, 2, 3;

 ni_value1 | ni_value2 | nullif
-----------+-----------+--------
 abc       | abc       |
 abc       | efg       | abc
(2 rows)
openGauss=# DROP TABLE null_if_t1;
```

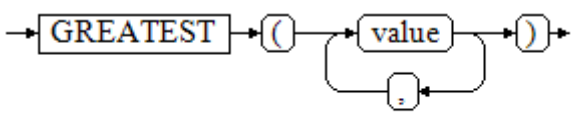
  如果value1等于value2则返回NULL，否则返回value1。

```
openGauss=# SELECT NULLIF('Hello','Hello World');
 nullif
--------
 Hello
(1 row)
```

- GREATEST（最大值），LEAST（最小值）

  GREATEST的语法图请参见图7-5。

  图 7-5 greatest::=

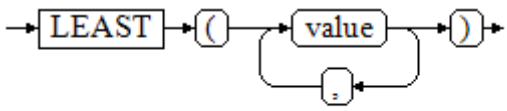
  

  从一个任意数字表达式的列表里选取最大的数值。


```
openGauss=# SELECT greatest(9000,155555,2.01);
 greatest
----------
   155555
(1 row)
```

LEAST的语法图请参见**图7-6**。

图 7-6 least::=



从一个任意数字表达式的列表里选取最小的数值。

以上的数字表达式必须都可以转换成一个普通的数据类型，该数据类型将是结果类型。

列表中的NULL值将被忽略。只有所有表达式的结果都是NULL的时候，结果才是NULL。

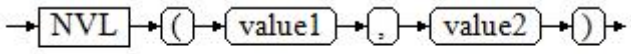
```
openGauss=# SELECT least(9000,2);
 least
-------
     2
(1 row)
```

示例请参见**条件表达式函数**。

- NVL

  NVL的语法图请参见**图7-7**。

  图 7-7 nvl::=

  如果value1为NULL则返回value2，如果value1非NULL，则返回value1。

  示例：

```
openGauss=# SELECT nvl(null,1);
nvl
-----
 1
(1 row)

openGauss=# SELECT nvl ('Hello World',1);
      nvl
---------------
 Hello World
(1 row)
```

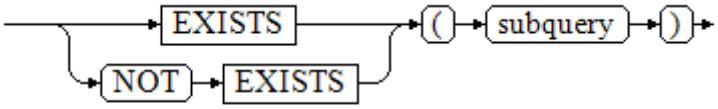
7.6.3 子查询表达式

子查询表达式主要有以下几种：

- EXISTS/NOT EXISTS

  EXISTS/NOT EXISTS的语法图请参见**图7-8**。

图 7-8 EXISTS/NOT EXISTS::=



EXISTS的参数是一个任意的SELECT语句，或者说子查询。系统对子查询进行运算以判断它是否返回行。如果它至少返回一行，则EXISTS结果就为“真”；如果子查询没有返回任何行， EXISTS的结果是“假”。

这个子查询通常只是运行到能判断它是否可以生成至少一行为止，而不是等到全部结束。

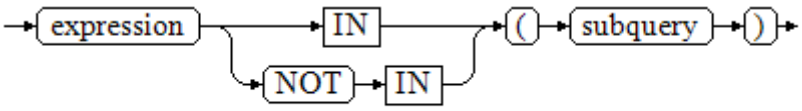
示例：

```
openGauss=# CREATE TABLE exists_t1(a int, b int);
openGauss=# INSERT INTO exists_t1 VALUES(1, 2),(2, 3),(3, 4),(4, 5);

openGauss=# CREATE TABLE exists_t2(a int, c int);
openGauss=# INSERT INTO exists_t2 VALUES(3, 4),(4, 5),(5, 6),(6, 7);

openGauss=# SELECT * FROM exists_t1 t1 WHERE EXISTS (SELECT * FROM exists_t2 t2 WHERE t2.a =
t1.a);
 a | b
---+---
 3 | 4
 4 | 5
(2 rows)

openGauss=# DROP TABLE exists_t1, exists_t2;
```

- IN/NOT IN

  IN/NOT IN的语法请参见**图7-9**。

图 7-9 IN/NOT IN::=



右边是一个圆括号括起来的子查询，它必须只返回一个字段。左边表达式对子查询结果的每一行进行一次计算和比较。如果找到任何相等的子查询行，则IN结果为“真”。如果没有找到任何相等行，则结果为“假”（包括子查询没有返回任何行的情况）。

表达式或子查询行里的NULL遵照SQL处理布尔值和NULL组合时的规则。如果两个行对应的字段都相等且非空，则这两行相等；如果任意对应字段不等且非空，则这两行不等；否则结果是未知（NULL）。如果每一行的结果都是不等或NULL，并且至少有一个NULL，则IN的结果是NULL。

示例：

```
openGauss=# CREATE TABLE in_t1(a int, b int);
openGauss=# INSERT INTO in_t1 VALUES(1, 2),(2, 3),(3, 4),(4, 5);

openGauss=# CREATE TABLE in_t2(a int, c int);
openGauss=# INSERT INTO in_t2 VALUES(3, 4),(4, 5),(5, 6),(6, 7);

openGauss=# SELECT * FROM in_t1 t1 WHERE t1.a IN (SELECT t2.a FROM in_t2 t2);
```

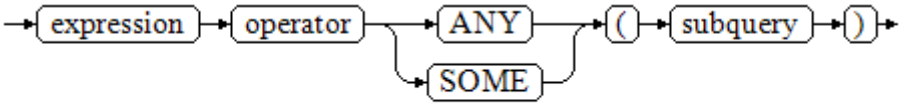


```
 a | b
---+---
 3 | 4
 4 | 5
(2 rows)

openGauss=# DROP TABLE in_t1, in_t2;
```

- ANY/SOME

  ANY/SOME的语法图请参见图7-10。

  图 7-10 any/some::=

  右边是一个圆括号括起来的子查询，它必须只返回一个字段。左边表达式使用operator对子查询结果的每一行进行一次计算和比较，其结果必须是布尔值。如果至少获得一个真值，则ANY结果为“真”。如果全部获得假值，则结果是“假”（包括子查询没有返回任何行的情况）。SOME是ANY的同义词。IN与ANY可以等效替换 。

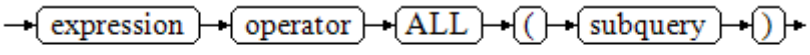
  示例：

```
openGauss=# CREATE TABLE any_t1(a int, b int);
openGauss=# INSERT INTO any_t1 VALUES(1, 2),(2, 3),(3, 4),(4, 5);

openGauss=# CREATE TABLE any_t2(a int, c int);
openGauss=# INSERT INTO any_t2 VALUES(3, 4),(4, 5),(5, 6),(6, 7);

openGauss=# SELECT * FROM any_t1 t1 WHERE t1.a < ANY(SELECT t2.a FROM any_t2 t2 where t2.a =
3 or t2.a = 4);
 a | b
---+---
 1 | 2
 2 | 3
 3 | 4
(3 rows)

openGauss=# DROP TABLE any_t1, any_t2;
```

- ALL

  ALL的语法请参见图7-11。

  图 7-11 all::=

  右边是一个圆括号括起来的子查询，它必须只返回一个字段。左边表达式使用operator对子查询结果的每一行进行一次计算和比较，其结果必须是布尔值。如果全部获得真值，ALL结果为“真”（包括子查询没有返回任何行的情况）。如果至少获得一个假值，则结果是“假”。

  示例：

```
openGauss=# CREATE TABLE all_t1(a int, b int);
openGauss=# INSERT INTO all_t1 VALUES(1, 2),(2, 3),(3, 4),(4, 5);
```



```
openGauss=# CREATE TABLE all_t2(a int, c int);
openGauss=# INSERT INTO all_t2 VALUES(3, 4),(4, 5),(5, 6),(6, 7);

openGauss=# SELECT * FROM all_t1 t1 WHERE t1.a < ALL(SELECT t2.a FROM all_t2 t2 where t2.a = 3
or t2.a = 4);
 a | b
---+---
 1 | 2
 2 | 3
(2 rows)

openGauss=# DROP TABLE all_t1, all_t2;
```

7.6.4 数组表达式

IN

expression **IN** *(value [, ...])*

右侧括号中的是一个表达式列表。左侧表达式的结果与表达式列表的内容进行比较。如果列表中的内容符合左侧表达式的结果，则IN的结果为true。如果没有相符的结果，则IN的结果为false。

示例如下：

```
openGauss=# SELECT 8000+500 IN (10000, 9000) AS RESULT;
 result
----------
 f
(1 row)
```

说明

如果表达式结果为null，或者表达式列表不符合表达式的条件且右侧表达式列表返回结果至少一处为空，则IN的返回结果为null，而不是false。这样的处理方式和SQL返回空值的布尔组合规则是一致的。

NOT IN

expression **NOT IN** *(value [, ...])*

右侧括号中的是一个表达式列表。左侧表达式的结果与表达式列表的内容进行比较。如果在列表中的内容没有符合左侧表达式结果的内容，则NOT IN的结果为true。如果有符合的内容，则NOT IN的结果为false。

示例如下：

```
openGauss=# SELECT 8000+500 NOT IN (10000, 9000) AS RESULT;
 result
----------
 t
(1 row)
```

说明

- 如果查询语句返回结果为空，或者表达式列表不符合表达式的条件且右侧表达式列表返回结果至少一处为空，则NOT IN的返回结果为null，而不是false。这样的处理方式和SQL返回空值的布尔组合规则是一致的。
- 在所有情况下X NOT IN Y等价于NOT(X IN Y)。

ANY/SOME (array)

expression operator **ANY** *(array expression)*

expression operator **SOME** *(array expression)*

右侧括号中的是一个数组表达式，它必须产生一个数组值。左侧表达式的结果使用操作符对数组表达式的每一行结果都进行计算和比较，比较结果必须是布尔值。

```
openGauss=# SELECT 8000+500 < SOME (array[10000,9000]) AS RESULT;
 result
----------
 t
(1 row)
openGauss=# SELECT 8000+500 < ANY (array[10000,9000]) AS RESULT;
 result
----------
 t
(1 row)
```

说明

- 如果对比结果至少获取一个真值，则ANY的结果为true。
- 如果对比结果没有真值，则ANY的结果为false。
- 如果结果没有真值，并且数组表达式生成至少一个值为null，则ANY的值为NULL，而不是false。这样的处理方式和SQL返回空值的布尔组合规则是一致的。
- SOME是ANY的同义词。

SOME是ANY的同义词。

ALL (array)

expression operator **ALL** *(array expression)*

右侧括号中的是一个数组表达式，它必须产生一个数组值。左侧表达式的结果使用操作符对数组表达式的每一行结果都进行计算和比较，比较结果必须是布尔值。

- 如果所有的比较结果都为真值（包括数组不含任何元素的情况），则ALL的结果为true。
- 如果存在一个或多个比较结果为假值，则ALL的结果为false。

如果数组表达式产生一个NULL数组，则ALL的结果为NULL。如果左边表达式的值为NULL，则ALL的结果通常也为NULL(某些不严格的比较操作符可能得到不同的结果)。另外，如果右边的数组表达式中包含null元素并且比较结果没有假值，则ALL的结果将是NULL(某些不严格的比较操作符可能得到不同的结果)， 而不是“真”。这样的处理方式和SQL返回空值的布尔组合规则是一致的。

```
openGauss=# SELECT 8000+500 < ALL (array[10000,9000]) AS RESULT;
 result
----------
 t
(1 row)
```

7.6.5 行表达式

语法：

row_constructor operator row_constructor

两边都是一个行构造器，两行值必须具有相同数目的字段，每一行都进行比较，行比较允许使用=，<>，<，<=，>=等操作符，或其中一个相似的语义符。

对于<，<=，>，> =的情况下，行中元素从左到右依次比较，直到遇到一对不相等的元素或者一对为空的元素。如果这对元素中存在至少一个null值，则比较结果是未知的（null），否则这对元素的比较结果为最终的结果。如果最终没有遇到不相等或者为空的元素，则认为这两行值相等，根据操作符含义判断最终结果。

不支持对XML类型数据操作。

示例：

```
openGauss=# SELECT ROW(1,2,NULL) < ROW(1,3,0) AS RESULT;
 result
----------
 t
(1 row)

openGauss=# select (4,5,6) > (3,2,1) as result;
 result
--------
 t
(1 row)

openGauss=# select (4,1,1) > (3,2,1) as result;
 result
--------
 t
(1 row)

openGauss=# select ('test','data') > ('data','data') as result;
 result
--------
 t
(1 row)

openGauss=# select (4,1,1) > (3,2,null) as result;
 result
--------
 t
(1 row)

openGauss=# select (null,1,1) > (3,2,1) as result;
 result
--------

(1 row)

openGauss=# select (null,5,6) > (null,5,6) as result;
 result
--------

(1 row)

openGauss=# select (4,5,6) > (4,5,6) as result;
 result
--------
 f
(1 row)

openGauss=# select (2,2,5) >= (2,2,3) as result;
 result
--------
 t
(1 row)

openGauss=# select (2,2,1) <= (2,2,3) as result;
 result
```



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

=，<>和别的操作符使用略有不同。如果两行值的所有字段都是非空并且符合操作符条件，则认为两行是符合操作符条件的；如果两行值的任意字段为非空并且不符合操作符条件，则认为两行是不符合操作符条件的；如果两行值的任意字段为空，则比较的结果是未知的（null）。

示例：

```
openGauss=# select (1,2,3) = (1,2,3) as result;
 result
--------
 t
(1 row)

openGauss=# select (1,2,3) <> (2,2,3) as result;
 result
--------
 t
(1 row)

openGauss=# select (2,2,3) <> (2,2,null) as result;
 result
--------

(1 row)

openGauss=# select (null,5,6) <> (null,5,6) as result;
 result
--------

(1 row)
```

7.7 类型转换

7.7.1 概述

背景信息

在SQL语言中，每个数据都与一个决定其行为和用法的数据类型相关。GaussDB提供一个可扩展的数据类型系统，该系统比其它SQL实现更具通用性和灵活性。因而，GaussDB中大多数类型转换是由通用规则来管理的，这种做法允许使用混合类型的表达式。

GaussDB扫描/分析器只将词法元素分解成五个基本种类：整数、浮点数、字符串、标识符和关键字。大多数非数字类型首先表现为字符串。SQL语言的定义允许将常量字符串声明为具体的类型。例，下面查询：

```
openGauss=# SELECT text 'Origin' AS "label", point '(0,0)' AS "value";
 label  | value
--------+-------
 Origin | (0,0)
(1 row)
```

示例中有两个文本常量，类型分别为text和point。如果没有为字符串文本声明类型，则该文本首先被定义成一个unknown类型。

在GaussDB分析器里，有四种基本的SQL结构需要独立的类型转换规则：

- 函数调用

  多数SQL类型系统是建筑在一套丰富的函数上的。函数调用可以有一个或多个参数。因为SQL允许函数重载，所以不能通过函数名直接找到要调用的函数，分析器必须根据函数提供的参数类型选择正确的函数。

- 操作符

  SQL允许在表达式上使用前缀或后缀（单目）操作符，也允许表达式内部使用双目操作符（两个参数）。像函数一样，操作符也可以被重载，因此操作符的选择也和函数一样取决于参数类型。

- 值存储

  INSERT和UPDATE语句将表达式结果存入表中。语句中的表达式类型必须和目标字段的类型一致或者可以转换为一致。

- UNION，CASE和相关构造

  因为联合SELECT语句中的所有查询结果必须在一列里显示出来，所以每个SELECT子句中的元素类型必须相互匹配并转换成一个统一类型。类似地，一个CASE构造的结果表达式必须转换成统一的类型，这样整个case表达式会有一个统一的输出类型。同样的要求也存在于ARRAY构造以及GREATEST和LEAST函数中。

系统表pg_cast存储了有关数据类型之间的转换关系以及如何执行这些转换的信息。详细信息请参见**PG_CAST**。

语义分析阶段会决定表达式的返回值类型并选择适当的转换行为。数据类型的基本类型分类，包括：Boolean，numeric，string，bitstring，datetime，timespan，geometric和network。每种类型都有一种或多种首选类型用于解决类型选择的问题。根据首选类型和可用的隐含转换，就可能保证有歧义的表达式（那些有多个候选解析方案的）得到有效的方式解决。

所有类型转换规则都是建立在下面几个基本原则上的：

- 隐含转换绝不能有奇怪的或不可预见的输出。
- 如果一个查询不需要隐含的类型转换，分析器和执行器不应该进行更多的额外操作。这就是说，任何一个类型匹配、格式清晰的查询不应该在分析器里耗费更多的时间，也不应该向查询中引入任何不必要的隐含类型转换调用。
- 另外，如果一个查询在调用某个函数时需要进行隐式转换，当用户定义了一个有正确参数的函数后，解释器应该选择使用新函数。

7.7.2 操作符

操作符类型解析

1. 从系统表pg_operator中选出要考虑的操作符。如果可以找到一个参数类型以及参数个数都一致的操作符，那么这个操作符就是最终使用的操作符。如果找到了多个备选的操作符，将从中选择一个最合适的。
2. 寻找最优匹配。
   a. 抛弃那些输入类型不匹配并且也不能隐式转换成匹配的候选操作符。unknown文本在这种情况下可以转换成任何东西。如果只剩下一个候选项，则用之，否则继续下一步。
   b. 遍历所有候选操作符，保留那些输入类型匹配最准确的。此时，域类型看做和域类型的基本类型相同。如果没有一个操作符能被保留，则保留所有候选。如果只剩下一个候选项，则用之，否则继续下一步。

c. 遍历所有候选操作符，保留那些需要类型转换时接受(属于输入数据类型的类型范畴的)首选类型位置最多的操作符。如果没有接受首选类型的操作符，则保留所有候选。如果只剩下一个候选项，则用之，否则继续下一步。

d. 如果有任何输入参数是unknown类型，检查剩余的候选操作符对应参数位置的类型范畴。在每一个能够接受字符串类型范畴的位置使用string类型（这种对字符串的偏爱是合适的，因为unknown文本确实像字符串）。另外，如果所有剩下的候选操作符都接受相同的类型范畴，则选择该类型范畴，否则抛出一个错误（因为在没有更多线索的条件下无法作出正确的选择）。现在抛弃不接受选定的类型范畴的候选操作符，然后，如果任意候选操作符在某个给定的参数位置接受一个首选类型，则抛弃那些在该参数位置接受非首选类型的候选操作符。如果没有一个操作符能被保留，则保留所有候选。如果只剩下一个候选项，则用之，否则继续下一步。

e. 如果同时有unknown和已知类型的参数，并且所有已知类型的参数都是相同的类型，那么假设unknown参数也是那种类型，并检查哪个候选操作符在unknown参数位置接受那个类型。如果只有一个操作符符合，那么使用它。否则，产生一个错误。

⚠ 注意

在找到一个操作符后，如果输入的参数类型和操作符的参数类型不一致，可能会发生隐式类型转换，转换后可能发生不可预知的行为。如果隐式转换后行为有问题，可以通过显式类型转换规避此问题。例如，定长类型bpchar转换为变长类型text后，会消除字符串行尾空格，如果再和其它字符串比较时可能会发生错误行为。

示例

示例1：阶乘操作符类型解析。在系统表中里只有一个阶乘操作符（后缀!），它以bigint作为参数。扫描器给下面查询表达式的参数赋予bigint的初始类型：

```
openGauss=# SELECT 40 ! AS "40 factorial";

                  40 factorial
-----------------------------------------------
 815915283247897734345611269596115894272000000000
(1 row)
```

分析器对参数做类型转换，查询等效于：

```
openGauss=# SELECT CAST(40 AS bigint) ! AS "40 factorial";
```

示例2：字符串连接操作符类型分析。一种字符串风格的语法既可以用于字符串也可以用于复杂的扩展类型。未声明类型的字符串将被所有可能的候选操作符匹配。有一个未声明的参数的例子：

```
openGauss=# SELECT text 'abc' || 'def' AS "text and unknown";
 text and unknown
------------------
 abcdef
(1 row)
```

本例中分析器寻找两个参数都是text的操作符。确实有这样的操作符，两个参数都是text类型。

下面是连接两个未声明类型的值：

```
openGauss=# SELECT 'abc' || 'def' AS "unspecified";
 unspecified
```



```
-------------
 abcdef
(1 row)
```

说明

因为查询中没有声明任何类型，所以本例中对类型没有任何初始提示。因此，分析器查找所有候选操作符，发现既存在接受字符串类型范畴的操作符也存在接受位串类型范畴的操作符。因为字符串类型范畴是首选，所以选择字符串类型范畴的首选类型text作为解析未知类型文本的声明类型。

示例3：绝对值和取反操作符类型分析。GaussDB操作符表里面有几条记录对应于前缀操作符@，它们都用于为各种数值类型实现绝对值操作。其中之一用于float8类型，它是数值类型范畴中的首选类型。因此，在面对unknown输入的时候， GaussDB会使用该类型：

```
openGauss=# SELECT @ '-4.5' AS "abs";
 abs
-----
 4.5
(1 row)
```

此处，系统在应用选定的操作符之前隐式的转换unknown类型的文字为float8类型。

示例4：数组包含操作符类型分析。这里是解决一个操作符带有一个已知和一个未知类型输入的例子：

```
openGauss=# SELECT array[1,2] <@ '{1,2,3}' as "is subset";
 is subset
-----------
 t
(1 row)
```

说明

GaussDB操作符表有几条记录对应于中缀操作符<@，但是只有两个可以在左侧接受一个整数数组的操作符是数组包含(anyarray <@ anyarray) 和范围包含(anyelement <@ anyrange)的。因为没有多态的伪类型(参阅**伪类型**)是首选的，所以解析器不能解决这个基础上的歧义。然而，最后一个解析规则告诉用户，假设未知类型的文字是和另外一个输入相同的类型，也就是，整数数组。现在只有两个操作符中的一个可以匹配，所以选择数组包含。（如果用户选择了范围包含，用户将得到一个错误，因为字符串没有正确的格式成为范围的文字。）

7.7.3 函数

函数类型解析

1. 从系统表pg_proc中选择所有可能被选到的函数。如果使用了一个不带模式修饰的函数名称，那么认为该函数是那些在当前搜索路径中的函数。如果给出一个带修饰的函数名，那么只考虑指定模式中的函数。

   如果搜索路径中找到了多个不同参数类型的函数。将从中选择一个合适的函数。
2. 查找和输入参数类型完全匹配的函数。如果找到一个，则用之。如果输入的实参类型都是unknown类型，则不会找到匹配的函数。
3. 如果未找到完全匹配，请查看该函数是否为一个特殊的类型转换函数。
4. 寻找最优匹配。

   a. 抛弃那些输入类型不匹配并且也不能隐式转换成匹配的候选函数。unknown文本在这种情况下可以转换成任何东西。如果只剩下一个候选项，则用之，否则继续下一步。

b. 遍历所有候选函数，保留那些输入类型匹配最准确的。此时，域被看作和它们的基本类型相同。如果没有一个函数能准确匹配，则保留所有候选。如果只剩下一个候选项，则用之，否则继续下一步。

c. 遍历所有候选函数，保留那些需要类型转换时接受首选类型位置最多的函数。如果没有接受首选类型的函数，则保留所有候选。如果只剩下一个候选项，则用之，否则继续下一步。

d. 如果有任何输入参数是unknown类型，检查剩余的候选函数对应参数位置的类型范畴。在每一个能够接受字符串类型范畴的位置使用string类型（这种对字符串的偏爱是合适的，因为unknown文本确实像字符串）。另外，如果所有剩下的候选函数都接受相同的类型范畴，则选择该类型范畴，否则抛出一个错误（因为在没有更多线索的条件下无法做出正确的选择）。现在抛弃不接受选定的类型范畴的候选函数，然后，如果任意候选函数在那个范畴接受一个首选类型，则抛弃那些在该参数位置接受非首选类型的候选函数。如果没有一个候选符合这些测试则保留所有候选。如果只有一个候选函数符合，则使用它；否则，继续下一步。

e. 如果同时有unknown和已知类型的参数，并且所有已知类型的参数有相同的类型，假设unknown参数也是这种类型，检查哪个候选函数可以在unknown参数位置接受这种类型。如果正好一个候选符合，那么使用它。否则，产生一个错误。

示例

示例1：圆整函数参数类型解析。只有一个round函数有两个参数（第一个是numeric，第二个是integer）。所以下面的查询自动把第一个类型为integer的参数转换成numeric类型。

```
openGauss=# SELECT round(4, 4);
 round
--------
 4.0000
(1 row)
```

实际上它被分析器转换成：

```
openGauss=# SELECT round(CAST (4 AS numeric), 4);
```

因为带小数点的数值常量初始时被赋予numeric类型，因此下面的查询将不需要类型转换，并且可能会略微高效一些：

```
openGauss=# SELECT round(4.0, 4);
```

示例2：子字符串函数类型解析。有好几个substr函数，其中一个接受text和integer类型。如果用一个未声明类型的字符串常量调用它，系统将选择接受string类型范畴的首选类型（也就是text类型）的候选函数。

```
openGauss=# SELECT substr('1234', 3);
 substr
--------
    34
(1 row)
```

如果该字符串声明为varchar类型，就像从表中取出来的数据一样，分析器将试着将其转换成text类型：

```
openGauss=# SELECT substr(varchar '1234', 3);
 substr
--------
    34
(1 row)
```


被分析器转换后实际上变成：

```
openGauss=# SELECT substr(CAST (varchar '1234' AS text), 3);
```

说明

分析器从pg_cast表中了解到text和varchar是二进制兼容的，意思是说一个可以传递给接受另一个的函数而不需要做任何物理转换。因此，在这种情况下，实际上没有做任何类型转换。

而且，如果以integer为参数调用函数，分析器将试图将其转换成text类型：

```
openGauss=# SELECT substr(1234, 3);
substr
--------
 34
(1 row)
```

被分析器转换后实际上变成：

```
openGauss=# SELECT substr(CAST (1234 AS text), 3);
 substr
--------
    34
(1 row)
```

7.7.4 值存储

值存储数据类型解析

1. 查找与目标字段准确的匹配。
2. 试着将表达式直接转换成目标类型。如果已知这两种类型之间存在一个已注册的转换函数，那么直接调用该转换函数即可。如果表达式是一个未知类型文本，该文本字符串的内容将交给目标类型的输入转换过程。
3. 检查目标类型是否有长度转换。长度转换是一个从某类型到自身的转换。如果在pg_cast表里面找到一个，那么在存储到目标字段之前先在表达式上应用。这样的转换函数总是接受一个额外的类型为integer的参数，它接收目标字段的atttypmod值（实际上是其声明长度，atttypmod的解释随不同的数据类型而不同），并且它可能接受一个Boolean类型的第三个参数，表示转换是显式的还是隐式的。转换函数负责施加那些长度相关的语义，比如长度检查或者截断。

示例

character存储类型转换。对一个目标列定义为character(20)的语句，下面的语句显示存储值的长度正确：

```
openGauss=#  CREATE SCHEMA tpcds;
openGauss=# CREATE TABLE tpcds.value_storage_t1 (
    VS_COL1 CHARACTER(20)
)DISTRIBUTE BY HASH (VS_COL1);
openGauss=# INSERT INTO tpcds.value_storage_t1 VALUES('abcdef');
openGauss=# SELECT VS_COL1, octet_length(VS_COL1) FROM tpcds.value_storage_t1;
       vs_col1        | octet_length
----------------------+--------------
 abcdef               |           20
(1 row)
)

openGauss=# DROP TABLE tpcds.value_storage_t1;
openGauss=#  DROP SCHEMA tpcds;
```


说明

这里真正发生的事情是两个unknown文本缺省解析成text，这样就允许||操作符解析成text连接。然后操作符的text结果转换成bpchar("空白填充的字符型"， character类型内部名称)以匹配目标字段类型。不过，从text到bpchar的转换是二进制兼容的，这样的转换是隐含的并且实际上不做任何函数调用。最后，在系统表里找到长度转换函数bpchar(bpchar, integer, Boolean)并且应用于该操作符的结果和存储的字段长。这个类型相关的函数执行所需的长度检查和额外的空白填充。

7.7.5 UNION，CASE 和相关构造

SQL UNION构造必须把那些可能不太相似的类型匹配起来成为一个结果集。解析算法分别应用于联合查询的每个输出字段。INTERSECT和EXCEPT构造对不相同的类型使用和UNION相同的算法进行解析。CASE、ARRAY、VALUES、GREATEST和LEAST构造也使用同样的算法匹配它的部件表达式并且选择一个结果数据类型。

UNION，CASE 和相关构造解析

- 如果所有输入都是相同的类型，并且不是unknown类型，那么解析成这种类型。
- 如果所有输入都是unknown类型则解析成text类型（字符串类型范畴的首选类型）。否则，忽略unknown输入。
- 如果输入不属于同一个类型范畴，失败。（unknown类型除外）
- 如果输入类型是同一个类型范畴，则选择该类型范畴的首选类型。（例外：union操作会选择第一个分支的类型作为所选类型。）

  说明

  系统表pg_type中typcategory表示数据类型范畴， typispreferred表示是否是typcategory分类中的首选类型。
- 把所有输入转换为所选的类型（对于字符串保持原有长度）。如果从给定的输入到所选的类型没有隐式转换则失败。
- 若输入中含json、txid_snapshot、sys_refcursor或几何类型，则不能进行union。

对于 case 和 coalesce，在 TD 兼容模式下的处理

- 如果所有输入都是相同的类型，并且不是unknown类型，那么解析成这种类型。
- 如果所有输入都是unknown类型则解析成text类型。
- 如果输入字符串（包括unknown，unknown当text来处理）和数字类型，那么解析成字符串类型，如果是其他不同的类型范畴，则报错。
- 如果输入类型是同一个类型范畴，则选择该类型的优先级较高的类型。
- 把所有输入转换为所选的类型。如果从给定的输入到所选的类型没有隐式转换则失败。

对于 case，在 ORA 兼容模式下的处理

“decode(expr, search1, result1, search2, result2, ..., defresult)”，在设置参数sql_beta_feature = a_style_coerce时，按ORA兼容模式下的处理，将整个表达式最终的返回值类型定为result1的数据类型，或者与result1同类型范畴的更高精度的数据类型。（例如，numeric与int同属数值类型范畴，但numeric比int精度要高，具有更高优先级）。对于CASE WHEN，ORA兼容性下与默认行为相同。

- 如果所有输入都是相同的类型，并且不是unknown类型，那么解析成这种类型。否则，进入后续步骤。

- 将result1的数据类型置为最终的返回值类型preferType，其所属类型范畴为preferCategory。
- 依次考虑result2、result3直至defresult的数据类型。如果其类型范畴也是preferCategory，即与result1具有相同的类型范畴，则判断其精度（优先级）是否高于preferType，如果高于，则将preferType更新为更高精度的数据类型；如果其类型范畴不是preferCategory，则判断其数据类型是否可以隐式转换为preferType，不可以则报错。
- 将最终preferType记录的数据类型作为整个表达式最终的返回值类型；表达式的结果向此类型进行隐式转换。

注1：

为了兼容一种特殊情况，即表示了超大数字的字符类型向数值类型转换的情况，例如select decode(1, 2, 2, '534654656764654546575676786 76')，大数超过了bigint、double等的表示范围。所以，当result1的类型范畴为数值类型，且不满足上述"所有输入都是相同的类型"条件时，将返回值的类型直接置为numeric，以兼容此种特殊情况。

注2：

数值类型的优先级排序：numeric>float8>float4>int8>int4>int2>int1

字符类型的优先级排序：text>varchar=nvarchar2>bpchar>char

日期类型的优先级排序：
timestamptz>timestamp>smalldatetime>date>abstime>timetz>time

日期跨度类型的优先级排序：interval>tinterval>reltime

注3：

ORA兼容模式，当参数set sql_beta_feature的值设置为'a_style_coerce'时，所支持的隐式类型转换见下图，\代表不需要转换，yes表示支持，空白表示不支持：

	bool	int1	int2	int4	int8	float4	float8	numeric	money	char	bpchar	varchar2	nvarchar2	text/clob	raw	blob	date	time	timetz	timestamp	timestamptz	smalldatetime	interval	reltime	abstime
bool	\																								
int1		\	yes	yes	yes	yes	yes	yes		yes	yes	yes	yes	yes											
int2		yes	\	yes	yes	yes	yes	yes		yes	yes	yes	yes	yes											
int4		yes	yes	\	yes	yes	yes	yes		yes	yes	yes	yes	yes											
int8		yes	yes	yes	\	yes	yes	yes		yes	yes	yes	yes	yes											
float4		yes	yes	yes	yes	\	yes	yes		yes	yes	yes	yes	yes											
float8		yes	yes	yes	yes	yes	\	yes		yes	yes	yes	yes	yes											
numeric		yes	yes	yes	yes	yes	yes	\		yes	yes	yes	yes	yes											
money									\																
char		yes	yes	yes	yes	yes	yes	yes		\	yes	yes	yes	yes											
bpchar		yes	yes	yes	yes	yes	yes	yes		yes	\	yes	yes	yes											
varchar2		yes	yes	yes	yes	yes	yes	yes		yes	yes	\	yes	yes	yes										
nvarchar2		yes	yes	yes	yes	yes	yes	yes		yes	yes	yes	\	yes											
text/clob		yes	yes	yes	yes	yes	yes	yes		yes	yes	yes	yes	\											
raw												yes		yes	\	yes									
blob															yes	\									
date												yes	yes	yes			\			yes	yes	yes			yes
time														yes				\	yes						
timetz														yes				yes	\						
timestamp												yes	yes	yes			yes			\	yes	yes			yes
timestamptz														yes			yes			yes	\	yes			yes
smalldatetime												yes		yes			yes			yes	yes	\			yes
interval												yes	yes	yes									\	yes	
reltime														yes									yes	\	
abstime														yes			yes			yes	yes	yes			\

示例

示例1：Union中的待定类型解析。这里，unknown类型文本'b'将被解析成text类型。

```
openGauss=# SELECT text 'a' AS "text" UNION SELECT 'b';
 text
------
 a
 b
(2 rows)
```

示例2：简单Union中的类型解析。文本1.2的类型为numeric，而且integer类型的1可以隐含地转换为numeric，因此使用这个类型。

```
openGauss=# SELECT 1.2 AS "numeric" UNION SELECT 1;
 numeric
---------
       1
     1.2
(2 rows)
```

示例3：转置Union中的类型解析。这里，因为类型real不能被隐含转换成integer，但是integer可以隐含转换成real，那么联合的结果类型将是real。

```
openGauss=# SELECT 1 AS "real" UNION SELECT CAST('2.2' AS REAL);
 real
------
    1
  2.2
(2 rows)
```

示例4：TD模式下，coalesce参数输入int和varchar类型，那么解析成varchar类型。ORA模式下会报错。

```
--在oracle模式下，创建ORA兼容模式的数据库oracle_1。
openGauss=# CREATE DATABASE oracle_1 dbcompatibility = 'ORA';

--切换数据库为oracle_1。
openGauss=# \c oracle_1

--创建表t1。
oracle_1=# CREATE TABLE t1(a int, b varchar(10));

--查看coalesce参数输入int和varchar类型的查询语句的执行计划。
oracle_1=# EXPLAIN SELECT coalesce(a, b) FROM t1;
ERROR:  COALESCE types integer and character varying cannot be matched
LINE 1: EXPLAIN SELECT coalesce(a, b) FROM t1;
                                ^
CONTEXT:  referenced column: coalesce

--删除表。
oracle_1=# DROP TABLE t1;

--切换数据库为testdb。
oracle_1=# \c testdb

--在TD模式下，创建TD兼容模式的数据库td_1。
openGauss=# CREATE DATABASE td_1 dbcompatibility = 'TD';

--切换数据库为td_1。
openGauss=# \c td_1

--创建表t2。
td_1=# CREATE TABLE t2(a int, b varchar(10));

--查看coalesce参数输入int和varchar类型的查询语句的执行计划。
td_1=# EXPLAIN VERBOSE select coalesce(a, b) from t2;
```



```
                                QUERY PLAN
-------------------------------------------------------------------------------------------
 Data Node Scan  (cost=0.00..0.00 rows=0 width=0)
   Output: (COALESCE((t2.a)::character varying, t2.b))
   Node/s: All datanodes
   Remote query: SELECT COALESCE(a::character varying, b) AS "coalesce" FROM public.t2
(4 rows)

--删除表。
td_1=# DROP TABLE t2;

--切换数据库为testdb。
td_1=# \c testdb

--删除Oracle和TD模式的数据库。
openGauss=# DROP DATABASE oracle_1;
openGauss=# DROP DATABASE td_1;
```

示例5：ORA模式下，将整个表达式最终的返回值类型定为result1的数据类型，或者与result1同类型范畴的更高精度的数据类型。

```
--在ORA模式下，创建ORA兼容模式的数据库ora_1。
openGauss=# CREATE DATABASE ora_1 dbcompatibility = 'A';

--切换数据库为ora_1。
openGauss=# \c ora_1

--开启Decode兼容性参数。
set sql_beta_feature='a_style_coerce';

--创建表t1。
ora_1=# CREATE TABLE t1(c_int int, c_float8 float8, c_char char(10), c_text text, c_date date);

--插入数据。
ora_1=# INSERT INTO t1 VALUES(1, 2, '3', '4', date '12-10-2010');

--result1类型为char，defresult类型为text，text精度更高，返回值的类型由char更新为text。
ora_1=# SELECT decode(1, 2, c_char, c_text) AS result, pg_typeof(result) FROM t1;
 result | pg_typeof
--------+-----------
 4      | text
(1 row)

--result1类型为int，属于数值类型范畴，返回值的类型置为numeric。
ora_1=# SELECT decode(1, 2, c_int, c_float8) AS result, pg_typeof(result) FROM t1;
 result | pg_typeof
--------+-----------
      2 | numeric
(1 row)

--不存在defresult数据类型向result1数据类型之间的隐式转换，报错处理。
ora_1=# SELECT decode(1, 2, c_int, c_date) FROM t1;
ERROR:  CASE types integer and timestamp without time zone cannot be matched
LINE 1: SELECT decode(1, 2, c_int, c_date) FROM t1;
                                   ^
CONTEXT:  referenced column: c_date

--关闭Decode兼容性参数。
set sql_beta_feature='none';

--删除表。
ora_1=# DROP TABLE t1;
DROP TABLE

--切换数据库为testdb。
ora_1=# \c testdb

--删除ORA模式的数据库。
openGauss=# DROP DATABASE ora_1;
DROP DATABASE
```


7.8 系统操作

GaussDB通过SQL语句执行不同的系统操作，比如：设置变量，显示执行计划和垃圾收集等操作。

设置变量

设置会话或事务中需要使用的各种参数，请参考**SET**。

显示执行计划

显示GaussDB为SQL语句规划的执行计划，请参考**EXPLAIN**。

事务日志检查点

预写式日志（WAL）缺省时在事务日志中每隔一段时间放置一个检查点。CHECKPOINT会立即进行检查，而不是等到下一次调度时的检查点。请参考**CHECKPOINT**。

垃圾收集

进行垃圾收集以及可选择的对数据库进行分析。请参考**VACUUM**。

收集统计信息

收集与数据库中表内容相关的统计信息。请参考**ANALYZE | ANALYSE**。

设置当前事务的约束检查模式

设置当前事务里的约束检查的特性。请参考**SET CONSTRAINTS**。

7.9 事务控制

事务是用户定义的一个数据库操作序列，这些操作要么全做要么全不做，是一个不可分割的工作单位。

启动事务

GaussDB通过START TRANSACTION和BEGIN语法启动事务，请参考**START TRANSACTION**和**BEGIN**。

设置事务

GaussDB通过SET TRANSACTION或者SET LOCAL TRANSACTION语法设置事务，请参考**SET TRANSACTION**。

提交事务

GaussDB通过COMMIT或者END可完成提交事务的功能，即提交事务的所有操作，请参考**COMMIT | END**。

回滚事务

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

说明

数据库中收到的一次执行请求（不在事务块中），如果含有多条语句，将会被打包成一个事务，如果其中有一个语句失败，那么整个请求都将会被回滚。

7.10 DDL 语法一览表

DDL（Data Definition Language数据定义语言），用于定义或修改数据库中的对象。如：表、索引、视图等。

说明

GaussDB不支持CN不完整时进行DDL操作。例如：集群中有1个CN故障时执行新建数据库、表等操作都会失败。

定义数据库

数据库是组织、存储和管理数据的仓库，而数据库定义主要包括：创建数据库、修改数据库属性，以及删除数据库。所涉及的SQL语句，请参考**表7-97**。

表 7-97 数据库定义相关 SQL

功能	相关SQL
创建数据库	**CREATE DATABASE**
修改数据库属性	**ALTER DATABASE**
删除数据库	**DROP DATABASE**

定义模式

模式是一组数据库对象的集合，主要用于控制对数据库对象的访问。所涉及的SQL语句，请参考**表7-98**。

表 7-98 模式定义相关 SQL

功能	相关SQL
创建模式	**CREATE SCHEMA**
修改模式属性	**ALTER SCHEMA**
删除模式	**DROP SCHEMA**

定义表空间

表空间用于管理数据对象，与磁盘上的一个目录对应。所涉及的SQL语句，请参考**表7-99**。

表 7-99 表空间定义相关 SQL

功能	相关SQL
创建表空间	**CREATE TABLESPACE**
修改表空间属性	**ALTER TABLESPACE**
删除表空间	**DROP TABLESPACE**

定义表

表是数据库中的一种特殊数据结构，用于存储数据对象以及对象之间的关系。所涉及的SQL语句，请参考**表7-100**。

表 7-100 表定义相关 SQL

功能	相关SQL
创建表	**CREATE TABLE**
修改表属性	**ALTER TABLE**
删除表	**DROP TABLE**

定义分区表

分区表是一种逻辑表，数据是由普通表存储的，主要用于提升查询性能。所涉及的SQL语句，请参考**表7-101**。

表 7-101 分区表定义相关 SQL

功能	相关SQL
创建分区表	**CREATE TABLE PARTITION**
创建分区	**ALTER TABLE PARTITION**
修改分区表属性	**ALTER TABLE PARTITION**
删除分区	**ALTER TABLE PARTITION**
删除分区表	**DROP TABLE**

定义索引

索引是对数据库表中一列或多列的值进行排序的一种结构，使用索引可快速访问数据库表中的特定信息。所涉及的SQL语句，请参考表7-102。

表 7-102 索引定义相关 SQL

功能	相关SQL
创建索引	CREATE INDEX
修改索引属性	ALTER INDEX
删除索引	DROP INDEX
重建索引	REINDEX

定义存储过程

存储过程是一组为了完成特定功能的SQL语句集，经编译后存储在数据库中，用户通过指定存储过程的名称并给出参数（如果该存储过程带有参数）来执行它。所涉及的SQL语句，请参考表7-103。

表 7-103 存储过程定义相关 SQL

功能	相关SQL
创建存储过程	CREATE PROCEDURE
删除存储过程	DROP PROCEDURE

定义函数

在GaussDB中，它和存储过程类似，也是一组SQL语句集，使用上没有差别。所涉及的SQL语句，请参考表7-104。

表 7-104 函数定义相关 SQL

功能	相关SQL
创建函数	CREATE FUNCTION
修改函数属性	ALTER FUNCTION
删除函数	DROP FUNCTION

定义视图

视图是从一个或几个基本表中导出的虚表，可用于控制用户对数据访问，请参考表7-105。

表 7-105 视图定义相关 SQL

功能	相关SQL
创建视图	CREATE VIEW
删除视图	DROP VIEW

定义游标

为了处理SQL语句，存储过程进程分配一段内存区域来保存上下文联系。游标是指向上下文区域的句柄或指针。借助游标，存储过程可以控制上下文区域的变化，请参考表7-106。

表 7-106 游标定义相关 SQL

功能	相关SQL
创建游标	CURSOR
移动游标	MOVE
从游标中提取数据	FETCH
关闭游标	CLOSE

7.11 DML 语法一览表

DML（Data Manipulation Language数据操作语言），用于对数据库表中的数据进行操作。如：插入、更新、查询、删除。

插入数据

插入数据是往数据库表中添加一条或多条记录，请参考INSERT。

修改数据

修改数据是修改数据库表中的一条或多条记录，请参考UPDATE。

查询数据

数据库查询语句SELECT是用于在数据库中检索适合条件的信息，请参考SELECT。

删除数据

GaussDB提供了两种删除表数据的语句：删除表中指定条件的数据，请参考DELETE；或删除表的所有数据，请参考TRUNCATE。

TRUNCATE快速地从表中删除所有行，它和在每个表上进行无条件的DELETE有同样的效果，不过因为它不做表扫描，因而快得多。在大表上最有用。

拷贝数据

GaussDB提供了在表和文件之间拷贝数据的语句，请参考**COPY**。

锁定表

GaussDB提供了多种锁模式用于控制对表中数据的并发访问，请参考**LOCK**。

调用函数

GaussDB提供了三个用于调用函数的语句，它们在语法结构上没有差别，请参考**CALL**。

操作会话

用户与数据库之间建立的连接称为会话，请参考**表7-107**。

表 7-107 会话相关 SQL

功能	相关SQL
修改会话	**ALTER SESSION**
结束会话	**ALTER SYSTEM KILL SESSION**

7.12 DCL 语法一览表

DCL（Data Control Language数据控制语言），是用来创建用户角色、设置或更改数据库用户或角色权限的语句。

定义角色

角色是用来管理权限的，从数据库安全的角度考虑，可以把所有的管理和操作权限划分到不同的角色上。所涉及的SQL语句，请参考**表7-108**。

表 7-108 角色定义相关 SQL

功能	相关SQL
创建角色	**CREATE ROLE**
修改角色属性	**ALTER ROLE**
删除角色	**DROP ROLE**

定义用户

用户是用来登录数据库的，通过对用户赋予不同的权限，可以方便地管理用户对数据库的访问及操作。所涉及的SQL语句，请参考**表7-109**。

表 7-109 用户定义相关 SQL

功能	相关SQL
创建用户	**CREATE USER**
修改用户属性	**ALTER USER**
删除用户	**DROP USER**

授权

GaussDB提供了针对数据对象和角色授权的语句，请参考**GRANT**。

收回权限

GaussDB提供了收回权限的语句，请参考**REVOKE**。

设置默认权限

GaussDB允许设置应用于将来创建的对象的权限，请参考**ALTER DEFAULT PRIVILEGES**。

7.13 SQL 语法

7.13.1 ABORT

功能描述

回滚当前事务并且撤销所有当前事务中所做的更改。

作用等同于**ROLLBACK**，早期SQL有用ABORT，现在推荐使用ROLLBACK。

注意事项

在事务外部执行ABORT语句不会影响事务的执行，但是会产生一个警告信息。

语法格式

```
ABORT [ WORK | TRANSACTION ] ;
```

参数说明

- **WORK | TRANSACTION**
  可选关键字，除了增加可读性没有其他任何作用。

示例

```
--创建表customer_demographics_t1。
openGauss=# CREATE TABLE customer_demographics_t1
(
    CD_DEMO_SK              INTEGER               NOT NULL,
```



```
    CD_GENDER               CHAR(1)                 ,
    CD_MARITAL_STATUS       CHAR(1)                   ,
    CD_EDUCATION_STATUS     CHAR(20)                  ,
    CD_PURCHASE_ESTIMATE    INTEGER                   ,
    CD_CREDIT_RATING        CHAR(10)                  ,
    CD_DEP_COUNT            INTEGER                   ,
    CD_DEP_EMPLOYED_COUNT   INTEGER                   ,
    CD_DEP_COLLEGE_COUNT    INTEGER
)
DISTRIBUTE BY HASH (CD_DEMO_SK);

--插入记录。
openGauss=# INSERT INTO customer_demographics_t1 VALUES(1920801,'M', 'U', 'DOCTOR DEGREE', 200,
'GOOD', 1, 0,0);

--开启事务。
openGauss=# START TRANSACTION;

--更新字段值。
openGauss=# UPDATE customer_demographics_t1 SET cd_education_status= 'Unknown';

--终止事务，上面所执行的更新会被撤销掉。
openGauss=# ABORT;

--查询数据。
openGauss=# SELECT * FROM customer_demographics_t1 WHERE cd_demo_sk = 1920801;
cd_demo_sk | cd_gender | cd_marital_status | cd_education_status  | cd_purchase_estimate | cd_credit_rating
| cd_dep_count | cd_dep_employed_count | cd_dep_college_count
------------+-----------+-------------------+----------------------+----------------------+-----------------
+--------------+------------------------+----------------------
   1920801 | M         | U                 | DOCTOR DEGREE        |                  200 | GOOD             |            1
|                  0 |                      0
(1 row)

--删除表。
openGauss=# DROP TABLE customer_demographics_t1;
```

相关链接

SET TRANSACTION，**COMMIT | END**，**ROLLBACK**

7.13.2 ALTER AUDIT POLICY

功能描述

修改统一审计策略。

注意事项

- 只有poladmin，sysadmin或初始用户才能进行此操作。
- 需要打开enable_security_policy开关统一审计策略才可以生效。

语法格式

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name { ADD | REMOVE } { [ privilege_audit_clause ]
[ access_audit_clause ] };
ALTER AUDIT POLICY [ IF EXISTS ] policy_name MODIFY ( filter_group_clause );
ALTER AUDIT POLICY [ IF EXISTS ] policy_name DROP FILTER;
ALTER AUDIT POLICY [ IF EXISTS ] policy_name COMMENTS policy_comments;
ALTER AUDIT POLICY [ IF EXISTS ] policy_name { ENABLE | DISABLE };
```

- privilege_audit_clause：
  ```
  PRIVILEGES ({ DDL | ALL } [ ON LABEL ( resource_label_name [, ... ] ) ])
  ```


- access_audit_clause：
  ```
  ACCESS ({ DML | ALL } [ ON LABEL ( resource_label_name [, ... ] ) ])
  ```
- filter_group_clause
  ```
  FILTER ON { filter_type ( filter_value [, ... ] ) } [, ... ]
  ```

参数说明

- **policy_name**

  审计策略名称，需要唯一，不可重复；

  取值范围：字符串，要符合**标识符命名规范**。
- **DDL**

  指的是针对数据库执行如下操作时进行审计，目前支持：ALTER、ANALYZE、COMMENT、CREATE、DROP、GRANT、REVOKE、SET、SHOW、LOGIN_ACCESS、LOGIN_FAILURE、LOGOUT、LOGIN。
- **DML**

  指的是针对数据库执行如下操作时进行审计，目前支持：COPY、DEALLOCATE、DELETE_P、EXECUTE、REINDEX、INSERT、PREPARE、SELECT、TRUNCATE、UPDATE。
- **ALL**

  指的是上述DDL或DML中支持的所有对数据库的操作。当形式为{ DDL | ALL }时，ALL指所有DDL操作；当形式为{ DML | ALL }时，ALL指所有DML操作。
- **filter_type**

  指定审计策略的过滤信息，过滤类型包括：IP、ROLES、APP。
- **filter_value**

  指具体过滤信息内容。
- **policy_comments**

  用于记录策略相关的描述信息。
- **ENABLE|DISABLE**

  可以打开或关闭统一审计策略。若不指定ENABLE|DISABLE，语句默认为ENABLE。

示例

请参考CREATE AUDIT POLICY的**示例**。

相关链接

CREATE AUDIT POLICY，**DROP AUDIT POLICY**。

7.13.3 ALTER COORDINATOR

功能描述

修改pgxc_node系统表中指定节点的nodeis_active字段值，可以在集群任意一个正常的CN上执行，并且指定在哪些节点上修改系统表。

注意事项

- *ALTER COORDINATOR*是修改系统表的语句，限制只有管理员用户和内部维护模式（例如CM集群管理）可以执行。这个语句是CN剔除特性专用，要配合其他操作，不要单独使用，不建议用户自己执行。
- 该语句执行完成后，需要调用select reload_active_coordinator()更新发生系统表修改的节点连接池信息。

语法格式

```
ALTER COORDINATOR nodename SET status
    WITH (nodename1[, nodename2, nodename3 ...]);
```

参数说明

- **nodename**

  节点名，对应在pgxc_node系统表的一行记录，指定后将修改记录中的nodeis_active字段值。

  取值范围：字符串，只支持CN，并且要保证该节点名在pgxc_node系统表中有对应的记录。

- **status**

  pgxc_node系统表中nodeis_acitve字段的更新值。

  取值范围：

  - FALSE

  - TRUE

- **nodename1[, nodename2, nodename3 ...]**

  该SQL执行的节点范围，ALTER COORDINATOR执行时会自动下发到范围内的所有节点，需要包含当前执行节点。

  取值范围：字符串，只支持CN，要保证该节点名在pgxc_node系统表中有对应的记录，并且节点状态正常，否则SQL执行失败。

示例

集群有3个CN，cn_5001、cn_5002、cn_5003，均处于正常工作状态。

cn_5001发生故障且满足剔除时间要求后，需要将cn_5001从集群中剔除，执行SQL在cn_5002和cn5003节点上刷新pgxc_node系统表中cn_5001对应记录的nodeis_active为false：

```
ALTER COORDINATOR cn_5001 SET False WITH (cn_5002,cn_5003);
```

cn_5001故障解除后，为了在集群中加回cn_5001，执行SQL在cn_5002和cn5003节点上刷新pgxc_node系统表中cn_5001对应记录的nodeis_active为true：

```
ALTER COORDINATOR cn_5001 SET True WITH (cn_5002,cn_5003);
```

7.13.4 ALTER DATABASE

功能描述

修改数据库的属性，包括它的名称、所有者、连接数限制、对象隔离属性等。

注意事项

- 只有数据库的所有者或者被授予了数据库ALTER权限的用户才能执行ALTER DATABASE命令，系统管理员默认拥有此权限。针对所要修改属性的不同，还有以下权限约束：
  - 修改数据库名称，必须拥有CREATEDB权限。
  - 修改数据库所有者，当前用户必须是该DATABASE的所有者或者系统管理员，必须拥有CREATEDB权限，且该用户是新所有者角色的成员。
  - 修改数据库默认表空间，该用户必须拥有新表空间的CREATE权限。这个语句会从物理上将一个数据库原来缺省表空间上的表和索引移至新的表空间。注意不在缺省表空间的表和索引不受此影响。
- 不能重命名当前使用的数据库，如果需要重新命名，须连接至其他数据库上。

语法格式

- 修改数据库的最大连接数。
```
ALTER DATABASE database_name
    [ WITH ] CONNECTION LIMIT connlimit;
```
- 修改数据库名称。
```
ALTER DATABASE database_name
    RENAME TO new_name;
```
- 修改数据库所有者。
```
ALTER DATABASE database_name
    OWNER TO new_owner;
```
- 修改数据库默认表空间。
```
ALTER DATABASE database_name
    SET TABLESPACE new_tablespace;
```

  说明

  如果该数据库中的某些表或对象已经创建在new_tablespace下，则无法将该数据库的默认表空间修改为new_tablespace，执行会报错。

- 修改数据库对象隔离属性。
```
ALTER DATABASE database_name [ WITH ] { ENABLE | DISABLE } PRIVATE OBJECT;
```

  说明

  - 修改数据库的对象隔离属性时须连接至该数据库，否则无法更改。
  - 新创建的数据库，对象隔离属性默认是关闭的。当开启数据库对象隔离属性后，数据库会为系统表PG_CLASS、PG_ATTRIBUTE、PG_PROC、PG_NAMESPACE、PGXC_SLICE和PG_PARTITION自动添加行级访问控制策略，普通用户只能查看这些系统表中有权访问的对象（表、函数、视图、字段等）。对象隔离特性对管理员用户不生效，当开启对象隔离特性后，管理员也可以查看到全量的数据库对象。

参数说明

- **database_name**

  需要修改属性的数据库名称。

  取值范围：字符串，要符合**标识符命名规范**。

- **connlimit**

  数据库可以接收的最大并发连接数（管理员用户连接除外）。

  取值范围：[-1, 2^31-1]的整数，建议填写1~50的整数。-1（缺省）表示没有限制。

- **new_name**

  数据库的新名称。

  取值范围：字符串，要符合**标识符命名规范**。

- **new_owner**

  数据库的新所有者。

  取值范围：字符串，有效的用户名。

- **new_tablespace**

  数据库新的默认表空间，该表空间为数据库中已经存在的表空间。默认的表空间为pg_default。

  取值范围：字符串，有效的表空间名。

- **configuration_parameter**

  - **value**

    把指定的数据库会话参数值设置为给定的值。如果value是DEFAULT或者RESET，则在新的会话中使用系统的缺省设置。OFF关闭设置。

    须知

    当前版本不支持设置数据库级别参数。

    取值范围：字符串

    - DEFAULT
    - OFF
    - RESET

  - **FROM CURRENT**

    根据当前会话连接的数据库设置该参数的值。

- **RESET configuration_parameter**

  重置指定的数据库会话参数值。

  须知

  当前版本不支持重置数据库级别参数。

- **RESET ALL**

  重置全部的数据库会话参数值。

  须知

  当前版本不支持重置数据库级别参数。

说明

- 修改数据库默认表空间，会将旧表空间中的所有表和索引转移到新表空间中，该操作不会影响其他非默认表空间中的表和索引。
- 修改的数据库会话参数值，将在下一次会话中生效。
- 执行完参数设置后，需要手动执行CLEAN CONNECTION清理旧连接，否则可能存在集群节点间参数值不一致。

示例

请参考CREATE DATABASE的**示例**。

相关链接

CREATE DATABASE，**DROP DATABASE**

7.13.5 ALTER DEFAULT PRIVILEGES

功能描述

设置应用于将来创建的对象的权限（这不会影响分配到已有对象中的权限）。

注意事项

目前只支持表（包括视图）、序列、函数，类型的权限更改。

语法格式

```
ALTER DEFAULT PRIVILEGES
   [ FOR { ROLE | USER } target_role [, ...] ]
   [ IN SCHEMA schema_name [, ...] ]
   abbreviated_grant_or_revoke;
```

- 其中abbreviated_grant_or_revoke子句用于指定对哪些对象进行授权或回收权限。

  ```
  grant_on_tables_clause
    | grant_on_sequences_clause
    | grant_on_functions_clause
    | grant_on_types_clause
    | grant_on_client_master_keys_clause
    | grant_on_column_encryption_keys_clause
    | revoke_on_tables_clause
    | revoke_on_sequences_clause
    | revoke_on_functions_clause
    | revoke_on_types_clause
    | revoke_on_client_master_keys_clause
    | revoke_on_column_encryption_keys_clause
  ```

- 其中grant_on_tables_clause子句用于对表授权。

  ```
  GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | ALTER | DROP |
  COMMENT | INDEX | VACUUM }
     [, ...] | ALL [ PRIVILEGES ] }
     ON TABLES
     TO { [ GROUP ] role_name | PUBLIC } [, ...]
     [ WITH GRANT OPTION ]
  ```

- 其中grant_on_sequences_clause子句用于对序列授权。

  ```
  GRANT { { SELECT | UPDATE | USAGE | ALTER | DROP | COMMENT }
     [, ...] | ALL [ PRIVILEGES ] }
     ON SEQUENCES
  ```



```
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ]
```

- 其中grant_on_functions_clause子句用于对函数授权。

```
GRANT { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON FUNCTIONS
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ]
```

- 其中grant_on_types_clause子句用于对类型授权。

```
GRANT { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON TYPES
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ]
```

- 其中grant_on_client_master_keys_clause子句用于对客户端主密钥授权。

```
GRANT { { USAGE  | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON CLIENT_MASTER_KEYS
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ]
```

- 其中grant_on_column_encryption_keys_clause子句用于对列加密密钥授权。

```
GRANT { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON COLUMN_ENCRYPTION_KEYS
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ]
```

- 其中revoke_on_tables_clause子句用于回收表对象的权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | ALTER | DROP | COMMENT |
INDEX | VACUUM }
    [, ...] | ALL [ PRIVILEGES ] }
    ON TABLES
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

- 其中revoke_on_sequences_clause子句用于回收序列的权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | UPDATE | USAGE | ALTER | DROP | COMMENT }
    [, ...] | ALL [ PRIVILEGES ] }
    ON SEQUENCES
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

- 其中revoke_on_functions_clause子句用于回收函数的权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON FUNCTIONS
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

- 其中revoke_on_types_clause子句用于回收类型的权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON TYPES
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

- 其中revoke_on_client_master_keys_clause子句用于回收客户端主密钥的权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON CLIENT_MASTER_KEYS
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

- 其中revoke_on_column_encryption_keys_clause子句用于回收列加密密钥的权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON COLUMN_ENCRYPTION_KEYS
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```


参数说明

- **target_role**
  已有角色的名称。如果省略FOR ROLE/USER，则缺省值为当前角色/用户。
  取值范围：已有角色的名称。
- **schema_name**
  现有模式的名称。
  target_role必须有schema_name的CREATE权限。
  取值范围：现有模式的名称。
- **role_name**
  被授予或者取消权限角色的名称。
  取值范围：已存在的角色名称。

须知

如果想删除一个被赋予了默认权限的角色，有必要恢复改变的缺省权限或者使用DROP OWNED BY来为角色脱离缺省的权限记录。

示例

```
--将创建在模式tpcds里的所有表（和视图）的SELECT权限授予每一个用户。
openGauss=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds GRANT SELECT ON TABLES TO PUBLIC;

--创建用户普通用户jack。
openGauss=# CREATE USER jack PASSWORD 'xxxxxxxxxx';

--将tpcds下的所有表的插入权限授予用户jack。
openGauss=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds GRANT INSERT ON TABLES TO jack;

--撤销上述权限。
openGauss=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds REVOKE SELECT ON TABLES FROM PUBLIC;
openGauss=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds REVOKE INSERT ON TABLES FROM jack;

--删除用户jack。
openGauss=# DROP USER jack CASCADE;

--删除SCHEMA。
openGauss=# DROP SCHEMA tpcds;
```

相关链接

GRANT，**REVOKE**

7.13.6 ALTER DIRECTORY

功能描述

对directory属性进行修改。

注意事项

- 目前只支持修改directory属主。

- 当enable_access_server_directory=off时，只允许初始用户修改directory属主；当enable_access_server_directory=on时，具有SYSADMIN权限的用户和directory对象的属主可以修改directory，且要求该用户是新属主的成员。

语法格式

```
ALTER DIRECTORY directory_name
    OWNER TO new_owner;
```

参数描述

- **directory_name**

  需要修改的目录名称，范围为已经存在的目录名称。

- **new_owner**

  目录的新属主。

示例

```
--创建目录。
openGauss=# CREATE OR REPLACE DIRECTORY  dir  as '/tmp/';

--创建用户。
openGauss=# CREATE USER system PASSWORD '********';

--修改目录的owner。
openGauss=# ALTER DIRECTORY dir OWNER TO system;

--删除目录。
openGauss=# DROP DIRECTORY dir;
```

相关链接

CREATE DIRECTORY，**DROP DIRECTORY**

7.13.7 ALTER FUNCTION

功能描述

修改自定义函数的属性。

注意事项

只有函数的所有者或者被授予了函数ALTER权限的用户才能执行ALTER FUNCTION命令，系统管理员默认拥有该权限。针对所要修改属性的不同，还有以下权限约束：

- 如果函数中涉及对临时表相关的操作，则无法使用ALTER FUNCTION。
- 修改函数的所有者或修改函数的模式，当前用户必须是该函数的所有者或者系统管理员，且该用户是新所有者角色的成员。
- 只有系统管理员和初始化用户可以将function的schema修改成public。

语法格式

- 修改自定义函数的附加参数。

  ```
  ALTER FUNCTION function_name ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    action [ ... ] [ RESTRICT ];
  ```

  其中附加参数action子句语法为。


```
{CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT}
 | {IMMUTABLE | STABLE | VOLATILE}
 | {SHIPPABLE | NOT SHIPPABLE}
 | {NOT FENCED | FENCED}
 | [ NOT ] LEAKPROOF
 | { [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER }
 | AUTHID { DEFINER | CURRENT_USER }
 | COST execution_cost
 | ROWS result_rows
 | SET configuration_parameter { { TO | = } { value | DEFAULT }| FROM CURRENT}
 | RESET {configuration_parameter | ALL}
```

- 修改自定义函数的名称。

```
ALTER FUNCTION function_name ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    RENAME TO new_name;
```

- 修改自定义函数的所属者。

```
ALTER FUNCTION function_name ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    OWNER TO new_owner;
```

- 修改自定义函数的模式。

```
ALTER FUNCTION function_name ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    SET SCHEMA new_schema;
```

参数说明

- **function_name**

  要修改的函数名称。

  取值范围：已存在的函数名。

- **argmode**

  标识该参数是输入、输出参数。

  取值范围：

  - IN：声明入参。
  - OUT：声明出参。
  - INOUT：声明出入参。

- **argname**

  参数名称。

  取值范围：字符串，符合**标识符命名规范**。

- **argtype**

  参数类型。

  取值范围：有效的类型，请参考**数据类型**。

- **CALLED ON NULL INPUT**

  表明该函数的某些参数是NULL的时候可以按照正常的方式调用。缺省时与指定此参数的作用相同。

- **RETURNS NULL ON NULL INPUT**

  STRICT

  STRICT用于指定如果函数的某个参数是NULL，此函数总是返回NULL。如果声明了这个参数，则如果存在NULL参数时不会执行该函数；而只是自动假设一个NULL结果。

  RETURNS NULL ON NULL INPUT和STRICT的功能相同。

- **IMMUTABLE**

  表示该函数在给出同样的参数值时总是返回同样的结果。

- **STABLE**

  表示该函数不能修改数据库，对相同参数值，在同一次表扫描里，该函数的返回值不变，但是返回值可能在不同SQL语句之间变化。

- **VOLATILE**

  表示该函数值可以在一次表扫描内改变，不会做任何优化。

- **SHIPPABLE**
- **NOT SHIPPABLE**

  表示该函数是否可以下推到DN上执行。

  对于IMMUTABLE类型的函数，函数始终可以下推到DN上执行。

  对于STABLE/VOLATILE类型的函数，仅当函数的属性是SHIPPABLE的时候，函数可以下推到DN执行。

- **LEAKPROOF**

  表示该函数没有副作用，指出参数只包括返回值。LEAKROOF只能由系统管理员设置。

- **EXTERNAL**

  （可选）目的是和SQL兼容，这个特性适合于所有函数，而不仅是外部函数

- **SECURITY INVOKER**

  AUTHID CURRENT_USER

  表明该函数将以调用它的用户的权限执行。缺省时与指定此参数的作用相同。

  SECURITY INVOKER和AUTHID CURRENT_USER的功能相同。

- **SECURITY DEFINER**

  AUTHID DEFINER

  声明该函数将以创建它的用户的权限执行。

  AUTHID DEFINER和SECURITY DEFINER的功能相同。

- **COST execution_cost**

  用来估计函数的执行成本。

  execution_cost以cpu_operator_cost为单位。

  取值范围：正数

- **ROWS result_rows**

  估计函数返回的行数。用于函数返回的是一个集合。

  取值范围：正数，默认值是1000行。

- **configuration_parameter**
  - **value**

    把指定的数据库会话参数值设置为给定的值。如果value是DEFAULT或者RESET，则在新的会话中使用系统的缺省设置。OFF关闭设置。

    取值范围：字符串

    - DEFAULT
    - OFF
    - RESET

    - 指定默认值。
  - **FROM CURRENT**
    取当前会话中的值设置为configuration_parameter的值。
- **new_name**
  函数的新名称。要修改函数的所属模式，必须拥有新模式的CREATE权限。
  取值范围：字符串，符合**标识符命名规范**。
- **new_owner**
  函数的新所有者。要修改函数的所有者，新所有者必须拥有该函数所属模式的CREATE权限。
  取值范围：已存在的用户角色。
- **new_schema**
  函数的新模式。
  取值范围：已存在的模式。

示例

请参见CREATE FUNCTION的**示例**。

相关链接

CREATE FUNCTION，**DROP FUNCTION**

7.13.8 ALTER GLOBAL CONFIGURATION

功能描述

新增、修改系统表gs_global_config的key-value值。如果修改的参数已经存在，则修改；如果不存在则新增。

注意事项

- 仅支持数据库初始用户运行此命令。
- 参数名称不能为weak_password、undostoragetype。

语法格式

```
ALTER GLOBAL CONFIGURATION with(name=value, name=value...);
```

参数说明

- **name**
  参数名称，text类型，不能为weak_password、undostoragetype，除此之外没有限制。
- **value**
  参数值，text类型。

示例

```
--插入内容。
openGauss=# ALTER GLOBAL CONFIGURATION with(redis_is_ok = true);
```



```
--查询。
openGauss=# SELECT * FROM gs_global_config;
      name       | value
-----------------+-------
 buckets_len     | 16384
 undostoragetype | page
 redis_is_ok     | true
(3 rows)

--修改内容。
openGauss=# ALTER GLOBAL CONFIGURATION with(redis_is_ok = false);

--查询。
openGauss=# SELECT * FROM gs_global_config;
      name       | value
-----------------+-------
 buckets_len     | 16384
 undostoragetype | page
 redis_is_ok     | false
(3 rows)

--删除内容。
openGauss=# DROP GLOBAL CONFIGURATION redis_is_ok;

--查询。
openGauss=# SELECT * FROM gs_global_config;
      name       | value
-----------------+-------
 buckets_len     | 16384
 undostoragetype | page
(2 rows)
```

相关链接

DROP GLOBAL CONFIGURATION

7.13.9 ALTER GROUP

功能描述

修改一个用户组的属性。

注意事项

ALTER GROUP是ALTER ROLE的别名，非SQL标准语法，不推荐使用，建议用户直接使用ALTER ROLE替代。

语法格式

- 向用户组中添加用户。
  ```
  ALTER GROUP group_name
     ADD USER user_name [, ... ];
  ```
- 从用户组中删除用户。
  ```
  ALTER GROUP group_name
     DROP USER user_name [, ... ];
  ```
- 修改用户组的名称。
  ```
  ALTER GROUP group_name
     RENAME TO new_name;
  ```


参数说明

请参考ALTER ROLE的**参数说明**。

示例

```
--创建用户组。
openGauss=# CREATE GROUP super_users WITH PASSWORD "********";

--创建用户。
openGauss=# CREATE ROLE lche WITH PASSWORD "********";

--创建用户。
openGauss=# CREATE ROLE jim WITH PASSWORD "********";

--向用户组中添加用户。
openGauss=# ALTER GROUP super_users ADD USER lche, jim;

--从用户组中删除用户。
openGauss=# ALTER GROUP super_users DROP USER jim;

--修改用户组的名称。
openGauss=# ALTER GROUP super_users RENAME TO normal_users;

--删除用户。
openGauss=# DROP ROLE lche, jim;

--删除用户组。
openGauss=# DROP GROUP normal_users;
```

相关链接

CREATE GROUP，**DROP GROUP**，**ALTER ROLE**

7.13.10 ALTER INDEX

功能描述

ALTER INDEX用于修改现有索引的定义。

它有几种子形式：

- IF EXISTS

  如果指定的索引不存在，则发出一个notice而不是error。
- RENAME TO

  只改变索引的名称。对存储的数据没有影响。
- SET TABLESPACE

  这个选项会改变索引的表空间为指定表空间，并且把索引相关的数据文件移动到新的表空间里。
- SET ({ STORAGE_PARAMETER = value } [, ...])

  改变索引的一个或多个索引方法特定的存储参数。 需要注意的是索引内容不会被这个命令立即修改，根据参数的不同，可能需要使用REINDEX重建索引来获得期望的效果。
- RESET ({ storage_parameter } [, ...])

  重置索引的一个或多个索引方法特定的存储参数为缺省值。与SET一样，可能需要使用REINDEX来完全更新索引。

- [MODIFY PARTITION index_partition_name] UNUSABLE
  用于设置表或者索引分区上的索引不可用。
- REBUILD [PARTITION index_partition_name]
  用于重建表或者索引分区上的索引。
- RENAME PARTITION
  用于重命名索引分区
- MOVE PARTITION
  用于修改索引分区的所属表空间。

注意事项

只有索引的所有者或者拥有索引所在表的INDEX权限的用户有权限执行此命令，系统管理员默认拥有此权限。

语法格式

- 重命名表索引的名称。
```
ALTER INDEX [ IF EXISTS ] index_name
    RENAME TO new_name;
```
- 修改表索引的所属空间。
```
ALTER INDEX [ IF EXISTS ] index_name
    SET TABLESPACE tablespace_name;
```
- 修改表索引的存储参数。
```
ALTER INDEX [ IF EXISTS ] index_name
    SET ( {storage_parameter = value} [, ... ] );
```
- 重置表索引的存储参数。
```
ALTER INDEX [ IF EXISTS ] index_name
    RESET ( storage_parameter [, ... ] ) ;
```
- 设置表索引或索引分区不可用。
```
ALTER INDEX [ IF EXISTS ] index_name
    [ MODIFY PARTITION index_partition_name ] UNUSABLE;
```
- 重建表索引或索引分区。
```
ALTER INDEX index_name
    REBUILD [ PARTITION index_partition_name ];
```
- 重命名索引分区。
```
ALTER INDEX [ IF EXISTS ] index_name
    RENAME PARTITION index_partition_name TO new_index_partition_name;
```
- 修改索引分区的所属表空间。
```
ALTER INDEX [ IF EXISTS ] index_name
    MOVE PARTITION index_partition_name TABLESPACE new_tablespace;
```

参数说明

- **index_name**
  要修改的索引名。
- **new_name**
  新的索引名。
  取值范围：字符串，且符合**标识符命名规范**。
- **tablespace_name**
  表空间的名称。

取值范围：已存在的表空间。

- **storage_parameter**
  索引方法特定的参数名。
- **value**
  索引方法特定的存储参数的新值。根据参数的不同，这可能是一个数字或单词。
- **new_index_partition_name**
  新索引分区名。
- **index_partition_name**
  索引分区名。
- **new_tablespace**
  新表空间。

示例

请参见CREATE INDEX的**示例**。

相关链接

CREATE INDEX，**DROP INDEX**，**REINDEX**

7.13.11 ALTER LANGUAGE

本版本暂不支持使用该语法。

7.13.12 ALTER MASKING POLICY

功能描述

修改脱敏策略。

注意事项

- 只有poladmin，sysadmin或初始用户才能执行此操作。
- 需要打开enable_security_policy开关脱敏策略才可以生效。

语法格式

- 修改策略描述：
```
ALTER MASKING POLICY policy_name COMMENTS policy_comments;
```
- 修改脱敏方式：
```
ALTER MASKING POLICY policy_name [ADD | REMOVE | MODIFY] masking_actions[, ...]*;
其中masking_action:
    masking_function ON LABEL(label_name[, ...]*)
```
- 修改脱敏策略生效场景：
```
ALTER MASKING POLICY policy_name MODIFY(FILTER ON FILTER_TYPE(filter_value[, ...]*)[, ...]*);
```
- 移除脱敏策略生效场景，使策略对所用场景生效：
```
ALTER MASKING POLICY policy_name DROP FILTER;
```
- 修改脱敏策略开启/关闭：
```
ALTER MASKING POLICY policy_name [ENABLE | DISABLE];
```


参数说明

- **policy_name**
  脱敏策略名称，需要唯一，不可重复。
  取值范围：字符串，要符合**标识符命名规范**。
- **policy_comments**
  需要为脱敏策略添加或修改的描述信息。
- **masking_function**
  指的是预置的八种脱敏方式或者用户自定义的函数，支持模式。
  maskall不是预置函数，硬编码在代码中，不支持\df展示。
  预置时脱敏方式如下：

  ```
  maskall | randommasking | creditcardmasking | basicemailmasking | fullemailmasking |
  shufflemasking | alldigitsmasking | regexpmasking
  ```

- **label_name**
  资源标签名称。
- **FILTER_TYPE**
  指定脱敏策略的过滤信息，过滤类型包括：IP、ROLES、APP。
- **filter_value**
  指具体过滤信息内容，例如具体的IP，具体的APP名称，具体的用户名。
- **ENABLE|DISABLE**
  可以打开或关闭脱敏策略。若不指定ENABLE|DISABLE，语句默认为ENABLE。

示例

```
--创建dev_mask和bob_mask用户。
openGauss=# CREATE USER dev_mask PASSWORD 'xxxxxxxxxx';
openGauss=# CREATE USER bob_mask PASSWORD 'xxxxxxxxxx';

--创建一个表tb_for_masking。
openGauss=# CREATE TABLE tb_for_masking(col1 text, col2 text, col3 text);

--创建资源标签标记敏感列col1。
openGauss=# CREATE RESOURCE LABEL mask_lb1 ADD COLUMN(tb_for_masking.col1);

--创建资源标签标记敏感列col2。
openGauss=# CREATE RESOURCE LABEL mask_lb2 ADD COLUMN(tb_for_masking.col2);

--对访问敏感列col1的操作创建脱敏策略。
openGauss=# CREATE MASKING POLICY maskpol1 maskall ON LABEL(mask_lb1);

--为脱敏策略maskpol1添加描述。
openGauss=# ALTER MASKING POLICY maskpol1 COMMENTS 'masking policy for tb_for_masking.col1';

--修改脱敏策略maskpol1，新增一项脱敏方式。
openGauss=# ALTER MASKING POLICY maskpol1 ADD randommasking ON LABEL(mask_lb2);

--修改脱敏策略maskpol1，移除一项脱敏方式。
openGauss=# ALTER MASKING POLICY maskpol1 REMOVE randommasking ON LABEL(mask_lb2);

--修改脱敏策略maskpol1，修改一项脱敏方式。
openGauss=# ALTER MASKING POLICY maskpol1 MODIFY randommasking ON LABEL(mask_lb1);

--修改脱敏策略maskpol1使之仅对用户dev_mask和bob_mask,客户端工具为gsql，IP地址为'10.20.30.40',
'127.0.0.0/24'场景生效。
openGauss=# ALTER MASKING POLICY maskpol1 MODIFY (FILTER ON ROLES(dev_mask, bob_mask),
APP(gsql), IP('10.20.30.40', '127.0.0.0/24'));
```



```
--修改脱敏策略maskpol1，使之对所有用户场景生效。
openGauss=# ALTER MASKING POLICY maskpol1 DROP FILTER;

--禁用脱敏策略maskpol1。
openGauss=# ALTER MASKING POLICY maskpol1 DISABLE;

--删除脱敏策略。
openGauss=# DROP MASKING POLICY maskpol1;

--删除资源标签。
openGauss=# DROP RESOURCE LABEL mask_lb1, mask_lb2;

--删除表tb_for_masking。
openGauss=# DROP TABLE tb_for_masking;

--删除用户dev_mask,和bob_mask。
openGauss=# DROP USER dev_mask, bob_mask;
```

相关链接

CREATE MASKING POLICY，**DROP MASKING POLICY**。

7.13.13 ALTER MATERIALIZED VIEW

功能描述

更改一个现有物化视图的多个辅助属性。

可用于ALTER MATERIALIZED VIEW的语句形式和动作是ALTER TABLE的一个子集，并且在用于物化视图时具有相同的含义。详见**ALTER TABLE**。

注意事项

- 只有物化视图的所有者有权限执行ALTER MATERIALIZED VIEW命令，系统管理员默认拥有此权限。
- 不支持更改物化视图结构。

语法格式

- 修改物化视图的所属用户。
  ```
  ALTER MATERIALIZED VIEW [ IF EXISTS ] mv_name
    OWNER TO new_owner;
  ```
- 修改物化视图的列。
  ```
  ALTER MATERIALIZED VIEW [ IF EXISTS ] mv_name
    RENAME [ COLUMN ] column_name TO new_column_name;
  ```
- 重命名物化视图。
  ```
  ALTER MATERIALIZED VIEW [ IF EXISTS ] mv_name
    RENAME TO new_name;
  ```

参数说明

- **mv_name**

  一个现有物化视图的名称，可以用模式修饰。

  取值范围：字符串，符合**标识符命名规范**。
- **column_name**

  一个新的或者现有的列的名称。

取值范围：字符串，符合**标识符命名规范**。

- **new_column_name**
  一个现有列的新名称。
- **new_owner**
  该物化视图的新拥有者的用户名。
- **new_name**
  该物化视图的新名称。

示例

```
--创建表。
openGauss=# CREATE TABLE my_table (c1 int, c2 int) WITH(STORAGE_TYPE=ASTORE);

--创建全量物化视图。
openGauss=# CREATE MATERIALIZED VIEW foo AS SELECT * FROM my_table;

--把物化视图foo重命名为bar。
openGauss=# ALTER MATERIALIZED VIEW foo RENAME TO bar;

--删除全量物化视图。
openGauss=# DROP MATERIALIZED VIEW bar;

--删除表my_table。
openGauss=# DROP TABLE my_table;
```

相关链接

CREATE INCREMENTAL MATERIALIZED VIEW，**CREATE MATERIALIZED VIEW**，**DROP MATERIALIZED VIEW**，**REFRESH INCREMENTAL MATERIALIZED VIEW**，**REFRESH MATERIALIZED VIEW**

7.13.14 ALTER NODE

功能描述

修改一个现有节点的定义。

注意事项

ALTER NODE是集群管理工具封装的接口，用来实现集群管理，管理员用户才有权限使用该接口。该接口不建议用户直接使用，以免对集群状态造成影响。

语法格式

```
ALTER NODE nodename WITH
 (
   [ TYPE = nodetype,]
   [ HOST = hostname,]
   [ PORT = portnum,]
   [ HOST1 = 'hostname',]
   [ PORT1 = portnum,]
   [ HOSTPRIMARY [ = boolean ],]
   [ PRIMARY [ = boolean ],]
   [ PREFERRED [ = boolean ],]
   [ SCTP_PORT = portnum,]
   [ CONTROL_PORT = portnum,]
   [ SCTP_PORT1 = portnum,]
   [ CONTROL_PORT1 = portnum, ]
```



```
  [ NODEIS_CENTRAL [ = boolean ]]
);
```

说明

PORT选项指定的端口号为节点间内部通信绑定的端口号，不同于外部客户端连接节点的端口号，可通过pgxc_node表查询。

参数说明

请参见CREATE NODE的**参数说明**。

相关链接

CREATE NODE，**DROP NODE**

7.13.15 ALTER NODE GROUP

功能描述

修改一个node group的信息。

注意事项

- 只有系统管理员或者被授予了node group的ALTER权限的用户可以修改node group信息。
- 修改node group操作都是系统内部操作，除了SET DEFAULT语法之外，其他操作都需要在维护模式下（调用set xc_maintenance_mode=on;）。
- ALTER NODE GROUP语法仅仅应该在数据库内部使用，使用者不应该手动调用这些SQL语句，否则会导致数据库系统数据不一致。

语法格式

```
ALTER NODE GROUP groupname
{
SET DEFAULT
   | RENAME TO new_group_name
   | SET VCGROUP RENAME TO new_group_name
   | SET NOT VCGROUP
   | SET TABLE GROUP new_group_name
   | COPY BUCKETS FROM src_group_name
   | ADD NODE ( nodename [, ... ] )
   | DELETE NODE ( nodename [, ... ] )
   | RESIZE TO dest_group_name
   | SET VCGROUP WITH GROUP new_group_name
}
```

参数说明

- **groupname**

  需要修改的node group名称。

  取值范围：字符串，要符合**标识符命名规范**。

- **SET DEFAULT**

  将系统中除了groupname指定的node group之外的其他node group对象的in_redistribution字段设置为'y'。考虑到兼容以前版本，该语法仍然保留，且不需要设置维护模式。

- **RENAME TO new_group_name**

  将groupname指定的node group的名称修改为new_group_name。

- **SET TABLE GROUP new_group_name**

  将所有CN节点的pgxc_class表中pgroup字段是group_name的记录修改为new_group_name。

- **COPY BUCKETS FROM src_group_name**

  从src_group_name表示的NodeGroup中，将group_members字段和group_buckets字段的内容拷贝到groupname所表示的NodeGroup中。

- **ADD NODE (nodename [, ...])**

  从groupname指定的NodeGroup中增加指定的节点，这些新增节点在PGXC_NODE系统表中存在。该语句仅仅修改系统表，不会进行实际的节点添加和数据重分布，用户不应该直接调用该SQL语句。

- **DELETE NODE (nodename [, ...])**

  从groupname指定的NodeGroup中，将指定的节点移除，这些被移除的节点仍然存在于PGXC_NODE系统表中。该语句仅仅修改系统表，不会进行实际的节点移除和数据重分布，用户不应该直接调用该SQL语句。

- **RESIZE TO dest_group_name**

  设置集群resize操作标志，将groupname所表示的NodeGroup设置为重分布的源NodeGroup，并取消is_installation标志；同时将desst_group_name设置为重分布的目的NodeGroup，并设置is_installation标志。

相关链接

CREATE NODE GROUP，DROP NODE GROUP

7.13.16 ALTER RESOURCE LABEL

功能描述

修改资源标签。

注意事项

只有poladmin、sysadmin或初始用户才能执行此操作。

语法格式

```
ALTER RESOURCE LABEL label_name (ADD|REMOVE)
  label_item_list[, ...]*;
```

- label_item_list：
  ```
  resource_type(resource_path[, ...]*)
  ```
- resource_type：
  ```
  TABLE | COLUMN | SCHEMA | VIEW | FUNCTION
  ```

参数说明

- **label_name**

  资源标签名称。

  取值范围：字符串，要符合标识符命名规范。

- **resource_type**
  指的是要标记的数据库资源类型。
- **resource_path**
  指的是描述具体的数据库资源的路径。

示例

```
--创建基本表table_for_label。
openGauss=# CREATE TABLE table_for_label(col1 int, col2 text);

--创建资源标签table_label。
openGauss=# CREATE RESOURCE LABEL table_label ADD COLUMN(table_for_label.col1);

--将col2添加至资源标签table_label中。
openGauss=# ALTER RESOURCE LABEL table_label ADD COLUMN(table_for_label.col2)

--将资源标签table_label中的一项移除。
openGauss=# ALTER RESOURCE LABEL table_label REMOVE COLUMN(table_for_label.col1);

--删除资源标签table_label。
openGauss=# DROP RESOURCE LABEL table_label;

--删除基本表table_for_label。
openGauss=# DROP TABLE table_for_label;
```

相关链接

CREATE RESOURCE LABEL，**DROP RESOURCE LABEL**。

7.13.17 ALTER ROLE

功能描述

修改角色属性。

注意事项

无。

语法格式

- 修改角色的权限。
  ```
  ALTER ROLE role_name [ [ WITH ] option [ ... ] ];
  ```
  其中权限项子句option为：
  ```
  {CREATEDB | NOCREATEDB}
    | {CREATEROLE | NOCREATEROLE}
    | {INHERIT | NOINHERIT}
    | {AUDITADMIN | NOAUDITADMIN}
    | {SYSADMIN | NOSYSADMIN}
    | {MONADMIN | NOMONADMIN}
    | {OPRADMIN | NOOPRADMIN}
    | {POLADMIN | NOPOLADMIN}
    | {USEFT | NOUSEFT}
    | {LOGIN | NOLOGIN}
    | {REPLICATION | NOREPLICATION}
    | {VCADMIN | NOVCADMIN}
    | {PERSISTENCE | NOPERSISTENCE}
    | CONNECTION LIMIT connlimit
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'[EXPIRED]
    | [ ENCRYPTED | UNENCRYPTED ] IDENTIFIED BY 'password' [ REPLACE 'old_password' | EXPIRED ]
  ```



```
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD { 'password' | DISABLE | EXPIRED }
    | [ ENCRYPTED | UNENCRYPTED ] IDENTIFIED BY { 'password' [ REPLACE 'old_password' ] |
DISABLE }
    | VALID BEGIN 'timestamp'
    | VALID UNTIL 'timestamp'
    | USER GROUP 'groupuser'
    | PERM SPACE 'spacelimit'
    | TEMP SPACE 'tmpspacelimit'
    | SPILL SPACE 'spillspacelimit'
    | NODE GROUP logic_cluster_name
    | PGUSER
```

- 修改角色的名称。

```
ALTER ROLE role_name
    RENAME TO new_name;
```

- 锁定或解锁。

```
ALTER ROLE role_name
    ACCOUNT { LOCK | UNLOCK };
```

- 设置角色的配置参数。

```
ALTER ROLE role_name [ IN DATABASE database_name ]
    SET configuration_parameter {{ TO | = } { value | DEFAULT } | FROM CURRENT};
```

- 重置角色的配置参数。

```
ALTER ROLE role_name
    [ IN DATABASE database_name ] RESET {configuration_parameter|ALL};
```

参数说明

- **role_name**

  现有角色名。

  取值范围：已存在的角色名，如果角色名中包含大写字母则需要使用双引号括起来。

- **IN DATABASE database_name**

  表示修改角色在指定数据库上的参数。

- **SET configuration_parameter {{ TO | = } { value | DEFAULT } | FROM CURRENT}**

  设置角色的参数。ALTER ROLE中修改的会话参数只针对指定的角色，且在下一次该角色启动的会话中有效。

  须知

  当前版本不支持设置用户级别参数。

  取值范围：

  configuration_parameter和value的取值请参见**SET**。

  DEFAULT：表示清除configuration_parameter参数的值，configuration_parameter参数的值将继承本角色新产生的SESSION的默认值。

  FROM CURRENT：取当前会话中的值设置为configuration_parameter参数的值。

- **RESET {configuration_parameter|ALL}**

  清除configuration_parameter参数的值。与SET configuration_parameter TO DEFAULT的效果相同。

须知

当前版本不支持重置用户级别参数。

取值范围：ALL表示清除所有参数的值。

- **ACCOUNT LOCK | ACCOUNT UNLOCK**
  - ACCOUNT LOCK：锁定账户，禁止登录数据库。
  - ACCOUNT UNLOCK：解锁账户，允许登录数据库。
- **PGUSER**

  当前版本不允许修改角色的PGUSER属性。
- **PASSWORD/IDENTIFIED BY 'password'**

  重置或修改用户密码。除了初始用户外其他管理员或普通用户修改自己的密码需要输入正确的旧密码。只有初始用户、系统管理员（sysadmin）或拥有创建用户（CREATEROLE）权限的用户才可以重置普通用户密码，无需输入旧密码。初始用户可以重置系统管理员的密码，系统管理员不允许重置其他系统管理员的密码。
- **EXPIRED**

  设置密码失效。只有初始用户、系统管理员（sysadmin）或拥有创建用户（CREATEROLE）权限的用户才可以设置用户密码失效，其中系统管理员也可以设置自己或其他系统管理员密码失效。不允许设置初始用户密码失效。

  密码失效的用户可以登录数据库但不能执行查询操作，只有修改密码或由管理员重置密码后才可以恢复正常查询操作。

其他参数请参见CREATE ROLE的**参数说明**。

示例

请参见CREATE ROLE的**示例**。

相关链接

CREATE ROLE，**DROP ROLE**，**SET ROLE**

7.13.18 ALTER ROW LEVEL SECURITY POLICY

功能描述

对已存在的行访问控制策略（包括行访问控制策略的名称，行访问控制指定的用户，行访问控制的策略表达式）进行修改。

注意事项

表的所有者或管理员用户才能进行此操作。

语法格式

```
ALTER [ ROW LEVEL SECURITY ] POLICY [ IF EXISTS ] policy_name ON table_name RENAME TO new_policy_name;

ALTER [ ROW LEVEL SECURITY ] POLICY policy_name ON table_name
```



```
    [ TO { role_name | PUBLIC } [, ...] ]
    [ USING ( using_expression ) ];
```

参数说明

- policy_name
  行访问控制策略名称。
- table_name
  行访问控制策略的表名。
- new_policy_name
  新的行访问控制策略名称。
- role_name
  行访问控制策略应用的数据库用户，可以指定多个用户，PUBLIC表示应用到所有用户。
- using_expression
  行访问控制的表达式，返回值为boolean类型。

示例

```
--创建数据表all_data。
openGauss=# CREATE TABLE all_data(id int, role varchar(100), data varchar(100));

--创建行访问控制策略，当前用户只能查看用户自身的数据。
openGauss=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role =
CURRENT_USER);
openGauss=# \d+ all_data
                        Table "public.all_data"
 Column |          Type          | Modifiers | Storage  | Stats target | Description
--------+------------------------+-----------+----------+--------------+-------------
 id     | integer                |           | plain    |              |
 role   | character varying(100) |           | extended |              |
 data   | character varying(100) |           | extended |              |
Row Level Security Policies:
    POLICY "all_data_rls"
      USING (((role)::name = "current_user"()))
Has OIDs: no
Distribute By: HASH(id)
Location Nodes: ALL DATANODES
Options: orientation=row

--修改行访问控制all_data_rls的名称。
openGauss=# ALTER ROW LEVEL SECURITY POLICY all_data_rls ON all_data RENAME TO all_data_new_rls;

--创建用户alice, bob。
openGauss=# CREATE ROLE alice WITH PASSWORD "********";
openGauss=# CREATE ROLE bob WITH PASSWORD "********";

--修改行访问控制策略影响的用户。
openGauss=# ALTER ROW LEVEL SECURITY POLICY all_data_new_rls ON all_data TO alice, bob;
openGauss=# \d+ all_data
                        Table "public.all_data"
 Column |          Type          | Modifiers | Storage  | Stats target | Description
--------+------------------------+-----------+----------+--------------+-------------
 id     | integer                |           | plain    |              |
 role   | character varying(100) |           | extended |              |
 data   | character varying(100) |           | extended |              |
Row Level Security Policies:
    POLICY "all_data_new_rls"
      TO alice,bob
      USING (((role)::name = "current_user"()))
Has OIDs: no
```



```
Distribute By: HASH(id)
Location Nodes: ALL DATANODES
Options: orientation=row, enable_rowsecurity=true

--修改行访问控制策略表达式。
openGauss=# ALTER ROW LEVEL SECURITY POLICY all_data_new_rls ON all_data USING (id > 100 AND role
= current_user);
openGauss=# \d+ all_data
                          Table "public.all_data"
 Column |          Type          | Modifiers | Storage  | Stats target | Description
--------+------------------------+-----------+----------+--------------+-------------
 id     | integer                |           | plain    |              |
 role   | character varying(100) |           | extended |              |
 data   | character varying(100) |           | extended |              |
Row Level Security Policies:
    POLICY "all_data_new_rls"
      TO alice,bob
      USING (((id > 100) AND ((role)::name = "current_user"())))
Has OIDs: no
Distribute By: HASH(id)
Location Nodes: ALL DATANODES
Options: orientation=row, enable_rowsecurity=true

--删除用户alice, bob。
openGauss=# DROP ROLE alice, bob;

--删除访问控制策略。
openGauss=# DROP ROW LEVEL SECURITY POLICY all_data_new_rls ON all_data;

--删除数据表all_data。
openGauss=# DROP TABLE all_data;
```

相关链接

CREATE ROW LEVEL SECURITY POLICY，**DROP ROW LEVEL SECURITY POLICY**

7.13.19 ALTER SCHEMA

功能描述

修改模式属性。

注意事项

- 只有模式的所有者或者被授予了模式ALTER权限的用户有权限执行ALTER SCHEMA命令，系统管理员默认拥有此权限。但要修改模式的所有者，当前用户必须是该模式的所有者或者系统管理员，且该用户是新所有者角色的成员。
- 对于系统模式pg_catalog，只允许初始用户修改模式的所有者。修改系统自带模式的名称可能会导致部分功能不可用甚至影响数据库正常运行，默认情况下不允许修改系统自带模式的名称，考虑到前向兼容性，仅允许当系统在启动或升级过程中或参数allow_system_table_mods为on时修改。

语法格式

- 修改模式的名称。

```
ALTER SCHEMA schema_name
    RENAME TO new_name;
```

- 修改模式的所有者。

```
ALTER SCHEMA schema_name
    OWNER TO new_owner;
```


参数说明

- **schema_name**

  现有模式的名称。

  取值范围：已存在的模式名。

- **RENAME TO new_name**

  修改模式的名称。非系统管理员要改变模式的名称，则该用户必须在此数据库上有CREATE权限。

  new_name：模式的新名称。

  取值范围：字符串，要符合**标识符命名规范**。

- **OWNER TO new_owner**

  修改模式的所有者。非系统管理员要改变模式的所有者，该用户还必须是新的所有角色的直接或间接成员， 并且该成员必须在此数据库上有CREATE权限。

  new_owner：模式的新所有者。

  取值范围：已存在的用户名/角色名。

示例

```
--创建模式ds。
openGauss=# CREATE SCHEMA ds;

--将当前模式ds更名为ds_new。
openGauss=# ALTER SCHEMA ds RENAME TO ds_new;

--创建用户jack。
openGauss=# CREATE USER jack PASSWORD '********';

--将DS_NEW的所有者修改为jack。
openGauss=# ALTER SCHEMA ds_new OWNER TO jack;


--删除用户jack和模式ds_new。
openGauss=# DROP SCHEMA ds_new;
openGauss=# DROP USER jack;
```

相关链接

CREATE SCHEMA，**DROP SCHEMA**

7.13.20 ALTER SEQUENCE

功能描述

修改一个现有的序列的参数。

注意事项

- 只有序列的所有者或者被授予了序列ALTER权限的用户才能执行ALTER SEQUENCE命令，系统管理员默认拥有该权限。但要修改序列的所有者，当前用户必须是该序列的所有者或者系统管理员，且该用户是新所有者角色的成员。
- 当前版本仅支持修改拥有者、归属列和最大值。若要修改其他参数，可以删除重建，并用Setval函数恢复当前值。
- ALTER SEQUENCE MAXVALUE不支持在事务、函数和存储过程中使用。

- 修改序列的最大值后，会清空该序列在所有会话的cache。
- ALTER SEQUENCE会阻塞nextval、setval、currval和lastval的调用。

语法格式

- 修改序列归属列

```
ALTER SEQUENCE [ IF EXISTS ] name
    [MAXVALUE maxvalue | NO MAXVALUE | NOMAXVALUE]
    [ OWNED BY { table_name.column_name | NONE } ] ;
```

- 修改序列的拥有者

```
ALTER SEQUENCE [ IF EXISTS ] name OWNER TO new_owner;
```

参数说明

- **name**

  将要修改的序列名称。

- **IF EXISTS**

  当序列不存在时使用该选项不会出现错误消息，仅有一个通知。

- **OWNED BY**

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

  如果序列已经和表有关联后，使用这个选项后新的关联关系会覆盖旧的关联。

  关联的表和序列的所有者必须是同一个用户，并且在同一个模式中。

  使用OWNED BY NONE将删除任何已经存在的关联。

- **new_owner**

  序列新所有者的用户名。用户要修改序列的所有者，必须是新角色的直接或者间接成员，并且那个角色必须有序列所在模式上的CREATE权限。

示例

```
--创建一个名为serial的递增序列，从101开始。
openGauss=# CREATE SEQUENCE serial START 101;

--创建一个表，定义默认值。
openGauss=# CREATE TABLE T1(C1 bigint default nextval('serial'));

--将序列serial的归属列变为T1.C1。
openGauss=# ALTER SEQUENCE serial OWNED BY T1.C1;

--删除序列
openGauss=# DROP SEQUENCE serial cascade;
openGauss=# DROP TABLE T1;
```

相关链接

CREATE SEQUENCE，**DROP SEQUENCE**

7.13.21 ALTER SERVER

功能描述

增加、修改和删除一个现有server的参数。现有server可以从pg_foreign_server系统表查询。

注意事项

只有server的所有者或者被授予了server的ALTER权限的用户才可以执行ALTER SERVER命令，系统管理员默认拥有该权限。但要修改server的所有者，当前用户必须是该server的所有者或者系统管理员，且该用户是新所有者角色的成员。

OPTIONS中的敏感字段（如password、secret_access_key）在使用多层引号时，语义和不带引号的场景是不同的，因此不会被识别为敏感字段进行脱敏。

语法格式

- 修改外部服务的参数。

```
ALTER SERVER server_name [ VERSION 'new_version' ]
    [ OPTIONS ( {[ ADD | SET | DROP ] option ['value']} [, ... ] ) ];
```

在OPTIONS选项里，ADD、SET和DROP指定要执行的操作，未指定时默认为ADD操作。option和value为对应操作的参数。

- 修改外部服务的所有者。

```
ALTER SERVER server_name
    OWNER TO new_owner;
```

- 修改外部服务的名称。

```
ALTER SERVER server_name
    RENAME TO new_name;
```

参数说明

修改server的参数如下所示：

- **server_name**

  所修改的server的名称。

- **new_version**

  修改后server的新版本名称。

- **OPTIONS**

  更改该服务器的选项。ADD、SET和 DROP指定要执行的动作。如果没有显式地指定操作， 将会假定为ADD。选项名称必须唯一，名称和值也会使用该服务器的外部数据封装器库进行验证。

  - encrypt

    是否对数据进行加密，该参数仅支持在type为OBS时设置。默认值为off。

    取值范围：

    - on表示对数据进行加密。
    - off表示不对数据进行加密。

  - access_key

    OBS访问协议对应的AK值（OBS云服务界面由用户获取），创建外表时AK值会加密保存到数据库的元数据表中。该参数仅支持type为OBS时设置。

  - secret_access_key

    OBS访问协议对应的SK值（OBS云服务界面由用户获取），创建外表时SK值会加密保存到数据库的元数据表中。该参数仅支持type为OBS时设置。

- **new_owner**

修改后server的新拥有者。更改所有者，你必须是外部服务器的所有者并且也是新的所有者角色的直接或者间接成员，并且你必须对外部服务器的外部数据封装器有USAGE权限。

- **new_name**

  修改后server的新名称。

示例

```
--创建my_server。
openGauss=# CREATE SERVER my_server FOREIGN DATA WRAPPER log_fdw;

--修改外部服务的名称。
openGauss=# ALTER SERVER my_server RENAME TO my_server_1;

--删除my_server_1。
openGauss=# DROP SERVER my_server_1;
```

相关链接

CREATE SERVER，**DROP SERVER**

7.13.22 ALTER SESSION

功能描述

ALTER SESSION命令用于定义或修改那些对当前会话有影响的条件或参数。修改后的会话参数会一直保持，直到断开当前会话。

注意事项

- 如果执行SET TRANSACTION之前没有执行START TRANSACTION，则事务立即结束，命令无法显示效果。
- 可以用START TRANSACTION里面声明所需要的transaction_mode(s)的方法来避免使用SET TRANSACTION。

语法格式

- 设置会话的事务参数。

  ```
  ALTER SESSION SET [ SESSION CHARACTERISTICS AS ] TRANSACTION
     { ISOLATION LEVEL { READ COMMITTED | READ UNCOMMITTED } | { READ ONLY  | READ
  WRITE } } [, ...] ;
  ```

- 设置会话的其他运行时参数。

  ```
  ALTER SESSION SET
    {{config_parameter { { TO  | =  }  { value | DEFAULT }
      | FROM CURRENT }} | CURRENT_SCHEMA [ TO | = ] { schema | DEFAULT }
      | TIME ZONE time_zone
      | SCHEMA 'schema'
      | NAMES encoding_name
      | ROLE role_name PASSWORD 'password'
      | SESSION AUTHORIZATION { role_name PASSWORD 'password' | DEFAULT }
      | XML OPTION { DOCUMENT | CONTENT }
    } ;
  ```

参数说明

- **SESSION**

  声明的参数只对当前会话起作用。如果SESSION和LOCAL都没出现，则SESSION为缺省值。

如果在事务中执行了此命令，命令的产生影响将在事务回滚之后消失。如果该事务已提交，影响将持续到会话的结束，除非被另外一个SET命令重置参数。

- **TIME ZONE timezone**

  用于指定当前会话的本地时区。

  取值范围：有效的本地时区。该选项对应的运行时参数名称为TimeZone，DEFAULT缺省值为PRC。

- **CURRENT_SCHEMA**

  schema

  CURRENT_SCHEMA用于指定当前的模式。

  取值范围：已存在模式名称。如果模式名不存在，会导致CURRENT_SCHEMA值为空。

- **SCHEMA schema**

  同CURRENT_SCHEMA。此处的schema是个字符串。

  例如：set schema 'public';

- **NAMES encoding_name**

  用于设置客户端的字符编码。等价于set client_encoding to encoding_name。

  取值范围：有效的字符编码。该选项对应的运行时参数名称为client_encoding，默认编码为UTF8。

- **XML OPTION option**

  用于设置XML的解析方式。

  取值范围：CONTENT（缺省）、DOCUMENT

- **config_parameter**

  可设置的运行时参数的名称。可用的运行时参数可以使用SHOW ALL命令查看。

示例

```
-- 创建模式ds。
openGauss=# CREATE SCHEMA ds;

--设置模式搜索路径。
openGauss=# SET SEARCH_PATH TO ds, public;

--设置日期时间风格为传统的POSTGRES风格（日在月前）。
openGauss=# SET DATESTYLE TO postgres, dmy;

--设置当前会话的字符编码为UTF8。
openGauss=# ALTER SESSION SET NAMES 'UTF8';

--设置时区为加州伯克利。
openGauss=# SET TIME ZONE 'PST8PDT';

--设置时区为意大利。
openGauss=# SET TIME ZONE 'Europe/Rome';

--设置当前模式。
openGauss=# ALTER SESSION SET CURRENT_SCHEMA TO tpcds;

--设置XML OPTION为DOCUMENT。
openGauss=# ALTER SESSION SET XML OPTION DOCUMENT;

--创建角色joe，并设置会话的角色为joe。
openGauss=# CREATE ROLE joe WITH PASSWORD '********';
openGauss=# ALTER SESSION SET SESSION AUTHORIZATION joe PASSWORD '********';
```



```
--切换到默认用户。
openGauss=> ALTER SESSION SET SESSION AUTHORIZATION default;

--删除ds模式。
openGauss=# DROP SCHEMA ds;

--删除joe。
openGauss=# DROP ROLE joe;

--开启事务,设置事务级别
openGauss=# START TRANSACTION;
openGauss=# ALTER SESSION SET TRANSACTION READ ONLY;
openGauss=# END;
```

相关链接

SET

7.13.23 ALTER SYNONYM

功能描述

修改SYNONYM对象的属性。

注意事项

- 目前仅支持修改SYNONYM对象的属主。
- 只有系统管理员有权限修改SYNONYM对象的属主信息。
- 新属主必须具有SYNONYM对象所在模式的CREATE权限。

语法格式

```
ALTER SYNONYM synonym_name
    OWNER TO new_owner;
```

参数描述

- **synonym_name**
  待修改的同义词名字，可以带模式名。
  取值范围：字符串，需要符合**标识符命名规范**。
- **new_owner**
  同义词对象的新所有者。
  取值范围：字符串，有效的用户名。

示例

```
--创建同义词t1。
openGauss=# CREATE OR REPLACE SYNONYM t1 FOR ot.t1;

--创建新用户u1。
openGauss=# CREATE USER u1 PASSWORD '********';

--修改同义词t1的owner为u1。
openGauss=# ALTER SYNONYM t1 OWNER TO u1;

--删除同义词t1。
openGauss=# DROP SYNONYM t1;
```



```
--删除用户u1。
openGauss=# DROP USER u1;
```

相关链接

CREATE SYNONYM，DROP SYNONYM

7.13.24 ALTER SYSTEM KILL SESSION

功能描述

ALTER SYSTEM KILL SESSION命令用于结束一个会话。

注意事项

无。

语法格式

```
ALTER SYSTEM KILL SESSION 'session_sid, serial' [ IMMEDIATE ];
```

参数说明

- **session_sid, serial**

  会话的SID和SERIAL（格式请参考示例）。

  取值范围：通过查看系统表dv_sessions可查看所有会话的SID和SERIAL。

- **IMMEDIATE**

  表明会话将在命令执行后立即结束。

示例

```
--查询会话信息。
openGauss=# SELECT sid,serial#,username FROM dv_sessions;
       sid       | serial# | username
-----------------+---------+----------
 140131075880720 |       0 | omm
 140131025549072 |       0 | omm
 140131073779472 |       0 | omm
 140131071678224 |       0 | omm
 140131125774096 |       0 |
 140131127875344 |       0 |
 140131113629456 |       0 |
 140131094742800 |       0 |
(8 rows)

--结束SID为140131075880720的会话。
openGauss=# ALTER SYSTEM KILL SESSION '140131075880720,0' IMMEDIATE;
```

7.13.25 ALTER TABLE

功能描述

修改表，包括修改表的定义、重命名表、重命名表中指定的列、重命名表的约束、设置表的所属模式、添加/更新多个列、打开/关闭行访问控制开关。

注意事项

- 表的所有者、被授予了表ALTER权限的用户或被授予ALTER ANY TABLE权限的用户有权限执行ALTER TABLE命令，系统管理员默认拥有此权限。但要修改表的所有者或者修改表的模式，当前用户必须是该表的所有者或者系统管理员，且该用户是新所有者角色的成员。
- 不能修改分区表的TABLESPACE，但可以修改分区的TABLESPACE。
- 不支持修改存储参数ORIENTATION。
- SET SCHEMA操作不支持修改为系统内部模式，当前仅支持用户模式之间的修改。
- 不允许对表的分布列（distribute column）进行修改。
- 不支持增加自增列，或者增加DEFAULT值中包含nextval()表达式的列。
- 不支持对外表、临时表开启行访问控制开关。
- 通过约束名删除PRIMARY KEY约束时，不会删除NOT NULL约束，如果有需要，请手动删除NOT NULL约束。
- 使用JDBC时，支持通过PrepareStatement对DEFAULT值进行参数化设置。
- 如果用ADD COLUMN增加一个字段，那么所有表中现有行都初始化为该字段的缺省值 (如果没有声明DEFAULT子句，那么就是 NULL)。

  新增列没有声明DEFAULT值时，默认值为NULL，不会触发全表更新。

  新增列如果有DEFAULT值，必须符合以下所有要求，否则会带来全表更新开销，影响在线业务：

  1. 数据类型为以下类型中的一种：BOOL, BYTEA, SMALLINT, BIGINT, SMALLINT, INTEGER, NUMERIC, FLOAT, DOUBLE PRECISION, CHAR, VARCHAR, TEXT, TIMESTAMPTZ, TIMESTAMP, DATE, TIME, TIMETZ, INTERVAL；

  2. 新增列的DEFAULT值长度不超过128个字节；

  3. 新增列DEFAULT值不包含易变（volatile）函数；

  4. 新增列设置有DEFAULT值，且DEFAULT值不为NULL。

  如果不确定是否满足条件3，可以查询PG_RPOC系统表中函数的provolatile属性是否为'v'。

语法格式

- 修改表的定义。

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }
    action [, ... ];
```

其中具体表操作action可以是以下子句之一：

```
column_clause
    | ADD table_constraint [ NOT VALID ]
    | ADD table_constraint_using_index
    | VALIDATE CONSTRAINT constraint_name
    | DROP CONSTRAINT [ IF EXISTS ]  constraint_name [ RESTRICT | CASCADE ]
    | CLUSTER ON index_name
    | SET WITHOUT CLUSTER
    | SET ( {storage_parameter = value} [, ... ] )
    | RESET ( storage_parameter [, ... ] )
    | OWNER TO new_owner
    | SET TABLESPACE new_tablespace
    | TO { GROUP groupname | NODE ( nodename [, ... ] ) }
    | ADD NODE ( nodename [, ... ] )
    | DELETE NODE ( nodename [, ... ] )
    | UPDATE SLICE LIKE table_name
    | DISABLE TRIGGER [ trigger_name | ALL | USER ]
```



```
| ENABLE TRIGGER [ trigger_name | ALL | USER ]
| ENABLE REPLICA TRIGGER trigger_name
| ENABLE ALWAYS TRIGGER trigger_name
| DISABLE/ENABLE [ REPLICA | ALWAYS ] RULE
| DISABLE ROW LEVEL SECURITY
| ENABLE ROW LEVEL SECURITY
| FORCE ROW LEVEL SECURITY
| NO FORCE ROW LEVEL SECURITY

| INHERIT parent_table
| NO INHERIT parent_table
| DISTRIBUTE BY { REPLICATION | { [ HASH ] ( column_name ) } }
| OF type_name
| NOT OF
| REPLICA IDENTITY { DEFAULT | USING INDEX index_name | FULL | NOTHING }
```


说明

- **ADD table_constraint [NOT VALID]**

  给表增加一个新的约束。

- **ADD table_constraint_using_index**

  根据已有唯一索引为表增加主键约束或唯一约束。

- **VALIDATE CONSTRAINT constraint_name**

  验证一个使用NOT VALID选项创建的检查类约束，通过扫描全表来保证所有记录都符合约束条件。如果约束已标记为有效时，什么操作也不会发生。

- **DROP CONSTRAINT [IF EXISTS] constraint_name [RESTRICT | CASCADE]**

  删除一个表上的约束。

- **CLUSTER ON index_name**

  为将来的CLUSTER（聚簇）操作选择默认索引。实际上并没有重新盘簇化处理该表。

- **SET WITHOUT CLUSTER**

  从表中删除最新使用的CLUSTER索引。这样会影响将来那些没有声明索引的CLUSTER（聚簇）操作。

- **SET ({storage_parameter = value} [, ...])**

  修改表的一个或多个存储参数。

- **RESET (storage_parameter [, ...])**

  重置表的一个或多个存储参数。与SET一样，根据参数的不同可能需要重写表才能获得想要的效果。

- **OWNER TO new_owner**

  将表、序列、视图的属主改变成指定的用户。

- **SET TABLESPACE new_tablespace**

  这种形式将表空间修改为指定的表空间并将相关的数据文件移动到新的表空间。但是表上的所有索引都不会被移动，索引可以通过ALTER INDEX语法的SET TABLESPACE选项来修改索引的表空间。

- **TO { GROUP groupname | NODE (nodename [, ...]) }**

  此语法仅在扩展模式（GUC参数support_extended_features为on时）下可用。该模式谨慎打开，主要供内部扩容工具使用，一般用户不应使用该模式。该命令只会修改表分布节点的逻辑映射关系，并未真正在DN节点上迁移表的元数据和数据。

- **ADD NODE (nodename [, ...])**

  此语法主要供内部扩容工具使用，一般用户不建议使用。

- **DELETE NODE (nodename [, ...])**

  此语法主要供内部缩容工具使用，一般用户不建议使用。

- **UPDATE SLICE LIKE table_name**

  此语法主要供内部扩缩容工具使用，一般用户不可以使用。

- **DISABLE TRIGGER [trigger_name | ALL | USER]**

  禁用trigger_name所表示的单个触发器，或禁用所有触发器，或仅禁用用户触发器（此选项不包括内部生成的约束触发器，例如，可延迟唯一性和排除约束的约束触发器）。

  应谨慎使用此功能，因为如果不执行触发器，则无法保证原先期望的约束的完整性。

- **| ENABLE TRIGGER [trigger_name | ALL | USER]**

  启用trigger_name所表示的单个触发器，或启用所有触发器，或仅启用用户触发器。

- **| ENABLE REPLICA TRIGGER trigger_name**

  触发器触发机制受配置变量session_replication_role的影响，当复制角色为“origin”（默认值）或“local”时，将触发简单启用的触发器。

  配置为ENABLE REPLICA的触发器仅在会话处于“replica”模式时触发。

- **| ENABLE ALWAYS TRIGGER trigger_name**

  无论当前复制模式如何，配置为ENABLE ALWAYS的触发器都将触发。

- **| { DISABLE | ENABLE } [REPLICA | ALWAYS] RULE**

  配置属于表的重写规则,已禁用的规则对系统来说仍然是可见的，只是在查询重写期间不被应用。语义为关闭/启动规则。由于关系到视图的实现，ON SELECT规则不可禁用。 配置为ENABLE REPLICA的规则将会仅在会话为"replica" 模式时启动，而配置为ENABLE ALWAYS的触发器将总是会启动，不考虑当前复制模式。规则触发机制也受配置变量session_replication_role的影响，类似于上述触发器。

- **| { DISABLE | ENABLE } ROW LEVEL SECURITY**

  开启或关闭表的行访问控制开关。

  当开启行访问控制开关时，如果未在该数据表定义相关行访问控制策略，数据表的行级访问将不受影响；如果关闭表的行访问控制开关，即使定义了行访问控制策略，数据表的行访问也不受影响。详细信息参见**CREATE ROW LEVEL SECURITY POLICY**章节。

- **| { NO FORCE | FORCE } ROW LEVEL SECURITY**

  强制开启或关闭表的行访问控制开关。

  默认情况，表所有者不受行访问控制特性影响，但当强制开启表的行访问控制开关时，表的所有者（不包含系统管理员用户）会受影响。系统管理员可以绕过所有的行访问控制策略，不受影响。

- **INHERIT parent_table**

  将目标资料表加到指定的父资料表中成为新的子资料表。之后，针对父资料表的查询将会包含目标资料表的资料。要作为子资料表加入前，目标资料表必须已经包含父资料表的所有栏位。这些栏位必须具有可匹配的资料类别，并且如果在父资料表中具有NOT NULL的限制条件，那么必须在子资料表中也具有NOT NULL的限制条件。对于父资料表的所有CHECK限制条件，必须还有相对应的子资料表限制条件，除非父资料表中标记为不可继承。

- **NO INHERIT parent_table**

  从指定的父资料表的子资料表中产出目标资料表。针对父资料表的查询将不再包含从目标资料表中所产生的记录。

- **DISTRIBUTE BY { REPLICATION | { [HASH] (column_name) } }**

  指定表如何在节点之间分布或者复制。

- **OF type_name**

  将表连接至一种复合类型，与CREATE TABLE OF选项创建表一样。表的字段的名称和类型必须精确匹配复合类型中的定义，不过oid系统字段允许不一样。表不能是从任何其他表继承的。这些限制确保CREATE TABLE OF选项允许一个相同的表定义。

- **NOT OF**

  将一个与某类型进行关联的表进行关联的解除。

- **REPLICA IDENTITY { DEFAULT | USING INDEX index_name | FULL | NOTHING }**

  在逻辑复制场景下，指定该表的UPDATE和DELETE操作中旧元组的记录级别。

  - DEFAULT记录主键的列的旧值，没有主键则不记录。
  - USING INDEX记录命名索引覆盖的列的旧值，这些值必须是唯一的，不局部的，不可延迟的，并且仅包括标记为NOT NULL的列。
  - FULL记录该行中所有列的旧值。
  - NOTHING不记录有关旧行的信息。

  在逻辑复制场景，解析该表的UPDATE和DELETE操作语句时，以此方法记录的信息组成解析出的旧元组。对于有主键表该选项可设置为DEFAULT或FULL。对于无主键表该选项需设置为FULL，否则解码时旧元组将解析为空。一般场景不建议设置为NOTHING，旧元组会始终解析为空。

– 其中列相关的操作column_clause可以是以下子句之一：

```
ADD [ COLUMN ] column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]
| MODIFY column_name data_type
| MODIFY column_name [ CONSTRAINT constraint_name ] NOT NULL [ ENABLE ]
| MODIFY column_name [ CONSTRAINT constraint_name ] NULL
```



```
| DROP [ COLUMN ] [ IF EXISTS ] column_name [ RESTRICT | CASCADE ]
| ALTER [ COLUMN ] column_name [ SET DATA ] TYPE data_type [ COLLATE collation ] [ USING
expression ]
| ALTER [ COLUMN ] column_name { SET DEFAULT expression | DROP DEFAULT }
| ALTER [ COLUMN ] column_name { SET | DROP } NOT NULL
| ALTER [ COLUMN ] column_name SET STATISTICS [PERCENT] integer
| ADD STATISTICS (( column_1_name, column_2_name [, ...] ))
| DELETE STATISTICS (( column_1_name, column_2_name [, ...] ))
| ALTER [ COLUMN ] column_name SET ( {attribute_option = value} [, ... ] )
| ALTER [ COLUMN ] column_name RESET ( attribute_option [, ... ] )
| ALTER [ COLUMN ] column_name SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }
```


说明

- **ADD [COLUMN] column_name data_type [COLLATE collation] [column_constraint [...]]**

  向表中增加一个新的字段。用ADD COLUMN增加一个字段，所有表中现有行都初始化为该字段的缺省值（如果没有声明DEFAULT子句，值为NULL）。

- **ADD ({ column_name data_type} [, ...])**

  向表中增加多列。

- **MODIFY ({ column_name data_type | column_name [CONSTRAINT constraint_name] NOT NULL [ENABLE] | column_name [CONSTRAINT constraint_name] NULL } [, ...])**

  修改表已存在字段的数据类型。此命令会导致该字段的统计信息清空，建议在修改后重新收集该列的统计信息。

- **DROP [COLUMN] [IF EXISTS] column_name [RESTRICT | CASCADE]**

  从表中删除一个字段，和这个字段相关的索引和表约束也会被自动删除。如果任何表之外的对象依赖于这个字段，必须声明CASCADE ，比如视图等。

  DROP COLUMN命令并不是物理上把字段删除，而只是简单地把它标记为对SQL操作不可见。随后对该表的插入和更新将在该字段存储一个NULL。因此，删除一个字段是很快的，但是它不会立即释放表在磁盘上的空间，因为被删除了的字段占据的空间还没有回收。这些空间将在执行VACUUM时而得到回收。

- **ALTER [COLUMN] column_name [SET DATA] TYPE data_type [COLLATE collation] [USING expression]**

  改变表字段的数据类型。该字段涉及的索引和简单的表约束将被自动地转换为使用新的字段类型，方法是重新分析最初提供的表达式。

  当字段的原始数据类型和修改后的数据类型二进制兼容时，执行该语句不需要对整表进行重写，其他场景下会进行整表重写。原类型和目标类型是否二进制兼容可以在PG_CAST系统表中查看，如果castmethod为'b'则为二进制兼容。例如源表中数据类型是text类型，如果转为int类型则会触发表重写；转为clob类型则不会触发表重写。如果表重写被触发，该表上被删除的空间也将被立刻回收。

  此命令会导致该字段的统计信息清空，建议在修改后重新收集该列的统计信息。

- **ALTER [COLUMN] column_name { SET DEFAULT expression | DROP DEFAULT }**

  为一个字段设置或者删除缺省值。请注意缺省值只应用于随后的INSERT命令，它们不会修改表中已经存在的行。也可以为视图创建缺省，这个时候它们是在视图的ON INSERT规则应用之前插入到INSERT句中的。

- **ALTER [COLUMN] column_name { SET | DROP } NOT NULL**

  修改一个字段是否允许NULL值或者拒绝NULL值。如果表在字段中包含非NULL，则只能使用SET NOT NULL。

- **ALTER [COLUMN] column_name SET STATISTICS [PERCENT] integer**

  为随后的ANALYZE操作设置针对每个字段的统计收集目标。目标的范围可以在0到10000之内设置。设置为-1时表示重新恢复到使用系统缺省的统计目标。

- **ALTER [COLUMN] column_name SET ({attribute_option = value} [, ...])**

  ALTER [COLUMN] column_name RESET (attribute_option [, ...])

  设置/重置属性选项。

  目前，属性选项只定义了n_distinct和n_distinct_inherited。n_distinct影响表本身的统计值，而n_distinct_inherited影响表及其继承子表的统计。目前，只支持SET/RESET n_distinct参数，禁止SET/RESET n_distinct_inherited参数。

- **ALTER [COLUMN] column_name SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }**

  为一个字段设置存储模式。这个设置控制这个字段是内联保存还是保存在一个附属的表里，以及数据是否要压缩。仅支持对行存表的设置。SET STORAGE本身并不改变表上的任何东西，只是设置将来的表操作时，建议使用的策略。

  - 其中列约束column_constraint为：

```
[ CONSTRAINT constraint_name ]
    { NOT NULL |
      NULL |
      CHECK ( expression ) |
      DEFAULT default_expr  |
      UNIQUE index_parameters |
      PRIMARY KEY index_parameters |
      REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL |  MATCH PARTIAL | MATCH
SIMPLE ]
        [ ON DELETE action ] [ ON UPDATE action ] }
    [ DEFERRABLE | NOT DEFERRABLE ][ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

- 其中根据已有唯一索引为表增加主键约束或唯一约束table_constraint_using_index为：

```
[ CONSTRAINT constraint_name ]
    { UNIQUE | PRIMARY KEY } USING INDEX index_name
    [ DEFERRABLE | NOT DEFERRABLE ][ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

- 其中表约束table_constraint为：

```
[ CONSTRAINT constraint_name ]
    { CHECK ( expression ) |
      UNIQUE ( column_name [, ... ] ) index_parameters |
      PRIMARY KEY ( column_name [, ... ] ) index_parameters |
      PARTIAL CLUSTER KEY ( column_name [, ... ] )}
    [ DEFERRABLE | NOT DEFERRABLE ][ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

  其中索引参数index_parameters为：

```
[ WITH ( {storage_parameter = value} [, ... ] ) ]
    [ USING INDEX TABLESPACE tablespace_name ]
```

- 重命名表。对名称的修改不会影响所存储的数据。

```
ALTER TABLE [ IF EXISTS ] table_name
    RENAME TO new_table_name;
```

- 重命名表中指定的列。

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name )}
    RENAME [ COLUMN ] column_name TO new_column_name;
```

- 重命名表的约束。

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }
    RENAME CONSTRAINT constraint_name TO new_constraint_name;
```

- 设置表的所属模式。

```
ALTER TABLE [ IF EXISTS ] table_name
    SET SCHEMA new_schema;
```

  说明

  - 这种形式把表移动到另外一个模式。相关的索引、约束都跟着移动。目前序列不支持改变schema。 若该表拥有序列，需要将序列删除，重建，或者取消拥有关系， 才能将表schema更改成功。
  - 要修改一个表的模式，用户必须在新模式上拥有CREATE权限。要把该表添加为一个父表的新子表，用户必须同时又是父表的所有者。要修改所有者，用户还必须是新的所有角色的直接或间接成员，并且该成员必须在该表的模式上有CREATE权限。这些限制规定了该用户不能做出了重建和删除表之外的事情。不过，系统管理员可以以任何方式修改任意表的所有权限。
  - 除了RENAME和SET SCHEMA之外所有动作都可以捆绑在一个经过多次修改的列表中并行使用。比如，可以在一个命令里增加几个字段或修改几个字段的类型。对于大表，此种操作带来的效率提升更明显，原因在于只需要对该大表做一次处理。
  - 增加一个CHECK或NOT NULL约束将会扫描该表，以保证现有的行符合约束要求。
  - 用一个非空缺省值增加一个字段或者改变一个字段的现有类型会重写整个表。对于大表来说，这个操作可能会花很长时间，并且它还临时需要两倍的磁盘空间。

- 添加多个列。

```
ALTER TABLE [ IF EXISTS ] table_name
    ADD ( { column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]} [, ...] );
```


- 更新多个列。

```
ALTER TABLE [ IF EXISTS ] table_name
    MODIFY ( { column_name data_type | column_name [ CONSTRAINT constraint_name ] NOT NULL
[ ENABLE ] | column_name [ CONSTRAINT constraint_name ] NULL } [, ...] );
```

参数说明

- **IF EXISTS**

  如果不存在相同名称的表，不会抛出一个错误，而会发出一个通知，告知表不存在。

- **table_name [*] | ONLY table_name | ONLY (table_name)**

  table_name是需要修改的表名。

  若声明了ONLY选项，则只有那个表被更改。若未声明ONLY，该表及其所有子表都将会被更改。另外，可以在表名称后面显示的增加*选项来指定包括子表，即表示所有后代表都被扫描，这是默认行为。

- **constraint_name**

  要删除的现有约束的名称。

- **index_name**

  索引名称。

- **storage_parameter**

  表的存储参数的名称。

  在线扩容新增的三个选项：

  - append_mode(枚举类型)

    设置表上扩容方式为在线扩容，离线扩容，非扩容方式。在线扩容时允许对表进行部分的修改操作；离线扩容时，在扩容时不允许对表进行操作。

    正在扩容的表上需要新增数据，要以追加的方式写入，便于记录增量数据。

    - on：标记表为在线扩容模式，在线扩容时，设置后后续数据以追加方式写入。
    - off：关闭扩容模式，设置后表上数据按正常方式写入，并且在pg_class.reloptions中不显示在线扩容相关的选项。
    - read_only：标记表为离线扩容。离线扩容时，不允许对表进行操作。
    - end_catchup：最后一轮追增的写报错模式，写业务报错，读业务正常执行。

  - rel_cn_oid（OID类型）

    记录当前CN节点中表的OID，用于在DN节点上生成delete_delta表

    当append_mode=on时，必须同时指定rel_cn_oid.

    这append_mode ,rel_cn_oid两个选项只在在线扩容工具中使用，不建议用户使用。

  - exec_step（整型）

    记录断点续传的步骤，记录在临时表的relOptions中。

    取值范围：[1,4]

    只支持数据重分布工具使用。

  - create_time（长整型）

记录断点续传时临时表创建时间，记录在临时表的relOptions中。

只支持数据重分布工具使用。

- wait_clean_cbi (字符串类型)

  标记当前全局索引中含有扩容bucket搬迁产生的残留tuple，扩容后会设置(wait_clean_cbi=y), 在vacuum流程清理残留tuple后设置（wait_clean_cbi=n）。

  此选项只在扩容工具中使用，不建议用户使用。

创建索引新增一个选项：

- parallel_workers（int类型）

  表示创建索引时起的bgworker线程数量，例如2就表示将会起2个bgworker线程并发创建索引。

  取值范围：[0,32]，0表示关闭该功能。

  默认值：不设置该参数，表示未开启并行建索引功能。

复制表新增一个选项：

- primarynode（bool类型）

  默认值：off

  当primarynode=on时，将为复制表选择primary node，通常是pgxc_class表nodeoids字段记录的第一个节点。当复制表执行IUD操作时，将先下发到primarynode节点执行，收到结果后再下发到其它DN。

- **new_owner**

  表新拥有者的名称。

- **new_tablespace**

  表所属新的表空间名称。

- **column_name，column_1_name， column_2_name**

  现存的或新字段的名称。

- **data_type**

  新字段的类型，或者现存字段的新类型。

- **collation**

  字段排序规则名称。可选字段COLLATE指定了新字段的排序规则，如果省略，排序规则为新字段的默认类型。排序规则可以使用“select * from pg_collation”命令从pg_collation系统表中查询，默认的排序规则为查询结果中以default开始的行。

- **USING expression**

  USING子句声明如何从旧的字段值里计算新的字段值；如果省略，缺省从旧类型向新类型的赋值转换。如果从旧数据类型到新类型没有隐含或者赋值的转换，则必须提供一个USING子句。

  说明

  ALTER TYPE的USING选项实际上可以声明涉及该行旧值的任何表达式，即它可以引用除了正在被转换的字段之外其他的字段。这样，就可以用ALTER TYPE语法做非常普遍性的转换。因为这个灵活性，USING表达式并没有作用于该字段的缺省值（如果有的话），结果可能不是缺省表达式要求的常量表达式。这就意味着如果从旧类型到新类型没有隐含或者赋值转换的话，即使存在USING子句，ALTER TYPE也可能无法把缺省值转换成新的类型。在这种情况下，应该用DROP DEFAULT先删除缺省，执行ALTER TYPE，然后使用SET DEFAULT增加一个合适的新缺省值。类似的考虑也适用于涉及该字段的索引和约束。

- **NOT NULL | NULL**

  设置列是否允许空值。

- **ENABLE**

  表示启动该约束，缺省时默认启用。

- **integer**

  带符号的整数常值。当使用PERCENT时表示按照表数据的百分比收集统计信息，integer的取值范围为0-100。

- **attribute_option**

  属性选项。

- **PLAIN | EXTERNAL | EXTENDED | MAIN**

  字段存储模式。

  - PLAIN必须用于定长的数值（比如integer）并且是内联的、不压缩的。
  - MAIN用于内联、可压缩的数据。
  - EXTERNAL用于外部保存、不压缩的数据。使用EXTERNAL将令在text和bytea字段上的子字符串操作更快，但付出的代价是增加了存储空间。
  - EXTENDED用于外部的压缩数据，EXTENDED是大多数支持非PLAIN存储的数据的缺省。

- **CHECK (expression)**

  每次将要插入的新行或者将要被更新的行必须使表达式结果为真才能成功，否则会抛出一个异常并且不会修改数据库。

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

  目前，CHECK表达式不能包含子查询也不能引用除当前行字段之外的变量。

- **DEFAULT default_expr**

  给字段指定缺省值。

  缺省表达式的数据类型必须和字段类型匹配。

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

- **UNIQUE index_parameters**

  UNIQUE (column_name [, ...]) index_parameters

  UNIQUE约束表示表里的一个或多个字段的组合必须在全表范围内唯一。

- **PRIMARY KEY index_parameters**

  PRIMARY KEY (column_name [, ...]) index_parameters

  主键约束表明表中的一个或者一些字段只能包含唯一（不重复）的非NULL值。

- **DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE**

  设置该约束是否可推迟。

  - DEFERRABLE：可以推迟到事务结尾使用SET CONSTRAINTS命令检查。
  - NOT DEFERRABLE：在每条命令之后马上检查。
  - INITIALLY IMMEDIATE：在每条语句之后就立即检查它。
  - INITIALLY DEFERRED：只有在事务结尾才检查它。

- **WITH ({storage_parameter = value} [, ...])**
  为表或索引指定一个可选的存储参数。
- **tablespace_name**
  索引所在表空间的名称。
- **new_table_name**
  修改后新的表名称。
- **new_column_name**
  表中指定列修改后新的列名称。
- **new_constraint_name**
  修改后表约束的新名称。
- **new_schema**
  修改后新的模式名称。
- **CASCADE**
  级联删除依赖于被依赖字段或者约束的对象（比如引用该字段的视图）。
- **RESTRICT**
  如果字段或者约束还有任何依赖的对象，则拒绝删除该字段。这是缺省行为。
- **schema_name**
  表所在的模式名称。

修改表示例

- **重命名表**

```
openGauss=# CREATE TABLE aa(c1 int, c2 int);
openGauss=# ALTER TABLE IF EXISTS aa RENAME TO test_alt1;
```

- **修改表所属模式**

```
--创建模式test_schema。
openGauss=# CREATE SCHEMA test_schema;
--把表test_alt1的所属模式修改为test_schema。
openGauss=# ALTER TABLE test_alt1 SET SCHEMA test_schema;
--查询表信息。
openGauss=# SELECT schemaname,tablename FROM pg_tables WHERE tablename = 'test_alt1';
 schemaname  | tablename
-------------+-----------
 test_schema | test_alt1
(1 row)
```

- **修改表的所有者**

```
--创建用户test_user。
openGauss=# CREATE USER test_user PASSWORD 'XXXXXXXXXX';
-- 修改test_alt1表的所有者为test_user;
openGauss=# ALTER TABLE IF EXISTS test_schema.test_alt1 OWNER TO test_user;
-- 查看
openGauss=# SELECT tablename, schemaname, tableowner FROM pg_tables WHERE tablename =
'test_alt1';
 tablename | schemaname  | tableowner
-----------+-------------+------------
 test_alt1 | test_schema | test_user
(1 row)
```

- **修改表的表空间**

```
--创建表空间tbs_data1。
openGauss=# CREATE TABLESPACE tbs_data1 RELATIVE LOCATION 'tablespace1/tbs_data1';
--修改test_alt1表的空间为tbs_data1。
openGauss=# ALTER TABLE test_schema.test_alt1 SET TABLESPACE tbs_data1;
-- 查看。
```



```
openGauss=# SELECT tablename, tablespace FROM pg_tables WHERE tablename = 'test_alt1';
 tablename | tablespace
-----------+------------
 test_alt1 | tbs_data1
(1 row)

--删除。
openGauss=# DROP TABLE test_schema.test_alt1;
openGauss=# DROP TABLESPACE tbs_data1;
openGauss=# DROP SCHEMA test_schema;
openGauss=# DROP USER test_user;
```

修改列示例

- **修改列名**

```
-- 建表
openGauss=# CREATE TABLE test_alt2(c1 INT,c2 INT);
-- 修改列名
openGauss=# ALTER TABLE test_alt2 RENAME c1 TO id;
openGauss=# ALTER TABLE test_alt2 RENAME COLUMN c2 to areaid;
-- 查看
openGauss=# \d test_alt1
   Table "public.test_alt1"
 Column |  Type   | Modifiers
--------+---------+-----------
 id     | integer |
 areaid | integer |
```

- **增加列**

```
-- 表test_alt1增加列
openGauss=# ALTER TABLE IF EXISTS test_alt2 ADD COLUMN name VARCHAR(20);
-- 查看
openGauss=# \d test_alt2
           Table "public.test_alt1"
  Column  |          Type          | Modifiers
----------+------------------------+-----------
 id       | integer                |
 areacode | integer                |
 name     | character varying(20) |
```

- **修改列的数据类型**

```
-- 修改test_alt1表中name字段的类型
openGauss=# ALTER TABLE test_alt1 MODIFY name VARCHAR(50);
-- 查看
openGauss=# \d test_alt1
          Table "public.test_alt2"
 Column |          Type          | Modifiers
--------+------------------------+-----------
 id     | integer                |
 areaid | integer                |
 name   | character varying(50) |
-- 修改test_alt1表中name字段的类型
openGauss=# ALTER TABLE test_alt2 ALTER COLUMN name TYPE VARCHAR(25);
-- 查看
openGauss=# \d test_alt2
          Table "public.test_alt2"
 Column |          Type          | Modifiers
--------+------------------------+-----------
 id     | integer                |
 areaid | integer                |
 name   | character varying(25) |
```

- **删除列**

```
-- 删除test_alt1中areaid字段
openGauss=# ALTER TABLE test_alt2 DROP COLUMN areaid;
-- 查看
openGauss=# \d test_alt2
          Table "public.test_alt2"
 Column |          Type          | Modifiers
--------+------------------------+-----------
```



```
 id     | integer               |
 name   | character varying(25) |
```

- **修改字段存储模式**

```
--查看表详细信息。
openGauss=# \d+ test_alt2
                          Table "public.test_alt2"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+-------------
 id     | integer               |           | plain    |              |
 name   | character varying(25) |           | extended |              |
Has OIDs: no
Options: orientation=row, storage_type=USTORE

--修改test_alt2表中name字段的存储模式。
openGauss=# ALTER TABLE test_alt2 ALTER COLUMN name SET STORAGE PLAIN;

--查看。
openGauss=# \d+ test_alt2
                          Table "public.test_alt2"
 Column |         Type          | Modifiers | Storage | Stats target | Description
--------+-----------------------+-----------+---------+--------------+-------------
 id     | integer               |           | plain   |              |
 name   | character varying(25) |           | plain   |              |
Has OIDs: no
Options: orientation=row, storage_type=USTORE

--删除。
openGauss=# DROP TABLE test_alt2;
```

修改约束示例

- **为列添加非空约束**

```
--建表。
openGauss=# CREATE TABLE test_alt3(pid INT, areaid CHAR(5), name VARCHAR(20));
--为pid添加非空约束。
openGauss=# ALTER TABLE test_alt3 MODIFY pid NOT NULL;
--查看。
openGauss=# \d test_alt3
          Table "public.test_alt3"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 pid    | integer               | not null
 areaid | character(5)          |
 name   | character varying(20) |
```

- **取消列的非空约束**

```
openGauss=# ALTER TABLE test_alt3 MODIFY pid NULL;
--查看。
openGauss=# \d test_alt3
          Table "public.test_alt3"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 pid    | integer               |
 areaid | character(5)          |
 name   | character varying(20) |
```

- **修改字段默认值**

```
--修改test_alt1表中id的默认值。
openGauss=# ALTER TABLE test_alt3 ALTER COLUMN areaid SET DEFAULT '00000';
--查看。
openGauss=# \d test_alt3
                Table "public.test_alt3"
 Column |         Type          |        Modifiers
--------+-----------------------+--------------------------
 pid    | integer               |
 areaid | character(5)          | default '00000'::bpchar
 name   | character varying(20) |
--删除id的默认值。
openGauss=# ALTER TABLE test_alt3 ALTER COLUMN areaid DROP DEFAULT;
```



```
--查看。
openGauss=# \d test_alt3
          Table "public.test_alt3"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 pid    | integer               |
 areaid | character(5)          |
 name   | character varying(20) |
```

- **添加表级约束**
  - 直接添加约束

    ```
    --给表添加主键约束。
    openGauss=# ALTER TABLE test_alt3 ADD CONSTRAINT pk_test3_pid PRIMARY KEY (pid);
    --查看。
    openGauss=# \d test_alt3
              Table "public.test_alt3"
     Column |         Type          | Modifiers
    --------+-----------------------+-----------
     pid    | integer               | not null
     areaid | integer               |
     name   | character varying(20) |
    Indexes:
        "pk_test3_pid" PRIMARY KEY, btree (pid) TABLESPACE pg_default
    ```

  - 先创建索引然后再添加约束

    ```
    --建表
    openGauss=# CREATE TABLE test_alt4(c1 INT, c2 INT);
    --建索引。
    openGauss=# CREATE UNIQUE INDEX pk_test4_c1 ON test_alt4(c1);
    --添加约束时关联已经创建的索引。
    openGauss=# ALTER TABLE test_alt4 ADD CONSTRAINT pk_test4_c1 PRIMARY KEY USING
    INDEX pk_test4_c1;
    --查看。
    openGauss=# \d test_alt4
       Table "public.test_alt4"
     Column |  Type   | Modifiers
    --------+---------+-----------
     c1     | integer | not null
     c2     | integer |
    Indexes:
        "pk_test4_c1" PRIMARY KEY, btree (c1) TABLESPACE pg_default
    --删除。
    openGauss=# DROP TABLE test_alt4;
    ```

- **删除表级约束**

```
--删除约束。
openGauss=# ALTER TABLE test_alt3 DROP CONSTRAINT IF EXISTS pk_test3_pid;
--查看。
openGauss=# \d test_alt3
          Table "public.test_alt3"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 pid    | integer               | not null
 areaid | integer               |
 name   | character varying(20) |

--删除。
openGauss=# DROP TABLE test_alt3;
```

相关链接

CREATE TABLE，**DROP TABLE**

7.13.26 ALTER TABLE PARTITION

功能描述

修改表分区，包括增加/删除分区、切割/合并分区、清空分区、移动分区表空间、交换分区、重命名分区，以及修改分区属性等。

注意事项

- 只有分区表的所有者或者被授予了分区表ALTER权限的用户有权限执行ALTER TABLE PARTITION命令，系统管理员默认拥有此权限。
- 添加分区的表空间不能是PG_GLOBAL。
- 添加分区的名称不能与该分区表已有分区的名称相同。
- 添加分区的分区键值要和分区表的分区键的类型一致，且要大于分区表中最后一个范围分区的上边界。
- 如果目标分区表中已有分区数达到了最大值（1048575），则不能继续添加分区。
- 当分区表只有一个分区时，不能删除该分区。
- 选择分区使用PARTITION FOR()，括号里指定值个数应该与定义分区时使用的列个数相同，并且一一对应。
- Value分区表不支持相应的Alter Partition操作。
- 若设置参数enable_gpi_auto_update为on，即使不声明UPDATE GLOBAL INDEX子句，也会自动更新Global索引。

语法格式

- 修改表分区主语法。

```
ALTER TABLE [ IF EXISTS ] { table_name  [*] | ONLY table_name | ONLY ( table_name  )}
    action [, ... ];
```

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

```
move_clause |
    exchange_clause |
    row_clause |
    merge_clause |
    modify_clause |
    split_clause |
    add_clause |
    drop_clause
```

  - move_clause子语法用于移动分区到新的表空间。

```
MOVE PARTITION { partion_name | FOR ( partition_value [, ...] ) } TABLESPACE tablespacename
```

  - exchange_clause子语法用于把普通表的数据迁移到指定的分区。

```
EXCHANGE PARTITION { ( partition_name ) | FOR ( partition_value [, ...] ) }
    WITH TABLE {[ ONLY ] ordinary_table_name | ordinary_table_name * | ONLY
( ordinary_table_name )}
    [ { WITH | WITHOUT } VALIDATION ] [ VERBOSE ] [ UPDATE GLOBAL INDEX ]
```

    进行交换的普通表和分区必须满足如下条件：

    - 普通表和分区的列数目相同，对应列的信息严格一致，包括：列名、列的数据类型、列约束、列的Collation信息、列的存储参数等。

- 普通表和分区的分布列信息严格一致。
- 普通表和分区的索引个数相同，且对应索引的信息严格一致。
- 普通表和分区的表约束个数相同，且对应表约束的信息严格一致。
- 普通表不可以是临时表。
- 在内置安全策略开关开启的情况下，普通表不可以包含绑定了动态数据脱敏策略的列。

完成交换后，普通表和分区的数据被置换，同时普通表和分区的表空间信息被置换。此时，普通表和分区的统计信息变得不可靠，需要对普通表和分区重新执行analyze。如果在普通表/分区表上进行了drop column操作，被删除的列依然物理存在，所以需要保证普通表和分区的被删除列也严格对齐才能交换成功。

– row_clause子语法用于设置分区表的行迁移开关。

```
{ ENABLE | DISABLE } ROW MOVEMENT
```

– merge_clause子语法用于把多个分区合并成一个分区。

```
MERGE PARTITIONS { partition_name } [, ...] INTO PARTITION partition_name
    [ TABLESPACE tablespacename ] [ UPDATE GLOBAL INDEX ]
```

– modify_clause子语法用于设置分区索引是否可用。

```
MODIFY PARTITION partition_name { UNUSABLE LOCAL INDEXES | REBUILD UNUSABLE LOCAL INDEXES }
```

– split_clause子语法用于把一个分区切割成多个分区。

```
SPLIT PARTITION { partition_name | FOR ( partition_value [, ...] ) } { split_point_clause | no_split_point_clause } [ UPDATE GLOBAL INDEX ]
```

- 指定切割点split_point_clause的语法为：

```
AT ( partition_value ) INTO ( PARTITION partition_name [ TABLESPACE tablespacename ] , PARTITION partition_name [ TABLESPACE tablespacename ] )
```

须知

- 切割点的大小要位于正在被切割的分区的分区键范围内，指定切割点的方式只能把一个分区切割成两个新分区。

- 不指定切割点no_split_point_clause的语法为：

```
INTO { ( partition_less_than_item [, ...] ) | ( partition_start_end_item [, ...] ) }
```


须知

- 不指定切割点的方式，partition_less_than_item指定的第一个新分区的分区键要大于正在被切割的分区的前一个分区（如果存在的话）的分区键，partition_less_than_item指定的最后一个分区的分区键要等于正在被切割的分区的分区键大小。
- 不指定切割点的方式，partition_start_end_item指定的第一个新分区的起始点（如果存在的话）必须等于正在被切割的分区的前一个分区（如果存在的话）的分区键，partition_start_end_item指定的最后一个分区的终止点（如果存在的话）必须等于正在被切割的分区的分区键。
- partition_less_than_item支持的分区键个数最多为4，而partition_start_end_item仅支持1个分区键，其支持的数据类型参见**•PARTITION BY RANGE(partition_key)**。
- 在同一语句中partition_less_than_item和partition_start_end_item两者不可同时使用；不同split语句之间没有限制。

  - 分区项partition_less_than_item的语法为：

    ```
    PARTITION partition_name VALUES LESS THAN ( { partition_value | MAXVALUE }  [, ...] )
       [ TABLESPACE tablespacename ]
    ```

  - 分区项partition_start_end_item的语法为，其约束参见**START END语法描述**。

    ```
    PARTITION partition_name {
          {START(partition_value) END (partition_value) EVERY (interval_value)} |
          {START(partition_value) END ({partition_value | MAXVALUE})} |
          {START(partition_value)} |
          {END({partition_value | MAXVALUE})}
    } [TABLESPACE tablespace_name]
    ```

- add_clause子语法用于为指定的分区表添加一个或多个分区。

  ```
  ADD PARTITION ( partition_col1_name = partition_col1_value [, partition_col2_name =
  partition_col2_value ] [, ...] )
     [ LOCATION 'location1' ]
     [ PARTITION (partition_colA_name = partition_colA_value [, partition_colB_name =
  partition_colB_value ] [, ...] ) ]
     [ LOCATION 'location2' ]
  ADD {partition_less_than_item | partition_start_end_item}
  ```

- drop_clause子语法用于删除分区表中的指定分区。

  ```
  DROP PARTITION  { partition_name | FOR (  partition_value [, ...] )  } [ UPDATE GLOBAL INDEX ]
  ```

- 修改表分区名称的语法。

  ```
  ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name  )}
     RENAME PARTITION { partion_name | FOR ( partition_value [, ...] ) } TO partition_new_name;
  ```

参数说明

- **table_name**

  分区表名。

  取值范围：已存在的分区表名。

- **partition_name**

  分区名。

  取值范围：已存在的分区名。

- **tablespacename**

  指定分区要移动到哪个表空间。

取值范围：已存在的表空间名。

- **partition_value**

  分区键值。

  通过PARTITION FOR（partition_value [, ...])子句指定的这一组值，可以唯一确定一个分区。

  取值范围：需要进行重命名的分区的分区键的取值范围。

- **UNUSABLE LOCAL INDEXES**

  设置该分区上的所有索引不可用。

- **REBUILD UNUSABLE LOCAL INDEXES**

  重建该分区上的所有索引。

- **{ ENABLE | DISABLE } ROW MOVEMET**

  行迁移开关。

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

  取值范围：

  - ENABLE：打开行迁移开关。
  - DISABLE：关闭行迁移开关。

  默认是关闭状态。

- **ordinary_table_name**

  进行迁移的普通表的名称。

  取值范围：已存在的普通表名。

- **{ WITH | WITHOUT } VALIDATION**

  在进行数据迁移时，是否检查普通表中的数据满足指定分区的分区键范围。

  取值范围：

  - WITH：对于普通表中的数据要检查是否满足分区的分区键范围，如果有数据不满足，则报错。
  - WITHOUT：对于普通表中的数据不检查是否满足分区的分区键范围。

  默认是WITH状态。

  由于检查比较耗时，特别是当数据量很大的情况下更甚。所以在保证当前普通表中的数据满足分区的分区键范围时，可以加上WITHOUT来指明不进行检查。

- **VERBOSE**

  在VALIDATION是WITH状态时，如果检查出普通表有不满足要交换分区的分区键范围的数据，那么把这些数据插入到正确的分区，如果路由不到任何分区，再报错。

  须知

  只有在VALIDATION是WITH状态时，才可以指定VERBOSE。

- **partition_new_name**

  分区的新名称。

  取值范围：字符串，要符合**标识符命名规范**。

- **UPDATE GLOBAL INDEX**

  如果使用该参数，则会更新分区表上的所有全局索引，以确保使用全局索引可以查询出正确的数据。

  如果不使用该参数，则分区表上的所有全局索引将会失效。

示例

请参考CREATE TABLE PARTITION的**示例**。

相关链接

CREATE TABLE PARTITION，**DROP TABLE**

7.13.27 ALTER TABLESPACE

功能描述

修改表空间的属性。

注意事项

- 当前版本禁止使用ALTER TABLESPACE语法。
- 只有表空间的所有者或者被授予了表空间ALTER权限的用户有权限执行ALTER TABLESPACE命令，系统管理员默认拥有此权限。但要修改表空间的所有者，当前用户必须是该表空间的所有者或系统管理员，且该用户是新所有者角色的成员。
- 对行存表的ALTER TABLESPACE操作不支持在事务块中执行。
- 要修改表空间的所有者A为B，则A必须是B的直接或者间接成员。

  说明

  如果new_owner与old_owner一致，此处不再校验当前执行操作的用户是否具有修改权限，而直接显示ALTER成功。

语法格式

- 重命名表空间的语法。

```
ALTER TABLESPACE tablespace_name
    RENAME TO new_tablespace_name;
```

- 设置表空间所有者的语法。

```
ALTER TABLESPACE tablespace_name
    OWNER TO new_owner;
```

- 设置表空间属性的语法。

```
ALTER TABLESPACE tablespace_name
    SET ( {tablespace_option = value} [, ... ] );
```

- 重置表空间属性的语法。

```
ALTER TABLESPACE tablespace_name
    RESET ( { tablespace_option } [, ...] );
```

- 设置表空间限额的语法

```
ALTER TABLESPACE tablespace_name
    RESIZE MAXSIZE { UNLIMITED | 'space_size'};
```


参数说明

- **tablespace_name**

  要修改的表空间。

  取值范围：已存在的表空间名。

- **new_tablespace_name**

  表空间的新名称。

  新名称不能以"PG_"开头。

  取值范围：字符串，符合**标识符命名规范**。

- **new_owner**

  表空间的新所有者。

  取值范围：已存在的用户名。

- **tablespace_option**

  设置或者重置表空间的参数。

  取值范围：

  - seq_page_cost：设置优化器计算一次顺序获取磁盘页面的开销。缺省为1.0。
  - random_page_cost：设置优化器计算一次非顺序获取磁盘页面的开销。缺省为4.0。

    说明

    - random_page_cost是相对于seq_page_cost的取值，等于或者小于seq_page_cost时毫无意义。
    - 默认值为4.0的前提条件是，优化器采用索引来扫描表数据，并且表数据在cache中命中率可以90%左右。
    - 如果表数据空间要比物理内存小，那么减小该值到一个适当水平；相反地，如果表数据在cache中命中率要低于90%，那么适当增大该值。
    - 如果采用了类似于SSD的随机访问代价较小的存储器，可以适当减小该值，以反映真正的随机扫描代价。

  value的取值范围：浮点类型的正数。

- **RESIZE MAXSIZE**

  重新设置表空间限额的数值。

  取值范围：

  - UNLIMITED，该表空间不设置限额。
  - 由space_size来确定，其格式参考**CREATE TABLESPACE**。

    说明

    - 若调整后的限额值比当前表空间实际使用的值要小，调整操作可以执行成功，后续用户需要将该表空间的使用值降低到新限额值之下，才能继续往该表空间中写入数据。
    - 修改参数MAXSIZE时也可使用：

      ```
      ALTER TABLESPACE tablespace_name RESIZE MAXSIZE
       { 'UNLIMITED' | 'space_size'};
      ```

示例

请参考CREATE TABLESPACE的**示例**。

相关链接

CREATE TABLESPACE，DROP TABLESPACE

7.13.28 ALTER TRIGGER

功能描述

修改触发器名称。

说明

目前只支持修改名称。

注意事项

只有触发器所在表的所有者可以执行ALTER TRIGGER操作，系统管理员默认拥有此权限。

语法格式

```
ALTER TRIGGER trigger_name ON table_name RENAME TO new_name;
```

参数说明

- **trigger_name**

  要修改的触发器名称。

  取值范围：已存在的触发器。

- **table_name**

  要修改的触发器所在的表名称。

  取值范围：已存在的含触发器的表。

- **new_name**

  修改后的新名称。

  取值范围：符合**标识符命名规范**的字符串，最大长度不超过63个字符，且不能与所在表上其他触发器同名。

示例

请参见**CREATE TRIGGER**的示例。

相关链接

CREATE TRIGGER，DROP TRIGGER，ALTER TABLE

7.13.29 ALTER TYPE

功能描述

修改一个类型的定义。

注意事项

只有类型的所有者或者被授予了类型ALTER权限的用户可以执行ALTER TYPE命令，系统管理员默认拥有此权限。但要修改类型的所有者或者修改类型的模式，当前用户必须是该类型的所有者或者系统管理员，且该用户是新所有者角色的成员。

语法格式

- 修改类型
  ```
  ALTER TYPE name action [, ... ];
  ```
  其中action对应的子句如下:
  - 给复合类型增加新的属性。
    ```
    ADD ATTRIBUTE attribute_name data_type [ COLLATE collation ] [ CASCADE | RESTRICT ]
    ```
  - 从复合类型中删除一个属性。
    ```
    DROP ATTRIBUTE [ IF EXISTS ] attribute_name [ CASCADE | RESTRICT ]
    ```
  - 改变一种复合类型中某个属性的类型。
    ```
    ALTER ATTRIBUTE attribute_name [ SET DATA ] TYPE data_type [ COLLATE collation ]
    [ CASCADE | RESTRICT ]
    ```
- 给复合类型增加新的属性。
  ```
  ALTER TYPE name ADD ATTRIBUTE attribute_name data_type [ COLLATE collation ] [ CASCADE |
  RESTRICT ];
  ```
- 从复合类型删除一个属性。
  ```
  ALTER TYPE name DROP ATTRIBUTE [ IF EXISTS ] attribute_name [ CASCADE | RESTRICT ];
  ```
- 改变一种复合类型中某个属性的类型。
  ```
  ALTER TYPE name ALTER ATTRIBUTE attribute_name [ SET DATA ] TYPE data_type [ COLLATE
  collation ] [ CASCADE | RESTRICT ];
  ```
- 改变类型的所有者。
  ```
  ALTER TYPE name OWNER TO { new_owner | CURRENT_USER | SESSION_USER };
  ```
- 改变类型的名称。
  ```
  ALTER TYPE name RENAME TO new_name;
  ```
- 改变一个复合类型中一个属性的名称。
  ```
  ALTER TYPE name RENAME ATTRIBUTE attribute_name TO new_attribute_name [ CASCADE |
  RESTRICT ];
  ```
- 将类型移至一个新的模式中。
  ```
  ALTER TYPE name SET SCHEMA new_schema;
  ```
- 为枚举类型增加一个新值。
  ```
  ALTER TYPE name ADD VALUE [ IF NOT EXISTS ] new_enum_value [ { BEFORE | AFTER }
  neighbor_enum_value ];
  ```
- 重命名枚举类型的一个标签值。
  ```
  ALTER TYPE name RENAME VALUE existing_enum_value TO new_enum_value;
  ```

参数说明

- **name**

  一个需要修改的现有的类型的名称(可以有模式修饰) 。
- **new_name**

  该类型的新名称。
- **new_owner**

  新所有者的用户名 。
- **new_schema**

  该类型的新模式 。

- **attribute_name**
  拟增加、更改或删除的属性的名称。
- **new_attribute_name**
  拟改名的属性的新名称。
- **data_type**
  拟新增属性的数据类型，或是拟更改的属性的新类型名。
- **new_enum_value**
  枚举类型新增加的标签值，是一个非空的长度不超过64个字节的字符串。
- **neighbor_enum_value**
  一个已有枚举标签值，新值应该被增加在紧接着该枚举值之前或者之后的位置上。
- **existing_enum_value**
  现有的要重命名的枚举值，是一个非空的长度不超过64个字节的字符串
- **CASCADE**
  自动级联更新需更新类型以及相关联的记录和继承它们的子表。
- **RESTRICT**
  如果需联动更新类型是已更新类型的关联记录，则拒绝更新。这是缺省选项。

须知

- ADD ATTRIBUTE、DROP ATTRIBUTE和ALTER ATTRIBUTE选项可以组合成一个列表同时处理。 例如，在一条命令中同时增加几个属性或是更改几个属性的类型是可以实现的。
- 要修改一个类型的模式，必须在新模式上拥有CREATE权限。要修改所有者，必须是新的所有角色的直接或间接成员，并且该成员必须在此类型的模式上有CREATE权限（这些限制强制了修改所有者不会做任何通过删除和重建类型不能做的事情。不过，系统管理员可以以任何方式修改任意类型的所有权）。要增加一个属性或是修改一个属性的类型，也必须有该类型的USAGE权限。

示例

请参考CREATE TYPE的**示例**。

相关链接

CREATE TYPE，**DROP TYPE**

7.13.30 ALTER USER

功能描述

修改数据库用户的属性。

注意事项

ALTER USER中修改的会话参数只针对指定的用户，且在下一次会话中有效。

语法格式

- 修改用户的权限等信息。

```
ALTER USER user_name [ [ WITH ] option [ ... ] ];
```

其中option子句为。

```
{ CREATEDB | NOCREATEDB }
    | { CREATEROLE | NOCREATEROLE }
    | { INHERIT | NOINHERIT }
    | { AUDITADMIN | NOAUDITADMIN }
    | { SYSADMIN | NOSYSADMIN }
    | {MONADMIN | NOMONADMIN}
    | {OPRADMIN | NOOPRADMIN}
    | {POLADMIN | NOPOLADMIN}
    | { USEFT | NOUSEFT }
    | { LOGIN | NOLOGIN }
    | { REPLICATION | NOREPLICATION }
    | {VCADMIN | NOVCADMIN}
    | {PERSISTENCE | NOPERSISTENCE}
    | CONNECTION LIMIT connlimit
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD { 'password' [EXPIRED] | DISABLE | EXPIRED }
    | [ ENCRYPTED | UNENCRYPTED ] IDENTIFIED BY { 'password' [ REPLACE 'old_password' |
EXPIRED ] | DISABLE }
    | VALID BEGIN 'timestamp'
    | VALID UNTIL 'timestamp'
    | USER GROUP 'groupuser'
    | PERM SPACE 'spacelimit'
    | TEMP SPACE 'tmpspacelimit'
    | SPILL SPACE 'spillspacelimit'
    | NODE GROUP logic_cluster_name
    | PGUSER
```

- 修改用户名。

```
ALTER USER user_name
    RENAME TO new_name;
```

- 锁定或解锁。

```
ALTER USER user_name
    ACCOUNT { LOCK | UNLOCK };
```

参数说明

- **user_name**

  现有用户名。

  取值范围：已存在的用户名，如果用户名中包含大写字母则需要使用双引号括起来。

- **new_password**

  新密码。

  密码规则如下：

  - 不能与当前密码相同。
  - 密码默认不少于8个字符。
  - 不能与用户名及用户名倒序相同。
  - 至少包含大写字母（A-Z），小写字母（a-z），数字（0-9），非字母数字字符（限定为~!@#$%^&*()-_=+\|[{}];:,<.>/?）四类字符中的三类字符。当密码中包含的字符不属于上述四种字符范围内时语句执行会报错。

  取值范围：字符串。

- **old_password**

  旧密码。

- **ACCOUNT { LOCK | UNLOCK }**
  - ACCOUNT LOCK：锁定账户，禁止登录数据库。
  - ACCOUNT UNLOCK：解锁账户，允许登录数据库。
- **PGUSER**
  当前版本不允许修改用户的PGUSER属性。

其他参数请参见**CREATE ROLE**和**ALTER ROLE**的参数说明。

须知

当前版本不支持设置用户级别参数。

示例

请参考CREATE USER的**示例**。

相关链接

CREATE ROLE，**CREATE USER**，**DROP USER**

7.13.31 ALTER VIEW

功能描述

ALTER VIEW更改视图的各种辅助属性（如果用户是更改视图的查询定义，要使用CREATE OR REPLACE VIEW）。

注意事项

只有视图的所有者或者被授予了视图ALTER权限的用户才可以执行ALTER VIEW命令，系统管理员默认拥有该权限。针对所要修改属性的不同，对其还有以下权限约束：

- 修改视图的模式，当前用户必须是视图的所有者或者系统管理员，且要有新模式的CREATE权限。
- 修改视图的所有者，当前用户必须是视图的所有者或者系统管理员，且该用户必须是新所有者角色的成员，并且此角色必须有视图所在模式的CREATE权限。
- 禁止修改视图中列的类型。

语法格式

- 设置视图列的默认值。
```
ALTER VIEW [ IF EXISTS ] view_name
    ALTER [ COLUMN ] column_name SET DEFAULT expression;
```
- 取消列视图列的默认值。
```
ALTER VIEW [ IF EXISTS ] view_name
    ALTER [ COLUMN ] column_name DROP DEFAULT;
```
- 修改视图的所有者。
```
ALTER VIEW [ IF EXISTS ] view_name
    OWNER TO new_owner;
```
- 重命名视图。


```
ALTER VIEW [ IF EXISTS ] view_name
    RENAME TO new_name;
```

- 设置视图的所属模式。

```
ALTER VIEW [ IF EXISTS ] view_name
    SET SCHEMA new_schema;
```

- 设置视图的选项。

```
ALTER VIEW [ IF EXISTS ] view_name
    SET ( { view_option_name [ = view_option_value ] } [, ... ] );
```

- 重置视图的选项。

```
ALTER VIEW [ IF EXISTS ] view_name
    RESET ( view_option_name [, ... ] );
```

参数说明

- **IF EXISTS**

  使用这个选项，如果视图不存在时不会产生错误，仅会有一个提示信息。

- **view_name**

  视图名称，可以用模式修饰。

  取值范围：字符串，符合**标识符命名规范**。

- **column_name**

  可选的名称列表，视图的字段名。如果没有给出，字段名取自查询中的字段名。

  取值范围：字符串，符合**标识符命名规范**。

- **SET/DROP DEFAULT**

  设置或删除一个列的缺省值，该参数暂无实际意义。

- **new_owner**

  视图新所有者的用户名称。

- **new_name**

  视图的新名称。

- **new_schema**

  视图的新模式。

- **view_option_name [= view_option_value]**

  该子句为视图指定一个可选的参数。

  目前view_option_name支持的参数仅有security_barrier，当VIEW视图提供行级安全时，应使用该参数。

  取值范围：Boolean类型，TRUE、FALSE。

示例

```
--创建SCHEMA。
openGauss=# CREATE SCHEMA tpcds;

--创建表tpcds.customer。
openGauss=# CREATE TABLE tpcds.customer
(
c_customer_sk         INTEGER          NOT NULL,
c_customer_id         CHARACTER(16)    NOT NULL
);

--向表中插入多条记录。
openGauss=# INSERT INTO tpcds.customer VALUES (1, 'AAAAAAAABAAAAAAA'),(100,
'AAAAAAAACAAAAAAA'),(150, 'AAAAAAAADAAAAAAA');
```



```
--创建一个由c_customer_sk小于150的内容组成的视图。
openGauss=# CREATE VIEW tpcds.customer_details_view_v1 AS
    SELECT * FROM tpcds.customer
    WHERE c_customer_sk < 150;

--修改视图名称。
openGauss=# ALTER VIEW tpcds.customer_details_view_v1 RENAME TO customer_details_view_v2;

--修改视图所属schema。
openGauss=# ALTER VIEW tpcds.customer_details_view_v2 SET schema public;

--删除视图。
openGauss=# DROP VIEW public.customer_details_view_v2;

--删除表tpcds.customer。
openGauss=# DROP TABLE tpcds.customer;

--删除SCHEMA。
openGauss=# DROP SCHEMA tpcds CASCADE;
```

相关链接

CREATE VIEW，**DROP VIEW**

7.13.32 ANALYZE | ANALYSE

功能描述

- 用于收集与数据库中普通表内容相关的统计信息，统计结果存储在系统表PG_STATISTIC、PG_STATISTIC_EXT下，执行ANALYZE命令后，可在上述系统表或系统视图PG_STATS、PG_EXT_STATS内查询收集到的统计信息。执行计划生成器会使用这些统计数据，以确定最有效的执行计划。
- 如果没有指定参数，ANALYZE会分析当前数据库中的每个表和分区表。同时也可以通过指定table_name、column_name和partition_name参数把分析限定在特定的表、列或分区表中。
- ANALYZE|ANALYSE VERIFY用于检测数据库中普通表（行存表）的数据文件是否损坏。

注意事项

- ANALYZE非临时表不能在一个匿名块、事务块、函数或存储过程内被执行。支持存储过程中ANALYZE临时表，不支持统计信息回滚操作。
- ANALYZE VERIFY场景不触发远程读，因此远程读参数不生效。对于关键系统表出现错误被系统检测出页面损坏时，将直接报错不再继续检测。
- 如果没有指定参数，ANALYZE处理当前数据库里用户拥有相应权限的每个表。如果参数指定了一个表，ANALYZE只处理指定的那个表。
- 要对一个表进行ANALYZE操作，通常用户必须是表的所有者或者被授予了指定表VACUUM权限的用户，默认系统管理员有该权限。数据库的所有者允许对数据库中除了共享目录以外的所有表进行ANALYZE操作（该限制意味着只有系统管理员才能真正对一个数据库进行ANALYZE操作）。ANALYZE命令会跳过那些用户没有权限的表。
- ANALYZE不收集无法做比较或等值运算的列，例如cursor类型。

语法格式

- 收集表的统计信息。
```
{ ANALYZE | ANALYSE } [ VERBOSE ]
    [ table_name [ ( column_name [, ...] ) ] ];
```
- 收集分区表的分区统计信息，该语法在功能上尚不支持。
```
{ ANALYZE | ANALYSE } [ VERBOSE ]
    table_name [ ( column_name [, ...] ) ] PARTITION ( patrition_name ) ;
```

  说明

  普通分区表目前支持针对某个分区的统计信息的语法，但功能上不支持针对某个分区的统计信息收集。

- 检测当前库的数据文件
```
{ ANALYZE | ANALYSE } VERIFY { FAST | COMPLETE };
```

  说明

  - Fast模式校验时，需要对校验的表有并发的DML操作，会导致校验过程中有误报的问题，因为当前Fast模式是直接从磁盘上读取，有其他线程并发修改文件时，会导致获取的数据不准确，建议离线操作。
  - 支持对全库进行操作，由于涉及的表较多，建议以重定向保存结果。
    ```
    gsql -d database -p port -f sqlfile> sqllog.txt  2>&1
    ```
  - 不支持临时表和unlog表。
  - 只有对外可见的表对外提示NOTICE，内部表的检测会包含在它所依赖的外部表，不对外显示和呈现。
  - 此命令的处理可容错ERROR级别的处理。
  - 对于全库操作时，当关键系统表出现损坏则直接报错，不再继续执行。

- 检测表和索引的数据文件
```
{ ANALYZE | ANALYSE } VERIFY { FAST  |COMPLETE } { table_name | index_name } [ CASCADE ];
```

  说明

  - 支持对普通表的操作和索引表的操作，但不支持对索引表index使用CASCADE操作。原因是由于CASCADE模式用于处理主表的所有索引表，当单独对索引表进行检测时，无需使用CASCADE模式。
  - 不支持临时表和unlog表。
  - 对于主表的检测会同步检测主表的内部表，例如toast表、cudesc表等。
  - 当提示索引表损坏时，建议使用reindex命令进行重建索引操作。

- 检测表分区的数据文件
```
{ ANALYZE | ANALYSE } VERIFY { FAST | COMPLETE } table_name PARTITION (patrition_name) [ CASCADE ];
```

  说明

  - 支持对表的单独分区进行检测操作，但不支持对索引表index使用CASCADE操作。
  - 不支持临时表和unlog表。

参数说明

- **VERBOSE**

  启用显示进度信息。

  说明

  如果指定了VERBOSE，ANALYZE发出进度信息，表明目前正在处理的表。各种有关表的统计信息也会打印出来。

- **table_name**

  需要分析的特定表的表名（可能会带模式名），如果省略，将对数据库中的所有表（非外部表）进行分析。

  对于ANALYZE收集统计信息，目前仅支持行存表的外表。

  取值范围：已有的表名。

- **column_name，column_1_name，column_2_name**

  需要分析特定列的列名，默认为所有列。

  取值范围：已有的列名。

- **partition_name**

  如果table为分区表，在关键字PARTITION后面指定分区名partition_name表示分析该分区表的统计信息。目前语法上支持分区表做ANALYZE，但功能实现上暂不支持对指定分区统计信息的分析。

  取值范围：表的某一个分区名。

- **foreign_table_name**

  需要分析的特定表的表名（可能会带模式名）。

  取值范围：已有的表名。

- **index_name**

  需要分析的特定索引表的表名（可能会带模式名）。

  取值范围：已有的表名。

- **FAST|COMPLETE**

  对于行存表，FAST模式下主要对于行存表的CRC和page header进行校验，如果校验失败则会告警; 而COMPLETE模式下，则主要对行存表的指针、tuple进行解析校验。

- **CASCADE**

  CASCADE模式下会对当前表的所有索引进行检测处理。

示例

--- 创建表。

```
openGauss=# CREATE TABLE customer_info
(
WR_RETURNED_DATE_SK      INTEGER                  ,
WR_RETURNED_TIME_SK      INTEGER                  ,
WR_ITEM_SK               INTEGER           NOT NULL,
WR_REFUNDED_CUSTOMER_SK  INTEGER
)
DISTRIBUTE BY HASH (WR_ITEM_SK);
```

--- 创建分区表。

```
openGauss=# CREATE TABLE customer_par
(
WR_RETURNED_DATE_SK      INTEGER                  ,
WR_RETURNED_TIME_SK      INTEGER                  ,
WR_ITEM_SK               INTEGER           NOT NULL,
WR_REFUNDED_CUSTOMER_SK  INTEGER
)
DISTRIBUTE BY HASH (WR_ITEM_SK)
PARTITION BY RANGE(WR_RETURNED_DATE_SK)
(
PARTITION P1 VALUES LESS THAN(2452275),
PARTITION P2 VALUES LESS THAN(2452640),
```



```
PARTITION P3 VALUES LESS THAN(2453000),
PARTITION P4 VALUES LESS THAN(MAXVALUE)
)
ENABLE ROW MOVEMENT;
```

--- 使用ANALYZE语句更新统计信息。

```
openGauss=# ANALYZE customer_info;
```

--- 使用ANALYZE VERBOSE语句更新统计信息，并输出表的相关信息。

```
openGauss=# ANALYZE VERBOSE customer_info;
INFO:  analyzing "cstore.pg_delta_3394584009"(cn_5002 pid=53078)
INFO:  analyzing "public.customer_info"(cn_5002 pid=53078)
INFO:  analyzing "public.customer_info" inheritance tree(cn_5002 pid=53078)
ANALYZE
```

说明

若环境若有故障，需查看CN的log。

--- 删除表。

```
openGauss=# DROP TABLE customer_info;
openGauss=# DROP TABLE customer_par;
```

7.13.33 BEGIN

功能描述

BEGIN可以用于开始一个匿名块，也可以用于开始一个事务。本节描述用BEGIN开始匿名块的语法，以BEGIN开始事务的语法见**START TRANSACTION**。

匿名块是能够动态地创建和执行过程代码的结构，而不需要以持久化的方式将代码作为数据库对象储存在数据库中。

注意事项

无。

语法格式

- 开启匿名块

```
[DECLARE [declare_statements]]
BEGIN
execution_statements
END;
/
```

- 开启事务

```
BEGIN [ WORK | TRANSACTION ]
 [
  {
    ISOLATION LEVEL { READ COMMITTED | READ UNCOMMITTED | SERIALIZABLE | REPEATABLE
READ }
    | { READ WRITE | READ ONLY }
  } [, ...]
 ];
```

参数说明

- **declare_statements**

  声明变量，包括变量名和变量类型，如“sales_cnt int”。

- **execution_statements**

  匿名块中要执行的语句。

  取值范围：已存在的函数名称。

示例

```
--使用匿名块输出字符串。
openGauss=# BEGIN
dbe_output.print_line('Hello');
END;
/
```

相关链接

START TRANSACTION

7.13.34 CALL

功能描述

使用CALL命令可以调用已定义的函数和存储过程。

注意事项

函数或存储过程的所有者、被授予了函数或存储过程EXECUTE权限的用户或被授予EXECUTE ANY FUNCTION权限的用户有权调用函数或存储过程，系统管理员默认拥有此权限。

语法格式

```
CALL [ schema. ] { func_name | procedure_name } ( param_expr );
```

参数说明

- **schema**

  函数或存储过程所在的模式名称。

- **func_name**

  所调用函数或存储过程的名称。

  取值范围：已存在的函数名称。

- **param_expr**

  参数列表可以用符号":="或者"=>"将参数名和参数值隔开，这种方法的好处是参数可以以任意顺序排列。若参数列表中仅出现参数值，则参数值的排列顺序必须和函数或存储过程定义时的相同。

  取值范围：已存在的函数参数名称或存储过程参数名称。

  说明

  参数可以包含入参（参数名和类型之间指定“IN”关键字）和出参（参数名和类型之间指定“OUT”关键字），使用CALL命令调用函数或存储过程时，对于非重载的函数，参数列表必须包含出参，出参可以传入一个变量或者任一常量，详见**示例**。对于重载的package函数，参数列表里可以忽略出参，忽略出参时可能会导致函数找不到。包含出参时，出参只能是常量。

示例

```
--创建一个函数func_add_sql，计算两个整数的和，并返回结果。
openGauss=# CREATE FUNCTION func_add_sql(num1 integer, num2 integer) RETURN integer
AS
BEGIN
RETURN num1 + num2;
END;
/

--按参数值传递。
openGauss=# CALL func_add_sql(1, 3);

--使用命名标记法传参。
openGauss=# CALL func_add_sql(num1 => 1,num2 => 3);
openGauss=# CALL func_add_sql(num2 := 2, num1 := 3);

--删除函数。
openGauss=# DROP FUNCTION func_add_sql;

--创建带出参的函数。
openGauss=# CREATE FUNCTION func_increment_sql(num1 IN integer, num2 IN integer, res OUT integer)
RETURN integer
AS
BEGIN
res := num1 + num2;
END;
/

--出参传入常量。
openGauss=# CALL func_increment_sql(1,2,1);

--出参传入变量。
openGauss=# DECLARE
res int;
BEGIN
func_increment_sql(1, 2, res);
dbe_output.print_line(res);
END;
/

--创建重载的函数。
openGauss=# CREATE OR REPLACE PROCEDURE package_func_overload(col int, col2 out int) PACKAGE
AS
DECLARE
    col_type text;
BEGIN
     col := 122;
         dbe_output.print_line('two out parameters ' || col2);
END;
/

openGauss=# CREATE OR REPLACE PROCEDURE package_func_overload(col int, col2 out varchar) PACKAGE
AS
DECLARE
    col_type text;
BEGIN
     col2 := '122';
         dbe_output.print_line('two varchar parameters ' || col2);
END;
/

--函数调用。
openGauss=# CALL package_func_overload(1, 'test');
openGauss=# CALL package_func_overload(1, 1);

--删除函数。
openGauss=# DROP FUNCTION package_func_overload;
openGauss=# DROP FUNCTION func_increment_sql;
```


相关链接

CREATE FUNCTION，CREATE PROCEDURE

7.13.35 CHECKPOINT

功能描述

检查点（CHECKPOINT）是一个事务日志中的点，所有数据文件都在该点被更新以反映日志中的信息，所有数据文件都将被刷新到磁盘。

设置事务日志检查点。预写式日志（WAL）缺省时在事务日志中每隔一段时间放置一个检查点。可以使用gs_guc命令设置相关运行时参数（checkpoint_segments, checkpoint_timeout和incremental_checkpoint_timeout）来调整这个原子化检查点的间隔。

注意事项

- 只有系统管理员和运维管理员可以调用CHECKPOINT。
- CHECKPOINT会立即进行检查，而不是等到下一次调度时的检查点。

语法格式

```
CHECKPOINT;
```

参数说明

无。

示例

```
--设置检查点。
openGauss=# CHECKPOINT;
```

7.13.36 CLEAN CONNECTION

功能描述

清理当前CN节点到其他指定数据库节点(CN/DN)的空闲或无效网络连接。允许在指定CN节点上清理当前CN中缓存的指定数据库、指定用户的相关空闲/无效连接。

注意事项

- 在非force模式下，该功能只清理数据库集群节点(CN/DN)之间的连接，不会影响客户端连接。
- 该功能只清理CN中已缓存的空闲/无效的连接，正在使用的正常连接不做清理。
- 该功能只在CN上执行有效，在DN中不生效。
- 可以通过查询PG_STAT_GET_POOLER_STATUS()函数查看缓存的连接，检验清理的效果。
- 建议只在数据库出现网络连接异常时执行此功能。

语法格式

```
CLEAN CONNECTION
   TO { COORDINATOR ( nodename [, ... ] ) | NODE ( nodename [, ... ] )| ALL [ CHECK ] [ FORCE ] }
   { FOR DATABASE dbname | TO USER username | FOR DATABASE dbname TO USER username };
```

参数说明

- **CHECK**

  仅在节点列表为TO ALL时可以指定。如果指定该参数，会在清理连接之前检查数据库是否被其他会话连接访问。此参数主要用于DROP DATABASE之前的连接访问检查，如果发现有其他会话连接，则将报错并停止删除数据库。

- **FORCE**

  仅在节点列表为TO ALL时可以指定，如果指定该参数，当前CN中所有和指定dbname和username相关的线程都会收到SIGTERM信号，相应的会话被强制关闭，事务会中止，网络连接被清理。

- **COORDINATOR (nodename ,nodename ... }) | NODE (nodename , nodename ...) | ALL**

  删除当前CN节点与指定节点的空闲/无效连接。有三种场景：

  - **COORDINATOR**：删除当前CN到指定CN节点上的空闲/无效连接。
  - **NODE**：删除当前CN到指定DN节点上的空闲/无效连接。
  - **ALL**：删除当前CN到所有节点上的空闲/无效连接，包括CN和DN。

  取值范围：可替换其中的nodename为已存在的节点名。

- **dbname**

  删除当前CN节点中指定数据库相关的连接。如果不指定该属性，则删除所有数据库相关的连接。

  取值范围：系统中已存在数据库名称。

- **username**

  删除当前CN节点中指定用户相关连接。如果不指定，则删除所有用户相关的连接。

  取值范围：已存在的用户。

示例

```
--创建数据库test_clean_connection。
openGauss=# CREATE DATABASE test_clean_connection;

--创建jack用户。
openGauss=# CREATE USER jack PASSWORD '********';

--删除与数据库template1相关的当前CN节点与dn1和dn2节点的空闲/无效连接。
openGauss=# CLEAN CONNECTION TO NODE (dn_6001_6002,dn_6003_6004) FOR DATABASE template1;

--删除与用户jack相关的当前CN节点与dn1节点的空闲/无效连接。
openGauss=# CLEAN CONNECTION TO NODE (dn_6001_6002) TO USER jack;

--删除与数据库test_clean_connection相关的当前CN节点与所有节点的连接。
openGauss=# CLEAN CONNECTION TO ALL FORCE FOR DATABASE test_clean_connection;

--删除用户jack。
openGauss=# DROP USER jack;

--删除数据库test_clean_connection。
openGauss=# DROP DATABASE test_clean_connection;
```


7.13.37 CLOSE

功能描述

CLOSE释放和一个游标关联的所有资源。

注意事项

- 不允许对一个已关闭的游标再做任何操作。
- 一个不再使用的游标应该尽早关闭。
- 当创建游标的事务用COMMIT或ROLLBACK终止之后，每个不可保持的已打开游标都隐含关闭。
- 当创建游标的事务通过ROLLBACK退出之后，每个可以保持的游标都将隐含关闭。
- 当创建游标的事务成功提交，可保持的游标将保持打开，直到执行一个明确的CLOSE或者客户端断开。
- GaussDB没有明确打开游标的OPEN语句，因为一个游标在使用CURSOR命令定义的时候就打开了。可以通过查询系统视图pg_cursors看到所有可用的游标。

语法格式

```
CLOSE { cursor_name | ALL } ;
```

参数说明

- **cursor_name**

  一个待关闭的游标名称。

- **ALL**

  关闭所有已打开的游标。

示例

请参考FETCH的**示例**。

相关链接

FETCH，**MOVE**

7.13.38 CLUSTER

功能描述

- 根据一个索引对表进行聚簇排序。
- CLUSTER指定GaussDB通过索引名指定的索引聚簇由表名指定的表。表名上必须已经定义该索引。
- 当对一个表聚簇后，该表将基于索引信息进行物理存储。聚簇是一次性操作：当表被更新之后，更改的内容不会被聚簇。也就是说，系统不会试图按照索引顺序对新的存储内容及更新记录进行重新聚簇。
- 在对一个表聚簇之后，GaussDB会记录该表在哪一个索引上建立了聚簇。CLUSTER table_name将在该表之前记录过的聚簇索引上重新聚簇。用户也可以用

ALTER TABLE table_name CLUSTER on index_name来设置指定表用于后续聚簇操作的索引，或使用ALTER TABLE table_name SET WITHOUT CLUSTER来清除指定表之前设置的聚簇索引。

- 不含参数的CLUSTER命令会将当前用户所拥有的数据库中的先前做过聚簇的所有表重新处理。如果系统管理员调用这个命令，则对所有进行过聚簇的表重新聚簇。
- 在对一个表进行聚簇的时候，会在其上请求一个ACCESS EXCLUSIVE锁。这样就避免了在CLUSTER完成之前对该表执行其它的操作(包括读写)。

注意事项

- 只有行存B-tree索引支持CLUSTER操作。
- 如果用户只是随机访问表中的行，那么表中数据的实际存储顺序是无关紧要的。但是，如果对某些特定数据的访问次数较多，而且有一个索引将这些数据分组，那么使用CLUSTER索引对性能会有所提升。
- 如果一个请求从表中查找的索引是一个范围，或者是一个索引值对应多行，CLUSTER也会有助于应用，因为如果索引标识出了第一匹配行所在的存储页，所有其它行也可能也已经在同一个存储页里了，这样便节省了磁盘访问的时间，加速了查询。
- 在聚簇过程中，系统会先创建一个按照索引顺序建立的表的临时备份，同时也建立表上的每个索引的临时备份。因此，聚簇过程中需要保证磁盘上有足够的剩余空间，至少是表大小与全部索引大小之和。
- 因为CLUSTER记录着哪些索引曾被用于聚簇，所以用户可以在第一次手动指定索引，对指定表进行聚簇，然后设置一个周期化执行的维护脚本，只需执行不带参数的CLUSTER命令，就可以实现对想要周期性聚簇的表进行自动更新。
- 因为优化器记录着有关表的排序的统计，在表上执行聚簇操作后，需运行ANALYZE操作以确保优化器具备最新的排序信息，否则，优化器可能会选择非最优的查询规划。
- CLUSTER不允许在事务中执行。
- 如果没有将GUC参数xc_maintenance_mode设置为on，那么CLUSTER操作将跳过所有系统表。

语法格式

- 对一个表进行聚簇排序。
```
CLUSTER [ VERBOSE ] table_name [ USING index_name ];
```
- 对一个分区进行聚簇排序。
```
CLUSTER [ VERBOSE ] table_name PARTITION ( partition_name ) [ USING index_name ];
```
- 对已做过聚簇的表重新进行聚簇。
```
CLUSTER [ VERBOSE ];
```

参数说明

- **VERBOSE**

  启用显示进度信息。

- **table_name**

  表名称。

  取值范围：已存在的表名称。

- **index_name**
  索引名称。
  取值范围：已存在的索引名称。
- **partition_name**
  分区名称。
  取值范围：已存在的分区名称。

示例

```
--创建SCHEMA。
openGauss=# CREATE SCHEMA tpcds;

-- 创建一个分区表。
openGauss=# CREATE TABLE tpcds.inventory_p1
(
   INV_DATE_SK             INTEGER              NOT NULL,
   INV_ITEM_SK             INTEGER              NOT NULL,
   INV_WAREHOUSE_SK        INTEGER              NOT NULL,
   INV_QUANTITY_ON_HAND    INTEGER
)
DISTRIBUTE BY HASH(INV_ITEM_SK)
PARTITION BY RANGE(INV_DATE_SK)
(
      PARTITION P1 VALUES LESS THAN(2451179),
      PARTITION P2 VALUES LESS THAN(2451544),
      PARTITION P3 VALUES LESS THAN(2451910),
      PARTITION P4 VALUES LESS THAN(2452275),
      PARTITION P5 VALUES LESS THAN(2452640),
      PARTITION P6 VALUES LESS THAN(2453005),
      PARTITION P7 VALUES LESS THAN(MAXVALUE)
);

-- 创建索引ds_inventory_p1_index1。
openGauss=# CREATE INDEX ds_inventory_p1_index1 ON tpcds.inventory_p1 (INV_ITEM_SK) LOCAL;

-- 对表tpcds.inventory_p1进行聚簇。
openGauss=# CLUSTER tpcds.inventory_p1 USING ds_inventory_p1_index1;

-- 对分区p3进行聚簇。
openGauss=# CLUSTER tpcds.inventory_p1 PARTITION (p3) USING ds_inventory_p1_index1;

-- 对已做过聚簇的表重新进行聚簇。
openGauss=# CLUSTER;

--删除索引。
openGauss=# DROP INDEX tpcds.ds_inventory_p1_index1;

--删除分区表。
openGauss=# DROP TABLE tpcds.inventory_p1;

--删除SCHEMA。
openGauss=# DROP SCHEMA tpcds CASCADE;
```

7.13.39 COMMENT

功能描述

定义或修改一个对象的注释。

注意事项

- 每个对象只存储一条注释，因此要修改一个注释，对同一个对象发出一条新的COMMENT命令即可。要删除注释，在文本字符串的位置写上NULL即可。当删除对象时，注释自动被删除掉。
- 目前注释浏览没有安全机制：任何连接到某数据库上的用户都可以看到所有该数据库对象的注释。共享对象（比如数据库、角色、表空间）的注释是全局存储的，连接到任何数据库的任何用户都可以看到它们。因此，不要在注释里存放与安全有关的敏感信息。
- 对大多数对象，只有对象的所有者或者被授予了对象COMMENT权限的用户可以设置注释，系统管理员默认拥有该权限。
- 角色没有所有者，所以COMMENT ON ROLE命令仅可以由系统管理员对系统管理员角色执行，有CREATEROLE权限的角色也可以为非系统管理员角色设置注释。系统管理员可以对所有对象进行注释。

语法格式

```
COMMENT ON
{
  AGGREGATE agg_name (agg_type [, ...] ) |
  CAST (source_type AS target_type) |
  COLLATION object_name |
  COLUMN { table_name.column_name | view_name.column_name } |
  CONSTRAINT constraint_name ON table_name |
  CONVERSION object_name |
  DATABASE object_name |
  DOMAIN object_name |
  EXTENSION object_name |
  FUNCTION function_name ( [ {[ argname ] [ argmode ] argtype} [, ...] ] ) |
  INDEX object_name |
  LARGE OBJECT large_object_oid |
  OPERATOR operator_name (left_type, right_type) |
  OPERATOR CLASS object_name USING index_method |
  OPERATOR FAMILY object_name USING index_method |
  [ PROCEDURAL ] LANGUAGE object_name |
  ROLE object_name |
  SCHEMA object_name |
  SERVER object_name |
  TABLE object_name |
  TABLESPACE object_name |
  TEXT SEARCH CONFIGURATION object_name |
  TEXT SEARCH DICTIONARY object_name |
  TEXT SEARCH PARSER object_name |
  TEXT SEARCH TEMPLATE object_name |
  TYPE object_name |
  VIEW object_name |
  TRIGGER trigger_name ON table_name
}
   IS 'text';
```

参数说明

- **agg_name**

  聚集函数的名称

- **agg_type**

  聚集函数参数的类型

- **source_type**

  类型转换的源数据类型。

- **target_type**
  类型转换的目标数据类型。
- **object_name**
  对象名。
- **table_name.column_name**
  view_name.column_name
  列名称。前缀可加表名称或者视图名称。
- **constraint_name**
  表约束的名称。
- **table_name**
  表的名称。
- **function_name**
  函数名称。
- **argmode,argname,argtype**
  函数参数的模式、名称、类型。
- **large_object_oid**
  大对象的OID。
- **operator_name**
  操作符名称。
- **left_type,right_type**
  操作参数的数据类型（可以用模式修饰）。当前置或者后置操作符不存在时，可以增加NONE选项。
- **text**
  注释。

示例

```
--创建SCHEMA。
openGauss=# CREATE SCHEMA tpcds;

--创建表tpcds.customer。
openGauss=# CREATE TABLE tpcds.customer
(
c_customer_sk         INTEGER        NOT NULL,
c_customer_id         CHAR(16)       NOT NULL
);

--向表中插入多条记录。
openGauss=# INSERT INTO tpcds.customer VALUES (50, 'AAAAAAAABAAAAAAA'),(100,
'AAAAAAAACAAAAAAA'),(150, 'AAAAAAAADAAAAAAA');

--创建表tpcds.customer_demographics_t2。
openGauss=# CREATE TABLE tpcds.customer_demographics_t2
(
    CD_DEMO_SK               INTEGER               NOT NULL,
    CD_GENDER                CHAR(1)                     ,
    CD_MARITAL_STATUS        CHAR(1)                     ,
    CD_EDUCATION_STATUS      CHAR(20)                    ,
    CD_PURCHASE_ESTIMATE     INTEGER                     ,
    CD_CREDIT_RATING         CHAR(10)                    ,
    CD_DEP_COUNT             INTEGER                     ,
    CD_DEP_EMPLOYED_COUNT    INTEGER                     ,
```



```
    CD_DEP_COLLEGE_COUNT      INTEGER
)
DISTRIBUTE BY HASH (CD_DEMO_SK);

-- 为tpcds.customer_demographics_t2.cd_demo_sk列加注释。
openGauss=# COMMENT ON COLUMN tpcds.customer_demographics_t2.cd_demo_sk IS 'Primary key of
customer demographics table.';

--创建一个由c_customer_sk小于150的内容组成的视图。
openGauss=# CREATE VIEW tpcds.customer_details_view_v2 AS
    SELECT *
    FROM tpcds.customer
    WHERE c_customer_sk < 150;

-- 为tpcds.customer_details_view_v2视图加注释。
openGauss=# COMMENT ON VIEW tpcds.customer_details_view_v2 IS 'View of customer detail';

-- 删除view。
openGauss=# DROP VIEW tpcds.customer_details_view_v2;

-- 删除tpcds.customer_demographics_t2。
openGauss=# DROP TABLE tpcds.customer_demographics_t2;

--删除表tpcds.customer。
openGauss=# DROP TABLE tpcds.customer;

--删除SCHEMA。
openGauss=# DROP SCHEMA tpcds CASCADE;
```

7.13.40 COMMIT | END

功能描述

通过COMMIT或者END可完成提交事务的功能，即提交事务的所有操作。

注意事项

执行COMMIT这个命令的时候，命令执行者必须是该事务的创建者或系统管理员，且创建和提交操作可以不在同一个会话中。

语法格式

```
{ COMMIT | END } [ WORK | TRANSACTION ] ;
```

参数说明

- **COMMIT | END**

  提交当前事务，让所有当前事务的更改为其他事务可见。

- **WORK | TRANSACTION**

  可选关键字，除了增加可读性没有其他任何作用。

示例

```
--创建SCHEMA。
openGauss=# CREATE SCHEMA tpcds;

--创建表。
openGauss=# CREATE TABLE tpcds.customer_demographics_t2
(
    CD_DEMO_SK              INTEGER              NOT NULL,
    CD_GENDER               CHAR(1)                      ,
```



```
    CD_MARITAL_STATUS         CHAR(1)                   ,
    CD_EDUCATION_STATUS       CHAR(20)                  ,
    CD_PURCHASE_ESTIMATE      INTEGER                   ,
    CD_CREDIT_RATING          CHAR(10)                  ,
    CD_DEP_COUNT              INTEGER                   ,
    CD_DEP_EMPLOYED_COUNT     INTEGER                   ,
    CD_DEP_COLLEGE_COUNT      INTEGER
)
DISTRIBUTE BY HASH (CD_DEMO_SK);

--开启事务。
openGauss=# START TRANSACTION;

--插入数据。
openGauss=# INSERT INTO tpcds.customer_demographics_t2 VALUES(1,'M', 'U', 'DOCTOR DEGREE', 1200,
'GOOD', 1, 0, 0);
openGauss=# INSERT INTO tpcds.customer_demographics_t2 VALUES(2,'F', 'U', 'MASTER DEGREE', 300,
'BAD', 1, 0, 0);

--提交事务，让所有更改永久化。
openGauss=# COMMIT;

--查询数据。
openGauss=# SELECT * FROM tpcds.customer_demographics_t2;

--删除表tpcds.customer_demographics_t2。
openGauss=# DROP TABLE tpcds.customer_demographics_t2;

--删除SCHEMA。
openGauss=# DROP SCHEMA tpcds CASCADE;
```

相关链接

ROLLBACK

7.13.41 COMMIT PREPARED

功能描述

提交一个早先为两阶段提交准备好的事务。

注意事项

- 该功能仅在维护模式(GUC参数xc_maintenance_mode为on时)下可用。该模式谨慎打开，一般供维护人员排查问题使用，一般用户不应使用该模式。
- 命令执行者必须是该事务的创建者或系统管理员，且创建和提交操作可以不在同一个会话中。
- 事务功能由数据库自动维护，不应显式使用事务功能。

语法格式

```
COMMIT PREPARED transaction_id [ WITH commit_sequence_number ];
```

参数说明

- **transaction_id**
  待提交事务的标识符。它不能和任何当前预备事务已经使用了的标识符同名。
- **commit_sequence_number**
  待提交事务的序列号。它是一个64位递增无符号数。

示例

```
--开始。
openGauss=# begin;

--准备标识符为的trans_test的事务。
openGauss=# PREPARE TRANSACTION 'trans_test';

--创建表。
openGauss=# CREATE TABLE item1(id int);

--提交标识符为的trans_test的事务。
openGauss=# COMMIT PREPARED 'trans_test';

--删除表。
openGauss=# DROP TABLE item1;
```

相关链接

PREPARE TRANSACTION，**ROLLBACK PREPARED**。

7.13.42 COPY

功能描述

通过COPY命令实现在表和文件之间拷贝数据。

COPY FROM从一个文件拷贝数据到一个表，COPY TO把一个表的数据拷贝到一个文件。

注意事项

- 当参数enable_copy_server_files关闭时，只允许初始用户执行COPY FROM FILENAME或COPY TO FILENAME命令，当参数enable_copy_server_files打开时，允许具有SYSADMIN权限的用户或继承了内置角色gs_role_copy_files权限的用户执行，但默认禁止对数据库配置文件，密钥文件，证书文件和审计日志执行COPY FROM FILENAME或COPY TO FILENAME，以防止用户越权查看或修改敏感文件。同时enable_copy_server_files打开时，管理员可以通过guc参数safe_data_path设置普通用户可以导入导出的路径必须为设置路径的子路径，未设置此guc参数时候（默认情况），不对普通用户使用的路径进行拦截。
- COPY只能用于表，不能用于视图。
- COPY TO需要读取的表的select权限，COPY FROM需要插入的表的INSERT权限。
- 如果声明了一个字段列表，COPY将只在文件和表之间拷贝已声明字段的数据。如果表中有任何不在字段列表里的字段，COPY FROM将为那些字段插入缺省值。
- 如果声明了数据源文件，服务器必须可以访问该文件；如果指定了STDIN，数据将在客户前端和服务器之间流动，输入时，表的列与列之间使用TAB键分隔，在新的一行中以反斜杠和句点（\.）表示输入结束。
- 如果数据文件的任意行包含比预期多或者少的字段，COPY FROM将抛出一个错误。
- 数据的结束可以用一个只包含反斜杠和句点（\.）的行表示。如果从文件中读取数据，数据结束的标记是不必要的；如果在客户端应用之间拷贝数据，必须要有结束标记。
- COPY FROM中\N为空字符串，如果要输入实际数据值\N ，使用\\N。

- COPY FROM 支持通过列表达式对数据做预处理，但是列表达式中不支持子查询这类能力。
- COPY FROM在遇到数据格式错误时会回滚事务，但没有足够的错误信息，不方便用户从大量的原始数据中定位错误数据。
- COPY FROM/TO适合低并发，本地小数据量导入导出。
- 在COPY TO导出的过程中，如果表内字段数据存在“\0”字符，则字段数据在导出时会发生截断，字段中只有“\0”之前的数据会被导出。

语法格式

- 从一个文件拷贝数据到一个表。

```
COPY table_name [ ( column_name [, ...] ) ]
    FROM { 'filename' | STDIN }
    [ [ USING ] DELIMITERS 'delimiters' ]
    [ WITHOUT ESCAPING ]
    [ LOG ERRORS ]
    [ REJECT LIMIT 'limit' ]
    [ [ WITH ] ( option [, ...] ) ]
    | copy_option
    | TRANSFORM  ( { column_name [ data_type ] [ AS transform_expr ] } [, ...] )
    | FIXED FORMATTER ( { column_name( offset, length ) } [, ...] ) [ ( option [, ...] ) | copy_option
[  ...] ] ];
```

  说明

  语法中的FIXED FORMATTER ({ column_name(offset, length) } [, ...])以及[copy_option [...]] 的无冲突项可以任意排列组合。

- 把一个表的数据拷贝到一个文件。

```
COPY table_name [ ( column_name [, ...] ) ]
    TO { 'filename' | STDOUT }
    [ [ USING ] DELIMITERS 'delimiters' ]
    [ WITHOUT ESCAPING ]
    [ [ WITH ] ( option [, ...] ) ]
    | copy_option
    | FIXED FORMATTER ( { column_name( offset, length ) } [, ...] ) [ ( option [, ...] ) | copy_option
[  ...] ] ];

COPY query {(SELECT) | (VALUES)}
    TO { 'filename' | STDOUT }
    [ WITHOUT ESCAPING ]
    [ [ WITH ] ( option [, ...] ) ]
    | copy_option
    | FIXED FORMATTER ( { column_name( offset, length ) } [, ...] ) [ ( option [, ...] ) | copy_option
[  ...] ] ];
```

  说明

  1. COPY TO语法形式约束如下：

     (query)与[USING] DELIMITERS不兼容，即若COPY TO的数据来自于一个query的查询结果，那么COPY TO语法不能再指定[USING] DELIMITERS语法子句。
  2. 对于FIXED FORMATTER语法后面跟随的copy_option是以空格进行分隔的。
  3. copy_option是指COPY原生的参数形式，而option是兼容外表导入的参数形式。
  4. 语法中的FIXED FORMATTER ({ column_name(offset, length) } [, ...])以及[copy_option [...]] 的无冲突项可以任意排列组合。

  其中可选参数option子句语法为：

```
FORMAT 'format_name'
| OIDS [ boolean ]
| DELIMITER 'delimiter_character'
| NULL 'null_string'
| HEADER [ boolean ]
| USEEOF [ boolean ]
```



```
| FILEHEADER 'header_file_string'
| FREEZE [ boolean ]
| QUOTE 'quote_character'
| ESCAPE 'escape_character'
| EOL 'newline_character'
| NOESCAPING [ boolean ]
| FORCE_QUOTE { ( column_name [, ...] ) | * }
| FORCE_NOT_NULL ( column_name [, ...] )
| ENCODING 'encoding_name'
| IGNORE_EXTRA_DATA [ boolean ]
| FILL_MISSING_FIELDS [ boolean ]
| COMPATIBLE_ILLEGAL_CHARS [ boolean ]
| DATE_FORMAT 'date_format_string'
| TIME_FORMAT 'time_format_string'
| TIMESTAMP_FORMAT 'timestamp_format_string'
| SMALLDATETIME_FORMAT 'smalldatetime_format_string'
```

其中可选参数copy_option子句语法为：

```
OIDS
| NULL 'null_string'
| HEADER
| USEEOF
| FILEHEADER 'header_file_string'
| FREEZE
| FORCE_NOT_NULL column_name [, ...]
| FORCE_QUOTE { column_name [, ...] | * }
| BINARY
| CSV
| QUOTE [ AS ] 'quote_character'
| ESCAPE [ AS ] 'escape_character'
| EOL 'newline_character'
| ENCODING 'encoding_name'
| IGNORE_EXTRA_DATA
| FILL_MISSING_FIELDS
| COMPATIBLE_ILLEGAL_CHARS
| DATE_FORMAT 'date_format_string'
| TIME_FORMAT 'time_format_string'
| TIMESTAMP_FORMAT 'timestamp_format_string'
| SMALLDATETIME_FORMAT 'smalldatetime_format_string'
```

参数说明

- **query**

  其结果将被拷贝。

  取值范围：仅支持一个SELECT或VALUES命令，命令结尾不需要分号。

- **table_name**

  表的名称（可以有模式修饰）。

  取值范围：已存在的表名。

- **column_name**

  可选的待拷贝字段列表。

  取值范围：如果没有声明字段列表，将使用所有字段。

- **STDIN**

  声明输入是来自标准输入。

- **STDOUT**

  声明输出打印到标准输出。

- **FIXED**

  打开字段固定长度模式。在字段固定长度模式下，不能声明DELIMITER，NULL，CSV选项。指定FIXED类型后，不能再通过option或copy_option指定BINARY、CSV、TEXT等类型。

说明

定长格式定义如下：

1. 每条记录的每个字段长度相同。
2. 长度不足的字段以空格填充，数字类型字段左对齐，字符字段右对齐。
3. 字段和字段之间没有分隔符。

- **[USING] DELIMITERS 'delimiters'**

  在文件中分隔各个字段的字符串，分隔符最大长度不超过10个字节。

  取值范围：不允许包含\.abcdefghijklmnopqrstuvwxyz0123456789中的任何一个字符。

  缺省值：在文本模式下，缺省是水平制表符，在CSV模式下是一个逗号。

- **WITHOUT ESCAPING**

  在TEXT格式中，不对'\'和后面的字符进行转义。

  取值范围：仅支持TEXT格式。

- **LOG ERRORS**

  若指定，则开启对于COPY FROM语句中数据类型错误的容错机制，相关错误行的错误记录会记录到此库中public.pgxc_copy_error_log表中，备后续查阅。

  取值范围：仅支持导入（即COPY FROM）时指定。

  说明

  此容错选项的使用限制如下：

  - 此容错机制仅捕捉COPY FROM过程中CN节点上数据解析过程中相关的数据类型错误（DATA_EXCEPTION），诸如CN与DN之间的网络交互错误或者是DN上的表达式转换错误等CN数据解析逻辑之外的过程无法涵盖在内。
  - 在每个库第一次使用时COPY FROM容错时，请先行检查public.pgxc_copy_error_log（COPY错误表）是否存在，若不存在请调用copy_error_log_create() 函数创建；若存在，请转移该表数据并删除这张表后，调用copy_error_log_create() 函数创建。更多关于表public.pgxc_copy_error_log的字段信息，请参见**表7-53**。
  - 若在指定了LOG ERRORS的COPY FROM运行时，public.pgxc_copy_error_log不存在（未创建或者已删除）或表定义不符合copy_error_log_create() 中的预设表定义，则会报错。因此请确定此COPY错误表是使用copy_error_log_create() 函数创建的，否则可能导致容错的COPY FROM语句无法正常执行。
  - COPY已有的容错选项（如IGNORE_EXTRA_DATA）开启时，对应类型的错误会按照已有的方式处理而不会报出异常，因此错误表也不会有相应数据。

- **LOG ERRORS DATA**

  LOG ERRORS DATA和LOG ERRORS的区别:

  a. LOG ERRORS DATA会填充容错表的rawrecord字段。

  b. 只有super权限的用户才能使用LOG ERRORS DATA参数选项。

  须知

  使用“LOG ERRORS DATA”时，若错误内容过于复杂可能存在写入容错表失败的风险，导致任务失败。

  对于以某种编码无法读起来的错误，对应ERRCODE_CHARACTER_NOT_IN_REPERTOIRE和ERRCODE_UNTRANSLATABLE_CHARACTER两种错误码，不记录rawrecord字段。

- **REJECT LIMIT 'limit'**

  与LOG ERROR选项共同使用，对COPY FROM的容错机制设置数值上限，一旦此COPY FROM语句错误数据超过选项指定条数，则会按照原有机制报错。

  取值范围：正整数（1-2147483647），'unlimited'（无最大值限制）

  缺省值：若未指定LOG ERRORS，则会报错；若指定LOG ERRORS，则默认为0。

  说明

  如上述LOG ERRORS中描述的容错机制，REJECT LIMIT的计数也是按照执行COPY FROM的CN上遇到的解析错误数量计算，而不是每个DN上的错误数量，这点请与GDS容错机制区别开。

- **FORMATTER**

  在固定长度模式中，定义每一个字段在数据文件中的位置。按照column(offset,length)格式定义每一列在数据文件中的位置。

  取值范围：

  - offset取值不能小于0，以字节为单位。
  - length取值不能小于0，以字节为单位。

  所有列的总长度和不能大于1GB。

  文件中没有出现的列默认以空值代替。

- **OPTION { option_name ' value ' }**

  用于指定兼容外表的各类参数。

  - FORMAT

    数据源文件的格式。

    取值范围：CSV、TEXT、FIXED、BINARY。

    - CSV格式的文件，可以有效处理数据列中的换行符，但对一些特殊字符处理有欠缺。
    - TEXT格式的文件，可以有效处理一些特殊字符，但无法正确处理数据列中的换行符。
    - FIXED格式的文件，适用于每条数据的数据列都比较固定的数据，长度不足的列会添加空格补齐，过长的列则会自动截断。
    - BINARY形式的选项会使得所有的数据被存储/读作二进制格式而不是文本。 这比TEXT和CSV格式的要快一些，但是一个BINARY格式文件可移植性比较差。

    缺省值：TEXT

  - OIDS

    为每行拷贝内部对象标识（oid）。

    说明

    若COPY FROM对象为query或者对于没有oid的表，指定oids标识报错。

    取值范围：true/on，false/off。

    缺省值：false

  - DELIMITER

    指定数据文件行数据的字段分隔符。

说明

- 分隔符不能是\r和\n。
- 分隔符不能和null参数相同，CSV格式数据的分隔符不能和quote参数相同。
- TEXT格式数据的分隔符不能包含： 小写字母、数字和特殊字符.\。
- 数据文件中单行数据长度需<1GB，如果分隔符较长且数据列较多的情况下，会影响导出有效数据的长度。
- 分隔符推荐使用多字符和不可见字符。多字符例如'$^&'；不可见字符例如0x07，0x08，0x1b等。
- 如使用TAB制表符进行CSV文件分隔，使用E'\t'。

取值范围：支持多字符分隔符，但分隔符不能超过10个字节。

缺省值：

- TEXT格式的默认分隔符是水平制表符（tab）。
- CSV格式的默认分隔符为“,”。
- FIXED格式没有分隔符。

– NULL

用来指定数据文件中空值的表示。

取值范围：

- null值不能是\r和\n，最大为100个字符。
- null值不能和分隔符、quote参数相同。

缺省值：

- CSV格式下默认值是一个没有引号的空字符串。
- 在TEXT格式下默认值是\N。

– HEADER

指定导出数据文件是否包含标题行，标题行一般用来描述表中每个字段的信息。header只能用于CSV，FIXED格式的文件中。

在导入数据时，如果header选项为on，则数据文件中第一行会被识别为标题行，会忽略此行。如果header为off，而数据文件中第一行会被识别为数据。

在导出数据时，如果header选项为on，则需要指定fileheader。如果header为off，则导出数据文件不包含标题行。

取值范围：true/on，false/off。

缺省值：false

– USEEOF

不对导入数据中的”\.”做报错处理。

取值范围：true/on，false/off。

缺省值：false

– QUOTE

CSV格式文件下的引号字符。

缺省值：双引号

说明

- quote参数不能和分隔符、null参数相同。
- quote参数只能是单字节的字符。
- 推荐不可见字符作为quote，例如0x07，0x08，0x1b等。

- ESCAPE

  CSV格式下，用来指定逃逸字符，逃逸字符只能指定为单字节字符。

  缺省值：双引号。当与quote值相同时，会被替换为'\0'。

- EOL 'newline_character'

  指定导入导出数据文件换行符样式。

  取值范围：支持多字符换行符，但换行符不能超过10个字节。常见的换行符，如\r、\n、\r\n（设成0x0D、0x0A、0x0D0A效果是相同的），其他字符或字符串，如$、#。

  说明

  - EOL参数只能用于TEXT格式的导入导出，不支持CSV格式和FIXED格式导入。为了兼容原有EOL参数，仍然支持导出CSV格式和FIXED格式时指定EOL参数为0x0D或0x0D0A。
  - EOL参数不能和分隔符、null参数相同。
  - EOL参数不能包含：.abcdefghijklmnopqrstuvwxyz0123456789。

- FORCE_QUOTE { (column_name [, ...]) | * }

  在CSV COPY TO模式下，强制在每个声明的字段周围对所有非NULL值都使用引号包围。NULL输出不会被引号包围。

  取值范围：已存在的字段。

- FORCE_NOT_NULL (column_name [, ...])

  在CSV COPY FROM模式下，指定的字段输入不能为空。

  取值范围：已存在的字段。

- ENCODING

  指定数据文件的编码格式名称，缺省为当前数据库编码格式。

- IGNORE_EXTRA_DATA

  若数据源文件比外表定义列数多，是否会忽略对多出的列。该参数只在数据导入过程中使用。

  取值范围：true/on、false/off。

  - 参数为true/on，若数据源文件比外表定义列数多，则忽略行尾多出来的列。
  - 参数为false/off，若数据源文件比外表定义列数多，会显示如下错误信息。

    ```
    extra data after last expected column
    ```

  缺省值：false。

须知

如果行尾换行符丢失，使两行变成一行时，设置此参数为true将导致后一行数据被忽略掉。

- COMPATIBLE_ILLEGAL_CHARS

  导入非法字符容错参数。此语法仅对COPY FROM导入有效。

  取值范围：true/on，false/off。

  - 参数为true/on，则导入时遇到非法字符进行容错处理，非法字符转换后入库，不报错，不中断导入。
  - 参数为false/off，导入时遇到非法字符进行报错，中断导入。

  缺省值：false/off

  说明

  导入非法字符容错规则如下：

  （1）对于'\0'，容错后转换为空格；

  （2）对于其他非法字符，容错后转换为问号；

  （3）若compatible_illegal_chars为true/on标识导入时对于非法字符进行容错处理，则若NULL、DELIMITER、QUOTE、ESCAPE设置为空格或问号则会通过如"illegal chars conversion may confuse COPY escape 0x20"等报错信息提示用户修改可能引起混淆的参数以避免导入错误。

- FILL_MISSING_FIELDS

  当数据加载时，若数据源文件中一行的最后一个字段缺失的处理方式。

  取值范围：true/on，false/off。

  缺省值：false/off

- DATE_FORMAT

  导入对于DATE类型指定格式。此参数不支持BINARY格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对COPY FROM导入有效。

  取值范围：合法DATE格式。可参考**时间和日期处理函数和操作符**。

  说明

  对于指定为ORACLE兼容类型的数据库，则DATE类型内建为TIMESTAMP类型。在导入的时候，若需指定格式，可以参考下面的timestamp_format参数。

- TIME_FORMAT

  导入对于TIME类型指定格式。此参数不支持BINARY格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对COPY FROM导入有效。

  取值范围：合法TIME格式，不支持时区。可参考**时间和日期处理函数和操作符**。

- TIMESTAMP_FORMAT

  导入对于TIMESTAMP类型指定格式。此参数不支持BINARY格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对COPY FROM导入有效。

  取值范围：合法TIMESTAMP格式，不支持时区。可参考**时间和日期处理函数和操作符**。

- SMALLDATETIME_FORMAT

  导入对于SMALLDATETIME类型指定格式。此参数不支持BINARY格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对COPY FROM导入有效。

取值范围：合法SMALLDATETIME格式。可参考**时间和日期处理函数和操作符**。

- **COPY_OPTION { option_name ' value ' }**

  用于指定COPY原生的各类参数。

  - OIDS

    为每行拷贝内部对象标识（oid）。

    说明

    若COPY FROM对象为query或者对于没有oid的表，指定oids标识报错。

  - NULL null_string

    用来指定数据文件中空值的表示。

    须知

    在使用COPY FROM的时候，任何匹配这个字符串的字符串将被存储为NULL值，所以应该确保指定的字符串和COPY TO相同。

    取值范围：

    - null值不能是\r和\n，最大为100个字符。
    - null值不能和分隔符、quote参数相同。

    缺省值：

    - 在TEXT格式下默认值是\N。
    - CSV格式下默认值是一个没有引号的空字符串。

  - HEADER

    指定导出数据文件是否包含标题行，标题行一般用来描述表中每个字段的信息。header只能用于CSV，FIXED格式的文件中。

    在导入数据时，如果header选项为on，则数据文件中第一行会被识别为标题行，会忽略此行。如果header为off，而数据文件中第一行会被识别为数据。

    在导出数据时，如果header选项为on，则需要指定fileheader。如果header为off，则导出数据文件不包含标题行。

  - USEEOF

    不对导入数据中的”\.”做报错处理。

  - FILEHEADER

    导出数据时用于定义标题行的文件，一般用来描述每一列的数据信息。

须知

- 仅在header为on或true的情况下有效。
- fileheader指定的是绝对路径。
- 该文件只能包含一行标题信息，并以换行符结尾，多余的行将被丢弃（标题信息不能包含换行符）。
- 该文件包括换行符在内长度不超过1M。

- FREEZE

  将COPY加载的数据行设置为已经被frozen，就像这些数据行执行过VACUUM FREEZE。

  这是一个初始数据加载的性能选项。仅当以下三个条件同时满足时，数据行会被frozen：

  - 在同一事务中create或truncate这张表之后执行COPY。
  - 当前事务中没有打开的游标。
  - 当前事务中没有原有的快照。

  说明

  COPY完成后，所有其他会话将会立刻看到这些数据。但是这违反了MVCC可见性的一般原则，用户应当了解这样会导致潜在的风险。

- FORCE NOT NULL column_name [, ...]

  在CSV COPY FROM模式下，指定的字段输入不能为空。

  取值范围：已存在的字段。

- FORCE QUOTE { column_name [, ...] | * }

  在CSV COPY TO模式下，强制在每个声明的字段周围对所有非NULL值都使用引号包围。NULL输出不会被引号包围。

  取值范围：已存在的字段。

- BINARY

  使用二进制格式存储和读取，而不是以文本的方式。在二进制模式下，不能声明DELIMITER，NULL，CSV选项。指定BINARY类型后，不能再通过option或copy_option指定CSV、FIXED、TEXT等类型。

- CSV

  打开逗号分隔变量（CSV）模式。指定CSV类型后，不能再通过option或copy_option指定BINARY、FIXED、TEXT等类型。

- QUOTE [AS] 'quote_character'

  CSV格式文件下的引号字符。

  缺省值：双引号。

  说明

  - quote参数不能和分隔符、null参数相同。
  - quote参数只能是单字节的字符。
  - 推荐不可见字符作为quote，例如0x07，0x08，0x1b等。

- ESCAPE [AS] 'escape_character'

CSV格式下，用来指定逃逸字符，逃逸字符只能指定为单字节字符。

默认值为双引号。当与quote值相同时，会被替换为'\0'。

- EOL 'newline_character'

  指定导入导出数据文件换行符样式。

  取值范围：支持多字符换行符，但换行符不能超过10个字节。常见的换行符，如\r、\n、\r\n（设成0x0D、0x0A、0x0D0A效果是相同的），其他字符或字符串，如$、#。

  说明

  - EOL参数只能用于TEXT格式的导入导出，不支持CSV格式和FIXED格式。为了兼容原有EOL参数，仍然支持导出CSV格式和FIXED格式时指定EOL参数为0x0D或0x0D0A。
  - EOL参数不能和分隔符、null参数相同。
  - EOL参数不能包含：.abcdefghijklmnopqrstuvwxyz0123456789。

- ENCODING 'encoding_name'

  指定文件编码格式名称。

  取值范围：有效的编码格式。

  缺省值：当前编码格式。

- IGNORE_EXTRA_DATA

  指定当数据源文件比外表定义列数多时，忽略行尾多出来的列。该参数只在数据导入过程中使用。

  若不使用该参数，在数据源文件比外表定义列数多，会显示如下错误信息。

  ```
  extra data after last expected column
  ```

- COMPATIBLE_ILLEGAL_CHARS

  指定导入时对非法字符进行容错处理，非法字符转换后入库。不报错，不中断导入。此参数不支持BINARY格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对COPY FROM导入有效。

  若不使用该参数，导入时遇到非法字符进行报错，中断导入。

  说明

  导入非法字符容错规则如下：

  （1）对于'\0'，容错后转换为空格；

  （2）对于其他非法字符，容错后转换为问号；

  （3）若compatible_illegal_chars为true/on标识，导入时对于非法字符进行容错处理，则若NULL、DELIMITER、QUOTE、ESCAPE设置为空格或问号则会通过如"illegal chars conversion may confuse COPY escape 0x20"等报错信息提示用户修改可能引起混淆的参数以避免导入错误。

- FILL_MISSING_FIELDS

  当数据加载时，若数据源文件中一行的最后一个字段缺失的处理方式。

  取值范围：true/on，false/off。

  缺省值：false/off。

须知

目前COPY指定此Option实际不会生效，即不会有相应的容错处理效果（不生效）。需要额外注意的是，打开此选项会导致解析器在CN数据解析阶段（即COPY错误表容错的涵盖范围）忽略此数据问题，而到DN重新报错，从而使得COPY错误表（打开LOG ERRORS REJECT LIMIT）在此选项打开的情况下无法成功捕获这类少列的数据异常。因此请不要指定此选项。

- DATE_FORMAT 'date_format_string'

  导入对于DATE类型指定格式。此参数不支持BINARY格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对COPY FROM导入有效。

  取值范围：合法DATE格式。可参考**时间和日期处理函数和操作符**

  说明

  对于指定为ORACLE兼容类型的数据库，则DATE类型内建为TIMESTAMP类型。在导入的时候，若需指定格式，可以参考下面的timestamp_format参数。

- TIME_FORMAT 'time_format_string'

  导入对于TIME类型指定格式。此参数不支持BINARY格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对COPY FROM导入有效。

  取值范围：合法TIME格式，不支持时区。可参考**时间和日期处理函数和操作符**。

- TIMESTAMP_FORMAT 'timestamp_format_string'

  导入对于TIMESTAMP类型指定格式。此参数不支持BINARY格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对COPY FROM导入有效。

  取值范围：合法TIMESTAMP格式，不支持时区。可参考**时间和日期处理函数和操作符**。

- SMALLDATETIME_FORMAT 'smalldatetime_format_string'

  导入对于SMALLDATETIME类型指定格式。此参数不支持BINARY格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对COPY FROM导入有效。

  取值范围：合法SMALLDATETIME格式。可参考**时间和日期处理函数和操作符**。

- TRANSFORM ({ column_name [data_type] [AS transform_expr] } [, ...])

  指定表中各个列的转换表达式；其中data_type指定该列在表达式参数中的数据类型；transform_expr为目标表达式，返回与表中目标列数据类型一致的结果值，表达式可参考**表达式**。

  说明

  COPY FROM目前不支持对分布列指定表达式转换。

COPY FROM能够识别的特殊反斜杠序列如下所示。

- **\b**：反斜杠 （ASCII 8）
- **\f**：换页（ASCII 12）

- **\n**：换行符（ASCII 10）
- **\r**：回车符（ASCII 13）
- **\t**：水平制表符（ASCII 9）
- **\v**：垂直制表符（ASCII 11）
- **\digits**：反斜杠后面跟着一到三个八进制数，表示ASCII值为该数的字符。
- **\xdigits**：反斜杠x后面跟着一个或两个十六进制位声明指定数值编码的字符。

权限控制示例

```
openGauss=> copy t1 from '/home/xy/t1.csv';
ERROR:  COPY to or from a file is prohibited for security concerns
HINT:  Anyone can COPY to stdout or from stdin. gsql's \copy command also works for anyone.
openGauss=> grant gs_role_copy_files to xxx;
```

此错误为非初始用户没有使用copy的权限示例，解决方式为打开enable_copy_server_files参数，则管理员可以使用copy功能，普通用户需要在此基础上加入gs_role_copy_files群组。

示例

```
--创建SCHEMA。
openGauss=# CREATE SCHEMA tpcds;

--创建tpcds.ship_mode表。
openGauss=# CREATE TABLE tpcds.ship_mode
(
    SM_SHIP_MODE_SK           INTEGER               NOT NULL,
    SM_SHIP_MODE_ID           CHAR(16)              NOT NULL,
    SM_TYPE                   CHAR(30)                      ,
    SM_CODE                   CHAR(10)                      ,
    SM_CARRIER                CHAR(20)                      ,
    SM_CONTRACT               CHAR(20)
)
DISTRIBUTE BY HASH(SM_SHIP_MODE_SK);

--向tpcds.ship_mode表插入一条数据。
openGauss=# INSERT INTO tpcds.ship_mode VALUES (1,'a','b','c','d','e');

--将tpcds.ship_mode中的数据复制到/home/omm/ds_ship_mode.dat文件中。
openGauss=# COPY tpcds.ship_mode TO '/home/omm/ds_ship_mode.dat';

--将tpcds.ship_mode 输出到STDOUT。
openGauss=# COPY tpcds.ship_mode TO STDOUT;

--将tpcds.ship_mode 的数据输出到STDOUT，使用参数如下：分隔符为','(delimiter ',')，编码格式为UTF8(encoding 'utf8')。
openGauss=# COPY tpcds.ship_mode TO STDOUT WITH (delimiter ',', encoding 'utf8');

--将tpcds.ship_mode 的数据输出到STDOUT，使用参数如下：导入格式为CSV（format 'CSV'），引号包围SM_SHIP_MODE_SK字段的导出内容(force_quote(SM_SHIP_MODE_SK))。
openGauss=# COPY tpcds.ship_mode TO STDOUT WITH (format 'CSV', force_quote(SM_SHIP_MODE_SK));

--创建tpcds.ship_mode_t1表。
openGauss=# CREATE TABLE tpcds.ship_mode_t1
(
    SM_SHIP_MODE_SK           INTEGER               NOT NULL,
    SM_SHIP_MODE_ID           CHAR(16)              NOT NULL,
    SM_TYPE                   CHAR(30)                      ,
    SM_CODE                   CHAR(10)                      ,
    SM_CARRIER                CHAR(20)                      ,
    SM_CONTRACT               CHAR(20)
)
DISTRIBUTE BY HASH(SM_SHIP_MODE_SK );
```



```
--从STDIN复制数据到表tpcds.ship_mode_t1。
openGauss=# COPY tpcds.ship_mode_t1 FROM STDIN;

--从/home/omm/ds_ship_mode.dat文件复制数据到表tpcds.ship_mode_t1。
openGauss=# COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat';

--从/home/omm/ds_ship_mode.dat文件复制数据到表tpcds.ship_mode_t1，应用TRANSFORM表达式转换，取SM_TYPE列左边10个字符插入到表中。
openGauss=# COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat' TRANSFORM (SM_TYPE AS LEFT(SM_TYPE, 10));

--从/home/omm/ds_ship_mode.dat文件复制数据到表tpcds.ship_mode_t1，使用参数如下：导入格式为TEXT（format 'text'），分隔符为'\t'（delimiter E'\t'），忽略多余列（ignore_extra_data 'true'），不指定转义（noescaping 'true'）。
openGauss=# COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat' WITH(format 'text', delimiter E'\t', ignore_extra_data 'true', noescaping 'true');

--从/home/omm/ds_ship_mode.dat文件复制数据到表tpcds.ship_mode_t1，使用参数如下：导入格式为FIXED（FIXED），指定定长格式（FORMATTER(SM_SHIP_MODE_SK(0, 2), SM_SHIP_MODE_ID(2,16), SM_TYPE(18,30), SM_CODE(50,10), SM_CARRIER(61,20), SM_CONTRACT(82,20)）），忽略多余列（ignore_extra_data），有数据头（header）。
openGauss=# COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat' FIXED FORMATTER(SM_SHIP_MODE_SK(0, 2), SM_SHIP_MODE_ID(2,16), SM_TYPE(18,30), SM_CODE(50,10), SM_CARRIER(61,20), SM_CONTRACT(82,20)) header ignore_extra_data;

--删除表和SCHEMA。
openGauss=# DROP TABLE tpcds.ship_mode;
openGauss=# DROP TABLE tpcds.ship_mode_t1;
openGauss=# DROP SCHEMA tpcds;
```

7.13.43 CREATE AUDIT POLICY

功能描述

创建统一审计策略。

注意事项

只有poladmin，sysadmin或初始用户能进行此操作。

需要开启安全策略开关，即设置GUC参数enable_security_policy=on，脱敏策略才可以生效。

语法格式

```
CREATE AUDIT POLICY [ IF NOT EXISTS ] policy_name { privilege_audit_clause | access_audit_clause } [, ... ] [ filter_group_clause ] [ ENABLE | DISABLE ];
```

- privilege_audit_clause：
  ```
  PRIVILEGES { DDL | ALL } [ ON LABEL ( resource_label_name [, ... ] ) ]
  ```
- access_audit_clause：
  ```
  ACCESS { DML | ALL } [ ON LABEL ( resource_label_name [, ... ] ) ]
  ```
- filter_group_clause：
  ```
  FILTER ON { FILTER_TYPE ( filter_value [, ... ] ) } [, ... ]
  ```

参数说明

- **policy_name**

  审计策略名称，需要唯一，不可重复；

  取值范围：字符串，要符合**标识符命名规范**。

- **resource_label_name**

  资源标签名称。

- **DDL**

  指的是针对数据库执行如下操作时进行审计，目前支持：CREATE、ALTER、DROP、ANALYZE、COMMENT、GRANT、REVOKE、SET、SHOW、LOGIN_ANY、LOGIN_FAILURE、LOGIN_SUCCESS、LOGOUT。

- **DML**

  指的是针对数据库执行如下操作时进行审计，目前支持：SELECT、COPY、DEALLOCATE、DELETE、EXECUTE、INSERT、PREPARE、REINDEX、TRUNCATE、UPDATE。

- **ALL**

  指的是上述DDL或DML中支持的所有对数据库的操作。当形式为{ DDL | ALL }时，ALL指所有DDL操作；当形式为{ DML | ALL }时，ALL指所有DML操作。

- **FILTER_TYPE**

  描述策略过滤的条件类型，包括APP、ROLES、IP。

- **filter_value**

  指具体过滤信息内容。

- **ENABLE|DISABLE**

  可以打开或关闭统一审计策略。若不指定ENABLE|DISABLE，语句默认为ENABLE。

示例

```
--创建dev_audit和bob_audit用户。
openGauss=# CREATE USER dev_audit PASSWORD 'xxxxxxxxxx';
openGauss=# CREATE USER bob_audit password 'xxxxxxxxxx';

--创建一个表tb_for_audit。
openGauss=# CREATE TABLE tb_for_audit(col1 text, col2 text, col3 text);

--创建资源标签。
openGauss=# CREATE RESOURCE LABEL adt_lb0 ADD TABLE(tb_for_audit);

--对数据库执行create操作创建审计策略。
openGauss=# CREATE AUDIT POLICY adt1 PRIVILEGES CREATE;

--对数据库执行select操作创建审计策略。
openGauss=# CREATE AUDIT POLICY adt2 ACCESS SELECT;

--仅审计记录用户dev_audit和bob_audit在执行针对adt_lb0资源进行的create操作数据库创建审计策略。
openGauss=# CREATE AUDIT POLICY adt3 PRIVILEGES CREATE ON LABEL(adt_lb0) FILTER ON
ROLES(dev_audit, bob_audit);

--仅审计记录用户dev_audit和bob_audit,客户端工具为gsql，IP地址为'10.20.30.40', '127.0.0.0/24'，在执行针对adt_lb0资源进行的select、insert、delete操作数据库创建审计策略。
openGauss=# CREATE AUDIT POLICY adt4 ACCESS SELECT ON LABEL(adt_lb0), INSERT ON LABEL(adt_lb0),
DELETE FILTER ON ROLES(dev_audit, bob_audit), APP(gsql), IP('10.20.30.40', '127.0.0.0/24');

--删除审计策略。
openGauss=# DROP AUDIT POLICY adt1, adt2, adt3, adt4;

--删除资源标签。
openGauss=# DROP RESOURCE LABEL adt_lb0;

--删除表tb_for_audit。
openGauss=# DROP TABLE tb_for_audit;
```



```
--删除dev_audit和bob_audit用户。
openGauss=# DROP USER dev_audit, bob_audit;
```

相关链接

ALTER AUDIT POLICY，**DROP AUDIT POLICY**。

7.13.44 CREATE BARRIER

功能描述

创建一个新集群节点间的同步点。该同步点可用于数据恢复。

注意事项

CREATE BARRIER通常只用于备份恢复操作，因此只允许在以下两种场景下执行。

- 数据库初始化用户可执行。
- 在CN节点上开启备份恢复模式，即GUC参数operation_mode=on的前提下，具备OPRADMIN权限的用户可执行。

语法格式

```
CREATE BARRIER [ barrier_name  ] ;
```

参数说明

- **barrier_name**

  可选参数。同步点名称。

  取值范围：字符串，要符合**标识符命名规范**。

示例

```
--创建一个barrier，不指定名称。
openGauss=# CREATE BARRIER;

--指定barrier名称。
openGauss=# CREATE BARRIER 'barrier1';
```

7.13.45 CREATE CONVERSION

功能描述

定义一种两个字符集编码之间的新转换。

注意事项

- 参数DEFAULT将在客户端和服务器之间默认执行源编码到目标编码之间的转换。要支持这个用法，需要定义双向转换，即从A到B和从B到A之间的转换。
- 创建转换需拥有函数的EXECUTE权限及目标模式的CREATE权限。
- 源编码和目标编码都不可以使用SQL_ASCII，因为在涉及SQL_ASCII “encoding”的情况下，服务器的行为是硬连接的。
- 使用DROP CONVERSION可以移除用户定义的转换。

语法格式

```
CREATE [ DEFAULT ] CONVERSION name
    FOR source_encoding TO dest_encoding FROM function_name
```

参数说明

- **DEFAULT**

  DEFAULT子句表示这个转换是从源编码到目标编码的默认转换。在一个模式中对于每一个编码对，只应该有一个默认转换。

- **name**

  转换的名称，可以被模式限定。如果没有被模式限定，该转换被定义在当前模式中。在一个模式中，转换名称必须唯一。

- **source_encoding**

  源编码名称。

- **dest_encoding**

  目标编码名称。

- **function_name**

  被用来执行转换的函数。函数名可以被模式限定。如果没有，将在路径中查找该函数。

  ```
  conv_proc(
      integer, -- 原编码ID
      integer, -- 目标编码ID
      cstring, -- 源字符串（空值终止的C字符串）
      internal,-- 目标（用一个空值终止的C字符串填充）
      integer  -- 源字符串长度
  ) RETURNS void;
  ```

示例

```
--使用myfunc函数创建一个编码UTF8到LATIN1的转换。
CREATE CONVERSION myconv FOR 'UTF8' TO 'LATIN1' FROM myfunc;
```

7.13.46 CREATE DATABASE

功能描述

创建一个新的数据库。缺省情况下新数据库将通过复制标准系统数据库template0来创建，且仅支持使用template0来创建。

注意事项

- 只有拥有CREATEDB权限的用户才可以创建新数据库，系统管理员默认拥有此权限。
- 不能在事务块中执行创建数据库语句。
- 在创建数据库过程中，若出现类似“could not initialize database directory”的错误提示，可能是由于文件系统上数据目录的权限不足或磁盘满等原因引起。

语法格式

```
CREATE DATABASE database_name
    [ [ WITH ] { [ OWNER [=] user_name ] |
                 [ TEMPLATE [=] template ] |
                 [ ENCODING [=] 'encoding' ] |
```



```
    [ LC_COLLATE [=] 'lc_collate' ] |
    [ LC_CTYPE [=] 'lc_ctype' ] |
    [ DBCOMPATIBILITY [=] 'compatibilty_type' ] |
    [ TABLESPACE [=] tablespace_name ] |
    [ CONNECTION LIMIT [=] connlimit ]}[...] ];
```

参数说明

- **database_name**

  数据库名称。

  取值范围：字符串，要符合**标识符命名规范**。

- **OWNER [=] user_name**

  数据库所有者。缺省时，新数据库的所有者是当前用户。

  取值范围：已存在的用户名。

- **TEMPLATE [=] template**

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

  取值范围：仅template0。

- **ENCODING [=] 'encoding'**

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

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

  常用取值：GBK、UTF8、Latin1、GB18030等，具体支持的字符集如下。

  表 7-110 支持的字符集

名称	描述	语言	是否服务器端?	ICU（International Components for Unicode）?	字节/字符	别名
BIG5	Big Five	繁体中文	否	否	1-2	WIN950, Windows950
EUC_CN	扩展UNIX编码-中国	简体中文	是	是	1-3	-
EUC_JP	扩展UNIX编码-日本	日文	是	是	1-3	-

名称	描述	语言	是否服务器端?	ICU（International Components for Unicode）?	字节/字符	别名
EUC_JIS_2004	扩展UNIX编码-日本, JIS X 0213	日文	是	否	1-3	-
EUC_KR	扩展UNIX编码-韩国	韩文	是	是	1-3	-
EUC_TW	扩展UNIX编码-中国台湾	繁体中文	是	是	1-3	-
GB18030	国家标准	中文	是	否	1-4	-
GBK	扩展国家标准	简体中文	是	否	1-2	WIN936, Windows936
ISO_8859_5	ISO 8859-5, ECMA 113	拉丁语/西里尔语	是	是	1	-
ISO_8859_6	ISO 8859-6, ECMA 114	拉丁语/阿拉伯语	是	是	1	-
ISO_8859_7	ISO 8859-7, ECMA 118	拉丁语/希腊语	是	是	1	-
ISO_8859_8	ISO 8859-8, ECMA 121	拉丁语/希伯来语	是	是	1	-
JOHAB	JOHAB	韩语	否	否	1-3	-
KOI8R	KOI8-R	西里尔语（俄语）	是	是	1	KOI8

名称	描述	语言	是否服务器端?	ICU（International Components for Unicode）?	字节/字符	别名
KOI8U	KOI8-U	西里尔语（乌克兰语）	是	是	1	-
LATIN1	ISO 8859-1, ECMA 94	西欧	是	是	1	ISO88591
LATIN2	ISO 8859-2, ECMA 94	中欧	是	是	1	ISO88592
LATIN3	ISO 8859-3, ECMA 94	南欧	是	是	1	ISO88593
LATIN4	ISO 8859-4, ECMA 94	北欧	是	是	1	ISO88594
LATIN5	ISO 8859-9, ECMA 128	土耳其语	是	是	1	ISO88599
LATIN6	ISO 8859-10, ECMA 144	日耳曼语	是	是	1	ISO885910
LATIN7	ISO 8859-13	波罗的海	是	是	1	ISO885913
LATIN8	ISO 8859-14	凯尔特语	是	是	1	ISO885914
LATIN9	ISO 8859-15	带欧罗巴和口音的LATIN1	是	是	1	ISO885915

名称	描述	语言	是否服务器端?	ICU（International Components for Unicode）?	字节/字符	别名
LATIN10	ISO 8859-16, ASRO SR 14111	罗马尼亚语	是	否	1	ISO885916
MULE_INTERNAL	Mule内部编码	多语种编辑器	是	否	1-4	-
SJIS	Shift JIS	日语	否	否	1-2	Mskanji, ShiftJIS, WIN932, Windows932
SHIFT_JIS_2004	Shift JIS, JIS X 0213	日语	否	否	1-2	-
SQL_ASCII	未指定（见文本）	*任意*	是	否	1	-
UHC	统一韩语编码	韩语	否	否	1-2	WIN949, Windows949
UTF8	Unicode, 8-bit	*所有*	是	是	1-4	Unicode
WIN866	Windows CP866	西里尔语	是	是	1	ALT
WIN874	Windows CP874	泰语	是	否	1	-
WIN1250	Windows CP1250	中欧	是	是	1	-
WIN1251	Windows CP1251	西里尔语	是	是	1	WIN
WIN1252	Windows CP1252	西欧	是	是	1	-

名称	描述	语言	是否服务器端?	ICU（International Components for Unicode）?	字节/字符	别名
WIN1253	Windows CP1253	希腊语	是	是	1	-
WIN1254	Windows CP1254	土耳其语	是	是	1	-
WIN1255	Windows CP1255	希伯来语	是	是	1	-
WIN1256	Windows CP1256	阿拉伯语	是	是	1	-
WIN1257	Windows CP1257	波罗的海	是	是	1	-
WIN1258	Windows CP1258	越南语	是	是	1	ABC, TCVN, TCVN5712, VSCII

须知

需要注意并非所有的客户端API都支持上面列出的字符集。

SQL_ASCII设置与其他设置表现存下差异。如果服务器字符集是SQL_ASCII，服务器把字节值0-127根据 ASCII标准解释，而字节值128-255则当作无法解析的字符。如果设置为SQL_ASCII，就不会有编码转换。因此，这个设置基本不是用来声明所使用的指定编码， 因为这个声明会忽略编码。在大多数情况下，如果使用了任何非ASCII数据，那么请不要再使用SQL_ASCII进行设置，因为数据库无法转换或者校验非ASCII字符。

须知

- 指定新的数据库字符集编码必须与所选择的本地环境中（LC_COLLATE和LC_CTYPE）的设置兼容。
- 当指定的字符编码集为GBK时，部分中文生僻字无法直接作为对象名。这是因为GBK第二个字节的编码范围在0x40-0x7E之间时，字节编码与ASCII字符@A-Z[\]^_`a-z{|}重叠。其中@[\]^_'{|}是数据库中的操作符，直接作为对象名时，会语法报错。例如“侅”字，GBK16进制编码为0x8240，第二个字节为0x40，与ASCII“@”符号编码相同，因此无法直接作为对象名使用。如果确实要使用，可以在创建和访问对象时，通过增加双引号来规避这个问题。
- 若客户端编码为A，服务器端编码为B，则需要满足数据库中存在编码格式A与B的转换。数据库能够支持的所有的编码格式转换详见系统表PG_CONVERSION（若无法转换，则建议客户端编码与服务器端编码保持一致，客户端编码可通过GUC参数client_encoding修改）。

- **LC_COLLATE [=] 'lc_collate'**

  指定新数据库使用的字符集。例如，通过lc_collate = 'zh_CN.gbk'设定该参数。

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

  取值范围：操作系统支持的字符集。

- **LC_CTYPE [=] 'lc_ctype'**

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

  取值范围：操作系统支持的字符分类。

  说明

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

- **DBCOMPATIBILITY [=] 'compatibilty_type'**

  指定兼容的数据库的类型，默认兼容MySQL。

  取值范围：MYSQL、TD、ORA、PG。分别表示兼容MySQL、TD（Teradata）、Oracle和PostgreSQL。

  说明

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

- **TABLESPACE [=] tablespace_name**

  指定数据库对应的表空间。

  取值范围：已存在表空间名。

- **CONNECTION LIMIT [=] connlimit**

数据库可以接受的并发连接数。

须知

- 系统管理员不受此参数的限制。
- connlimit每个CN单独统计，集群整体的连接数 = connlimit * 当前正常CN节点个数。

取值范围：[-1, 2^31-1]的整数。默认值为-1，表示没有限制。

有关字符编码的一些限制：

- 若区域设置为C（或POSIX），则允许所有的编码类型，但是对于其他的区域设置，字符编码必须和区域设置相同。
- 编码和区域设置必须匹配模板数据库，除了将template0当作模板。 因为其他数据库可能会包含不匹配指定编码的数据，或者可能包含排序顺序受LC_COLLATE和LC_CTYPE影响的索引。复制这些数据会导致在新数据库中的索引失效。template0是不包含任何会受到影响的数据或者索引。

示例

```
--创建jim和tom用户。
openGauss=# CREATE USER jim PASSWORD '********';
openGauss=# CREATE USER tom PASSWORD '********';

--创建一个GBK编码的数据库music（本地环境的编码格式必须也为GBK）。
openGauss=# CREATE DATABASE music ENCODING 'GBK' template = template0;

--创建数据库music2，并指定所有者为jim。
openGauss=# CREATE DATABASE music2 OWNER jim;

--用模板template0创建数据库music3，并指定所有者为jim。
openGauss=# CREATE DATABASE music3 OWNER jim TEMPLATE template0;

--设置music数据库的连接数为10。
openGauss=# ALTER DATABASE music CONNECTION LIMIT= 10;

--将music名称改为music4。
openGauss=# ALTER DATABASE music RENAME TO music4;

--将数据库music2的所属者改为tom。
openGauss=# ALTER DATABASE music2 OWNER TO tom;

--删除数据库。
openGauss=# DROP DATABASE music2;
openGauss=# DROP DATABASE music3;
openGauss=# DROP DATABASE music4;

--删除jim和tom用户。
openGauss=# DROP USER jim;
openGauss=# DROP USER tom;

--创建兼容TD格式的数据库。
openGauss=# CREATE DATABASE td_compatible_db DBCOMPATIBILITY 'TD';

--创建兼容ORA格式的数据库。
openGauss=# CREATE DATABASE ora_compatible_db DBCOMPATIBILITY 'ORA';

--删除兼容TD、ORA格式的数据库。
openGauss=# DROP DATABASE td_compatible_db;
openGauss=# DROP DATABASE ora_compatible_db;
```


相关链接

ALTER DATABASE，**DROP DATABASE**

优化建议

- **create database**

  事务中不支持创建database。

- **ENCODING**

  当新建数据库Encoding与模板数据库（SQL_ASCII）不匹配（为'GBK' /'UTF8'/'LATIN1'/'GB18030'）时，必须指定template [=] template0。

7.13.47 CREATE DIRECTORY

功能描述

使用CREATE DIRECTORY语句创建一个目录对象，该目录对象定义了服务器文件系统上目录的别名，用于存放用户使用的数据文件，用户可以通过dbe_file高级包来读写这些文件。

该目录对象对于指定用户可以赋予READ和WRITE的操作权限，用于给dbe_file提供权限控制。

注意事项

- 当enable_access_server_directory=off时，只允许初始用户创建directory对象；当enable_access_server_directory=on时，具有SYSADMIN权限的用户和继承了内置角色gs_role_directory_create权限的用户可以创建directory对象。
- 创建用户默认拥有此路径的READ和WRITE操作权限。
- 目录的默认owner为创建directory的用户。
- 以下路径禁止创建：
  - 路径含特殊字符。
  - 路径是相对路径。
  - 路径是符号连接。
- 创建目录时会进行以下合法性校验：
  - 创建时会检查添加路径是否为操作系统实际存在路径，如不存在会提示用户使用风险。
  - 创建时会校验数据库初始化（omm）用户对于添加路径的权限(即操作系统目录权限，读/写/执行 - R/W/X)，如果权限不全，会提示用户使用风险。
- 在集群环境下用户指定的路径需要用户保证各节点上路径的一致性，否则在不同节点上执行会产生找不到路径的问题。

语法格式

```
CREATE [OR REPLACE] DIRECTORY directory_name
AS 'path_name';
```

参数说明

- **directory_name**

目录名称。

取值范围：字符串，要符标识符的命名规范。

- **path_name**

  操作系统的路径。

  取值范围： 有效的操作系统路径。

示例

```
--创建目录。
openGauss=# CREATE OR REPLACE DIRECTORY  dir  as '/tmp/';
```

相关链接

ALTER DIRECTORY，**DROP DIRECTORY**

7.13.48 CREATE FUNCTION

功能描述

创建一个函数。

注意事项

- 如果创建函数时参数或返回值带有精度，不进行精度检测。
- 创建函数时，函数定义中对表对象的操作建议都显式指定模式，否则可能会导致函数执行异常。
- 在创建函数时，函数内部通过SET语句设置current_schema和search_path无效。执行完函数后的search_path和current_schema与执行函数前的search_path和current_schema保持一致。
- 如果函数参数中带有出参，SELECT调用函数必须缺省出参，CALL调用函数必须指定出参，对于调用重载的带有PACKAGE属性的函数，CALL调用函数可以缺省出参，具体信息参见**CALL**的示例。
- 兼容PostgreSQL风格的函数或者带有PACKAGE属性的函数支持重载。在指定REPLACE的时候，如果参数个数、类型、返回值有变化，不会替换原有函数，而是会建立新的函数。
- SELECT调用可以指定不同参数来进行同名函数调用。语法CALL不支持调用不带有PACKAGE属性的同名函数。
- 在创建function时，不能在avg函数外面嵌套其他agg函数，或者其他系统函数。
- 在普通集群模式下，暂不支持将返回值、参数以及变量设置为建在非系统默认安装Node Group的表，sql function内部语句暂不支持对建在非系统默认安装Node Group的表操作。
- 新创建的函数默认会给PUBLIC授予执行权限（详见**GRANT**）。用户默认继承PUBLIC角色权限，因此其他用户也会有函数的执行权限并可以查看函数的定义，另外执行函数时还需要具备函数所在schema的USAGE权限。用户在创建函数时可以选择收回PUBLIC默认执行权限，然后根据需要将执行权限授予其他用户，为了避免出现新函数能被所有人访问的时间窗口，应在一个事务中创建函数并且设置函数执行权限。开启数据库对象隔离属性后，普通用户只能查看有权限执行的函数定义。
- 函数定义时如果指定为IMMUTABLE和SHIPPABLE类型，应该尽量避免函数中存在INSERT，UPDATE，DELETE，MERGE和DDL操作，因为上述操作应该由CN判断

对应的执行节点，否则执行结果可能产生错误。如果在声明为IMMUTABLE和SHIPPABLE类型的函数中下推执行了DDL，可能会导致各节点数据库对象不一致。修复此类问题可以在CN上创建VOLATILE PL/SQL函数，函数定义中使用execute语句动态执行用于修复系统对象的DDL，再使用EXECUTE DIRECT ON语法在指定的DN上执行修复函数调用，从而解决引入的问题。

- 在函数内部调用其它无参数的函数时，可以省略括号，直接使用函数名进行调用。
- 在函数内部调用其他有出参的函数，如果在赋值表达式中调用时，被调函数的出参可以省略，给出了也会被忽略。
- 兼容Oracle风格的函数支持参数注释的查看与导出、导入。
- 兼容Oracle风格的函数支持介于IS/AS与plsql_body之间的注释的查看与导出、导入。
- 被授予CREATE ANY FUNCTION权限的用户，可以在用户模式下创建/替换函数。
- 函数默认为SECURITY INVOKER权限，如果想将默认行为改为SECURITY DEFINER权限，需要设置guc参数behavior_compat_options='plsql_security_definer'。
- 不支持形参仅在自定义ref cursor类型和sys_refcursor类型不同的重载。
- 带OUT模式参数的函数不支持嵌套调用，比如 b := func(a,func(c,1));，建议改为tmp := func(c,1); b := func(a,tmp);。
- 如果将定义者权限的函数创建到其他用户SCHEMA下，则会以其他用户的权限执行该函数，有越权风险，请谨慎使用。

语法格式

- 兼容PostgreSQL风格的创建自定义函数语法。

```
CREATE [ OR REPLACE  ] FUNCTION function_name
   ( [  { argname [ argmode  ] argtype [  { DEFAULT  | :=  | =  } expression  ]}  [, ...]  ] )
   [ RETURNS rettype [ DETERMINISTIC  ]  | RETURNS TABLE (  { column_name column_type  }
[, ...] )]
   LANGUAGE lang_name
   [
     {IMMUTABLE  | STABLE  | VOLATILE }
      | {SHIPPABLE | NOT SHIPPABLE}
      | WINDOW
      | [ NOT  ] LEAKPROOF
      | {CALLED ON NULL INPUT  | RETURNS NULL ON NULL INPUT | STRICT }
      | {[ EXTERNAL  ] SECURITY INVOKER | [ EXTERNAL  ] SECURITY DEFINER | AUTHID DEFINER  |
AUTHID CURRENT_USER}
      | {fenced | not fenced}
      | {PACKAGE}

      | COST execution_cost
      | ROWS result_rows
      | SET configuration_parameter { {TO | =} value | FROM CURRENT }}
   ][...]
   {
      AS 'definition'
      | AS 'obj_file', 'link_symbol'
   }
```

- 兼容Oracle风格的创建自定义函数的语法。

```
CREATE [ OR REPLACE  ] FUNCTION function_name
   ( [  { argname [ argmode  ] argtype [  { DEFAULT | := | =  } expression  ] }  [, ...]  ] )
   RETURN rettype [ DETERMINISTIC  ]
   [
      {IMMUTABLE  | STABLE  | VOLATILE }
      | {SHIPPABLE | NOT SHIPPABLE}
      | {PACKAGE}
      | {FENCED | NOT FENCED}
      | [ NOT  ] LEAKPROOF
```



```
      | {CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
      | {[ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER |
AUTHID DEFINER | AUTHID CURRENT_USER
}
      | COST execution_cost
      | ROWS result_rows
      | SET configuration_parameter { {TO | =} value | FROM CURRENT
      | LANGUAGE lang_name
   ][...]

   {
     IS | AS
} plsql_body
/
```

参数说明

- **function_name**

  要创建的函数名称（可以用模式修饰）。

  取值范围：字符串，要符合**标识符命名规范**。

- **argname**

  函数参数的名称。

  取值范围：字符串，要符合**标识符命名规范**。

- **argmode**

  函数参数的模式。

  取值范围：IN，OUT，INOUT或VARIADIC。缺省值是IN。只有OUT模式的参数后面能跟VARIADIC。并且OUT和INOUT模式的参数不能用在RETURNS TABLE的函数定义中。

  说明

  VARIADIC用于声明数组类型的参数。

- **argtype**

  函数参数的类型。可以使用%ROWTYPE间接引用表的类型，或者使用%TYPE间接引用表或复合类型中某一列的类型。

- **expression**

  参数的默认表达式。

  说明

  推荐使用方式：将所有默认值参数定义在所有非默认值参数后。

- **rettype**

  函数返回值的数据类型。与argtype相同，可以使用%ROWTYPE或者%TYPE间接引用类型。

  如果存在OUT或IN OUT参数，可以省略RETURNS子句。如果存在，该子句必须和输出参数所表示的结果类型一致：如果有多个输出参数，则为RECORD，否则与单个输出参数的类型相同。

  SETOF修饰词表示该函数将返回一个集合，而不是单独一项。

- **column_name**

  字段名称。

- **column_type**

  字段类型。

- **definition**

  一个定义函数的字符串常量，含义取决于语言。它可以是一个内部函数名称、一个指向某个目标文件的路径、一个SQL查询、一个过程语言文本。

- **LANGUAGE lang_name**

  用以实现函数的语言的名称。可以是SQL，C，internal，或者是用户定义的过程语言名称。为了保证向下兼容，该名称可以用单引号（包围）。若采用单引号，则引号内必须为大写。

  由于兼容性问题，O风格的语法无论指定任何语言，最终创建的语言都为PL/SQL。

- **WINDOW**

  表示该函数是窗口函数，通常只用于C语言编写的函数。替换函数定义时不能改变WINDOW属性。

  须知

  自定义窗口函数只支持LANGUAGE是internal，并且引用的内部函数必须是窗口函数。

- **IMMUTABLE**

  表示该函数在给出同样的参数值时总是返回同样的结果。

- **STABLE**

  表示该函数不能修改数据库，对相同参数值，在同一次表扫描里，该函数的返回值不变，但是返回值可能在不同SQL语句之间变化。

- **VOLATILE**

  表示该函数值可以在一次表扫描内改变，因此不会做任何优化。

- **SHIPPABLE**

  NOT SHIPPABLE

  表示该函数是否可以下推到DN上执行。

  - 对于IMMUTABLE类型的函数，函数始终可以下推到DN上执行。
  - 对于STABLE/VOLATILE类型的函数，仅当函数的属性是SHIPPABLE的时候，函数可以下推到DN执行。

  须知

  对于指定了SHIPPABLE/IMMUTABLE的函数或者存储过程，其不能包含EXCEPTION或调用含有EXCEPTION的函数或者存储过程。

- PACKAGE

  表示该函数是否支持重载。

  - 不允许package函数和非package函数重载或者替换。
  - package函数不支持VARIADIC类型的参数。
  - 不允许修改函数的package属性。

- **LEAKPROOF**

  指出该函数的参数只包括返回值。LEAKPROOF只能由系统管理员设置。

- **CALLED ON NULL INPUT**

  表明该函数的某些参数是NULL的时候可以按照正常的方式调用。该参数可以省略。

- **RETURNS NULL ON NULL INPUT**

  STRICT

  STRICT用于指定如果函数的某个参数是NULL，此函数总是返回NULL。如果声明了这个参数，当有NULL值参数时该函数不会被执行；而只是自动返回一个NULL结果。

  RETURNS NULL ON NULL INPUT和STRICT的功能相同。

- **EXTERNAL**

  目的是和SQL兼容，是可选的，这个特性适合于所有函数，而不仅是外部函数。

- **SECURITY INVOKER**

  AUTHID CURRENT_USER

  表明该函数将带着调用它的用户的权限执行。该参数可以省略。

  SECURITY INVOKER和AUTHID CURRENT_USER的功能相同。

- **SECURITY DEFINER**

  AUTHID DEFINER

  声明该函数将以创建它的用户的权限执行。

  AUTHID DEFINER和SECURITY DEFINER的功能相同。

- **FENCED**

  NOT FENCED

  该参数用于声明函数是在保护模式还是非保护模式下执行。如果函数声明为NOT FENCED模式，则函数的执行在CN或者DN进程中进行。如果函数声明为FENCED模式，则函数在新fork的进程执行，这样函数的异常不会影响CN或者DN进程。

  FENCED/NOT FENCED模式的选择：

  - 正在开发或者调试的Function使用FENCED模式。开发测试完成，使用NOT FENCED模式执行，减少fork进程以及通信的开销。
  - 复杂的操作系统操作，例：打开文件，信号处理，线程处理等操作，使用FENCED模式。否则可能影响GaussDB数据库的执行。
  - 用户自定义PL/SQL函数，如果不指定该参数，默认为NOT FENCED，且不支持指定为FENCED执行模式。

- **COST execution_cost**

  用来估计函数的执行成本。

  execution_cost以cpu_operator_cost为单位。

  取值范围：正数

- **ROWS result_rows**

  估计函数返回的行数。用于函数返回的是一个集合。

  取值范围：正数，默认值是1000行。

- **configuration_parameter**
  - **value**

    把指定的数据库会话参数值设置为给定的值。如果value是DEFAULT或者RESET，则在新的会话中使用系统的缺省设置。OFF关闭设置。

    取值范围：字符串

  - DEFAULT
  - OFF
  - RESET
  - 指定默认值。
- **FROM CURRENT**

  取当前会话中的值设置为configuration_parameter的值。

- **obj_file, link_symbol**

  obj_file指定了动态库的绝对路径；link_sympol指定了该函数的链接符号，即函数在C代码中的函数名称。

- **plsql_body**

  PL/SQL存储过程体。

须知

当在函数体中进行创建用户、修改密码或加解密等涉及密码或密钥相关操作时，系统表及日志中会记录密码或密钥的明文信息。为防止敏感信息泄露，不建议用户在函数体中进行涉及密码或密钥等敏感信息的相关操作。

示例

```
--定义函数为SQL查询。
openGauss=# CREATE FUNCTION func_add_sql(integer, integer) RETURNS integer
    AS 'select $1 + $2;'
    LANGUAGE SQL
    IMMUTABLE
    RETURNS NULL ON NULL INPUT;

--利用参数名用 plpgsql 自增一个整数。
openGauss=# CREATE OR REPLACE FUNCTION func_increment_plsql(i integer) RETURNS integer AS $$
        BEGIN
                RETURN i + 1;
        END;
$$ LANGUAGE plpgsql;

--返回RECORD类型。
openGauss=# CREATE OR REPLACE FUNCTION compute(i int, out result_1 bigint, out result_2 bigint)
RETURNS SETOF RECORD
AS $$
BEGIN
    result_1 = i + 1;
    result_2 = i * 10;
RETURN next;
END;
$$LANGUAGE plpgsql;

--返回一个包含多个输出参数的记录。
openGauss=# CREATE FUNCTION func_dup_sql(in int, out f1 int, out f2 text)
    AS $$ SELECT $1, CAST($1 AS text) || ' is text' $$
    LANGUAGE SQL;

openGauss=# SELECT * FROM func_dup_sql(42);

--计算两个整数的和，并返回结果。如果输入为null，则返回null。
openGauss=# CREATE FUNCTION func_add_sql2(num1 integer, num2 integer) RETURN integer
AS
```



```
BEGIN
RETURN num1 + num2;
END;
/
--创建package属性的重载函数
openGauss=# CREATE OR REPLACE FUNCTION package_func_overload(col int, col2  int)
RETURN integer package
AS
DECLARE
    col_type text;
BEGIN
     col := 122;
        dbe_output.print_line('two int parameters ' || col2);
        return 0;
END;
/

openGauss=# CREATE OR REPLACE FUNCTION package_func_overload(col int, col2 smallint)
RETURN integer package
AS
DECLARE
    col_type text;
BEGIN
     col := 122;
        dbe_output.print_line('two smallint parameters ' || col2);
        RETURN 0;
END;
/

--修改函数add的执行规则为IMMUTABLE，即参数不变时返回相同结果。
openGauss=# ALTER FUNCTION func_add_sql2(INTEGER, INTEGER) IMMUTABLE;

--将函数add的名称修改为add_two_number。
openGauss=# ALTER FUNCTION func_add_sql2(INTEGER, INTEGER) RENAME TO add_two_number;

--将函数add的属者改为omm。
openGauss=# ALTER FUNCTION add_two_number(INTEGER, INTEGER) OWNER TO omm;

--删除函数。
openGauss=# DROP FUNCTION add_two_number;
openGauss=# DROP FUNCTION func_increment_sql;
openGauss=# DROP FUNCTION func_dup_sql;
openGauss=# DROP FUNCTION func_increment_plsql;
openGauss=# DROP FUNCTION func_add_sql;
```

相关链接

ALTER FUNCTION，**DROP FUNCTION**

优化建议

- analyse | analyze
  - 不支持在事务或匿名块中执行analyze 。
  - 不支持在函数或存储过程中执行analyze操作。

7.13.49 CREATE GROUP

功能描述

创建一个新用户组。

注意事项

CREATE GROUP是CREATE ROLE的别名，非SQL标准语法，不推荐使用，建议用户直接使用CREATE ROLE替代。

语法格式

```
CREATE GROUP group_name [ [ WITH ] option [ ... ] ]
    [ ENCRYPTED | UNENCRYPTED ] { PASSWORD | IDENTIFIED BY } { 'password' [ EXPIRED ] | DISABLE };
```

其中可选项action子句语法为：

```
{SYSADMIN | NOSYSADMIN}
    | {MONADMIN | NOMONADMIN}
    | {OPRADMIN | NOOPRADMIN}
    | {POLADMIN | NOPOLADMIN}
    | {AUDITADMIN | NOAUDITADMIN}
    | {CREATEDB | NOCREATEDB}
    | {USEFT | NOUSEFT}
    | {CREATEROLE | NOCREATEROLE}
    | {INHERIT | NOINHERIT}
    | {LOGIN | NOLOGIN}
    | {REPLICATION | NOREPLICATION}
    | {VCADMIN | NOVCADMIN}
    | {PERSISTENCE | NOPERSISTENCE}
    | CONNECTION LIMIT connlimit
    | VALID BEGIN 'timestamp'
    | VALID UNTIL 'timestamp'
    | USER GROUP 'groupuser'
    | PERM SPACE 'spacelimit'
    | NODE GROUP logic_group_name
    | IN ROLE role_name [, ...]
    | IN GROUP role_name [, ...]
    | ROLE role_name [, ...]
    | ADMIN role_name [, ...]
    | USER role_name [, ...]
    | SYSID uid
    | DEFAULT TABLESPACE tablespace_name
    | PROFILE DEFAULT
    | PROFILE profile_name
    | PGUSER
```

参数说明

请参考CREATE ROLE的**参数说明**。

示例

```
--创建组my_group。
openGauss=# CREATE GROUP my_group PASSWORD '********';

--删除组。
openGauss=# DROP GROUP my_group;
```

相关链接

ALTER GROUP，**DROP GROUP**，**CREATE ROLE**

7.13.50 CREATE INCREMENTAL MATERIALIZED VIEW

功能描述

CREATE INCREMENTAL MATERIALIZED VIEW会创建一个增量物化视图，后续可以使用REFRESH MATERIALIZED VIEW（全量刷新）和REFRESH INCREMENTAL MATERIALIZED VIEW（增量刷新）刷新物化视图的数据。

CREATE INCREMENTAL MATERIALIZED VIEW类似于CREATE TABLE AS，不过它会记住被用来初始化该视图的查询， 因此它可以在后续中进行数据刷新。一个物化视图有很多和表相同的属性，但是不支持临时物化视图。

注意事项

- 增量物化视图不可以在临时表或全局临时表上创建。
- 增量物化视图仅支持简单过滤查询和基表UNION ALL查询。
- 创建增量物化视图不可指定分布列。
- 创建增量物化视图后，基表中的绝大多数DDL操作不再支持。
- 不支持对增量物化视图进行IUD操作。
- 增量物化视图创建后，当基表数据发生变化时，需要使用刷新（REFRESH）命令保持物化视图与基表同步。

语法格式

```
CREATE INCREMENTAL MATERIALIZED VIEW mv_name
   [ (column_name [, ...] ) ]
   [ TABLESPACE tablespace_name ]
   AS query;
```

参数说明

- **mv_name**

  要创建的物化视图的名称（可以被模式限定）。

  取值范围：字符串，要符合**标识符命名规范**。

- **column_name**

  新物化视图中的一个列名。物化视图支持指定列，指定列需要和后面的查询语句结果的列数量保持一致；如果没有提供列名，会从查询的输出列名中获取列名。

  取值范围：字符串，要符合**标识符命名规范**。

- **TABLESPACE tablespace_name**

  指定新建物化视图所属表空间。如果没有声明，将使用默认表空间。

- **AS query**

  一个SELECT或者TABLE 命令。这个查询将在一个安全受限的操作中运行。

示例

```
--创建一个普通表。
openGauss=# CREATE TABLE my_table (c1 int, c2 int);

--创建增量物化视图。
openGauss=# CREATE INCREMENTAL MATERIALIZED VIEW my_imv AS SELECT * FROM my_table;

--基表写入数据。
```



```
openGauss=# INSERT INTO my_table VALUES(1,1),(2,2);

--对增量物化视图my_imv进行增量刷新。
openGauss=# REFRESH INCREMENTAL MATERIALIZED VIEW my_imv;

--删除增量物化视图。
openGauss=# DROP MATERIALIZED VIEW my_imv;

--删除普通表my_table。
openGauss=# DROP TABLE my_table;
```

相关链接

ALTER MATERIALIZED VIEW，**CREATE MATERIALIZED VIEW**，**CREATE TABLE**，**DROP MATERIALIZED VIEW**，**REFRESH INCREMENTAL MATERIALIZED VIEW**，**REFRESH MATERIALIZED VIEW**

7.13.51 CREATE INDEX

功能描述

在指定的表上创建索引。

索引可以用来提高数据库查询性能，但是不恰当的使用将导致数据库性能下降。建议仅在匹配如下某条原则时创建索引：

- 经常执行查询的字段。
- 在连接条件上创建索引，对于存在多字段连接的查询，建议在这些字段上建立组合索引。例如，select * from t1 join t2 on t1.a=t2.a and t1.b=t2.b，可以在t1表上的a，b字段上建立组合索引。
- where子句的过滤条件字段上（尤其是范围条件）。
- 在经常出现在order by、group by和distinct后的字段。

在分区表上创建索引与在普通表上创建索引的语法不太一样，使用时请注意，如当索引带GLOBAL/LOCAL关键字或者创建索引为GLOBAL索引时不支持创建部分索引。

注意事项

- 索引自身也占用存储空间、消耗计算资源，创建过多的索引将对数据库性能造成负面影响（尤其影响数据导入的性能，建议在数据导入后再建索引）。因此，仅在必要时创建索引。
- 索引定义里的所有函数和操作符都必须是immutable类型的，即它们的结果必须只能依赖于它们的输入参数，而不受任何外部的影响（如另外一个表的内容或者当前时间）。这个限制可以确保该索引的行为是定义良好的。要在一个索引上或WHERE中使用用户定义函数，请把它标记为immutable类型函数。
- 在分区表上创建唯一索引时，索引项必须包含分布列。索引项不包含分区键只能创建全局分区索引。
- 被授予CREATE ANY INDEX权限的用户，可以在public模式和用户模式下创建索引。
- 如果表达式索引中调用的是用户自定义函数，按照函数创建者权限执行表达式索引函数。

语法格式

- 在表上创建索引。

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ [schemaname.]index_name ] ON table_name
[ USING method ]
    ({ { column_name | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ NULLS
{ FIRST | LAST } ] }[, ...] )
    [ WITH ( {storage_parameter = value} [, ... ] ) ]
    [ TABLESPACE tablespace_name ]
    [ WHERE predicate ];
```

- 在分区表上创建索引。

```
CREATE [ UNIQUE ] INDEX [ [schemaname.]index_name ] ON table_name [ USING method ]
    ( { { column_name | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ NULLS
LAST ] } [, ...] )
    [ LOCAL [ ( { PARTITION index_partition_name [ TABLESPACE index_partition_tablespace ] }
[, ...] ) ] | GLOBAL ]
    [ WITH ( { storage_parameter = value } [, ...] ) ]
    [ TABLESPACE tablespace_name ]
    [ WHERE predicate ];
```

参数说明

- **UNIQUE**

  创建唯一性索引，每次添加数据时检测表中是否有重复值。如果插入或更新的值会引起重复的记录时，将导致一个错误。

  目前只有行存表B-tree及UBtree索引支持唯一索引。

- **CONCURRENTLY**

  以不阻塞DML的方式创建索引（加ShareUpdateExclusiveLock锁）。创建索引时，一般会阻塞其他语句对该索引所依赖表的访问。指定此关键字，可以实现创建过程中不阻塞DML。

  - 此选项只能指定一个索引的名称。
  - 普通CREATE INDEX命令可以在事务内执行，但是CREATE INDEX CONCURRENTLY不可以在事务内执行。
  - 分区表不支持CONCURRENTLY方式创建索引，对于临时表，支持使用CONCURRENTLY关键字创建索引，但是实际创建过程中，采用的是阻塞式的创建方式，因为没有其他会话会并发访问临时表，并且阻塞式创建成本更低。

说明

- 创建索引时指定此关键字，需要执行先后两次对该表的全表扫描来完成build，第一次扫描的时候创建索引，不阻塞读写操作；第二次扫描的时候合并更新第一次扫描到目前为止发生的变更。
- 由于需要执行两次对表的扫描和build，而且必须等待现有的所有可能对该表执行修改的事务结束。这意味着该索引的创建比正常耗时更长，同时因此带来的CPU和I/O消耗对其他业务也会造成影响。
- 如果在索引构建时发生失败，那会留下一个“不可用”的索引。这个索引会被查询忽略，但它仍消耗更新开销。这种情况推荐的恢复方法是通过DROP INDEX IF EXISTS语法删除该索引并尝试再次CONCURRENTLY建索引。
- 由于在第二次扫描之后，索引构建必须等待任何持有早于第二次扫描拿的快照的事务终止，而且建索引时加的ShareUpdateExclusiveLock锁（4级）会和大于等于4级的锁冲突，在创建这类索引时，容易引发卡住（hang）或者死锁问题。例如：
  - 两个会话对同一个表创建CONCURRENTLY索引，会引起死锁问题。
  - 两个会话，一个对表创建CONCURRENTLY索引，一个drop table，会引起死锁问题。
  - 三个会话，会话1先对表a加锁，不提交，会话2接着对表b创建CONCURRENTLY索引，会话3接着对表a执行写入操作，在会话1事务未提交之前，会话2会一直被阻塞。
  - 创建CONCURRENTLY索引与同一个表的TRUNCATE操作并发，会引起死锁问题。
  - 将事务隔离级别设置成可重复读（默认为读已提交），起两个会话，会话1起事务对表a执行写入操作，不提交，会话2对表b创建CONCURRENTLY索引，在会话1事务未提交之前，会话2会一直被阻塞。
- 索引构建过程中或者构建失败的情况下，需要确认索引进度或状态，可以通过查询函数gs_get_index_status('schema_name', 'index_name')来确认当前所有节点上索引的状态，其中入参为schema_name和index_name，分别用来指定索引的模式名称和索引名称，返回值为node_name，indisready和indisvalid，分别表示节点名称，索引在该节点上是否可插入，以及索引在该节点上是否可用，只有当所有节点indisready和indisvalid均为true的情况下，索引才是“可用的”，否则请等待索引创建完成，或者构建失败情况下，删除索引重新创建。
- 在I/O、CPU不受限的情况下，在线创建索引对业务性能的劣化一般可以控制在10%以内，但在特殊场景下劣化可能会超过此数值。这是因为在线创建索引本身是一种消耗I/O、CPU资源较多的长事务，需要比离线创建索引消耗更多的资源。在线创建索引事务持续时间越长，对业务性能的影响越大。在线创建索引时间与基表数据量、并发DML产生的数据量正相关，在I/O、CPU不受限的情况下，在线创建索引时间大约是离线创建索引的2~6倍，但当并发事务量较大（>10000tps）或存在资源争抢的情况时，可能会超过此数值。可以使用并行创建索引来缩短创建索引时间；在线并行创建索引性能随着并行工作线程数量增加而提升到一定值后稳定，相比串行创建索引性能一般可提升30%左右。建议在业务低谷期进行在线创建索引，以避免对业务造成较大影响。虽然在线创建索引在一定程度上提供了业务不中断的能力，但仍然需要谨慎实施。

- **schema_name**

  模式的名称。

  取值范围：已存在模式名。

- **index_name**

  要创建的索引名，不能包含模式名，索引的模式与表相同。

  取值范围：字符串，要符合**标识符命名规范**。

- **table_name**

  需要为其创建索引的表的名称，可以用模式修饰。

  取值范围：已存在的表名。

- **USING method**

  指定创建索引的方法。

取值范围：

- btree：B-tree索引使用一种类似于B+树的结构来存储数据的键值，通过这种结构能够快速的查找索引。btree适合支持比较查询以及查询范围。创建索引时，当表为ustore时会自动变换为ubtree。

行存表支持的索引类型：btree（行存表缺省值）。

- **column_name**

  表中需要创建索引的列的名称（字段名）。

  如果索引方式支持多字段索引，可以声明多个字段。全局索引最多可以声明31个字段，其他索引最多可以声明32个字段。

- **expression**

  创建一个基于该表的一个或多个字段的表达式索引，通常必须写在圆括号中。如果表达式有函数调用的形式，圆括号可以省略。

  表达式索引可用于获取对基本数据的某种变形的快速访问。比如，一个在upper(col)上的函数索引将允许WHERE upper(col) = 'JIM'子句使用索引。

  在创建表达式索引时，如果表达式中包含IS NULL子句，则这种索引是无效的。此时，建议用户尝试创建一个部分索引。

- **COLLATE collation**

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

- **opclass**

  操作符类的名称。对于索引的每一列可以指定一个操作符类，操作符类标识了索引那一列的使用的操作符。例如一个B-tree索引在一个四字节整数上可以使用int4_ops；这个操作符类包括四字节整数的比较函数。实际上对于列上的数据类型默认的操作符类是足够用的。操作符类主要用于一些有多种排序的数据。例如，用户想按照绝对值或者实数部分排序一个复数，可以通过定义两个操作符类，然后在建立索引时选择合适的类。

- **ASC**

  指定按升序排序（默认）。

- **DESC**

  指定按降序排序。

- **NULLS FIRST**

  指定空值在排序中排在非空值之前，当指定DESC排序时，本选项为默认的。

- **NULLS LAST**

  指定空值在排序中排在非空值之后，未指定DESC排序时，本选项为默认的。

- **WITH ({storage_parameter = value} [, ...])**

  指定索引方法的存储参数。

  取值范围：

  Psort之外的索引都支持FILLFACTOR参数。

  - FILLFACTOR

    一个索引的填充因子（fillfactor）是一个介于10和100之间的百分数。

    取值范围：10~100

  - CROSSBUCKET

    索引是否使用跨hashbucket索引。仅支持B-Tree索引。

    取值范围：ON，OFF

    默认值：ON

- **TABLESPACE tablespace_name**

  指定索引的表空间，如果没有声明则使用默认的表空间。

  取值范围：已存在的表空间名。

- **WHERE predicate**

  创建一个部分索引。部分索引是一个只包含表的一部分记录的索引，通常是该表中比其他部分数据更有用的部分。例如，有一个表，表里包含已记账和未记账的订单，未记账的订单只占表的一小部分而且这部分是最常用的部分，此时就可以通过只在未记账部分创建一个索引来改善性能。另外一个可能的用途是使用带有UNIQUE的WHERE强制一个表的某个子集的唯一性。

  取值范围：predicate表达式只能引用表的字段，它可以使用所有字段，而不仅是被索引的字段。目前，子查询和聚集表达式不能出现在WHERE子句里。不建议使用int等数值类型作为predicate，因为int等数值类型可以隐式转换为bool值（非0值隐式转换为true，0转换为false），可能导致非预期的结果。

  对于分区表索引，当创建索引带GLOBAL/LOCAL关键字，或者最终创建的索引类型为GLOBAL索引时，不支持带WHERE子句创建索引。

- **PARTITION index_partition_name**

  索引分区的名称。

  取值范围：字符串，要符合**标识符命名规范**。

- **TABLESPACE index_partition_tablespace**

  索引分区的表空间。

  取值范围：如果没有声明，将使用分区表索引的表空间index_tablespace。

示例

```
--创建表tpcds.ship_mode_t1。
openGauss=# CREATE SCHEMA tpcds;
openGauss=# CREATE TABLE tpcds.ship_mode_t1
(
    SM_SHIP_MODE_SK          INTEGER              NOT NULL,
    SM_SHIP_MODE_ID          CHAR(16)             NOT NULL,
    SM_TYPE                  CHAR(30)                     ,
    SM_CODE                  CHAR(10)                     ,
    SM_CARRIER               CHAR(20)                     ,
    SM_CONTRACT              CHAR(20)
)
DISTRIBUTE BY HASH(SM_SHIP_MODE_SK);

--在表tpcds.ship_mode_t1上的SM_SHIP_MODE_SK字段上创建普通的唯一索引。
openGauss=# CREATE UNIQUE INDEX ds_ship_mode_t1_index1 ON
tpcds.ship_mode_t1(SM_SHIP_MODE_SK);

--在表tpcds.ship_mode_t1上的SM_SHIP_MODE_SK字段上创建指定B-tree索引。
openGauss=# CREATE INDEX ds_ship_mode_t1_index4 ON tpcds.ship_mode_t1 USING
btree(SM_SHIP_MODE_SK);

--在表tpcds.ship_mode_t1上SM_CODE字段上创建表达式索引。
openGauss=# CREATE INDEX ds_ship_mode_t1_index2 ON tpcds.ship_mode_t1(SUBSTR(SM_CODE,1 ,4));

--在表tpcds.ship_mode_t1上的SM_SHIP_MODE_SK字段上创建SM_SHIP_MODE_SK大于10的部分索引。
openGauss=# CREATE UNIQUE INDEX ds_ship_mode_t1_index3 ON
tpcds.ship_mode_t1(SM_SHIP_MODE_SK) WHERE SM_SHIP_MODE_SK>10;
```



```
--在表tpcds.ship_mode_t1上的SM_SHIP_MODE_SK字段上以不阻塞DML的方式创建索引。
openGauss=# CREATE INDEX CONCURRENTLY ds_ship_mode_t1_index4 ON
tpcds.ship_mode_t1(SM_SHIP_MODE_SK);

--重命名一个现有的索引。
openGauss=# ALTER INDEX tpcds.ds_ship_mode_t1_index1 RENAME TO ds_ship_mode_t1_index5;

--设置索引不可用。
openGauss=# ALTER INDEX tpcds.ds_ship_mode_t1_index2 UNUSABLE;

--重建索引。
openGauss=# ALTER INDEX tpcds.ds_ship_mode_t1_index2 REBUILD;

--删除一个现有的索引。
openGauss=# DROP INDEX tpcds.ds_ship_mode_t1_index2;

--删除表。
openGauss=# DROP TABLE tpcds.ship_mode_t1;

--创建表空间。
openGauss=# CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace1/tablespace_1';
openGauss=# CREATE TABLESPACE example2 RELATIVE LOCATION 'tablespace2/tablespace_2';
openGauss=# CREATE TABLESPACE example3 RELATIVE LOCATION 'tablespace3/tablespace_3';
openGauss=# CREATE TABLESPACE example4 RELATIVE LOCATION 'tablespace4/tablespace_4';
--创建表tpcds.customer_address_p1。
openGauss=# CREATE TABLE tpcds.customer_address_p1
(
    CA_ADDRESS_SK             INTEGER               NOT NULL,
    CA_ADDRESS_ID             CHAR(16)              NOT NULL,
    CA_STREET_NUMBER          CHAR(10)                      ,
    CA_STREET_NAME            VARCHAR(60)                   ,
    CA_STREET_TYPE            CHAR(15)                      ,
    CA_SUITE_NUMBER           CHAR(10)                      ,
    CA_CITY                   VARCHAR(60)                   ,
    CA_COUNTY                 VARCHAR(30)                   ,
    CA_STATE                  CHAR(2)                       ,
    CA_ZIP                    CHAR(10)                      ,
    CA_COUNTRY                VARCHAR(20)                   ,
    CA_GMT_OFFSET             DECIMAL(5,2)                  ,
    CA_LOCATION_TYPE          CHAR(20)
)
TABLESPACE example1
DISTRIBUTE BY HASH(CA_ADDRESS_SK)
PARTITION BY RANGE(CA_ADDRESS_SK)
(
   PARTITION p1 VALUES LESS THAN (3000),
   PARTITION p2 VALUES LESS THAN (5000) TABLESPACE example1,
   PARTITION p3 VALUES LESS THAN (MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;
--创建分区表索引ds_customer_address_p1_index1，不指定索引分区的名称。
openGauss=# CREATE INDEX ds_customer_address_p1_index1 ON
tpcds.customer_address_p1(CA_ADDRESS_SK) LOCAL;
--创建分区表索引ds_customer_address_p1_index2，并指定索引分区的名称。
openGauss=# CREATE INDEX ds_customer_address_p1_index2 ON
tpcds.customer_address_p1(CA_ADDRESS_SK) LOCAL
(
    PARTITION CA_ADDRESS_SK_index1,
    PARTITION CA_ADDRESS_SK_index2 TABLESPACE example3,
    PARTITION CA_ADDRESS_SK_index3 TABLESPACE example4
)
TABLESPACE example2;

--修改分区表索引CA_ADDRESS_SK_index2的表空间为example1。
openGauss=# ALTER INDEX tpcds.ds_customer_address_p1_index2 MOVE PARTITION
CA_ADDRESS_SK_index2 TABLESPACE example1;

--修改分区表索引CA_ADDRESS_SK_index3的表空间为example2。
```



```
openGauss=# ALTER INDEX tpcds.ds_customer_address_p1_index2 MOVE PARTITION
CA_ADDRESS_SK_index3 TABLESPACE example2;

--重命名分区表索引。
openGauss=# ALTER INDEX tpcds.ds_customer_address_p1_index2 RENAME PARTITION
CA_ADDRESS_SK_index1 TO CA_ADDRESS_SK_index4;

--删除索引和分区表。
openGauss=# DROP INDEX tpcds.ds_customer_address_p1_index1;
openGauss=# DROP INDEX tpcds.ds_customer_address_p1_index2;
openGauss=# DROP TABLE tpcds.customer_address_p1;
--删除表空间。
openGauss=# DROP TABLESPACE example1;
openGauss=# DROP TABLESPACE example2;
openGauss=# DROP TABLESPACE example3;
openGauss=# DROP TABLESPACE example4;
```

相关链接

ALTER INDEX，**DROP INDEX**

优化建议

- create index

  建议仅在匹配如下条件之一时创建索引：

  - 经常执行查询的字段。
  - 在连接条件上创建索引，对于存在多字段连接的查询，建议在这些字段上建立组合索引。例如，select * from t1 join t2 on t1.a=t2.a and t1.b=t2.b，可以在t1表上的a，b字段上建立组合索引。
  - where子句的过滤条件字段上（尤其是范围条件）。
  - 在经常出现在order by、group by和distinct后的字段。

  约束限制：

  - 普通表的索引支持最大列数为32列；分区表的GLOBAL索引支持最大列数为31列。
  - 单个索引大小不能超过索引页面大小（8k），其中B-tree、UBtree不能超过页面大小的三分之一。
  - 分区表上不支持创建部分索引。
  - 在分区表上创建唯一索引时，索引项中必须包含分布列。索引项不包含分区键只能创建全局分区索引。

7.13.52 CREATE LANGUAGE

本版本暂不支持使用该语法。

7.13.53 CREATE MASKING POLICY

功能描述

创建脱敏策略。

注意事项

只有poladmin，sysadmin或初始用户能执行此操作。

需要开启安全策略开关，即设置GUC参数enable_security_policy=on，脱敏策略才可以生效。

语法格式

```
CREATE MASKING POLICY policy_name masking_clause[, ...]  [ policy_filter_clause ] [ENABLE | DISABLE];
```

- masking_clause：
  ```
  masking_function ON LABEL(label_name[, ...])
  ```
- masking_function：

  maskall不是预置函数，硬编码在代码中，不支持\df展示。

  预置时脱敏方式如下：
  ```
  { maskall | randommasking | creditcardmasking | basicemailmasking | fullemailmasking |
  shufflemasking | alldigitsmasking | regexpmasking }
  ```
- policy_filter_clause:
  ```
  FILTER ON { FILTER_TYPE ( filter_value [, ...] ) } [, ...]
  ```
- FILTER_TYPE:
  ```
  IP | APP | ROLES
  ```

参数说明

- **policy_name**

  审计策略名称，需要唯一，不可重复。

  取值范围：字符串，要符合**标识符命名规范**。
- **label_name**

  资源标签名称。
- **masking_clause**

  指出使用何种脱敏函数对被label_name标签标记的数据库资源进行脱敏，支持用schema.function的方式指定脱敏函数。
- **policy_filter**

  指出该脱敏策略对何种身份的用户生效，若为空表示对所有用户生效。
- **FILTER_TYPE**

  描述策略过滤的条件类型，包括IP | APP | ROLES。
- **filter_value**

  指具体过滤信息内容，例如具体的IP，具体的APP名称，具体的用户名。
- **ENABLE|DISABLE**

  可以打开或关闭脱敏策略。若不指定ENABLE|DISABLE，语句默认为ENABLE。

示例

```
--创建dev_mask和bob_mask用户。
openGauss=# CREATE USER dev_mask PASSWORD 'xxxxxxxxxx';
openGauss=# CREATE USER bob_mask PASSWORD 'xxxxxxxxxx';

--创建一个表tb_for_masking。
openGauss=# CREATE TABLE tb_for_masking(col1 text, col2 text, col3 text);

--创建资源标签标记敏感列col1。
openGauss=# CREATE RESOURCE LABEL mask_lb1 ADD COLUMN(tb_for_masking.col1);

--创建资源标签标记敏感列col2。
openGauss=# CREATE RESOURCE LABEL mask_lb2 ADD COLUMN(tb_for_masking.col2);
```



```
--对访问敏感列col1的操作创建脱敏策略。
openGauss=# CREATE MASKING POLICY maskpol1 maskall ON LABEL(mask_lb1);

--创建仅对用户dev_mask和bob_mask,客户端工具为gsql，IP地址为'10.20.30.40', '127.0.0.0/24'场景下生效的脱敏策略。
openGauss=# CREATE MASKING POLICY maskpol2 randommasking ON LABEL(mask_lb2) FILTER ON ROLES(dev_mask, bob_mask), APP(gsql), IP('10.20.30.40', '127.0.0.0/24');

--删除脱敏策略。
openGauss=# DROP MASKING POLICY maskpol1, maskpol2;

--删除资源标签。
openGauss=# DROP RESOURCE LABEL mask_lb1, mask_lb2;

--删除表tb_for_masking。
openGauss=# DROP TABLE tb_for_masking;

--删除用户dev_mask和bob_mask。
openGauss=# DROP USER dev_mask, bob_mask;
```

相关链接

5.1.13.14.14-ALTER MASKING POLICY，**5.1.13.14.96-DROP MASKING POLICY**。

7.13.54 CREATE MATERIALIZED VIEW

CREATE MATERIALIZED VIEW会创建一个全量物化视图，并且后续可以使用REFRESH MATERIALIZED VIEW（全量刷新）刷新物化视图的数据。

CREATE MATERIALIZED VIEW类似于CREATE TABLE AS，不过它会记住被用来初始化该视图的查询， 因此它可以在后续中进行数据刷新。一个物化视图有很多和表相同的属性，但是不支持临时物化视图。

注意事项

- 全量物化视图不可以在临时表或全局临时表上创建。
- 全量物化视图不支持nodegroup。
- 创建全量物化视图后，基表中的绝大多数DDL操作不再支持。
- 不支持对全量物化视图进行IUD操作。
- 全量物化视图创建后，当基表数据发生变化时，需要使用刷新（REFRESH）命令保持物化视图与基表同步。

语法格式

```
CREATE MATERIALIZED VIEW mv_name
   [ (column_name [, ...] ) ]
   [ WITH ( {storage_parameter = value} [, ... ] ) ]
   [ TABLESPACE tablespace_name ]
   AS query;
```

参数说明

- **mv_name**

  要创建的物化视图的名称（可以被模式限定）。

  取值范围：字符串，要符合**标识符命名规范**。

- **column_name**

新物化视图中的一个列名。物化视图支持指定列，指定列需要和后面的查询语句结果的列数量保持一致；如果没有提供列名，会从查询的输出列名中获取列名。

取值范围：字符串，要符合**标识符命名规范**。

- **WITH (storage_parameter [= value] [, ...])**

  这个子句为表或索引指定一个可选的存储参数。详见**CREATE TABLE**。

- **TABLESPACE tablespace_name**

  指定新建物化视图所属表空间。如果没有声明，将使用默认表空间。

- **AS query**

  一个SELECT、TABLE 或者VALUES命令。这个查询将在一个安全受限的操作中运行。

示例

```
--创建一个普通表。
openGauss=# CREATE TABLE my_table (c1 int, c2 int);

--创建全量物化视图。
openGauss=# CREATE MATERIALIZED VIEW my_mv AS SELECT * FROM my_table;

--基表写入数据。
openGauss=# INSERT INTO my_table VALUES(1,1),(2,2);

--对全量物化视图my_mv进行全量刷新。
openGauss=# REFRESH MATERIALIZED VIEW my_mv;

--删除全量物化视图。
openGauss=# DROP MATERIALIZED VIEW my_mv;

--删除普通表my_table。
openGauss=# DROP TABLE my_table;
```

相关链接

ALTER MATERIALIZED VIEW，**CREATE INCREMENTAL MATERIALIZED VIEW**，**CREATE TABLE**，**DROP MATERIALIZED VIEW**，**REFRESH INCREMENTAL MATERIALIZED VIEW**，**REFRESH MATERIALIZED VIEW**

7.13.55 CREATE NODE

功能描述

创建一个新的集群节点。

注意事项

CREATE NODE是集群管理工具封装的接口，用来实现集群管理。该接口不建议用户直接使用，以免对集群状态造成影响。管理员用户才有权限使用该接口。

语法格式

```
CREATE NODE nodename WITH
 (
   [ TYPE = nodetype,]
   [ HOST = hostname,]
   [ PORT = portnum,]
   [ HOST1 = 'hostname',]
   [ PORT1 = portnum,]
```



```
  [ HOSTPRIMARY [ = boolean ],]
  [ PRIMARY [ = boolean ],]
  [ PREFERRED [ = boolean ],]
  [ SCTP_PORT = portnum,]
  [ CONTROL_PORT = portnum,]
  [ SCTP_PORT1 = portnum,]
  [ CONTROL_PORT1 = portnum ]
);
```

参数说明

- **nodename**
  节点名称。
  取值范围：字符串，要符合**标识符命名规范**。
- **TYPE = nodetype**
  指定节点的类型。
  取值范围：
  - 'coordinator'
  - 'datanode'
- **HOST = hostname**
  指定节点对应的主机名称或者IP地址。
- **PORT = portnum**
  指定节点绑定的主机端口号。
- **HOST1 = hostname**
  指定节点对应的备机名称或者IP地址。
- **PORT1 = portnum**
  指定节点绑定的备机端口号。
- **HOSTPRIMARY**
- **PRIMARY = boolean**
  声明该节点是否为主节点。主节点允许做读写操作，否则只允许读操作。
  取值范围：
  - true
  - false（默认值）
- **PREFERRED = boolean**
  声明该节点是否为读操作的首选节点。
  取值范围：
  - true
  - false（默认值）
- **SCTP_PORT = portnum**
  主机TCP代理通信库使用的数据传输通道侦听端口，使用TCP协议侦听连接。
- **CONTROL_PORT = portnum**
  主机TCP代理通信库使用的控制传输通道侦听端口，使用TCP协议侦听连接。
- **SCTP_PORT1 = portnum**
  备机TCP代理通信库使用的数据传输通道侦听端口，使用TCP协议侦听连接。

- **CONTROL_PORT 1= portnum**
  备机TCP代理通信库使用的控制传输通道侦听端口，使用TCP协议侦听连接。

相关链接

ALTER NODE，**DROP NODE**。

7.13.56 CREATE NODE GROUP

功能描述

创建一个新的集群节点组。

注意事项

- CREATE NODE GROUP是集群管理工具封装的接口，用来实现集群管理。
- 该接口仅对管理员用户开放使用。

语法格式

```
CREATE NODE GROUP groupname
   [WITH ( nodename [, ... ] )] [bucketcnt bucket_cnt]
   [ BUCKETS [ ( bucketnumber [, ... ] ) ] ] [VCGROUP] [DISTRIBUTE FROM src_group_name] [groupparent parent_group_name];
```

参数说明

- **groupname**
  节点组名称。
  取值范围：字符串，要符合**标识符命名规范**。且最大长度不超过63个字符。

  说明

  节点组命名支持ASCII字符集上所有字符，但是建议用户按照标识符命名规范命名。

- **nodename**
  节点名称。
  取值范围：字符串，要符合**标识符命名规范**。且最大长度不超过63个字符。
  不指定该参数时，需要指定bucketcnt值，表示创建属于installation node group的child node group

- bucketcnt bucket_cnt
  bucket_cnt表示bucket桶数量。
  取值范围：[32,16384)，并且必须是2的幂次方。
  不指定该参数时，需要指定WITH的值。

- **BUCKETS [(bucketnumber [, ...])]**
  BUCKETS子句是集群管理工具的内部用法，该子句不建议用户直接使用，以免对集群的正常使用造成影响。

- groupparent parent_group_name
  parent_group_name表示当前child node group所属的parent node group名字。

相关链接

DROP NODE GROUP

7.13.57 CREATE PROCEDURE

功能描述

创建一个新的存储过程。

注意事项

- 如果创建存储过程时参数或返回值带有精度，不进行精度检测。
- 创建存储过程时，存储过程定义中对表对象的操作建议都显示指定模式，否则可能会导致存储过程执行异常。
- 在创建存储过程时，存储过程内部通过SET语句设置current_schema和search_path无效。执行完函数search_path和current_schema与执行函数前的search_path和current_schema保持一致。
- 如果存储过程参数中带有出参，SELECT调用存储过程必须缺省出参，CALL调用存储过程时调用非重载函数必须指定出参，对于重载的package函数，out参数可以缺省，具体信息参见CALL的示例。
- 存储过程指定package属性时支持重载。
- 在创建procedure时，不能在avg函数外面嵌套其他agg函数，或者其他系统函数。
- 函数定义时如果指定为IMMUTABLE和SHIPPABLE类型，应该尽量避免函数中存在INSERT，UPDATE，DELETE，MERGE和DDL操作，因为上述操作应该由CN判断对应的执行节点，否则执行结果可能产生错误。
- 存储过程中不支持需要return集合的操作。
- 在存储过程内部调用其它无参数的存储过程时，可以省略括号，直接使用存储过程名进行调用。
- 在存储过程内部调用其他有出参的函数，如果在赋值表达式中调用时，被调函数的出参可以省略，给出了也会被忽略。
- 存储过程支持参数注释的查看与导出、导入。
- 存储过程支持介于IS/AS与plsql_body之间的注释的查看与导出、导入。
- 被授予CREATE ANY FUNCTION权限的用户，可以在用户模式下创建/替换存储过程。
- 存储过程默认为SECURITY INVOKER权限，如果想将默认行为改为SECURITY DEFINER权限，需要设置guc参数behavior_compat_options='plsql_security_definer'。
- 如果将定义者权限的存储过程创建到其他用户SCHEMA下，则会以其他用户的权限执行该存储过程，有越权风险，请谨慎使用。

语法格式

```
openGauss=# CREATE [ OR REPLACE ] PROCEDURE procedure_name
   [ ( {[ argname ] [ argmode ] argtype [ { DEFAULT | := | = } expression ]}[,...]) ]
   [
     { IMMUTABLE | STABLE | VOLATILE }
     | { SHIPPABLE | NOT SHIPPABLE }
     | {PACKAGE}
```



```
      | [ NOT ] LEAKPROOF
      | { CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
      | {[ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER | AUTHID DEFINER | AUTHID
CURRENT_USER}
      | COST execution_cost
      | SET configuration_parameter { [ TO | = ] value | FROM CURRENT }
   ][ ... ]
 { IS | AS }
plsql_body
/
```

参数说明

- **OR REPLACE**

  当存在同名的存储过程时，替换原来的定义。

- **procedure_name**

  创建的存储过程名称，可以带有模式名。

  取值范围：字符串，要符合**标识符命名规范**。

- **argmode**

  参数的模式。

  须知

  VARIADIC用于声明数组类型的参数。

  取值范围： IN，OUT，INOUT或VARIADIC。缺省值是IN。只有OUT模式的参数后面能跟VARIADIC。并且OUT和INOUT模式的参数不能用在RETURNS TABLE的过程定义中。

- **argname**

  参数的名称。

  取值范围：字符串，要符合**标识符命名规范**。

- **argtype**

  参数的数据类型。可以使用%ROWTYPE间接引用表的类型，或者使用%TYPE间接引用表或复合类型中某一列的类型。

  取值范围：可用的数据类型。

- **expression**

  参数的默认表达式。

  说明

  - 推荐使用方式：将所有默认值参数定义在所有非默认值参数后。

- **IMMUTABLE、STABLE等**

  行为约束可选项。各参数的功能与CREATE FUNCTION类似，详细说明见**CREATE FUNCTION**

- **plsql_body**

  PL/SQL存储过程体。

须知

当在存储过程体中进行创建用户、修改密码或加解密等涉及密码或密钥相关操作时，系统表及日志中会记录密码或密钥的明文信息。为防止敏感信息泄露，不建议用户在存储过程体中进行涉及密码或密钥等敏感信息的相关操作。

说明

argname和argmode的顺序没有严格要求，推荐按照argname、argmode、argtype的顺序使用。

示例

```
--创建一个存储过程。
openGauss=# CREATE OR REPLACE PROCEDURE prc_add
(
    param1    IN   INTEGER,
    param2    IN OUT  INTEGER
)
AS
BEGIN
   param2:= param1 + param2;
   dbe_output.print_line('result is: '||to_char(param2));
END;
/

--调用此存储过程。
openGauss=# SELECT prc_add(2,3);

--创建一个参数模式为VARIADIC的存储过程。
openGauss=# CREATE OR REPLACE PROCEDURE pro_variadic (var1 VARCHAR2(10) DEFAULT 'hello!',var4
VARIADIC int4[])
AS
BEGIN
    dbe_output.print_line(var1);
END;
/

--执行此存储过程。
openGauss=# SELECT pro_variadic(var1=>'hello', VARIADIC var4=> array[1,2,3,4]);

--创建一个存储过程，将带着调用它的用户的权限执行。
openGauss=# CREATE TABLE tb1(a integer);
openGauss=#  CREATE PROCEDURE insert_data(v integer)
SECURITY INVOKER
AS
BEGIN
    INSERT INTO tb1 VALUES(v);
END;
/

--调用此存储过程。
openGauss=#  CALL insert_data(1);

--创建带有package属性的存储过程。
openGauss=# create or replace procedure package_func_overload(col int, col2 out varchar)
package
as
declare
    col_type text;
begin
     col2 := '122';
        dbe_output.print_line('two varchar parameters ' || col2);
end;
/
--删除存储过程。
```



```
openGauss=# DROP PROCEDURE prc_add;
openGauss=# DROP PROCEDURE pro_variadic;
openGauss=# DROP PROCEDURE insert_data;
openGauss=# DROP PROCEDURE package_func_overload;
```

相关链接

DROP PROCEDURE

优化建议

- analyse | analyze
  - 不支持在事务或匿名块中执行analyze 。
  - 不支持在函数或存储过程中执行analyze操作。

7.13.58 CREATE RESOURCE LABEL

功能描述

创建资源标签。

注意事项

只有poladmin，sysadmin或初始用户能正常执行此操作。

语法格式

```
CREATE RESOURCE LABEL [IF NOT EXISTS] label_name ADD label_item_list[, ...]*;
```

- label_item_list：
  ```
  resource_type(resource_path[, ...]*)
  ```
- resource_type：
  ```
  TABLE | COLUMN | SCHEMA | VIEW | FUNCTION
  ```

参数说明

- **IF NOT EXISTS**

  如果已经存在相同名称的资源标签，不会抛出错误，而是发出一个通知，告知此资源标签已存在。
- **label_name**

  资源标签名称，创建时要求不能与已有标签重名。

  取值范围：字符串，要符合**标识符命名规范**。
- **resource_type**

  指的是要标记的数据库资源的类型。
- **resource_path**

  指的是描述具体的数据库资源的路径。

示例

```
--创建一个表tb_for_label。
openGauss=# CREATE TABLE tb_for_label(col1 text, col2 text, col3 text);

--创建一个模式schema_for_label。
```



```
openGauss=# CREATE SCHEMA schema_for_label;

--创建一个视图view_for_label。
openGauss=# CREATE VIEW view_for_label AS SELECT 1;

--创建一个函数func_for_label。
openGauss=# CREATE FUNCTION func_for_label RETURNS TEXT AS $$ SELECT col1 FROM tb_for_label; $$
LANGUAGE SQL;

--基于表创建资源标签。
openGauss=# CREATE RESOURCE LABEL IF NOT EXISTS table_label add TABLE(public.tb_for_label);

--基于列创建资源标签。
openGauss=# CREATE RESOURCE LABEL IF NOT EXISTS column_label add
COLUMN(public.tb_for_label.col1);

--基于模式创建资源标签。
openGauss=# CREATE RESOURCE LABEL IF NOT EXISTS schema_label add SCHEMA(schema_for_label);

--基于视图创建资源标签。
openGauss=# CREATE RESOURCE LABEL IF NOT EXISTS view_label add VIEW(view_for_label);

--基于函数创建资源标签。
openGauss=# CREATE RESOURCE LABEL IF NOT EXISTS func_label add FUNCTION(func_for_label);

--删除资源标签。
openGauss=# DROP RESOURCE LABEL func_label, view_label, schema_label, column_label, table_label;

--删除函数func_for_label。
openGauss=# DROP FUNCTION func_for_label;

--删除视图view_for_label。
openGauss=# DROP VIEW view_for_label;

--删除模式schema_for_label。
openGauss=# DROP SCHEMA schema_for_label;

--删除表tb_for_label。
openGauss=# DROP TABLE tb_for_label;
```

相关链接

5.1.13.14.17-ALTER RESOURCE LABEL，**5.1.13.14.102-DROP RESOURCE LABEL**。

7.13.59 CREATE ROLE

功能描述

创建角色。

角色是拥有数据库对象和权限的实体。在不同的环境中角色可以认为是一个用户，一个组或者兼顾两者。

注意事项

- 在数据库中添加一个新角色，角色无登录权限。
- 创建角色的用户必须具备CREATE ROLE的权限或者是系统管理员。

语法格式

```
CREATE ROLE role_name [ [ WITH ] option [ ... ] ] [ ENCRYPTED | UNENCRYPTED ] { PASSWORD |
IDENTIFIED BY } { 'password' [EXPIRED] | DISABLE };
```


其中角色信息设置子句option语法为：

```
{SYSADMIN | NOSYSADMIN}
    | {MONADMIN | NOMONADMIN}
    | {OPRADMIN | NOOPRADMIN}
    | {POLADMIN | NOPOLADMIN}
    | {AUDITADMIN | NOAUDITADMIN}
    | {CREATEDB | NOCREATEDB}
    | {USEFT | NOUSEFT}
    | {CREATEROLE | NOCREATEROLE}
    | {INHERIT | NOINHERIT}
    | {LOGIN | NOLOGIN}
    | {REPLICATION | NOREPLICATION}
    | {VCADMIN | NOVCADMIN}
    | {PERSISTENCE | NOPERSISTENCE}
    | CONNECTION LIMIT connlimit
    | VALID BEGIN 'timestamp'
    | VALID UNTIL 'timestamp'
    | USER GROUP 'groupuser'
    | PERM SPACE 'spacelimit'
    | TEMP SPACE 'tmpspacelimit'
    | SPILL SPACE 'spillspacelimit'
    | NODE GROUP logic_cluster_name
    | IN ROLE role_name [, ...]
    | IN GROUP role_name [, ...]
    | ROLE role_name [, ...]
    | ADMIN rol e_name [, ...]
    | USER role_name [, ...]
    | SYSID uid
    | DEFAULT TABLESPACE tablespace_name
    | PROFILE DEFAULT
    | PROFILE profile_name
    | PGUSER
```

参数说明

- **role_name**

  角色名称。

  取值范围：字符串，要符合**标识符命名规范**，且最多为63个字符。若超过63个字符，数据库会截断并保留前63个字符当做角色名称。当角色名中包含大写字母时数据库会自动转换为小写字母，如果需要创建包含大写字母的角色名则需要使用双引号括起来。

- **password**

  登录密码。

  密码规则如下：

  - 密码默认不少于8个字符。
  - 不能与角色名及角色名倒序相同。
  - 至少包含大写字母（A-Z），小写字母（a-z），数字（0-9），非字母数字字符（限定为~!@#$%^&*()-_=+\|[{}];:,<.>/?）四类字符中的三类字符。当密码中包含的字符不属于上述四种字符范围内时语句执行会报错。
  - 密码也可以是符合格式要求的密文字符串，这种情况主要用于用户数据导入场景，不推荐用户直接使用。如果直接使用密文密码，用户需要知道密文密码对应的明文，并且保证明文密码复杂度，数据库不会校验密文密码复杂度，直接使用密文密码的安全性由用户保证。
  - 创建角色时，应当使用单引号将用户密码括起来。

  取值范围：不为空的字符串。

- **EXPIRED**

在创建用户时可选EXPIRED，即创建密码失效用户，该用户不允许执行简单查询和扩展查询。只有在修改自身密码后才可正常执行语句。

- DISABLE

  默认情况下，用户可以更改自己的密码，除非密码被禁用。要禁用用户的密码，请指定DISABLE。禁用某个用户的密码后，将从系统中删除该密码，此类用户只能通过外部认证来连接数据库，例如：kerberos认证。只有管理员才能启用或禁用密码。普通用户不能禁用初始用户的密码。要启用密码，请运行ALTER USER并指定密码。

- **ENCRYPTED | UNENCRYPTED**

  控制密码存储在系统表里的口令是否加密。按照产品安全要求，密码必须加密存储，所以，UNENCRYPTED在GaussDB中禁止使用。因为系统无法对指定的加密口令字符串进行解密，所以如果目前的口令字符串已经用SHA256加密的格式，则会继续照此存放，而不管是否声明了ENCRYPTED或UNENCRYPTED。这样就允许在dump/restore的时候重新加载加密的口令。

- **SYSADMIN | NOSYSADMIN**

  决定一个新角色是否为“系统管理员”，具有SYSADMIN属性的角色拥有系统最高权限。

  缺省为NOSYSADMIN。

  三权分立打开时，具有SYSADMIN属性的用户无权创建用户。

- **MONADMIN | NOMONADMIN**

  定义角色是否是监控管理员。

  缺省为NOMONADMIN。

- **OPRADMIN | NOOPRADMIN**

  定义角色是否是运维管理员。

  缺省为NOOPRADMIN。

- **POLADMIN | NOPOLADMIN**

  定义角色是否是安全策略管理员。

  缺省为NOPOLADMIN。

- **AUDITADMIN | NOAUDITADMIN**

  定义角色是否有审计管理属性。

  缺省为NOAUDITADMIN。

- **CREATEDB | NOCREATEDB**

  决定一个新角色是否能创建数据库。

  新角色没有创建数据库的权限。

  缺省为NOCREATEDB。

- **USEFT | NOUSEFT**

  该参数为保留参数，暂未启用。

- **CREATEROLE | NOCREATEROLE**

  决定一个角色是否可以创建新角色（也就是执行CREATE ROLE和CREATE USER）。一个拥有CREATEROLE权限的角色也可以修改和删除其他角色。

  缺省为NOCREATEROLE。

  - 三权分立关闭时，具有CREATEROLE属性的用户有权限创建具有CREATEROLE、AUDITADMIN、MONADMIN、POLADMIN、CREATEDB属性的用户和普通用户。

  - 三权分立打开时，具有CREATEROLE属性的用户有权限创建具有CREATEROLE、MONADMIN、POLADMIN、CREATEDB属性的用户和普通用户。

- **INHERIT | NOINHERIT**

  这些子句决定一个角色是否“继承”它所在组的角色的权限。不推荐使用。

- **LOGIN | NOLOGIN**

  具有LOGIN属性的角色才可以登录数据库。一个拥有LOGIN属性的角色可以认为是一个用户。

  缺省为NOLOGIN。

- **REPLICATION | NOREPLICATION**

  定义角色是否允许流复制或设置系统为备份模式。REPLICATION属性是特定的角色，仅用于复制。

  缺省为NOREPLICATION。

- **PERSISTENCE | NOPERSISTENCE**

  定义永久用户。仅允许初始用户创建、修改和删除具有PERSISTENCE属性的永久用户。

- **CONNECTION LIMIT**

  声明该角色可以使用的并发连接数量。

  须知

  - 系统管理员不受此参数的限制。
  - connlimit每个CN单独统计，集群整体的连接数= connlimit * 当前正常CN节点个数。

  取值范围：[-1, 2^31-1]的整数。缺省值为-1，表示没有限制。

- **VALID BEGIN**

  设置角色生效的时间戳。如果省略了该子句，角色无有效开始时间限制。

- **VALID UNTIL**

  设置角色失效的时间戳。如果省略了该子句，角色无有效结束时间限制。

- **USER GROUP 'groupuser'**

  创建一个user的子用户。

- **PERM SPACE**

  设置用户使用空间的大小。

- **TEMP SPACE**

  设置用户临时表存储空间限额。

- **SPILL SPACE**

  设置用户算子落盘空间限额。

- **IN ROLE**

  新角色立即拥有IN ROLE子句中列出的一个或多个现有角色拥有的权限。不推荐使用。

- **IN GROUP**

IN GROUP是IN ROLE过时的拼法。不推荐使用。

- **ROLE**

  ROLE子句列出一个或多个现有的角色，它们将自动添加为这个新角色的成员，拥有新角色所有的权限。

- **ADMIN**

  ADMIN子句类似ROLE子句，不同的是ADMIN后的角色可以把新角色的权限赋给其他角色。

- **USER**

  USER子句是ROLE子句过时的拼法。

- **SYSID**

  SYSID子句将被忽略，无实际意义。

- **DEFAULT TABLESPACE**

  DEFAULT TABLESPACE子句将被忽略，无实际意义。

- **PROFILE**

  PROFILE子句将被忽略，无实际意义。

- **PGUSER**

  当前版本该属性没有实际意义，仅为了语法的前向兼容而保留。

示例

```
--创建一个角色，名为manager，密码为********。
openGauss=# CREATE ROLE manager IDENTIFIED BY '********';

--创建一个角色，从2015年1月1日开始生效，到2026年1月1日失效。
openGauss=# CREATE ROLE miriam WITH LOGIN PASSWORD '********' VALID BEGIN '2015-01-01' VALID
UNTIL '2026-01-01';

--修改角色manager的密码为**********。
openGauss=# ALTER ROLE manager IDENTIFIED BY '**********' REPLACE '********';

--修改角色manager为系统管理员。
openGauss=# ALTER ROLE manager SYSADMIN;

--删除角色manager。
openGauss=# DROP ROLE manager;

--删除角色miriam。
openGauss=# DROP GROUP miriam;
```

相关链接

SET ROLE，**ALTER ROLE**，**DROP ROLE**，**GRANT**，**REVOKE**

7.13.60 CREATE ROW LEVEL SECURITY POLICY

功能描述

对表创建行访问控制策略。

当对表创建了行访问控制策略，只有打开该表的行访问控制开关(ALTER TABLE ... ENABLE ROW LEVEL SECURITY)，策略才能生效。否则不生效。

当前行访问控制影响数据表的读取操作(SELECT、UPDATE、DELETE)，暂不影响数据表的写入操作(INSERT、MERGE INTO)。表所有者或系统管理员可以在USING子句中

创建表达式，在客户端执行数据表读取操作时，数据库后台在查询重写阶段会将满足条件的表达式拼接并应用到执行计划中。针对数据表的每一条元组，当USING表达式返回TRUE时，元组对当前用户可见，当USING表达式返回FALSE或NULL时，元组对当前用户不可见。

行访问控制策略名称是针对表的，同一个数据表上不能有同名的行访问控制策略；对不同的数据表，可以有同名的行访问控制策略。

行访问控制策略可以应用到指定的操作(SELECT、UPDATE、DELETE、ALL)，ALL表示会影响SELECT、UPDATE、DELETE三种操作；定义行访问控制策略时，若未指定受影响的相关操作，默认为ALL。

行访问控制策略可以应用到指定的用户(角色)，也可应用到全部用户(PUBLIC)；定义行访问控制策略时，若未指定受影响的用户，默认为PUBLIC。

注意事项

- 支持对行存表、行存分区表、复制表、unlogged表、hash表定义行访问控制策略。
- 不支持外表、临时表定义行访问控制策略。
- 不支持对视图定义行访问控制策略。
- 同一张表上可以创建多个行访问控制策略，一张表最多创建100个行访问控制策略。
- 系统管理员不受行访问控制影响，可以查看表的全量数据。
- 通过SQL语句、视图、函数、存储过程查询包含行访问控制策略的表，都会受影响。
- 不支持对添加了行级访问控制策略的表字段进行修改数据类型操作。

语法格式

```
CREATE [ ROW LEVEL SECURITY ] POLICY policy_name ON table_name
    [ AS { PERMISSIVE | RESTRICTIVE } ]
    [ FOR { ALL | SELECT | UPDATE | DELETE } ]
    [ TO { role_name | PUBLIC | CURRENT_USER | SESSION_USER } [, ...] ]
    USING ( using_expression )
```

参数说明

- **policy_name**

  行访问控制策略名称，同一个数据表上行访问控制策略名称不能相同。

- **table_name**

  行访问控制策略的表名。

- PERMISSIVE | RESTRICTIVE

  PERMISSIVE指定行访问控制策略为宽容性策略，宽容性策略的条件用OR表达式拼接。RESTRICTIVE指定行访问控制策略为限制性策略，限制性策略的条件用AND表达式拼接。拼接方式如下：

  (using_expression_permissive_1 OR using_expression_permissive_2 ...) AND (using_expression_restrictive_1 AND using_expression_restrictive_2 ...)

  缺省默认为PERMISSIVE。

- **command**

  当前行访问控制影响的SQL操作，可指定操作包括：ALL、SELECT、UPDATE、DELETE。当未指定时，ALL为默认值，涵盖SELECT、UPDATE、DELETE操作。

当command为SELECT时，SELECT类操作受行访问控制的影响，只能查看到满足条件(using_expression返回值为TRUE)的元组数据，受影响的操作包括SELECT，UPDATE ... RETURNING，DELETE ... RETURNING。

当command为UPDATE时，UPDATE类操作受行访问控制的影响，只能更新满足条件(using_expression返回值为TRUE)的元组数据，受影响的操作包括UPDATE, UPDATE ... RETURNING, SELECT ... FOR UPDATE/SHARE。

当command为DELETE时，DELETE类操作受行访问控制的影响，只能删除满足条件(using_expression返回值为TRUE)的元组数据，受影响的操作包括DELETE, DELETE ... RETURNING。

行访问控制策略与适配的SQL语法关系参见下表：

表 7-111 ROW LEVEL SECURITY 策略与适配 SQL 语法关系

Command	SELECT/ALL policy	UPDATE/ALL policy	DELETE/ALL policy
SELECT	Existing row	No	No
SELECT FOR UPDATE/SHARE	Existing row	Existing row	No
UPDATE	No	Existing row	No
UPDATE RETURNING	Existing row	Existing row	No
DELETE	No	No	Existing row
DELETE RETURNING	Existing row	No	Existing row

- **role_name**

  行访问控制影响的数据库用户。

  CURRENT_USER表示当前执行环境的用户名；SESSION_USER则表示会话用户名；当未指定时，PUBLIC为默认值，PUBLIC表示影响所有数据库用户，可以指定多个受影响的数据库用户。

  须知

  系统管理员不受行访问控制特性影响。

- **using_expression**

  行访问控制的表达式（返回boolean值）。

  条件表达式中不能包含AGG函数和窗口（WINDOW）函数。在查询重写阶段，如果数据表的行访问控制开关打开，满足条件的表达式会添加到计划树中。针对数据表的每条元组，会进行表达式计算，只有表达式返回值为TRUE时，行数据对用户才可见（SELECT、UPDATE、DELETE）；当表达式返回FALSE时，该元组对当前用户不可见，用户无法通过SELECT语句查看此元组，无法通过UPDATE语句更新此元组，无法通过DELETE语句删除此元组。

示例

```
--创建用户alice。
openGauss=# CREATE USER alice PASSWORD 'xxxxxxxxx';

--创建用户bob。
openGauss=# CREATE USER bob PASSWORD 'xxxxxxxxx';

--创建数据表all_data。
openGauss=# CREATE TABLE public.all_data(id int, role varchar(100), data varchar(100));

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

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

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

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

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

--当前用户执行SELECT操作。
openGauss=# SELECT * FROM all_data;
 id | role  |    data
----+-------+------------
  1 | alice | alice data
  2 | bob   | bob data
  3 | peter | peter data
(3 rows)

--给用户登录权限。
openGauss=# ALTER USER alice LOGIN;

openGauss=# EXPLAIN(COSTS OFF) SELECT * FROM all_data;
         QUERY PLAN
----------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Seq Scan on all_data
(3 rows)

--切换至alice用户执行SELECT操作。
openGauss=# SELECT * FROM all_data;
 id | role  |    data
----+-------+------------
  1 | alice | alice data
(1 row)

openGauss=# EXPLAIN(COSTS OFF) SELECT * FROM all_data;
```



```
                        QUERY PLAN
------------------------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Seq Scan on all_data
         Filter: ((role)::name = 'alice'::name)
 Notice: This query is influenced by row level security feature
(5 rows)

--删除行访问控制策略。
openGauss=# DROP ROW LEVEL SECURITY POLICY all_data_rls ON all_data;

--删除数据表all_data。
openGauss=# DROP TABLE public.all_data;

--删除用户alice, bob。
openGauss=# DROP USER alice, bob;
```

相关链接

DROP ROW LEVEL SECURITY POLICY

7.13.61 CREATE SCHEMA

功能描述

创建模式。

访问命名对象时可以使用模式名作为前缀进行访问，如果无模式名前缀，则访问当前模式下的命名对象。创建命名对象时也可用模式名作为前缀修饰。

另外，CREATE SCHEMA可以包括在新模式中创建对象的子命令，这些子命令和那些在创建完模式后发出的命令没有任何区别。如果使用了AUTHORIZATION子句，则所有创建的对象都将被该用户所拥有。

注意事项

- 只要用户对当前数据库有CREATE权限，就可以创建模式。
- 系统管理员在普通用户同名schema下创建的对象，所有者为schema的同名用户（非系统管理员）。

语法格式

- 根据指定的名称创建模式。
  ```
  CREATE SCHEMA schema_name
     [ AUTHORIZATION user_name ]  [ schema_element [ ... ] ];
  ```
- 根据用户名创建模式。
  ```
  CREATE SCHEMA AUTHORIZATION user_name [ schema_element [ ... ] ];
  ```

参数说明

- **schema_name**

  模式名称。

须知

模式名不能和当前数据库里其他的模式重名。

模式的名称不可以“pg_”开头。

取值范围：字符串，要符合**标识符命名规范**。

- **AUTHORIZATION user_name**

  指定模式的所有者。当不指定schema_name时，把user_name当作模式名，此时user_name只能是角色名。

  取值范围：已存在的用户名/角色名。

- **schema_element**

  在模式里创建对象的SQL语句。目前仅支持CREATE TABLE、CREATE VIEW、CREATE INDEX、CREATE PARTITION、GRANT子句。

  子命令所创建的对象都被AUTHORIZATION子句指定的用户所拥有。

说明

如果当前搜索路径上的模式中存在同名对象时，需要明确指定引用对象所在的模式。可以通过命令SHOW SEARCH_PATH来查看当前搜索路径上的模式。

示例

```
--创建一个角色role1。
openGauss=# CREATE ROLE role1 IDENTIFIED BY '********';

-- 为用户role1创建一个同名schema，子命令创建的表films和winners的拥有者为role1。
openGauss=# CREATE SCHEMA AUTHORIZATION role1
    CREATE TABLE films (title text, release date, awards text[])
    CREATE VIEW winners AS
    SELECT title, release FROM films WHERE awards IS NOT NULL;

--删除schema。
openGauss=# DROP SCHEMA role1 CASCADE;
--删除用户。
openGauss=# DROP USER role1 CASCADE;
```

相关链接

ALTER SCHEMA，**DROP SCHEMA**

7.13.62 CREATE SEQUENCE

功能描述

CREATE SEQUENCE用于向当前数据库里增加一个新的序列。序列的Owner为创建此序列的用户。

注意事项

- Sequence是一个存放等差数列的特殊表，该表受DBMS控制。这个表没有实际意义，通常用于为行或者表生成唯一的标识符。
- 如果给出一个模式名，则该序列就在给定的模式中创建，否则会在当前模式中创建。序列名必须和同一个模式中的其他序列、表、索引、视图或外表的名称不同。

- 创建序列后，在表中使用序列的nextval()函数和generate_series(1,N)函数对表插入数据，请保证nextval的可调用次数大于等于N+1次，否则会因为generate_series()函数会调用N+1次而导致报错。
- 被授予CREATE ANY SEQUENCE权限的用户，可以在public模式和用户模式下创建序列。

语法格式

```
CREATE SEQUENCE name [ INCREMENT [ BY ] increment ]
    [ MINVALUE minvalue | NO MINVALUE | NOMINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE |
NOMAXVALUE]
    [ START [ WITH ] start ] [ CACHE cache ] [ [ NO ] CYCLE | NOCYCLE ]
    [ OWNED BY { table_name.column_name | NONE } ];
```

参数说明

- **name**

  将要创建的序列名称。

  取值范围: 仅可以使用小写字母（a~z）、 大写字母（A~Z），数字和特殊字符"#"，"_"，"$"的组合。

- **increment**

  指定序列的步长。一个正数将生成一个递增的序列，一个负数将生成一个递减的序列。

  缺省值为1。

- **MINVALUE minvalue | NO MINVALUE| NOMINVALUE**

  执行序列的最小值。如果没有声明minvalue或者声明了NO MINVALUE，则递增序列的缺省值为1，递减序列的缺省值为-2^{63}-1。NOMINVALUE等价于NO MINVALUE

- **MAXVALUE maxvalue | NO MAXVALUE| NOMAXVALUE**

  执行序列的最大值。如果没有声明maxvalue或者声明了NO MAXVALUE，则递增序列的缺省值为2^{63}-1，递减序列的缺省值为-1。NOMAXVALUE等价于NO MAXVALUE

- **start**

  指定序列的起始值。缺省值：对于递增序列为minvalue，递减序列为maxvalue。

- **cache**

  为了快速访问，而在内存中预先存储序列号的个数。

  缺省值为1，表示一次只能生成一个值，也就是没有缓存。

  说明

  - 不建议同时定义cache和maxvalue或minvalue。因为定义cache后不能保证序列的连续性，可能会产生空洞，造成序列号段浪费。如对并发性能有要求，请同时参考guc参数session_sequence_cache。
  - cache指定了单CN/DN一次向GTM中申请的值；session_sequence_cache指定的是单个会话一次向CN/DN申请缓存的值，会话结束后会自动丢弃。

- **CYCLE**

  用于使序列达到maxvalue或者minvalue后可循环并继续下去。

  如果声明了NO CYCLE，则在序列达到其最大值后任何对nextval的调用都会返回一个错误。

NOCYCLE的作用等价于NO CYCLE。

缺省值为NO CYCLE。

若定义序列为CYCLE，则不能保证序列的唯一性。

- **OWNED BY**-

  将序列和一个表的指定字段进行关联。这样，在删除那个字段或其所在表的时候会自动删除已关联的序列。关联的表和序列的所有者必须是同一个用户，并且在同一个模式中。需要注意的是，通过指定OWNED BY，仅仅是建立了表的对应列和sequence之间关联关系，并不会在插入数据时在该列上产生自增序列。

  缺省值为OWNED BY NONE，表示不存在这样的关联。

须知

通过OWNED BY创建的Sequence不建议用于其他表，如果希望多个表共享Sequence，该Sequence不应该从属于特定表。

示例

创建一个名为serial的递增序列，从101开始：

```
openGauss=# CREATE SEQUENCE serial
 START 101
 CACHE 20;
```

从序列中选出下一个数字：

```
openGauss=# SELECT nextval('serial');
 nextval
---------
     101
```

从序列中选出下一个数字：

```
openGauss=# SELECT nextval('serial');
 nextval
---------
     102
```

创建与表关联的序列：

```
openGauss=# CREATE TABLE customer_address
(
    ca_address_sk             integer               not null,
    ca_address_id             char(16)              not null,
    ca_street_number          char(10)                      ,
    ca_street_name            varchar(60)                   ,
    ca_street_type            char(15)                      ,
    ca_suite_number           char(10)                      ,
    ca_city                   varchar(60)                   ,
    ca_county                 varchar(30)                   ,
    ca_state                  char(2)                       ,
    ca_zip                    char(10)                      ,
    ca_country                varchar(20)                   ,
    ca_gmt_offset             decimal(5,2)                  ,
    ca_location_type          char(20)
);

openGauss=# CREATE SEQUENCE serial1
 START 101
 CACHE 20
OWNED BY customer_address.ca_address_sk;
```



```
--删除序列
openGauss=# DROP TABLE customer_address;
openGauss=# DROP SEQUENCE serial cascade;
openGauss=# DROP SEQUENCE serial1 cascade;
```

相关链接

DROP SEQUENCE，**ALTER SEQUENCE**

7.13.63 CREATE SERVER

功能描述

创建一个外部服务器。

外部服务器是存储OBS服务器信息或其他同构集群信息的载体。

注意事项

默认只有系统管理员才可以创建外部服务器，否则需要对所使用的FOREIGN DATA WRAPPER授权才可以创建，授权语法为：

```
GRANT USAGE ON FOREIGN DATA WRAPPER fdw_name TO username
```

其中fdw_name为FOREIGN DATA WRAPPER的名称，username为创建SERVER的用户名。

OPTIONS中的敏感字段（如password、secret_access_key）在使用多层引号时，语义和不带引号的场景是不同的，因此不会被识别为敏感字段进行脱敏。

语法格式

```
CREATE SERVER server_name
   FOREIGN DATA WRAPPER fdw_name
   OPTIONS ( { option_name ' value ' } [, ...] ) ;
```

参数说明

- **server_name**

  server的名称。

  取值范围：长度必须小于等于63。

- **FOREIGN DATA WRAPPER fdw_name**

  指定外部数据封装器的名称。

  取值范围：fdw_name是数据库初始化时系统创建的数据封装器，对于其他同构集群，fdw_name为gc_fdw。还可以创建dist_fdw、file_fdw、log_fdw。

- **OPTIONS ({ option_name ' value ' } [, ...])**

  用于指定外部服务器的各类参数，详细的参数说明如下所示。

  - encrypt

    是否对数据进行加密，该参数仅支持type为OBS时设置。默认值为on。

    取值范围：

    - on表示对数据进行加密，使用HTTPS协议通信。

    - off表示不对数据进行加密，使用HTTP协议通信。
  - access_key

    OBS访问协议对应的AK值（OBS云服务界面由用户获取），创建外表时AK值会加密保存到数据库的元数据表中。该参数仅支持type为OBS时设置。
  - secret_access_key

    OBS访问协议对应的SK值（OBS云服务界面由用户获取），创建外表时SK值会加密保存到数据库的元数据表中。该参数仅支持type为OBS时设置。

示例

建立一个my_server，其中file_fdw为数据库中存在的foreign data wrapper。

```
--创建my_server。
openGauss=# CREATE SERVER my_server FOREIGN DATA WRAPPER file_fdw;

--删除my_server。
openGauss=# DROP SERVER my_server;
```

建立另外一个同构集群的server，其中gc_fdw为数据库中存在的foreign data wrapper。

```
--创建server。
openGauss=# CREATE SERVER server_remote FOREIGN DATA WRAPPER GC_FDW OPTIONS
   (address '10.146.187.231:8000,10.180.157.130:8000' ,
  dbname 'test',
  username 'test',
  password 'xxxxxxxx'
);

--删除server。
openGauss=# DROP SERVER server_remote;
```

相关链接

ALTER SERVER， **DROP SERVER**

7.13.64 CREATE SYNONYM

功能描述

创建一个同义词对象。同义词是数据库对象的别名，用于记录与其他数据库对象名间的映射关系，用户可以使用同义词访问关联的数据库对象。

注意事项

- 定义同义词的用户成为其所有者。
- 若指定模式名称，则同义词在指定模式中创建。否则，在当前模式创建。
- 支持通过同义词访问的数据库对象包括：表、视图、函数和存储过程。
- 使用同义词时，用户需要具有对关联对象的相应权限。
- 支持使用同义词的DML语句包括：SELECT、INSERT、UPDATE、DELETE、EXPLAIN、CALL。
- 不支持关联函数或存储过程的CREATE SYNONYM语句出现在存储过程中，建议存储过程中使用系统表pg_synonym中已存在的同义词对象。

- 不建议对临时表创建同义词。如果需要创建的话，需要指定同义词的目标临时表的模式名，否则无法正常使用该同义词，并且在当前会话结束前执行DROP SYNONYM命令。
- 删除原对象后，与之关联同义词不会被级联删除，继续访问该同义词会报错，并提示已失效。

语法格式

```
CREATE [ OR REPLACE ] SYNONYM synonym_name
    FOR object_name;
```

参数说明

- **synonym**

  创建的同义词名字，可以带模式名。

  取值范围：字符串，要符合**标识符命名规范**。

- **object_name**

  关联的对象名字，可以带模式名。

  取值范围：字符串，要符合**标识符命名规范**。

  说明

  object_name可以是不存在的对象名称。

注意

避免对包含口令等敏感信息的函数，如加解密类函数gs_encrypt、gs_decrypt等创建别名并且使用别名调用，防止敏感信息泄露。

示例

```
--创建模式ot。
openGauss=# CREATE SCHEMA ot;

--创建表ot.t1及其同义词t1。
openGauss=# CREATE TABLE ot.t1(id int, name varchar2(10)) DISTRIBUTE BY hash(id);
openGauss=# CREATE OR REPLACE SYNONYM t1 FOR ot.t1;

--使用同义词t1。
openGauss=# SELECT * FROM t1;
openGauss=# INSERT INTO t1 VALUES (1, 'ada'), (2, 'bob');
openGauss=# UPDATE t1 SET t1.name = 'cici' WHERE t1.id = 2;

--创建同义词v1及其关联视图ot.v_t1。
openGauss=# CREATE SYNONYM v1 FOR ot.v_t1;
openGauss=# CREATE VIEW ot.v_t1 AS SELECT * FROM ot.t1;

--使用同义词v1。
openGauss=# SELECT * FROM v1;

--创建重载函数ot.add及其同义词add。
openGauss=# CREATE OR REPLACE FUNCTION ot.add(a integer, b integer) RETURNS integer AS
$$
SELECT $1 + $2
$$
LANGUAGE sql;

openGauss=# CREATE OR REPLACE FUNCTION ot.add(a decimal(5,2), b decimal(5,2)) RETURNS
```



```
decimal(5,2) AS
$$
SELECT $1 + $2
$$
LANGUAGE sql;

openGauss=# CREATE OR REPLACE SYNONYM add FOR ot.add;

--使用同义词add。
openGauss=# SELECT add(1,2);
openGauss=# SELECT add(1.2,2.3);

--创建存储过程ot.register及其同义词register。
openGauss=# CREATE PROCEDURE ot.register(n_id integer, n_name varchar2(10))
SECURITY INVOKER
AS
BEGIN
    INSERT INTO ot.t1 VALUES(n_id, n_name);
END;
/

openGauss=# CREATE OR REPLACE SYNONYM register FOR ot.register;

--使用同义词register，调用存储过程。
openGauss=# CALL register(3,'mia');

--删除同义词。
openGauss=# DROP SYNONYM t1;
openGauss=# DROP SYNONYM IF EXISTS v1;
openGauss=# DROP SYNONYM IF EXISTS add;
openGauss=# DROP SYNONYM register;
openGauss=# DROP SCHEMA ot CASCADE;
```

相关链接

ALTER SYNONYM，**DROP SYNONYM**

7.13.65 CREATE TABLE

功能描述

在当前数据库中创建一个新的空白表，该表由命令执行者所有。

注意事项

- 表中的主键约束和唯一约束必须包含分布列。
- 分布列不支持更新（UPDATE）操作。
- 如果在建表过程中数据库系统发生故障，系统恢复后可能无法自动清除之前已创建的、大小为0的磁盘文件。此种情况出现概率小，不影响数据库系统的正常运行。
- 使用JDBC时，支持通过PrepareStatement对DEFAULT值进行参数化设置。
- 行存表的表级约束不支持外键。
- 依据并发控制策略，drop table if exist和create if exist操作相同的表并发场景时，有一个会回滚。
- 被授予CREATE ANY TABLE权限的用户，可以在public模式和用户模式下创建表。如果想要创建包含serial类型列的表，还需要授予CREATE ANY SEQUENCE创建序列的权限。

须知

如果在GaussDB中无限创建表，可能会对CN（Coordinator Node）造成以下影响：

- 资源耗尽：每个表都会占用一定的磁盘空间，无限创建表会导致大量的内存和磁盘空间被占用，可能会导致CN的资源耗尽，从而导致系统崩溃或变得不稳定。
- 性能下降：无限创建表会导致大量的I/O操作和CPU计算，数据库的元数据信息将会变得十分庞大，可能会导致CN的性能下降，包括插入、查询、更新和删除等操作，因此导致系统响应变慢或无法满足业务需求。
- 安全问题：过多的表会导致数据库的管理和维护变得困难，无限创建表可能会导致数据泄露或数据丢失等安全问题，数据库的稳定性会降低从而给企业带来不可估量的损失。

因此，对于应该合理规划表的数量和大小，避免无限创建表，从而保证系统的稳定性、可靠性和安全性。

语法格式

- 创建表。

```
CREATE [ [ GLOBAL | LOCAL ] [ TEMPORARY | TEMP ] | UNLOGGED ] TABLE [ IF NOT EXISTS ]
table_name
    ({ column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]
       | table_constraint
       | LIKE source_table [ like_option [...] ] }
       [, ... ])
    [ WITH ( {storage_parameter = value} [, ... ] ) ]
    [ ON COMMIT { PRESERVE ROWS | DELETE ROWS } ]
    [ TABLESPACE tablespace_name ]
    [ DISTRIBUTE BY { REPLICATION | HASH ( column_name [, ...] )
    | RANGE ( column_name [, ...] ) { SLICE REFERENCES tablename | ( slice_less_than_item [, ...] )
    | ( slice_start_end_item [, ...] ) }
    | LIST ( column_name [, ...] ) { SLICE REFERENCES tablename | ( slice_values_item [, ...] ) }
    } ]
    [ TO { GROUP groupname | NODE ( nodename [, ... ] ) } ];
```

  - 其中列约束column_constraint为：

```
[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  CHECK ( expression ) |
  DEFAULT default_expr |
  UNIQUE [ index_parameters ] |
  PRIMARY KEY [ index_parameters ] |
  REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
    [ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

  - 其中表约束table_constraint为：

```
[ CONSTRAINT constraint_name ]
{ CHECK ( expression ) |
  UNIQUE ( column_name [, ... ] ) [ index_parameters ] |
  PRIMARY KEY ( column_name [, ... ] ) [ index_parameters ] |
  PARTIAL CLUSTER KEY ( column_name [, ... ] ) }
[ DEFERRABLE | NOT DEFERRABLE| INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

  - 其中like选项like_option为：

```
{ INCLUDING | EXCLUDING } { DEFAULTS | CONSTRAINTS | INDEXES | STORAGE | COMMENTS |
PARTITION | RELOPTIONS | DISTRIBUTION | ALL }
```

  - 其中RANGE分布规则

    slice_less_than_item为：

SLICE slice_name VALUES LESS THAN ({ literal | MAXVALUE } [, ...])
[DATANODE dn_name]

slice_start_end_item为：

SLICE slice_name_prefix {

{ START (literal) END (literal) EVERY (literal) } |

{ START (literal) END ({ literal | MAXVALUE }) } |

{ START (literal) } |

{ END ({ literal | MAXVALUE }) }

}

– 其中LIST分布规则slice_values_item为：

SLICE slice_name VALUES (list_values_item) [DATANODE dn_name]

list_values_item为：

{ DEFAULT | { partition_values_list [, ...] } }

partition_values_list为:

{ (literal [, ...]) }

其中索引参数index_parameters为：

```
[ WITH ( {storage_parameter = value} [, ... ] ) ]
[ USING INDEX TABLESPACE tablespace_name ]
```

参数说明

- **UNLOGGED**

  如果指定此关键字，则创建的表为非日志表。在非日志表中写入的数据不会被写入到预写日志中，这样就会比普通表快很多。但是在冲突、执行操作系统重启、数据库重启、主备切换、切断电源操作或异常关机导致数据库重启后，非日志表数据会被清空，有数据丢失的风险。非日志表中的内容也不会被复制到备服务器中。在非日志表中创建的索引也不会被自动记录。

  使用场景：非日志表不能保证数据的安全性，用户应该在确保数据已经做好备份的前提下使用，例如系统升级时进行数据的备份。

  故障处理：当异常关机等操作导致非日志表上的索引发生数据丢失时，用户应该对发生错误的索引进行重建。

- **GLOBAL | LOCAL**

  创建临时表时可以在TEMP或TEMPORARY前指定GLOBAL或LOCAL关键字。目前这两个关键字的设立，仅是为了兼容SQL标准，实际上无论指定GLOBAL还是LOCAL，GaussDB都会创建本地临时表。

- **TEMPORARY | TEMP**

  如果指定TEMP或TEMPORARY关键字，则创建的表为临时表。临时表只在当前会话可见，本会话结束后会自动删除。因此，在除当前会话连接的CN以外的其他CN故障时，仍然可以在当前会话上创建和使用临时表。由于临时表只在当前会话创建，对于涉及对临时表操作的DDL语句，会产生DDL失败的报错。因此，建议DDL语句中不要对临时表进行操作。TEMP和TEMPORARY等价。

须知

- 临时表通过每个会话独立的以pg_temp开头的schema来保证只对当前会话可见，因此，不建议用户在日常操作中手动删除以pg_temp，pg_toast_temp开头的schema。
- 如果建表时不指定TEMPORARY/TEMP关键字，而指定表的schema为当前会话的pg_temp_开头的schema，则该表会被创建为临时表。
- 临时表只对当前会话可见，因此不支持与\parallel on并行执行一起使用。
- 临时表不支持DN故障或者主备切换。

- **IF NOT EXISTS**

  如果已经存在相同名称的表，不会报出错误，而会发出通知，告知通知该表已存在。
- **table_name**

  要创建的表名。

须知

物化视图的一些处理逻辑会通过表名的前缀来识别是不是物化视图日志表和物化视图关联表。因此，用户不要创建表名以mlog_或matviewmap_为前缀的表，否则会影响此表的一些功能。

- **column_name**

  新表中要创建的字段名。
- **data_type**

  字段的数据类型。
- **COLLATE collation**

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

  LIKE子句声明一个表，新表自动从这个表中继承所有字段名及其数据类型和非空约束，以及声明为serial的缺省表达式。

  新表与源表之间在创建动作完毕之后是完全无关的。在源表做的任何修改都不会传播到新表中，并且也不可能在扫描源表的时候包含新表的数据。

  被复制的列和约束并不使用相同的名称进行融合。如果明确的指定了相同的名称或者在另外一个LIKE子句中，将会报错。

  - 源表上除serial外的字段缺省表达式只有在指定INCLUDING DEFAULTS时，才会复制到新表中。缺省是不包含缺省表达式的，即新表中的所有字段的缺省值都是NULL。
  - 源表上的CHECK约束仅在指定INCLUDING CONSTRAINTS时，会复制到新表中，而其他类型的约束永远不会复制到新表中。非空约束总是复制到新表中。此规则同时适用于表约束和列约束。
  - 如果指定了INCLUDING INDEXES，则源表上的索引也将在新表上创建，默认不建立索引。

- 如果指定了INCLUDING STORAGE，则复制列的STORAGE设置会复制到新表中，默认情况下不包含STORAGE设置。
- 如果指定了INCLUDING COMMENTS，则源表列、约束和索引的注释会复制到新表中。默认情况下，不复制源表的注释。
- 如果指定了INCLUDING PARTITION，则源表的分区定义会复制到新表中，同时新表将不能再使用PARTITION BY子句。默认情况下，不拷贝源表的分区定义。
- 如果指定了INCLUDING RELOPTIONS，则源表的存储参数（即源表的WITH子句）会复制到新表中。默认情况下，不复制源表的存储参数。
- 如果指定了INCLUDING DISTRIBUTION，则源表的分布信息会复制到新表中，包括分布类型和分布列，同时新表将不能再使用DISTRIBUTE BY子句。默认情况下，不拷贝源表的分布信息。
- INCLUDING ALL包含了INCLUDING DEFAULTS、INCLUDING CONSTRAINTS、INCLUDING INDEXES、INCLUDING STORAGE、INCLUDING COMMENTS、INCLUDING PARTITION、INCLUDING RELOPTIONS和INCLUDING DISTRIBUTION的内容。

须知

- 如果源表包含serial、bigserial、smallseriral类型，或者源表字段的默认值是sequence，且sequence属于源表（通过CREATE SEQUENCE ... OWNED BY创建），这些Sequence不会关联到新表中，新表中会重新创建属于自己的sequence。这和之前版本的处理逻辑不同。如果用户希望源表和新表共享Sequence，需要首先创建一个共享的Sequence（避免使用OWNED BY），并配置为源表字段默认值，这样创建的新表会和源表共享该Sequence。
- 不建议将其他表私有的Sequence配置为源表字段的默认值，尤其是其他表只分布在特定的NodeGroup上，这可能导致CREATE TABLE ... LIKE执行失败。另外，如果源表配置其他表私有的Sequence，当该表删除时Sequence也会连带删除，这样源表的Sequence将不可用。如果用户希望多个表共享Sequence，建议创建共享的Sequence。

- **WITH ({ storage_parameter = value } [, ...])**

  这个子句为表或索引指定一个可选的存储参数。用于表的WITH子句还可以包含OIDS=TRUE或者单独的OIDS来指定给新表中的每一行都分配一个OID（对象标识符），或者OIDS=FALSE表示不分配OID。

  说明

  使用任意精度类型Numeric定义列时，建议指定精度p以及刻度s。在不指定精度和刻度时，会按输入的显示出来。

  参数的详细描述如下所示。

  - FILLFACTOR

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

    取值范围：10~100

- ORIENTATION

  指定表数据的存储方式，即行存方式，该参数设置成功后就不再支持修改。

  取值范围：

  - ROW，表示表的数据将以行式存储。

    行存储适合于OLTP业务，此类型的表上交互事务比较多，一次交互会涉及表中的多个列，用行存查询效率较高。

  默认值：

  若指定表空间为普通表空间，默认值为ROW。

- segment

  使用段页式的方式存储。本参数仅支持行存表。不支持临时表、unlog表。不支持ustore存储引擎。

  取值范围：on/off

  默认值：off

- hashbucket

  创建hash bucket存储。本参数仅支持行存表和行存range表。

  取值范围：on/off

  默认值：off

须知

- 当前版本hashbucket表相关DDL操作性能受限，不建议频繁对hashbucket表进行DDL操作。
- hashbucket表绑定段页式存储，即hashbucket=on隐含segment=on。

- bucketcnt

  创建bucket表时，指定表的bucket数目。该参数指定的值必须有和其对应的Child Node Group存在。

  取值范围：32 ~ 16384,，且是2的整数次方。

  默认值：16384。

- parallel_workers

  表示创建索引时起的bgworker线程数量，例如2就表示将会起2个bgworker线程并发创建索引。

  取值范围：[0,32]，int类型，0表示关闭该功能。

  默认值：不设置该参数，表示未开启并行建索引功能。

- hasuids

  参数开启：更新表元组时，为元组分配表级唯一标识id。

  取值范围：on/off。

  默认值：off。

- **WITHOUT OIDS**

  等价于WITH（OIDS=FALSE）的语法。

- **ON COMMIT { PRESERVE ROWS | DELETE ROWS }**

ON COMMIT选项决定在事务中执行创建临时表操作，当事务提交时，此临时表的后续操作，当前支持PRESERVE ROWS和DELETE ROWS选项。

- PRESERVE ROWS（缺省值）：提交时不对临时表做任何操作，临时表及其表数据保持不变。
- DELETE ROWS：提交时删除临时表中数据。

- **TABLESPACE tablespace_name**

  创建新表时指定此关键字，表示新表将要在指定表空间内创建。如果没有声明，将使用默认表空间。

- **DISTRIBUTE BY**

  指定表如何在节点之间分布或者复制。

  取值范围：

  - REPLICATION：表的每一行存在所有数据节点（DN）中，即每个数据节点都有完整的表数据。
  - HASH（column_name）：对指定的列进行Hash，通过映射，把数据分布到指定DN。
  - RANGE(column_name) 对指定列按照范围进行映射，把数据分布到对应DN。
  - LIST(column_name) 对指定列按照具体值进行映射，把数据分布到对应DN。

    说明

    - 当指定DISTRIBUTE BY { HASH | RANGE | LIST } (column_name)参数时，创建主键和唯一索引必须包含“ column_name”列。
    - 对于从句是VALUE LESS THAN语法格式的RANGE分布策略，分布键最多支持4列。分布规则如下：

      1. 从插入值的第一列开始比较。
      2. 如果插入值的第一列小于待插入的分片的当前列的边界值，则直接插入。
      3. 如果插入值的第一列等于待插入的分片的当前列的边界值，则比较插入值的下一列与待插入的分片的下一列的边界值，如果小于，则直接插入。如果相等，继续比较插入值的下一列与待插入的分片的下一列的边界值，直至小于并插入。
      4. 如果插入值的所有列大于待插入的分片的所有列的边界值，则比较下一分片。

  默认值：HASH(column_name)，column_name取表的主键列（如果有的话）或首个数据类型支持作为分布列的列。

  column_name的数据类型必须是以下类型之一：

  - INTEGER TYPES：TINYINT，SMALLINT，INT，BIGINT，NUMERIC/DECIMAL
  - CHARACTER TYPES：CHAR，BPCHAR，VARCHAR，VARCHAR2，NVARCHAR2，TEXT
  - DATE/TIME TYPES：DATE，TIME，TIMETZ，TIMESTAMP，TIMESTAMPTZ，INTERVAL，SMALLDATETIME

说明

在建表时，选择分布列和分区键可对SQL查询性能产生重大影响。因此，需要根据一定策略选择合适的分布列和分区键。

- 选择合适的分布列

  对于采用散列（Hash）方式的数据分布表，一个合适的分布列应将一个表内的数据，均匀分散存储在多个DN内，避免出现数据倾斜现象（即多个DN内数据分布不均）。请按照如下原则判定合适的分布列：

  1. 判断是否已发生数据倾斜现象。

     连接数据库，执行如下语句，查看各DN内元组数目。命令中的斜体部分tablename，请填入待分析的表名。

     ```
     openGauss=# SELECT a.count,b.node_name FROM (SELECT count(*) AS count,xc_node_id FROM tablename GROUP BY xc_node_id) a, pgxc_node b WHERE a.xc_node_id=b.node_id ORDER BY a.count DESC;
     ```

     如果各DN内元组数目相差较大（如相差数倍、数十倍），则表明已发生数据倾斜现象，请按照下面原则调整分布列。

  2. 重新选择分布列，重新建表。当前不支持通过ALTER TABLE语句调整分布列，因此调整分布列时需要重新建表。

     选择原则如下：

     分布列的列值应比较离散，以便数据能够均匀分布到各个DN。例如，考虑选择表的主键为分布列，如在人员信息表中选择身份证号码为分布列。

     在满足上面原则的情况下，考虑选择查询中的连接条件为分布列，以便Join任务能够下推到DN中执行，且减少DN之间的通信数据量。

- 选择合适的分区键

  数据分区功能，可根据表的一列或者多列，将要插入表的记录分为若干个范围（这些范围在不同的分区里没有重叠）。然后为每个范围创建一个分区，用来存储相应的数据。

  调整分区键，使每次查询结果尽可能存储在相同或者最少的分区内（称为“分区剪枝”），通过获取连续I/O大幅度提升查询性能。

  实际业务中，经常将时间作为查询对象的过滤条件，因此，可考虑选择时间列为分区键，键值范围可根据总数据量、一次查询数据量调整。

- RANGE/LIST分布

  当没有为RANGE/LIST分布表的分片显示指定DN时，数据库内部为分片分配DN是采用roundrobin的算法。另外，在使用RANGE/LIST分布的场景中，考虑到后续扩容的需要，建议用户在建表时定义尽可能多的分片数，因为如果定义的分片数小于扩容前的DN节点数，数据重分布时则无法落入新的DN节点。需要特别注意的是，由于是由用户自行设计分片规则，在某些极端情况下，扩容也可能无法解决存储空间不足的问题。

- **TO { GROUP groupname | NODE (nodename [, ...]) }**

  TO GROUP指定创建表所在的Node Group。TO NODE主要供内部扩容工具使用，一般用户不应该使用。

- **CONSTRAINT constraint_name**

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

  定义约束有两种方法：

  - 列约束：作为一个列定义的一部分，仅影响该列。
  - 表约束：不和某个列绑在一起，可以作用于多个列。

- **NOT NULL**

  字段值不允许为NULL。

- **NULL**

  字段值允许为NULL ，这是缺省值。

这个子句只是为和非标准SQL数据库兼容。不建议使用。

- **CHECK (expression)**

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

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

  说明

  expression表达式中，如果存在“<>NULL”或“！=NULL”，这种写法是无效的，需要写成“is NOT NULL”。

- **DEFAULT default_expr**

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

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

- **UNIQUE index_parameters**

  UNIQUE (column_name [, ...]) index_parameters

  UNIQUE约束表示表里的一个字段或多个字段的组合必须在全表范围内唯一。

  对于唯一约束，NULL被认为是互不相等的。

  说明

  如果没有声明DISTRIBUTE BY REPLICATION，则唯一约束的列集合中必须包含分布列。

- **PRIMARY KEY index_parameters**

  PRIMARY KEY (column_name [, ...]) index_parameters

  主键约束声明表中的一个或者多个字段只能包含唯一的非NULL值。

  一个表只能声明一个主键。

  说明

  如果没有声明DISTRIBUTE BY REPLICATION，则主键约束的列集合中必须包含分布列。

- **REFERENCES**

  当前版本分布式数据库暂不支持REFERENCES子句。

- **DEFERRABLE | NOT DEFERRABLE**

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

- **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

  如果约束是可推迟的，则这个子句声明检查约束的缺省时间。

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

  约束检查的时间可以用SET CONSTRAINTS命令修改。

- **USING INDEX TABLESPACE tablespace_name**

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

示例

```
--创建简单的表。
openGauss=# CREATE TABLE tpcds.warehouse_t1
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
);

openGauss=# CREATE TABLE tpcds.warehouse_t2
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)         DICTIONARY,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
);
--创建表，并指定W_STATE字段的缺省值为GA。
openGauss=# CREATE TABLE tpcds.warehouse_t3
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)           DEFAULT 'GA',
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
);

--创建表，并在事务结束时检查W_WAREHOUSE_NAME字段是否有重复。
openGauss=# CREATE TABLE tpcds.warehouse_t4
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)   UNIQUE DEFERRABLE,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
```



```
    W_STREET_NUMBER           CHAR(10)                    ,
    W_STREET_NAME             VARCHAR(60)                 ,
    W_STREET_TYPE             CHAR(15)                   ,
    W_SUITE_NUMBER            CHAR(10)                    ,
    W_CITY                    VARCHAR(60)                ,
    W_COUNTY                  VARCHAR(30)                 ,
    W_STATE                   CHAR(2)                    ,
    W_ZIP                     CHAR(10)                   ,
    W_COUNTRY                 VARCHAR(20)                 ,
    W_GMT_OFFSET              DECIMAL(5,2)
);
--创建一个带有70%填充因子的表。
openGauss=# CREATE TABLE tpcds.warehouse_t5
(
    W_WAREHOUSE_SK            INTEGER                NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)               NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT         INTEGER                     ,
    W_STREET_NUMBER           CHAR(10)                    ,
    W_STREET_NAME             VARCHAR(60)                 ,
    W_STREET_TYPE             CHAR(15)                   ,
    W_SUITE_NUMBER            CHAR(10)                    ,
    W_CITY                    VARCHAR(60)                ,
    W_COUNTY                  VARCHAR(30)                 ,
    W_STATE                   CHAR(2)                    ,
    W_ZIP                     CHAR(10)                   ,
    W_COUNTRY                 VARCHAR(20)                 ,
    W_GMT_OFFSET              DECIMAL(5,2),
    UNIQUE(W_WAREHOUSE_NAME) WITH(fillfactor=70)
);

--或者用下面的语法。
openGauss=# CREATE TABLE tpcds.warehouse_t6
(
    W_WAREHOUSE_SK            INTEGER                NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)               NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)            UNIQUE,
    W_WAREHOUSE_SQ_FT         INTEGER                     ,
    W_STREET_NUMBER           CHAR(10)                    ,
    W_STREET_NAME             VARCHAR(60)                 ,
    W_STREET_TYPE             CHAR(15)                   ,
    W_SUITE_NUMBER            CHAR(10)                    ,
    W_CITY                    VARCHAR(60)                ,
    W_COUNTY                  VARCHAR(30)                 ,
    W_STATE                   CHAR(2)                    ,
    W_ZIP                     CHAR(10)                   ,
    W_COUNTRY                 VARCHAR(20)                 ,
    W_GMT_OFFSET              DECIMAL(5,2)
) WITH(fillfactor=70);

--创建表，并指定该表数据不写入预写日志。
openGauss=# CREATE UNLOGGED TABLE tpcds.warehouse_t7
(
    W_WAREHOUSE_SK            INTEGER              NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)             NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT         INTEGER                     ,
    W_STREET_NUMBER           CHAR(10)                    ,
    W_STREET_NAME             VARCHAR(60)                 ,
    W_STREET_TYPE             CHAR(15)                   ,
    W_SUITE_NUMBER            CHAR(10)                    ,
    W_CITY                    VARCHAR(60)                ,
    W_COUNTY                  VARCHAR(30)                 ,
    W_STATE                   CHAR(2)                    ,
    W_ZIP                     CHAR(10)                   ,
    W_COUNTRY                 VARCHAR(20)                 ,
    W_GMT_OFFSET              DECIMAL(5,2)
);
```



```
--创建表临时表。
openGauss=# CREATE TEMPORARY TABLE warehouse_t24
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME            VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT           INTEGER                     ,
    W_STREET_NUMBER           CHAR(10)                    ,
    W_STREET_NAME             VARCHAR(60)                 ,
    W_STREET_TYPE             CHAR(15)                    ,
    W_SUITE_NUMBER              CHAR(10)                  ,
    W_CITY                    VARCHAR(60)                 ,
    W_COUNTY                    VARCHAR(30)               ,
    W_STATE                   CHAR(2)                     ,
    W_ZIP                     CHAR(10)                    ,
    W_COUNTRY                   VARCHAR(20)               ,
    W_GMT_OFFSET                DECIMAL(5,2)
);

--事务中创建表临时表，并指定提交事务时删除该临时表数据。
openGauss=# CREATE TEMPORARY TABLE warehouse_t25
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME            VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT           INTEGER                     ,
    W_STREET_NUMBER           CHAR(10)                    ,
    W_STREET_NAME             VARCHAR(60)                 ,
    W_STREET_TYPE             CHAR(15)                    ,
    W_SUITE_NUMBER              CHAR(10)                  ,
    W_CITY                    VARCHAR(60)                 ,
    W_COUNTY                    VARCHAR(30)               ,
    W_STATE                   CHAR(2)                     ,
    W_ZIP                     CHAR(10)                    ,
    W_COUNTRY                   VARCHAR(20)               ,
    W_GMT_OFFSET                DECIMAL(5,2)
) ON COMMIT DELETE ROWS;

--创建表时，不希望因为表已存在而报错。
openGauss=# CREATE TABLE IF NOT EXISTS tpcds.warehouse_t8
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME            VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT           INTEGER                     ,
    W_STREET_NUMBER           CHAR(10)                    ,
    W_STREET_NAME             VARCHAR(60)                 ,
    W_STREET_TYPE             CHAR(15)                    ,
    W_SUITE_NUMBER              CHAR(10)                  ,
    W_CITY                    VARCHAR(60)                 ,
    W_COUNTY                    VARCHAR(30)               ,
    W_STATE                   CHAR(2)                     ,
    W_ZIP                     CHAR(10)                    ,
    W_COUNTRY                   VARCHAR(20)               ,
    W_GMT_OFFSET                DECIMAL(5,2)
);

--创建普通表空间。
openGauss=# CREATE TABLESPACE DS_TABLESPACE1 RELATIVE LOCATION 'tablespace/tablespace_1';
--创建表时，指定表空间。
openGauss=# CREATE TABLE tpcds.warehouse_t9
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME            VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT           INTEGER                     ,
    W_STREET_NUMBER           CHAR(10)                    ,
    W_STREET_NAME             VARCHAR(60)                 ,
    W_STREET_TYPE             CHAR(15)                    ,
```



```
    W_SUITE_NUMBER            CHAR(10)                    ,
    W_CITY                    VARCHAR(60)                 ,
    W_COUNTY                  VARCHAR(30)                 ,
    W_STATE                   CHAR(2)                     ,
    W_ZIP                     CHAR(10)                    ,
    W_COUNTRY                 VARCHAR(20)                 ,
    W_GMT_OFFSET              DECIMAL(5,2)
) TABLESPACE DS_TABLESPACE1;

--创建表时，单独指定W_WAREHOUSE_NAME的索引表空间。
openGauss=# CREATE TABLE tpcds.warehouse_t10
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)           UNIQUE USING INDEX TABLESPACE
DS_TABLESPACE1,
    W_WAREHOUSE_SQ_FT         INTEGER                     ,
    W_STREET_NUMBER           CHAR(10)                    ,
    W_STREET_NAME             VARCHAR(60)                 ,
    W_STREET_TYPE             CHAR(15)                    ,
    W_SUITE_NUMBER            CHAR(10)                    ,
    W_CITY                    VARCHAR(60)                 ,
    W_COUNTY                  VARCHAR(30)                 ,
    W_STATE                   CHAR(2)                     ,
    W_ZIP                     CHAR(10)                    ,
    W_COUNTRY                 VARCHAR(20)                 ,
    W_GMT_OFFSET              DECIMAL(5,2)
);
--创建一个有主键约束的表。
openGauss=# CREATE TABLE tpcds.warehouse_t11
(
    W_WAREHOUSE_SK            INTEGER             PRIMARY KEY,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT         INTEGER                     ,
    W_STREET_NUMBER           CHAR(10)                    ,
    W_STREET_NAME             VARCHAR(60)                 ,
    W_STREET_TYPE             CHAR(15)                    ,
    W_SUITE_NUMBER            CHAR(10)                    ,
    W_CITY                    VARCHAR(60)                 ,
    W_COUNTY                  VARCHAR(30)                 ,
    W_STATE                   CHAR(2)                     ,
    W_ZIP                     CHAR(10)                    ,
    W_COUNTRY                 VARCHAR(20)                 ,
    W_GMT_OFFSET              DECIMAL(5,2)
);

---或者使用下面的语法，效果完全一样。
openGauss=# CREATE TABLE tpcds.warehouse_t12
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT         INTEGER                     ,
    W_STREET_NUMBER           CHAR(10)                    ,
    W_STREET_NAME             VARCHAR(60)                 ,
    W_STREET_TYPE             CHAR(15)                    ,
    W_SUITE_NUMBER            CHAR(10)                    ,
    W_CITY                    VARCHAR(60)                 ,
    W_COUNTY                  VARCHAR(30)                 ,
    W_STATE                   CHAR(2)                     ,
    W_ZIP                     CHAR(10)                    ,
    W_COUNTRY                 VARCHAR(20)                 ,
    W_GMT_OFFSET              DECIMAL(5,2),
    PRIMARY KEY(W_WAREHOUSE_SK)
);

--或使用下面的语法，指定约束的名称。
openGauss=# CREATE TABLE tpcds.warehouse_t13
```



```
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2),
    CONSTRAINT W_CSTR_KEY1 PRIMARY KEY(W_WAREHOUSE_SK)
);

--创建一个有复合主键约束的表。
openGauss=# CREATE TABLE tpcds.warehouse_t14
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2),
    CONSTRAINT W_CSTR_KEY2 PRIMARY KEY(W_WAREHOUSE_SK, W_WAREHOUSE_ID)
);

--定义一个检查列约束。
openGauss=# CREATE TABLE tpcds.warehouse_t19
(
    W_WAREHOUSE_SK            INTEGER               PRIMARY KEY CHECK (W_WAREHOUSE_SK > 0),
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)           CHECK (W_WAREHOUSE_NAME IS NOT NULL),
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
);

openGauss=# CREATE TABLE tpcds.warehouse_t20
(
    W_WAREHOUSE_SK            INTEGER               PRIMARY KEY,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)           CHECK (W_WAREHOUSE_NAME IS NOT NULL),
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
```



```
    W_STATE                 CHAR(2)                 ,
    W_ZIP                   CHAR(10)                ,
    W_COUNTRY               VARCHAR(20)             ,
    W_GMT_OFFSET            DECIMAL(5,2),
    CONSTRAINT W_CONSTR_KEY2 CHECK(W_WAREHOUSE_SK > 0 AND W_WAREHOUSE_NAME IS NOT
NULL)
);

--定义一个表，表中每一个行存在所有DN中。
openGauss=# CREATE TABLE tpcds.warehouse_t21
(
    W_WAREHOUSE_SK          INTEGER               NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT       INTEGER                       ,
    W_STREET_NUMBER         CHAR(10)                      ,
    W_STREET_NAME           VARCHAR(60)                   ,
    W_STREET_TYPE           CHAR(15)                      ,
    W_SUITE_NUMBER          CHAR(10)                      ,
    W_CITY                  VARCHAR(60)                   ,
    W_COUNTY                VARCHAR(30)                   ,
    W_STATE                 CHAR(2)                       ,
    W_ZIP                   CHAR(10)                      ,
    W_COUNTRY               VARCHAR(20)                   ,
    W_GMT_OFFSET            DECIMAL(5,2)
)DISTRIBUTE BY REPLICATION;

打开复制表primarynode 选项
openGauss=# ALTER TABLE tpcds.warehouse_t21 SET (primarynode=on);

查看是否打开选项(Options 显示的内容不同版本略有区别)
openGauss=# \d+ tpcds.warehouse_t21
                            Table "tpcds.warehouse_t21"
     Column       |         Type          | Modifiers | Storage  | Stats target | Description
------------------+-----------------------+-----------+----------+--------------+-------------
 w_warehouse_sk    | integer               | not null  | plain    |              |
 w_warehouse_id    | character(16)         | not null  | extended |              |
 w_warehouse_name  | character varying(20) |           | extended |              |
 w_warehouse_sq_ft | integer               |           | plain    |              |
 w_street_number   | character(10)         |           | extended |              |
 w_street_name     | character varying(60) |           | extended |              |
 w_street_type     | character(15)         |           | extended |              |
 w_suite_number    | character(10)         |           | extended |              |
 w_city            | character varying(60) |           | extended |              |
 w_county          | character varying(30) |           | extended |              |
 w_state           | character(2)          |           | extended |              |
 w_zip             | character(10)         |           | extended |              |
 w_country         | character varying(20) |           | extended |              |
 w_gmt_offset      | numeric(5,2)          |           | main     |              |
Has OIDs: no
Distribute By: REPLICATION
Location Nodes: ALL DATANODES
Options: primarynode=on

--定义一个表，使用HASH分布。
openGauss=# CREATE TABLE tpcds.warehouse_t22
(
    W_WAREHOUSE_SK          INTEGER               NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT       INTEGER                       ,
    W_STREET_NUMBER         CHAR(10)                      ,
    W_STREET_NAME           VARCHAR(60)                   ,
    W_STREET_TYPE           CHAR(15)                      ,
    W_SUITE_NUMBER          CHAR(10)                      ,
    W_CITY                  VARCHAR(60)                   ,
    W_COUNTY                VARCHAR(30)                   ,
    W_STATE                 CHAR(2)                       ,
    W_ZIP                   CHAR(10)                      ,
```

```
    W_COUNTRY                 VARCHAR(20)                 ,
    W_GMT_OFFSET              DECIMAL(5,2),
    CONSTRAINT W_CONSTR_KEY3 UNIQUE(W_WAREHOUSE_SK)
)DISTRIBUTE BY HASH(W_WAREHOUSE_SK);

--查看DN信息
openGauss=# select node_name from pgxc_node;
  node_name
--------------
 coordinator1
 datanode1
 datanode2
 datanode3
 datanode4
 datanode5
 datanode6
(7 rows)

--定义一个表，使用RANGE分布
openGauss=# CREATE TABLE tpcds.warehouse_t26
(
W_WAREHOUSE_SK            INTEGER               NOT NULL,
W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
W_WAREHOUSE_NAME            VARCHAR(20)                 ,
W_WAREHOUSE_SQ_FT         INTEGER                     ,
W_STREET_NUMBER           CHAR(10)                    ,
W_STREET_NAME             VARCHAR(60)                 ,
W_STREET_TYPE             CHAR(15)                  ,
W_SUITE_NUMBER            CHAR(10)                    ,
W_CITY                    VARCHAR(60)               ,
W_COUNTY                  VARCHAR(30)                 ,
W_STATE                   CHAR(2)                   ,
W_ZIP                     CHAR(10)                  ,
W_COUNTRY                 VARCHAR(20)                 ,
W_GMT_OFFSET              DECIMAL(5,2)
)DISTRIBUTE BY RANGE(W_WAREHOUSE_ID)
(
   SLICE s1 VALUES LESS THAN (10) DATANODE datanode1,
   SLICE s2 VALUES LESS THAN (20) DATANODE datanode2,
   SLICE s3 VALUES LESS THAN (30) DATANODE datanode3,
   SLICE s4 VALUES LESS THAN (MAXVALUE) DATANODE datanode4
);

--多列RANGE分区策略示例
openGauss=# create table t_ran1(c1 int, c2 int, c3 int, c4 int, c5 int)
distribute by range(c1,c2)
(
SLICE s1 VALUES LESS THAN (10,10) DATANODE datanode1,
SLICE s2 VALUES LESS THAN (10,20) DATANODE datanode2,
SLICE s3 VALUES LESS THAN (20,10) DATANODE datanode3
);
openGauss=# insert into t_ran1 values(9,5,'a');
openGauss=# insert into t_ran1 values(9,20,'a');
openGauss=# insert into t_ran1 values(9,21,'a');
openGauss=# insert into t_ran1 values(10,5,'a');
openGauss=# insert into t_ran1 values(10,15,'a');
openGauss=# insert into t_ran1 values(10,20,'a');
openGauss=# insert into t_ran1 values(10,21,'a');
openGauss=# insert into t_ran1 values(11,5,'a');
openGauss=# insert into t_ran1 values(11,20,'a');
openGauss=# insert into t_ran1 values(11,21,'a');
openGauss=# select node_name,node_type,node_id from pgxc_node;
  node_name   | node_type |    node_id
--------------+-----------+-------------
 coordinator1 | C         |  1938253334
 datanode1    | D         |   888802358
 datanode2    | D         |  -905831925
 datanode3    | D         | -1894792127
(4 rows)
```



```
openGauss=# select xc_node_id,* from t_ran1;
 xc_node_id  | c1 | c2 | c3 | c4 | c5
-------------+----+----+----+----+----
   888802358 |  9 |  5 |  0 |    |
   888802358 |  9 | 20 |  0 |    |
   888802358 |  9 | 21 |  0 |    |
   888802358 | 10 |  5 |  0 |    |
  -905831925 | 10 | 15 |  0 |    |
 -1894792127 | 10 | 20 |  0 |    |
 -1894792127 | 10 | 21 |  0 |    |
 -1894792127 | 11 |  5 |  0 |    |
 -1894792127 | 11 | 20 |  0 |    |
 -1894792127 | 11 | 21 |  0 |    |
(10 rows)

--利用SLICE REFERENCES建表
openGauss=# CREATE TABLE tpcds.warehouse_t27
(
W_WAREHOUSE_SK            INTEGER               NOT NULL,
W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
W_WAREHOUSE_NAME            VARCHAR(20)                 ,
W_WAREHOUSE_SQ_FT         INTEGER                     ,
W_STREET_NUMBER           CHAR(10)                    ,
W_STREET_NAME             VARCHAR(60)                 ,
W_STREET_TYPE             CHAR(15)                    ,
W_SUITE_NUMBER            CHAR(10)                    ,
W_CITY                    VARCHAR(60)                 ,
W_COUNTY                  VARCHAR(30)                 ,
W_STATE                   CHAR(2)                     ,
W_ZIP                     CHAR(10)                    ,
W_COUNTRY                 VARCHAR(20)                 ,
W_GMT_OFFSET              DECIMAL(5,2)
)DISTRIBUTE BY RANGE(W_WAREHOUSE_ID) SLICE REFERENCES warehouse_t26;

--定义一个表，使用LIST分布
openGauss=# CREATE TABLE tpcds.warehouse_t28
(
W_WAREHOUSE_SK            INTEGER               NOT NULL,
W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
W_WAREHOUSE_NAME            VARCHAR(20)                 ,
W_WAREHOUSE_SQ_FT         INTEGER                     ,
W_STREET_NUMBER           CHAR(10)                    ,
W_STREET_NAME             VARCHAR(60)                 ,
W_STREET_TYPE             CHAR(15)                    ,
W_SUITE_NUMBER            CHAR(10)                    ,
W_CITY                    VARCHAR(60)                 ,
W_COUNTY                  VARCHAR(30)                 ,
W_STATE                   CHAR(2)                     ,
W_ZIP                     CHAR(10)                    ,
W_COUNTRY                 VARCHAR(20)                 ,
W_GMT_OFFSET              DECIMAL(5,2)
)DISTRIBUTE BY LIST(W_COUNTRY)
(
    SLICE s1 VALUES ('USA') DATANODE datanode1,
    SLICE s2 VALUES ('CANADA') DATANODE datanode2,
    SLICE s3 VALUES ('UK') DATANODE datanode3,
    SLICE s4 VALUES (DEFAULT) DATANODE datanode4
);
--向tpcds.warehouse_t19表中增加一个varchar列。
openGauss=# ALTER TABLE tpcds.warehouse_t19 ADD W_GOODS_CATEGORY varchar(30);

--给tpcds.warehouse_t19表增加一个检查约束。
openGauss=# ALTER TABLE tpcds.warehouse_t19 ADD CONSTRAINT W_CONSTR_KEY4 CHECK (W_STATE IS
NOT NULL);

--在一个操作中改变两个现存字段的类型。
openGauss=# ALTER TABLE tpcds.warehouse_t19
    ALTER COLUMN W_GOODS_CATEGORY TYPE varchar(80),
    ALTER COLUMN W_STREET_NAME TYPE varchar(100);
```



```
--此语句与上面语句等效。
openGauss=# ALTER TABLE tpcds.warehouse_t19 MODIFY (W_GOODS_CATEGORY varchar(30),
W_STREET_NAME varchar(60));

--给一个已存在字段添加非空约束。
openGauss=# ALTER TABLE tpcds.warehouse_t19 ALTER COLUMN W_GOODS_CATEGORY SET NOT NULL;

--移除已存在字段的非空约束。
openGauss=# ALTER TABLE tpcds.warehouse_t19 ALTER COLUMN W_GOODS_CATEGORY DROP NOT NULL;

--将表移动到另一个表空间。
openGauss=# ALTER TABLE tpcds.warehouse_t19 SET TABLESPACE PG_DEFAULT;
--创建模式joe。
openGauss=# CREATE SCHEMA joe;

--将表移动到另一个模式中。
openGauss=# ALTER TABLE tpcds.warehouse_t19 SET SCHEMA joe;

--重命名已存在的表。
openGauss=# ALTER TABLE joe.warehouse_t19 RENAME TO warehouse_t23;

--从warehouse_t23表中删除一个字段。
openGauss=# ALTER TABLE joe.warehouse_t23 DROP COLUMN W_STREET_NAME;


--删除表空间、模式joe和模式表warehouse。
openGauss=# DROP TABLE tpcds.warehouse_t1;
openGauss=# DROP TABLE tpcds.warehouse_t2;
openGauss=# DROP TABLE tpcds.warehouse_t3;
openGauss=# DROP TABLE tpcds.warehouse_t4;
openGauss=# DROP TABLE tpcds.warehouse_t5;
openGauss=# DROP TABLE tpcds.warehouse_t6;
openGauss=# DROP TABLE tpcds.warehouse_t7;
openGauss=# DROP TABLE tpcds.warehouse_t8;
openGauss=# DROP TABLE tpcds.warehouse_t9;
openGauss=# DROP TABLE tpcds.warehouse_t10;
openGauss=# DROP TABLE tpcds.warehouse_t11;
openGauss=# DROP TABLE tpcds.warehouse_t12;
openGauss=# DROP TABLE tpcds.warehouse_t13;
openGauss=# DROP TABLE tpcds.warehouse_t14;
openGauss=# DROP TABLE tpcds.warehouse_t15;
openGauss=# DROP TABLE tpcds.warehouse_t16;
openGauss=# DROP TABLE tpcds.warehouse_t17;
openGauss=# DROP TABLE tpcds.warehouse_t18;
openGauss=# DROP TABLE tpcds.warehouse_t20;
openGauss=# DROP TABLE tpcds.warehouse_t21;
openGauss=# DROP TABLE tpcds.warehouse_t22;
openGauss=# DROP TABLE joe.warehouse_t23;
openGauss=# DROP TABLE tpcds.warehouse_t24;
openGauss=# DROP TABLE tpcds.warehouse_t25;
openGauss=# DROP TABLE tpcds.warehouse_t26;
openGauss=# DROP TABLE tpcds.warehouse_t27;
openGauss=# DROP TABLE tpcds.warehouse_t28;
openGauss=# DROP TABLE creditcard_info;
openGauss=# DROP TABLESPACE DS_TABLESPACE1;
openGauss=# DROP SCHEMA IF EXISTS joe CASCADE;
```

相关链接

ALTER TABLE，**DROP TABLE**，**CREATE TABLESPACE**

优化建议

- UNLOGGED

  - UNLOGGED表和表上的索引因为数据写入时不通过WAL日志机制，写入速度远高于普通表。因此，可以用于缓冲存储复杂查询的中间结果集，增强复杂查询的性能。
  - UNLOGGED表无主备机制，在系统故障或异常断点等情况下导致数据库重启，会清除UNLOGGED表数据，会有数据丢失风险，因此，不可用来存储基础数据。
- TEMPORARY | TEMP
  - 临时表只在当前会话可见，会话结束后会自动删除。
  - 除了当前CN外，其他CN对于该临时表不可见。
- LIKE
  - 新表自动从这个表中继承所有字段名及其数据类型和非空约束，新表与源表之间在创建动作完毕之后是完全无关的。
- LIKE INCLUDING DEFAULTS
  - 源表上的字段缺省表达式只有在指定INCLUDING DEFAULTS时，才会复制到新表中。缺省是不包含缺省表达式的，即新表中的所有字段的缺省值都是NULL。
- LIKE INCLUDING CONSTRAINTS
  - 源表上的CHECK约束仅在指定INCLUDING CONSTRAINTS时，会复制到新表中，而其他类型的约束永远不会复制到新表中。非空约束总是复制到新表中。此规则同时适用于表约束和列约束。
- LIKE INCLUDING INDEXES
  - 如果指定了INCLUDING INDEXES，则源表上的索引也将在新表上创建，默认不建立索引。
- LIKE INCLUDING STORAGE
  - 如果指定了INCLUDING STORAGE，则复制列的STORAGE设置会复制到新表中，默认情况下不包含STORAGE设置。
- LIKE INCLUDING COMMENTS
  - 如果指定了INCLUDING COMMENTS，则源表列、约束和索引的注释会复制到新表中。默认情况下，不复制源表的注释。
- LIKE INCLUDING PARTITION
  - 如果指定了INCLUDING PARTITION，则源表的分区定义会复制到新表中，同时新表将不能再使用PARTITION BY子句。默认情况下，不拷贝源表的分区定义。
- LIKE INCLUDING RELOPTIONS
  - 如果指定了INCLUDING RELOPTIONS，则源表的存储参数（即源表的WITH子句）会复制到新表中。默认情况下，不复制源表的存储参数。
- LIKE INCLUDING DISTRIBUTION
  - 如果指定了INCLUDING DISTRIBUTION，则源表的分布信息会复制到新表中，包括分布类型和分布列，同时新表将不能再使用DISTRIBUTE BY子句。默认情况下，不拷贝源表的分布信息。
- LIKE INCLUDING ALL
  - INCLUDING ALL包含了INCLUDING DEFAULTS、INCLUDING CONSTRAINTS、INCLUDING INDEXES、INCLUDING STORAGE、INCLUDING COMMENTS、INCLUDING PARTITION、INCLUDING RELOPTIONS和INCLUDING DISTRIBUTION的内容。

- ORIENTATION ROW
  - 创建行存表，行存储适合于OLTP业务，此类型的表上交互事务比较多，一次交互会涉及表中的多个列，用行存查询效率较高。
- DISTRIBUTE BY
  - 事实表或者数据量较大的维度表建议创建为分布表。对指定的列进行Hash，通过映射，把数据分布到指定DN。语法为:distribute by hash(column_name)。
  - 数据量较小的维度表建议创建为复制表。表的每条记录存在所有数据节点（DN）中，即每个数据节点都有完整的表数据。语法为: distribute by replication。

7.13.66 CREATE TABLESPACE

功能描述

在数据库中创建一个新的表空间。

注意事项

- 系统管理员或者继承了内置角色gs_role_tablespace权限的用户可以创建表空间。
- 不允许在一个事务块内部执行CREATE TABLESPACE。
- 执行CREATE TABLESPACE失败，如果内部创建目录（文件）操作成功了就会产生残留的目录（文件），重新创建时需要用户手动清理表空间指定的目录下残留的内容。如果在创建过程中涉及到数据目录下的表空间软连接残留，需要先将软连接的残留文件删除，再重新执行OM相关操作。
- CREATE TABLESPACE不支持两阶段事务，如果部分节点执行失败，不支持回滚。
- 创建表空间前的准备工作参考下述参数说明。
- 在公有云场景下一般不建议用户使用自定义的表空间。

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

语法格式

```
CREATE TABLESPACE tablespace_name
    [ OWNER user_name ] [ RELATIVE ] LOCATION 'directory' [ MAXSIZE 'space_size' ]
    [with_option_clause];
```

其中普通表空间的with_option_clause为：

```
WITH ( {filesystem= { 'general'| "general" | general} |
    random_page_cost = { 'value ' | value } |
    seq_page_cost = { 'value ' | value }}[,...])
```

参数说明

- **tablespace_name**

  要创建的表空间名称。

  表空间名称不能和数据集群中的其他表空间重名，且名称不能以"pg"开头，这样的名称留给系统表空间使用。

取值范围：字符串，要符合**标识符命名规范**。

- **OWNER user_name**

  指定该表空间的所有者。缺省时，新表空间的所有者是当前用户。

  只有系统管理员可以创建表空间，但是可以通过OWNER子句把表空间的所有权赋给其他非系统管理员。

  取值范围：字符串，已存在的用户。

- **RELATIVE**

  若指定该参数，表示使用相对路径，LOCATION目录是相对于各个CN/DN数据目录下的。

  目录层次：CN和DN的数据目录/pg_location/相对路径。相对路径最多指定两层。

  若没有指定该参数，表示使用绝对表空间路径，LOCATION目录需要使用绝对路径。

- **LOCATION directory**

  用于表空间的目录。当创建绝对表空间路径时，对于目录有如下要求：

  - GaussDB系统用户必须对该目录拥有读写权限，并且目录为空。如果该目录不存在，将由系统自动创建。
  - 目录必须是绝对路径，目录中不得含有特殊字符（如$,> 等）。
  - 目录不允许指定在数据库数据目录下。
  - 目录需为本地路径。

  取值范围：字符串，有效的目录。

- **MAXSIZE 'space_size'**

  指定表空间在单个DN上的最大值。

  取值范围：字符串格式为正整数+单位，单位当前支持K/M/G/T/P。解析后的数值以K为单位，且范围不能够超过64比特表示的有符号整数，即1KB~9007199254740991KB。

- **random_page_cost**

  指定随机读取page的开销。

  取值范围：0~1.79769e+308。

  默认值：使用GUC参数random_page_cost的值。

- **seq_page_cost**

  指定顺序读取page的开销。

  取值范围：0~1.79769e+308。

  默认值：使用GUC参数seq_page_cost的值。

示例

```
--创建表空间。
openGauss=# CREATE TABLESPACE ds_location1 RELATIVE LOCATION 'test_tablespace/test_tablespace_1';

--创建用户joe。
openGauss=# CREATE ROLE joe IDENTIFIED BY '********';

--创建用户jay。
openGauss=# CREATE ROLE jay IDENTIFIED BY '********';

--创建表空间，且所有者指定为用户joe。
```



```
openGauss=# CREATE TABLESPACE ds_location2 OWNER joe RELATIVE LOCATION 'test_tablespace/
test_tablespace_2';

--把表空间ds_location1重命名为ds_location3。
openGauss=# ALTER TABLESPACE ds_location1 RENAME TO ds_location3;

--改变表空间ds_location2的所有者。
openGauss=# ALTER TABLESPACE ds_location2 OWNER TO jay;

--删除表空间。
openGauss=# DROP TABLESPACE ds_location2;
openGauss=# DROP TABLESPACE ds_location3;

--删除用户。
openGauss=# DROP ROLE joe;
openGauss=# DROP ROLE jay;
```

相关链接

CREATE DATABASE，**CREATE TABLE**，**CREATE INDEX**，**DROP TABLESPACE**，**ALTER TABLESPACE**

优化建议

- create tablespace

  不建议在事务内部创建表空间。

7.13.67 CREATE TABLE AS

功能描述

根据查询结果创建表。

CREATE TABLE AS创建一个表并且用来自SELECT命令的结果填充该表。该表的字段和SELECT输出字段的名称及数据类型相关。不过用户可以通过明确地给出一个字段名称列表来覆盖SELECT输出字段的名称。

CREATE TABLE AS对源表进行一次查询，然后将数据写入新表中，而查询视图结果会根据源表的变化而有所改变。相比之下，每次做查询的时候，视图都重新计算定义它的SELECT语句。

注意事项

- 分区表不能采用此方式进行创建。
- 如果在建表过程中数据库系统发生故障，系统恢复后可能无法自动清除之前已创建的、大小非0的磁盘文件。此种情况出现概率小，不影响数据库系统的正常运行。

语法格式

```
CREATE [ UNLOGGED ] TABLE table_name
    [ (column_name [, ...] ) ]
    [ WITH ( {storage_parameter = value} [, ... ] ) ]
    [ TABLESPACE tablespace_name ]
    [ DISTRIBUTE BY { REPLICATION | { [HASH ] ( column_name ) } } ]
    [ TO { GROUP groupname | NODE ( nodename [, ... ] ) } ]
    AS query
    [ WITH [ NO ] DATA ];
```


参数说明

- **UNLOGGED**

  指定表为非日志表。在非日志表中写入的数据不会被写入到预写日志中，这样就会比普通表快很多。但是，它也是不安全的，在冲突或异常关机导致数据库重启后，非日志表数据会被清空。非日志表中的内容也不会被复制到备用服务器中。在该类表中创建的索引也不会被自动记录。

  - 使用场景：非日志表不能保证数据的安全性，用户应该在确保数据已经做好备份的前提下使用，例如系统升级时进行数据的备份。
  - 故障处理：当异常关机等操作导致非日志表上的索引发生数据丢失时，用户应该对发生错误的索引进行重建。

- **GLOBAL | LOCAL**

  创建临时表时可以在TEMP或TEMPORARY前指定GLOBAL或LOCAL关键字。目前这两个关键字的设立，仅是为了兼容SQL标准，实际上无论指定GLOBAL还是LOCAL，GaussDB都会创建本地临时表。

- **TEMPORARY | TEMP**

  如果指定TEMP或TEMPORARY关键字，则创建的表为临时表。临时表只在当前会话可见，本会话结束后会自动删除。因此，在除当前会话连接的CN以外的其他CN故障时，仍然可以在当前会话上创建和使用临时表。由于临时表只在当前会话创建，对于涉及对临时表操作的DDL语句，会产生DDL失败的报错。因此，建议DDL语句中不要对临时表进行操作。TEMP和TEMPORARY等价。

  须知

  - 临时表通过每个会话独立的以pg_temp开头的schema来保证只对当前会话可见，因此，不建议用户在日常操作中手动删除以pg_temp，pg_toast_temp开头的schema。
  - 如果建表时不指定TEMPORARY/TEMP关键字，而指定表的schema为当前会话的pg_temp_开头的schema，则该表会被创建为临时表。
  - 临时表只对当前会话可见，因此不支持与\parallel on并行执行一起使用。
  - 临时表不支持DN故障或者主备切换。

- **table_name**

  要创建的表名。

  取值范围：字符串，要符合**标识符命名规范**。

- **column_name**

  新表中要创建的字段名。

  取值范围：字符串，要符合**标识符命名规范**。

- **WITH (storage_parameter [= value] [, ...])**

  这个子句为表或索引指定一个可选的存储参数。参数的详细说明如下所示。

  - FILLFACTOR

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

择，但是对于频繁更新的表，选择较小的填充因子则更加合适。该参数只对行存表有效。

取值范围：10~100

- ORIENTATION

  取值范围：

  COLUMN：表的数据将以列式存储。

  ROW（缺省值）：表的数据将以行式存储。

- hashbucket

  创建hash bucket存储。本参数仅支持行存表和行存range表。

  取值范围：on/off

  默认值：off

须知

当前版本hashbucket表相关DDL操作性能受限，不建议频繁对hashbucket表进行DDL操作。

- **TABLESPACE tablespace_name**

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

- **DISTRIBUTE BY**

  指定表如何在节点之间分布或者复制。

  - REPLICATION：表的每一行存在所有数据节点(DN)中，即每个数据节点都有完整的表数据。
  - HASH (column_name)：对指定的列进行Hash，通过映射，把数据分布到指定DN。

须知

当指定DISTRIBUTE BY HASH (column_name)参数时，创建主键和唯一索引必须包含“ column_name”列。

缺省值：HASH(column_name)，column_name取表的主键列（如果有的话）或首个数据类型支持作为分布列的列。

column_name的数据类型必须是以下类型之一：

- INTEGER TYPES：TINYINT，SMALLINT，INT，BIGINT，NUMERIC/DECIMAL
- CHARACTER TYPES: CHAR，BPCHAR，VARCHAR，VARCHAR2，NVARCHAR2
- DATE/TIME TYPES: DATE，TIME，TIMETZ，TIMESTAMP，TIMESTAMPTZ，INTERVAL，SMALLDATETIME

- **TO { GROUP groupname | NODE (nodename [, ...]) }**

  TO GROUP指定创建表所在的Node Group。TO NODE主要供内部扩容工具使用，一般用户不应该使用。

- **AS query**

  一个SELECT VALUES命令或者一个运行预备好的SELECT或VALUES查询的EXECUTE命令。

- **[WITH [NO] DATA]**

  创建表时，是否也插入查询到的数据。默认是要数据，选择“NO”参数时，则不要数据。

示例

```
--创建一个SCHEMA。
openGauss=# CREATE SCHEMA tpcds;

--创建一个表tpcds.store_returns表。
openGauss=# CREATE TABLE tpcds.store_returns
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    sr_item_sk                VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT         INTEGER
);

--向表中插入一条记录。
openGauss=# INSERT INTO tpcds.store_returns(W_WAREHOUSE_SK, W_WAREHOUSE_ID, sr_item_sk,
W_WAREHOUSE_SQ_FT) VALUES (1, 'AAAAAAAABAAAAAAA', '4800', '20');

--创建一个表tpcds.store_returns_t1并插入tpcds.store_returns表中sr_item_sk字段中大于4795的数值。
openGauss=# CREATE TABLE tpcds.store_returns_t1 AS SELECT * FROM tpcds.store_returns WHERE
sr_item_sk > '4795';

--使用tpcds.store_returns拷贝一个新表tpcds.store_returns_t2。
openGauss=# CREATE TABLE tpcds.store_returns_t2 AS table tpcds.store_returns;

--删除表。
openGauss=# DROP TABLE tpcds.store_returns_t1 ;
openGauss=# DROP TABLE tpcds.store_returns_t2 ;
openGauss=# DROP TABLE tpcds.store_returns;

--删除schema。
openGauss=# DROP SCHEMA tpcds CASCADE;
```

相关链接

CREATE TABLE，**SELECT**

7.13.68 CREATE TABLE PARTITION

功能描述

创建分区表。分区表是把逻辑上的一张表根据某种方案分成几张物理块进行存储，这张逻辑上的表称之为分区表，物理块称之为分区。分区表是一张逻辑表，不存储数据，数据实际是存储在分区上的。

常见的分区方案有范围分区（Range Partitioning）、哈希分区（Hash Partitioning）、列表分区（List Partitioning）、数值分区（Value Partition）等。目前行存表仅支持范围分区。

范围分区是根据表的一列或者多列，将要插入表的记录分为若干个范围，这些范围在不同的分区里没有重叠。为每个范围创建一个分区，用来存储相应的数据。

范围分区的分区策略是指记录插入分区的方式。目前范围分区仅支持范围分区策略。

范围分区策略：根据分区键值将记录映射到已创建的某个分区上，如果可以映射到已创建的某一分区上，则把记录插入到对应的分区上，否则给出报错和提示信息。这是最常用的分区策略。

分区可以提供若干好处：

- 某些类型的查询性能可以得到极大提升。特别是表中访问率较高的行位于一个单独分区或少数几个分区上的情况下。分区可以减少数据的搜索空间，提高数据访问效率。
- 当查询或更新一个分区的大部分记录时，连续扫描那个分区而不是访问整个表可以获得巨大的性能提升。
- 如果需要大量加载或者删除的记录位于单独的分区上，则可以通过直接读取或删除那个分区以获得巨大的性能提升，同时还可以避免由于大量DELETE导致的VACUUM超载。

注意事项

- 指定分区查询时，如select * from tablename partition (partitionname)，关键字partition注意不要写错。如果写错，查询不会报错，这时查询会变为对表起别名进行查询。
- 分布式下指定分区只支持Select，其它语法进行指定分区操作时会报错，不会转为别名操作。
- 对于分区表进行UPDATE或DELETE时，如果生成的计划不是FQS或Stream计划，语句执行效率会比较差。建议排查语句，消除不可下推因素，从而生成FQS或Stream计划。

语法格式

```
CREATE TABLE [ IF NOT EXISTS ] partition_table_name
( [
   { column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]
   | table_constraint
   | LIKE source_table [ like_option [...] ] }
   [, ... ]
] )
   [ WITH ( {storage_parameter = value} [, ... ] ) ]
   [ TABLESPACE tablespace_name ]
   [ DISTRIBUTE BY { REPLICATION | { [ HASH ] ( column_name ) } } ]
   [ TO { GROUP groupname | NODE ( nodename [, ... ] ) } ]
   PARTITION BY {
      {RANGE (partition_key) ( partition_less_than_item [, ... ] )} |
      {RANGE (partition_key) ( partition_start_end_item [, ... ] )}
   } [ { ENABLE | DISABLE } ROW MOVEMENT ];
```

- 列约束column_constraint：

```
[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  CHECK ( expression ) |
  DEFAULT default_expr |
  UNIQUE [ index_parameters ] |
  PRIMARY KEY [ index_parameters |
  REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
    [ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE ][ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

- 表约束table_constraint：

```
[ CONSTRAINT constraint_name ]
{ CHECK ( expression ) |
  UNIQUE ( column_name [, ... ] ) [ index_parameters ] |
  PRIMARY KEY ( column_name [, ... ] ) [ index_parameters]}
[ DEFERRABLE | NOT DEFERRABLE ][ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```


- like选项like_option：
  ```
  { INCLUDING | EXCLUDING } { DEFAULTS | CONSTRAINTS | INDEXES | STORAGE | COMMENTS | RELOPTIONS | DISTRIBUTION | ALL }
  ```
- 索引存储参数index_parameters：
  ```
  [ WITH ( {storage_parameter = value} [, ... ] ) ]
  [ USING INDEX TABLESPACE tablespace_name ]
  ```
- partition_less_than_item：
  ```
  PARTITION partition_name VALUES LESS THAN ( { partition_value | MAXVALUE } ) [TABLESPACE tablespace_name]
  ```
- partition_start_end_item：
  ```
  PARTITION partition_name {
        {START(partition_value) END (partition_value) EVERY (interval_value)} |
        {START(partition_value) END ({partition_value | MAXVALUE})} |
        {START(partition_value)} |
        {END({partition_value | MAXVALUE})}
  } [TABLESPACE tablespace_name]
  ```

参数说明

- **IF NOT EXISTS**

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

  分区表的名称。

  取值范围：字符串，要符合**标识符命名规范**。
- **column_name**

  新表中要创建的字段名。

  取值范围：字符串，要符合**标识符命名规范**。
- **data_type**

  字段的数据类型。
- **COLLATE collation**

  COLLATE子句指定列的排序规则（该列必须是可排列的数据类型）。如果没有指定，则使用默认的排序规则。
- **CONSTRAINT constraint_name**

  列约束或表约束的名称。可选的约束子句用于声明约束，新行或者更新的行必须满足这些约束才能成功插入或更新。排序规则可以使用“select * from pg_collation”命令从pg_collation系统表中查询，默认的排序规则为查询结果中以default开始的行。

  定义约束有两种方法：

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

  LIKE子句声明一个表，新表自动从这个表里面继承所有字段名及其数据类型和非空约束。

  和INHERITS不同，新表与原来的表之间在创建动作完毕之后是完全无关的。在源表做的任何修改都不会传播到新表中，并且也不可能在扫描源表的时候包含新表的数据。

  字段缺省表达式只有在声明了INCLUDING DEFAULTS之后才会包含进来。缺省是不包含缺省表达式的，即新表中所有字段的缺省值都是NULL。

非空约束将总是复制到新表中，CHECK约束则仅在指定了INCLUDING CONSTRAINTS的时候才复制，而其他类型的约束则永远也不会被复制。此规则同时适用于表约束和列约束。

和INHERITS不同，被复制的列和约束并不使用相同的名称进行融合。如果明确的指定了相同的名称或者在另外一个LIKE子句中，将会报错。

- 如果指定了INCLUDING INDEXES，则源表上的索引也将在新表上创建，默认不建立索引。
- 如果指定了INCLUDING STORAGE，则拷贝列的STORAGE设置也将被拷贝，默认情况下不包含STORAGE设置。
- 如果指定了INCLUDING COMMENTS，则源表列、约束和索引的注释也会被拷贝过来。默认情况下，不拷贝源表的注释。
- 如果指定了INCLUDING RELOPTIONS，则源表的存储参数（即源表的WITH子句）也将拷贝至新表。默认情况下，不拷贝源表的存储参数。
- 如果指定了INCLUDING DISTRIBUTION，则新表将拷贝源表的分布信息，包括分布类型和分布列，同时新表将不能再使用DISTRIBUTE BY子句。默认情况下，不拷贝源表的分布信息。
- INCLUDING ALL是INCLUDING DEFAULTS INCLUDING CONSTRAINTS INCLUDING INDEXES INCLUDING STORAGE INCLUDING COMMENTS INCLUDING RELOPTIONS INCLUDING DISTRIBUTION的简写形式。

- **WITH (storage_parameter [= value] [, ...])**

  这个子句为表或索引指定一个可选的存储参数。参数的详细描述如下所示：

  - FILLFACTOR

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

    取值范围：10~100

  - ORIENTATION

    决定了表的数据的存储方式。

    取值范围：

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

    须知

    orientation不支持修改。

  - STORAGE_TYPE

    指定存储引擎类型，该参数设置成功后就不再支持修改。

    取值范围：

    - USTORE，表示表支持Inplace-Update存储引擎。特别需要注意，使用USTORE表，必须要开启track_counts和track_activities参数，否则会引起空间膨胀。

  - ASTORE，表示表支持Append-Only存储引擎。

  默认值：不指定表时，默认是Append-Only存储。

- hashbucket

  创建hash bucket存储。本参数仅支持行存表和行存range表。

  取值范围：on/off

  默认值：off

须知

当前版本hashbucket表相关DDL操作性能受限，不建议频繁对hashbucket表进行DDL操作。

- **TABLESPACE tablespace_name**

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

- **DISTRIBUTE BY**

  指定表如何在节点之间分布或者复制。

  取值范围：

  - REPLICATION：表的每一行存在所有数据节点(DN)中，即每个数据节点都有完整的表数据。
  - HASH（column_name）：对指定的列进行Hash，通过映射，把数据分布到指定DN。

须知

当指定DISTRIBUTE BY HASH (column_name)参数时，创建主键和唯一索引必须包含“ column_name”列。

  缺省值：HASH(column_name)，column_name取表的主键列（如果有的话）或首个数据类型支持作为分布列的列。

  column_name的数据类型必须是以下类型之一：

  - INTEGER TYPES: TINYINT, SMALLINT, INT, BIGINT, NUMERIC/DECIMAL
  - CHARACTER TYPES: CHAR, BPCHAR, VARCHAR, VARCHAR2, NVARCHAR2
  - DATA/TIME TYPES: DATE, TIME, TIMETZ, TIMESTAMP, TIMESTAMPTZ, INTERVAL, SMALLDATETIME

- **TO { GROUP groupname | NODE (nodename [, ...]) }**

  TO GROUP指定创建表所在的Node Group。TO NODE主要供内部扩容工具使用，一般用户不应该使用。

- **PARTITION BY RANGE(partition_key)**

  创建范围分区。partition_key为分区键的名称。

  （1）对于从句是VALUES LESS THAN的语法格式：

须知

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

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

（2）对于从句是START END的语法格式：

须知

对于从句是START END的语法格式，范围分区策略的分区键仅支持1列。

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

- **PARTITION partition_name VALUES LESS THAN ({ partition_value | MAXVALUE })**

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

  须知

  - 每个分区都需要指定一个上边界。
  - 分区上边界的类型应当和分区键的类型一致。
  - 分区列表是按照分区上边界升序排列的，值较小的分区位于值较大的分区之前。

- **PARTITION partition_name {START (partition_value) END (partition_value) EVERY (interval_value)} | {START (partition_value) END (partition_value|MAXVALUE)} | {START(partition_value)} | {END (partition_value | MAXVALUE)}**

  指定各分区的信息，各参数意义如下：

  - partition_name：范围分区的名称或名称前缀，除以下情形外（假定其中的partition_name是p1），均为分区的名称。
    - 若该定义是START+END+EVERY从句，则语义上定义的分区的名称依次为p1_1, p1_2, ...。例如对于定义“PARTITION p1 START(1) END(4) EVERY(1)”，则生成的分区是：[1, 2), [2, 3) 和 [3, 4)，名称依次为p1_1, p1_2和p1_3，即此处的p1是名称前缀。
    - 若该定义是第一个分区定义，且该定义有START值，则范围（MINVALUE, START）将自动作为第一个实际分区，其名称为p1_0，然后该定义语义描述的分区名称依次为p1_1, p1_2, ...。例如对于完整定义

“PARTITION p1 START(1), PARTITION p2 START(2)”，则生成的分区是：(MINVALUE, 1), [1, 2) 和 [2, MAXVALUE)，其名称依次为p1_0, p1_1和p2，即此处p1是名称前缀，p2是分区名称。这里MINVALUE表示最小值。

- partition_value：范围分区的端点值（起始或终点），取值依赖于partition_key的类型，不可是MAXVALUE。
- interval_value：对[START，END) 表示的范围进行切分，interval_value是指定切分后每个分区的宽度，不可是MAXVALUE；如果（END-START）值不能整除以EVERY值，则仅最后一个分区的宽度小于EVERY值。
- MAXVALUE：表示最大值，它通常用于设置最后一个范围分区的上边界。

须知

1. 在创建分区表若第一个分区定义含START值，则范围（MINVALUE，START）将自动作为实际的第一个分区。
2. START END语法需要遵循以下限制：
   - 每个partition_start_end_item中的START值（如果有的话，下同）必须小于其END值；
   - 相邻的两个partition_start_end_item，第一个的END值必须等于第二个的START值；
   - 每个partition_start_end_item中的EVERY值必须是正向递增的，且必须小于（END-START）值；
   - 每个分区包含起始值，不包含终点值，即形如：[起始值，终点值)，起始值是MINVALUE时则不包含；
   - 一个partition_start_end_item创建的每个分区所属的TABLESPACE一样；
   - partition_name作为分区名称前缀时，其长度不要超过57字节，超过时自动截断；
   - 在创建、修改分区表时请注意分区表的分区总数不可超过最大限制（32767）；
3. 在创建分区表时START END与LESS THAN语法不可混合使用。
4. 即使创建分区表时使用START END语法，备份（gs_dump）出的SQL语句也是VALUES LESS THAN语法格式。

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

  行迁移开关。

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

  取值范围：

  - ENABLE：行迁移开关打开。
  - DISABLE（缺省值）：行迁移开关关闭。

  在打开行迁移开关情况下，并发update、delete操作可能会报错，原因如下：

  update和delete操作对于旧数据都是标记为已删除。在打开行迁移开关情况下，如果更新分区键时，导致了跨分区更新，内核会把旧分区中旧数据标记为已删除，在新分区中新增加一条数据，无法通过旧数据找到新数据。

  在update和update并发、delete和delete并发、update和delete并发三个并发场景下，如果并发操作同一行数据时，数据跨分区和非跨分区结果有不同的行为。

a. 对于数据非跨分区结果，第一个操作执行完后，第二个操作不会报错。

如果第一个操作是update，第二个操作能成功找到最新的数据，之后对新数据操作。

如果第一个操作是delete，第二个操作看到当前数据已经被删除而且找不到最新数据，就终止操作。

b. 对于数据跨分区结果，第一个操作执行完后，第二个操作会报错。

如果第一个操作是update，由于新数据在新分区中，第二个操作不能成功找到最新的数据，就无法操作，之后会报错。

如果第一个操作是delete，第二个操作看到当前数据已经被删除而且找不到最新数据，但无法判断删除旧数据的操作是update还是delete。如果是update，报错处理。如果是delete，终止操作。为了保持数据的正确性，只能报错处理。

如果是update和update并发，update和delete并发场景，需要串行执行才能解决问题，如果是delete和delete并发，关闭行迁移开关可以解决问题。

- **NOT NULL**

  字段值不允许为NULL。ENABLE用于语法兼容，可省略。

- **NULL**

  字段值允许NULL ，这是缺省。

  这个子句只是为和非标准SQL数据库兼容。不建议使用。

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

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

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

  用NO INHERIT标记的约束将不会传递到子表中去。

  ENABLE用于语法兼容，可省略。

- **DEFAULT default_expr**

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

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

- **UNIQUE index_parameters**

  UNIQUE (column_name [, ...]) index_parameters

  UNIQUE约束表示表里的一个字段或多个字段的组合必须在全表范围内唯一。

  对于唯一约束，NULL被认为是互不相等的。

  说明

  如果没有声明DISTRIBUTE BY REPLICATION，则唯一约束的列集合中必须包含分布列。

- **PRIMARY KEY index_parameters**

  PRIMARY KEY (column_name [, ...]) index_parameters

  主键约束声明表中的一个或者多个字段只能包含唯一的非NULL值。

  一个表只能声明一个主键。

说明

如果没有声明DISTRIBUTE BY REPLICATION，则主键约束的列集合中必须包含分布列。

- **DEFERRABLE | NOT DEFERRABLE**

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

- **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

  如果约束是可推迟的，则这个子句声明检查约束的缺省时间。

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

  约束检查的时间可以用SET CONSTRAINTS命令修改。

- **USING INDEX TABLESPACE tablespace_name**

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

示例

- 示例1：创建范围分区表tpcds.web_returns_p1，含有8个分区，分区键为integer类型。 分区的范围分别为：wr_returned_date_sk< 2450815，2450815<= wr_returned_date_sk< 2451179，2451179<=wr_returned_date_sk< 2451544，2451544 <= wr_returned_date_sk< 2451910，2451910 <= wr_returned_date_sk< 2452275，2452275 <= wr_returned_date_sk< 2452640，2452640 <= wr_returned_date_sk< 2453005，wr_returned_date_sk>=2453005。

```
-- 创建临时schema。
openGauss=# CREATE SCHEMA tpcds;
openGauss=# SET CURRENT_SCHEMA TO tpcds;

--创建表tpcds.web_returns。
openGauss=# CREATE TABLE tpcds.web_returns
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
);

--创建分区表tpcds.web_returns_p1。
openGauss=# CREATE TABLE tpcds.web_returns_p1
(
    WR_RETURNED_DATE_SK       INTEGER                       ,
    WR_RETURNED_TIME_SK       INTEGER                       ,
    WR_ITEM_SK                INTEGER               NOT NULL,
    WR_REFUNDED_CUSTOMER_SK   INTEGER                       ,
```



```
    WR_REFUNDED_CDEMO_SK      INTEGER                   ,
    WR_REFUNDED_HDEMO_SK      INTEGER                   ,
    WR_REFUNDED_ADDR_SK       INTEGER                   ,
    WR_RETURNING_CUSTOMER_SK  INTEGER                   ,
    WR_RETURNING_CDEMO_SK     INTEGER                   ,
    WR_RETURNING_HDEMO_SK     INTEGER                   ,
    WR_RETURNING_ADDR_SK      INTEGER                   ,
    WR_WEB_PAGE_SK            INTEGER                   ,
    WR_REASON_SK              INTEGER                   ,
    WR_ORDER_NUMBER           BIGINT               NOT NULL,
    WR_RETURN_QUANTITY        INTEGER                   ,
    WR_RETURN_AMT             DECIMAL(7,2)              ,
    WR_RETURN_TAX             DECIMAL(7,2)              ,
    WR_RETURN_AMT_INC_TAX     DECIMAL(7,2)              ,
    WR_FEE                    DECIMAL(7,2)              ,
    WR_RETURN_SHIP_COST       DECIMAL(7,2)              ,
    WR_REFUNDED_CASH          DECIMAL(7,2)              ,
    WR_REVERSED_CHARGE        DECIMAL(7,2)              ,
    WR_ACCOUNT_CREDIT         DECIMAL(7,2)              ,
    WR_NET_LOSS               DECIMAL(7,2)
)
DISTRIBUTE BY HASH (WR_ITEM_SK)
PARTITION BY RANGE(WR_RETURNED_DATE_SK)
(
        PARTITION P1 VALUES LESS THAN(2450815),
        PARTITION P2 VALUES LESS THAN(2451179),
        PARTITION P3 VALUES LESS THAN(2451544),
        PARTITION P4 VALUES LESS THAN(2451910),
        PARTITION P5 VALUES LESS THAN(2452275),
        PARTITION P6 VALUES LESS THAN(2452640),
        PARTITION P7 VALUES LESS THAN(2453005),
        PARTITION P8 VALUES LESS THAN(MAXVALUE)
);

--从示例数据表导入数据。
openGauss=# INSERT INTO tpcds.web_returns_p1 SELECT * FROM tpcds.web_returns;

--删除分区P8。
openGauss=# ALTER TABLE tpcds.web_returns_p1 DROP PARTITION P8;

--增加分区WR_RETURNED_DATE_SK介于2453005和2453105之间。
openGauss=# ALTER TABLE tpcds.web_returns_p1 ADD PARTITION P8 VALUES LESS THAN (2453105);

--增加分区WR_RETURNED_DATE_SK介于2453105和MAXVALUE之间。
openGauss=# ALTER TABLE tpcds.web_returns_p1 ADD PARTITION P9 VALUES LESS THAN
(MAXVALUE);

--删除分区P8。
openGauss=# ALTER TABLE tpcds.web_returns_p1 DROP PARTITION FOR (2453005);

--分区P7重命名为P10。
openGauss=# ALTER TABLE tpcds.web_returns_p1 RENAME PARTITION P7 TO P10;

--分区P6重命名为P11。
openGauss=# ALTER TABLE tpcds.web_returns_p1 RENAME PARTITION FOR (2452639) TO P11;

--查询分区P10的行数。
openGauss=# SELECT count(*) FROM tpcds.web_returns_p1 PARTITION (P10);
 count
--------
 0
(1 row)

--查询分区P1的行数。
openGauss=# SELECT COUNT(*) FROM tpcds.web_returns_p1 PARTITION FOR (2450815);
 count
--------
 0
(1 row)
```



```
--删除表和表空间。
openGauss=# DROP TABLE tpcds.web_returns_p1;
openGauss=# DROP TABLE tpcds.web_returns;
openGauss=# DROP SCHEMA tpcds CASCADE;
```

- 示例2：创建范围分区表tpcds.web_returns_p2，含有8个分区，分区键类型为integer类型，其中第8个分区上边界为MAXVALUE。

  八个分区的范围分别为： wr_returned_date_sk< 2450815，2450815<= wr_returned_date_sk< 2451179，2451179<=wr_returned_date_sk< 2451544，2451544 <= wr_returned_date_sk< 2451910，2451910 <= wr_returned_date_sk< 2452275，2452275 <= wr_returned_date_sk< 2452640，2452640 <= wr_returned_date_sk< 2453005，wr_returned_date_sk>=2453005。

  分区表tpcds.web_returns_p2的表空间为example1；分区P1到P7没有声明表空间，使用采用分区表tpcds.web_returns_p2的表空间example1；指定分区P8的表空间为example2。

  假定CN和DN的数据目录/pg_location/mount1/path1，CN和DN的数据目录/pg_location/mount2/path2，CN和DN的数据目录/pg_location/mount3/path3，CN和DN的数据目录/pg_location/mount4/path4是dwsadmin用户拥有读写权限的空目录。

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

-- 创建临时schema。
openGauss=# CREATE SCHEMA tpcds;
openGauss=# SET CURRENT_SCHEMA TO tpcds;
openGauss=# CREATE TABLE tpcds.web_returns_p2
(
    WR_RETURNED_DATE_SK       INTEGER                   ,
    WR_RETURNED_TIME_SK       INTEGER                   ,
    WR_ITEM_SK                INTEGER             NOT NULL,
    WR_REFUNDED_CUSTOMER_SK   INTEGER                     ,
    WR_REFUNDED_CDEMO_SK      INTEGER                     ,
    WR_REFUNDED_HDEMO_SK      INTEGER                     ,
    WR_REFUNDED_ADDR_SK       INTEGER                     ,
    WR_RETURNING_CUSTOMER_SK  INTEGER                     ,
    WR_RETURNING_CDEMO_SK     INTEGER                     ,
    WR_RETURNING_HDEMO_SK     INTEGER                     ,
    WR_RETURNING_ADDR_SK      INTEGER                     ,
    WR_WEB_PAGE_SK            INTEGER                   ,
    WR_REASON_SK              INTEGER                   ,
    WR_ORDER_NUMBER           BIGINT              NOT NULL,
    WR_RETURN_QUANTITY        INTEGER                   ,
    WR_RETURN_AMT             DECIMAL(7,2)              ,
    WR_RETURN_TAX             DECIMAL(7,2)              ,
    WR_RETURN_AMT_INC_TAX     DECIMAL(7,2)              ,
    WR_FEE                    DECIMAL(7,2)              ,
    WR_RETURN_SHIP_COST       DECIMAL(7,2)              ,
    WR_REFUNDED_CASH          DECIMAL(7,2)              ,
    WR_REVERSED_CHARGE        DECIMAL(7,2)              ,
    WR_ACCOUNT_CREDIT         DECIMAL(7,2)              ,
    WR_NET_LOSS               DECIMAL(7,2)
)
TABLESPACE example1
DISTRIBUTE BY HASH (WR_ITEM_SK)
PARTITION BY RANGE(WR_RETURNED_DATE_SK)
(
    PARTITION P1 VALUES LESS THAN(2450815),
    PARTITION P2 VALUES LESS THAN(2451179),
    PARTITION P3 VALUES LESS THAN(2451544),
    PARTITION P4 VALUES LESS THAN(2451910),
```



```
    PARTITION P5 VALUES LESS THAN(2452275),
    PARTITION P6 VALUES LESS THAN(2452640),
    PARTITION P7 VALUES LESS THAN(2453005),
    PARTITION P8 VALUES LESS THAN(MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;

--以like方式创建一个分区表。
openGauss=# CREATE TABLE tpcds.web_returns_p3 (LIKE tpcds.web_returns_p2 INCLUDING
PARTITION);

--修改分区P1的表空间为example2。
openGauss=# ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P1 TABLESPACE example2;

--修改分区P2的表空间为example3。
openGauss=# ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P2 TABLESPACE example3;

--以2453010为分割点切分P8。
openGauss=# ALTER TABLE tpcds.web_returns_p2 SPLIT PARTITION P8 AT (2453010) INTO
(
    PARTITION P9,
    PARTITION P10
);

--将P6，P7合并为一个分区。
openGauss=# ALTER TABLE tpcds.web_returns_p2 MERGE PARTITIONS P6, P7 INTO PARTITION P8;

--修改分区表迁移属性。
openGauss=# ALTER TABLE tpcds.web_returns_p2 DISABLE ROW MOVEMENT;

--删除表和表空间。
openGauss=# DROP TABLE tpcds.web_returns_p1;
openGauss=# DROP TABLE tpcds.web_returns_p2;
openGauss=# DROP TABLE tpcds.web_returns_p3;
openGauss=# DROP SCHEMA tpcds CASCADE;
openGauss=# DROP TABLESPACE example1;
openGauss=# DROP TABLESPACE example2;
openGauss=# DROP TABLESPACE example3;
openGauss=# DROP TABLESPACE example4;
```

- 示例3：START END语法创建、修改Range分区表。

  假定/home/omm/startend_tbs1，/home/omm/startend_tbs2，/home/omm/startend_tbs3，/home/omm/startend_tbs4是omm用户拥有读写权限的空目录。

```
-- 创建表空间
openGauss=# CREATE TABLESPACE startend_tbs1 LOCATION '/home/omm/startend_tbs1';
openGauss=# CREATE TABLESPACE startend_tbs2 LOCATION '/home/omm/startend_tbs2';
openGauss=# CREATE TABLESPACE startend_tbs3 LOCATION '/home/omm/startend_tbs3';
openGauss=# CREATE TABLESPACE startend_tbs4 LOCATION '/home/omm/startend_tbs4';

-- 创建临时schema
openGauss=# CREATE SCHEMA tpcds;
openGauss=# SET CURRENT_SCHEMA TO tpcds;

-- 创建分区表，分区键是integer类型
openGauss=# CREATE TABLE tpcds.startend_pt (c1 INT, c2 INT)
TABLESPACE startend_tbs1
DISTRIBUTE BY HASH (c1)
PARTITION BY RANGE (c2) (
   PARTITION p1 START(1) END(1000) EVERY(200) TABLESPACE startend_tbs2,
   PARTITION p2 END(2000),
   PARTITION p3 START(2000) END(2500) TABLESPACE startend_tbs3,
   PARTITION p4 START(2500),
   PARTITION p5 START(3000) END(5000) EVERY(1000) TABLESPACE startend_tbs4
)
ENABLE ROW MOVEMENT;

-- 查看分区表信息
```



```
openGauss=# SELECT relname, boundaries, spcname FROM pg_partition p JOIN pg_tablespace t ON
p.reltablespace=t.oid and p.parentid='tpcds.startend_pt'::regclass ORDER BY 1;
   relname   | boundaries |    spcname
-------------+------------+---------------
 p1_0        | {1}        | startend_tbs2
 p1_1        | {201}      | startend_tbs2
 p1_2        | {401}      | startend_tbs2
 p1_3        | {601}      | startend_tbs2
 p1_4        | {801}      | startend_tbs2
 p1_5        | {1000}     | startend_tbs2
 p2          | {2000}     | startend_tbs1
 p3          | {2500}     | startend_tbs3
 p4          | {3000}     | startend_tbs1
 p5_1        | {4000}     | startend_tbs4
 p5_2        | {5000}     | startend_tbs4
 startend_pt |            | startend_tbs1
(12 rows)

-- 导入数据，查看分区数据量
openGauss=# INSERT INTO tpcds.startend_pt VALUES (GENERATE_SERIES(0, 4999),
GENERATE_SERIES(0, 4999));
openGauss=# SELECT COUNT(*) FROM tpcds.startend_pt PARTITION FOR (0);
 count
-------
     1
(1 row)

openGauss=# SELECT COUNT(*) FROM tpcds.startend_pt PARTITION (p3);
 count
-------
   500
(1 row)

-- 增加分区: [5000, 5300), [5300, 5600), [5600, 5900), [5900, 6000)
openGauss=# ALTER TABLE tpcds.startend_pt ADD PARTITION p6 START(5000) END(6000)
EVERY(300) TABLESPACE startend_tbs4;

-- 增加MAXVALUE分区: p7
openGauss=# ALTER TABLE tpcds.startend_pt ADD PARTITION p7 END(MAXVALUE);

-- 重命名分区p7为p8
openGauss=# ALTER TABLE tpcds.startend_pt RENAME PARTITION p7 TO p8;

-- 删除分区p8
openGauss=# ALTER TABLE tpcds.startend_pt DROP PARTITION p8;

-- 重命名5950所在的分区为：p71
openGauss=# ALTER TABLE tpcds.startend_pt RENAME PARTITION FOR(5950) TO p71;

-- 分裂4500所在的分区[4000, 5000)
openGauss=# ALTER TABLE tpcds.startend_pt SPLIT PARTITION FOR(4500) INTO(PARTITION q1
START(4000) END(5000) EVERY(250) TABLESPACE startend_tbs3);

-- 修改分区p2的表空间为startend_tbs4
openGauss=# ALTER TABLE tpcds.startend_pt MOVE PARTITION p2 TABLESPACE startend_tbs4;

-- 查看分区情形
openGauss=# SELECT relname, boundaries, spcname FROM pg_partition p JOIN pg_tablespace t ON
p.reltablespace=t.oid and p.parentid='tpcds.startend_pt'::regclass ORDER BY 1;
   relname   | boundaries |    spcname
-------------+------------+---------------
 p1_0        | {1}        | startend_tbs2
 p1_1        | {201}      | startend_tbs2
 p1_2        | {401}      | startend_tbs2
 p1_3        | {601}      | startend_tbs2
 p1_4        | {801}      | startend_tbs2
 p1_5        | {1000}     | startend_tbs2
 p2          | {2000}     | startend_tbs4
 p3          | {2500}     | startend_tbs3
```



```
 p4          | {3000}     | startend_tbs1
 p5_1        | {4000}     | startend_tbs4
 p6_1        | {5300}     | startend_tbs4
 p6_2        | {5600}     | startend_tbs4
 p6_3        | {5900}     | startend_tbs4
 p71         | {6000}     | startend_tbs4
 q1_1        | {4250}     | startend_tbs3
 q1_2        | {4500}     | startend_tbs3
 q1_3        | {4750}     | startend_tbs3
 q1_4        | {5000}     | startend_tbs3
 startend_pt |            | startend_tbs1
(19 rows)

-- 删除表和表空间
openGauss=# DROP TABLE tpcds.startend_pt;
openGauss=# DROP SCHEMA tpcds CASCADE;
openGauss=# DROP TABLESPACE startend_tbs1;
openGauss=# DROP TABLESPACE startend_tbs2;
openGauss=# DROP TABLESPACE startend_tbs3;
openGauss=# DROP TABLESPACE startend_tbs4;
```

相关链接

ALTER TABLE PARTITION，**DROP TABLE**

7.13.69 CREATE TRIGGER

功能描述

创建一个触发器。 触发器将与指定的表或视图关联，并在特定条件下执行指定的函数。

注意事项

- 当前仅支持在普通行存表上创建触发器，不支持在临时表、unlogged表等类型表上创建触发器。
- 如果为同一事件定义了多个相同类型的触发器，则按触发器的名称字母顺序触发它们。
- 触发器常用于多表间数据关联同步场景，对SQL执行性能影响较大，不建议在大数据量同步及对性能要求高的场景中使用。
- 当触发器满足如下条件时，触发语句能和触发器一起下推到DN执行并提升触发器执行性能：
  - 开关enable_trigger_shipping和enable_fast_query_shipping开启。
  - 源表触发器使用的触发器函数为plpgsql类型（推荐类型）。
  - 源表与触发表分布键的类型、数量完全相同，均为行存表，且所属相同的nodegroup。
  - 原INSERT/UPDATE/DELETE语句条件中包含所有分布键与NEW/OLD等值比较表达式。
  - 原INSERT/UPDATE/DELETE语句在没有触发器的情况下原本就能query shipping。
  - 源表上只有INSERT BEFORE FOR EACH ROW/INSERT AFTER FOR EACH ROW/UPDATE BEFORE FOR EACH ROW/UPDATE AFTER FOR EACH ROW/DELETE BEFORE FOR EACH ROW/DELETE AFTER FOR EACH ROW六类触发器，且所有触发器都可下推。

- INSERT ON DUPLICATE KEY UPDATE语句无法触发触发器。
- 执行触发器语句时，是使用触发器创建者的身份进行权限判断的。
- 执行创建触发器操作的用户需要拥有指定表的TRIGGER权限。

语法格式

```
CREATE [ CONSTRAINT ] TRIGGER name { BEFORE | AFTER | INSTEAD OF } { event [ OR ... ] }
   ON table_name
   [ FROM referenced_table_name ]
   { NOT DEFERRABLE | [ DEFERRABLE ] { INITIALLY IMMEDIATE | INITIALLY DEFERRED } }
   [ FOR [ EACH ] { ROW | STATEMENT } ]
   [ WHEN ( condition ) ]
   EXECUTE PROCEDURE function_name ( arguments );
```

其中event包含以下几种：

```
   INSERT
   UPDATE [ OF column_name [, ... ] ]
   DELETE
   TRUNCATE
```

参数说明

- **CONSTRAINT**

  可选项，指定此参数将创建约束触发器，即触发器作为约束来使用。除了可以使用SET CONSTRAINTS调整触发器触发的时间之外，这与常规触发器相同。 约束触发器必须是AFTER ROW触发器。

- **name**

  触发器名称，该名称不能限定模式，因为触发器自动继承其所在表的模式，且同一个表的触发器不能重名。 对于约束触发器，使用**SET CONSTRAINTS**修改触发器行为时也使用此名称。

  取值范围：符合**标识符命名规范**的字符串，且最大长度不超过63个字符。

- **BEFORE**

  触发器函数是在触发事件发生前执行。

- **AFTER**

  触发器函数是在触发事件发生后执行，约束触发器只能指定为AFTER。

- **INSTEAD OF**

  触发器函数直接替代触发事件。

- **event**

  启动触发器的事件，取值范围包括：INSERT、UPDATE、DELETE或TRUNCATE，也可以通过OR同时指定多个触发事件。

  对于UPDATE事件类型，可以使用下面语法指定列：

  ```
  UPDATE OF column_name1 [, column_name2 ... ]
  ```

  表示只有这些列作为UPDATE语句的目标列时，才会启动触发器，但是INSTEAD OF UPDATE类型不支持指定列信息。如果UPDATE OF指定的列包含生成列，当生成列依赖的列是UPDATE语句的目标列时，也会启动触发器。

- **table_name**

  需要创建触发器的表名称。

  取值范围：数据库中已经存在的表名称。

- **referenced_table_name**

约束引用的另一个表的名称。 只能为约束触发器指定，常见于外键约束。由于当前不支持外键，因此不建议使用。

取值范围：数据库中已经存在的表名称。

- **DEFERRABLE | NOT DEFERRABLE**

  约束触发器的启动时机，仅作用于约束触发器。这两个关键字设置该约束是否可推迟。

  详细介绍请参见**CREATE TABLE**。

- **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

  如果约束是可推迟的，则这个子句声明检查约束的缺省时间，仅作用于约束触发器。

  详细介绍请参见**CREATE TABLE**。

- **FOR EACH ROW | FOR EACH STATEMENT**

  触发器的触发频率。

  - FOR EACH ROW是指该触发器是受触发事件影响的每一行触发一次。
  - FOR EACH STATEMENT是指该触发器是每个SQL语句只触发一次。

  未指定时默认值为FOR EACH STATEMENT。约束触发器只能指定为FOR EACH ROW。

- **condition**

  决定是否实际执行触发器函数的条件表达式。当指定WHEN时，只有在条件返回true时才会调用该函数。

  在FOR EACH ROW触发器中，WHEN条件可以通过分别写入OLD.column_name或NEW.column_name来引用旧行或新行值的列。INSERT触发器不能引用OLD和DELETE触发器不能引用NEW。

  INSTEAD OF触发器不支持WHEN条件。

  WHEN表达式不能包含子查询。

  对于约束触发器，WHEN条件的评估不会延迟，而是在执行更新操作后立即发生。 如果条件返回值不为true，则触发器不会排队等待延迟执行。

- **function_name**

  用户定义的函数，必须声明为不带参数并返回类型为触发器，在触发器触发时执行。

- **arguments**

  执行触发器时要提供给函数的可选的以逗号分隔的参数列表。参数是文字字符串常量，简单的名称和数字常量也可以写在这里，但它们都将被转换为字符串。 请检查触发器函数的实现语言的描述，以了解如何在函数内访问这些参数。

  说明

  关于触发器种类：

  - INSTEAD OF的触发器必须标记为FOR EACH ROW，并且只能在视图上定义。
  - BEFORE和AFTER触发器作用在视图上时，只能标记为FOR EACH STATEMENT。
  - TRUNCATE类型触发器仅限FOR EACH STATEMENT。

表 7-112 表和视图上支持的触发器种类：

触发时机	触发事件	行级	语句级
BEFORE	INSERT/UPDATE/DELETE	表	表和视图
	TRUNCATE	不支持	表
AFTER	INSERT/UPDATE/DELETE	表	表和视图
	TRUNCATE	不支持	表
INSTEAD OF	INSERT/UPDATE/DELETE	视图	不支持
	TRUNCATE	不支持	不支持

表 7-113 plpgsql 类型触发器函数特殊变量：

变量名	变量含义
NEW	INSERT及UPDATE操作涉及tuple信息中的新值，对DELETE为空。
OLD	UPDATE及DELETE操作涉及tuple信息中的旧值，对INSERT为空。
TG_NAME	触发器名称。
TG_WHEN	触发器触发时机（BEFORE/AFTER/INSTEAD OF）。
TG_LEVEL	触发频率（ROW/STATEMENT）。
TG_OP	触发操作（INSERT/UPDATE/DELETE/TRUNCATE）。
TG_RELID	触发器所在表OID。
TG_RELNAME	触发器所在表名（已废弃，现用TG_TABLE_NAME替代）。
TG_TABLE_NAME	触发器所在表名。
TG_TABLE_SCHEMA	触发器所在表的SCHEMA信息。
TG_NARGS	触发器函数参数个数。
TG_ARGV[]	触发器函数参数列表。

示例

```
--创建源表及触发表
openGauss=# CREATE TABLE test_trigger_src_tbl(id1 INT, id2 INT, id3 INT);
openGauss=# CREATE TABLE test_trigger_des_tbl(id1 INT, id2 INT, id3 INT);
```



```
--创建触发器函数
openGauss=# CREATE OR REPLACE FUNCTION tri_insert_func() RETURNS TRIGGER AS
        $$
        DECLARE
        BEGIN
                INSERT INTO test_trigger_des_tbl VALUES(NEW.id1, NEW.id2, NEW.id3);
                RETURN NEW;
        END
        $$ LANGUAGE plpgsql;

openGauss=# CREATE OR REPLACE FUNCTION tri_update_func() RETURNS TRIGGER AS
        $$
        DECLARE
        BEGIN
                UPDATE test_trigger_des_tbl SET id3 = NEW.id3 WHERE id1=OLD.id1;
                RETURN OLD;
        END
        $$ LANGUAGE plpgsql;

openGauss=# CREATE OR REPLACE FUNCTION TRI_DELETE_FUNC() RETURNS TRIGGER AS
        $$
        DECLARE
        BEGIN
                DELETE FROM test_trigger_des_tbl WHERE id1=OLD.id1;
                RETURN OLD;
        END
        $$ LANGUAGE plpgsql;

--创建INSERT触发器
openGauss=# CREATE TRIGGER insert_trigger
        BEFORE INSERT ON test_trigger_src_tbl
        FOR EACH ROW
        EXECUTE PROCEDURE tri_insert_func();

--创建UPDATE触发器
openGauss=# CREATE TRIGGER update_trigger
        AFTER UPDATE ON test_trigger_src_tbl
        FOR EACH ROW
        EXECUTE PROCEDURE tri_update_func();

--创建DELETE触发器
openGauss=# CREATE TRIGGER delete_trigger
        BEFORE DELETE ON test_trigger_src_tbl
        FOR EACH ROW
        EXECUTE PROCEDURE tri_delete_func();

--执行INSERT触发事件并检查触发结果
openGauss=# INSERT INTO test_trigger_src_tbl VALUES(100,200,300);
openGauss=# SELECT * FROM test_trigger_src_tbl;
openGauss=# SELECT * FROM test_trigger_des_tbl;  //查看触发操作是否生效。

--执行UPDATE触发事件并检查触发结果
openGauss=# UPDATE test_trigger_src_tbl SET id3=400 WHERE id1=100;
openGauss=# SELECT * FROM test_trigger_src_tbl;
openGauss=# SELECT * FROM test_trigger_des_tbl;  //查看触发操作是否生效

--执行DELETE触发事件并检查触发结果
openGauss=# DELETE FROM test_trigger_src_tbl WHERE id1=100;
openGauss=# SELECT * FROM test_trigger_src_tbl;
openGauss=# SELECT * FROM test_trigger_des_tbl;  //查看触发操作是否生效

--修改触发器
openGauss=# ALTER TRIGGER delete_trigger ON test_trigger_src_tbl RENAME TO delete_trigger_renamed;

--禁用insert_trigger触发器
openGauss=# ALTER TABLE test_trigger_src_tbl DISABLE TRIGGER insert_trigger;

--禁用当前表上所有触发器
```



```
openGauss=# ALTER TABLE test_trigger_src_tbl DISABLE TRIGGER ALL;

--删除触发器
openGauss=# DROP TRIGGER insert_trigger ON test_trigger_src_tbl;
openGauss=# DROP TRIGGER update_trigger ON test_trigger_src_tbl;
openGauss=# DROP TRIGGER delete_trigger_renamed ON test_trigger_src_tbl;
```

相关链接

ALTER TRIGGER，**DROP TRIGGER**，**ALTER TABLE**

7.13.70 CREATE TYPE

功能描述

在当前数据库中定义一种新的数据类型。定义数据类型的用户将成为该数据类型的拥有者。类型只适用于行存表

有五种形式的CREATE TYPE，分别为：复合类型、基本类型、shell类型、枚举类型和集合类型。

- 复合类型

  复合类型由一个属性名和数据类型的列表指定。如果属性的数据类型是可排序的，也可以指定该属性的排序规则。复合类型本质上和表的行类型相同，但是如果只想定义一种类型，使用CREATE TYPE避免了创建一个实际的表。单独的复合类型也是很有用的，例如可以作为函数的参数或者返回类型。

  为了能够创建复合类型，必须拥有在其所有属性类型上的USAGE特权。

- 基本类型

  用户可以自定义一种新的基本类型（标量类型）。通常来说这些函数必须是用C或者另外一种低层语言所编写。

- shell类型

  shell类型是一种用于后面要定义的类型的占位符，通过发出一个不带除类型名之外其他参数的CREATE TYPE命令可以创建这种类型。在创建基本类型时，需要shell类型作为一种向前引用。

- 枚举类型

  由若干个标签构成的列表，每一个标签值都是一个非空字符串，且字符串长度必须不超过64个字节。

- 集合类型

  类似数组，但是没有长度限制，主要在存储过程中使用。

- 被授予CREATE ANY TYPE权限的用户，可以在public模式和用户模式下创建类型。

注意事项

如果给定一个模式名，那么该类型将被创建在指定的模式中。否则它会被创建在当前模式中。类型名称必须与同一个模式中任何现有的类型或者域相区别（因为表具有相关的数据类型，类型名称也必须与同一个模式中任何现有表的名称不同）。

语法格式

```
CREATE TYPE name AS
    ( [ attribute_name data_type [ COLLATE collation ] [, ... ] ] )
```



```
CREATE TYPE name (
    INPUT = input_function,
    OUTPUT = output_function
    [ , RECEIVE = receive_function ]
    [ , SEND = send_function ]
    [ , TYPMOD_IN = type_modifier_input_function ]
    [ , TYPMOD_OUT = type_modifier_output_function ]
    [ , ANALYZE = analyze_function ]
    [ , INTERNALLENGTH = { internallength | VARIABLE } ]
    [ , PASSEDBYVALUE ]
    [ , ALIGNMENT = alignment ]
    [ , STORAGE = storage ]
    [ , LIKE = like_type ]
    [ , CATEGORY = category ]
    [ , PREFERRED = preferred ]
    [ , DEFAULT = default ]
    [ , ELEMENT = element ]
    [ , DELIMITER = delimiter ]
    [ , COLLATABLE = collatable ]
)

CREATE TYPE name

CREATE TYPE name AS ENUM
    ( [ 'label' [, ... ] ] )

CREATE TYPE name AS TABLE OF data_type
```

参数说明

复合类型

- **name**

  要创建的类型的名称（可以被模式限定）。

- **attribute_name**

  复合类型的一个属性（列）的名称。

- **data_type**

  要成为复合类型的一个列的现有数据类型的名称。可以使用%ROWTYPE间接引用表的类型，或者使用%TYPE间接引用表或复合类型中某一列的类型。

- **collation**

  要关联到复合类型的一列的现有排序规则的名称。排序规则可以使用“select * from pg_collation”命令从pg_collation系统表中查询，默认的排序规则为查询结果中以default开始的行。

基本类型

自定义基本类型时，参数可以以任意顺序出现，input_function和output_function为必选参数，其它为可选参数。

- **input_function**

  将数据从类型的外部文本形式转换为内部形式的函数名。

  输入函数可以被声明为有一个cstring类型的参数，或者有三个类型分别为cstring、 oid、integer的参数。

  - cstring参数是以C字符串存在的输入文本。
  - oid参数是该类型自身的OID（对于数组类型则是其元素类型的OID）。
  - integer参数是目标列的typmod（如果知道，不知道则将传递 -1）。

输入函数必须返回一个该数据类型本身的值。通常，一个输入函数应该被声明为STRICT。 如果不是这样，在读到一个NULL输入值时，调用输入函数时第一个参数会是NULL。在这种情况下，该函数必须仍然返回NULL，除非调用函数发生了错误（这种情况主要是想支持域输入函数，域输入函数可能需要拒绝NULL输入）。

说明

输入和输出函数能被声明为具有新类型的结果或参数是因为：必须在创建新类型之前创建这两个函数。而新类型应该首先被定义为一种shell type，它是一种占位符类型，除了名称和拥有者之外它没有其他属性。这可以通过不带额外参数的命令CREATE TYPE name做到。然后用C写的I/O函数可以被定义为引用这种shell type。最后，用带有完整定义的CREATE TYPE把该shell type替换为一个完全的、合法的类型定义，之后新类型就可以正常使用了。

- **output_function**

  将数据从类型的内部形式转换为外部文本形式的函数名。

  输出函数必须被声明为有一个新数据类型的参数。输出函数必须返回类型cstring。对于NULL值不会调用输出函数。

- **receive_function**

  可选参数。将数据从类型的外部二进制形式转换成内部形式的函数名。

  如果没有该函数，该类型不能参与到二进制输入中。二进制表达转换成内部形式代价更低，然而却更容易移植（例如，标准的整数数据类型使用网络字节序作为外部二进制表达，而内部表达是机器本地的字节序）。receive_function应该执行足够的检查以确保该值是有效的。

  接收函数可以被声明为有一个internal类型的参数，或者有三个类型分别为internal、oid、integer的参数。

  - internal参数是一个指向StringInfo缓冲区的指针，其中保存着接收到的字节串。
  - oid和integer参数和文本输入函数的相同。

  接收函数必须返回一个该数据类型本身的值。通常，一个接收函数应该被声明为STRICT。如果不是这样，在读到一个NULL输入值时调用接收函数时第一个参数会是NULL。在这种情况下，该函数必须仍然返回NULL，除非接收函数发生了错误（这种情况主要是想支持域接收函数，域接收函数可能需要拒绝NULL输入）。

- **send_function**

  可选参数。将数据从类型的内部形式转换为外部二进制形式的函数名。

  如果没有该函数，该类型将不能参与到二进制输出中。发送函数必须被声明为有一个新数据类型的参数。发送函数必须返回类型bytea。对于NULL值不会调用发送函数。

- **type_modifier_input_function**

  可选参数。将类型的修饰符数组转换为内部形式的函数名。

- **type_modifier_output_function**

  可选参数。将类型的修饰符的内部形式转换为外部文本形式的函数名。

说明

如果该类型支持修饰符（附加在类型声明上的可选约束，例如，char(5)或numeric(30,2)），则需要可选的type_modifier_input_function以及type_modifier_output_function。GaussDB允许用户定义的类型有一个或者多个简单常量或者标识符作为修饰符。不过，为了存储在系统目录中，该信息必须能被打包到一个非负整数值中。所声明的修饰符会被以cstring数组的形式传递给type_modifier_input_function。 type_modifier_input_function必须检查该值的合法性（如果值错误就抛出一个错误），如果值正确，要返回一个非负integer值，该值将被存储在“typmod”列中。如果类型没有 type_modifier_input_function则类型修饰符将被拒绝。type_modifier_output_function把内部的整数typmod值转换回正确的形式用于用户显示。type_modifier_output_function必须返回一个cstring值，该值就是追加到类型名称后的字符串。例如，numeric的函数可能会返回(30,2)。如果默认的显示格式就是只把存储的typmod整数值放在圆括号内，则允许省略type_modifier_output_function。

- **analyze_function**

  可选参数。为该数据类型执行统计分析的函数名的可选参数。

  默认情况下，如果该类型有一个默认的B-tree操作符类，ANALYZE将尝试用类型的“equals”和“less-than”操作符来收集统计信息。这种行为对于非标量类型并不合适，因此可以通过指定一个自定义分析函数来覆盖这种行为。分析函数必须被声明为有一个类型为internal的参数，并且返回一个boolean结果。

- **internallength**

  可选参数。一个数字常量，用于指定新类型的内部表达的字节长度。默认为变长。

  虽然只有I/O函数和其他为该类型创建的函数才知道新类型的内部表达的细节， 但是内部表达的一些属性必须被向GaussDB声明。其中最重要的是internallength。基本数据类型可以是定长的（这种情况下internallength是一个正整数）或者是变长的（把internallength设置为VARIABLE，在内部通过把typlen设置为-1表示）。所有变长类型的内部表达都必须以一个4字节整数开始，internallength定义了总长度。

- **PASSEDBYVALUE**

  可选参数。表示这种数据类型的值需要被传值而不是传引用。传值的类型必须是定长的，并且它们的内部表达不能超过Datum类型（某些机器上是4字节，其他机器上是8字节）的尺寸。

- **alignment**

  可选参数。该参数指定数据类型的存储对齐需求。如果被指定，必须是char、int2、int4或者double。默认是int4。

  允许的值等同于以1、2、4或8字节边界对齐。要注意变长类型的alignment参数必须至少为4，因为它们需要包含一个int4作为它们的第一个组成部分。

- **storage**

  可选参数。该数据类型的存储策略。

  如果被指定，必须是plain、external、extended或者main。 默认是plain。

  - plain指定该类型的数据将总是被存储在线内并且不会被压缩。（对定长类型只允许plain）
  - extended 指定系统将首先尝试压缩一个长的数据值，并且将在数据仍然太长的情况下把值移出主表行。
  - external允许值被移出主表， 但是系统将不会尝试对它进行压缩。
  - main允许压缩，但是不鼓励把值移出主表（如果没有其他办法让行的大小变得合适，具有这种存储策略的数据项仍将被移出主表，但比起extended以及external项来，这种存储策略的数据项会被优先考虑保留在主表中）。

除plain之外所有的storage值都暗示该数据类型的函数能处理被TOAST过的值。指定的值仅仅是决定一种可TOAST数据类型的列的默认TOAST存储策略，用户可以使用ALTER TABLE SET STORAGE为列选取其他策略。

- **like_type**

  可选参数。与新类型具有相同表达的现有数据类型的名称。会从这个类型中复制internallength、 passedbyvalue、 alignment以及storage的值（ 除非在这个CREATE TYPE命令的其他地方用显式说明覆盖）。

  当新类型的底层实现是以一种现有的类型为参考时，用这种方式指定表达特别有用。

- **category**

  可选参数。这种类型的分类码（一个ASCII 字符）。 默认是“用户定义类型”的'U'。为了创建自定义分类， 也可以选择其他 ASCII字符。

- **preferred**

  可选参数。如果这种类型是其类型分类中的优先类型则为TRUE，否则为FALSE。默认为假。在一个现有类型分类中创建一种新的优先类型要非常谨慎， 因为这可能会导致很大的改变。

  说明

  category和preferred参数可以被用来帮助控制在混淆的情况下应用哪一种隐式造型。每一种数据类型都属于一个用单个ASCII 字符命名的分类，并且每一种类型可以是其所属分类中的“首选”。当有助于解决重载函数或操作符时，解析器将优先造型到首选类型（但是只能从同类的其他类型造型）。对于没有隐式转换到或来自任意其他类型的类型，让这些设置保持默认即可。不过，对于有隐式转换的相关类型的组，把它们都标记为属于同一个类别并且选择一种或两种“最常用”的类型作为该类别的首选通常是很有用的。在把一种用户定义的类型增加到一个现有的内建类别（例如，数字或者字符串类型）中时，category参数特别有用。不过，也可以创建新的全部是用户定义类型的类别。对这样的类别，可选择除大写字母之外的任何ASCII 字符。

- **default**

  可选参数。数据类型的默认值。如果被省略，默认值是空。

  如果用户希望该数据类型的列被默认为某种非空值，可以指定一个默认值。默认值可以用DEFAULT关键词指定（这样一个默认值可以被附加到一个特定列的显式DEFAULT子句覆盖）。

- **element**

  可选参数。被创建的类型是一个数组，element指定了数组元素的类型。例如，要定义一个4字节整数的数组（int4）， 应指定ELEMENT = int4。

- **delimiter**

  可选参数。指定这种类型组成的数组中分隔值的定界符。

  可以把delimiter设置为一个特定字符，默认的定界符是逗号（,）。注意定界符是与数组元素类型相关的，而不是数组类型本身相关。

- **collatable**

  可选参数。如果这个类型的操作可以使用排序规则信息，则为TRUE。默认为FALSE。

  如果collatable为TRUE，这种类型的列定义和表达式可能通过使用COLLATE子句携带有排序规则信息。在该类型上操作的函数的实现负责真正利用这些信息，仅把类型标记为可排序的并不会让它们自动地去使用这类信息。

- **label**

  可选参数。与枚举类型的一个值相关的文本标签，其值为长度不超过63个字符的非空字符串。

说明

在创建用户定义类型的时候， GaussDB会自动创建一个与之关联的数组类型，其名称由该元素类型的名称前缀一个下划线组成。

示例

```
--创建一种复合类型，建表并插入数据以及查询：
openGauss=# CREATE TYPE compfoo AS (f1 int, f2 text);
openGauss=# CREATE TABLE t1_compfoo(a int, b compfoo);
openGauss=# CREATE TABLE t2_compfoo(a int, b compfoo);
openGauss=# INSERT INTO t1_compfoo values(1,(1,'demo'));
openGauss=# INSERT INTO t2_compfoo select * from t1_typ5;
openGauss=# SELECT (b).f1 FROM t1_compfoo;
openGauss=# SELECT * FROM t1_compfoo t1 join t2_compfoo t2 on (t1.b).f1=(t1.b).f1;

--重命名数据类型：
openGauss=# ALTER TYPE compfoo RENAME TO compfoo1;

--要改变一个用户定义类型compfoo1的所有者为usr1：
CREATE USER usr1 PASSWORD '********';
openGauss=# ALTER TYPE compfoo1 OWNER TO usr1;

--把用户定义类型compfoo1的模式改变为usr1:
openGauss=# ALTER TYPE compfoo1 SET SCHEMA usr1;

给一个数据类型增加一个新的属性：
openGauss=# ALTER TYPE usr1.compfoo1 ADD ATTRIBUTE f3 int;

删除compfoo1类型：
openGauss=# DROP TYPE usr1.compfoo1 cascade;

删除相关表和用户:
openGauss=# DROP TABLE t1_compfoo;
openGauss=# DROP TABLE t2_compfoo;
openGauss=# DROP SCHEMA usr1;
openGauss=# DROP USER usr1;

--创建一个枚举类型
openGauss=# CREATE TYPE bugstatus AS ENUM ('create', 'modify', 'closed');

--添加一个标签值
openGauss=# ALTER TYPE bugstatus ADD VALUE IF NOT EXISTS 'regress' BEFORE 'closed';

--重命名一个标签值
openGauss=# ALTER TYPE bugstatus RENAME VALUE 'create' TO 'new';

--创建一个集合类型
openGauss=# CREATE TYPE compfoo_table AS TABLE OF compfoo;
```

相关链接

ALTER TYPE，**DROP TYPE**

7.13.71 CREATE USER

功能描述

创建一个用户。

注意事项

- 通过CREATE USER创建的用户，默认具有LOGIN权限。

- 通过CREATE USER创建用户的同时，系统会在执行该命令的数据库中，为该用户创建一个同名的SCHEMA。
- 系统管理员在普通用户同名schema下创建的对象，所有者为schema的同名用户（非系统管理员）。

语法格式

```
CREATE USER user_name [ [ WITH ] option [ ... ] ] [ ENCRYPTED | UNENCRYPTED ] { PASSWORD | IDENTIFIED BY } { 'password' [EXPIRED] | DISABLE };
```

其中option子句用于设置权限及属性等信息。

```
{SYSADMIN | NOSYSADMIN}
    | {MONADMIN | NOMONADMIN}
    | {OPRADMIN | NOOPRADMIN}
    | {POLADMIN | NOPOLADMIN}
    | {AUDITADMIN | NOAUDITADMIN}
    | {CREATEDB | NOCREATEDB}
    | {USEFT | NOUSEFT}
    | {CREATEROLE | NOCREATEROLE}
    | {INHERIT | NOINHERIT}
    | {LOGIN | NOLOGIN}
    | {REPLICATION | NOREPLICATION}
    | {VCADMIN | NOVCADMIN}
    | {PERSISTENCE | NOPERSISTENCE}
    | CONNECTION LIMIT connlimit
    | VALID BEGIN 'timestamp'
    | VALID UNTIL 'timestamp'
    | USER GROUP 'groupuser'
    | PERM SPACE 'spacelimit'
    | TEMP SPACE 'tmpspacelimit'
    | SPILL SPACE 'spillspacelimit'
    | NODE GROUP logic_cluster_name
    | IN ROLE role_name [, ...]
    | IN GROUP role_name [, ...]
    | ROLE role_name [, ...]
    | ADMIN role_name [, ...]
    | USER role_name [, ...]
    | SYSID uid
    | DEFAULT TABLESPACE tablespace_name
    | PROFILE DEFAULT
    | PROFILE profile_name
    | PGUSER
```

参数说明

- **user_name**

  用户名称。

  取值范围：字符串，要符合**标识符命名规范**。且最大长度不超过63个字符。当用户名中包含大写字母时数据库将自动转换为小写字母，如果需要创建包含大写字母的用户名则需要使用双引号括起来。

- **password**

  登录密码。

  密码规则如下：

  - 密码默认不少于8个字符。
  - 不能与用户名及用户名倒序相同。
  - 至少包含大写字母（A-Z），小写字母（a-z），数字（0-9），非字母数字字符（限定为~!@#$%^&*()-_=+\|[{}];:,<.>/?）四类字符中的三类字符。当密码中包含的字符不属于上述四种字符范围内时语句执行会报错。

- 密码也可以是符合格式要求的密文字符串，这种情况主要用于用户数据导入场景，不推荐用户直接使用。如果直接使用密文密码，用户需要知道密文密码对应的明文，并且保证明文密码复杂度，数据库不会校验密文密码复杂度，直接使用密文密码的安全性由用户保证。
- 创建用户时，应当使用单引号将用户密码括起来。

取值范围：字符串。

CREATE USER的其他参数值请参考**CREATE ROLE参数说明**。

示例

```
--创建用户jim，登录密码为********。
openGauss=# CREATE USER jim PASSWORD '********';

--下面语句与上面的等价。
openGauss=# CREATE USER kim IDENTIFIED BY '********';

--如果创建有“创建数据库”权限的用户，则需要加CREATEDB关键字。
openGauss=# CREATE USER dim CREATEDB PASSWORD '********';

--将用户jim的登录密码由********修改为**********。
openGauss=# ALTER USER jim IDENTIFIED BY '**********' REPLACE '********';

--为用户jim追加CREATEROLE权限。
openGauss=# ALTER USER jim CREATEROLE;

--锁定jim账户。
openGauss=# ALTER USER jim ACCOUNT LOCK;

--删除用户。
openGauss=# DROP USER kim CASCADE;
openGauss=# DROP USER jim CASCADE;
openGauss=# DROP USER dim CASCADE;
```

相关链接

ALTER USER，**CREATE ROLE**，**DROP USER**

7.13.72 CREATE VIEW

功能描述

创建一个视图。视图与基本表不同，是一个虚拟的表。数据库中仅存放视图的定义，而不存放视图对应的数据，这些数据仍存放在原来的基本表中。若基本表中的数据发生变化，从视图中查询出的数据也随之改变。从这个意义上讲，视图就像一个窗口，透过它可以看到数据库中用户感兴趣的数据及变化。

注意事项

被授予CREATE ANY TABLE权限的用户，可以在public模式和用户模式下创建视图。

语法格式

```
CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] VIEW view_name [ ( column_name [, ...] ) ]
    [ WITH ( {view_option_name [= view_option_value]} [, ... ] ) ]
    AS query;
```


说明

创建视图时使用WITH(security_barrier)可以创建一个相对安全的视图，避免攻击者利用低成本函数的RAISE语句打印出基表数据。

当视图创建后，不允许使用REPLACE修改本视图当中的列名，也不允许删除列。

参数说明

- **OR REPLACE**

  如果视图已存在，则重新定义。

- **TEMP | TEMPORARY**

  创建临时视图。

- **view_name**

  要创建的视图名称。可以用模式修饰。

  取值范围：字符串，符合**标识符命名规范**。

- **column_name**

  可选的名称列表，用作视图的字段名。如果没有给出，字段名取自查询中的字段名。

  取值范围：字符串，符合**标识符命名规范**。

- **view_option_name [= view_option_value]**

  该子句为视图指定一个可选的参数。

  目前view_option_name支持的参数仅有security_barrier，当VIEW视图提供行级安全时，应使用该参数。

  取值范围：Boolean类型，TRUE、FALSE

- **query**

  为视图提供行和列的SELECT或VALUES语句。

须知

若query包含指定分区表分区的子句，创建视图会将所指定分区的OID硬编码到系统表中。如果使用导致指定分区的OID发生变更的分区DDL语法，如DROP/SPLIT/MERGE该分区，则会导致视图不可用。需要重新创建视图。

示例

```
--创建字段spcname为pg_default组成的视图。
openGauss=# CREATE VIEW myView AS
    SELECT * FROM pg_tablespace WHERE spcname = 'pg_default';

--查看视图。
openGauss=# SELECT * FROM myView ;

--删除视图myView。
openGauss=# DROP VIEW myView;
```

相关链接

ALTER VIEW，**DROP VIEW**

7.13.73 CREATE WEAK PASSWORD DICTIONARY

功能描述

向gs_global_config表中插入一个或者多个弱口令。

注意事项

- 只有初始用户、系统管理员和安全管理员拥有权限执行本语法。
- 弱口令字典中的口令存放在gs_global_config系统表中。
- 弱口令字典默认为空，用户通过本语法可以新增一条或多条弱口令。
- 当用户尝试通过本语法插入gs_global_config表中已存在的弱口令时，会只在表中保留一条该弱口令。

语法格式

```
CREATE WEAK PASSWORD DICTIONARY
    [WITH VALUES] ( {'weak_password'} [, ...] );
```

参数说明

weak_password

弱口令。

范围：字符串。

示例

```
--向gs_global_config系统表中插入单个弱口令。
openGauss=# CREATE WEAK PASSWORD DICTIONARY WITH VALUES ('password1');

--向gs_global_config系统表中插入多个弱口令。
openGauss=# CREATE WEAK PASSWORD DICTIONARY WITH VALUES ('password2'),('password3');

--清空gs_global_config系统表中所有弱口令。
openGauss=# DROP WEAK PASSWORD DICTIONARY;

--查看现有弱口令。
openGauss=# SELECT * FROM gs_global_config WHERE NAME LIKE 'weak_password';
```

相关链接

13.14.119-DROP WEAK PASSWORD DICTIONARY

7.13.74 CURSOR

功能描述

CURSOR命令定义一个游标，用于在一个大的查询里面检索少数几行数据。

为了处理SQL语句，存储过程进程分配一段内存区域来保存上下文联系。游标是指向上下文区域的句柄或指针。借助游标，存储过程可以控制上下文区域的变化。

注意事项

- 游标命令只能在事务块里使用。
- 通常游标和SELECT一样返回文本格式。因为数据在系统内部是用二进制格式存储的，系统必须对数据做一定转换以生成文本格式。一旦数据是以文本形式返回，客户端应用需要把它们转换成二进制进行操作。使用FETCH语句，游标可以返回文本或二进制格式。
- 应该小心使用二进制游标。文本格式一般都比对应的二进制格式占用的存储空间大。二进制游标返回内部二进制形态的数据，可能更易于操作。如果想以文本方式显示数据，则以文本方式检索会为用户节约很多客户端的工作。比如，如果查询从某个整数列返回1，在缺省的游标里将获得一个字符串1，但在二进制游标里将得到一个4字节的包含该数值内部形式的数值（大端顺序）。

语法格式

```
CURSOR cursor_name
    [ BINARY ]  [ NO SCROLL ]  [ { WITH | WITHOUT } HOLD ]
    FOR query ;
```

参数说明

- **cursor_name**

  将要创建的游标名。

  取值范围：遵循数据库对象命名规范。

- **BINARY**

  指明游标以二进制而不是文本格式返回数据。

- **NO SCROLL**

  声明游标检索数据行的方式。

  - NO SCROLL：声明该游标不能用于以倒序的方式检索数据行。
  - 未声明：根据执行计划的不同，自动判断该游标是否可以用于以倒序的方式检索数据行。

- **WITH HOLD | WITHOUT HOLD**

  声明当创建游标的事务结束后，游标是否能继续使用。

  - WITH HOLD：声明该游标在创建它的事务结束后仍可继续使用。
  - WITHOUT HOLD：声明该游标在创建它的事务之外不能再继续使用，此游标将在事务结束时被自动关闭。
  - 如果不指定WITH HOLD或WITHOUT HOLD，默认行为是WITHOUT HOLD。
  - 跨节点事务不支持WITH HOLD（例如在多Coordinator部署集群中所创建的含有DDL的事务属于跨节点事务）。

- **query**

  使用SELECT或VALUES子句指定游标返回的行。

  取值范围：SELECT或VALUES子句。

示例

请参考FETCH的**示例**。

相关链接

FETCH

7.13.75 DEALLOCATE

功能描述

DEALLOCATE用于删除前面编写的预备语句。如果用户没有明确删除一个预备语句，那么它将在会话结束的时候被删除。

PREPARE关键字总被忽略。

注意事项

无。

语法格式

```
DEALLOCATE [ PREPARE ] { name | ALL };
```

参数说明

- **name**
  将要删除的预备语句。
- **ALL**
  删除所有预备语句。

示例

无。

7.13.76 DECLARE

功能描述

DECLARE命令既可以定义一个游标，用于在一个大的查询里面检索少数几行数据，也可以作为一个匿名块的开始。

本节主要描述定义为游标的用法，定义为匿名块的用法见**BEGIN**。

为了处理SQL语句，存储过程进程分配一段内存区域来保存上下文联系。游标是指向上下文区域的句柄或指针。借助游标，存储过程可以控制上下文区域的变化。

通常游标和SELECT一样返回文本格式。因为数据在系统内部是用二进制格式存储的，系统必须对数据做一定转换以生成文本格式。一旦数据是以文本形式返回，客户端应用需要把它们转换成二进制进行操作。使用FETCH语句，游标可以返回文本或二进制格式。

注意事项

- 游标命令只能在事务块里使用。
- 应该小心使用二进制游标。文本格式一般都比对应的二进制格式占用的存储空间大。二进制游标返回内部二进制形态的数据，可能更易于操作。如果想以文本方

式显示数据，则以文本方式检索会为用户节约很多客户端的工作。比如，如果查询从某个整数列返回1，在缺省的游标里将获得一个字符串1，但在二进制游标里将得到一个4字节的包含该数值内部形式的数值（大端顺序）。

语法格式

- 定义游标

```
DECLARE cursor_name [ BINARY ] [ NO SCROLL ]
    CURSOR [ { WITH | WITHOUT } HOLD ] FOR query ;
```

- 开启匿名块

```
[DECLARE [declare_statements]]
BEGIN
execution_statements
END;
/
```

参数说明

- **cursor_name**

  将要创建的游标名。

  取值范围：遵循数据库对象命名规范。

- **BINARY**

  指明游标以二进制而不是文本格式返回数据。

- **NO SCROLL**

  声明游标检索数据行的方式。

  - NO SCROLL：声明该游标不能用于以倒序的方式检索数据行。
  - 未声明：根据执行计划的不同，自动判断该游标是否可以用于以倒序的方式检索数据行。

- **WITH HOLD**

  WITHOUT HOLD

  声明当创建游标的事务结束后，游标是否能继续使用。

  - WITH HOLD：声明该游标在创建它的事务结束后仍可继续使用。
  - WITHOUT HOLD：声明该游标在创建它的事务之外不能再继续使用，此游标将在事务结束时被自动关闭。
  - 如果不指定WITH HOLD或WITHOUT HOLD，默认行为是WITHOUT HOLD。

- **query**

  使用SELECT或VALUES子句指定游标返回的行。

  取值范围：SELECT或VALUES子句。

- **declare_statements**

  声明变量，包括变量名和变量类型，如“sales_cnt int”。

- **execution_statements**

  匿名块中要执行的语句。

  取值范围：已存在的函数名称。

示例

开启匿名块示例请参考BEGIN的**示例**。

定义游标示例请参考FETCH的**示例**。

相关链接

BEGIN，**FETCH**

7.13.77 DELETE

功能描述

DELETE从指定的表里删除满足WHERE子句的行。如果WHERE子句不存在，将删除表中所有行，结果只保留表结构。

注意事项

- 表的所有者、被授予了表DELETE权限的用户或被授予DELETE ANY TABLE权限的用户有权删除表中数据，系统管理员默认拥有此权限。同时也必须有USING子句引用的表以及condition上读取的表的SELECT权限。
- 对于行存复制表，仅支持两种场景下的delete操作：1）有主键约束的场景；2）执行计划能下推的场景。

语法格式

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
DELETE [/*+ plan_hint */] [FROM] [ ONLY ] table_name [ * ] [ [ AS ] alias ]
    [ USING using_list ]
    [ WHERE condition | WHERE CURRENT OF cursor_name ]
    [ RETURNING { * | { output_expr [ [ AS ] output_name ] } [, ...] } ];
```

其中with_query的详细格式为：

```
with_query_name [ ( column_name [, ...] ) ] AS [ [ NOT ] MATERIALIZED ]
( {select | values | insert | update | delete} )
```

参数说明

- **WITH [RECURSIVE] with_query [, ...]**

  用于声明一个或多个可以在主查询中通过名称引用的子查询，相当于临时表。

  如果声明了RECURSIVE，那么允许SELECT子查询通过名称引用它自己。

  – with_query_name指定子查询生成的结果集名称，在查询中可使用该名称访问子查询的结果集。

  – column_name指定子查询结果集中显示的列名。

  – 每个子查询可以是SELECT，VALUES，INSERT，UPDATE或DELETE语句。

  – 用户可以使用MATERIALIZED / NOT MATERIALIZED对CTE进行修饰。

  - 如果声明为MATERIALIZED，WITH查询将被物化，生成一个子查询结果集的拷贝，在引用处直接查询该拷贝，因此WITH子查询无法和主干SELECT语句进行联合优化（如谓词下推、等价类传递等），对于此类场景可以使用NOT MATERIALIZED进行修饰，如果WITH查询语义上可以作为子查询内联执行，则可以进行上述优化。
  - 如果用户没有显式声明物化属性则遵守以下规则：如果CTE只在所属主干语句中被引用一次，且语义上支持内联执行，则会被改写为子查询内联执行，否则以CTE Scan的方式物化执行。

- **plan_hint子句**

  以/*+ */的形式在DELETE关键字后，用于对DELETE对应的语句块生成的计划进行hint调优，详细用法请参见章节**使用Plan Hint进行调优**。每条语句中只有第一个/*+ plan_hint */注释块会作为hint生效，里面可以写多条hint。

- **ONLY**

  如果指定ONLY则只有该表被删除；如果没有声明，则该表和它的所有子表将都被删除。

- **table_name**

  目标表的名称（可以有模式修饰）。

  取值范围：已存在的表名。

- **alias**

  目标表的别名。

  取值范围：字符串，符合**标识符命名规范**。

- **using_list**

  using子句。

- **condition**

  一个返回Boolean值的表达式，用于判断哪些行需要被删除。建议不要使用int等数值类型作为condition，因为int等数值类型可以隐式转换为bool值（非0值隐式转换为true，0转换为false），可能导致非预期的结果。

- **WHERE CURRENT OF cursor_name**

  当前不支持，仅保留语法接口。

- **output_expr**

  DELETE命令删除行之后计算输出结果的表达式，该表达式可以使用表的任意字段，可以使用*返回被删除行的所有字段。

- **output_name**

  一个字段的输出名称。

  取值范围：字符串，符合**标识符命名规范**。

示例

```
--创建一个SCHEMA。
openGauss=# CREATE SCHEMA tpcds;

--创建表tpcds.customer_address。
openGauss=# CREATE TABLE tpcds.customer_address
(
ca_address_sk         INTEGER           NOT NULL,
ca_address_id         CHARACTER(16)     NOT NULL,
ca_street_number      INTEGER                   ,
ca_street_name        CHARACTER (20)
);

--向表中插入多条记录。
openGauss=# INSERT INTO tpcds.customer_address VALUES (1, 'AAAAAAAABAAAAAAA', '18', 'Jackson'),
(10000, 'AAAAAAAACAAAAAAA', '362', 'Washington 6th'),(15000, 'AAAAAAAADAAAAAAA', '585', 'Dogwood
Washington');

--创建表tpcds.customer_address_bak。
openGauss=# CREATE TABLE tpcds.customer_address_bak AS TABLE tpcds.customer_address;

--删除tpcds.customer_address_bak中ca_address_sk小于14888的职员。
openGauss=# DELETE FROM tpcds.customer_address_bak WHERE ca_address_sk < 14888;
```



```
--删除tpcds.customer_address_bak中所有数据。
openGauss=# DELETE FROM tpcds.customer_address_bak;

--删除tpcds.customer_address_bak表。
openGauss=# DROP TABLE tpcds.customer_address_bak;

--删除tpcds.customer_address表。
openGauss=# DROP TABLE tpcds.customer_address;

--删除SCHEMA。
openGauss=# DROP SCHEMA tpcds CASCADE;
```

优化建议

- delete
  如果要删除表中的所有记录，建议使用truncate语法。

7.13.78 DO

功能描述

执行匿名代码块。

代码块被看作是没有参数的一段函数体，返回值类型是void。它的解析和执行是同一时刻发生的。

注意事项

- 程序语言在使用之前，必须通过命令CREATE LANGUAGE安装到当前的数据库中。 plpgsql是默认的安装语言，其它语言安装时必须指定。
- 如果语言是不受信任的，用户必须有使用程序语言的USAGE权限，或者是系统管理员权限。

语法格式

```
DO [ LANGUAGE lang_name ] code;
```

参数说明

- **lang_name**
  用来解析代码的程序语言的名称，如果缺省，默认的语言是plpgsql。
- **code**
  可以被执行的程序语言代码，必须指定为字符串。

示例

```
--创建用户webuser。
openGauss=# CREATE USER webuser PASSWORD '********';

--授予用户webuser对模式tpcds下视图的所有操作权限。
openGauss=# DO $$DECLARE r record;
BEGIN
    FOR r IN SELECT c.relname,n.nspname FROM pg_class c,pg_namespace n
            WHERE c.relnamespace = n.oid AND n.nspname = 'tpcds' AND relkind IN ('r','v')
    LOOP
        EXECUTE 'GRANT ALL ON ' || quote_ident(r.table_schema) || '.' || quote_ident(r.table_name) || ' TO
webuser';
    END LOOP;
```



```
END$$;


--删除用户webuser。
openGauss=# DROP USER webuser CASCADE;
```

7.13.79 DROP AUDIT POLICY

功能描述

删除一个审计策略。

注意事项

只有poladmin，sysadmin或初始用户才能进行此操作。

语法格式

```
DROP AUDIT POLICY [IF EXISTS] policy_name;
```

参数说明

- **policy_name**
  审计策略名称，需要唯一，不可重复。
  取值范围：字符串，要符合**标识符命名规范**。

示例

请参考CREATE AUDIT POLICY的**示例**。

相关链接

CREATE AUDIT POLICYALTER AUDIT POLICY。

7.13.80 DROP DATABASE

功能描述

删除一个数据库。

注意事项

- 只有数据库所有者或者被授予了数据库DROP权限的用户有权限执行DROP DATABASE命令，系统管理员默认拥有此权限。
- 不能对系统默认安装的三个数据库（POSTGRES、TEMPLATE0和TEMPLATE1）执行删除操作，系统做了保护。如果想查看当前服务中有哪几个数据库，可以用gsql的\l命令查看。
- 如果有用户正在与要删除的数据库连接，则删除操作失败。如果要查看当前存在哪些数据库连接，可以通过视图DV_SESSIONS查看。
- 不能在事务块中执行DROP DATABASE命令。
- 确定删除数据库前需要执行“CLEAN CONNECTION TO ALL FORCE FOR DATABASE XXXX;”命令，用于强制停止当前已有的用户连接及后台线程，防止

因为有后台线程未完全退出而导致的删库失败问题。此处需要注意，强制停止后台线程可能导致当前数据库数据一致性问题，此命令仅在确定删库阶段执行。

- 如果执行DROP DATABASE失败，事务回滚，需要再次执行一次DROP DATABASE IF EXISTS。

须知

DROP DATABASE一旦执行将无法撤销，请谨慎使用。

语法格式

```
DROP DATABASE [ IF EXISTS ] database_name ;
```

参数说明

- **IF EXISTS**
  如果指定的数据库不存在，则发出一个notice而不是抛出一个错误。
- **database_name**
  要删除的数据库名称。
  取值范围：字符串，已存在的数据库名称。

示例

请参见CREATE DATABASE的**示例**。

相关链接

CREATE DATABASE

优化建议

- drop database
  不支持在事务中删除database。

7.13.81 DROP DIRECTORY

功能描述

删除指定的directory对象。

注意事项

当enable_access_server_directory=off时，只允许初始用户删除directory对象；当enable_access_server_directory=on时，具有SYSADMIN权限的用户、directory对象的属主、被授予了该directory的DROP权限的用户或者继承了内置角色gs_role_directory_drop权限的用户可以删除directory对象。

语法格式

```
DROP DIRECTORY [ IF EXISTS ] directory_name;
```


参数说明

- **directory_name**

  目录名称。

  取值范围：已经存在的目录名。

示例

```
--创建目录。
openGauss=# CREATE OR REPLACE DIRECTORY  dir  as '/tmp/';

--删除外部表。
openGauss=# DROP DIRECTORY dir;
```

相关链接

CREATE DIRECTORY，**ALTER DIRECTORY**

7.13.82 DROP FUNCTION

功能描述

删除一个已存在的函数。

注意事项

如果函数中涉及对临时表的相关操作，则无法使用DROP FUNCTION删除函数。

只有函数的所有者或者被授予了函数DROP权限的用户才能执行DROP FUNCTION命令，系统管理员默认拥有该权限。

语法格式

```
DROP FUNCTION [ IF EXISTS ] function_name
[ ( [ { [ argname ] [ argmode ] argtype} [, ...] ] ) [ CASCADE | RESTRICT ] ];
```

参数说明

- **IF EXISTS**

  IF EXISTS表示，如果函数存在则执行删除操作，函数不存在也不会报错，只是发出一个notice。

- **function_name**

  要删除的函数名称。

  取值范围：已存在的函数名。

- **argmode**

  函数参数的模式。

- **argname**

  函数参数的名称。

- **argtype**

  函数参数的类型

- **CASCADE | RESTRICT**

- CASCADE：级联删除依赖于函数的对象 。
- RESTRICT：如果有任何依赖对象存在，则拒绝删除该函数（缺省行为）。

示例

请参见CREATE FUNCTION的**示例**。

相关链接

ALTER FUNCTION，**CREATE FUNCTION**

7.13.83 DROP GLOBAL CONFIGURATION

功能描述

删除系统表gs_global_config中的参数值。

注意事项

仅支持数据库初始用户运行此命令。

参数名称不能为weak_password、undostoragetype。

语法格式

```
DROP GLOBAL CONFIGURATION name [, ...];
```

参数说明

- **name**
  参数名称必须是gs_global_config中已经存在的，删除不存在的参数将报错。

相关链接

ALTER GLOBAL CONFIGURATION

7.13.84 DROP GROUP

功能描述

删除用户组。DROP GROUP是DROP ROLE的别名。

注意事项

仅对有CREATE ROLE权限的用户开放，CREATE ROLE权限通过管理员用户赋予。

语法格式

```
DROP GROUP [ IF EXISTS ] group_name [, ...];
```

参数说明

- **IF EXISTS**

如果指定的角色不存在，则发出一个notice而不是抛出一个错误。

- **group_name**

  要删除的角色名称。

  取值范围：已存在的角色。

示例

请参见CREATE ROLE的**示例**。

相关链接

CREATE GROUP，**ALTER GROUP**，**DROP ROLE**

7.13.85 DROP INDEX

功能描述

删除索引。

注意事项

索引的所有者、索引所在模式或者拥有索引所在表的INDEX权限的用户有权限执行DROP INDEX命令，系统管理员默认拥有此权限。

语法格式

```
DROP INDEX [ IF EXISTS ]
  index_name [, ...] [ CASCADE | RESTRICT ];
```

参数说明

- **IF EXISTS**

  如果指定的索引不存在，则发出一个notice而不是抛出一个错误。

- **index_name**

  要删除的索引名。

  取值范围：已存在的索引。

- **CASCADE | RESTRICT**
  - CASCADE：表示允许级联删除依赖于该索引的对象。
  - RESTRICT：表示有依赖于此索引的对象存在时，该索引无法被删除。此选项为缺省值。

示例

请参见CREATE INDEX的**示例**。

相关链接

ALTER INDEX，**CREATE INDEX**

7.13.86 DROP LANGUAGE

本版本暂不支持使用该语法。

7.13.87 DROP MASKING POLICY

功能描述

删除脱敏策略。

注意事项

只有poladmin，sysadmin或初始用户才能执行此操作。

语法格式

```
DROP MASKING POLICY [ IF EXISTS ] policy_name;
```

参数说明

- **policy_name**
  审计策略名称，需要唯一，不可重复。
  取值范围：字符串，要符合**标识符命名规范**。已存在的策略名称。

示例

```
--删除一个脱敏策略。
openGauss=# DROP MASKING POLICY IF EXISTS maskpol1;

--删除一组脱敏策略。
openGauss=# DROP MASKING POLICY IF EXISTS maskpol1, maskpol2, maskpol3;
```

相关链接

ALTER MASKING POLICY，**CREATE MASKING POLICY**。

7.13.88 DROP MATERIALIZED VIEW

功能描述

删除数据库中已有的物化视图。

注意事项

物化视图的所有者、物化视图所在模式的所有者、被授予了物化视图DROP权限的用户或拥有DROP ANY TABLE权限的用户才有权限执行DROP MATERIALIZED VIEW命令，系统管理员默认拥有此权限。

语法格式

```
DROP MATERIALIZED VIEW [ IF EXISTS ] mv_name [, ...] [ CASCADE | RESTRICT ];
```


参数说明

- **IF EXISTS**
  如果指定的物化视图不存在，则发出一个notice而不是抛出一个错误。
- **mv_name**
  要删除的物化视图名称。
- **CASCADE | RESTRICT**
  - CASCADE：级联删除依赖此物化视图的对象。
  - RESTRICT：如果有依赖对象存在，则拒绝删除此物化视图。此选项为缺省值。

示例

```
--创建表。
openGauss=# CREATE TABLE my_table (c1 int, c2 int)
WITH(STORAGE_TYPE=ASTORE);

--创建名为my_mv的物化视图。
openGauss=# CREATE MATERIALIZED VIEW my_mv AS SELECT * FROM my_table;

--删除名为my_mv的物化视图。
openGauss=# DROP MATERIALIZED VIEW my_mv;

--删除表。
openGauss=# DROP TABLE my_table;
```

相关链接

ALTER MATERIALIZED VIEW，**CREATE INCREMENTAL MATERIALIZED VIEW**，**CREATE MATERIALIZED VIEW**，**CREATE TABLE**，**REFRESH INCREMENTAL MATERIALIZED VIEW**，**REFRESH MATERIALIZED VIEW**

7.13.89 DROP NODE

功能描述

删除节点。

注意事项

DROP NODE是集群管理工具封装的接口，用来实现集群管理。该接口不建议用户直接使用，以免对集群状态造成影响。管理员用户才有权限使用该接口。

语法格式

```
DROP NODE [ IF EXISTS ] nodename [WITH ( cnnodename [, ... ] )];
```

参数说明

- **IF EXISTS**
  如果指定的节点不存在，则发出一个notice而不是抛出一个错误。
- **nodename**
  要删除的节点名。
  取值范围：已存在的节点nodename。

- **cnnodename**

  CN名称。如果定义了该参数，则除当前连接CN外，还将在该节点上执行。否则，如果是删除DN，将在所有CN上执行；如果是删除CN，将在除待删除CN外所有CN上执行。

  取值范围：已存在的CN的nodename。

相关链接

CREATE NODE，ALTER NODE。

7.13.90 DROP NODE GROUP

功能描述

删除节点组。

注意事项

- DROP NODE GROUP是集群管理工具封装的接口，用来实现集群管理。
- 只有系统管理员或者被授予了节点组DROP权限的用户才能执行该操作。

语法格式

```
DROP NODE GROUP groupname [DISTRIBUTE FROM src_group_name];
```

参数说明

- **groupname**

  要删除的节点组名。

  取值范围：已存在的节点组。

- **DISTRIBUTE FROM src_group_name**

  如果被删除的节点组是从src_group_name Node group节点组重分布过来的，删除该节点组需要指定src_group_name，以便将重分布后的节点分布信息同步到src_group_name指定的Node group节点组。该语句仅仅用于扩容重分布，用户不建议直接使用，以免导致数据分布错误和Node group不可用。

相关链接

CREATE NODE GROUP

7.13.91 DROP OWNED

功能描述

删除一个数据库角色所拥有的数据库对象的权限。

注意事项

- 所有该角色在当前数据库里和共享对象（数据库，表空间）上的所有对象上的权限都将被撤销。

- DROP OWNED常常被用来为移除一个或者多个角色做准备。因为DROP OWNED只影响当前数据库中的对象，通常需要在包含将被移除角色所拥有的对象的每一个数据库中都执行这个命令。
- 使用CASCADE选项可能导致这个命令递归去删除由其他用户所拥有的对象。
- 角色所拥有的数据库、表空间将不会被移除。

语法格式

```
DROP OWNED BY name [, ...] [ CASCADE | RESTRICT ];
```

参数说明

- **name**

  角色名。
- **CASCADE | RESTRICT**
  - CASCADE：级联删除所有依赖于被删除对象的对象。
  - RESTRICT（缺省值）：拒绝删除那些有任何依赖对象存在的对象。

相关链接

REASSIGN OWNED，**DROP ROLE**

7.13.92 DROP PROCEDURE

功能描述

删除已存在的存储过程。

语法格式

```
DROP PROCEDURE [ IF EXISTS ] procedure_name ;
```

参数说明

- **IF EXISTS**

  如果指定的存储过程不存在，发出一个notice而不是抛出一个错误。
- **procedure_name**

  要删除的存储过程名称。

  取值范围：已存在的存储过程名。

示例

请参见CREATE PROCEDURE的**示例**。

相关链接

CREATE PROCEDURE

7.13.93 DROP RESOURCE LABEL

功能描述

删除资源标签。

注意事项

只有poladmin，sysadmin或初始用户才能执行此操作。

语法格式

```
DROP RESOURCE LABEL [IF EXISTS] label_name[, ...];
```

参数说明

- **label_name**
  资源标签名称；
  取值范围：字符串，要符合**标识符命名规范**。

示例

```
--删除一个资源标签。
openGauss=# DROP RESOURCE LABEL IF EXISTS res_label1;

--删除一组资源标签。
openGauss=# DROP RESOURCE LABEL IF EXISTS res_label1, res_label2, res_label3;
```

相关链接

ALTER RESOURCE LABEL，**CREATE RESOURCE LABEL**。

7.13.94 DROP ROLE

功能描述

删除指定的角色。

语法格式

```
DROP ROLE [ IF EXISTS ] role_name [, ...];
```

参数说明

- **IF EXISTS**
  如果指定的角色不存在，则发出一个notice而不是抛出一个错误。
- **role_name**
  要删除的角色名称。
  取值范围：已存在的角色。

示例

```
--创建一个角色，名为manager，密码为********。
openGauss=# CREATE ROLE manager IDENTIFIED BY '********';
```



```
--创建一个角色，从2015年1月1日开始生效，到2026年1月1日失效。
openGauss=# CREATE ROLE miriam WITH LOGIN PASSWORD '********' VALID BEGIN '2015-01-01' VALID
UNTIL '2026-01-01';

--修改角色manager的密码为**********。
openGauss=# ALTER ROLE manager IDENTIFIED BY '**********' REPLACE '********';

--修改角色manager为系统管理员。
openGauss=# ALTER ROLE manager SYSADMIN;

--删除角色manager。
openGauss=# DROP ROLE manager;

--删除角色miriam。
openGauss=# DROP GROUP miriam;
```

相关链接

CREATE ROLE，**ALTER ROLE**，**SET ROLE**

7.13.95 DROP ROW LEVEL SECURITY POLICY

功能描述

删除表上某个行访问控制策略。

注意事项

仅表的所有者或者管理员用户才能删除表的行访问控制策略。

语法格式

```
DROP [ ROW LEVEL SECURITY ] POLICY [ IF EXISTS ] policy_name ON table_name [ CASCADE | RESTRICT ]
```

参数说明

- **IF EXISTS**
  如果指定的行访问控制策略不存在，发出一个notice而不是抛出一个错误。
- **policy_name**
  要删除的行访问控制策略的名称。
- **table_name**
  行访问控制策略所在的数据表名。
- **CASCADE | RESTRICT**
  仅适配此语法，无对象依赖于该行访问控制策略，CASCADE和RESTRICT效果相同。

示例

```
--创建数据表all_data。
openGauss=# CREATE TABLE all_data(id int, role varchar(100), data varchar(100));

--创建行访问控制策略。
openGauss=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role =
CURRENT_USER);

--删除行访问控制策略。
```



```
openGauss=# DROP ROW LEVEL SECURITY POLICY all_data_rls ON all_data;

--删除数据表all_data。
openGauss=# DROP TABLE all_data;
```

相关链接

ALTER ROW LEVEL SECURITY POLICY，CREATE ROW LEVEL SECURITY POLICY

7.13.96 DROP SCHEMA

功能描述

从数据库中删除模式。

注意事项

只有模式的所有者或者被授予了模式DROP权限的用户有权限执行DROP SCHEMA命令，系统管理员默认拥有此权限。

语法格式

```
DROP SCHEMA [ IF EXISTS ] schema_name [, ...] [ CASCADE | RESTRICT ];
```

参数说明

- **IF EXISTS**
  如果指定的模式不存在，发出一个notice而不是抛出一个错误。
- **schema_name**
  模式的名称。
  取值范围：已存在模式名。
- **CASCADE | RESTRICT**
  - CASCADE：自动删除包含在模式中的对象。
  - RESTRICT：如果模式包含任何对象，则删除失败（缺省行为）。

须知

不要随意删除pg_temp或pg_toast_temp开头的模式，这些模式是系统内部使用的，如果删除，可能导致无法预知的结果。

说明

无法删除当前模式。如果要删除当前模式，须切换到其他模式下。

示例

请参见CREATE SCHEMA的示例。

相关链接

ALTER SCHEMA，CREATE SCHEMA。

7.13.97 DROP SEQUENCE

功能描述

从当前数据库里删除序列。

注意事项

序列的所有者、序列所在模式或者被授予了序列DROP权限的用户才能删除，系统管理员默认拥有该权限。

语法格式

```
DROP SEQUENCE [ IF EXISTS ] { [schema.] sequence_name } [ , ... ] [ CASCADE | RESTRICT ];
```

参数说明

- **IF EXISTS**
  如果指定的序列不存在，则发出一个notice而不是抛出一个错误。
- **sequence_name**
  序列名称。
- **CASCADE**
  级联删除依赖序列的对象。
- **RESTRICT**
  如果存在任何依赖的对象，则拒绝删除序列。此项是缺省值。

示例

```
--创建一个名为serial的递增序列，从101开始。
openGauss=# CREATE SEQUENCE serial START 101;

--删除序列。
openGauss=# DROP SEQUENCE serial;
```

相关链接

ALTER SEQUENCE， **DROP SEQUENCE**

7.13.98 DROP SERVER

功能描述

删除现有的一个数据服务器。

注意事项

只有server的所有者或者被授予了server的DROP权限的用户才可以删除，系统管理员默认拥有该权限。

语法格式

```
DROP SERVER [ IF EXISTS ] server_name [ { CASCADE | RESTRICT } ] ;
```


参数描述

- **IF EXISTS**
  如果指定的数据服务器不存在，则发出一个notice而不是抛出一个错误。
- **server_name**
  服务器名称。
- **CASCADE | RESTRICT**
  - CASCADE：级联删除依赖于server的对象 。
  - RESTRICT（缺省值）：如果存在依赖对象，则拒绝删除该server。

示例

```
--创建my_server。
openGauss=# CREATE SERVER my_server FOREIGN DATA WRAPPER file_fdw;

--删除my_server。
openGauss=# DROP SERVER my_server;
```

相关链接

ALTER SERVER，**CREATE SERVER**

7.13.99 DROP SYNONYM

功能描述

删除指定的SYNONYM对象。

注意事项

只有SYNONYM的所有者有权限执行DROP SYNONYM命令，系统管理员默认拥有此权限。

语法格式

```
DROP SYNONYM [ IF EXISTS ] synonym_name [ CASCADE | RESTRICT ];
```

参数描述

- **IF EXISTS**
  如果指定的同义词不存在，则发出一个notice而不是抛出一个错误。
- **synonym_name**
  同义词名字，可以带模式名。
- **CASCADE | RESTRICT**
  - CASCADE：级联删除依赖同义词的对象（比如视图）。
  - RESTRICT：如果有依赖对象存在，则拒绝删除同义词。此选项为缺省值。

示例

请参考CREATE SYNONYM的**示例**。

相关链接

ALTER SYNONYM，CREATE SYNONYM

7.13.100 DROP TABLE

功能描述

删除指定的表。

注意事项

DROP TABLE会强制删除指定的表，删除表后，依赖该表的索引会被删除，而使用到该表的函数和存储过程将无法执行。删除分区表，会同时删除分区表中的所有分区。

表的所有者、表所在模式的所有者、被授予了表的DROP权限的用户或被授予DROP ANY TABLE权限的用户，有权删除指定表，系统管理员默认拥有该权限。

语法格式

```
DROP TABLE [ IF EXISTS ]
  { [schema.]table_name } [, ...] [ CASCADE | RESTRICT ];
```

参数说明

- **IF EXISTS**

  如果指定的表不存在，则发出一个notice而不是抛出一个错误。
- **schema**

  模式名称。
- **table_name**

  表名称。
- **CASCADE | RESTRICT**
  - CASCADE：表示允许级联删除依赖于该表的对象（比如视图、触发器、索引；注：关联的表对象无法被级联删除）。
  - RESTRICT（缺省项）：表示有依赖于该表的对象存在时，该表无法被删除。此选项为缺省值。

示例

请参考CREATE TABLE的示例。

相关链接

ALTER TABLE，CREATE TABLE

7.13.101 DROP TABLESPACE

功能描述

删除一个表空间。

注意事项

- 只有表空间所有者或者被授予了表空间DROP权限的用户有权限执行DROP TABLESPACE命令，系统管理员默认拥有此权限。
- 在删除一个表空间之前，表空间里面不能有任何数据库对象，否则会报错。
- DROP TABLESPACE不支持回滚，因此，不能出现在事务块内部。
- 执行DROP TABLESPACE操作时，如果有另外的会话执行\db查询操作，可能会由于tablespace事务的原因导致查询失败，请重新执行\db查询操作。
- 如果执行DROP TABLESPACE失败，需要再次执行一次DROP TABLESPACE IF EXISTS。

语法格式

```
DROP TABLESPACE [ IF EXISTS ] tablespace_name;
```

参数说明

- **IF EXISTS**

  如果指定的表空间不存在，则发出一个notice而不是抛出一个错误。
- **tablespace_name**

  表空间的名称。

  取值范围：已存在的表空间的名称。

示例

请参见CREATE TABLESPACE的**示例**。

相关链接

ALTER TABLESPACE，**CREATE TABLESPACE**

优化建议

- drop tablespace

  不支持在事务中删除tablespace。

7.13.102 DROP TRIGGER

功能描述

删除触发器。

注意事项

只有触发器的所有者可以执行DROP TRIGGER操作，系统管理员默认拥有此权限。

语法格式

```
DROP TRIGGER [ IF EXISTS ] trigger_name ON table_name [ CASCADE | RESTRICT ];
```


参数说明

- **IF EXISTS**
  如果指定的触发器不存在，则发出一个notice而不是抛出一个错误。
- **trigger_name**
  要删除的触发器名称。
  取值范围：已存在的触发器。
- **table_name**
  要删除的触发器所在的表名称。
  取值范围：已存在的含触发器的表。
- **CASCADE | RESTRICT**
  - CASCADE：级联删除依赖此触发器的对象。
  - RESTRICT：如果有依赖对象存在，则拒绝删除此触发器。此选项为缺省值。

示例

请参见**CREATE TRIGGER**的**示例**。

相关链接

CREATE TRIGGER，**ALTER TRIGGER**，**ALTER TABLE**

7.13.103 DROP TYPE

功能描述

删除一个用户定义的数据类型。

注意事项

只有类型的所有者、被授予了类型DROP权限的用户有权限执行DROP TYPE命令，系统管理员默认拥有此权限。

语法格式

```
DROP TYPE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

参数说明

- **IF EXISTS**
  如果指定的类型不存在，那么发出一个notice而不是抛出一个错误。
- **name**
  要删除的类型名(可以有模式修饰)。
- **CASCADE**
  级联删除依赖该类型的对象(比如字段、函数、操作符等)
  RESTRICT
  如果有依赖对象，则拒绝删除该类型（缺省行为）。

示例

请参考CREATE TYPE的**示例**。

相关链接

CREATE TYPE，**ALTER TYPE**

7.13.104 DROP USER

功能描述

删除用户，同时会删除同名的schema。

注意事项

- 须使用CASCADE级联删除依赖用户的对象（除数据库外）。当删除用户的级联对象时，如果级联对象处于锁定状态，则此级联对象无法被删除，直到对象被解锁或锁定级联对象的进程被终止。
- 在GaussDB中，存在一个配置参数enable_kill_query，此参数在配置文件postgresql.conf中。此参数影响级联删除用户对象的行为：
  - 当参数enable_kill_query为on ，且使用CASCADE模式删除用户时，会自动kill锁定用户级联对象的进程，并删除用户。
  - 当参数enable_kill_query为off，且使用CASCADE模式删除用户时，会等待锁定级联对象的进程结束之后再删除用户。
- 在数据库中删除用户时，如果依赖用户的对象在其他数据库中或者依赖用户的对象是其他数据库，请用户先手动删除其他数据库中的依赖对象或直接删除依赖数据库，再删除用户。即DROP USER不支持跨数据库进行级联删除。
- 在删除用户时，需要先删除该用户拥有的所有对象并且收回该用户在其他对象上的权限，或者通过指定CASCADE级联删除该用户拥有的对象和被授予的权限。
- 在多租户场景下，删除组用户时，业务用户也会同时被删除，如果指定CASCADE级联删除，那么删除业务用户时同时也指定CASCADE。如果在删除某个用户失败时，会报错，同时其他用户也无法成功删除。
- 如果用户下存在创建GDS外表时指定的错误表，则无法通过DROP USER指定CASCADE关键字直接删除用户。

语法格式

```
DROP USER [ IF EXISTS ] user_name [, ...] [ CASCADE | RESTRICT ];
```

参数说明

- **IF EXISTS**

  如果指定的用户不存在，发出一个notice而不是抛出一个错误。
- **user_name**

  待删除的用户名。

  取值范围：已存在的用户名。
- **CASCADE | RESTRICT**
  - CASCADE：级联删除依赖用户的对象，并收回授予该用户的权限。

- RESTRICT：如果用户还有任何依赖的对象或被授予了其他对象的权限，则拒绝删除该用户（缺省行为）。

  说明

  在GaussDB中，存在一个配置参数enable_kill_query，此参数在配置文件postgresql.conf中。此参数影响级联删除用户对象的行为：

  - 当参数enable_kill_query为on，且使用CASCADE模式删除用户时，会自动kill锁定用户级联对象的进程，并删除用户。
  - 当参数enable_kill_query为off，且使用CASCADE模式删除用户时，会等待锁定级联对象的进程结束之后再删除用户。

示例

请参考CREATE USER的**示例**。

相关链接

ALTER USER，**CREATE USER**

7.13.105 DROP VIEW

功能描述

数据库中强制删除已有的视图。

注意事项

视图的所有者、视图所在模式的所有者、被授予了视图DROP权限的用户或拥有DROP ANY TABLE权限的用户，有权限执行DROP VIEW的命令，系统管理员默认拥有此权限。

语法格式

```
DROP VIEW [ IF EXISTS ] view_name [, ...] [ CASCADE | RESTRICT ];
```

参数说明

- **IF EXISTS**

  如果指定的视图不存在，则发出一个notice而不是抛出一个错误。

- **view_name**

  要删除的视图名称。

  取值范围：已存在的视图。

- **CASCADE | RESTRICT**
  - CASCADE：级联删除依赖此视图的对象（比如其他视图）。
  - RESTRICT：如果有依赖对象存在，则拒绝删除此视图。此选项为缺省值。

示例

请参见CREATE VIEW的**示例**。

相关链接

ALTER VIEW，CREATE VIEW

7.13.106 DROP WEAK PASSWORD DICTIONARY

功能描述

清空gs_global_config中的所有弱口令。

注意事项

只有初始用户、系统管理员和安全管理员拥有权限执行本语法。

语法格式

```
DROP WEAK PASSWORD DICTIONARY;
```

参数说明

无。

示例

参见CREATE WEAK PASSWORD DICTIONARY的示例。

相关链接

13.14.82-CREATE WEAK PASSWORD DICTIONARY

7.13.107 EXECUTE

功能描述

执行一个前面准备好的预备语句。因为一个预备语句只在会话的生命期里存在，所以预备语句必须是在当前会话的前些时候用PREPARE语句创建的。

注意事项

如果创建预备语句时，PREPARE语句声明了一些参数，那么传递给EXECUTE语句的必须是一个兼容的参数集，否则会出现错误。

语法格式

```
EXECUTE name [ ( parameter [, ...] ) ];
```

参数说明

- **name**

  要执行的预备语句的名称。

- **parameter**

  给预备语句的参数的具体数值。它必须是一个和生成与创建这个预备语句时指定参数的数据类型相兼容的值的表达式。

示例

```
--创建schema。
openGauss=# CREATE SCHEMA tpcds;

--创建表reason。
openGauss=# CREATE TABLE tpcds.reason (
    CD_DEMO_SK          INTEGER          NOT NULL,
    CD_GENDER           character(16)            ,
    CD_MARITAL_STATUS   character(100)
);

--插入数据。
openGauss=# INSERT INTO tpcds.reason VALUES(51, 'AAAAAAAADDAAAAAA', 'reason 51');

--创建表reason_t1。
openGauss=# CREATE TABLE tpcds.reason_t1 AS TABLE tpcds.reason;

--为一个INSERT语句创建一个预备语句然后执行它。
openGauss=# PREPARE insert_reason(integer,character(16),character(100)) AS INSERT INTO
tpcds.reason_t1 VALUES($1,$2,$3);

openGauss=# EXECUTE insert_reason(52, 'AAAAAAAADDAAAAAA', 'reason 52');

--删除表reason和reason_t1。
openGauss=# DROP TABLE tpcds.reason;
openGauss=# DROP TABLE tpcds.reason_t1;

--删除schema。
openGauss=# DROP SCHEMA tpcds CASCADE;
```

7.13.108 EXECUTE DIRECT

功能描述

在指定的节点上执行SQL语句。一般情况下，SQL语句的执行是由集群负载自动分配到合适的节点上，EXECUTE DIRECT主要用于数据库维护和测试。

注意事项

- 当enable_nonsysadmin_execute_direct=off时，只有系统管理员和监控管理员才能执行EXECUTE DIRECT。
- 为了各个节点上数据的一致性，SQL语句仅支持SELECT，不允许执行事务语句、DDL、DML。
- 使用此类型语句在指定的DN执行stddev聚集计算时，返回结果集是以三元数组形式返回，如{3, 8, 30}，表示count结果为3，sum结果为8，平方和为30。使用此类型语句在指定的DN执行AVG聚集计算时，返回结果集以二元组形式返回，如{4，2}，表示count结果为4，sum结果为2。
- 当指定多个节点时，不支持agg函数，当query中包含agg函数时，会返回“EXECUTE DIRECT on multinode not support agg functions.”
- 由于CN节点不存储用户表数据，不允许指定CN节点执行用户表上的SELECT查询。
- 不允许执行嵌套的EXECUTE DIRECT语句，即执行的SQL语句不能同样是EXECUTE DIRECT语句，此时可直接执行最内层EXECUTE DIRECT语句代替。
- agg函数查询结果与直接在CN上查询不一致，会返回多个信息，不支持array_avg函数。

语法格式

```
EXECUTE DIRECT ON ( nodename [, ... ] ) query ;
EXECUTE DIRECT ON { COORDINATORS | DATANODES | ALL } query;
```

参数说明

- **nodename**
  节点名称。
  取值范围：已存在的节点。
- **query**
  要执行查询语句。
- **COORDINATORS**
  在所有coordinator执行查询语句
- **DATANODES**
  在所有datanode执行查询语句
- **ALL**
  在所有coordinator和datanode执行查询语句

示例

```
--查询当前集群的节点分布状态。
openGauss=# SELECT * FROM pgxc_node;
 node_name   | node_type | node_port |   node_host    | node_port1 |   node_host1   | hostis_primary |
nodeis_primary | nodeis_preferred |    node_id   | sctp_port | control_port | sctp_port1 | control_port1
--------------+-----------+-----------+----------------+------------+----------------+----------------+----------------
+------------------+-------------+-----------+--------------+------------+---------------
 cn_5001      | C         |      8050 | 10.180.155.74  |       8050 | 10.180.155.74  | t              | f              |
f                |  1120683504 |         0 |            0 |          0 |              0
 cn_5003      | C         |      8050 | 10.180.157.130 |       8050 | 10.180.157.130 | t              | f              |
f                |  -125853378 |         0 |            0 |          0 |              0
 dn_6001_6002 | D         |     40050 | 10.180.155.74  |      45050 | 10.146.187.231 | t              | f              |
f                |  1644780306 |     40052 |        40052 |      45052 |          45052
 dn_6003_6004 | D         |     40050 | 10.146.187.231 |      45050 | 10.180.157.130 | t              | f              |
f                |  -966646068 |     40052 |        40052 |      45052 |          45052
 dn_6005_6006 | D         |     40050 | 10.180.157.130 |      45050 | 10.180.155.74  | t              | f              |
f                |   868850011 |     40052 |        40052 |      45052 |          45052
 cn_5002      | C         |      8050 | localhost      |       8050 | localhost      | t              | f              | f              |
-1736975100 |         0 |            0 |          0 |              0
(6 rows)

--查询dn_6001_6002上tpcds.customer_address记录。
openGauss=# EXECUTE DIRECT ON(dn_6001_6002) 'select count(*) from tpcds.customer_address';
 count
-------
 16922
(1 row)

--查询tpcds.customer_address所有记录。
openGauss=# SELECT count(*) FROM tpcds.customer_address;
 count
-------
 50000
(1 row)
```


7.13.109 EXPLAIN

功能描述

显示SQL语句的执行计划。

执行计划将显示SQL语句所引用的表会采用什么样的扫描方式，如：简单的顺序扫描、索引扫描等。如果引用了多个表，执行计划还会显示用到的JOIN算法。

执行计划的最关键的部分是语句的预计执行开销，是指计划生成器估算执行该语句将花费多长的时间。

若指定了ANALYZE选项，则该语句会被执行，然后根据实际的运行结果显示统计数据，包括每个计划节点内时间总开销（毫秒为单位）和实际返回的总行数。这对于判断计划生成器的估计是否接近现实非常有用。

注意事项

在指定ANALYZE选项时，语句会被执行。如果用户想使用EXPLAIN分析INSERT，UPDATE，DELETE，CREATE TABLE AS或EXECUTE语句，而不想改动数据（执行这些语句会影响数据），请使用这种方法：

```
START TRANSACTION;
EXPLAIN ANALYZE ...;
ROLLBACK;
```

语法格式

- 显示SQL语句的执行计划，支持多种选项，对选项顺序无要求。

  ```
  EXPLAIN [ ( option [, ...] ) ] statement;
  ```

  其中选项option子句的语法为：

  ```
  ANALYZE [ boolean ] |
    ANALYSE [ boolean ] |
    VERBOSE [ boolean ] |
    COSTS [ boolean ] |
    CPU [ boolean ] |
    DETAIL [ boolean ] |
    NODES [ boolean ] |
    NUM_NODES [ boolean ] |
    BUFFERS [ boolean ] |
    TIMING [ boolean ] |
    PLAN [ boolean ] |
    FORMAT { TEXT | XML | JSON | YAML }
  ```

- 显示SQL语句的执行计划，且要按顺序给出选项。

  ```
  EXPLAIN { [ ANALYZE | ANALYSE ] [ VERBOSE ] | PERFORMANCE } statement;
  ```

参数说明

- **statement**

  指定要分析的SQL语句。

- **ANALYZE boolean | ANALYSE boolean**

  显示实际运行时间和其他统计数据。当两个参数同时使用时，在option中排在后面的参数生效。

  取值范围：

  - TRUE（缺省值）：显示实际运行时间和其他统计数据。

  - FALSE：不显示。
- **VERBOSE boolean**
  显示有关计划的额外信息。
  取值范围：
  - TRUE（缺省值）：显示额外信息。
  - FALSE：不显示。
- **COSTS boolean**
  包括每个规划节点的估计总成本，以及估计的行数和每行的宽度。
  取值范围：
  - TRUE（缺省值）：显示估计总成本和宽度。
  - FALSE：不显示。
- **CPU boolean**
  打印CPU的使用情况的信息。
  取值范围：
  - TRUE（缺省值）：显示CPU的使用情况。
  - FALSE：不显示。
- **DETAIL boolean**
  打印DN上的信息。
  取值范围：
  - TRUE（缺省值）：打印DN的信息。
  - FALSE：不打印。
- **NODES boolean**
  打印query执行的节点信息。
  取值范围：
  - TRUE（缺省值）：打印执行的节点的信息。
  - FALSE：不打印。
- **NUM_NODES boolean**
  打印执行中的节点的个数信息。
  取值范围：
  - TRUE（缺省值）：打印DN个数的信息。
  - FALSE：不打印。
- **BUFFERS boolean**
  包括缓冲区的使用情况的信息。
  取值范围：
  - TRUE：显示缓冲区的使用情况。
  - FALSE（缺省值）：不显示。
- **TIMING boolean**
  包括实际的启动时间和花费在输出节点上的时间信息。
  取值范围：
  - TRUE（缺省值）：显示启动时间和花费在输出节点上的时间信息。

  - FALSE：不显示。

- **PLAN boolean**

  是否将执行计划存储在PLAN_TABLE中。当该选项开启时，会将执行计划存储在PLAN_TABLE中，不打印到当前屏幕，因此该选项为on时，不能与其他选项同时使用。

  取值范围：

  - TRUE（缺省值）：将执行计划存储在PLAN_TABLE中，不打印到当前屏幕。执行成功返回EXPLAIN SUCCESS。
  - FALSE：不存储执行计划，将执行计划打印到当前屏幕。

- **FORMAT**

  指定输出格式。

  取值范围：TEXT，XML，JSON和YAML。

  默认值：TEXT。

- **PERFORMANCE**

  使用此选项时，即打印执行中的所有相关信息。

示例

```
--创建SCHEMA。
openGauss=# CREATE SCHEMA tpcds;

--创建表tpcds.customer_address。
openGauss=# CREATE TABLE tpcds.customer_address
(
ca_address_sk         INTEGER           NOT NULL,
ca_address_id         CHARACTER(16)     NOT NULL
);

--向表中插入多条记录。
openGauss=# INSERT INTO tpcds.customer_address VALUES (5000, 'AAAAAAAABAAAAAAA'),(10000,
'AAAAAAAACAAAAAAA');

--创建一个表tpcds.customer_address_p1。
openGauss=# CREATE TABLE tpcds.customer_address_p1 AS TABLE tpcds.customer_address;

--修改explain_perf_mode为normal。
openGauss=# SET explain_perf_mode=normal;

--显示表简单查询的执行计划。
openGauss=# EXPLAIN SELECT * FROM tpcds.customer_address_p1;
QUERY PLAN
--------------------------------------------------
Data Node Scan  (cost=0.00..0.00 rows=0 width=0)
Node/s: All datanodes
(2 rows)

--以JSON格式输出的执行计划（explain_perf_mode为normal时）。
openGauss=# EXPLAIN(FORMAT JSON) SELECT * FROM tpcds.customer_address_p1;
              QUERY PLAN
--------------------------------------
[                              +
  {                            +
    "Plan": {                    +
      "Node Type": "Data Node Scan",+
      "Startup Cost": 0.00,        +
      "Total Cost": 0.00,          +
      "Plan Rows": 0,              +
      "Plan Width": 0,             +
      "Node/s": "All datanodes"    +
    }                              +
```



```
  }                          +
 ]
(1 row)

--如果有一个索引，当使用一个带索引WHERE条件的查询，可能会显示一个不同的计划。
openGauss=# EXPLAIN SELECT * FROM tpcds.customer_address_p1 WHERE ca_address_sk=10000;
QUERY PLAN
------------------------------------------------
Data Node Scan  (cost=0.00..0.00 rows=0 width=0)
Node/s: dn_6005_6006
(2 rows)

--以YAML格式输出的执行计划（explain_perf_mode为normal时）。
openGauss=# EXPLAIN(FORMAT YAML) SELECT * FROM tpcds.customer_address_p1 WHERE
ca_address_sk=10000;
          QUERY PLAN
--------------------------------
 - Plan:                      +
     Node Type: "Data Node Scan"+
     Startup Cost: 0.00        +
     Total Cost: 0.00          +
     Plan Rows: 0              +
     Plan Width: 0             +
     Node/s: "dn_6005_6006"
(1 row)

--禁止开销估计的执行计划。
openGauss=# EXPLAIN(COSTS FALSE)SELECT * FROM tpcds.customer_address_p1 WHERE
ca_address_sk=10000;
      QUERY PLAN
------------------------
 Data Node Scan
   Node/s: dn_6005_6006
(2 rows)

--带有聚集函数查询的执行计划。
openGauss=# EXPLAIN SELECT SUM(ca_address_sk) FROM tpcds.customer_address_p1 WHERE
ca_address_sk<10000;
                                QUERY PLAN
-------------------------------------------------------------------------------------
 Aggregate  (cost=18.19..14.32 rows=1 width=4)
   ->  Streaming (type: GATHER)  (cost=18.19..14.32 rows=3 width=4)
         Node/s: All datanodes
         ->  Aggregate  (cost=14.19..14.20 rows=3 width=4)
               ->  Seq Scan on customer_address_p1  (cost=0.00..14.18 rows=10 width=4)
                     Filter: (ca_address_sk < 10000)
(6 rows)

--删除表tpcds.customer_address_p1。
openGauss=# DROP TABLE tpcds.customer_address_p1;

--删除表tpcds.customer_address。
openGauss=# DROP TABLE tpcds.customer_address;

--删除SCHEMA。
openGauss=# DROP SCHEMA tpcds CASCADE;
```

相关链接

ANALYZE | ANALYSE

7.13.110 EXPLAIN PLAN

功能描述

通过EXPLAIN PLAN命令可以将查询执行的计划信息存储于PLAN_TABLE表中。与EXPLAIN命令不同的是，EXPLAIN PLAN仅将计划信息进行存储，而不会打印到屏幕。

语法格式

```
EXPLAIN PLAN
[ SET STATEMENT_ID = name ]
FOR statement ;
```

参数说明

- **name**

  查询标签。

  取值范围：字符串

- EXPLAIN中的PLAN选项表示需要将计划信息存储于PLAN_TABLE中，存储成功将返回“EXPLAIN SUCCESS”。
- 用户可通过STATEMENT_ID对查询设置标签，输入的标签信息也将存储于PLAN_TABLE中。

  说明

  用户在执行EXPLAIN PLAN时，如果没有设置STATEMENT_ID，则默认为空值。同时，用户可输入的STATEMENT_ID最大长度为30个字节，超过长度将会产生报错。

注意事项

- EXPLAIN PLAN不支持在DN上执行。
- 对于执行错误的SQL语句无法进行计划信息的收集。
- PLAN_TABLE中的数据是session级生命周期并且session隔离和用户隔离，用户只能看到当前session、当前用户的数据。
- PLAN_TABLE无法与GDS外表进行关联查询。
- 对于不能下推的查询，无法收集到具体的object信息，object只能收集到REMOTE_QUERY或CTE等信息。详见**示例 2**。

示例 1

使用EXPLAIN PLAN收集SQL语句的执行计划，通常包括以下步骤：

步骤1 执行EXPLAIN PLAN。

说明

执行EXPLAIN PLAN 后会将计划信息自动存储于PLAN_TABLE中，不支持对PLAN_TABLE进行INSERT、UPDATE、ANALYZE等操作。

PLAN_TABLE详细介绍见**PLAN_TABLE**。

```
--创建表foo1,foo2。
openGauss=# CREATE TABLE foo1(f1 int, f2 text, f3 text[]);
openGauss=# CREATE TABLE foo2(f1 int, f2 text, f3 text[]);

--执行explain plan。
openGauss=# EXPLAIN PLAN SET STATEMENT_ID = 'TPCH-Q4' FOR SELECT f1, count(*) FROM foo1 WHERE f1 > 1 AND f1 < 3 AND EXISTS (SELECT * FROM foo2) GROUP BY f1;
```

步骤2 查询PLAN_TABLE。

```
openGauss=# SELECT * FROM plan_table;
```

步骤3 清理PLAN_TABLE表中的数据。


```
openGauss=# DELETE FROM plan_table WHERE STATEMENT_ID = 'TPCH-Q4';
openGauss=# DROP TABLE foo1;
openGauss=# DROP TABLE foo2;
```

----结束

示例 2

对于不能下推的查询，执行explain plan后plan_table中object仅收集到REMOTE_QUERY或CTE等信息。

场景一：优化器生成下发语句的计划，此时仅能收集到REMOTE_QUERY。

```
 explain plan set statement_id = 'test remote query' for
 select
 current_user
 from
 customer;
```

查询PLAN_TABLE。

```
SELECT * FROM PLAN_TABLE;
```

```
   statement_id    | plan_id  | id |   operation    |  options  |   object_name    | object_type  | object_owner |  projection
-------------------+----------+----+----------------+-----------+------------------+--------------+--------------+--------------
 test remote query | 29360133 |  1 | NESTED LOOPS   | CARTESIAN |                  |              |              | 'apple'::name
 test remote query | 29360133 |  2 | DATA NODE SCAN |           | customer         | REMOTE_QUERY |              |
 test remote query | 29360133 |  3 | DATA NODE SCAN |           | customer_address | REMOTE_QUERY |              |
(3 rows)
```

场景二：对于with recursive场景中不能下推的查询，仅能收集到CTE。

关闭enable_stream_recursive，使得查询不能下推。

```
set enable_stream_recursive = off;
```

执行explain plan SQL

```
explain plan set statement_id = 'cte can not be push down'
for
with recursive rq as
(
  select id, name from  chinamap where id = 11
  union all
  select origin.id, rq.name || ' > ' || origin.name
  from rq join chinamap origin on origin.pid = rq.id
)
select id, name from rq order by 1;
```

查询PLAN_TABLE。

```
SELECT * FROM PLAN_TABLE;
```

```
       statement_id       | plan_id  | id | operation | options | object_name | object_type | object_owner |   projection
--------------------------+----------+----+-----------+---------+-------------+-------------+--------------+----------------
 cte can not be push down | 25166071 |  1 | SORT      |         |             |             |              | rq.id, rq.name
 cte can not be push down | 25166071 |  2 | CTE SCAN  |         | rq          | CTE         |              | rq.id, rq.name
(2 rows)
```

7.13.111 FETCH

功能描述

FETCH通过已创建的游标来检索数据。

每个游标都有一个供FETCH使用的关联位置。游标的关联位置可以在查询结果的第一行之前，或者在结果中的任意行，或者在结果的最后一行之后：

- 游标刚创建完之后，关联位置在第一行之前。
- 在抓取了一些移动行之后，关联位置在检索到的最后一行上。

- 如果FETCH抓取完了所有可用行，它就停在最后一行后面，或者在反向抓取的情况下是停在第一行前面。
- FETCH ALL或FETCH BACKWARD ALL总是把游标的关联位置放在最后一行或者在第一行前面。

注意事项

- 如果游标定义了NO SCROLL，则不允许使用例如FETCH BACKWARD之类的反向抓取。
- NEXT，PRIOR，FIRST，LAST，ABSOLUTE，RELATIVE形式在恰当地移动游标之后抓取一条记录。如果后面没有数据行，就返回一个空的结果，此时游标就会停在查询结果的最后一行之后（向后查询时）或者第一行之前（向前查询时）。
- FORWARD和BACKWARD形式在向前或者向后移动的过程中抓取指定的行数，然后把游标定位在最后返回的行上；或者，如果count大于可用的行数，则在所有行之后（向后查询时）或者之前（向前查询时）。
- RELATIVE 0，FORWARD 0，BACKWARD 0都要求在不移动游标的前提下抓取当前行，也就是重新抓取最近刚抓取过的行。除非游标定位在第一行之前或者最后一行之后，否则这个动作都应该成功。而当游标定位在第一行之前或者最后一行之后，不返回任何行。
- 当FETCH的游标上涉及非系统表时，不支持BACKWARD、PRIOR、FIRST等涉及反向获取操作。

语法格式

```
FETCH [ direction { FROM | IN } ] cursor_name;
```

其中direction子句为可选参数。

```
NEXT
  | PRIOR
  | FIRST
  | LAST
  | ABSOLUTE count
  | RELATIVE count
  | count
  | ALL
  | FORWARD
  | FORWARD count
  | FORWARD ALL
  | BACKWARD
  | BACKWARD count
  | BACKWARD ALL
```

参数说明

- **direction**

  定义抓取数据的方向。

  取值范围：

  - NEXT（缺省值）

    从当前关联位置开始，抓取下一行。
  - PRIOR

    从当前关联位置开始，抓取上一行。
  - FIRST

    抓取查询的第一行（和ABSOLUTE 1相同）。

- LAST
  抓取查询的最后一行（和ABSOLUTE -1相同）。
- ABSOLUTE count
  抓取查询中第count行。
  ABSOLUTE抓取不会比用相对位移移动到需要的数据行更快，因为下层的实现必须遍历所有中间的行。
  count取值范围：有符号的整数
  - count为正数，就从查询结果的第一行开始，抓取第count行。当count小于当前游标位置时，涉及到rewind操作，暂不支持。
  - count为负数或0，涉及到反向扫描操作，暂不支持。
- RELATIVE count
  从当前关联位置开始，抓取随后或前面的第count行。
  取值范围：有符号的整数
  - count为正数就抓取当前关联位置之后的第count行。
  - count为负数或0，涉及到反向扫描操作，暂不支持。
  - 如果当前行没有数据的话，RELATIVE 0返回空。
- count
  抓取随后的count行（和FORWARD count一样）。
- ALL
  从当前关联位置开始，抓取所有剩余的行（和FORWARD ALL一样）。
- FORWARD
  抓取下一行（和NEXT一样）。
- FORWARD count
  与RELATIVE count的效果相同，从当前关联位置开始，抓取随后或前面的第count行。
- FORWARD ALL
  从当前关联位置开始，抓取所有剩余行。
- BACKWARD
  从当前关联位置开始，抓取前面一行(和PRIOR一样) 。
- BACKWARD count
  从当前关联位置开始，抓取前面的count行（向后扫描）。
  取值范围：有符号的整数
  - count为正数就抓取当前关联位置之前的第count行。
  - count为负数就抓取当前关联位置之后的第abs（count）行。
  - 如果有数据的话，BACKWARD 0重新抓取当前行。
- BACKWARD ALL
  从当前关联位置开始，抓取所有前面的行（向后扫描） 。

- **{ FROM | IN } cursor_name**

使用关键字FROM或IN指定游标名称。

取值范围：已创建的游标的名称。

示例

```
--创建一个SCHEMA。
openGauss=# CREATE SCHEMA tpcds;

--创建表tpcds.customer_address。
openGauss=# CREATE TABLE tpcds.customer_address
(
ca_address_sk         INTEGER           NOT NULL,
ca_address_id         CHARACTER(16)     NOT NULL,
ca_street_number      INTEGER                   ,
ca_street_name        CHARACTER (20)
);

--向表中插入多条记录。
openGauss=# INSERT INTO tpcds.customer_address VALUES (1, 'AAAAAAAABAAAAAAA', '18', 'Jackson'),(2,
'AAAAAAAACAAAAAAA', '362', 'Washington 6th'),(3, 'AAAAAAAADAAAAAAA', '585', 'Dogwood Washington');

--SELECT语句，用一个游标读取一个表。开始一个事务。
openGauss=# START TRANSACTION;

--建立一个名为cursor1的游标。
openGauss=# CURSOR cursor1 FOR SELECT * FROM tpcds.customer_address ORDER BY 1;

--抓取头3行到游标cursor1里。
openGauss=# FETCH FORWARD 3 FROM cursor1;
 ca_address_sk |  ca_address_id   | ca_street_number |   ca_street_name
---------------+------------------+------------------+--------------------
             1 | AAAAAAAABAAAAAAA | 18               | Jackson
             2 | AAAAAAAACAAAAAAA | 362              | Washington 6th
             3 | AAAAAAAADAAAAAAA | 585              | Dogwood Washington
(3 rows)

--关闭游标并提交事务。
openGauss=# CLOSE cursor1;

--结束一个事务。
openGauss=# END;

--VALUES子句，用一个游标读取VALUES子句中的内容。开始一个事务。
openGauss=# START TRANSACTION;

--建立一个名为cursor2的游标。
openGauss=# CURSOR cursor2 FOR VALUES(1,2),(0,3) ORDER BY 1;

--抓取头2行到游标cursor2里。
openGauss=# FETCH FORWARD 2 FROM cursor2;
column1 | column2
---------+---------
0 |       3
1 |       2
(2 rows)

--关闭游标并提交事务。
openGauss=# CLOSE cursor2;

--结束一个事务。
openGauss=# END;

--WITH HOLD游标的使用，开启事务。
openGauss=# START TRANSACTION;

--创建一个with hold游标。
openGauss=# DECLARE cursor1 CURSOR WITH HOLD FOR SELECT * FROM tpcds.customer_address ORDER
BY 1;
```



```
--抓取头2行到游标cursor1里。
openGauss=# FETCH FORWARD 2 FROM cursor1;
 ca_address_sk |  ca_address_id   | ca_street_number |   ca_street_name
---------------+------------------+------------------+--------------------
             1 | AAAAAAAABAAAAAAA | 18               | Jackson
             2 | AAAAAAAACAAAAAAA | 362              | Washington 6th
(2 rows)

--结束事务。
openGauss=# END;

--抓取下一行到游标cursor1里。
openGauss=# FETCH FORWARD 1 FROM cursor1;
 ca_address_sk |  ca_address_id   | ca_street_number |   ca_street_name
---------------+------------------+------------------+--------------------
             3 | AAAAAAAADAAAAAAA | 585              | Dogwood Washington
(1 row)

--关闭游标。
openGauss=# CLOSE cursor1;

--删除表tpcds.customer_address。
openGauss=# DROP TABLE tpcds.customer_address;

--删除SCHEMA。
openGauss=# DROP SCHEMA tpcds CASCADE;
```

相关链接

CLOSE，**MOVE**

7.13.112 GRANT

功能描述

对角色和用户进行授权操作。

使用GRANT命令进行用户授权包括以下场景：

- **将系统权限授权给角色或用户**

  系统权限又称为用户属性，包括SYSADMIN、CREATEDB、CREATEROLE、AUDITADMIN、MONADMIN、OPRADMIN、POLADMIN、INHERIT、REPLICATION、VCADMIN和LOGIN等。

  系统权限一般通过CREATE/ALTER ROLE语法来指定。其中，SYSADMIN权限可以通过GRANT/REVOKE ALL PRIVILEGE授予或撤销。但系统权限无法通过ROLE和USER的权限被继承，也无法授予PUBLIC。

- **将数据库对象授权给角色或用户**

  将数据库对象（表、视图、指定字段、数据库、函数、模式、表空间等）的相关权限授予特定角色或用户；

  GRANT命令将数据库对象的特定权限授予一个或多个角色。这些权限会追加到已有的权限上。

  关键字PUBLIC表示该权限要赋予所有角色，包括以后创建的用户。PUBLIC可以看做是一个隐含定义好的组，它总是包括所有角色。任何角色或用户都将拥有通过GRANT直接赋予的权限和所属的权限，再加上PUBLIC的权限。

  如果声明了WITH GRANT OPTION，则被授权的用户也可以将此权限赋予他人，否则就不能授权给他人。这个选项不能赋予PUBLIC，这是GaussDB特有的属性。

GaussDB会将某些类型的对象上的权限授予PUBLIC。默认情况下，对表、表字段、序列、外部数据源、外部服务器、模式或表空间对象的权限不会授予PUBLIC，而以下这些对象的权限会授予PUBLIC：数据库的CONNECT权限和CREATE TEMP TABLE权限、函数的EXECUTE特权、语言和数据类型（包括域）的USAGE特权。对象拥有者可以撤销默认授予PUBLIC的权限并专门授予权限给其他用户。为了更安全，建议在同一个事务中创建对象并设置权限，这样其他用户就没有时间窗口使用该对象。这些初始的默认权限可以使用ALTER DEFAULT PRIVILEGES命令修改。

对象的所有者缺省具有该对象上的所有权限，出于安全考虑所有者可以舍弃部分权限，但ALTER、DROP、COMMENT、INDEX、VACUUM以及对象的可再授予权限属于所有者固有的权限，隐式拥有。

- **将角色或用户的权限授权给其他角色或用户**

  将一个角色或用户的权限授予一个或多个其他角色或用户。在这种情况下，每个角色或用户都可视为拥有一个或多个数据库权限的集合。

  当声明了WITH ADMIN OPTION，被授权的用户可以将该权限再次授予其他角色或用户，以及撤销所有由该角色或用户继承到的权限。当授权的角色或用户发生变更或被撤销时，所有继承该角色或用户权限的用户拥有的权限都会随之发生变更。

  数据库系统管理员可以给任何角色或用户授予/撤销任何权限。拥有CREATEROLE权限的角色可以赋予或者撤销任何非系统管理员角色的权限。

- **将ANY权限授予给角色或用户**

  将ANY权限授予特定的角色和用户，ANY权限的取值范围参见语法格式。当声明了WITH ADMIN OPTION，被授权的用户可以将该ANY权限再次授予其他角色/用户，或从其他角色/用户处回收该ANY权限。ANY权限可以通过角色被继承，但不能赋予PUBLIC。初始用户和三权分立关闭时的系统管理员用户可以给任何角色/用户授予或撤销ANY权限。

  目前支持以下ANY权限：CREATE ANY TABLE、ALTER ANY TABLE、DROP ANY TABLE、SELECT ANY TABLE、INSERT ANY TABLE、UPDATE ANY TABLE、DELETE ANY TABLE、CREATE ANY SEQUENCE、CREATE ANY INDEX、CREATE ANY FUNCTION、EXECUTE ANY FUNCTION、 CREATE ANY PACKAGE、EXECUTE ANY PACKAGE、CREATE ANY TYPE。详细的ANY权限范围描述参考**表7-114**。

注意事项

- 不允许将ANY权限授予PUBLIC，也不允许从PUBLIC回收ANY权限。
- ANY权限属于数据库内的权限，只对授予该权限的数据库内的对象有效，例如SELECT ANY TABLE只允许用户查看当前数据库内的所有用户表数据，对其他数据库内的用户表无查看权限。
- ANY权限与原有的权限相互无影响。
- 如果用户被授予CREATE ANY TABLE权限，在同名schema下创建表的属主是该schema的创建者，用户对表进行其他操作时，需要授予相应的操作权限。
- 需要谨慎授予用户CREATE ANY FUNCTION或CREATE ANY PACKAGE的权限，以免其他用户利用DEFINER类型的函数或PACKAGE进行权限提升。

语法格式

- 将表或视图的访问权限赋予指定的用户或角色。

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER | ALTER | DROP | COMMENT | INDEX | VACUUM } [, ...]
```



```
    | ALL [ PRIVILEGES ] }
  ON { [ TABLE ] table_name [, ...]
    | ALL TABLES IN SCHEMA schema_name [, ...] }
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

- 将表中字段的访问权限赋予指定的用户或角色。

```
GRANT { {{ SELECT | INSERT | UPDATE | REFERENCES | COMMENT } ( column_name [, ...] )} [, ...]
    | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
  ON [ TABLE ] table_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

- 将序列的访问权限赋予指定的用户或角色。

```
GRANT { { SELECT | UPDATE | USAGE | ALTER | DROP | COMMENT } [, ...]
    | ALL [ PRIVILEGES ] }
  ON { [ SEQUENCE ] sequence_name [, ...]
    | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

- 将数据库的访问权限赋予指定的用户或角色。

```
GRANT { { CREATE | CONNECT | TEMPORARY | TEMP | ALTER | DROP | COMMENT } [, ...]
    | ALL [ PRIVILEGES ] }
  ON DATABASE database_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

- 将域的访问权限赋予指定的用户或角色。

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
  ON DOMAIN domain_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

说明

本版本暂时不支持赋予域的访问权限。

- 将外部数据源的访问权限赋予给指定的用户或角色。

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
  ON FOREIGN DATA WRAPPER fdw_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

- 将外部服务器的访问权限赋予给指定的用户或角色。

```
GRANT { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
  ON FOREIGN SERVER server_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

- 将函数的访问权限赋予给指定的用户或角色。

```
GRANT { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
  ON { FUNCTION {function_name ( [ {[ argmode ] [ arg_name ] arg_type} [, ...] ] )} [, ...]
    | ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

- 将存储过程的访问权限赋予给指定的用户或角色。

```
GRANT { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
  ON { PROCEDURE {proc_name ( [ {[ argmode ] [ arg_name ] arg_type} [, ...] ] )} [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

- 将过程语言的访问权限赋予给指定的用户或角色。

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
  ON LANGUAGE lang_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

说明

Java和Internal只支持拥有sysadmin权限的用户创建。

- 将子集群的访问权限赋予指定的用户或角色。

```
GRANT { { CREATE | USAGE | COMPUTE | ALTER | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON NODE GROUP group_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

  说明

  将子集群的create权限赋予指定用户或角色时，会默认把usage和compute权限赋予指定用户或角色。

- 将模式的访问权限赋予指定的用户或角色。

```
GRANT { { CREATE | USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON SCHEMA schema_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

  说明

  将模式中的表或者视图对象授权给其他用户时，需要将表或视图所属的模式的USAGE权限同时授予该用户，若没有该权限，则只能看到这些对象的名称，并不能实际进行对象访问。

- 将大对象的访问权限赋予指定的用户或角色。

```
GRANT { { SELECT | UPDATE } [, ...] | ALL [ PRIVILEGES ] }
    ON LARGE OBJECT loid [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

  说明

  本版本暂时不支持大对象。

- 将表空间的访问权限赋予指定的用户或角色。

```
GRANT { { CREATE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON TABLESPACE tablespace_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

- 将类型的访问权限赋予指定的用户或角色。

```
GRANT { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON TYPE type_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

  说明

  本版本暂时不支持赋予类型的访问权限。

- 将directory对象的权限赋予指定的角色。

```
GRANT { { READ | WRITE | ALTER | DROP } [, ...] | ALL [PRIVILEGES] }
  ON DIRECTORY directory_name [, ...]
  TO { [GROUP] role_name | PUBLIC } [, ...]
  [WITH GRANT OPTION];
```

- 将package对象的权限赋予指定的角色。

```
GRANT { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [PRIVILEGES] }
  ON PACKAGE package_name [, ...]
  TO { [GROUP] role_name | PUBLIC } [, ...]
  [WITH GRANT OPTION];
```

  说明

  本版本暂不支持package对象。

- 将角色的权限赋予其他用户或角色的语法。

```
GRANT role_name [, ...]
  TO role_name [, ...]
  [ WITH ADMIN OPTION ];
```


- 将sysadmin权限赋予指定的角色。
```
GRANT ALL { PRIVILEGES | PRIVILEGE }
  TO role_name;
```
- 将ANY权限赋予其他用户或角色的语法。
```
GRANT { CREATE ANY TABLE | ALTER ANY TABLE | DROP ANY TABLE | SELECT ANY TABLE | INSERT
ANY TABLE | UPDATE ANY TABLE |
  DELETE ANY TABLE | CREATE ANY SEQUENCE | CREATE ANY INDEX | CREATE ANY FUNCTION |
EXECUTE ANY FUNCTION |
  CREATE ANY PACKAGE | EXECUTE ANY PACKAGE | CREATE ANY TYPE } [, ...]
  TO [ GROUP ] role_name [, ...]
  [ WITH ADMIN OPTION ];
```

说明

本版本暂不支持package对象。

参数说明

GRANT的权限分类如下所示。

- **SELECT**

  允许对指定的表、视图、序列执行SELECT语句。
- **INSERT**

  允许对指定的表执行INSERT语句。
- **UPDATE**

  允许对声明的表中任意字段执行UPDATE语句。SELECT… FOR UPDATE和SELECT… FOR SHARE除了需要SELECT权限外，还需要UPDATE权限。
- **DELETE**

  允许执行DELETE语句删除指定表中的数据。
- **TRUNCATE**

  允许执行TRUNCATE语句删除指定表中的所有记录。
- **REFERENCES**

  创建一个外键约束，必须拥有参考表和被参考表的REFERENCES权限，分布式场景暂不支持。
- **TRIGGER**

  允许在指定的表上创建触发器。
- **CREATE**
  - 对于数据库，允许在数据库里创建新的模式。
  - 对于模式，允许在模式中创建新的对象。如果要重命名一个对象，用户除了必须是该对象的所有者外，还必须拥有该对象所在模式的CREATE权限。
  - 对于表空间，允许在表空间中创建表，允许在创建数据库和模式的时候把该表空间指定为缺省表空间。
  - 对于子集群，允许在子集群中创建表对象。
- **CONNECT**

  允许用户连接到指定的数据库。
- **EXECUTE**

  允许使用指定的函数，以及利用这些函数实现的操作符。

- **USAGE**
  - 对于过程语言，允许用户在创建函数的时候指定过程语言。
  - 对于模式，USAGE允许访问包含在指定模式中的对象，若没有该权限，则只能看到这些对象的名称。
  - 对于序列，USAGE允许使用nextval函数。
  - 对于子集群， 对包含在指定模式中的对象有访问权限时，USAGE允许访问指定子集群下的表对象。
  - 对于密钥对象，USAGE是使用密钥的权限。
- **COMPUTE**

  针对计算子集群，允许用户在具有compute权限的计算子集群上进行弹性计算。
- **ALTER**

  允许用户修改指定对象的属性，但不包括修改对象的所有者和修改对象所在的模式。
- **DROP**

  允许用户删除指定的对象。
- **COMMENT**

  允许用户定义或修改指定对象的注释。
- **INDEX**

  允许用户在指定表上创建索引，并管理指定表上的索引，还允许用户对指定表执行REINDEX和CLUSTER操作。
- **VACUUM**

  允许用户对指定的表执行ANALYZE和VACUUM操作。
- **ALL PRIVILEGES**

  一次性给指定用户/角色赋予所有可赋予的权限。只有系统管理员有权执行GRANT ALL PRIVILEGES。

GRANT的参数说明如下所示。

- **role_name**

  已存在用户名称。
- **table_name**

  已存在表名称。
- **column_name**

  已存在字段名称。
- **schema_name**

  已存在模式名称。
- **database_name**

  已存在数据库名称。
- **function_name**

  已存在函数名称。
- **sequence_name**

  已存在序列名称。

- **domain_name**
  已存在域类型名称。
- **fdw_name**
  已存在外部数据包名称。
- **lang_name**
  已存在语言名称。
- **type_name**
  已存在类型名称。
- **group_name**
  已存在的子集群名称。
- **argmode**
  参数模式。
  取值范围：字符串，要符合**标识符命名规范**。
- **arg_name**
  参数名称。
  取值范围：字符串，要符合**标识符命名规范**。
- **arg_type**
  参数类型。
  取值范围：字符串，要符合**标识符命名规范**。
- **loid**
  包含本页的大对象的标识符。
  取值范围：字符串，要符合**标识符命名规范**。
- **tablespace_name**
  表空间名称。
- **client_master_key**
  客户端加密主密钥的名称。
  取值范围：字符串，要符合**标识符命名规范**。
- **column_encryption_key**
  列加密密钥的名称。
  取值范围：字符串，要符合**标识符命名规范**。
- **directory_name**
  目录名称。
  取值范围：字符串，要符合**标识符命名规范**。
- **WITH GRANT OPTION**
  如果声明了WITH GRANT OPTION，则被授权的用户也可以将此权限赋予他人，否则就不能授权给他人。这个选项不能赋予PUBLIC。

非对象所有者给其他用户授予对象权限时，命令按照以下规则执行：

- 如果用户没有该对象上指定的权限，命令立即失败。
- 如果用户有该对象上的部分权限，则GRANT命令只授予他有授权选项的权限。

- 如果用户没有可用的授权选项，GRANT ALL PRIVILEGES形式将发出一个警告信息，其他命令形式将发出在命令中提到的且没有授权选项的相关警告信息。

说明

数据库系统管理员可以访问所有对象，而不会受对象的权限设置影响。除了必要的情况外，建议不要总是以系统管理员身份进行操作。

- **WITH ADMIN OPTION**

  对于角色，当声明了WITH ADMIN OPTION，被授权的用户可以将该角色再授予其他角色/用户，或从其他角色/用户回收该角色。

  对于ANY权限，当声明了WITH ADMIN OPTION，被授权的用户可以将该ANY权限再授予其他角色/用户，或从其他角色/用户回收该ANY权限。

表 7-114 ANY 权限列表

ANY权限名称	描述
CREATE ANY TABLE	用户能够在public模式和用户模式下创建表或视图。如果想要创建serial类型列的表，还需要授予创建序列的权限。
ALTER ANY TABLE	用户拥有对public模式和用户模式下表或视图的ALTER权限。如果想要修改表的唯一索引为表增加主键约束或唯一约束，还需要授予该表的索引权限。
DROP ANY TABLE	用户拥有对public模式和用户模式下表或视图的DROP权限。
SELECT ANY TABLE	用户拥有对public模式和用户模式下表或视图的SELECT权限，仍然受行级访问控制限制。
UPDATE ANY TABLE	用户拥有对public模式和用户模式下表或视图的UPDATE权限，仍然受行级访问控制限制。
INSERT ANY TABLE	用户拥有对public模式和用户模式下表或视图的INSERT权限。
DELETE ANY TABLE	用户拥有对public模式和用户模式下表或视图的DELETE权限，仍然受行级访问控制限制。
CREATE ANY FUNCTION	用户能够在用户模式下创建函数或存储过程。
EXECUTE ANY FUNCTION	用户拥有在public模式和用户模式下函数或存储过程的EXECUTE权限。
CREATE ANY PACKAGE	本版本暂不支持package对象。
EXECUTE ANY PACKAGE	本版本暂不支持package对象。
CREATE ANY TYPE	用户能够在public模式和用户模式下创建类型。
CREATE ANY SEQUENCE	用户能够在public模式和用户模式下创建序列。

ANY权限名称	描述
CREATE ANY INDEX	用户能够在public模式和用户模式下创建索引。如果在某表空间创建分区表索引，需要授予用户该表空间的创建权限。

说明

用户被授予任何一种ANY权限后，用户对public模式和用户模式具有USAGE权限，对表13-1中除public之外的系统模式没有USAGE权限。

示例

示例：将系统权限授权给用户或者角色。

创建名为joe的用户，并将sysadmin权限授权给他。

```
openGauss=# CREATE USER joe PASSWORD 'xxxxxxxxxx';
openGauss=# GRANT ALL PRIVILEGES TO joe;
```

授权成功后，用户joe会拥有sysadmin的所有权限。

示例：将对象权限授权给用户或者角色。

1. 撤销joe用户的sysadmin权限，然后将模式tpcds的使用权限和表tpcds.reason的所有权限授权给用户joe。

   ```
   openGauss=# CREATE SCHEMA tpcds;
   openGauss=# CREATE TABLE tpcds.reason
   (
   r_reason_sk         INTEGER      NOT NULL,
   r_reason_id         CHAR(16)     NOT NULL,
   r_reason_desc       VARCHAR(20)
   );
   openGauss=# REVOKE ALL PRIVILEGES FROM joe;
   openGauss=# GRANT USAGE ON SCHEMA tpcds TO joe;
   openGauss=# GRANT ALL PRIVILEGES ON tpcds.reason TO joe;
   ```

   授权成功后，joe用户就拥有了tpcds.reason表的所有权限，包括增删改查等权限。

2. 将tpcds.reason表中r_reason_sk、r_reason_id、r_reason_desc列的查询权限，r_reason_desc的更新权限授权给joe。

   ```
   openGauss=# GRANT select (r_reason_sk,r_reason_id,r_reason_desc),update (r_reason_desc) ON tpcds.reason TO joe;
   ```

   授权成功后，用户joe对tpcds.reason表中r_reason_sk，r_reason_id，r_reason_desc的查询权限会立即生效。如果joe用户需要拥有将这些权限授权给其他用户的权限，可以通过以下语法对joe用户进行授权。

   ```
   openGauss=# GRANT select (r_reason_sk, r_reason_id) ON tpcds.reason TO joe WITH GRANT OPTION;
   ```

   将数据库testdb的连接权限授权给用户joe，并给予其在testdb中创建schema的权限，而且允许joe将此权限授权给其他用户。

   ```
   openGauss=# CREATE DATABASE testdb;
   openGauss=# GRANT create,connect on database testdb TO joe WITH GRANT OPTION;
   ```

   创建角色tpcds_manager，将模式tpcds的访问权限授权给角色tpcds_manager，并授予该角色在tpcds下创建对象的权限，不允许该角色中的用户将权限授权给其他人。

   ```
   openGauss=# CREATE ROLE tpcds_manager PASSWORD 'xxxxxxxxxx';
   openGauss=# GRANT USAGE,CREATE ON SCHEMA tpcds TO tpcds_manager;
   ```


将表空间tpcds_tbspc的所有权限授权给用户joe，但用户joe无法将权限继续授予其他用户。

```
openGauss=# CREATE TABLESPACE tpcds_tbspc RELATIVE LOCATION 'tablespace/tablespace_1';
openGauss=# GRANT ALL ON TABLESPACE tpcds_tbspc TO joe;
```

示例：将用户或者角色的权限授权给其他用户或角色。

1. 创建角色manager，将joe的权限授权给manager，并允许该角色将权限授权给其他人。
   ```
   openGauss=# CREATE ROLE manager PASSWORD 'xxxxxxxxxx';
   openGauss=# GRANT joe TO manager WITH ADMIN OPTION;
   ```
2. 创建用户senior_manager，将用户manager的权限授权给该用户。
   ```
   openGauss=# CREATE ROLE senior_manager PASSWORD 'xxxxxxxxxx';
   openGauss=# GRANT manager TO senior_manager;
   ```
3. 撤销权限，并清理用户。
   ```
   openGauss=# REVOKE joe FROM manager;
   openGauss=# REVOKE manager FROM senior_manager;
   openGauss=# DROP USER manager;
   ```

示例：撤销上述授予的权限，并清理角色和用户。

```
openGauss=# REVOKE ALL PRIVILEGES ON tpcds.reason FROM joe;
openGauss=# REVOKE ALL PRIVILEGES ON SCHEMA tpcds FROM joe;
openGauss=# REVOKE ALL ON TABLESPACE tpcds_tbspc FROM joe;
openGauss=# DROP TABLESPACE tpcds_tbspc;
openGauss=# REVOKE USAGE,CREATE ON SCHEMA tpcds FROM tpcds_manager;
openGauss=# DROP ROLE tpcds_manager;
openGauss=# DROP ROLE senior_manager;
openGauss=# DROP USER joe CASCADE;
openGauss=# DROP TABLE tpcds.reason;
openGauss=# DROP SCHEMA tpcds CASCADE;
```

相关链接

REVOKE，**ALTER DEFAULT PRIVILEGES**

7.13.113 INSERT

功能描述

向表中添加一行或多行数据。

注意事项

- 表的所有者、拥有表INSERT权限的用户或拥有INSERT ANY TABLE权限的用户，可向表中插入数据，系统管理员默认拥有此权限。
- 如果使用RETURNING子句，用户必须要有该表的SELECT权限。
- 如果使用ON DUPLICATE KEY UPDATE，用户必须要有该表的INSERT、UPDATE权限，UPDATE子句中列的SELECT权限。
- 如果使用QUERY子句插入来自查询里的数据行，用户还需要拥有在查询里使用的表的SELECT权限。
- 如果使用QUERY子句插入来自查询动态数据脱敏列的数据，插入的结果即为脱敏后的值，无法被还原。
- 当连接到TD兼容的数据库时，td_compatible_truncation参数设置为on时，将启用超长字符串自动截断功能，在后续的INSERT语句中（不包含外表的场景下），对目标表中char和varchar类型的列上插入超长字符串时，系统会自动按照目标表中相应列定义的最大长度对超长字符串进行截断。

> **说明**
>
> 如果向字符集为字节类型编码（SQL_ASCII，LATIN1等）的数据库中插入多字节字符数据（如汉字等），且字符数据跨越截断位置，这种情况下，按照字节长度自动截断，自动截断后会在尾部产生非预期结果。如果用户有对于截断结果正确性的要求，建议用户采用UTF8等能够按照字符截断的输入字符集作为数据库的编码集。

语法格式

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
INSERT [/*+ plan_hint */] INTO table_name [ AS alias ] [ ( column_name [, ...] ) ]
    { DEFAULT VALUES
    | VALUES {( { expression | DEFAULT } [, ...] ) }[, ...]
    | query }
    [ ON DUPLICATE KEY UPDATE { NOTHING | { column_name = { expression | DEFAULT } } [, ...] [ WHERE condition ] } ]
    [ RETURNING {* | {output_expression [ [ AS ] output_name ] }[, ...]} ];
```

参数说明

- **WITH [RECURSIVE] with_query [, ...]**

  用于声明一个或多个可以在主查询中通过名称引用的子查询，相当于临时表。

  如果声明了RECURSIVE，那么允许SELECT子查询通过名称引用它自己。

  其中with_query的详细格式为：

  ```
  with_query_name [ ( column_name [, ...] ) ] AS [ [ NOT ] MATERIALIZED ]
  ( {SELECT | VALUES | INSERT | UPDATE | DELETE} )
  ```

  – with_query_name指定子查询生成的结果集名称，在查询中可使用该名称访问子查询的结果集。

  – column_name指定子查询结果集中显示的列名。

  – 每个子查询可以是SELECT，VALUES，INSERT，UPDATE或DELETE语句。

  – 用户可以使用MATERIALIZED / NOT MATERIALIZED对CTE进行修饰。

  - 如果声明为MATERIALIZED，WITH查询将被物化，生成一个子查询结果集的拷贝，在引用处直接查询该拷贝，因此WITH子查询无法和主干SELECT语句进行联合优化（如谓词下推、等价类传递等），对于此类场景可以使用NOT MATERIALIZED进行修饰，如果WITH查询语义上可以作为子查询内联执行，则可以进行上述优化。
  - 如果用户没有显示声明物化属性则遵守以下规则：如果CTE只在所属主干语句中被引用一次，且语义上支持内联执行，则会被改写为子查询内联执行，否则以CTE Scan的方式物化执行。

  > **说明**
  >
  > INSERT ON DUPLICATE KEY UPDATE不支持WITH及WITH RECURSIVE子句。

- **plan_hint子句**

  以/*+ */的形式在INSERT关键字后，用于对INSERT对应的语句块生成的计划进行hint调优，详细用法请参见章节**使用Plan Hint进行调优**。每条语句中只有第一个/*+ plan_hint */注释块会作为hint生效，里面可以写多条hint。

- **table_name**

  要插入数据的目标表名。

  取值范围：已存在的表名。

- **column_name**

  目标表中的字段名：

  - 字段名可以使用子字段名或者数组下标修饰。
  - 没有在字段列表中出现的每个字段，将由系统默认值，或者声明时的默认值填充，若都没有则用NULL填充。例如，向一个复合类型中的某些字段插入数据的话，其他字段将是NULL。
  - 目标字段（column_name）可以按顺序排列。如果没有列出任何字段，则默认全部字段，且顺序为表声明时的顺序。
  - 如果value子句和query中只提供了N个字段，则目标字段为前N个字段。
  - value子句和query提供的值在表中从左到右关联到对应列。

  取值范围：已存在的字段名。

- **expression**

  赋予对应column的一个有效表达式或值：

  - 如果是INSERT ON DUPLICATE KEY UPDATE语句下，expression可以为VALUES(column_name)或EXCLUDED.column_name用来表示引用冲突行对应的column_name字段的值。需注意，其中VALUES(column_name)不支持嵌套在表达式中（例如VALUES(column_name)+1），但EXCLUDED不受此限制。
  - 向表中字段插入单引号 " ' "时需要使用单引号自身进行转义。
  - 如果插入行的表达式不是正确的数据类型，系统试图进行类型转换，若转换不成功，则插入数据失败，系统返回错误信息。

- **DEFAULT**

  对应字段名的缺省值。如果没有缺省值，则为NULL。

- **query**

  一个查询语句（SELECT语句），将查询结果作为插入的数据。

- **RETURNING**

  返回实际插入的行，RETURNING列表的语法与SELECT的输出列表一致。注意：INSERT ON DUPLICATE KEY UPDATE不支持RETURNING子句。

- **output_expression**

  INSERT命令在每一行都被插入之后用于计算输出结果的表达式。

  取值范围：该表达式可以使用table的任意字段。可以使用*返回被插入行的所有字段。

- **output_name**

  字段的输出名称。

  取值范围：字符串，符合**标识符命名规范**。

- **ON DUPLICATE KEY UPDATE**

  对于带有唯一约束（UNIQUE INDEX或PRIMARY KEY）的表，如果插入数据违反唯一约束，则对冲突行执行UPDATE子句完成更新。如果UPDATE子句为NOTHING，则不做任何操作。

  对于不带唯一约束的表，则仅执行插入。

  - 不支持触发器，不会触发目标表的INSERT或UPDATE触发器。
  - 不支持延迟生效（DEFERRABLE）的唯一约束或主键。
  - 如果表中存在多个唯一约束，如果所插入数据违反多个唯一约束，对于检测到冲突的第一行进行更新，其他冲突行不更新（检查顺序与索引维护具有强相关性，一般先创建的索引先进行冲突检查）。

- 分布列、唯一索引列不允许UPDATE。
- UPDATE的WHERE子句不支持包含子链接。

示例

```
--创建SCHEMA。
openGauss=# CREATE SCHEMA tpcds;

--创建表tpcds.reason。
openGauss=# CREATE TABLE tpcds.reason
(
  r_reason_sk    integer,
  r_reason_id    character(16),
  r_reason_desc  character(100)
);

--向表中插入一条记录。
openGauss=# INSERT INTO tpcds.reason(r_reason_sk, r_reason_id, r_reason_desc) VALUES (0,
'AAAAAAAAAAAAAAAA', 'reason0');

--创建表tpcds.reason_t2。
openGauss=# CREATE TABLE tpcds.reason_t2
(
  r_reason_sk    integer,
  r_reason_id    character(16),
  r_reason_desc  character(100)
);

--向表中插入一条记录。
openGauss=# INSERT INTO tpcds.reason_t2(r_reason_sk, r_reason_id, r_reason_desc) VALUES (1,
'AAAAAAAABAAAAAAA', 'reason1');

--向表中插入一条记录，和上一条语法等效。
openGauss=# INSERT INTO tpcds.reason_t2 VALUES (2, 'AAAAAAAABAAAAAAA', 'reason2');

--向表中插入多条记录。
openGauss=# INSERT INTO tpcds.reason_t2 VALUES (3, 'AAAAAAAACAAAAAAA','reason3'),(4,
'AAAAAAAADAAAAAAA', 'reason4'),(5, 'AAAAAAAAEAAAAAAA','reason5');

--向表中插入tpcds.reason中r_reason_sk小于5的记录。
openGauss=# INSERT INTO tpcds.reason_t2 SELECT * FROM tpcds.reason WHERE r_reason_sk <5;

--对表创建唯一索引。
openGauss=# CREATE UNIQUE INDEX reason_t2_u_index ON tpcds.reason_t2(r_reason_sk);

--向表中插入多条记录，如果冲突则更新冲突数据行中r_reason_id字段为'BBBBBBBBCAAAAAAA'。
openGauss=# INSERT INTO tpcds.reason_t2 VALUES (5, 'BBBBBBBBCAAAAAAA','reason5'),(6,
'AAAAAAAADAAAAAAA', 'reason6') ON DUPLICATE KEY UPDATE r_reason_id = 'BBBBBBBBCAAAAAAA';

--删除表tpcds.reason_t2。
openGauss=# DROP TABLE tpcds.reason_t2;

--删除表tpcds.reason。
openGauss=# DROP TABLE tpcds.reason;

--删除SCHEMA。
openGauss=# DROP SCHEMA tpcds CASCADE;
```

优化建议

- VALUES

通过INSERT语句批量插入数据时，建议将多条记录合并入一条语句中执行插入，以提高数据加载性能。例如，INSERT INTO sections VALUES (30, 'Administration', 31, 1900),(40, 'Development', 35, 2000), (50, 'Development' , 60 , 2001);

如果INSERT多VALUES语句中VALUES的值分布在一个DN上，GaussDB可以把语句下推到对应DN执行。目前只支持VALUES中值为常量，简单表达式和可下推函数（pg_proc中字段provolatile为'i'）。如果表中列带有DEFAULT值，只支持DEFAULT值为常量，简单表达式。单VALUES不能下推单DN的语句，多VALUES同样不支持下推。

7.13.114 LOCK

功能描述

LOCK TABLE获取表级锁。

GaussDB在为一个引用了表的命令自动请求锁时，尽可能选择最小限制的锁模式。如果用户需要一种更为严格的锁模式，可以使用LOCK命令。例如，一个应用是在Read Committed隔离级别上运行事务，并且它需要保证表中的数据在事务的运行过程中不被修改。为实现这个目的，则可以在查询之前对表使用SHARE锁模式进行锁定。这样将防止数据不被并发修改，从而保证后续的查询可以读到已提交的持久化的数据。因为SHARE锁模式与任何写操作需要的ROW EXCLUSIVE模式冲突，并且LOCK TABLE name IN SHARE MODE语句将等到所有当前持有ROW EXCLUSIVE模式锁的事务提交或回滚后才能执行。因此，一旦获得该锁，就不会存在未提交的写操作，并且其他操作也只能等到该锁释放之后才能开始。

注意事项

- LOCK TABLE只能在一个事务块的内部有用，因为锁在事务结束时就会被释放。出现在任意事务块外面的LOCK TABLE都会报错。
- 如果没有声明锁模式，缺省为最严格的模式ACCESS EXCLUSIVE。
- LOCK TABLE ... IN ACCESS SHARE MODE需要在目标表上有SELECT权限。所有其他形式的LOCK需要UPDATE或DELETE权限。
- 没有UNLOCK TABLE命令，锁总是在事务结束时释放。
- LOCK TABLE只处理表级的锁，因此那些带“ROW”字样的锁模式都是有歧义的。这些模式名称通常可理解为用户试图在一个被锁定的表中获取行级的锁。同样，ROW EXCLUSIVE模式也是一个可共享的表级锁。注意，只要是涉及到LOCK TABLE，所有锁模式都有相同的语意，区别仅在于规则中锁与锁之间是否冲突，规则请参见**表7-115**。
- 如果没有打开xc_maintenance_mode参数，那么对系统表申请ACCESS EXCLUSIVE级别锁将报错。
- 自动CANCEL业务接口只允许重分布工具使用。

语法格式

```
LOCK [ TABLE ] {[ ONLY ] name [, ...]| {name [ * ]} [, ...]}
    [ IN {ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE | SHARE | SHARE ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE} MODE ]
    [ NOWAIT ][CANCELABLE];
```


参数说明

表 7-115 冲突的锁模式

请求的锁模式/当前锁模式	ACCESS SHARE	ROW SHARE	ROW EXCLUSIVE	SHARE UPDATE EXCLUSIVE	SHARE	SHARE ROW EXCLUSIVE	EXCLUSIVE	ACCESS EXCLUSIVE
ACCESS SHARE	-	-	-	-	-	-	-	X
ROW SHARE	-	-	-	-	-	-	X	X
ROW EXCLUSIVE	-	-	-	-	X	X	X	X
SHARE UPDATE EXCLUSIVE	-	-	-	X	X	X	X	X
SHARE	-	-	X	X	-	X	X	X
SHARE ROW EXCLUSIVE	-	-	X	X	X	X	X	X
EXCLUSIVE	-	X	X	X	X	X	X	X
ACCESS EXCLUSIVE	X	X	X	X	X	X	X	X

LOCK的参数说明如下所示：

- **name**

  要锁定的表的名称，可以有模式修饰。

  LOCK TABLE命令中声明的表的顺序就是上锁的顺序。

  取值范围：已存在的表名。

- **ONLY**

  如果指定ONLY，只有该表被锁定。如果没有声明，该表和他的所有子表将都被锁定。

- **ACCESS SHARE**

  ACCESS锁只允许对表进行读取，而禁止对表进行修改。所有对表进行读取而不修改的SQL语句都会自动请求这种锁。例如，SELECT命令会自动在被引用的表上请求一个这种锁。

- **ROW SHARE**

  ROW SHARE锁允许对表进行并发读取，禁止对表进行其他操作。

  SELECT FOR UPDATE和SELECT FOR SHARE命令会自动在目标表上请求ROW SHARE锁（且所有被引用但不是FOR SHARE/FOR UPDATE的其他表上，还会自动加上ACCESS SHARE锁）。

  对于分区表，SELECT FOR SHARE操作还会在DN上获取partition对象的ROW EXCLUSIVE锁进行并发控制。

- **ROW EXCLUSIVE**

  与ROW SHARE锁相同，ROW EXCLUSIVE允许并发读取表，但是禁止修改表中数据。UPDATE，DELETE，INSERT命令会自动在目标表上请求这个锁（且所有被引用的其他表上还会自动加上的ACCESS SHARE锁）。通常情况下，所有会修改表数据的命令都会请求表的ROW EXCLUSIVE锁。

- **SHARE UPDATE EXCLUSIVE**

  这个模式保护一个表的模式不被并发修改，以及禁止在目标表上执行垃圾回收命令（VACUUM）。

  VACUUM（不带FULL选项）、ANALYZE、CREATE INDEX CONCURRENTLY命令会自动请求这样的锁。

- **SHARE**

  SHARE锁允许并发的查询，但是禁止对表进行修改。

  CREATE INDEX（不带CONCURRENTLY）语句会自动请求这种锁。

- **SHARE ROW EXCLUSIVE**

  SHARE ROW EXCLUSIVE锁禁止对表进行任何的并发修改，而且是独占锁，因此一个会话中只能获取一次。

  任何SQL语句都不会自动请求这个锁模式。

- **EXCLUSIVE**

  EXCLUSIVE锁允许对目标表进行并发查询，但是禁止任何其他操作。

  这个模式只允许并发加ACCESS SHARE锁，也就是说，只有对表的读动作可以和持有这个锁模式的事务并发执行。

  任何SQL语句都不会在用户表上自动请求这个锁模式。然而在某些操作的时候，会在某些系统表上请求它。

- **ACCESS EXCLUSIVE**

  这个模式保证其所有者（事务）是可以访问该表的唯一事务。

  ALTER TABLE，DROP TABLE，TRUNCATE，REINDEX，CLUSTER，VACUUM FULL命令会自动请求这种锁。

  在LOCK TABLE命令没有明确声明需要的锁模式时，它是缺省锁模式。

- **NOWAIT**

  声明LOCK TABLE不去等待任何冲突的锁释放，如果无法立即获取该锁，该命令退出并且发出一个错误信息。

  在不指定NOWAIT的情况下获取表级锁时，如果有其他互斥锁存在的话，则等待其他锁的释放。

- **CANCELABLE**
  通过指定该参数允许等锁线程给持锁线程和等锁线程发送CANCEL信号。
  只允许重分布工具使用，其它用户使用将报错。

示例

```
--创建SCHEMA。
openGauss=# CREATE SCHEMA tpcds;

--创建表tpcds.reason。
openGauss=# CREATE TABLE tpcds.reason
(
r_reason_sk         INTEGER      NOT NULL,
r_reason_id         CHAR(16)     NOT NULL,
r_reason_desc       INTEGER
);

--向表中插入多条记录。
openGauss=# INSERT INTO tpcds.reason VALUES (1, 'AAAAAAAABAAAAAAA', '18'),(5,
'AAAAAAAACAAAAAAA', '362'),(7, 'AAAAAAAADAAAAAAA', '585');

--在执行删除操作时对一个有主键的表进行 SHARE ROW EXCLUSIVE 锁。
openGauss=# CREATE TABLE tpcds.reason_t1 AS TABLE tpcds.reason;

openGauss=# START TRANSACTION;

openGauss=# LOCK TABLE tpcds.reason_t1 IN SHARE ROW EXCLUSIVE MODE;

openGauss=# DELETE FROM tpcds.reason_t1 WHERE r_reason_desc IN(SELECT r_reason_desc FROM
tpcds.reason_t1 WHERE r_reason_sk < 6 );

openGauss=# DELETE FROM tpcds.reason_t1 WHERE r_reason_sk = 7;

openGauss=# COMMIT;

--删除表tpcds.reason_t1。
openGauss=# DROP TABLE tpcds.reason_t1;

--删除表。
openGauss=# DROP TABLE tpcds.reason;

--删除SCHEMA。
openGauss=# DROP SCHEMA tpcds CASCADE;
```

7.13.115 MOVE

功能描述

MOVE在不检索数据的情况下重新定位一个游标。MOVE的作用类似于**FETCH**命令，但只是重定位游标而不返回行。

语法格式

```
MOVE [ direction [ FROM | IN ] ] cursor_name;
```

其中direction子句为可选参数。

```
NEXT
  | PRIOR
  | FIRST
  | LAST
  | ABSOLUTE count
  | RELATIVE count
  | count
```



```
| ALL
| FORWARD
| FORWARD count
| FORWARD ALL
| BACKWARD
| BACKWARD count
| BACKWARD ALL
```

参数说明

MOVE命令的参数与FETCH的相同，详细请参见FETCH的**参数说明**。

说明

成功完成时，MOVE命令将返回一个“MOVE count”的标签，count是一个使用相同参数的FETCH命令会返回的行数（可能为零）。

示例

```
--创建SCHEMA。
openGauss=# CREATE SCHEMA tpcds;

--创建表tpcds.reason。
openGauss=# CREATE TABLE tpcds.reason
(
r_reason_sk         INTEGER      NOT NULL,
r_reason_id         CHAR(16)     NOT NULL,
r_reason_desc       VARCHAR(40)
);

--向表中插入多条记录。
openGauss=# INSERT INTO tpcds.reason VALUES (1, 'AAAAAAAABAAAAAAA', 'Xxxxxxxxx'),(2,
'AAAAAAAACAAAAAAA', ' Xxxxxxxxx'),(3, 'AAAAAAAADAAAAAAA', ' Xxxxxxxxx'),(4, 'AAAAAAAAEAAAAAAA',
'Not the product that was ordered'),(5, 'AAAAAAAAFAAAAAAA', 'Parts missing'),(6, 'AAAAAAAAGAAAAAAA',
'Does not work with a product that I have'),(7, 'AAAAAAAAHAAAAAAA', 'Gift exchange');

--开始一个事务。
openGauss=# START TRANSACTION;

--定义一个名为cursor1的游标。
openGauss=# CURSOR cursor1 FOR SELECT * FROM tpcds.reason;

--忽略游标cursor1的前3行。
openGauss=# MOVE FORWARD 3 FROM cursor1;

--抓取游标cursor1的前4行。
openGauss=# FETCH 4 FROM cursor1;
 r_reason_sk |   r_reason_id    |                                          r_reason_desc
-------------+------------------
+-----------------------------------------------------------------------------------------------------
           4 | AAAAAAAAEAAAAAAA | Not the product that was
ordred
           5 | AAAAAAAAFAAAAAAA | Parts missing
           6 | AAAAAAAAGAAAAAAA | Does not work with a product that I
have
           7 | AAAAAAAAHAAAAAAA | Gift
exchange
(4 rows)

--关闭游标。
openGauss=# CLOSE cursor1;

--结束一个事务。
openGauss=# END;

--删除表。
openGauss=# DROP TABLE tpcds.reason;
```



```
--删除SCHEMA。
openGauss=# DROP SCHEMA tpcds CASCADE;
```

相关链接

CLOSE，FETCH

7.13.116 MERGE INTO

功能描述

通过MERGE INTO语句，将目标表和源表中的数据针对关联条件进行匹配，若关联条件匹配时对目标表进行UPDATE，无法匹配时对目标表执行INSERT。此语法可以很方便地用来合并执行UPDATE和INSERT，避免多次执行。

注意事项

- 进行MERGE INTO操作的用户需要同时拥有目标表的UPDATE和INSERT权限，以及源表的SELECT权限。
- 不支持重分布过程中MERGE INTO。
- 如果MERGE INTO操作的源表包含被动态数据脱敏的数据列，则执行过程中向目标表中插入或更新的结果即为脱敏后的值，无法被还原。

语法格式

```
MERGE [/*+ plan_hint */] INTO table_name [ [ AS ] alias ]
USING { { table_name | view_name } | subquery } [ [ AS ] alias ]
ON ( condition )
[
  WHEN MATCHED THEN
  UPDATE SET { column_name = { expression | subquery | DEFAULT } |
         ( column_name [, ...] ) = ( { expression | subquery | DEFAULT } [, ...] ) } [, ...]
  [ WHERE condition ]
]
[
  WHEN NOT MATCHED THEN
  INSERT { DEFAULT VALUES |
  [ ( column_name [, ...] ) ] VALUES ( { expression | subquery | DEFAULT } [, ...] ) [, ...] [ WHERE condition ] }
];
NOTICE: 'subquery' in the UPDATE and INSERT clauses are only avaliable in CENTRALIZED mode!
```

参数说明

- **plan_hint子句**

  以/*+ */的形式在MERGE关键字后，用于对MERGE对应的语句块生成的计划进行hint调优，详细用法请参见章节**使用Plan Hint进行调优**。每条语句中只有第一个/*+ plan_hint */注释块会作为hint生效，里面可以写多条hint。

- **INTO子句**

  指定正在更新或插入的目标表。目标表为复制表时，暂不支持目标表中某列默认值为volatile函数（如自增列），enable_stream_operator=off时目标表需要包含主键或带有unique not null。

  - **table_name**

    目标表的表名。

  - **alias**

目标表的别名。

取值范围：字符串，符合**标识符命名规范**。

- **USING子句**

  指定源表，源表可以为表、视图或子查询。目标表为复制表时，暂不支持USING子句中包含非复制表。

- **ON子句**

  关联条件，用于指定目标表和源表的关联条件。不支持更新关联条件中的字段。

- **WHEN MATCHED子句**

  当源表和目标表中数据针对关联条件可以匹配上时，选择WHEN MATCHED子句进行UPDATE操作。

  不支持更新分布列。不支持更新系统表、系统列。

- **WHEN NOT MATCHED子句**

  当源表和目标表中数据针对关联条件无法匹配时，选择WHEN NOT MATCHED子句进行INSERT操作。

  不支持INSERT子句中包含多个VALUES。

  WHEN MATCHED和WHEN NOT MATCHED子句顺序可以交换，可以缺省其中一个，但不能同时缺省，不支持同时指定两个WHEN MATCHED或WHEN NOT MATCHED子句。

- **DEFAULT**

  用对应字段的缺省值填充该字段。

  如果没有缺省值，则为NULL。

- **WHERE condition**

  UPDATE子句和INSERT子句的条件，只有在条件满足时才进行更新操作，可缺省。不支持WHERE条件中引用系统列。不建议使用int等数值类型作为condition，因为int等数值类型可以隐式转换为bool值（非0值隐式转换为true，0转换为false），可能导致非预期的结果。

示例

```
-- 创建目标表products和源表newproducts，并插入数据
openGauss=# CREATE TABLE products
(
product_id INTEGER,
product_name VARCHAR2(60),
category VARCHAR2(60)
);

openGauss=# INSERT INTO products VALUES (1501, 'vivitar 35mm', 'electrncs');
openGauss=# INSERT INTO products VALUES (1502, 'olympus is50', 'electrncs');
openGauss=# INSERT INTO products VALUES (1600, 'play gym', 'toys');
openGauss=# INSERT INTO products VALUES (1601, 'lamaze', 'toys');
openGauss=# INSERT INTO products VALUES (1666, 'harry potter', 'dvd');

openGauss=# CREATE TABLE newproducts
(
product_id INTEGER,
product_name VARCHAR2(60),
category VARCHAR2(60)
);

openGauss=# INSERT INTO newproducts VALUES (1502, 'olympus camera', 'electrncs');
openGauss=# INSERT INTO newproducts VALUES (1601, 'lamaze', 'toys');
openGauss=# INSERT INTO newproducts VALUES (1666, 'harry potter', 'toys');
openGauss=# INSERT INTO newproducts VALUES (1700, 'wait interface', 'books');
```



```
-- 进行MERGE INTO操作
openGauss=# MERGE INTO products p
USING newproducts np
ON (p.product_id = np.product_id)
WHEN MATCHED THEN
  UPDATE SET p.product_name = np.product_name, p.category = np.category WHERE p.product_name !=
'play gym'
WHEN NOT MATCHED THEN
  INSERT VALUES (np.product_id, np.product_name, np.category) WHERE np.category = 'books';
MERGE 4

-- 查询更新后的结果
openGauss=# SELECT * FROM products ORDER BY product_id;
 product_id |  product_name  | category
------------+----------------+-----------
       1501 | vivitar 35mm   | electrncs
       1502 | olympus camera | electrncs
       1600 | play gym       | toys
       1601 | lamaze         | toys
       1666 | harry potter   | toys
       1700 | wait interface | books
(6 rows)

-- 删除表
openGauss=# DROP TABLE products;
openGauss=# DROP TABLE newproducts;
```

7.13.117 PREPARE

功能描述

创建一个预备语句。

预备语句是服务端的对象，可以用于优化性能。在执行PREPARE语句的时候，指定的查询被解析、分析、重写。当随后发出EXECUTE语句的时候，预备语句被规划和执行。这种设计避免了重复解析、分析工作。PREPARE语句创建后在整个数据库会话期间一直存在，一旦创建成功，即便是在事务块中创建，事务回滚，PREPARE也不会删除。只能通过显式调用**DEALLOCATE**进行删除，会话结束时，PREPARE也会自动删除。

语法格式

```
PREPARE name [ ( data_type [, ...] ) ] AS statement;
```

参数说明

- **name**

  指定预备语句的名称。它必须在该会话中是唯一的。
- **data_type**

  参数的数据类型。
- **statement**

  是SELECT、INSERT、UPDATE、DELETE、MERGE INTO或VALUES语句之一。

示例

请参见EXECUTE的**示例**。

相关链接

DEALLOCATE

7.13.118 PREPARE TRANSACTION

功能描述

为当前事务做两阶段提交的准备。

在命令之后，事务就不再和当前会话关联了；它的状态完全保存在磁盘上，它被提交成功的可能性非常高，即使是在请求提交之前数据库发生了崩溃也如此。

一旦准备好了，一个事务就可以在稍后用COMMIT PREPARED或 ROLLBACK PREPARED命令分别进行提交或者回滚。这些命令可以从任何会话中发出，而不光是最初执行事务的那个会话。

从发出命令的会话的角度来看，PREPARE TRANSACTION不同于ROLLBACK：在执行它之后，就不再有活跃的当前事务了，并且预备事务的效果无法见到 (在事务提交的时候其效果会再次可见)。

如果PREPARE TRANSACTION因为某些原因失败，那么它就会变成一个ROLLBACK，当前事务被取消。

注意事项

- 事务功能由数据库自动维护，不应显式使用事务功能。
- 分布式当前不支持客户调用自定义PREPARE TRANSACTION操作。
- 在运行PREPARE TRANSACTION命令时，必须在postgresql.conf配置文件中增大max_prepared_transactions的数值。建议至少将其设置为等于max_connections，这样每个会话都可以有一个等待中的预备事务。

语法格式

```
PREPARE TRANSACTION transaction_id;
```

参数说明

- **transaction_id**

  待提交事务的标识符，用于后面在COMMIT PREPARED或ROLLBACK PREPARED的时候标识这个事务。它不能和任何当前预备事务已经使用了的标识符同名。

  取值范围：标识符必须以字符串文本的方式书写，并且必须小于200字节长。

相关链接

COMMIT PREPARED，ROLLBACK PREPARED

7.13.119 REASSIGN OWNED

功能描述

修改数据库对象的属主。

REASSIGN OWNED要求系统将所有旧角色拥有的数据库对象的属主更改为新角色。

注意事项

- REASSIGN OWNED常用于在删除角色之前的准备工作。
- 执行REASSIGN OWNED需要有旧角色和目标角色上的权限。

语法格式

```
REASSIGN OWNED BY old_role [, ...] TO new_role;
```

参数说明

- **old_role**

  旧属主的角色名。

- **new_role**

  将要成为这些对象属主的新角色的名称。注意：只有初始用户才可以通过REASSIGN OWNED语法将属主修改为初始化用户。

示例

无。

7.13.120 REINDEX

功能描述

为表中的数据重建索引。

在以下几种情况下需要使用REINDEX重建索引：

- 索引崩溃，并且不再包含有效的数据。
- 索引变得“臃肿”，包含大量的空页或接近空页。
- 为索引更改了存储参数（例如填充因子），并且希望这个更改完全生效。

注意事项

REINDEX DATABASE和SYSTEM这种形式的重建索引不能在事务块中执行。

语法格式

- 重建普通索引。

  ```
  REINDEX { INDEX | [INTERNAL] TABLE | DATABASE | SYSTEM } name [ FORCE ];
  ```

- 重建索引分区。

  ```
  REINDEX  { INDEX|  [INTERNAL] TABLE} name
     PARTITION partition_name [ FORCE  ];
  ```

参数说明

- **INDEX**

  重新建立指定的索引。

- **TABLE**

  重新建立指定表的所有索引，如果表有从属的“TOAST”表，则这个表也会重建索引。如果表上有索引已经被alter unusable失效，则这个索引无法被重新创建。

- **DATABASE**
  重建当前数据库里的所有索引。
- **SYSTEM**
  在当前数据库上重建所有系统表上的索引。不会处理在用户表上的索引。
- **name**
  需要重建索引的索引、表、数据库的名称。表和索引可以有模式修饰。

  说明

  REINDEX DATABASE和SYSTEM只能重建当前数据库的索引，所以name必须和当前数据库名称相同。
- **FORCE**
  废弃选项，仅为保持前向兼容，故继续保留。
- **partition_name**
  需要重建索引的分区的名称或者索引分区的名称。
  取值范围：
  - 如果前面是REINDEX INDEX，则这里应该指定索引分区的名称；
  - 如果前面是REINDEX TABLE，则这里应该指定分区的名称；

须知

- REINDEX DATABASE和SYSTEM这种形式的重建索引不能在事务块中执行。
- REINDEX不支持单独操作toast表或toast索引。

示例

```
--创建SCHEMA。
openGauss=# CREATE SCHEMA tpcds;

--创建表tpcds.customer。
openGauss=# CREATE TABLE tpcds.customer
(
c_customer_sk         INTEGER        NOT NULL,
c_customer_id         CHAR(16)       NOT NULL
);

--向表中插入多条记录。
openGauss=# INSERT INTO tpcds.customer VALUES (1, 'AAAAAAAABAAAAAAA'),(5,
'AAAAAAAACAAAAAAA'),(10, 'AAAAAAAADAAAAAAA');

--创建一个行存表tpcds.customer_t1，并在tpcds.customer_t1表上的c_customer_sk字段创建索引。
openGauss=# CREATE TABLE tpcds.customer_t1
(
    c_customer_sk             integer               not null,
    c_customer_id             char(16)               not null,
    c_current_cdemo_sk        integer                       ,
    c_current_hdemo_sk        integer                       ,
    c_current_addr_sk         integer                       ,
    c_first_shipto_date_sk    integer                       ,
    c_first_sales_date_sk     integer                       ,
    c_salutation              char(10)                      ,
    c_first_name              char(20)                      ,
    c_last_name               char(30)                      ,
    c_preferred_cust_flag     char(1)                       ,
    c_birth_day               integer                       ,
```



```
    c_birth_month             integer                   ,
    c_birth_year              integer                   ,
    c_birth_country           varchar(20)               ,
    c_login                   char(13)                  ,
    c_email_address           char(50)                  ,
    c_last_review_date        char(10)
)
WITH (orientation = row);

openGauss=# CREATE INDEX tpcds_customer_index1 ON tpcds.customer_t1 (c_customer_sk);

openGauss=# INSERT INTO tpcds.customer_t1 SELECT * FROM tpcds.customer WHERE c_customer_sk < 10;

--重建一个单独索引。
openGauss=# REINDEX INDEX tpcds.tpcds_customer_index1;

--重建表tpcds.customer_t1上的所有索引。
openGauss=# REINDEX TABLE tpcds.customer_t1;

--删除tpcds.customer_t1表。
openGauss=# DROP TABLE tpcds.customer_t1;

--删除表。
openGauss=# DROP TABLE tpcds.customer;

--删除SCHEMA。
openGauss=# DROP SCHEMA tpcds CASCADE;
```

优化建议

- INTERNAL TABLE
  此种情况大多用于故障恢复，不建议进行并发操作。
- DATABASE
  不建议在事务中REINDEX DATABASE。
- SYSTEM
  不建议在事务中REINDEX系统表。

7.13.121 REFRESH INCREMENTAL MATERIALIZED VIEW

功能描述

REFRESH INCREMENTAL MATERIALIZED VIEW会以增量刷新的方式对物化视图进行刷新。

注意事项

- 增量刷新仅支持增量物化视图。
- 刷新物化视图需要当前用户拥有基表的SELECT权限。

语法格式

```
REFRESH INCREMENTAL MATERIALIZED VIEW mv_name;
```

参数说明

- **mv_name**
  要刷新的物化视图的名称。

示例

```
--创建一个普通表。
openGauss=# CREATE TABLE my_table (c1 int, c2 int);

--创建增量物化视图。
openGauss=# CREATE INCREMENTAL MATERIALIZED VIEW my_imv AS SELECT * FROM my_table;

--基表写入数据。
openGauss=# INSERT INTO my_table VALUES(1,1),(2,2);

--对增量物化视图my_imv进行增量刷新。
openGauss=# REFRESH INCREMENTAL MATERIALIZED VIEW my_imv;

--删除增量物化视图。
openGauss=# DROP MATERIALIZED VIEW my_imv;

--删除表my_table。
openGauss=# DROP TABLE my_table;
```

相关链接

ALTER MATERIALIZED VIEW，**CREATE INCREMENTAL MATERIALIZED VIEW**，**CREATE MATERIALIZED VIEW**，**CREATE TABLE**，**DROP MATERIALIZED VIEW**，**REFRESH MATERIALIZED VIEW**

7.13.122 REFRESH MATERIALIZED VIEW

功能描述

REFRESH MATERIALIZED VIEW会以全量刷新的方式对物化视图进行刷新。

注意事项

- 全量刷新既可以对全量物化视图执行，也可以对增量物化视图执行。
- 刷新物化视图需要当前用户拥有基表的SELECT权限。

语法格式

```
REFRESH MATERIALIZED VIEW mv_name;
```

参数说明

- **mv_name**
  要刷新的物化视图的名称。

示例

```
--创建一个普通表。
openGauss=# CREATE TABLE my_table (c1 int, c2 int);

--创建全量物化视图。
openGauss=# CREATE MATERIALIZED VIEW my_mv AS SELECT * FROM my_table;

--创建增量物化视图。
openGauss=# CREATE INCREMENTAL MATERIALIZED VIEW my_imv AS SELECT * FROM my_table;

--基表写入数据。
openGauss=# INSERT INTO my_table VALUES(1,1),(2,2);

--对全量物化视图my_mv进行全量刷新。
```



```
openGauss=# REFRESH MATERIALIZED VIEW my_mv;

--对增量物化视图my_imv进行全量刷新。
openGauss=# REFRESH MATERIALIZED VIEW my_imv;

--删除增量物化视图。
openGauss=# DROP MATERIALIZED VIEW my_imv;

--删除全量物化视图。
openGauss=# DROP MATERIALIZED VIEW my_mv;

--删除表my_table。
openGauss=# DROP TABLE my_table;
```

相关链接

ALTER MATERIALIZED VIEW，**CREATE INCREMENTAL MATERIALIZED VIEW**，**CREATE MATERIALIZED VIEW**，**CREATE TABLE**，**DROP MATERIALIZED VIEW**，**REFRESH INCREMENTAL MATERIALIZED VIEW**

7.13.123 RELEASE SAVEPOINT

功能描述

RELEASE SAVEPOINT删除一个当前事务先前定义的保存点。

把一个保存点删除就令其无法作为回滚点使用，除此之外它没有其它用户可见的行为。它并不能撤销在保存点建立起来之后执行的命令的影响，要撤销那些命令可以使用ROLLBACK TO SAVEPOINT 。当不再需要的时候删除一个保存点可以令系统在事务结束之前提前回收一些资源。

RELEASE SAVEPOINT也删除所有在指定的保存点建立之后的所有保存点。

注意事项

- 不能RELEASE一个没有定义的保存点，语法上会报错。
- 如果事务在回滚状态，则不能释放保存点。
- 如果多个保存点拥有同样的名称，只有最近定义的那个才被释放。

语法格式

```
RELEASE [ SAVEPOINT ] savepoint_name;
```

参数说明

savepoint_name

要删除的保存点的名称

示例

```
--创建SCHEMA。
openGauss=# CREATE SCHEMA tpcds;

--创建一个新表。
openGauss=# CREATE TABLE tpcds.table1(a int);

--开启事务。
openGauss=# START TRANSACTION;
```



```
--插入数据。
openGauss=# INSERT INTO tpcds.table1 VALUES (3);

--建立保存点。
openGauss=# SAVEPOINT my_savepoint;

--插入数据。
openGauss=# INSERT INTO tpcds.table1 VALUES (4);

--删除保存点。
openGauss=# RELEASE SAVEPOINT my_savepoint;

--提交事务。
openGauss=# COMMIT;

--查询表的内容，会同时看到3和4。
openGauss=# SELECT * FROM tpcds.table1;

--删除表。
openGauss=# DROP TABLE tpcds.table1;

--删除SCHEMA。
openGauss=# DROP SCHEMA tpcds CASCADE;
```

相关链接

SAVEPOINT，**ROLLBACK TO SAVEPOINT**

7.13.124 RESET

功能描述

RESET将指定的运行时参数恢复为缺省值。这些参数的缺省值是指postgresql.conf配置文件中所描述的参数缺省值。

RESET命令与如下命令的作用相同：

```
SET configuration_parameter TO DEFAULT;
```

注意事项

RESET的事务性行为和SET相同，它的影响将会被事务回滚撤销。

语法格式

```
RESET {configuration_parameter | CURRENT_SCHEMA | TIME ZONE | TRANSACTION ISOLATION LEVEL |
SESSION AUTHORIZATION | ALL };
```

参数说明

- **configuration_parameter**
  运行时参数的名称。
  取值范围：可以使用SHOW ALL命令查看运行时参数。

  说明

  部分通过SHOW ALL查看的参数不能通过SET设置。如max_datanodes。

- **CURRENT_SCHEMA**
  当前模式

- **TIME ZONE**
  时区。
- **TRANSACTION ISOLATION LEVEL**
  事务的隔离级别。
- **SESSION AUTHORIZATION**
  当前会话的用户标识符。
- **ALL**
  所有运行时参数。

示例

```
--把timezone设为缺省值。
openGauss=# RESET timezone;

--把所有参数设置为缺省值。
openGauss=# RESET ALL;
```

相关链接

SET，**SHOW**

7.13.125 REVOKE

功能描述

REVOKE用于撤销一个或多个角色的权限。

注意事项

非对象所有者试图在对象上REVOKE权限，命令按照以下规则执行：

- 如果授权用户没有该对象上的权限，则命令立即失败。
- 如果授权用户有部分权限，则只撤销那些有授权选项的权限。
- 如果授权用户没有授权选项，REVOKE ALL PRIVILEGES形式将发出一个错误信息，而对于其他形式的命令而言，如果是命令中指定名称的权限没有相应的授权选项，该命令将发出一个警告。

语法格式

- 回收指定表或视图上权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | ALTER | DROP | COMMENT |
INDEX | VACUUM }[, ...]
    | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...]
        | ALL TABLES IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收表上指定字段权限。

```
REVOKE [ GRANT OPTION FOR ]
    { {{ SELECT | INSERT | UPDATE | REFERENCES | COMMENT } ( column_name [, ...] )}[, ...]
    | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
    ON [ TABLE ] table_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```


- 回收指定序列上权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | UPDATE | ALTER | DROP | COMMENT }[, ...]
    | ALL [ PRIVILEGES ] }
    ON { [ SEQUENCE ] sequence_name [, ...]
        | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收指定数据库上权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { CREATE | CONNECT | TEMPORARY | TEMP | ALTER | DROP | COMMENT } [, ...]
    | ALL [ PRIVILEGES ] }
    ON DATABASE database_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收指定域上权限。

```
REVOKE [ GRANT OPTION FOR ]
    { USAGE | ALL [ PRIVILEGES ] }
    ON DOMAIN domain_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收指定客户端加密主密钥上的权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | DROP } [, ...] | ALL [PRIVILEGES] }
    ON CLIENT_MASTER_KEYS client_master_keys_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收指定列加密密钥上的权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | DROP } [, ...] | ALL [PRIVILEGES]}
    ON COLUMN_ENCRYPTION_KEYS column_encryption_keys_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收指定目录上权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { READ | WRITE | ALTER | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON DIRECTORY directory_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收指定外部数据源上权限。

```
REVOKE [ GRANT OPTION FOR ]
   { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN DATA WRAPPER fdw_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收指定外部服务器上权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON FOREIGN SERVER server_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收指定函数上权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON { FUNCTION {function_name ( [ {[ argmode ] [ arg_name ] arg_type} [, ...] ] )} [, ...]
        | ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收指定存储过程上权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON { PROCEDURE {proc_name ( [ {[ argmode ] [ arg_name ] arg_type} [, ...] ] )} [, ...]
```



```
    | ALL PROCEDURE IN SCHEMA schema_name [, ...] }
  FROM { [ GROUP ] role_name | PUBLIC } [, ...]
  [ CASCADE | RESTRICT ];
```

- 回收指定过程语言上权限。

```
REVOKE [ GRANT OPTION FOR ]
  { USAGE | ALL [ PRIVILEGES ] }
   ON LANGUAGE lang_name [, ...]
   FROM { [ GROUP ] role_name | PUBLIC } [, ...]
   [ CASCADE | RESTRICT ];
```

- 回收指定大对象上权限。

```
REVOKE [ GRANT OPTION FOR ]
   { { SELECT | UPDATE } [, ...] | ALL [ PRIVILEGES ] }
   ON LARGE OBJECT loid [, ...]
   FROM { [ GROUP ] role_name | PUBLIC } [, ...]
   [ CASCADE | RESTRICT ];
```

- 回收指定模式上权限。

```
REVOKE [ GRANT OPTION FOR ]
   { { CREATE | USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
   ON SCHEMA schema_name [, ...]
   FROM { [ GROUP ] role_name | PUBLIC } [, ...]
   [ CASCADE | RESTRICT ];
```

- 回收指定表空间上权限。

```
REVOKE [ GRANT OPTION FOR ]
   { { CREATE  | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
   ON TABLESPACE tablespace_name [, ...]
   FROM { [ GROUP ] role_name | PUBLIC } [, ...]
   [ CASCADE | RESTRICT ];
```

- 回收指定类型上权限。

```
REVOKE [ GRANT OPTION FOR ]
  { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
   ON TYPE type_name [, ...]
   FROM { [ GROUP ] role_name | PUBLIC } [, ...]
   [ CASCADE | RESTRICT ];
```

- 回收指定子集群上权限

```
REVOKE [ GRANT OPTION FOR ]
   { { CREATE | USAGE | COMPUTE  | ALTER | DROP } [, ...] | ALL [ PRIVILEGES ] }
   ON NODE GROUP group_name [, ...]
   FROM { [ GROUP ] role_name | PUBLIC } [, ...]
   [ CASCADE | RESTRICT ];
```

说明

回收子集群的create权限时，会默认回收usage和compute权限。

- 回收directory对象的权限。

```
REVOKE [ GRANT OPTION FOR ]
  { { READ | WRITE } [, ...] | ALL [PRIVILEGES] }
  ON DIRECTORY directory_name [, ...]
  FROM {[GROUP] role_name | PUBLIC} [, ...]
  [ CASCADE | RESTRICT ];
```

- 回收package对象的权限。

```
REVOKE [ GRANT OPTION FOR ]
  { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [PRIVILEGES] }
  ON PACKAGE package_name [, ...]
  FROM {[GROUP] role_name | PUBLIC} [, ...]
  [ CASCADE | RESTRICT ];
```

- 按角色回收角色上的权限。

```
REVOKE [ ADMIN OPTION FOR ]
   role_name [, ...] FROM role_name [, ...]
   [ CASCADE | RESTRICT ];
```

- 回收角色上的sysadmin权限。

```
REVOKE ALL { PRIVILEGES | PRIVILEGE } FROM role_name;
```


- 回收ANY权限。

```
REVOKE [ ADMIN OPTION FOR ]
  { CREATE ANY TABLE | ALTER ANY TABLE | DROP ANY TABLE | SELECT ANY TABLE | INSERT ANY
TABLE | UPDATE ANY TABLE |
  DELETE ANY TABLE | CREATE ANY SEQUENCE | CREATE ANY INDEX | CREATE ANY FUNCTION |
EXECUTE ANY FUNCTION |
  CREATE ANY PACKAGE | EXECUTE ANY PACKAGE | CREATE ANY TYPE } [, ...]
  FROM [ GROUP ] role_name [, ...];
```

参数说明

关键字PUBLIC表示一个隐式定义的拥有所有角色的组。

权限类别和参数说明，请参见GRANT的**参数说明**。

任何特定角色拥有的特权包括直接授予该角色的特权、从该角色作为其成员的角色中得到的权限以及授予给PUBLIC的权限。因此，从PUBLIC收回SELECT特权并不一定会意味着所有角色都会失去在该对象上的SELECT特权，那些直接被授予的或者通过另一个角色被授予的角色仍然会拥有它。类似地，从一个用户收回SELECT后，如果PUBLIC仍有SELECT权限，该用户还是可以使用SELECT。

指定GRANT OPTION FOR时，只撤销对该权限授权的权力，而不撤销该权限本身。

如用户A拥有某个表的UPDATE权限以及WITH GRANT OPTION选项，同时A把这个权限赋予了用户B，则用户B持有的权限称为依赖性权限。当用户A持有的权限或者授权选项被撤销时，依赖性权限仍然存在，但如果声明了CASCADE，则所有依赖性权限都被撤销。

一个用户只能撤销由它自己直接赋予的权限。例如，如果用户A被指定授权（WITH ADMIN OPTION）选项，且把一个权限赋予了用户B，然后用户B又赋予了用户C，则用户A不能直接将C的权限撤销。但是，用户A可以撤销用户B的授权选项，并且使用CASCADE。这样，用户C的权限就会自动被撤销。另外一个例子：如果A和B都赋予了C同样的权限，则A可以撤销他自己的授权选项，但是不能撤销B的，因此C仍然拥有该权限。

如果执行REVOKE的角色持有的权限是通过多层成员关系获得的，则具体是哪个包含的角色执行的该命令是不确定的。在这种场合下，建议的方法是使用SET ROLE成为特定角色，然后执行REVOKE，否则可能导致删除了不想删除的权限，或者是任何权限都没有删除。

示例

请参考GRANT的**示例**。

相关链接

GRANT

7.13.126 ROLLBACK

功能描述

回滚当前事务并取消当前事务中的所有更新。

在事务运行的过程中发生了某种故障，事务不能继续执行，系统将事务中对数据库的所有已完成的操作全部撤销，数据库状态回到事务开始时。

注意事项

如果不在一个事务内部发出ROLLBACK不会有问题，但是将抛出一个NOTICE信息。

语法格式

```
ROLLBACK [ WORK | TRANSACTION ];
```

参数说明

- **WORK | TRANSACTION**

  可选关键字。除了增加可读性，没有任何其他作用。

示例

```
--开启一个事务
openGauss=# START TRANSACTION;

--取消所有更改
openGauss=# ROLLBACK;
```

相关链接

COMMIT | END

7.13.127 ROLLBACK PREPARED

功能描述

取消一个先前为两阶段提交准备好的事务。

注意事项

- 该功能仅在维护模式(GUC参数xc_maintenance_mode为on时)下可用。该模式谨慎打开，一般供维护人员排查问题使用，一般用户不应使用该模式。
- 要想回滚一个预备事务，必须是最初发起事务的用户，或者是系统管理员。
- 事务功能由数据库自动维护，不应显式使用事务功能。

语法格式

```
ROLLBACK PREPARED transaction_id ;
```

参数说明

- **transaction_id**

  待提交事务的标识符。它不能和任何当前预备事务已经使用了的标识符同名。

相关链接

COMMIT PREPARED，**PREPARE TRANSACTION**。

7.13.128 ROLLBACK TO SAVEPOINT

功能描述

ROLLBACK TO SAVEPOINT用于回滚到一个保存点，隐含地删除所有在该保存点之后建立的保存点。

回滚所有指定保存点建立之后执行的命令。保存点仍然有效，并且需要时可以再次回滚到该点。

注意事项

- 不能回滚到一个未定义的保存点，语法上会报错。
- 在保存点方面，游标有一些非事务性的行为。任何在保存点里打开的游标都会在回滚掉这个保存点之后关闭。如果一个前面打开了的游标在保存点里面，并且游标被一个FETCH命令影响，而这个保存点稍后回滚了，那么这个游标的位置仍然在FETCH让它指向的位置(也就是FETCH不会被回滚)。关闭一个游标的行为也不会被回滚给撤销掉。如果一个游标的操作导致事务回滚，那么这个游标就会置于不可执行状态，所以，尽管一个事务可以用ROLLBACK TO SAVEPOINT重新恢复，但是游标不能再使用了。
- 使用ROLLBACK TO SAVEPOINT回滚到一个保存点。使用RELEASE SAVEPOINT删除一个保存点，但是保留该保存点建立后执行的命令的效果。

语法格式

```
ROLLBACK [ WORK | TRANSACTION ] TO [ SAVEPOINT ] savepoint_name;
```

参数说明

savepoint_name

回滚截至的保存点

示例

```
--撤销 my_savepoint 建立之后执行的命令的影响。
openGauss=# START TRANSACTION;
openGauss=# SAVEPOINT my_savepoint;
openGauss=# ROLLBACK TO SAVEPOINT my_savepoint;
--游标位置不受保存点回滚的影响。
openGauss=# DECLARE foo CURSOR FOR SELECT 1 UNION SELECT 2;
openGauss=# SAVEPOINT foo;
openGauss=# FETCH 1 FROM foo;
 ?column?
----------
        1
openGauss=# ROLLBACK TO SAVEPOINT foo;
openGauss=# FETCH 1 FROM foo;
 ?column?
----------
        2
openGauss=# RELEASE SAVEPOINT my_savepoint;
openGauss=# COMMIT;
```

相关链接

SAVEPOINT，**RELEASE SAVEPOINT**

7.13.129 SAVEPOINT

功能描述

SAVEPOINT用于在当前事务里建立一个新的保存点。

保存点是事务中的一个特殊记号，它允许将那些在它建立后执行的命令全部回滚，把事务的状态恢复到保存点所在的时刻。

注意事项

- 使用ROLLBACK TO SAVEPOINT回滚到一个保存点。使用RELEASE SAVEPOINT删除一个保存点，但是保留该保存点建立后执行的命令的效果。
- 保存点只能在一个事务块里面建立。在一个事务里面可以定义多个保存点。
- 由于节点故障或者通信故障引起的分布式节点线程或进程退出导致的报错，以及由于COPY FROM操作中源数据与目标表的表结构不一致导致的报错，均不能正常回滚到保存点之前，而是整个事务回滚。
- SQL标准要求，使用SAVEPOINT建立一个同名保存点时，需要自动删除前面那个同名保存点。在GaussDB数据库里，将保留旧的保存点，但是在回滚或者释放的时候，只使用最近的那个。释放了新的保存点将导致旧的再次成为ROLLBACK TO SAVEPOINT和RELEASE SAVEPOINT可以访问的保存点。除此之外，SAVEPOINT是完全符合SQL标准的。

语法格式

```
SAVEPOINT savepoint_name;
```

参数说明

savepoint_name

新建保存点的名称。

须知

使用SAVEPOINT时，建议及时RELEASE SAVEPOINT，避免子事务的嵌套个数过大；建议嵌套个数不要超过10000，当嵌套数过大时，可能引发当前事务性能劣化。

示例

```
--创建一个新表。
openGauss=# CREATE TABLE table1(a int);

--开启事务。
openGauss=# START TRANSACTION;

--插入数据。
openGauss=# INSERT INTO table1 VALUES (1);

--建立保存点。
openGauss=# SAVEPOINT my_savepoint;

--插入数据。
openGauss=# INSERT INTO table1 VALUES (2);
```



```
--回滚保存点。
openGauss=# ROLLBACK TO SAVEPOINT my_savepoint;

--插入数据。
openGauss=# INSERT INTO table1 VALUES (3);

--提交事务。
openGauss=# COMMIT;

--查询表的内容，会同时看到1和3,不能看到2，因为2被回滚。
openGauss=# SELECT * FROM table1;

--删除表。
openGauss=# DROP TABLE table1;

--创建一个新表。
openGauss=# CREATE TABLE table2(a int);

--开启事务。
openGauss=# START TRANSACTION;

--插入数据。
openGauss=# INSERT INTO table2 VALUES (3);

--建立保存点。
openGauss=# SAVEPOINT my_savepoint;

--插入数据。
openGauss=# INSERT INTO table2 VALUES (4);

--回滚保存点。
openGauss=# RELEASE SAVEPOINT my_savepoint;

--提交事务。
openGauss=# COMMIT;

--查询表的内容，会同时看到3和4。
openGauss=# SELECT * FROM table2;

--删除表。
openGauss=# DROP TABLE table2;
```

相关链接

RELEASE SAVEPOINT，**ROLLBACK TO SAVEPOINT**

7.13.130 SELECT

功能描述

SELECT用于从表或视图中取出数据。

SELECT语句就像叠加在数据库表上的过滤器，利用SQL关键字从数据表中过滤出用户需要的数据。

注意事项

- 表的所有者、拥有表SELECT权限的用户或拥有SELECT ANY TABLE权限的用户，有权读取表或视图中数据，系统管理员默认拥有此权限。
- SELECT支持普通表的JOIN，不支持普通表和GDS外表的JOIN。即SELECT语句中不能同时出现普通表和GDS外表。
- 必须对每个在SELECT命令中使用的字段有SELECT权限。

- 使用FOR UPDATE或FOR SHARE除了SELECT权限外还要求UPDATE权限。

语法格式

- 查询数据

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT [/*+ plan_hint */] [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    { * | {expression [ [ AS ] output_name ]} [, ...] }
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY grouping_element [, ...] ]
    [ HAVING condition [, ...] ]
    [ WINDOW {window_name AS ( window_definition )} [, ...] ]
    [ { UNION | INTERSECT | EXCEPT | MINUS } [ ALL | DISTINCT ] select ]
    [ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS { FIRST |
LAST } ]} [, ...] ]
    [ LIMIT { [offset,] count | ALL } ]
    [ OFFSET start [ ROW | ROWS ] ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
    [ {FOR { UPDATE | SHARE } [ OF table_name [, ...] ] [ NOWAIT | WAIT n]} [...] ]
    TABLE { ONLY { (table_name) | table_name } | table_name [ * ]};
```

说明

condition和expression中可以使用targetlist中表达式的别名。

- 只能同一层引用。
- 只能引用targetlist中的别名。
- 只能是后面的表达式引用前面的表达式。
- 不能包含volatile函数。
- 不能包含Window function函数。
- 不支持在JOIN ON条件中引用别名。
- targetlist中有多个要应用的别名则报错。

- 其中子查询with_query为：

```
with_query_name [ ( column_name [, ...] ) ]
    AS [ [ NOT ] MATERIALIZED ] ( {select | values | insert | update | delete} )
```

- 其中指定查询源from_item为：

```
{[ ONLY ] table_name [ * ] [ partition_clause ] [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
[ TABLESAMPLE sampling_method ( argument [, ...] ) [ REPEATABLE ( seed ) ] ]
|( select ) [ AS ] alias [ ( column_alias [, ...] ) ]
|with_query_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
|function_name ( [ argument [, ...] ] ) [ AS ] alias [ ( column_alias [, ...] | column_definition [, ...] ) ]
|function_name ( [ argument [, ...] ] ) AS ( column_definition [, ...] )
|from_item [ NATURAL ] join_type from_item [ ON join_condition | USING ( join_column [, ...] ) ]}
```

- 其中group子句为：

```
( )
| expression
| ( expression [, ...] )
| ROLLUP ( { expression | ( expression [, ...] ) } [, ...] )
| CUBE ( { expression | ( expression [, ...] ) } [, ...] )
| GROUPING SETS ( grouping_element [, ...] )
```

- from_item中指定分区partition_clause为：

```
PARTITION { ( partition_name ) |
      FOR (  partition_value [, ...] ) }
```

说明

指定分区只适合分区表。

- 其中设置排序方式nlssort_expression_clause为：

```
NLSSORT ( column_name, ' NLS_SORT = { SCHINESE_PINYIN_M | generic_m_ci } ' )
```


第二个参数可选generic_m_ci，仅支持纯英文不区分大小写排序。

- 简化版查询语法，功能相当于SELECT * FROM table_name。
  ```
  TABLE { ONLY {(table_name)| table_name} | table_name [ * ]};
  ```

参数说明

- **WITH [RECURSIVE] with_query [, ...]**

  用于声明一个或多个可以在主查询中通过名称引用的子查询，相当于临时表。这种子查询语句结构称为CTE（Common Table Expression）结构，应用这种结构时，执行计划中将存在CTE SCAN的内容。

  如果声明了RECURSIVE，那么允许SELECT子查询通过名称引用它自己。

  其中with_query的详细格式为：

  ```
  with_query_name [ ( column_name [, ...] ) ] AS [ [ NOT ] MATERIALIZED ] ( {select | values | insert | update | delete} )
  ```

  - with_query_name指定子查询生成的结果集名称，在查询中可使用该名称访问子查询的结果集。
  - column_name指定子查询结果集中显示的列名。
  - 每个子查询可以是SELECT，VALUES，INSERT，UPDATE或DELETE语句。
  - 用户可以使用MATERIALIZED / NOT MATERIALIZED对CTE进行修饰。对于Stream计划，目前仅支持内联执行一种方式，此时该语法不生效。
    - 如果声明为MATERIALIZED，WITH查询将被物化，生成一个子查询结果集的拷贝，在引用处直接查询该拷贝，因此WITH子查询无法和主干SELECT语句进行联合优化（如谓词下推、等价类传递等）。当使用NOT MATERIALIZED进行修饰，如果WITH查询语义上可以作为子查询内联执行，则可以进行上述优化。
    - 如果用户没有显式声明物化属性则遵守以下规则：如果CTE只在所属SELECT主干中被引用一次，且语义上支持内联执行，则会被改写为子查询内联执行，否则以CTE Scan的方式物化执行。

- **plan_hint子句**

  以/*+ */的形式在SELECT关键字后，用于对SELECT对应语句块生成的计划进行hint调优，详细用法请参见章节**使用Plan Hint进行调优**。每条语句中只有第一个/*+ plan_hint */注释块会作为hint生效，里面可以写多条hint。

- **ALL**

  声明返回所有符合条件的行，是默认行为，可以省略该关键字。

- **DISTINCT [ON (expression [, ...])]**

  从SELECT的结果集中删除所有重复的行，使结果集中的每行都是唯一的。

  ON（expression [, ...]）只保留那些在给出的表达式上运算出相同结果的行集合中的第一行。

  须知

  DISTINCT ON表达式是使用与ORDER BY相同的规则进行解释的。除非使用了ORDER BY来保证需要的行首先出现，否则，"第一行"是不可预测的。

- **SELECT列表**

  指定查询表中列名，可以是部分列或者是全部（使用通配符*表示）。

通过使用子句AS output_name可以为输出字段取个别名，这个别名通常用于输出字段的显示。支持关键字name、value和type作为列别名。

列名可以用下面几种形式表达：

- 手动输入列名，多个列之间用英文逗号（,）分隔。
- 可以是FROM子句里面计算出来的字段。

- **FROM子句**

  为SELECT声明一个或者多个源表。

  FROM子句涉及的元素如下所示。

  - table_name

    表名或视图名，名称前可加上模式名，如：schema_name.table_name。

  - alias

    给表或复杂的表引用起一个临时的表别名，以便被其余的查询引用。

    别名用于缩写或者在自连接中消除歧义。如果提供了别名，它就会完全代替表的实际名称。

  - TABLESAMPLE sampling_method（argument [, ...]）[REPEATABLE（seed）]

    table_name之后的TABLESAMPLE子句表示应该用指定的sampling_method来检索表中行的子集。

    可选的REPEATABLE子句指定一个用于产生采样方法中随机数的种子数。种子值可以是任何非空常量值。如果查询时表没有被更改，指定相同种子和argument值的两个查询将会选择该表相同的采样。但是不同的种子值通常将会产生不同的采样。如果没有给出REPEATABLE，则会基于一个系统产生的种子为每一个查询选择一个新的随机采样。

  - column_alias

    列别名。

  - PARTITION

    查询分区表的某个分区的数据。

  - partition_name

    分区名。

  - partition_value

    指定的分区键值。在创建分区表时，如果指定了多个分区键，可以通过PARTITION FOR子句指定的这一组分区键的值，唯一确定一个分区。

  - subquery

    FROM子句中可以出现子查询，创建一个临时表保存子查询的输出。

  - with_query_name

    WITH子句同样可以作为FROM子句的源，可以通过WITH查询的名称对其进行引用。

  - function_name

    函数名称。函数调用也可以出现在FROM子句中。

  - join_type

    有5种类型，如下所示。

    - [INNER] JOIN

一个JOIN子句组合两个FROM项。可使用圆括号以决定嵌套的顺序。如果没有圆括号，JOIN从左向右嵌套。

- LEFT [OUTER] JOIN

  返回笛卡尔积中所有符合连接条件的行，再加上左表中通过连接条件没有匹配到右表行的那些行。这样，左边的行将扩展为生成表的全长，方法是在那些右表对应的字段位置填上NULL。请注意，只在计算匹配的时候，才使用JOIN子句的条件，外层的条件是在计算完毕之后施加的。

- RIGHT [OUTER] JOIN

  返回所有内连接的结果行，加上每个不匹配的右边行（左边用NULL扩展）。

  这只是一个符号上的方便，因为总是可以把它转换成一个LEFT OUTER JOIN，只要把左边和右边的输入互换位置即可。

- FULL [OUTER] JOIN

  返回所有内连接的结果行，加上每个不匹配的左边行（右边用NULL扩展），再加上每个不匹配的右边行（左边用NULL扩展）。

- CROSS JOIN

  CROSS JOIN等效于INNER JOIN ON（TRUE），即没有被条件删除的行。这种连接类型只是符号上的方便，因为它们与简单的FROM和WHERE的效果相同。

  说明

  必须为INNER和OUTER连接类型声明一个连接条件，即NATURAL ON，join_condition，USING (join_column [， ...]) 之一。但是它们不能出现在CROSS JOIN中。

其中CROSS JOIN和INNER JOIN生成一个简单的笛卡尔积，和在FROM的顶层列出两个项的结果相同。

– ON join_condition

  连接条件，用于限定连接中的哪些行是匹配的。如：ON left_table.a = right_table.a。不建议使用int等数值类型作为join_condition，因为int等数值类型可以隐式转换为bool值（非0值隐式转换为true，0转换为false），可能导致非预期的结果。

– USING(join_column[，...])

  ON left_table.a = right_table.a AND left_table.b = right_table.b ... 的简写。要求对应的列必须同名。

– NATURAL

  NATURAL是具有相同名称的两个表的所有列的USING列表的简写。

– from item

  用于连接的查询源对象的名称。

- **WHERE子句**

  WHERE子句构成一个行选择表达式，用来缩小SELECT查询的范围。condition是返回值为布尔型的任意表达式，任何不满足该条件的行都不会被检索。不建议使用int等数值类型作为condition，因为int等数值类型可以隐式转换为bool值（非0值隐式转换为true，0转换为false），可能导致非预期的结果。

WHERE子句中可以通过指定"(+)"操作符的方法将表的连接关系转换为外连接。但是不建议用户使用这种用法，因为这并不是SQL的标准语法，在做平台迁移的时候可能面临语法兼容性的问题。同时，使用"(+)"有很多限制：

a. "(+)"只能出现在where子句中。
b. 如果from子句中已经有指定表连接关系，那么不能再在where子句中使用"(+)"。
c. "(+)"只能作用在表或者视图的列上，不能作用在表达式上。
d. 如果表A和表B有多个连接条件，那么必须在所有的连接条件中指定"(+)"，否则"(+)"将不会生效，表连接会转化成内连接，并且不给出任何提示信息。
e. "(+)"作用的连接条件中的表不能跨查询或者子查询。如果"(+)"作用的表，不在当前查询或者子查询的from子句中，则会报错。如果"(+)"作用的对端的表不存在，则不报错，同时连接关系会转化为内连接。
f. "(+)"作用的表达式不能直接通过"OR"连接。
g. 如果"(+)"作用的列是和一个常量的比较关系， 那么这个表达式会成为JOIN条件的一部分。
h. 同一个表不能对应多个外表。
i. "(+)"只能出现"比较表达式"，"NOT表达式"，“ANY表达式”，“ALL表达式”，“IN表达式”，“NULLIF表达式”，“IS DISTINCT FROM表达式”，“IS OF”表达式。"(+)"不能出现在其他类型表达式中，并且这些表达式中不允许出现通过“AND”和“OR”连接的表达式。
j. "(+)"只能转化为左外连接或者右外连接，不能转化为全连接，即不能在一个表达式的两个表上同时指定"(+)"

须知

对于WHERE子句的LIKE操作符，当LIKE中要查询特殊字符“%”、“_”、“\”的时候需要使用反斜杠“\”来进行转义。

- **GROUP BY子句**

  将查询结果按某一列或多列的值分组，值相等的为一组。

  – CUBE ({ expression | (expression [, ...]) } [, ...])

    CUBE是自动对GROUP BY子句中列出的字段进行分组汇总，结果集将包含维度列中各值的所有可能组合，以及与这些维度值组合相匹配的基础行中的聚合值。它会为每个分组返回一行汇总信息， 用户可以使用CUBE来产生交叉表值。比如，在CUBE子句中给出三个表达式（n = 3），运算结果为$2^n = 2^3$ = 8组。 以n个表达式的值分组的行称为常规行，其余的行称为超级聚集行。

  – GROUPING SETS (grouping_element [, ...])

    GROUPING SETS子句是GROUP BY子句的进一步扩展，它可以使用户指定多个GROUP BY选项。 这样做可以通过裁剪用户不需要的数据组来提高效率。当用户指定了所需的数据组时，数据库不需要执行完整CUBE或ROLLUP生成的聚合集合。

须知

- 如果SELECT列表的表达式中引用了那些没有分组的字段，则会报错，除非使用了聚集函数，因为对于未分组的字段，可能返回多个数值。
- 如果SELECT列表的表达式中引用了常量，则无需在GROUP BY子句中对该常量进行分组，否则会报错。

- **HAVING子句**

  与GROUP BY子句配合用来选择特殊的组。HAVING子句将组的一些属性与一个常数值比较，只有满足HAVING子句中的逻辑表达式的组才会被提取出来。

- **WINDOW子句**

  一般形式为WINDOW window_name AS（window_definition）[， ...]，window_name是可以被随后的窗口定义所引用的名称，window_definition可以是以下的形式：

  ```
  [ existing_window_name ]
  [ PARTITION BY expression [, ...] ]
  [ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS { FIRST | LAST } ] [, ...] ]
  [ frame_clause ]
  ```

  frame_clause为窗函数定义一个窗口框架window frame，窗函数（并非所有）依赖于框架，window frame是当前查询行的一组相关行。frame_clause可以是以下的形式：

  ```
  [ RANGE | ROWS ] frame_start
  [ RANGE | ROWS ] BETWEEN frame_start AND frame_end
  ```

  frame_start和frame_end可以是：

  ```
  UNBOUNDED PRECEDING
  value PRECEDING
  CURRENT ROW
  value FOLLOWING
  UNBOUNDED FOLLOWING
  ```

- **UNION子句**

  UNION计算多个SELECT语句返回行集合的并集。

  UNION子句有如下约束条件：

  - 除非声明了ALL子句，否则缺省的UNION结果不包含重复的行。
  - 同一个SELECT语句中的多个UNION操作符是从左向右计算的，除非用圆括号进行了标识。
  - FOR UPDATE不能在UNION的结果或输入中声明。

  一般表达式：

  ```
  select_statement UNION [ALL] select_statement
  ```

  - select_statement可以是任何没有ORDER BY、LIMIT、FOR UPDATE子句的SELECT语句。
  - 如果用圆括号包围，ORDER BY和LIMIT可以附着在子表达式里。

- **INTERSECT子句**

  INTERSECT计算多个SELECT语句返回行集合的交集，不含重复的记录。

  INTERSECT子句有如下约束条件：

  - 同一个SELECT语句中的多个INTERSECT操作符是从左向右计算的，除非用圆括号进行了标识。
  - 当对多个SELECT语句的执行结果进行UNION和INTERSECT操作的时候，会优先处理INTERSECT。

一般形式：

```
select_statement INTERSECT select_statement
```

select_statement可以是任何没有FOR UPDATE子句的SELECT语句。

- **EXCEPT子句**

  EXCEPT子句有如下的通用形式：

  ```
  select_statement EXCEPT [ ALL ] select_statement
  ```

  select_statement是任何没有FOR UPDATE子句的SELECT表达式。

  EXCEPT操作符计算存在于左边SELECT语句的输出而不存在于右边SELECT语句输出的行。

  EXCEPT的结果不包含任何重复的行，除非声明了ALL选项。使用ALL时，一个在左边表中有m个重复而在右边表中有n个重复的行将在结果中出现max(m-n,0) 次。

  除非用圆括号指明顺序，否则同一个SELECT语句中的多个EXCEPT操作符是从左向右计算的。EXCEPT和UNION的绑定级别相同。

  目前，不能给EXCEPT的结果或者任何EXCEPT的输入声明FOR UPDATE子句。

- **MINUS子句**

  与EXCEPT子句具有相同的功能和用法。

- **ORDER BY子句**

  对SELECT语句检索得到的数据进行升序或降序排序。对于ORDER BY表达式中包含多列的情况：

  - 首先根据最左边的列进行排序，如果这一列的值相同，则根据下一个表达式进行比较。
  - 如果对于所有声明的表达式都相同，则按随机顺序返回。
  - 在与DISTINCT关键字一起使用的情况下，ORDER BY中排序的列必须包括在SELECT语句所检索的结果集的列中。
  - 在与GROUP BY子句一起使用的情况下，ORDER BY中排序的列必须包括在SELECT语句所检索的结果集的列中。

  须知

  如果要支持中文拼音排序，需要在初始化数据库时指定编码格式为UTF-8、GB18030或GBK。命令如下:

  initdb –E UTF8 –D ../data –locale=zh_CN.UTF-8、initdb -E GB18030 -D ../data -locale=zh_CN.GB18030或initdb –E GBK –D ../data –locale=zh_CN.GBK。

- **LIMIT子句**

  LIMIT子句由两个独立的子句组成：

  LIMIT { count | ALL }限制返回行数，count为指定行数，LIMIT ALL的效果和省略LIMIT子句一样。

  OFFSET start count声明返回的最大行数，而start声明开始返回行之前忽略的行数。如果两个都指定了，会在开始计算count个返回行之前先跳过start行。

- **OFFSET子句**

  SQL：2008开始提出一种不同的语法：

  OFFSET start { ROW | ROWS }

  start声明开始返回行之前忽略的行数。

- **FETCH { FIRST | NEXT } [count] { ROW | ROWS } ONLY**

  如果不指定count，默认值为1，FETCH子句限定返回查询结果从第一行开始的总行数。

- **FOR UPDATE子句**

  FOR UPDATE子句将对SELECT检索出来的行进行加锁。这样避免它们在当前事务结束前被其他事务修改或者删除，即其他企图UPDATE、 DELETE、 SELECT FOR UPDATE这些行的事务将被阻塞，直到当前事务结束。

  为了避免操作等待其他事务提交，可使用NOWAIT选项，如果被选择的行不能立即被锁住，执行SELECT FOR UPDATE NOWAIT将会立即报错，而不是等待；WAIT N选项，如果被选择的行不能立即被锁住，等待n秒（其中，n为int类型，取值范围：0 <= n <= 2147483），n秒内获取锁则正常执行，否则报错。

  FOR SHARE的行为类似，只是它在每个检索出来的行上要求一个共享锁，而不是一个排他锁。一个共享锁阻塞其它事务执行UPDATE、DELETE、SELECT，不阻塞SELECT FOR SHARE。

  如果在FOR UPDATE或FOR SHARE中明确指定了表名称，则只有这些指定的表被锁定，其他在SELECT中使用的表将不会被锁定。否则，将锁定该命令中所有使用的表。

  如果FOR UPDATE或FOR SHARE应用于一个视图或者子查询，它同样将锁定所有该视图或子查询中使用到的表。

  多个FOR UPDATE和FOR SHARE子句可以用于为不同的表指定不同的锁定模式。

  如果一个表中同时出现（或隐含同时出现）在FOR UPDATE和FOR SHARE子句中，则按照FOR UPDATE处理。类似的，如果影响一个表的任意子句中出现了NOWAIT，该表将按照NOWAIT处理。

  须知

  对于FOR UPDATE/SHARE，执行计划不能下推的SQL，直接返回报错信息；对于执行计划可以下推的，下推到DN执行。

  分布式场景下不支持对多表FOR UPDATE/SHAR。

- **NLS_SORT**

  指定某字段按照特殊方式排序。目前仅支持中文拼音格式排序和不区分大小写排序。如果要支持此排序方式，在创建数据库时需要指定编码格式为“UTF8”、”GB18030”或“GBK”；如果指定为其他编码，例如SQL_ASCII，则可能报错或者排序无效。

  取值范围：

  - SCHINESE_PINYIN_M，按照中文拼音排序。
  - generic_m_ci，不区分大小写排序（可选，仅支持纯英文不区分大小写排序）。

- **PARTITION子句**

  查询某个分区表中相应分区的数据。

示例

```
--创建SCHEMA。
openGauss=# CREATE SCHEMA tpcds;

--创建表tpcds.reason。
```



```
openGauss=# CREATE TABLE tpcds.reason
(
  r_reason_sk       integer,
  r_reason_id       character(16),
  r_reason_desc     character(100)
);

--向表中插入多条记录。
openGauss=# INSERT INTO tpcds.reason values(3,'AAAAAAAABAAAAAAA','reason 1'),
(10,'AAAAAAAABAAAAAAA','reason 2'),(4,'AAAAAAAABAAAAAAA','reason 3'),
(10,'AAAAAAAABAAAAAAA','reason 4'),(10,'AAAAAAAABAAAAAAA','reason 5'),
(20,'AAAAAAAACAAAAAAA','N%reason 6'),(30,'AAAAAAAACAAAAAAA','W%reason 7');

--先通过子查询得到一张临时表temp_t，然后查询表temp_t中的所有数据。
openGauss=# WITH temp_t(name,isdba) AS (SELECT usename,usesuper FROM pg_user) SELECT * FROM
temp_t;

--查询tpcds.reason表的所有r_reason_sk记录，且去除重复。
openGauss=# SELECT DISTINCT(r_reason_sk) FROM tpcds.reason;

--LIMIT子句示例：获取表中一条记录。
openGauss=# SELECT * FROM tpcds.reason LIMIT 1;

--查询所有记录，且按字母升序排列。
openGauss=# SELECT r_reason_desc FROM tpcds.reason ORDER BY r_reason_desc;

--通过表别名，从pg_user和pg_user_status这两张表中获取数据。
openGauss=# SELECT a.usename,b.locktime FROM pg_user a,pg_user_status b WHERE a.usesysid=b.roloid;

--FULL JOIN子句示例：将pg_user和pg_user_status这两张表的数据进行全连接显示，即数据的合集。
openGauss=# SELECT a.usename,b.locktime,a.usesuper FROM pg_user a FULL JOIN pg_user_status b on
a.usesysid=b.roloid;

--GROUP BY子句示例：根据查询条件过滤，并对结果进行分组。
openGauss=# SELECT r_reason_id, AVG(r_reason_sk) FROM tpcds.reason GROUP BY r_reason_id HAVING
AVG(r_reason_sk) > 25;

--GROUP BY CUBE子句示例：根据查询条件过滤，并对结果进行分组汇总。
openGauss=# SELECT r_reason_id,AVG(r_reason_sk) FROM tpcds.reason GROUP BY
CUBE(r_reason_id,r_reason_sk);

--GROUP BY GROUPING SETS子句示例:根据查询条件过滤，并对结果进行分组汇总。
openGauss=# SELECT r_reason_id,AVG(r_reason_sk) FROM tpcds.reason GROUP BY GROUPING
SETS((r_reason_id,r_reason_sk),r_reason_sk);

--UNION子句示例：将表tpcds.reason里r_reason_desc字段中的内容以W开头和以N开头的进行合并。
openGauss=# SELECT r_reason_sk, tpcds.reason.r_reason_desc
    FROM tpcds.reason
    WHERE tpcds.reason.r_reason_desc LIKE 'W%'
UNION
SELECT r_reason_sk, tpcds.reason.r_reason_desc
    FROM tpcds.reason
    WHERE tpcds.reason.r_reason_desc LIKE 'N%';

--NLS_SORT子句示例：中文拼音排序。
openGauss=# SELECT * FROM tpcds.reason ORDER BY NLSSORT( r_reason_desc, 'NLS_SORT =
SCHINESE_PINYIN_M');


--不区分大小写排序（可选，仅支持纯英文不区分大小写排序）：
openGauss=# SELECT * FROM tpcds.reason ORDER BY NLSSORT( r_reason_desc, 'NLS_SORT =
generic_m_ci');

--创建分区表tpcds.reason_p。
openGauss=# CREATE TABLE tpcds.reason_p
(
  r_reason_sk integer,
  r_reason_id character(16),
  r_reason_desc character(100)
```



```
)
PARTITION BY RANGE (r_reason_sk)
(
  partition P_05_BEFORE values less than (05),
  partition P_15 values less than (15),
  partition P_25 values less than (25),
  partition P_35 values less than (35),
  partition P_45_AFTER values less than (MAXVALUE)
)
;

--插入数据。
openGauss=# INSERT INTO tpcds.reason_p values(3,'AAAAAAAABAAAAAAA','reason 1'),
(10,'AAAAAAAABAAAAAAA','reason 2'),(4,'AAAAAAAABAAAAAAA','reason 3'),
(10,'AAAAAAAABAAAAAAA','reason 4'),(10,'AAAAAAAABAAAAAAA','reason 5'),
(20,'AAAAAAAACAAAAAAA','reason 6'),(30,'AAAAAAAACAAAAAAA','reason 7');

--PARTITION子句示例：从tpcds.reason_p的表分区P_05_BEFORE中获取数据。
openGauss=#  SELECT * FROM tpcds.reason_p PARTITION (P_05_BEFORE);
 r_reason_sk |   r_reason_id    |   r_reason_desc
-------------+------------------+----------------------------------
           3 | AAAAAAAABAAAAAAA | reason 1
           4 | AAAAAAAABAAAAAAA | reason 3
(2 rows)

--GROUP BY子句示例：按r_reason_id分组统计tpcds.reason_p表中的记录数。
openGauss=# SELECT COUNT(*),r_reason_id FROM tpcds.reason_p GROUP BY r_reason_id;
 count |   r_reason_id
-------+------------------
     2 | AAAAAAAACAAAAAAA
     5 | AAAAAAAABAAAAAAA
(2 rows)

--GROUP BY CUBE子句示例：根据查询条件过滤，并对查询结果分组汇总。
openGauss=# SELECT * FROM tpcds.reason GROUP BY  CUBE (r_reason_id,r_reason_sk,r_reason_desc);

--GROUP BY GROUPING SETS子句示例：根据查询条件过滤，并对查询结果分组汇总。
openGauss=# SELECT * FROM tpcds.reason GROUP BY  GROUPING SETS
((r_reason_id,r_reason_sk),r_reason_desc);

--HAVING子句示例：按r_reason_id分组统计tpcds.reason_p表中的记录，并只显示r_reason_id个数大于2的信息。
openGauss=# SELECT COUNT(*) c,r_reason_id FROM tpcds.reason_p GROUP BY r_reason_id HAVING c>2;
 c |   r_reason_id
---+------------------
 5 | AAAAAAAABAAAAAAA
(1 row)

--IN子句示例：按r_reason_id分组统计tpcds.reason_p表中的r_reason_id个数，并只显示r_reason_id值为AAAAAAAABAAAAAAA或AAAAAAAADAAAAAAA的个数。
openGauss=# SELECT COUNT(*),r_reason_id FROM tpcds.reason_p GROUP BY r_reason_id HAVING
r_reason_id IN('AAAAAAAABAAAAAAA','AAAAAAAADAAAAAAA');
count |   r_reason_id
-------+------------------
     5 | AAAAAAAABAAAAAAA
(1 row)

--INTERSECT子句示例：查询r_reason_id等于AAAAAAAABAAAAAAA，并且r_reason_sk小于5的信息。
openGauss=# SELECT * FROM tpcds.reason_p WHERE r_reason_id='AAAAAAAABAAAAAAA' INTERSECT
SELECT * FROM tpcds.reason_p WHERE r_reason_sk<5;
 r_reason_sk |   r_reason_id    |     r_reason_desc
-------------+------------------+----------------------------------
           4 | AAAAAAAABAAAAAAA | reason 3
           3 | AAAAAAAABAAAAAAA | reason 1
(2 rows)

--EXCEPT子句示例：查询r_reason_id等于AAAAAAAABAAAAAAA，并且去除r_reason_sk小于4的信息。
openGauss=# SELECT * FROM tpcds.reason_p WHERE r_reason_id='AAAAAAAABAAAAAAA' EXCEPT SELECT *
FROM tpcds.reason_p WHERE r_reason_sk<4;
```



```
r_reason_sk |   r_reason_id    |      r_reason_desc
-------------+------------------+----------------------------------
          10 | AAAAAAAABAAAAAAA | reason 5
          10 | AAAAAAAABAAAAAAA | reason 4
           4 | AAAAAAAABAAAAAAA | reason 3
          10 | AAAAAAAABAAAAAAA | reason 2
(4 rows)

--创建表store_returns、customer
openGauss=# CREATE TABLE tpcds.store_returns (sr_item_sk int, sr_customer_id varchar(50),sr_customer_sk
int);
openGauss=# CREATE TABLE tpcds.customer (c_item_sk int, c_customer_id varchar(50),c_customer_sk int);
openGauss=# INSERT INTO tpcds.store_returns VALUES(18000);
openGauss=# INSERT INTO tpcds.customer (c_customer_id) VALUES('AAAAAAAAJNGEBAAA');

--通过在where子句中指定"(+)"来实现左连接。
openGauss=#
SELECT t1.sr_item_sk ,t2.c_customer_id FROM tpcds.store_returns t1, tpcds.customer t2 WHERE
t1.sr_customer_sk  = t2.c_customer_sk(+) ORDER BY 1 DESC LIMIT 1;
 sr_item_sk | c_customer_id
------------+---------------
      18000 |
(1 row)

--通过在where子句中指定"(+)"来实现右连接。
openGauss=#
SELECT t1.sr_item_sk ,t2.c_customer_id FROM tpcds.store_returns t1, tpcds.customer t2 WHERE
t1.sr_customer_sk(+)  = t2.c_customer_sk
ORDER BY 1 DESC LIMIT 1
 sr_item_sk |  c_customer_id
------------+------------------
            | AAAAAAAAJNGEBAAA
(1 row)

--通过在where子句中指定"(+)"来实现左连接，并且增加连接条件。
openGauss=# SELECT t1.sr_item_sk ,t2.c_customer_id FROM tpcds.store_returns t1, tpcds.customer t2
WHERE t1.sr_customer_sk  = t2.c_customer_sk(+) AND t2.c_customer_sk(+) < 1 ORDER BY 1 LIMIT 1;
sr_item_sk | c_customer_id
------------+---------------
          1 |
(1 row)

--不支持在where子句中指定"(+)"的同时使用内层嵌套AND/OR的表达式。
openGauss=#
SELECT t1.sr_item_sk ,t2.c_customer_id FROM tpcds.store_returns t1, tpcds.customer t2 WHERE
NOT(t1.sr_customer_sk  = t2.c_customer_sk(+) AND t2.c_customer_sk(+) < 1);
ERROR:  Operator "(+)" can not be used in nesting expression.
LINE 1: ...tomer_id from store_returns t1, customer t2 where not(t1.sr_...
                                                             ^
--where子句在不支持表达式宏指定"(+)"会报错。
openGauss=#
SELECT t1.sr_item_sk ,t2.c_customer_id FROM tpcds.store_returns t1, tpcds.customer t2 WHERE
(t1.sr_customer_sk  = t2.c_customer_sk(+))::bool;
ERROR:  Operator "(+)" can only be used in common expression.

--where子句在表达式的两边都指定"(+)"会报错。
openGauss=# SELECT t1.sr_item_sk ,t2.c_customer_id FROM tpcds.store_returns t1, tpcds.customer t2
WHERE t1.sr_customer_sk(+)  = t2.c_customer_sk(+);
ERROR:  Operator "(+)" can't be specified on more than one relation in one join condition
HINT:  "t1", "t2"...are specified Operator "(+)" in one condition.

--删除表。
openGauss=# DROP TABLE tpcds.reason_p, tpcds.reason;

--删除SCHEMA。
openGauss=# DROP SCHEMA tpcds CASCADE;
```


7.13.131 SELECT INTO

功能描述

SELECT INTO用于根据查询结果创建一个新表，并且将查询到的数据插入到新表中。

数据并不返回给客户端，这一点和普通的SELECT不同。新表的字段具有和SELECT的输出字段相同的名称和数据类型。

注意事项

CREATE TABLE AS的作用和SELECT INTO类似，且提供了SELECT INTO所提供功能的超集。建议使用CREATE TABLE AS语法替代SELECT INTO，因为SELECT INTO不能在存储过程中使用。

语法格式

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    { * | {expression [ [ AS ] output_name ]} [, ...] }
    INTO [ UNLOGGED ] [ TABLE ] new_table
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY expression [, ...] ]
    [ HAVING condition [, ...] ]
    [ WINDOW {window_name AS ( window_definition )} [, ...] ]
    [ { UNION | INTERSECT | EXCEPT | MINUS } [ ALL | DISTINCT ] select ]
    [ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS { FIRST |
LAST } ]} [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start [ ROW | ROWS ] ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
    [ {FOR { UPDATE | SHARE } [ OF table_name [, ...] ] [ NOWAIT | WAIT N]} [...] ];
```

参数说明

- **new_table**

  new_table指定新建表的名称。

- **UNLOGGED**

  指定表为非日志表。在非日志表中写入的数据不会被写入到预写日志中，这样就会比普通表快很多。但是，它也是不安全的，在冲突或异常关机导致数据库重启后，非日志表数据会被清空。非日志表中的内容也不会被复制到备用服务器中。在该类表中创建的索引也不会被自动记录。

  - 使用场景：非日志表不能保证数据的安全性，用户应该在确保数据已经做好备份的前提下使用，例如系统升级时进行数据的备份。
  - 故障处理：当异常关机等操作导致非日志表上的索引发生数据丢失时，用户应该对发生错误的索引进行重建。

- **INTO [UNLOGGED] [TABLE] new_table**

  UNLOGGED指定表为非日志表。在非日志表中写入的数据不会被写入到预写日志中，这样就会比普通表快很多。但是，它也是不安全的，在冲突或异常关机导致数据库重启后，非日志表数据会被清空。非日志表中的内容也不会被复制到备用服务器中。在该类表中创建的索引也不会被自动记录。

  new_table指定新建表的名称。

说明

SELECT INTO的其它参数可参考SELECT的**参数说明**。

示例

```
--创建SCHEMA。
openGauss=# CREATE SCHEMA tpcds;

--创建表tpcds.reason。
openGauss=# CREATE TABLE tpcds.reason
(
  r_reason_sk       integer,
  r_reason_id       character(16),
  r_reason_desc     character(100)
);

--向表中插入多条记录。
openGauss=# INSERT INTO tpcds.reason values(1,'AAAAAAAABAAAAAAA','reason 1'),
(2,'AAAAAAAABAAAAAAA','reason 2'),(3,'AAAAAAAABAAAAAAA','reason 3'),
(4,'AAAAAAAABAAAAAAA','reason 4'),(4,'AAAAAAAABAAAAAAA','reason 5'),
(4,'AAAAAAAACAAAAAAA','reason 6'),(5,'AAAAAAAACAAAAAAA','reason 7');

--将tpcds.reason表中r_reason_sk小于5的值加入到新建表中。
openGauss=# SELECT * INTO tpcds.reason_t1 FROM tpcds.reason WHERE r_reason_sk < 5;
INSERT 0 6

--删除表。
openGauss=# DROP TABLE tpcds.reason_t1, tpcds.reason;

--删除SCHEMA。
openGauss=# DROP SCHEMA tpcds CASCADE;
```

相关链接

SELECT

7.13.132 SET

功能描述

用于修改运行时配置参数。

注意事项

大多数运行时参数都可以用SET在运行时设置，但有些则在服务运行过程中或会话开始之后不能修改。

语法格式

- 设置所处的时区。
  ```
  SET [ SESSION | LOCAL ] TIME ZONE { timezone | LOCAL | DEFAULT };
  ```
- 设置所属的模式。
  ```
  SET [ SESSION | LOCAL ]
    {CURRENT_SCHEMA { TO | = } { schema | DEFAULT }
    | SCHEMA 'schema'};
  ```
- 设置客户端编码集。
  ```
  SET [ SESSION | LOCAL ] NAMES encoding_name;
  ```
- 设置XML的解析方式。
  ```
  SET [ SESSION | LOCAL ] XML OPTION { DOCUMENT | CONTENT };
  ```


- 设置其他运行时参数。

```
SET [ LOCAL | SESSION ]
    {config_parameter { { TO | = } { value | DEFAULT }
                      | FROM CURRENT }};
```

参数说明

- **SESSION**

  声明的参数只对当前会话起作用。如果SESSION和LOCAL都没出现，则SESSION为缺省值。

  如果在事务中执行了此命令，命令的产生影响将在事务回滚之后消失。如果该事务已提交，影响将持续到会话的结束，除非被另外一个SET命令重置参数。

- **LOCAL**

  声明的参数只在当前事务中有效。在COMMIT或ROLLBACK之后，会话级别的设置将再次生效。

  不论事务是否提交，此命令的影响只持续到当前事务结束。一个特例是：在一个事务里面，既有SET命令，又有SET LOCAL命令，且SET LOCAL在SET后面，则在事务结束之前，SET LOCAL命令会起作用，但事务提交之后，则是SET命令会生效。

- **TIME ZONE timezone**

  用于指定当前会话的本地时区。

  取值范围：有效的本地时区。该选项对应的运行时参数名称为TimeZone，DEFAULT缺省值为PRC。

- **CURRENT_SCHEMA schema**

  CURRENT_SCHEMA用于指定当前的模式。

  取值范围：已存在模式名称。

- **SCHEMA schema**

  同CURRENT_SCHEMA。此处的schema是个字符串。

  例如：set schema 'public';

- **NAMES encoding_name**

  用于设置客户端的字符编码。等价于set client_encoding to encoding_name。

  取值范围：有效的字符编码。该选项对应的运行时参数名称为client_encoding，默认编码为UTF8。

- **XML OPTION option**

  用于设置XML的解析方式。

  取值范围：CONTENT（缺省）、DOCUMENT

- **config_parameter**

  可设置的运行时参数的名称。可用的运行时参数可以使用SHOW ALL命令查看。

  说明

  部分通过SHOW ALL查看的参数不能通过SET设置。如max_datanodes。

- **value**

  config_parameter的新值。可以声明为字符串常量、标识符、数字，或者逗号分隔的列表。DEFAULT用于把这些参数设置为它们的缺省值。

示例

```
--设置模式搜索路径。
openGauss=# SET search_path TO tpcds, public;

--把日期时间风格设置为传统的 POSTGRES 风格(日在月前)。
openGauss=# SET datestyle TO postgres;
```

相关链接

RESET，**SHOW**

7.13.133 SET CONSTRAINTS

功能描述

SET CONSTRAINTS设置当前事务检查行为的约束条件。

IMMEDIATE约束是在每条语句后面进行检查。DEFERRED约束一直到事务提交时才检查。每个约束都有自己的模式。

从创建约束条件开始，一个约束总是设定为DEFERRABLE INITIALLY DEFERRED，DEFERRABLE INITIALLY IMMEDIATE，NOT DEFERRABLE三个特性之一。第三种总是IMMEDIATE，并且不会受SET CONSTRAINTS影响。前两种以指定的方式启动每个事务，但是其行为可以在事务里用SET CONSTRAINTS改变。

带着一个约束名列表的SET CONSTRAINTS改变这些约束的模式（都必须是可推迟的）。如果有多个约束匹配某个名称，则所有都会被影响。SET CONSTRAINTS ALL改变所有可推迟约束的模式。

当SET CONSTRAINTS把一个约束从DEFERRED改成IMMEDIATE的时候，新模式反作用式地起作用：任何将在事务结束准备进行的数据修改都将在SET CONSTRAINTS的时候执行检查。如果违反了任何约束，SET CONSTRAINTS都会失败（并且不会修改约束模式）。因此，SET CONSTRAINTS可以用于强制在事务中某一点进行约束检查。

检查和唯一约束总是不可推迟的。

注意事项

SET CONSTRAINTS只在当前事务里设置约束的行为。因此，如果用户在事务块之外（START TRANSACTION/COMMIT对）执行这个命令，它将没有任何作用。

语法格式

```
SET CONSTRAINTS { ALL | { name } [, ...] } { DEFERRED | IMMEDIATE };
```

参数说明

- **name**
  约束名。
  取值范围：已存在的约束名。可以在系统表pg_constraint中查到。
- **ALL**
  所有约束。
- **DEFERRED**
  约束一直到事务提交时才检查。

- **IMMEDIATE**
  约束在每条语句后进行检查。

示例

```
--设置所有约束在事务提交时检查。
openGauss=# SET CONSTRAINTS ALL DEFERRED;
```

7.13.134 SET ROLE

功能描述

设置当前会话的当前用户标识符。

注意事项

- 当前会话的用户必须是指定的rolename角色的成员，当三权分立关闭时，系统管理员可以选择任何角色。
- 使用这条命令，它可能会增加一个用户的权限，也可能会限制一个用户的权限。如果会话用户的角色有INHERITS属性，则它自动拥有它能SET ROLE变成的角色的所有权限；在这种情况下，SET ROLE实际上是删除了所有直接赋予会话用户的权限，以及它的所属角色的权限，只剩下指定角色的权限。另一方面，如果会话用户的角色有NOINHERITS属性，SET ROLE删除直接赋予会话用户的权限，而获取指定角色的权限。

语法格式

- 设置当前会话的当前用户标识符。
  ```
  SET [ SESSION | LOCAL ] ROLE role_name PASSWORD 'password';
  ```
- 重置当前用户标识为当前会话用户标识符。
  ```
  RESET ROLE;
  ```

参数说明

- **SESSION**
  声明这个命令只对当前会话起作用，此参数为缺省值。
- **LOCAL**
  声明该命令只在当前事务中有效。
- **role_name**
  角色名。
  取值范围：字符串，要符合**标识符命名规范**。
- **password**
  角色的密码。要求符合密码的命名规则。不支持直接使用密文密码。
- **RESET ROLE**
  用于重置当前用户标识。

示例

```
--创建角色paul。
openGauss=# CREATE ROLE paul IDENTIFIED BY '********';
```



```
--设置当前用户为paul。
openGauss=# SET ROLE paul PASSWORD '********';

--查看当前会话用户，当前用户。
openGauss=> SELECT SESSION_USER, CURRENT_USER;

--重置当前用户。
openGauss=> RESET ROLE;

--删除用户。
openGauss=# DROP USER paul;
```

7.13.135 SET SESSION AUTHORIZATION

功能描述

把当前会话里的会话用户标识和当前用户标识都设置为指定的用户。

注意事项

只有在初始会话用户有系统管理员权限的时候，会话用户标识符才能改变。否则，只有在指定了被认证的用户名的情况下，系统才接受该命令。

语法格式

- 为当前会话设置会话用户标识符和当前用户标识符。
  ```
  SET [ SESSION | LOCAL ] SESSION AUTHORIZATION role_name PASSWORD 'password';
  ```
- 重置会话和当前用户标识符为初始认证的用户名。
  ```
  {SET [ SESSION | LOCAL ] SESSION AUTHORIZATION DEFAULT
      | RESET SESSION AUTHORIZATION};
  ```

参数说明

- **SESSION**
  声明这个命令只对当前会话起作用。
- **LOCAL**
  声明该命令只在当前事务中有效。
- **role_name**
  用户名。
  取值范围：字符串，要符合**标识符命名规范**。
- **password**
  角色的密码。要求符合密码的命名规则。不支持直接使用密文密码。
- **DEFAULT**
  重置会话和当前用户标识符为初始认证的用户名。

示例

```
--创建角色paul。
openGauss=# CREATE ROLE paul IDENTIFIED BY '********';

--设置当前用户为paul。
openGauss=# SET SESSION AUTHORIZATION paul password '********';

--查看当前会话用户，当前用户。
openGauss=> SELECT SESSION_USER, CURRENT_USER;
```



```
--重置当前用户。
openGauss=> RESET SESSION AUTHORIZATION;

--删除用户。
openGauss=# DROP USER paul;
```

相关参考

SET ROLE

7.13.136 SET TRANSACTION

功能描述

为事务设置特性。事务特性包括事务隔离级别、事务访问模式(读/写或者只读)。可以设置当前事务的特性（LOCAL)，也可以设置会话的默认事务特性(SESSION)。

注意事项

设置当前事务特性需要在事务中执行（即执行SET TRANSACTION之前需要执行START TRANSACTION或者BEGIN），否则设置不生效。

语法格式

设置事务的隔离级别、读写模式。

```
{ SET [ LOCAL ] TRANSACTION|SET SESSION CHARACTERISTICS AS TRANSACTION }
  { ISOLATION LEVEL { READ COMMITTED | READ UNCOMMITTED | SERIALIZABLE | REPEATABLE READ }
  | { READ WRITE | READ ONLY } } [, ...]
```

参数说明

- **LOCAL**

  声明该命令只在当前事务中有效。

- **SESSION**

  声明这个命令只对当前会话起作用。

- **ISOLATION_LEVEL_CLAUSE**

  指定事务隔离级别，该参数决定当一个事务中存在其他并发运行事务时能够看到什么数据。

  说明

  - 在事务中第一个数据修改语句（INSERT，DELETE，UPDATE，FETCH，COPY）执行之后，当前事务的隔离级别就不能再次设置。

  取值范围：

  - READ COMMITTED：读已提交隔离级别，只能读到已经提交的数据，而不会读到未提交的数据。这是缺省值。
  - READ UNCOMMITTED：读未提交隔离级别，可能会读到未提交的数据。提供这个隔离级别可用于在存在某协调节点CN故障等情况下应急使用，建议这种隔离级别下仅做只读操作，避免造成数据不一致。
  - REPEATABLE READ：可重复读隔离级别，仅仅能看到事务开始之前提交的数据，不能看到未提交的数据，以及在事务执行期间由其它并发事务提交的修改。

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

- **READ WRITE | READ ONLY**

  指定事务访问模式（读/写或者只读）。

  说明

  会话的默认事务特性的访问模式只能在启动数据库时或者通过发送HUP信号进行设置。

示例

```
--开启一个事务，设置事务的隔离级别为READ COMMITTED，访问模式为READ ONLY。
openGauss=# START TRANSACTION;
openGauss=# SET LOCAL TRANSACTION ISOLATION LEVEL READ COMMITTED READ ONLY;
openGauss=# COMMIT;
```

7.13.137 SHOW

功能描述

SHOW将显示当前运行时参数的数值。

语法格式

```
SHOW
 {
   [VARIABLES LIKE] configuration_parameter |
   CURRENT_SCHEMA |
   TIME ZONE |
   TRANSACTION ISOLATION LEVEL |
   SESSION AUTHORIZATION |
   ALL
 };
```

参数说明

显示变量的参数请参见RESET的**参数说明**。

示例

```
--显示 timezone 参数值。
openGauss=# SHOW timezone;

--显示所有参数。
openGauss=# SHOW ALL;

--显示参数名中包含”var”的所有参数
openGauss=# SHOW VARIABLES LIKE var;
```

相关链接

SET，**RESET**

7.13.138 SHUTDOWN

功能描述

SHUTDOWN将关闭当前连接的数据库节点。

注意事项

- 仅拥有管理员权限的用户可以运行此命令。
- 分布式数据库不支持SHUTDOWN命令。

语法格式

```
SHUTDOWN [FAST | IMMEDIATE];
```

参数说明

- **""**

  不指定关闭模式，默认为fast。

- **fast**

  不等待客户端中断连接，将所有活跃事务回滚并且强制断开客户端，然后关闭数据库节点。

- **immediate**

  强行关闭，在下次重新启动的时候将导致故障恢复。

示例

```
--关闭当前数据库节点。
openGauss=# SHUTDOWN;

--使用fast模式关闭当前数据库节点。
openGauss=# SHUTDOWN FAST;
```

7.13.139 START TRANSACTION

功能描述

通过START TRANSACTION启动事务。如果声明了隔离级别、读写模式，那么新事务就使用这些特性，类似执行了**SET TRANSACTION**。

语法格式

格式一：START TRANSACTION格式

```
START TRANSACTION
 [
  {
    ISOLATION LEVEL { READ COMMITTED | READ UNCOMMITTED | SERIALIZABLE | REPEATABLE READ }
    | { READ WRITE | READ ONLY }
  } [, ...]
 ];
```

格式二：BEGIN格式

```
BEGIN [ WORK | TRANSACTION ]
 [
  {
    ISOLATION LEVEL { READ COMMITTED | READ UNCOMMITTED | SERIALIZABLE | REPEATABLE READ }
    | { READ WRITE | READ ONLY }
  } [, ...]
 ];
```


参数说明

- **WORK | TRANSACTION**

  BEGIN格式中的可选关键字，没有实际作用。

- **ISOLATION LEVEL**

  指定事务隔离级别，它决定当一个事务中存在其他并发运行事务时它能够看到什么数据。

  说明

  在事务中第一个数据修改语句（INSERT，DELETE，UPDATE，FETCH，COPY）执行之后，事务隔离级别就不能再次设置。

  取值范围：

  - READ COMMITTED：读已提交隔离级别，只能读到已经提交的数据，而不会读到未提交的数据。这是缺省值。
  - READ UNCOMMITTED：读未提交隔离级别，可能会读到未提交的数据。提供这个隔离级别可用于在存在某协调节点CN故障等情况下应急使用，建议这种隔离级别下仅做只读操作，避免造成数据不一致。
  - REPEATABLE READ： 可重复读隔离级别，仅仅看到事务开始之前提交的数据，它不能看到未提交的数据，以及在事务执行期间由其它并发事务提交的修改。
  - SERIALIZABLE：GaussDB目前功能上不支持此隔离级别，等价于REPEATABLE READ。

- **READ WRITE | READ ONLY**

  指定事务访问模式（读/写或者只读）。

示例

```
--以默认方式启动事务。
openGauss=# START TRANSACTION;
openGauss=# SELECT * FROM tpcds.reason;
openGauss=# END;

--以默认方式启动事务。
openGauss=# BEGIN;
openGauss=# SELECT * FROM tpcds.reason;
openGauss=# END;

--以隔离级别为READ COMMITTED，读/写方式启动事务。
openGauss=# START TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
openGauss=# SELECT * FROM tpcds.reason;
openGauss=# COMMIT;
```

相关链接

COMMIT | END，**ROLLBACK**，**SET TRANSACTION**

7.13.140 TRUNCATE

功能描述

清理表数据，TRUNCATE快速地从表中删除所有行。

它和在目标表上进行无条件的DELETE有同样的效果，但由于TRUNCATE不做表扫描，因而快得多。在大表上操作效果更明显。

注意事项

- TRUNCATE TABLE在功能上与不带WHERE子句DELETE语句相同：二者均删除表中的全部行。
- TRUNCATE TABLE比DELETE速度快且使用系统和事务日志资源少：
  - DELETE语句每次删除一行，并在事务日志中为所删除每行记录一项。
  - TRUNCATE TABLE通过释放存储表数据所用数据页来删除数据，并且只在事务日志中记录页的释放。
- TRUNCATE，DELETE，DROP三者的差异如下：
  - TRUNCATE TABLE，删除内容，释放空间，但不删除定义。
  - DELETE TABLE，删除内容，不删除定义，不释放空间。
  - DROP TABLE，删除内容和定义，释放空间。

语法格式

- 清理表数据。

```
TRUNCATE [ TABLE ] [ ONLY ] {table_name [ * ]} [, ... ]
    [ CONTINUE IDENTITY ] [ CASCADE | RESTRICT ];
```

- 清理表分区的数据。

```
ALTER TABLE [ IF EXISTS  ] { [ ONLY  ] table_name
                           | table_name *
                           | ONLY ( table_name )  }
    TRUNCATE PARTITION { partition_name
                       | FOR (  partition_value  [, ...] )  } [ UPDATE GLOBAL INDEX ];
```

参数说明

- **ONLY**

  如果声明ONLY，只有指定的表会被清空。如果没有声明ONLY，这个表以及其所有子表（若有）会被清空。

- **table_name**

  目标表的名称（可以有模式修饰）。

  取值范围：已存在的表名。

- **CONTINUE IDENTITY**

  不改变序列的值。这是缺省值。

- **CASCADE | RESTRICT**
  - CASCADE：级联清空所有由于CASCADE而被添加到组中的表。
  - RESTRICT（缺省值）：如果其他表在该表上有外键引用则拒绝清空（分布式场景暂不支持）。

- **partition_name**

  目标分区表的分区名。

  取值范围：已存在的分区名。

- **partition_value**

  指定的分区键值。

  通过PARTITION FOR子句指定的这一组值，可以唯一确定一个分区。

  取值范围：需要进行删除数据分区的分区键的取值范围。

须知

使用PARTITION FOR子句时，partition_value所在的整个分区会被清空。

- **UPDATE GLOBAL INDEX**

  如果使用该参数，则会更新分区表上的所有全局索引，以确保使用全局索引可以查询出正确的数据。

  如果不使用该参数，则分区表上的所有全局索引将会失效。

示例

```
--创建SCHEMA。
openGauss=# CREATE SCHEMA tpcds;

--创建表tpcds.reason。
openGauss=# CREATE TABLE tpcds.reason
(
  r_reason_sk       integer,
  r_reason_id       character(16),
  r_reason_desc     character(100)
);

--向表中插入多条记录。
openGauss=# INSERT INTO tpcds.reason values(1,'AAAAAAAABAAAAAAA','reason 1'),
(5,'AAAAAAAABAAAAAAA','reason 2'),(15,'AAAAAAAABAAAAAAA','reason 3'),
(25,'AAAAAAAABAAAAAAA','reason 4'),(35,'AAAAAAAABAAAAAAA','reason 5'),
(45,'AAAAAAAACAAAAAAA','reason 6'),(55,'AAAAAAAACAAAAAAA','reason 7');

--创建表。
openGauss=# CREATE TABLE tpcds.reason_t1 AS TABLE tpcds.reason;

--清空表tpcds.reason_t1。
openGauss=# TRUNCATE TABLE tpcds.reason_t1;

--删除表。
openGauss=# DROP TABLE tpcds.reason_t1;
--创建分区表。
openGauss=# CREATE TABLE tpcds.reason_p
(
  r_reason_sk integer,
  r_reason_id character(16),
  r_reason_desc character(100)
)PARTITION BY RANGE (r_reason_sk)
(
  partition p_05_before values less than (05),
  partition p_15 values less than (15),
  partition p_25 values less than (25),
  partition p_35 values less than (35),
  partition p_45_after values less than (MAXVALUE)
);

--插入数据。
openGauss=# INSERT INTO tpcds.reason_p SELECT * FROM tpcds.reason;

--清空分区p_05_before。
openGauss=# ALTER TABLE tpcds.reason_p TRUNCATE PARTITION p_05_before;

--清空分区p_15。
openGauss=# ALTER TABLE tpcds.reason_p TRUNCATE PARTITION for (15);

--清空分区表。
openGauss=# TRUNCATE TABLE tpcds.reason_p;

--删除表。
openGauss=# DROP TABLE tpcds.reason_p;
```



```
--删除表。
openGauss=# DROP TABLE tpcds.reason;

--删除SCHEMA。
openGauss=# DROP SCHEMA tpcds CASCADE;
```

7.13.141 UPDATE

功能描述

更新表中的数据，UPDATE修改满足条件的所有行中指定的字段值，WHERE子句声明条件，SET子句指定的字段会被修改，没有出现的字段则保持它们的原值。

注意事项

- 表的所有者、拥有表UPDATE权限的用户或拥有UPDATE ANY TABLE权限的用户，有权更新表中的数据，系统管理员默认拥有此权限。
- 对expression或condition条件里涉及到的任何表要有SELECT权限。
- 不允许对表的分布列（distribute column）进行修改。

语法格式

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
UPDATE [/*+ plan_hint */] [ ONLY ] table_name [ * ] [ [ AS ] alias ]
SET {column_name = { expression | DEFAULT }
   |( column_name [, ...] ) = {( { expression | DEFAULT } [, ...] ) |sub_query }}[, ...]
   [ FROM from_list] [ WHERE condition ]
   [ RETURNING {*
            | {output_expression [ [ AS ] output_name ]} [, ...] }];

where sub_query can be:
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
{ * | {expression [ [ AS ] output_name ]} [, ...] }
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY grouping_element [, ...] ]
[ HAVING condition [, ...] ]
[ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS { FIRST |
LAST } ]} [, ...] ]
[ LIMIT { [offset,] count | ALL } ]
```

参数说明

- **WITH [RECURSIVE] with_query [, ...]**

  用于声明一个或多个可以在主查询中通过名称引用的子查询，相当于临时表。

  如果声明了RECURSIVE，那么允许SELECT子查询通过名称引用它自己。

  其中with_query的详细格式为：

  ```
  with_query_name [ ( column_name [, ...] ) ] AS [ [ NOT ] MATERIALIZED ] ( {select | values | insert |
  update | delete} )
  ```

  - with_query_name指定子查询生成的结果集名称，在查询中可使用该名称访问子查询的结果集。
  - column_name指定子查询结果集中显示的列名。
  - 每个子查询可以是SELECT、VALUES、INSERT、UPDATE或DELETE语句。
  - 用户可以使用MATERIALIZED / NOT MATERIALIZED对CTE进行修饰。

  - 如果声明为MATERIALIZED，WITH查询将被物化，生成一个子查询结果集的拷贝，在引用处直接查询该拷贝，因此WITH子查询无法和主干SELECT语句进行联合优化（如谓词下推、等价类传递等），对于此类场景可以使用NOT MATERIALIZED进行修饰，如果WITH查询语义上可以作为子查询内联执行，则可以进行上述优化。
  - 如果用户没有显式声明物化属性则遵守以下规则：如果CTE只在所属SELECT主干中被引用一次，且语义上支持内联执行，则会被改写为子查询内联执行，否则以CTE Scan的方式物化执行。

- **plan_hint子句**

  以/*+ */的形式在UPDATE关键字后，用于对UPDATE对应的语句块生成的计划进行hint调优，详细用法请参见章节**使用Plan Hint进行调优**。每条语句中只有第一个/*+ plan_hint */注释块会作为hint生效，里面可以写多条hint。

- **table_name**

  要更新的表名，可以使用模式修饰。

  取值范围：已存在的表名称。

- **alias**

  目标表的别名。

  取值范围：字符串，符合**标识符命名规范**。

- **column_name**

  要修改的字段名。

  支持使用目标表的别名加字段名来引用这个字段。例如：

  UPDATE foo AS f SET f.col_name = 'namecol';

  取值范围：已存在的字段名。

- **expression**

  赋给字段的值或表达式。

- **DEFAULT**

  用对应字段的缺省值填充该字段。

  如果没有缺省值，则为NULL。

- **sub_query**

  子查询。

  使用同一数据库里其他表的信息来更新一个表可以使用子查询的方法。其中SELECT子句具体介绍请参考**SELECT**。

  在update单列时，支持使用order by子句与limit子句；而在update多列时，则不支持使用order by子句与limit子句。

- **from_list**

  一个表的表达式列表，允许在WHERE条件里使用其他表的字段。与在一个SELECT语句的FROM子句里声明表列表类似。

须知

目标表不能出现在from_list里，除非在使用一个自连接（此时它必须以from_list的别名出现）。

- **condition**

  一个返回Boolean类型结果的表达式。只有这个表达式返回true的行才会被更新。不建议使用int等数值类型作为condition，因为int等数值类型可以隐式转换为bool值（非0值隐式转换为true，0转换为false），可能导致非预期的结果。

- **output_expression**

  在所有需要更新的行都被更新之后，UPDATE命令用于计算返回值的表达式。

  取值范围：使用任何table以及FROM中列出的表的字段。*表示返回所有字段。

- **output_name**

  字段的返回名称。

示例

```
--创建表student1。
openGauss=# CREATE TABLE student1
(
  stuno     int,
  classno   int
)
DISTRIBUTE BY hash(stuno);

--插入数据。
openGauss=# INSERT INTO student1 VALUES(1,1);
openGauss=# INSERT INTO student1 VALUES(2,2);
openGauss=# INSERT INTO student1 VALUES(3,3);

--查看数据。
openGauss=# SELECT * FROM student1;

--直接更新所有记录的值。
openGauss=# UPDATE student1 SET classno = classno*2;

--查看数据。
openGauss=# SELECT * FROM student1;

--删除表。
openGauss=# DROP TABLE student1;
```

7.13.142 VACUUM

功能描述

VACUUM回收表或B-Tree索引中已经删除的行所占据的存储空间。在一般的数据库操作里，那些已经DELETE的行并没有从它们所属的表中物理删除；在完成VACUUM之前它们仍然存在。因此有必要周期地运行VACUUM，特别是在经常更新的表上。

注意事项

- 如果没有参数，VACUUM处理当前数据库里用户拥有相应权限的每个表。如果参数指定了一个表，VACUUM只处理指定的那个表。
- 要对一个表进行VACUUM操作，通常用户必须是表的所有者或者被授予了指定表VACUUM权限的用户，默认系统管理员有该权限。数据库的所有者允许对数据库中除了共享目录以外的所有表进行VACUUM操作（该限制意味着只有系统管理员才能真正对一个数据库进行VACUUM操作）。VACUUM命令会跳过那些用户没有权限的表进行垃圾回收操作。
- VACUUM不能在事务块内执行。

- 建议生产数据库经常清理（至少每晚一次），以保证不断地删除失效的行。尤其是在增删了大量记录之后，对受影响的表执行VACUUM ANALYZE命令是一个很好的习惯。这样将更新系统目录为最近的更改，并且允许查询优化器在规划用户查询时有更优选择。
- 不建议日常使用FULL选项，但是可以在特殊情况下使用。例如在用户删除了一个表的大部分行之后，希望从物理上缩小该表以减少磁盘空间占用。VACUUM FULL通常要比单纯的VACUUM收缩更多的表尺寸。FULL选项并不清理索引，所以推荐周期性的运行**REINDEX**命令。实际上，首先删除所有索引，再运行VACUUM FULL命令，最后重建索引通常是更快的选择。如果执行此命令后所占用物理空间无变化（未减少），请确认是否有其他活跃事务（删除数据事务开始之前开始的事务，并在VACUUM FULL执行前未结束）存在，如果有等其他活跃事务退出进行重试。
- VACUUM会导致I/O流量的大幅增加，这可能会影响其他活动会话的性能。因此，有时候会建议使用基于开销的VACUUM延迟特性。
- 如果指定了VERBOSE选项，VACUUM将打印处理过程中的信息，以表明当前正在处理的表。各种有关当前表的统计信息也会打印出来。
- 当含有带括号的选项列表时，选项可以以任何顺序写入。如果没有括号，则选项必须按语法显示的顺序给出。
- VACUUM和VACUUM FULL时，会根据参数vacuum_defer_cleanup_age延迟清理行存表记录，即不会立即清理刚刚删除的元组。
- VACUUM ANALYZE先执行一个VACUUM操作，然后给每个选定的表执行一个ANALYZE。对于日常维护脚本而言，这是一个很方便的组合。
- 简单的VACUUM（不带FULL选项）只是简单地回收空间并且令其可以再次使用。这种形式的命令可以和对表的普通读写并发操作，因为没有请求排他锁。VACUUM FULL执行更广泛的处理，包括跨块移动行，以便把表压缩到最少的磁盘块数目里。这种形式要慢许多并且在处理的时候需要在表上施加一个排他锁。
- 如果没有打开xc_maintenance_mode参数，那么VACUUM FULL会跳过所有系统表。
- 执行DELETE后立即执行VACUUM FULL命令不会回收空间。执行DELETE后再执行1000个非SELECT事务，或者等待1s后再执行1个事务，之后再执行VACUUM FULL命令空间才会回收。
- 为保证性能和统计信息的准确性，避免VACUUM ANALYZE、AUTOANALYZE、手动ANALYZE等涉及ANALYZE的命令同时执行或执行过于频繁。

语法格式

- 回收空间并更新统计信息，对关键字顺序无要求。

```
VACUUM [ ( { FULL | FREEZE | VERBOSE | {ANALYZE | ANALYSE }} [,...] ) ]
    [ table_name [ (column_name [, ...] ) ] ] [ PARTITION ( partition_name ) ];
```

- 仅回收空间，不更新统计信息。

```
VACUUM [ FULL [COMPACT] ] [ FREEZE ] [ VERBOSE ] [ table_name ] [ PARTITION
( partition_name ) ];
```

- 回收空间并更新统计信息，且对关键字顺序有要求。

```
VACUUM [ FULL ] [ FREEZE ] [ VERBOSE ] { ANALYZE | ANALYSE } [ VERBOSE ]
    [ table_name [ (column_name [, ...] ) ] ] [ PARTITION ( partition_name ) ];
```

参数说明

- **FULL**

  选择“FULL”清理，这样可以恢复更多的空间，但是需要耗时更多，并且在表上施加了排他锁。

> **说明**
>
> 使用FULL参数会导致统计信息丢失，如果需要收集统计信息，请在VACUUM FULL语句中加上ANALYZE关键字。

- **FREEZE**

  指定FREEZE相当于执行VACUUM时将vacuum_freeze_min_age参数设为0。

- **VERBOSE**

  为每个表打印一份详细的清理工作报告。

- **ANALYZE | ANALYSE**

  更新用于优化器的统计信息，以决定执行查询的最有效方法。

- **table_name**

  要清理的表的名称（可以有模式修饰）。

  取值范围：要清理的表的名称。缺省时为当前数据库中的所有表。

- **column_name**

  要分析的具体的字段名称，需要配合analyze选项使用。

  取值范围：要分析的具体的字段名称。缺省时为所有字段。

  > **说明**
  >
  > 由于VACUUM ANALYZE语句的机制是依次执行VACUUM和ANALYZE，因此当column_name错误时，会存在VACUUM执行成功但ANALYZE执行失败的情况；对于分区表，则会出现对某个分区成功执行VACUUM之后但ANALYZE执行失败的情况。

- **PARTITION**

  COMPACT和PARTITION参数不能同时使用。

- **partition_name**

  要清理的表的分区名称。缺省时为所有分区。

示例

```
--创建SCHEMA。
openGauss=# CREATE SCHEMA tpcds;

--创建表tpcds.reason。
openGauss=# CREATE TABLE tpcds.reason
(
  r_reason_sk       integer,
  r_reason_id       character(16),
  r_reason_desc     character(100)
);

--向表中插入多条记录。
openGauss=# INSERT INTO tpcds.reason values(1,'AAAAAAAABAAAAAAA','reason 1'),
(2,'AAAAAAAABAAAAAAA','reason 2');

--在表tpcds.reason上创建索引。
openGauss=# CREATE UNIQUE INDEX ds_reason_index1 ON tpcds.reason(r_reason_sk);

--对带索引的表tpcds.reason执行VACUUM操作。
openGauss=# VACUUM (VERBOSE, ANALYZE) tpcds.reason;

--删除索引。
openGauss=# DROP INDEX tpcds.ds_reason_index1 CASCADE;
openGauss=# DROP TABLE tpcds.reason;
openGauss=# DROP SCHEMA tpcds CASCADE;
```


优化建议

- vacuum
  - VACUUM不能在事务块内执行。
  - 建议生产数据库经常清理（至少每晚一次），以保证不断地删除失效的行。尤其是在增删了大量记录后，对相关表执行VACUUM ANALYZE命令。
  - 不建议日常使用FULL选项，但是可以在特殊情况下使用。例如，一个例子就是在用户删除了一个表的大部分行之后，希望从物理上缩小该表以减少磁盘空间占用。
  - 执行VACUUM FULL操作时，建议首先删除相关表上的所有索引，再运行VACUUM FULL命令，最后重建索引。

7.13.143 VALUES

功能描述

根据给定的值表达式计算一个或一组行的值。它通常用于在一个较大的命令内生成一个“常数表”。

注意事项

- 应当避免使用VALUES返回数量非常大的结果行，否则可能会出现内存耗尽或者性能低下的情况。出现在INSERT中的VALUES是一个特殊情况，因为目标字段类型可以从INSERT的目标表获知，并不需要通过扫描VALUES列表来推测，所以在此情况下可以处理非常大的结果行。
- 如果指定了多行，那么每一行都必须拥有相同的元素个数。

语法格式

```
VALUES {( expression [, ...] )} [, ...]
    [ ORDER BY { sort_expression [ ASC | DESC | USING operator ] } [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start [ ROW | ROWS ] ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ];
```

参数说明

- **expression**

  用于计算或插入结果表指定地点的常量或者表达式。

  在一个出现在INSERT顶层的VALUES列表中，expression可以被DEFAULT替换以表示插入目的字段的缺省值。除此以外，当VALUES出现在其他场合的时候是不能使用DEFAULT的。

- **sort_expression**

  一个表示如何排序结果行的表达式或者整数常量。

- **ASC**

  指定按照升序排列。

- **DESC**

  指定按照降序排列。

- **operator**

  一个排序操作符。

- **count**
  返回的最大行数。
- **OFFSET start [ROW | ROWS]**
  声明返回的最大行数，而start声明开始返回行之前忽略的行数。
- **FETCH { FIRST | NEXT } [count] { ROW | ROWS } ONLY**
  FETCH子句限定返回查询结果从第一行开始的总行数，count的缺省值为1。

示例

请参见INSERT的**示例**。

7.14 附录

7.14.1 扩展函数

下表列举了GaussDB中支持的扩展函数，不作为商用特性交付，仅供参考。

分类	函数名称	描述
触发器函数	pg_get_triggerdef(trigger_oid)	为触发器获取CREATE [CONSTRAINT] TRIGGER命令
	pg_get_triggerdef(trigger_oid, pretty_bool)	为触发器获取CREATE [CONSTRAINT] TRIGGER命令

8 最佳实践

8.1 表设计最佳实践

8.1.1 选择存储模型

进行数据库设计时，表设计上的一些关键项将严重影响后续整库的查询性能。表设计对数据存储也有影响：好的表设计能够减少I/O操作及最小化内存使用，进而提升查询性能。

表的存储模型选择是表定义的第一步。客户业务属性是表的存储模型的决定性因素，依据下面表格选择适合当前业务的存储模型。

存储模型	适用场景
行存	点查询（返回记录少，基于索引的简单查询）。 增删改比较多的场景。

8.1.2 选择分布方式

复制表（Replication）方式将表中的全量数据在集群的每一个DN实例上保留一份。主要适用于记录集较小的表。这种存储方式的优点是每个DN上都有该表的全量数据，在join操作中可以避免数据重分布操作，从而减小网络开销，同时减少了plan segment(每个plan segment都会起对应的线程)；缺点是每个DN都保留了表的完整数据，造成数据的冗余。一般情况下只有较小的维度表才会定义为Replication表。

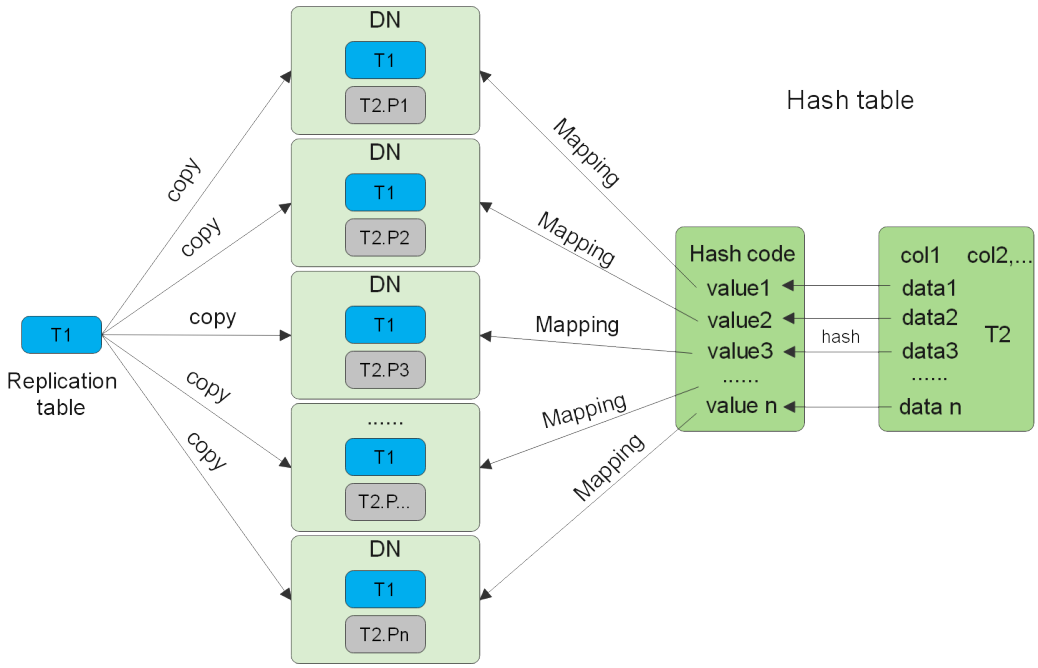
哈希（Hash）表将表中某一个或几个字段进行hash运算后，生成对应的hash值，根据DN实例与哈希值的映射关系获得该元组的目标存储位置。对于Hash分布表，在读/写数据时可以利用各个节点的I/O资源，大大提升表的读/写速度。一般情况下大表定义为Hash表。

范围（Range）和列表（List）分布是由用户自定义的分布策略，根据分布列的取值落入满足一定范围或者具体值的对应目标DN，这两种分布方式便于用户灵活地进行数据管理，但对用户本身的数据抽象能力有一定的要求。

策略	描述	适用场景
Hash	表数据通过hash方式散列到集群中的所有DN实例上。	数据量较大的事实表。
Replication	集群中每一个DN实例上都有一份全量表数据。	小表、维度表。
Range	表数据对指定列按照范围进行映射，分布到对应DN。	用户需要自定义分布规则的场景。
List	表数据对指定列按照具体值进行映射，分布到对应DN。	用户需要自定义分布规则的场景。

如图8-1所示，复制表如图中的表T1，哈希表如图中的表T2。

图 8-1 复制表和哈希表



8.1.3 选择分布列

Hash分布表的分布列选取至关重要，需要满足以下原则：

1. **列值应比较离散，以便数据能够均匀分布到各个DN**。例如，考虑选择表的主键为分布列，如在人员信息表中选择身份证号码为分布列。
2. **在满足上述条件的情况下，考虑选择查询中的连接条件为分布列**，以便Join任务能够下推到DN中执行，且减少DN之间的通信数据量。

对于Hash分表策略，如果分布列选择不当，可能导致数据倾斜，查询时出现部分DN的I/O短板，从而影响整体查询性能。因此在采用Hash分表策略之后需对表的数据进行数据倾斜性检查，以确保数据在各个DN上是均匀分布的。可以使用以下SQL检查数据倾斜性。

```
select
xc_node_id, count(1)
from tablename
group by xc_node_id
order by xc_node_id desc;
```

示例如下：

```
CREATE TABLE t1(c1 int) distribute by hash(c1);
INSERT INTO t1 values(generate_series(1,100));
select xc_node_id, count(1) from t1 group by xc_node_id order by xc_node_id desc;
DROP TABLE t1;
```

其中xc_node_id对应DN，一般来说，**不同DN的数据量相差5%以上即可视为倾斜，如果相差10%以上就必须要调整分布列**。

GaussDB支持多分布列特性，可以更好地满足数据分布的均匀性要求。

Range/List分布表的分布列由用户根据实际需要进行选择。除了需选择合适的分布列，还需要注意分布规则对数据分布的影响。

8.1.4 使用分区表

分区表是把逻辑上的一张表根据某种方案分成几张物理块进行存储。这张逻辑上的表称之为分区表，物理块称之为分区。分区表是一张逻辑表，不存储数据，数据实际是存储在分区上的。分区表和普通表相比具有以下优点：

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

GaussDB支持的分区表为范围分区表。

范围分区表：将数据基于范围映射到每一个分区。这个范围是由创建分区表时指定的分区键决定的。分区键经常采用日期，例如将销售数据按照月份进行分区。

8.1.5 选择数据类型

高效数据类型，主要包括以下三方面：

1. **尽量使用执行效率比较高的数据类型**

   一般来说整型数据运算(包括=、＞、＜、≧、≦、≠等常规的比较运算，以及group by)的效率比字符串、浮点数要高。

2. **尽量使用短字段的数据类型**

   长度较短的数据类型不仅可以减小数据文件的大小，提升I/O性能；同时也可以减小相关计算时的内存消耗，提升计算性能。比如对于整型数据，如果可以用smallint就尽量不用int，如果可以用int就尽量不用bigint。

3. **使用一致的数据类型**

   表关联列尽量使用相同的数据类型。如果表关联列数据类型不同，数据库必须动态地转化为相同的数据类型进行比较，这种转换会带来一定的性能开销。

8.1.6 查看表所在节点

用户在建表时可以指定表如何在节点之间分布或者复制，详情请参考**•DISTRIBUTEBY**，分布方式介绍可参阅**选择分布方式**。

用户在建表时也可设置“Node Group”来指定表所在的Group，详情请参考**•TO{GROUPgroupname|...**。

用户还可以通过以下命令查看表所在实例。

1. 查询表所在的schema。
   ```
   select t1.nspname,t2.relname from pg_namespace t1,pg_class t2 where t1.oid = t2.relnamespace and t2.relname = 'table1';
   ```
   上述命令中，“nspname”为schema的名称，“relname”为表、索引、视图等对象的名称，“oid”为行标识符，“relnamespace”为包含这个关系的名称空间的OID，“table1”为表名称。
2. 查看表的relname和nodeoids。
   ```
   select t1.relname,t2.nodeoids from pg_class t1, pgxc_class t2, pg_namespace t3  where t1.relfilenode =  t2.pcrelid and t1.relnamespace=t3.oid and t1.relname = 'table1' and t3.nspname ='schema1';
   ```
   上述命令中，“nodeoids”为表分布的节点OID列表，“relfilenode”为这个关系在磁盘上的文件的名称，“pcrelid”为表的OID，“schema1”为**1**中查询出的该表所在schema。
3. 根据查询到的表分布的节点，查询表所在实例。
   ```
   select * from pgxc_node where oid in (nodeoids1, nodeoids2, nodeoids3);
   ```
   上述命令中的“nodeoids1, nodeoids2, nodeoids3”为**2**中查询到的3个nodeoids，操作时以实际查询到的为准，各nodeoids间以“,”隔开。

8.2 SQL 查询最佳实践

根据数据库的SQL执行机制以及大量的实践总结发现：通过一定的规则调整SQL语句，在保证结果正确的基础上，能够提高SQL执行效率。

- **使用union all代替union**

  union在合并两个集合时会执行去重操作，而union all则直接将两个结果集合并、不执行去重。执行去重会消耗大量的时间，因此，在一些实际应用场景中，如果通过业务逻辑已确认两个集合不存在重叠，可用union all替代union以便提升性能。

- **join列增加非空过滤条件**

  若join列上的NULL值较多，则可以加上is not null过滤条件，以实现数据的提前过滤，提高join效率。

- **not in转not exists**

  NOT IN语句需要使用NESTLOOP ANTI JOIN来实现，而NOT EXISTS则可以通过HASH ANTI JOIN来实现。在join列不存在NULL值的情况下，NOT EXISTS和NOT IN等价。因此在没有NULL值时，可以通过将NOT IN转换为NOT EXISTS，通过生成HASH JOIN来提升查询效率。

  建表语句如下：

  ```
  DROP SCHEMA IF EXISTS no_in_to_no_exists_test CASCADE;
  CREATE SCHEMA no_in_to_no_exists_test;
  SET CURRENT_SCHEMA=no_in_to_no_exists_test;
  CREATE TABLE t1(c1 int, c2 int, c3 int);
  CREATE TABLE t2(d1 int, d2 int NOT NULL, d3 int);
  ```

  使用NOT IN实现查询语句如下：

  ```
  SELECT * FROM t1 WHERE  c1 NOT IN (SELECT d2 FROM t2);
  ```

  其计划如下所示：

  ```
  openGauss=# EXPLAIN SELECT * FROM t1 WHERE  c1 NOT IN (SELECT d2 FROM t2);
                              QUERY PLAN
  ```



```
-------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=0.06..38.57 rows=3 width=12)
   Node/s: All datanodes
   ->  Nested Loop Anti Join  (cost=0.00..38.44 rows=3 width=12)
         Join Filter: ((t1.c1 = t2.d2) OR (t1.c1 IS NULL))
         ->  Seq Scan on t1  (cost=0.00..14.14 rows=30 width=12)
         ->  Materialize  (cost=0.00..18.08 rows=90 width=4)
               ->  Streaming(type: BROADCAST)  (cost=0.00..17.93 rows=90 width=4)
                     Spawn on: All datanodes
                     ->  Seq Scan on t2  (cost=0.00..14.14 rows=30 width=4)
(9 rows)
```

因为t2.d2字段中没有NULL值（t2.d2字段在表定义中为NOT NULL），所以查询可以等价修改如下：

```
SELECT * FROM t1 WHERE  NOT EXISTS (SELECT * FROM t2 WHERE t1.c1=t2.d2);
```

其生成的计划如下：

```
openGauss=# EXPLAIN SELECT * FROM t1 WHERE  NOT EXISTS (SELECT * FROM t2 WHERE
t1.c1=t2.d2);
                              QUERY PLAN
-------------------------------------------------------------------------------
 Streaming (type: GATHER)  (cost=14.38..29.99 rows=3 width=12)
   Node/s: All datanodes
   ->  Hash Right Anti Join  (cost=14.32..29.86 rows=3 width=12)
         Hash Cond: (t2.d2 = t1.c1)
         ->  Streaming(type: REDISTRIBUTE)  (cost=0.00..15.49 rows=30 width=4)
               Spawn on: All datanodes
               ->  Seq Scan on t2  (cost=0.00..14.14 rows=30 width=4)
         ->  Hash  (cost=14.14..14.14 rows=29 width=12)
               ->  Seq Scan on t1  (cost=0.00..14.14 rows=30 width=12)
(9 rows)
```

- **选择hashagg。**

  查询语句中如果存在GROUP BY条件则生成的计划（Plan）中可能存在排序操作，即计划中包含GroupAgg+Sort算子，导致性能较差。可以通过设置GUC参数work_mem增大可用内存，生成带有HashAgg的计划（Plan）避免排序操作从而提升性能。work_mem设置请联系管理员。

- **尝试将函数替换为case语句。**

  GaussDB函数调用性能较低，如果出现过多的函数调用导致性能下降很多，可以根据情况把可下推函数的函数改成CASE表达式。

- **避免对索引使用函数或表达式运算。**

  对索引使用函数或表达式运算会停止使用索引转而执行全表扫描。

- **尽量避免在where子句中使用!=或<>操作符、null值判断、or连接、参数隐式转换。**

- **对复杂SQL语句进行拆分。**

  对于过于复杂并且不易通过以上方法调整性能的SQL可以考虑拆分的方法，把SQL中某一部分拆分成独立的SQL并把执行结果存入临时表，拆分常见的场景包括但不限于：

  - 作业中多个SQL有同样的子查询，并且子查询数据量较大。
  - Plan cost计算不准，导致子查询hash bucket太小，比如实际数据1000W行，hash bucket只有1000。
  - 函数（如substr、to_number）导致大数据量子查询选择度计算不准。
  - 多DN环境下对大表做broadcast的子查询。

其他更多调优点，请参考**典型SQL调优点**。

8.3 数据倾斜查询最佳实践

8.3.1 快速定位查询存储倾斜的表

目前提供的倾斜查询接口有函数：**table_distribution(schemaname text, tablename text)**、**table_distribution()**以及视图**PGXC_GET_TABLE_SKEWNESS**，客户可以根据自身业务情况来选择使用。

场景一：磁盘满后快速定位存储倾斜的表

首先，通过**pg_stat_get_last_data_changed_time(oid)**函数查询出近期发生过数据变更的表，鉴于表的最后修改时间只在进行IUD操作的CN记录，要查询库内1天(间隔可在函数中调整)内被修改的所有表，可以使用如下封装函数：

```
CREATE OR REPLACE FUNCTION get_last_changed_table(OUT schemaname text, OUT relname text)
RETURNS setof record
AS $$
DECLARE
   row_data record;
   row_name record;
   query_str text;
   query_str_nodes text;
   BEGIN
      query_str_nodes := 'SELECT node_name FROM pgxc_node where node_type = ''C''';
      FOR row_name IN EXECUTE(query_str_nodes) LOOP
         query_str := 'EXECUTE DIRECT ON (' || row_name.node_name || ') ''SELECT b.nspname,a.relname
FROM pg_class a INNER JOIN pg_namespace b on a.relnamespace = b.oid where
pg_stat_get_last_data_changed_time(a.oid) BETWEEN current_timestamp - 1 AND current_timestamp;''';
         FOR row_data IN EXECUTE(query_str) LOOP
            schemaname = row_data.nspname;
            relname = row_data.relname;
            return next;
         END LOOP;
      END LOOP;
      return;
   END; $$
LANGUAGE 'plpgsql';
```

然后，通过**table_distribution(schemaname text, tablename text)**查询出表在各个DN占用的存储空间。

```
SELECT table_distribution(schemaname,relname) FROM get_last_changed_table();
```

场景二：常规数据倾斜巡检

- 在库中表个数少于1W的场景，直接使用倾斜视图查询当前库内所有表的数据倾斜情况。
  ```
  SELECT * FROM pgxc_get_table_skewness ORDER BY totalsize DESC;
  ```
- 在库中表个数非常多（至少大于1W）的场景，因**PGXC_GET_TABLE_SKEWNESS**涉及全库查并计算非常全面的倾斜字段，所以可能会花费比较长的时间（小时级），建议参考**PGXC_GET_TABLE_SKEWNESS**视图定义，直接使用table_distribution()函数自定义输出，减少输出列进行计算优化，例如：
  ```
  SELECT schemaname,tablename,max(dnsize) AS maxsize, min(dnsize) AS minsize
  FROM pg_catalog.pg_class c
  INNER JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
  INNER JOIN pg_catalog.table_distribution() s ON s.schemaname = n.nspname AND s.tablename =
  c.relname
  ```



```
INNER JOIN pg_catalog.pgxc_class x ON c.oid = x.pcrelid AND x.pclocatortype = 'H'
GROUP BY schemaname,tablename;
```


9 用户自定义函数

集群启动时，除了启动CN、DN之外，还会启动UDF master进程。当需要执行fenced模式的UDF时，UDF master进程会fork出UDF worker进程，UDF worker进程执行fenced模式的UDF。

9.1 PL/SQL 语言函数

PL/SQL是一种可载入的过程语言。

用PL/SQL创建的函数可以被用在任何可以使用内建函数的地方。例如，可以创建复杂条件的计算函数并且后面用它们来定义操作符或把它们用于索引表达式。

SQL被大多数数据库用作查询语言。它是可移植的并且容易学习。但是每一个SQL语句必须由数据库服务器单独执行。

这意味着客户端应用对于每一个查询都要执行以下过程：发送查询到数据库服务器、等待查询被接收、接收并处理结果、进行相关计算、然后发送更多查询给服务器。如果客户端和数据库服务器不在同一台机器上，那么这个过程还会引起进程间通信并且将带来网络负担。

通过PL/SQL，可以将一整块计算和一系列查询分组在数据库服务器内部，这样就有了一种过程语言的能力并且使SQL更易用，同时能节省的客户端/服务器通信开销。

- 客户端和服务器之间的额外往返通信被消除。
- 客户端不需要的中间结果不必被整理或者在服务器和客户端之间传送。
- 多轮的查询解析可以被避免。

PL/SQL可以使用SQL中所有的数据类型、操作符和函数。应用PL/SQL创建函数的语法为**CREATE FUNCTION**。

PL/SQL是一种可载入的过程语言。其应用方法与**存储过程**相似，但存储过程无返回值，PL/SQL语言函数有返回值。

10 存储过程

10.1 存储过程

商业规则和业务逻辑可以通过程序存储在GaussDB中，这个程序就是存储过程。

存储过程是SQL、PL/SQL、Java语句的组合。存储过程使执行商业规则的代码可以从应用程序中移动到数据库。从而，代码存储一次能够被多个程序使用。

存储过程的创建及调用办法请参考**CREATE PROCEDURE**。

PL/SQL语言函数节所提到的PL/SQL语言创建的函数与存储过程的应用方法相同。下面各节中，除非特别声明，否则内容通用于存储过程和PL/SQL语言函数。

10.2 数据类型

数据类型是一组值的集合以及定义在这个值集上的一组操作。GaussDB数据库是由表的集合组成的，而各表中的列定义了该表，每一列都属于一种数据类型，GaussDB根据数据类型有相应函数对其内容进行操作，例如GaussDB可对数值型数据进行加、减、乘、除操作。

10.3 数据类型转换

数据库中允许有些数据类型进行隐式类型转换（赋值、函数调用的参数等），有些数据类型间不允许进行隐式数据类型转换，可尝试使用GaussDB提供的类型转换函数，例如CAST进行数据类型强转。

GaussDB数据库常见的隐式类型转换，请参见**表10-1**。

须知

GaussDB支持的DATE的效限范围是：公元前4713年到公元294276年。

表 10-1 隐式类型转换表

原始数据类型	目标数据类型	备注
CHAR	VARCHAR2	-
CHAR	NUMBER	原数据必须由数字组成。
CHAR	DATE	原数据不能超出合法日期范围。
CHAR	RAW	-
CHAR	CLOB	-
VARCHAR2	CHAR	-
VARCHAR2	NUMBER	原数据必须由数字组成。
VARCHAR2	DATE	原数据不能超出合法日期范围。
VARCHAR2	CLOB	-
NUMBER	CHAR	-
NUMBER	VARCHAR2	-
DATE	CHAR	-
DATE	VARCHAR2	-
RAW	CHAR	-
RAW	VARCHAR2	-
CLOB	CHAR	-
CLOB	VARCHAR2	-
CLOB	NUMBER	原数据必须由数字组成。
INT4	CHAR	-

10.4 数组和 record

10.4.1 数组

数组类型的使用

在使用数组之前，需要自定义一个数组类型。

在存储过程中紧跟AS关键字后面定义数组类型。定义方法为：

```
TYPE array_type IS VARRAY(size) OF data_type;
```

其中：

- array_type：要定义的数组类型名。
- VARRAY：表示要定义的数组类型。
- size：取值为正整数，表示可以容纳的成员的最大数量。
- data_type：要创建的数组中成员的类型。

说明

- 在GaussDB中，数组会自动增长，访问越界会返回一个NULL，不会报错。
- 在存储过程中定义的数组类型，其作用域仅在该存储过程中。
- 建议选择上述定义方法的一种来自定义数组类型，当同时使用两种方法定义同名的数组类型时，GaussDB会优先选择存储过程中定义的数组类型来声明数组变量。
- 当data_type为varchar、numeric等可以定义长度和精度的类型，如果package内创建varray类型数组，则在package外部调用该varray类型数组时，varchar、numeric等长度或精度限制会失效。
- array类型的构造器仅支持在O兼容模式下使用。

GaussDB支持使用圆括号来访问数组元素，且还支持一些特有的函数，如extend，count，first，last, prior, next, exists, trim, delete来访问数组的内容。

说明

存储过程中如果有DML语句（SELECT、UPDATE、INSERT、DELETE），DML语句只能使用中括号来访问数组元素，这样可以和函数表达式区分开。

示例

```
--演示在存储过程中对数组进行操作。
openGauss=# CREATE OR REPLACE PROCEDURE array_proc AS
DECLARE
    TYPE ARRAY_INTEGER IS VARRAY(1024) OF INTEGER;--定义数组类型
    ARRINT ARRAY_INTEGER := ARRAY_INTEGER();  --声明数组类型的变量
BEGIN
    ARRINT.EXTEND(10);
    FOR I IN 1..10 LOOP
        ARRINT(I) := I;
    END LOOP;
    DBE_OUTPUT.PRINT_LINE(ARRINT.COUNT);
    DBE_OUTPUT.PRINT_LINE(ARRINT(1));
    DBE_OUTPUT.PRINT_LINE(ARRINT(10));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.FIRST));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.LAST));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.NEXT(ARRINT.FIRST)));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.PRIOR(ARRINT.LAST)));
    ARRINT.TRIM();
    IF ARRINT.EXISTS(10) THEN
       DBE_OUTPUT.PRINT_LINE('Exist 10th element');
    ELSE
       DBE_OUTPUT.PRINT_LINE('Not exist 10th element');
    END IF;
    DBE_OUTPUT.PRINT_LINE(ARRINT.COUNT);
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.FIRST));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.LAST));
    ARRINT.DELETE();
END;
/

--调用该存储过程。
openGauss=# CALL array_proc();

--删除存储过程。
openGauss=# DROP PROCEDURE array_proc;
```


10.4.2 record

record 类型的变量

创建一个record变量的方式：

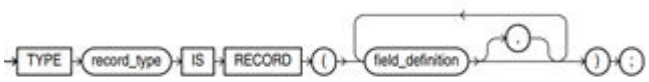
定义一个record类型，然后使用该类型来声明一个变量。

语法

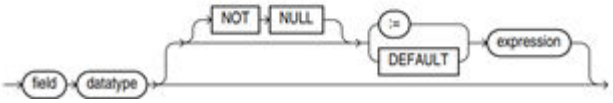
record类型的语法参见图10-1。

图 10-1 record 类型的语法

record_type_definition ::=



field_definition ::=



对以上语法格式的解释如下：

- record_type：声明的类型名称。
- field：record类型中的成员名称。
- datatype：record类型中成员的类型。
- expression：设置默认值的表达式。

说明

在GaussDB中：

- record类型的变量的赋值支持：
  - 在函数或存储过程的声明阶段，声明一个record类型，并且可以在该类型中定义成员变量。
  - 一个record变量到另一个record变量的赋值。
  - SELECT INTO和FETCH向一个record类型的变量中赋值。
  - 将一个NULL值赋值给一个record变量。
- 不支持INSERT和UPDATE语句使用record变量进行插入数据和更新数据。
- 如果成员有record类型，则内层record类型的默认值不支持传递至外层record类型中。
- 如果成员有复合类型，在声明阶段不支持指定默认值，该行为同声明阶段的变量一样。

示例

```
下面存储过程中用到的表定义如下：
openGauss=# \d emp_rec
              Table "public.emp_rec"
```



```
 Column  |              Type              | Modifiers
---------+--------------------------------+-----------
 empno   | numeric(4,0)                   | not null
 ename   | character varying(10)          |
 job     | character varying(9)           |
 mgr     | numeric(4,0)                   |
 hiredate | timestamp(0) without time zone |
 sal     | numeric(7,2)                   |
 comm    | numeric(7,2)                   |
 deptno  | numeric(2,0)                   |

--演示在存储过程中对数组进行操作。
openGauss=# CREATE OR REPLACE FUNCTION regress_record(p_w VARCHAR2)
RETURNS
VARCHAR2  AS $$
DECLARE

  --声明一个record类型.
  type rec_type is record (name  varchar2(100), epno int);
  employer rec_type;

  --使用%type声明record类型
  type rec_type1 is record (name  emp_rec.ename%type, epno int not null :=10);
  employer1 rec_type1;

  --声明带有默认值的record类型
  type rec_type2 is record (
       name varchar2 not null := 'SCOTT',
       epno int not null :=10);
   employer2 rec_type2;
   CURSOR C1 IS  select ename,empno from emp_rec order by 1 limit 1;

BEGIN
     --对一个record类型的变量的成员赋值。
    employer.name := 'WARD';
    employer.epno = 18;
    raise info 'employer name: % , epno:%', employer.name, employer.epno;

    --将一个record类型的变量赋值给另一个变量。
    employer1 := employer;
    raise info 'employer1 name: % , epno: %',employer1.name, employer1.epno;

     --将一个record类型变量赋值为NULL。
    employer1 := NULL;
    raise info 'employer1 name: % , epno: %',employer1.name, employer1.epno;

     --获取record变量的默认值。
    raise info 'employer2 name: % ,epno: %', employer2.name, employer2.epno;

     --在for循环中使用record变量
     for employer in select ename,empno from emp_rec order by 1  limit 1
         loop
             raise info 'employer name: % , epno: %', employer.name, employer.epno;
        end loop;

     --在select into 中使用record变量。
     select ename,empno  into employer2 from emp_rec order by 1 limit 1;
     raise info 'employer name: % , epno: %', employer2.name, employer2.epno;

     --在cursor中使用record变量。
     OPEN C1;
     FETCH C1 INTO employer2;
     raise info 'employer name: % , epno: %', employer2.name, employer2.epno;
     CLOSE C1;
     RETURN employer.name;
END;
$$
LANGUAGE plpgsql;
```



```
--调用该存储过程。
openGauss=# CALL regress_record('abc');

--删除存储过程。
openGauss=# DROP PROCEDURE regress_record;
```

10.5 声明语法

10.5.1 基本结构

结构

PL/SQL块中可以包含子块，子块可以位于PL/SQL中任何部分。PL/SQL块的结构如下：

- 声明部分：声明PL/SQL用到的变量，类型及游标，以及局部的存储过程和函数。
  ```
  DECLARE
  ```

  说明

  不涉及变量声明时声明部分可以没有。
  - 对匿名块来说，没有变量声明部分时，可以省去DECLARE关键字。
  - 对存储过程来说，没有DECLARE， AS相当于DECLARE。即便没有变量声明的部分，关键字AS也必须保留。
- 执行部分：过程及SQL语句，程序的主要部分。必选。
  ```
  BEGIN
  ```
- 执行异常部分：错误处理。可选。
  ```
  EXCEPTION
  ```
- 结束。必选。
  ```
  END;
  /
  ```

须知

禁止在PL/SQL块中使用连续的Tab，连续的Tab可能会造成在使用gsql工具带“-r”参数执行PL/SQL块时出现异常。

分类

PL/SQL块可以分为以下几类：

- 匿名块：动态构造，只能执行一次。语法请参考图10-2。
- 子程序：存储在数据库中的存储过程、函数和操作符及高级包等。当在数据库上建立好后，可以在其他程序中调用它们。

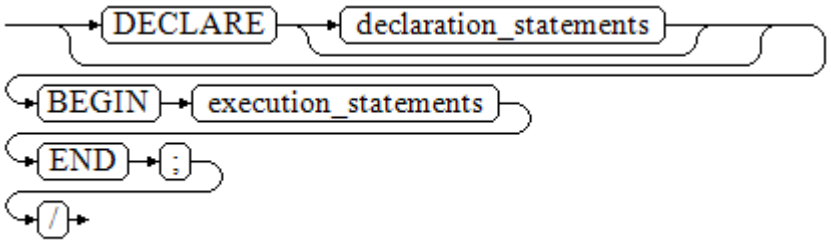
10.5.2 匿名块

匿名块（Anonymous Block）一般用于不频繁执行的脚本或不重复进行的活动。它们在一个会话中执行，并不被存储。

语法

匿名块的语法参见图10-2。

图 10-2 anonymous_block::=



对以上语法图的解释如下：

- 匿名块程序实施部分，以BEGIN语句开始，以END语句停顿，以一个分号结束。输入“/”按回车执行它。

  须知

  最后的结束符“/”必须独占一行，不能直接跟在END后面。

- 声明部分包括变量定义、类型、游标定义等。
- 最简单的匿名块不执行任何命令。但一定要在任意实施块里至少有一个语句，甚至是一个NULL语句。

示例

下面列举了基本的匿名块程序：

```
--空语句块
openGauss=# BEGIN
    NULL;
END;
/

--将信息打印到控制台：
openGauss=# BEGIN
    dbe_output.print_line('hello world!');
END;
/

--将变量内容打印到控制台：
openGauss=# DECLARE
    my_var VARCHAR2(30);
BEGIN
    my_var :='world';
    dbe_output.print_line('hello'||my_var);
END;
/
```

10.5.3 子程序

存储在数据库中的存储过程、函数和操作符及高级包等。当在数据库上建立好后，可以在其他程序中调用它们。

10.6 基本语句

在编写PL/SQL过程中，会定义一些变量，给变量赋值，调用其他存储过程等。介绍PL/SQL中的基本语句，包括定义变量、赋值语句、调用语句以及返回语句。

说明

尽量不要在存储过程中调用包含密码的SQL语句，因为存储在数据库中的存储过程文本可能被其他有权限的用户看到导致密码信息被泄漏。如果存储过程中包含其他敏感信息也需要配置存储过程的访问权限，保证敏感信息不会泄漏。

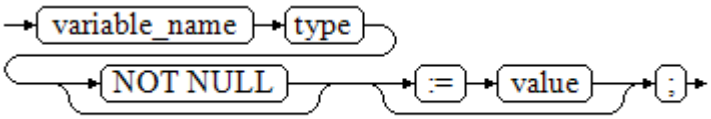
10.6.1 定义变量

介绍PL/SQL中变量的声明，以及该变量在代码中的作用域。

变量声明

变量声明语法请参见图10-3。

图 10-3 declare_variable::=



对以上语法格式的解释如下：

- variable_name，为变量名。
- type，为变量类型。
- value，是该变量的初始值（如果不给定初始值，则初始为NULL）。value也可以是表达式。

示例

```
openGauss=# DECLARE
    emp_id  INTEGER := 7788;  --定义变量并赋值
BEGIN
    emp_id := 5*7784;  --变量赋值
END;
/
```

变量类型除了支持基本类型，还可使用%TYPE和%ROWTYPE去声明一些与其他表字段或表结构本身相关的变量。

%TYPE属性

%TYPE主要用于声明某个与其他变量类型（例如，表中某列的类型）相同的变量。假如定义一个my_name变量，它的变量类型与employee的firstname类型相同，可以通过如下定义：

```
my_name employee.firstname%TYPE
```

这样定义可以带来两个好处，首先，不用预先知道employee表的firstname类型具体是什么。其次，即使之后firstname类型有了变化，也不需要再次修改my_name的类型。

%ROWTYPE属性

%ROWTYPE属性主要用于对一组数据的类型声明，用于存储表中的一行数据，或从游标匹配的结果。假如，需要一组数据，该组数据的字段名称与字段类型都与employee表相同。可以通过如下定义：

```
my_employee employee%ROWTYPE
```

说明

多个CN的环境下，存储过程中无法声明临时表的%ROWTYPE及%TYPE属性。因为临时表仅在当前session有效，在编译阶段其他CN无法看到当前CN的临时表。故多个CN的环境下，会提示该临时表不存在。

变量作用域

变量的作用域表示变量在代码块中的可访问性和可用性。只有在它的作用域内，变量才有效。

- 变量必须在declare部分声明，即必须建立BEGIN-END块。块结构也强制变量必须先声明后使用，即变量在过程内有不同作用域、不同的生存期。
- 同一变量可以在不同的作用域内定义多次，内层的定义会覆盖外层的定义。
- 在外部块定义的变量，可以在嵌套块中使用。但外部块不能访问嵌套块中的变量。

示例

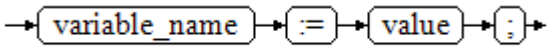
```
openGauss=# DECLARE
    emp_id  INTEGER :=7788;  --定义变量并赋值
    outer_var  INTEGER :=6688;  --定义变量并赋值
BEGIN
    DECLARE
        emp_id INTEGER :=7799;  --定义变量并赋值
        inner_var  INTEGER :=6688;  --定义变量并赋值
    BEGIN
        dbe_output.print_line('inner emp_id ='||emp_id);  --显示值为7799
        dbe_output.print_line('outer_var ='||outer_var);  --引用外部块的变量
    END;
    dbe_output.print_line('outer emp_id ='||emp_id);  --显示值为7788
END;
/
```

10.6.2 赋值语句

变量语法

给变量赋值的语法请参见图10-4。

图 10-4 assignment_value::=



对以上语法格式的解释如下：

- variable_name，为变量名。
- value，可以是值或表达式。值value的类型需要和变量variable_name的类型兼容才能正确赋值。

变量赋值示例

```
openGauss=# DECLARE
    emp_id  INTEGER := 7788;--赋值
BEGIN
    emp_id := 5;--赋值
    emp_id := 5*7784;
END;
/
```

INTO/BULK COLLECT INTO

将存储过程内语句返回的值存储到变量内，BULK COLLECT INTO允许将部分或全部返回值暂存到数组内部。

示例

```
openGauss=# DROP TABLE IF EXISTS customers;
openGauss=# CREATE TABLE customers(id int,name varchar);
openGauss=# INSERT INTO customers VALUES(1,'ab');
openGauss=# DECLARE
    my_id integer;
BEGIN
    select id into my_id from customers limit 1; -- 赋值
END;
/

openGauss=# DECLARE
    type id_list is varray(6) of customers.id%type;
    id_arr id_list;
BEGIN
    select id bulk collect into id_arr from customers order by id DESC limit 20; -- 批量赋值
END;
/
```

须知

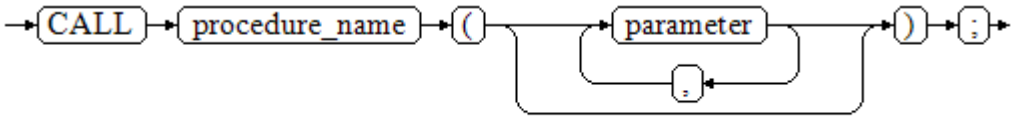
BULK COLLECT INTO 只支持批量赋值给数组。合理使用LIMIT字段避免操作过量数据导致性能下降。

10.6.3 调用语句

语法

调用一个语句的语法请参见图10-5。

图 10-5 call_clause::=



对以上语法格式的解释如下：

- procedure_name，为存储过程名。

- parameter，为存储过程的参数，可以没有或者有多个参数。

示例

```
--建表
openGauss=# CREATE SCHEMA hr;
openGauss=# SET CURRENT_SCHEMA = hr;
openGauss=# CREATE TABLE staffs
(
  section_id INTEGER,
  salary INTEGER
);
openGauss=# INSERT INTO staffs VALUES (30, 10);
openGauss=# INSERT INTO staffs VALUES (30, 20);

--创建存储过程proc_staffs
openGauss=# CREATE OR REPLACE PROCEDURE proc_staffs
(
section     NUMBER(6),
salary_sum out NUMBER(8,2),
staffs_count out INTEGER
)
IS
BEGIN
SELECT sum(salary), count(*) INTO salary_sum, staffs_count FROM hr.staffs where section_id = section;
END;
/

--创建存储过程proc_return.
openGauss=# CREATE OR REPLACE PROCEDURE proc_return
AS
v_num NUMBER(8,2);
v_sum INTEGER;
BEGIN
proc_staffs(30, v_sum, v_num);  --调用语句
dbe_output.print_line(v_sum||'#'||v_num);
RETURN;  --返回语句
END;
/

--调用存储过程proc_return.
openGauss=# CALL proc_return();

--清除存储过程
openGauss=# DROP PROCEDURE proc_staffs;
openGauss=# DROP PROCEDURE proc_return;

--创建函数func_return.
openGauss=# CREATE OR REPLACE FUNCTION func_return returns void
language plpgsql
AS $$
DECLARE
v_num INTEGER := 1;
BEGIN
dbe_output.print_line(v_num);
RETURN;  --返回语句
END $$;


-- 调用函数func_return
openGauss=#  CALL func_return();

-- 清除函数
openGauss=#  DROP FUNCTION func_return;

-- 清除当前数据库模式
openGauss=#  DROP SCHEMA hr CASCADE;
```


10.7 动态语句

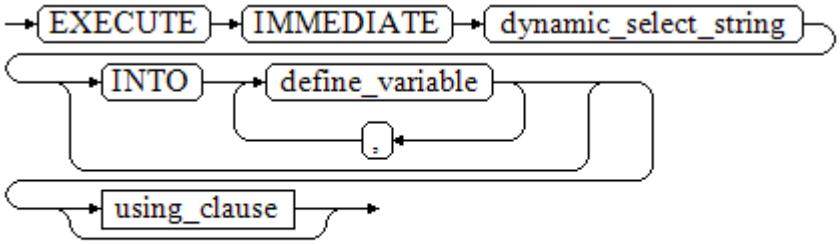
10.7.1 执行动态查询语句

介绍执行动态查询语句。GaussDB提供两种方式：使用EXECUTE IMMEDIATE、OPEN FOR实现动态查询。前者通过动态执行SELECT语句，后者结合了游标的使用。当需要将查询的结果保存在一个数据集用于提取时，可使用OPEN FOR实现动态查询。

EXECUTE IMMEDIATE

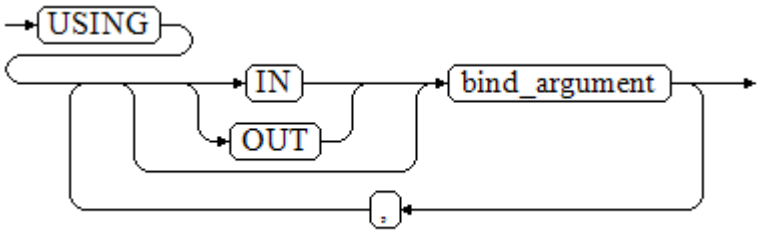
语法图请参见图10-6。

图 10-6 EXECUTE IMMEDIATE dynamic_select_clause::=



using_clause子句的语法图参见图10-7。

图 10-7 using_clause::=



对以上语法格式的解释如下：

- define_variable，用于指定存放单行查询结果的变量。
- USING IN bind_argument，用于指定存放传递给动态SQL值的变量，即在dynamic_select_string中存在占位符时使用。
- USING OUT bind_argument，用于指定存放动态SQL返回值的变量。

须知

- 查询语句中，into和out不能同时存在；
- 占位符命名以“:”开始，后面可跟数字、字符或字符串，与USING子句的bind_argument一一对应；
- bind_argument只能是值、变量或表达式，不能是表名、列名、数据类型等数据库对象，即不支持使用bind_argument为动态SQL语句传递模式对象。如果存储过程需要通过声明参数传递数据库对象来构造动态SQL语句（常见于执行DDL语句时），建议采用连接运算符“||”拼接dynamic_select_clause；
- 动态PL/SQL块允许出现重复的占位符，即相同占位符只能与USING子句的一个bind_argument按位置对应。

示例

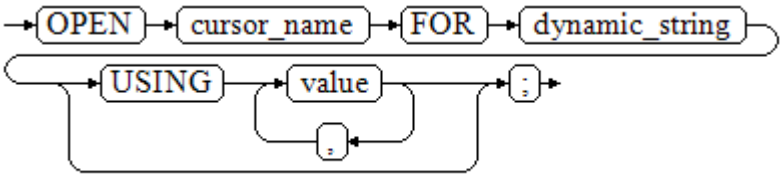
```
openGauss=# DROP SCHEMA IF EXISTS hr CASCADE;
openGauss=# CREATE SCHEMA hr;
openGauss=# SET CURRENT_SCHEMA = hr;
openGauss=# CREATE TABLE staffs
(
  staff_id NUMBER,
  first_name VARCHAR2,
  salary NUMBER
);
openGauss=# INSERT INTO staffs VALUES (200, 'mike', 5800);
openGauss=# INSERT INTO staffs VALUES (201, 'lily', 3000);
openGauss=# INSERT INTO staffs VALUES (202, 'john', 4400);

--从动态语句检索值（INTO 子句）：
openGauss=# DECLARE
   staff_count  VARCHAR2(20);
BEGIN
   EXECUTE IMMEDIATE 'select count(*) from hr.staffs'
      INTO staff_count;
   dbe_output.print_line(staff_count);
END;
/

--传递并检索值（INTO子句用在USING子句前）：
openGauss=# CREATE OR REPLACE PROCEDURE dynamic_proc
AS
   staff_id     NUMBER(6) := 200;
   first_name   VARCHAR2(20);
   salary       NUMBER(8,2);
BEGIN
   EXECUTE IMMEDIATE 'select first_name, salary from hr.staffs where staff_id = :1'
      INTO first_name, salary
      USING IN staff_id;
   dbe_output.print_line(first_name || ' ' || salary);
END;
/

--调用存储过程
openGauss=# CALL dynamic_proc();

--删除存储过程
openGauss=# DROP PROCEDURE dynamic_proc;
```

OPEN FOR

动态查询语句还可以使用OPEN FOR打开动态游标来执行。

语法参见**图10-8**。

图 10-8 open_for::=



参数说明：

- cursor_name：要打开的游标名。
- dynamic_string：动态查询语句。
- USING value：在dynamic_string中存在占位符时使用。

游标的使用请参考**游标**。

示例

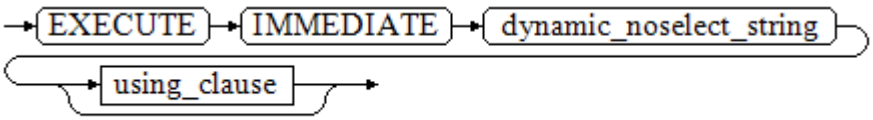
```
openGauss=# DECLARE
    name           VARCHAR2(20);
    phone_number   VARCHAR2(20);
    salary         NUMBER(8,2);
    sqlstr         VARCHAR2(1024);

    TYPE app_ref_cur_type IS REF CURSOR;  --定义游标类型
    my_cur app_ref_cur_type;  --定义游标变量

BEGIN
    sqlstr := 'select first_name,phone_number,salary from hr.staffs
        where section_id = :1';
    OPEN my_cur FOR sqlstr USING '30';  --打开游标, using是可选的
    FETCH my_cur INTO name, phone_number, salary; --获取数据
    WHILE my_cur%FOUND LOOP
        dbe_output.print_line(name||'#'||phone_number||'#'||salary);
        FETCH my_cur INTO name, phone_number, salary;
    END LOOP;
    CLOSE my_cur;   --关闭游标
END;
/

-- 清除当前数据库模式
openGauss=# DROP SCHEMA hr CASCADE;
```

10.7.2 执行动态非查询语句

语法

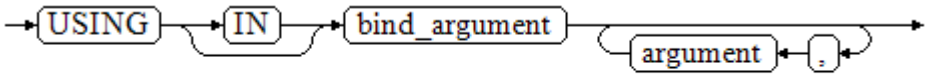
语法请参见**图10-9**。

图 10-9 noselect::=



using_clause子句的语法参见**图10-10**。

图 10-10 using_clause::=



对以上语法格式的解释如下：

USING IN bind_argument用于指定存放传递给动态SQL值的变量，在dynamic_noselect_string中存在占位符时使用，即动态SQL语句执行时，bind_argument将替换相对应的占位符。要注意的是，bind_argument只能是值、变量或表达式，不能是表名、列名、数据类型等数据库对象。如果存储过程需要通过声明参数传递数据库对象来构造动态SQL语句（常见于执行DDL语句时），建议采用连接运算符“||”拼接dynamic_select_clause。另外，动态语句允许出现重复的占位符，相同占位符只能与唯一一个bind_argument按位置一一对应。

示例

```
--创建表
openGauss=# CREATE TABLE sections_t1
(
   section       NUMBER(4) ,
   section_name  VARCHAR2(30),
   manager_id    NUMBER(6),
   place_id      NUMBER(4)
)
DISTRIBUTE BY hash(manager_id);

--声明变量
openGauss=# DECLARE
   section       NUMBER(4) := 280;
   section_name  VARCHAR2(30) := 'Info support';
   manager_id    NUMBER(6) := 103;
   place_id      NUMBER(4) := 1400;
   new_colname   VARCHAR2(10) := 'sec_name';
BEGIN
--执行查询
    EXECUTE IMMEDIATE 'insert into sections_t1 values(:1, :2, :3, :4)'
       USING section, section_name, manager_id,place_id;
--执行查询（重复占位符）
    EXECUTE IMMEDIATE 'insert into sections_t1 values(:1, :2, :3, :1)'
       USING section, section_name, manager_id;
--执行ALTER语句（建议采用“||”拼接数据库对象构造DDL语句）
    EXECUTE IMMEDIATE 'alter table sections_t1 rename section_name to ' || new_colname;
END;
/

--查询数据
openGauss=# SELECT * FROM sections_t1;

--删除表
openGauss=# DROP TABLE sections_t1;
```

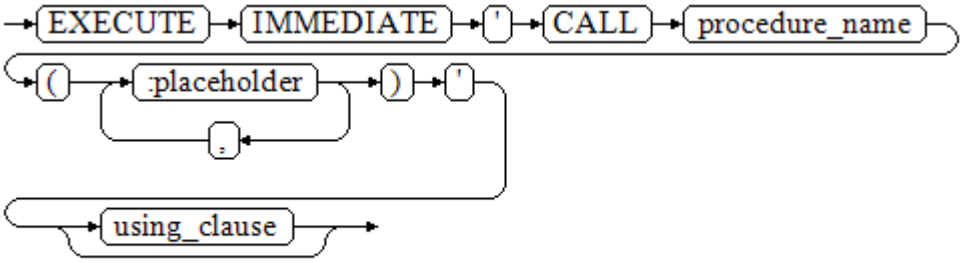
10.7.3 动态调用存储过程

动态调用存储过程必须使用匿名的语句块将存储过程或语句块包在里面，使用EXECUTE IMMEDIATE…USING语句后面带IN、OUT来输入、输出参数。

语法

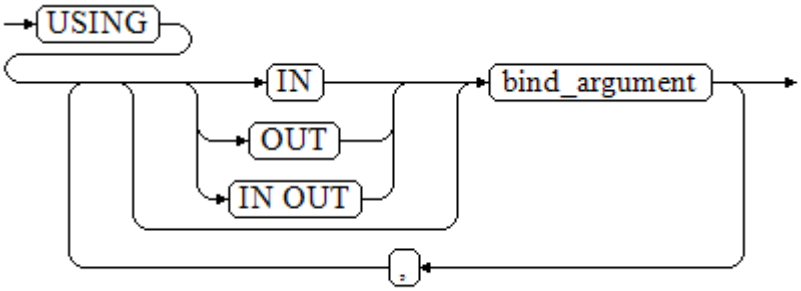
语法请参见**图10-11**。

图 10-11 call_procedure::=



using_clause子句的语法参见图10-12。

图 10-12 using_clause::=



对以上语法格式的解释如下：

- CALL procedure_name，调用存储过程。
- [:placeholder1，:placeholder2，…]，存储过程参数占位符列表。占位符个数与参数个数相同。
- USING [IN|OUT|IN OUT] bind_argument，用于指定存放传递给存储过程参数值的变量。bind_argument前的修饰符与对应参数的修饰符一致。

示例

```
--创建存储过程proc_add。
openGauss=# CREATE OR REPLACE PROCEDURE proc_add
(
    param1    in   INTEGER,
    param2    out  INTEGER,
    param3    in   INTEGER
)
AS
BEGIN
   param2:= param1 + param3;
END;
/

openGauss=# DECLARE
    input1 INTEGER:=1;
    input2 INTEGER:=2;
    statement  VARCHAR2(200);
    param2     INTEGER;
BEGIN
   --声明调用语句
```



```
    statement := 'call proc_add(:col_1, :col_2, :col_3)';
    --执行语句
    EXECUTE IMMEDIATE statement
        USING IN input1, OUT param2, IN input2;
    dbe_output.print_line('result is: '||to_char(param2));
END;
/

--删除存储过程
openGauss=# DROP PROCEDURE proc_add;
```

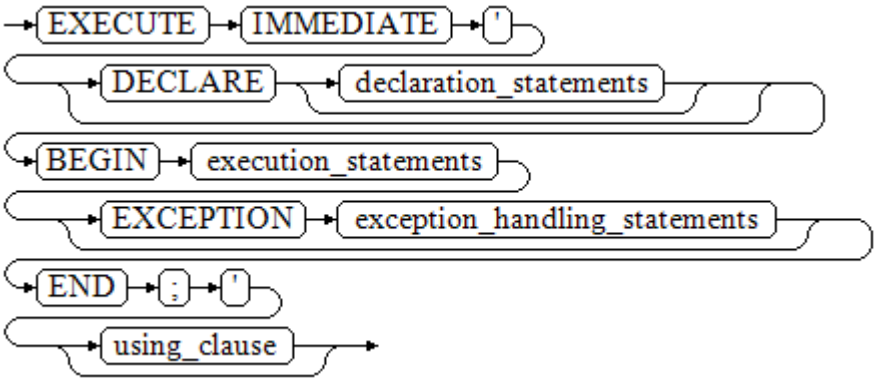
10.7.4 动态调用匿名块

动态调用匿名块是指在动态语句中执行匿名块，使用EXECUTE IMMEDIATE…USING语句后面带IN、OUT来输入、输出参数。

语法

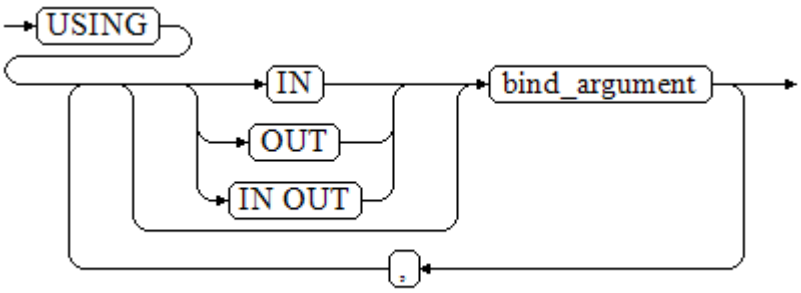
语法请参见图10-13。

图 10-13 call_anonymous_block::=



using_clause子句的语法参见图10-14。

图 10-14 using_clause::=



对以上语法格式的解释如下：

- 匿名块程序实施部分，以BEGIN语句开始，以END语句停顿，以一个分号结束。
- USING [IN|OUT|IN OUT] bind_argument，用于指定存放传递给存储过程参数值的变量。bind_argument前的修饰符与对应参数的修饰符一致。

- 匿名块中间的输入输出参数使用占位符来指明，要求占位符个数与参数个数相同，并且占位符所对应参数的顺序和USING中参数的顺序一致。
- 目前GaussDB在动态语句调用匿名块时，EXCEPTION语句中暂不支持使用占位符进行输入输出参数的传递。
- 不支持调用带有占位符的重载函数。
- 不支持同一条语句同时使用匿名块内声明的变量和绑定参数。
- 仅支持匿名块中调用SQL语句绑定参数，其余绑定参数场景皆不支持。例如：匿名块中调用存储过程，匿名块中使用表达式以及cursor等、匿名块中嵌套调用动态语句。

示例

```
--建表
openGauss=# CREATE SCHEMA hr;
openGauss=# SET CURRENT_SCHEMA = hr;
--创建存储过程dynamic_proc
openGauss=# CREATE OR REPLACE PROCEDURE dynamic_proc
AS
   staff_id     NUMBER(6) := 200;
   first_name   VARCHAR2(20);
   salary       NUMBER(8,2);
BEGIN
    --执行匿名块
    EXECUTE IMMEDIATE 'begin select first_name, salary into :first_name, :salary from hr.staffs where
staff_id= :dno; end;'
       USING OUT first_name, OUT salary, IN staff_id;
   dbe_output.print_line(first_name|| ' ' || salary);
END;
/

--调用存储过程
openGauss=# CALL dynamic_proc();

--删除存储过程
openGauss=# DROP PROCEDURE dynamic_proc;

-- 清除当前数据库模式
openGauss=# SET CURRENT_SCHEMA = public;
openGauss=#  DROP SCHEMA hr CASCADE;
```

10.8 控制语句

10.8.1 返回语句

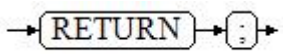
GaussDB提供两种方式返回数据：RETURN或RETURN NEXT及RETURN QUERY。其中，RETURN NEXT和RETURN QUERY只适用于函数，不适用存储过程。

10.8.1.1 RETURN

语法

返回语句的语法请参见图10-15。

图 10-15 return_clause::=



对以上语法的解释如下：

用于将控制从存储过程或函数返回给调用者。

示例

请参见调用语句的**示例**。

10.8.1.2 RETURN NEXT 及 RETURN QUERY

语法

创建函数时需要指定返回值SETOF datatype。

return_next_clause::=

return_query_clause::=

对以上语法的解释如下：

当需要函数返回一个集合时，使用RETURN NEXT或者RETURN QUERY向结果集追加结果，然后继续执行函数的下一条语句。随着后续的RETURN NEXT或RETURN QUERY命令的执行，结果集中会有多个结果。函数执行完成后会一起返回所有结果。

RETURN NEXT可用于标量和复合数据类型。

RETURN QUERY有一种变体RETURN QUERY EXECUTE，后面还可以增加动态查询，通过USING向查询插入参数。

示例

```
openGauss=# CREATE TABLE t1(a int);
openGauss=# INSERT INTO t1 VALUES(1),(10);

--RETURN NEXT
openGauss=# CREATE OR REPLACE FUNCTION fun_for_return_next() RETURNS SETOF t1 AS $$
DECLARE
  r t1%ROWTYPE;
BEGIN
  FOR r IN select * from t1
  LOOP
    RETURN NEXT r;
  END LOOP;
  RETURN;
END;
$$ LANGUAGE plpgsql;
openGauss=# call fun_for_return_next();
 a
---
 1
 10
(2 rows)

-- RETURN QUERY
openGauss=# CREATE OR REPLACE FUNCTION fun_for_return_query() RETURNS SETOF t1 AS $$
```



```
DECLARE
  r t1%ROWTYPE;
BEGIN
  RETURN QUERY select * from t1;
END;
$$
language plpgsql;
openGauss=# call fun_for_return_next();
 a
---
 1
 10
(2 rows)

openGauss=# DROP PROCEDURE IF EXISTS fun_for_return_next();
DROP PROCEDURE
openGauss=# DROP FUNCTION IF EXISTS fun_for_return_query();
DROP FUNCTION
openGauss=# DROP TABLE t1;
DROP TABLE
```

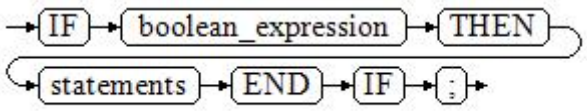
10.8.2 条件语句

条件语句的主要作用判断参数或者语句是否满足已给定的条件，根据判定结果执行相应的操作。

GaussDB有五种形式的IF：

- IF_THEN

  图 10-16 IF_THEN::=

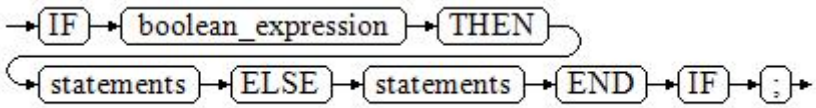
  IF_THEN语句是IF的最简单形式。如果条件为真，statements将被执行。否则，将忽略它们的结果使该IF_THEN语句执行结束。

  示例

  ```
  openGauss=#
  DECLARE
      v_user_id integer default 1;
  BEGIN
  IF v_user_id <> 0 THEN
      raise info 'v_user_id is NOT 0';

  END IF;
  END;
  /
  INFO:  v_user_id is NOT 0
  ```

- IF_THEN_ELSE

  图 10-17 IF_THEN_ELSE::=

IF_THEN_ELSE语句增加了ELSE的分支，可以声明在条件为假的时候执行的语句。

示例

```
openGauss=#
DECLARE
    v_user_id integer default 1;
BEGIN
    IF v_user_id <> 0 THEN
        raise info 'v_user_id is NOT 0';
    ELSE
        raise info 'v_user_id is 0';
    END IF;
END;
/
INFO:  v_user_id is NOT 0
```

- IF_THEN_ELSE IF

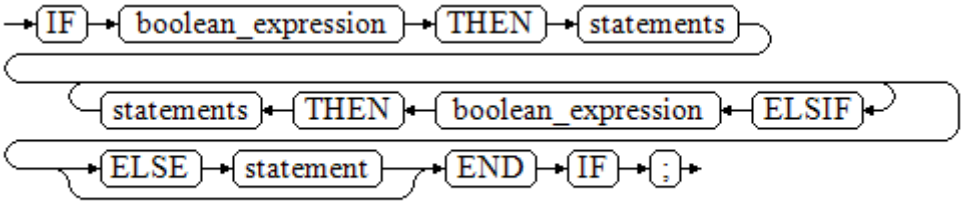
  IF语句可以嵌套，嵌套方式如下：

```
openGauss=#
DECLARE
    v_user_id integer default 1;
BEGIN
    IF v_user_id = 0 THEN
        raise info 'v_user_id is  0';
    ELSE
        IF v_user_id > 0 THEN
            raise info 'v_user_id > 0';
        END IF;
    END IF;
END;
/
INFO:  v_user_id > 0
```

  这种形式实际上就是在一个IF语句的ELSE部分嵌套了另一个IF语句。因此需要一个END IF语句给每个嵌套的IF，另外还需要一个END IF语句结束父IF-ELSE。如果有多个选项，可使用下面的形式。

- IF_THEN_ELSIF_ELSE

  图 10-18 IF_THEN_ELSIF_ELSE::=



  示例

```
DECLARE
    v_user_id integer default NULL;
BEGIN
    IF v_user_id = 0 THEN
        raise info 'v_user_id is  0';
    ELSIF v_user_id > 0 THEN
        raise info 'v_user_id > 0';
    ELSIF v_user_id < 0 THEN
        raise info 'v_user_id < 0';
    ELSE
        raise info 'v_user_id is NULL';
    END IF;
END;
```



```
/
INFO:  v_user_id is NULL
```

- IF_THEN_ELSEIF_ELSE

  ELSEIF是ELSIF的别名。

  综合示例

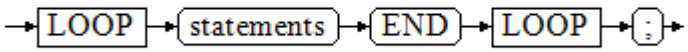
```
CREATE OR REPLACE PROCEDURE proc_control_structure(i in integer)
AS
    BEGIN
        IF i > 0 THEN
            raise info 'i:% is greater than 0. ',i;
        ELSIF i < 0 THEN
            raise info 'i:% is smaller than 0. ',i;
        ELSE
            raise info 'i:% is equal to 0. ',i;
        END IF;
        RETURN;
    END;
/

CALL proc_control_structure(3);

--删除存储过程
DROP PROCEDURE proc_control_structure;
```

10.8.3 循环语句

简单 LOOP 语句

语法图

图 10-19 loop::=



示例

```
CREATE OR REPLACE PROCEDURE proc_loop(i in integer, count out integer)
AS
    BEGIN
        count:=0;
        LOOP
        IF count > i THEN
            raise info 'count is %. ', count;
            EXIT;
        ELSE
            count:=count+1;
        END IF;
        END LOOP;
    END;
/

CALL proc_loop(10,5);
```

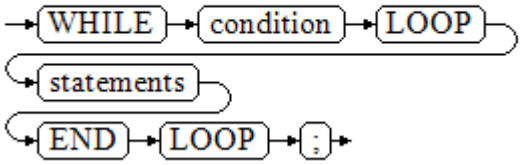
须知

该循环必须要结合EXIT使用，否则将陷入死循环。

WHILE_LOOP 语句

语法图

图 10-20 while_loop::=



只要条件表达式为真，WHILE语句就会不停地在一系列语句上进行循环，在每次进入循环体的时候进行条件判断。

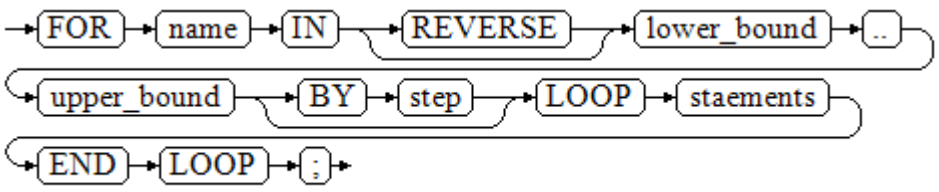
示例

```
CREATE TABLE integertable(c1 integer) DISTRIBUTE BY hash(c1);
CREATE OR REPLACE PROCEDURE proc_while_loop(maxval in integer)
AS
    DECLARE
    i int :=1;
    BEGIN
        WHILE i < maxval LOOP
            INSERT INTO integertable VALUES(i);
            i:=i+1;
        END LOOP;
    END;
/

--调用函数
CALL proc_while_loop(10);

--删除存储过程和表
DROP PROCEDURE proc_while_loop;
DROP TABLE integertable;
```

FOR_LOOP（integer 变量）语句

语法图

图 10-21 for_loop::=



说明

- 变量name会自动定义为integer类型并且只在此循环里存在。变量name介于lower_bound和upper_bound之间。
- 当使用REVERSE关键字时，lower_bound必须大于等于upper_bound，否则循环体不会被执行。

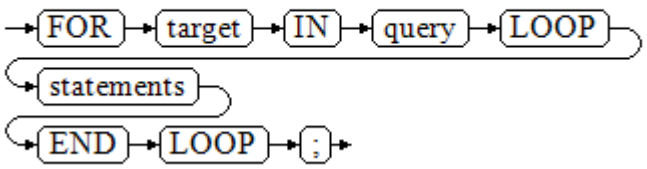
示例

```
--从0到5进行循环
CREATE OR REPLACE PROCEDURE proc_for_loop()
AS
    BEGIN
    FOR I IN 0..5 LOOP
        DBE_OUTPUT.PRINT_LINE('It is '||to_char(I) || ' time;') ;
    END LOOP;
END;
/

--调用函数
CALL proc_for_loop();

--删除存储过程
DROP PROCEDURE proc_for_loop;
```

FOR_LOOP 查询语句

语法图

图 10-22 for_loop_query::=



说明

变量target会自动定义，类型和query的查询结果的类型一致，并且只在此循环中有效。target的取值就是query的查询结果。

示例

```
--循环输出查询结果。
CREATE OR REPLACE PROCEDURE proc_for_loop_query()
AS
    record VARCHAR2(50);
BEGIN
    FOR record IN SELECT spcname FROM pg_tablespace LOOP
    dbe_output.print_line(record);
    END LOOP;
END;
/

--调用函数
CALL proc_for_loop_query();
```

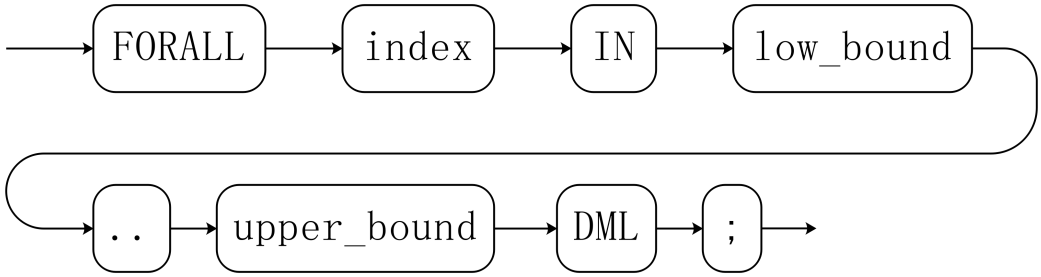


```
--删除存储过程
DROP PROCEDURE proc_for_loop_query;
```

FORALL 批量查询语句

语法图

图 10-23 forall::=



说明

变量index会自动定义为integer类型并且只在此循环里存在。index的取值介于low_bound和upper_bound之间。

示例

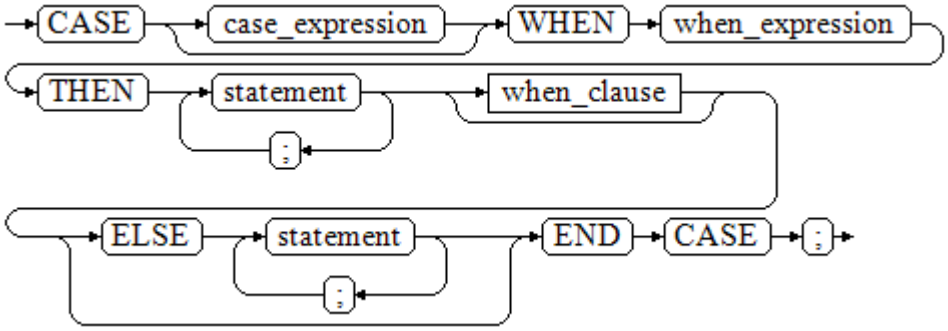
```
CREATE TABLE TEST_t1 (
  title NUMBER(6),
  did VARCHAR2(20),
  data_period VARCHAR2(25),
  kind VARCHAR2(25),
  interval VARCHAR2(20),
  time DATE,
  isModified VARCHAR2(10)
)
DISTRIBUTE BY hash(did);

INSERT INTO TEST_t1 VALUES( 8, 'Donald', 'OConnell', 'DOCONNEL', '650.507.9833', to_date('21-06-1999',
'dd-mm-yyyy'), 'SH_CLERK' );

CREATE OR REPLACE PROCEDURE proc_forall()
AS
BEGIN
    FORALL i IN 100..120
        update TEST_t1 set title = title + 100*i;
END;
/

--调用函数
CALL proc_forall();

--查询存储过程调用结果
SELECT * FROM TEST_t1 WHERE title BETWEEN 100 AND 120;

--删除存储过程和表
DROP PROCEDURE proc_forall;
DROP TABLE TEST_t1;
```

10.8.4 分支语句

语法

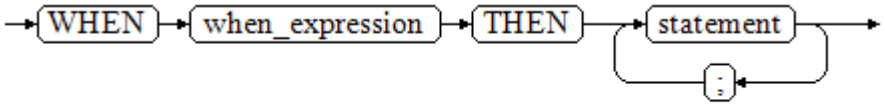
分支语句的语法请参见图10-24。

图 10-24 case_when::=



when_clause子句的语法图参见图10-25。

图 10-25 when_clause::=



参数说明：

- case_expression：变量或表达式。
- when_expression：常量或者条件表达式。
- statement：执行语句。

示例

```
CREATE OR REPLACE PROCEDURE proc_case_branch(pi_result in integer, pi_return out integer)
AS
   BEGIN
      CASE pi_result
         WHEN 1 THEN
            pi_return := 111;
         WHEN 2 THEN
            pi_return := 222;
         WHEN 3 THEN
            pi_return := 333;
         WHEN 6 THEN
            pi_return := 444;
         WHEN 7 THEN
            pi_return := 555;
         WHEN 8 THEN
            pi_return := 666;
         WHEN 9 THEN
            pi_return := 777;
         WHEN 10 THEN
            pi_return := 888;
         ELSE
            pi_return := 999;
      END CASE;
      raise info 'pi_return : %',pi_return ;
END;
/

CALL proc_case_branch(3,0);
```



```
--删除存储过程
DROP PROCEDURE proc_case_branch;
```

10.8.5 空语句

在PL/SQL程序中，可以用NULL语句来说明“不用做任何事情”，相当于一个占位符，可以使某些语句变得有意义，提高程序的可读性。

语法

空语句的用法如下：

```
DECLARE
   ...
BEGIN
   ...
   IF v_num IS NULL THEN
      NULL; -- 不需要处理任何数据。
   END IF;
END;
/
```

10.8.6 错误捕获语句

缺省时，当PL/SQL函数执行过程中发生错误时退出函数执行，并且周围的事务也会回滚。可以用一个带有EXCEPTION子句的BEGIN块捕获错误并且从中恢复。其语法是正常的BEGIN块语法的一个扩展：

```
[<<label>>]
[DECLARE
   declarations]
BEGIN
   statements
EXCEPTION
   WHEN condition [OR condition ...] THEN
      handler_statements
   [WHEN condition [OR condition ...] THEN
      handler_statements
   ...]
END;
```

如果没有发生错误，这种形式的块儿只是简单地执行所有语句，然后转到END之后的下一个语句。但是如果在执行的语句内部发生了一个错误，则这个语句将会回滚，然后转到EXCEPTION列表，寻找匹配错误的第一个条件。若找到匹配，则执行对应的handler_statements，然后转到END之后的下一个语句。如果没有找到匹配，则会向事务的外层报告错误，和没有EXCEPTION子句一样。错误码可以捕获同一类的其他错误码。

也就是说该错误可以被一个包围块用EXCEPTION捕获，如果没有包围块，则进行退出函数处理。

condition的名称可以是《 错误码参考 》的SQL标准错误码编号说明的任意值。特殊的条件名OTHERS匹配除了QUERY_CANCELED之外的所有错误类型。

如果在选中的handler_statements里发生了新错误，则不能被这个EXCEPTION子句捕获，而是向事务的外层报告错误。一个外层的EXCEPTION子句可以捕获它。

如果一个错误被EXCEPTION捕获，PL/SQL函数的局部变量保持错误发生时的原值，但是所有该块中想写入数据库中的状态都回滚。

示例：

```
CREATE TABLE mytab(id INT,firstname VARCHAR(20),lastname VARCHAR(20)) DISTRIBUTE BY hash(id);
```



```
INSERT INTO mytab(firstname, lastname) VALUES('Tom', 'Jones');

CREATE FUNCTION fun_exp() RETURNS INT
AS $$
DECLARE
    x INT :=0;
    y INT;
BEGIN
    UPDATE mytab SET firstname = 'Joe' WHERE lastname = 'Jones';
    x := x + 1;
    y := x / 0;
EXCEPTION
    WHEN division_by_zero THEN
        RAISE NOTICE 'caught division_by_zero';
        RETURN x;
END;$$
LANGUAGE plpgsql;

call fun_exp();
NOTICE:  caught division_by_zero
 fun_exp
---------
       1
(1 row)

select * from mytab;
 id | firstname | lastname
----+-----------+----------
    | Tom       | Jones
(1 row)

DROP FUNCTION fun_exp();
DROP TABLE mytab;
```

当控制到达给y赋值的地方时，会有一个division_by_zero错误失败。这个错误将被EXCEPTION子句捕获。而在RETURN语句里返回的数值将是x的增量值。

说明

进入和退出一个包含EXCEPTION子句的块要比不包含的块开销大的多。因此，不必要的时候不要使用EXCEPTION。

在下列场景中，无法捕获处理异常，整个存储过程回滚：节点故障、网络故障引起的存储过程参与节点线程退出以及COPY FROM操作中源数据与目标表的表结构不一致造成的异常。

示例： UPDATE/INSERT异常

这个例子根据使用异常处理器执行恰当的UPDATE或INSERT 。

```
CREATE TABLE db (a INT, b TEXT);

CREATE FUNCTION merge_db(key INT, data TEXT) RETURNS VOID AS
$$
BEGIN
    LOOP

--第一次尝试更新key
        UPDATE db SET b = data WHERE a = key;
        IF found THEN
            RETURN;
        END IF;
--不存在，所以尝试插入key，如果其他人同时插入相同的key，可能得到唯一key失败。
        BEGIN
            INSERT INTO db(a,b) VALUES (key, data);
            RETURN;
        EXCEPTION WHEN unique_violation THEN
        --什么也不做，并且循环尝试再次更新。
        END;
    END LOOP;
```



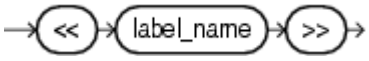
```
END;
$$
LANGUAGE plpgsql;

SELECT merge_db(1, 'david');
SELECT merge_db(1, 'dennis');

--删除FUNCTION和TABLE
DROP FUNCTION merge_db;
DROP TABLE db ;
```

10.8.7 GOTO 语句

GOTO语句可以实现从GOTO位置到目标语句的无条件跳转。GOTO语句会改变原本的执行逻辑，因此应该慎重使用，或者也可以使用EXCEPTION处理特殊场景。当执行GOTO语句时，目标Label必须是唯一的。

语法

label declaration ::=



goto statement ::=

示例

```
openGauss=# CREATE OR REPLACE PROCEDURE GOTO_test()
AS
DECLARE
    v1  int;
BEGIN
    v1  := 0;
        LOOP
        EXIT WHEN v1 > 100;
                v1 := v1 + 2;
                if v1 > 25 THEN
                        GOTO pos1;
                END IF;
        END LOOP;
<<pos1>>
v1 := v1 + 10;
raise info 'v1 is %. ', v1;
END;
/

call GOTO_test();
```

限制场景

GOTO使用有以下限制场景

- 不支持有多个相同的GOTO labels目标场景，无论是否在同一个block中。

  ```
  BEGIN
    GOTO pos1;
    <<pos1>>
    SELECT * FROM ...
    <<pos1>>
    UPDATE t1 SET ...
  END;
  ```


- 不支持GOTO跳转到IF语句，CASE语句，LOOP语句中。

```
BEGIN
  GOTO pos1;
  IF valid THEN
    <<pos1>>
    SELECT * FROM ...
  END IF;
END;
```

- 不支持GOTO语句从一个IF子句跳转到另一个IF子句，或从一个CASE语句的WHEN子句跳转到另一个WHEN子句。

```
BEGIN
  IF valid THEN
    GOTO pos1;
    SELECT * FROM ...
  ELSE
    <<pos1>>
    UPDATE t1 SET ...
  END IF;
END;
```

- 不支持从外部块跳转到内部的BEGIN-END块。

```
BEGIN
  GOTO pos1;
  BEGIN
    <<pos1>>
    UPDATE t1 SET ...
  END;
END;
```

- 不支持从异常处理部分跳转到当前的BEGIN-END块。但可以跳转到上层BEGIN-END块。

```
BEGIN
  <<pos1>>
  UPDATE t1 SET ...
  EXCEPTION
    WHEN condition THEN
      GOTO pos1;
END;
```

- 如果从GOTO到一个不包含执行语句的位置，需要添加NULL语句。

```
DECLARE
  done  BOOLEAN;
BEGIN
  FOR i IN 1..50 LOOP
    IF done THEN
      GOTO end_loop;
    END IF;
    <<end_loop>>  -- not allowed unless an executable statement follows
    NULL; -- add NULL statement to avoid error
  END LOOP;  -- raises an error without the previous NULL
END;
/
```

10.9 事务语句

存储过程本身会自动处于一个事务中。调用最外围存储过程开始时会自动开启一个事务，同时在调用结束时自动提交或者中间异常时回滚。除了系统自动的事务控制外，也可以使用COMMIT/ROLLBACK来控制存储过程中的事务。在存储过程中调用COMMIT/ROLLBACK命令，将提交/回滚当前事务并自动开启一个新的事务，后续的所有操作都会运行在此新的事务中。

保存点SAVEPOINT是事务中的一个特殊记号，它允许将那些在它建立后执行的命令全部回滚，把事务的状态恢复到保存点所在的时刻。存储过程中允许使用保存点来进行事务管理，当前支持保存点的创建、回滚和释放操作。需要特别注意，存储过程中使

用回滚保存点只是回退当前事务的修改，而不会改变存储过程的执行流程，也不会回退存储过程中的局部变量值等。

须知

支持调用的上下文环境：

1. 支持在PL/SQL的存储过程/函数内使用COMMIT/ROLLBACK/SAVEPOINT。
2. 支持含有EXCEPTION的存储过程/函数使用COMMIT/ROLLBACK/SAVEPOINT。
3. 支持在存储过程的EXCEPTION语句内使用COMMIT/ROLLBACK/SAVEPOINT。
4. 支持在事务块里调用含有COMMIT/ROLLBACK/SAVEPOINT的存储过程，即通过/BEGIN/START/END等开启控制的外部事务。
5. 支持在子事务中调用含SAVEPOINT的存储过程并使用外部定义的SAVEPOINT，回退事务状态到存储过程外定义的SAVEPOINT位置。
6. 支持在存储过程外部可见存储过程内部定义的SAVEPOINT，即存储过程外可以将事务修改回滚到存储过程中定义SAVEPOINT的位置。
7. 支持多数PL/SQL的上下文和语句内调用COMMIT/ROLLBACK/SAVEPOINT，包括常用的IF/FOR/CURSOR LOOP/WHILE。

支持提交/回滚的内容：

1. 支持DDL在COMMIT/ROLLBACK后的提交/回滚。
2. 支持DML的COMMIT/ROLLBACK后的提交。
3. 支持存储过程内GUC参数的回滚提交。

语法

```
定义保存点
    SAVEPOINT savepoint_name;
回滚保存点
    ROLLBACK TO [SAVEPOINT] savepoint_name;
释放保存点
    RELEASE [SAVEPOINT] savepoint_name;
```

示例

说明

支持在PL/SQL的存储过程内使用COMMIT/ROLLBACK。

```
CREATE TABLE EXAMPLE1(COL1 INT);

CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE()
AS
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1(COL1) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
END;
/
```


说明

- 支持含有EXCEPTION的存储过程使用COMMIT/ROLLBACK。
- 支持在存储过程的EXCEPTION语句内使用COMMIT/ROLLBACK。
- 支持DDL在COMMIT/ROLLBACK后的提交/回滚。

```
CREATE OR REPLACE PROCEDURE TEST_COMMIT_INSERT_EXCEPTION_ROLLBACK()
AS
BEGIN
 DROP TABLE IF EXISTS TEST_COMMIT;
 CREATE TABLE TEST_COMMIT(A INT, B INT);
 INSERT INTO TEST_COMMIT SELECT 1, 1;
 COMMIT;
       CREATE TABLE TEST_ROLLBACK(A INT, B INT);
 RAISE EXCEPTION 'RAISE EXCEPTION AFTER COMMIT';
EXCEPTION
   WHEN OTHERS THEN
 INSERT INTO TEST_COMMIT SELECT 2, 2;
 ROLLBACK;
END;
/
```

说明

支持在事务块里调用含有COMMIT/ROLLBACK的存储过程，即通过/BEGIN/START/END等开启控制的外部事务。

```
BEGIN;
   CALL TEST_COMMIT_INSERT_EXCEPTION_ROLLBACK();
END;
```

说明

支持多数PL/SQL的上下文和语句内调用COMMIT/ROLLBACK，包括常用的IF/FOR/CURSOR LOOP/WHILE。

```
CREATE OR REPLACE PROCEDURE TEST_COMMIT2()
IS
BEGIN
   DROP TABLE IF EXISTS TEST_COMMIT;
   CREATE TABLE TEST_COMMIT(A INT);
   FOR I IN REVERSE 3..0 LOOP
 INSERT INTO TEST_COMMIT SELECT I;
 COMMIT;
   END LOOP;
   FOR I IN REVERSE 2..4 LOOP
 UPDATE TEST_COMMIT SET A=I;
 COMMIT;
   END LOOP;
EXCEPTION
WHEN OTHERS THEN
 INSERT INTO TEST_COMMIT SELECT 4;
   COMMIT;
END;
/
```

说明

- 支持存储过程内GUC参数的回滚提交。
- enable_force_vector_engine GUC参数不建议开发者使用。

```
SHOW explain_perf_mode;
SHOW enable_force_vector_engine;

CREATE OR REPLACE PROCEDURE GUC_ROLLBACK()
AS
BEGIN
   SET enable_force_vector_engine = on;
```



```
    COMMIT;
    SET explain_perf_mode TO pretty;
    ROLLBACK;
END;
/

call GUC_ROLLBACK();
SHOW explain_perf_mode;
SHOW enable_force_vector_engine;
SET enable_force_vector_engine = off;
```

说明

支持在PL/SQL的存储过程内使用保存点回退事务部分修改。

```
CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE1()
AS
BEGIN
    INSERT INTO EXAMPLE1 VALUES(1);
    SAVEPOINT s1;
    INSERT INTO EXAMPLE1 VALUES(2);
    ROLLBACK TO s1;  -- 回退插入记录2
    INSERT INTO EXAMPLE1 VALUES(3);
END;
/
```

说明

支持在PL/SQL的存储过程中使用保存点回退到存储过程外部定义的保存点。

```
CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE2()
AS
BEGIN
    INSERT INTO EXAMPLE1 VALUES(2);
    ROLLBACK TO s1;  -- 回退插入记录2
    INSERT INTO EXAMPLE1 VALUES(3);
END;
/

BEGIN;
INSERT INTO EXAMPLE1 VALUES(1);
SAVEPOINT s1;
CALL STP_SAVEPOINT_EXAMPLE2();
SELECT * FROM EXAMPLE1;
COMMIT;
```

说明

支持在存储过程外部回退到在PL/SQL存储过程内部定义的保存点。

```
CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE3()
AS
BEGIN
    INSERT INTO EXAMPLE1 VALUES(1);
    SAVEPOINT s1;
    INSERT INTO EXAMPLE1 VALUES(2);
END;
/

BEGIN;
INSERT INTO EXAMPLE1 VALUES(3);
CALL STP_SAVEPOINT_EXAMPLE3();
ROLLBACK TO SAVEPOINT s1; --回退存储过程中插入记录2
SELECT * FROM EXAMPLE1;
COMMIT;
```

说明

支持函数(Function)中调用commit/rollback语句。


```
CREATE OR REPLACE FUNCTION FUNCTION_EXAMPLE1() RETURN INT
AS
EXP INT;
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1(col1) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
    SELECT COUNT(*) FROM EXAMPLE1 INTO EXP;
    RETURN EXP;
END;
/
```

限制场景

⚠ 注意

- 不支持调用的上下文环境：
  1. 不支持除PL/SQL的其他存储过程中调用COMMIT/ROLLBACK/SAVEPOINT，例如PLPYTHON等。
  2. 不支持事务块中调用了SAVEPOINT后，调用含有COMMIT/ROLLBACK的存储过程。
  3. 不支持TRIGGER中调用含有COMMIT/ROLLBACK/SAVEPOINT语句的存储过程。
  4. 不支持EXECUTE语句中调用COMMIT/ROLLBACK/SAVEPOINT语句。
  5. 不支持在CURSOR语句中打开一个含有COMMIT/ROLLBACK/SAVEPOINT的存储过程。
  6. 不支持带有IMMUTABLE以及SHIPPABLE的存储过程调用COMMIT/ROLLBACK/SAVEPOINT，或调用带有COMMIT/ROLLBACK/SAVEPOINT语句的存储过程。
  7. 不支持SQL中调用含有COMMIT/ROLLBACK/SAVEPOINT语句的存储过程，除了SELECT PROC以及CALL PROC。
  8. 存储过程头带有GUC参数设置的不允许调用COMMIT/ROLLBACK/SAVEPOINT语句。
  9. 不支持CURSOR/EXECUTE语句，以及各类表达式内调用COMMIT/ROLLBACK/SAVEPOINT。
  10. 不支持存储过程返回值与表达式计算中调用含有COMMIT/ROLLBACK/SAVEPOINT的存储过程。
  11. 不支持存储过程中释放存储过程外部定义的保存点。
  12. 存储过程事务和其中的自治事务是两个独立的事务，不能互相使用对方事务中定义的保存点
- 不支持提交回滚的内容：
  1. 不支持存储过程内声明变量以及传入变量的提交/回滚。
  2. 不支持存储过程内必须重启生效的GUC参数的提交/回滚。

在存储过程使用commit/rollback有以下限制场景：

说明

不允许Trigger的存储过程包含commit/rollback语句，或调用带有commit/rollback语句的存储过程。

```
CREATE OR REPLACE FUNCTION FUNCTION_TRI_EXAMPLE2() RETURN TRIGGER
AS
EXP INT;
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1(col1) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
    SELECT COUNT(*) FROM EXAMPLE1 INTO EXP;
END;
/

CREATE TRIGGER TRIGGER_EXAMPLE AFTER DELETE ON EXAMPLE1
FOR EACH ROW EXECUTE PROCEDURE FUNCTION_TRI_EXAMPLE2();

DELETE FROM EXAMPLE1;
```

说明

不支持带有IMMUTABLE以及SHIPPABLE的存储过程调用commit/rollblack，或调用带有commit/rollback语句的存储过程。

```
CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE1()
IMMUTABLE
AS
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1 (col1) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
END;
/
```

说明

不支持存储过程中任何变量的提交，包括存储过程内声明的变量或者传入的参数。

```
CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE2(EXP_OUT OUT INT)
AS
EXP INT;
BEGIN
    EXP_OUT := 0;
    COMMIT;
    DBE_OUTPUT.PRINT_LINE('EXP IS:'||EXP);
    EXP_OUT := 1;
    ROLLBACK;
    DBE_OUTPUT.PRINT_LINE('EXP IS:'||EXP);
END;
/
```

说明

不支持出现在SQL中的调用（除了Select Procedure）。

```
CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE3()
AS
BEGIN
```



```
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1 (col1) VALUES (i);
        IF i % 2 = 0 THEN
            EXECUTE IMMEDIATE 'COMMIT';
        ELSE
            EXECUTE IMMEDIATE 'ROLLBACK';
        END IF;
    END LOOP;
END;
/
```

说明

存储过程头带有GUC参数设置的不允许调用commit/rollback语句。

```
CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE4()
SET ARRAY_NULLS TO "ON"
AS
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1 (col1) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
END;
/
```

说明

游标open的对象不允许为带有commit/rollback语句的存储过程。

```
CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE5(INTIN IN INT, INTOUT OUT INT)
AS
BEGIN
INTOUT := INTIN + 1;
COMMIT;
END;
/

CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE6()
AS
CURSOR CURSOR1(EXPIN INT)
IS SELECT TRANSACTION_EXAMPLE5(EXPIN);
INTEXP INT;
BEGIN
    FOR i IN 0..20 LOOP
        OPEN CURSOR1(i);
        FETCH CURSOR1 INTO INTEXP;
        INSERT INTO EXAMPLE1(COL1) VALUES (INTEXP);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
        CLOSE CURSOR1;
    END LOOP;
END;
/
```

说明

不支持CURSOR/EXECUTE语句，以及各类表达式内调用COMMIT/ROLLBACK。

```
CREATE OR REPLACE PROCEDURE exec_func1()
AS
BEGIN
    CREATE TABLE TEST_exec(A INT);
COMMIT;
```



```
END;
/
CREATE OR REPLACE PROCEDURE exec_func2()
AS
BEGIN
EXECUTE exec_func1();
COMMIT;
END;
/
```

说明

不支持存储过程返回值与表达式计算。

```
CREATE OR REPLACE PROCEDURE exec_func3(RET_NUM OUT INT)
AS
BEGIN
    RET_NUM := 1+1;
COMMIT;
END;
/
CREATE OR REPLACE PROCEDURE exec_func4(ADD_NUM IN INT)
AS
SUM_NUM INT;
BEGIN
SUM_NUM := ADD_NUM + exec_func3();
COMMIT;
END;
/
```

说明

不支持存储过程中释放存储过程外部定义的保存点。

```
CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE3()
AS
BEGIN
    INSERT INTO EXAMPLE1 VALUES(2);
    RELEASE SAVEPOINT s1;  -- 释放存储过程外部定义的保存点
    INSERT INTO EXAMPLE1 VALUES(3);
END;
/

BEGIN;
INSERT INTO EXAMPLE1 VALUES(1);
SAVEPOINT s1;
CALL STP_SAVEPOINT_EXAMPLE3();
COMMIT;
```

10.10 其他语句

10.10.1 锁操作

GaussDB提供了多种锁模式用于控制对表中数据的并发访问。这些模式可以用在MVCC（多版本并发控制）无法给出期望行为的场合。同样，大多数GaussDB命令自动施加恰当的锁，以保证被引用的表在命令的执行过程中不会以一种不兼容的方式被删除或者修改。比如，在存在其他并发操作的时候，ALTER TABLE是不能在同一个表上执行的。

完整的锁操作请参见**LOCK**。

10.10.2 游标操作

GaussDB中游标（cursor）是系统为用户开设的一个数据缓冲区，存放着SQL语句的执行结果。每个游标区都有一个名称。用户可以用SQL语句逐一从游标中获取记录，并赋给主变量，交由主语言进一步处理。

游标的操作主要有游标的定义、打开、获取和关闭。

完整的游标操作示例可参考**显式游标**。

10.11 游标

10.11.1 游标概述

为了处理SQL语句，存储过程进程分配一段内存区域来保存上下文联系。游标是指向上下文区域的句柄或指针。借助游标，存储过程可以控制上下文区域的变化。

须知

当游标作为存储过程的返回值时，如果使用JDBC调用该存储过程，返回的游标将不可用。

游标的使用分为显式游标和隐式游标。对于不同的SQL语句，游标的使用情况不同，详细信息请参见**表10-2**。

表 10-2 游标使用情况

SQL语句	游标
非查询语句	隐式的
结果是单行的查询语句	隐式的或显式的
结果是多行的查询语句	显式的

10.11.2 显式游标

显式游标主要用于对查询语句的处理，尤其是在查询结果为多条记录的情况下。

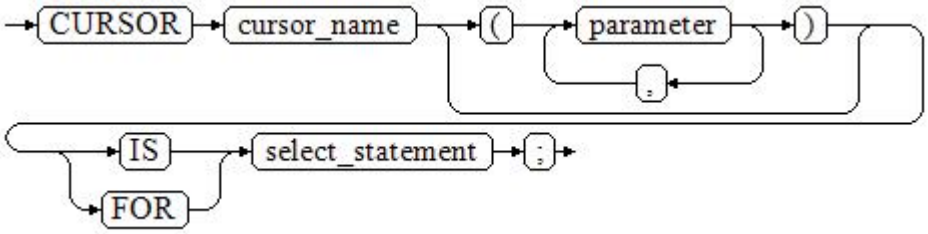
处理步骤

显式游标处理需六个PL/SQL步骤：

步骤1 **定义静态游标：**就是定义一个游标名，以及与其相对应的SELECT语句。

定义静态游标的语法图，请参见**图10-26**。

图 10-26 static_cursor_define::=



参数说明：

- cursor_name：定义的游标名。
- parameter：游标参数，只能为输入参数，其格式为：
  ```
  parameter_name datatype
  ```
- select_statement：查询语句。

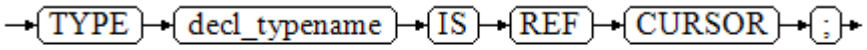
说明

- 根据执行计划的不同，系统会自动判断该游标是否可以用于以倒序的方式检索数据行。
- 语法上支持parameter为输出参数，但其行为与输入参数保持一致。

定义动态游标：指ref游标，可以通过一组静态的SQL语句动态的打开游标。首先定义ref游标类型，然后定义该游标类型的游标变量，在打开游标时通过OPEN FOR动态绑定SELECT语句。

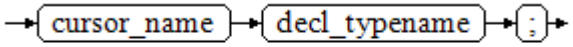
定义动态游标的语法图，请参见图10-27和图10-28。

图 10-27 cursor_typename::=



GaussDB支持sys_refcursor动态游标类型，函数或存储过程可以通过sys_refcursor参数传入或传出游标结果集合，函数也可以通过返回sys_refcursor来返回游标结果集合。

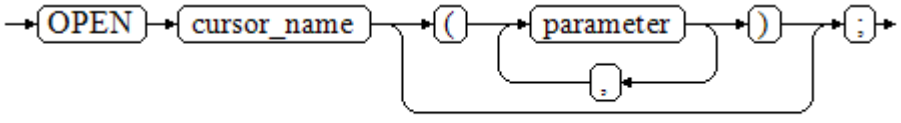
图 10-28 dynamic_cursor_define::=



步骤2 **打开静态游标：**就是执行游标所对应的SELECT语句，将其查询结果放入工作区，并且指针指向工作区的首部，标识游标结果集合。如果游标查询语句中带有FOR UPDATE选项，OPEN语句还将锁定数据库表中游标结果集合对应的数据行。

打开静态游标的语法图，请参见图10-29。

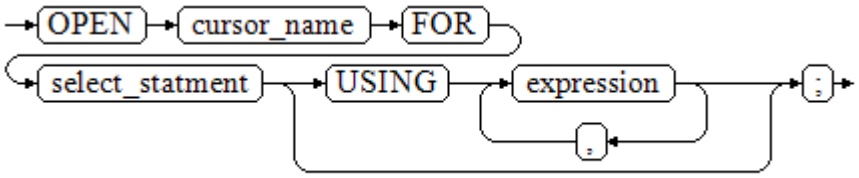
图 10-29 open_static_cursor::=



打开动态游标：可以通过OPEN FOR语句打开动态游标，动态绑定SQL语句。

打开动态游标的语法图，请参见图10-30。

图 10-30 open_dynamic_cursor::=

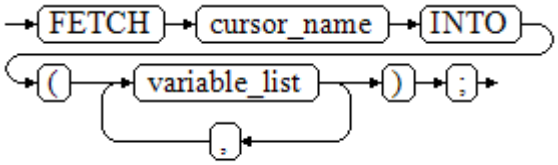


PL/SQL程序不能用OPEN语句重复打开一个游标。

步骤3 提取游标数据：检索结果集合中的数据行，放入指定的输出变量中。

提取游标数据的语法图，请参见图10-31。

图 10-31 fetch_cursor::=



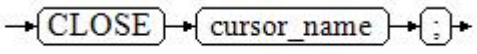
步骤4 对该记录进行处理。

步骤5 继续处理，直到活动集合中没有记录。

步骤6 关闭游标：当提取和处理完游标结果集合数据后，应及时关闭游标，以释放该游标所占用的系统资源，并使该游标的工作区变成无效，不能再使用FETCH语句获取其中数据。关闭后的游标可以使用OPEN语句重新打开。

关闭游标的语法图，请参见图10-32。

图 10-32 close_cursor::=



----结束

属性

游标的属性用于控制程序流程或者了解程序的状态。当运行DML语句时，PL/SQL打开一个内建游标并处理结果，游标是维护查询结果的内存中的一个区域，游标在运行DML语句时打开，完成后关闭。显式游标的属性为：

- %FOUND布尔型属性：当最近一次读记录时成功返回，则值为TRUE。
- %NOTFOUND布尔型属性：与%FOUND相反。
- %ISOPEN布尔型属性：当游标已打开时返回TRUE。

- %ROWCOUNT数值型属性：返回已从游标中读取的记录数。

示例

前置DDL、DML，本节后续示例依赖此用例。

```
DROP SCHEMA IF EXISTS hr CASCADE;
CREATE SCHEMA hr;
SET current_schema = hr;
DROP TABLE IF EXISTS sections;
DROP TABLE IF EXISTS staffs;
DROP TABLE IF EXISTS department;
--创建部门表
CREATE TABLE sections(
    section_name varchar(100),
    place_id     int,
    section_id   int
);
INSERT INTO sections VALUES ('hr',1,1);

--创建员工表
CREATE TABLE staffs(
    staff_id number(6),
    salary number(8,2),
    section_id int,
    first_name varchar(20)
);
INSERT INTO staffs VALUES (1,100,1,'Tom');

--创建部门表
create table department(
    section_id int
);
--游标参数的传递方法。
CREATE OR REPLACE PROCEDURE cursor_proc1()
AS
DECLARE
    DEPT_NAME VARCHAR(100);
    DEPT_LOC NUMBER(4);
    --定义游标
    CURSOR C1 IS
        SELECT section_name, place_id FROM hr.sections WHERE section_id <= 50;
    CURSOR C2(sect_id INTEGER) IS
        SELECT section_name, place_id FROM hr.sections WHERE section_id <= sect_id;
    TYPE CURSOR_TYPE IS REF CURSOR;
    C3 CURSOR_TYPE;
    SQL_STR VARCHAR(100);
BEGIN
    OPEN C1;--打开游标
    LOOP
        --通过游标取值
        FETCH C1 INTO DEPT_NAME, DEPT_LOC;
        EXIT WHEN C1%NOTFOUND;
        DBE_OUTPUT.PRINT_LINE(DEPT_NAME||'---'||DEPT_LOC);
    END LOOP;
    CLOSE C1;--关闭游标

    OPEN C2(10);
    LOOP
        FETCH C2 INTO DEPT_NAME, DEPT_LOC;
        EXIT WHEN C2%NOTFOUND;
        DBE_OUTPUT.PRINT_LINE(DEPT_NAME||'---'||DEPT_LOC);
    END LOOP;
    CLOSE C2;

    SQL_STR := 'SELECT section_name, place_id FROM hr.sections WHERE section_id <= :DEPT_NO;';
    OPEN C3 FOR SQL_STR USING 50;
    LOOP
```



```
        FETCH C3 INTO DEPT_NAME, DEPT_LOC;
        EXIT WHEN C3%NOTFOUND;
        DBE_OUTPUT.PRINT_LINE(DEPT_NAME||'---'||DEPT_LOC);
    END LOOP;
    CLOSE C3;
END;
/
CREATE PROCEDURE
CALL cursor_proc1();
hr---1
hr---1
hr---1
 cursor_proc1
--------------

(1 row)
DROP PROCEDURE cursor_proc1;
DROP PROCEDURE
--给工资低于3000的员工增加工资500。
CREATE TABLE hr.staffs_t1 AS TABLE hr.staffs;

CREATE OR REPLACE PROCEDURE cursor_proc2()
AS
DECLARE
   V_EMPNO  NUMBER(6);
   V_SAL    NUMBER(8,2);
   CURSOR C IS SELECT staff_id, salary FROM hr.staffs_t1;
BEGIN
   OPEN C;
   LOOP
      FETCH C INTO V_EMPNO, V_SAL;
      EXIT WHEN C%NOTFOUND;
      IF V_SAL<=3000 THEN
           UPDATE hr.staffs_t1 SET salary =salary + 500 WHERE staff_id = V_EMPNO;
      END IF;
   END LOOP;
   CLOSE C;
END;
/
CREATE PROCEDURE
CALL cursor_proc2();
 cursor_proc2
--------------

(1 row)
--删除存储过程
DROP PROCEDURE cursor_proc2;
DROP PROCEDURE
DROP TABLE hr.staffs_t1;
DROP TABLE
--SYS_REFCURSOR类型作为函数参数
CREATE OR REPLACE PROCEDURE proc_sys_ref(O OUT SYS_REFCURSOR)
IS
C1 SYS_REFCURSOR;
BEGIN
OPEN C1 FOR SELECT section_ID FROM HR.sections ORDER BY section_ID;
O := C1;
END;
/

DECLARE
C1 SYS_REFCURSOR;
TEMP NUMBER(4);
BEGIN
proc_sys_ref(C1);
LOOP
  FETCH C1 INTO TEMP;
  DBE_OUTPUT.PRINT_LINE(C1%ROWCOUNT);
  EXIT WHEN C1%NOTFOUND;
```



```
END LOOP;
END;
/

--删除存储过程
DROP PROCEDURE proc_sys_ref;
```

10.11.3 隐式游标

对于非查询语句，如修改、删除操作，则由系统自动地为这些操作设置游标并创建其工作区，这些由系统隐含创建的游标称为隐式游标，隐式游标的名称为SQL，这是由系统定义的。

简介

对于隐式游标的操作，如定义、打开、取值及关闭操作，都由系统自动地完成，无需用户进行处理。用户只能通过隐式游标的相关属性，来完成相应的操作。在隐式游标的工作区中，所存放的数据是最新处理的一条SQL语句所包含的数据，与用户自定义的显式游标无关。

格式调用为： SQL%

说明

INSERT，UPDATE，DELETE，SELECT语句中不必明确定义游标。

属性

隐式游标属性为：

- SQL%FOUND布尔型属性：当最近一次读记录时成功返回，则值为TRUE。
- SQL%NOTFOUND布尔型属性：与%FOUND相反。
- SQL%ROWCOUNT数值型属性：返回已从游标中读取的记录数。
- SQL%ISOPEN布尔型属性：取值总是FALSE。SQL语句执行完毕立即关闭隐式游标。

示例

```
--删除EMP表中某部门的所有员工，如果该部门中已没有员工，则在DEPT表中删除该部门。
CREATE TABLE hr.staffs_t1 AS TABLE hr.staffs;
CREATE TABLE hr.sections_t1 AS TABLE hr.sections;

CREATE OR REPLACE PROCEDURE proc_cursor3()
AS
   DECLARE
   V_DEPTNO NUMBER(4) := 100;
   BEGIN
      DELETE FROM hr.staffs WHERE section_ID = V_DEPTNO;
      --根据游标状态做进一步处理
      IF SQL%NOTFOUND THEN
      DELETE FROM hr.sections_t1 WHERE section_ID = V_DEPTNO;
      END IF;
   END;
/

CALL proc_cursor3();

--删除存储过程和临时表
DROP PROCEDURE proc_cursor3;
DROP TABLE hr.staffs_t1;
DROP TABLE hr.sections_t1;
```

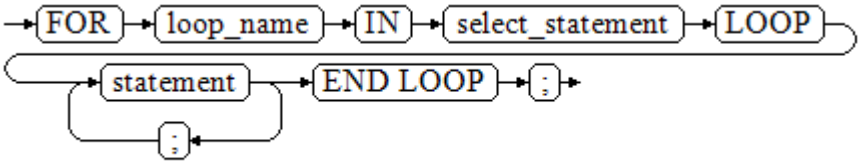

10.11.4 游标循环

游标在WHILE语句、LOOP语句中的使用称为游标循环，一般这种循环都需要使用OPEN、FETCH和CLOSE语句。下面要介绍的一种循环不需要这些操作，可以简化游标循环的操作，这种循环方式适用于静态游标的循环，不用执行静态游标的四个步骤。

语法

FOR AS循环的语法请参见图10-33。

图 10-33 FOR_AS_loop::=



注意事项

- 不能在该循环语句中对查询的表进行更新操作。
- 变量loop_name会自动定义且只在此循环中有效，类型和select_statement的查询结果类型一致。loop_name的取值就是select_statement的查询结果。
- 游标的属性中%FOUND、%NOTFOUND、%ROWCOUNT在GaussDB数据库中都是访问同一个内部变量，事务和匿名块不支持多个游标同时访问。

示例

```
BEGIN
FOR ROW_TRANS IN
        SELECT first_name FROM hr.staffs
    LOOP
        DBE_OUTPUT.PRINT_LINE (ROW_TRANS.first_name );
    END LOOP;
END;
/

--创建表
CREATE TABLE integerTable1( A INTEGER) DISTRIBUTE BY hash(A);
CREATE TABLE integerTable2( B INTEGER) DISTRIBUTE BY hash(B);
INSERT INTO integerTable2 VALUES(2);

--多游标共享游标属性的标量
DECLARE
   CURSOR C1 IS SELECT A FROM integerTable1;--声明游标
   CURSOR C2 IS SELECT B FROM integerTable2;
   PI_A INTEGER;
   PI_B INTEGER;
BEGIN
   OPEN C1;--打开游标
   OPEN C2;
   FETCH C1 INTO PI_A; ----  C1%FOUND 和 C2%FOUND 值为 FALSE
   FETCH C2 INTO PI_B; ----  C1%FOUND 和 C2%FOUND 的值都为 TRUE
   --判断游标状态
   IF C1%FOUND THEN
      IF C2%FOUND THEN
        DBE_OUTPUT.PRINT_LINE('Dual cursor share paremeter.');
      END IF;
   END IF;
```



```
  CLOSE C1;--关闭游标
  CLOSE C2;
END;
/

--删除临时表
DROP TABLE integerTable1;
DROP TABLE integerTable2;

DROP SCHEMA hr CASCADE;
```

10.12 高级包

高级包现有两套接口，第一套为基础接口，第二套是为了提高易用性做了二次封装的接口，推荐使用第二套接口。

10.12.1 基础接口

10.12.1.1 PKG_SERVICE

PKG_SERVICE支持的所有接口请参见**表10-3**。

表 10-3 PKG_SERVICE

接口名称	描述
PKG_SERVICE.SQL_IS_CONTEXT_ACTIVE	确认该CONTEXT是否已注册。
PKG_SERVICE.SQL_CLEAN_ALL_CONTEXTS	取消所有注册的CONTEXT。
PKG_SERVICE.SQL_REGISTER_CONTEXT	注册一个CONTEXT。
PKG_SERVICE.SQL_UNREGISTER_CONTEXT	取消注册该CONTEXT。
PKG_SERVICE.SQL_SET_SQL	向CONTEXT设置一条SQL语句，目前只支持SELECT。
PKG_SERVICE.SQL_RUN	在一个CONTEXT上执行设置的SQL语句。
PKG_SERVICE.SQL_NEXT_ROW	读取该CONTEXT中的下一行数据。
PKG_SERVICE.SQL_GET_VALUE	读取该CONTEXT中动态定义的列值
PKG_SERVICE.SQL_SET_RESULT_TYPE	根据类型OID动态定义该CONTEXT的一个列。
PKG_SERVICE.JOB_CANCEL	通过任务ID来删除定时任务。
PKG_SERVICE.JOB_FINISH	禁用或者启用定时任务。
PKG_SERVICE.JOB_SUBMIT	提交一个定时任务。作业号由系统自动生成或由用户指定。

接口名称	描述
PKG_SERVICE.JOB_UPDATE	修改定时任务的属性，包括任务内容、下次执行时间、执行间隔。
PKG_SERVICE.SUBMIT_ON_NODES	提交一个任务到所有节点，作业号由系统自动生成。
PKG_SERVICE.ISUBMIT_ON_NODES	提交一个任务到所有节点，作业号由用户指定。
PKG_SERVICE.SQL_GET_ARRAY_RESULT	获取该CONTEXT中返回的数组值。
PKG_SERVICE.SQL_GET_VARIABLE_RESULT	获取该CONTEXT中返回的列值。

- PKG_SERVICE.SQL_IS_CONTEXT_ACTIVE

  该函数用来确认一个CONTEXT是否已注册。该函数传入想查找的CONTEXT ID，如果该CONTEXT存在返回TRUE，反之返回FALSE。

  PKG_SERVICE.SQL_IS_CONTEXT_ACTIVE函数原型为：

  ```
  PKG_SERVICE.SQL_IS_CONTEXT_ACTIVE(
   context_id      IN INTEGER
  )
  RETURN BOOLEAN;
  ```

  表 10-4 PKG_SERVICE.SQL_IS_CONTEXT_ACTIVE 接口说明

参数名称	描述
context_id	想查找的CONTEXT ID号

- PKG_SERVICE.SQL_CLEAN_ALL_CONTEXTS

  该函数用来取消所有CONTEXT

  PKG_SERVICE.SQL_CLEAN_ALL_CONTEXTS函数原型为：

  ```
  PKG_SERVICE.SQL_CLEAN_ALL_CONTEXTS(
  )
  RETURN VOID;
  ```

- PKG_SERVICE.SQL_REGISTER_CONTEXT

  该函数用来打开一个CONTEXT，是后续对该CONTEXT进行各项操作的前提。该函数不传入任何参数，内部自动递增生成CONTEXT ID，并作为返回值返回给integer定义的变量。

  PKG_SERVICE.SQL_REGISTER_CONTEXT函数原型为：

  ```
  DBE_SQL.REGISTER_CONTEXT(
  )
  RETURN INTEGER;
  ```

- PKG_SERVICE.SQL_UNREGISTER_CONTEXT

  该函数用来关闭一个CONTEXT，是该CONTEXT中各项操作的结束。如果在存储过程结束时没有调用该函数，则该CONTEXT占用的内存仍然会保存，因此关闭

CONTEXT非常重要。由于异常情况的发生会中途退出存储过程，导致CONTEXT未能关闭，因此建议存储过程中有异常处理，将该接口包含在内。

PKG_SERVICE.SQL_UNREGISTER_CONTEXT函数原型为：

```
PKG_SERVICE.SQL_UNREGISTER_CONTEXT(
 context_id      IN INTEGER
)
RETURN INTEGER;
```

表 10-5 PKG_SERVICE.SQL_UNREGISTER_CONTEXT 接口说明

参数名称	描述
context_id	打算关闭的CONTEXT ID号

- PKG_SERVICE.SQL_SET_SQL

  该函数用来解析给定游标的查询语句，被传入的查询语句会立即执行。目前仅支持SELECT查询语句的解析，且语句参数仅可通过text类型传递，长度不大于1G。

  PKG_SERVICE.SQL_SET_SQL函数的原型为：

  ```
  PKG_SERVICE.SQL_SET_SQL(
  context_id      IN INTEGER,
  query_string    IN TEXT,
  language_flag   IN INTEGER
  )
  RETURN BOOLEAN;
  ```

  表 10-6 PKG_SERVICE.SQL_SET_SQL 接口说明

参数名称	描述
context_id	执行查询语句解析的CONTEXT ID
query_string	执行的查询语句
language_flag	版本语言号，目前只支持1

- PKG_SERVICE.SQL_RUN

  该函数用来执行一个给定的CONTEXT。该函数接收一个CONTEXT ID，运行后获得的数据用于后续操作。目前仅支持SELECT查询语句的执行。

  PKG_SERVICE.SQL_RUN函数的原型为：

  ```
  PKG_SERVICE.SQL_RUN(
  context_id      IN INTEGER,
  )
  RETURN INTEGER;
  ```

  表 10-7 PKG_SERVICE.SQL_RUN 接口说明

参数名称	描述
context_id	执行查询语句解析的CONTEXT ID

- PKG_SERVICE.SQL_NEXT_ROW

  该函数返回执行SQL实际返回的数据行数，每一次运行该接口都会获取到新的行数的集合，直到数据读取完毕获取不到新行为止。

PKG_SERVICE.SQL_NEXT_ROW函数的原型为：

```
PKG_SERVICE.SQL_NEXT_ROW(
context_id      IN INTEGER,
)
RETURN INTEGER;
```

表 10-8 PKG_SERVICE.SQL_NEXT_ROW 接口说明

参数名称	描述
context_id	执行的CONTEXT ID

- PKG_SERVICE.SQL_GET_VALUE

  该函数用来返回给定CONTEXT中给定位置的CONTEXT元素值，该接口访问的是PKG_SERVICE.SQL_NEXT_ROW获取的数据。

  PKG_SERVICE.SQL_GET_VALUE函数的原型为：

```
PKG_SERVICE.SQL_GET_VALUE(
context_id          IN     INTEGER,
pos                 IN     INTEGER,
col_type            IN     ANYELEMENT
)
RETURN ANYELEMENT;
```

表 10-9 PKG_SERVICE.SQL_GET_VALUE 接口说明

参数名称	描述
context_id	执行的CONTEXT ID
pos	动态定义列在查询中的位置
col_type	任意类型变量，定义列的返回值类型

- PKG_SERVICE.SQL_SET_RESULT_TYPE

  该函数用来定义从给定CONTEXT返回的列，该接口只能应用于SELECT定义的CONTEXT中。定义的列通过查询列表的相对位置来标识，
  PKG_SERVICE.SQL_SET_RESULT_TYPE函数的原型为：

```
PKG_SERVICE.SQL_SET_RESULT_TYPE(
context_id      IN INTEGER,
pos             IN INTEGER,
coltype_oid     IN ANYELEMENT,
maxsize         IN INTEGER
)
RETURN INTEGER;
```

表 10-10 PKG_SERVICE.SQL_SET_RESULT_TYPE 接口说明

参数名称	描述
context_id	执行的CONTEXT ID。
pos	动态定义列在查询中的位置。
coltype_oid	任意类型的变量，可根据变量类型得到对应类型OID。

参数名称	描述
maxsize	定义的列的长度。

- PKG_SERVICE.JOB_CANCEL

  存储过程CANCEL删除指定的定时任务。

  PKG_SERVICE.JOB_CANCEL函数原型为：

  ```
  PKG_SERVICE.JOB_CANCEL(
  job  IN  INTEGER);
  ```

  表 10-11 PKG_SERVICE.JOB_CANCEL 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	integer	IN	否	指定的作业号。

  示例：

  ```
  SELECT PKG_SERVICE.JOB_SUBMIT (101, 'insert_msg_statistic1;', sysdate, 'sysdate+3.0/24');
   job_submit
  ------------
         101
  (1 row)
  CALL PKG_SERVICE.JOB_CANCEL(101);
   job_cancel
  ------------

  (1 row)
  ```

- PKG_SERVICE.JOB_FINISH

  存储过程FINISH禁用或者启用定时任务。

  PKG_SERVICE.JOB_FINISH函数原型为：

  ```
  PKG_SERVICE.JOB_FINISH(
  id          IN   INTEGER,
  broken      IN   BOOLEAN,
  next_time   IN   TIMESTAMP  DEFAULT  sysdate);
  ```

  表 10-12 PKG_SERVICE.JOB_FINISH 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	integer	IN	否	指定的作业号。
broken	Boolean	IN	否	状态标志位，true代表禁用，false代表启用。根据true或false值更新当前job；如果为空值，则不改变原有job的状态。

参数	类型	入参/出参	是否可以为空	描述
next_time	timestamp	IN	是	下次运行时间，默认为当前系统时间。如果参数broken状态为true，则更新该参数为'4000-1-1'；如果参数broken状态为false，且如果参数next_time不为空值，则更新指定job的next_time值，如果next_time为空值，则不更新next_time值。该参数可以省略，为默认值。

- PKG_SERVICE.JOB_SUBMIT

  存储过程JOB_SUBMIT提交一个系统提供的定时任务。

  PKG_SERVICE.JOB_SUBMIT函数原型为：

```
PKG_SERVICE.JOB_SUBMIT(
id            IN   BIGINT,
content       IN   TEXT,
next_date     IN   TIMESTAMP DEFAULT sysdate,
interval_time IN   TEXT  DEFAULT 'null',
job           OUT  INTEGER);
```

  说明

  当创建一个定时任务（JOB）时，系统默认将当前数据库和用户名与当前创建的定时任务绑定起来。该接口函数可以通过call或select调用，如果通过select调用，可以不填写出参。如果在存储过程中，则需要通过perform调用该接口函数。如果提交的sql语句任务使用到非public的schema，应该指定表或者函数的schema，或者在sql语句前添加set current_schema = xxx;语句。

  表 10-13 PKG_SERVICE.JOB_SUBMIT 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	bigint	IN	否	作业号。如果传入id为NULL，则内部会生成作业ID。
context	text	IN	否	要执行的SQL语句。支持一个或多个‘DML’，‘匿名块’，‘调用存储过程的语句’或3种混合的场景。
next_time	timestamp	IN	否	下次作业运行时间。默认值为当前系统时间（sysdate）。如果是过去时间，在提交作业时表示立即执行。

参数	类型	入参/出参	是否可以为空	描述
interval_time	text	IN	是	用来计算下次作业运行时间的时间表达式，可以是interval表达式，也可以是sysdate加上一个numeric值（例如：sysdate+1.0/24）。如果为空值或字符串"null"表示只执行一次，执行后JOB状态STATUS变成'd' 不再执行。
job	integer	OUT	否	作业号。范围为1～32767。当使用select调用pkg_service.job_submit时，该参数可以省略。

示例：

```
CREATE TABLE test_table(a int);
CREATE TABLE

CREATE OR REPLACE PROCEDURE test_job(a in int) IS
BEGIN
INSERT INTO test_table VALUES(a);
COMMIT;
END;
/
CREATE PROCEDURE
SELECT PKG_SERVICE.JOB_SUBMIT(NULL, 'call pro_xxx();', to_date('20180101','yyyymmdd'),'sysdate
+1');
 job_submit
------------
      28269
(1 row)
SELECT PKG_SERVICE.JOB_SUBMIT(NULL, 'call pro_xxx();', to_date('20180101','yyyymmdd'),'sysdate
+1.0/24');
 job_submit
------------
       1506
(1 row)

CALL PKG_SERVICE.JOB_SUBMIT(NULL, 'INSERT INTO T_JOB  VALUES(1);  call pro_1(); call pro_2();',
add_months(to_date('201701','yyyymm'),1), 'date_trunc(''day'',SYSDATE) + 1 +(8*60+30.0)/
(24*60)' ,:jobid);
  job
-------
 14131
(1 row)

SELECT PKG_SERVICE.JOB_SUBMIT (101, 'insert_msg_statistic1;', sysdate, 'sysdate+3.0/24');
 job_submit
------------
       101
(1 row)
```

- PKG_SERVICE.JOB_UPDATE

  存储过程UPDATE修改定时任务的属性，包括任务内容、下次执行时间、执行间隔。

  PKG_SERVICE.JOB_UPDATE函数原型为：

```
PKG_SERVICE.JOB_UPDATE(
id             IN   BIGINT,
next_time      IN   TIMESTAMP,
```



```
interval_time  IN    TEXT,
content        IN    TEXT);
```

表 10-14 PKG_SERVICE.JOB_UPDATE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	integer	IN	否	指定的作业号。
next_time	timestamp	IN	是	下次运行时间。如果该参数为空值，则不更新指定job的next_time值，否则更新指定job的next_time值。
interval_time	text	IN	是	用来计算下次作业运行时间的时间表达式。如果该参数为空值，则不更新指定job的interval_time值；如果该参数不为空值，会校验interval_time是否为有效的时间类型或interval类型，则更新指定job的interval_time值。如果为字符串"null"表示只执行一次，执行后JOB状态STATUS变成'd' 不再执行。
content	text	IN	是	执行的存储过程名或者sql语句块。如果该参数为空值，则不更新指定job的content值，否则更新指定job的content值。

示例：

```
CALL PKG_SERVICE.JOB_UPDATE(101, 'call userproc();', sysdate, 'sysdate + 1.0/1440');
 job_update
------------
(1 row)
CALL PKG_SERVICE.JOB_UPDATE(101, 'insert into tbl_a values(sysdate);', sysdate, 'sysdate + 1.0/1440');
 job_update
------------
(1 row)
```

- PKG_SERVICE.SUBMIT_ON_NODES

  存储过程SUBMIT_ON_NODES创建一个所有CN/DN上的定时任务，仅sysadmin/monitor admin有此权限。

  PKG_SERVICE.SUBMIT_ON_NODES函数原型为：

  ```
  PKG_SERVICE.SUBMIT_ON_NODES(
  node_name    IN   NAME,
  database     IN   NAME,
  what         IN   TEXT,
  next_date    IN   TIMESTAMP WITHOUT TIME ZONE,
  job_interval IN   TEXT,
  job          OUT  INTEGER);
  ```


表 10-15 PKG_SERVICE.SUBMIT_ON_NODES 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
node_name	text	IN	否	指定作业的执行节点，当前仅支持值为'ALL_NODE'（在所有节点执行）与'CCN'（在central coordinator执行）。
database	text	IN	否	集群作业所使用的database，节点类型为'ALL_NODE'时仅支持值为'postgres'。
what	text	IN	否	要执行的SQL语句。支持一个或多个‘DML’，‘匿名块’，‘调用存储过程的语句’或3种混合的场景。
nextdate	timestamp	IN	否	下次作业运行时间。默认值为当前系统时间（sysdate）。如果是过去时间，在提交作业时表示立即执行。
job_interval	text	IN	否	用来计算下次作业运行时间的时间表达式，可以是interval表达式，也可以是sysdate加上一个numeric值（例如：sysdate+1.0/24）。如果为空值或字符串"null"表示只执行一次，执行后JOB状态STATUS变成'd'不再执行。
job	integer	OUT	否	作业号。范围为1～32767。当使用select调用dbms.submit_on_nodes时，该参数可以省略。

示例：

```
select pkg_service.submit_on_nodes('ALL_NODE', 'postgres', 'select
capture_view_to_json(''dbe_perf.statement'', 0);', sysdate, 'interval ''60 second''');
 submit_on_nodes
-----------------
           25376
(1 row)
select pkg_service.submit_on_nodes('CCN', 'postgres', 'select
capture_view_to_json(''dbe_perf.statement'', 0);', sysdate, 'interval ''60 second''');
 submit_on_nodes
-----------------
            4973
(1 row)
```

- PKG_SERVICE.ISUBMIT_ON_NODES

  ISUBMIT_ON_NODES与SUBMIT_ON_NODES语法功能相同，但其第一个参数是入参，即指定的作业号，SUBMIT最后一个参数是出参，表示系统自动生成的作业号。仅sysadmin/monitor admin有此权限。

- PKG_SERVICE.SQL_GET_ARRAY_RESULT

  该函数用来返回绑定的数组类型的OUT参数的值，可以用来获取存储过程中的OUT参数。

  PKG_SERVICE.SQL_GET_ARRAY_RESULT函数原型为：

```
PKG_SERVICE.SQL_GET_ARRAY_RESULT(
    context_id in int,
```



```
    pos in VARCHAR2,
    column_value inout anyarray,
    result_type in anyelement
);
```

表 10-16 PKG_SERVICE.SQL_GET_ARRAY_RESULT 接口说明

参数名称	描述
context_id	想查找的CONTEXT ID号。
pos	绑定的参数名。
column_value	返回值。
result_type	返回值类型。

- PKG_SERVICE.SQL_GET_VARIABLE_RESULT

  该函数用来返回绑定的非数组类型的OUT参数的值，可以用来获取存储过程中的OUT参数。

  PKG_SERVICE.SQL_GET_VARIABLE_RESULT函数原型为：

```
PKG_SERVICE.SQL_GET_VARIABLE_RESULT(
    context_id in int,
    pos in VARCHAR2,
    result_type in anyelement
)
RETURNS anyelement;
```

表 10-17 PKG_SERVICE.SQL_GET_VARIABLE_RESULT 接口说明

参数名称	描述
context_id	想查找的CONTEXT ID号。
pos	绑定的参数名。
result_type	返回值类型。

10.12.1.2 PKG_UTIL

PKG_UTIL支持的所有接口请参见**表10-18**：

表 10-18 PKG_UTIL

接口名称	描述
PKG_UTIL.LOB_GET_LENGTH	获取lob的长度。
PKG_UTIL.LOB_READ	读取lob对象的一部分。
PKG_UTIL.LOB_WRITE	将源对象按照指定格式写入到目标对象。
PKG_UTIL.LOB_APPEND	将lob源对象指定个数的字符追加到目标lob对象。

接口名称	描述
PKG_UTIL.LOB_COMPARE	根据指定长度比较两个lob对象。
PKG_UTIL.LOB_MATCH	返回一个字符串在LOB中第N次出现的位置。
PKG_UTIL.LOB_RESET	将lob的指定位置重置为指定字符。
PKG_UTIL.IO_PRINT	将字符串打印输出。
PKG_UTIL.RAW_GET_LENGTH	获取raw的长度。
PKG_UTIL.RAW_CAST_FROM_VARCHAR2	将varchar2转化为raw。
PKG_UTIL.RAW_CAST_FROM_BINARY_INTEGER	将binary integer转化为raw。
PKG_UTIL.RAW_CAST_TO_BINARY_INTEGER	将raw转化为binary integer。
PKG_UTIL.RANDOM_SET_SEED	设置随机种子。
PKG_UTIL.RANDOM_GET_VALUE	返回随机值。
PKG_UTIL.FILE_SET_DIRNAME	设置当前操作的目录。
PKG_UTIL.FILE_OPEN	根据指定文件名和设置的目录打开一个文件。
PKG_UTIL.FILE_SET_MAX_LINE_SIZE	设置写入文件一行的最大长度。
PKG_UTIL.FILE_IS_CLOSE	检测一个文件句柄是否关闭。
PKG_UTIL.FILE_READ	从一个打开的文件句柄中读取指定长度的数据。
PKG_UTIL.FILE_READLINE	从一个打开的文件句柄中读取一行数据。
PKG_UTIL.FILE_WRITE	将BUFFER中的数据写入到文件中。
PKG_UTIL.FILE_WRITELINE	将BUFFER写入文件，并追加换行符。
PKG_UTIL.FILE_NEWLINE	新起一行。
PKG_UTIL.FILE_READ_RAW	从一个打开的文件句柄中读取指定长度的二进制数据。
PKG_UTIL.FILE_WRITE_RAW	将二进制数据写入到文件中。
PKG_UTIL.FILE_FLUSH	将一个文件句柄中的数据写入到物理文件中。
PKG_UTIL.FILE_CLOSE	关闭一个打开的文件句柄。
PKG_UTIL.FILE_REMOVE	删除一个物理文件，操作需要有对应权限。

接口名称	描述
PKG_UTIL.FILE_RENAME	对于磁盘上的文件进行重命名，类似UNIX的mv。
PKG_UTIL.FILE_SIZE	返回文件大小。
PKG_UTIL.FILE_BLOCK_SIZE	返回文件含有的块数量。
PKG_UTIL.FILE_EXISTS	判断文件是否存在。
PKG_UTIL.FILE_GETPOS	返回文件的偏移量，单位字节。
PKG_UTIL.FILE_SEEK	设置文件位置为指定偏移。
PKG_UTIL.FILE_CLOSE_ALL	关闭一个会话中打开的所有文件句柄。
PKG_UTIL.EXCEPTION_REPORT_ERROR	抛出一个异常。

- PKG_UTIL.LOB_GET_LENGTH

  该函数LOB_GET_LENGTH获取输入数据的长度。

  PKG_UTIL.LOB_GET_LENGTH函数原型为：

  ```
  PKG_UTIL.LOB_GET_LENGTH(
  lob       IN    anyelement
  )
  RETURN INTEGER;
  ```

  表 10-19 PKG_UTIL.LOB_GET_LENGTH 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
lob	clob / blob	IN	否	待获取长度的对象。

- PKG_UTIL.LOB_READ

  该函数LOB_READ读取一个对象，并返回指定部分。

  PKG_UTIL.LOB_READ函数原型为：

  ```
  PKG_UTIL.LOB_READ(
  lob       IN    anyelement,
  len       IN    int,
  start     IN    int,
  mode      IN    int
  )
  RETURN ANYELEMENT
  ```

表 10-20 PKG_UTIL.LOB_READ 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
lob	clob/blob	IN	否	clob或者blob类型数据。
len	int	IN	否	返回结果长度。
start	int	IN	否	相较于第一个字符的偏移量。
mode	int	IN	否	判断读取操作的类型， 0： read; 1 : trim; 2 : substr。

- PKG_UTIL.LOB_WRITE

  该函数LOB_WRITE将源对象按照指定的参数写入目标对象，并返回目标对象。

  PKG_UTIL.LOB_WRITE函数原型为：

```
PKG_UTIL.LOB_WRITE(
dest_lob    INOUT   blob,
src_lob     IN      raw
len         IN      int,
start_pos   IN      int
)
RETURN BLOB;
PKG_UTIL.LOB_WRITE(
dest_lob    INOUT   clob,
src_lob     IN      varchar2
len         IN      int,
start_pos   IN      int
)
RETURN CLOB;
```

表 10-21 PKG_UTIL.LOB_WRITE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
dest_lob	clob/blob	INOUT	否	写入的目标对象。
src_lob	clob/blob	IN	否	被写入的源对象。
len	int	IN	否	源对象的写入长度。
start_pos	int	IN	否	目标对象的写入起始位置。

- PKG_UTIL.LOB_APPEND

  该函数LOB_APPEND将源blob/clob对象追加到目标blob/clob对象, 并返回目标对象。

  PKG_UTIL.LOB_APPEND函数原型为：

```
PKG_UTIL.LOB_APPEND(
dest_lob    INOUT   blob,
```



```
src_lob     IN      blob,
len         IN      int default NULL
)
RETURN BLOB;

PKG_UTIL.LOB_APPEND(
dest_lob    INOUT   clob,
src_lob     IN      clob,
len         IN      int default NULL
)
RETURN CLOB;
```

表 10-22 PKG_UTIL.LOB_APPEND 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
dest_lob	blob / clob	INOUT	否	写入的目标blob/clob对象。
src_lob	blob / clob	IN	否	被写入的源blob/clob对象。
len	int	IN	是	写入源对象的长度，为NULL则默认写入源对象全部。

- PKG_UTIL.LOB_COMPARE

  该函数LOB_COMPARE按照指定的起始位置、个数比较对象是否相同，lob1大则返回1，lob2大返回-1，相等返回0。

  PKG_UTIL.LOB_COMPARE函数原型为：

```
PKG_UTIL.LOB_COMPARE(
lob1        IN   anyelement,
lob2        IN   anyelement,
len         IN   int default 1073741771,
start_pos1      IN   int default 1,
start_pos2      IN   int default 1
)
RETURN INTEGER;
```

表 10-23 PKG_UTIL.LOB_COMPARE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
lob1	clob/ blob	IN	否	待比较的字符串。
lob2	clob/ blob	IN	否	待比较的字符串。
len	int	IN	否	比较的长度。
start_pos1	int	IN	否	lob1起始偏移量。

参数	类型	入参/出参	是否可以为空	描述
start_pos2	int	IN	否	lob2起始偏移量。

- PKG_UTIL.LOB_MATCH

  该函数LOB_MATCH返回pattern出现在lob对象中第match_nth次的位置。

  PKG_UTIL.LOB_MATCH函数原型为：

```
PKG_UTIL.LOB_MATCH(
lob          IN   anyelement,
pattern      IN   anyelement,
start        IN   int,
match_nth    IN   int default 1
)
RETURN INTEGER;
```

表 10-24 PKG_UTIL.LOB_MATCH 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
lob	clob/blob	IN	否	待比较的字符串。
pattern	clob/blob	IN	否	待匹配的pattern。
start	int	IN	否	lob的起始比较位置。
match_nth	int	IN	否	第几次匹配到。

- PKG_UTIL.LOB_RESET

  该函数LOB_RESET清除一段数据为字符value。

  PKG_UTIL.LOB_RESET函数原型为：

```
PKG_UTIL.LOB_RESET(
lob          IN   blob,
len          IN   int,
start        IN   int,
value         IN   int default 0
)
RETURN record;
```

表 10-25 PKG_UTIL.LOB_RESET 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
lob	blob	IN	否	待重置的字符串。

参数	类型	入参/出参	是否可以为空	描述
len	int	IN	否	重置的长度。
start	int	IN	否	重置的起始位置。
value	int	IN	是	设置的字符。默认值‘0’。

- PKG_UTIL.IO_PRINT

  该函数IO_PRINT将一段字符串打印输出。

  PKG_UTIL.IO_PRINT函数原型为：

```
PKG_UTIL.IO_PRINT(
format       IN   text,
is_one_line  IN   boolean
)
RETURN void;
```

  表 10-26 PKG_UTIL.IO_PRINT 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
format	text	IN	否	待打印输出的字符串。
is_one_line	boolean	IN	否	是否输出为一行。

- PKG_UTIL.RAW_GET_LENGTH

  该函数RAW_GET_LENGTH获取raw的长度。

  PKG_UTIL.RAW_GET_LENGTH函数原型为：

```
PKG_UTIL.RAW_GET_LENGTH(
value       IN   raw
)
RETURN integer;
```

  表 10-27 PKG_UTIL.RAW_GET_LENGTH 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
raw	raw	IN	否	待获取长度的对象。

- PKG_UTIL.RAW_CAST_FROM_VARCHAR2

  该函数RAW_CAST_FROM_VARCHAR2将varchar2转化为raw。

  PKG_UTIL.RAW_CAST_FROM_VARCHAR2函数原型为：

```
PKG_UTIL.RAW_CAST_FROM_VARCHAR2(
str       IN   varchar2
)
RETURN raw;
```


表 10-28 PKG_UTIL.RAW_CAST_FROM_VARCHAR2 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
str	varchar2	IN	否	需要转化的源数据。

- PKG_UTIL.RAW_CAST_FROM_BINARY_INTEGER

  该函数RAW_CAST_FROM_BINARY_INTEGER将BIGINT转化为RAW。

  PKG_UTIL.RAW_CAST_FROM_BINARY_INTEGER函数原型为：

```
PKG_UTIL.RAW_CAST_FROM_BINARY_INTEGER(
value       IN   BIGINT,
endianess   IN   INTEGER
)
RETURN RAW;
```

表 10-29 PKG_UTIL.RAW_CAST_FROM_BINARY_INTEGER 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
value	BIGINT	IN	否	需要转化的源数据。
endianess	INTEGER	IN	否	表示字典序的整型值，当前支持1或2（1代表BIG_ENDIAN，2代表LITTLE_ENDIAN）。

- PKG_UTIL.RAW_CAST_TO_BINARY_INTEGER

  该函数RAW_CAST_TO_BINARY_INTEGER将RAW转化为BINARY_INTEGER。

  PKG_UTIL.RAW_CAST_TO_BINARY_INTEGER函数原型为：

```
PKG_UTIL.RAW_CAST_TO_BINARY_INTEGER(
value       IN   RAW,
endianess   IN   INTEGER
)
RETURN INTEGER;
```

表 10-30 PKG_UTIL.RAW_CAST_TO_BINARY_INTEGER 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
value	RAW	IN	否	需要转化的源数据。
endianess	INTEGER	IN	否	表示字典序的整型值，当前支持1或2（1代表BIG_ENDIAN，2代表LITTLE_ENDIAN）。

- PKG_UTIL.RANDOM_SET_SEED

该函数RANDOM_SET_SEED设置随机数种子。

PKG_UTIL.RANDOM_SET_SEED函数原型为：

```
PKG_UTIL.RANDOM_SET_SEED(
seed        IN   int
)
RETURN integer;
```

表 10-31 PKG_UTIL.RANDOM_SET_SEED 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
seed	int	IN	否	随机数种子。

- PKG_UTIL.RANDOM_GET_VALUE

  该函数RANDOM_GET_VALUE返回0~1区间的一个随机数，其有效数字为15位。

  PKG_UTIL.RANDOM_GET_VALUE函数原型为：

  ```
  PKG_UTIL.RANDOM_GET_VALUE(
  )
  RETURN numeric;
  ```

- PKG_UTIL.FILE_SET_DIRNAME

  设置当前操作的目录，基本上所有涉及单个目录的操作，都需要调用此方法先设置操作的目录。

  PKG_UTIL.FILE_SET_DIRNAME函数原型为：

  ```
  PKG_UTIL.FILE_SET_DIRNAME(
  dir  IN  text
  )
  RETURN bool
  ```

  表 10-32 PKG_UTIL.FILE_SET_DIRNAME 接口参数说明

参数	描述
dirname	文件的目录位置，这个字符串是一个目录对象名。 **说明** 文件目录的位置，需要添加到系统表**PG_DIRECTORY**中，如果传入的路径和**PG_DIRECTORY**中的路径不匹配，会报路径不存在的错误， 下面的涉及location作为参数的函数也是同样的情况。

- PKG_UTIL.FILE_OPEN

  该函数用来打开一个文件，最多可以同时打开50个文件。并且该函数返回INTEGER类型的一个句柄。

  PKG_UTIL.FILE_OPEN函数原型为：

  ```
  PKG_UTIL.FILE_OPEN(
  file_name    IN  text,
  open_mode    IN  integer)
  ```


表 10-33 PKG_UTIL.FILE_OPEN 接口参数说明

参数	描述
file_name	文件名，包含扩展（文件类型），不包括路径名。如果文件名中包含路径，在OPEN中会被忽略，在UNIX系统中，文件名不能以/.结尾。
open_mode	指定文件的打开模式，包含r：read text，w: write text和a: append text。 **说明** 对于写操作，会检测文件类型，如果写入elf文件，将会报错并退出。

- PKG_UTIL.FILE_SET_MAX_LINE_SIZE

  设置写入文件一行的最大长度。

  PKG_UTIL.FILE_SET_MAX_LINE_SIZE函数原型为：

  ```
  PKG_UTIL.FILE_SET_MAX_LINE_SIZE(
  max_line_size in integer)
  RETURN BOOL
  ```

  表 10-34 PKG_UTIL.FILE_SET_MAX_LINE_SIZE 接口参数说明

参数	描述
max_line_size	每行最大字符数，包含换行符（最小值是1，最大值是32767）。如果没有指定，会指定一个默认值1024。

- PKG_UTIL.FILE_IS_CLOSE

  检测一个文件句柄是否关闭。

  PKG_UTIL.FILE_IS_CLOSE函数原型为：

  ```
  PKG_UTIL.FILE_IS_CLOSE(
  file in integer
  )
  RETURN BOOL
  ```

  表 10-35 PKG_UTIL.FILE_IS_CLOSE 接口参数说明

参数	描述
file	一个打开的文件句柄。

- PKG_UTIL.FILE_READ

  根据指定的长度从一个打开的文件句柄中读取出数据。

  PKG_UTIL.FILE_READ函数原型为：

  ```
  PKG_UTIL.FILE_READ(
  file     IN   integer,
  buffer   OUT  text,
  len      IN   bigint default 1024)
  ```


表 10-36 PKG_UTIL.FILE_READ 接口参数说明

参数	描述
file	通过调用OPEN打开的文件句柄，文件必须以读的模式打开，否则会抛出INVALID_OPERATION的异常。
buffer	用于接收数据的BUFFER。
len	从文件中读取的字节数。

- PKG_UTIL.FILE_READLINE

  根据指定的长度从一个打开的文件句柄中读取出一行数据。

  PKG_UTIL.FILE_READLINE函数原型为：

```
PKG_UTIL.FILE_READLINE(
file    IN  integer,
buffer  OUT text,
len     IN  integer default NULL)
```

表 10-37 PKG_UTIL.FILE_READLINE 接口参数说明

参数	描述
file	通过调用OPEN打开的文件句柄，文件必须以读的模式打开，否则会抛出INVALID_OPERATION的异常。
buffer	用于接收数据的BUFFER。
len	从文件中读取的字节数，默认是NULL。如果是默认NULL，会使用max_line_size来指定大小。

- PKG_UTIL.FILE_WRITE

  将BUFFER中指定的数据写入到文件中。

  PKG_UTIL.FILE_WRITE函数原型为：

```
PKG_UTIL.FILE_WRITE(
file in integer,
buffer in text
)
RETURN BOOL
```

表 10-38 PKG_UTIL.FILE_WRITE 接口参数说明

参数	描述
file	一个打开的文件句柄。
buffer	要写入文件的文本数据，BUFFER的最大值是32767个字节。如果没有指定值，默认是1024个字节，没有刷新到文件之前，一系列的PUT操作的BUFFER总和不能超过32767个字节。 **说明** 对于写操作，会检测文件类型，如果写入elf文件，将会报错并退出。

- PKG_UTIL.FILE_NEWLINE

向一个打开的文件中写入一个行终结符。行终结符和平台相关。

PKG_UTIL.FILE_NEWLINE函数原型为：

```
PKG_UTIL.FILE_NEWLINE(
file in integer
)
RETURN BOOL
```

表 10-39 PKG_UTIL.FILE_NEWLINE 接口参数说明

参数	描述
file	一个打开的文件句柄。

- PKG_UTIL.FILE_WRITELINE

  向一个打开的文件中写入一行。

  PKG_UTIL.FILE_WRITELINE函数原型为：

  ```
  PKG_UTIL.FILE_WRITELINE(
  file in integer,
  buffer in text
  )
  RETURN BOOL
  ```

  表 10-40 PKG_UTIL.FILE_WRITELINE 接口参数说明

参数	描述
file	一个打开的文件句柄。
buffer	要写入的内容

- PKG_UTIL.FILE_READ_RAW

  从一个打开的文件句柄中读取指定长度的二进制数据，返回读取的二进制数据，返回类型为raw。

  PKG_UTIL.FILE_READ_RAW函数原型为：

  ```
  PKG_UTIL.FILE_READ_RAW(
  file      in integer,
  length    in integer default NULL
  )
  RETURN raw
  ```

  表 10-41 PKG_UTIL.FILE_READ_RAW 接口参数说明

参数	描述
file	一个打开的文件句柄。
length	要读取的长度，默认为NULL。默认情况下读取文件中所有数据，最大为1G。

- PKG_UTIL.FILE_WRITE_RAW

  向一个打开的文件中写入传入二进制对象RAW。插入成功返回true。

  PKG_UTIL.FILE_WRITE_RAW函数原型为：


```
PKG_UTIL.FILE_WRITE_RAW(
file in integer,
r    in raw
)
RETURN BOOL
```

表 10-42 PKG_UTIL.FILE_WRITE_RAW 接口参数说明

参数	描述
file	一个打开的文件句柄。
r	准备传入文件的数据 **说明** 对于写操作，会检测文件类型，如果写入elf文件，将会报错并退出。

- PKG_UTIL.FILE_FLUSH

  一个文件句柄中的数据要写入到物理文件中，缓冲区中的数据必须要有一个行终结符。当文件必须在打开时读取，刷新非常有用。例如，调试信息可以刷新到文件中，以便立即读取。

  PKG_UTIL.FILE_FLUSH函数原型为：

  ```
  PKG_UTIL.FILE_FLUSH（
  file in integer
  ）
  RETURN VOID
  ```

  表 10-43 PKG_UTIL.FILE_FLUSH 接口参数说明

参数	描述
file	一个打开的文件句柄。

- PKG_UTIL.FILE_CLOSE

  关闭一个打开的文件句柄。

  PKG_UTIL.FILE_CLOSE函数原型为：

  ```
  PKG_UTIL.FILE_CLOSE（
  file in integer
  ）
  RETURN BOOL
  ```

  表 10-44 PKG_UTIL.FILE_CLOSE 接口参数说明

参数	描述
file	一个打开的文件句柄。

- PKG_UTIL.FILE_REMOVE

  删除一个磁盘文件，操作的时候需要有充分的权限。

  PKG_UTIL.FILE_REMOVE函数原型为：

  ```
  PKG_UTIL.FILE_REMOVE(
  file_name in text
  )
  RETURN VOID
  ```


表 10-45 PKG_UTIL.FILE_REMOVE 接口参数说明

参数	描述
filen_ame	要删除的文件名

- PKG_UTIL.FILE_RENAME

  对于磁盘上的文件进行重命名，类似UNIX的mv。

  PKG_UTIL.FILE_RENAME函数原型为：

```
PKG_UTIL.FILE_RENAME(
src_dir in text,
src_file_name in text,
dest_dir in text,
dest_file_name in text,
overwrite boolean default false)
```

表 10-46 PKG_UTIL.FILE_RENAME 接口参数说明

参数	描述
src_dir	源文件目录（大小写敏感）。 **说明** • 文件目录的位置，需要添加到系统表**PG_DIRECTORY**中，如果传入的路径和**PG_DIRECTORY**中的路径不匹配，会报路径不存在的错误，下面的涉及location作为参数的函数也是同样的情况。 • 在打开guc参数safe_data_path时，用户只能通过高级包读写safe_data_path指定文件路径下的文件。
src_file_name	源文件名。
dest_dir	目标文件目录（大小写敏感）。 **说明** • 文件目录的位置，需要添加到系统表**PG_DIRECTORY**中，如果传入的路径和**PG_DIRECTORY**中的路径不匹配，会报路径不存在的错误，下面的涉及location作为参数的函数也是同样的情况。 • 在打开guc参数safe_data_path时，用户只能通过高级包读写safe_data_path指定文件路径下的文件。
dest_file_name	目标文件名。
overwrite	默认是false，如果目的目录下存在一个同名的文件，不会进行重写。

- PKG_UTIL.FILE_SIZE

  返回指定的文件大小。

  PKG_UTIL.FILE_SIZE函数原型为：

```
bigint PKG_UTIL.FILE_SIZE(
file_name in text
)return bigint
```


表 10-47 PKG_UTIL.FILE_SIZE 接口参数说明

参数	描述
file_name	文件名

- PKG_UTIL.FILE_BLOCK_SIZE
  返回指定的文件含有的块数量。
  PKG_UTIL.FILE_BLOCK_SIZE函数原型为：

```
bigint PKG_UTIL.FILE_BLOCK_SIZE(
file_name in text
)return bigint
```

表 10-48 PKG_UTIL.FILE_BLOCK_SIZE 接口参数说明

参数	描述
file_name	文件名

- PKG_UTIL.FILE_EXISTS
  判断指定的文件是否存在。
  PKG_UTIL.FILE_EXISTS函数原型为：

```
PKG_UTIL.FILE_EXISTS(
file_name in text
)
RETURN BOOL
```

表 10-49 PKG_UTIL.FILE_EXISTS 接口参数说明

参数	描述
file_name	文件名

- PKG_UTIL.FILE_GETPOS
  返回文件的偏移量，单位字节。
  PKG_UTIL.FILE_GETPOS函数原型为：

```
PKG_UTIL.FILE_GETPOS(
file in integer
)
RETURN BIGINT
```

表 10-50 PKG_UTIL.FILE_GETPOS 接口参数说明

参数	描述
file	一个打开的文件句柄。

- PKG_UTIL.FILE_SEEK
  根据用户指定的字节数向前或者向后调整文件指针的位置。
  PKG_UTIL.FILE_SEEK函数原型为：


```
void PKG_UTIL.FILE_SEEK(
file in integer,
start in bigint
)
RETURN VOID
```

表 10-51 PKG_UTIL.FILE_SEEK 接口参数说明

参数	描述
file	一个打开的文件句柄。
start	文件偏移，字节。

- PKG_UTIL.FILE_CLOSE_ALL

  关闭一个会话中打开的所有的文件句柄。

  PKG_UTIL.FILE_CLOSE_ALL函数原型为：

  ```
  PKG_UTIL.FILE_CLOSE_ALL(
  )
  RETURN VOID↵
  ```

  表 10-52 PKG_UTIL.FILE_CLOSE_ALL 接口参数说明

参数	描述
无	无

- PKG_UTIL.EXCEPTION_REPORT_ERROR

  抛出一个异常。

  PKG_UTIL.EXCEPTION_REPORT_ERROR函数原型为：

  ```
  PKG_UTIL.EXCEPTION_REPORT_ERROR(
  code integer,
  log text,
  flag boolean DEFAULT false
  )
  RETURN INTEGER
  ```

  表 10-53 PKG_UTIL.EXCEPTION_REPORT_ERROR 接口参数说明

参数	描述
code	抛出异常所打印的错误码。
log	抛出异常所打印的日志提示信息。
flag	保留字段，默认为false。

- PKG_UTIL.app_read_client_info

  读取client_info信息

  PKG_UTIL.app_read_client_info函数原型为：

  ```
  PKG_UTIL.app_read_client_info(
  OUT buffer text
  )return text
  ```


表 10-54 PKG_UTIL.app_read_client_info 接口参数说明

参数	描述
buffer	返回的client_info信息

- PKG_UTIL.app_set_client_info
  设置client_info信息
  PKG_UTIL.app_set_client_info函数原型为：

```
PKG_UTIL.app_set_client_info(
str text
)
```

表 10-55 PKG_UTIL.app_set_client_info 接口参数说明

参数	描述
str	要设置的client_info信息

- PKG_UTIL.lob_converttoblob
  将clob转成blob，amount为要转换的长度
  PKG_UTIL.lob_converttoblob函数原型为：

```
PKG_UTIL.lob_converttoblob(
dest_lob blob,
src_clob clob,
amount integer,
dest_offset integer,
src_offset integer
)return raw
```

表 10-56 PKG_UTIL.lob_converttoblob 接口参数说明

参数	描述
dest_lob	目标lob
src_clob	要转换的clob
amount	转换的长度
dest_offset	目标lob的起始位置
src_offset	源clob的起始位置

- PKG_UTIL.lob_converttoclob
  将blob转成clob，amount为要转换的长度
  PKG_UTIL.lob_converttoclob函数原型为：

```
PKG_UTIL.lob_converttoclob(
dest_lob clob,
src_blob blob,
amount integer,
dest_offset integer,
src_offset integer
)return text
```


表 10-57 PKG_UTIL.lob_converttoclob 接口参数说明

参数	描述
dest_lob	目标lob
src_blob	要转换的blob
amount	转换的长度
dest_offset	目标lob的起始位置
src_offset	源clob的起始位置

- PKG_UTIL.lob_texttoraw

  将text转成raw

  PKG_UTIL.lob_texttoraw函数原型为：

  ```
  PKG_UTIL.lob_texttoraw(
  src_lob clob
  )
  RETURN raw
  ```

  表 10-58 PKG_UTIL.lob_texttoraw 接口参数说明

参数	描述
src_lob	要转换的lob数据

- PKG_UTIL.match_edit_distance_similarity

  计算两个字符串的差别

  PKG_UTIL.match_edit_distance_similarity函数原型为：

  ```
  PKG_UTIL.match_edit_distance_similarity(
  str1 text,
  str2 text
  )
  RETURN INTEGER
  ```

  表 10-59 PKG_UTIL.match_edit_distance_similarity 接口参数说明

参数	描述
str1	第一个字符串
str2	第二个字符串

- PKG_UTIL.raw_cast_to_varchar2

  raw类型转成varchar2。

  PKG_UTIL.raw_cast_to_varchar2函数原型为：

  ```
  PKG_UTIL.raw_cast_to_varchar2(
  str raw
  )
  RETURN varchar2
  ```


表 10-60 PKG_UTIL.raw_cast_to_varchar2 接口参数说明

参数	描述
str	十六进制字符串

- PKG_UTIL.session_clear_context

  清除session_context信息

  PKG_UTIL.session_clear_context函数原型为：

```
PKG_UTIL.session_clear_context(
namespace text,
client_identifier text,
attribute text
)
RETURN INTEGER
```

表 10-61 PKG_UTIL.session_clear_context 接口参数说明

参数	描述
namespace	属性的命名空间
client_identifier	client_identifier，一般与namespace即可，当为null时，默认修改所有namesapce
attribute	要清除的属性值

- PKG_UTIL.session_search_context

  查找属性值

  PKG_UTIL.session_clear_context函数原型为：

```
PKG_UTIL.session_clear_context(
namespace text,
attribute text
)
RETURN INTEGER
```

表 10-62 PKG_UTIL.session_clear_context 接口参数说明

参数	描述
namespace	属性的命名空间
attribute	要清除的属性值

- PKG_UTIL.session_set_context

  设置属性值

  PKG_UTIL.session_set_context函数原型为：

```
PKG_UTIL.session_set_context(
namespace text,
attribute text,
value text
)
RETURN INTEGER
```


表 10-63 PKG_UTIL.session_set_context 接口参数说明

参数	描述
namespace	属性的命名空间
attribute	要设置的属性
value	属性对应的值

- PKG_UTIL.utility_get_time
  打印UNIX时间戳。
  PKG_UTIL.utility_get_time函数原型为：
  ```
  PKG_UTIL.utility_get_time()
  RETURN bigint
  ```
- PKG_UTIL.utility_format_error_backtrace
  查看存储过程的错误堆栈。
  PKG_UTIL.utility_format_error_backtrace函数原型为：
  ```
  PKG_UTIL.utility_format_error_backtrace()
  RETURN text
  ```
- PKG_UTIL.utility_format_error_stack
  查看存储过程的报错信息。
  PKG_UTIL.utility_format_error_stack函数原型为：
  ```
  PKG_UTIL.utility_format_error_stack()
  RETURN text
  ```
- PKG_UTIL.utility_format_call_stack
  查看存储过程调用堆栈。
  PKG_UTIL.utility_format_call_stack函数原型为：
  ```
  PKG_UTIL.utility_format_call_stack()
  RETURN text
  ```

10.12.2 二次封装接口(推荐)

10.12.2.1 DBE_LOB

接口介绍

高级功能包DBE_LOB支持的所有接口参见表10-64。

表 10-64 DBE_LOB

接口名称	描述
DBE_LOB.GET_LENGTH	获取并返回指定的LOB类型对象的长度。
DBE_LOB.OPEN	打开一个LOB返回一个LOB的描述符。
DBE_LOB.READ	根据指定的长度及起始位置偏移读取LOB内容的一部分到BUFFER缓冲区。

接口名称	描述
DBE_LOB.WRITE	根据指定长度及起始位置偏移将BUFFER中内容写入到LOB中。
DBE_LOB.WRITE_APPEND	根据指定长度将BUFFER中内容写入到LOB的尾部。
DBE_LOB.COPY	根据指定长度及起始位置偏移将BLOB内容写入到另一个BLOB中。
DBE_LOB.ERASE	根据指定长度及起始位置偏移删除BLOB中的内容。
DBE_LOB.CLOSE	关闭已经打开的LOB描述符。
DBE_LOB.MATCH	返回一个字符串在LOB中第N次出现的位置。
DBE_LOB.COMPARE	比较两个LOB或者两个LOB的某一部分。
DBE_LOB.SUBSTR	用于读取一个LOB的子串，返回读取的字节个数或者字符个数。
DBE_LOB.STRIP	用于截断指定长度的LOB，执行完会将LOB的长度设置为参数指定的长度。
DBE_LOB.CREATE_TEMPORARY	创建一个临时的BLOB或者CLOB对象。
DBE_LOB.FREETEMPORARY	删除一个临时的BLOB或者CLOB对象。
DBE_LOB.APPEND	将源LOB的内容拼接到目的LOB中。
DBE_LOB.FILEOPEN	打开一个数据库外部文件，并返回文件表述符。
DBE_LOB.FILECLOSE	关闭由FILEOPEN打开的外部文件。
DBE_LOB.LOADFROMFILE	读取数据库外部文件到BLOB文件中。
DBE_LOB.LOADBLOBFROMFILE	读取数据库外部文件到BLOB文件中。
DBE_LOB.LOADCLOBFROMFILE	读取数据库外部文件到CLOB文件中。
DBE_LOB.CONVERTTOBLOB	将CLOB类型文件转换为BLOB类型文件。
DBE_LOB.CONVERTTOCLOB	将BLOB类型文件转换为CLOB类型文件。

- DBE_LOB.GET_LENGTH
  存储过程GET_LENGTH获取并返回指定的LOB类型对象的长度。
  DBE_LOB.GET_LENGTH函数原型为：

```
DBE_LOB.GET_LENGTH (
lob    IN   BLOB)
RETURN INTEGER;
```



```
DBE_LOB.GET_LENGTH (
lob    IN   CLOB)
RETURN INTEGER;
```

表 10-65 DBE_LOB.GET_LENGTH 接口参数说明

参数	描述
lob	待获取长度的BLOB/CLOB类型对象。

- DBE_LOB.OPEN

  存储过程打开一个LOB，并返回一个LOB描述符，该过程无实际意义，仅用于兼容。

  DBE_LOB.OPEN函数原型为：

```
DBE_LOB.OPEN (
lob    INOUT   BLOB
);

DBE_LOB.OPEN (
lob  INOUT    CLOB
);

DBE_LOB.OPEN (
bfile dbe_lob.bfile,
open_mode text DEFAULT 'null'::text
);
```

表 10-66 DBE_LOB.OPEN 接口参数说明

参数	描述
lob	被打开的BLOB或者CLOB对象。

- DBE_LOB.READ

  存储过程READ根据指定长度及起始位置偏移读取LOB内容的一部分到BUFFER缓冲区。

  DBE_LOB.READ函数原型为：

```
DBE_LOB.READ (
lob     IN     BLOB,
len     IN     INTEGER,
start   IN     INTEGER,
buffer  OUT    RAW);

DBE_LOB.READ (
lob     IN     CLOB,
len     INOUT  INTEGER,
start   IN     INTEGER,
buffer  OUT    VARCHAR2);
```

表 10-67 DBE_LOB.READ 接口参数说明

参数	说明
lob	待读入的BLOB/CLOB类型对象。

参数	说明
len	读入长度。 **说明** 如果读入长度为负，会收到错误提示“ERROR: argument 2 is null, invalid, or out of range.”
start	指定从LOB内容的哪个位置开始读取的偏移（即相对LOB内容起始位置的字节数）。
buffer	读取LOB内容后存放的目标缓冲区。

- DBE_LOB.WRITE

  存储过程WRITE根据指定长度及起始位置将BUFFER中内容写入到LOB变量中。

  DBE_LOB.WRITE函数原型为：

```
DBE_LOB.WRITE (
dest_lob   INOUT    BLOB,
len        IN       INTEGER,
start      IN       INTEGER,
src_lob    IN       RAW);

DBE_LOB.WRITE (
dest_lob   INOUT   CLOB,
len        IN      INTEGER,
start      IN      INTEGER,
src_lob    IN      VARCHAR2);
```

  表 10-68 DBE_LOB.WRITE 接口参数说明

参数	说明
dest_lob	待写入的BLOB/CLOB类型对象。
len	写入长度，最大支持32767字符。 **说明** 如果写入长度小于1或写入长度大于待写入的内容长度，则报错。
start	指定从LOB内容的哪个位置开始写入的偏移（即相对LOB内容起始位置的字节数）。 **说明** 如果偏移量小于1，则报错；如果偏移量大于LOB类型最大长度时，不会报错。
src_lob	待写入的内容。

- DBE_LOB.WRITE_APPEND

  存储过程WRITE_APPEND根据指定长度将BUFFER中内容写入到LOB的尾部。

  DBE_LOB.WRITE_APPEND函数原型为：

```
DBE_LOB.WRITE_APPEND (
dest_lob    INOUT    BLOB,
len         IN       INTEGER,
src_lob     IN       RAW);

DBE_LOB.WRITE_APPEND (
dest_lob    INOUT     CLOB,
```

```
len         IN        INTEGER,
src_lob     IN        VARCHAR2);
```

表 10-69 DBE_LOB.WRITE_APPEND 接口参数说明

参数	说明
dest_lob	待写入的指定BLOB/CLOB类型对象。
len	写入长度。 **说明** 如果写入长度小于1或写入长度大于待写入的内容长度，则报错。
src_lob	待写入的内容。

- DBE_LOB.COPY

  存储过程COPY根据指定长度及起始位置偏移将BLOB内容拷贝到另一个BLOB中。

  DBE_LOB.COPY函数原型为：

```
DBE_LOB.COPY (
dest_lob     INOUT    BLOB,
src_lob      IN       BLOB,
len          IN       INTEGER,
dest_start   IN       INTEGER  DEFAULT 1,
src_start    IN       INTEGER  DEFAULT 1);
```

表 10-70 DBE_LOB.COPY 接口参数说明

参数	说明
dest_lob	待拷入的BLOB类型对象。
src_lob	待拷出的BLOB类型对象。
len	拷贝长度。 **说明** 如果拷入长度小于1或拷入长度大于BLOB类型最大长度，则报错。
dest_start	指定从BLOB内容的哪个位置开始拷入的偏移（即相对BLOB内容起始位置的字节数）。 **说明** 如果偏移量小于1或偏移量大于BLOB类型最大长度，则报错。
src_start	指定从BLOB内容的哪个位置开始拷出的偏移（即相对BLOB内容起始位置的字节数）。 **说明** 如果偏移量小于1或偏移量大于拷贝来源BLOB的长度，则报错。

- DBE_LOB.ERASE

  存储过程ERASE根据指定长度及起始位置偏移删除BLOB中的内容。

  DBE_LOB.ERASE函数原型为：

```
DBE_LOB.ERASE (
lob       INOUT   BLOB,
len       INOUT   INTEGER,
start     IN      INTEGER DEFAULT 1);
```


表 10-71 DBE_LOB.ERASE 接口参数说明

参数	说明
lob	待删除内容的BLOB类型对象。
len	待删除的长度。 **说明** 如果删除长度小于1或删除长度大于BLOB类型最大长度，则报错。
start	指定从BLOB内容的哪个位置开始删除的偏移（即相对BLOB内容起始位置的字节数）。 **说明** 如果偏移量小于1或偏移量大于BLOB类型最大长度，则报错。

- DBE_LOB.CLOSE

  存储过程CLOSE关闭已经打开的LOB描述符。

  DBE_LOB.CLOSE函数原型为：

```
DBE_LOB.CLOSE(
lob     IN      BLOB);

DBE_LOB.CLOSE (
lob     IN      CLOB);

DBE_LOB.CLOSE (
file    IN      INTEGER);
```

表 10-72 DBE_LOB.CLOSE 接口参数说明

参数	说明
lob	待关闭的BLOB/CLOB类型对象。

- DBE_LOB.MATCH

  该函数返回pattern在LOB中第N次出现的位置，如果输入的是一些无效值会返回NULL值。offset < 1 or offset > LOBMAXSIZE，nth < 1，nth > LOBMAXSIZE。

  DBE_LOB.MATCH函数原型为：

```
DBE_LOB.MATCH (
lob             IN     BLOB,
pattern         IN     RAW,
start_index     IN     INTEGER  DEFAULT 1,
match_index     IN     INTEGER  DEFAULT 1)
RETURN INTEGER;

DBE_LOB.MATCH (
lob             IN     CLOB,
pattern         IN     VARCHAR2 ,
start_index     IN     INTEGER  DEFAULT 1,
match_index     IN     INTEGER  DEFAULT 1)
RETURN INTEGER;
```


表 10-73 DBE_LOB.match 接口参数说明

参数	说明
lob	要查找的BLOB/CLOB描述符。
pattern	要匹配的模式，对于BLOB是由一组RAW类型的数据组成，对于CLOB是由一组text类型的数据组成。
start_index	对于BLOB是以字节为单位的绝对偏移量，对于CLOB是以字符为单位的偏移量，模式匹配的起始位置是1。
match_index	模式匹配的次数，最小值为1。

- DBE_LOB.COMPARE

  这个函数比较两个LOB或者两个LOB的一部分。

  - 如果比较的结果相等返回0，否则返回非零的值。
  - 如果第一个LOB比第二个小，返回-1；如果第一个LOB比第二个大，返回1。
  - 如果len，start1，start2这几个参数有无效参数返回NULL，有效的偏移量范围是1~LOBMAXSIZE。

  DBE_LOB.COMPARE函数原型为：

```
DBE_LOB.COMPARE (
lob1     IN     BLOB,
lob2     IN     BLOB,
len      IN     INTEGER DEFAULT DBE_LOB.LOBMAXSIZE,
start1   IN     INTEGER DEFAULT 1,
start2   IN     INTEGER DEFAULT 1)
RETURN INTEGER;

DBE_LOB.COMPARE (
lob1     IN     CLOB,
lob2     IN     CLOB,
len      IN     INTEGER DEFAULT DBE_LOB.LOBMAXSIZE,
start1   IN     INTEGER DEFAULT 1,
start2   IN     INTEGER DEFAULT 1)
RETURN INTEGER;
```

表 10-74 DBE_LOB.COMPARE 接口参数说明

参数	说明
lob1	第一个要比较的BLOB/CLOB类型对象。
lob2	第二个要比较的BLOB/CLOB类型对象。
len	要比较的字符数或者字节数，最大值为DBE_LOB.LOBMAXSIZE。
start1	第一个LOB描述符的偏移量，初始位置是1。
start2	第二个LOB描述符的偏移量，初始位置是1。

- DBE_LOB.SUBSTR

  用于读取一个LOB的子串，返回读取的字节个数或者字符个数，当amount > 1或者amount < 32767，offset < 1或者offset > LOBMAXSIZE的时候返回值是NULL。

DBE_LOB.SUBSTR函数原型为：

```
DBE_LOB.SUBSTR (
lob       IN     BLOB,
len       IN     INTEGER DEFAULT  32767,
start     IN     INTEGER DEFAULT  1)
RETURN RAW;

DBE_LOB.SUBSTR (
lob       IN     CLOB,
len       IN     INTEGER DEFAULT  32767,
start     IN     INTEGER DEFAULT  1)
RETURN VARCHAR2;
```

表 10-75 DBE_LOB.SUBSTR 接口参数说明

参数	说明
lob	将要读取子串的LOB描述符，对于BLOB类型的返回值是读取的字节个数，对于CLOB类型的返回值是字符个数。
len	要读取的字节数或者字符数量。
start	从开始位置偏移的字符数或者字节数量。

- DBE_LOB.STRIP

  这个存储过程用于截断指定长度的LOB，执行完这个存储过程会将LOB的长度设置为newlen参数指定的长度。如果对一个空的LOB执行截断操作，不会有任何执行结果；如果指定的长度比LOB的长度长，会产生一个异常。

  DBE_LOB.STRIP函数原型为：

  ```
  DBE_LOB.STRIP (
  lob       IN OUT     BLOB,
  newlen    IN       INTEGER);

  DBE_LOB.STRIP (
  lob       IN OUT     CLOB,
  newlen    IN       INTEGER);
  ```

  表 10-76 DBE_LOB.STRIP 接口参数说明

参数	说明
lob	待读入的指定BLOB类型对象。
newlen	截断后LOB的新长度，对于BLOB是字节数，对于CLOB是字符数。

- DBE_LOB.CREATE_TEMPORARY

  这个存储过程创建一个临时的BLOB或者CLOB，这个存储过程仅用于语法上的兼容，并无实际意义。

  DBE_LOB.CREATE_TEMPORARY函数原型为：

  ```
  DBE_LOB.CREATE_TEMPORARY (
  locator           INOUT      BLOB,
  cache             IN         BOOLEAN,
  keep_alive_time   IN         INTEGER);

  DBE_LOB.CREATE_TEMPORARY (
  locator           INOUT      CLOB,
  ```



```
cache            IN        BOOLEAN,
keep_alive_time  IN        INTEGER);
```

表 10-77 DBE_LOB.CREATE_TEMPORARY 接口参数说明

参数	说明
locator	LOB描述符。
cache	仅用于语法上的兼容。
keep_alive_time	仅用于语法上的兼容。

- DBE_LOB.APPEND

  存储过程APPEND根据指定长度及起始位置偏移读取BLOB内容的一部分到BUFFER缓冲区。

  DBE_LOB.APPEND函数原型为：

```
DBE_LOB.APPEND (
dest_lob    INOUT      BLOB,
src_lob     IN         BLOB);

DBE_LOB.APPEND (
dest_lob    INOUT      CLOB,
src_lob     IN         CLOB);
```

表 10-78 DBE_LOB.APPEND 接口参数说明

参数	说明
dest_lob	要写入的BLOB/CLOB对象。
src_lob	读取的BLOB/CLOB对象。

- DBE_LOB.FREETEMPORARY

  存储过程用于释放由CREATE_TEMPORARY创建的LOB文件。

  DBE_LOB.FREETEMPORARY函数原型为：

```
DBE_LOB.FREETEMPORARY (
lob_loc    INOUT      BLOB);

DBE_LOB.FREETEMPORARY (
lob_loc    INOUT      CLOB);
```

表 10-79 DBE_LOB.FREETEMPORARY 接口参数说明

参数	说明
lob_loc	要释放的BLOB/CLOB对象。

- DBE_LOB.FILEOPEN

  这个函数用于打开数据库外部BFILE类型文件，并返回这个文件对用的文件描述符（fd）。

  BFILE类型定义为：


```
DBE_LOB.BFILE (
directory    text,
filename     text);
```

DBE_LOB.FILEOPEN函数原型为：

```
DBE_LOB.FILEOPEN (
file          IN    DBE_LOB.BFILE,
open_mode     IN    text)
RETURN integer;
```

表 10-80 DBE_LOB.FILEOPEN 接口参数说明

参数	说明
file	要打开的数据库外部文件（BFILE类型包含了文件路径和文件名）。
open_mode	文件打开模式（w、r、a）。

- DBE_LOB.FILECLOSE

  这个函数用于关闭数据外部BFILE类型文件。

  DBE_LOB.FILECLOSE函数原型为：

  ```
  DBE_LOB.FILECLOSE (
  file     IN     integer);
  ```

  表 10-81 DBE_LOB.FILECLOSE 接口参数说明

参数	说明
file	要关闭的数据库外部文件（由FILEOPEN返回的文件描述符）。

- DBE_LOB.LOADFROMFILE

  用于将BFILE类型外部文件读取到BLOB文件中。

  DBE_LOB.LOADFROMFILE函数原型为：

  ```
  DBE_LOB.LOADFROMFILE (
  dest_lob      IN     BLOB,
  src_file      IN     INTEGER,
  amount        IN     INTEGER,
  dest_offset   IN     INTEGER,
  src_offset    IN     INTEGER)
  RETURN raw;
  ```

  表 10-82 DBE_LOB.LOADFROMFILE 接口参数说明

参数	说明
dest_lob	目标blob文件，bfile文件将读取到这个文件中。
src_bfile	需要读取的源bfile文件。
amount	blob文件的长度，超过这个阈值的文件将不会保存到blob中。

参数	说明
dest_offset	blob文件的偏移长度，dest_offset=1将从文件起始位置开始载入，以此类推。
src_offset	bfile文件的偏移长度，src_offset=1将从文件起始位置开始读取，以此类推。

- DBE_LOB.LOADBLOBFROMFILE

  用于将BFILE类型外部文件读取到BLOB文件中。

  DBE_LOB.LOADBLOBFROMFILE函数原型为：

```
DBE_LOB.LOADBLOBFROMFILE (
dest_lob      IN    BLOB,
src_file     IN    INTEGER,
amount        IN    INTEGER,
dest_offset   IN    INTEGER,
src_offset    IN    INTEGER)
RETURN raw;
```

表 10-83 DBE_LOB.LOADBLOBFROMFILE 接口参数说明

参数	说明
dest_lob	目标blob文件，bfile文件将读取到这个文件中。
src_bfile	需要读取的源bfile文件。
amount	blob文件的长度，超过这个阈值的文件将不会保存到blob中。
dest_offset	blob文件的偏移长度，dest_offset=1将从文件起始位置开始载入，以此类推。
src_offset	bfile文件的偏移长度，src_offset=1将从文件起始位置开始读取，以此类推。

- DBE_LOB.LOADCLOBFROMFILE

  用于将BFILE类型外部文件读取到CLOB文件中。

  DBE_LOB.LOADCLOBFROMFILE函数原型为：

```
DBE_LOB.LOADCLOBFROMFILE (
dest_lob      IN    CLOB,
src_file     IN    INTEGER,
amount        IN    INTEGER,
dest_offset   IN    INTEGER,
src_offset    IN    INTEGER)
RETURN raw;
```

表 10-84 DBE_LOB.LOADCLOBFROMFILE 接口参数说明

参数	说明
dest_lob	目标clob文件，bfile文件将读取到这个文件中。
src_bfile	需要读取的源bfile文件。

参数	说明
amount	clob文件的长度，超过这个阈值的文件将不会保存到clob中。
dest_offset	clob文件的偏移长度，dest_offset=1将从文件起始位置开始载入，以此类推。
src_offset	bfile文件的偏移长度，src_offset=1将从文件起始位置开始读取，以此类推。

- DBE_LOB.CONVERTTOBLOB

  这个函数将clob文件转换成blob文件。

  DBE_LOB.CONVERTTOBLOB函数原型为：

```
DBE_LOB.CONVERTTOBLOB(
dest_blob     IN   BLOB,
src_clob      IN   CLOB,
amount        IN   INTEGER default 32767,
dest_offset   IN   INTEGER default 1,
src_offset    IN   INTEGER default 1)
RETURN raw;
```

  表 10-85 DBE_LOB.CONVERTTOBLOB 接口参数说明

参数	说明
dest_lob	目标blob文件，clob转换后的文件。
src_bfile	需要读取的源clob文件。
amount	blob文件的长度，超过这个阈值的文件将不会保存到blob中。
dest_offset	blob文件的偏移长度，dest_offset=1将从文件起始位置开始载入，以此类推。
src_offset	clob文件的偏移长度，src_offset=1将从文件起始位置开始读取，以此类推。

- DBE_LOB.CONVERTTOCLOB

  这个函数将blob文件转换成clob文件。

  DBE_LOB.CONVERTTOCLOB函数原型为：

```
DBE_LOB.CONVERTTOCLOB(
dest_clob     IN   CLOB,
src_blob      IN   BLOB,
amount        IN   INTEGER default 32767,
dest_offset   IN   INTEGER default 1,
src_offset    IN   INTEGER default 1)
RETURN text;
```

  表 10-86 DBE_LOB.CONVERTTOCLOB 接口参数说明

参数	说明
dest_lob	目标clob文件，blob转换后的文件。

参数	说明
src_bfile	需要读取的源blob文件。
amount	clob文件的长度，超过这个阈值的文件将不会保存到clob中。
dest_offset	clob文件的偏移长度，dest_offset=1将从文件起始位置开始载入，以此类推。
src_offset	blob文件的偏移长度，src_offset=1将从文件起始位置开始读取，以此类推。

示例

```
--获取字符串的长度
SELECT DBE_LOB.GET_LENGTH('12345678');
 get_length
------------
          8
(1 row)
DECLARE
myraw  RAW(100);
amount INTEGER :=2;
buffer INTEGER :=1;
begin
DBE_LOB.READ('123456789012345',amount,buffer,myraw);
dbe_output.print_line(myraw);
end;
/
0123
ANONYMOUS BLOCK EXECUTE

CREATE TABLE blob_Table (t1 blob) DISTRIBUTE BY REPLICATION;
CREATE TABLE blob_Table_bak (t2 blob) DISTRIBUTE BY REPLICATION;
INSERT INTO blob_Table VALUES('abcdef');
INSERT INTO blob_Table_bak VALUES('22222');

DECLARE
str varchar2(100) := 'abcdef';
source raw(100);
dest blob;
copyto blob;
amount int;
PSV_SQL varchar2(100);
PSV_SQL1 varchar2(100);
a int :=1;
len int;
BEGIN
source := dbe_raw.cast_from_varchar2_to_raw(str);
amount := dbe_raw.get_length(source);

PSV_SQL :='select * from blob_Table for update';
PSV_SQL1 := 'select * from blob_Table_bak for update';

EXECUTE IMMEDIATE PSV_SQL into dest;
EXECUTE IMMEDIATE PSV_SQL1 into copyto;

DBE_LOB.WRITE(dest, amount, 1, source);
DBE_LOB.WRITE_APPEND(dest, amount, source);

DBE_LOB.ERASE(dest, a, 1);
DBE_OUTPUT.PRINT_LINE(a);
DBE_LOB.COPY(copyto, dest, amount, 10, 1);
perform DBE_LOB.CLOSE(dest);
```



```
RETURN;
END;
/
1
ANONYMOUS BLOCK EXECUTE
--删除表
DROP TABLE blob_Table;
DROP TABLE
DROP TABLE blob_Table_bak;
DROP TABLE
```

10.12.2.2 DBE_RANDOM

接口介绍

高级功能包DBE_RANDOM支持的所有接口请参见**表 DBE_RANDOM接口参数说明**。

表 10-87 DBE_RANDOM 接口参数说明

接口名称	描述
DBE_RANDOM.SET_SEED	设置一个随机数的种子。
DBE_RANDOM.GET_VALUE	生成一个大小介于指定的low及high之间的随机数。

- DBE_RANDOM.SET_SEED

  存储过程SEED用于设置一个随机数的种子。DBE_RANDOM.SET_SEED函数原型为：

  ```
  DBE_RANDOM.SET_SEED (seed  IN  INTEGER);
  ```

  表 10-88 DBE_RANDOM.SET_SEED 接口参数说明

参数	描述
seed	用于产生一个随机数的种子。

- DBE_RANDOM.GET_VALUE

  函数GET_VALUE生成一个大小介于指定的low及high之间的随机数。
  DBE_RANDOM.GET_VALUE函数原型为：

  ```
  DBE_RANDOM.GET_VALUE(
  min  IN  NUMBER default 0,
  max  IN  NUMBER default 1)
  RETURN NUMBER;
  ```

  表 10-89 DBE_RANDOM.GET_VALUE 接口参数说明

参数	描述
min	指定随机数大小的下边界，生成的随机数大于或等于min。
max	指定随机数大小的上边界，生成的随机数小于max。

说明

- 实际上，只要求这里的参数类型是NUMERIC即可，对于左右边界的大小并没有要求。
- DBE_RANDOM实现的是伪随机，所以若使用的初值（种子）不变，那么伪随机数的数序也不变，使用时需要注意。
- 生成的随机数有效数字为15位。

示例

```
--产生0到1之间的随机数：
SELECT DBE_RANDOM.GET_VALUE(0,1);
    get_value
------------------
 .917468812743886(1 row)
--对于指定范围内的整数，要加入参数min和max，并从结果中截取较小的数（最大值不能被作为可能的值）。所以对于0到99之间的整数，使用下面的代码：
SELECT TRUNC(DBE_RANDOM.GET_VALUE(0,100));
 trunc
-------
    26
(1 row)
```

10.12.2.3 DBE_OUTPUT

接口介绍

高级功能包DBE_OUTPUT支持的所有接口请参见**表 DBE_OUTPUT**。

表 10-90 DBE_OUTPUT

接口名称	描述
DBE_OUTPUT.PRINT_LINE	输出指定的文本，并添加换行符。
DBE_OUTPUT.PRINT	输出指定的文本，不添加换行符。
DBE_OUTPUT.SET_BUFFER_SIZE	设置输出缓冲区的大小，如果不指定则缓冲区最大能容忍20000字节，如果指定小于等于2000字节，则缓冲区允许容纳2000字节。

- DBE_OUTPUT.PRINT_LINE

存储过程PRINT_LINE向消息缓冲区写入一行带有行结束符的文本。
DBE_OUTPUT.PRINT_LINE函数原型为：

```
DBE_OUTPUT.PRINT_LINE (
format IN VARCHAR2);
```

表 10-91 DBE_OUTPUT.PRINT_LINE 接口参数说明

参数	描述
format	写入消息缓冲区的文本。

- DBE_OUTPUT.PRINT

存储过程PRINT将指定的文本输出到指定文本的前面，不添加换行符。
DBE_OUTPUT.PRINT函数原型为：

```
DBE_OUTPUT.PRINT (
format IN VARCHAR2);
```

表 10-92 DBE_OUTPUT.PRINT 接口参数说明

参数	描述
format	写入指定文本前的文本。

- DBE_OUTPUT.SET_BUFFER_SIZE

存储过程SET_BUFFER_SIZE设置输出缓冲区的大小，如果不指定的话缓冲区最大只能容纳20000字节。DBE_OUTPUT.SET_BUFFER_SIZE函数原型为：

```
DBE_OUTPUT.SET_BUFFER_SIZE (
size IN INTEGER default 20000);
```

表 10-93 DBE_OUTPUT.SET_BUFFER_SIZE 接口参数说明

参数	描述
size	设置输出缓冲区的大小。

示例

```
BEGIN
    DBE_OUTPUT.SET_BUFFER_SIZE(50);
    DBE_OUTPUT.PRINT('hello, ');
    DBE_OUTPUT.PRINT_LINE('database!');--输出hello, database!
END;
/
```

10.12.2.4 DBE_RAW

接口介绍

高级功能包DBE_RAW支持的所有接口请参见**表 DBE_RAW**。

表 10-94 DBE_RAW

接口名称	描述
DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW	将INTEGER类型值转换为二进制表示形式（即RAW类型）。

接口名称	描述
DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER	将二进制表示形式的整型值（即RAW类型）转换为INTEGER类型。
DBE_RAW.GET_LENGTH	获取RAW类型对象的长度。
DBE_RAW.CAST_FROM_VARCHAR2_TO_RAW	将VARCHAR2类型值转化为二进制表示形式（即RAW类型）。
DBE_RAW.CAST_TO_VARCHAR2	将RAW类型值转换成VARCHAR2类型。
DBE_RAW.SUBSTR	求RAW类型子串。
DBE_RAW.BIT_OR	RAW类型按位或。

须知

RAW类型的外部表现形式是十六进制，内部存储形式是二进制。例如一个RAW类型的数据11001011的表现形式为‘CB’，即在实际的类型转换中输入的是‘CB’。

- DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW

  存储过程CAST_FROM_BINARY_INTEGER_TO_RAW将INTEGER类型值转换为二进制表示形式（即RAW类型）。

  DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW函数原型为：

  ```
  DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW (
  value         IN  INTEGER,
  endianess     IN  INTEGER DEFAULT 1)
  RETURN RAW;
  ```

  表 10-95 DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW 接口参数说明

参数	描述
value	待转成RAW类型的整型数值。
endianess	表示字节序的整型值1或2（1代表BIG_ENDIAN，2代表LITTLE-ENDIAN）。

- DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER

  存储过程CAST_FROM_RAW_TO_BINARY_INTEGER将二进制表示形式的整型值（即RAW类型）转换为INTEGER类型。

  DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER函数原型为：

  ```
  DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER (
  value         IN  RAW,
  endianess     IN  INTEGER DEFAULT 1)
  RETURN BINARY_INTEGER;
  ```


表 10-96 DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER 接口参数说明

参数	描述
value	二进制表示形式的整型值（即RAW类型）。
endianess	表示字节序的整型值1或2（1代表BIG_ENDIAN，2代表LITTLE-ENDIAN）。

- DBE_RAW.GET_LENGTH

  存储过程GET_LENGTH返回RAW类型对象的长度。

  DBE_RAW.GET_LENGTH函数原型为：

  ```
  DBE_RAW.GET_LENGTH(
  value     IN     RAW)
  RETURN INTEGER;
  ```

  表 10-97 DBE_RAW.GET_LENGTH 接口参数说明

参数	描述
value	RAW类型对象

- DBE_RAW.CAST_FROM_VARCHAR2_TO_RAW

  存储过程CAST_FROM_VARCHAR2_TO_RAW将VARCHAR2类型的对象转换成RAW类型。

  DBE_RAW.CAST_FROM_VARCHAR2_TO_RAW函数原型为：

  ```
  DBE_RAW.CAST_TO_RAW(
  str     IN     VARCHAR2)
  RETURN RAW;
  ```

  表 10-98 DBE_RAW.CAST_FROM_VARCHAR2_TO_RAW 接口参数说明

参数	描述
c	待转换的VARCHAR2类型对象

- DBE_RAW.CAST_TO_VARCHAR2

  存储过程CAST_TO_VARCHAR2将RAW类型的对象转换成VARCHAR2类型。

  DBE_RAW.CAST_TO_VARCHAR2函数原型为：

  ```
  DBE_RAW.CAST_TO_VARCHAR2(
  str     IN     RAW)
  RETURN VARCHAR2;
  ```

  表 10-99 DBE_RAW.CAST_TO_VARCHAR2 接口参数说明

参数	描述
str	待转换的RAW类型对象

- DBE_RAW.BIT_OR

存储过程BIT_OR求两个RAW按位或的结果。

DBE_RAW.BIT_OR函数原型为：

```
DBE_RAW.BIT_OR(
str1     IN    RAW,
str2     IN    RAW)
RETURN RAW;
```

表 10-100 DBE_RAW.BIT_OR 接口参数说明

参数	描述
str1	按位或的第一个字符串
str2	按位或的第二个字符串

- DBE_RAW.SUBSTR

  存储过程SUBSTR将RAW类型的对象按起始位和长度截取。

  DBE_RAW.SUBSTR函数原型为：

  ```
  DBE_RAW.SUBSTR(
     IN lob_loc raw,
     IN off_set integer default 1,
     IN amount integer default 32767)
  RETURN RAW;
  ```

  表 10-101 DBE_RAW.SUBSTR 接口参数说明

参数	描述
lob_loc	源raw字符串
off_set	子串的起始位置，默认值1
amount	子串的长度，默认值32767

示例

```
--在存储过程中操作RAW数据
CREATE OR REPLACE PROCEDURE proc_raw
AS
str varchar2(100) := 'abcdef';
source raw(100);
amount integer;
BEGIN
source := dbe_raw.cast_from_varchar2_to_raw(str);--类型转换
amount := dbe_raw.get_length(source);--获取长度
dbe_output.print_line(amount);
END;
/

--调用存储过程
CALL proc_raw();

--删除存储过程
DROP PROCEDURE proc_raw;
```


10.12.2.5 DBE_TASK

接口介绍

高级功能包DBE_TASK支持的所有接口请参见**表 DBE_TASK**。

表 10-102 DBE_TASK

接口名称	描述
DBE_TASK.SUBMIT	提交一个定时任务。作业号由系统自动生成。
DBE_TASK.JOB_SUBMIT	同DBE_TASK.SUBMIT。但提供语法兼容参数。
DBE_TASK.ID_SUBMIT	提交一个定时任务。作业号由用户指定。
DBE_TASK.CANCEL	通过作业号来删除定时任务。
DBE_TASK.RUN	运行定时任务。
DBE_TASK.FINISH	禁用或者启用定时任务。
DBE_TASK.UPDATE	修改定时任务的属性，包括任务内容、下次执行时间、执行间隔。
DBE_TASK.CHANGE	同DBE_TASK.UPDATE。但提供语法兼容参数。
DBE_TASK.CONTENT	修改定时任务的任务内容属性。
DBE_TASK.NEXT_TIME	修改定时任务的下次执行时间属性。
DBE_TASK.INTERVAL	修改定时任务的执行间隔属性。

- DBE_TASK.SUBMIT

  存储过程SUBMIT提交一个系统提供的定时任务。

  DBE_TASK.SUBMIT函数原型为：

  ```
  DBE_TASK.SUBMIT(
  what          IN   TEXT,
  next_time     IN   TIMESTAMP DEFAULT sysdate,
  interval_time IN   TEXT  DEFAULT 'null',
  id            OUT  INTEGER
  )RETURN INTEGER;
  ```

  说明

  当创建一个定时任务（DBE_TASK）时，系统默认将当前数据库和用户名与当前创建的定时任务（DBE_TASK）绑定起来。该接口函数可以通过call或select调用，如果通过select调用，可以不填写出参。如果在存储过程中则需要用通过perform调用该接口函数。如果提交的sql语句任务使用到非public的schema，应该指定表或者函数的schema，或者在sql语句前添加set current_schema = xxx;语句。

表 10-103 DBE_TASK.SUBMIT 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
what	text	IN	否	要执行的SQL语句。支持一个或多个‘DDL’(不支持DB相关操作)，‘DML’，‘匿名块’，‘调用存储过程的语句’或4种混合的场景。
next_time	timestamp	IN	否	下次作业运行时间。默认值为当前系统时间（sysdate）。如果是过去时间，在提交作业时表示立即执行。
interval_time	text	IN	是	用来计算下次作业运行时间的时间表达式，可以是interval表达式，也可以是sysdate加上一个numeric值（例如：sysdate+1.0/24）。如果为空值或字符串"null"表示只执行一次，执行后JOB状态STATUS变成'd' 不再执行。
id	integer	OUT	否	作业号。范围为1～32767。当使用select调用时，该参数不能添加，当使用call调用时，该参数必须添加。

须知

当在TASK的参数what中创建用户时，日志会记录密码的明文。因此不建议在TASK任务中创建用户。该接口创建的任务不能保证高可用，建议使用PKG_SERVICE.SUBMIT_ON_NODES创建任务，并将job执行节点指定为CCN。

示例：

```
openGauss=# SELECT DBE_TASK.SUBMIT('call pro_xxx();', to_date('20180101','yyyymmdd'),'sysdate
+1');
 submit
--------
  31031
(1 row)

openGauss=# SELECT DBE_TASK.SUBMIT('call pro_xxx();', to_date('20180101','yyyymmdd'),'sysdate
+1.0/24');
 submit
--------
    512
(1 row)

openGauss=# DECLARE
openGauss-#     jobid int;
openGauss-# BEGIN
openGauss$#     PERFORM DBE_TASK.SUBMIT('call pro_xxx();', sysdate, 'interval ''5 minute''', jobid);
openGauss$# END;
openGauss$# /
ANONYMOUS BLOCK EXECUTE
```

- DBE_TASK.JOB_SUBMIT

  存储过程SUBMIT提交一个系统提供的定时任务。并提供了额外的兼容性参数。

DBE_TASK.JOB_SUBMIT函数原型为：

```
DBE_TASK.JOB_SUBMIT(
job           OUT  INTEGER,
what           IN   TEXT,
next_date      IN   TIMESTAMP  DEFAULT sysdate,
job_interval  IN   TEXT       DEFAULT 'null',
no_parse       IN   BOOLEAN    DEFAULT false,
instance       IN   INTEGER    DEFAULT 0,
force          IN   BOOLEAN    DEFAULT false
)RETURN INTEGER;
```

表 10-104 DBE_TASK.JOB_SUBMIT 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
job	integer	OUT	否	作业号。范围为1～32767。当使用select调用dbe.job_submit时，该参数可以省略。
what	text	IN	否	要执行的SQL语句。支持一个或多个‘DDL’(不支持DB相关操作)，‘DML’，‘匿名块’，‘调用存储过程的语句’或4种混合的场景。
next_date	timestamp	IN	是	下次作业运行时间。默认值为当前系统时间（sysdate）。如果是过去时间，在提交作业时表示立即执行。
job_interval	text	IN	是	用来计算下次作业运行时间的时间表达式，可以是interval表达式，也可以是sysdate加上一个numeric值（例如：sysdate+1.0/24）。如果为空值或字符串"null"表示只执行一次，执行后JOB状态STATUS变成'd'不再执行。
no_parse	boolean	IN	是	默认值false，仅用于语法上的兼容。
instance	integer	IN	是	默认值0，仅用于语法上的兼容。
force	boolean	IN	是	默认值false，仅用于语法上的兼容。

示例：

```
openGauss=# DECLARE
openGauss-#     id integer;
openGauss-# BEGIN
openGauss$#     id = DBE_TASK.JOB_SUBMIT(
openGauss$#         what => 'insert into t1 values (1, 2)',
openGauss$#         job_interval => 'sysdate + 1' --daily
openGauss$#     );
openGauss$# END;
openGauss$# /
ANONYMOUS BLOCK EXECUTE
```


- DBE_TASK.ID_SUBMIT

  ID_SUBMIT与SUBMIT语法功能相同，但其第一个参数是入参，即指定的作业号，SUBMIT最后一个参数是出参，表示系统自动生成的作业号。

  ```
  DBE_TASK.ID_SUBMIT(
  id            IN   BIGINT,
  what          IN   TEXT,
  next_time     IN   TIMESTAMP DEFAULT sysdate,
  interval_time IN   TEXT  DEFAULT 'null');
  ```

  示例：

  ```
  openGauss=# CALL dbe_task.id_submit(101, 'insert_msg_statistic1;', sysdate, 'sysdate+3.0/24');
   id_submit
  -----------

  (1 row)
  ```

- DBE_TASK.CANCEL

  存储过程CANCEL删除指定的定时任务。

  DBE_TASK.CANCEL函数原型为：

  ```
  CANCEL(id  IN  INTEGER);
  ```

  表 10-105 DBE_TASK.CANCEL 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	integer	IN	否	指定的作业号。

  示例：

  ```
  openGauss=# CALL dbe_task.cancel(101);
   cancel
  --------

  (1 row)
  ```

- DBE_TASK.RUN

  存储过程RUN运行定时任务。

  DBE_TASK.RUN函数原型为：

  ```
  DBE_TASK.RUN(
  job          IN   BIGINT,
  force        IN   BOOLEAN DEFAULT FALSE);
  ```

  表 10-106 DBE_TASK.RUN 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
job	bigint	IN	否	指定的作业号。
force	boolean	IN	是	仅用于语法上的兼容。

示例：

```
openGauss=# BEGIN
openGauss$#     DBE_TASK.ID_SUBMIT(12345, 'insert_msg_statistic1;', sysdate, 'sysdate+3.0/24');
openGauss$#     DBE_TASK.RUN(12345);
openGauss$# END;
openGauss$# /
ANONYMOUS BLOCK EXECUTE
```

- DBE_TASK.FINISH

  存储过程FINISH禁用或者启用定时任务。

  DBE_TASK.FINISH函数原型为：

  ```
  DBE_TASK.FINISH(
  id          IN   INTEGER,
  broken      IN   BOOLEAN,
  next_time   IN   TIMESTAMP  DEFAULT  sysdate);
  ```

  表 10-107 DBE_TASK.FINISH 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	integer	IN	否	指定的作业号。
broken	boolean	IN	否	状态标志位，true代表禁用，false代表启用。具体true或false值更新当前job；如果为空值，则不改变原有job的状态。
next_time	timestamp	IN	是	下次运行时间，默认为当前系统时间。如果参数broken状态为true，则更新该参数为'4000-1-1'；如果参数broken状态为false，且如果参数next_time不为空值，则更新指定job的next_time值，如果next_time为空值，则不更新next_time值。该参数可以省略，为默认值。

  示例：

  ```
  openGauss=# CALL dbe_task.id_submit(101, 'insert_msg_statistic1;', sysdate, 'sysdate+3.0/24');
   id_submit
  -----------

  (1 row)

  openGauss=# CALL dbe_task.finish(101, true);
   finish
  --------

  (1 row)

  openGauss=# CALL dbe_task.finish(101, false, sysdate);
   finish
  --------
  ```



```
(1 row)
```

- DBE_TASK.UPDATE

  存储过程UPDATE修改定时任务的属性，包括任务内容、下次执行时间、执行间隔。

  DBE_TASK.UPDATE函数原型为：

```
dbe_task.UPDATE(
id              IN     INTEGER,
content         IN     TEXT,
next_time       IN     TIMESTAMP,
interval_time   IN  TEXT);
```

  表 10-108 DBE_TASK.UPDATE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	integer	IN	否	指定的作业号。
content	text	IN	是	执行的存储过程名或者sql语句块。如果该参数为空值，则不更新指定job的content值，否则更新指定job的content值。
next_time	timestamp	IN	是	下次运行时间。如果该参数为空值，则不更新指定job的next_time值，否则更新指定job的next_time值。
interval_time	text	IN	是	用来计算下次作业运行时间的时间表达式。如果该参数为空值，则不更新指定job的interval_time值；如果该参数不为空值，会校验interval_time是否为有效的时间类型或interval类型，则更新指定job的interval_time值。如果为字符串"null"表示只执行一次，执行后JOB状态STATUS变成'd' 不再执行。

  示例：

```
openGauss=# CALL dbe_task.update(101, 'call userproc();', sysdate, 'sysdate + 1.0/1440');
 update
--------


(1 row)

openGauss=# CALL dbe_task.update(101, 'insert into tbl_a values(sysdate);', sysdate, 'sysdate +
1.0/1440');
 update
--------


(1 row)
```

- DBE_TASK.CHANGE

  存储过程UPDATE修改定时任务的属性，包括任务内容、下次执行时间、执行间隔。

  DBE_TASK.CHANGE函数原型为：


```
DBE_TASK.CHANGE(
job             IN    INTEGER,
what            IN    TEXT       DEFAULT NULL,
next_date       IN    TIMESTAMP  DEFAULT NULL,
job_interval    IN    TEXT       DEFAULT NULL,
instance        IN    INTEGER    DEFAULT NULL,
force           IN    BOOLEAN    DEFAULT false);
```

表 10-109 DBE_TASK.CHANGE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
job	integer	IN	否	指定的作业号。
what	text	IN	是	执行的存储过程名或者sql语句块。如果该参数为空值，则不更新指定job的what值，否则更新指定job的what值。
next_date	timestamp	IN	是	下次运行时间。如果该参数为空值，则不更新指定job的next_date值，否则更新指定job的next_date值。
job_interval	text	IN	是	用来计算下次作业运行时间的时间表达式。如果该参数为空值，则不更新指定job的job_interval值；如果该参数不为空值，会校验job_interval是否为有效的时间类型或interval类型，则更新指定job的job_interval值。如果为字符串"null"表示只执行一次，执行后JOB状态STATUS变成'd' 不再执行。
instance	integer	IN	是	仅用于语法上的兼容。
force	boolean	IN	否	仅用于语法上的兼容。

示例：

```
openGauss=# BEGIN
openGauss$#     DBE_TASK.CHANGE(
openGauss$#         job => 101,
openGauss$#         what => 'insert into t2 values (2);'
openGauss$#     );
openGauss$# END;
openGauss$# /
ANONYMOUS BLOCK EXECUTE
```

- DBE_TASK.CONTENT

  存储过程CONTENT修改定时任务的任务内容属性。

  DBE_TASK.CONTENT函数原型为：

```
DBE_TASK.CONTENT(
id          IN     INTEGER,
content     IN TEXT);
```


表 10-110 DBE_TASK.CONTENT 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	integer	IN	否	指定的作业号。
content	text	IN	否	执行的存储过程调用或者sql语句块或者程序块。

说明

- 当content参数是一个或多个可以执行成功的sql语句/程序块/调用存储过程时，该接口函数才能被执行成功，否则会执行失败。
- 若content参数为一个简单的insert、update等语句，需要在表前加模式名。

示例：

```
openGauss=# CALL dbe_task.content(101, 'call userproc();');
 content
---------

(1 row)

openGauss=# CALL dbe_task.content(101, 'insert into tbl_a values(sysdate);');
 content
---------

(1 row)
```

- DBE_TASK.NEXT_TIME

  存储过程NEXT_TIME修改定时任务的下次执行时间属性。

  DBE_TASK.NEXT_TIME函数原型为：

```
DBE_TASK.NEXT_TIME(
id         IN    BIGINT,
next_time  IN    TEXT);
```

表 10-111 DBE_TASK.NEXT_TIME 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	bigint	IN	否	指定的作业号。
next_time	text	IN	否	下次运行时间。

说明

如果输入的next_time的值小于当前日期值，该job会立即执行一次。

示例：

```
openGauss=# CALL dbe_task.next_time(101, sysdate);
 next_time
-----------
```



```
(1 row)
```

- DBE_TASK.INTERVAL

  存储过程INTERVAL修改定时任务的执行间隔属性。

  DBE_TASK.INTERVAL函数原型为：

  ```
  DBE_TASK.INTERVAL(
  id             IN      INTEGER,
  interval_time      IN    TEXT);
  ```

  表 10-112 DBE_TASK.INTERVAL 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	integer	IN	否	指定的作业号。
interval_time	text	IN	是	用来计算下次作业运行时间的时间表达式。如果为空值或字符串"null"表示只执行一次，执行后JOB状态STATUS变成'd' 不再执行。作业间隔需要为有效的时间类型或interval类型。

  示例：

  ```
  openGauss=# CALL dbe_task.interval(101, 'sysdate + 1.0/1440');
   interval
  ----------

  (1 row)

  openGauss=# CALL dbe_task.cancel(101);
   cancel
  --------

  (1 row)
  ```

  说明

  对于指定job正在运行状态（即job_status为'r'）时，不允许通过cancel、update、next_time、content、interval等接口删除或修改job的参数信息。

约束说明

1. 使用summit/id_submit创建一个新job后，该job从属于当前coordinator（即：该job仅在当前coordinator上调度和执行），其他coordinator不会调度和执行该job，如果出现coordinator节点故障，无法保证job正常执行。建议使用**PKG_SERVICE.SUBMIT_ON_NO...**接口，将job执行节点指定为CCN，以保证节点故障时job仍然可用。不是所有coordinator都可以查看、修改、删除其他CN创建的job。
2. job只能通过dbe_task高级包提供的接口进行创建、更新、删除操作，因为高级包的接口中会考虑所有CN间job信息的同步和pg_job与pg_job_proc表主键的关联操作，如果通过DML语句对pg_job表进行增删改，会导致job信息在CN间不一致和系统表无法关联变更的混乱问题，会严重影响job内部的管理。

3. 由于用户创建的每个任务和CN绑定，当任务运行过程中，该CN故障，则该任务的状态无法实时刷新，仍为’r’状态，需要等CN启动正常后才能刷新为’s’状态。如果在任务未执行时CN故障，则该CN上的任务都得不到正常的调度和执行，需要人为干预让该CN恢复正常，或进行节点删除/替换、job才能正常的调度和执行。
4. job在定时执行过程中，需要在当前job所属的CN上实时更新该job的运行状态、最近执行开始时间、最近执行结束时间、下次开始时间、失败次数（如果job执行失败）等相关参数信息到pg_job系统表中，并同步到其他CN，保证job信息的一致性。如果其他CN存在节点故障，那么job所属CN会同步超时重发的处理，导致job执行时间变长，但CN间同步超时失败后，原CN上pg_job表中job的相关信息仍然能正常更新，且job能正常执行成功。当故障CN恢复正常后，可能出现该CN上pg_job表中当前job的执行时间、运行状态等参数与原CN上不一致的情况，需要原CN上再次执行该job后才能保证job信息的同步。
5. 对于并发同时有多个job到达执行时间的场景，由于会为每个job创建一个线程来执行job，由于系统内部启动每个线程的时间会有延迟，因此会导致同时并发执行的job的开始时间有延迟，每个job的延迟时间在0.1ms左右。

10.12.2.6 DBE_UTILITY

接口介绍

高级功能包DBE_UTILITY支持的所有接口请参见**表10-113**。

表 10-113 DBE_UTILITY

接口名称	描述
DBE_UTILITY.FORMAT_ERROR_BACKTRACE	输出存储过程异常的调用堆栈。
DBE_UTILITY.FORMAT_ERROR_STACK	输出存储过程异常的具体信息。
DBE_UTILITY.FORMAT_CALL_STACK	输出存储过程的调用堆栈。
DBE_UTILITY.GET_TIME	输出当前时间，一般用于做差得到执行时长。

- DBE_UTILITY.FORMAT_ERROR_BACKTRACE

存储过程FORMAT_ERROR_BACKTRACE返回在执行过程中出现错误时，出现错误位置的调用堆栈。DBE_UTILITY.FORMAT_ERROR_BACKTRACE函数原型为：

```
DBE_UTILITY.FORMAT_ERROR_BACKTRACE()
RETURN TEXT;
```

- DBE_UTILITY.FORMAT_ERROR_STACK

存储过程FORMAT_ERROR_STACK返回在执行过程中出现错误时，出现错误位置的具体信息。DBE_UTILITY.FORMAT_ERROR_STACK函数原型为：


```
DBE_UTILITY.FORMAT_ERROR_STACK()
RETURN TEXT;
```

- DBE_UTILITY.FORMAT_CALL_STACK

存储过程FORMAT_CALL_STACK设置输出函数调用堆栈。
DBE_UTILITY.FORMAT_CALL_STACK函数原型为：

```
DBE_UTILITY.FORMAT_CALL_STACK()
RETURN TEXT;
```

- DBE_UTILITY.GET_TIME

存储过程GET_TIME设置输出时间，通常用于做差，单独的返回值没有意义。
DBE_UTILITY.GET_TIME函数原型为：

```
DBE_UTILITY.GET_TIME()
RETURN BIGINT;
```

示例

```
CREATE OR REPLACE PROCEDURE test_get_time1()
AS
declare
    start_time  bigint;
    end_time  bigint;
BEGIN
    start_time:= dbe_utility.get_time ();
    pg_sleep(1);
    end_time:=dbe_utility.get_time ();
    dbe_output.print_line(end_time - start_time);
END;
/
```

10.12.2.7 DBE_SQL

接口介绍

高级功能包DBE_SQL支持的接口请参见**表10-114**。

表 10-114 DBE_SQL

接口名称	描述
DBE_SQL.REGISTER_CONTEXT	打开一个游标。
DBE_SQL.SQL_UNREGISTER_CONTEXT	关闭一个已打开的游标。
DBE_SQL.SQL_SET_SQL	向游标传递一组SQL语句。
DBE_SQL.SQL_RUN	在游标上执行一组动态定义操作。
DBE_SQL.NEXT_ROW	读取游标一行数据。
DBE_SQL.SET_RESULT_TYPE	动态定义一个列。
DBE_SQL.SET_RESULT_TYPE_CHAR	动态定义一个char类型的列。
DBE_SQL.SET_RESULT_TYPE_INT	动态定义一个int类型的列。
DBE_SQL.SET_RESULT_TYPE_LONG	动态定义一个long类型的列。

接口名称	描述
DBE_SQL.SET_RESULT_TYPE_RAW	动态定义一个raw类型的列。
DBE_SQL.SET_RESULT_TYPE_TEXT	动态定义一个text类型的列。
DBE_SQL.SET_RESULT_TYPE_UNKNOWN	动态定义一个未知列（类型不识别时入此接口）。
DBE_SQL.GET_RESULT	读取一个已动态定义的列值。
DBE_SQL.GET_RESULT_CHAR	读取一个已动态定义的列值（指定char类型）。
DBE_SQL.GET_RESULT_INT	读取一个已动态定义的列值（指定int类型）。
DBE_SQL.GET_RESULT_LONG	读取一个已动态定义的列值（指定long类型）。
DBE_SQL.GET_RESULT_RAW	读取一个已动态定义的列值（指定raw类型）。
DBE_SQL.GET_RESULT_TEXT	读取一个已动态定义的列值（指定text类型）。
DBE_SQL.GET_RESULT_UNKNOWN	读取一个已动态定义的列值（类型不识别时入此接口）。
DBE_SQL.DBE_SQL_GET_RESULT_CHAR	读取一个已动态定义的列值（指定char类型）。
DBE_SQL.DBE_SQL_GET_RESULT_LONG	读取一个已动态定义的列值（指定long类型）。
DBE_SQL.DBE_SQL_GET_RESULT_RAW	读取一个已动态定义的列值（指定raw类型）。
DBE_SQL.IS_ACTIVE	检查游标是否已打开。
DBE_SQL.LAST_ROW_COUNT	兼容接口，暂不支持该功能。
DBE_SQL.RUN_AND_NEXT	预留接口，暂不支持该功能。
DBE_SQL.SQL_BIND_VARIABLE	根据语句中的变量，绑定一个值到该变量。
DBE_SQL.SQL_BIND_ARRAY	根据语句中的变量，绑定一组值到该变量。
DBE_SQL.SET_RESULT_TYPE_INTS	动态定义一个int数组类型的列。
DBE_SQL.SET_RESULT_TYPE_TEXTS	动态定义一个text数组类型的列。
DBE_SQL.SET_RESULT_TYPE_RAWS	动态定义一个raw数组类型的列。
DBE_SQL.SET_RESULT_TYPE_BYTEAS	动态定义一个bytea数组类型的列。
DBE_SQL.SET_RESULT_TYPE_CHARS	动态定义一个char数组类型的列。

接口名称	描述
DBE_SQL.SET_RESULTS_TYPE	动态定义一个数组类型的列。
DBE_SQL.GET_RESULTS_INT	读取一个已动态定义的列值（指定int数组类型）。
DBE_SQL.GET_RESULTS_TEXT	读取一个已动态定义的列值（指定text数组类型）。
DBE_SQL.GET_RESULTS_RAW	读取一个已动态定义的列值（指定raw数组类型）。
DBE_SQL.GET_RESULTS_BYTEA	读取一个已动态定义的列值（指定bytea数组类型）。
DBE_SQL.GET_RESULTS_CHAR	读取一个已动态定义的列值（指定char数组类型）。
DBE_SQL.GET_RESULTS	读取一个已动态定义的列值。
DBE_SQL.SQL_DESCRIBE_COLUMNS	描述游标读取的列信息。
DBE_SQL.DESC_REC	存储游标读取的列信息的类型。
DBE_SQL.DESC_TAB	DESC_REC的TABLE类型。
DBE_SQL.DATE_TABLE	DATE的TABLE类型。
DBE_SQL.NUMBER_TABLE	NUMBER的TABLE类型。
DBE_SQL.VARCHAR2_TABLE	VARCHAR2的TABLE类型。
DBE_SQL.BIND_VARIABLE	绑定参数接口。
DBE_SQL.SQL_SET_RESULTS_TYPE_C	动态定义一个数组类型的列。
DBE_SQL.SQL_GET_VALUES_C	读取一个已动态定义的列值。
DBE_SQL.GET_VARIABLE_RESULT	读取一个SQL语句执行后的返回值。
DBE_SQL.GET_VARIABLE_RESULT_CHAR	读取一个SQL语句执行后的返回值（指定char类型）。
DBE_SQL.GET_VARIABLE_RESULT_RAW	读取一个SQL语句执行后的返回值（指定raw类型）。
DBE_SQL.GET_VARIABLE_RESULT_TEXT	读取一个SQL语句执行后的返回值（指定text类型）。
DBE_SQL.GET_VARIABLE_RESULT_INT	读取一个SQL语句执行后的返回值（指定int类型）。
DBE_SQL.GET_ARRAY_RESULT_TEXT	读取一个SQL语句执行后的返回值（指定text数组类型）。
DBE_SQL.GET_ARRAY_RESULT_RAW	读取一个SQL语句执行后的返回值（指定raw数组类型）。

接口名称	描述
DBE_SQL.GET_ARRAY_RESULT_CHAR	读取一个SQL语句执行后的返回值（指定char数组类型）。
DBE_SQL.GET_ARRAY_RESULT_INT	读取一个SQL语句执行后的返回值（指定int数组类型）。

说明

- 建议使用dbe_sql.set_result_type及dbe_sql.get_result定义参数列。
- 当结果集大于work_mem设定值时会触发结果集临时下盘，但最大阈值不超过512MB。

- DBE_SQL.REGISTER_CONTEXT

  该函数用来打开一个游标，是后续dbe_sql各项操作的前提。该函数不传入任何参数，内部自动递增生成游标ID，并作为返回值返回给integer定义的变量。

  DBE_SQL.REGISTER_CONTEXT函数原型为：

  ```
  DBE_SQL.REGISTER_CONTEXT(
  )
  RETURN INTEGER;
  ```

- DBE_SQL.SQL_UNREGISTER_CONTEXT

  该函数用来关闭一个游标，是dbe_sql各项操作的结束。如果在存储过程结束时没有调用该函数，则该游标占用的内存仍然会保存，因此关闭游标非常重要。由于异常情况的发生会中途退出存储过程，导致游标未能关闭，因此建议存储过程中有异常处理，将该接口包含在内。

  DBE_SQL.SQL_UNREGISTER_CONTEXT函数原型为：

  ```
  DBE_SQL.SQL_UNREGISTER_CONTEXT(
  context_id      IN INTEGER
  )
  RETURN INTEGER;
  ```

  表 10-115 DBE_SQL.SQL_UNREGISTER_CONTEXT 接口说明

参数名称	描述
context_id	打算关闭的游标ID号

- DBE_SQL.SQL_SET_SQL

  该函数用来解析给定游标的查询语句。目前语句参数仅可通过text类型传递，长度不大于1G。

  DBE_SQL.SQL_SET_SQL函数的原型为：

  ```
  DBE_SQL.SQL_SET_SQL(
  context_id      IN INTEGER,
  query_string IN TEXT,
  language_flag        IN INTEGER
  )
  RETURN BOOLEAN;
  ```


表 10-116 DBE_SQL.SQL_SET_SQL 接口说明

参数名称	描述
context_id	执行查询语句解析的游标ID
query_string	执行的查询语句
language_flag	版本语言号，目前只支持1

- DBE_SQL.SQL_RUN

  该函数用来执行一个给定的游标。该函数接收一个游标ID，运行后获得的数据用于后续操作。

  DBE_SQL.SQL_RUN函数的原型为：

```
DBE_SQL.SQL_RUN(
context_id     IN INTEGER,
)
RETURN INTEGER;
```

表 10-117 DBE_SQL.SQL_RUN 接口说明

参数名称	描述
context_id	执行查询语句解析的游标ID

- DBE_SQL.NEXT_ROW

  该函数返回符合查询条件的数据行数，每一次运行该接口都会获取到新的行数的集合，直到数据读取完毕获取不到新行为止。

  DBE_SQL.NEXT_ROW函数的原型为：

```
DBE_SQL.NEXT_ROW(
context_id     IN INTEGER,
)
RETURN INTEGER;
```

表 10-118 DBE_SQL.NEXT_ROW 接口说明

参数名称	描述
context_id	执行的游标ID

- DBE_SQL.SET_RESULT_TYPE

  该函数用来定义从给定游标返回的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。

  DBE_SQL.SET_RESULT_TYPE函数的原型为：

```
DBE_SQL.SET_RESULT_TYPE(
context_id     IN INTEGER,
pos     IN INTEGER,
column_ref   IN ANYELEMENT,
maxsize     IN INTEGER default 1024
)
RETURN INTEGER;
```


表 10-119 DBE_SQL.SET_RESULT_TYPE 接口说明

参数名称	描述
context_id	执行的游标ID
pos	动态定义列在查询中的位置
column_ref	任意类型的变量，可根据变量类型选择适当的接口动态定义列
maxsize	定义的列的长度

- DBE_SQL.SET_RESULT_TYPE_CHAR

  该函数用来定义从给定游标返回的CHAR类型的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。

  DBE_SQL.SET_RESULT_TYPE_CHAR函数的原型为：

```
DBE_SQL.SET_RESULT_TYPE_CHAR(
context_id     IN INTEGER,
pos     IN INTEGER,
column_ref       IN TEXT,
column_size     IN INTEGER
)
RETURN INTEGER;
```

表 10-120 DBE_SQL.SET_RESULT_TYPE_CHAR 接口说明

参数名称	描述
context_id	执行的游标ID
pos	动态定义列在查询中的位置
column_ref	需要定义的某类型的参数变量
column_size	动态定义列长度

- DBE_SQL.SET_RESULT_TYPE_INT

  该函数用来定义从给定游标返回的INT类型的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。

  DBE_SQL.SET_RESULT_TYPE_INT函数的原型为：

```
DBE_SQL.SET_RESULT_TYPE_INT(
context_id     IN INTEGER,
pos     IN INTEGER
)
RETURN INTEGER;
```

表 10-121 DBE_SQL.SET_RESULT_TYPE_INT 接口说明

参数名称	描述
context_id	执行的游标ID
pos	动态定义列在查询中的位置

- DBE_SQL.SET_RESULT_TYPE_LONG

  该函数用来定义从给定游标返回的长列类型（非数据类型long）的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。长列的大小限制为1G。

  DBE_SQL.SET_RESULT_TYPE_LONG函数的原型为：

```
DBE_SQL.SET_RESULT_TYPE_LONG(
context_id     IN INTEGER,
pos     IN INTEGER
)
RETURN INTEGER;
```

  表 10-122 DBE_SQL.SET_RESULT_TYPE_LONG 接口说明

参数名称	描述
context_id	执行的游标ID
pos	动态定义列在查询中的位置

- DBE_SQL.SET_RESULT_TYPE_RAW

  该函数用来定义从给定游标返回的RAW类型的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。

  DBE_SQL.SET_RESULT_TYPE_RAW函数的原型为：

```
DBE_SQL.SET_RESULT_TYPE_RAW(
context_id     IN INTEGER,
pos     IN INTEGER,
column_ref       IN RAW,
column_size     IN INTEGER
)
RETURN INTEGER;
```

  表 10-123 DBE_SQL.SET_RESULT_TYPE_RAW 接口说明

参数名称	描述
context_id	执行的游标ID
pos	动态定义列在查询中的位置
column_ref	RAW类型的参数变量
column_size	列的长度

- DBE_SQL.SET_RESULT_TYPE_TEXT

  该函数用来定义从给定游标返回的TEXT类型的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。

  DBE_SQL.SET_RESULT_TYPE_TEXT函数的原型为：

```
DBE_SQL.DEFINE_COLUMN_CHAR(
context_id     IN INTEGER,
pos     IN INTEGER,
maxsize     IN INTEGER
)
RETURN INTEGER;
```


表 10-124 DBE_SQL.SET_RESULT_TYPE_TEXT 接口说明

参数名称	描述
context_id	执行的游标ID
pos	动态定义列在查询中的位置
maxsize	定义的TEXT类型的最大长度

- DBE_SQL.SET_RESULT_TYPE_UNKNOWN

  该函数用来处理从给定游标返回的未知数据类型的列，该接口仅用于类型不识别时的报错退出。

  DBE_SQL.SET_RESULT_TYPE_UNKNOWN函数的原型为：

```
DBE_SQL.SET_RESULT_TYPE_UNKNOWN(
context_id      IN INTEGER,
pos     IN INTEGER,
col_type        IN TEXT
)
RETURN INTEGER;
```

表 10-125 DBE_SQL.SET_RESULT_TYPE_UNKNOWN 接口说明

参数名称	描述
context_id	执行的游标ID
posn	动态定义列在查询中的位置
col_type	动态定义的参数

- DBE_SQL.GET_RESULT

  该函数用来返回给定游标给定位置的游标元素值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  DBE_SQL.GET_RESULT函数的原型为：

```
DBE_SQL.GET_RESULT(
context_id              IN     INTEGER,
pos             IN     INTEGER,
column_value            INOUT ANYELEMENT
)
RETURN ANYELEMENT;
```

表 10-126 DBE_SQL.GET_RESULT 接口说明

参数名称	描述
context_id	执行的游标ID
pos	动态定义列在查询中的位置
column_value	定义的列的返回值

- DBE_SQL.GET_RESULT_CHAR

  该存储过程用来返回给定游标给定位置的游标CHAR类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

DBE_SQL.GET_RESULT_CHAR函数的原型为：

```
DBE_SQL.GET_RESULT_CHAR(
context_id          IN    INTEGER,
pos                 IN    INTEGER,
tr                  INOUT CHARACTER,
err                 INOUT NUMERIC,
actual_length       INOUT INTEGER
);
```

表 10-127 DBE_SQL.GET_RESULT_CHAR 接口说明

参数名称	描述
context_id	执行的游标ID
pos	动态定义列在查询中的位置
tr	返回值
err	错误号。传出参数，须传入变量做参数。目前未实现，固定传出-1。
actual_length	返回值的实际长度

DBE_SQL.GET_RESULT_CHAR函数的重载函数为：

```
DBE_SQL.GET_RESULT_CHAR(
context_id          IN    INTEGER,
pos                 IN    INTEGER,
tr                  INOUT CHARACTER
);
```

表 10-128 DBE_SQL.GET_RESULT_CHAR 接口说明

参数名称	描述
context_id	执行的游标ID
pos	动态定义列在查询中的位置
tr	返回值

- DBE_SQL.GET_RESULT_INT

  该函数用来返回给定游标给定位置的游标INT类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。DBE_SQL.GET_RESULT_INT函数的原型为：

  ```
  DBE_SQL.GET_RESULT_INT(
  context_id              IN    INTEGER,
  pos                 IN    INTEGER
  )
  RETURN INTEGER;
  ```

  表 10-129 DBE_SQL.GET_RESULT_INT 接口说明

参数名称	描述
context_id	执行的游标ID

参数名称	描述
pos	动态定义列在查询中的位置

- DBE_SQL.GET_RESULT_LONG

  该函数用来返回给定游标给定位置的游标长列（非long/bigint整型）类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  DBE_SQL.GET_RESULT_LONG函数的原型为：

```
DBE_SQL.GET_RESULT_LONG(
context_id          IN INTEGER,
pos                 IN      INTEGER,
lgth                IN    INTEGER,
off_set             IN   INTEGER,
vl                INOUT     TEXT,
vl_length           INOUT    INTEGER
)
RETURN RECORD;
```

  表 10-130 DBE_SQL.GET_RESULT_LONG 接口说明

参数名称	描述
context_id	执行的游标ID
pos	动态定义列在查询中的位置
lgth	返回值的长度
off_set	返回值的起始位置
vl	返回值
vl_length	实际返回值的长度

- DBE_SQL.GET_RESULT_RAW

  该存储过程用来返回给定游标给定位置的游标RAW类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  DBE_SQL.GET_RESULT_RAW存储过程的原型为：

```
DBE_SQL.GET_RESULT_RAW(
context_id          IN  INTEGER,
pos                 IN      INTEGER,
tr                INOUT     RAW,
err               INOUT     NUMERIC,
actual_length       INOUT INTEGER
);
```

  表 10-131 DBE_SQL.GET_RESULT_RAW 接口说明

参数名称	描述
context_id	执行的游标ID
pos	动态定义列在查询中的位置
tr	返回的列值

参数名称	描述
err	错误号。传出参数，须传入变量做参数。目前未实现，固定传出-1。
actual_length	返回值的实际长度，不能长于此值，否则截断。

DBE_SQL.GET_RESULT_RAW函数的重载函数为：

```
DBE_SQL.GET_RESULT_RAW(
context_id          IN  INTEGER,
pos                 IN      INTEGER,
tr                INOUT     RAW
);
```

表 10-132 DBE_SQL.GET_RESULT_RAW 接口说明

参数名称	描述
context_id	执行的游标ID
pos	动态定义列在查询中的位置
tr	返回的列值

- DBE_SQL.GET_RESULT_TEXT

  该函数用来返回给定游标给定位置的游标TEXT类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  DBE_SQL.GET_RESULT_TEXT函数的原型为：

  ```
  DBE_SQL.GET_RESULT_TEXT(
  context_id          IN  INTEGER,
  pos                 IN      INTEGER
  )
  RETURN TEXT;
  ```

  表 10-133 DBE_SQL.GET_RESULT_TEXT 接口说明

参数名称	描述
context_id	执行的游标ID
pos	动态定义列在查询中的位置

- DBE_SQL.GET_RESULT_UNKNOWN

  该函数用来返回给定游标给定位置的游标未知类型的值，该接口为类型不支持时的报错处理接口。

  DBE_SQL.GET_RESULT_UNKNOWN函数的原型为：

  ```
  DBE_SQL.GET_RESULT_UNKNOWN(
  context_id          IN    INTEGER,
  pos                 IN          INTEGER,
  col_type          IN      TEXT
  )
  RETURN TEXT;
  ```


表 10-134 DBE_SQL.GET_RESULT_UNKNOWN 接口说明

参数名称	描述
context_id	执行的游标ID
pos	动态定义列在查询中的位置
col_type	返回的参数类型

- DBE_SQL.DBE_SQL_GET_RESULT_CHAR

  该函数用来返回给定游标给定位置的游标CHAR类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。和DBE_SQL.GET_RESULT_CHAR的区别是，不设置返回值长度，返回整个字符串。

  DBE_SQL.DBE_SQL_GET_RESULT_CHAR函数的原型为：

```
DBE_SQL.DBE_SQL_GET_RESULT_CHAR(
context_id     IN    INTEGER,
pos            IN    INTEGER
)
RETURN CHARACTER;
```

表 10-135 DBE_SQL.GET_RESULT_CHAR 接口说明

参数名称	描述
context_id	执行的游标ID
pos	动态定义列在查询中的位置

- DBE_SQL.DBE_SQL_GET_RESULT_LONG

  该函数用来返回给定游标给定位置的游标长列（非long/bigint整型）类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  和DBE_SQL.GET_RESULT_LONG的区别是，不设置返回值长度，返回整个BIGINT值。

  DBE_SQL.DBE_SQL_GET_RESULT_LONG函数的原型为：

```
DBE_SQL.DBE_SQL_GET_RESULT_LONG(
context_id          IN INTEGER,
pos                 IN INTEGER
)
RETURN BIGINT;
```

表 10-136 DBE_SQL.DBE_SQL_GET_RESULT_LONG 接口说明

参数名称	描述
context_id	执行的游标ID
pos	动态定义列在查询中的位置

- DBE_SQL.DBE_SQL_GET_RESULT_RAW

  该函数用来返回给定游标给定位置的游标RAW类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  和函数DBE_SQL.GET_RESULT_RAW的区别是，不设置返回值长度，返回整个字符串。

DBE_SQL.DBE_SQL_GET_RESULT_RAW函数的原型为：

```
DBE_SQL.GET_RESULT_RAW(
context_id          IN  INTEGER,
pos                 IN          INTEGER,
tr                  INOUT       RAW
)
RETURN RAW;
```

表 10-137 DBE_SQL.GET_RESULT_RAW 接口说明

参数名称	描述
context_id	执行的游标ID
pos	动态定义列在查询中的位置

- DBE_SQL.IS_ACTIVE

该函数用来返回游标的当前状态：打开、解析、执行、定义。取值是为TRUE，关闭后为FALSE，未知时报错，其余默认为关闭。

DBE_SQL.IS_ACTIVE函数的原型为：

```
DBE_SQL.IS_ACTIVE(
context_id                IN    INTEGER
)
RETURN BOOLEAN;
```

表 10-138 DBE_SQL.IS_ACTIVE 接口说明

参数名称	描述
context_id	被查询的游标ID

- DBE_SQL.SQL_BIND_VARIABLE

  该函数用来绑定一个参数到SQL语句，当执行SQL语句时，会根据该绑定的值来执行。

  DBE_SQL.SQL_BIND_VARIABLE函数的原型为：

```
DBE_SQL.SQL_BIND_VARIABLE(
    context_id in int,
    query_string in text,
    language_flag in anyelement,
    out_value_size in int default null
)
RETURNS void;
```

表 10-139 DBE_SQL.SQL_BIND_VARIABLE 接口说明

参数名称	描述
context_id	被查询的游标ID。
query_string	绑定的变量名。
language_flag	绑定的值。

参数名称	描述
out_value_size	返回值的大小，默认值为null。

- DBE_SQL.SQL_BIND_ARRAY

  该函数用来绑定一组参数到SQL语句，当执行SQL语句时，会根据该绑定的数组来执行。

  DBE_SQL.SQL_BIND_ARRAY函数的原型为：

```
DBE_SQL.SQL_BIND_ARRAY(
    IN context_id int,
    IN query_string text,
    IN value anyarray
)
RETURNS void;
DBE_SQL.SQL_BIND_ARRAY(
    IN context_id int,
    IN query_string text,
    IN value anyarray,
    IN lower_index int,
    IN higher_index int
)
RETURNS void;
```

  表 10-140 DBE_SQL.SQL_BIND_ARRAY 接口说明

参数名称	描述
context_id	被查询的游标ID。
query_string	绑定的变量名
value	绑定的数组
lower_index	绑定数组的最小下标
higher_index	绑定数组的最大下标

- DBE_SQL.SET_RESULT_TYPE_INTS

  该函数用来定义从给定游标返回的INT数组类型的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。

  DBE_SQL.SET_RESULT_TYPE_INTS函数的原型为：

```
DBE_SQL.SET_RESULT_TYPE_INTS(
    IN context_id int,
    IN pos int,
    IN column_ref anyarray,
    IN cnt int,
    IN lower_bnd int
)
RETURNS integer;
```

  表 10-141 DBE_SQL.SET_RESULT_TYPE_INTS 接口说明

参数名称	描述
context_id	被查询的游标ID

参数名称	描述
pos	动态定义列在查询中的位置
column_ref	标记返回的数组类型
cnt	标记一次获取多少个值
lower_bnd	标记返回数组时的开始下标

- DBE_SQL.SET_RESULT_TYPE_TEXTS

  该函数用来定义从给定游标返回的TEXT数组类型的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。

  DBE_SQL.SET_RESULT_TYPE_TEXTS函数的原型为：

```
DBE_SQL.SET_RESULT_TYPE_TEXTS(
    IN context_id int,
    IN pos int,
    IN column_ref anyarray,
    IN cnt int,
    IN lower_bnd int,
    IN maxsize int
)
RETURNS integer;
```

  表 10-142 DBE_SQL.SET_RESULT_TYPE_TEXTS 接口说明

参数名称	描述
context_id	被查询的游标ID
pos	动态定义列在查询中的位置
column_ref	标记返回的数组类型
cnt	标记一次获取多少个值
lower_bnd	标记返回数组时的开始下标
maxsize	定义的TEXT类型的最大长度

- DBE_SQL.SET_RESULT_TYPE_RAWS

  该函数用来定义从给定游标返回的RAW数组类型的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。

  DBE_SQL.SET_RESULT_TYPE_RAWS函数的原型为：

```
DBE_SQL.set_result_type_raws(
    IN context_id int,
    IN pos int,
    IN column_ref anyarray,
    IN cnt int,
    IN lower_bnd int,
    IN column_size int
)
RETURNS integer;
```


表 10-143 DBE_SQL.SET_RESULT_TYPE_RAWS 接口说明

参数名称	描述
context_id	被查询的游标ID
pos	动态定义列在查询中的位置
column_ref	标记返回的数组类型
cnt	标记一次获取多少个值
lower_bnd	标记返回数组时的开始下标
column_size	列的长度

- DBE_SQL.SET_RESULT_TYPE_BYTEAS

  该函数用来定义从给定游标返回的BYTEA数组类型的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。

  DBE_SQL.SET_RESULT_TYPE_BYTEAS函数的原型为：

```
DBE_SQL.set_result_type_byteas(
    IN context_id int,
    IN pos int,
    IN column_ref anyarray,
    IN cnt int,
    IN lower_bnd int,
    IN column_size int
)
RETURNS integer;
```

表 10-144 DBE_SQL.SET_RESULT_TYPE_BYTEAS 接口说明

参数名称	描述
context_id	被查询的游标ID
pos	动态定义列在查询中的位置
column_ref	标记返回的数组类型
cnt	标记一次获取多少个值
lower_bnd	标记返回数组时的开始下标
column_size	列的长度

- DBE_SQL.SET_RESULT_TYPE_CHARS

  该函数用来定义从给定游标返回的CHAR数组类型的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。

  DBE_SQL.SET_RESULT_TYPE_CHARS函数的原型为：

```
DBE_SQL.SET_RESULT_TYPE_CHARS(
    IN context_id int,
    IN pos int,
    IN column_ref anyarray,
    IN cnt int,
```



```
    IN lower_bnd int,
    IN column_size int
)
RETURNS integer;
```

表 10-145 DBE_SQL.SET_RESULT_TYPE_CHARS 接口说明

参数名称	描述
context_id	被查询的游标ID
pos	动态定义列在查询中的位置
column_ref	标记返回的数组类型
cnt	标记一次获取多少个值
lower_bnd	标记返回数组时的开始下标
column_size	列的长度

- DBE_SQL.SET_RESULTS_TYPE

  该函数用来定义从给定游标返回的列，该接口只能应用于SELECT定义的游标中。定义的列通过查询列表的相对位置来标识，传入变量的数据类型决定了该列被定义的类型。

  DBE_SQL.SET_RESULTS_TYPE函数的原型为：

```
DBE_SQL.SET_RESULTS_TYPE(
    IN context_id int,
    IN pos int,
    IN column_ref anyarray,
    IN cnt int,
    IN lower_bnd int,
    IN maxsize int  DEFAULT 1024
) returns void;
```

表 10-146 DBE_SQL.SET_RESULTS_TYPE 接口说明

参数名称	描述
context_id	被查询的游标ID
pos	动态定义列在查询中的位置
column_ref	标记返回的数组类型
cnt	标记一次获取多少个值
lower_bnd	标记返回数组时的开始下标
maxsize	定义的类型的最大长度

- DBE_SQL.GET_RESULTS_INT

  该函数用来返回给定游标给定位置的游标INT数组类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  DBE_SQL.GET_RESULTS_INT函数的原型为：

```
DBE_SQL.GET_RESULTS_INT(
    IN context_id int,
```



```
    IN pos int,
    INOUT column_value anyarray
);
```

表 10-147 DBE_SQL.GET_RESULTS_INT 接口说明

参数名称	描述
context_id	被查询的游标ID
pos	动态定义列在查询中的位置
column_value	返回值

- DBE_SQL.GET_RESULTS_TEXT

  该函数用来返回给定游标给定位置的游标TEXT数组类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  DBE_SQL.GET_RESULTS_TEXT函数的原型为：

```
DBE_SQL.GET_RESULTS_TEXT(
    IN context_id int,
    IN pos int,
    INOUT column_value anyarray
);
```

表 10-148 DBE_SQL.GET_RESULTS_TEXT 接口说明

参数名称	描述
context_id	被查询的游标ID
pos	动态定义列在查询中的位置
column_value	返回值

- DBE_SQL.GET_RESULTS_RAW

  该函数用来返回给定游标给定位置的游标RAW数组类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  DBE_SQL.GET_RESULTS_RAW函数的原型为：

```
DBE_SQL.GET_RESULTS_RAW(
    IN context_id int,
    IN pos int,
    INOUT column_value anyarray
);
```

表 10-149 DBE_SQL.GET_RESULTS_RAW 接口说明

参数名称	描述
context_id	被查询的游标ID
pos	动态定义列在查询中的位置
column_value	返回值

- DBE_SQL.GET_RESULTS_BYTEA

  该函数用来返回给定游标给定位置的游标BYTEA数组类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  DBE_SQL.GET_RESULTS_BYTEA函数的原型为：

```
DBE_SQL.GET_RESULTS_BYTEA(
    IN context_id int,
    IN pos int,
    INOUT column_value anyarray
);
```

  表 10-150 DBE_SQL.GET_RESULTS_BYTEA 接口说明

参数名称	描述
context_id	被查询的游标ID
pos	动态定义列在查询中的位置
column_value	返回值

- DBE_SQL.GET_RESULTS_CHAR

  该函数用来返回给定游标给定位置的游标CHAR数组类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  DBE_SQL.GET_RESULTS_CHAR函数的原型为：

```
DBE_SQL.GET_RESULTS_CHAR(
    IN context_id int,
    IN pos int,
    INOUT column_value anyarray
);
```

  表 10-151 DBE_SQL.GET_RESULTS_CHAR 接口说明

参数名称	描述
context_id	被查询的游标ID
pos	动态定义列在查询中的位置
column_value	返回值

- DBE_SQL.GET_RESULTS

  该函数用来返回给定游标给定位置的游标数组类型的值，该接口访问的是DBE_SQL.NEXT_ROW获取的数据。

  说明

  由于DBE_SQL.GET_RESULTS底层机制通过数组实现，当用不同的数组获取同一列的返回值时，会由于内部索引的不连续向数组中填充NULL值来确保数组本身索引的连续性，这会导致返回结果数组的长度和Oracle的不一致。

  DBE_SQL.GET_RESULTS函数的原型为：

```
DBE_SQL.GET_RESULTS(
    IN context_id int,
    IN pos int,
    INOUT column_value anyarray
);
```


表 10-152 DBE_SQL.GET_RESULTS 接口说明

参数名称	描述
context_id	被查询的游标ID
pos	动态定义列在查询中的位置
column_value	返回值

- DBE_SQL.SQL_DESCRIBE_COLUMNS
  该函数用来描述列信息，该接口只能应用于SELECT定义的游标中。
  DBE_SQL.SQL_DESCRIBE_COLUMNS函数的原型为：

```
DBE_SQL.SQL_DESCRIBE_COLUMNS(
    context_id in int,
    col_cnt inout int,
    desc_t inout dbe_sql.desc_tab
)RETURNS record ;
```

表 10-153 DBE_SQL.SQL_DESCRIBE_COLUMNS 接口说明

参数名称	描述
context_id	被查询的游标ID
col_cnt	返回的列的数量
desc_t	返回的列的描述信息

- DBE_SQL.DESC_REC
  该类型是复合类型，用来存储SQL_DESCRIBE_COLUMNS接口中的描述信息。
  DBE_SQL.DESC_REC函数的原型为：

```
CREATE TYPE DBE_SQL.DESC_REC AS (
    col_type            int,
    col_max_len         int,
    col_name            VARCHAR2(32),
    col_name_len        int,
    col_schema_name     VARCHAR2(32),
    col_schema_name_len int,
    col_precision       int,
    col_scale           int,
    col_charsetid       int,
    col_charsetform     int,
    col_null_ok         BOOLEAN
);
```

- DBE_SQL.DESC_TAB
  该类型是DESC_REC的TABLE类型，通过TABLE OF语法实现。
  DBE_SQL.DESC_TAB函数的原型为：

```
CREATE TYPE DBE_SQL.DESC_TAB AS TABLE OF DBE_SQL.DESC_REC;
```

- DBE_SQL.DATE_TABLE
  该类型是DATE的TABLE类型，通过TABLE OF语法实现。
  DBE_SQL.DATE_TABLE函数的原型为：

```
CREATE TYPE DBE_SQL.DATE_TABLE AS TABLE OF DATE;
```


- DBE_SQL.NUMBER_TABLE
  该类型是NUMBER的TABLE类型，通过TABLE OF语法实现。
  DBE_SQL.NUMBER_TABLE函数的原型为：
  ```
  CREATE TYPE DBE_SQL.NUMBER_TABLE AS TABLE OF NUMBER;
  ```
- DBE_SQL.VARCHAR2_TABLE
  该类型是VARCHAR2的TABLE类型，通过TABLE OF语法实现。
  DBE_SQL.VARCHAR2函数的原型为：
  ```
  CREATE TYPE DBE_SQL.VARCHAR2_TABLE AS TABLE OF VARCHAR2(2000);
  ```
- DBE_SQL.BIND_VARIABLE
  该函数是绑定参数接口，建议使用DBE_SQL.SQL_BIND_VARIABLE。
- DBE_SQL.SQL_SET_RESULTS_TYPE_C
  该函数是动态定义一个数组类型的列，不建议用户使用。
  DBE_SQL.SQL_SET_RESULTS_TYPE_C函数的原型为：
  ```
  DBE_SQL.sql_set_results_type_c(
      context_id in int,
      pos in int,
      column_ref in anyarray,
      cnt in int,
      lower_bnd in int,
      col_type in anyelement,
      maxsize in int
  )return integer;
  ```

  表 10-154 DBE_SQL.SQL_SET_RESULTS_TYPE_C 接口说明

参数名称	描述
context_id	被查询的游标ID
pos	动态定义列在查询中的位置
column_ref	标记返回的数组类型
cnt	标记一次获取多少个值
lower_bnd	标记返回数组时的开始下标
col_type	标记返回的数组类型对应的变量类型
maxsize	定义的类型的最大长度

- DBE_SQL.SQL_GET_VALUES_C
  该函数是读取一个已动态定义的列值，不建议用户使用。
  DBE_SQL.SQL_GET_VALUES_C函数的原型为：
  ```
  DBE_SQL.sql_get_values_c(
      context_id in int,
      pos in int,
      results_type inout anyarray,
      result_type in anyelement
  )return anyarray;
  ```


表 10-155 DBE_SQL.SQL_GET_VALUES_C 接口说明

参数名称	描述
context_id	被查询的游标ID
pos	参数位置信息
results_type	获取的结果
result_type	获取的结果类型

- DBE_SQL.GET_VARIABLE_RESULT

  该函数用来返回绑定的OUT参数的值，可以用来获取存储过程中的OUT参数。

  DBE_SQL.GET_VARIABLE_RESULT函数的原型为：

```
DBE_SQL.get_variable_result(
    IN context_id int,
    IN pos VARCHAR2,
    INOUT column_value anyelement
);
```

表 10-156 DBE_SQL.GET_VARIABLE_RESULT 接口说明

参数名称	描述
context_id	被查询的游标ID
pos	绑定的参数名
column_value	返回值

- DBE_SQL.GET_VARIABLE_RESULT_CHAR

  该函数用来返回绑定的CHAR类型的OUT参数的值，可以用来获取存储过程中的OUT参数。

  DBE_SQL.GET_VARIABLE_RESULT_CHAR函数的原型为：

```
DBE_SQL.get_variable_result_char(
    IN context_id int,
    IN pos VARCHAR2
)
RETURNS char
```

表 10-157 DBE_SQL.GET_VARIABLE_RESULT_CHAR 接口说明

参数名称	描述
context_id	被查询的游标ID
pos	绑定的参数名

- DBE_SQL.GET_VARIABLE_RESULT_RAW

  该函数用来返回绑定的RAW类型的OUT参数的值，可以用来获取存储过程中的OUT参数。

  DBE_SQL.GET_VARIABLE_RESULT_RAW函数的原型为：


```
CREATE OR REPLACE FUNCTION DBE_SQL.get_variable_result_raw(
    IN context_id int,
    IN pos VARCHAR2,
    INOUT value anyelement
)
RETURNS anyelement
```

表 10-158 DBE_SQL.GET_VARIABLE_RESULT_RAW 接口说明

参数名称	描述
context_id	被查询的游标ID
pos	绑定的参数名
value	返回值

- DBE_SQL.GET_VARIABLE_RESULT_TEXT

  该函数用来返回绑定的TEXT类型的OUT参数的值，可以用来获取存储过程中的OUT参数。

  DBE_SQL.GET_VARIABLE_RESULT_TEXT函数的原型为：

```
CREATE OR REPLACE FUNCTION DBE_SQL.get_variable_result_text(
    IN context_id int,
    IN pos VARCHAR2
)
RETURNS text
```

表 10-159 DBE_SQL.GET_VARIABLE_RESULT_TEXT 接口说明

参数名称	描述
context_id	被查询的游标ID
pos	绑定的参数名

- DBE_SQL.GET_VARIABLE_RESULT_INT

  该函数用来返回绑定的INT类型的OUT参数的值，可以用来获取存储过程中的OUT参数。

  DBE_SQL.GET_VARIABLE_RESULT_INT函数的原型为：

```
DBE_SQL.get_variable_result_int(
    IN context_id int,
    IN pos VARCHAR2,
    INOUT value anyelement
)
RETURNS anyelement
```

表 10-160 DBE_SQL.GET_VARIABLE_RESULT_INT 接口说明

参数名称	描述
context_id	被查询的游标ID
pos	绑定的参数名
value	返回值

- DBE_SQL.GET_ARRAY_RESULT_TEXT

  该函数用来返回绑定的TEXT数组类型的OUT参数的值，可以用来获取存储过程中的OUT参数。

  DBE_SQL.GET_ARRAY_RESULT_TEXT函数的原型为：

  ```
  DBE_SQL.get_array_result_text(
      IN context_id int,
      IN pos VARCHAR2,
      INOUT column_value anyarray
  )
  ```

  表 10-161 DBE_SQL.GET_ARRAY_RESULT_TEXT 接口说明

参数名称	描述
context_id	被查询的游标ID
pos	绑定的参数名
column_value	返回值

- DBE_SQL.GET_ARRAY_RESULT_RAW

  该函数用来返回绑定的RAW数组类型的OUT参数的值，可以用来获取存储过程中的OUT参数。

  DBE_SQL.GET_ARRAY_RESULT_RAW函数的原型为：

  ```
  DBE_SQL.get_array_result_raw(
      IN context_id int,
      IN pos VARCHAR2,
      INOUT column_value anyarray
  )
  ```

  表 10-162 DBE_SQL.GET_ARRAY_RESULT_RAW 接口说明

参数名称	描述
context_id	被查询的游标ID
pos	绑定的参数名
column_value	返回值

- DBE_SQL.GET_ARRAY_RESULT_CHAR

  该函数用来返回绑定的CHAR数组类型的OUT参数的值，可以用来获取存储过程中的OUT参数。

  DBE_SQL.GET_ARRAY_RESULT_CHAR函数的原型为：

  ```
  DBE_SQL.get_array_result_char(
      IN context_id int,
      IN pos VARCHAR2,
      INOUT column_value anyarray
  )
  ```


表 10-163 DBE_SQL.GET_ARRAY_RESULT_CHAR 接口说明

参数名称	描述
context_id	被查询的游标ID
pos	绑定的参数名
column_value	返回值

- DBE_SQL.GET_ARRAY_RESULT_INT

  该函数用来返回绑定的INT数组类型的OUT参数的值，可以用来获取存储过程中的OUT参数。

  DBE_SQL.GET_ARRAY_RESULT_INT函数的原型为：

```
DBE_SQL.get_array_result_int(
    IN context_id int,
    IN pos VARCHAR2,
    INOUT column_value anyarray
)
```

表 10-164 DBE_SQL.GET_ARRAY_RESULT_INT 接口说明

参数名称	描述
context_id	被查询的游标ID
pos	绑定的参数名
column_value	返回值

示例

```
--在存储过程中操作raw数据
openGauss=# create or replace procedure pro_dbe_sql_all_02(in_raw raw,v_in int,v_offset int)
as
context_id int;
v_id int;
v_info bytea :=1;
query varchar(2000);
execute_ret int;
define_column_ret_raw bytea :='1';
define_column_ret int;
begin
drop table if exists pro_dbe_sql_all_tb1_02 ;
create table pro_dbe_sql_all_tb1_02(a int ,b blob);
insert into pro_dbe_sql_all_tb1_02 values(1,HEXTORAW('DEADBEEE'));
insert into pro_dbe_sql_all_tb1_02 values(2,in_raw);
query := 'select * from pro_dbe_sql_all_tb1_02 order by 1';
--打开游标
context_id := dbe_sql.register_context();
--编译游标
dbe_sql.sql_set_sql(context_id, query, 1);
--定义列
define_column_ret:= dbe_sql.set_result_type(context_id,1,v_id);
define_column_ret_raw:= dbe_sql.set_result_type_raw(context_id,2,v_info,10);
--执行
execute_ret := dbe_sql.sql_run(context_id);
loop
exit when (dbe_sql.next_row(cursorid) <= 0);
--获取值
```



```
dbe_sql.get_result(context_id,1,v_id);
dbe_sql.get_result_raw(context_id,2,v_info,v_in,v_offset);
--输出结果
dbe_output.print_line('id:'|| v_id || ' info:' || v_info);
end loop;
--关闭游标
dbe_sql.sql_unregister_context(context_id);
end;
/
--调用存储过程
openGauss=# call pro_dbe_sql_all_02(HEXTORAW('DEADBEEF'),0,1);

--删除存储过程
openGauss=# DROP PROCEDURE pro_dbe_sql_all_02;
```

10.12.2.8 DBE_FILE

DBE_FILE包为存储过程提供了读、写操作系统文本文件的能力。

接口介绍

高级功能包DBE_FILE支持的所有接口请参见**表10-165**。

表 10-165 DBE_FILE

接口名称	描述
DBE_FILE.OPEN	根据指定的目录和文件名打开一个文件。
DBE_FILE.IS_CLOSE	检测一个文件句柄是否关闭。
DBE_FILE.IS_OPEN	检测一个文件句柄是否打开。
DBE_FILE.READ_LINE	从一个打开的文件句柄中读取一行指定长度的数据。
DBE_FILE.WRITE	将数据写入到一个打开的文件的缓冲区中。
DBE_FILE.NEW_LINE	将一个或者多个行终结符写入到一个打开的文件的缓冲区中。
DBE_FILE.WRITE_LINE	将数据写入到一个打开的文件的缓冲区中，并自动追加一个行终结符。
DBE_FILE.FORMAT_WRITE	将数据按指定格式写入到一个打开的文件的缓冲区中。
DBE_FILE.GET_RAW	从一个打开的文件中读取指定字节数的RAW类型数据。
DBE_FILE.PUT_RAW	将RAW类型数据写入到一个打开的文件的缓冲区中。
DBE_FILE.FLUSH	将缓存区中的数据写入到物理文件中。
DBE_FILE.CLOSE	关闭一个打开的文件句柄。
DBE_FILE.CLOSE_ALL	关闭一个会话中打开的所有的文件句柄。

接口名称	描述
DBE_FILE.REMOVE	根据指定的目录和文件名删除一个磁盘文件，操作的时候需要有充分的权限。
DBE_FILE.RENAME	重命名一个磁盘文件，类似Unix的mv指令。
DBE_FILE.COPY	复制一个连续区域的内容到一个新创建的文件中，如果忽略了start_line和end_line会复制整个文件。
DBE_FILE.GET_ATTR	读取并返回一个磁盘文件的属性。
DBE_FILE.SEEK	根据用户指定的字节数向前或者向后调整文件指针的位置。
DBE_FILE.GET_POS	以字节为单位返回文件当前的偏移量。

- DBE_FILE.OPEN

  该函数用来打开一个文件，可以指定最大行的大小，最多可以同时打开50个文件。并且该函数返回INTEGER类型的一个句柄。

  DBE_FILE.OPEN函数原型为：

```
DBE_FILE.OPEN (
dir             IN    VARCHAR2,
file_name       IN    VARCHAR2,
open_mode       IN    VARCHAR2,
max_line_size   IN    INTEGER DEFAULT 1024)
RETURN INTEGER;
```

  表 10-166 DBE_FILE.OPEN 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
dir	VARCHAR2	IN	否	文件的目录位置，这个字符串是一个目录对象名。 **说明** • 文件目录的位置，需要添加到系统表PG_DIRECTORY中，如果传入的路径和PG_DIRECTORY中的路径不匹配，会报路径不存在的错误，下面的涉及location作为参数的函数也是同样的情况。 • 在打开guc参数safe_data_path时，用户只能通过高级包读写safe_data_path指定文件路径下的文件。

参数	类型	入参/出参	是否可以为空	描述
file_name	VARCHAR2	IN	否	文件名，包含扩展（文件类型），不包括路径名。如果文件名中包含路径，在OPEN中会被忽略，在UNIX系统中，文件名不能以/.结尾。
open_mode	VARCHAR2	IN	否	指定文件的打开模式，包含r：read text，w: write text和a: append text。 **说明** 对于写操作，会检测写入文件类型，如果为elf类型文件，会报错退出。
max_line_size	INTEGER	IN	是	每行最大字符数，包含换行符（最小值是1，最大值是32767）。如果没有指定，会指定一个默认值1024。

- DBE_FILE.IS_OPEN

  函数DBE_FILE.IS_OPEN用于检测一个文件是否已经打开，返回一个布尔值，异常情况是INVALID_FILEHANDLE。

  DBE_FILE.IS_OPEN函数原型为：

  ```
  DBE_FILE.IS_OPEN(
     file IN INTEGER)
  RETURN BOOLEAN;
  ```

  表 10-167 DBE_FILE.IS_OPEN 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	是	待检测的文件句柄，为空时DBE_FILE.IS_OPEN接口返回FALSE。

- DBE_FILE.IS_CLOSE

  函数DBE_FILE.IS_CLOSE用于检测一个文件句柄是否已经关闭，返回布尔值，异常情况是INVALID_FILEHANDLE。

  DBE_FILE.IS_CLOSE函数原型为：

  ```
  DBE_FILE.IS_CLOSE (
  file      IN      INTEGER)
  RETURN BOOLEAN;
  ```


表 10-168 DBE_FILE.IS_CLOSE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	是	传入一个要检测的文件句柄。

- DBE_FILE.READ_LINE

  存储过程DBE_FILE.READ_LINE从一个打开的文件读取数据，并把读取的结果存放到BUFFER中。读取的时候会读取到行尾，但不包含行终结符，或者读取到文件末尾，或者读取到len参数指定的大小。读取的长度不能超过OPEN的时候指定的max_line_size。

  DBE_FILE.READ_LINE函数原型为：

  ```
  DBE_FILE.READ_LINE (
  file      IN   INTEGER,
  buffer    OUT  VARCHAR2,
  len       IN   INTEGER DEFAULT NULL)
  ```

表 10-169 DBE_FILE.READ_LINE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	通过调用OPEN打开的文件句柄，文件必须以读的模式打开，否则会抛出INVALID_OPERATION的异常。
buffer	VARCHAR2	OUT	否	用于接收数据的BUFFER。
len	INTEGE	IN	是	从文件中读取的字节数，默认是NULL。如果是默认NULL，会使用max_line_size来指定大小。

- DBE_FILE.WRITE

  函数DBE_FILE.WRITE用于向文件对应的缓冲区中写入BUFFER中的数据，文件必须以写模式打开，这个操作不会写入行终结符。

DBE_FILE.WRITE函数原型为：

```
DBE_FILE.WRITE (
file    IN    INTEGER,
buffer  IN    TEXT)
RETURN BOOLEAN;
```

表 10-170 DBE_FILE.WRITE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	通过OPEN打开的文件句柄，要写入的文件必须以写模式打开，这个操作不会写入行终结符。
buffer	TEXT	IN	是	要写入文件的文本数据，BUFFER的最大值是32767个字节。如果在open的时候没有指定值，默认是1024个字节，没有刷新到文件之前，一系列的WRITE操作的BUFFER总和不能超过32767个字节。 **说明** 对于写操作，会检测写入文件类型，如果为elf类型文件，会报错退出。

- DBE_FILE.NEW_LINE

  函数DBE_FILE.NEW_LINE用于向文件对应的缓冲区中写入一个或者多个行终结符，行终结符和平台相关。

  DBE_FILE.NEW_LINE函数原型为：

  ```
  DBE_FILE.NEW_LINE (
  file        IN    INTEGER,
  line_nums   IN    INTEGER := 1)
  RETURN BOOLEAN;
  ```

  表 10-171 DBE_FILE.NEW_LINE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	通过OPEN打开的文件句柄。

参数	类型	入参/出参	是否可以为空	描述
line_nums	INTEGER	IN	是	写入到文件中的终结符的数量，默认值为1。

- DBE_FILE.WRITE_LINE

  函数DBE_FILE.WRITE_LINE用于向文件对应的缓冲区中写入BUFFER中的数据，文件必须以写模式打开，这个操作会自动追加行终结符。

  DBE_FILE.WRITE_LINE函数原型为：

```
DBE_FILE.WRITE_LINE(
file       IN     INTEGER,
buffer     IN     TEXT,
flush      IN     BOOLEAN DEFAULT FALSE)
RETURN BOOLEAN;
```

表 10-172 DBE_FILE.WRITE_LINE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	通过OPEN打开的文件句柄。
buffer	TEXT	IN	是	要写入文件的文本数据，BUFFER的最大值是32767个字节。如果在open的时候没有指定值，默认是1024个字节，没有刷新到文件之前，一系列的PUT操作的BUFFER总和不能超过32767个字节。 **说明** 对于写操作，会检测写入文件类型，如果为elf类型文件，会报错退出。
flush	BOOLEAN	IN	是	在write后是否要刷到磁盘。

- DBE_FILE.FORMAT_WRITE

函数DBE_FILE.FORMAT_WRITE将格式化数据写入到一个打开的文件对应的缓冲区中，是允许格式化的DBE_FILE.WRITE接口。

DBE_FILE.FORMAT_WRITE函数原型为：

```
DBE_FILE.FORMAT_WRITE (
file   IN INTEGER,
format IN VARCHAR2,
arg1   IN VARCHAR2 DEFAULT NULL,
. . .
arg6 IN VARCHAR2 DEFAULT NULL])
RETURN BOOLEAN;
```

表 10-173 DBE_FILE.FORMAT_WRITE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	通过OPEN打开的文件句柄。
format	TEXT	IN	是	一个要进行格式化的字符串包含，文本和格式符\n和%s。
[arg1. . .arg6]	TEXT	IN	是	从1到6个可选的参数串，参数和格式化字符的位置是一一对应的，如果存在格式化字符而没有提供参数，会使用空串来替代%s。

- DBE_FILE.GET_RAW

  该函数用于从打开的文件描述符中读取二进制数据，从r中返回。

  DBE_FILE.GET_RAW函数原型为：

  ```
  DBE_FILE.GET_RAW (
  file    IN   INTEGER,
  r       OUT  RAW,
  length  IN   INTEGER DEFAULT NULL)
  RETURN RAW;
  ```

  表 10-174 DBE_FILE.GET_RAW 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	通过OPEN打开的文件句柄。

参数	类型	入参/出参	是否可以为空	描述
r	RAW	OUT	否	输出的二进制数据
length	INTEGER	IN	是	要读取文件的长度，默认值为NULL，读取文件中所有数据，最大长度为1G。

- DBE_FILE.PUT_RAW

  函数DBE_FILE.PUT_RAW用于向文件对应的缓冲区中写入RAW类型数据。

  DBE_FILE.PUT_RAW函数原型为：

```
DBE_FILE.PUT_RAW (
file    IN  INTEGER,
r       IN  RAW,
flush   IN  BOOLEAN DEFAULT FALSE)
RETURN BOOLEAN;
```

  表 10-175 DBE_FILE.PUT_RAW 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	通过OPEN打开的文件句柄。
r	RAW	IN	否	输出的二进制数据 **说明** 对于写操作，会检测写入文件类型，如果为elf类型文件，会报错退出。
flush	BOOLEAN	IN	是	是否flush到文件中，默认为false。

- DBE_FILE.FLUSH

  函数DBE_FILE.FLUSH将缓冲区中的数据写入到物理文件中，缓存中的数据必须要有一个行终结符。该函数可以将缓冲区的数据及时写入到对应的物理文件中。

DBE_FILE.FLUSH函数原型为：

```
DBE_FILE.FLUSH (
file    IN    INTEGER)
RETURN VOID;
```

表 10-176 DBE_FILE.FLUSH 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	一个打开的文件句柄。

- DBE_FILE.CLOSE

  函数DBE_FILE.CLOSE用于关闭一个打开的文件句柄，当调用这个函数的时候，如果还有等待写入的缓存的数据，可能会收到异常信息。

  DBE_FILE.CLOSE函数原型为：

  ```
  DBE_FILE.CLOSE (
  file IN INTEGER
  )RETURN INTEGER;
  ```

  表 10-177 DBE_FILE.CLOSE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	通过OPEN打开的文件句柄。

- DBE_FILE.CLOSE_ALL

  函数DBE_FILE.CLOSE_ALL关闭一个会话中打开的所有的文件句柄，可用于紧急的清理操作。

  DBE_FILE.CLOSE_ALL函数原型为：

  ```
  DBE_FILE.CLOSE_ALL()
  RETRUN VOID;
  ```


表 10-178 DBE_FILE.CLOSE_ALL 接口参数说明

参数	描述
无	无

- DBE_FILE.REMOVE

  函数DBE_FILE.REMOVE删除一个磁盘文件，使用的时候需要有充分的权限。

  DBE_FILE.REMOVE函数原型为：

```
DBE_FILE.REMOVE (
dir          IN     VARCHAR2,
file_name    IN     VARCHAR2)
RETURN VOID;
```

表 10-179 DBE_FILE.REMOVE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
dir	VARCHAR2	IN	否	文件的目录位置，这个字符串是一个目录对象名。 **说明** • 文件目录的位置，需要添加到系统表PG_DIRECTORY中，如果传入的路径和PG_DIRECTORY中的路径不匹配，会报路径不存在的错误，下面的涉及location作为参数的函数也是同样的情况。 • 在打开guc参数safe_data_path时，用户只能通过高级包读写safe_data_path指定文件路径下的文件。
file_name	VARCHAR2	IN	否	文件名。

- DBE_FILE.RENAME

  函数DBE_FILE.RENAME重命名一个磁盘文件，类似Unix的mv指令。

  DBE_FILE.RENAME函数原型为：

```
DBE_FILE.RENAME (
src_dir        IN   VARCHAR2,
src_file_name  IN   VARCHAR2,
dest_dir       IN   VARCHAR2,
dest_file_name IN   VARCHAR2,
overwrite      IN   BOOLEAN DEFAULT FALSE)
RETURN VOID;
```


表 10-180 DBE_FILE.RENAME 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
src_dir	VARCHAR2	IN	否	源文件的目录位置（大小写敏感）。 **说明** • 文件目录的位置，需要添加到系统表PG_DIRECTORY中，如果传入的路径和PG_DIRECTORY中的路径不匹配，会报路径不存在的错误，下面的涉及location作为参数的函数也是同样的情况。 • 在打开guc参数safe_data_path时，用户只能通过高级包读写safe_data_path指定文件路径下的文件。
src_file_name	VARCHAR2	IN	否	要进行命名的源文件。
dest_dir	VARCHAR2	IN	否	目的目录位置（大小写敏感）。 **说明** • 文件目录的位置，需要添加到系统表PG_DIRECTORY中，如果传入的路径和PG_DIRECTORY中的路径不匹配，会报路径不存在的错误，下面的涉及location作为参数的函数也是同样的情况。 • 在打开guc参数safe_data_path时，用户只能通过高级包读写safe_data_path指定文件路径下的文件。
dest_file_name	VARCHAR2	IN	否	新的文件名。
overwrite	BOOLEAN	IN	是	是否重写，参数指定为空或者不指定时表示不重写。在不重写的情况下，如果目的目录下已存在同名文件会报错。

- DBE_FILE.COPY

  函数DBE_FILE.COPY复制一个连续区域的内容到一个新创建的文件中，如果忽略了start_line和end_line会复制整个文件。

  DBE_FILE.COPY函数原型为：

```
DBE_FILE.COPY (
src_dir           IN     VARCHAR2,
src_file_name     IN     VARCHAR2,
dest_dir          IN     VARCHAR2,
dest_file_name    IN     VARCHAR2,
start_line        IN     INTEGER DEFAULT 1,
end_line          IN     INTEGER DEFAULT NULL)
RETURN VOID;
```

表 10-181 DBE_FILE.COPY 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
src_dir	VARCHAR2	IN	否	源文件所在的目录。 **说明** • 文件目录的位置，需要添加到系统表PG_DIRECTORY中，如果传入的路径和PG_DIRECTORY中的路径不匹配，会报路径不存在的错误，下面的涉及location作为参数的函数也是同样的情况。 • 在打开guc参数safe_data_path时，用户只能通过高级包读写safe_data_path指定文件路径下的文件。
src_file_name	VARCHAR2	IN	否	要拷贝的源文件。
dest_dir	VARCHAR2	IN	否	目的文件所在的目录。 **说明** • 文件目录的位置，需要添加到系统表PG_DIRECTORY中，如果传入的路径和PG_DIRECTORY中的路径不匹配，会报路径不存在的错误，下面的涉及location作为参数的函数也是同样的情况。 • 在打开guc参数safe_data_path时，用户只能通过高级包读写safe_data_path指定文件路径下的文件。
dest_file_name	VARCHAR2	IN	否	目的文件名。 **说明** 对于写操作，会检测写入文件类型，如果为elf类型文件，会报错退出。
start_line	INTEGER	IN	否	拷贝开始的行号，默认是1。

参数	类型	入参/出参	是否可以为空	描述
end_line	INTEGER	IN	是	拷贝结束的行号，默认是NULL，如果是NULL，则指定到文件尾。

- DBE_FILE.GET_ATTR

  函数DBE_FILE.GET_ATTR读取并返回一个磁盘文件的属性。

  DBE_FILE.GET_ATTR函数原型为：

```
DBE_FILE.GET_ATTR(
location IN text,
filename IN text,
OUT fexists boolean,
OUT file_length bigint,
OUT block_size integer);
```

表 10-182 DBE_FILE.GET_ATTR 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
location	TEXT	IN	否	文件所在的目录。 **说明** • 文件目录的位置，需要添加到系统表PG_DIRECTORY中，如果传入的路径和PG_DIRECTORY中的路径不匹配，会报路径不存在的错误，下面的涉及location作为参数的函数也是同样的情况。 • 在打开guc参数safe_data_path时，用户只能通过高级包读写safe_data_path指定文件路径下的文件。
filename	TEXT	IN	否	文件名。
fexists	BOOLEAN	OUT	否	文件是否存在。

参数	类型	入参/出参	是否可以为空	描述
file_length	BIGINT	OUT	否	文件的字节长度，如果文件不存在返回NULL。
block_size	INTEGER	OUT	否	文件系统的块大小（单位字节），如果文件不存在返回NULL。

- DBE_FILE.SEEK

  函数DBE_FILE.SEEK根据用户指定的字节数向前或者向后调整文件指针的位置。

  DBE_FILE.SEEK函数原型为：

```
DBE_FILE.SEEK (
file             IN INTEGER,
absolute_start  IN      BIGINT DEFAULT NULL,
relative_start  IN      BIGINT DEFAULT NULL)
RETURN VOID;
```

  表 10-183 DBE_FILE.SEEK 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	通过OPEN打开的文件句柄。
absolute_start	BIGINT	IN	是	文件偏移的绝对位置，默认值为NULL。
relative_start	BIGINT	IN	是	文件偏移的相对位置。如果这个值是正数，向前偏移；如果是负数，向后偏移；默认值为NULL。如果和absolute_start参数同时指定，以absolute_start参数为准。

- DBE_FILE.GET_POS

  函数DBE_FILE.GET_POS以字节为单位返回文件当前的偏移量。

DBE_FILE.FGETPOS函数原型为：

```
DBE_FILE.GET_POS (
file    IN     INTEGER)
RETURN BIGINT;
```

表 10-184 DBE_FILE.GET_POS 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
file	INTEGER	IN	否	通过OPEN打开的文件句柄。

示例

```
--系统管理员向PG_DIRECTORY系统表中加入目录/temp/：
CREATE OR REPLACE DIRECTORY dir AS '/tmp/';
-- 预期结果为：
CREATE DIRECTORY

--打开一个文件并向文件中写入数
据
DECLARE
  f integer;
  dir text := 'dir';
BEGIN
  f := dbe_file.open(dir, 'sample.txt', 'w');
  PERFORM dbe_file.write_line(f, 'ABC');
  PERFORM dbe_file.write_line(f, '123'::numeric);
  PERFORM dbe_file.write_line(f, '-----');
  PERFORM dbe_file.new_line(f);
  PERFORM dbe_file.write_line(f, '*******');
  PERFORM dbe_file.new_line(f, 0);
  PERFORM dbe_file.write_line(f, '++++++++');
  PERFORM dbe_file.new_line(f, 2);
  PERFORM dbe_file.write_line(f, '########');
  PERFORM dbe_file.write(f, 'A');
  PERFORM dbe_file.write(f, 'B');
  PERFORM dbe_file.new_line(f);
  PERFORM dbe_file.format_write(f, '[1 -> %s, 2 -> %s, 3 -> %s, 4 -> %s, 5 -> %s]', 'gaussdb', 'dbe', 'file',
'get', 'line');
  PERFORM dbe_file.new_line(f);
  PERFORM dbe_file.write_line(f, '1234567890');
  f := dbe_file.close(f);
END;
/
-- 预期结果为：
ANONYMOUS BLOCK EXECUTE

--在上面写入的文件中读取数据。
DECLARE
  f integer;
  dir text := 'dir';
BEGIN
  f := dbe_file.open(dir, 'sample.txt', 'r');
  FOR i IN 1..11 LOOP
      RAISE INFO '[%] : %', i, dbe_file.read_line(f);
```



```
  END LOOP;
END;
/
-- 预期结果为：
INFO:  [1] : ABC
INFO:  [2] : 123
INFO:  [3] : -----
INFO:  [4] :
INFO:  [5] : *******
INFO:  [6] : ++++++++
INFO:  [7] :
INFO:  [8] :
INFO:  [9] : ########
INFO:  [10] : AB
INFO:  [11] : [1 -> gaussdb, 2 -> dbe, 3 -> file, 4 -> get, 5 -> line]
ANONYMOUS BLOCK EXECUTE

-- 对文件句柄执行位置偏移，并获取文件的当前位
置。
DECLARE
    l_file   integer;
    l_buffer VARCHAR2(32767);
    dir text := 'dir';
    abs_offset number := 100;
    rel_offset number := NULL;
BEGIN
    l_file := dbe_file.open(dir  => dir,  file_name  => 'sample.txt',open_mode => 'R');
    dbe_output.print_line('before seek: current position is ' || dbe_file.get_pos(file => l_file)); -- before seek:
current position is 0
    dbe_file.seek(file => l_file, absolute_start=>abs_offset, relative_start=>rel_offset);
    dbe_output.print_line('fseek: current position is ' || dbe_file.get_pos(file => l_file)); -- seek: current
position is 100
    l_file := dbe_file.close(file => l_file);
END;
/
-- 预期结果为：
before seek: current position is 0
fseek: current position is 100
ANONYMOUS BLOCK EXECUTE
```

10.12.2.9 DBE_SESSION

接口介绍

高级功能包DBE_SESSION支持的所有接口请参见**表10-185**。DBE_SESSION作用范围是session级别。

表 10-185 DBE_SESSION

接口名称	描述
DBE_SESSION.SET_CONTEXT	设置指定context下，某一属性(attribute)的值(value)。
DBE_SESSION.CLEAR_CONTEXT	清除指定context下，某一属性(attribute)的值(value)。
DBE_SESSION.SEARCH_CONTEXT	查找指定context下，某一属性(attribute)的值(value)。

- DBE_SESSION.SET_CONTEXT

向指定namespace(context)下，设置某一属性(attribute)的值(value)。
DBE_SESSION.SET_CONTEXT函数原型为：

```
DBE_SESSION.SET_CONTEXT(
    namespace text,
    attribute text,
    value text
)returns void;
```

表 10-186 DBE_SESSION.SET_CONTEXT 接口参数说明

参数	描述
namespace	需要设置的context名称，当context不存在时，新建context，最长支持128个字符
attribute	属性名称，最长支持128个字符
value	要设置的值的名称，最长支持128个字符

- DBE_SESSION.CLEAR_CONTEXT

清除指定namespace(context)下，某一属性(attribute)的值(value)。
DBE_SESSION.CLEAR_CONTEXT函数原型为：

```
DBE_SESSION.CLEAR_CONTEXT (
    namespace text,
    client_identifier text default 'null',
    attribute text
)returns void ;
```

表 10-187 DBE_SESSION.CLEAR_CONTEXT 接口参数说明

参数	描述
namespace	用户指定的context
client_identifier	客户端认证，默认'null'，通常情况用户无需手动设置
attribute	要清除的属性

- DBE_SESSION.SEARCH_CONTEXT

查找指定namespace(context)下，某一属性(attribute)的值(value)。
DBE_SESSION.SEARCH_CONTEXT函数原型为：

```
DBE_SESSION.SEARCH_CONTEXT (
    namespace text,
    attribute text
)returns text;
```


表 10-188 DBE_SESSION.SEARCH_CONTEXT 接口参数说明

参数	描述
namespace	用户指定的context
attribute	要查找的属性

示例

```
BEGIN
    select DBE_SESSION.set_context('test', 'gaussdb', 'one');    --设置名为test的context下属性为gaussdb的值为
one
    select DBE_SESSION.search_context('test', 'gaussdb');
    select DBE_SESSION.clear_context('test', 'test','gaussdb');
END;
/
```

10.12.2.10 DBE_MATCH

接口介绍

高级功能包DBE_MATCH支持的所有接口请参见**表10-189**。

表 10-189 DBE_MATCH

接口名称	描述
DBE_MATCH.EDIT_DISTANCE_SIMILARITY	比较两个字符串的差距(删除、新增、变换的最小步骤)，并归一化到0-100（100表示完全一致，0表示完全不一致）。

- DBE_MATCH.EDIT_DISTANCE_SIMILARITY

比较两个字符串的差距(删除、新增、变换的最小步骤)，并归一化到0-100（100表示完全一致，0表示完全不一致）,DBE_MATCH.EDIT_DISTANCE_SIMILARITY函数原型为：

```
DBE_MATCH.EDIT_DISTANCE_SIMILARITY(
    str1 IN text,
    str2 IN text
)returns integer ;
```

表 10-190 DBE_MATCH.EDIT_DISTANCE_SIMILARITY 接口参数说明

参数	描述
str1	第一个字符串，如果为null，直接输出0
str2	第二个字符串，如果为null，直接输出0

10.12.2.11 DBE_SCHEDULER

接口介绍

高级功能包DBE_SCHEDULER支持通过调度（schedule）和程序（program）更加灵活的创建定时任务。支持的所有接口请见**表10-191**。

须知

DBE_SCHEDULER尚不支持节点间同步定时任务，若要创建多节点定时任务请使用**DBE_TASK**接口实现。

表 10-191 DBE_SCHEDULER

接口名称	描述
DBE_SCHEDULER.CREATE_JOB	创建定时任务。
DBE_SCHEDULER.DROP_JOB	删除定时任务。
DBE_SCHEDULER.DROP_SINGLE_JOB	删除单个定时任务。
DBE_SCHEDULER.SET_ATTRIBUTE	设置对象属性。
DBE_SCHEDULER.RUN_JOB	运行定时任务。
DBE_SCHEDULER.RUN_BACKEND_JOB	后台运行定时任务。
DBE_SCHEDULER.RUN_FOREGROUND_JOB	前台运行定时任务。
DBE_SCHEDULER.STOP_JOB	停止定时任务。
DBE_SCHEDULER.STOP_SINGLE_JOB	停止单个定时任务。
DBE_SCHEDULER.GENERATE_JOB_NAME	生成定时任务名。
DBE_SCHEDULER.CREATE_PROGRAM	创建程序。
DBE_SCHEDULER.DEFINE_PROGRAM_ARGUMENT	定义程序参数。
DBE_SCHEDULER.DROP_PROGRAM	删除程序。
DBE_SCHEDULER.DROP_SINGLE_PROGRAM	删除单个程序。
DBE_SCHEDULER.SET_JOB_ARGUMENT_VALUE	设置定时任务参数值。

接口名称	描述
DBE_SCHEDULER.CREATE_SCHEDULE	创建调度。
DBE_SCHEDULER.DROP_SCHEDULE	删除调度。
DBE_SCHEDULER.DROP_SINGLE_SCHEDULE	删除单个调度。
DBE_SCHEDULER.CREATE_JOB_CLASS	创建定时任务类。
DBE_SCHEDULER.DROP_JOB_CLASS	删除定时任务类。
DBE_SCHEDULER.DROP_SINGLE_JOB_CLASS	删除单个定时任务类。
DBE_SCHEDULER.GRANT_USER_AUTHORIZATION	赋予用户特殊权限。
DBE_SCHEDULER.REVOKE_USER_AUTHORIZATION	撤销用户特殊权限。
DBE_SCHEDULER.CREATE_CREDENTIAL	创建证书。
DBE_SCHEDULER.DROP_CREDENTIAL	销毁证书。
DBE_SCHEDULER.ENABLE	启用对象。
DBE_SCHEDULER.ENABLE_SINGLE	启用单个对象。
DBE_SCHEDULER.DISABLE	停用对象。
DBE_SCHEDULER.DISABLE_SINGLE	停用单个对象。
DBE_SCHEDULER.EVAL_CALENDAR_STRING	分析调度任务周期。
DBE_SCHEDULER.EVALUATE_CALENDAR_STRING	分析调度任务周期。

- DBE_SCHEDULER.CREATE_JOB

  创建一个定时任务。

  DBE_SCHEDULER.CREATE_JOB函数原型可以分为4种：

```
-- 内联调度和程序的定时任务
DBE_SCHEDULER.CREATE_JOB(
job_name TEXT,
job_type TEXT,
job_action TEXT,
number_of_arguments INTEGER             DEFAULT 0,
```



```
start_date TIMESTAMP WITH TIME ZONE     DEFAULT NULL,
repeat_interval TEXT                    DEFAULT NULL,
end_date TIMESTAMP WITH TIME ZONE       DEFAULT NULL,
job_class TEXT            DEFAULT 'DEFAULT_JOB_CLASS',
enabled BOOLEAN                          DEFAULT FALSE,
auto_drop BOOLEAN                         DEFAULT TRUE,
comments TEXT                           DEFAULT NULL,
credential_name TEXT                     DEFAULT NULL,
destination_name TEXT                    DEFAULT NULL
)

-- 引用创建好的调度和程序的定时任务
DBE_SCHEDULER.CREATE_JOB(
job_name TEXT,
program_name TEXT,
schedule_name TEXT,
job_class TEXT            DEFAULT 'DEFAULT_JOB_CLASS',
enabled BOOLEAN                          DEFAULT FALSE,
auto_drop BOOLEAN                         DEFAULT TRUE,
comments TEXT                           DEFAULT NULL,
job_style TEXT                         DEFAULT 'REGULAR',
credential_name TEXT                     DEFAULT NULL,
destination_name TEXT                    DEFAULT NULL
)

-- 引用创建好的程序，内联调度的定时任务
DBE_SCHEDULER.CREATE_JOB(
job_name text,
program_name TEXT,
start_date TIMESTAMP WITH TIME ZONE     DEFAULT NULL,
repeat_interval TEXT                    DEFAULT NULL,
end_date TIMESTAMP WITH TIME ZONE       DEFAULT NULL,
job_class TEXT            DEFAULT 'DEFAULT_JOB_CLASS',
enabled BOOLEAN                          DEFAULT FALSE,
auto_drop BOOLEAN                         DEFAULT TRUE,
comments TEXT                           DEFAULT NULL,
job_style TEXT                         DEFAULT 'REGULAR',
credential_name TEXT                     DEFAULT NULL,
destination_name TEXT                    DEFAULT NULL
)

-- 引用创建好的调度，内联程序的定时任务
DBE_SCHEDULER.CREATE_JOB(
job_name TEXT,
schedule_name TEXT,
job_type TEXT,
job_action TEXT,
number_of_arguments INTEGER             DEFAULT 0,
job_class TEXT            DEFAULT 'DEFAULT_JOB_CLASS',
enabled BOOLEAN                          DEFAULT FALSE,
auto_drop BOOLEAN                         DEFAULT TRUE,
comments TEXT                           DEFAULT NULL,
credential_name TEXT                     DEFAULT NULL,
destination_name TEXT                    DEFAULT NULL
)
```

说明

利用DBE_SCHEDULER创建的定时任务不会与DBE_TASK中的定时任务相冲突。

DBE_SCHEDULER创建的定时任务会生成对应的job_id，但是在使用过程中这个id并没有实际意义。

对于create类型接口，不做入参类型合法性校验，创建成功不代表会执行成功，通过系统表pg_job查询当前任务的执行状态。

表 10-192 DBE_SCHEDULER.CREATE_JOB 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
job_name	text	IN	否	定时任务名称。
job_type	text	IN	否	定时任务内联程序类型，可用类型为： ● 'PLSQL_BLOCK'：匿名存储过程快。 ● 'STORED_PROCEDURE'：保存的存储过程。 ● 'EXTERNAL_SCRIPT'：外部脚本。
job_action	text	IN	否	定时任务内联程序执行内容。
number_of_arguments	integer	IN	否	定时任务内联程序参数个数。
program_name	text	IN	否	定时任务引用程序名称。
start_date	timestamp with time zone	IN	是	定时任务内联调度起始时间。
repeat_interval	text	IN	是	定时任务内联调度任务周期。
end_date	timestamp with time zone	IN	是	定时任务内联调度失效时间。
schedule_name	text	IN	否	定时任务引用调度名称。
job_class	text	IN	否	定时任务类名。
enabled	boolean	IN	否	定时任务启用状态。
auto_drop	boolean	IN	否	定时任务自动删除。
comments	text	IN	是	备注。
job_style	text	IN	否	定时任务行为模式，仅支持‘REGULAR'。

参数	类型	入参/出参	是否可以为空	描述
credential_name	text	IN	是	定时任务证书名。
destination_name	text	IN	是	定时任务目标名。

示例：

```
openGauss=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'select
pg_sleep(1);', 3, false, 'test');
 create_program
----------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.create_schedule('schedule1', NULL, 'sysdate', NULL, 'test');
 create_schedule
-----------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.create_job(job_name=>'job1', program_name=>'program1',
schedule_name=>'schedule1');
 create_job
------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_schedule('schedule1');
 drop_schedule
---------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_program('program1', false);
 drop_program
--------------

(1 row)
```

须知

创建'EXTERNAL_SCRIPT'类型的定时任务需要管理员赋予相关的权限和证书，且需要数据库启动用户对该外部脚本有读取权限才可以正常生效。

- DBE_SCHEDULER.DROP_JOB

  删除定时任务。

  DBE_SCHEDULER.DROP_JOB函数原型为：


```
DBE_SCHEDULER.drop_job(
job_name text,
force boolean                    default false,
defer boolean                    default false,
commit_semantics text      default 'STOP_ON_FIRST_ERROR'
)
```

说明

DBE_SCHEDULER.DROP_JOB可以指定一个或多个任务，或指定任务类进行定时任务删除。

表 10-193 DBE_SCHEDULER.DROP_JOB 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
job_name	text	IN	否	定时任务或定时任务类名称，可以指定一个或多个，当指定多个定时任务时需要利用逗号隔开。
force	boolean	IN	否	删除定时任务行为标记位： ● true：尝试先停止当前定时任务，再进行删除。 ● false：如果定时任务正在运行会删除失败。
defer	boolean	IN	否	删除定时任务行为标记位： ● true：允许定时任务完成后再进行删除。 ● false：不允许定时任务继续执行，尝试进行删除。
commit_semantics	text	IN	否	提交规则： ● 'STOP_ON_FIRST_ERROR'：在第一个报错之前的删除操作会提交。 ● 'TRANSACTIONAL'：事务级提交，报错前的删除操作会回滚。 ● 'ABSORB_ERRORS'：尝试越过报错，将成功的删除操作提交。

示例：

```
openGauss=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'select
pg_sleep(1);', 3, false, 'test');
 create_program
----------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.create_schedule('schedule1', NULL, 'sysdate', NULL, 'test');
 create_schedule
-----------------
```



```
(1 row)

openGauss=# CALL DBE_SCHEDULER.create_job(job_name=>'job1', program_name=>'program1',
schedule_name=>'schedule1');
 create_job
------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_schedule('schedule1');
 drop_schedule
---------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_program('program1', false);
 drop_program
--------------

(1 row)
```

须知

commit_semantic中的'TRANSACTIONAL'选项必须在force为false的情况下才会生效。

- DBE_SCHEDULER.DROP_SINGLE_JOB

  删除一个定时任务。

  DBE_SCHEDULER.DROP_SINGLE_JOB函数原型为：

```
DBE_SCHEDULER.drop_single_job(
job_name text,
force boolean                   default false,
defer boolean                   default false
)
```

表 10-194 DBE_SCHEDULER.DROP_SINGLE_JOB 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
job_name	text	IN	否	定时任务或定时任务类名称。
force	boolean	IN	否	删除定时任务行为标记位： ● true：尝试先停止当前定时任务，再进行删除。 ● false：如果定时任务正在运行会删除失败。

参数	类型	入参/出参	是否可以为空	描述
defer	boolean	IN	否	删除定时任务行为标记位： ● true：允许定时任务完成后再进行删除。 ● false：不允许定时任务继续执行，尝试进行删除。

示例：

```
openGauss=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'select
pg_sleep(1);', 0, false, 'test');
 create_program
----------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.create_job('job1', 'program1', '2021-07-20', 'interval ''3 minute''',
'2121-07-20', 'DEFAULT_JOB_CLASS', false, false,'test', 'style', NULL, NULL);
 create_job
------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_single_job('job1', false, false);
 drop_single_job
-----------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_program('program1', false);
 drop_program
--------------

(1 row)
```

- DBE_SCHEDULER.SET_ATTRIBUTE

  修改定时任务属性。

  DBE_SCHEDULER.SET_ATTRIBUTE函数4种原型为：

```
DBE_SCHEDULER.set_attribute(
name                 text,
attribute            text,
value                boolean
)

DBE_SCHEDULER.set_attribute(
name                 text,
attribute            text,
value                text
)

DBE_SCHEDULER.set_attribute(
name                 text,
attribute            text,
value                timestamp
)

DBE_SCHEDULER.set_attribute(
name                 text,
attribute            text,
value                timestamp with time zone
```



```
)

DBE_SCHEDULER.set_attribute(
name text,
attribute text,
value text,
value2 text                    default NULL
)
```

说明

name在这里可以指定任何DBE_SCHEDULER内部的对象。

表 10-195 DBE_SCHEDULER.SET_ATTRIBUTE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
name	text	IN	否	对象名。
attribute	text	IN	否	属性名。
value	boolean/date/timestamp/timestamp with time zone/text	IN	否	属性值，可选属性如下： 定时任务相关：job_type, job_action, number_of_arguments, start_date, repeat_interval, end_date, job_class, enabled, auto_drop, comments, credential_name, destination_name, program_name, schedule_name, job_style。 调度相关：program_action, program_type, number_of_arguments, comments。 程序相关：start_date, repeat_interval, end_date, comments。
value2	text	IN	是	额外属性值。保留参数位，目前尚不支持拥有额外属性值的目标属性。

示例：

```
openGauss=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'select
pg_sleep(1);', 3, false, 'test');
 create_program
----------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.set_attribute('program1', 'number_of_arguments', 0);
 set_attribute
---------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.set_attribute('program1', 'program_type',
'STORED_PROCEDURE');
 set_attribute
```



```
---------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_program('program1', false);
 drop_program
--------------

(1 row)
```

须知

不要使用DBE_SCHEDULER.SET_ATTRIBUTE来将参数置空。
对象名不能通过DBE_SCHEDULER.SET_ATTRIBUTE来更改。
内联对象不能通过DBE_SCHEDULER.SET_ATTRIBUTE来更改。

- DBE_SCHEDULER.RUN_JOB
  运行定时任务。
  DBE_SCHEDULER.RUN_JOB函数原型为：

```
DBE_SCHEDULER.run_job(
job_name text,
use_current_session boolean             default true
)
```

说明

DBE_SCHEDULER.RUN_JOB主要用于立即运行定时作业，独立于定时任务本身的调度，甚至可以同时运行。

表 10-196 DBE_SCHEDULER.RUN_JOB 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
job_name	text	IN	否	定时任务名称，可以指定一个或多个，当指定多个定时任务时需要利用逗号隔开。
use_current_session	boolean	IN	否	运行定时任务标志位： • true：使用当前会话运行，主要用于查看定时任务是否可以正常运行。 • false：后台拉起定时任务，运行结果会打印到日志中。

示例：

```
openGauss=# SELECT dbe_scheduler.create_job('job1','PLSQL_BLOCK','begin insert into test1
values(12); end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
 create_job
------------

(1 row)
```



```
openGauss=# CALL DBE_SCHEDULER.run_job('job1', false);
 run_job
---------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)
```

须知

use_current_session目前只作用于job_type为'EXTERNAL_SCRIPT'的定时任务上。

- DBE_SCHEDULER.RUN_BACKEND_JOB

  后台运行定时任务。

  DBE_SCHEDULER.RUN_BACKEND_JOB函数原型为：

  ```
  DBE_SCHEDULER.run_backend_job(
  job_name text
  )
  ```

  表 10-197 DBE_SCHEDULER.RUN_BACKEND_JOB 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
job_name	text	IN	否	定时任务名称。

  示例：

  ```
  openGauss=# SELECT dbe_scheduler.create_job('job1','PLSQL_BLOCK','begin insert into test1
  values(12); end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
   create_job
  ------------

  (1 row)

  openGauss=# CALL DBE_SCHEDULER.run_backend_job('job1');
   run_backend_job
  -----------------

  (1 row)

  openGauss=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
   drop_job
  ----------

  (1 row)
  ```

- DBE_SCHEDULER.RUN_FOREGROUND_JOB

  当前会话运行定时任务。

  仅支持运行external 类型任务。

  返回值：text。

  DBE_SCHEDULER.RUN_FOREGROUND_JOB函数原型为：


```
DBE_SCHEDULER.run_foreground_job(
job_name text
)return text
```

表 10-198 DBE_SCHEDULER.RUN_FOREGROUND_JOB 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
job_name	text	IN	否	定时任务名称。

示例：

```
openGauss=# create user test1 identified by '*********';
NOTICE:  The encrypted password contains MD5 ciphertext, which is not secure.
CREATE ROLE
openGauss=# select DBE_SCHEDULER.create_credential('cre_1', 'test1', '*********');
 create_credential
-------------------

(1 row)

openGauss=# select DBE_SCHEDULER.create_job(job_name=>'job1', job_type=>'EXTERNAL_SCRIPT',
job_action=>'/usr/bin/pwd', enabled=>true, auto_drop=>false, credential_name => 'cre_1');
 create_job
------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.run_foreground_job('job1');
  run_foreground_job
---------------------------------
 Host key verification failed.\r+

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_credential('cre_1', false);
 drop_credential
-----------------

(1 row)

openGauss=# drop user test1;
DROP ROLE
```

- DBE_SCHEDULER.STOP_JOB

  终止定时任务。

  DBE_SCHEDULER.STOP_JOB函数原型为：

  ```
  DBE_SCHEDULER.stop_job(
  job_name text,
  force boolean                    default false,
  commit_semantics text                default 'STOP_ON_FIRST_ERROR'
  )
  ```


表 10-199 DBE_SCHEDULER.STOP_JOB 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
job_name	text	IN	否	定时任务或定时任务类名称，可以指定一个或多个，当指定多个定时任务时需要利用逗号隔开。
force	boolean	IN	否	删除定时任务行为标记位： ● true：调度器会发送终止信号立即结束任务线程。 ● false：调度器会尝试利用打断信号终止定时任务线程。
commit_semantics	text	IN	否	提交规则： ● 'STOP_ON_FIRST_ERROR'：在第一个报错之前的打断操作会提交。 ● 'ABSORB_ERRORS'：尝试越过报错，将成功的打断操作提交。

示例：

```
openGauss=# SELECT dbe_scheduler.create_job('job1','PLSQL_BLOCK','begin insert into test1
values(12); end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
 create_job
------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.stop_job('job1', true, 'STOP_ON_FIRST_ERROR');
 stop_job
----------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)
```

- DBE_SCHEDULER.STOP_SINGLE_JOB

  终止单个定时任务。

  DBE_SCHEDULER.STOP_SINGLE_JOB函数原型为：

  ```
  DBE_SCHEDULER.stop_single_job(
  job_name text,
  force boolean                    default false
  )
  ```


表 10-200 DBE_SCHEDULER.STOP_SINGLE_JOB 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
job_name	text	IN	否	定时任务或定时任务类名称。
force	boolean	IN	否	删除定时任务行为标记位： ● true：调度器会发送终止信号立即结束任务线程。 ● false：调度器会尝试利用打断信号终止定时任务线程。

示例：

```
openGauss=# SELECT dbe_scheduler.create_job('job1','PLSQL_BLOCK','begin insert into test1
values(12); end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
 create_job
------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.stop_single_job('job1', true);
 stop_single_job
-----------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)
```

- DBE_SCHEDULER.GENERATE_JOB_NAME

  生成定时任务名称。

  DBE_SCHEDULER.GENERATE_JOB_NAME函数原型为：

  ```
  DBE_SCHEDULER.generate_job_name(
  prefix text                     default 'JOB$_'
  )return text
  ```

  表 10-201 DBE_SCHEDULER.GENERATE_JOB_NAME 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
prefix	text	IN	否	生成名称的前缀，默认为'JOB$_'，反复执行生成的定时任务名为：job$_1, job$_2, job$_3 ...。

  示例：


```
openGauss=# CALL DBE_SCHEDULER.generate_job_name();
 generate_job_name
-------------------
 JOB$_1
(1 row)

openGauss=# CALL DBE_SCHEDULER.generate_job_name();
 generate_job_name
-------------------
 JOB$_2
(1 row)

openGauss=# CALL DBE_SCHEDULER.generate_job_name('job');
 generate_job_name
-------------------
 job3
(1 row)

openGauss=# CALL DBE_SCHEDULER.generate_job_name('job');
 generate_job_name
-------------------
 job4
(1 row)
```

须知

首次执行DBE_SCHEDULER.GENERATE_JOB_NAME会在public下创建一个临时序列用于保存当前名称的序号。由于普通用户没有在public下create权限，因此如果普通用户为当前db下第一次调用该函数，会失败，需要授权该普通用户在public下的create权限，或者使用有该权限的用户调用该接口以创建临时序列。

- DBE_SCHEDULER.CREATE_PROGRAM

  创建程序。

  DBE_SCHEDULER.CREATE_PROGRAM函数原型为：

```
DBE_SCHEDULER.create_program(
program_name text,
program_type text,
program_action text,
number_of_arguments integer          default 0,
enabled boolean                    default false,
comments text                      default NULL
)
```

表 10-202 DBE_SCHEDULER.CREATE_PROGRAM 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
program_name	text	IN	否	程序名称。

参数	类型	入参/出参	是否可以为空	描述
program_type	text	IN	否	程序类型，可用类型为： ● 'PLSQL_BLOCK'：匿名存储过程快。 ● 'STORED_PROCEDURE'：保存的存储过程。 ● 'EXTERNAL_SCRIPT'：外部脚本。
program_action	text	IN	否	程序操作。
number_of_arguments	integer	IN	否	程序采用的参数数量。
enabled	boolean	IN	否	程序启用状态。
comments	text	IN	是	备注

示例：

```
openGauss=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'select
pg_sleep(1);', 3, false, 'test');
 create_program
----------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_program('program1', false);
 drop_program
--------------

(1 row)
```

- DBE_SCHEDULER.DEFINE_PROGRAM_ARGUMENT

  定义程序参数。

  修正带默认值default_value字段的接口默认转换成小写行为，对字符大小写做出区分。

  DBE_SCHEDULER.DEFINE_PROGRAM_ARGUMENT函数原型为：

```
DBE_SCHEDULER.define_program_argument(
program_name text,
argument_position integer,
argument_name text                  default NULL,
argument_type text,
out_argument boolean                 default false
)

-- 带有默认值 --
DBE_SCHEDULER.define_program_argument(
program_name text,
argument_position integer,
argument_name text                  default NULL,
argument_type text,
default_value text,
```



```
out_argument boolean                default false
)
```

表 10-203 DBE_SCHEDULER.DEFINE_PROGRAM_ARGUMENT 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
program_name	text	IN	否	程序名称。
argument_position	integer	IN	否	参数位置。
argument_name	text	IN	否	参数名称。
argument_type	text	IN	否	参数类型。
default_value	text	IN	否	默认值。
out_argument	boolean	IN	否	预留参数。

示例：

```
openGauss=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'select
pg_sleep(1);', 2, false, 'test');
create_program
----------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.define_program_argument('program1', 1, 'pa1', 'type1', false);
 define_program_argument
-------------------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.define_program_argument('program1', 1, 'pa1', 'type1', 'value1',
false);
 define_program_argument
-------------------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_program('program1', false);
 drop_program
--------------

(1 row)
```

- DBE_SCHEDULER.DROP_PROGRAM

  删除程序。

  DBE_SCHEDULER.DROP_PROGRAM函数原型为：

  ```
  DBE_SCHEDULER.drop_program(
  program_name text,
  ```



```
force boolean                    default false
)
```

表 10-204 DBE_SCHEDULER.DROP_PROGRAM 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
program_name	text	IN	否	程序名称。
force	boolean	IN	否	删除程序行为标记位： ● true：在删除程序之前，将禁用应用该程序的所有作业。 ● false：该程序不能被任何作业引用，否则会发送错误。

示例：

```
openGauss=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'select
pg_sleep(1);', 2, false, 'test');
create_program
----------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_program('program1', false);
 drop_program
--------------

(1 row)
```

- DBE_SCHEDULER.DROP_SINGLE_PROGRAM

  删除单个程序。

  DBE_SCHEDULER.DROP_SINGLE_PROGRAM函数原型为：

```
DBE_SCHEDULER.drop_single_program(
program_name text,
force boolean                    default false
)
```

表 10-205 DBE_SCHEDULER.DROP_SINGLE_PROGRAM 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
program_name	text	IN	否	程序名称。

参数	类型	入参/出参	是否可以为空	描述
force	boolean	IN	否	删除程序行为标记位： ● true：在删除程序之前，将禁用应用该程序的所有作业。 ● false：该程序不能被任何作业引用，否则会发送错误。

示例：

```
openGauss=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'select
pg_sleep(1);', 2, false, 'test');
create_program
----------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_single_program('program1', false);
 drop_single_program
---------------------

(1 row)
```

- DBE_SCHEDULER.SET_JOB_ARGUMENT_VALUE

  设置定时任务参数值。argument_value字段支持赋空入参。

  DBE_SCHEDULER.SET_JOB_ARGUMENT_VALUE函数原型为：

```
DBE_SCHEDULER.set_job_argument_value(
job_name text,
argument_position integer,
argument_value text
)

DBE_SCHEDULER.set_job_argument_value(
job_name text,
argument_name text,
argument_value text
)
```

表 10-206 DBE_SCHEDULER.SET_JOB_ARGUMENT_VALUE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
job_name	text	IN	否	定时任务名称。
argument_position	integer	IN	否	参数位置。
argument_name	text	IN	是	参数名称。
argument_value	text	IN	是	参数值。

示例：

```
openGauss=# CALL dbe_scheduler.create_job('job1','EXTERNAL_SCRIPT','begin insert into test1
values(12); end;',2,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
 create_job
------------


(1 row)

openGauss=# CALL DBE_SCHEDULER.set_job_argument_value('job1', 1, 'value1');
 set_job_argument_value
------------------------


(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------


(1 row)
```

- DBE_SCHEDULER.CREATE_SCHEDULE

  创建调度。

  DBE_SCHEDULER.CREATE_SCHEDULE函数原型为：

```
DBE_SCHEDULER.create_schedule(
schedule_name text,
start_date timestamp with time zone      default NULL,
repeat_interval text,
end_date timestamp with time zone        default NULL,
comments text                            default NULL
)
```

表 10-207 DBE_SCHEDULER.CREATE_SCHEDULE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
schedule_name	text	IN	否	调度名称。
start_date	timestamp with time zone	IN	是	调度开始时间。
repeat_interval	text	IN	否	调度重复频率。
end_date	timestamp with time zone	IN	是	调度结束时间。
comments	text	IN	是	备注

示例：

```
openGauss=# CALL DBE_SCHEDULER.create_schedule('schedule1', sysdate, 'sysdate + 3 / (24 * 60 *
60)', null, 'test1');
 create_schedule
-----------------


(1 row)

openGauss=# CALL DBE_SCHEDULER.create_schedule('schedule2', sysdate, 'FREQ=DAILY; BYHOUR=6;',
null, 'test1');
```



```
 create_schedule
-----------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.create_schedule('schedule3', sysdate, 'FREQ=DAILY;
BYHOUR=6;');
 create_schedule
-----------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_schedule('schedule1');
 drop_single_schedule
----------------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_schedule('schedule2', false);
 drop_single_schedule
----------------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_schedule('schedule3', true);
 drop_single_schedule
----------------------

(1 row)
```

- DBE_SCHEDULER.DROP_SCHEDULE

  删除调度。

  DBE_SCHEDULER.DROP_SCHEDULE函数原型为：

```
DBE_SCHEDULER.drop_schedule(
schedule_name text,
force boolean                    default false
)
```

表 10-208 DBE_SCHEDULER.DROP_SCHEDULE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
schedule_name	text	IN	否	调度名称。
force	boolean	IN	否	删除调度行为标记位： ● true：在删除调度之前，将禁用使用此调度的任何作业或窗口。 ● false：调度不能被任何作业或窗口引用，否则会发生错误。

示例：

```
openGauss=# CALL DBE_SCHEDULER.create_schedule('schedule1', sysdate, 'sysdate + 3 / (24 * 60 *
60)', null, 'test1');
 create_schedule
-----------------
```



```
(1 row)

openGauss=# CALL DBE_SCHEDULER.create_schedule('schedule2', sysdate, 'FREQ=DAILY; BYHOUR=6;',
null, 'test1');
 create_schedule
-----------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.create_schedule('schedule3', sysdate, 'FREQ=DAILY;
BYHOUR=6;');
 create_schedule
-----------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_schedule('schedule1');
 drop_single_schedule
----------------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_schedule('schedule2', false);
 drop_single_schedule
----------------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_schedule('schedule3', true);
 drop_single_schedule
----------------------

(1 row)
```

- DBE_SCHEDULER.DROP_SINGLE_SCHEDULE

  删除单个调度。

  DBE_SCHEDULER.DROP_SINGLE_SCHEDULE函数原型为：

```
DBE_SCHEDULER.drop_single_schedule(
schedule_name text,
force boolean                    default false
)
```

表 10-209 DBE_SCHEDULER.DROP_SINGLE_SCHEDULE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
schedule_name	text	IN	否	调度名称。
force	boolean	IN	否	删除调度行为标记位： ● true：在删除调度之前，将禁用使用此调度的任何作业或窗口。 ● false：调度不能被任何作业或窗口引用，否则会发生错误。

示例：

```
openGauss=# CALL DBE_SCHEDULER.create_schedule('schedule1', sysdate, 'sysdate + 3 / (24 * 60 *
60)', null, 'test1');
```



```
 create_schedule
-----------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.create_schedule('schedule2', sysdate, 'FREQ=DAILY; BYHOUR=6;',
null, 'test1');
 create_schedule
-----------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.create_schedule('schedule3', sysdate, 'FREQ=DAILY;
BYHOUR=6;');
 create_schedule
-----------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_single_schedule('schedule1');
 drop_single_schedule
----------------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_single_schedule('schedule2', false);
 drop_single_schedule
----------------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_single_schedule('schedule3', true);
 drop_single_schedule
----------------------

(1 row)
```

- DBE_SCHEDULER.CREATE_JOB_CLASS

  创建定时任务类。

  DBE_SCHEDULER.CREATE_JOB_CLASS函数原型为：

```
DBE_SCHEDULER.create_job_class(
job_class_name text,
resource_consumer_group text           default NULL,
service text                        default NULL,
logging_level integer                default 0,
log_history integer                  default NULL,
comments text                        default NULL
)
```

表 10-210 DBE_SCHEDULER.CREATE_JOB_CLASS 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
job_class_name	text	IN	否	定时任务类名称。
resource_consumer_group	text	IN	是	定时任务类内联资源消费组。
service	text	IN	是	定时任务类内联数据库服务。

参数	类型	入参/出参	是否可以为空	描述
logging_level	integer	IN	否	定时任务类记录信息个数。
log_history	integer	IN	是	定时任务类记录信息保留天数。
comments	text	IN	是	备注

示例：

```
openGauss=# CALL DBE_SCHEDULER.create_job_class(job_class_name => 'jc1',
resource_consumer_group => '123');
 create_job_class
------------------


(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_job_class('jc1', false);
 drop_job_class
----------------


(1 row)
```

- DBE_SCHEDULER.DROP_JOB_CLASS

  删除定时任务类。

  DBE_SCHEDULER.DROP_JOB_CLASS函数原型为：

  ```
  DBE_SCHEDULER.drop_job_class(
  job_class_name text,
  force boolean                    default false
  )
  ```

  表 10-211 DBE_SCHEDULER.DROP_JOB_CLASS 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
job_class_name	text	IN	否	定时任务类名称
force	boolean	IN	否	删除定时任务类行为标记位： ● true：该类的作业将被禁用，并且其他类将设置为默认类，只有在成功的情况下，才会删除该类。 ● false：被删除的类不得被任何作业引用，否则会发生错误。

  示例：

  ```
  openGauss=# CALL DBE_SCHEDULER.create_job_class(job_class_name => 'jc1',
  resource_consumer_group => '123');
  ```



```
 create_job_class
------------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_job_class('jc1', false);
 drop_job_class
----------------

(1 row)
```

- DBE_SCHEDULER.DROP_SINGLE_JOB_CLASS

  删除单个定时任务类。

  DBE_SCHEDULER.DROP_SINGLE_JOB_CLASS函数原型为：

  ```
  DBE_SCHEDULER.drop_single_job_class(
  job_class_name text,
  force boolean                    default false
  )
  ```

  表 10-212 DBE_SCHEDULER.DROP_SINGLE_JOB_CLASS 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
job_class_name	text	IN	否	定时任务类名称。
force	boolean	IN	否	删除定时任务类行为标记位： ● true：该类的作业将被禁用，并且其他类将设置为默认类，只有在成功的情况下，才会删除该类。 ● false：被删除的类不得被任何作业引用，否则会发生错误。

  示例：

  ```
  openGauss=# CALL DBE_SCHEDULER.create_job_class(job_class_name => 'jc1',
  resource_consumer_group => '123');
   create_job_class
  ------------------

  (1 row)

  openGauss=# CALL DBE_SCHEDULER.drop_single_job_class('jc1', false);
   drop_single_job_class
  -----------------------

  (1 row)
  ```

- DBE_SCHEDULER.GRANT_USER_AUTHORIZATION

  为数据库用户提供定时任务权限。调用该函数的用户需要具有SYSADMIN权限。

  DBE_SCHEDULER.GRANT_USER_AUTHORIZATION函数原型为：

  ```
  DBE_SCHEDULER.grant_user_authorization(
  username              text,
  privilege             text
  )
  ```


表 10-213 DBE_SCHEDULER.GRANT_USER_AUTHORIZATION 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
username	text	IN	否	数据库用户名称。
privilege	text	IN	否	定时任务权限。

示例：

```
openGauss=# create user user1 password '1*s*****';
NOTICE:  The encrypted password contains MD5 ciphertext, which is not secure.
CREATE ROLE
openGauss=# CALL DBE_SCHEDULER.grant_user_authorization('user1', 'create job');
 grant_user_authorization
--------------------------

(1 row)
openGauss=# drop user user1;
DROP ROLE
```

- DBE_SCHEDULER.REVOKE_USER_AUTHORIZATION

  撤销数据库用户的定时任务权限。调用该函数的用户需要具有SYSADMIN权限。

  DBE_SCHEDULER.REVOKE_USER_AUTHORIZATION函数原型为：

  ```
  DBE_SCHEDULER.revoke_user_authorization(
  username              text,
  privilege             text
  )
  ```

  表 10-214 DBE_SCHEDULER.REVOKE_USER_AUTHORIZATION 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
username	text	IN	否	数据库用户名称。
privilege	text	IN	否	定时任务权限。

  示例：

  ```
  openGauss=# create user user1 password '1*s*****';
  NOTICE:  The encrypted password contains MD5 ciphertext, which is not secure.
  CREATE ROLE
  openGauss=# CALL DBE_SCHEDULER.grant_user_authorization('user1', 'create job');
   grant_user_authorization
  --------------------------

  (1 row)

  openGauss=# CALL DBE_SCHEDULER.revoke_user_authorization('user1', 'create job');
   revoke_user_authorization
  ---------------------------

  (1 row)
  openGauss=# drop user user1;
  DROP ROLE
  ```


- DBE_SCHEDULER.CREATE_CREDENTIAL
  创建授权证书。调用该函数的用户需要具有SYSADMIN权限。
  DBE_SCHEDULER.CREATE_CREDENTIAL函数原型为：

```
DBE_SCHEDULER.create_credential(
credential_name      text,
username             text,
password             text          default NULL,
database_role        text          default NULL,
windows_domain       text          default NULL,
comments             text          default NULL
)
```

表 10-215 DBE_SCHEDULER.CREATE_CREDENTIAL 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
credential_name	text	IN	否	授权证书名称。
username	text	IN	否	数据库用户名称。
password	text	IN	是	用户密码。
database_role	text	IN	是	数据库系统权限。
windows_domain	text	IN	是	Windows用户所属域。
comments	text	IN	是	备注

示例：

```
openGauss=# CALL DBE_SCHEDULER.create_credential('cre_1', 'user1', '');
 create_credential
-------------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_credential('cre_1', false);
 drop_credential
-----------------

(1 row)
```

须知

DBE_SCHEDULER.CREATE_CREDENTIAL的password字段请传NULL或者'******',该参数仅做兼容性，不代表实际含义。禁止使用安装用户对应的os用户名创建证书。

- DBE_SCHEDULER.DROP_CREDENTIAL
  销毁授权证书。调用该函数的用户需要具有SYSADMIN权限。
  DBE_SCHEDULER.DROP_CREDENTIAL函数原型为：

```
DBE_SCHEDULER.drop_credential(
credential_name      text,
```



```
force boolean default false
)
```

表 10-216 DBE_SCHEDULER.DROP_CREDENTIAL 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
credential_name	text	IN	否	授权证书名称。
force	boolean	IN	否	删除授权证书行为标记位： ● true：无论是否有作业引用该证书，都会被删除。 ● false：任何作业都无法引用该证书，否则会发生错误。

示例：

```
openGauss=# CALL DBE_SCHEDULER.create_credential('cre_1', 'user1', '');
 create_credential
-------------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_credential('cre_1', false);
 drop_credential
-----------------

(1 row)
```

- DBE_SCHEDULER.ENABLE

  启用对象。

  DBE_SCHEDULER.ENABLE函数原型为：

  ```
  DBE_SCHEDULER.enable(
  name text,
  commit_semantics text                default 'STOP_ON_FIRST_ERROR'
  )
  ```

  表 10-217 DBE_SCHEDULER.ENABLE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
name	text	IN	否	对象名称，可以指定一个或多个，当指定多个程序时需利用逗号隔开。

参数	类型	入参/出参	是否可以为空	描述
commit_semantics	text	IN	否	提交规则。支持以下类型： ● 'STOP_ON_FIRST_ERROR'：在第一个报错之前的启用操作会提交。 ● 'TRANSACTIONAL'：事务级提交，报错前的启用操作会回滚。 ● 'ABSORB_ERRORS'：尝试越过报错，将成功的启用操作提交。

示例：

```
openGauss=# CALL dbe_scheduler.create_job('job1','PLSQL_BLOCK','begin insert into test1 values(12);
end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
lse, 'test');
 create_job
------------


(1 row)

openGauss=# CALL DBE_SCHEDULER.create_program('program1', 'stored_procedure', 'insert into
tb_job_test(key) values(null);', 0, false, '');
 create_program
----------------


(1 row)

openGauss=# CALL DBE_SCHEDULER.enable('job1');
 enable
--------


(1 row)

openGauss=# CALL DBE_SCHEDULER.enable('program1', 'STOP_ON_FIRST_ERROR');
 enable
--------


(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------


(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_program('program1', false);
 drop_program
--------------


(1 row)
```

- DBE_SCHEDULER.ENABLE_SINGLE

  启用单个对象。

  DBE_SCHEDULER.ENABLE_SINGLE函数原型为：

  ```
  DBE_SCHEDULER.enable_single(
  name text
  )
  ```

表 10-218 DBE_SCHEDULER.ENABLE_SINGLE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
name	text	IN	否	对象名称。

示例：

```
openGauss=# CALL dbe_scheduler.create_job('job1','PLSQL_BLOCK','begin insert into test1 values(12);
end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
lse, 'test');
 create_job
------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.enable_single('job1');
 enable_single
---------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)
```

- DBE_SCHEDULER.DISABLE

  禁用多个对象，name为逗号分隔的字符串，每个逗号分隔的字符串为一个对象。

  DBE_SCHEDULER.DISABLE函数原型为：

  ```
  DBE_SCHEDULER.disable(
  name text,
  force boolean                    default false,
  commit_semantics text                default 'STOP_ON_FIRST_ERROR'
  )
  ```

表 10-219 DBE_SCHEDULER.DISABLE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
name	text	IN	否	对象名称。
force	boolean	IN	否	禁用对象行为标记位： ● true：无论是否有其他对象依赖于该对象，也会被禁用。 ● false：任何对象都无法依赖于该对象，否则会发生错误。

参数	类型	入参/出参	是否可以为空	描述
commit_semantics	text	IN	否	提交规则。支持以下类型： ● 'STOP_ON_FIRST_ERROR'：在第一个报错之前的禁用操作会提交。 ● 'TRANSACTIONAL'：事务级提交，报错前的禁用操作会回滚。 ● 'ABSORB_ERRORS'：尝试越过报错，将成功的禁用操作提交。

示例：

```
openGauss=# CALL dbe_scheduler.create_job('job1','PLSQL_BLOCK','begin insert into test1 values(12);
end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
 create_job
------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.create_program('program1', 'stored_procedure', 'insert into
tb_job_test(key) values(null);', 0, false, '');
 create_program
----------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.disable('job1');
 disable
---------

(1 row)

openGauss=# CALL DBE_SCHEDULER.disable('program1', false, 'STOP_ON_FIRST_ERROR');
 disable
---------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_program('program1', false);
 drop_program
--------------

(1 row)
```

- DBE_SCHEDULER.DISABLE_SINGLE

  禁用单个对象。

  DBE_SCHEDULER.DISABLE_SINGLE函数原型为：

  ```
  DBE_SCHEDULER.disable_single(
  name text,
  force boolean                    default false
  )
  ```


表 10-220 DBE_SCHEDULER.DISABLE_SINGLE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
name	text	IN	否	对象名称。
force	boolean	IN	否	禁用对象行为标记位： ● true：无论是否有其他对象依赖于该对象，也会被禁用。 ● false：任何对象都无法依赖于该对象，否则会发生错误。

示例：

```
openGauss=# CALL dbe_scheduler.create_job('job1','PLSQL_BLOCK','begin insert into test1 values(12);
end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
 create_job
------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.disable_single('job1', false);
 disable_single
----------------

(1 row)

openGauss=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)
```

- DBE_SCHEDULER.EVAL_CALENDAR_STRING

  分析调度任务周期。

  返回值类型：timestamp with time zone

  DBE_SCHEDULER.EVAL_CALENDAR_STRING函数原型为：

  ```
  DBE_SCHEDULER.evaluate_calendar_string(
  IN calendar_string text,
  IN start_date timestamp with time zone,
  IN return_date_after timestamp with time zone
  )return timestamp with time zone
  ```

  表 10-221 DBE_SCHEDULER.EVAL_CALENDAR_STRING 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
calendar_string	text	IN	否	定时任务日期字符串。
start_date	timestamp with time zone	IN	否	定时任务开始时间。

参数	类型	入参/出参	是否可以为空	描述
return_date_after	timestamp with time zone	IN	否	定时任务返回日期。

示例：

```
openGauss=# CALL DBE_SCHEDULER.eval_calendar_string('FREQ=DAILY; BYHOUR=6;', sysdate,
sysdate);
 eval_calendar_string
-----------------------
 2023-09-15 06:47:24+08
(1 row)
```

- DBE_SCHEDULER.EVALUATE_CALENDAR_STRING

  分析调度任务周期。

  返回值类型: timestamp with time zone

  DBE_SCHEDULER.EVALUATE_CALENDAR_STRING函数原型为：

  ```
  DBE_SCHEDULER.evaluate_calendar_string(
  IN calendar_string text,
  IN start_date timestamp with time zone,
  IN return_date_after timestamp with time zone,
  OUT next_run_date timestamp with time zone
  )return timestamp with time zone
  ```

  表 10-222 DBE_SCHEDULER.EVALUATE_CALENDAR_STRING 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
calendar_string	text	IN	否	定时任务日期字符串。
start_date	timestamp with time zone	IN	否	定时任务开始时间。
return_date_after	timestamp with time zone	IN	否	定时任务返回日期。
next_run_date	timestamp with time zone	OUT	否	定时任务返回下一个日期。

  示例：

  ```
  openGauss=# CREATE OR REPLACE PROCEDURE pr1(calendar_str text) as
  DECLARE
     start_date         timestamp with time zone;
     return_date_after timestamp with time zone;
     next_run_date      timestamp with time zone;
  BEGIN
     start_date := '2003-2-1 10:30:00.111111+8'::timestamp with time zone;
     return_date_after := start_date;
     DBE_SCHEDULER.evaluate_calendar_string(
        calendar_str,
        start_date, return_date_after, next_run_date);
     DBE_OUTPUT.PRINT_LINE('next_run_date: ' || next_run_date);
  ```



```
    return_date_after := next_run_date;
END;
/
CREATE PROCEDURE

openGauss=# CALL pr1('FREQ=hourly;INTERVAL=2;BYHOUR=6,10;BYMINUTE=0;BYSECOND=0');
next_run_date: 2003-02-02 06:00:00+08
 pr1
-----

(1 row)
```

10.12.2.12 DBE_APPLICATION_INFO

接口介绍

高级功能包DBE_APPLICATION_INFO支持的所有接口请参见**表10-223**。
DBE_APPLICATION_INFO作用范围是当前session。

表 10-223 DBE_APPLICATION_INFO

接口名称	描述
DBE_APPLICATION_INFO.SET_CLIENT_INFO	写入客户端信息
DBE_APPLICATION_INFO.READ_CLIENT_INFO	读取客户端信息

- DBE_APPLICATION_INFO.SET_CLIENT_INFO

写入客户端信息。DBE_APPLICATION_INFO.SET_CLIENT_INFO函数原型为：

```
DBE_APPLICATION_INFO.SET_CLIENT_INFO(
    str text
)returns void;
```

表 10-224 DBE_APPLICATION_INFO.SET_CLIENT_INFO 接口参数说明

参数	描述
str	写入的客户端信息

- DBE_APPLICATION_INFO.READ_CLIENT_INFO

读取客户端信息DBE_APPLICATION_INFO.READ_CLIENT_INFO函数原型为：

```
DBE_APPLICATION_INFO.READ_CLIENT_INFO(
 OUT client_info text);
```


表 10-225 DBE_APPLICATION_INFO.READ_CLIENT_INFO 接口参数说明

参数	描述
client_info	客户端信息

10.13 Retry 管理

Retry是数据库在SQL或存储过程（包含匿名块）执行失败时，在数据库内部进行重新执行的过程，以提高执行成功率和用户体验。数据库内部通过检查发生错误时的错误码及Retry相关配置，决定是否进行重试。

- 失败时回滚之前执行的语句，并重新执行存储过程进行Retry。

  示例：

```
openGauss=# CREATE OR REPLACE PROCEDURE retry_basic ( IN  x INT)
AS
BEGIN
    INSERT INTO t1 (a) VALUES (x);
    INSERT INTO t1 (a) VALUES (x+1);
END;
/
openGauss=# CALL retry_basic(1);
```

10.14 调试

语法

RAISE有以下五种语法格式：

图 10-34 raise_format::=

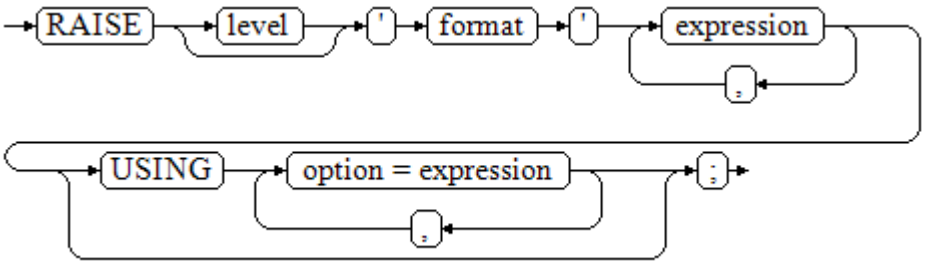


图 10-35 raise_condition::=

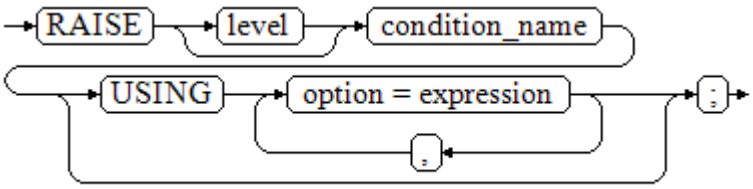


图 10-36 raise_sqlstate::=

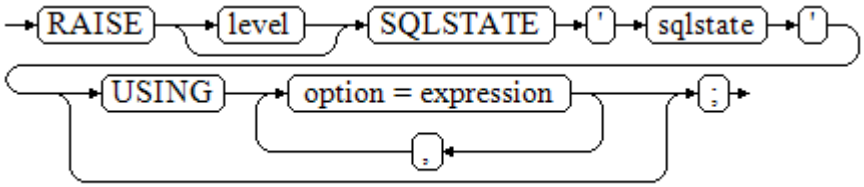


图 10-37 raise_option::=

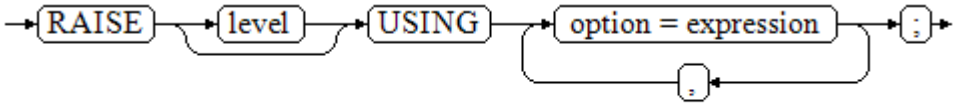
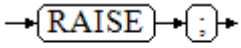


图 10-38 raise::=



参数说明：

- level选项用于指定错误级别，有DEBUG，LOG，INFO，NOTICE，WARNING以及EXCEPTION（默认值）。EXCEPTION抛出一个正常终止当前事务的异常，其他的仅产生不同异常级别的信息。特殊级别的错误信息是否报告到客户端、写到服务器日志由log_min_messages和client_min_messages这两个配置参数控制。
- format：格式字符串，指定要报告的错误消息文本。格式字符串后可跟表达式，用于向消息文本中插入。在格式字符串中，%由format后面跟着的参数的值替换，%%用于打印出%。例如：

  ```
  --v_job_id 将替换字符串中的 %:
  RAISE NOTICE 'Calling cs_create_job(%)',v_job_id;
  ```

- option = expression：向错误报告中添加另外的信息。关键字option可以是MESSAGE、DETAIL、HINT以及ERRCODE，并且每一个expression可以是任意的字符串。
  - MESSAGE，指定错误消息文本，这个选项不能用于在USING前包含一个格式字符串的RAISE语句中。
  - DETAIL，说明错误的详细信息。
  - HINT，用于打印出提示信息。
  - ERRCODE，向报告中指定错误码（SQLSTATE）。可以使用条件名称或者直接用五位字符的SQLSTATE错误码。
- condition_name：错误码对应的条件名。
- sqlstate：错误码。

如果在RAISE EXCEPTION命令中既没有指定条件名也没有指定SQLSTATE，默认用RAISE EXCEPTION (P0001)。如果没有指定消息文本，默认用条件名或者SQLSTATE作为消息文本。

须知

- 当由SQLSTATE指定了错误码，则不局限于已定义的错误码，可以选择任意包含五个数字或者大写的ASCII字母的错误码，而不是00000。建议避免使用以三个0结尾的错误码，因为这种错误码是类别码，会被整个种类捕获。
- 兼容O模式下，SQLCODE等于SQLSTATE。

说明

图10-38所示的语法不接任何参数。这种形式仅用于一个BEGIN块中的EXCEPTION语句，它使得错误重新被处理。

示例

终止事务时，给出错误和提示信息：

```
CREATE OR REPLACE PROCEDURE proc_raise1(user_id in integer)
AS
BEGIN
RAISE EXCEPTION 'Noexistence ID --> %',user_id USING HINT = 'Please check your user ID';
END;
/

call proc_raise1(300011);

--执行结果
ERROR:  Noexistence ID --> 300011
HINT:  Please check your user ID
```

两种设置SQLSTATE的方式：

```
CREATE OR REPLACE PROCEDURE proc_raise2(user_id in integer)
AS
BEGIN
RAISE 'Duplicate user ID: %',user_id USING ERRCODE = 'unique_violation';
END;
/

\set VERBOSITY verbose
call proc_raise2(300011);

--执行结果
ERROR:  Duplicate user ID: 300011
SQLSTATE: 23505
```

如果主要的参数是条件名或者是SQLSTATE，可以使用：

RAISE division_by_zero;

RAISE SQLSTATE '22012';

例如：

```
CREATE OR REPLACE PROCEDURE division(div in integer, dividend in integer)
AS
DECLARE
res int;
    BEGIN
    IF dividend=0 THEN
        RAISE division_by_zero;
        RETURN;
    ELSE
        res := div/dividend;
        RAISE INFO 'division result: %', res;
        RETURN;
```



```
    END IF;
    END;
/
call division(3,0);

--执行结果
ERROR:  division_by_zero
```

或者另一种方式：

```
RAISE unique_violation USING MESSAGE = 'Duplicate user ID: ' || user_id;
```


11 自治事务

自治事务（Autonomous Transaction），在主事务执行过程中新启的独立的事务。自治事务的提交和回滚不会影响主事务已提交的数据，同时自治事务也不受主事务影响。

自治事务在存储过程、函数和匿名块中定义，用PRAGMA AUTONOMOUS_TRANSACTION关键字来声明。

11.1 存储过程支持自治事务

自治事务可以在存储过程中定义，标识符为PRAGMA AUTONOMOUS_TRANSACTION，其余语法与创建存储过程语法相同，示例如下。

```
--建表
create table t2(a int, b int);
insert into t2 values(1,2);
select * from t2;

--创建包含自治事务的存储过程
CREATE OR REPLACE PROCEDURE autonomous_4(a int, b int)  AS
DECLARE
    num3 int := a;
    num4 int := b;
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    insert into t2 values(num3, num4);
    dbe_output.print_line('just use call.');
END;
/
--创建调用自治事务存储过程的普通存储过程
CREATE OR REPLACE PROCEDURE autonomous_5(a int, b int)  AS
DECLARE
BEGIN
    dbe_output.print_line('just no use call.');
    insert into t2 values(666, 666);
    autonomous_4(a,b);
    rollback;
END;
/
--调用普通存储过程
select autonomous_5(11,22);
--查看表结果
select * from t2 order by a;
```

上述例子，最后在回滚的事务块中执行包含自治事务的存储过程，直接说明了自治事务的特性，即主事务的回滚，不会影响自治事务已经提交的内容。

11.2 匿名块支持自治事务

自治事务可以在匿名块中定义，标识符为PRAGMA AUTONOMOUS_TRANSACTION，其余语法与创建匿名块语法相同，示例如下。

```
create table t1(a int ,b text);

START TRANSACTION;
DECLARE
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    dbe_output.print_line('just use call.');
    insert into t1 values(1,'you are so cute,will commit!');
END;
/
insert into t1 values(1,'you will rollback!');
rollback;

select * from t1;
```

上述例子，最后在回滚的事务块前执行包含自治事务的匿名块，也能直接说明了自治事务的特性，即主事务的回滚，不会影响自治事务已经提交的内容。

11.3 函数支持自治事务

自治事务可以在函数中定义，标识符为PRAGMA AUTONOMOUS_TRANSACTION，其余语法与函数语法相同，示例如下。

```
create table t4(a int, b int, c text);

CREATE OR REPLACE function autonomous_32(a int ,b int ,c text) RETURN int AS
DECLARE
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    insert into t4 values(a, b, c);
    return 1;
END;
/
CREATE OR REPLACE function autonomous_33(num1 int) RETURN int AS
DECLARE
    num3 int := 220;
    tmp int;
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    num3 := num3/num1;
    return num3;
EXCEPTION
    WHEN division_by_zero THEN
        select autonomous_32(num3, num1, sqlerrm) into tmp;
        return 0;
END;
/

select autonomous_33(0);

select * from t4;
```


11.4 规格约束

⚠ 注意

- 自治事务执行时，将会在后台启动自治事务session，可以通过max_concurrent_autonomous_transactions设置自治事务执行的最大并行数量，取值范围：0~1024，默认值：10。
- 当max_concurrent_autonomous_transactions参数设置为0时，自治事务将无法执行。
- 自治事务新启session后，将使用默认session参数，不共享主session下对象（包括session级别变量，本地临时变量，全局临时表的数据等）。
- 自治事务受通信缓冲区影响，返回给客户端的信息大小受限于通信缓冲区长度，超过通信缓冲区长度时报错。

- 触发器函数不支持自治事务。

```
CREATE TABLE test_trigger_des_tbl(id1 INT, id2 INT, id3 INT);

CREATE OR REPLACE FUNCTION tri_insert_func() RETURNS TRIGGER AS
$$
DECLARE
 PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
INSERT INTO test_trigger_des_tbl VALUES(NEW.id1, NEW.id2, NEW.id3);
RETURN NEW;
END
$$ LANGUAGE plpgsql;
```

- 自治事务不支持非顶层匿名块调用（仅支持顶层自治事务,包括存储过程、函数、匿名块）。

```
create table t1(a int ,b text);

DECLARE
 --PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
 DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
 BEGIN
  dbe_output.print_line('just use call.');
  insert into t1 values(1,'can you rollback!');
 END;
 insert into t1 values(2,'I will rollback!');
 rollback;
END;
/

select * from t1;
```

- 自治事务不支持ref_cursor参数传递。

```
create table sections(section_ID int);
insert into sections values(1);
insert into sections values(1);
insert into sections values(1);
insert into sections values(1);

CREATE OR REPLACE function proc_sys_ref()
return SYS_REFCURSOR
IS
declare
  PRAGMA AUTONOMOUS_TRANSACTION;
```



```
        C1 SYS_REFCURSOR;
BEGIN
 OPEN C1 FOR SELECT section_ID FROM sections ORDER BY section_ID;
 return C1;
END;
/

CREATE OR REPLACE function proc_sys_ref(OUT C2 SYS_REFCURSOR, OUT a int)
return SYS_REFCURSOR
IS
declare
  PRAGMA AUTONOMOUS_TRANSACTION;
        C1 SYS_REFCURSOR;
BEGIN
 OPEN C1 FOR SELECT section_ID FROM sections ORDER BY section_ID;
 return C1;
END;
/
```

- 分布式自治事务不支持下推（IMMUTABLE,STABLE类型）。

```
CREATE OR REPLACE procedure autonomous_test_in_p_116(num1 int )
IMMUTABLE
AS
DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
perform pg_sleep(1);
END;
/

CREATE OR REPLACE procedure autonomous_test_in_p_117(num1 int ) STABLE AS
DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
perform pg_sleep(1);
END;
/
```

- 分布式不支持检测（死锁时，有锁等待超时报错）。

```
create table test_lock (id int,a date);
insert into test_lock values (10,sysdate),(11,sysdate),(12,sysdate);
CREATE OR REPLACE FUNCTION autonomous_test_lock(num1 int,num2 int) RETURNS
integer LANGUAGE plpgsql AS $$
DECLARE num3 int := 4;
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
update test_lock set a=sysdate where id =11;
RETURN num1+num2+num3;
END;
$$;
start transaction;
update test_lock set a=sysdate where id =11;
call autonomous_test_lock(1,1);
END;
```

- 自治事务函数不支持返回非out形式的record类型。
- 不支持修改自治事务的隔离级别。
- 不支持自治事务返回集合类型（setof）。

12 系统表和系统视图

12.1 系统表和系统视图概述

系统表是GaussDB存放结构元数据的地方，是GaussDB数据库系统运行控制信息的来源，是数据库系统的核心组成部分。

系统视图提供了查询系统表和访问数据库内部状态的方法。

系统表和系统视图要么只对管理员可见，要么对所有用户可见。下面的系统表和视图有些标识了需要管理员权限，这些系统表和视图只有管理员可以查询。

用户可以删除后重新创建这些表、增加列、插入和更新数值，但是用户修改系统表会导致系统信息的不一致，从而导致系统控制紊乱。正常情况下不应该由用户手工修改系统表或系统视图，或者手工重命名系统表或系统视图所在的模式，而是由SQL语句关联的系统表操作自动维护系统表信息。

须知

- 不建议用户修改系统表和系统视图的权限。
- 用户应该禁止对系统表进行增删改等操作，人为对系统表的修改或破坏可能会导致系统各种异常情况甚至集群不可用。
- gs_package系统表在分布式场景下能够查询到但是无意义。
- 系统表和系统视图中与外键相关的字段暂不支持。
- 系统表和系统视图中的字段类型详见**数据类型**章节介绍。

12.2 系统表

12.2.1 GS_AUDITING_POLICY

GS_AUDITING_POLICY系统表记录统一审计的主体信息，每条记录对应一个设计策略。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

表 12-1 GS_AUDITING_POLICY 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
polname	name	策略名称，需要唯一，不可重复。
polcomments	name	策略描述字段，记录策略相关的描述信息，通过COMMENTS关键字体现。
modifydate	timestamp without time zone	策略创建或修改的最新时间戳。
polenabled	boolean	用来表示策略启动开关。 • t（true）：表示策略启动。 • f（false）：表示策略没有启动。

12.2.2 GS_AUDITING_POLICY_ACCESS

GS_AUDITING_POLICY_ACCESS系统表记录与DML数据库相关操作的统一审计信息。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

表 12-2 GS_AUDITING_POLICY_ACCESS 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
accesstype	name	DML数据库操作相关类型。例如SELECT、INSERT、DELETE等。
labelname	name	资源标签名称。对应系统表GS_AUDITING_POLICY中的polname字段。
policyoid	oid	所属审计策略的oid。对应系统表GS_AUDITING_POLICY中的oid。
modifydate	timestamp without time zone	创建或修改的最新时间戳。

12.2.3 GS_AUDITING_POLICY_FILTERS

GS_AUDITING_POLICY_FILTERS系统表记录统一审计相关的过滤策略相关信息，每条记录对应一个设计策略。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

表 12-3 GS_AUDITING_POLICY_FILTERS 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
filtertype	name	过滤类型。目前值仅为logical_expr。
labelname	name	名称。目前值仅为logical_expr。
policyoid	oid	所属审计策略的oid，对应审计策略系统表GS_AUDITING_POLICY中的oid。
modifydate	timestamp without time zone	创建或修改的最新时间戳。
logicaloperator	text	过滤条件的逻辑字符串。

12.2.4 GS_AUDITING_POLICY_PRIVILEGES

GS_AUDITING_POLICY_PRIVILEGES系统表记录统一审计DDL数据库相关操作信息，每条记录对应一个设计策略。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

表 12-4 GS_AUDITING_POLICY_PRIVI 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
privilegetype	name	DDL数据库操作相关类型。例如CREATE、ALTER、DROP等。
labelname	name	资源标签名称。对应系统表GS_AUDITING_POLICY中的polname字段。
policyoid	oid	对应审计策略系统表GS_AUDITING_POLICY中的oid。
modifydate	timestamp without time zone	创建或修改的最新时间戳。

12.2.5 GS_ASP

GS_ASP显示被持久化的ACTIVE SESSION PROFILE样本。该系统表只能在系统库中查询。

表 12-5 GS_ASP 字段

名称	类型	描述
sampleid	bigint	采样ID。
sample_time	timestamp with time zone	采样的时间。
need_flush_sample	boolean	该样本是否需要刷新到磁盘。 ● t（true）：表示需要。 ● f（false）：表示不需要。
databaseid	oid	数据库ID。
thread_id	bigint	线程的ID。
sessionid	bigint	会话的ID。
start_time	timestamp with time zone	会话的启动时间。
event	text	具体的事件名称。
lwtid	integer	当前线程的轻量级线程号。
psessionid	bigint	streaming线程的父线程。
tlevel	integer	streaming线程的层级。与执行计划的层级(id)相对应。
smpid	integer	smp执行模式下并行线程的并行编号。
userid	oid	session用户的id。
application_name	text	应用的名字。
client_addr	inet	client端的地址。
client_hostname	text	client端的名字。
client_port	integer	客户端用于与后端通讯的TCP端口号。
query_id	bigint	debug query id。
unique_query_id	bigint	unique query id。
user_id	oid	unique query的key中的user_id。
cn_id	integer	表示该unique sql来自哪个CN节点。unique query的key中的cn_id。
unique_query	text	-规范化后的Unique SQL文本串。
locktag	text	会话等待锁信息，可通过locktag_decode解析。

名称	类型	描述
lockmode	text	会话等待锁模式： ● LW_EXCLUSIVE：排他锁 ● LW_SHARED：共享锁 ● LW_WAIT_UNTIL_FREE：等待LW_EXCLUSIVE可用
block_sessionid	bigint	如果会话正在等待锁，阻塞该会话获取锁的会话标识。
wait_status	text	描述event列的更多详细信息。
global_sessionid	text	全局会话ID。
xact_start_time	timestamp with time zone	事务开始时间。
query_start_time	timestamp with time zone	语句开始执行时间。
state	text	当前语句状态。 可能取值为：active、idle in transaction、fastpath function call、idle in transaction (aborted)、disabled、retrying。

12.2.6 GS_DB_PRIVILEGE

GS_DB_PRIVILEGE系统表记录ANY权限的授予情况，每条记录对应一条授权信息。

表 12-6 GS_DB_PRIVILEGE 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
roleid	oid	用户标识。
privilege_type	text	用户拥有的ANY权限，取值参考**表7-114**。
admin_option	boolean	是否具有privilege_type列记录的ANY权限的再授权权限。 ● t：表示具有。 ● f：表示不具有。

12.2.7 GS_GLOBAL_CONFIG

GS_GLOBAL_CONFIG记录了集群初始化时，用户指定的参数值。除此之外，还存放了用户设置的弱口令，支持数据库初始用户通过ALTER和DROP语法对系统表中的参数进行写入、修改和删除。

表 12-7 GS_GLOBAL_CONFIG 字段

名称	类型	描述
name	name	集群初始化时系统内置的指定参数名称、弱口令名称、或用户需要使用的参数。
value	text	集群初始化时系统内置的指定参数值、弱口令、或用户需要使用的参数值。

12.2.8 GS_JOB_ATTRIBUTE

GS_JOB_ATTRIBUTE系统表提供了DBE_SCHEDULER定时任务的相关属性信息，其中包括定时任务，定时任务类，证书，授权，程序和调度的基本属性。

表 12-8 GS_JOB_ATTRIBUTE 字段

名称	类型	描述
oid	oid	行标识符（隐含字段）。
job_name	text	定时任务，定时任务类，证书，程序和调度的名字，授权的用户名。
attribute_name	text	定时任务，定时任务类，证书，程序和调度的属性名，授权的内容。
attribute_value	text	定时任务，定时任务类，证书，程序和调度的属性值。

12.2.9 GS_JOB_ARGUMENT

GS_JOB_ARGUMENT系统表提供了DBE_SCHEDULER定时任务和程序的参数属性。

表 12-9 GS_JOB_ARGUMENT 字段

名称	类型	描述
oid	oid	行标识符（隐含字段）。
argument_position	integer	定时任务或程序的参数位置。
argument_type	name	定时任务或程序的参数类型。
job_name	text	定时任务或程序名。

名称	类型	描述
argument_name	text	定时任务或程序的参数名（定时任务继承了程序的参数名，所以为空）。
argument_value	text	定时任务的参数值（程序本身无法绑定值）。
default_value	text	程序的参数默认值。

12.2.10 GS_MASKING_POLICY

GS_MASKING_POLICY系统表记录动态数据脱敏策略的主体信息，每条记录对应一个脱敏策略。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

表 12-10 GS_MASKING_POLICY 表字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
polname	name	策略名称，唯一不可重复。
polcomments	name	策略描述字段，记录策略相关的描述信息，通过COMMENTS关键字体现。
modifydate	timestamp without time zone	策略创建或修改的最新时间戳。
polenabled	boolean	策略启动开关。 ● t（true）：表示策略启动。 ● f（false）：表示策略没有启动。

12.2.11 GS_MASKING_POLICY_ACTIONS

GS_MASKING_POLICY_ACTIONS系统表记录动态数据脱敏策略中相应的脱敏策略包含的脱敏行为，一个脱敏策略对应着该表的一行或多行记录。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

表 12-11 GS_MASKING_POLICY_ACTIONS 表字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
actiontype	name	脱敏函数，标识脱敏策略使用的脱敏函数。

名称	类型	描述
actparams	name	向脱敏函数中传递的参数信息。
actlabelname	name	被脱敏的label名称。
policyoid	oid	该条记录所属的脱敏策略的oid，对应GS_MASKING_POLICY中的oid。
actmodifydate	timestamp without time zone	该条记录创建或修改的最新时间戳。

12.2.12 GS_MASKING_POLICY_FILTERS

GS_MASKING_POLICY_FILTERS系统表记录动态数据脱敏策略对应的用户过滤条件，当用户条件满足FILTER条件时，对应的脱敏策略才会生效。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

表 12-12 GS_MASKING_POLICY_FILTERS 表字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
filtertype	name	过滤类型。目前值仅为logical_expr。
filterlabelname	name	过滤范围。目前值仅为logical_expr。
policyoid	oid	该条记录所属的脱敏策略的oid，对应GS_MASKING_POLICY中的oid。
modifydate	timestamp without time zone	该条用户过滤条件创建或修改的最新时间戳。
logicaloperator	text	过滤条件的波兰表达式。

12.2.13 GS_MATVIEW

GS_MATVIEW系统表提供了关于数据库中每一个物化视图的信息。

表 12-13 GS_MATVIEW 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
matviewid	oid	物化视图的oid。

名称	类型	描述
mapid	oid	物化视图map表的oid，map表为物化视图关联表，与物化视图一一对应。全量物化视图不存在对应的map表，该字段为0。
ivm	boolean	物化视图的类型，t为增量物化视图，f为全量物化视图。
needrefresh	boolean	保留字段。
refreshtime	timestamp	物化视图上一次刷新时间，若未刷新则为null。仅对DN上的增量物化视图维护该字段，其余情况均为null。

12.2.14 GS_MATVIEW_DEPENDENCY

GS_MATVIEW_DEPENDENCY系统表提供了关于数据库中每一个增量物化视图、基表和mlog表的关联信息。全量物化视图不存在与基表对应的mlog表，不会写入记录。

表 12-14 GS_MATVIEW_DEPENDENCY 字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
matviewid	oid	物化视图的oid。
relid	oid	物化视图基表的oid。
mlogid	oid	物化视图mlog表的oid，mlog表为物化视图日志表，与基表一一对应。
mxmin	int4	保留字段。

12.2.15 GS_OPT_MODEL

GS_OPT_MODEL是启用AiEngine执行计划时间预测功能时的数据表，记录机器学习模型的配置、训练结果、功能、对应系统函数、训练历史等相关信息。

说明

分布式场景下提供此系统表，但AI能力不可用。

12.2.16 GS_POLICY_LABEL

GS_POLICY_LABEL系统表记录资源标签配置信息，一个资源标签对应着一条或多条记录，每条记录标记了数据库资源所属的资源标签。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

FQDN（Fully Qualified Domain Name）标识了数据库资源所属的绝对路径。

表 12-15 GS_POLICY_LABEL 表字段

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
labelname	name	资源标签名称。
labeltype	name	资源标签类型，目前仅为RESOURCE。
fqdnnamespace	oid	被标识的数据库资源所属的namespace oid。
fqdnid	oid	被标识的数据库资源的oid，若数据库资源为列，则该列为所属表的oid。
relcolumn	name	列名，若被标识的数据库资源为列，该列指出列名，否则该列为空。
fqdntype	name	被标识的数据库资源的类型名称，例如schema、table、column、view等。

12.2.17 GS_RECYCLEBIN

gs_recyclebin描述了闪回特性回收站对象的详细信息，目前分布式不支持闪回特性。

12.2.18 GS_SQL_PATCH

GS_SQL_PATCH系统表存储所有SQL_PATCH的状态信息，当前分布式下暂不支持该功能。

表 12-16 GS_SQL_PATCH 字段

名称	类型	描述
patch_name	name	PATCH名称。
unique_sql_id	bigint	查询全局唯一ID。
owner	oid	PATCH的创建用户ID。
enable	boolean	PATCH是否生效。

名称	类型	描述
status	"char"	PATCH的状态（预留字段）。
abort	boolean	是否是AbortHint。
hint_string	text	Hint文本。
hint_node	pg_node_tree	Hint解析&序列化的结果。
original_query	text	原始语句（预留字段）。
patched_query	text	PATCH之后的语句（预留字段）。
original_query_tree	pg_node_tree	原始语句的解析结果（预留字段）。
patched_query_tree	pg_node_tree	PATCH之后语句的解析结果（预留字段）。
description	text	PATCH的备注。

12.2.19 GS_TXN_SNAPSHOT

GS_TXN_SNAPSHOT是“时间戳-CSN”映射表，周期性采样，并维护适当的时间范围，用于估算范围内的时间戳对应的CSN值。目前分布式不支持闪回特性。

12.2.20 GS_UID

GS_UID系统表存储了数据库中使用hasuids属性表的唯一标识元信息。

表 12-17 GS_UID 字段

名称	类型	描述
relid	oid	表的oid信息。
uid_backup	bigint	当前可以为表分配唯一标识的最大值。

12.2.21 PG_AGGREGATE

PG_AGGREGATE系统表存储与聚集函数有关的信息。PG_AGGREGATE里的每条记录都是一条pg_proc里面的记录的扩展。PG_PROC记录承载该聚集的名称、输入和输出数据类型，以及其它一些和普通函数类似的信息。

表 12-18 PG_AGGREGATE 字段

名称	类型	引用	描述
aggfnoid	regproc	PG_PROC.proname	此聚集函数的PG_PROC proname。
aggtransfn	regproc	PG_PROC.proname	转换函数。
aggcollectfn	regproc	PG_PROC.proname	收集函数。
aggfinalfn	regproc	PG_PROC.proname	最终处理函数（如果没有则为零）。
aggsortop	oid	PG_OPERATOR.oid	关联排序操作符（如果没有则为零）。
aggtranstype	oid	PG_TYPE.oid	此聚集函数的内部转换（状态）数据的数据类型。 可能取值及其含义见于pg_type.h中type定义，主要分为多态（isPolymorphicType）和非多态两类。
agginitval	text	-	转换状态的初始值。这是一个文本数据域，它包含初始值的外部字符串表现形式。如果数据域是null，则转换状态值从null开始。
agginitcollect	text	-	收集状态的初始值。这是一个文本数据域，它包含初始值的外部字符串表现形式。如果数据域是null，则收集状态值从null开始。
aggkind	"char"	-	此聚集函数类型： • 'n'：表示Normal Agg • 'o'：表示Ordered Set Agg
aggnumdirectargs	smallint	-	Ordered Set Agg类型聚集函数的直接参数（非聚集相关参数）数量。对Normal Agg类型聚集函数，该值为0。

12.2.22 PG_AM

PG_AM系统表存储有关索引访问方法的信息。系统支持的每种索引访问方法都有一行。

表 12-19 PG_AM 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
amname	name	-	访问方法的名称。
amstrategies	smallint	-	访问方法的操作符策略个数，或者如果访问方法没有一个固定的操作符策略集则为0。
amsupport	smallint	-	访问方法的支持过程个数。
amcanorder	boolean	-	这种访问方式是否支持通过索引字段值的命令扫描排序。 ● t（true）：表示支持。 ● f（false）：表示不支持。
amcanorderbyop	boolean	-	这种访问方式是否支持通过索引字段上操作符的结果的命令扫描排序。 t（true）：表示支持。 f（false）：表示不支持。
amcanbackward	boolean	-	访问方式是否支持向后扫描。 ● t（true）：表示支持。 ● f（false）：表示不支持。
amcanunique	boolean	-	访问方式是否支持唯一索引。 ● t（true）：表示支持。 ● f（false）：表示不支持。
amcanmulticol	boolean	-	访问方式是否支持多字段索引。 ● t（true）：表示支持。 ● f（false）：表示不支持。
amoptionalkey	boolean	-	访问方式是否支持第一个索引字段上没有任何约束的扫描。 ● t（true）：表示支持。 ● f（false）：表示不支持。
amsearcharray	boolean	-	访问方式是否支持ScalarArrayOpExpr搜索。 t（true）：表示支持。 f（false）：表示不支持。
amsearchnulls	boolean	-	访问方式是否支持IS NULL/NOT NULL搜索。 ● t（true）：表示支持。 ● f（false）：表示不支持。

名称	类型	引用	描述
amstorage	boolean	-	是否允许索引存储的数据类型与列的数据类型不同。 ● t（true）：表示允许。 ● f（false）：表示不允许。
amclusterable	boolean	-	是否允许在一个这种类型的索引上聚簇。 ● t（true）：表示允许。 ● f（false）：表示不允许。
ampredlocks	boolean	-	是否允许这种类型的一个索引管理细粒度的谓词锁定。 ● t（true）：表示允许。 ● f（false）：表示不允许
amkeytype	oid	PG_TYPE.oid	存储在索引里数据的类型，如果不是一个固定的类型则为0。
aminsert	regproc	PG_PROC.proname	“插入这个行”函数。
ambeginscan	regproc	PG_PROC.proname	“准备索引扫描” 函数。
amgettuple	regproc	PG_PROC.proname	“下一个有效行”函数，如果没有则为0。
amgetbitmap	regproc	PG_PROC.proname	“抓取所有的有效行” 函数，如果没有则为0。
amrescan	regproc	PG_PROC.proname	“（重新）开始索引扫描”函数。
amendscan	regproc	PG_PROC.proname	“索引扫描后清理” 函数。
ammarkpos	regproc	PG_PROC.proname	“标记当前扫描位置”函数。
amrestrpos	regproc	PG_PROC.proname	“恢复已标记的扫描位置”函数。
ammerge	regproc	PG_PROC.proname	“归并多个索引对象”函数。
ambuild	regproc	PG_PROC.proname	“建立新索引”函数。
ambuildempty	regproc	PG_PROC.proname	“建立空索引”函数。
ambulkdelete	regproc	PG_PROC.proname	批量删除函数。

名称	类型	引用	描述
amvacuumcleanup	regproc	PG_PROC.proname	VACUUM后的清理函数。
amcanreturn	regproc	PG_PROC.proname	检查是否索引支持唯一索引扫描的函数，如果没有则为0。
amcostestimate	regproc	PG_PROC.proname	估计一个索引扫描开销的函数。
amoptions	regproc	PG_PROC.proname	为一个索引分析和确认reloptions的函数。

12.2.23 PG_AMOP

PG_AMOP系统表存储有关和访问方法操作符族关联的信息。如果一个操作符是一个操作符族中的成员，则在这个表中会占据一行。一个族成员是一个search操作符或一个ordering操作符。一个操作符可以在多个族中出现，但是不能在一个族中的多个搜索位置或多个排序位置中出现。

表 12-20 PG_AMOP 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含字段，必须明确选择）。
amopfamily	oid	PG_OPFAMILY.oid	这个项的操作符族。
amoplefttype	oid	PG_TYPE.oid	操作符的左输入类型。可能取值及其描述见于数据类型。
amoprighttype	oid	PG_TYPE.oid	操作符的右输入类型。可能取值及其描述见于数据类型。
amopstrategy	smallint	-	操作符策略数。
amoppurpose	"char"	-	操作符目的。 • s：表示搜索。 • o：表示排序。
amopopr	oid	PG_OPERATOR.oid	该操作符的OID。
amopmethod	oid	PG_AM.oid	索引访问方式操作符族。
amopsortfamily	oid	PG_OPFAMILY.oid	如果是一个排序操作符，则为这个项排序所依据的btree操作符族；如果是一个搜索操作符，则为0。

search操作符表明这个操作符族的一个索引可以被搜索，找到所有满足WHERE indexed_column operator constant的行。显然，这样的操作符必须返回布尔值，并且它的左输入类型必须匹配索引的字段数据类型。

ordering操作符表明这个操作符族的一个索引可以被扫描，返回以ORDER BY indexed_column operator constant顺序表示的行。这样的操作符可以返回任意可排序的数据类型，它的左输入类型也必须匹配索引的字段数据类型。 ORDER BY的确切的语义是由amopsortfamily字段指定的，该字段必须为操作符的返回类型引用一个btree操作符族。

12.2.24 PG_AMPROC

PG_AMPROC系统表存储有关与访问方法操作符族相关联的支持过程的信息。每个属于某个操作符族的支持过程都占有一行。

表 12-21 PG_AMPROC 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含字段，必须明确选择）。
amprocfamily	oid	**PG_OPFAMILY**.oid	该项的操作符族。
amproclefttype	oid	**PG_TYPE**.oid	相关操作符的左输入数据类型。 常见的数据类型请参见**数据类型**。
amprocrighttype	oid	**PG_TYPE**.oid	相关操作符的右输入数据类型。 常见的数据类型请参见**数据类型**。
amprocnum	smallint	-	支持过程编号。
amproc	regproc	**PG_PROC** .proname	过程的OID。

amproclefttype和amprocrighttype字段的习惯解释，标识一个特定支持过程支持的操作符的左和右输入类型。对于某些访问方式，匹配支持过程本身的输入数据类型，对其他的则不这样。有一个对索引的“缺省”支持过程的概念，amproclefttype和amprocrighttype都等于索引操作符类的opcintype。

12.2.25 PG_APP_WORKLOADGROUP_MAPPING

PG_APP_WORKLOADGROUP_MAPPING系统表提供了数据库负载映射组的信息。

表 12-22 PG_APP_WORKLOADGROUP_MAPPING 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
appname	name	应用名称。
workload_gpname	name	映射到的负载组名称。

12.2.26 PG_ATTRDEF

PG_ATTRDEF系统表存储列的默认值。

表 12-23 PG_ATTRDEF 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
adrelid	oid	该列的所属表。
adnum	smallint	该列的数目。
adbin	pg_node_tree	字段缺省值的内部表现形式。
adsrc	text	人类可读的缺省值的内部表现形式。
adgencol	"char"	标识该列是否为生成列。取值为's'表示该列为生成列，取值为'\0'表示该列为普通列，默认值为'\0'。

12.2.27 PG_ATTRIBUTE

PG_ATTRIBUTE系统表存储关于表字段的信息。

表 12-24 PG_ATTRIBUTE 字段

名称	类型	描述
attrelid	oid	此字段所属表。
attname	name	字段名。
atttypid	oid	字段类型。

名称	类型	描述
attstattarget	integer	控制ANALYZE为这个字段积累的统计细节的级别。 ● 零值表示不收集统计信息。 ● 负数表示使用系统缺省的统计对象。 ● 正数值的确切信息是和数据类型相关的。 对于标量数据类型，ATTSTATTARGET既是要收集的"最常用数值"的目标数目，也是要创建的柱状图的目标数量。
attlen	smallint	是本字段类型的**PG_TYPE**中typlen的拷贝。
attnum	smallint	字段编号。
attndims	integer	如果该字段是数组，则是维数，否则是0 。
attcacheoff	integer	在磁盘上的时候总是-1 ，但是如果加载入内存中的行描述器中，它可能会被更新以缓冲在行中字段的偏移量。
atttypmod	integer	记录创建新表时支持的类型特定的数据（比如一个varchar字段的最大长度）。它传递给类型相关的输入和长度转换函数当做第三个参数。其值对那些不需要ATTTYPMOD的类型通常为-1。
attbyval	boolean	这个字段类型的**PG_TYPE**中typbyval的拷贝。
attstorage	"char"	这个字段类型的**PG_TYPE**中typstorage的拷贝。
attalign	"char"	这个字段类型的**PG_TYPE**中typalign的拷贝。
attnotnull	boolean	这代表一个非空约束。可以改变这个字段以打开或者关闭这个约束。
atthasdef	boolean	这个字段有一个缺省值，此时它对应**PG_ATTRDEF**表里实际定义此值的记录。
attisdropped	boolean	这个字段已经被删除了，不再有效。一个已经删除的字段物理上仍然存在表中，但会被分析器忽略，因此不能再通过SQL访问。
attislocal	boolean	这个字段是局部定义在关系中的。请注意一个字段可以同时是局部定义和继承的。
attinhcount	integer	这个字段所拥有的直接父表的个数。如果一个字段的父表个数非零，则它就不能被删除或重命名。
attcollation	oid	对此列定义的校对列。
attacl	aclitem[]	列级访问权限控制。

名称	类型	描述
attoptions	text[]	字段属性。目前支持以下两种属性： ● n_distinct，表示该字段的distinct值数量（不包含子表）。 ● n_distinct_inherited，表示该字段的distinct值数量（包含子表）。
attfdwoptions	text[]	外表字段属性。当前支持的dist_fdw、file_fdw、log_fdw未使用外表字段属性。
attinitdefval	bytea	存储了此列默认的值表达式。行存表的ADD COLUMN需要使用此字段。
attkvtype	tinyint	对某一列指定key value类型。类型包括： 0. ATT_KV_UNDEFINED : 默认 1. ATT_KV_TAG : 维度 2. ATT_KV_FIELD : 指标 3. ATT_KV_TIMETAG :时间列 4. ATT_KV_HIDETAG: 隐藏分布列

12.2.28 PG_AUTHID

PG_AUTHID系统表存储有关数据库认证标识符（角色）的信息。角色把“用户”的概念包含在内。一个用户实际上就是一个rolcanlogin标志被设置的角色。任何角色（不管rolcanlogin设置与否）都能够把其他角色作为成员。

在一个集群中只有一份pg_authid，不是每个数据库有一份。需要有系统管理员权限才可以访问此系统表。

表 12-25 PG_AUTHID 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
rolname	name	角色名称。
rolsuper	boolean	角色是否是拥有最高权限的初始系统管理员。 ● t（true）：表示是。 ● f（false）：表示不是。
rolinherit	boolean	角色是否自动继承其所属角色的权限。 ● t（true）：表示自动继承。 ● f（false）：表示不自动继承。

名称	类型	描述
rolcreaterole	boolean	角色是否可以创建更多角色。 ● t（true）：表示可以。 ● f（false）：表示不可以。
rolcreatedb	boolean	角色是否可以创建数据库。 ● t（true）：表示可以。 ● f（false）：表示不可以。
rolcatupdate	boolean	角色是否可以直接更新系统表。只有usesysid=10的初始系统管理员拥有此权限。其他用户无法获得此权限。 ● t（true）：表示可以。 ● f（false）：表示不可以。
rolcanlogin	boolean	角色是否可以登录，也就是说，这个角色可以给予会话认证标识符。 ● t（true）：表示可以。 ● f（false）：表示不可以。
rolreplication	boolean	角色是否具有复制权限。 ● t（true）：表示有。 ● f（false）：表示没有。
rolauditadmin	boolean	角色是否具有审计管理员权限。 ● t（true）：表示有。 ● f（false）：表示没有。
rolsystemadmin	boolean	角色是否具有系统管理员权限。 ● t（true）：表示有。 ● f（false）：表示没有。
rolconnlimit	integer	对于可以登录的角色，限制其最大并发连接数量。 -1 表示没有限制。
rolpassword	text	口令密文，如果没有口令，则为NULL。
rolvalidbegin	timestamp with time zone	账户的有效开始时间，如果没有开始时间，则为NULL。
rolvaliduntil	timestamp with time zone	账户的有效结束时间，如果没有结束时间，则为NULL。
roluseft	boolean	角色是否可以操作外表。 ● t（true）：表示可以。 ● f（false）：表示不可以。
rolparentid	oid	用户所在组用户的OID。

名称	类型	描述
roltabspace	text	用户数据表的最大空间限额。
rolkind	"char"	特殊用户种类，包括Node group管理员、永久用户和普通用户。
roltempspace	text	用户临时表的最大空间限额，单位 KB。
rolspillspace	text	用户执行作业时下盘数据的最大空间限额，单位 KB。
rolexcpdata	text	用户可以设置的查询规则(当前未使用)。
rolmonitoradmin	boolean	角色是否具有监控管理员权限。 ● t（true）：表示有。 ● f（false）：表示没有。
roloperatoradmin	boolean	角色是否具有运维管理员权限。 ● t（true）：表示有。 ● f（false）：表示没有。
rolpolicyadmin	boolean	角色是否具有安全策略管理员权限。 ● t（true）：表示有。 ● f（false）：表示没有。

12.2.29 PG_AUTH_HISTORY

PG_AUTH_HISTORY系统表记录了角色的认证历史。需要有系统管理员权限才可以访问此系统表。

表 12-26 PG_AUTH_HISTORY 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
roloid	oid	角色标识。
passwordtime	timestamp with time zone	创建和修改密码的时间。
rolpassword	text	角色密码密文，加密方式由GUC参数password_encryption_type确定。

12.2.30 PG_AUTH_MEMBERS

PG_AUTH_MEMBERS系统表存储显示角色之间的成员关系。

表 12-27 PG_AUTH_MEMBERS 字段

名称	类型	描述
roleid	oid	拥有成员的角色ID。
member	oid	属于ROLEID角色的一个成员的角色ID。
grantor	oid	赋予此成员关系的角色ID。
admin_option	boolean	如果MEMBER可以把ROLEID角色的成员关系赋予其他角色，则为真，不可以则为假。

12.2.31 PG_CAST

PG_CAST系统表存储数据类型之间的转化关系。

表 12-28 PG_CAST 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
castsource	oid	源数据类型的OID。
casttarget	oid	目标数据类型的OID。
castfunc	oid	转化函数的OID。如果为零表明不需要转化函数。
castcontext	"char"	源数据类型和目标数据类型间的转化方式： ● 'e'：表示只能进行显式转化（使用CAST或::语法）。 ● 'i'：表示能进行隐式转化。 ● 'a'：表示类型间同时支持隐式和显式转化。
castmethod	"char"	转化方法： ● 'f'：使用castfunc字段中指定的函数进行转化。 ● 'b'：类型间是二进制强制转化，不使用castfunc。

12.2.32 PG_CLASS

PG_CLASS系统表存储数据库对象信息及其之间的关系。

表 12-29 PG_CLASS 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。

名称	类型	描述
relname	name	表、索引、视图等对象的名称。
relnamespace	oid	包含这个关系的名称空间的OID。
reltype	oid	对应这个表的行类型的数据类型（索引为零，因为索引没有pg_type记录）。
reloftype	oid	复合类型的OID，0表示其他类型。
relowner	oid	关系所有者。
relam	oid	如果行是索引，则就是所用的访问模式（B-tree，hash等）。
relfilenode	oid	这个关系在磁盘上的文件的名称，如果没有则为0。
reltablespace	oid	这个关系存储所在的表空间。如果为零，则意味着使用该数据库的缺省表空间。如果关系在磁盘上没有文件，则这个字段没有什么意义。
relpages	double precision	以页(大小为BLCKSZ)为单位的该表在磁盘上的大小，它只是优化器用的一个近似值。
reltuples	double precision	表中行的数目，只是优化器使用的一个估计值。
relallvisible	integer	被标识为全可见的表中的页的数量。此字段是优化器用来做SQL执行优化使用的。VACUUM、ANALYZE和一些DDL语句（例如，CREATE INDEX）会引起此字段更新。
reltoastrelid	oid	与该表关联的TOAST表的OID ，如果没有则为0。 TOAST表在一个从属表里“离线”存储大字段。
reltoastidxid	oid	对于TOAST表是它的索引的OID，如果不是TOAST表则为0。
relhasindex	boolean	如果它是一个表而且至少有（或者最近有过）一个索引，则为真。 它是由CREATE INDEX设置的，但DROP INDEX不会立即将它清除。如果VACUUM进程检测一个表没有索引，将会把它将清理relhasindex字段，将值设置为假。
relisshared	boolean	如果该表在整个集群中由所有数据库共享则为真，否则为假。只有某些系统表（比如pg_database）是共享的。

名称	类型	描述
relpersistence	"char"	• p：表示永久表。 • u：表示非日志表。 • t：表示临时表。 • g：表示全局临时表。
relkind	"char"	• r：表示普通表。 • i：表示索引。 • S：表示序列。 • v：表示视图。 • c：表示复合类型。 • t：表示TOAST表。 • f：表示外表。 • m：表示物化视图。 • e：表示STREAM对象。 • o：表示CONTVIEW对象。
relnatts	smallint	关系中用户字段数目（除了系统字段以外）。在PG_ATTRIBUTE里有相同数目对应行。
relchecks	smallint	表里的检查约束的数目，参阅PG_CONSTRAINT表。
relhasoids	boolean	如果为关系中每行都生成一个OID则为真，否则为假。
relhaspkey	boolean	如果这个表有一个（或者曾经有一个）主键，则为真，否则为假。
relhasrules	boolean	如表有规则就为真。是否有规则可参考系统表PG_REWRITE。
relhastriggers	boolean	True表示表中有触发器，或者曾经有过触发器。系统表PG_TRIGGER中记录了表和视图的触发器。
relhassubclass	boolean	如果有（或者曾经有）任何继承的子表，为真，否则为假。
relrowmovement	boolean	针对分区表进行update操作时，是否允许行迁移。 • true：表示允许行迁移。 • false：表示不允许行迁移。
parttype	"char"	表或者索引是否具有分区表的性质。 • p：表示带有分区表性质。 • n：表示没有分区表特性。

名称	类型	描述
relfrozenxid	xid32	该表中所有在这个之前的事务ID已经被一个固定的（"frozen"）事务ID替换。该字段用于跟踪该表是否需要为了防止事务ID重叠（或者允许收缩pg_clog）而进行清理。如果该关系不是表则为零（InvalidTransactionId）。 为保持前向兼容，保留此字段，新增relfrozenxid64用于记录此信息。
relacl	aclitem[]	访问权限。 查询的回显结果为以下形式： rolename=xxxx/yyyy --赋予一个角色的权限 =xxxx/yyyy --赋予public的权限 xxxx表示赋予的权限，yyyy表示授予这个权限的角色。权限的参数说明请参见**表12-30**。
reloptions	text[]	表或索引的访问方法，使用"keyword=value"格式的字符串。
relreplident	"char"	逻辑解码中解码列的标识： • d：默认 (主键，如果存在)。 • n：无。 • f：所有列。 • i：索引的indisreplident被设置或者为默认。
relfrozenxid64	xid	该表中所有在这个之前的事务ID已经被一个固定的（"frozen"）事务ID替换。该字段用于跟踪该表是否需要为了防止事务ID重叠（或者允许收缩pg_clog）而进行清理。如果该关系不是表则为零（InvalidTransactionId）。
relbucket	oid	当前表是否包含hash bucket分片。有效的OID指向pg_hashbucket表中记录的具体分片信息。NULL表示不包含hash bucket分片。
relbucketkey	int2vector	表示hash分区列信息，NULL表示不包含。
relminmxid	xid	该表中所有在这个之前的多事务ID已经被一个事务ID替换。该字段用于跟踪该表是否需要为了防止多事务ID重叠或者允许收缩pg_clog而进行清理。如果该关系不是表则为零（InvalidTransactionId）。

表 12-30 权限的参数说明

参数	参数说明
r	SELECT（读）
w	UPDATE（写）

参数	参数说明
a	INSERT（插入）
d	DELETE
D	TRUNCATE
x	REFERENCES
t	TRIGGER
X	EXECUTE
U	USAGE
C	CREATE
c	CONNECT
T	TEMPORARY
A	ALTER
P	DROP
m	COMMENT
i	INDEX
v	VACUUM
*	给前面权限的授权选项

12.2.33 PG_COLLATION

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

表 12-31 PG_COLLATION 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含字段，必须明确选择）。
collname	name	-	排序规则名（每个名称空间和编码唯一）。
collnamespace	oid	PG_NAMESPACE.oid	包含这个排序规则的名称空间的OID。
collowner	oid	PG_AUTHID.oid	排序规则的所有者。

名称	类型	引用	描述
collencoding	integer	-	排序规则可用的编码，兼容PostgreSQL所有的字符编码类型，如果适用于任意编码为-1。
collcollate	name	-	这个排序规则对象的LC_COLLATE。
collctype	name	-	这个排序规则对象的LC_CTYPE。

12.2.34 PG_CONSTRAINT

PG_CONSTRAINT系统表存储表上的检查约束、主键、唯一约束和外键约束。

表 12-32 PG_CONSTRAINT 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
conname	name	约束名称（不一定是唯一的）。
connamespace	oid	包含这个约束的名称空间的OID。
contype	"char"	• c：检查约束。 • p：主键约束。 • u：唯一约束。 • t：触发器约束。 • x：互斥约束。 • f：外键约束。 • s：聚簇约束。 • i：无效约束。
condeferrable	boolean	这个约束是否可以推迟。 • true：表示可以。 • false：表示不可以。
condeferred	boolean	缺省时这个约束是否可以推迟。 • true：表示可以。 • false：表示不可以。
convalidated	boolean	约束是否有效。目前，只有外键和CHECK约束可将其设置为FALSE。 • true：表示有效。 • false：表示无效。

名称	类型	描述
conrelid	oid	这个约束所在的表，如果不是表约束则为0。
contypid	oid	这个约束所在的域，如果不是一个域约束则为0。
conindid	oid	与约束关联的索引ID。
confrelid	oid	如果是外键，则为参考的表，否则为0。
confupdtype	"char"	外键更新动作代码。 ● a：没动作。 ● r：限制。 ● c：级联。 ● n：设置为null。 ● d：设置为缺省。
confdeltype	"char"	外键删除动作代码。 ● a：没动作。 ● r：限制。 ● c：级联。 ● n：设置为null。 ● d：设置为缺省。
confmatchtype	"char"	外键匹配类型。 ● f：全部。 ● p：部分。 ● u：未指定（在f的基础上允许匹配NULL值）。
conislocal	boolean	是否是为关系创建的本地约束。 ● true：表示是。 ● false：表示不是。
coninhcount	integer	约束直接继承父表的数目。继承父表数非零时，不能删除或重命名该约束。
connoinherit	boolean	是否可以被继承。 ● true：表示可以。 ● false：表示不可以。
consoft	boolean	是否为信息约束(Informational Constraint)。 ● true：表示是。 ● false：表示不是。

名称	类型	描述
conopt	boolean	是否使用信息约束优化执行计划。 ● true：表示使用。 ● false：表示不使用。
conkey	smallint[]	如果是表约束，则是约束控制的字段列表。
confkey	smallint[]	如果是一个外键，是参考的字段的列表。
conpfeqop	oid[]	如果是一个外键，是做PK=FK比较的相等操作符ID的列表。
conppeqop	oid[]	如果是一个外键，是做PK=PK比较的相等操作符ID的列表。
conffeqop	oid[]	如果是一个外键，是做FK=FK比较的相等操作符ID的列表。
conexclop	oid[]	如果是一个排他约束，是列的排他操作符ID列表。
conbin	pg_node_tree	如果是检查约束，那就是其表达式的内部形式。
consrc	text	如果是检查约束，则是表达式的可读形式。
conincluding	smallint[]	不用做约束，但是会包含在INDEX中的属性列。

须知

- consrc在被引用的对象改变之后不会被更新，它不会跟踪字段的名称修改。建议使用pg_get_constraintdef()来抽取一个检查约束的定义。
- PG_CLASS的relchecks需要和在该表上为给定关系找到的检查约束的数目一致。

12.2.35 PG_CONVERSION

PG_CONVERSION系统表描述编码转换信息。

表 12-33 PG_CONVERSION 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含字段，必须明确选择）。
conname	name	-	转换名称（在一个名称空间里是唯一的）。
connamespace	oid	PG_NAMESPACE.oid	包含这个转换的名称空间的OID。

名称	类型	引用	描述
conowner	oid	PG_AUTHID.oid	编码转换的属主。
conforencoding	integer	-	源编码ID。
contoencoding	integer	-	目的编码ID。
conproc	regproc	PG_PROC.proname	转换过程。
condefault	boolean	-	如果这是缺省转换则为真，否则为假。

12.2.36 PG_DATABASE

PG_DATABASE系统表存储关于可用数据库的信息。

表 12-34 PG_DATABASE 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
datname	name	数据库名称。
datdba	oid	数据库所有人，通常为其创建者。
encoding	integer	数据库的字符编码方式。
datcollate	name	数据库使用的排序顺序。
datctype	name	数据库使用的字符分类。
datistemplate	boolean	是否允许作为模板数据库。 • true：表示允许。 • false：表示不允许。
datallowconn	boolean	这个字段用于保护template0数据库不被更改。 • true：表示用户可以连接到这个数据库。 • false：表示没有用户可以连接到这个数据库。
datconnlimit	integer	该数据库上允许的最大并发连接数，-1表示无限制。
datlastsysoid	oid	数据库里最后一个系统OID 。
datfrozenxid	xid32	用于跟踪该数据库是否需要为了防止事务ID重叠而进行清理。当前版本该字段已经废弃使用，为保持前向兼容，保留此字段，新增datfrozenxid64用于记录此信息。

名称	类型	描述
dattablespace	oid	数据库的缺省表空间。
datcompatibility	name	数据库兼容模式。当前支持四种兼容模式：PG、ORA、MYSQL、TD。
datacl	aclitem[]	访问权限。
datfrozenxid64	xid	用于跟踪该数据库是否需要为了防止事务ID重叠而进行清理。
datminmxid	xid	该数据库中所有在这之前的多事务ID已经被一个事务ID替换。用于跟踪该数据库是否需要为了防止事务ID重叠或者允许收缩pg_clog而进行清理。它是此数据库中所有表的PG_CLASS中relminmxid的最小值。

12.2.37 PG_DB_ROLE_SETTING

PG_DB_ROLE_SETTING系统表存储数据库运行时每个角色与数据绑定的配置项的默认值 。

表 12-35 PG_DB_ROLE_SETTING 字段

名称	类型	描述
setdatabase	oid	配置项所对应的数据库，如果未指定数据库，则为0。
setrole	oid	配置项所对应的角色，如果未指定角色，则为0。
setconfig	text[]	运行时配置项的默认值。

12.2.38 PG_DEFAULT_ACL

PG_DEFAULT_ACL系统表存储为新建对象设置的初始权限。

表 12-36 PG_DEFAULT_ACL 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
defaclrole	oid	与此权限相关的角色ID。
defaclnamespace	oid	与此权限相关的名称空间，如果没有，则为0。

名称	类型	描述
defaclobjtype	"char"	此权限的对象类型。 ● r：表示表或视图。 ● S：表示序列。 ● f：表示函数。 ● T：表示类型。 ● K：表示客户端主密钥。 ● k：表示列加密密钥。
defaclacl	aclitem[]	创建该类型时所拥有的访问权限。

12.2.39 PG_DEPEND

PG_DEPEND系统表记录数据库对象之间的依赖关系。这个信息允许DROP命令找出哪些其它对象必须由DROP CASCADE删除，或者是在DROP RESTRICT的情况下避免删除。

这个表的功能类似**PG_SHDEPEND**，用于记录那些在数据库集群之间共享的对象之间的依赖性关系。

表 12-37 PG_DEPEND 字段

名称	类型	引用	描述
classid	oid	**PG_CLASS**.oid	有依赖对象所在系统表的OID。
objid	oid	任意OID属性	指定的依赖对象的OID。
objsubid	integer	-	对于表字段，这个是该属性的字段数（objid和classid引用表本身）。对于所有其它对象类型，目前这个字段是0。
refclassid	oid	**PG_CLASS**.oid	被引用对象所在的系统表的OID。
refobjid	oid	任意OID属性	指定的被引用对象的OID。
refobjsubid	integer	-	对于表字段，这个是该字段的字段号（refobjid和refclassid引用表本身）。对于所有其它对象类型，目前这个字段是0。
deptype	"char"	-	一个定义这个依赖关系特定语义的代码。

在所有情况下，一个PG_DEPEND记录表示被引用的对象不能在有依赖的对象被删除前删除。不过，这里还有几种由deptype定义的情况：

- DEPENDENCY_NORMAL (n)：独立创建的对象之间的一般关系。有依赖的对象可以在不影响被引用对象的情况下删除。被引用对象只有在声明了CASCADE的情

况下删除，这时有依赖的对象也被删除。例子：一个表字段对其数据类型有一般依赖关系。

- DEPENDENCY_AUTO (a)：有依赖对象可以和被引用对象分别删除，并且如果删除了被引用对象则应该被自动删除（不管是RESTRICT或CASCADE模式）。例子：一个表上面的命名约束是在该表上的自动依赖关系，因此如果删除了表，它也会被删除。
- DEPENDENCY_INTERNAL (i)：有依赖的对象是作为被引用对象的一部分创建的，实际上只是它的内部实现的一部分。 DROP有依赖对象是不能直接允许的（将告诉用户发出一条删除被引用对象的DROP）。一个对被引用对象的DROP将传播到有依赖对象，不管是否声明了CASCADE。例子：一个创建来强制外键约束的触发器在该约束的**PG_CONSTRAINT**记录上是标记为内部依赖的。
- DEPENDENCY_EXTENSION (e)：依赖对象是被依赖对象extension的一个成员。依赖对象只可以通过在被依赖对象上DROP EXTENSION删除。函数上这个依赖类型和内部依赖一样动作，但是它为了清晰和简化gs_dump保持分开。
- DEPENDENCY_PIN (p)：没有依赖对象；这种类型的记录标志着系统本身依赖于被引用对象，因此这个对象绝不能被删除。这种类型的记录只有在initdb的时候创建。有依赖对象的字段里是零。

12.2.40 PG_DESCRIPTION

PG_DESCRIPTION系统表可以给每个数据库对象存储一个可选的描述（注释）。许多内置的系统对象的描述提供了PG_DESCRIPTION的初始内容。

这个表的功能类似**PG_SHDESCRIPTION**，用于记录整个集群范围内共享对象的注释。

表 12-38 PG_DESCRIPTION 字段

名称	类型	引用	描述
objoid	oid	任意OID属性	这条描述所描述的对象的OID。
classoid	oid	**PG_CLASS**.oid	这个对象出现的系统表的OID。
objsubid	integer	-	对于一个表字段的注释，它是字段号（objoid和classoid指向表自身）。对于其它对象类型，它是零。
description	text	-	对该对象描述的任意文本。

12.2.41 PG_DIRECTORY

PG_DIRECTORY系统表用于保存用户添加的directory对象可以通过CREATE DIRECTORY语句向该表中添加记录，当enable_access_server_directory=off时，只允许初始用户创建directory对象；当enable_access_server_directory=on时，具有SYSADMIN权限的用户和继承了内置角色gs_role_directory_create权限的用户可以创建directory对象。

表 12-39 PG_DIRECTORY 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
dirname	name	目录对象的名称。
owner	oid	目录对象的所有者。
dirpath	text	目录路径。
diracl	aclitem[]	访问权限。

12.2.42 PG_ENUM

PG_ENUM系统表包含显示每个枚举类型值和标签的记录。给定枚举类型的内部表示实际上是PG_ENUM里面相关行的OID。

表 12-40 PG_ENUM 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含字段，必须明确选择）。
enumtypid	oid	PG_TYPE.oid	拥有这个枚举值的PG_TYPE记录的OID。
enumsortorder	real	-	这个枚举值在它的枚举类型中的排序位置。
enumlabel	name	-	这个枚举值的文本标签。

PG_ENUM行的OID跟着一个特殊规则：偶数的OID保证用和它们的枚举类型一样的排序顺序排序。也就是，如果两个偶数OID属于相同的枚举类型，那么较小的OID必须有较小\enumsortorder值。奇数OID需要毫无关系的排序顺序。这个规则允许枚举比较例程在许多常见情况下避开目录查找。创建和修改枚举类型的例程只要可能就尝试分配偶数OID给枚举值。

当创建了一个枚举类型时，它的成员赋予了排序顺序位置1到n。但是随后添加的成员可能会分配enumsortorder的负值或分数值。对这些值的唯一要求是它们要正确的排序和在每个枚举类型中唯一。

12.2.43 PG_FOREIGN_SERVER

PG_FOREIGN_SERVER系统表存储外部服务器定义。一个外部服务器描述了一个外部数据源，例如一个远程服务器。外部服务器通过外部数据封装器访问。

表 12-41 PG_FOREIGN_SERVER 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含字段，必须明确选择）。
srvname	name	-	外部服务器名。
srvowner	oid	PG_AUTHID.oid	外部服务器的所有者。
srvfdw	oid	PG_FOREIGN_DATA_WRAPPER.oid	这个外部服务器的外部数据封装器的OID。
srvtype	text	-	服务器的类型（可选）。
srvversion	text	-	服务器的版本（可选）。
srvacl	aclitem[]	-	访问权限。
srvoptions	text[]	-	外部服务器指定选项，使用“keyword=value”格式的字符串。

12.2.44 PG_HASHBUCKET

PG_HASHBUCKET系统表存储hash bucket信息。

表 12-42 PG_HASHBUCKET 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
bucketid	oid	对bucketvector计算的hash值，通过hash值可以加速对bucketvector的查找。
bucketcnt	integer	包含分片的个数。
bucketmapsize	integer	所有DN上包含的分片总数。
bucketref	integer	预留字段，默认值为1。
bucketvector	oidvector_extend	记录此行bucket信息包含的所有bucket的id，在此列上建立唯一索引，具有相同bucketid信息的表共享同一行pg_hashbucket数据。

12.2.45 PG_INDEX

PG_INDEX系统表存储索引的一部分信息，其他的信息大多数在PG_CLASS中。

表 12-43 PG_INDEX 字段

名称	类型	描述
indexrelid	oid	这个索引在PG_CLASS里的记录的OID。
indrelid	oid	使用这个索引的表在PG_CLASS里的记录的OID。
indnatts	smallint	索引中的字段数目。
indisunique	boolean	是否为唯一索引。 ● true：是唯一索引。 ● false：不是唯一索引。
indisprimary	boolean	该索引是否为该表的主键。 ● true：该索引是该表的主键。这个字段为真的时候indisunique总是为真。 ● false：该索引不是该表的主键。
indisexclusion	boolean	该索引是否支持排他约束。 ● true：该索引支持排他约束。 ● false：该索引不支持排他约束。
indimmediate	boolean	插入数据时是否进行唯一性检查。 ● true：在插入数据时会立即进行唯一性检查。 ● false：在插入数据时不会进行唯一性检查。
indisclustered	boolean	该表是否在这个索引上建簇。 ● true：该表在这个索引上建簇。 ● false：该表没有在这个索引上建簇。
indisusable	boolean	该索引对INSERT/SELECT是否可用。 ● true：该索引对INSERT/SELECT可用。 ● false：该索引对INSERT/SELECT不可用。
indisvalid	boolean	● true：该索引可以用于查询。 ● false：该索引可能不完整，仍然必须在INSERT/UPDATE操作时进行更新，不过不能安全的用于查询。如果是唯一索引，则唯一属性也将不为真。
indcheckxmin	boolean	● true：查询不能使用索引，直到pg_index此行的xmin低于其快照的TransactionXmin，因为该表可能包含它们能看到的不兼容行断开的HOT链。 ● false：查询可以用索引。
indisready	boolean	● true：此索引对插入数据是可用的。 ● false：在插入或修改数据时忽略此索引。

名称	类型	描述
indkey	int2vector	这是一个包含indnatts值的数组，这些数组值表示这个索引所建立的表字段。比如一个值为1 3的意思是第一个字段和第三个字段组成这个索引键字。这个数组里的零表明对应的索引属性是在这个表字段上的一个表达式，而不是一个简单的字段引用。
indcollation	oidvector	索引用到的各列的ID。
indclass	oidvector	对于索引键字里面的每个字段，这个字段都包含一个指向所使用的操作符类的OID，参阅PG_OPCLASS获取细节。
indoption	int2vector	存储列前标识位，该标识位是由索引的访问方法定义。
indexprs	pg_node_tree	表达式树（以nodeToString()形式表现）用于那些非简单字段引用的索引属性。它是一个列表，个数与indkey中的零值个数相同。如果所有索引属性都是简单的引用，则为空。
indpred	pg_node_tree	部分索引断言的表达式树（以nodeToString()的形式表现）。如果不是部分索引，则是空字符串。
indisreplident	boolean	此索引的列是否为逻辑解码的解码列。 • true：此索引的列成为逻辑解码的解码列。 • false：此索引的列不是逻辑解码的解码列。
indnkeyatts	smallint	索引中的总字段数，超出indnatts的部分不参与索引查询。

12.2.46 PG_INHERITS

PG_INHERITS系统表记录关于表继承层次的信息。数据库里每个直接的子系表都有一条记录。间接的继承可以通过追溯记录链来判断。

表 12-44 PG_INHERITS 字段

名称	类型	引用	描述
inhrelid	oid	PG_CLASS.oid	子表的OID。
inhparent	oid	PG_CLASS.oid	父表的OID。
inhseqno	integer	-	如果一个子表存在多个直系父表（多重继承），这个数字表明此继承字段的排列顺序。计数从1开始。

12.2.47 PG_JOB

PG_JOB系统表存储用户创建的定时任务的任务详细信息，定时任务线程定时轮询PG_JOB系统表中的时间，当任务到期会触发任务的执行，并更新PG_JOB表中的任务状态。该系统表属于Shared Relation，所有创建的job记录对所有数据库可见。

表 12-45 PG_JOB 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
job_id	bigint	作业ID，主键，是唯一的（有唯一索引）。
current_postgres_pid	bigint	如果当前任务已被执行，那么此处记录运行此任务的线程ID。默认为 -1，表示此任务未被执行过。
log_user	name	创建者的UserName。
priv_user	name	作业执行者的UserName。
dbname	name	标识作业要在哪个数据库执行的数据库名称。
node_name	name	标识当前作业是在哪个CN上创建和执行。
job_status	"char"	当前任务的执行状态，默认为's'，各取值含义： ● 'r'：running ● 's'：successfully finished ● 'f'：job failed ● 'd'：disable 当job连续执行失败16次，会将job_status自动设置为失效状态'd'，后续不再执行该job。 注：当用户将定时任务关闭（即：guc参数job_queue_processes为0时），由于监控job执行的线程不会启动，所以该状态不会根据job的实时状态进行设置，用户不需要关注此状态。只有当开启定时任务功能（即：guc参数job_queue_processes为非0时），系统才会根据当前job的实时状态刷新该字段值。
start_date	timestamp without time zone	作业第一次开始执行时间，时间精确到毫秒。
next_run_date	timestamp without time zone	定时任务下次执行的时间，时间精确到毫秒。
failure_count	smallint	失败计数，作业连续执行失败16次，不再继续执行。
interval	text	作业执行的重复时间间隔。

名称	类型	描述
last_start_date	timestamp without time zone	上次运行开始时间，时间精确到毫秒。
last_end_date	timestamp without time zone	上次运行的结束时间，时间精确到毫秒。
last_suc_date	timestamp without time zone	上次成功运行的开始时间，时间精确到毫秒。
this_run_date	timestamp without time zone	正在运行任务的开始时间，时间精确到毫秒。
nspname	name	标识作业执行时的schema的名称。
job_name	text	DBE_SCHEDULER定时任务专用，定时任务名称。
end_date	timestamp without time zone	DBE_SCHEDULER定时任务专用，定时任务失效时间，时间精确到毫秒。
enable	boolean	DBE_SCHEDULER定时任务专用，定时任务启用状态： • true：启用。 • false：未启用。
failure_msg	text	最新一次执行任务报错信息。

12.2.48 PG_JOB_PROC

PG_JOB_PROC系统表对应PG_JOB表中每个任务的作业内容（包括：PL/SQL代码块、匿名块）。将存储过程信息独立出来，是因为如果放到PG_JOB中，被加载到共享内存的时候，会占用不必要的空间，所以在使用的时候再进行查询获取。

表 12-46 PG_JOB_PROC 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
job_id	integer	外键，关联PG_JOB中的job_id。
what	text	作业内容，DBE_SCHEDULER定时任务中的程序内容。
job_name	text	DBE_SCHEDULER定时任务专用，定时任务或程序名称。

12.2.49 PG_LANGUAGE

PG_LANGUAGE系统表登记编程语言，用户可以用这些语言或接口写函数或者存储过程。

表 12-47 PG_LANGUAGE 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含字段；必须明确选择）。
lanname	name	-	语言的名称。
lanowner	oid	PG_AUTHID.oid	语言的所有者。
lanispl	boolean	-	• true：表示用户定义的语言。 • false：表示内部语言，比如SQL。 目前，gs_dump仍然使用该字段判断哪种语言需要转储，但是这些可能在将来被其它机制取代。
lanpltrusted	boolean	-	• true：这是可信语言，意味着系统相信它不会被授予任何正常SQL执行环境之外的权限。 • false：这是不可信语言。只有初始用户可以用不可信语言创建函数。
lanplcallfoid	oid	PG_PROC.oid	对于非内部语言，这是指向该语言处理器的引用，语言处理器是一个特殊函数， 负责执行以某种语言写的所有函数。
laninline	oid	PG_PROC.oid	这个字段引用一个负责执行“inline”匿名代码块的函数（DO块）。如果不支持内联块则为零。
lanvalidator	oid	PG_PROC.oid	这个字段引用一个语言校验器函数，它负责检查新创建的函数的语法和有效性。如果没有提供校验器，则为零。
lanacl	aclitem[]	-	访问权限。

12.2.50 PG_LARGEOBJECT

PG_LARGEOBJECT系统表保存那些标记着“大对象”的数据。一个大对象是使用其创建时分配的OID标识的。每个大对象都分解成足够小的小段或者“页面”以便以行的形式存储在PG_LARGEOBJECT里。每页的数据定义为LOBLKSIZE。

需要有系统管理员权限才可以访问此系统表。

表 12-48 PG_LARGEOBJECT 字段

名称	类型	引用	描述
loid	oid	PG_LARGEOBJECT_METADATA.oid	包含本页的大对象的标识符。
pageno	integer	-	本页在其大对象数据中的页码，从零开始计算。
data	bytea	-	存储在大对象中的实际数据。这些数据绝不会超过LOBLKSIZE字节，而且可能更少。

PG_LARGEOBJECT的每一行保存一个大对象的一个页面，从该对象内部的字节偏移（pageno * LOBLKSIZE）开始。这种实现允许松散的存储：页面可以丢失，而且可以比LOBLKSIZE字节少（即使它们不是对象的最后一页）。大对象内丢失的部分读做零。

12.2.51 PG_LARGEOBJECT_METADATA

PG_LARGEOBJECT_METADATA系统表存储与大数据相关的元数据。实际的大对象数据存储在PG_LARGEOBJECT里。

表 12-49 PG_LARGEOBJECT_METADATA 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含字段，必须明确选择）。
lomowner	oid	PG_AUTHID.oid	大对象的所有者。
lomacl	aclitem[]	-	访问权限。

12.2.52 PG_NAMESPACE

PG_NAMESPACE系统表存储名称空间，即存储schema相关的信息。如果开启数据库对象隔离属性，用户只能查看自己有权限访问的schema信息。

表 12-50 PG_NAMESPACE 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
nspname	name	名称空间的名称。
nspowner	oid	名称空间的所有者。

名称	类型	描述
nsptimeline	bigint	在DN上创建此命名空间时的时间线。此字段为内部使用，仅在DN上有效。
nspacl	aclitem[]	访问权限。具体请参见**GRANT**和**REVOKE**。
in_redistribution	"char"	是否处于重发布状态。 ● t（true）：表示处于重发布状态。 ● f（false）：表示没有处于重发布状态。
nspblockchain	boolean	● 如果为真，则该模式为防篡改模式。 ● 如果为假，则此模式为非防篡改模式。

12.2.53 PG_OBJECT

PG_OBJECT系统表存储限定类型对象（普通表，索引，序列，视图，存储过程和函数）的创建用户、创建时间和最后修改时间。

表 12-51 PG_OBJECT 字段

名称	类型	描述
object_oid	oid	对象标识符。
object_type	"char"	对象类型： ● r：表示普通表。 ● i：表示索引。 ● s：表示序列。 ● v：表示视图。 ● p：表示存储过程和函数。
creator	oid	创建用户的标识符。
ctime	timestamp with time zone	对象的创建时间。
mtime	timestamp with time zone	对象的最后修改时间，修改行为包括ALTER操作和GRANT、REVOKE操作。
createcsn	bigint	对象创建时的CSN。
changecsn	bigint	对表或索引执行DDL操作时的CSN。

须知

- 无法记录初始化数据库（initdb）过程中所创建或修改的对象，即PG_OBJECT无法查询到该对象记录。
- 对于升级至该版本的数据库，无法记录升级以前所创建的对象，即PG_OBJECT无法查询到该对象记录。
- 对于上述两类对象再次修改时，会记录其修改时间（mtime），由于无法得知该对象创建时间，因此ctime为空。
- 对于升级前创建的对象，再次修改时会记录其修改时间（mtime），对表或索引执行DDL操作时会记录其所属事务的事务提交序列号（changecsn）。由于无法得知该对象创建时间，因此ctime和createcsn为空。
- ctime和mtime所记录的时间为用户当次操作所属事务的起始时间。
- 由扩容引起的对象修改时间也会被记录。
- createcsn和changecsn记录的是用户当次操作所属事务的事务提交序列号。

12.2.54 PG_OPCLASS

PG_OPCLASS系统表定义索引访问方法操作符类。

每个操作符类为一种特定数据类型和一种特定索引访问方法定义索引字段的语义。一个操作符类本质上指定一个特定的操作符族适用于一个特定的可索引的字段数据类型。索引的字段实际可用的族中的操作符集是接受字段的数据类型作为它们的左边的输入的那个。

表 12-52 PG_OPCLASS 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含字段，必须明确选择）。
opcmethod	oid	PG_AM.oid	操作符类所服务的索引访问方法。
opcname	name	-	这个操作符类的名称。
opcnamespace	oid	PG_NAMESPACE.oid	这个操作符类的名称空间。
opcowner	oid	PG_AUTHID.oid	操作符类属主。
opcfamily	oid	PG_OPFAMILY.oid	包含该操作符类的操作符族。
opcintype	oid	PG_TYPE.oid	操作符类索引的数据类型。
opcdefault	boolean	-	如果操作符类是opcintype的缺省，则为真。
opckeytype	oid	PG_TYPE.oid	索引数据的类型，如果和opcintype相同则为零。

一个操作符类的opcmethod必须匹配包含它的操作符族的opfmethod。

12.2.55 PG_OPERATOR

PG_OPERATOR系统表存储有关操作符的信息。

表 12-53 PG_OPERATOR 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含字段，必须明确选择）。
oprname	name	-	操作符的名称。
oprnamespace	oid	PG_NAMESPACE.oid	包含此操作符的名称空间的OID。
oprowner	oid	PG_AUTHID.oid	操作符所有者。
oprkind	"char"	-	• b=中缀(“两边”) • l=前缀(“左边”) • r=后缀(“右边”)
oprcanmerge	boolean	-	这个操作符是否支持合并连接。 • t（true）：表示支持合并连接。 • f（false）：表示不支持合并连接。
oprcanhash	boolean	-	这个操作符是否支持Hash连接。 • t（true）：表示支持Hash连接。 • f（false）：表示不支持Hash连接。
oprleft	oid	PG_TYPE.oid	左操作数的类型。
oprright	oid	PG_TYPE.oid	右操作数的类型。
oprresult	oid	PG_TYPE.oid	结果类型。
oprcom	oid	PG_OPERATOR.oid	如果存在的话，值为此操作符的交换符。不存在的话，值为0。
oprnegate	oid	PG_OPERATOR.oid	如果存在的话，值为此操作符的反转器。不存在的话，值为0。
oprcode	regproc	PG_PROC.proname	实现这个操作符的函数。

名称	类型	引用	描述
oprrest	regproc	PG_PROC.proname	此操作符的约束选择性计算函数。
oprjoin	regproc	PG_PROC.proname	此操作符的连接选择性计算函数。

12.2.56 PG_OPFAMILY

PG_OPFAMILY系统表定义操作符族。

每个操作符族是一个操作符和相关支持例程的集合，其中的例程实现为一个特定的索引访问方式指定的语义。另外，族中的操作符都是“兼容的”，通过由访问方式指定的方法。操作符族的概念允许交叉数据类型操作符和索引一起使用，并且合理的使用访问方式的语义的知识。

表 12-54 PG_OPFAMILY 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含字段，必须明确选择）。
opfmethod	oid	PG_AM.oid	操作符族使用的索引方法。
opfname	name	-	这个操作符族的名称。
opfnamespace	oid	PG_NAMESPACE.oid	这个操作符的名称空间。
opfowner	oid	PG_AUTHID.oid	操作符族的所有者。

定义一个操作符族的大多数信息不在它的PG_OPFAMILY行里面，而是在相关的行PG_AMOP，PG_AMPROC和PG_OPCLASS里。

12.2.57 PG_PARTITION

PG_PARTITION系统表存储数据库内所有分区表(partitioned table)、分区(table partition)、分区上toast表和分区索引(index partition)四类对象的信息。分区表索引(partitioned index)的信息不在PG_PARTITION系统表中保存。

表 12-55 PG_PARTITION 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
relname	name	分区表、分区、分区上toast表和分区索引的名称。

名称	类型	描述
parttype	"char"	对象类型： ● 'r'：partitioned table ● 'p'：table partition ● 'x'：index partition ● 't'：toast table
parentid	oid	当对象为分区表或分区时，此字段表示分区表在PG_CLASS中的OID。 当对象为index partition时，此字段表示所属分区表索引(partitioned index)的OID。
rangenum	integer	保留字段。
intervalnum	integer	保留字段。
partstrategy	"char"	分区表分区策略，现在仅支持： ● 'r'：范围分区。 ● 'v'：数值分区。
relfilenode	oid	table partition、index partition、分区上toast表的物理存储位置。
reltablespace	oid	table partition、index partition、分区上toast表所属表空间的OID。
relpages	double precision	统计信息：table partition、index partition的数据页数量。
reltuples	double precision	统计信息：table partition、index partition的元组数。
relallvisible	integer	统计信息：table partition、index partition的可见数据页数。
reltoastrelid	oid	table partition所对应toast表的OID。
reltoastidxid	oid	table partition所对应toast表的索引的OID。
indextblid	oid	index partition对应table partition的OID。
indisusable	boolean	分区索引是否可用。 ● t（true）：表示可用。 ● f（false）：表示不可用。
reldeltarelid	oid	Delta表的OID。
reldeltaidx	oid	Delta表的索引表的OID。
relcudescrelid	oid	CU描述表的OID。
relcudescidx	oid	CU描述表的索引表的OID。

名称	类型	描述
relfrozenxid	xid32	冻结事务ID号。 为保持前向兼容，保留此字段，新增relfrozenxid64用于记录此信息。
intspnum	integer	间隔分区所属表空间的个数。
partkey	int2vector	分区键的列号。
intervaltablespace	oidvector	间隔分区所属的表空间，间隔分区以round-robin方式落在这些表空间内。
interval	text[]	间隔分区的间隔值。
boundaries	text[]	范围分区和间隔分区的上边界。
transit	text[]	间隔分区的跳转点。
reloptions	text[]	设置partition的存储属性，与pg_class.reloptions的形态一样，用"keyword=value"格式的字符串来表示，目前用于在线扩容的信息搜集。
relfrozenxid64	xid	冻结事务ID号。
relminmxid	xid	冻结多事务ID号。

12.2.58 PG_PLTEMPLATE

PG_PLTEMPLATE系统表存储过程语言的“模板”信息。

表 12-56 PG_PLTEMPLATE 字段

名称	类型	描述
tmplname	name	这个模板所应用的语言的名称。
tmpltrusted	boolean	如果语言被认为是可信的，则为真。否则为假。
tmpldbacreate	boolean	如果语言是由数据库所有者创建的，则为真。否则为假。
tmplhandler	text	调用处理器函数的名称。
tmplinline	text	匿名块处理器的名称，若没有则为NULL。
tmplvalidator	text	校验函数的名称，如果没有则为NULL。
tmpllibrary	text	实现语言的共享库的路径。
tmplacl	aclitem[]	模板的访问权限（未使用）。

12.2.59 PG_PROC

PG_PROC系统表存储函数或过程的信息。

表 12-57 PG_PROC 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
proname	name	函数名称。
pronamespace	oid	包含该函数名称空间的OID。
proowner	oid	函数的所有者。
prolang	oid	这个函数的实现语言或调用接口。
procost	real	估算的执行成本。
prorows	real	估算的影响行的数目。
provariadic	oid	参数元素的数据类型。
protransform	regproc	此函数的简化调用方式。
proisagg	boolean	函数是否是聚集函数。 ● t（true）：表示是。 ● f（false）：表示不是。
proiswindow	boolean	函数是否是窗口函数。 ● t（true）：表示是。 ● f（false）：表示不是。
prosecdef	boolean	函数是否是一个安全定义器（也就是一个“setuid”函数）。 ● t（true）：表示是。 ● f（false）：表示不是。
proleakproof	boolean	函数是否没有副作用。如果函数没有对参数进行防泄露处理，则会抛出错误。 ● t（true）：表示没副作用。 ● f（false）：表示有副作用。
proisstrict	boolean	● t（true）：使用该函数时，如果入参有空值，则函数实际上不调用直接返回空。 ● f（false）：使用该函数时，即使入参有空值，也要调用。所以该函数必须处理空输入。
proretset	boolean	函数返回值是否为一个集合（也就是说，指定数据类型的多个数值）。 ● true：函数返回值是一个集合。 ● false：函数返回值不是一个集合。

名称	类型	描述
provolatile	"char"	该函数的结果是否只依赖于它的输入参数，或者还会被外界因素影响。 ● i：“不可变的”（immutable）函数，这样的函数对于相同的输入总是产生相同的结果。 ● s：“稳定的”（stable）函数，（对于固定输入）其结果在一次扫描里不变。 ● v：“易变”（volatile）函数，其结果可能在任何时候变化也用于那些有副作用的函数，因此调用它们无法得到优化。
pronargs	smallint	参数数目。
pronargdefaults	smallint	有默认值的参数数目。
prorettype	oid	返回值的数据类型。
proargtypes	oidvector	一个存放函数参数的数据类型的数组。数组里只包括输入参数（包括INOUT参数），代表该函数的调用签名（接口）。
proallargtypes	oid[]	一个包含函数参数的数据类型的数组。数组里包括所有参数的类型（包括OUT和INOUT参数），如果所有参数都是IN参数，则这个字段就会是空。请注意数组下标是以1为起点的，而因为历史原因，proargtypes的下标起点为0。
proargmodes	"char"[]	一个保存函数参数模式的数组，编码如下： ● i表示IN参数。 ● o表示OUT参数。 ● b表示INOUT参数。 ● v表示VARIADIC参数。 如果所有参数都是IN参数，则这个字段为空。请注意，下标对应的是proallargtypes的位置，而不是proargtypes。
proargnames	text[]	一个保存函数参数的名称的数组。没有名称的参数在数组里设置为空字符串。如果没有一个参数有名称，这个字段将是空。请注意，此数组的下标对应proallargtypes而不是proargtypes。
proargdefaults	pg_node_tree	默认值的表达式树。是PRONARGDEFAULTS元素的列表。
prosrc	text	描述函数或存储过程的定义。例如，对于解释型语言来说就是函数的源程序，或者一个连接符号，一个文件名，或者函数和存储过程创建时指定的其他任何函数体内容，具体取决于语言/调用习惯的实现。
probin	text	关于如何调用该函数的附加信息。同样，其含义也是和语言相关的。

名称	类型	描述
proconfig	text[]	函数针对运行时配置变量的本地设置。
proacl	aclitem[]	访问权限。具体请参见**GRANT**和**REVOKE**。
prodefaultargpos	int2vector	函数具有默认值的入参的位置。
fencedmode	boolean	函数的执行模式，表示函数是在fence还是not fence模式下执行。 ● true：fence模式，函数的执行会在重新fork的进程中执行。 ● false：not fence模式。 系统内建函数，fencedmode字段均为false，即not fence模式。
proshippable	boolean	表示该函数是否可以下推到DN上执行，默认值是false。 ● 对于IMMUTABLE类型的函数，函数始终可以下推到DN上执行。 ● 对于STABLE/VOLATILE类型的函数，仅当函数的属性是SHIPPABLE的时候，函数可以下推到DN执行。
propackage	boolean	表示该函数是否支持重载，默认值是false。 ● t（true）：表示支持。 ● f（false）：表示不支持。
prokind	"char"	表示该对象为函数还是存储过程。 ● 'f'：表示该对象为函数。 ● 'p'：表示该对象为存储过程。
proargsrc	text	描述兼容ORA语法定义的函数或存储过程的参数输入字符串，包括参数注释。默认值为NULL。
proisprivate	boolean	描述函数是否是PACKAGE内的私有函数，默认为false。
propackageid	oid	函数所属的package oid，如果不在package内，则为0。
proargtypesext	oidvector_extend	当函数参数较多时，用来存放函数参数的数据类型的数组。数组里只包括输入参数（包括INOUT参数），代表该函数的调用签名（接口）。
prodefaultargposext	int2vector_extend	当函数参数较多时，函数具有默认值的入参的位置。
allargtypes	oidvector	用来存放存储过程参数的数据类型的数组，包含存储过程所有参数（入参、出参、INOUT参数）。

名称	类型	描述
allargtypesext	oidvector_extend	当函数参数较多（大于666个）时，用来存放函数参数的数据类型的数组，包含所有参数（入参、出参、INOUT参数）。

12.2.60 PG_RANGE

PG_RANGE系统表存储关于范围类型的信息，除了**PG_TYPE**里类型的记录。

表 12-58 PG_RANGE 字段

名称	类型	引用	描述
rngtypid	oid	**PG_TYPE**.oid	范围类型的OID。
rngsubtype	oid	**PG_TYPE**.oid	这个范围类型的元素类型（子类型）的OID。
rngcollation	oid	**PG_COLLATION**.oid	用于范围比较的排序规则的OID，如果没有则为零。
rngsubopc	oid	**PG_OPCLASS**.oid	用于范围比较的子类型的操作符类的OID。
rngcanonical	regproc	**PG_PROC**.proname	转换范围类型为规范格式的函数名，如果没有则为0。
rngsubdiff	regproc	**PG_PROC**.proname	表示用于计算范围两元素差值的函数的名字，该函数的返回值为双精度类型，如果不存在该函数则rngsubdiff字段值为0。

若元素类型是离散的，则rngcanonical决定用于范围类型的排序顺序；若元素类型是不可排序的，则rngsubopc决定用于范围类型的排序顺序；若元素类型是可排序的，则rngsubopc和rngcollation决定用于范围类型的排序顺序。

12.2.61 PG_REPLICATION_ORIGIN

PG_REPLICATION_ORIGIN系统表包含所有已创建的复制源，该表在一个集群的所有数据库之间共享，即每个集群只有一份，而不是每个数据库一份。

表 12-59 PG_REPLICATION_ORIGIN 字段

名称	类型	描述
roident	oid	一个集群范围内唯一的复制源标识符。

名称	类型	描述
roname	text	外部的由用户定义的复制源名称。

12.2.62 PG_RESOURCE_POOL

PG_RESOURCE_POOL系统表提供了数据库资源池的信息。

表 12-60 PG_RESOURCE_POOL 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
respool_name	name	资源池名称。
mem_percent	integer	内存配置的百分比。
cpu_affinity	bigint	CPU绑定core的数值。
control_group	name	资源池所在的control group名称。
active_statements	integer	资源池上最大的并发数。
max_dop	integer	最大并发度。用作扩容的接口，表示数据重分布时，扫描并发度。
memory_limit	name	资源池最大的内存。
parentid	oid	父资源池OID。
io_limits	integer	每秒触发I/O的次数上限。行存单位是万次/s。
io_priority	name	I/O利用率高达90%时，重消耗I/O作业进行I/O资源管控时关联的优先级等级。
nodegroup	name	表示资源池所在的Node group的名称。
is_foreign	boolean	表示资源池是否用于Node group之外的用户。如果为true，表示资源池用来控制不属于当前资源池的普通用户的资源。如果为false，表示不控制不属于当前资源池的普通用户的资源。
max_worker	integer	只用于扩容的接口，表示扩容数据重分布时，表内插入并发度。

12.2.63 PG_REWRITE

PG_REWRITE系统表存储表和视图定义的重写规则。

表 12-61 PG_REWRITE 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
rulename	name	规则名称。
ev_class	oid	使用这条规则的表名称。
ev_attr	smallint	这条规则适用的字段（目前总是为零，表示整个表）。
ev_type	"char"	规则适用的事件类型。 ● 1：SELECT ● 2：UPDATE ● 3：INSERT ● 4：DELETE
ev_enabled	"char"	用于控制复制的触发。 ● O：“origin”和“local”模式时触发。 ● D：禁用触发。 ● R：“replica”时触发。 ● A：任何模式都会触发。
is_instead	boolean	如果该规则是INSTEAD规则，则为真，否则为假。
ev_qual	pg_node_tree	规则的资格条件的表达式树（以nodeToString() 形式存在）。
ev_action	pg_node_tree	规则动作的查询树（以nodeToString() 形式存在）。

12.2.64 PG_RLSPOLICY

PG_RLSPOLICY系统表存储行级访问控制策略。

表 12-62 PG_RLSPOLICY 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
polname	name	行级访问控制策略的名称。
polrelid	oid	行级访问控制策略作用的表对象oid。
polcmd	"char"	行级访问控制策略影响的SQL操作。

名称	类型	描述
polpermissive	boolean	行级访问控制策略的属性。 ● t：表达式OR条件拼接。 ● f：表达式AND条件拼接。
polroles	oid[]	行级访问控制策略影响的用户oid列表，不指定表示影响所有的用户。
polqual	pg_node_tree	行级访问控制策略的表达式。

12.2.65 PG_SECLABEL

PG_SECLABEL系统表存储数据对象上的安全标签。

PG_SHSECLABEL的作用类似，只是它是用于在一个数据库集群内共享的数据库对象的安全标签上的。

表 12-63 PG_SECLABEL 字段

名称	类型	引用	描述
objoid	oid	任意OID属性	这个安全标签所属的对象的OID。
classoid	oid	PG_CLASS.oid	出现这个对象的系统目录的OID。
objsubid	integer	-	出现在这个对象中的列的序号。
provider	text	-	与这个标签相关的标签提供程序。
label	text	-	应用于这个对象的安全标签。

12.2.66 PG_SHDEPEND

PG_SHDEPEND系统表记录数据库对象和共享对象（比如角色）之间的依赖性关系。这些信息允许GaussDB保证在企图删除这些对象之前，这些对象是没有被引用的。

PG_DEPEND的作用类似，只是它是用于在一个数据库内部的对象的依赖性关系的。

PG_SHDEPEND是在集群的所有数据库之间共享的，即每个集群只有一个，而不是每个数据库一个。

表 12-64 PG_SHDEPEND 字段

名称	类型	引用	描述
dbid	oid	PG_DATABASE.oid	依赖对象所在的数据库的OID ，如果是共享对象，则为零。
classid	oid	PG_CLASS.oid	依赖对象所在的系统表的OID。

名称	类型	引用	描述
objid	oid	任意OID属性	指定的依赖对象的OID。
objsubid	integer	-	对于一个表字段，这是字段号（objid和classid参考表本身）。对于所有其他对象类型，这个字段为零。
refclassid	oid	PG_CLASS.oid	被引用对象所在的系统表的OID(必须是一个共享表)。
refobjid	oid	任意OID属性	指定的被引用对象的OID。
deptype	"char"	-	一段代码，定义了这个依赖性关系的特定语义。参阅下文。

在任何情况下，一条PG_SHDEPEND记录就表明这个被引用的对象不能在未删除依赖对象的前提下删除。不过，deptype同时还标出了几种不同的子风格：

- SHARED_DEPENDENCY_OWNER (o)

  被引用的对象（必须是一个角色）是依赖对象的所有者。
- SHARED_DEPENDENCY_ACL (a)

  被引用的对象（必须是一个角色）在依赖对象的ACL（访问控制列表，也就是权限列表）里提到。SHARED_DEPENDENCY_ACL不会在对象的所有者上添加，因为所有者会有一个SHARED_DEPENDENCY_OWNER记录。
- SHARED_DEPENDENCY_PIN (p)

  这类记录标识系统自身依赖于该被依赖对象，因此这样的对象不能被删除。这种类型的记录只是由initdb创建。这样的依赖对象的字段都是零。
- SHARED_DEPENDENCY_ DBPRIV(d)

  被引用的对象（必须是一个角色）具有依赖对象所对应的ANY权限（指定的依赖对象的OID对应的是系统表GS_DB_PRIVILEGE中一行）。

12.2.67 PG_SHDESCRIPTION

PG_SHDESCRIPTION系统表为共享数据库对象存储可选的注释。可以使用COMMENT命令操作注释的内容，使用\d命令查看注释内容。

PG_DESCRIPTION提供了类似的功能，它记录了单个数据库中对象的注释。

PG_SHDESCRIPTION是在集群的所有数据库之间共享的，即每个集群只有一个，而不是每个数据库一个。

表 12-65 PG_SHDESCRIPTION 字段

名称	类型	引用	描述
objoid	oid	任意OID属性	所描述的对象的OID。
classoid	oid	PG_CLASS.oid	该对象出现的系统表的OID。
description	text	-	该对象的描述信息。

12.2.68 PG_SHSECLABEL

PG_SHSECLABEL系统表存储在共享数据库对象上的安全标签。安全标签可以用SECURITY LABEL命令操作。

查看安全标签的简单点的方法，请参阅**PG_SECLABELS**。

PG_SECLABEL的作用类似，只是它是用于在单个数据库内部的对象的安全标签的。

不同于大多数的系统表，PG_SHSECLABEL在一个集群中的所有数据库中共享：每个数据库集群只有一个PG_SHSECLABEL，而不是每个数据库一个。

表 12-66 PG_SHSECLABEL 字段

名称	类型	引用	描述
objoid	oid	任意OID属性	这个安全标签所属的对象的OID。
classoid	oid	**PG_CLASS**.oid	出现这个对象的系统目录的OID。
provider	text	-	与这个标签相关的标签提供程序。
label	text	-	应用于这个对象的安全标签。

12.2.69 PG_STATISTIC

PG_STATISTIC系统表存储有关该数据库中表和索引列的统计数据。默认只有系统管理员权限才可以访问此系统表，普通用户需要授权才可以访问。

表 12-67 PG_STATISTIC 字段

名称	类型	描述
starelid	oid	所描述字段所属的表或者索引。
starelkind	"char"	所属对象的类型。
staattnum	smallint	所描述字段在表中的编号，从1开始。
stainherit	boolean	是否统计有继承关系的对象。
stanullfrac	real	所描述字段中为NULL的记录的比率。
stawidth	integer	所描述字段非NULL记录的平均存储宽度，以字节计。
stadistinct	real	标识全局统计信息中所有DN上字段里唯一的非NULL数据值的数目。 • 一个大于零的数值是独立数值的实际数目。 • 一个小于零的值是distinct值所占总行数的比例，比如stadistinct=-0.5时，它的实际distinct值是总行数*0.5。 • 零值表示独立数值的数目未知。

名称	类型	描述
stakindN	smallint	一个编码，表示这种类型的统计存储在pg_statistic行的第N个“槽位”。 N的取值范围：1～5
staopN	oid	一个用于生成这些存储在第N个“槽位”的统计信息的操作符。比如，一个柱面图槽位会显示<操作符，该操作符定义了该数据的排序顺序。 N的取值范围：1～5
stanumbersN	real[]	第N个“槽位”的相关类型的数值类型统计，如果该槽位和数值类型没有关系，则就是NULL。 N的取值范围：1～5
stavaluesN	anyarray	第N个“槽位”类型的字段数据值，如果该槽位类型不存储任何数据值，则就是NULL。每个数组的元素值实际上都是指定字段的数据类型，因此，除了把这些字段的类型定义成anyarray之外，没有更好的办法。 N的取值范围：1～5
stadndistinct	real	标识DN1上字段里唯一的非NULL数据值的数目。 ● 一个大于零的数值是独立数值的实际数目。 ● 一个小于零的值是distinct值所占总行数的比例，比如stadndistinct=-0.5时，它的实际distinct值是总行数*0.5。 ● 零值表示独立数值的数目未知。
staextinfo	text	统计信息的扩展信息。预留字段。

须知

PG_STATISTIC系统表存储了统计对象的一些敏感信息，如高频值MCV。系统管理员和授权后的其他用户可以通过访问PG_STATISTIC系统表查询到统计对象的这些敏感信息。

12.2.70 PG_STATISTIC_EXT

PG_STATISTIC_EXT系统表存储有关该数据库中表的扩展统计数据。由用户指定收集哪些扩展统计数据。需要有系统管理员权限才可以访问此系统表。

表 12-68 PG_STATISTIC_EXT 字段

名称	类型	描述
starelid	oid	所描述字段所属的表或者索引。
starelkind	"char"	所属对象的类型。'c'表示表，'p'表示分区。

名称	类型	描述
stainherit	boolean	是否统计有继承关系的对象。 ● t（true）：表示统计。 ● f（false）：表示不统计。
stanullfrac	real	所描述字段中为NULL的记录的比率。
stawidth	integer	所描述字段非NULL记录的平均存储宽度，以字节计。
stadistinct	real	标识全局统计信息中所有DN上字段里唯一的非NULL数据值的数目。 ● 一个大于零的数值是独立数值的实际数目。 ● 一个小于零的值是distinct值所占总行数的比例，比如stadistinct=-0.5时，它的实际distinct值是总行数*0.5。 ● 零值表示独立数值的数目未知。
stadndistinct	real	标识DN1上字段里唯一的非NULL数据值的数目。 ● 一个大于零的数值是独立数值的实际数目。 ● 一个小于零的值是distinct值所占总行数的比例，比如stadndistinct=-0.5时，它的实际distinct值是总行数*0.5。 ● 零值表示独立数值的数目未知。
stakindN	smallint	一个编码，表示这种类型的统计存储在pg_statistic行的第N个“槽位”。 N的取值范围：1～5
staopN	oid	一个用于生成这些存储在第N个“槽位”的统计信息的操作符。比如，一个柱面图槽位会显示<操作符，该操作符定义了该数据的排序顺序。 N的取值范围：1～5
stakey	int2vector	所描述的字段编号的数组。
stanumbersN	real[]	第N个“槽位”的相关类型的数值类型统计，如果该槽位和数值类型没有关系，则就是NULL。 N的取值范围：1～5
stavaluesN	anyarray	第N个“槽位”类型的字段数据值，如果该槽位类型不存储任何数据值，则就是NULL。每个数组的元素值实际上都是指定字段的数据类型，因此，除了把这些字段的类型定义成anyarray之外，没有更好的办法。 N的取值范围：1～5
staexprs	pg_node_tree	扩展统计信息对应的表达式。

须知

PG_STATISTIC_EXT系统表存储了统计对象的一些敏感信息，如高频值MCV。系统管理员和授权后的其他用户可以通过访问PG_STATISTIC_EXT系统表查询到统计对象的这些敏感信息。

12.2.71 PG_SYNONYM

PG_SYNONYM系统表存储同义词对象名与其他数据库对象名间的映射信息。

表 12-69 PG_SYNONYM 字段

名称	类型	描述
oid	oid	数据库对象id。
synname	name	同义词名称。
synnamespace	oid	包含该同义词的名字空间的OID。
synowner	oid	同义词的所有者，通常是创建它的用户OID。
synobjschema	name	关联对象指定的模式名。
synobjname	name	关联对象名。

12.2.72 PG_TABLESPACE

PG_TABLESPACE系统表存储表空间信息。

表 12-70 PG_TABLESPACE 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
spcname	name	表空间名称。
spcowner	oid	表空间的所有者，通常是创建它的人。
spcacl	aclitem[]	访问权限。具体请参见**GRANT**和**REVOKE**。
spcoptions	text[]	表空间的选项。
spcmaxsize	text	可使用的最大磁盘空间大小，单位Byte。
relative	boolean	标识表空间指定的存储路径是否为相对路径。 • t（true）：表示是。 • f（false）：表示不是。

12.2.73 PG_TRIGGER

PG_TRIGGER系统表存储触发器信息。

表 12-71 PG_TRIGGER 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
tgrelid	oid	触发器所在表的OID。
tgname	name	触发器名。
tgfoid	oid	要被触发器调用的函数。
tgtype	smallint	触发器类型。
tgenabled	"char"	O =触发器在“origin”和“local”模式下触发。 D =触发器被禁用。 R =触发器在“replica”模式下触发。 A =触发器始终触发。
tgisinternal	boolean	内部触发器标识，如果为true表示内部触发器。
tgconstrrelid	oid	完整性约束引用的表。
tgconstrindid	oid	完整性约束的索引。
tgconstraint	oid	约束触发器在pg_constraint中的OID。
tgdeferrable	boolean	约束触发器是为DEFERRABLE类型。 ● t（true）：表示是。 ● f（false）：表示不是。
tginitdeferred	boolean	约束触发器是否为INITIALLY DEFERRED类型。 ● t（true）：表示是。 ● f（false）：表示不是。
tgnargs	smallint	触发器函数入参个数。
tgattr	int2vector	当触发器指定列时的列号，未指定则为空数组。
tgargs	bytea	传递给触发器的参数。
tgqual	pg_node_tree	表示触发器的WHEN条件，如果没有则为null。
tgowner	oid	触发器所有者

12.2.74 PG_TS_CONFIG

PG_TS_CONFIG系统表包含表示文本搜索配置的记录。一个配置指定一个特定的文本搜索解析器和一个为了每个解析器的输出类型使用的字典的列表。

解析器在PG_TS_CONFIG记录中显示，但是字典映射的标记是由PG_TS_CONFIG_MAP里面的辅助记录定义的。

表 12-72 PG_TS_CONFIG 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含字段，必须明确选择）。
cfgname	name	-	文本搜索配置名。
cfgnamespace	oid	PG_NAMESPACE.oid	包含这个配置的名称空间的OID。
cfgowner	oid	PG_AUTHID.oid	配置的所有者。
cfgparser	oid	PG_TS_PARSER.oid	这个配置的文本搜索解析器的OID。
cfoptions	text[]	-	分词相关配置选项。

12.2.75 PG_TS_CONFIG_MAP

PG_TS_CONFIG_MAP系统表包含为每个文本搜索配置的解析器的每个输出符号类型，显示哪个文本搜索字典应该被咨询、以什么顺序搜索的记录。

表 12-73 PG_TS_CONFIG_MAP 字段

名称	类型	引用	描述
mapcfg	oid	PG_TS_CONFIG.oid	拥有这个映射记录的PG_TS_CONFIG记录的OID。
maptokentype	integer	-	由配置的解析器产生的一个符号类型值。
mapseqno	integer	-	在相同mapcfg或maptokentype值的情况下，该符号类型的顺序号。
mapdict	oid	PG_TS_DICT.oid	要咨询的文本搜索字典的OID。

12.2.76 PG_TS_DICT

PG_TS_DICT系统表包含定义文本搜索字典的记录。字典取决于文本搜索模板，该模板声明所有需要的实现函数；字典本身提供模板支持的用户可设置的参数的值。

这种分工允许字典通过非权限用户创建。参数由文本字符串dictinitoption指定，参数的格式和意义取决于模板。

表 12-74 PG_TS_DICT 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含字段，必须明确选择）。
dictname	name	-	文本搜索字典名。
dictnamespace	oid	PG_NAMESPACE.oid	包含这个字典的名称空间的OID。
dictowner	oid	PG_AUTHID.oid	字典的所有者。
dicttemplate	oid	PG_TS_TEMPLATE.oid	这个字典的文本搜索模板的OID。
dictinitoption	text	-	该模板的初始化选项字符串。

12.2.77 PG_TS_PARSER

PG_TS_PARSER系统表包含定义文本解析器的记录。解析器负责分裂输入文本为词位，并且为每个词位分配标记类型。因为解析器必须通过C语言级别的函数实现，所以新解析器必须由数据库系统管理员创建。

表 12-75 PG_TS_PARSER 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含字段；必须明确选择）。
prsname	name	-	文本搜索解析器名。
prsnamespace	oid	PG_NAMESPACE.oid	包含这个解析器的名称空间的OID。
prsstart	regproc	PG_PROC.proname	解析器的启动函数名。
prstoken	regproc	PG_PROC.proname	解析器的下一个标记函数名。
prsend	regproc	PG_PROC.proname	解析器的关闭函数名。
prsheadline	regproc	PG_PROC.proname	解析器的标题函数名。
prslextype	regproc	PG_PROC.proname	解析器的lextype函数名。

12.2.78 PG_TS_TEMPLATE

PG_TS_TEMPLATE系统表包含定义文本搜索模板的记录。模板是文本搜索字典的类的实现框架。因为模板必须通过C语言级别的函数实现，索引新模板的创建必须由数据库系统管理员创建。

表 12-76 PG_TS_TEMPLATE 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含字段；必须明确选择）。
tmplname	name	-	文本搜索模板名。
tmplnamespace	oid	PG_NAMESPACE.oid	包含这个模板的名称空间的OID。
tmplinit	regproc	PG_PROC.proname	模板的初始化函数名。
tmpllexize	regproc	PG_PROC.proname	模板的lexize函数名。

12.2.79 PG_TYPE

PG_TYPE系统表存储数据类型的相关信息。

表 12-77 PG_TYPE 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
typname	name	数据类型名称。
typnamespace	oid	包含这个类型的名称空间的OID。
typowner	oid	该类型的所有者。
typlen	smallint	对于定长类型是该类型内部表现形式的字节数目。对于变长类型是负数。 ● -1：表示一种“变长”（有长度字属性的数据）。 ● -2：表示这是一个NULL结尾的C字符串。
typbyval	boolean	● true：指定内部传递这个类型的数值时是传值。 ● false：指定内部传递这个类型的数值时是传引用。 如果该类型的typlen不是1、2、4、8，typbyval建议传引用，也可以传值。变长类型通常是传引用，也可以传值。

名称	类型	描述
typtype	"char"	• b：基础类型。 • c：复合类型（比如，一个表的行类型）。 • d：域类型。 • p：伪类型。 • r：范围类型。 • e：枚举类型。 • o：集合类型。 参见typrelid和typbasetype。
typcategory	"char"	是数据类型的模糊分类，可用于解析器作为数据转换的依据。
typispreferred	boolean	• true：数据符合typcategory所指定的转换规则时进行转换。 • false：不进行转换。
typisdefined	boolean	• true：表示类型已定义。 • false：表示是一种尚未定义的类型的占位符，此时，除了该类型的名称、名称空间和OID之外没有可靠的信息。
typdelim	"char"	当分析数组输入时，分隔两个此类型数值的字符请注意该分隔符是与数组元素数据类型相关联的，而不是和数组数据类型关联。
typrelid	oid	如果是复合类型（请参见typtype），则这个字段指向PG_CLASS中定义该表的行。对于自由存在的复合类型，pg_class记录并不表示一个表，但是总需要它来查找该类型连接的PG_ATTRIBUTE记录。对于非复合类型为零。
typelem	oid	如果不为0，则它标识pg_type里面的另外一行。当前类型可以当做一个产生类型为typelem的数组来描述。一个“真正的”数组类型是变长的（typlen= -1），但是一些定长的（typlen > 0）类型也拥有非零的typelem（比如name和point）。如果一个定长类型拥有一个typelem，则他的内部形式必须是typelem数据类型的某个数目的个数值，不能有其它数据。变长数组类型有一个该数组子过程定义的头（文件）。
typarray	oid	如果不为0，则表示在pg_type中有对应的类型记录。
typinput	regproc	输入转换函数（文本格式）。
typoutput	regproc	输出转换函数（文本格式）。
typreceive	regproc	输入转换函数（二进制格式），如果没有则为0。
typsend	regproc	输出转换函数（二进制格式），如果没有则为0。
typmodin	regproc	输入类型修改符函数，如果没有则为0。

名称	类型	描述
typmodout	regproc	输出类型修改符函数，如果没有则为0。
typanalyze	regproc	自定义的ANALYZE函数，如果使用标准函数，则为0。
typalign	"char"	当存储此类型的数值时要求的对齐性质。它应用于磁盘存储以及该值的大多数形式。如果数值是连续存放的，比如在磁盘上以完全的裸数据的形式存放时，则先在此类型的数据前填充空白，这样它就可以按照要求的界限存储。对齐引用是该序列中第一个数据的开头。可能的值包含： ● c：表示char对齐，也就是不需要对齐。 ● s：表示short对齐（在大多数机器上是2字节）。 ● i：表示int对齐（在大多数机器上是4字节）。 ● d：表示double对齐（在大多数机器上是8字节，但不一定是全部）。 **须知** 对于在系统表里使用的类型，在pg_type里定义的尺寸和对齐必须和编译器在一个表示表的一行的结构里的布局一样。
typstorage	"char"	指明一个变长类型（那些有typlen = -1）是否准备好应付非常规值，以及对这种属性的类型的缺省策略是什么。可能的值包含： ● p：数值总是以简单方式存储。 ● e：数值可以存储在一个“次要”关系中（如果该关系有这么一个，请参见PG_CLASS.reltoastrelid）。 ● m：数值可以以内联的压缩方式存储。 ● x：数值可以以内联的压缩方式或者在“次要”表里存储。 **须知** m域也可以移到从属表里存储，但只是最后的解决方法（e和x域先移走）。
typnotnull	boolean	该类型是否存在NOTNULL约束。目前只用于域。
typbasetype	oid	如果这是一个衍生类型（请参见typtype），则该标识作为这个类型的基础的类型。如果不是衍生类型则为零。
typtypmod	integer	域使用typtypmod记录要作用到它们的基础类型上的typmod（如果基础类型不使用typmod则为-1）。如果这种类型不是域，则为-1。
typndims	integer	如果一个域是数组，则typndims是数组维数的数值。非域非数组域为零。
typcollation	oid	指定类型的排序规则，取值参考PG_COLLATION系统表。如果为0，则表示不支持排序。
typdefaultbin	pg_node_tree	如果为非NULL，则它是该类型缺省表达式的nodeToString() 表现形式。目前这个字段只用于域。

名称	类型	描述
typdefault	text	如果某类型没有相关缺省值，则取值是NULL。如果typdefaultbin不是NULL，则typdefault必须包含一个typdefaultbin代表的缺省表达式。如果typdefaultbin为NULL但typdefault不是，typdefault则是该类型缺省值的外部表现形式，可以把它交给该类型的输入转换器生成一个常量。
typacl	aclitem[]	访问权限。

12.2.80 PG_USER_MAPPING

PG_USER_MAPPING系统表存储从本地用户到远程的映射。

需要有系统管理员权限才可以访问此系统表。普通用户可以使用视图PG_USER_MAPPINGS进行查询。

表 12-78 PG_USER_MAPPING 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含字段，必须明确选择）。
umuser	oid	PG_AUTHID.oid	被映射的本地用户的OID，如果用户映射是公共的则为0。
umserver	oid	PG_FOREIGN_SERVER.oid	包含这个映射的外部服务器的OID。
umoptions	text[]	-	用户映射指定选项，使用“keyword=value”格式的字符串。

12.2.81 PG_USER_STATUS

PG_USER_STATUS系统表存储访问数据库用户的状态信息。需要有系统管理员权限才可以访问此系统表。

表 12-79 PG_USER_STATUS 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
roloid	oid	角色的标识。
failcount	integer	尝试失败次数。
locktime	timestamp with time zone	角色被锁定的时间点。

名称	类型	描述
rolstatus	smallint	角色的状态。 ● 0：正常状态。 ● 1：由于登录失败次数超过阈值被锁定了一定的时间。 ● 2：被管理员锁定。
permspace	bigint	角色已经使用的永久表存储空间大小。
tempspace	bigint	角色已经使用的临时表存储空间大小。
password expired	smallint	密码是否失效。 ● 0：密码有效。 ● 1：密码失效。

12.2.82 PG_WORKLOAD_GROUP

PG_WORKLOAD_GROUP系统表提供了数据库负载组的信息。

表 12-80 PG_WORKLOAD_GROUP 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
workload_gpname	name	负载组名称。
respool_oid	oid	绑定到的资源池的id。
act_statements	integer	负载组内最大的活跃语句数。

12.2.83 PGXC_CLASS

PGXC_CLASS系统表存储每张表的复制或分布信息。

表 12-81 PGXC_CLASS 字段

名称	类型	描述
pcrelid	oid	表的OID。

名称	类型	描述
pclocatortype	"char"	定位器类型。 ● H：hash ● G：Range ● L：List ● M：Modulo ● N：Round Robin ● R：Replication
pchashalgorithm	smallint	使用哈希算法分布元组。
pchashbuckets	smallint	哈希容器的值。
pgroup	name	节点群的名称。
redistributed	"char"	表已经完成重分布。
redis_order	integer	重分布的顺序。该值等于0的表在本轮重分布过程中不进行重分布。
pcattnum	int2vector	用作分布键的列标号。
nodeoids	oidvector_extend	表分布的节点OID列表。
options	text	系统内部保留字段，存储扩展状态信息。

12.2.84 PGXC_GROUP

PGXC_GROUP系统表存储节点组信息。

表 12-82 PGXC_GROUP 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
group_name	name	节点组名称。
in_redistribution	"char"	是否需要重分布。取值包括： ● n：表示NodeGroup没有再进行重分布。 ● y：表示NodeGroup是重分布过程中的源节点组。 ● t：表示NodeGroup是重分布过程中的目的节点组。
group_members	oidvector_extend	节点组的节点OID列表。
group_buckets	text	分布数据桶的集合。

名称	类型	描述
is_installation	boolean	是否安装子集群。 ● t（true）：表示安装。 ● f（false）：表示不安装。
group_acl	aclitem[]	访问权限。
group_kind	"char"	node group类型，取值包括： ● i：表示installation node group。 ● n：表示普通集群node group。 ● e：表示弹性集群。
group_parent	oid	如果是子node group，该字段表示父node group的OID，如果是父node group，该字段值为空。

12.2.85 PGXC_NODE

PGXC_NODE系统表存储集群节点信息。

表 12-83 PGXC_NODE 字段

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
node_name	name	节点名称。
node_type	"char"	节点类型。 ● C：协调节点。 ● D：数据节点。 ● S：数据节点的备节点。
node_port	integer	节点的端口号。
node_host	name	节点的主机名称或者IP（如配置为虚拟IP，则为虚拟IP）。
node_port1	integer	复制节点的端口号。
node_host1	name	复制节点的主机名称或者IP（如配置为虚拟IP，则为虚拟IP）。
hostis_primary	boolean	表明当前节点是否发生主备切换。 ● t（true）：表示发生。 ● f（false）：表示不发生。

名称	类型	描述
nodeis_primary	boolean	在replication表下，是否优选当前节点作为优先执行的节点进行非查询操作。 ● t（true）：表示优选。 ● f（false）：表示不优选。
nodeis_preferred	boolean	在replication表下，是否优选当前节点作为首选的节点进行查询。 ● t（true）：表示优选。 ● f（false）：表示不优选。
node_id	integer	节点标识符。由node_name经过hash函数计算后得到。
sctp_port	integer	主节点使用TCP代理通信库的数据通道侦听端口（当前版本已经不再支持SCTP通信库）。
control_port	integer	主节点使用TCP代理通信库的控制通道侦听端口。
sctp_port1	integer	备节点使用TCP代理通信库的数据通道侦听端口（当前版本已经不再支持SCTP通信库）。
control_port1	integer	备节点使用TCP代理通信库的控制通道侦听端口。
nodeis_central	boolean	表明当前节点是否为中心控制节点，只用于CN，对DN无效。 ● t（true）：表示是。 ● f（false）：表示不是。
nodeis_active	boolean	表明当前节点是否是正常状态，用于标记CN是否被剔除，对DN无效。 ● t（true）：表示是。 ● f（false）：表示不是。

12.2.86 PGXC_REDISTB

PGXC_REDISTB表是在扩容期间创建的表，每个数据库创建一张，用来记录用户表的重分布状态，扩容结束后会被删除。只有具有connect权限的用户可查看。

表 12-84

名称	类型	描述
relname	name	用户表的名称
nspname	name	表所在的表空间名称
pcrelid	oid	表的oid

名称	类型	描述
pclocatortype	character	定位器类型： H: hash M: Modulo N: Round Robin R: Replicate
pchashalgorithm	smallint	使用哈希算法分布元组
pchashbuckets	smallint	哈希容器的值
pgroup	name	表所属的node group
redistributed	character	表当前的状态 i：表示正在进行重分布 y：表示已经完成重分布 n：表示还未进行重分布 d：表示重分布已完成，临时表还未删除
redis_order	interger	表重分布的顺序（默认值1024，设置为0表示该表不进行重分布；数值越小，越先进行重分布）
pcattnum	int2vector	用作分布键的列标号
nodeoids	oidvector_extend	表所在node group的节点id
internal_mask	integer	reloption中是否包含内部信息。当前支持的取值如下： 0x0，未开启内部掩码。 0x8000，开启内部掩码。 0x01，该表禁止insert。 0x02，该表禁止delete。 0x04，该表禁止alter。 0x08，该表禁止select。 0x0100，该表禁止update。
table_size	bigint	表的大小，单位字节。

12.2.87 PGXC_SLICE

PGXC_SLICE表是针对range范围分布和list分布创建的系统表，用来记录分布具体信息，当前不支持range interval自动扩展分片，不过在系统表中预留。

表 12-85 PGXC_SLICE 字段

名称	类型	描述
relname	name	表名或者分片名，通过type区分。
type	"char"	● 't'：relname是表名。 ● 's'：relname是分片的名字。
strategy	"char"	● 'r'：为range分布表。 ● 'l'：为list分布表。 后续interval分片会扩展该值。
relid	oid	该tuple隶属的分布表oid。
referenceoid	oid	所参考分布表的oid，用于slice reference建表语法。
sindex	integer	当为list分布表时，表示当前boundary在某个分片内的位置。
interval	text[]	预留字段。
transitboundary	text[]	预留字段。
transitno	integer	预留字段。
nodeoid	oid	当relname为分片名时，表示该分片的数据存放在哪个DN上，nodeoid表示这个DN的oid。
boundaries	text[]	当relname为分片名时，对应该分片的边界值。
specified	boolean	当前分片对应的DN是否是用户在DDL中显示指定的。
sliceorder	integer	用户定义分片的顺序。

12.2.88 PLAN_TABLE_DATA

PLAN_TABLE_DATA存储了用户通过执行EXPLAIN PLAN收集到的计划信息。与PLAN_TABLE视图不同的是PLAN_TABLE_DATA表存储了所有session和user执行EXPLAIN PLAN收集的计划信息。

表 12-86 PLAN_TABLE_DATA 字段

名称	类型	描述
session_id	text	表示插入该条数据的会话，由服务线程启动时间戳和服务线程ID组成。受非空约束限制。

名称	类型	描述
user_id	oid	用户ID，用于标识触发插入该条数据的用户。受非空约束限制。
statement_id	character varying(30)	用户输入的查询标签。
plan_id	bigint	查询计划标识。该标识在计划生成阶段自动产生，供内核工程师调试使用。
id	integer	计划中的节点编号。
operation	character varying(30)	操作描述。
options	character varying(255)	操作选项。
object_name	name	操作对应的对象名，来自于用户定义。
object_type	character varying(30)	对象类型。
object_owner	name	对象所属schema，来自于用户定义。
projection	character varying(4000)	操作输出的列信息。
cost	double precision	优化器对算子估算的执行代价。
cardinality	double precision	优化器对算子估算的结果行数。

说明

- PLAN_TABLE_DATA中包含了当前节点所有用户、所有会话的数据，仅管理员有访问权限。普通用户可以通过**PLAN_TABLE**视图查看属于自己的数据。
- 对于不活跃（已退出）的会话，其在PLAN_TABLE_DATA中的数据会在一定时间（默认5min）后被gs_clean清理。用户也可以手动执行gs_clean -C选项对表中不活跃的会话数据进行清理。
- PLAN_TABLE_DATA中的数据是用户通过执行EXPLAIN PLAN命令后由系统自动插入表中，因此禁止用户手动对数据进行插入或更新，否则会引起表中的数据混乱。需要对表中数据删除时，建议通过**PLAN_TABLE**视图进行。
- statement_id、object_name、object_owner和projection字段内容遵循用户定义的大小写存储，其它字段内容采用大写存储。

12.2.89 STATEMENT_HISTORY

获得当前节点的执行语句的信息。查询系统表必须具有sysadmin权限。只可在系统库中查询到结果，用户库中无法查询。

对于此系统表查询有如下约束：

- 必须在postgres库内查询，其它库中不存数据。
- 此系统表受track_stmt_stat_level控制，默认为"OFF,L0"，第一部分控制Full SQL，第二部分控制Slow SQL，具体字段记录级别见下表。
- 对于Slow SQL，当track_stmt_stat_level的值为非OFF时，且SQL执行时间超过log_min_duration_statement，会记录为慢SQL。

表 12-87 STATEMENT_HISTORY 字段

名称	类型	描述	记录级别
db_name	name	数据库名称。	L0
schema_name	name	schema名称。	L0
origin_node	integer	节点名称。	L0
user_name	name	用户名。	L0
application_name	text	用户发起的请求的应用程序名称。	L0
client_addr	text	用户发起的请求的客户端地址。	L0
client_port	integer	用户发起的请求的客户端端口。	L0
unique_query_id	bigint	归一化SQL ID。	L0
debug_query_id	bigint	唯一SQL ID。	L0
query	text	归一化SQL(仅在CN上有值)。	L0
start_time	timestamp with time zone	语句启动的时间。	L0
finish_time	timestamp with time zone	语句结束的时间。	L0
slow_sql_threshold	bigint	语句执行时慢SQL的标准。	L0
transaction_id	bigint	事务ID。	L0
thread_id	bigint	执行线程ID。	L0

名称	类型	描述	记录级别
session_id	bigint	用户session id。	L0
n_soft_parse	bigint	软解析次数, n_soft_parse + n_hard_parse可 能大于n_calls, 因为子查询未计入n_calls。	L0
n_hard_parse	bigint	硬解析次数, n_soft_parse + n_hard_parse可 能大于n_calls, 因为子查询未计入n_calls。	L0
query_plan	text	语句执行计划。	L1
n_returned_rows	bigint	SELECT返回的结果集行数。	L0
n_tuples_fetched	bigint	随机扫描行。	L0
n_tuples_returned	bigint	顺序扫描行。	L0
n_tuples_inserted	bigint	插入行。	L0
n_tuples_updated	bigint	更新行。	L0
n_tuples_deleted	bigint	删除行。	L0
n_blocks_fetched	bigint	buffer的块访问次数。	L0
n_blocks_hit	bigint	buffer的块命中次数。	L0
db_time	bigint	有效的DB时间花费，多线程将累加（单位：微秒）。	L0
cpu_time	bigint	CPU消耗时间（单位：微秒）。	L0
execution_time	bigint	执行器内执行时间（单位：微秒）。	L0
parse_time	bigint	SQL解析时间（单位：微秒）。	L0
plan_time	bigint	SQL生成计划时间（单位：微秒）。	L0
rewrite_time	bigint	SQL重写时间（单位：微秒）。	L0
pl_execution_time	bigint	plpgsql上的执行时间（单位：微秒）。	L0

名称	类型	描述	记录级别
pl_compilation_time	bigint	plpgsql上的编译时间（单位：微秒）。	L0
data_io_time	bigint	I/O上的时间花费（单位：微秒）。	L0
net_send_info	text	通过物理连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。在分布式数据库中，CN与CN、CN与客户端以及CN与DN之间都是通过物理连接进行通信，通过该字段可以分析SQL在分布式系统下的网络开销。例如：{"time":xxx, "n_calls":xxx, "size":xxx}。	L0
net_recv_info	text	通过物理连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。在分布式数据库中，CN与CN、CN与客户端以及CN与DN之间都是通过物理连接进行通信，通过该字段可以分析SQL在分布式系统下的网络开销。例如：{"time":xxx, "n_calls":xxx, "size":xxx}。	L0
net_stream_send_info	text	通过逻辑连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。在分布式数据库中，不同分片的DN之间通过逻辑连接进行通信，通过该字段可以分析SQL在分布式系统下的网络开销。 例如：{"time":xxx, "n_calls":xxx, "size":xxx}。	L0
net_stream_recv_info	text	通过逻辑连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。在分布式数据库中，不同分片的DN之间通过逻辑连接进行通信，通过该字段可以分析SQL在分布式系统下的网络开销。 例如：{"time":xxx, "n_calls":xxx, "size":xxx}。	L0
lock_count	bigint	加锁次数。	L0
lock_time	bigint	加锁耗时。	L1
lock_wait_count	bigint	加锁等待次数。	L0
lock_wait_time	bigint	加锁等待耗时。	L1

名称	类型	描述	记录级别
lock_max_count	bigint	最大持锁数量。	L0
lwlock_count	bigint	轻量级加锁次数（预留）。	L0
lwlock_wait_count	bigint	轻量级等锁次数。	L0
lwlock_time	bigint	轻量级加锁时间（预留）。	L1
lwlock_wait_time	bigint	轻量级等锁时间。	L1
details	bytea	语句锁事件的列表，该列表按时间顺序记录事件，记录的数量受参数track_stmt_details_size的影响，该字段为二进制，需要借助解析函数pg_catalog.statement_detail_decode读取，见（表7-60）。 事件包括： • 加锁开始 • 加锁结束 • 等锁开始 • 等锁结束 • 放锁开始 • 放锁结束 • 轻量级等锁开始 • 轻量级等锁结束	L2
is_slow_sql	boolean	该SQL是否为slow SQL。 • t（true）：表示是。 • f（false）：表示不是。	L0
trace_id	text	驱动传入的trace id，与应用的一次请求相关联。	L0

名称	类型	描述	记录级别
advise	text	可能导致该SQL为slow SQL的风险信息（可能同时存在多种风险）。 ● Cast Function Cause Index Miss.：表示存在隐式转换导致索引匹配失败的风险。 ● Limit too much rows.：表示存在limit值过大导致SQL变慢的风险。 ● Proleakproof of function is false.：表示函数的proleakproof值为false，此时函数在生成计划时因存在数据泄露的风险而不会使用统计信息，影响生成计划的准确性，从而存在SQL变慢的风险。	L0

12.2.90 STREAMING_STREAM

STREAMING_STREAM系统表存储所有STREAM对象的元数据信息。

表 12-88 STREAMING_STREAM 字段

名称	类型	描述
relid	oid	STREAM对象的标识。
queries	bytea	该STREAM对应CONTVIEW的位图映射。

12.2.91 STREAMING_CONT_QUERY

STREAMING_CONT_QUERY系统表存储所有CONTVIEW对象的元数据信息。

表 12-89 STREAMING_CONT_QUERY 字段

名称	类型	描述
id	integer	CONTVIEW对象唯一的标识符，不可重复。
type	"char"	标识CONTVIEW的类型。 ● 'r'：表示该CONTVIEW是基于行存存储模型。
relid	oid	CONTVIEW对象的标识。
defrelid	oid	CONTVIEW对应的持续计算规则VIEW的标识。

名称	类型	描述
active	boolean	标识CONTVIEW是否处于持续计算状态。 ● t（true）：表示是。 ● f（false）：表示不是。
streamrelid	oid	CONTVIEW对应的STREAM的标识。
matrelid	oid	CONTVIEW对应物化表的标识。
lookupidxid	oid	CONTVIEW对应GROUP LOOK UP INDEX的标识，此字段内部使用，仅行存具有。
step_factor	smallint	标识CONTVIEW的步进模式。主要取值为0（无重叠窗口）和1（滑动窗口，步长为1）。
ttl	integer	CONTVIEW设置的ttl_interval参数值。
ttl_attno	smallint	CONTVIEW设置的TTL功能对应时间列的字段编号。
dictrelid	oid	CONTVIEW对应字典表的标识。
grpnum	smallint	CONTVIEW持续计算规则中维度列的个数，此字段内部使用。
grpidx	int2vector	CONTVIEW持续计算规则中维度列在TARGET LIST的索引，此字段内部使用。

12.2.92 STREAMING_REAPER_STATUS

STREAMING_REAPER_STATUS系统表存储流引擎reaper线程的状态信息。由于规格变更，当前版本已经不再支持本特性，请不要使用。

表 12-90 STREAMING_REAPER_STATUS 字段

名称	类型	描述
id	integer	CONTVIEW对象唯一的标识符，不可重复。
contquery_name	name	CONTVIEW对象的名称。
gather_interval	text	CONTVIEW对象设置的gather_interval参数值（自动聚合特定时间前历史数据的时间参数）。
gather_completion_time	text	CONTVIEW对象最近一次的GATHER（历史数据聚合）的完成时间。

12.3 系统视图

12.3.1 ADM_COL_COMMENTS

ADM_COL_COMMENTS视图显示数据库表中字段的注释信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-91 ADM_COL_COMMENTS 字段

名称	类型	描述
owner	character varying(64)	表的所有者。
table_name	character varying(64)	表名。
column_name	character varying(64)	列名。
comments	text	注释。

12.3.2 ADM_CONS_COLUMNS

ADM_CONS_COLUMNS视图显示数据库表中约束字段的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-92 ADM_CONS_COLUMNS 字段

名称	类型	描述
owner	character varying(64)	约束创建者。
column_name	character varying(64)	约束相关的列名。
constraint_name	character varying(64)	约束名。
position	smallint	表中列的位置。
table_name	character varying(64)	约束相关的表名。

12.3.3 ADM_CONSTRAINTS

ADM_CONSTRAINTS视图显示数据库表中约束的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-93 ADM_CONSTRAINTS 字段

名称	类型	描述
owner	character varying(64)	约束创建者。
constraint_name	character varying(64)	约束名。
constraint_type	text	约束类型： ● c：表示检查约束。 ● f：表示外键约束。 ● p：表示主键约束。 ● u：表示唯一约束。
table_name	character varying(64)	约束相关的表名。
index_owner	character varying(64)	约束相关的索引的所有者（只针对唯一约束和主键约束）。
index_name	character varying(64)	约束相关的索引名（只针对唯一约束和主键约束）。

12.3.4 ADM_DATA_FILES

ADM_DATA_FILES视图显示数据库文件的描述。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-94 ADM_DATA_FILES 字段

名称	类型	描述
tablespace_name	name	文件所属的表空间的名称。
bytes	double precision	文件的字节长度。

12.3.5 ADM_HIST_SNAPSHOT

ADM_HIST_SNAPSHOT视图记录当前系统中存储的WDR快照数据的索引信息、开始时间。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。

表 12-95 ADM_HIST_SNAPSHOT 字段

名称	类型	描述
snap_id	bigint	WDR快照序号。
begin_interval_time	timestamp	WDR快照的开始时间。

12.3.6 ADM_HIST_SQL_PLAN

ADM_HIST_SQL_PLAN视图描述当前用户通过执行EXPLAIN PLAN收集到的计划信息。

表 12-96 ADM_HIST_SQL_PLAN 字段

名称	类型	描述
sql_id	character varying(30)	表示插入该条数据的会话，由服务线程启动时间戳和服务线程ID组成。受非空约束限制。
plan_hash_value	bigint	查询标识。
operation	character varying(30)	操作描述。
options	character varying(255)	操作选项。
object_name	name	操作对应的对象名，来自于用户定义。

12.3.7 ADM_HIST_SQLSTAT

ADM_HIST_SQLSTAT视图描述当前节点的执行语句的信息。

表 12-97 ADM_HIST_SQLSTAT 字段

名称	类型	描述
instance_number	integer	快照的实例编号。
sql_id	bigint	查询标识。
plan_hash_value	bigint	归一化SQL ID。
module	text	包含第一次解析 SQL 语句时正在执行的模块的名称。

名称	类型	描述
elapsed_time_delta	bigint	有效的DB时间花费，多线程将累加（单位：微秒）。
cpu_time_delta	bigint	CPU时间（单位：微秒）。
executions_delta	integer	自从它被带入库缓存以来在此对象上发生的执行次数增量。
iowait_delta	bigint	I/O上的时间花费（单位：微秒）。
apwait_delta	integer	应用程序等待时间的Delta值。
rows_processed_delta	bigint	SELECT返回的结果集行数。
snap_id	integer	唯一快照 ID。

12.3.8 ADM_IND_COLUMNS

ADM_IND_COLUMNS视图显示数据库中所有索引字段的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-98 ADM_IND_COLUMNS 字段

名称	类型	描述
index_owner	character varying(64)	索引的所有者。
index_name	character varying(64)	索引名。
table_owner	character varying(64)	表的所有者。
table_name	character varying(64)	表名。
column_name	name	列名。
column_position	smallint	索引中列的位置。

12.3.9 ADM_IND_EXPRESSIONS

ADM_IND_EXPRESSIONS视图显示数据库中表达式索引的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-99 ADM_IND_EXPRESSIONS 字段

名称	类型	描述
table_owner	character varying(64)	表的所有者。
table_name	character varying(64)	表名。
index_owner	character varying(64)	索引的所有者。
index_name	character varying(64)	索引名。
column_expression	text	定义列的基于函数的索引表达式。
column_position	smallint	索引中列的位置。

12.3.10 ADM_IND_PARTITIONS

ADM_IND_PARTITIONS视图显示数据库中所有索引分区的信息。数据库中每个分区表的每个索引分区（如果存在的话）在ADM_IND_PARTITIONS里都会有一行记录。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-100 ADM_IND_PARTITIONS 字段

名称	类型	描述
index_owner	character varying(64)	索引分区所属分区表索引的所有者的名称。
index_name	character varying(64)	索引分区所属分区表索引的名称。
partition_name	character varying(64)	索引分区的名称。
def_tablespace_name	name	索引分区的表空间名称。
high_value	text	索引分区所对应分区的上边界。
index_partition_usable	boolean	索引分区是否可用。 ● t（true）：表示可用。 ● f（false）：表示不可用。
schema	character varying(64)	索引分区所属分区表索引的模式。

12.3.11 ADM_INDEXES

ADM_INDEXES视图显示数据库下所有索引的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-101 ADM_INDEXES 字段

名称	类型	描述
owner	character varying(64)	索引的所有者。
index_name	character varying(64)	索引名称。
table_name	character varying(64)	索引对应的表名。
uniqueness	text	表示该索引是否为唯一索引。
partitioned	character(3)	表示该索引是否具有分区表的性质。
generated	character varying(1)	表示该索引的名称是否为系统生成。

12.3.12 ADM_OBJECTS

ADM_OBJECTS视图显示数据库中所有数据库对象的信息。需要有系统管理员权限才可以访问，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-102 ADM_OBJECTS 字段

名称	类型	描述
owner	name	对象的所有者。
object_name	name	对象的名称。
object_id	oid	对象的OID。
object_type	name	对象的类型。例如table，schema，index等。
namespace	oid	对象所在的命名空间。
created	timestamp with time zone	对象的创建时间
last_ddl_time	timestamp with time zone	对象的最后修改时间

须知

created和last_ddl_time支持的范围参见**PG_OBJECT**中的记录范围。

12.3.13 ADM_PART_INDEXES

ADM_PART_INDEXES视图显示数据库中所有分区表索引的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-103 ADM_PART_INDEXES 字段

名称	类型	描述
def_tablespace_name	name	分区表索引的表空间名称。
index_owner	character varying(64)	分区表索引的所有者名称。
index_name	character varying(64)	分区表索引的名称。
partition_count	bigint	分区表索引的索引分区的个数。
partitioning_key_count	integer	分区表的分区键个数。
partitioning_type	text	分区表的分区策略。 **说明** 当前分区表策略仅支持范围分区（Range Partitioning）。
schema	character varying(64)	分区表索引所属模式名称。
table_name	character varying(64)	分区表索引所属的分区表名称。

12.3.14 ADM_PART_TABLES

ADM_PART_TABLES视图显示数据中所有分区表的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-104 ADM_PART_TABLES 字段

名称	类型	描述
table_owner	character varying(64)	分区表的所有者名称。
table_name	character varying(64)	分区表的名称。
partitioning_type	text	分区表的分区策略。 **说明** 当前分区表策略仅支持范围分区（Range Partitioning）。

名称	类型	描述
partition_count	bigint	分区表的分区个数。
partitioning_key_count	integer	分区表的分区键个数。
def_tablespace_name	name	分区表的表空间名称。
schema	character varying(64)	分区表的模式。

12.3.15 ADM_PROCEDURES

ADM_PROCEDURES视图显示数据库中所有存储过程或函数的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-105 ADM_PROCEDURES 字段

名称	类型	描述
owner	character varying(64)	存储过程或函数的所有者。
object_name	character varying(64)	存储过程或函数名称。
argument_number	smallint	存储过程入参个数。

12.3.16 ADM_SCHEDULER_JOBS

ADM_SCHEDULER_JOBS视图显示数据库中所有DBE_SCHEDULER定时任务的信息。

表 12-106 ADM_SCHEDULER_JOBS 字段

名称	类型	描述
owner	name	定时任务所有者。
job_name	text	定时任务名。
job_style	text	定时任务行为模式。
job_creator	name	定时任务创建者。
program_name	text	定时任务引用的程序的名称。
job_action	text	定时任务的程序内容。
number_of_arguments	text	定时任务的参数个数。
schedule_name	text	定时任务引用的调度的名称。

名称	类型	描述
start_date	timestamp without time zone	定时任务的起始时间。
repeat_interval	text	定时任务的任务周期。
end_date	timestamp without time zone	定时任务的失效时间。
job_class	text	定时任务所属的定时任务类的名称。
enabled	boolean	定时任务的启用状态。
auto_drop	text	定时任务的自动删除功能状态。
state	"char"	定时任务的状态。
failure_count	smallint	定时任务失败次数统计。
last_start_date	timestamp without time zone	定时任务上次拉起时间。
next_run_date	timestamp without time zone	定时任务下次执行时间。
destination	text	定时任务目标名称。
credential_name	text	定时任务证书名称。
comments	text	定时任务的备注。

12.3.17 ADM_SEQUENCES

ADM_SEQUENCES视图显示数据库下的所有序列信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-107 ADM_SEQUENCES 字段

名称	类型	描述
sequence_owner	character varying(64)	序列的所有者。
sequence_name	character varying(64)	序列名称。
min_value	int16	序列的最小值。
max_value	int16	序列的最大值。
increment_by	int16	序列的递增值。
last_number	int16	上一序列的值。

名称	类型	描述
cache_size	int16	序列磁盘缓存大小。
cycle_flag	character(1)	表示序列是否是循环序列，取值为Y或N： ● Y表示是循环序列。 ● N表示不是循环序列。

12.3.18 ADM_SOURCE

ADM_SOURCE视图显示数据库中所有存储过程或函数的信息，且提供存储过程或函数定义的字段。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-108 ADM_SOURCE 字段

名称	类型	描述
owner	character varying(64)	存储过程或函数的所有者。
name	character varying(64)	存储过程或函数名称。
text	text	存储过程或函数的定义。

12.3.19 ADM_SYNONYMS

ADM_SYNONYMS视图显示数据库中所有同义词的信息。需要有系统管理员权限才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-109 ADM_SYNONYMS 字段

名称	类型	描述
owner	text	同义词的所有者。
schema_name	text	同义词所属模式名。
synonym_name	text	同义词的名称。
table_owner	text	关联对象的所有者。尽管该列称为table_owner，但它拥有的该关联对象不一定是表，可以是任何数据库通用对象，例如视图、存储过程、同义词等。

名称	类型	描述
table_name	text	关联对象名。尽管该列称为table_name，但此关联对象不一定是表，可以是任何数据库通用对象，例如视图、存储过程、同义词等。
table_schema_name	text	关联对象所属模式名。尽管该列称为table_schema_name，但此schema下的该关联对象不一定是表，可以是任何数据库通用对象，例如视图、存储过程、同义词等。

12.3.20 ADM_TAB_COLUMNS

ADM_TAB_COLUMNS视图显示关于表的字段的信息。数据库中每个表的每一个字段在ADM_TAB_COLUMNS里各有一行对应的数据。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-110 ADM_TAB_COLUMNS 字段

名称	类型	描述
owner	character varying(64)	表的所有者。
table_name	character varying(64)	表的名称。
column_name	character varying(64)	列名。
data_type	character varying(128)	列的数据类型。
data_length	integer	列的字节长度。
data_precision	integer	数据类型的精度，对于numeric数据类型有效，其他类型为NULL。
data_scale	integer	小数点右边的位数，对于numeric数据类型有效，其他类型为0。
nullable	bpchar	该列是否允许为空，对于主键约束和非空约束，该值为n。
column_id	integer	创建表时列的序号。
avg_col_len	numeric	列的平均长度（单位字节）。

名称	类型	描述
char_length	numeric	列的长度（以字符计），只对varchar，nvarchar2，bpchar，char类型有效。
comments	text	注释。

12.3.21 ADM_TAB_COMMENTS

ADM_TAB_COMMENTS视图显示数据库中所有表和视图的注释信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-111 ADM_TAB_COMMENTS 字段

名称	类型	描述
owner	character varying(64)	表或视图的所有者。
table_name	character varying(64)	表或视图的名称。
comments	text	注释。

12.3.22 ADM_TAB_PARTITIONS

ADM_TAB_PARTITIONS视图显示数据库中所有的分区信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-112 ADM_TAB_PARTITIONS 字段

名称	类型	描述
table_owner	character varying(64)	角色名称。
table_name	character varying(64)	关系表名称。
partition_name	character varying(64)	分区名称。
high_value	text	范围分区和间隔分区的上边界。
tablespace_name	name	分区表的表空间名称。
schema	character varying(64)	名称空间的名称。

12.3.23 ADM_TABLES

ADM_TABLES视图显示数据库中所有表的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-113 ADM_TABLES 字段

名称	类型	描述
owner	character varying(64)	表的所有者。
table_name	character varying(64)	表名称。
tablespace_name	character varying(64)	存储表的表空间名称。
dropped	character varying	当前表是否已删除： ● YES：表示已删除。 ● NO：表示未删除。
num_rows	numeric	表的估计行数。
status	character varying(8)	当前表是否有效。 ● VALID：当前表有效。 ● UNUSABLE：当前表不可用。
temporary	character(1)	表是否为临时表： ● Y：表示是临时表。 ● N：表示不是临时表。

12.3.24 ADM_TABLESPACES

ADM_TABLESPACES视图显示有关可用的表空间的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-114 ADM_TABLESPACES 字段

名称	类型	描述
tablespace_name	character varying(64)	表空间名称。

12.3.25 ADM_TRIGGERS

ADM_TRIGGERS视图显示数据库中触发器的信息。默认只有系统管理员权限才可以访问此系统表，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-115 ADM_TRIGGERS 字段

名称	类型	描述
trigger_name	character varying(64)	触发器名称。
table_owner	character varying(64)	角色名称。
table_name	character varying(64)	关系表名称。
status	character varying(64)	• O：触发器在“origin”和“local”模式下触发。 • D：触发器被禁用。 • R：触发器在“replica”模式下触发。 • A：触发器始终触发。

12.3.26 ADM_TYPE_ATTRS

ADM_TYPE_ATTRS视图描述当前数据库对象类型的属性。

表 12-116 ADM_TYPE_ATTRS 字段

名称	类型	描述
owner	oid	该类型的所有者。
type_name	name	数据类型名称。
attr_name	name	字段名。
attr_type_mod	integer	记录创建新表时支持的类型特定的数据（比如一个varchar字段的最大长度）。它传递给类型相关的输入和长度转换函数当做第三个参数。其值对那些不需要ATTTYPMOD的类型通常为-1。
attr_type_owner	oid	该类型属性的所有者。
attr_type_name	name	数据类型属性名称。
length	smallint	对于定长类型是该类型内部表现形式的字节数目。对于变长类型是负数。 • -1表示一种“变长”（有长度字属性的数据）。 • -2表示这是一个NULL结尾的C字符串。
precision	integer	数字类型的精度。
scale	integer	数字类型的范围。
character_set_name	character(1)	属性的字符集名称（c或n）。

名称	类型	描述
attr_no	smalllint	属性编号。
inherited	character(1)	表示属性是否继承自超级类型 （Y或N)。

12.3.27 ADM_USERS

ADM_USERS视图显示所有数据库用户的信息。需要有系统管理员权限才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-117 ADM_USERS 字段

名称	类型	描述
username	character varying(64)	用户名称。

12.3.28 ADM_VIEWS

ADM_VIEWS视图显示数据库中视图的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-118 ADM_VIEWS 字段

名称	类型	描述
owner	character varying(64)	视图的所有者。
view_name	character varying(64)	视图名称。

12.3.29 COMM_CLIENT_INFO

COMM_CLIENT_INFO视图显示单个节点活跃的客户端连接信息（DN上查询该视图显示 CN连接DN的信息），默认只有系统管理员权限才可以访问此系统视图。

表 12-119 COMM_CLIENT_INFO 字段

名称	类型	描述
node_name	text	当前DN节点的名称，如dn_6001_6002_6003。
app	text	DN上查询该视图，app显示为连接当前DN的客户端，如coordinator(CN)、GTM、DN等。
tid	bigint	当前线程的线程号。

名称	类型	描述
lwtid	integer	当前线程的轻量级线程号。
query_id	bigint	查询ID，对应debug_query_id。
socket	integer	如果是物理连接，显示socket fd。
remote_ip	text	对端节点IP。
remote_port	text	对端节点port。
logic_id	integer	如果是逻辑连接，显示sid。

12.3.30 DB_ALL_TABLES

DB_ALL_TABLES视图显示当前用户所能访问的表或视图的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-120 DB_ALL_TABLES 字段

名称	类型	描述
owner	name	表或视图的所有者。
table_name	name	表或视图的名称。
tablespace_name	name	表或视图所在的表空间。

12.3.31 DB_COL_COMMENTS

DB_COL_COMMENTS视图显示当前用户可访问的表中字段的注释信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-121 DB_COL_COMMENTS 字段

名称	类型	描述
owner	character varying(64)	表的所有者。
table_name	character varying(64)	表名。
column_name	character varying(64)	列名。
comments	text	注释。

12.3.32 DB_CONS_COLUMNS

DB_CONS_COLUMNS视图显示当前用户可访问的约束字段的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-122 DB_CONS_COLUMNS 字段

名称	类型	描述
owner	character varying(64)	约束创建者。
constraint_name	character varying(64)	约束名。
table_name	character varying(64)	约束相关的表名。
column_name	character varying(64)	约束相关的列名。
position	smallint	表中列的位置。

12.3.33 DB_CONSTRAINTS

DB_CONSTRAINTS视图显示当前用户可访问的约束的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-123 DB_CONSTRAINTS 字段

名称	类型	描述
owner	character varying(64)	约束创建者。
constraint_name	character varying(64)	约束名。
constraint_type	text	约束类型： ● c表示检查约束。 ● f表示外键约束。 ● p表示主键约束。 ● u表示唯一约束。
table_name	character varying(64)	约束相关的表名。
index_owner	character varying(64)	约束相关的索引的所有者（只针对唯一约束和主键约束）。
index_name	character varying(64)	约束相关的索引名（只针对唯一约束和主键约束）。

12.3.34 DB_DEPENDENCIES

DB_DEPENDENCIES视图显示当前用户可访问的函数、高级包之间的依赖关系。该视图同时存在于PG_CATALOG和SYS Schema下。

须知

因为相关信息的限制，在目前GaussDB中，该表为一空表，表内没有任何记录。

表 12-124 DB_DEPENDENCIES 字段

名称	类型	描述
owner	character varying(30)	对象的所有者。
name	character varying(30)	对象的名称。
type	character varying(17)	对象的类型。
referenced_owner	character varying(30)	引用对象的所有者。
referenced_name	character varying(64)	引用对象的名称。
referenced_type	character varying(17)	引用对象的类型。
referenced_link_name	character varying(128)	到引用对象的链接的名称。
schemaid	numeric	当前schema的ID。
dependency_type	character varying(4)	依赖类型（REF(软引用）或HARD（直接描述））。

12.3.35 DB_IND_COLUMNS

DB_IND_COLUMNS视图显示当前用户可访问的所有索引的字段信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-125 DB_IND_COLUMNS 字段

名称	类型	描述
index_owner	character varying(64)	索引的所有者。
index_name	character varying(64)	索引名。
table_owner	character varying(64)	表的所有者。
table_name	character varying(64)	表名。
column_name	name	列名。
column_position	smallint	索引中列的位置。

12.3.36 DB_IND_EXPRESSIONS

DB_IND_EXPRESSIONS视图显示当前用户可访问的表达式索引的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-126 DB_IND_EXPRESSIONS 字段

名称	类型	描述
index_owner	character varying(64)	索引的所有者。
index_name	character varying(64)	索引名。
table_owner	character varying(64)	表的所有者。
table_name	character varying(64)	表名。
column_expression	text	定义列的基于函数的索引表达式。
column_position	smallint	索引中列的位置。

12.3.37 DB_INDEXES

DB_INDEXES视图显示当前用户可访问的索引信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-127 DB_INDEXES 字段

名称	类型	描述
owner	character varying(64)	索引的所有者。
index_name	character varying(64)	索引名。
table_name	character varying(64)	索引对应的表名。
uniqueness	text	表示这个索引是否为唯一索引。
partitioned	character(3)	表示这个索引是否具有分区表的性质。
generated	character varying(1)	表示这个索引的名称是否为系统生成。

12.3.38 DB_OBJECTS

DB_OBJECTS视图记录了当前用户可访问的数据库对象的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-128 DB_OBJECTS 字段

名称	类型	描述
owner	name	对象的所有者。
object_name	name	对象的名称。
object_id	oid	对象的OID。
object_type	name	对象的类型。
namespace	oid	对象所在的命名空间的ID。
created	timestamp with time zone	对象的创建时间
last_ddl_time	timestamp with time zone	对象的最后修改时间

须知

created和last_ddl_time支持的范围参见**PG_OBJECT**中的记录范围。

12.3.39 DB_PROCEDURES

DB_PROCEDURES视图显示当前用户可访问的所有存储过程或函数信息的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-129 DB_PROCEDURES 字段

名称	类型	描述
owner	name	对象的所有者。
object_name	name	对象的名称。

12.3.40 DB_SEQUENCES

DB_SEQUENCES视图显示当前用户能够访问的所有序列的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-130 DB_SEQUENCES 字段

名称	类型	描述
sequence_owner	name	序列所有者。

名称	类型	描述
sequence_name	name	序列的名称。
min_value	int16	序列最小值。
max_value	int16	序列最大值。
last_number	int16	上一序列的值。
cache_size	int16	序列磁盘缓存大小。
increment_by	int16	序列的增量。
cycle_flag	character(1)	表示序列是否是循环序列，取值为Y或N： ● Y表示是循环序列。 ● N表示不是循环序列。

12.3.41 DB_SOURCE

DB_SOURCE视图显示当前用户可访问的存储过程或函数信息，且提供存储过程或函数定义的字段。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-131 DB_SOURCE 字段

名称	类型	描述
owner	name	对象的所有者。
name	name	对象的名称。
type	name	对象的类型。
text	text	对象的定义。

12.3.42 DB_SYNONYMS

DB_SYNONYMS视图显示当前用户可访问的所有同义词信息。

表 12-132 DB_SYNONYMS 字段

名称	类型	描述
owner	text	同义词的所有者。
schema_name	text	同义词所属模式名。
synonym_name	text	同义词的名称。

名称	类型	描述
table_owner	text	关联对象的所有者。尽管该列称为table_owner，但它拥有的该关联对象不一定是表，可以是任何数据库通用对象，例如视图、存储过程、同义词等。
table_name	text	关联对象名。尽管该列称为table_name，但此关联对象不一定是表，可以是任何数据库通用对象，例如视图、存储过程、同义词等。
table_schema_name	text	关联对象所属模式名。尽管该列称为table_schema_name，但此schema下的该关联对象不一定是表，可以是任何数据库通用对象，例如视图、存储过程、同义词等。

12.3.43 DB_TAB_COLUMNS

DB_TAB_COLUMNS视图显示当前用户可访问的表的列的描述信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-133 DB_TAB_COLUMNS 字段

名称	类型	描述
owner	character varying(64)	表的所有者。
table_name	character varying(64)	表的名称。
column_name	character varying(64)	列的名称。
data_type	character varying(128)	列的数据类型。
data_length	integer	列的字节长度。
data_precision	integer	数据类型的精度，对于numeric数据类型有效，其他类型为NULL。
data_scale	integer	小数点右边的位数，对于numeric数据类型有效，其他类型为0。
nullable	bpchar	该列是否允许为空，对于主键约束和非空约束，该值为n。
column_id	integer	对象创建或增加列时列的序号。
char_length	numeric	列的长度（单位字符），只对varchar，nvarchar2，bpchar，char类型有效。

名称	类型	描述
avg_col_len	numeric	列的平均长度（单位字节）。
comments	text	注释。

12.3.44 DB_TAB_COMMENTS

DB_TAB_COMMENTS视图显示当前用户可访问的所有表和视图的注释信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-134 DB_TAB_COMMENTS 字段

名称	类型	描述
owner	character varying(64)	表或视图的所有者。
table_name	character varying(64)	表或视图的名称。
comments	text	注释。

12.3.45 DB_TABLES

DB_TABLES视图显示当前用户可访问的所有表的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-135 DB_TABLES 字段

名称	类型	描述
owner	character varying(64)	表的所有者。
table_name	character varying(64)	表名。
tablespace_name	character varying(64)	存储表的表空间名称。
num_rows	numeric	表的估计行数。
status	character varying(8)	当前表是否有效。 ● VALID：当前表有效。 ● UNUSABLE：当前表不可用。
temporary	character(1)	表是否为临时表： ● Y：表示是临时表。 ● N：表示不是临时表。

名称	类型	描述
dropped	character varying	当前表是否已删除： ● YES：表示已删除。 ● NO：表示未删除。

12.3.46 DB_TRIGGERS

DB_TRIGGERS视图显示当前用户能访问到的触发器的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-136 DB_TRIGGERS 字段

名称	类型	描述
trigger_name	character varying(64)	触发器名称。
table_owner	character varying(64)	角色名称。
table_name	character varying(64)	关系表名称。
status	character varying(64)	● O：触发器在“origin”和“local”模式下触发。 ● D：触发器被禁用。 ● R：触发器在“replica”模式下触发。 ● A：触发器始终触发。

12.3.47 DB_USERS

DB_USERS视图记录数据库中所有用户，但不对用户信息进行详细的描述。默认只有系统管理员可以访问。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-137 DB_USERS 字段

名称	类型	描述
user_id	oid	用户的OID。
username	name	用户的名称。

12.3.48 DB_VIEWS

DB_VIEWS视图显示当前用户可访问的所有视图的描述信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-138 DB_VIEWS 字段

名称	类型	描述
owner	name	视图的所有者。
view_name	name	视图的名称。
text	text	视图文本。
text_length	integer	视图文本长度。

12.3.49 DV_SESSION_LONGOPS

DV_SESSION_LONGOPS视图显示当前正在执行的操作的进度。该视图需要授权访问。

表 12-139 DV_SESSION_LONGOPS 字段

名称	类型	描述
sid	bigint	当前正在执行的后台进程的OID。
serial#	integer	当前正在执行的后台进程的序号，在GaussDB中为0。
sofar	integer	目前完成的工作量，在GaussDB中为空。
totalwork	integer	工作总量，在GaussDB中为空。

12.3.50 DV_SESSIONS

DV_SESSIONS视图显示当前所有活动的后台线程的信息。默认只有系统管理员权限才可以访问此系统视图，普通用户需要授权才可以访问。

表 12-140 DV_SESSIONS 字段

名称	类型	描述
sid	bigint	当前活动的后台线程的OID。
serial#	integer	当前活动的后台线程的序号，在GaussDB中为0。
user#	oid	登录此后台线程的用户的OID。oid 为0表示此后台线程为全局辅助线程（auxiliary）。

名称	类型	描述
username	name	登录此后台线程的用户名。username为空表示此后台线程为全局辅助线程（auxiliary）。 可以通过和pg_stat_get_activity()关联查询，识别出application_name: 例如： SELECT s.*,a.application_name FROM DV_SESSIONS AS s LEFT JOIN pg_stat_get_activity(NULL) AS a ON s.sid=a.sessionid;

12.3.51 GET_GLOBAL_PREPARED_XACTS

GET_GLOBAL_PREPARED_XACTS视图可用来获取全局所有节点上的两阶段残留事务信息。

表 12-141 GET_GLOBAL_PREPARED_XACTS 字段

名称	类型	描述
transaction	xid	残留事务的xid。
gid	text	残留两阶段事务全局gid。
prepared	timestamp with time zone	残留两阶段事务prepared时间。
owner	name	残留两阶段事务的owner。
database	name	残留两阶段事务所属的database。
node_name	text	残留事务所在的节点名称。

12.3.52 GLOBAL_BAD_BLOCK_INFO

GLOBAL_BAD_BLOCK_INFO视图，在CN上执行，统计所有实例数据页面损坏的情况，查询信息会显示损坏页面的基本信息。在DN上执行结果为空。可根据这些信息利用**数据损坏检测修复函数**中的页面检测修复函数进行进一步修复操作。默认只有初始用户、具有sysadmin属性的用户、在运维模式下具有运维管理员属性的用户、以及监控用户可以查看，其余用户需要赋权后才可以使用。

表 12-142 GLOBAL_BAD_BLOCK_INFO 字段

名称	类型	描述
node_name	text	当前损坏页面的节点信息。
spc_node	oid	当前损坏页面对应的表空间id。
db_node	oid	当前损坏页面对应的数据库id。
rel_node	oid	当前损坏页面对应的relation的relfilenode。

名称	类型	描述
bucket_node	integer	当前损坏页面的bucket_node，非段页式表显示-1，段页式表是非0值，该字段在修复时作为是否是段页式表的判断依据。
block_num	oid	当前损坏页面的页面号。
fork_num	integer	当前损坏页面的文件forknum。
file_path	text	当前损坏页面的相对路径，段页式表显示的是逻辑路径，非真实存在的文件。
check_time	timestamp with time zone	当前损坏页面检测出有问题的时间。
repair_time	timestamp with time zone	当前损坏页面被修复的时间。

12.3.53 GLOBAL_CLEAR_BAD_BLOCK_INFO

GLOBAL_CLEAR_BAD_BLOCK_INFO视图，在CN上执行，用于清理所有实例中已修复的损坏页面的信息，在DN上执行结果为空。默认只有初始用户、具有sysadmin属性的用户、在运维模式下具有运维管理员属性的用户、以及监控用户可以查看，其余用户需要赋权后才可以使用。

表 12-143 GLOBAL_BAD_BLOCK_INFO 字段

名称	类型	描述
node_name	text	当前清理修复页面信息结果对应的节点信息。
result	boolean	当前实例清理修复页面执行的结果。

12.3.54 GLOBAL_COMM_CLIENT_INFO

GLOBAL_COMM_CLIENT_INFO视图可用来查询整个集群全局节点活跃的客户端连接信息，默认只有系统管理员权限才可以访问此系统视图。

表 12-144 GLOBAL_COMM_CLIENT_INFO 字段

名称	类型	描述
node_name	text	当前节点的名称
app	text	连接当前DN的客户端，如coordinator(CN)、GTM、DN等。
tid	bigint	当前线程的线程号

名称	类型	描述
lwtid	integer	当前线程的轻量级线程号
query_id	bigint	查询ID，对应debug_query_id
socket	integer	如果是物理连接，显示socket
remote_ip	text	对端节点IP
remote_port	text	对端节点port
logic_id	integer	如果是逻辑连接，显示sid

12.3.55 GLOBAL_STAT_HOTKEYS_INFO

GLOBAL_STAT_HOTKEYS_INFO视图可用来查询整个集群中热点key的统计信息，查询结果按照count从大到小排序。

表 12-145 GLOBAL_STAT_HOTKEYS_INFO 字段

名称	类型	描述
database_name	text	热点key所在database名称
schema_name	text	热点key所在schema名称
table_name	text	热点key所在table名称
key_value	text	热点key的value
hash_value	bigint	热点key在数据库中的哈希值，如果是List/Range分布表，该字段为0
count	numeric	热点key被访问频次

12.3.56 GLOBAL_WAL_SENDER_STATUS

GLOBAL_WAL_SENDER_STATUS视图显示当前集群主DN实例的redo日志传输和回放状态。该视图只有monitor admin和sysadmin权限可以查看。

表 12-146 GLOBAL_WAL_SENDER_STATUS 字段

名称	类型	描述
nodename	text	主节点名。
source_ip	text	主节点IP。
source_port	integer	主节点端口。

名称	类型	描述
dest_ip	text	备节点IP。
dest_port	integer	备节点端口。
sender_pid	integer	发送线程PID。
local_role	text	主节点类型。 ● UNKNOWN_MODE：表示状态未知。 ● NORMAL_MODE：表示单主机节点类型。 ● PRIMARY_MODE：表示节点类型为主节点。 ● STANDBY_MODE：表示节点类型为备节点。 ● CASCADE_STANDBY_MODE：表示节点类型为级联备节点。 ● PENDING_MODE：表示该节点在仲裁阶段。 ● RECOVERY_MODE：表示该节点在恢复阶段。 ● STANDBY_CLUSTER_MODE：表示该节点类型为备集群节点。 ● MAIN_STANDBY_MODE：表示该节点类型为首备集群节点。 **说明** 主节点类型期望值为：NORMAL_MODE，PRIMARY_MODE，PENDING_MODE，RECOVERY_MODE。如出现其他节点类型，请联系华为技术工程师提供技术支持。

名称	类型	描述
peer_role	text	备节点类型。 ● UNKNOWN_MODE：表示状态未知。 ● NORMAL_MODE：表示单主机节点类型。 ● PRIMARY_MODE：表示节点类型为主节点。 ● STANDBY_MODE：表示节点类型为备节点。 ● CASCADE_STANDBY_MODE：表示节点类型为级联备节点。 ● PENDING_MODE：表示该节点在仲裁阶段。 ● RECOVERY_MODE：表示该节点在恢复阶段。 ● STANDBY_CLUSTER_MODE：表示该节点类型为备集群节点。 ● MAIN_STANDBY_MODE：表示该节点类型为首备集群节点。 **说明** 备节点类型期望值为：NORMAL_MODE，STANDBY_MODE，CASCADE_STANDBY_MODE，PENDING_MODE，RECOVERY_MODE，STANDBY_CLUSTER_MODE，MAIN_STANDBY_MODE。如出现其他节点类型，请联系华为技术工程师提供技术支持。
peer_state	text	备节点状态。 ● UNKNOWN_STATE：节点状态未知。 ● NORMAL_STATE：表示节点启动正常。 ● NEEDREPAIR_STATE：当前节点需要修复。 ● STARTING_STATE：节点正在启动中。 ● WAITING_STATE：节点正等待升级中。 ● DEMOTING_STATE：节点正在降级中。 ● PROMOTING_STATE：备节点正在升级为主节点的状态 ● BUILDING_STATE：备机启动失败, 需要重建。 ● CATCHUP_STATE：备节点正在追赶主节点。 ● COREDUMP_STATE：节点崩溃。

名称	类型	描述
state	text	wal sender状态。 ● WALSNDSTATE_STARTUP：启动状态。 ● WALSNDSTATE_BACKUP：备份状态。 ● WALSNDSTATE_CATCHUP：追赶状态。 ● WALSNDSTATE_STREAMING：流复制状态。
sender_sent_location	text	主节点发送位置。
sender_write_location	text	主节点落盘位置。
sender_flush_location	text	主节点flush磁盘位置。
sender_replay_location	text	主节点redo位置。
receiver_received_location	text	备节点接收位置。
receiver_write_location	text	备节点落盘位置。
receiver_flush_location	text	备节点flush磁盘位置。
receiver_replay_location	text	备节点redo位置。

12.3.57 GS_ALL_CONTROL_GROUP_INFO

GS_ALL_CONTROL_GROUP_INFO视图显示数据库内所有控制组的信息。

表 12-147 GS_ALL_CONTROL_GROUP_INFO 字段

名称	类型	描述
name	text	控制组的名称。

名称	类型	描述
type	text	控制组的类型。 ● GROUP_NONE，无分组。 ● GROUP_TOP，顶级分组。 ● GROUP_CLASS，该资源的类分组，不控制任何线程。 ● GROUP_BAKWD，后端线程控制组。 ● GROUP_DEFWD， 默认控制组，仅控制该级别的查询线程。 ● GROUP_TSWD 每个用户的分时控制组，控制最底层的查询线程。
gid	bigint	控制组ID。
classgid	bigint	Workload所属Class的控制组ID。
class	text	Class控制组。
workload	text	Workload控制组。
shares	bigint	控制组分配的CPU资源配额。
limits	bigint	控制组分配的CPU资源限额。
wdlevel	bigint	Workload控制组层级。
cpucores	text	控制组使用的CPU核的信息。

12.3.58 GS_AUDITING

GS_AUDITING视图显示对数据库相关操作的所有审计信息。需要有系统管理员或安全策略管理员权限才可以访问此视图。

表 12-148 GS_AUDITING 字段

名称	类型	描述
polname	name	策略名称，需要唯一，不可重复。
pol_type	text	审计策略类型，值为‘access’或者‘privilege’。 ● access：表示审计DML操作。 ● privilege：表示审计DDL操作。
polenabled	boolean	用来表示策略启动开关。 ● t（true）：表示启动。 ● f（false）：表示不启动。
access_type	name	DML数据库操作相关类型。例如SELECT、INSERT、DELETE等。

名称	类型	描述
label_name	name	资源标签名称。对应系统表gs_auditing_policy中的polname字段。
priv_object	text	用来描述数据库资产的路径。
filter_name	text	过滤条件的逻辑字符串。

12.3.59 GS_AUDITING_ACCESS

GS_AUDITING_ACCESS视图显示对数据库DML相关操作的所有审计信息。需要有系统管理员或安全策略管理员权限才可以访问此视图。

表 12-149 GS_AUDITING_ACCESS 字段

名称	类型	描述
polname	name	策略名称，需要唯一，不可重复。
pol_type	text	审计策略类型，值为‘access’，表示审计DML操作。
polenabled	boolean	用来表示策略启动开关。 ● t（true）：表示启动。 ● f（false）：表示不启动。
access_type	name	DML数据库操作相关类型。例如SELECT、INSERT、DELETE等。
label_name	name	资源标签名称。对应系统表gs_auditing_policy中的polname字段。
access_object	text	用来描述数据库资产的路径。
filter_name	text	过滤条件的逻辑字符串。

12.3.60 GS_AUDITING_PRIVILEGE

GS_AUDITING_PRIVILEGE视图显示对数据库DDL相关操作的所有审计信息。需要有系统管理员或安全策略管理员权限才可以访问此视图。

表 12-150 GS_AUDITING_PRIVILEGE 字段

名称	类型	描述
polname	name	策略名称，需要唯一，不可重复。
pol_type	text	审计策略类型，值为‘privilege’，表示审计DDL操作。

名称	类型	描述
polenabled	boolean	用来表示策略启动开关。 ● t（true）：表示启动。 ● f（false）：表示不启动。
access_type	name	DDL数据库操作相关类型。例如CREATE、ALTER、DROP等。
label_name	name	资源标签名称。对应系统表gs_auditing_policy中的polname字段。
priv_object	text	带有数据库对象的全称域名。
filter_name	text	过滤条件的逻辑字符串。

12.3.61 GS_CLUSTER_RESOURCE_INFO

GS_CLUSTER_RESOURCE_INFO视图显示的是所有DN资源的汇总信息。该视图需要设置enable_dynamic_workload=on才能查询，并且该视图不支持在DN执行。查询该视图需要sysadmin权限。

表 12-151 GS_CLUSTER_RESOURCE_INFO 字段

名称	类型	描述
min_mem_util	integer	DN最小内存使用率。
max_mem_util	integer	DN最大内存使用率。
min_cpu_util	integer	DN最小CPU使用率。
max_cpu_util	integer	DN最大CPU使用率。
min_io_util	integer	DN最小I/O使用率。
max_io_util	integer	DN最大I/O使用率。
used_mem_rate	integer	物理节点最大内存使用率。

12.3.62 GS_DB_PRIVILEGES

GS_DB_PRIVILEGES系统视图记录ANY权限的授予情况，每条记录对应一条授权信息。

表 12-152 GS_DB_PRIVILEGES 字段

名称	类型	描述
rolename	name	用户名。

名称	类型	描述
privilege_type	text	用户拥有的ANY权限，取值参考**表7-114**。
admin_option	text	是否具有privilege_type列记录的ANY权限的再授权权限。 ● yes：表示具有。 ● no：表示不具有。

12.3.63 GS_GET_CONTROL_GROUP_INFO

GS_GET_CONTROL_GROUP_INFO视图显示所有控制组的信息。查询该视图需要sysadmin权限。

表 12-153 GS_GET_CONTROL_GROUP_INFO 字段

名称	类型	描述
group_name	text	控制组的名称。
group_type	text	控制组的类型。 ● GROUP_NONE，无分组。 ● GROUP_TOP，顶级分组。 ● GROUP_CLASS，该资源的类分组，不控制任何线程。 ● GROUP_BAKWD，后端线程控制组。 ● GROUP_DEFWD，默认控制组，仅控制该级别的查询线程。 ● GROUP_TSWD每个用户的分时控制组，控制最底层的查询线程。
gid	bigint	控制组ID。
classgid	bigint	Workload所属Class的控制组ID。
class	text	Class控制组。
group_workload	text	Workload控制组。
shares	bigint	控制组分配的CPU资源配额。
limits	bigint	控制组分配的CPU资源限额。
wdlevel	bigint	Workload控制组层级。

名称	类型	描述
cpucores	text	控制组使用的CPU核的信息。
nodegroup	text	node group名称。
group_kind	text	node group类型，取值包括i，n，v，e。 • i：表示installation node group。 • n：表示普通集群node group。 • e：表示弹性集群。

12.3.64 GS_GSC_MEMORY_DETAIL

GS_GSC_MEMORY_DETAIL视图显示当前节点当前进程的全局SysCache的内存占用情况。仅在开启GSC的模式下有数据。

需要注意的是，由于这个查询是以数据库内存上下文分隔的，因此会缺少一部分内存的统计，缺失的内存统计对应的内存上下文名称为GlobalSysDBCache。

表 12-154 GS_GSC_MEMORY_DETAIL 字段

名称	类型	描述
db_id	text	数据库id。
totalsize	numeric	共享内存总大小，单位Byte。
freesize	numeric	共享内存剩余大小，单位Byte。
usedsize	numeric	共享内存使用大小，单位Byte。

12.3.65 GS_LABELS

GS_LABELS视图显示所有已配置的资源标签信息。需要有系统管理员或安全策略管理员权限才可以访问此视图。

表 12-155 GS_LABELS 字段

名称	类型	描述
labelname	name	资源标签的名称。
labeltype	name	资源标签的类型。对应系统表GS_POLICY_LABEL中的labeltype字段。
fqdntype	name	数据库资源的类型。如table、schema、index等。

名称	类型	描述
schemaname	name	数据库资源所属的schema名称。
fqdnname	name	数据库资源名称。
columnname	name	数据库资源列名称，若标记的数据库资源不为表的列则该项为空。

12.3.66 GS_LSC_MEMORY_DETAIL

GS_LSC_MEMORY_DETAIL视图显示所有线程的本地SysCache的内存使用情况，以MemoryContext节点来统计。仅在开启GSC的模式下有数据。

表 12-156 GS_LSC_MEMORY_DETAIL 字段

名称	类型	描述
threadid	text	线程启动时间+线程标识（字符串信息为timestamp.sessionid）。
tid	bigint	线程标识。
thrdtype	text	线程类型。可以是系统内存在的任何线程类型，如postgresql、wlmmonitor等。
contextname	text	内存上下文名称。
level	smallint	当前上下文在整体内存上下文中的层级。
parent	text	父内存上下文名称。
totalsize	bigint	当前内存上下文的内存总数，单位Byte。
freesize	bigint	当前内存上下文中已释放的内存总数，单位Byte。
usedsize	bigint	当前内存上下文中已使用的内存总数，单位Byte。

12.3.67 GS_MASKING

GS_MASKING视图显示所有已配置的动态脱敏策略信息。需要有系统管理员或安全策略管理员权限才可以访问此视图。

表 12-157 GS_MASKING 字段

名称	类型	描述
polname	name	脱敏策略名称。
polenabled	boolean	脱敏策略开关。

名称	类型	描述
maskaction	name	脱敏函数。
labelname	name	脱敏函数作用的标签名称。
masking_object	text	脱敏数据库资源对象。
filter_name	text	过滤条件的逻辑表达式。

12.3.68 GS_MATVIEWS

GS_MATVIEWS视图显示数据库中每一个物化视图的信息。

表 12-158 GS_MATVIEWS 字段

名称	类型	引用	描述
schemaname	name	PG_NAMESPACE.nspname	物化视图的模式名。
matviewname	name	PG_CLASS.relname	物化视图名。
matviewowner	name	PG_AUTHID.rolname	物化视图的所有者。
tablespace	name	PG_TABLESPACE.spcname	物化视图的表空间名（如果使用数据库默认表空间则为空）
hasindexes	boolean	-	如果物化视图有（或者最近有过）任何索引，则此列为真。
definition	text	-	物化视图的定义（一个重构的SELECT查询）。

12.3.69 GS_SQL_COUNT

GS_SQL_COUNT视图显示数据库当前节点当前时刻执行的五类语句（SELECT、INSERT、UPDATE、DELETE、MERGE INTO）统计信息。

- 普通用户查询GS_SQL_COUNT视图仅能看到该用户当前节点的统计信息；管理员权限用户查询GS_SQL_COUNT视图则能看到所有用户当前节点的统计信息。
- 当集群或该节点重启时，计数将清零，并重新开始计数 。
- 计数以节点收到的查询数为准，包括集群内部进行的查询。例如，CN收到一条查询，若下发多条查询DN，那将在DN上进行相应次数的计数。

表 12-159 GS_SQL_COUNT 字段

名称	类型	描述
node_name	name	节点名称。
user_name	name	用户名。
select_count	bigint	SELECT语句统计结果。
update_count	bigint	UPDATE语句统计结果。
insert_count	bigint	INSERT语句统计结果。
delete_count	bigint	DELETE语句统计结果。
mergeinto_count	bigint	MERGE INTO语句统计结果。
ddl_count	bigint	DDL语句的数量。
dml_count	bigint	DML语句的数量。
dcl_count	bigint	DCL语句的数量。
total_select_elapse	bigint	总SELECT的时间花费（单位：微秒）。
avg_select_elapse	bigint	平均SELECT的时间花费（单位：微秒）。
max_select_elapse	bigint	最大SELECT的时间花费（单位：微秒）。
min_select_elapse	bigint	最小SELECT的时间花费（单位：微秒）。
total_update_elapse	bigint	总UPDATE的时间花费（单位：微秒）。
avg_update_elapse	bigint	平均UPDATE的时间花费（单位：微秒）。
max_update_elapse	bigint	最大UPDATE的时间花费（单位：微秒）。
min_update_elapse	bigint	最小UPDATE的时间花费（单位：微秒）。
total_insert_elapse	bigint	总INSERT的时间花费（单位：微秒）。
avg_insert_elapse	bigint	平均INSERT的时间花费（单位：微秒）。
max_insert_elapse	bigint	最大INSERT的时间花费（单位：微秒）。
min_insert_elapse	bigint	最小INSERT的时间花费（单位：微秒）。
total_delete_elapse	bigint	总DELETE的时间花费（单位：微秒）。

名称	类型	描述
avg_delete_elapse	bigint	平均DELETE的时间花费（单位：微秒）。
max_delete_elapse	bigint	最大DELETE的时间花费（单位：微秒）。
min_delete_elapse	bigint	最小DELETE的时间花费（单位：微秒）。

12.3.70 GS_STAT_DB_CU

GS_STAT_DB_CU视图可用来查询集群各个节点中每个数据库的CU命中情况。可以通过gs_stat_reset()进行清零。该视图只有monitor admin和sysadmin权限可以查看。

表 12-160 GS_STAT_DB_CU 字段

名称	类型	描述
node_name1	text	节点名称。
db_name	text	数据库名称。
mem_hit	bigint	内存命中次数。
hdd_sync_read	bigint	硬盘同步读次数。
hdd_asyn_read	bigint	硬盘异步读次数。

12.3.71 GS_STAT_SESSION_CU

GS_STAT_SESSION_CU视图可用来查询整个集群各个节点中当前运行session的CU命中情况。session退出相应的统计数据会清零。集群重启后，统计数据也会清零。该视图只有monitor admin和sysadmin权限可以查看。

表 12-161 GS_STAT_SESSION_CU 字段

名称	类型	描述
node_name1	text	节点名称。
mem_hit	integer	内存命中次数。
hdd_sync_read	integer	硬盘同步读次数。
hdd_asyn_read	integer	硬盘异步读次数。

12.3.72 GS_TOTAL_NODEGROUP_MEMORY_DETAIL

GS_TOTAL_NODEGROUP_MEMORY_DETAIL显示当前数据库Node group使用内存的信息，单位为MB，若GUC参数enable_memory_limit设置为off，则该视图不能使用。

表 12-162 GS_TOTAL_NODEGROUP_MEMORY_DETAIL 字段

名称	类型	描述
ngname	text	Node group名称。
memorytype	text	内存类型，包括以下几种： ● ng_total_memory：该Node group的总内存大小。 ● ng_used_memory：该Node group的实际使用内存大小。 ● ng_estimate_memory：该Node group的估算使用内存大小。 ● ng_foreignrp_memsize：该Node group的外部资源池的总内存大小。 ● ng_foreignrp_usedsize：该Node group的外部资源池实际使用内存大小。 ● ng_foreignrp_peaksize：该Node group的外部资源池使用内存的峰值。 ● ng_foreignrp_mempct：该Node group的外部资源池占该Node group总内存大小的百分比。 ● ng_foreignrp_estmsize：该Node group的外部资源池估算使用内存大小。
memorymbytes	integer	内存类型分配内存的大小。

12.3.73 GV_SESSION

GV_SESSION视图描述和当前用户查询相关的信息，字段保存的是上一次执行的信息。

表 12-163 GV_SESSION 字段

名称	类型	描述
sid	bigint	会话ID。
serial#	integer	当前活动的后台线程的序号，在GaussDB中为0。
schemaname	name	登录该后台的用户名。
user#	oid	登录此后台线程的用户的OID。oid 为0表示此后台线程为全局辅助线程(auxiliary)。

名称	类型	描述
username	name	登录此后台线程的用户名。username为空表示此后台线程为全局辅助线程(auxiliary)。
machine	text	客户端的主机名，这个字段是通过client_addr的反向DNS查找得到。这个字段只有在启动log_hostname且使用IP连接时才非空。
sql_id	bigint	查询语句的ID。
client_info	text	客户端的主机名，这个字段是通过client_addr的反向DNS查找得到。这个字段只有在启动log_hostname且使用IP连接时才非空。
event	text	语句当前排队状态。可能值是： ● waiting in queue：表示语句在排队中。 ● 空：表示语句正在运行。
sql_exec_start	timestamp with time zone	开始当前活跃查询的时间， 如果state的值不是active，则这个值是上一个查询的开始时间。
program	text	连接到该后台的应用名。
status	text	该后台当前总体状态。可能值是： ● active：后台正在执行一个查询。 ● idle：后台正在等待一个新的客户端命令。 ● idle in transaction：后台在事务中，但事务中没有语句在执行。 ● idle in transaction (aborted)：后台在事务中，但事务中有语句执行失败。 ● fastpath function call：后台正在执行一个fast-path函数。 ● disabled：如果后台禁用track_activities，则报告这个状态。

12.3.74 MPP_TABLES

MPP_TABLES视图显示PGXC_CLASS中表的信息。

表 12-164 MPP_TABLES 字段

名称	类型	描述
schemaname	name	表的模式名。
tablename	name	表名。
tableowner	name	表的所有者。

名称	类型	描述
tablespace	name	表所在的表空间。
pgroup	name	节点群的名称。
nodeoids	oidvector_extend	表分布的节点OID列表。

12.3.75 MY_COL_COMMENTS

MY_COL_COMMENTS视图显示当前用户下表的列注释信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-165 MY_COL_COMMENTS 字段

名称	类型	描述
owner	character varying(64)	表的所有者。
table_name	character varying(64)	表的名称。
column_name	character varying(64)	列名称。
comments	text	注释。

12.3.76 MY_CONS_COLUMNS

MY_CONS_COLUMNS视图显示当前用户下表中主键约束列的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-166 MY_CONS_COLUMNS 字段

名称	类型	描述
owner	character varying(64)	约束创建者。
table_name	character varying(64)	约束相关的表名。
column_name	character varying(64)	约束相关的列名。
constraint_name	character varying(64)	约束名。
position	smallint	表中列的位置。

12.3.77 MY_CONSTRAINTS

MY_CONSTRAINTS视图显示当前用户下表中约束的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-167 MY_CONSTRAINTS 字段

名称	类型	描述
owner	character varyiny(64)	约束创建者。
constraint_name	vcharacter varying(64)	约束名。
constraint_type	text	约束类型： • c表示检查约束。 • f表示外键约束。 • p表示主键约束。 • u表示唯一约束。
table_name	character varying(64)	约束相关的表名。
index_owner	character varying(64)	约束相关的索引的所有者（只针对唯一约束和主键约束）。
index_name	character varying(64)	约束相关的索引的名称（只针对唯一约束和主键约束）。

12.3.78 MY_IND_COLUMNS

MY_IND_COLUMNS视图显示当前用户下所有索引的字段信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-168 MY_IND_COLUMNS 字段

名称	类型	描述
index_owner	character varying(64)	索引的所有者。
index_name	character varying(64)	索引名。
table_owner	character varying(64)	表的所有者。
table_name	character varying(64)	表名。
column_name	name	列名。
column_position	smallint	索引中列的位置。

12.3.79 MY_IND_EXPRESSIONS

MY_IND_EXPRESSIONS视图显示当前用户下基于函数的表达式索引的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-169 MY_IND_EXPRESSIONS 字段

名称	类型	描述
table_owner	character varying(64)	表的所有者。
table_name	character varying(64)	表名。
index_owner	character varying(64)	索引的所有者。
index_name	character varying(64)	索引名。
column_expression	text	定义列的基于函数的索引表达式。
column_position	smallint	索引中列的位置。

12.3.80 MY_IND_PARTITIONS

MY_IND_PARTITIONS视图显示当前用户下的索引分区信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-170 MY_IND_PARTITIONS 字段

名称	类型	描述
index_owner	character varying(64)	索引分区所属分区表索引的所有者的名称。
index_name	character varying(64)	索引分区所属分区表索引的名称。
partition_name	character varying(64)	索引分区的名称。
def_tablespace_name	name	索引分区的表空间名称。
high_value	text	索引分区所对应分区的上边界。
index_partition_usable	boolean	索引分区是否可用。 • t（true）：表示是。 • f（false）：表示否。
schema	character varying(64)	索引分区所属分区表索引的模式。

12.3.81 MY_INDEXES

MY_INDEXES视图显示当前用户的索引信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-171 MY_INDEXES 字段

名称	类型	描述
owner	character varying(64)	索引的所有者。
index_name	character varying(64)	索引名称。
table_name	character varying(64)	索引对应的表名。
uniqueness	text	表示这个索引是否为唯一索引。
partitioned	character(3)	表示这个索引是否具有分区表的性质。
generated	character varying(1)	表示这个索引的名称是否为系统生成。

12.3.82 MY_JOBS

MY_JOBS视图显示当前用户拥有的定时任务的详细信息。该视图同时存在于PG_CATALOG和SYS schema下。

表 12-172 MY_JOBS 字段

名称	类型	描述
job	bigint	作业ID。
log_user	name	创建者的UserName。
priv_user	name	作业执行者的UserName。
dbname	name	创建作业的数据库名称。
start_date	timestamp without time zone	作业的开始时间。
start_suc	text	作业成功执行的开始时间。
last_date	timestamp without time zone	上次运行开始时间。
last_suc	text	上次成功运行的开始时间。
this_date	timestamp without time zone	正在运行任务的开始时间。
this_suc	text	正在运行任务成功的开始时间。
next_date	timestamp without time zone	任务下次执行时间。
next_suc	text	任务下次成功执行时间。

名称	类型	描述
broken	text	如果status字段取值为'd'，则为'y'，否则为'n'。
status	"char"	本步骤的执行状态，取值范围：（'r'、's'、'f'、'd'），默认为'r'，取值含义： ● r：运行中。 ● s：执行成功。 ● f：执行失败。 ● d：取消执行。
interval	text	用来计算下次运行时间的时间表达式，如果为null则表示定时任务只执行一次。
failures	smallint	失败计数，若作业连续执行失败16次，则不再继续执行。
what	text	可执行的作业。

12.3.83 MY_OBJECTS

MY_OBJECTS视图描述了当前用户拥有的数据库对象。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-173 MY_OBJECTS 字段

名称	类型	描述
object_name	name	对象的名称。
object_id	oid	对象的OID。
object_type	name	对象的类型，包括TABLE、INDEX、SEQUENCE、VIEW。
namespace	oid	对象所属的名称空间。
created	timestamp with time zone	对象的创建时间
last_ddl_time	timestamp with time zone	对象的最后修改时间

须知

created和last_ddl_time支持的范围参见PG_OBJECT中的记录范围。

12.3.84 MY_PART_INDEXES

MY_PART_INDEXES视图显示当前用户下分区表索引的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-174 MY_PART_INDEXES 字段

名称	类型	描述
def_tablespace_name	name	分区表索引的表空间名称。
index_owner	character varying(64)	分区表索引的所有者名称。
index_name	character varying(64)	分区表索引的名称。
partition_count	bigint	分区表索引的索引分区的个数。
partitioning_key_count	integer	分区表的分区键个数。
partitioning_type	text	分区表的分区策略。 **说明** 当前分区表策略仅支持范围分区（Range Partitioning）。
schema	character varying(64)	分区表索引的模式。
table_name	character varying(64)	分区表索引所属的分区表名称。

12.3.85 MY_PART_TABLES

MY_PART_TABLES视图显示当前用户下分区表的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-175 MY_PART_TABLES 字段

名称	类型	描述
table_owner	character varying(64)	分区表的所有者名称。
table_name	character varying(64)	分区表的名称。
partitioning_type	text	分区表的分区策略。 **说明** 当前分区表策略仅支持范围分区（Range Partitioning）。
partition_count	bigint	分区表的分区个数。

名称	类型	描述
partitioning_key_count	integer	分区表的分区键个数。
def_tablespace_name	name	分区表的表空间名称。
schema	character varying(64)	分区表的模式。

12.3.86 MY_PROCEDURES

MY_PROCEDURES视图描述了当前用户拥有的存储过程或函数的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-176 MY_PROCEDURES 字段

名称	类型	描述
owner	character varying(64)	存储过程或函数的所有者。
object_name	character varying(64)	存储过程或函数名称。
argument_number	smallint	存储过程入参个数。

12.3.87 MY_SEQUENCES

MY_SEQUENCES视图显示当前用户的序列信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-177 MY_SEQUENCES 字段

名称	类型	描述
sequence_owner	name	序列所有者。
sequence_name	name	序列的名称。
min_value	int16	序列最小值。
max_value	int16	序列最大值。
increment_by	int16	序列的增量。
cycle_flag	character(1)	表示序列是否是循环序列，取值： ● Y表示是循环序列。 ● N表示不是循环序列。
last_number	int16	上一序列的值。
cache_size	int16	序列磁盘缓存大小。

12.3.88 MY_SOURCE

MY_SOURCE视图显示当前用户拥有的存储过程或函数的信息，且提供存储过程或函数定义的字段。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-178 MY_SOURCE 字段

名称	类型	描述
owner	character varying(64)	存储过程或函数的所有者。
name	character varying(64)	存储过程或函数名称。
text	text	存储过程或函数的定义。

12.3.89 MY_SYNONYMS

MY_SYNONYMS视图显示当前用户可访问的同义词的信息。

表 12-179 MY_SYNONYMS 字段

名称	类型	描述
schema_name	text	同义词所属模式名。
synonym_name	text	同义词的名称。
table_owner	text	关联对象的所有者。尽管该列称为table_owner，但它拥有的该关联对象不一定是表，可以是任何数据库通用对象，例如视图、存储过程、同义词等。
table_name	text	关联对象名。尽管该列称为table_name，但此关联对象不一定是表，可以是任何数据库通用对象，例如视图、存储过程、同义词等。
table_schema_name	text	关联对象所属模式名。尽管该列称为table_schema_name，但此schema下的该关联对象不一定是表，可以是任何数据库通用对象，例如视图、存储过程、同义词等。

12.3.90 MY_TAB_COLUMNS

MY_TAB_COLUMNS视图显示当前用户可访问的表的字段信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-180 MY_TAB_COLUMNS 字段

名称	类型	描述
owner	character varying(64)	表的所有者。
table_name	character varying(64)	表名称。
column_name	character varying(64)	列名。
data_type	character varying(128)	列的数据类型。
data_length	integer	列的字节长度。
data_precision	integer	数据类型的精度，对于numeric数据类型有效，其他类型为NULL。
data_scale	integer	小数点右边的位数，对于numeric数据类型有效，其他类型为0。
nullable	bpchar	该列是否允许为空，对于主键约束和非空约束，该值为n。
column_id	integer	创建表时列的序号。
avg_col_len	numeric	列的平均长度（单位字节）。
char_length	numeric	列的长度（单位字符），只对varchar，nvarchar2，bpchar，char类型有效。
comments	text	注释。

12.3.91 MY_TAB_COMMENTS

MY_TAB_COMMENTS视图显示当前用户拥有的表和视图的注释信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-181 MY_TAB_COMMENTS 字段

名称	类型	描述
owner	character varying(64)	表或视图的所有者。
table_name	character varying(64)	表或视图的名称。
comments	text	注释。

12.3.92 MY_TAB_PARTITIONS

MY_TAB_PARTITIONS视图显示当前用户下所有分区的信息。当前用户下每个分区表的每个分区都会在USER_TAB_PARTITIONS中有一条记录。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-182 MY_TAB_PARTITIONS 字段

名称	类型	描述
table_owner	character varying(64)	分区表的所有者名称。
table_name	character varying(64)	分区表的名称。
partition_name	character varying(64)	分区的名称。
high_value	text	分区的上边界。
tablespace_name	name	分区的表空间名称。
schema	character varying(64)	分区表的模式。

12.3.93 MY_TABLES

MY_TABLES视图显示当前用户拥有的表的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-183 MY_TABLES 字段

名称	类型	描述
owner	character varying(64)	表的所有者。
table_name	character varying(64)	表名称。
tablespace_name	character varying(64)	存储表的表空间名称。
dropped	character varying	当前记录是否已删除： ● yes表示已删除。 ● no表示未删除。
num_rows	numeric	表的估计行数。
status	character varying(8)	当前记录是否有效： ● valid表示有效。
temporary	character(1)	是否为临时表。 ● y表示是临时表。 ● n表示不是临时表。

12.3.94 MY_TRIGGERS

MY_TRIGGERS视图显示当前用户拥有的触发器的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-184 MY_TRIGGERS 字段

名称	类型	描述
trigger_name	character varying(64)	触发器名称。
table_name	character varying(64)	关系表名称。
table_owner	character varying(64)	角色名称。
status	character varying(64)	触发器的状态。 ● O：触发器在“origin”和“local”模式下触发。 ● D：触发器被禁用。 ● R：触发器在“replica”模式下触发。 ● A：触发器始终触发。

12.3.95 MY_VIEWS

MY_VIEWS视图显示当前用户拥有的所有视图的信息。该视图同时存在于PG_CATALOG和SYS Schema下。

表 12-185 MY_VIEWS 字段

名称	类型	描述
owner	character varying(64)	视图的所有者。
view_name	character varying(64)	视图名称。

12.3.96 PG_COMM_DELAY

PG_COMM_DELAY视图展示单个DN的通信库时延状态。

表 12-186 PG_COMM_DELAY 字段

名称	类型	描述
node_name	text	节点名称。

名称	类型	描述
remote_name	text	连接对端节点名称。
remote_host	text	连接对端IP地址。
stream_num	integer	当前物理连接使用的stream逻辑连接数量。
min_delay	integer	当前物理连接一分钟内探测到的最小时延，单位微秒。 **说明** 负数结果无效，请重新等待时延状态更新后再执行。
average	integer	当前物理连接一分钟内探测时延的平均值，单位微秒。
max_delay	integer	当前物理连接一分钟内探测到的最大时延，单位微秒。

12.3.97 PG_COMM_RECV_STREAM

PG_COMM_RECV_STREAM视图展示单个DN上所有的通信库接收流状态。

表 12-187 PG_COMM_RECV_STREAM 字段

名称	类型	描述
node_name	text	节点名称。
local_tid	bigint	使用此通信流的线程ID。
remote_name	text	连接对端节点名称。
remote_tid	bigint	连接对端线程ID。
idx	integer	通信对端DN在本DN内的标识编号。
sid	integer	通信流在物理连接中的标识编号。
tcp_sock	integer	通信流所使用的tcp通信socket。
state	text	通信流当前的状态。 ● UNKNOWN：表示当前逻辑连接状态未知。 ● READY：表示逻辑连接已就绪。 ● RUN：表示逻辑连接发送报文正常。 ● HOLD：表示逻辑连接发送报文等待中。 ● CLOSED：表示关闭逻辑连接。 ● TO_CLOSED：表示将会关闭逻辑连接。
query_id	bigint	通信流对应的debug_query_id编号。

名称	类型	描述
pn_id	integer	通信流所执行查询的plan_node_id编号。
send_smp	integer	通信流所执行查询send端的smpid编号。
recv_smp	integer	通信流所执行查询recv端的smpid编号。
recv_bytes	bigint	通信流接收的数据总量，单位Byte。
time	bigint	通信流当前生命周期使用时长，单位ms。
speed	bigint	通信流的平均接收速率，单位Byte/s。
quota	bigint	通信流当前的通信配额值，单位Byte。
buff_usize	bigint	通信流当前缓存的数据大小，单位Byte。

12.3.98 PG_COMM_SEND_STREAM

PG_COMM_SEND_STREAM视图展示单个DN上所有的通信库发送流状态。

表 12-188 PG_COMM_SEND_STREAM 字段

名称	类型	描述
node_name	text	节点名称。
local_tid	bigint	使用此通信流的线程ID。
remote_name	text	连接对端节点名称。
remote_tid	bigint	连接对端线程ID。
idx	integer	通信对端DN在本DN内的标识编号。
sid	integer	通信流在物理连接中的标识编号。
tcp_sock	integer	通信流所使用的tcp通信socket。
state	text	通信流当前的状态。 ● UNKNOWN：表示当前逻辑连接状态未知。 ● READY：表示逻辑连接已就绪。 ● RUN：表示逻辑连接发送报文正常。 ● HOLD：表示逻辑连接发送报文等待中。 ● CLOSED：表示关闭逻辑连接。 ● TO_CLOSED：表示将会关闭逻辑连接。
query_id	bigint	通信流对应的debug_query_id编号。
pn_id	integer	通信流所执行查询的plan_node_id编号。

名称	类型	描述
send_smp	integer	通信流所执行查询send端的smpid编号。
recv_smp	integer	通信流所执行查询recv端的smpid编号。
send_bytes	bigint	通信流发送的数据总量，单位Byte。
time	bigint	通信流当前生命周期使用时长，单位ms。
speed	bigint	通信流的平均发送速率，单位Byte/s。
quota	bigint	通信流当前的通信配额值，单位Byte。
wait_quota	bigint	通信流等待quota值产生的额外时间开销，单位ms。

12.3.99 PG_COMM_STATUS

PG_COMM_STATUS视图展示单个DN的通信库状态。

表 12-189 PG_COMM_STATUS 字段

名称	类型	描述
node_name	text	节点名称。
rxpck_rate	integer	节点通信库接收速率，单位Byte/s。
txpck_rate	integer	节点通信库发送速率，单位Byte/s。
rxkbyte_rate	bigint	bigint节点通信库接收速率，单位KByte/s。
txkbyte_rate	bigint	bigint节点通信库发送速率，单位KByte/s。
buffer	bigint	cmailbox的buffer大小。
memkbyte_libcomm	bigint	libcomm进程通信内存大小，单位Byte。
memkbyte_libpq	bigint	libpq进程通信内存大小，单位Byte。
used_pm	integer	postmaster线程实时使用率。
used_sflow	integer	gs_sender_flow_controller线程实时使用率。
used_rflow	integer	gs_receiver_flow_controller线程实时使用率。
used_rloop	integer	多个gs_receivers_loop线程中高的实时使用率。
stream	integer	当前使用的逻辑连接总数。

12.3.100 PG_CONTROL_GROUP_CONFIG

PG_CONTROL_GROUP_CONFIG视图显示系统的控制组配置信息。查询该视图需要sysadmin权限。

表 12-190 PG_CONTROL_GROUP_CONFIG 字段

名称	类型	描述
pg_control_group_config	text	控制组的配置信息。

12.3.101 PG_CURSORS

PG_CURSORS视图列出了当前可用的游标。

表 12-191 PG_CURSORS 字段

名称	类型	描述
name	text	游标名。
statement	text	声明该游标时的查询语句。
is_holdable	boolean	如果该游标是持久的（就是在声明该游标的事务结束后仍然可以访问该游标）则为TRUE，否则为FALSE。
is_binary	boolean	如果该游标被声明为BINARY则为TRUE，否则为FALSE。
is_scrollable	boolean	如果该游标可以滚动（就是允许以不连续的方式检索）则为TRUE，否则为FALSE。
creation_time	timestamp with time zone	声明该游标的时间戳。

12.3.102 PG_EXT_STATS

PG_EXT_STATS视图可用来访问存储在**PG_STATISTIC_EXT**表里面的扩展统计信息。

表 12-192 PG_EXT_STATS 字段

名称	类型	引用	描述
schemaname	name	**PG_NAMESPACE**.nspname	表的模式名。
tablename	name	**PG_CLASS**.relname	表名。

名称	类型	引用	描述
attname	int2vector	PG_STATISTIC_EXT.stakey	统计信息扩展的多列信息。
inherited	boolean	-	暂不支持继承表，该字段为false。
null_frac	real	-	记录中字段组合为空的百分比。
avg_width	integer	-	字段组合记录以字节记的平均宽度。
n_distinct	real	-	• 如果大于零，表示字段组合中独立数值的估计数目。 • 如果小于零，表示独立数值的数目除以行数后乘-1得到的负数。比如，-1表示一个字段组合中独立数值的个数和行数相同。 1. 用负数形式是因为ANALYZE认为独立数值的数目是随着表增长而增长； 2. 正数的形式用于在字段看上去好像有固定的可能值数目的情况下。 • 如果等于零，表示独立数值的数目未知。
n_dndistinct	real	-	标识dn1上字段组合中非NULL的独立数值的数目。 • 如果大于零，表示独立数值的实际数目。 • 如果小于零，表示独立数值的数目除以行数后乘-1得到的负数。比如，一个字段组合的数值平均出现概率为两次，则可以表示为n_dndistinct=-0.5。 • 如果等于零，表示独立数值的数目未知。
most_common_vals	anyarray	-	一个字段组合里最常用数值的列表。如果该字段组合不存在最常用数值，则为NULL。本列保存的多列常用数值均不为NULL。
most_common_freqs	real[]	-	一个记录字段组合里最常用数值的出现频率的列表，频率由每个数值出现的次数除以行数得到。如果most_common_vals取值为NULL，则该字段取值也为NULL。

名称	类型	引用	描述
most_common_vals_null	anyarray	-	一个字段组合里最常用数值的列表。如果该字段组合不存在最常用数值，则为NULL。本列保存的多列常用数值中至少有一个值为NULL。
most_common_freqs_null	real[]	-	一个记录字段组合里最常用数值的出现频率的列表，频率由每个数值出现的次数除以行数得到。如果most_common_vals_null取值为NULL，则该字段取值也为NULL。
histogram_bounds	anyarray	-	直方图的边界值列表。

12.3.103 PG_GET_INVALID_BACKENDS

PG_GET_INVALID_BACKENDS视图显示CN上连接到当前DN备机的后台线程信息，只有系统管理员和监控管理员才可以访问。

表 12-193 PG_GET_INVALID_BACKENDS 字段

名称	类型	描述
pid	bigint	线程ID。
node_name	text	后台线程中连接的节点信息。
dbname	name	当前连接的数据库。
backend_start	timestamp with time zone	后台线程启动的时间。
query	text	后台线程正在执行的查询语句。

12.3.104 PG_GET_SENDERS_CATCHUP_TIME

PG_GET_SENDERS_CATCHUP_TIME视图显示DN上当前活跃的主备发送线程的追赶信息。

表 12-194 PG_GET_SENDERS_CATCHUP_TIME 字段

名称	类型	描述
pid	bigint	当前sender的线程ID。
lwpid	integer	当前sender的lwpid。

名称	类型	描述
local_role	text	本地的角色。
peer_role	text	对端的角色。
state	text	当前sender的复制状态。
type	text	当前sender的类型。
catchup_start	timestamp with time zone	catchup启动的时间。
catchup_end	timestamp with time zone	catchup结束的时间。

12.3.105 PG_GROUP

PG_GROUP视图可用来查看数据库认证角色和组之间的成员关系。

表 12-195 PG_GROUP 字段

名称	类型	描述
groname	name	组的名称。
grosysid	oid	组的ID。
grolist	oid[]	一个数组，包含这个组里面所有角色的ID。

12.3.106 PG_INDEXES

PG_INDEXES视图显示数据库中每个索引的信息。

表 12-196 PG_INDEXES 字段

名称	类型	引用	描述
schemaname	name	PG_NAMESPACE.nspname	包含表和索引的模式名称。
tablename	name	PG_CLASS.relname	此索引所服务的表的名称。
indexname	name	PG_CLASS.relname	索引的名称。
tablespace	name	PG_TABLESPACE.nspname	包含索引的表空间名称。

名称	类型	引用	描述
indexdef	text	-	索引定义（一个重建的CREATE INDEX命令）。

12.3.107 PG_LOCKS

PG_LOCKS视图显示各打开事务所持有的锁的信息。

表 12-197 PG_LOCKS 字段

名称	类型	引用	描述
locktype	text	-	被锁定对象的类型：relation、extend、page、tuple、transactionid、virtualxid、object、userlock、advisory。
database	oid	PG_DATABASE.oid	被锁定对象所在数据库的OID。 • 如果被锁定的对象是共享对象，则OID为0。 • 如果被锁定的对象是一个事务，则OID为NULL。
relation	oid	PG_CLASS.oid	关系的OID，如果锁定的对象不是关系，也不是关系的一部分，则为NULL。
page	integer	-	关系内部的页面编号，如果对象不是关系页或者不是行页，则为NULL。
tuple	smallint	-	页面里边的行编号，如果对象不是行，则为NULL。
bucket	integer	-	哈希桶编号。
virtualxid	text	-	虚拟事务的id，如果对象不是一个虚拟事务，则为NULL。
transactionid	xid	-	事务的ID，如果对象不是一个事务，则为NULL。
classid	oid	PG_CLASS.oid	包含该对象的系统表的OID，如果对象不是普通的数据库对象，则为NULL。
objid	oid	-	对象在其系统表内的OID，如果对象不是普通的数据库对象，则为NULL。

名称	类型	引用	描述
objsubid	smallint	-	对于表的一个字段，这是字段编号；对于其他对象类型，这个字段是零；如果这个对象不是普通数据库对象，则为NULL。
virtualtransaction	text	-	持有此锁或者在等待此锁的虚拟事务的ID。
pid	bigint	-	持有或者等待这个锁的服务器逻辑线程的ID。如果锁是被一个预备事务持有的，则为NULL。
sessionid	bigint	-	持有或者等待这个锁的会话的ID。
mode	text	-	这个线程持有的或者是期望的锁模式。
granted	boolean	-	• 如果锁是持有锁，则为TRUE。 • 如果锁是等待锁，则为FALSE。
fastpath	boolean	-	如果通过fast-path获得锁，则为TRUE；如果通过主要的锁表获得，则为FALSE。
locktag	text	-	会话等待锁信息，可通过locktag_decode()函数解析。
global_sessionid	text	-	全局会话ID。

12.3.108 PG_NODE_ENV

PG_NODE_ENV视图显示当前节点的环境变量信息。只有系统管理员或监控管理员用户才可以访问此系统视图。

表 12-198 PG_NODE_ENV 字段

名称	类型	描述
node_name	text	当前节点的名称。
host	text	当前节点的主机名称。
process	integer	当前节点的进程号。
port	integer	当前节点的端口号。
installpath	text	当前节点的安装目录。
datapath	text	当前节点的数据目录。
log_directory	text	当前节点的日志目录。

12.3.109 PG_OS_THREADS

PG_OS_THREADS视图提供当前节点下所有线程的状态信息。

表 12-199 PG_OS_THREADS 字段

名称	类型	描述
node_name	text	当前节点的名称。
pid	bigint	当前节点进程中正在运行的线程号。
lwpid	integer	与pid对应的轻量级线程号。
thread_name	text	与pid对应的线程名称。
creation_time	timestamp with time zone	与pid对应的线程创建的时间。

12.3.110 PG_POOLER_STATUS

PG_POOLER_STATUS视图查询pooler中的缓存连接状态。

表 12-200 PG_POOLER_STATUS 字段

名称	类型	描述
database	text	数据库名称。
user_name	text	用户名。
tid	bigint	非线程池逻辑下为连接CN的线程id，线程池逻辑下为连接CN的sessionid。
node_oid	bigint	连接的实例节点OID。
node_name	name	连接的实例节点名称。
in_use	boolean	连接是否正被使用。 ● t（true）：表示连接正在使用。 ● f（false）：表示连接没有使用。
node_port	integer	连接的实例节点端口号。
fdsock	bigint	对端socket。
remote_pid	bigint	对端线程号。
session_params	text	由此连接下发的GUC session参数。
used_count	bigint	该连接的复用次数。
idx	bigint	连接的实例节点逻辑连接id。
streamid	bigint	每个逻辑连接对应的流标识id。

PG_POOLER_STATUS只能在CN上执行查询，显示本地CN的pooler模块的连接缓存信息。

12.3.111 PG_PREPARED_STATEMENTS

PG_PREPARED_STATEMENTS视图显示当前会话所有可用的预备语句的信息。

表 12-201 PG_PREPARED_STATEMENTS 字段

名称	类型	描述
name	text	预备语句的标识符。
statement	text	创建该预备语句的查询字符串。对于从SQL创建的预备语句而言是客户端提交的PREPARE语句；对于通过前/后端协议创建的预备语句而言是预备语句自身的文本。
prepare_time	timestamp with time zone	创建该预备语句的时间戳。
parameter_types	regtype[]	该预备语句期望的参数类型，以regtype类型的数组格式出现。与该数组元素相对应的OID可以通过把regtype转换为OID值得到。
from_sql	boolean	• 如果该预备语句是通过PREPARE语句创建的则为true。 • 如果是通过前/后端协议创建的则为false。

12.3.112 PG_PREPARED_XACTS

PG_PREPARED_XACTS视图显示当前准备好进行两阶段提交的事务的信息。

表 12-202 PG_PREPARED_XACTS 字段

名称	类型	引用	描述
transaction	xid	-	预备事务的数字标识。
gid	text	-	预备事务的全局标识。
prepared	timestamp with time zone	-	事务准备好提交的时间。
owner	name	PG_AUTHID.rolname	执行该事务的用户的名称。

名称	类型	引用	描述
database	name	PG_DATABASE.datname	执行该事务的数据库的名称。

12.3.113 PG_REPLICATION_ORIGIN_STATUS

PG_REPLICATION_ORIGIN_STATUS视图可用来查看复制源的复制状态。

表 12-203 PG_REPLICATION_ORIGIN_STATUS 字段

名称	类型	描述
local_id	oid	复制源ID。
external_id	text	复制源名称。
remote_lsn	text	复制源的lsn位置。
local_lsn	text	本地的lsn位置。

12.3.114 PG_REPLICATION_SLOTS

PG_REPLICATION_SLOTS视图可用来查看复制槽的信息。

表 12-204 PG_REPLICATION_SLOTS 字段

名称	类型	描述
slot_name	text	复制槽的名称
plugin	text	逻辑复制槽对应的输出插件名称。
slot_type	text	复制槽的类型。 • physical：物理复制槽。 • logical：逻辑复制槽。
datoid	oid	复制槽所在的数据库OID。
database	name	复制槽所在的数据库名称。
active	boolean	复制槽是否为激活状态。 • t（true）：表示是。 • f（false）：表示不是。
xmin	xid	数据库需要为复制槽保留的最早事务的事务号。
catalog_xmin	xid	数据库需要为逻辑复制槽保留的最早的涉及系统表的事务的事务号。

名称	类型	描述
restart_lsn	text	复制槽需要的最早xLog的物理位置。
dummy_standby	boolean	实验室特性。
confirmed_flush	text	逻辑复制槽专用，客户端确认接收到的日志位置。

12.3.115 PG_RLSPOLICIES

PG_RLSPOLICIES视图显示行级访问控制策略的信息。初始化用户和具有sysadmin属性的用户可以查看全部的策略信息，其他用户只能查看自己所拥有表上的策略信息。

表 12-205 PG_RLSPOLICIES 字段

名称	类型	描述
schemaname	name	行级访问控制策略作用的表对象所属模式的名称。
tablename	name	行级访问控制策略作用的表对象名称。
policyname	name	行级访问控制策略名称。
policypermissive	text	行级访问控制策略的表达式拼接方式。取值范围： ● PERMISSIVE：宽容性策略，用OR表达式拼接。 ● RESTRICTIVE：限制性策略，用AND表达式拼接。
policyroles	name[]	行级访问控制策略影响的用户列表，不指定表示影响所有的用户。
policycmd	text	行级访问控制策略影响的SQL操作。
policyqual	text	行级访问控制策略的表达式。

12.3.116 PG_ROLES

PG_ROLES视图显示数据库角色的相关信息。初始化用户和具有sysadmin属性或createrole属性的用户可以查看全部角色的信息，其他用户只能查看自己的信息。

表 12-206 PG_ROLES 字段

名称	类型	引用	描述
rolname	name	-	角色名称。

名称	类型	引用	描述
rolsuper	boolean	-	该角色是否是拥有最高权限的初始系统管理员。 ● t（true）：表示是。 ● f（false）：表示否。
rolinherit	boolean	-	该角色是否继承角色的权限。 ● t（true）：表示是。 ● f（false）：表示否。
rolcreaterole	boolean	-	该角色是否可以创建其他的角色。 ● t（true）：表示是。 ● f（false）：表示否。
rolcreatedb	boolean	-	该角色是否可以创建数据库。 ● t（true）：表示是。 ● f（false）：表示否。
rolcatupdate	boolean	-	该角色是否可以直接更新系统表。只有usesysid=10的初始系统管理员拥有此权限。其他用户无法获得此权限。 ● t（true）：表示是。 ● f（false）：表示否。
rolcanlogin	boolean	-	该角色是否可以登录数据库。 ● t（true）：表示是。 ● f（false）：表示否。
rolreplication	boolean	-	该角色是否可以复制。 ● t（true）：表示是。 ● f（false）：表示否。
rolauditadmin	boolean	-	该角色是否为审计管理员。 ● t（true）：表示是。 ● f（false）：表示否。
rolsystemadmin	boolean	-	该角色是否为系统管理员。 ● t（true）：表示是。 ● f（false）：表示否。
rolconnlimit	integer	-	对于可以登录的角色，这里限制了该角色允许发起的最大并发连接数。 -1表示无限制。
rolpassword	text	-	密文存储后的用户密码，始终是********。

名称	类型	引用	描述
rolvalidbegin	timestamp with time zone	-	账户的有效开始时间；如果没有设置有效开始时间，则为NULL。
rolvaliduntil	timestamp with time zone	-	账户的有效结束时间；如果没有设置有效结束时间，则为NULL。
rolparentid	oid	PG_AUTHID.rolparentid	用户所在组用户的OID。
roltabspace	text	-	用户永久表存储空间限额，单位kB。
rolconfig	text[]	PG_DB_ROLE_SETTING.setconfig	运行时配置项的默认值。
oid	oid	PG_AUTHID.oid	角色的ID。
roluseft	boolean	PG_AUTHID.roluseft	角色是否可以操作外表。 ● t（true）：表示是。 ● f（false）：表示否。
rolkind	"char"	-	角色类型。 ● n：普通用户，即非永久用户。 ● p：永久用户。
nodegroup	name	-	角色所关联的Node group名称，如果没有关联Node group，该值为空。
roltempspace	text	-	用户临时表存储空间限额，单位kB。
rolspillspace	text	-	用户算子落盘空间限额，单位kB。
rolmonitoradmin	boolean	-	该角色是否为监控管理员。 ● t（true）：表示是。 ● f（false）：表示否。
roloperatoradmin	boolean	-	该角色是否为运维管理员。 ● t（true）：表示是。 ● f（false）：表示否。
rolpolicyadmin	boolean	-	该角色是否为安全策略管理员。 ● t（true）：表示是。 ● f（false）：表示否。

12.3.117 PG_RULES

PG_RULES视图可用来查询重写规则的有用信息访问。

表 12-207 PG_RULES 字段

名称	类型	描述
schemaname	name	表的模式的名称。
tablename	name	规则作用的表的名称。
rulename	name	规则的名称。
definition	text	规则的定义（由CREATE语句重新构造得来）。

12.3.118 PG_RUNNING_XACTS

PG_RUNNING_XACTS视图显示当前节点运行事务的信息。

表 12-208 PG_RUNNING_XACTS 字段

名称	类型	描述
handle	integer	事务在GTM对应的句柄。
gxid	xid	事务id号。
state	tinyint	事务状态。 ● 3：prepared。 ● 0：starting。
node	text	节点名称。
xmin	xid	节点上当前数据涉及的最小事务号。
vacuum	boolean	表示当前事务是否是lazy vacuum事务。 ● t（true）：表示是。 ● f（false）：表示不是。
timeline	bigint	数据库重启次数。
prepare_xid	xid	处于prepared状态的事务的id，若事务不在prepared状态，值为0。
pid	bigint	事务对应的线程id。
next_xid	xid	CN传给DN的事务id。

12.3.119 PG_SECLABELS

PG_SECLABELS视图显示安全标签的信息。

表 12-209 PG_SECLABELS 字段

名称	类型	引用	描述
objoid	oid	任意OID属性	该安全标签所属的对象的OID。
classoid	oid	PG_CLASS.oid	该安全标签所属的对象所在系统表的OID。
objsubid	integer	-	• 对于一个在表字段上的安全标签，该字段是字段序号（引用表本身的objoid和classoid）。 • 对于所有其他对象类型，该字段为0。
objtype	text	-	该标签所属的对象的类型，文本格式。例如： • table：表类型。 • column：列类型。
objnamespace	oid	PG_NAMESPACE.oid	该对象的名称空间的OID，如果不适用，取值为NULL。
objname	text	-	该标签所属的对象的名称，文本格式。
provider	text	PG_SECLABEL.provider	该标签的提供者。
label	text	PG_SECLABEL.label	安全标签名称。

12.3.120 PG_SETTINGS

PG_SETTINGS视图显示数据库运行时参数的相关信息。

表 12-210 PG_SETTINGS 字段

名称	类型	描述
name	text	参数名称。
setting	text	参数当前值。
unit	text	参数的单位。
category	text	参数的逻辑组。
short_desc	text	参数的简单描述。

名称	类型	描述
extra_desc	text	参数的详细描述。
context	text	设置参数值的上下文，包括internal、postmaster、sighup、backend、superuser、user。
vartype	text	参数类型，包括bool、enum、integer、real、string。
source	text	参数的赋值方式。
min_val	text	参数最小值。如果参数类型不是数值型，那么该字段值为null。
max_val	text	参数最大值。如果参数类型不是数值型，那么该字段值为null。
enumvals	text[]	enum类型参数合法值。如果参数类型不是enum型，那么该字段值为null。
boot_val	text	数据库启动时参数默认值。
reset_val	text	数据库重置时参数默认值。
sourcefile	text	设置参数值的配置文件。如果参数不是通过配置文件赋值，那么该字段值为null。
sourceline	integer	设置参数值的配置文件的行号。如果参数不是通过配置文件赋值，那么该字段值为null。

12.3.121 PG_SHADOW

PG_SHADOW视图显示了所有在**PG_AUTHID**中标记了rolcanlogin的角色的属性，只有系统管理员权限才可以访问此系统视图。

该视图的信息与**PG_USER**是基本一致的，区别在于后者对密码做了敏感化处理，统一显示为********。

表 12-211 PG_SHADOW 字段

名称	类型	引用	描述
usename	name	**PG_AUTHID**.rolname	用户名。
usesysid	oid	**PG_AUTHID**.oid	用户的ID。
usecreatedb	boolean	-	用户是否可以创建数据库。 • t（true）：表示是。 • f（false）：表示否。

名称	类型	引用	描述
usesuper	boolean	-	用户是否是系统管理员。 ● t（true）：表示是。 ● f（false）：表示否。
usecatupd	boolean	-	用户是否可以更新视图。即使是系统管理员，如果这个字段值不是t，也不能更新视图。 ● t（true）：表示是。 ● f（false）：表示否。
userepl	boolean	-	用户是否可以复制数据流。 ● t（true）：表示是。 ● f（false）：表示否。
passwd	text	PG_AUTHID.rolpassword	密码密文，如果没有密码，则为NULL。
valbegin	timestamp with time zone	-	账户的有效开始时间；如果没有设置有效开始时间，则为NULL。
valuntil	timestamp with time zone	-	账户的有效结束时间；如果没有设置有效结束时间，则为NULL。
respool	name	-	用户所在的资源池。
parent	oid	-	父用户OID。
spacelimit	text	-	永久表存储空间的限额，单位kB。
useconfig	text[]	PG_DB_ROLE_SETTING.setconfig	运行时配置项的默认值。
tempspacelimit	text	-	临时表存储空间的限额，单位kB。
spillspacelimit	text	-	算子落盘空间的限额，单位kB。
usemonitoradmin	boolean	-	用户是否是监控管理员。 ● t（true）：表示是。 ● f（false）：表示否。
useoperatoradmin	boolean	-	用户是否是运维管理员。 ● t（true）：表示是。 ● f（false）：表示否。

名称	类型	引用	描述
usepolicyadmin	boolean	-	用户是否是安全策略管理员。 ● t（true）：表示是。 ● f（false）：表示否。

12.3.122 PG_SHARED_MEMORY_DETAIL

PG_SHARED_MEMORY_DETAIL视图显示所有已产生的共享内存上下文的使用信息。

表 12-212 PG_SHARED_MEMORY_DETAIL 字段

名称	类型	描述
contextname	text	内存上下文的名称。
level	smallint	当前上下文在整体内存上下文中的层级。
parent	text	上级内存上下文。
totalsize	bigint	共享内存总大小，单位Byte。
freesize	bigint	共享内存剩余大小，单位Byte。
usedsize	bigint	共享内存使用大小，单位Byte。

12.3.123 PG_STATS

PG_STATS视图可用来查看存储在pg_statistic表里面的单列统计信息。该视图记录的统计信息更新时间间隔由参数autovacuum_naptime设置。

表 12-213 PG_STATS 字段

名称	类型	引用	描述
schemaname	name	PG_NAMESPACE.nspname	表的模式名。
tablename	name	PG_CLASS.relname	表名。
attname	name	PG_ATTRIBUTE.attname	字段的名称。
inherited	boolean	-	暂不支持继承表，该字段为false。
null_frac	real	-	记录中字段为空的百分比。
avg_width	integer	-	字段记录以字节记的平均宽度。

名称	类型	引用	描述
n_distinct	real	-	• 如果大于零，表示字段中独立数值的估计数目。 • 如果小于零，表示独立数值的数目除以行数后乘-1得到的负数。比如，-1表示一个唯一字段，独立数值的个数和行数相同。 1. 用负数形式是因为ANALYZE认为独立数值的数目是随着表增长而增长； 2. 正数的形式用于在字段看上去好像有固定的可能值数目的情况下。 • 如果等于零，表示独立数值的数目未知。
n_dndistinct	real	-	标识dn1上字段中非NULL的独立数值的数目。 • 如果大于零，表示独立数值的实际数目。 • 如果小于零，表示独立数值的数目被行数除的负数。比如，一个字段的数值平均出现概率为两次，则可以表示为n_dndistinct=-0.5。 • 如果等于零，表示独立数值的数目未知。
most_common_vals	anyarray	-	一个字段里最常用数值的列表。如果该字段不存在最常用数值，则为NULL。
most_common_freqs	real[]	-	一个记录字段里最常用数值的出现频率的列表，频率由每个数值出现的次数除以行数得到。如果most_common_vals是NULL，则为NULL。
histogram_bounds	anyarray	-	由排除了空值和MCV值之外的取值组成的等频直方图。如果某个数值出现在most_common_vals中，则不出现在直方图里。如果字段数据类型没有<操作符或者most_common_vals列表包含了该字段所有取值，则这个字段的直方图信息为NULL。

名称	类型	引用	描述
correlation	real	-	字段值的物理行序和逻辑行序的相关性。取值范围从-1到+1。该值接近-1或者+1的时候，因为减少了对磁盘的随机访问，索引扫描的开销比接近零的时候更少。如果字段数据类型没有<操作符，则这个字段的相关性为NULL。
most_common_elems	anyarray	-	一个记录最常用的非空元素的列表。
most_common_elem_freqs	real[]	-	一个记录最常用的非空元素的出现频率的列表。
elem_count_histogram	real[]	-	对于独立的非空元素的统计直方图。

12.3.124 PG_STAT_ACTIVITY

PG_STAT_ACTIVITY视图显示和当前用户查询相关的信息，字段保存的是上一次执行的信息。

表 12-214 PG_STAT_ACTIVITY 字段

名称	类型	描述
datid	oid	用户会话在后台连接到的数据库OID。
datname	name	用户会话在后台连接到的数据库名称。
pid	bigint	后台线程ID。
sessionid	bigint	会话ID。
usesysid	oid	登录该后台的用户OID。
usename	name	登录该后台的用户名。
application_name	text	连接到该后台的应用名。
client_addr	inet	连接到该后台的客户端的IP地址。 如果该字段取值是null，表明是通过服务器机器上UNIX套接字连接客户端或者这是内部线程，如autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过client_addr的反向DNS查找得到。这个字段只有在启动log_hostname且使用IP连接时才非空。

名称	类型	描述
client_port	integer	客户端用于与后台通讯的TCP端口号，如果使用UNIX套接字，则为-1。
backend_start	timestamp with time zone	该会话开始的时间，即客户端连接服务器的时间。
xact_start	timestamp with time zone	当前活跃事务开始的时间，如果没有事务是活跃的，则为null。如果当前查询是首个事务，则这列等同于query_start列。
query_start	timestamp with time zone	当前活跃查询开始的时间，如果state的值不是active，则这个值是上一个查询的开始时间。如果是存储过程、函数、PACKAGE，则显示的是第一个查询时间，不会随着存储过程内语句运行而改变。
state_change	timestamp with time zone	上次状态改变的时间。
waiting	boolean	如果后台当前正等待锁则为true。否则为false。
enqueue	text	语句当前排队状态。可能值是： ● waiting in queue：表示语句在排队中。 ● 空：表示语句正在运行。

名称	类型	描述
state	text	该后台当前总体状态。可能值是： ● active：后台正在执行一个查询。 ● idle：后台正在等待一个新的客户端命令。 ● idle in transaction：后台在事务中，但事务中没有语句在执行。 ● idle in transaction (aborted)：后台在事务中，但事务中有语句执行失败。 ● fastpath function call：后台正在执行一个fast-path函数。 ● disabled：如果后台禁用track_activities，则报告这个状态。 **说明** 普通用户只能查看到自己账户所对应的会话状态。即其他账户的state信息为空。例如以judy用户连接数据库后，在pg_stat_activity中查看到的普通用户joe及初始用户omm的state信息为空： SELECT datname, usename, usesysid, state,pid FROM pg_stat_activity; datname | usename | usesysid | state | pid ----------+---------+----------+--------+----------------- testdb | omm | 10 | | 139968752121616 testdb | omm | 10 | | 139968903116560 db_tpcds | judy | 16398 | active | 139968391403280 testdb | omm | 10 | | 139968643069712 testdb | omm | 10 | | 139968680818448 testdb | joe | 16390 | | 139968563377936 (6 rows)
resource_pool	name	用户使用的资源池。
query_id	bigint	查询语句的ID。
query	text	该后台的最新查询。如果state状态是active（活跃的），此字段显示当前正在执行的查询。所有其他情况表示上一个查询。
connection_info	text	json格式字符串，记录当前连接数据库的驱动类型、驱动版本号、当前驱动的部署路径、进程属主用户等信息。
global_sessionid	text	全局的会话ID。
unique_sql_id	bigint	语句的unique sql id。

名称	类型	描述
trace_id	text	驱动传入的trace id，与应用的一次请求相关联。

12.3.125 PG_STAT_ACTIVITY_NG

PG_STAT_ACTIVITY_NG视图显示在当前用户所属的Node group下，所有查询的相关信息。

表 12-215 PG_STAT_ACTIVITY_NG 字段

名称	类型	描述
datid	oid	用户会话在后台连接到的数据库OID。
datname	name	用户会话在后台连接到的数据库名称。
pid	bigint	后台线程ID。
sessionid	bigint	会话ID。
global_sessionid	text	全局会话ID。
usesysid	oid	登录该后台的用户OID。
usename	name	登录该后台的用户名。
application_name	text	连接到该后台的应用名。
client_addr	inet	连接到该后台的客户端的IP地址。 如果此字段是null，它表明通过服务器机器上UNIX套接字连接客户端或者这是内部进程，如autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过client_addr的反向DNS查找得到。这个字段只有在启动log_hostname且使用IP连接时才非空。
client_port	integer	客户端用于与后台通讯的TCP端口号，如果使用UNIX套接字，则为-1。
backend_start	timestamp with time zone	该会话开始的时间，即客户端连接服务器的时间。
xact_start	timestamp with time zone	当前活跃事务开始的时间，如果没有事务是活跃的，则为null。如果当前查询是首个事务，则这列等同于query_start列。

名称	类型	描述
query_start	timestamp with time zone	当前活跃查询开始的时间，如果state的值不是active，则这个值是上一个查询的开始时间。如果是存储过程、函数、PACKAGE，则显示的是第一个查询时间，不会随着存储过程内语句运行而改变。
state_change	timestamp with time zone	上次状态改变的时间。
waiting	boolean	如果后台当前正等待锁则为true。否则为false。
enqueue	text	语句当前排队状态。可能值是： ● waiting in queue：表示语句在排队中。 ● 空：表示语句正在运行。

名称	类型	描述
state	text	该后台当前总体状态。可能值是： ● active：后台正在执行一个查询。 ● idle：后台正在等待一个新的客户端命令。 ● idle in transaction：后台在事务中，但事务中没有语句在执行。 ● idle in transaction (aborted)：后台在事务中，但事务中有语句执行失败。 ● fastpath function call：后台正在执行一个fast-path函数。 ● disabled：如果后台禁用track_activities，则报告这个状态。 **说明** 普通用户只能查看到自己账户所对应的会话状态。即其他账户的state信息为空。例如以judy用户连接数据库后，在pg_stat_activity中查看到的普通用户joe及初始用户omm的state信息为空： SELECT datname, usename, usesysid, state,pid FROM pg_stat_activity_ng; datname | usename | usesysid | state | pid ----------+---------+----------+--------+----------------- testdb | omm | 10 | | 139968752121616 testdb | omm | 10 | | 139968903116560 db_tpcds | judy | 16398 | active | 139968391403280 testdb | omm | 10 | | 139968643069712 testdb | omm | 10 | | 139968680818448 testdb | joe | 16390 | | 139968563377936 (6 rows)
resource_pool	name	用户使用的资源池。
query_id	bigint	查询语句的ID。
query	text	该后台的最新查询。如果state状态是active（活跃的），此字段显示当前正在执行的查询。所有其他情况表示上一个查询。
node_group	text	语句所属用户对应的Node group。

12.3.126 PG_STAT_ALL_INDEXES

PG_STAT_ALL_INDEXES视图可用来查询当前数据库中的每个索引行，显示访问特定索引的统计。

索引可以通过简单的索引扫描或位图索引扫描进行使用。位图扫描中几个索引的输出可以通过AND或者OR规则进行组合， 因此当使用位图扫描的时候，很难将独立堆行抓取与特定索引进行组合， 因此，每一次位图扫描都会增加
pg_stat_all_indexes.idx_tup_read使用索引的计数，并且增加
pg_stat_all_tables.idx_tup_fetch表的计数，但不影响
pg_stat_all_indexes.idx_tup_fetch。

表 12-216 PG_STAT_ALL_INDEXES 字段

名称	类型	描述
relid	oid	索引所在的表的OID。
indexrelid	oid	索引的OID。
schemaname	name	索引的模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	通过索引上扫描返回的索引项数。
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

12.3.127 PG_STAT_ALL_TABLES

PG_STAT_ALL_TABLES视图可用来查询当前数据库中每个表的信息（包括TOAST表），显示访问特定表的统计信息。

表 12-217 PG_STAT_ALL_TABLES 字段

名称	类型	描述
relid	oid	表的OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。

名称	类型	描述
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（比如没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计不活跃行数。
last_vacuum	timestamp with time zone	最后一次清理该表的时间。
last_autovacuum	timestamp with time zone	这个表上次被autovacuum守护进程清理的时间。
last_analyze	timestamp with time zone	上次分析该表的时间。
last_autoanalyze	timestamp with time zone	这个表上次被autovacuum守护进程分析的时间。
vacuum_count	bigint	这个表被清理的次数。
autovacuum_count	bigint	这个表被autovacuum清理的次数。
analyze_count	bigint	这个表被手动分析的次数。
autoanalyze_count	bigint	这个表被autovacuum守护进程分析的次数。
last_data_changed	timestamp with time zone	记录这个表上一次数据发生变化的时间（引起数据变化的操作包括INSERT/UPDATE/DELETE、EXCHANGE/TRUNCATE/DROP partition，系统表不记录该字段），该列数据仅在本地CN记录。

12.3.128 PG_STAT_BAD_BLOCK

PG_STAT_BAD_BLOCK视图显示自节点启动后，读取数据时出现Page/CU校验失败的统计信息。

表 12-218 PG_STAT_BAD_BLOCK 字段

名称	类型	描述
nodename	text	节点名。

名称	类型	描述
databaseid	integer	数据库OID。
tablespaceid	integer	表空间OID。
relfilenode	integer	文件对象ID。
bucketid	smallint	一致性hash bucket ID。
forknum	integer	文件类型。取值如下： ● 0：数据主文件。 ● 1：FSM文件。 ● 2：VM文件。 ● 3：BCM文件。
error_count	integer	出现校验失败的次数。
first_time	timestamp with time zone	第一次出现时间。
last_time	timestamp with time zone	最后一次出现时间。

12.3.129 PG_STAT_BGWRITER

PG_STAT_BGWRITER视图显示后端写进程活动的统计信息。

表 12-219 PG_STAT_BGWRITER 字段

名称	类型	描述
checkpoints_timed	bigint	定期执行的检查点数。
checkpoints_req	bigint	主动执行的检查点数。
checkpoint_write_time	double precision	将文件写入到磁盘时，在检查点处理部分花费的时间总量，以毫秒为单位。
checkpoint_sync_time	double precision	将文件同步到磁盘时，在检查点处理部分花费的时间总量，以毫秒为单位。
buffers_checkpoint	bigint	检查点写入的缓冲区的数量。
buffers_clean	bigint	后端写进程写入的缓冲区的数量。
maxwritten_clean	bigint	后端写进程因写入的缓冲区过多导致的清理扫描停止的次数。

名称	类型	描述
buffers_backend	bigint	后端直接写入的缓冲区的数量。
buffers_backend_fsync	bigint	后端自己执行fsync调用的次数（通常情况下，即使后端自己执行了这些写入动作，后端写进程也会再处理一次）。
buffers_alloc	bigint	分配的缓冲区数量。
stats_reset	timestamp with time zone	这些统计被重置的时间。

12.3.130 PG_STAT_DATABASE

PG_STAT_DATABASE视图显示集群中每个数据库的统计信息。

表 12-220 PG_STAT_DATABASE 字段

名称	类型	描述
datid	oid	数据库的OID。
datname	name	数据库的名称。
numbackends	integer	当前连接到该数据库的后端数。 这是该视图中唯一一个返回当前状态值的字段，其他字段返回的都是自上次重置之后的累计值。
xact_commit	bigint	该数据库中已经提交的事务数。
xact_rollback	bigint	该数据库中已经回滚的事务数。
blks_read	bigint	在该数据库中读取的磁盘块的数量。
blks_hit	bigint	已在缓冲区缓存中找到磁盘块的次数，因此不需要读取（只统计在缓冲区缓存找到的，不包括在操作系统的文件系统缓存中找到的）。
tup_returned	bigint	通过数据库查询返回的行数。
tup_fetched	bigint	通过数据库查询抓取的行数。
tup_inserted	bigint	通过数据库查询插入的行数。
tup_updated	bigint	通过数据库查询更新的行数。
tup_deleted	bigint	通过数据库查询删除的行数。
conflicts	bigint	由于数据库恢复冲突取消的查询数量（只在备用服务器发生的冲突）。请参见PG_STAT_DATABASE_CONFLICTS获取更多信息。

名称	类型	描述
temp_files	bigint	通过数据库查询创建的临时文件数量。计算所有临时文件， 无论该临时文件为什么创建（比如排序或者哈希）， 也不管log_temp_files参数如何设置。
temp_bytes	bigint	通过数据库查询写入临时文件的数据总量。计算所有临时文件，无论该临时文件为什么创建，也不管log_temp_files参数如何设置。
deadlocks	bigint	该数据库中检测到的死锁数。
blk_read_time	double precision	通过数据库后端读取数据文件块花费的时间，以毫秒计算。
blk_write_time	double precision	通过数据库后端写入数据文件块花费的时间，以毫秒计算。
stats_reset	timestamp with time zone	当前状态统计被重置的时间。

12.3.131 PG_STAT_DATABASE_CONFLICTS

PG_STAT_DATABASE_CONFLICTS视图显示数据库冲突状态的统计信息。

表 12-221 PG_STAT_DATABASE_CONFLICTS 字段

名称	类型	描述
datid	oid	数据库标识。
datname	name	数据库名称。
confl_tablespace	bigint	冲突的表空间的数目。
confl_lock	bigint	冲突的锁数目。
confl_snapshot	bigint	冲突的快照数目。
confl_bufferpin	bigint	冲突的缓冲区数目。
confl_deadlock	bigint	冲突的死锁数目。

12.3.132 PG_STAT_REPLICATION

PG_STAT_REPLICATION视图显示日志同步线程的信息，如发起端发送日志位置，接收端接收日志位置等。

表 12-222 PG_STAT_REPLICATION 字段

名称	类型	描述
pid	bigint	线程的PID。
usesysid	oid	用户系统ID。
usename	name	用户名。
application_name	text	程序名称。
client_addr	inet	客户端地址。
client_hostname	text	客户端名。
client_port	integer	客户端端口。
backend_start	timestamp with time zone	程序启动时间。
state	text	日志同步线程的状态。 ● startup：线程正在启动。 ● catchup：线程正在建立备用服务器和主服务器的连接。 ● streaming：线程已建立备用服务器和主服务器的连接，正在进行数据的流复制。 ● backup：线程正在发送备份。 ● stopping：线程正在停止。
sender_sent_location	text	发送端发送日志位置。
receiver_write_location	text	接收端write日志位置。
receiver_flush_location	text	接收端flush日志位置。
receiver_replay_location	text	接收端replay日志位置。
sync_priority	integer	同步复制的优先级（0表示异步）。

名称	类型	描述
sync_state	text	同步状态： ● async：异步复制。 ● sync：同步复制。 ● potential：该备用服务器现在是异步的，但假如一个当前的同步服务器发生故障，该服务器会变成同步的。 ● qurom：在同步与异步之间切换，保证备机中有大于一定数量的同步备机，同步备机数量一般为(n+1)/2-1，n为总副本个数。是否为同步备机取决于是否先接到了日志。详情可参考synchronous_standby_names参数描述。

12.3.133 PG_STAT_SYS_INDEXES

PG_STAT_SYS_INDEXES视图显示pg_catalog、information_schema模式中所有系统表的索引状态信息。

表 12-223 PG_STAT_SYS_INDEXES 字段

名称	类型	描述
relid	oid	该索引所在的表的OID。
indexrelid	oid	索引的OID。
schemaname	name	索引的模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	该索引上扫描返回的索引项数。
idx_tup_fetch	bigint	使用该索引的简单索引扫描在原表中抓取的活表行数。

12.3.134 PG_STAT_SYS_TABLES

PG_STAT_SYS_TABLES视图显示pg_catalog、information_schema模式的所有命名空间中系统表的统计信息。

表 12-224 PG_STAT_SYS_TABLES 字段

名称	类型	描述
relid	oid	表的OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（比如没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计不活跃行数。
last_vacuum	timestamp with time zone	上次手动清理该表的时间（不计算VACUUM FULL）。
last_autovacuum	timestamp with time zone	上次被autovacuum守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析这个表的时间。
last_autoanalyze	timestamp with time zone	上次被autovacuum守护进程分析的时间。
vacuum_count	bigint	这个表被手动清理的次数（不计算VACUUM FULL）。
autovacuum_count	bigint	这个表被autovacuum清理的次数。
analyze_count	bigint	这个表被手动分析的次数。
autoanalyze_count	bigint	这个表被autovacuum守护进程分析的次数。

名称	类型	描述
last_data_changed	timestamp with time zone	这个表数据最近修改时间。

12.3.135 PG_STAT_USER_FUNCTIONS

PG_STAT_USER_FUNCTIONS视图显示命名空间中用户自定义函数（函数语言为非内部语言）的状态信息。

表 12-225 PG_STAT_USER_FUNCTIONS 字段

名称	类型	描述
funcid	oid	函数标识。
schemaname	name	模式的名称。
funcname	name	函数名称。
calls	bigint	函数被调用的次数。
total_time	double precision	函数的总执行时长。
self_time	double precision	当前线程调用函数的总的时长。

12.3.136 PG_STAT_USER_INDEXES

PG_STAT_USER_INDEXES视图显示数据库中用户自定义普通表和toast表的索引状态信息。

表 12-226 PG_STAT_USER_INDEXES 字段

名称	类型	描述
relid	oid	该索引所在的表的OID。
indexrelid	oid	索引的OID。
schemaname	name	索引的模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	该索引上扫描返回的索引项数。
idx_tup_fetch	bigint	使用该索引的简单索引扫描在原表中抓取的活表行数。

12.3.137 PG_STAT_USER_TABLES

PG_STAT_USER_TABLES视图显示所有命名空间中用户自定义普通表和toast表的状态信息。

表 12-227 PG_STAT_USER_TABLES 字段

名称	类型	描述
relid	oid	表的OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（即没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计不活跃行数。
last_vacuum	timestamp with time zone	上次手动清理该表的时间（不计算VACUUM FULL）。
last_autovacuum	timestamp with time zone	上次被autovacuum守护进程清理的表。
last_analyze	timestamp with time zone	上次手动分析这个表的时间。
last_autoanalyze	timestamp with time zone	上次被autovacuum守护线程分析的时间。
vacuum_count	bigint	这个表被手动清理的次数（不计算VACUUM FULL）。
autovacuum_count	bigint	这个表被autovacuum清理的次数。
analyze_count	bigint	这个表被手动分析的次数。
autoanalyze_count	bigint	这个表被autovacuum守护进程分析的次。

名称	类型	描述
last_data_changed	timestamp with time zone	这个表数据最近修改时间。

12.3.138 PG_STAT_XACT_ALL_TABLES

PG_STAT_XACT_ALL_TABLES视图显示命名空间中所有普通表和toast表的事务状态信息。

表 12-228 PG_STAT_XACT_ALL_TABLES 字段

名称	类型	描述
relid	oid	表的OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（比如没有更新所需的单独索引）。

12.3.139 PG_STAT_XACT_SYS_TABLES

PG_STAT_XACT_SYS_TABLES视图显示命名空间中系统表的事务状态信息。

表 12-229 PG_STAT_XACT_SYS_TABLES 字段

名称	类型	描述
relid	oid	表的OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。

名称	类型	描述
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（比如没有更新所需的单独索引）。

12.3.140 PG_STAT_XACT_USER_FUNCTIONS

PG_STAT_XACT_USER_FUNCTIONS视图包含每个函数的执行的统计信息。

表 12-230 PG_STAT_XACT_USER_FUNCTIONS 字段

名称	类型	描述
funcid	oid	函数标识。
schemaname	name	模式的名称。
funcname	name	函数名称。
calls	bigint	函数被调用的次数。
total_time	double precision	函数的总执行时长。
self_time	double precision	当前线程调用函数的总的时长。

12.3.141 PG_STAT_XACT_USER_TABLES

PG_STAT_XACT_USER_TABLES视图显示命名空间中用户表的事务状态信息。

表 12-231 PG_STAT_XACT_USER_TABLES 字段

名称	类型	描述
relid	oid	表的OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。

名称	类型	描述
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（比如没有更新所需的单独索引）。

12.3.142 PG_STATIO_ALL_INDEXES

PG_STATIO_ALL_INDEXES视图可用来查询当前数据库中的每个索引行的信息，显示特定索引的I/O的统计信息。

表 12-232 PG_STATIO_ALL_INDEXES 字段

名称	类型	描述
relid	oid	索引的表的OID。
indexrelid	oid	该索引的OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	bigint	从索引中读取的磁盘块数。
idx_blks_hit	bigint	索引命中缓存数。

12.3.143 PG_STATIO_ALL_SEQUENCES

PG_STATIO_ALL_SEQUENCES视图显示当前数据库中每个序列的I/O的统计信息。

表 12-233 PG_STATIO_ALL_SEQUENCES 字段

名称	类型	描述
relid	oid	序列OID。
schemaname	name	序列的模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列命中缓存数。

12.3.144 PG_STATIO_ALL_TABLES

PG_STATIO_ALL_TABLES视图可用来查询当前数据库中每个表（包括TOAST表）的I/O统计信息。

表 12-234 PG_STATIO_ALL_TABLES 字段

名称	类型	描述
relid	oid	表OID。
schemaname	name	该表的模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表命中缓存数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	从该表的TOAST表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的TOAST表命中缓存数（如果存在）。
tidx_blks_read	bigint	从该表的TOAST表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该表的TOAST表索引命中缓存数（如果存在）。

12.3.145 PG_STATIO_SYS_INDEXES

PG_STATIO_SYS_INDEXES视图显示命名空间中所有系统表索引的I/O状态信息。

表 12-235 PG_STATIO_SYS_INDEXES 字段

名称	类型	描述
relid	oid	该索引所在的表的OID。
indexrelid	oid	该索引的OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	bigint	从索引中读取的磁盘块数。
idx_blks_hit	bigint	索引命中缓存数。

12.3.146 PG_STATIO_SYS_SEQUENCES

PG_STATIO_SYS_SEQUENCES视图显示命名空间中所有序列的I/O状态信息。

表 12-236 PG_STATIO_SYS_SEQUENCES 字段

名称	类型	描述
relid	oid	序列OID。
schemaname	name	序列的模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列命中缓存数。

12.3.147 PG_STATIO_SYS_TABLES

PG_STATIO_SYS_TABLES视图显示命名空间中所有系统表的I/O状态信息。

表 12-237 PG_STATIO_SYS_TABLES 字段

名称	类型	描述
relid	oid	表OID。
schemaname	name	该表的模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表命中缓存数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	从该表的TOAST表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的TOAST表命中缓存数（如果存在）。
tidx_blks_read	bigint	从该表的TOAST表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该表的TOAST表索引命中缓存数（如果存在）。

12.3.148 PG_STATIO_USER_INDEXES

PG_STATIO_USER_INDEXES视图显示命名空间中所有用户关系表索引的I/O状态信息。

表 12-238 PG_STATIO_USER_INDEXES 字段

名称	类型	描述
relid	oid	该索引所在的表的OID。
indexrelid	oid	该索引的OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名.
indexrelname	name	索引名称。
idx_blks_read	bigint	从索引中读取的磁盘块数。
idx_blks_hit	bigint	索引命中缓存数。

12.3.149 PG_STATIO_USER_SEQUENCES

PG_STATIO_USER_SEQUENCES视图显示命名空间中所有用户关系表序列的I/O状态信息。

表 12-239 PG_STATIO_USER_SEQUENCES 字段

名称	类型	描述
relid	oid	序列OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列命中缓存数。

12.3.150 PG_STATIO_USER_TABLES

PG_STATIO_USER_TABLES视图显示命名空间中所有用户关系表的I/O状态信息。

表 12-240 PG_STATIO_USER_TABLES 字段

名称	类型	描述
relid	oid	表OID。

名称	类型	描述
schemaname	name	该表的模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表命中缓存数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	从该表的TOAST表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的TOAST表命中缓存数（如果存在）。
tidx_blks_read	bigint	从该表的TOAST表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该表的TOAST表索引命中缓存数（如果存在）。

12.3.151 PG_TABLES

PG_TABLES视图可用来查询数据库中每个表的有用信息。

表 12-241 PG_TABLES 字段

名称	类型	引用	描述
schemaname	name	PG_NAMESPACE.nspname	表的模式名。
tablename	name	PG_CLASS.relname	表名。
tableowner	name	pg_get_userbyid(PG_CLASS.relowner)	表的所有者。
tablespace	name	PG_TABLESPACE.spcname	包含表的表空间，默认为NULL。
hasindexes	boolean	PG_CLASS.relhasindex	如果表上有索引（或者最近拥有）则为TRUE，否则为FALSE。
hasrules	boolean	PG_CLASS.relhasruls	如果表上有规则，则为TRUE，否则为FALSE。
hastriggers	boolean	PG_CLASS.RELHASTRIGGERS	如果表上有触发器，则为TRUE，否则为FALSE。
tablecreator	name	pg_get_userbyid(PG_OBJECT.creator)	表的创建者。

名称	类型	引用	描述
created	timestamp with time zone	PG_OBJECT.ctime	表的创建时间。
last_ddl_time	timestamp with time zone	PG_OBJECT.mtime	最后一次对该表执行DDL操作的时间。

12.3.152 PG_TDE_INFO

PG_TDE_INFO视图显示整个集群的加密信息。

表 12-242 PG_TDE_INFO 字段

名称	类型	描述
is_encrypt	boolean	是否加密集群。 ● f：非加密集群。 ● t：加密集群。
g_tde_algo	text	加密算法。 ● SM4-CTR-128 ● AES-CTR-128
remain	text	保留字段。

12.3.153 PG_TIMEZONE_ABBREVS

PG_TIMEZONE_ABBREVS视图显示所有可用时区的信息。

表 12-243 PG_TIMEZONE_ABBREVS 字段

名称	类型	描述
abbrev	text	时区名缩写。
utc_offset	interval	相对于UTC的偏移量。
is_dst	boolean	如果当前正实行夏令时则为TRUE，否则为FALSE。

12.3.154 PG_TIMEZONE_NAMES

PG_TIMEZONE_NAMES视图显示所有能够被SET TIMEZONE语法识别的时区名及其缩写、UTC偏移量、是否实行夏令时。

表 12-244 PG_TIMEZONE_NAMES 字段

名称	类型	描述
name	text	时区名。
abbrev	text	时区名缩写。
utc_offset	interval	相对于UTC的偏移量。
is_dst	boolean	如果当前正实行夏令时则为TRUE，否则为FALSE。

12.3.155 PG_TOTAL_MEMORY_DETAIL

PG_TOTAL_MEMORY_DETAIL视图显示某个数据库节点内存使用情况。

表 12-245 PG_TOTAL_MEMORY_DETAIL 字段

名称	类型	描述
nodename	text	节点名称。
memorytype	text	内存的名称。
memorymbytes	integer	内存使用的大小，单位为MB。

12.3.156 PG_TOTAL_USER_RESOURCE_INFO

PG_TOTAL_USER_RESOURCE_INFO视图显示所有用户的资源使用情况，需要使用管理员用户进行查询。此视图在GUC参数use_workload_manager为on时才有效。其中，I/O相关监控项在参数enable_logical_io_statistics为on时才有效。

表 12-246 PG_TOTAL_USER_RESOURCE_INFO 字段

名称	类型	描述
username	name	用户名。
used_memory	integer	正在使用的内存大小，单位MB。
total_memory	integer	可以使用的内存大小，单位MB。值为0表示未限制最大可用内存，其限制取决于数据库最大可用内存。

名称	类型	描述
used_cpu	double precision	正在使用的CPU核数（仅统计复杂作业CPU使用情况，且该值为相关控制组的CPU使用统计值）。
total_cpu	integer	在该机器节点上，用户关联控制组的CPU核数总和。
used_space	bigint	已使用的永久表存储空间大小，单位KB。
total_space	bigint	可使用的永久表存储空间大小，单位KB，值为-1表示未限制最大存储空间。
used_temp_space	bigint	已使用的临时空间大小，单位KB。
total_temp_space	bigint	可使用的临时空间总大小，单位KB，值为-1表示未限制。
used_spill_space	bigint	已使用的算子落盘空间大小，单位KB。
total_spill_space	bigint	可使用的算子落盘空间总大小，单位KB，值为-1表示未限制。
read_kbytes	bigint	CN：过去5秒内，该用户在所有DN上复杂作业read的字节总数（单位KB）。 DN：实例启动至当前时间为止，该用户复杂作业read的字节总数（单位KB）。
write_kbytes	bigint	CN：过去5秒内，该用户在所有DN上复杂作业write的字节总数（单位KB）。 DN：实例启动至当前时间为止，该用户复杂作业write的字节总数（单位KB）。
read_counts	bigint	CN：过去5秒内，该用户在所有DN上复杂作业read的次数之和（单位次）。 DN：实例启动至当前时间为止，该用户复杂作业read的次数之和（单位次）。
write_counts	bigint	CN：过去5秒内，该用户在所有DN上复杂作业write的次数之和（单位次）。 DN：实例启动至当前时间为止，该用户复杂作业write的次数之和（单位次）。
read_speed	double precision	CN：过去5秒内，该用户在单个DN上复杂作业read平均速率（单位KB/s）。 DN：过去5秒内，该用户在该DN上复杂作业read平均速率（单位KB/s）。

名称	类型	描述
write_speed	double precision	CN：过去5秒内，该用户在单个DN上复杂作业write平均速率（单位KB/s）。 DN：过去5秒内，该用户在该DN上复杂作业write平均速率（单位KB/s）。

12.3.157 PG_TOTAL_USER_RESOURCE_INFO_OID

PG_TOTAL_USER_RESOURCE_INFO_OID视图显示所有用户资源使用情况，需要使用管理员用户进行查询。此视图在GUC参数use_workload_manager为on时才有效。

表 12-247 PG_TOTAL_USER_RESOURCE_INFO_OID 字段

名称	类型	描述
userid	oid	用户ID。
used_memory	integer	正在使用的内存大小，单位MB。
total_memory	integer	可以使用的内存大小，单位MB。值为0表示未限制最大可用内存，其限制取决于数据库最大可用内存。
used_cpu	double precision	正在使用的CPU核数。
total_cpu	integer	在该机器节点上，用户关联控制组的CPU核数总和。
used_space	bigint	已使用的存储空间大小，单位KB。
total_space	bigint	可使用的存储空间大小，单位KB，值为-1表示未限制最大存储空间。
used_temp_space	bigint	已使用的临时空间大小，单位KB
total_temp_space	bigint	可使用的临时空间总大小，单位KB，值为-1表示未限制。
used_spill_space	bigint	已使用的下盘空间大小。单位KB。
total_spill_space	bigint	可使用的下盘空间总大小，单位KB，值为-1表示未限制。
read_kbytes	bigint	读磁盘数据量，单位KB。
write_kbytes	bigint	写磁盘数据量，单位KB。
read_counts	bigint	读磁盘次数。
write_counts	bigint	写磁盘次数。
read_speed	double precision	读磁盘速率，单位B/ms。

名称	类型	描述
write_speed	double precision	写磁盘速率，单位B/ms。

12.3.158 PG_USER

PG_USER视图显示数据库用户的信息，默认只有初始化用户和具有sysadmin属性的用户可以查看，其余用户需要赋权后才可以查看。

表 12-248 PG_USER 字段

名称	类型	描述
usename	name	用户名。
usesysid	oid	用户的ID。
usecreatedb	boolean	用户是否可以创建数据库。 ● t（true）：表示是。 ● f（false）：表示否。
usesuper	boolean	用户是否是拥有最高权限的初始系统管理员。 ● t（true）：表示是。 ● f（false）：表示否。
usecatupd	boolean	用户是否可以直接更新系统表。只有usesysid=10的初始系统管理员拥有此权限。其他用户无法获得此权限。 ● t（true）：表示是。 ● f（false）：表示否。
userepl	boolean	用户是否可以复制数据流。 ● t（true）：表示是。 ● f（false）：表示否。
passwd	text	密文存储后的用户口令，始终为********。
valbegin	timestamp with time zone	账户的有效开始时间；如果没有设置有效开始时间，则为NULL。
valuntil	timestamp with time zone	账户的有效结束时间；如果没有设置有效结束时间，则为NULL。
respool	name	用户所在的资源池。
parent	oid	父用户OID。
spacelimit	text	永久表存储空间限额，单位KB。

名称	类型	描述
useconfig	text[]	运行时配置项的默认值。参考 PG_DB_ROLE_SETTING.setconfig。
nodegroup	name	用户关联的Node group名称，如果该用户没有管理Node group，则该字段为空。
tempspacelimit	text	临时表存储空间限额，单位KB。
spillspacelimit	text	算子落盘空间限额，单位KB。
usemonitoradmin	boolean	用户是否是监控管理员。 • t（true）：表示是。 • f（false）：表示否。
useoperatoradmin	boolean	用户是否是运维管理员。 • t（true）：表示是。 • f（false）：表示否。
usepolicyadmin	boolean	用户是否是安全策略管理员。 • t（true）：表示是。 • f（false）：表示否。

12.3.159 PG_USER_MAPPINGS

PG_USER_MAPPINGS视图显示用户映射的信息。所有用户均可查看。

表 12-249 PG_USER_MAPPINGS 字段

名称	类型	引用	描述
umid	oid	PG_USER_MAPPING.oid	用户映射的OID。
srvid	oid	PG_FOREIGN_SERVER.oid	包含这个映射的外部服务器的OID。
srvname	name	PG_FOREIGN_SERVER.srvname	外部服务器的名称。
umuser	oid	PG_AUTHID.oid	被映射的本地角色的OID，如果用户映射是公共的则为0。
usename	name	-	被映射的本地用户的名称。
umoptions	text[]	-	如果当前用户是外部服务器的所有者，则为用户映射指定选项， 使用“keyword=value”字符串，否则为null。

12.3.160 PG_VARIABLE_INFO

PG_VARIABLE_INFO视图用于查询集群中当前节点的xid、oid的状态。

表 12-250 PG_VARIABLE_INFO 字段

名称	类型	描述
node_name	text	节点名称。
next_oid	oid	该节点下一次生成的oid。
next_xid	xid	该节点下一次生成的事务号。
oldest_xid	xid	该节点最旧的事务号。
xid_vac_limit	xid	强制autovacuum的临界点。
oldest_xid_db	oid	该节点datafrozenxid最小的数据库oid。
last_extend_csn_logpage	xid	最后一次扩展csnlog的页面号。
start_extend_csn_logpage	xid	csnlog扩展的起始页面号。
next_commit_seqno	xid	该节点下次生成的csn号。
latest_completed_xid	xid	该节点提交或者回滚后节点上的最新事务号。
startup_max_xid	xid	该节点关机前的最后一个事务号。

12.3.161 PG_VIEWS

PG_VIEWS视图显示数据库中每个视图的有用信息。

表 12-251 PG_VIEWS 字段

名称	类型	引用	描述
schemaname	name	PG_NAMESPACE.nspname	视图的模式名。
viewname	name	PG_CLASS.relname	视图名。
viewowner	name	PG_AUTHID.Erolname	视图的所有者。
definition	text	-	视图的定义。

12.3.162 PGXC_COMM_DELAY

PGXC_COMM_DELAY视图展示所有DN的通信库时延状态，仅system admin和monitor admin可以查看。

表 12-252 PGXC_COMM_DELAY 字段

名称	类型	描述
node_name	text	节点名称。
remote_name	text	连接对端节点的对端节点名称。
remote_host	text	连接对端IP的对端地址。
stream_num	integer	当前物理连接使用的stream逻辑连接数量。
min_delay	integer	当前物理连接一分钟内探测到的最小时延，单位微秒。 **说明** 负数结果无效，请重新等待时延状态更新后再执行查询。
average	integer	当前物理连接一分钟内探测时延的平均值，单位微秒。
max_delay	integer	当前物理连接一分钟内探测到的最大时延，单位微秒。

12.3.163 PGXC_COMM_RECV_STREAM

PGXC_COMM_RECV_STREAM视图展示所有DN上的通信库接收流状态。

表 12-253 PGXC_COMM_RECV_STREAM 字段

名称	类型	描述
node_name	text	节点名称。
local_tid	bigint	使用此通信流的线程ID。
remote_name	text	连接对端节点名称。
remote_tid	bigint	连接对端线程ID。
idx	integer	通信对端DN在本DN内的标识编号。
sid	integer	通信流在物理连接中的标识编号。
tcp_sock	integer	通信流所使用的tcp通信socket。

名称	类型	描述
state	text	通信流当前的状态。 ● UNKNOWN：表示当前逻辑连接状态未知。 ● READY：表示逻辑连接已就绪。 ● RUN：表示逻辑连接发送报文正常。 ● HOLD：表示逻辑连接发送报文等待中。 ● CLOSED：表示关闭逻辑连接。 ● TO_CLOSED：表示将会关闭逻辑连接。
query_id	bigint	通信流对应的debug_query_id编号。
pn_id	integer	通信流所执行查询的plan_node_id编号。
send_smp	integer	通信流所执行查询send端的smpid编号。
recv_smp	integer	通信流所执行查询recv端的smpid编号。
recv_bytes	bigint	通信流接收的数据总量，单位Byte。
time	bigint	通信流当前生命周期使用时长，单位ms。
speed	bigint	通信流的平均接收速率，单位Byte/s。
quota	bigint	通信流当前的通信配额值，单位Byte。
buff_usize	bigint	通信流当前缓存的数据大小，单位Byte。

12.3.164 PGXC_COMM_SEND_STREAM

PGXC_COMM_SEND_STREAM视图展示所有DN上的通信库发送流状态。

表 12-254 PGXC_COMM_SEND_STREAM 字段

名称	类型	描述
node_name	text	节点名称。
local_tid	bigint	使用此通信流的线程ID。
remote_name	text	连接对端节点名称。
remote_tid	bigint	连接对端线程ID。
idx	integer	通信对端DN在本DN内的标识编号。
sid	integer	通信流在物理连接中的标识编号。
tcp_sock	integer	通信流所使用的tcp通信socket。

名称	类型	描述
state	text	通信流当前的状态。 状态值选择如下： ● UNKNOWN：表示当前连接状态未知。 ● READY：表示连接已就绪。 ● RUN：表示连接当前正常发送报文。 ● HOLD：表示连接正在等待报文发送。 ● CLOSED：表示连接关闭。 ● TO_CLOSED：表示将会关闭连接。
query_id	bigint	通信流对应的debug_query_id编号。
pn_id	integer	通信流所执行查询的plan_node_id编号。
send_smp	integer	通信流所执行查询send端的smpid编号。
recv_smp	integer	通信流所执行查询recv端的smpid编号。
send_bytes	bigint	通信流发送的数据总量，单位Byte。
time	bigint	通信流当前生命周期使用时长，单位ms。
speed	bigint	通信流的平均发送速率，单位Byte/s。
quota	bigint	通信流当前的通信配额值，单位Byte。
wait_quota	bigint	通信流等待quota值产生的额外时间开销，单位ms。

12.3.165 PGXC_COMM_STATUS

PGXC_COMM_STATUS视图展示所有DN的通信库状态。

表 12-255 PGXC_COMM_STATUS 字段

名称	类型	描述
node_name	text	节点名称。
rxpck_rate	integer	节点通信库接收速率，单位Byte/s。
txpck_rate	integer	节点通信库发送速率，单位Byte/s。
rxkbyte_rate	bigint	bigint节点通信库接收速率，单位KByte/s。
txkbyte_rate	bigint	bigint节点通信库发送速率，单位KByte/s。
buffer	bigint	cmailbox的buffer大小。
memkbyte_libcomm	bigint	libcomm进程通信内存大小，单位Byte。

名称	类型	描述
memkbyte_libpq	bigint	libpq进程通信内存大小，单位Byte。
used_pm	integer	postmaster线程实时使用率。
used_sflow	integer	gs_sender_flow_controller线程实时使用率。
used_rflow	integer	gs_receiver_flow_controller线程实时使用率。
used_rloop	integer	多个gs_receivers_loop线程中高的实时使用率。
stream	integer	当前使用的逻辑连接总数。

12.3.166 PGXC_GET_STAT_ALL_TABLES

PGXC_GET_STAT_ALL_TABLES视图显示各表的插入、更新、删除以及脏页率信息。对于高脏页率的系统表，建议在确认当前没有人操作该系统表时，再执行VACUUM FULL。此视图为GaussDB新增功能，升级到该版本后，升级前的插入、更新、删除信息不会被统计。建议对脏页率超过30%的非系统表执行VACUUM FULL，用户也可根据业务场景自行选择是否执行VACUUM FULL。该视图只有system admin和monitor admin用户有权限查看。

表 12-256 PGXC_GET_STAT_ALL_TABLES 字段

名称	类型	描述
relid	oid	表的oid。
relname	name	表名。
schemaname	name	表的模式名。
n_tup_ins	numeric	插入的元组条数。
n_tup_upd	numeric	更新的元组条数。
n_tup_del	numeric	删除的元组条数。
n_live_tup	numeric	存活元组的条数。
n_dead_tup	numeric	死亡元组的条数。
dirty_page_rate	numeric(5,2)	表的脏页率信息(%)。

12.3.167 PGXC_GET_TABLE_SKEWNESS

PGXC_GET_TABLE_SKEWNESS视图显示当前数据库中表的数据分布倾斜情况。只有系统管理员权限可以访问。

表 12-257 PGXC_GET_TABLE_SKEWNESS 字段

名称	类型	描述
schemaname	name	表所在的模式名。
tablename	name	表名。
totalsize	numeric	表的总大小，单位Byte。
avgsize	numeric(1000,0)	表大小平均值(totalsize/DN个数，该值为平均分布的理想情况下，表在各DN占用空间大小)。
maxratio	numeric(4,3)	单DN表大小最大值占比（表在各DN占用空间的最大值/totalsize）。
minratio	numeric(4,3)	单DN表大小最小值占比（表在各DN占用空间的最小值/totalsize）。
skewsize	bigint	表分布倾斜值（单DN表大小最大值 - 单DN表大小最小值）。
skewratio	numeric(4,3)	表分布倾斜率（skewsize/totalsize）。
skewstddev	numeric(1000,0)	表分布标准方差（在表大小一定的情况下，该值越大表明表的整体分布情况越倾斜）。

12.3.168 PGXC_NODE_ENV

PGXC_NODE_ENV视图显示集群中所有节点的环境变量信息。该视图只有monitor admin和sysadmin权限可以查看。

表 12-258 PGXC_NODE_ENV 字段

名称	类型	描述
node_name1	text	集群中节点的名称。
host1	text	集群中节点的主机名称。
process1	integer	集群中节点的进程号。
port1	integer	集群中节点的端口号。
installpath1	text	集群中节点的安装目录。
datapath1	text	集群中节点的数据目录。
log_directory1	text	集群中节点的日志目录。

12.3.169 PGXC_OS_THREADS

PGXC_OS_THREADS视图显示当前集群中所有正常节点下的线程状态信息。该视图只有monitor admin和sysadmin权限可以查看。

表 12-259 PGXC_OS_THREADS 字段

名称	类型	描述
node_name	text	当前集群中正常节点的名称。
pid	bigint	当前集群中正常节点进程中正在运行的线程号。
lwpid	integer	与pid对应的轻量级线程号。
thread_name	text	与pid对应的线程名称。
creation_time	timestamp with time zone	与pid对应的线程创建的时间。

12.3.170 PGXC_PREPARED_XACTS

PGXC_PREPARED_XACTS视图显示当前处于prepared阶段的两阶段事务。只有system admin和monitor admin用户有权限查看。

表 12-260 PGXC_PREPARED_XACTS 字段

名称	类型	描述
pgxc_prepared_xact	text	当前处于prepared阶段的两阶段事务。

12.3.171 PGXC_RUNNING_XACTS

PGXC_RUNNING_XACTS视图显示集群中各个节点运行事务的信息，字段内容和PG_RUNNING_XACTS相同。只有system admin和monitor admin用户有权限查看。

表 12-261 PGXC_RUNNING_XACTS 字段

名称	类型	描述
handle	integer	事务在GTM对应的句柄。
gxid	xid	事务id号。
state	tinyint	事务状态。 ● 3：prepared。 ● 0：starting。

名称	类型	描述
node	text	节点名称。
xmin	xid	节点上当前数据涉及的最小事务号xmin。
vacuum	boolean	表示当前事务是否是lazy vacuum事务。 • t（true）：表示是。 • f（false）：表示否。
timeline	bigint	数据库重启次数。
prepare_xid	xid	处于prepared状态事务的id号，若不在prepared状态，值为0。
pid	bigint	事务对应的线程id。
next_xid	xid	CN传给DN的事务id号。

12.3.172 PGXC_SQL_COUNT

PGXC_SQL_COUNT视图可用来查看SELECT、INSERT、UPDATE、DELETE、MERGE INTO五种SQL的节点级和用户级统计结果，识别当前业务负载较重的query类型，衡量整个集群和单个节点执行某种类型查询的能力。通过对以上五类SQL查询进行计数，获得指定时刻的统计结果，经计算可以得到指定QPS等统计信息。例如，T1时刻，USER1的SELECT计数结果为X1，T2时刻为X2，则可计算得到该用户SELECT查询的QPS值为(X2-X1)\(T2-T1)。由此，可获得集群用户级QPS曲线图和集群吞吐情况，追踪每个用户的业务负载是否发生剧烈变化。如果有剧烈变化，可以定位具体的语句类型(SELECT/INSERT/UPDATE/DELETE/MERGE INTO)。同时观测QPS曲线可以获知问题发生时间点，结合其它工具，定位问题点。能够为集群性能调优、问题定位等提供依据。该视图只有monitor admin和sysadmin权限可以查看。只能在CN上查询，不支持execute direct on (dn) 'select * from PGXC_SQL_COUNT';语句。

PGXC_SQL_COUNT视图的字段与GS_SQL_COUNT一致，具体请参见**表12-159**。

说明

当执行用户的MERGE INTO语句时，若能下推，在DN上收到的是MERGE INTO语句，将在DN节点上进行MERGE INTO类型计数，相应mergeinto_count列计数增加；若不能下推，在DN上收到的是UPDATE或INSERT语句，将在DN节点上进行UPDATE或INSERT类型计数，相应的update_count列或insert_count列计数增加。

12.3.173 PGXC_STAT_ACTIVITY

PGXC_STAT_ACTIVITY视图显示当前集群下所有CN的当前用户查询相关的信息，该视图只有monitor admin和sysadmin权限可以查看。

表 12-262 PGXC_STAT_ACTIVITY 字段

名称	类型	描述
coorname	text	当前集群下的CN名称。

名称	类型	描述
datid	oid	用户会话在后台连接到的数据库OID。
datname	text	用户会话在后台连接到的数据库名称。
pid	bigint	后台线程ID。
sessionid	bigint	会话ID。
usesysid	oid	登录该后台的用户OID。
usename	text	登录该后台的用户名。
application_name	text	连接到该后台的应用名。
client_addr	inet	连接到该后台的客户端的IP地址。 如果此字段是null，它表明通过服务器机器上UNIX套接字连接客户端或者这是内部进程，如autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过client_addr的反向DNS查找得到。这个字段只有在启动log_hostname且使用IP连接时才非空。
client_port	integer	客户端用于与后台通讯的TCP端口号，如果使用UNIX套接字，则为-1。
backend_start	timestamp with time zone	该过程开始的时间，即当客户端连接服务器的时间。
xact_start	timestamp with time zone	启动当前事务的时间，如果没有事务是活跃的，则为null。如果当前查询是首个事务，则这列等同于query_start列。
query_start	timestamp with time zone	开始当前活跃查询的时间， 如果state的值不是active，则这个值是上一个查询的开始时间。
state_change	timestamp with time zone	上次状态改变的时间。
waiting	boolean	如果后台当前正等待锁则为true。否则为false。
enqueue	text	语句当前排队状态。可能值是： ● waiting in queue：表示语句在排队中。 ● 空：表示语句正在运行。

名称	类型	描述
state	text	该后台当前总体状态。可能值是： ● active：后台正在执行一个查询。 ● idle：后台正在等待一个新的客户端命令。 ● idle in transaction：后台在事务中，但事务中没有语句在执行。 ● idle in transaction (aborted)：后台在事务中，但事务中有语句执行失败。 ● fastpath function call：后台正在执行一个fast-path函数。 ● disabled：如果后台禁用track_activities，则报告这个状态。 **说明** 只有系统管理员能查看到自己账户所对应的会话状态。其他账户的state信息为空。例如以judy用户连接数据库后，在pgxc_stat_activity中查看到的普通用户joe及初始用户omm的state信息为空： SELECT datname, usename, usesysid, state,pid FROM pgxc_stat_activity; datname | usename | usesysid | state | pid ----------+---------+----------+--------+----------------- testdb | omm | 10 | | 139968752121616 testdb | omm | 10 | | 139968903116560 db_tpcds | judy | 16398 | active | 139968391403280 testdb | omm | 10 | | 139968643069712 testdb | omm | 10 | | 139968680818448 testdb | joe | 16390 | | 139968563377936 (6 rows)
resource_pool	name	用户使用的资源池。
query_id	bigint	查询语句的ID。
query	text	该后台的最新查询。如果state状态是active（活跃的），此字段显示当前正在执行的查询。所有其他情况表示上一个查询。
connection_info	text	json格式字符串，记录当前连接数据库的驱动类型、驱动版本号、当前驱动的部署路径、进程属主用户等信息。
global_sessionid	text	全局会话ID。
unique_sql_id	bigint	语句的unique sql id。

名称	类型	描述
trace_id	text	驱动传入的trace id，与应用的一次请求相关联。

12.3.174 PGXC_STAT_BAD_BLOCK

PGXC_STAT_BAD_BLOCK视图显示集群所有节点从启动后，在读取数据时出现Page/CU校验失败的统计信息。该视图只有monitor admin和sysadmin权限可以查看。

表 12-263 PGXC_STAT_BAD_BLOCK 字段

名称	类型	描述
nodename	text	节点名。
databaseid	integer	数据库OID。
tablespaceid	integer	表空间OID。
relfilenode	integer	文件对象ID。
forknum	integer	文件类型。取值如下： ● 0：数据主文件。 ● 1：FSM文件。 ● 2：VM文件。 ● 3：BCM文件。
error_count	integer	出现校验失败的次数。
first_time	timestamp with time zone	第一次出现的时间。
last_time	timestamp with time zone	最近一次出现的时间。

12.3.175 PGXC_THREAD_WAIT_STATUS

PGXC_THREAD_WAIT_STATUS视图可用来查看集群全局各个节点上所有SQL语句产生的线程之间的调用层次关系，以及各个线程的阻塞等待状态，从而更容易定位进程停止响应问题以及类似现象的原因。

PGXC_THREAD_WAIT_STATUS视图和PG_THREAD_WAIT_STATUS视图列定义完全相同，这是由于PGXC_THREAD_WAIT_STATUS视图本质是到集群中各个节点上查询PG_THREAD_WAIT_STATUS视图汇总的结果。

表 12-264 PGXC_THREAD_WAIT_STATUS 字段

名称	类型	描述
node_name	text	当前节点的名称。
db_name	text	数据库名称。
thread_name	text	线程名称。
query_id	bigint	查询ID，对应debug_query_id。
tid	bigint	当前线程的线程号。
sessionid	bigint	会话ID。
lwtid	integer	当前线程的轻量级线程号。
psessionid	bigint	父会话ID。
tlevel	integer	streaming线程的层级。
smpid	integer	并行线程的ID。
wait_status	text	当前线程等待状态的详细信息。
wait_event	text	当前线程正在等待的事件。
locktag	text	当前线程正在等待锁的信息。
lockmode	text	当前线程正等待获取的锁模式。
block_sessionid	bigint	阻塞当前线程获取锁的会话标识。
global_sessionid	text	全局会话ID。

例如：

在coordinator1执行一条语句之后长时间没有响应。可以创建另外一个连接到coordinator1上，查询coordinator1上的线程状态。

```
openGauss=# SELECT * FROM pg_thread_wait_status WHERE query_id > 0;
  node_name   | db_name  | thread_name  | query_id |       tid       | lwtid | ptid  | tlevel | smpid |
wait_status   |    wait_event
--------------+----------+--------------+----------+-----------------+-------+-------+--------+-------
+----------------------
 coordinator1 | postgres | gsql         | 20971544 | 140274089064208 | 22579 |       |      0 |     0 | wait node:
datanode4 |
(1 rows)
openGauss=# SELECT * FROM pgxc_thread_wait_status WHERE query_id > 0;
  node_name   | db_name  | thread_name  |     query_id      |       tid       |   sessionid    | lwtid  | psessionid |
tlevel | smpid | wait_status | wait_event
--------------+----------+--------------+-------------------+-----------------+----------------+--------+------------
+--------+-------+-------------+------------
 coordinator1 | postgres | gsql         | 77687093572155050 | 47212704827136 | 47212704827136 |  63191
|            |      0 |     0 | none        |
 coordinator2 | postgres | coordinator1 | 77687093572155050 | 47403117319936 | 47403117319936 |
159322 |            |      0 |     0 | none        |
 data_node1   | postgres | coordinator1 | 77687093572155050 | 47723869374208 | 47723869374208 |
159320 |            |      0 |     0 | none        |
 data_node2   | postgres | coordinator1 | 77687093572155050 | 47852867290880 | 47852867290880 |
159321 |            |      0 |     0 | none        |
(4 rows)
```

12.3.176 PGXC_TOTAL_MEMORY_DETAIL

PGXC_TOTAL_MEMORY_DETAIL视图显示集群内存使用情况。只能在CN上查询，不支持execute direct on (dn) 'select * from PGXC_TOTAL_MEMORY_DETAIL';语句。该视图只有sysadmin和monitor admin权限可以查看。

表 12-265 PGXC_TOTAL_MEMORY_DETAIL 字段

名称	类型	描述
nodename	text	节点名称。
memorytype	text	内存使用的名称。 ● max_process_memory：GaussDB集群实例所占用的内存大小。 ● process_used_memory：GaussDB进程所使用的内存大小。 ● max_dynamic_memory：最大动态内存。 ● dynamic_used_memory：已使用的动态内存。 ● dynamic_peak_memory：内存的动态峰值。 ● dynamic_used_shrctx：最大动态共享内存上下文。 ● dynamic_peak_shrctx：共享内存上下文的动态峰值。 ● max_shared_memory：最大共享内存。 ● shared_used_memory：已使用的共享内存。 ● max_sctpcomm_memory：通信库所允许使用的最大内存。 ● sctpcomm_used_memory：通信库已使用的内存大小。 ● sctpcomm_peak_memory：通信库的内存峰值。 ● other_used_memory：其他已使用的内存大小。
memorymbytes	integer	内存使用的大小，单位为MB。

12.3.177 PGXC_VARIABLE_INFO

PGXC_VARIABLE_INFO视图用于查询集群中所有节点的xid、oid的状态。该视图只有monitor admin和sysadmin权限可以查看。只能在CN上查询，不支持execute direct on (dn) 'select * from PGXC_VARIABLE_INFO';语句。

表 12-266 PGXC_VARIABLE_INFO 字段

名称	类型	描述
node_name	text	节点名称。
next_oid	oid	该节点下一次生成的oid。
next_xid	xid	该节点下一次生成的事务号。
oldest_xid	xid	该节点最老的事务号。
xid_vac_limit	xid	强制autovacuum的临界点。
oldest_xid_db	oid	该节点datafrozenxid最小的数据库oid。
last_extend_csn_logpage	xid	最后一次扩展csnlog的页面号。
start_extend_csn_logpage	xid	csnlog扩展的起始页面号。
next_commit_seqno	xid	该节点下次生成的csn号。
latest_completed_xid	xid	该节点提交或者回滚后节点上的最新事务号。
startup_max_xid	xid	该节点关机前的最后一个事务号。

12.3.178 PLAN_TABLE

PLAN_TABLE显示用户通过执行EXPLAIN PLAN收集到的计划信息。计划信息的生命周期是session级别，session退出后相应的数据将被清除。同时不同session和不同user间的数据是相互隔离的。

表 12-267 PLAN_TABLE 字段

名称	类型	描述
statement_id	character varying(30)	用户输入的查询标签。
plan_id	bigint	查询标识。
id	int	查询生成的计划中的每一个执行算子的编号。
operation	character varying(30)	计划中算子的操作描述。
options	character varying(255)	操作选项。
object_name	name	操作对应的对象名，非查询中使用到的对象别名。来自于用户定义。

名称	类型	描述
object_type	character varying(30)	对象类型。
object_owner	name	对象所属schema，来自于用户定义。
projection	character varying(4000)	操作输出的列信息。
cost	float8	优化器对算子估算的执行代价。
cardinality	float8	优化器对算子估算的结果行数。

说明

- object_type取值范围为PG_CLASS中定义的relkind类型（TABLE普通表，INDEX索引，SEQUENCE序列，VIEW视图，COMPOSITE TYPE复合类型，TOASTVALUE TOAST表）和计划使用到的rtekind(SUBQUERY, JOIN, FUNCTION, VALUES, CTE, REMOTE_QUERY)。
- object_owner对于RTE来说是计划中使用的对象描述，非用户定义的类型不存在object_owner。
- statement_id、object_name、object_owner、projection字段内容遵循用户定义的大小写存储，其它字段内容采用大写存储。
- 支持用户对PLAN_TABLE进行SELECT和DELETE操作，不支持其它DML操作。

12.3.179 PV_FILE_STAT

PV_FILE_STAT视图通过对数据文件I/O的统计，反映数据的I/O性能，用以发现I/O操作异常等性能问题。

表 12-268 PV_FILE_STAT 字段

名称	类型	描述
filenum	oid	文件标识。
dbid	oid	数据库标识。
spcid	oid	表空间标识。
phyrds	bigint	读物理文件的数目。
phywrts	bigint	写物理文件的数目。
phyblkrd	bigint	读物理文件块的数目。
phyblkwrt	bigint	写物理文件块的数目。
readtim	bigint	读文件的总时长，单位微秒。
writetim	bigint	写文件的总时长，单位微秒。
avgiotim	bigint	读写文件的平均时长，单位微秒。

名称	类型	描述
lstiotim	bigint	最后一次读文件时长，单位微秒。
miniotim	bigint	读写文件的最小时长，单位微秒。
maxiowtm	bigint	读写文件的最大时长，单位微秒。

12.3.180 PV_INSTANCE_TIME

提供当前节点下的各种时间消耗信息，主要分为以下类型:

- DB_TIME: 作业在多核下的有效时间花销。
- CPU_TIME：CPU的时间花销。
- EXECUTION_TIME：执行器内的时间花销。
- PARSE_TIME：SQL解析的时间花销。
- PLAN_TIME：生成Plan的时间花销。
- REWRITE_TIME：SQL重写的时间花销。
- PL_EXECUTION_TIME ： PL/SQL（存储过程）执行的时间花销.
- PL_COMPILATION_TIME：PL/SQL（存储过程）编译的时间花销。
- NET_SEND_TIME：网络上的时间花销。
- DATA_IO_TIME：I/O上的时间花销。

表 12-269 PV_INSTANCE_TIME 字段

名称	类型	描述
stat_id	integer	统计编号。
stat_name	text	类型名称。
value	bigint	时间值(单位：微秒)。

12.3.181 PV_OS_RUN_INFO

PV_OS_RUN_INFO视图显示当前操作系统运行的状态信息。

表 12-270 PV_OS_RUN_INFO 字段

名称	类型	描述
id	integer	编号。
name	text	操作系统运行状态名称。
value	numeric	操作系统运行状态值。
comments	text	操作系统运行状态注释。

名称	类型	描述
cumulative	boolean	操作系统运行状态的值是否为累加值。

12.3.182 PV_REDO_STAT

PV_REDO_STAT视图显示会话线程的日志回放情况。

表 12-271 PV_REDO_STAT 字段

名称	类型	描述
phywrts	bigint	日志回放过程中写数据的次数。
phyblkwrt	bigint	日志回放过程中写数据的块数。
writetim	bigint	日志回放过程中写数据消耗的总时间。
avgiotim	bigint	日志回放过程中写一次数据平均消耗的时间。
lstiotim	bigint	日志回放过程中最后一次写数据消耗的时间。
miniotim	bigint	日志回放过程中单次写数据消耗的最短时间。
maxiowtm	bigint	日志回放过程中单次写数据消耗的最长时间。

12.3.183 PV_SESSION_MEMORY

PV_SESSION_MEMORY视图显示Session级别的内存使用情况，包含执行作业在数据节点上gaussdb线程和Stream线程分配的所有内存。

表 12-272 PV_SESSION_MEMORY 字段

名称	类型	描述
sessid	text	线程启动时间+线程标识。
init_mem	integer	当前正在执行作业进入执行器前已分配的内存，单位MB。
used_mem	integer	当前正在执行作业已分配的内存，单位MB。
peak_mem	integer	当前正在执行作业已分配的内存峰值，单位MB。

12.3.184 PV_SESSION_MEMORY_CONTEXT

PV_SESSION_MEMORY_CONTEXT视图显示所有会话的内存使用情况，以MemoryContext节点来统计。该视图仅在开启线程池（enable_thread_pool = on）时生效。

其中内存上下文“TempSmallContextGroup”，记录当前线程中所有内存上下文字段“totalsize”小于8192字节的信息汇总，并且内存上下文统计计数记录到“usedsize”字段中。所以在视图中，“TempSmallContextGroup”内存上下文中的“totalsize”和“freesize”是该线程中所有内存上下文“totalsize”小于8192字节的汇总总和，usedsize字段表示统计的内存上下文个数。

表 12-273 PV_SESSION_MEMORY_CONTEXT 字段

名称	类型	描述
sessid	text	会话启动时间+会话标识（字符串信息为timestamp.sessionid）。
threadid	bigint	会话绑定的线程标识，如果未绑定线程，该值为-1。
contextname	text	内存上下文名称。
level	smallint	当前上下文在整体内存上下文中的层级。
parent	text	父内存上下文名称。
totalsize	bigint	当前内存上下文的内存总数，单位Byte。
freesize	bigint	当前内存上下文中已释放的内存总数，单位Byte。
usedsize	bigint	当前内存上下文中已使用的内存总数，单位Byte；“TempSmallContextGroup”内存上下文中该字段含义为统计计数。

12.3.185 PV_SESSION_MEMORY_DETAIL

PV_SESSION_MEMORY_DETAIL视图显示会话的内存使用情况，以MemoryContext节点来统计。当开启线程池（enable_thread_pool = on）时，该视图包含所有的线程和会话的内存使用情况。

其中内存上下文“TempSmallContextGroup”，记录当前线程中所有内存上下文字段“totalsize”小于8192字节的信息汇总，并且内存上下文统计计数记录到“usedsize”字段中。所以在视图中，“TempSmallContextGroup”内存上下文中的“totalsize”和“freesize”是该线程中所有内存上下文“totalsize”小于8192字节的汇总总和，usedsize字段表示统计的内存上下文个数。

可通过"select * from pv_session_memctx_detail(threadid, '');"将某个线程所有内存上下文信息记录到“$GAUSSLOG/pg_log/${node_name}/dumpmem”目录下的“threadid_timestamp.log”文件中。其中threadid可通过下表sessid中获得。

表 12-274 PV_SESSION_MEMORY_DETAIL 字段

名称	类型	描述
sessid	text	1. 关闭线程池（enable_thread_pool = off）时该字段表示线程启动时间+session标识（字符串信息为timestamp.sessionid）。 2. 开启线程池（enable_thread_pool = on）时，内存上下文是线程级别的，则对应的该字段表示线程启动时间+线程标识（字符串信息为timestamp.threadid），内存上下文是session级别的，则对应的该字段表示线程启动时间+session标识（字符串信息为timestamp.sessionid）。
sesstype	text	线程名称。
contextname	text	内存上下文名称。
level	smallint	当前上下文在整体内存上下文中的层级。
parent	text	父内存上下文名称。
totalsize	bigint	当前内存上下文的内存总数，单位Byte。
freesize	bigint	当前内存上下文中已释放的内存总数，单位Byte。
usedsize	bigint	当前内存上下文中已使用的内存总数，单位Byte；“TempSmallContextGroup”内存上下文中该字段含义为统计计数。

12.3.186 PV_SESSION_STAT

PV_SESSION_STAT视图显示以会话线程或AutoVacuum线程为单位的统计会话状态信息。

表 12-275 PV_SESSION_STAT 字段

名称	类型	描述
sessid	text	线程标识+线程启动时间。
statid	integer	统计编号。
statname	text	统计会话名称。
statunit	text	统计会话单位。
value	bigint	统计会话值。

12.3.187 PV_SESSION_TIME

PV_SESSION_TIME视图显示会话线程的运行时间信息及各执行阶段所消耗的时间。

表 12-276 PV_SESSION_TIME 字段

名称	类型	描述
sessid	text	线程标识+线程启动时间。
stat_id	integer	统计编号。
stat_name	text	会话类型名称。 ● DB_TIME：作业在多核下的有效时间花费。 ● CPU_TIME：CPU时间的消耗。 ● EXECUTION_TIME：执行器内花费的时间。 ● PARSE_TIME：SQL解析的时间花费。 ● PLAN_TIME：生成Plan的时间花费。 ● REWRITE_TIME：SQL重写的时间消耗。 ● PL_EXECUTION_TIME：PL/SQL（存储过程）的执行时间。 ● PL_COMPILATION_TIME：PL/SQL（存储过程）编译时间。 ● NET_SEND_TIME：网络上的时间花销。 ● DATA_IO_TIME：I/O时间上的花销。
value	bigint	会话值。

12.3.188 PV_THREAD_MEMORY_CONTEXT

PV_THREAD_MEMORY_CONTEXT视图显示所有线程的内存使用情况，以MemoryContext节点来统计。该视图在关闭线程池（enable_thread_pool = off）时等价于PV_SESSION_MEMORY_DETAIL视图。

其中内存上下文“TempSmallContextGroup”，记录当前线程中所有内存上下文字段“totalsize”小于8192字节的信息汇总，并且内存上下文统计计数记录到“usedsize”字段中。所以在视图中，“TempSmallContextGroup”内存上下文中的“totalsize”和“freesize”是该线程中所有内存上下文“totalsize”小于8192字节的汇总总和，usedsize字段表示统计的内存上下文个数。

表 12-277 PV_THREAD_MEMORY_CONTEXT 字段

名称	类型	描述
threadid	text	线程启动时间+线程标识（字符串信息为timestamp.sessionid）。
tid	bigint	线程标识。

名称	类型	描述
thrdtype	text	线程类型。可以是系统内存在的任何线程类型，如wlmmonitor等。
contextname	text	内存上下文名称。
level	smallint	当前上下文在整体内存上下文中的层级。
parent	text	父内存上下文名称。
totalsize	bigint	当前内存上下文的内存总数，单位Byte。
freesize	bigint	当前内存上下文中已释放的内存总数，单位Byte。
usedsize	bigint	当前内存上下文中已使用的内存总数，单位Byte；“TempSmallContextGroup”内存上下文中该字段含义为统计计数。

12.3.189 PV_TOTAL_MEMORY_DETAIL

PV_TOTAL_MEMORY_DETAIL视图显示当前数据库节点使用内存的信息，单位为MB。

表 12-278 PV_TOTAL_MEMORY_DETAIL 字段

名称	类型	描述
nodename	text	节点名称。

名称	类型	描述
memorytype	text	内存类型，包括以下几种： • max_process_memory：GaussDB集群实例所占用的内存大小。 • process_used_memory：GaussDB进程所使用的内存大小。 • max_dynamic_memory：最大动态内存。 • dynamic_used_memory：已使用的动态内存。 • dynamic_peak_memory：内存的动态峰值。 • dynamic_used_shrctx：最大动态共享内存上下文。 • dynamic_peak_shrctx：共享内存上下文的动态峰值。 • max_backend_memory：使用HA端口执行业务可使用的最大内存上限。 • backend_used_memory：使用HA端口执行业务已使用的内存。 • max_shared_memory：最大共享内存。 • shared_used_memory：已使用的共享内存。 • max_sctpcomm_memory：通信库所允许使用的最大内存。 • sctpcomm_used_memory：通信库已使用的内存大小。 • sctpcomm_peak_memory：通信库的内存峰值。 • other_used_memory：其他已使用的内存大小。
memorymbytes	integer	内存类型分配内存的大小。

12.3.190 SYS_DUMMY

SYS_DUMMY视图是数据库根据数据字典自动创建的，它只有一个文本字段，且只有一行，用于保存表达式计算结果。任何用户都可以访问它。该视图同时存在于PG_CATALOG和SYS schema下。

表 12-279 SYS_DUMMY 字段

名称	类型	描述
dummy	text	表达式计算结果。

13 Schema

GaussDB的Schema如下表所示。

表 13-1 GaussDB 支持的 Schema

Schema名称	描述
dbe_perf	DBE_PERF Schema内视图主要用来诊断性能问题，也是WDR Snapshot的数据来源。数据库安装后，默认只有初始用户和监控管理员具有模式dbe_perf的权限，有权查看该模式下的视图和函数。
snapshot	用于管理WDR snapshot的相关的数据信息，默认初始化用户或监控管理员用户可以访问。
sqladvsior	用于分布列推荐，具体使用方法见**分布列推荐函数**。
sys	用于提供系统信息视图接口。
pg_catalog	用于维护系统的catalog信息，包含系统表和所有内置数据类型、函数、操作符。
pg_toast	用于存储大对象（系统内部使用）。
public	公共模式，用于存储公共对象。search_path参数缺省时， 如果存在用户同名的模式则将创建的表（以及其他对象）默认创建到同名模式下，不存在用户同名模式则自动放入public模式。
pkg_service	用于管理package服务相关信息。
pkg_util	用于管理package工具相关信息。
dbe_raw	高级功能包dbe_raw，用于raw类型数据的转化、取子串、求长度等操作。
dbe_session	高级功能包dbe_session，用于设置指定属性的值，并支持用户查询校验。
dbe_lob	高级功能包dbe_lob，用于大文件（clob/blob）的读取、写入、复制等操作。
dbe_match	高级功能包dbe_match，用于字符串相似度比较。

Schema名称	描述
dbe_task	高级功能包dbe_task，用于作业任务的调度包括提交任务、取消任务、同步任务状态、更新任务信息等可以使数据库定期执行特定的任务。
dbe_sql	高级功能包dbe_sql，用于执行动态sql，可以在应用的运行时间构建查询和其它的命令。
dbe_file	高级功能包dbe_file，用于数据库外部文件的读取、复制、写入、删除、重命名等。
dbe_output	高级功能包dbe_output，用于打印输出信息。
dbe_random	高级功能包dbe_random，用于生成随机种子和随机数。
dbe_application_info	高级功能包dbe_application_info，用于记录客户端信息。
dbe_utility	高级功能包dbe_utility，用于存储过程调用调试工具，例如打印错误堆栈等。
dbe_scheduler	高级功能包dbe_scheduler，用于创建定时任务，通过程序(program)和调度(schedule)使数据库定期执行特定的任务。也可以通过授权、提供证书执行数据库外部任务。
information_schema	用于存储有关当前数据库中定义的对象的信息。

表 13-2 GaussDB 目前禁用的 Schema

Schema名称	描述
dbe_pldebugger	用于调试PL/SQL函数及存储过程，目前暂不支持，该视图下接口调用报错unsupported。
db4ai	用于管理AI训练中不同版本的数据信息。
dbe_pldeveloper	用户存储过程编译调试。
dbe_sql_util	用于语句补丁的管理。

以下接口在分布式部署形态下功能暂不支持：

- bool dbe_sql_util.create_hint_sql_patch(name, bigint, text, text DEFAULT NULL::text, boolean DEFAULT true)
- bool dbe_sql_util.create_abort_sql_patch(name, bigint, text DEFAULT NULL::text, boolean DEFAULT true)
- bool dbe_sql_util.drop_sql_patch(name)
- bool dbe_sql_util.enable_sql_patch(name)

- bool dbe_sql_util.disable_sql_patch(name)
- record dbe_sql_util.show_sql_patch(patch_name name, OUT unique_sql_id bigint, OUT enable boolean, OUT abort boolean, OUT hint_str text)

13.1 Information Schema

信息模式本身是一个名为information_schema的模式。这个模式自动存在于所有数据库中。信息模式由一组视图构成，它们包含定义在当前数据库中对象的信息。这个模式的拥有者是初始数据库用户，但是所有用户仅有使用权限，没有创建表、函数等对象的权限。

信息模式兼容PG，包括：constraint_table_usage、domain_constraints、domain_udt_usage、domains、enabled_roles、key_column_usage、parameters、referential_constraints、applicable_roles、administrable_role_authorizations、attributes、character_sets、check_constraint_routine_usage、check_constraints、collations、collation_character_set_applicability、column_domain_usage、column_privileges、column_udt_usage、columns、constraint_column_usage、role_column_grants、routine_privileges、role_routine_grants、routines、schemata、sequences、table_constraints、table_privileges、role_table_grants、tables、triggered_update_columns、triggers、udt_privileges、role_udt_grants、usage_privileges、role_usage_grants、user_defined_types、view_column_usage、view_routine_usage、view_table_usage、views、data_type_privileges、element_types、column_options、foreign_data_wrapper_options、foreign_data_wrappers、foreign_server_options、foreign_servers、foreign_table_options、foreign_tables、user_mapping_options、user_mappings、sql_features、sql_implementation_info、sql_languages、sql_packages、sql_parts、sql_sizing、sql_sizing_profiles请参考上述链接描述。

下面章节只显示未在上述描述内的视图信息。

13.1.1 _PG_FOREIGN_DATA_WRAPPERS

显示外部数据封装器的信息。该视图只有sysadmin权限可以查看。

表 13-3 _PG_FOREIGN_DATA_WRAPPERS 字段

名称	类型	描述
oid	oid	外部数据封装器的oid。
fdwowner	oid	外部数据封装器的所有者的oid。
fdwoptions	text[]	外部数据封装器指定选项，使用“keyword=value”格式的字符串。
foreign_data_wrapper_catalog	information_schema.sql_identifier	外部封装器所在的数据库名称(永远为当前数据库)。
foreign_data_wrapper_name	information_schema.sql_identifier	外部数据封装器名称。

authorization_identifier	information_schema.sql_identifier	外部数据封装器所有者的角色名称。
foreign_data_wrapper_language	information_schema.character_data	外部数据封装器的实现语言。

13.1.2 _PG_FOREIGN_SERVERS

显示外部服务器的信息。该视图只有sysadmin权限可以查看。

表 13-4 _PG_FOREIGN_SERVERS 字段

名称	类型	描述
oid	oid	外部服务器的oid。
srvoptions	text[]	外部服务器指定选项，使用“keyword=value”格式的字符串。
foreign_server_catalog	information_schema.sql_identifier	外部服务器所在database名称(永远为当前数据库)。
foreign_server_name	information_schema.sql_identifier	外部服务器名称。
foreign_data_wrapper_catalog	information_schema.sql_identifier	外部数据封装器所在database名称(永远为当前数据库)。
foreign_data_wrapper_name	information_schema.sql_identifier	外部数据封装器名称。
foreign_server_type	information_schema.character_data	外部服务器的类型。
foreign_server_version	information_schema.character_data	外部服务器的版本。
authorization_identifier	information_schema.sql_identifier	外部服务器的所有者的角色名称。

13.1.3 _PG_FOREIGN_TABLE_COLUMNS

显示外部表的列信息。该视图只有sysadmin权限可以查看。

表 13-5 _PG_FOREIGN_TABLE_COLUMNS 字段

名称	类型	描述
nspname	name	schema名称。

relname	name	表名称。
attname	name	列名称。
attfdwoptions	text[]	外部数据封装器的属性选项，使用“keyword=value”格式的字符串。

13.1.4 _PG_FOREIGN_TABLES

存储所有的定义在本数据库的外部表信息。只显示当前用户有权访问的外部表信息。该视图只有sysadmin权限可以查看。

表 13-6 _PG_FOREIGN_TABLES 字段

名称	类型	描述
foreign_table_catalog	information_schema.sql_identifier	外部表所在的数据库名称(永远是当前数据库)。
foreign_table_schema	name	外部表的schema名称。
foreign_table_name	name	外部表的名称。
ftoptions	text[]	外部表的可选项。
foreign_server_catalog	information_schema.sql_identifier	外部服务器所在的数据库名称(永远是当前数据库)。
foreign_server_name	information_schema.sql_identifier	外部服务器的名称。
authorization_identifier	information_schema.sql_identifier	所有者的角色名称。

13.1.5 _PG_USER_MAPPINGS

存储从本地用户到远程的映射。该视图只有sysadmin权限可以查看。

表 13-7 _PG_USER_MAPPINGS 字段

名称	类型	描述
oid	oid	从本地用户到远程的映射的oid。
umoptions	text[]	用户映射指定选项，使用"keyword=value”格式的字符串。

umuser	oid	被映射的本地用户的oid，如果用户映射是公共的则为0。
authorization_identifier	information_schema.sql_identifier	本地用户角色名称。
foreign_server_catalog	information_schema.sql_identifier	外部服务器定义所在的database名称。
foreign_server_name	information_schema.sql_identifier	外部服务器名称。
srvowner	information_schema.sql_identifier	外部服务器所有者。

13.1.6 INFORMATION_SCHEMA_CATALOG_NAME

用来显示当前所在的database的名称。

表 13-8 INFORMATION_SCHEMA_CATALOG_NAME 字段

名称	类型	描述
catalog_name	information_schema.sql_identifier	当前database的名称。

13.2 DBE_PERF Schema

DBE_PERF Schema内视图主要用来诊断性能问题，也是WDR Snapshot的数据来源。数据库安装后，默认只有初始用户和监控管理员具有模式dbe_perf的权限，有权查看该模式下的视图和函数。若是由老版本升级而来，为保持权限的前向兼容，模式dbe_perf的权限与老版本保持一致。从OS、Instance、Memory等多个维度划分组织视图，并且符合如下命名规范：

- GLOBAL_开头的视图，代表从CN/DN请求数据，并将数据追加对外返回，不会处理数据。
- SUMMARY_开头的视图，代表是将集群内的数据概述，多数情况下是返回CN/DN(有时只有CN的)的数据，会对数据进行加工和汇聚。
- 非这两者开头的视图，一般代表本地视图，不会向其它CN/DN请求数据。

13.2.1 OS

13.2.1.1 OS_RUNTIME

显示当前操作系统运行的状态信息。

表 13-9 OS_RUNTIME 字段

名称	类型	描述
id	integer	编号。
name	text	操作系统运行状态名称。
value	numeric	操作系统运行状态值。
comments	text	操作系统运行状态注释。
cumulative	boolean	操作系统运行状态的值是否为累加值。

13.2.1.2 GLOBAL_OS_RUNTIME

提供整个集群中所有正常节点下的操作系统运行状态信息。

表 13-10 GLOBAL_OS_RUNTIME 字段

名称	类型	描述
node_name	name	节点名称。
id	integer	编号。
name	text	操作系统运行状态名称。
value	numeric	操作系统运行状态值。
comments	text	操作系统运行状态注释。
cumulative	boolean	操作系统运行状态的值是否为累加值。

13.2.1.3 OS_THREADS

提供当前节点下所有线程的状态信息。

表 13-11 OS_THREADS 字段

名称	类型	描述
node_name	text	当前节点的名称。
pid	bigint	当前节点进程中正在运行的线程号。
lwpid	integer	与pid对应的轻量级线程号。
thread_name	text	与pid对应的线程名称。
creation_time	timestamp with time zone	与pid对应的线程创建的时间。

13.2.1.4 GLOBAL_OS_THREADS

提供整个集群中所有正常节点下的线程状态信息。

表 13-12 GLOBAL_OS_THREADS 字段

名称	类型	描述
node_name	text	当前节点的名称。
pid	bigint	当前节点进程中正在运行的线程号。
lwpid	integer	与pid对应的轻量级线程号。
thread_name	text	与pid对应的线程名称。
creation_time	timestamp with time zone	与pid对应的线程创建的时间。

13.2.2 Instance

13.2.2.1 INSTANCE_TIME

提供当前集群节点下的各种时间消耗信息，主要分为以下类型：

- DB_TIME：作业在多核下的有效时间花费。
- CPU_TIME：CPU时间的消耗。
- EXECUTION_TIME：执行器内花费的时间。
- PARSE_TIME：SQL解析的时间花费。
- PLAN_TIME：生成Plan的时间花费。
- REWRITE_TIME：SQL重写的时间消耗。
- PL_EXECUTION_TIME ： PL/SQL（存储过程）的执行时间。
- PL_COMPILATION_TIME：PL/SQL（存储过程）编译时间。
- NET_SEND_TIME：网络上的时间花销。
- DATA_IO_TIME：I/O时间上的花销。

表 13-13 INSTANCE_TIME 字段

名称	类型	描述
stat_id	integer	统计编号。
stat_name	text	类型名称。
value	bigint	时间值（单位：微秒）。

13.2.2.2 GLOBAL_INSTANCE_TIME

提供整个集群中所有正常节点下的各种时间消耗信息(时间类型见instance_time视图)。

表 13-14 GLOBAL_INSTANCE_TIME 字段

名称	类型	描述
node_name	name	节点的名称。
stat_id	integer	统计编号。
stat_name	text	类型名称。
value	bigint	时间值（单位：微秒）。

13.2.3 Memory

13.2.3.1 MEMORY_NODE_DETAIL

显示某个数据库节点内存使用情况。

表 13-15 MEMORY_NODE_DETAIL 字段

名称	类型	描述
nodename	text	节点名称。

名称	类型	描述
memorytype	text	内存的名称。 ● max_process_memory：GaussDB实例所占用的内存大小。 ● process_used_memory：进程所使用的内存大小。 ● max_dynamic_memory：最大动态内存。 ● dynamic_used_memory：已使用的动态内存。 ● dynamic_peak_memory：内存的动态峰值。 ● dynamic_used_shrctx：最大动态共享内存上下文。 ● dynamic_peak_shrctx：共享内存上下文的动态峰值。 ● max_shared_memory：最大共享内存。 ● shared_used_memory：已使用的共享内存。 ● max_sctpcomm_memory：TCP代理通信所允许使用的最大内存。 ● sctpcomm_used_memory：TCP代理通信已使用的内存大小。 ● sctpcomm_peak_memory：TCP代理通信的内存峰值。 ● other_used_memory：其他已使用的内存大小。 ● gpu_max_dynamic_memory：GPU最大动态内存。 ● gpu_dynamic_used_memory：GPU已使用的动态内存。 ● gpu_dynamic_peak_memory：GPU内存的动态峰值。 ● pooler_conn_memory：连接池申请内存计数。 ● pooler_freeconn_memory：连接池空闲连接的内存计数。 ● storage_compress_memory：存储模块压缩使用的内存大小。 ● udf_reserved_memory：UDF预留的内存大小。
memorymbytes	integer	内存使用的大小，单位为MB。

13.2.3.2 GLOBAL_MEMORY_NODE_DETAIL

显示当前集群中所有正常节点下的内存使用情况。

表 13-16 GLOBAL_MEMORY_NODE_DETAIL 字段

名称	类型	描述
nodename	text	节点名称。
memorytype	text	内存使用的名称。 ● max_process_memory：数据库节点可用内存的最大值。 ● process_used_memory：进程所使用的内存大小。 ● max_dynamic_memory：最大动态内存。 ● dynamic_used_memory：已使用的动态内存。 ● dynamic_peak_memory：内存的动态峰值。 ● dynamic_used_shrctx：已使用的动态共享内存上下文。 ● dynamic_peak_shrctx：共享内存上下文的动态峰值。 ● max_shared_memory：最大共享内存。 ● shared_used_memory：已使用的共享内存。 ● max_sctpcomm_memory：TCP代理通信所允许使用的最大内存。 ● sctpcomm_used_memory：TCP代理通信已使用的内存大小。 ● sctpcomm_peak_memory：TCP代理通信的内存峰值。 ● other_used_memory：其他已使用的内存大小。 ● gpu_max_dynamic_memory：GPU最大动态内存。 ● gpu_dynamic_used_memory：GPU已使用的动态内存。 ● gpu_dynamic_peak_memory：GPU内存的动态峰值。 ● pooler_conn_memory：连接池申请内存计数。 ● pooler_freeconn_memory：连接池空闲连接的内存计数。 ● storage_compress_memory：存储模块压缩使用的内存大小。 ● udf_reserved_memory：UDF预留的内存大小。
memorymbytes	integer	内存使用的大小，单位为MB。

13.2.3.3 MEMORY_NODE_NG_DETAIL

nodegroup内存使用情况。

表 13-17 MEMORY_NODE_NG_DETAIL 字段

名称	类型	描述
ngname	text	node group名称。
memorytype	text	内存使用的名称： ● ng_total_memory：node group中设置的总内存。 ● ng_used_memory：已经用的内存。 ● ng_estimate_memory：优化器评估已经用的内存。 ● ng_foreignrp_memsize：外部资源池设置的内存大小。 ● ng_foreignrp_usedsize：外部资源池当前已用内存。 ● ng_foreignrp_peaksize：外部资源池已使用的内存峰值。 ● ng_foreignrp_mempct：外部资源池属性中设置的占用系统总内存的百分比。 ● ng_foreignrp_estmsize：外部资源池执行作业优化器评估的内存使用。
memorymbytes	integer	内存使用的大小，单位为MB。

13.2.3.4 SHARED_MEMORY_DETAIL

查询当前节点所有已产生的共享内存上下文的使用信息。

表 13-18 SHARED_MEMORY_DETAIL 字段

名称	类型	描述
contextname	text	内存上下文的名称。
level	smallint	内存上下文的级别。
parent	text	上级内存上下文。
totalsize	bigint	共享内存总大小(单位：字节)。
freesize	bigint	共享内存剩余大小(单位：字节)。
usedsize	bigint	共享内存使用大小(单位：字节)。

13.2.3.5 GLOBAL_SHARED_MEMORY_DETAIL

查询整个集群中所有正常节点下的共享内存上下文的使用信息。

表 13-19 GLOBAL_SHARED_MEMORY_DETAIL 字段

名称	类型	描述
node_name	name	节点名称。
contextname	text	内存上下文的名称。
level	smallint	内存上下文的级别。
parent	text	上级内存上下文。
totalsize	bigint	共享内存总大小(单位：字节)。
freesize	bigint	共享内存剩余大小(单位：字节)。
usedsize	bigint	共享内存使用大小(单位：字节)。

13.2.3.6 TRACK_MEMORY_CONTEXT_DETAIL

查询DBE_PERF.track_memory_context设置的内存上下文上的内存申请详细信息。只有初始用户或者具有monadmin权限的用户可以执行该函数。

表 13-20 TRACK_MEMORY_CONTEXT_DETAIL 字段

名称	类型	描述
context_name	text	内存上下文的名称。
file	text	内存申请位置所属的文件。
line	integer	内存申请位置的行号。
size	bigint	内存申请的总大小（单位：字节）。

13.2.4 File

13.2.4.1 FILE_IOSTAT

通过对数据文件I/O的统计，反映数据的I/O性能，用以发现I/O操作异常等性能问题。

表 13-21 FILE_IOSTAT 字段

名称	类型	描述
filenum	oid	文件标识。
dbid	oid	数据库标识。
spcid	oid	表空间标识。

名称	类型	描述
phyrds	bigint	读物理文件的数目。
phywrts	bigint	写物理文件的数目。
phyblkrd	bigint	读物理文件块的数目。
phyblkwrt	bigint	写物理文件块的数目。
readtim	bigint	读文件的总时长（单位：微秒）。
writetim	bigint	写文件的总时长（单位：微秒）。
avgiotim	bigint	读写文件的平均时长（单位：微秒）。
lstiotim	bigint	最后一次读文件时长（单位：微秒）。
miniotim	bigint	读写文件的最小时长（单位：微秒）。
maxiowtm	bigint	读写文件的最大时长（单位：微秒）。

13.2.4.2 SUMMARY_FILE_IOSTAT

通过集群内对数据文件汇聚I/O的统计，反映数据的I/O性能，用以发现I/O操作异常等性能问题。

表 13-22 SUMMARY_FILE_IOSTAT 字段

名称	类型	描述
filenum	oid	文件标识。
dbid	oid	数据库标识。
spcid	oid	表空间标识。
phyrds	numeric	读物理文件的数目。
phywrts	numeric	写物理文件的数目。
phyblkrd	numeric	读物理文件块的数目。
phyblkwrt	numeric	写物理文件块的数目。
readtim	numeric	读文件的总时长（单位：微秒）。
writetim	numeric	写文件的总时长（单位：微秒）。
avgiotim	bigint	读写文件的平均时长（单位：微秒）。
lstiotim	bigint	最后一次读文件时长（单位：微秒）。
miniotim	bigint	读写文件的最小时长（单位：微秒）。
maxiowtm	bigint	读写文件的最大时长（单位：微秒）。

13.2.4.3 GLOBAL_FILE_IOSTAT

显示所有节点上的数据文件I/O统计信息。

表 13-23 GLOBAL_FILE_IOSTAT 字段

名称	类型	描述
node_name	name	节点名称
filenum	oid	文件标识。
dbid	oid	数据库标识。
spcid	oid	表空间标识。
phyrds	bigint	读物理文件的数目。
phywrts	bigint	写物理文件的数目。
phyblkrd	bigint	读物理文件块的数目。
phyblkwrt	bigint	写物理文件块的数目。
readtim	bigint	读文件的总时长（单位：微秒）。
writetim	bigint	写文件的总时长（单位：微秒）。
avgiotim	bigint	读写文件的平均时长（单位：微秒）。
lstiotim	bigint	最后一次读文件时长（单位：微秒）。
miniotim	bigint	读写文件的最小时长（单位：微秒）。
maxiowtm	bigint	读写文件的最大时长（单位：微秒）。

13.2.4.4 FILE_REDO_IOSTAT

本节点Redo(WAL)相关的统计信息。

表 13-24 FILE_REDO_IOSTAT 字段

名称	类型	描述
phywrts	bigint	向wal buffer中写的次数。
phyblkwrt	bigint	向wal buffer中写的block的块数。
writetim	bigint	向xLog文件中写操作的时间（单位：微秒）。
avgiotim	bigint	平均写xLog的时间（writetim/phywrts，单位：微秒）。
lstiotim	bigint	最后一次写xLog的时间（单位：微秒）。
miniotim	bigint	最小的写xLog时间（单位：微秒）。

名称	类型	描述
maxiowtm	bigint	最大的写xLog时间（单位：微秒）。

13.2.4.5 SUMMARY_FILE_REDO_IOSTAT

集群内汇聚所有的Redo(WAL)相关的统计信息。

表 13-25 SUMMARY_FILE_REDO_IOSTAT 字段

名称	类型	描述
phywrts	numeric	向wal buffer中写的次数。
phyblkwrt	numeric	向wal buffer中写的block的块数。
writetim	numeric	向xLog文件中写操作的时间（单位：微秒）。
avgiotim	bigint	平均写xLog的时间（writetim/phywrts，单位：微秒）。
lstiotim	bigint	最后一次写xLog的时间（单位：微秒）。
miniotim	bigint	最小的写xLog时间（单位：微秒）。
maxiowtm	bigint	最大的写xLog时间（单位：微秒）。

13.2.4.6 GLOBAL_FILE_REDO_IOSTAT

显示集群内各节点的Redo(WAL)相关统计信息。

表 13-26 GLOBAL_FILE_REDO_IOSTAT 字段

名称	类型	描述
node_name	name	节点名称。
phywrts	bigint	向wal buffer中写的次数。
phyblkwrt	bigint	向wal buffer中写的block的块数。
writetim	bigint	向xLog文件中写操作的时间（单位：微秒）。
avgiotim	bigint	平均写xLog的时间（writetim/phywrts，单位：微秒）。
lstiotim	bigint	最后一次写xLog的时间（单位：微秒）。
miniotim	bigint	最小的写xLog时间（单位：微秒）。
maxiowtm	bigint	最大的写xLog时间（单位：微秒）。

13.2.4.7 LOCAL_REL_IOSTAT

获取当前节点中数据文件I/O状态的累计值，显示为所有数据文件I/O状态的总和。

表 13-27 LOCAL_REL_IOSTAT 字段

名称	类型	描述
phyrds	bigint	读物理文件的数目。
phywrts	bigint	写物理文件的数目。
phyblkrd	bigint	读物理文件的块的数目。
phyblkwrt	bigint	写物理文件的块的数目。

13.2.4.8 GLOBAL_REL_IOSTAT

获取所有节点上的数据文件I/O统计信息。

表 13-28 GLOBAL_REL_IOSTAT 字段

名称	类型	描述
node_name	name	节点名称
phyrds	bigint	读物理文件的数目。
phywrts	bigint	写物理文件的数目。
phyblkrd	bigint	读物理文件块的数目。
phyblkwrt	bigint	写物理文件块的数目。

13.2.4.9 SUMMARY_REL_IOSTAT

获取所有节点上的数据文件I/O统计信息。

表 13-29 SUMMARY_REL_IOSTAT 字段

名称	类型	描述
phyrds	numeric	读物理文件的数目。
phywrts	numeric	写物理文件的数目。
phyblkrd	numeric	读物理文件的块的数目。
phyblkwrt	numeric	写物理文件的块的数目。

13.2.5 Object

13.2.5.1 STAT_USER_TABLES

显示当前节点所有命名空间中用户自定义普通表的状态信息。

表 13-30 STAT_USER_TABLES 字段

名称	类型	描述
relid	oid	表的OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（即没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计死行数。
last_vacuum	timestamp with time zone	最后一次该表是手动清理的（不计算VACUUM FULL）时间。
last_autovacuum	timestamp with time zone	上次被autovacuum守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析该表的时间。
last_autoanalyze	timestamp with time zone	上次被autovacuum守护进程分析的时间。
vacuum_count	bigint	该表被手动清理的次数（不计算VACUUM FULL）。
autovacuum_count	bigint	该表被autovacuum清理的次数。
analyze_count	bigint	该表被手动分析的次数。

名称	类型	描述
autoanalyze_count	bigint	该表被autovacuum守护进程分析的次数。

13.2.5.2 SUMMARY_STAT_USER_TABLES

集群内所有命名空间中用户自定义普通表的状态信息。

表 13-31 SUMMARY_STAT_USER_TABLES

名称	类型	描述
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	numeric	该表发起的顺序扫描数。
seq_tup_read	numeric	顺序扫描抓取的活跃行数。
idx_scan	numeric	该表发起的索引扫描数。
idx_tup_fetch	numeric	索引扫描抓取的活跃行数。
n_tup_ins	numeric	插入行数。
n_tup_upd	numeric	更新行数。
n_tup_del	numeric	删除行数。
n_tup_hot_upd	numeric	HOT更新行数（即没有更新所需的单独索引）。
n_live_tup	numeric	估计活跃行数。
n_dead_tup	numeric	估计死行数。
last_vacuum	timestamp with time zone	最后一次该表是手动清理的（不计算VACUUM FULL）时间。
last_autovacuum	timestamp with time zone	上次被autovacuum守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析该表的时间。
last_autoanalyze	timestamp with time zone	上次被autovacuum守护进程分析的时间。
vacuum_count	numeric	该表被手动清理的次数（不计算VACUUM FULL）。
autovacuum_count	numeric	该表被autovacuum清理的次数。

名称	类型	描述
analyze_count	numeric	该表被手动分析的次数。
autoanalyze_count	numeric	该表被autovacuum守护进程分析的次数。

13.2.5.3 GLOBAL_STAT_USER_TABLES

显示各节点所有命名空间中用户自定义普通表的状态信息。

表 13-32 GLOBAL_STAT_USER_TABLES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	表的OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（即没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计死行数。
last_vacuum	timestamp with time zone	最后一次该表是手动清理的（不计算VACUUM FULL）时间。
last_autovacuum	timestamp with time zone	上次被autovacuum守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析该表的时间。
last_autoanalyze	timestamp with time zone	上次被autovacuum守护进程分析的时间。

名称	类型	描述
vacuum_count	bigint	该表被手动清理的次数（不计算VACUUM FULL）。
autovacuum_count	bigint	该表被autovacuum清理的次数。
analyze_count	bigint	该表被手动分析的次数。
autoanalyze_count	bigint	该表被autovacuum守护进程分析的次数。

13.2.5.4 STAT_USER_INDEXES

显示数据库中用户自定义普通表的索引状态信息。

表 13-33 STAT_USER_INDEXES 字段

名称	类型	描述
relid	oid	该索引的表的OID。
indexrelid	oid	索引的OID。
schemaname	name	索引的模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	通过索引上扫描返回的索引项数。
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

13.2.5.5 SUMMARY_STAT_USER_INDEXES

集群内汇聚所有数据库中用户自定义普通表的索引状态信息。

表 13-34 SUMMARY_STAT_USER_INDEXES 字段

名称	类型	描述
schemaname	name	索引的模式名。
relname	name	索引的表名。

名称	类型	描述
indexrelname	name	索引名。
idx_scan	numeric	索引上开始的索引扫描数。
idx_tup_read	numeric	通过索引上扫描返回的索引项数。
idx_tup_fetch	numeric	通过使用索引的简单索引扫描抓取的活表行数。

13.2.5.6 GLOBAL_STAT_USER_INDEXES

显示各节点数据库中用户自定义普通表的索引状态信息。

表 13-35 GLOBAL_STAT_USER_INDEXES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	该索引的表的OID。
indexrelid	oid	索引的OID。
schemaname	name	索引的模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	通过索引上扫描返回的索引项数。
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

13.2.5.7 STAT_SYS_TABLES

显示单节点内pg_catalog、information_schema以及pg_toast模式下所有系统表的统计信息。

表 13-36 STAT_SYS_TABLES 字段

名称	类型	描述
relid	oid	表的OID。
schemaname	name	该表的模式名。

名称	类型	描述
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（比如没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计死行数。
last_vacuum	timestamp with time zone	最后一次该表是手动清理的（不计算VACUUM FULL）时间。
last_autovacuum	timestamp with time zone	上次被autovacuum守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析该表的时间。
last_autoanalyze	timestamp with time zone	上次被autovacuum守护进程分析的时间。
vacuum_count	bigint	这个表被手动清理的次数（不计算VACUUM FULL）。
autovacuum_count	bigint	该表被autovacuum清理的次数。
analyze_count	bigint	该表被手动分析的次数。
autoanalyze_count	bigint	该表被autovacuum守护进程分析的次数。

13.2.5.8 SUMMARY_STAT_SYS_TABLES

集群内汇聚pg_catalog、information_schema以及pg_toast模式下所有系统表的统计信息。

表 13-37 SUMMARY_STAT_SYS_TABLES 字段

名称	类型	描述
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	numeric	该表发起的顺序扫描数。
seq_tup_read	numeric	顺序扫描抓取的活跃行数。
idx_scan	numeric	该表发起的索引扫描数。
idx_tup_fetch	numeric	索引扫描抓取的活跃行数。
n_tup_ins	numeric	插入行数。
n_tup_upd	numeric	更新行数。
n_tup_del	numeric	删除行数。
n_tup_hot_upd	numeric	HOT更新行数（比如没有更新所需的单独索引）。
n_live_tup	numeric	估计活跃行数。
n_dead_tup	numeric	估计死行数。
last_vacuum	timestamp with time zone	最后一次该表是手动清理的（不计算VACUUM FULL）时间。
last_autovacuum	timestamp with time zone	上次被autovacuum守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析该表的时间。
last_autoanalyze	timestamp with time zone	上次被autovacuum守护进程分析的时间。
vacuum_count	numeric	该表被手动清理的次数（不计算VACUUM FULL）。
autovacuum_count	numeric	该表被autovacuum清理的次数。
analyze_count	numeric	该表被手动分析的次数。
autoanalyze_count	numeric	该表被autovacuum守护进程分析的次数。

13.2.5.9 GLOBAL_STAT_SYS_TABLES

显示集群各个节点pg_catalog、information_schema以及pg_toast模式下所有系统表的统计信息。

表 13-38 GLOBAL_STAT_SYS_TABLES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	表的OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（比如没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计死行数。
last_vacuum	timestamp with time zone	最后一次该表是手动清理的（不计算VACUUM FULL）时间。
last_autovacuum	timestamp with time zone	上次被autovacuum守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析该表的时间。
last_autoanalyze	timestamp with time zone	上次被autovacuum守护进程分析的时间。
vacuum_count	bigint	该表被手动清理的次数（不计算VACUUM FULL）。
autovacuum_count	bigint	该表被autovacuum清理的次数。
analyze_count	bigint	该表被手动分析的次数。
autoanalyze_count	bigint	该表被autovacuum守护进程分析的次数。

13.2.5.10 STAT_SYS_INDEXES

显示pg_catalog、information_schema以及pg_toast模式中所有系统表的索引状态信息。

表 13-39 STAT_SYS_INDEXES 字段

名称	类型	描述
relid	oid	该索引的表的OID。
indexrelid	oid	索引的OID。
schemaname	name	索引的模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	通过索引上扫描返回的索引项数。
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

13.2.5.11 SUMMARY_STAT_SYS_INDEXES

集群内汇聚pg_catalog、information_schema以及pg_toast模式中所有系统表的索引状态信息。

表 13-40 SUMMARY_STAT_SYS_INDEXES 字段

名称	类型	描述
schemaname	name	索引的模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	numeric	索引上开始的索引扫描数。
idx_tup_read	numeric	通过索引上扫描返回的索引项数。
idx_tup_fetch	numeric	通过使用索引的简单索引扫描抓取的活表行数。

13.2.5.12 GLOBAL_STAT_SYS_INDEXES

显示各节点pg_catalog、information_schema以及pg_toast模式中所有系统表的索引状态信息。

表 13-41 GLOBAL_STAT_SYS_INDEXES 字段

名称	类型	描述
node_name	name	节点名称
relid	oid	这个索引的表的OID。
indexrelid	oid	索引的OID。
schemaname	name	索引的模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	通过索引上扫描返回的索引项数。
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

13.2.5.13 STAT_ALL_TABLES

本节点内数据库中每个表的一行（包括TOAST表）的统计信息。

表 13-42 STAT_ALL_TABLES 字段

名称	类型	描述
relid	oid	表的OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。

名称	类型	描述
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（比如没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计死行数。
last_vacuum	timestamp with time zone	最后一次该表是手动清理的（不计算VACUUM FULL）的时间。
last_autovacuum	timestamp with time zone	上次被autovacuum守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析该表的时间。
last_autoanalyze	timestamp with time zone	上次被autovacuum守护进程分析时间。
vacuum_count	bigint	该表被手动清理的次数（不计算VACUUM FULL）。
autovacuum_count	bigint	该表被autovacuum清理的次数。
analyze_count	bigint	该表被手动分析的次数。
autoanalyze_count	bigint	该表被autovacuum守护进程分析的次数。

13.2.5.14 SUMMARY_STAT_ALL_TABLES

显示集群内汇聚数据库中每个表（包括TOAST表）的一行的统计信息。

表 13-43 SUMMARY_STAT_ALL_TABLES 字段

名称	类型	描述
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	numeric	该表发起的顺序扫描数。
seq_tup_read	numeric	顺序扫描抓取的活跃行数。
idx_scan	numeric	该表发起的索引扫描数。
idx_tup_fetch	numeric	索引扫描抓取的活跃行数。
n_tup_ins	numeric	插入行数。

名称	类型	描述
n_tup_upd	numeric	更新行数。
n_tup_del	numeric	删除行数。
n_tup_hot_upd	numeric	HOT更新行数（比如没有更新所需的单独索引）。
n_live_tup	numeric	估计活跃行数。
n_dead_tup	numeric	估计死行数。
last_vacuum	timestamp with time zone	最后一次该表是手动清理的（不计算VACUUM FULL）的时间。
last_autovacuum	timestamp with time zone	上次被autovacuum守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析该表的时间。
last_autoanalyze	timestamp with time zone	上次被autovacuum守护进程分析时间。
vacuum_count	numeric	该表被手动清理的次数（不计算VACUUM FULL）。
autovacuum_count	numeric	该表被autovacuum清理的次数。
analyze_count	numeric	该表被手动分析的次数。
autoanalyze_count	numeric	该表被autovacuum守护进程分析的次数。

13.2.5.15 GLOBAL_STAT_ALL_TABLES

显示各节点数据中每个表（包括TOAST表）的一行的统计信息。

表 13-44 GLOBAL_STAT_ALL_TABLES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	表的OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。

名称	类型	描述
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（比如没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计死行数。
last_vacuum	timestamp with time zone	最后一次该表是手动清理的（不计算VACUUM FULL）的时间。
last_autovacuum	timestamp with time zone	上次被autovacuum守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析该表的时间。
last_autoanalyze	timestamp with time zone	上次被autovacuum守护进程分析时间。
vacuum_count	bigint	该表被手动清理的次数（不计算VACUUM FULL）。
autovacuum_count	bigint	该表被autovacuum清理的次数。
analyze_count	bigint	该表被手动分析的次数。
autoanalyze_count	bigint	该表被autovacuum守护进程分析的次数。

13.2.5.16 STAT_ALL_INDEXES

统计显示本节点数据库中的每个索引的访问信息。

表 13-45 STAT_ALL_INDEXES 字段

名称	类型	描述
relid	oid	该索引的表的OID。
indexrelid	oid	索引的OID。
schemaname	name	索引的模式名。

名称	类型	描述
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	通过索引上扫描返回的索引项数。
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

13.2.5.17 SUMMARY_STAT_ALL_INDEXES

显示集群级数据库的每个索引的访问统计信息。

表 13-46 SUMMARY_STAT_ALL_INDEXES 字段

名称	类型	描述
schemaname	name	索引的模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	numeric	索引上开始的索引扫描数。
idx_tup_read	numeric	通过索引上扫描返回的索引项数。
idx_tup_fetch	numeric	通过使用索引的简单索引扫描抓取的活表行数。

13.2.5.18 GLOBAL_STAT_ALL_INDEXES

显示集群内各节点的数据库中的每个索引的访问信息。

表 13-47 GLOBAL_STAT_ALL_INDEXES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	该索引的表的OID。
indexrelid	oid	索引的OID。
schemaname	name	索引的模式名。

名称	类型	描述
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	通过索引上扫描返回的索引项数。
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

13.2.5.19 STAT_DATABASE

视图将包含本节点中每个数据库的统计信息。

表 13-48 STAT_DATABASE 字段

名称	类型	描述
datid	oid	数据库的OID。
datname	name	该数据库的名称。
numbackends	integer	当前连接到该数据库的后端数。这是在返回一个反映目前状态值的视图中唯一的列；自上次重置所有其他列返回累积值。
xact_commit	bigint	此数据库中已经提交的事务数。
xact_rollback	bigint	此数据库中已经回滚的事务数。
blks_read	bigint	在这个数据库中读取的磁盘块的数量。
blks_hit	bigint	高速缓存中已经发现的磁盘块的次数，这样读取是不必要的（这只包括缓冲区高速缓存，没有操作系统的文件系统缓存）。
tup_returned	bigint	通过数据库查询返回的行数。
tup_fetched	bigint	通过数据库查询抓取的行数。
tup_inserted	bigint	通过数据库查询插入的行数。
tup_updated	bigint	通过数据库查询更新的行数。
tup_deleted	bigint	通过数据库查询删除的行数。

名称	类型	描述
conflicts	bigint	由于数据库恢复冲突取消的查询数量（只在备用服务器发生的冲突）。请参见STAT_DATABASE_CONFLICTS获取更多信息。
temp_files	bigint	通过数据库查询创建的临时文件数量。计算所有临时文件，不论为什么创建临时文件（比如排序或者哈希），而且不管log_temp_files设置。
temp_bytes	bigint	通过数据库查询写入临时文件的数据总量。计算所有临时文件，不论为什么创建临时文件，而且不管log_temp_files设置。
deadlocks	bigint	在该数据库中检索的死锁数。
blk_read_time	double precision	通过数据库后端读取数据文件块花费的时间，以毫秒计算。
blk_write_time	double precision	通过数据库后端写入数据文件块花费的时间，以毫秒计算。
stats_reset	timestamp with time zone	重置当前状态统计的时间。

13.2.5.20 SUMMARY_STAT_DATABASE

视图将包含集群内汇聚的每个数据库的统计信息。

表 13-49 SUMMARY_STAT_DATABASE

名称	类型	描述
datname	name	数据库的名称。
numbackends	bigint	当前连接到该数据库的后端数。这是在返回一个反映目前状态值的视图中唯一的列；自上次重置所有其他列返回累积值。
xact_commit	numeric	此数据库中已经提交的事务数。
xact_rollback	numeric	此数据库中已经回滚的事务数。
blks_read	numeric	在这个数据库中读取的磁盘块的数量。
blks_hit	numeric	高速缓存中已经发现的磁盘块的次数，这样读取是不必要的（这只包括缓冲区高速缓存，没有操作系统的文件系统缓存）。
tup_returned	numeric	通过数据库查询返回的行数。
tup_fetched	numeric	通过数据库查询抓取的行数。

名称	类型	描述
tup_inserted	bigint	通过数据库查询插入的行数。
tup_updated	bigint	通过数据库查询更新的行数。
tup_deleted	bigint	通过数据库查询删除的行数。
conflicts	bigint	由于数据库恢复冲突取消的查询数量（只在备用服务器发生的冲突）。请参见STAT_DATABASE_CONFLICTS获取更多信息。
temp_files	numeric	通过数据库查询创建的临时文件数量。计算所有临时文件，不论为什么创建临时文件（比如排序或者哈希），而且不管log_temp_files设置。
temp_bytes	numeric	通过数据库查询写入临时文件的数据总量。计算所有临时文件，不论为什么创建临时文件，而且不管log_temp_files设置。
deadlocks	bigint	在该数据库中检索的死锁数。
blk_read_time	double precision	通过数据库后端读取数据文件块花费的时间，以毫秒计算。
blk_write_time	double precision	通过数据库后端写入数据文件块花费的时间，以毫秒计算。
stats_reset	timestamp with time zone	重置当前状态统计的时间。

13.2.5.21 GLOBAL_STAT_DATABASE

视图将包含集群中各节点的每个数据库统计信息。

表 13-50 GLOBAL_STAT_DATABASE 字段

名称	类型	描述
node_name	name	节点名称。
datid	oid	数据库的OID。
datname	name	该数据库的名称。
numbackends	integer	当前连接到该数据库的后端数。这是在返回一个反映目前状态值的视图中唯一的列；自上次重置所有其他列返回累积值。
xact_commit	bigint	此数据库中已经提交的事务数。

名称	类型	描述
xact_rollback	bigint	此数据库中已经回滚的事务数。
blks_read	bigint	在这个数据库中读取的磁盘块的数量。
blks_hit	bigint	高速缓存中已经发现的磁盘块的次数，这样读取是不必要的（这只包括数据库内核缓冲区高速缓存，没有操作系统的文件系统缓存）。
tup_returned	bigint	通过数据库查询返回的行数。
tup_fetched	bigint	通过数据库查询抓取的行数。
tup_inserted	bigint	通过数据库查询插入的行数。
tup_updated	bigint	通过数据库查询更新的行数。
tup_deleted	bigint	通过数据库查询删除的行数。
conflicts	bigint	由于数据库恢复冲突取消的查询数量（只在备用服务器发生的冲突）。请参见STAT_DATABASE_CONFLICTS获取更多信息。
temp_files	bigint	通过数据库查询创建的临时文件数量。计算所有临时文件，不论为什么创建临时文件（比如排序或者哈希），而且不管log_temp_files设置。
temp_bytes	bigint	通过数据库查询写入临时文件的数据总量。计算所有临时文件，不论为什么创建临时文件，而且不管log_temp_files设置。
deadlocks	bigint	在该数据库中检索的死锁数。
blk_read_time	double precision	通过数据库后端读取数据文件块花费的时间，以毫秒计算。
blk_write_time	double precision	通过数据库后端写入数据文件块花费的时间，以毫秒计算。
stats_reset	timestamp with time zone	重置当前状态统计的时间。

13.2.5.22 STAT_DATABASE_CONFLICTS

显示当前节点数据库冲突状态的统计信息。

表 13-51 STAT_DATABASE_CONFLICTS 字段

名称	类型	描述
datid	oid	数据库标识。
datname	name	数据库名称。
confl_tablespace	bigint	冲突的表空间的数目。
confl_lock	bigint	冲突的锁数目。
confl_snapshot	bigint	冲突的快照数目。
confl_bufferpin	bigint	冲突的缓冲区数目。
confl_deadlock	bigint	冲突的死锁数目。

13.2.5.23 SUMMARY_STAT_DATABASE_CONFLICTS

显示集群内汇聚的数据库冲突状态的统计信息。

表 13-52 SUMMARY_STAT_DATABASE_CONFLICTS 字段

名称	类型	描述
datname	name	数据库名称。
confl_tablespace	bigint	冲突的表空间的数目。
confl_lock	bigint	冲突的锁数目。
confl_snapshot	bigint	冲突的快照数目。
confl_bufferpin	bigint	冲突的缓冲区数目。
confl_deadlock	bigint	冲突的死锁数目。

13.2.5.24 GLOBAL_STAT_DATABASE_CONFLICTS

显示每个节点的数据库冲突状态的统计信息。

表 13-53 GLOBAL_STAT_DATABASE_CONFLICTS 字段

名称	类型	描述
node_name	name	节点名称
datid	oid	数据库标识。

名称	类型	描述
datname	name	数据库名称。
confl_tablespace	bigint	冲突的表空间的数目。
confl_lock	bigint	冲突的锁数目。
confl_snapshot	bigint	冲突的快照数目。
confl_bufferpin	bigint	冲突的缓冲区数目。
confl_deadlock	bigint	冲突的死锁数目。

13.2.5.25 STAT_XACT_ALL_TABLES

显示命名空间中所有普通表和toast表的事务状态信息。

表 13-54 STAT_XACT_ALL_TABLES 字段

名称	类型	描述
relid	oid	表的OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（比如没有更新所需的单独索引）。

13.2.5.26 SUMMARY_STAT_XACT_ALL_TABLES

显示集群内汇聚的命名空间中所有普通表和toast表的事务状态信息。

表 13-55 SUMMARY_STAT_XACT_ALL_TABLES 字段

名称	类型	描述
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	numeric	该表发起的顺序扫描数。
seq_tup_read	numeric	顺序扫描抓取的活跃行数。
idx_scan	numeric	该表发起的索引扫描数。
idx_tup_fetch	numeric	索引扫描抓取的活跃行数。
n_tup_ins	numeric	插入行数。
n_tup_upd	numeric	更新行数。
n_tup_del	numeric	删除行数。
n_tup_hot_upd	numeric	HOT更新行数（比如没有更新所需的单独索引）。

13.2.5.27 GLOBAL_STAT_XACT_ALL_TABLES

显示各节点的命名空间中所有普通表和toast表的事务状态信息。

表 13-56 GLOBAL_STAT_XACT_ALL_TABLES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	表的OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。

名称	类型	描述
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（比如没有更新所需的单独索引）。

13.2.5.28 STAT_XACT_SYS_TABLES

显示当前节点命名空间中系统表的事务状态信息。

表 13-57 STAT_XACT_SYS_TABLES 字段

名称	类型	描述
relid	oid	表的OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（比如没有更新所需的单独索引）。

13.2.5.29 SUMMARY_STAT_XACT_SYS_TABLES

显示集群内汇聚的命名空间中系统表的事务状态信息。

表 13-58 SUMMARY_STAT_XACT_SYS_TABLES 字段

名称	类型	描述
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	numeric	该表发起的顺序扫描数。
seq_tup_read	numeric	顺序扫描抓取的活跃行数。
idx_scan	numeric	该表发起的索引扫描数。
idx_tup_fetch	numeric	索引扫描抓取的活跃行数。
n_tup_ins	numeric	插入行数。
n_tup_upd	numeric	更新行数。
n_tup_del	numeric	删除行数。
n_tup_hot_upd	numeric	HOT更新行数（比如没有更新所需的单独索引）。

13.2.5.30 GLOBAL_STAT_XACT_SYS_TABLES

显示各节点命名空间中系统表的事务状态信息。

表 13-59 GLOBAL_STAT_XACT_SYS_TABLES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	表的OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。

名称	类型	描述
n_tup_hot_upd	bigint	HOT更新行数（比如没有更新所需的单独索引）。

13.2.5.31 STAT_XACT_USER_TABLES

显示当前节点命名空间中用户表的事务状态信息。

表 13-60 STAT_XACT_USER_TABLES 字段

名称	类型	描述
relid	oid	表的OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（比如没有更新所需的单独索引）。

13.2.5.32 SUMMARY_STAT_XACT_USER_TABLES

显示集群内汇聚的命名空间中用户表的事务状态信息。

表 13-61 SUMMARY_STAT_XACT_USER_TABLES 字段

名称	类型	描述
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	numeric	该表发起的顺序扫描数。

名称	类型	描述
seq_tup_read	numeric	顺序扫描抓取的活跃行数。
idx_scan	numeric	该表发起的索引扫描数。
idx_tup_fetch	numeric	索引扫描抓取的活跃行数。
n_tup_ins	numeric	插入行数。
n_tup_upd	numeric	更新行数。
n_tup_del	numeric	删除行数。
n_tup_hot_upd	numeric	HOT更新行数（比如没有更新所需的单独索引）。

13.2.5.33 GLOBAL_STAT_XACT_USER_TABLES

显示各节点命名空间中用户表的事务状态信息。

表 13-62 GLOBAL_STAT_XACT_USER_TABLES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	表的OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT更新行数（比如没有更新所需的单独索引）。

13.2.5.34 STAT_XACT_USER_FUNCTIONS

视图包含当前节点本事务内函数执行的统计信息。

表 13-63 STAT_XACT_USER_FUNCTIONS 字段

名称	类型	描述
funcid	oid	函数标识。
schemaname	name	模式的名称。
funcname	name	函数名称。
calls	bigint	函数被调用的次数。
total_time	double precision	函数的总执行时长。
self_time	double precision	当前线程调用函数的总的时长。

13.2.5.35 SUMMARY_STAT_XACT_USER_FUNCTIONS

视图包含集群内汇聚的本事务内函数执行的统计信息。

表 13-64 SUMMARY_STAT_XACT_USER_FUNCTIONS 字段

名称	类型	描述
schemaname	name	模式的名称。
funcname	name	函数名称。
calls	numeric	函数被调用的次数。
total_time	double precision	函数的总执行时长。
self_time	double precision	当前线程调用函数的总的时长。

13.2.5.36 GLOBAL_STAT_XACT_USER_FUNCTIONS

视图包含各节点本事务内函数执行的统计信息。

表 13-65 GLOBAL_STAT_XACT_USER_FUNCTIONS 字段

名称	类型	描述
node_name	name	节点名称。
funcid	oid	函数标识。

名称	类型	描述
schemaname	name	模式的名称。
funcname	name	函数名称。
calls	bigint	函数被调用的次数。
total_time	double precision	此函数及其调用的所有其他函数所花费的总时间。
self_time	double precision	在此函数本身中花费的总时间（不包括它调用的其他函数）。

13.2.5.37 STAT_BAD_BLOCK

获得当前节点表、索引等文件的读取失败信息。

表 13-66 STAT_BAD_BLOCK 字段

名称	类型	描述
nodename	text	节点名称。
databaseid	integer	database的oid。
tablespaceid	integer	tablespace的oid。
relfilenode	integer	relation的file node。
forknum	integer	fork编号。
error_count	integer	error的数量。
first_time	timestamp with time zone	页面损坏第一次出现的时间。
last_time	timestamp with time zone	页面损坏最后出现的时间。

13.2.5.38 SUMMARY_STAT_BAD_BLOCK

获得集群内汇聚的表、索引等文件的读取失败信息。

表 13-67 SUMMARY_STAT_BAD_BLOCK 字段

名称	类型	描述
databaseid	integer	database的oid。
tablespaceid	integer	tablespace的oid。

名称	类型	描述
relfilenode	integer	relation的file node。
forknum	bigint	fork编号。
error_count	bigint	error的数量。
first_time	timestamp with time zone	页面损坏第一次出现的时间。
last_time	timestamp with time zone	页面损坏最后出现的时间。

13.2.5.39 GLOBAL_STAT_BAD_BLOCK

获得各节点的表、索引等文件的读取失败信息。

表 13-68 GLOBAL_STAT_BAD_BLOCK 字段

名称	类型	描述
node_name	text	节点名称。
databaseid	integer	database的oid。
tablespaceid	integer	tablespace的oid。
relfilenode	integer	relation的file node。
forknum	integer	fork编号。
error_count	integer	error的数量。
first_time	timestamp with time zone	页面损坏第一次出现的时间。
last_time	timestamp with time zone	页面损坏最后出现的时间。

13.2.5.40 STAT_USER_FUNCTIONS

STAT_USER_FUNCTIONS视图显示命名空间中用户自定义函数（函数语言为非内部语言）的状态信息。

表 13-69 STAT_USER_FUNCTIONS 字段

名称	类型	描述
funcid	oid	函数标识。
schemaname	name	schema的名称。

名称	类型	描述
funcname	name	用户function的名称。
calls	bigint	函数被调用的次数。
total_time	double precision	调用此function的总时间花费，包含调用其它function的时间（单位：毫秒）。
self_time	double precision	调用此function自己时间的花费，不包含调用其它function的时间（单位：毫秒）。

13.2.5.41 SUMMARY_STAT_USER_FUNCTIONS

SUMMARY_STAT_USER_FUNCTIONS用来统计CN/DN用户自定义视图的相关统计信息。

表 13-70 SUMMARY_STAT_USER_FUNCTIONS 字段

名称	类型	描述
schemaname	name	schema的名称。
funcname	name	用户function的名称。
calls	numeric	函数被调用的次数。
total_time	double precision	调用此function的总时间花费，包含调用其它function的时间(单位：毫秒)。
self_time	double precision	调用此function自己时间的花费，不包含调用其它function的时间(单位：毫秒)。

13.2.5.42 GLOBAL_STAT_USER_FUNCTIONS

提供整个集群中各个节点的用户所创建的函数的状态的统计信息。

表 13-71 GLOBAL_STAT_USER_FUNCTIONS 字段

名称	类型	描述
node_name	name	节点名称。
funcid	oid	函数的id。
schemaname	name	此函数所在模式的名称。
funcname	name	用户function的名称。

名称	类型	描述
calls	bigint	函数被调用的次数。
total_time	double precision	调用此function的总时间花费，包含调用其它function的时间(单位：毫秒)。
self_time	double precision	调用此function自己时间的花费，不包含调用其它function的时间(单位：毫秒)。

13.2.6 Workload

13.2.6.1 WORKLOAD_SQL_COUNT

显示当前节点workload上的SQL数量分布。普通用户只可以看到自己在workload上的SQL分布；monadmin可以看到总的workload的负载情况。

表 13-72 WORKLOAD_SQL_COUNT 字段

名称	类型	描述
workload	name	负载名称。
select_count	bigint	select数量。
update_count	bigint	update数量。
insert_count	bigint	insert数量。
delete_count	bigint	delete数量。
ddl_count	bigint	ddl数量。
dml_count	bigint	dml数量。
dcl_count	bigint	dcl数量。

13.2.6.2 SUMMARY_WORKLOAD_SQL_COUNT

显示集群内各CN的workload上的SQL数量分布。

表 13-73 SUMMARY_WORKLOAD_SQL_COUNT 字段

名称	类型	描述
node_name	name	节点名称。

名称	类型	描述
workload	name	负载名称。
select_count	bigint	select数量。
update_count	bigint	update数量。
insert_count	bigint	insert数量。
delete_count	bigint	delete数量。
ddl_count	bigint	ddl数量。
dml_count	bigint	dml数量。
dcl_count	bigint	dcl数量。

13.2.6.3 WORKLOAD_TRANSACTION

当前节点上负载的事务信息。

表 13-74 WORKLOAD_TRANSACTION 字段

名称	类型	描述
workload	name	负载的名称。
commit_counter	bigint	用户事务commit数量。
rollback_counter	bigint	用户事务rollback数量。
resp_min	bigint	用户事务最小响应时间（单位：微秒）。
resp_max	bigint	用户事务最大响应时间（单位：微秒）。
resp_avg	bigint	用户事务平均响应时间（单位：微秒）。
resp_total	bigint	用户事务总响应时间（单位：微秒）。
bg_commit_counter	bigint	后台事务commit数量。
bg_rollback_counter	bigint	后台事务rollback数量。
bg_resp_min	bigint	后台事务最小响应时间（单位：微秒）。
bg_resp_max	bigint	后台事务最大响应时间（单位：微秒）。
bg_resp_avg	bigint	后台事务平均响应时间（单位：微秒）。

名称	类型	描述
bg_resp_total	bigint	后台事务总响应时间（单位：微秒）。

13.2.6.4 SUMMARY_WORKLOAD_TRANSACTION

显示集群内汇聚的负载事务信息。

表 13-75 SUMMARY_WORKLOAD_TRANSACTION 字段

名称	类型	描述
workload	name	负载的名称。
commit_counter	numeric	用户事务commit数量。
rollback_counter	numeric	用户事务rollback数量。
resp_min	bigint	用户事务最小响应时间（单位：微秒）。
resp_max	bigint	用户事务最大响应时间（单位：微秒）。
resp_avg	bigint	用户事务平均响应时间（单位：微秒）。
resp_total	numeric	用户事务总响应时间（单位：微秒）。
bg_commit_counter	numeric	后台事务commit数量。
bg_rollback_counter	numeric	后台事务rollback数量。
bg_resp_min	bigint	后台事务最小响应时间（单位：微秒）。
bg_resp_max	bigint	后台事务最大响应时间（单位：微秒）。
bg_resp_avg	bigint	后台事务平均响应时间（单位：微秒）。
bg_resp_total	numeric	后台事务总响应时间（单位：微秒）。

13.2.6.5 GLOBAL_WORKLOAD_TRANSACTION

显示各节点上的workload的负载信息。

表 13-76 GLOBAL_WORKLOAD_TRANSACTION 字段

名称	类型	描述
node_name	name	节点名称。
workload	name	负载的名称。
commit_counter	bigint	用户事务commit数量。

名称	类型	描述
rollback_counter	bigint	用户事务rollback数量。
resp_min	bigint	用户事务最小响应时间（单位：微秒）。
resp_max	bigint	用户事务最大响应时间（单位：微秒）。
resp_avg	bigint	用户事务平均响应时间（单位：微秒）。
resp_total	bigint	用户事务总响应时间（单位：微秒）。
bg_commit_counter	bigint	后台事务commit数量。
bg_rollback_counter	bigint	后台事务rollback数量。
bg_resp_min	bigint	后台事务最小响应时间（单位：微秒）。
bg_resp_max	bigint	后台事务最大响应时间（单位：微秒）。
bg_resp_avg	bigint	后台事务平均响应时间（单位：微秒）。
bg_resp_total	bigint	后台事务总响应时间（单位：微秒）。

13.2.6.6 WORKLOAD_SQL_ELAPSE_TIME

WORKLOAD_SQL_ELAPSE_TIME用来统计workload（业务负载）上的SUID信息。

表 13-77 WORKLOAD_SQL_ELAPSE_TIME 字段

名称	类型	描述
workload	name	workload（业务负载）名称。
total_select_elapse	bigint	总select的时间花费（单位：微秒）。
max_select_elapse	bigint	最大select的时间花费（单位：微秒）。
min_select_elapse	bigint	最小select的时间花费（单位：微秒）。
avg_select_elapse	bigint	平均select的时间花费（单位：微秒）。
total_update_elapse	bigint	总update的时间花费（单位：微秒）。
max_update_elapse	bigint	最大update的时间花费（单位：微秒）。
min_update_elapse	bigint	最小update的时间花费（单位：微秒）。
avg_update_elapse	bigint	平均update的时间花费（单位：微秒）。
total_insert_elapse	bigint	总insert的时间花费（单位：微秒）。
max_insert_elapse	bigint	最大insert的时间花费（单位：微秒）。
min_insert_elapse	bigint	最小insert的时间花费（单位：微秒）。

名称	类型	描述
avg_insert_elapse	bigint	平均insert的时间花费（单位：微秒）。
total_delete_elapse	bigint	总delete的时间花费（单位：微秒）。
max_delete_elapse	bigint	最大delete的时间花费（单位：微秒）。
min_delete_elapse	bigint	最小delete的时间花费（单位：微秒）。
avg_delete_elapse	bigint	平均delete的时间花费（单位：微秒）。

13.2.6.7 SUMMARY_WORKLOAD_SQL_ELAPSE_TIME

SUMMARY_WORKLOAD_SQL_ELAPSE_TIME用来统计所有CN节点上workload（业务）负载的SUID信息。

表 13-78 SUMMARY_WORKLOAD_SQL_ELAPSE_TIME 字段

名称	类型	描述
node_name	name	节点名称。
workload	name	workload（业务负载）名称。
total_select_elapse	bigint	总select的时间花费（单位：微秒）。
max_select_elapse	bigint	最大select的时间花费（单位：微秒）。
min_select_elapse	bigint	最小select的时间花费（单位：微秒）。
avg_select_elapse	bigint	平均select的时间花费（单位：微秒）。
total_update_elapse	bigint	总update的时间花费（单位：微秒）。
max_update_elapse	bigint	最大update的时间花费（单位：微秒）。
min_update_elapse	bigint	最小update的时间花费（单位：微秒）。
avg_update_elapse	bigint	平均update的时间花费（单位：微秒）。
total_insert_elapse	bigint	总insert的时间花费（单位：微秒）。
max_insert_elapse	bigint	最大insert的时间花费（单位：微秒）。
min_insert_elapse	bigint	最小insert的时间花费（单位：微秒）。
avg_insert_elapse	bigint	平均insert的时间花费（单位：微秒）。
total_delete_elapse	bigint	总delete的时间花费（单位：微秒）。
max_delete_elapse	bigint	最大delete的时间花费（单位：微秒）。
min_delete_elapse	bigint	最小delete的时间花费（单位：微秒）。
avg_delete_elapse	bigint	平均delete的时间花费（单位：微秒）。

13.2.6.8 USER_TRANSACTION

USER_TRANSACTION用来统计用户执行的事务信息。monadmin用户能看到所有用户执行事务的信息，普通用户只能查询到自己执行的事务信息。

表 13-79 USER_TRANSACTION 字段

名称	类型	描述
username	name	用户的名称。
commit_counter	bigint	用户事务commit数量。
rollback_counter	bigint	用户事务rollback数量。
resp_min	bigint	用户事务最小响应时间（单位：微秒）。
resp_max	bigint	用户事务最大响应时间（单位：微秒）。
resp_avg	bigint	用户事务平均响应时间（单位：微秒）。
resp_total	bigint	用户事务总响应时间（单位：微秒）。
bg_commit_counter	bigint	后台事务commit数量。
bg_rollback_counter	bigint	后台事务rollback数量。
bg_resp_min	bigint	后台事务最小响应时间（单位：微秒）。
bg_resp_max	bigint	后台事务最大响应时间（单位：微秒）。
bg_resp_avg	bigint	后台事务平均响应时间（单位：微秒）。
bg_resp_total	bigint	后台事务总响应时间（单位：微秒）。

13.2.6.9 GLOBAL_USER_TRANSACTION

GLOBAL_USER_TRANSACTION用来统计全局用户执行的事务信息。

表 13-80 GLOBAL_USER_TRANSACTION 字段

名称	类型	描述
node_name	name	节点名称。
username	name	用户的名称。
commit_counter	bigint	用户事务commit数量。
rollback_counter	bigint	用户事务rollback数量。
resp_min	bigint	用户事务最小响应时间（单位：微秒）。
resp_max	bigint	用户事务最大响应时间（单位：微秒）。

名称	类型	描述
resp_avg	bigint	用户事务平均响应时间(单位：微秒）。
resp_total	bigint	用户事务总响应时间（单位：微秒）。
bg_commit_counter	bigint	后台事务commit数量。
bg_rollback_counter	bigint	后台事务rollback数量。
bg_resp_min	bigint	后台事务最小响应时间（单位：微秒）。
bg_resp_max	bigint	后台事务最大响应时间（单位：微秒）。
bg_resp_avg	bigint	后台事务平均响应时间（单位：微秒）。
bg_resp_total	bigint	后台事务总响应时间（单位：微秒）。

13.2.7 Session/Thread

13.2.7.1 SESSION_STAT

当前节点以会话线程或AutoVacuum线程为单位，统计会话状态信息。

表 13-81 SESSION_STAT 字段

名称	类型	描述
sessid	text	线程启动时间+线程标识。
statid	integer	统计编号。
statname	text	统计会话名称。
statunit	text	统计会话单位。
value	bigint	统计会话值。

13.2.7.2 GLOBAL_SESSION_STAT

各节点上以会话线程或AutoVacuum线程为单位，统计会话状态信息。

表 13-82 GLOBAL_SESSION_STAT 字段

名称	类型	描述
node_name	name	节点名称。
sessid	text	线程启动时间+线程标识。

名称	类型	描述
statid	integer	统计编号。
statname	text	统计会话名称。
statunit	text	统计会话单位。
value	bigint	统计会话值。

13.2.7.3 SESSION_TIME

用于统计当前节点会话线程的运行时间信息，及各执行阶段所消耗时间。

表 13-83 SESSION_TIME 字段

名称	类型	描述
sessid	text	线程启动时间+线程标识。
stat_id	integer	统计编号。
stat_name	text	会话类型名称。
value	bigint	会话值。

13.2.7.4 GLOBAL_SESSION_TIME

用于统计各节点会话线程的运行时间信息，及各执行阶段所消耗时间，如表13-84所示。

表 13-84 GLOBAL_SESSION_TIME 字段

名称	类型	描述
node_name	name	节点名称。
sessid	text	线程启动时间+线程标识。
stat_id	integer	统计编号。
stat_name	text	会话类型名称。
value	bigint	会话值。

13.2.7.5 SESSION_MEMORY

统计Session级别的内存使用情况，包含执行作业在当前节点上GaussDB线程和Stream线程分配的所有内存，单位为MB，如表13-85所示。

表 13-85 SESSION_MEMORY 字段

名称	类型	描述
sessid	text	线程启动时间+线程标识。
init_mem	integer	当前正在执行作业进入执行器前已分配的内存。
used_mem	integer	当前正在执行作业已分配的内存。
peak_mem	integer	当前正在执行作业已分配的内存峰值。

13.2.7.6 GLOBAL_SESSION_MEMORY

统计各节点的Session级别的内存使用情况，包含执行作业在数据节点上GaussDB线程和Stream线程分配的所有内存，单位为MB，如表13-86所示。

表 13-86 GLOBAL_SESSION_MEMORY 字段

名称	类型	描述
node_name	name	节点名称。
sessid	text	线程启动时间+线程标识。
init_mem	integer	当前正在执行作业进入执行器前已分配的内存。
used_mem	integer	当前正在执行作业已分配的内存。
peak_mem	integer	当前正在执行作业已分配的内存峰值。

13.2.7.7 SESSION_MEMORY_DETAIL

统计线程的内存使用情况，以MemoryContext粒度来统计当前节点的内存，如表13-87所示。

表 13-87 SESSION_MEMORY_DETAIL 字段

名称	类型	描述
sessid	text	线程启动时间+线程标识。
sesstype	text	线程名称。
contextname	text	内存上下文名称。
level	smallint	内存上下文的重要级别。
parent	text	父级内存上下文名称。
totalsize	bigint	总申请内存大小(单位：字节)。
freesize	bigint	空闲内存大小(单位：字节)。

名称	类型	描述
usedsize	bigint	使用内存大小(单位：字节)。

13.2.7.8 GLOBAL_SESSION_MEMORY_DETAIL

统计各节点的线程的内存使用情况，以MemoryContext节点来统计。

表 13-88 GLOBAL_SESSION_MEMORY_DETAIL 字段

名称	类型	描述
node_name	name	节点名称。
sessid	text	线程启动时间+线程标识。
sesstype	text	线程名称。
contextname	text	内存上下文名称。
level	smallint	内存上下文的重要级别。
parent	text	父级内存上下文名称。
totalsize	bigint	总申请内存大小(单位：字节)。
freesize	bigint	空闲内存大小(单位：字节)。
usedsize	bigint	使用内存大小(单位：字节)。

13.2.7.9 THREAD_WAIT_STATUS

通过该视图可以检测当前节点中工作线程（backend thread）以及辅助线程（auxiliary thread）的阻塞等待情况，。

表 13-89 THREAD_WAIT_STATUS 字段

名称	类型	描述
node_name	text	当前节点的名称。
db_name	text	数据库名称。
thread_name	text	线程名称。
query_id	bigint	查询ID，对应debug_query_id。
tid	bigint	当前线程的线程号。
sessionid	bigint	session的ID。
lwtid	integer	当前线程的轻量级线程号。

名称	类型	描述
psessionid	bigint	streaming线程的父线程。
tlevel	integer	streaming线程的层级。
smpid	integer	并行线程的ID。
wait_status	text	当前线程的等待状态。
wait_event	text	如果wait_status是acquire lock、acquire lwlock、wait io三种类型，此列描述具体的锁、轻量级锁、I/O的信息；否则为空。
locktag	text	当前线程正在等待锁的信息。
lockmode	text	当前线程正等待获取的锁模式。包含表级锁、行级锁、页级锁下的各模式。
block_sessionid	bigint	阻塞当前线程获取锁的会话标识。
global_sessionid	text	全局会话ID。

13.2.7.10 GLOBAL_THREAD_WAIT_STATUS

通过该视图可以检测所有节点上工作线程（backend thread）以及辅助线程（auxiliary thread）的阻塞等待情况。

通过GLOBAL_THREAD_WAIT_STATUS视图，可以查看集群全局各个节点上所有SQL语句产生的线程之间的调用层次关系，以及各个线程的阻塞等待状态，从而更容易定位hang以及类似现象的原因。

GLOBAL_THREAD_WAIT_STATUS视图和THREAD_WAIT_STATUS视图列定义完全相同，这是由于GLOBAL_THREAD_WAIT_STATUS视图本质是到集群中各个节点上查询THREAD_WAIT_STATUS视图汇总的结果。

表 13-90 GLOBAL_THREAD_WAIT_STATUS 字段

名称	类型	描述
node_name	text	节点名称。
db_name	text	数据库名称。
thread_name	text	线程名称。
query_id	bigint	查询ID，对应debug_query_id。
tid	bigint	当前线程的线程号。
sessionid	bigint	session的ID。
lwtid	integer	当前线程的轻量级线程号。

名称	类型	描述
psessionid	bigint	streaming线程的父线程。
tlevel	integer	streaming线程的层级。
smpid	integer	并行线程的ID。
wait_status	text	当前线程的等待状态。
wait_event	text	如果wait_status是acquire lock、acquire lwlock、wait io三种类型，此列描述具体的锁、轻量级锁、I/O的信息。否则是空。
locktag	text	当前线程正在等待锁的信息。
lockmode	text	当前线程正等待获取的锁模式。包含表级锁、行级锁、页级锁下的各模式。
block_sessionid	bigint	阻塞当前线程获取锁的会话标识。
global_sessionid	text	全局会话ID。

13.2.7.11 LOCAL_THREADPOOL_STATUS

LOCAL_THREADPOOL_STATUS视图显示线程池中工作线程及会话的状态信息。该视图仅在线程池开启（enable_thread_pool = on）时生效。

表 13-91 LOCAL_THREADPOOL_STATUS 字段

名称	类型	描述
node_name	text	节点名称。
group_id	integer	线程池组ID。
bind_numa_id	integer	该线程池组绑定的NUMA ID。
bind_cpu_number	integer	该线程池组绑定的CPU信息。如果未绑定CPU，该值为NULL。
listener	integer	该线程池组的Listener线程数量。
worker_info	text	线程池中线程相关信息，包括以下信息： ● default：该线程池组中的初始线程数量。 ● new：该线程池组中新增线程的数量。 ● expect：该线程池组中预期线程的数量。 ● actual：该线程池组中实际线程的数量。 ● idle：该线程池组中空闲线程的数量。 ● pending：该线程池组中等待线程的数量。

名称	类型	描述
session_info	text	线程池中会话相关信息，包括以下信息： ● total：该线程池组中所有的会话数量。 ● waiting：该线程池组中等待调度的会话数量。 ● running：该线程池中正在执行的会话数量。 ● idle：该线程池组中空闲的会话数量。
stream_info	text	Stream线程池相关信息，包括以下信息： ● total：该线程池组中所有的stream线程数量。 ● running：该线程池组中当前正在工作的stream线程数量。 ● idle：该线程池组中空闲的stream线程数量。

13.2.7.12 GLOBAL_THREADPOOL_STATUS

GLOBAL_THREADPOOL_STATUS视图显示在所有节点上的线程池中工作线程及会话的状态信息。具体的字段表13-91。

13.2.7.13 SESSION_CPU_RUNTIME

SESSION_CPU_RUNTIME视图显示当前用户执行复杂作业（正在运行）时的CPU使用信息。

表 13-92 SESSION_CPU_RUNTIME 字段

名称	类型	描述
datid	oid	连接后端的数据库OID。
usename	name	登录到该后端的用户名。
pid	bigint	后端线程ID。
start_time	timestamp with time zone	语句执行的开始时间。
min_cpu_time	bigint	语句在所有DN上的最小CPU时间，单位为ms。
max_cpu_time	bigint	语句在所有DN上的最大CPU时间，单位为ms。
total_cpu_time	bigint	语句在所有DN上的CPU总时间，单位为ms。

名称	类型	描述
query	text	正在执行的语句。
node_group	text	语句所属用户对应的Node group。
top_cpu_dn	text	cpu使用量topN信息。

13.2.7.14 SESSION_MEMORY_RUNTIME

SESSION_MEMORY_RUNTIME视图显示当前用户执行复杂作业（正在运行）时的内存使用信息。

表 13-93 SESSION_MEMORY_RUNTIME 字段

名称	类型	描述
datid	oid	连接后端的数据库OID。
usename	name	登录到该后端的用户名。
pid	bigint	后端线程ID。
start_time	timestamp with time zone	语句执行的开始时间。如果是存储过程、函数、PACKAGE，则查询的是第一个查询时间，不会随着存储过程内语句运行而改变。
min_peak_memory	integer	语句在所有DN上的最小内存峰值大小，单位MB。
max_peak_memory	integer	语句在所有DN上的最大内存峰值大小，单位MB。
spill_info	text	语句在所有DN上的下盘信息： ● None：所有DN均未下盘。 ● All：所有DN均下盘。 ● [a:b]：数量为b个DN中有a个DN下盘。
query	text	正在执行的语句。
node_group	text	语句所属用户对应的Node group。
top_mem_dn	text	mem使用量topN信息。

13.2.7.15 LOCAL_ACTIVE_SESSION

LOCAL_ACTIVE_SESSION视图显示本节点上的ACTIVE SESSION PROFILE内存中的样本。

表 13-94 LOCAL_ACTIVE_SESSION 字段

名称	类型	描述
sampleid	bigint	采样ID。
sample_time	timestamp with time zone	采样的时间。
need_flush_sample	boolean	该样本是否需要刷新到磁盘。
databaseid	oid	数据库ID
thread_id	bigint	线程的ID。
sessionid	bigint	会话的ID。
start_time	timestamp with time zone	会话的启动时间。
event	text	具体的事件名称。
lwtid	integer	当前线程的轻量级线程号。
psessionid	bigint	streaming线程的父线程。
tlevel	integer	streaming线程的层级。与执行计划的层级(id)相对应。
smpid	integer	smp执行模式下并行线程的并行编号。
userid	oid	session用户的id。
application_name	text	应用的名称。
client_addr	inet	client端的地址。
client_hostname	text	client端的名称。
client_port	integer	客户端用于与后端通讯的TCP端口号。
query_id	bigint	debug query id
unique_query_id	bigint	unique query id
user_id	oid	unique query的key中的user_id。
cn_id	integer	cn id，在DN上表示该unique sql来自该CN节点，unique query的key中的cn_id。
unique_query	text	规范化后的UniqueSQL文本串。
locktag	text	会话等待锁信息，可通过locktag_decode解析。

名称	类型	描述
lockmode	text	会话等待锁模式。
block_sessionid	bigint	如果会话正在等待锁，阻塞该会话获取锁的会话标识。
final_block_sessionid	bigint	表示源头阻塞会话id。
wait_status	text	描述event列的更多详细信息。
global_sessionid	text	全局会话ID。
xact_start_time	timestamp with time zone	事务开始时间。
query_start_time	timestamp with time zone	语句开始执行时间。
state	text	当前语句状态。 可能取值为：active、idle in transaction、fastpath function call、idle in transaction (aborted)、disabled、retrying。

13.2.7.16 GLOBAL_ACTIVE_SESSION

GLOBAL_ACTIVE_SESSION视图显示所有节点上的ACTIVE SESSION PROFILE内存中的样本的汇总。

表 13-95 GLOBAL_ACTIVE_SESSION 字段

名称	类型	描述
node_name	text	节点名称。
sampleid	bigint	采样ID。
sample_time	timestamp without time zone	采样的时间。
need_flush_sample	boolean	该样本是否需要刷新的磁盘。
databaseid	oid	数据库ID。
thread_id	bigint	线程的ID。
sessionid	bigint	会话的ID。
start_time	timestamp without time zone	会话的启动时间。
event	text	具体的事件名称。

名称	类型	描述
lwtid	integer	当前线程的轻量级线程号。
psessionid	bigint	streaming线程的父线程。
tlevel	integer	streaming线程的层级。与执行计划的层级(id)相对应。
smpid	integer	smp执行模式下并行线程的并行编号。
userid	oid	session用户的id。
application_name	text	应用的名称。
client_addr	inet	client端的地址。
client_hostname	text	client端的名称。
client_port	integer	客户端用于与后端通讯的TCP端口号。
query_id	bigint	debug query id。
unique_query_id	bigint	unique query id。
user_id	oid	unique query的key中的user_id。
cn_id	integer	cn id，在DN上表示该unique sql来自该CN节点，unique query的key中的cn_id。
unique_query	text	规范化后的UniqueSQL文本串。
locktag	text	会话等待锁信息，可通过locktag_decode解析。
lockmode	text	会话等待锁模式。
block_sessionid	bigint	如果会话正在等待锁，阻塞该会话获取锁的会话标识。
final_block_sessionid	bigint	表示源头阻塞会话id。
wait_status	text	描述event列的更多详细信息。
global_sessionid	text	全局会话ID。
xact_start_time	timestamp with time zone	事务开始时间。
query_start_time	timestamp with time zone	语句开始执行时间。

名称	类型	描述
state	text	当前语句状态。 可能取值为：active、idle in transaction、fastpath function call、idle in transaction (aborted)、disabled、retrying。

13.2.8 Transaction

13.2.8.1 TRANSACTIONS_RUNNING_XACTS

显示当前节点运行事务的信息。

表 13-96 TRANSACTIONS_RUNNING_XACTS 字段

名称	类型	描述
handle	integer	事务在GTM对应的句柄。
gxid	xid	事务id号。
state	tinyint	事务状态（3：prepared或者0：starting）。
node	text	节点名称。
xmin	xid	节点上当前数据涉及的最小事务号xmin。
vacuum	boolean	标志当前事务是否是lazy vacuum事务（lazy vacuum是一种vacuum机制，在需要时进行vacuum）。 ● true：表示是。 ● false：表示否。
timeline	bigint	标志数据库重启次数。
prepare_xid	xid	处于prepared状态的事务的id号，若不在prepared状态，值为0。
pid	bigint	事务对应的线程id。
next_xid	xid	CN传给DN的事务id号。

13.2.8.2 SUMMARY_TRANSACTIONS_RUNNING_XACTS

显示集群中各个CN节点运行事务的信息汇总，

表 13-97 SUMMARY_TRANSACTIONS_RUNNING_XACTS 字段

名称	类型	描述
handle	integer	事务在GTM对应的句柄。
gxid	xid	事务id号。
state	tinyint	事务状态（3：prepared或者0：starting）。
node	text	节点名称。
xmin	xid	节点上当前数据涉及的最小事务号xmin。
vacuum	boolean	标志当前事务是否是lazy vacuum事务（lazy vacuum是一种vacuum机制，在需要时进行vacuum）。 ● true：表示是。 ● false：表示否。
timeline	bigint	标志数据库重启次数。
prepare_xid	xid	处于prepared状态的事务的id号，若不在prepared状态，值为0。
pid	bigint	事务对应的线程id。
next_xid	xid	CN传给DN的事务id号。

13.2.8.3 GLOBAL_TRANSACTIONS_RUNNING_XACTS

显示集群中各个CN和DN节点运行事务的信息。

表 13-98 GLOBAL_TRANSACTIONS_RUNNING_XACTS 字段

名称	类型	描述
handle	integer	事务在GTM对应的句柄。
gxid	xid	事务id号。
state	tinyint	事务状态（3：prepared或者0：starting）。
node	text	节点名称。
xmin	xid	节点上当前数据涉及的最小事务号xmin。
vacuum	boolean	标志当前事务是否是lazy vacuum事务（lazy vacuum是一种vacuum机制，在需要时进行vacuum）。 ● true：表示是。 ● false：表示否。
timeline	bigint	标志数据库重启次数。
prepare_xid	xid	处于prepared状态的事务的id号，若不在prepared状态，值为0。

名称	类型	描述
pid	bigint	事务对应的线程id。
next_xid	xid	CN传给DN的事务id号。

13.2.8.4 TRANSACTIONS_PREPARED_XACTS

显示当前准备好进行两阶段提交的事务的信息。

表 13-99 TRANSACTIONS_PREPARED_XACTS 字段

名称	类型	描述
transaction	xid	预备事务的数字事务标识。
gid	text	赋予该事务的全局事务标识。
prepared	timestamp with time zone	事务准备好提交的时间。
owner	name	执行该事务的用户的名称。
database	name	执行该事务所在的数据库名。

13.2.8.5 SUMMARY_TRANSACTIONS_PREPARED_XACTS

显示集群中各CN节点当前准备好进行两阶段提交的事务的信息汇总。

表 13-100 SUMMARY_TRANSACTIONS_PREPARED_XACTS 字段

名称	类型	描述
transaction	xid	预备事务的数字事务标识。
gid	text	赋予该事务的全局事务标识。
prepared	timestamp with time zone	事务准备好提交的时间。
owner	name	执行该事务的用户的名称。
database	name	执行该事务所在的数据库名。

13.2.8.6 GLOBAL_TRANSACTIONS_PREPARED_XACTS

显示集群中各个CN和DN节点当前准备好进行两阶段提交的事务的信息汇总。

表 13-101 GLOBAL_TRANSACTIONS_PREPARED_XACTS 字段

名称	类型	描述
transaction	xid	预备事务的数字事务标识。
gid	text	赋予该事务的全局事务标识。
prepared	timestamp with time zone	事务准备好提交的时间。
owner	name	执行该事务的用户的名称。
database	name	执行该事务所在的数据库名。

13.2.9 Query

13.2.9.1 STATEMENT

获得当前节点的执行语句(归一化SQL)的信息。查询视图必须具有sysadmin权限或者monitor admin权限。CN上可以看到此CN接收到的归一化的SQL的全量统计信息（包含DN）；DN上仅可看到归一化的SQL的此节点执行的统计信息。

说明

不同的savepoint_name所生成的unique_sql_id不同，大量使用savepoint_name时会导致系统中产生的unique_sql_id信息快速上涨，若unique_sql_id数量高于instr_unique_sql_count数量时，新产生的unique_sql_id信息将不被统计。

表 13-102 STATEMENT 字段

名称	类型	描述
node_name	name	节点名称。
node_id	integer	节点的ID(pgxc_node中的node_id)。
user_name	name	用户名称。
user_id	oid	用户OID。
unique_sql_id	bigint	归一化的SQL ID。
query	text	归一化的SQL。 备注：长度受track_activity_query_size控制。
n_calls	bigint	调用次数。
min_elapse_time	bigint	SQL在内核内的最小运行时间（单位：微秒）。
max_elapse_time	bigint	SQL在内核内的最大运行时间（单位：微秒）。

名称	类型	描述
total_elapse_time	bigint	SQL在内核内的总运行时间（单位：微秒）。
n_returned_rows	bigint	SELECT返回的结果集行数。
n_tuples_fetched	bigint	随机扫描行。
n_tuples_returned	bigint	顺序扫描行。
n_tuples_inserted	bigint	插入行。
n_tuples_updated	bigint	更新行。
n_tuples_deleted	bigint	删除行。
n_blocks_fetched	bigint	逻辑读次数。
n_blocks_hit	bigint	内存中命中次数。
n_soft_parse	bigint	软解析次数, n_soft_parse + n_hard_parse可能大于n_calls, 因为子查询未计入n_calls。
n_hard_parse	bigint	硬解析次数, n_soft_parse + n_hard_parse可能大于n_calls, 因为子查询未计入n_calls。
db_time	bigint	有效的数据库内部时间花费，多线程将累加（单位：微秒）。
cpu_time	bigint	CPU时间（单位：微秒）。
execution_time	bigint	执行器内执行时间（单位：微秒）。
parse_time	bigint	SQL解析时间（单位：微秒）。
plan_time	bigint	SQL生成计划时间（单位：微秒）。
rewrite_time	bigint	SQL重写时间（单位：微秒）。
pl_execution_time	bigint	plpgsql上的执行时间（单位：微秒）。
pl_compilation_time	bigint	plpgsql上的编译时间（单位：微秒）。
data_io_time	bigint	I/O上的时间花费（单位：微秒）。
net_send_info	text	通过物理连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。在分布式数据库中，CN与CN、CN与客服端以及CN与DN之间都是通过物理连接进行通信，通过该字段可以分析SQL在分布式系统下的网络开销。 例如：{"time":xxx, "n_calls":xxx, "size":xxx}

名称	类型	描述
net_recv_info	text	通过物理连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。在分布式数据库中，CN与CN、CN与客服端以及CN与DN之间都是通过物理连接进行通信，通过该字段可以分析SQL在分布式系统下的网络开销。 例如：{"time":xxx, "n_calls":xxx, "size":xxx}
net_stream_send_info	text	通过逻辑连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。在分布式数据库中，不同分片的DN之间通过逻辑连接进行通信，通过该字段可以分析SQL在分布式系统下的网络开销。 例如：{"time":xxx, "n_calls":xxx, "size":xxx}
net_stream_recv_info	text	通过逻辑连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。在分布式数据库中，不同分片的DN之间通过逻辑连接进行通信，通过该字段可以分析SQL在分布式系统下的网络开销。 例如：{"time":xxx, "n_calls":xxx, "size":xxx}
last_updated	timestamp with time zone	最后一次更新该语句的时间。
sort_count	bigint	排序执行的次数。
sort_time	bigint	排序执行的时间（单位：微秒）。
sort_mem_used	bigint	排序过程中使用的work memory大小（单位：KB）。
sort_spill_count	bigint	排序过程中，若发生落盘，写文件的次数。
sort_spill_size	bigint	排序过程中，若发生落盘，使用的文件大小（单位：KB）。
hash_count	bigint	hash执行的次数。
hash_time	bigint	hash执行的时间（单位：微秒）。
hash_mem_used	bigint	hash过程中使用的work memory大小（单位：KB）。
hash_spill_count	bigint	hash过程中，若发生落盘，写文件的次数。
hash_spill_size	bigint	hash过程中，若发生落盘，使用的文件大小（单位：KB）。

13.2.9.2 SUMMARY_STATEMENT

获得各CN节点的执行语句(归一化SQL)的全量信息(包含DN)。

表 13-103 SUMMARY_STATEMENT 字段

名称	类型	描述
node_name	name	节点名称。
node_id	integer	节点的ID(pgxc_node中的node_id)。
user_name	name	用户名称。
user_id	oid	用户OID。
unique_sql_id	bigint	归一化的SQL ID。
query	text	归一化的SQL。 备注：长度受track_activity_query_size控制。
n_calls	bigint	调用次数。
min_elapse_time	bigint	SQL在内核内的最小运行时间（单位：微秒）。
max_elapse_time	bigint	SQL在内核内的最大运行时间（单位：微秒）。
total_elapse_time	bigint	SQL在内核内的总运行时间（单位：微秒）。
n_returned_rows	bigint	SELECT返回的结果集行数。
n_tuples_fetched	bigint	随机扫描行。
n_tuples_returned	bigint	顺序扫描行。
n_tuples_inserted	bigint	插入行。
n_tuples_updated	bigint	更新行。
n_tuples_deleted	bigint	删除行。
n_blocks_fetched	bigint	逻辑读次数。
n_blocks_hit	bigint	内存中命中次数。
n_soft_parse	bigint	软解析次数。
n_hard_parse	bigint	硬解析次数。
db_time	bigint	有效的数据库内部时间花费，多线程将累加（单位：微秒）。
cpu_time	bigint	CPU时间（单位：微秒）。
execution_time	bigint	执行器内执行时间（单位：微秒）。
parse_time	bigint	SQL解析时间（单位：微秒）。
plan_time	bigint	SQL生成计划时间（单位：微秒）。
rewrite_time	bigint	SQL重写时间（单位：微秒）。

名称	类型	描述
pl_execution_time	bigint	plpgsql上的执行时间（单位：微秒）。
pl_compilation_time	bigint	plpgsql上的编译时间（单位：微秒）。
data_io_time	bigint	I/O上的时间花费（单位：微秒）。
net_send_info	text	通过物理连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。在分布式数据库中，CN与CN、CN与客服端以及CN与DN之间都是通过物理连接进行通信，通过该字段可以分析SQL在分布式系统下的网络开销。 例如：{"time":xxx, "n_calls":xxx, "size":xxx}
net_recv_info	text	通过物理连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。在分布式数据库中，CN与CN、CN与客服端以及CN与DN之间都是通过物理连接进行通信，通过该字段可以分析SQL在分布式系统下的网络开销。 例如：{"time":xxx, "n_calls":xxx, "size":xxx}
net_stream_send_info	text	通过逻辑连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。在分布式数据库中，不同分片的DN之间通过逻辑连接进行通信，通过该字段可以分析SQL在分布式系统下的网络开销。 例如：{"time":xxx, "n_calls":xxx, "size":xxx}
net_stream_recv_info	text	通过逻辑连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。在分布式数据库中，不同分片的DN之间通过逻辑连接进行通信，通过该字段可以分析SQL在分布式系统下的网络开销。 例如：{"time":xxx, "n_calls":xxx, "size":xxx}
last_updated	timestamp with time zone	最后一次更新该语句的时间。
sort_count	bigint	排序执行的次数。
sort_time	bigint	排序执行的时间（单位：微秒）。
sort_mem_used	bigint	排序过程中使用的work memory大小（单位：KB）。
sort_spill_count	bigint	排序过程中，若发生落盘，写文件的次数。
sort_spill_size	bigint	排序过程中，若发生落盘，使用的文件大小（单位：KB）。

名称	类型	描述
hash_count	bigint	hash执行的次数。
hash_time	bigint	hash执行的时间（单位：微秒）。
hash_mem_used	bigint	hash过程中使用的work memory大小（单位：KB）。
hash_spill_count	bigint	hash过程中，若发生落盘，写文件的次数。
hash_spill_size	bigint	hash过程中，若发生落盘，使用的文件大小（单位：KB）。

13.2.9.3 STATEMENT_COUNT

显示数据库当前节点当前时刻执行的五类语句（SELECT、INSERT、UPDATE、DELETE、MERGE INTO）和(DDL、DML、DCL)统计信息。

说明

普通用户查询STATEMENT_COUNT视图仅能看到该用户当前节点的统计信息；管理员权限用户查询STATEMENT_COUNT视图则能看到所有用户当前节点的统计信息。当集群或该节点重启时，计数将清零，并重新开始计数。计数以节点收到的查询数为准，包括集群内部进行的查询。例如，CN收到一条查询，如果一条查询语句包含多个子查询，这些子查询内容下发到DN时会被分别计数（无论单个DN还是多个DN）。

表 13-104 STATEMENT_COUNT 字段

名称	类型	描述
node_name	text	节点名称。
user_name	text	用户名。
select_count	bigint	select语句统计结果。
update_count	bigint	update语句统计结果。
insert_count	bigint	insert语句统计结果。
delete_count	bigint	delete语句统计结果。
mergeinto_count	bigint	merge into语句统计结果。
ddl_count	bigint	DDL语句的数量。
dml_count	bigint	DML语句的数量。
dcl_count	bigint	DCL语句的数量。
total_select_elapse	bigint	总select的时间花费（单位：微秒）。
avg_select_elapse	bigint	平均select的时间花费（单位：微秒）。
max_select_elapse	bigint	最大select的时间花费(单位：微秒)。

名称	类型	描述
min_select_elapse	bigint	最小select的时间花费（单位：微秒）。
total_update_elapse	bigint	总update的时间花费（单位：微秒）。
avg_update_elapse	bigint	平均update的时间花费(单位：微秒)。
max_update_elapse	bigint	最大update的时间花费（单位：微秒）。
min_update_elapse	bigint	最小update的时间花费（单位：微秒）。
total_insert_elapse	bigint	总insert的时间花费（单位：微秒）。
avg_insert_elapse	bigint	平均insert的时间花费（单位：微秒）。
max_insert_elapse	bigint	最大insert的时间花费（单位：微秒）。
min_insert_elapse	bigint	最小insert的时间花费（单位：微秒）。
total_delete_elapse	bigint	总delete的时间花费（单位：微秒）。
avg_delete_elapse	bigint	平均delete的时间花费（单位：微秒）。
max_delete_elapse	bigint	最大delete的时间花费（单位：微秒）。
min_delete_elapse	bigint	最小delete的时间花费（单位：微秒）。

13.2.9.4 GLOBAL_STATEMENT_COUNT

显示数据库各节点当前时刻执行的五类语句（SELECT、INSERT、UPDATE、DELETE、MERGE INTO）和(DDL、DML、DCL)统计信息。

表 13-105 GLOBAL_STATEMENT_COUNT 字段

名称	类型	描述
node_name	text	节点名称。
user_name	text	用户名。
select_count	bigint	select语句统计结果。
update_count	bigint	update语句统计结果。
insert_count	bigint	insert语句统计结果。
delete_count	bigint	delete语句统计结果。
mergeinto_count	bigint	merge into语句统计结果。
ddl_count	bigint	DDL语句的数量。
dml_count	bigint	DML语句的数量。
dcl_count	bigint	DCL语句的数量。

名称	类型	描述
total_select_elapse	bigint	总select的时间花费（单位：微秒）。
avg_select_elapse	bigint	平均select的时间花费（单位：微秒）。
max_select_elapse	bigint	最大select的时间花费(单位：微秒)。
min_select_elapse	bigint	最小select的时间花费（单位：微秒）。
total_update_elapse	bigint	总update的时间花费(单位：微秒)。
avg_update_elapse	bigint	平均update的时间花费（单位：微秒）。
max_update_elapse	bigint	最大update的时间花费（单位：微秒）。
min_update_elapse	bigint	最小update的时间花费（单位：微秒）。
total_insert_elapse	bigint	总insert的时间花费（单位：微秒）。
avg_insert_elapse	bigint	平均insert的时间花费（单位：微秒）。
max_insert_elapse	bigint	最大insert的时间花费(单位：微秒)。
min_insert_elapse	bigint	最小insert的时间花费（单位：微秒）。
total_delete_elapse	bigint	总delete的时间花费（单位：微秒）。
avg_delete_elapse	bigint	平均delete的时间花费(单位：微秒)。
max_delete_elapse	bigint	最大delete的时间花费（单位：微秒）。
min_delete_elapse	bigint	最小delete的时间花费（单位：微秒）。

13.2.9.5 SUMMARY_STATEMENT_COUNT

显示数据库各节点执行的五类语句（SELECT、INSERT、UPDATE、DELETE、MERGE INTO）和（DDL、DML、DCL）的汇总统计信息。

表 13-106 SUMMARY_STATEMENT_COUNT 字段

名称	类型	描述
user_name	text	用户名。
select_count	numeric	select语句统计结果。
update_count	numeric	update语句统计结果。
insert_count	numeric	insert语句统计结果。
delete_count	numeric	delete语句统计结果。
mergeinto_count	numeric	merge into语句统计结果。
ddl_count	numeric	DDL语句的数量。

名称	类型	描述
dml_count	numeric	DML语句的数量。
dcl_count	numeric	DCL语句的数量。
total_select_elapse	numeric	总select的时间花费（单位：微秒）。
avg_select_elapse	bigint	平均select的时间花费（单位：微秒）。
max_select_elapse	bigint	最大select的时间花费（单位：微秒）。
min_select_elapse	bigint	最小select的时间花费（单位：微秒）。
total_update_elapse	numeric	总update的时间花费（单位：微秒）。
avg_update_elapse	bigint	平均update的时间花费（单位：微秒）。
max_update_elapse	bigint	最大update的时间花费（单位：微秒）。
min_update_elapse	bigint	最小update的时间花费（单位：微秒）。
total_insert_elapse	numeric	总insert的时间花费(单位：微秒)。
avg_insert_elapse	bigint	平均insert的时间花费（单位：微秒）。
max_insert_elapse	bigint	最大insert的时间花费（单位：微秒）。
min_insert_elapse	bigint	最小insert的时间花费（单位：微秒）。
total_delete_elapse	numeric	总delete的时间花费（单位：微秒）。
avg_delete_elapse	bigint	平均delete的时间花费（单位：微秒）。
max_delete_elapse	bigint	最大delete的时间花费（单位：微秒）。
min_delete_elapse	bigint	最小delete的时间花费（单位：微秒）。

13.2.9.6 STATEMENT_RESPONSETIME_PERCENTILE

获取集群SQL响应时间P80，P95分布信息。

表 13-107 STATEMENT_RESPONSETIME_PERCENTILE 的字段

名称	类型	描述
p80	bigint	集群80%的SQL的响应时间（单位：微秒）。
p95	bigint	集群95%的SQL的响应时间（单位：微秒）。

13.2.9.7 GS_SLOW_QUERY_INFO（废弃）

GS_SLOW_QUERY_INFO视图显示当前节点上已经转储的慢查询信息。此数据是从内核中转储到系统表中的数据。当设置GUC参数enable_resource_record为on时，系统

会定时（周期为3分钟）将内核中query信息导入GS_WLM_SESSION_QUERY_INFO_ALL系统表，开启此功能会占用系统存储空间并对性能有一定影响。用户通过查询GS_SLOW_QUERY_INFO视图，可以查看已经转储的慢查询信息，本版本中已废弃。

表 13-108 GS_SLOW_QUERY_INFO 字段

名称	类型	描述
dbname	text	数据库名称。
schemaname	text	schema名称。
nodename	text	节点名称。
username	text	用户名。
queryid	bigint	归一化ID。
query	text	用户语句。
start_time	timestamp with time zone	开始执行时间。
finish_time	timestamp with time zone	结束执行时间。
duration	bigint	执行持续时间（毫秒）。
query_plan	text	计划信息。
n_returned_rows	bigint	Select返回的结果集行数。
n_tuples_fetched	bigint	随机扫描行数。
n_tuples_returned	bigint	顺序扫描行数。
n_tuples_inserted	bigint	插入行数。
n_tuples_updated	bigint	更新行数。
n_tuples_deleted	bigint	删除行数。
n_blocks_fetched	bigint	缓存加载次数。
n_blocks_hit	bigint	缓存命中数。
db_time	bigint	有效的数据库时间花费，多线程将累加（单位：微秒）。
cpu_time	bigint	CPU消耗时间（单位：微秒）。
execution_time	bigint	执行器内执行时间（单位：微秒）。

名称	类型	描述
parse_time	bigint	SQL解析时间（单位：微秒）。
plan_time	bigint	SQL生成计划时间（单位：微秒）。
rewrite_time	bigint	SQL重写时间（单位：微秒）。
pl_execution_time	bigint	plpgsql上的执行时间（单位：微秒）。
pl_compilation_time	bigint	plpgsql上的编译时间（单位：微秒）。
net_send_time	bigint	网络上的时间花费（单位：微秒）。
data_io_time	bigint	I/O上的时间花费(单位：微秒)。

13.2.9.8 GS_SLOW_QUERY_HISTORY（废弃）

GS_SLOW_QUERY_HISTORY显示当前节点上未转储的慢查询信息。具体字段信息请参考18.9.15 GS_SLOW_QUERY_INFO。该视图只有system admin和monitor admin用户有权限查询，本版本中已废弃。

13.2.9.9 GLOBAL_SLOW_QUERY_HISTORY（废弃）

GS_SLOW_QUERY_HISTORY显示所有节点上未转储的慢查询信息，本版本中已废弃。具体字段信息请参考18.9.15 GS_SLOW_QUERY_INFO。

13.2.9.10 GLOBAL_SLOW_QUERY_INFO（废弃）

GS_SLOW_QUERY_HISTORY显示所有节点上已经转储的慢查询信息，本版本中已废弃。具体字段信息请参考18.9.15 GS_SLOW_QUERY_INFO。

13.2.9.11 STATEMENT_HISTORY

获得当前节点的执行语句的信息。查询视图必须具有sysadmin权限或者monitor admin权限。只可在系统库中查询到结果，用户库中无法查询。

表 13-109 STATEMENT_HISTORY 字段

名称	类型	描述
dbname	name	数据库名称。
schemaname	name	schema名称。
origin_node	integer	节点名称。

名称	类型	描述
user_name	name	用户名。
application_name	text	用户发起的请求的应用程序名称。
client_addr	text	用户发起的请求的客户端地址。
client_port	integer	用户发起的请求的客户端端口。
unique_query_id	bigint	归一化SQL ID。
debug_query_id	bigint	唯一SQL ID。
query	text	归一化SQL(仅CN上有值)。
start_time	timestamp with time zone	语句启动的时间。
finish_time	timestamp with time zone	语句结束的时间。
slow_sql_threshold	bigint	语句执行时慢SQL的标准。
transaction_id	bigint	事务ID。
thread_id	bigint	执行线程ID。
session_id	bigint	用户session id。
n_soft_parse	bigint	软解析次数, n_soft_parse + n_hard_parse可能大于n_calls, 因为子查询未计入n_calls。
n_hard_parse	bigint	硬解析次数, n_soft_parse + n_hard_parse可能大于n_calls, 因为子查询未计入n_calls。
query_plan	text	语句执行计划。
n_returned_rows	bigint	SELECT返回的结果集行数。
n_tuples_fetched	bigint	随机扫描行。
n_tuples_returned	bigint	顺序扫描行。
n_tuples_inserted	bigint	插入行。
n_tuples_updated	bigint	更新行。
n_tuples_deleted	bigint	删除行。

名称	类型	描述
n_blocks_fetched	bigint	buffer的块访问次数。
n_blocks_hit	bigint	buffer的块命中次数。
db_time	bigint	有效的DB时间花费，多线程将累加（单位：微秒）。
cpu_time	bigint	CPU消耗时间（单位：微秒）。
execution_time	bigint	执行器内执行时间（单位：微秒）。
parse_time	bigint	SQL解析时间（单位：微秒）。
plan_time	bigint	SQL生成计划时间（单位：微秒）。
rewrite_time	bigint	SQL重写时间（单位：微秒）。
pl_execution_time	bigint	plpgsql上的执行时间（单位：微秒）。
pl_compilation_time	bigint	plpgsql上的编译时间（单位：微秒）。
data_io_time	bigint	I/O上的时间花费（单位：微秒）。
net_send_info	text	通过物理连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。在分布式数据库中，CN与CN、CN与客服端以及CN与DN之间都是通过物理连接进行通信，通过该字段可以分析SQL在分布式系统下的网络开销。例如：{"time":xxx, "n_calls":xxx, "size":xxx}。
net_recv_info	text	通过物理连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。在分布式数据库中，CN与CN、CN与客服端以及CN与DN之间都是通过物理连接进行通信，通过该字段可以分析SQL在分布式系统下的网络开销。例如：{"time":xxx, "n_calls":xxx, "size":xxx}。
net_stream_send_info	text	通过逻辑连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。在分布式数据库中，不同分片的DN之间通过逻辑连接进行通信，通过该字段可以分析SQL在分布式系统下的网络开销。 例如：{"time":xxx, "n_calls":xxx, "size":xxx}。

名称	类型	描述
net_stream_recv_info	text	通过逻辑连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。在分布式数据库中，不同分片的DN之间通过逻辑连接进行通信，通过该字段可以分析SQL在分布式系统下的网络开销。 例如：{"time":xxx, "n_calls":xxx, "size":xxx}。
lock_count	bigint	加锁次数。
lock_time	bigint	加锁耗时。
lock_wait_count	bigint	加锁等待次数。
lock_wait_time	bigint	加锁等待耗时。
lock_max_count	bigint	最大持锁数量。
lwlock_count	bigint	轻量级加锁次数（预留）。
lwlock_wait_count	bigint	轻量级等锁次数。
lwlock_time	bigint	轻量级加锁时间（预留）。
lwlock_wait_time	bigint	轻量级等锁时间。
details	bytea	语句锁事件的列表，该列表按时间顺序记录事件，记录的数量受参数track_stmt_details_size的影响, 该字段为二进制，需要借助解析函数pg_catalog.statement_detail_decode读取，见（**其它函数**） 事件包括： 加锁开始 加锁结束 等锁开始 等锁结束 放锁开始 放锁结束 轻量级等锁开始 轻量级等锁结束
is_slow_sql	boolean	该SQL是否为slow SQL。
trace_id	text	驱动传入的trace id，与应用的一次请求相关联。

名称	类型	描述
advise	text	可能导致该SQL为slow SQL的风险信息（可能同时存在多种风险）。 ● Cast Function Cause Index Miss.：表示存在隐式转换导致索引匹配失败的风险。 ● Limit too much rows.：表示存在limit值过大导致SQL变慢的风险。 ● Proleakproof of function is false.：表示函数的proleakproof值为false，此时函数在生成计划时因存在数据泄露的风险而不会使用统计信息，影响生成计划的准确性，从而存在SQL变慢的风险。

13.2.10 Cache/IO

13.2.10.1 STATIO_USER_TABLES

STATIO_USER_TABLES视图显示命名空间中所有用户关系表的I/O状态信息。

表 13-110 STATIO_USER_TABLES 字段

名称	类型	描述
relid	oid	表OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	该表的TOAST表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的TOAST表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	该表的TOAST表索引读取的磁盘块数（如果存在）。

名称	类型	描述
tidx_blks_hit	bigint	该表的TOAST表索引命中缓冲区数（如果存在）。

13.2.10.2 SUMMARY_STATIO_USER_TABLES

SUMMARY_STATIO_USER_TABLES视图显示集群内各节点的用户关系表的I/O状态汇总信息。

表 13-111 SUMMARY_STATIO_USER_TABLES 字段

名称	类型	描述
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	numeric	从该表中读取的磁盘块数。
heap_blks_hit	numeric	该表缓存命中数。
idx_blks_read	numeric	从表中所有索引读取的磁盘块数。
idx_blks_hit	numeric	表中所有索引命中缓存数。
toast_blks_read	numeric	该表的TOAST表读取的磁盘块数（如果存在）。
toast_blks_hit	numeric	该表的TOAST表命中缓冲区数（如果存在）。
tidx_blks_read	numeric	该表的TOAST表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	numeric	该表的TOAST表索引命中缓冲区数（如果存在）。

13.2.10.3 GLOBAL_STATIO_USER_TABLES

GLOBAL_STATIO_USER_TABLES视图显示各节点的命名空间中所有用户关系表的I/O状态信息。

表 13-112 GLOBAL_STATIO_USER_TABLES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	表OID。
schemaname	name	该表模式名。
relname	name	表名。

名称	类型	描述
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	该表的TOAST表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的TOAST表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	该表的TOAST表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该表的TOAST表索引命中缓冲区数（如果存在）。

13.2.10.4 STATIO_USER_INDEXES

STATIO_USER_INDEXES视图显示当前节点命名空间中所有用户关系表索引的I/O状态信息。

表 13-113 STATIO_USER_INDEXES 字段

名称	类型	描述
relid	oid	索引的表的OID。
indexrelid	oid	该索引的OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	bigint	从索引中读取的磁盘块数。
idx_blks_hit	bigint	索引命中缓存数。

13.2.10.5 SUMMARY_STATIO_USER_INDEXES

SUMMARY_STATIO_USER_INDEXES视图显示集群命名空间中所有用户关系表索引的I/O状态汇总信息。

表 13-114 SUMMARY_STATIO_USER_INDEXES 字段

名称	类型	描述
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	numeric	从索引中读取的磁盘块数。
idx_blks_hit	numeric	索引命中缓存数。

13.2.10.6 GLOBAL_STATIO_USER_INDEXES

GLOBAL_STATIO_USER_INDEXES视图显示各节点的命名空间中所有用户关系表索引的I/O状态信息。

表 13-115 GLOBAL_STATIO_USER_INDEXES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	索引的表的OID。
indexrelid	oid	该索引的OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	numeric	从索引中读取的磁盘块数。
idx_blks_hit	numeric	索引命中缓存数。

13.2.10.7 STATIO_USER_SEQUENCES

STATIO_USER_SEQUENCE视图显示当前节点的命名空间中所有用户关系表类型为序列的I/O状态信息。

表 13-116 STATIO_USER_SEQUENCE 字段

名称	类型	描述
relid	oid	序列OID。

名称	类型	描述
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

13.2.10.8 SUMMARY_STATIO_USER_SEQUENCES

SUMMARY_STATIO_USER_SEQUENCES视图显示集群内汇聚的命名空间中所有用户关系表类型为序列的I/O状态信息。

表 13-117 SUMMARY_STATIO_USER_SEQUENCES 字段

名称	类型	描述
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	numeric	从序列中读取的磁盘块数。
blks_hit	numeric	序列中缓存命中数。

13.2.10.9 GLOBAL_STATIO_USER_SEQUENCES

GLOBAL_STATIO_USER_SEQUENCES视图显示各节点的命名空间中所有用户关系表类型为序列的I/O状态信息。

表 13-118 GLOBAL_STATIO_USER_SEQUENCES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	序列OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

13.2.10.10 STATIO_SYS_TABLES

STATIO_SYS_TABLES视图显示当前节点的命名空间中所有系统表的I/O状态信息。

表 13-119 STATIO_SYS_TABLES 字段

名称	类型	描述
relid	oid	表OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	该表的TOAST表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的TOAST表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	该表的TOAST表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该表的TOAST表索引命中缓冲区数（如果存在）。

13.2.10.11 SUMMARY_STATIO_SYS_TABLES

SUMMARY_STATIO_SYS_TABLES视图显示集群内各节点的系统表的I/O状态汇总信息。

表 13-120 SUMMARY_STATIO_SYS_TABLES 字段

名称	类型	描述
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	numeric	从该表中读取的磁盘块数。
heap_blks_hit	numeric	该表缓存命中数。

名称	类型	描述
idx_blks_read	numeric	从表中所有索引读取的磁盘块数。
idx_blks_hit	numeric	表中所有索引命中缓存数。
toast_blks_read	numeric	该表的TOAST表读取的磁盘块数（如果存在）。
toast_blks_hit	numeric	该表的TOAST表命中缓冲区数（如果存在）。
tidx_blks_read	numeric	该表的TOAST表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	numeric	该表的TOAST表索引命中缓冲区数（如果存在）。

13.2.10.12 GLOBAL_STATIO_SYS_TABLES

GLOBAL_STATIO_SYS_TABLES视图显示各节点的命名空间中所有系统表的I/O状态信息。

表 13-121 GLOBAL_STATIO_SYS_TABLES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	表OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	该表的TOAST表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的TOAST表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	该表的TOAST表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该表的TOAST表索引命中缓冲区数（如果存在）。

13.2.10.13 STATIO_SYS_INDEXES

STATIO_SYS_INDEXES显示命名空间中所有系统表索引的I/O状态信息。

表 13-122 STATIO_SYS_INDEXES 字段

名称	类型	描述
relid	oid	索引的表的OID。
indexrelid	oid	该索引的OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	bigint	从索引中读取的磁盘块数。
idx_blks_hit	bigint	索引命中缓存数。

13.2.10.14 SUMMARY_STATIO_SYS_INDEXES

SUMMARY_STATIO_SYS_INDEXES视图显示集群命名空间中所有系统表索引的I/O状态汇总信息。

表 13-123 SUMMARY_STATIO_SYS_INDEXES 字段

名称	类型	描述
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	numeric	从索引中读取的磁盘块数。
idx_blks_hit	numeric	索引命中缓存数。

13.2.10.15 GLOBAL_STATIO_SYS_INDEXES

GLOBAL_STATIO_SYS_INDEXES视图显示各节点的命名空间中所有系统表索引的I/O状态信息。

表 13-124 GLOBAL_STATIO_SYS_INDEXES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	索引的表的OID。
indexrelid	oid	该索引的OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	numeric	从索引中读取的磁盘块数。
idx_blks_hit	numeric	索引命中缓存数。

13.2.10.16 STATIO_SYS_SEQUENCES

STATIO_SYS_SEQUENCES显示命名空间中所有系统序列的I/O状态信息。

表 13-125 STATIO_SYS_SEQUENCES 字段

名称	类型	描述
relid	oid	序列OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

13.2.10.17 SUMMARY_STATIO_SYS_SEQUENCES

SUMMARY_STATIO_SYS_SEQUENCES视图显示集群内汇聚的命名空间中所有系统序列的I/O状态信息。

表 13-126 SUMMARY_STATIO_SYS_SEQUENCES 字段

名称	类型	描述
schemaname	name	序列中模式名。

名称	类型	描述
relname	name	序列名。
blks_read	numeric	从序列中读取的磁盘块数。
blks_hit	numeric	序列中缓存命中数。

13.2.10.18 GLOBAL_STATIO_SYS_SEQUENCES

GLOBAL_STATIO_SYS_SEQUENCES视图显示各节点的命名空间中所有系统序列的I/O状态信息。

表 13-127 GLOBAL_STATIO_SYS_SEQUENCES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	序列OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

13.2.10.19 STATIO_ALL_TABLES

STATIO_ALL_TABLES视图显示数据库当前节点每张表（包括TOAST表）的I/O状态信息。

表 13-128 STATIO_ALL_TABLES 字段

名称	类型	描述
relid	oid	表OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表缓存命中数。

名称	类型	描述
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	该表的TOAST表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的TOAST表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	该表的TOAST表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该表的TOAST表索引命中缓冲区数（如果存在）。

13.2.10.20 SUMMARY_STATIO_ALL_TABLES

SUMMARY_STATIO_ALL_TABLES视图显示集群内各节点每张表（包括TOAST表）的I/O状态信息的汇总结果。

表 13-129 SUMMARY_STATIO_ALL_TABLES 字段

名称	类型	描述
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	numeric	从该表中读取的磁盘块数。
heap_blks_hit	numeric	该表缓存命中数。
idx_blks_read	numeric	从表中所有索引读取的磁盘块数。
idx_blks_hit	numeric	表中所有索引命中缓存数。
toast_blks_read	numeric	该表的TOAST表读取的磁盘块数（如果存在）。
toast_blks_hit	numeric	该表的TOAST表命中缓冲区数（如果存在）。
tidx_blks_read	numeric	该表的TOAST表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	numeric	该表的TOAST表索引命中缓冲区数（如果存在）。

13.2.10.21 GLOBAL_STATIO_ALL_TABLES

GLOBAL_STATIO_ALL_TABLES视图将包含各节点的数据库中每个表（包括TOAST表）的I/O的统计。

表 13-130 GLOBAL_STATIO_ALL_TABLES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	表OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	该表的TOAST表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的TOAST表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	该表的TOAST表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该表的TOAST表索引命中缓冲区数（如果存在）。

13.2.10.22 STATIO_ALL_INDEXES

STATIO_ALL_INDEXES视图包含数据库中的每个索引行，显示特定索引的I/O的统计。

表 13-131 STATIO_ALL_INDEXES 字段

名称	类型	描述
relid	oid	索引的表的OID。
indexrelid	oid	该索引的OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。

名称	类型	描述
idx_blks_read	bigint	从索引中读取的磁盘块数。
idx_blks_hit	bigint	索引命中缓存数。

13.2.10.23 SUMMARY_STATIO_ALL_INDEXES

SUMMARY_STATIO_ALL_INDEXES视图包含集群中的每个索引行， 显示特定索引I/O的统计汇总结果。

表 13-132 SUMMARY_STATIO_ALL_INDEXES 字段

名称	类型	描述
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	numeric	从索引中读取的磁盘块数。
idx_blks_hit	numeric	索引命中缓存数。

13.2.10.24 GLOBAL_STATIO_ALL_INDEXES

GLOBAL_STATIO_ALL_INDEXES视图包含各节点的数据库中的每个索引行，显示特定索引的I/O的统计。

表 13-133 GLOBAL_STATIO_ALL_INDEXES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	索引的表的OID。
indexrelid	oid	该索引的OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	numeric	从索引中读取的磁盘块数。

名称	类型	描述
idx_blks_hit	numeric	索引命中缓存数。

13.2.10.25 STATIO_ALL_SEQUENCES

STATIO_ALL_SEQUENCES视图包含数据库中每个序列的每一行，显示特定序列关于I/O的统计。

表 13-134 STATIO_ALL_SEQUENCES 字段

名称	类型	描述
relid	oid	序列OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

13.2.10.26 SUMMARY_STATIO_ALL_SEQUENCES

SUMMARY_STATIO_ALL_SEQUENCES视图包含集群内汇聚的数据库中每个序列的每一行,显示特定序列关于I/O的统计。

表 13-135 SUMMARY_STATIO_ALL_SEQUENCES 字段

名称	类型	描述
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	numeric	从序列中读取的磁盘块数。
blks_hit	numeric	序列中缓存命中数。

13.2.10.27 GLOBAL_STATIO_ALL_SEQUENCES

GLOBAL_STATIO_ALL_SEQUENCES包含各节点的数据库中每个序列的每一行，显示特定序列关于I/O的统计。

表 13-136 GLOBAL_STATIO_ALL_SEQUENCES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	序列OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

13.2.10.28 GLOBAL_STAT_DB_CU

GLOBAL_STAT_DB_CU视图用于查询集群各个节点中每个数据库的CU命中情况。可以通过gs_stat_reset()进行清零。

表 13-137 GLOBAL_STAT_DB_CU 字段

名称	类型	描述
node_name1	text	节点名称。
db_name	text	数据库名。
mem_hit	bigint	内存命中次数。
hdd_sync_read	bigint	硬盘同步读次数。
hdd_asyn_read	bigint	硬盘异步读次数。

13.2.10.29 GLOBAL_STAT_SESSION_CU

GLOBAL_STAT_SESSION_CU用于查询整个集群各个节点，当前运行session的CU命中情况。session退出相应的统计数据会清零。集群重启后，统计数据也会清零。

表 13-138 GLOBAL_STAT_SESSION_CU 字段

名称	类型	描述
node_name1	text	节点名称。
mem_hit	integer	内存命中次数。
hdd_sync_read	integer	硬盘同步读次数。

名称	类型	描述
hdd_asyn_read	integer	硬盘异步读次数。

13.2.11 Comm

13.2.11.1 COMM_DELAY

COMM_DELAY视图展示单个DN的TCP代理通信库时延状态。

表 13-139 COMM_DELAY 字段

名称	类型	描述
node_name	text	节点名称。
remote_name	text	连接对端节点名称。
remote_host	text	连接对端IP地址。
stream_num	integer	当前物理连接使用的stream逻辑连接数量。
min_delay	integer	当前物理连接一分钟内探测到的最小时延（单位：微秒）。 **说明** 负数结果无效，请重新等待时延状态更新后再执行。
average	integer	当前物理连接一分钟内探测时延的平均值（单位：微秒）。
max_delay	integer	当前物理连接一分钟内探测到的最大时延（单位：微秒）。

13.2.11.2 GLOBAL_COMM_DELAY

GLOBAL_COMM_DELAY视图展示所有DN的TCP代理通信库时延状态。

表 13-140 GLOBAL_COMM_DELAY 字段

名称	类型	描述
node_name	text	节点名称。
remote_name	text	连接对端节点名称。
remote_host	text	连接对端IP地址。

名称	类型	描述
stream_num	integer	当前物理连接使用的stream逻辑连接数量。
min_delay	integer	当前物理连接一分钟内探测到的最小时延（单位：微秒）。 **说明** 负数结果无效，请重新等待时延状态更新后再执行。
average	integer	当前物理连接一分钟内探测时延的平均值（单位：微秒）。
max_delay	integer	当前物理连接一分钟内探测到的最大时延（单位：微秒）。

13.2.11.3 COMM_RECV_STREAM

COMM_RECV_STREAM视图展示单个DN上所有的TCP代理通信库接收流状态。

表 13-141 COMM_RECV_STREAM 字段

名称	类型	描述
node_name	text	节点名称。
local_tid	bigint	使用此通信流的线程ID。
remote_name	text	连接对端节点名称。
remote_tid	bigint	连接对端线程ID。
idx	integer	通信对端DN在本DN内的标识编号。
sid	integer	通信流在物理连接中的标识编号。
tcp_sock	integer	通信流所使用的tcp通信socket。
state	text	通信流当前的状态。
query_id	bigint	通信流对应的debug_query_id编号。
pn_id	integer	通信流所执行查询的plan_node_id编号。
send_smp	integer	通信流所执行查询send端的smpid编号。
recv_smp	integer	通信流所执行查询recv端的smpid编号。
recv_bytes	bigint	通信流接收的数据总量（单位：Byte）。
time	bigint	通信流当前生命周期使用时长（单位：毫秒）。
speed	bigint	通信流的平均接收速率（单位：Byte/s）。
quota	bigint	通信流当前的通信配额值（单位：Byte）。

名称	类型	描述
buff_usize	bigint	通信流当前缓存的数据大小（单位：Byte）。

13.2.11.4 GLOBAL_COMM_RECV_STREAM

GLOBAL_COMM_RECV_STREAM视图展示所有DN上所有的TCP代理通信库接收流状态。

表 13-142 GLOBAL_COMM_RECV_STREAM 字段

名称	类型	描述
node_name	text	节点名称。
local_tid	bigint	使用此通信流的线程ID。
remote_name	text	连接对端节点名称。
remote_tid	bigint	连接对端线程ID。
idx	integer	通信对端DN在本DN内的标识编号。
sid	integer	通信流在物理连接中的标识编号。
tcp_sock	integer	通信流所使用的tcp通信socket。
state	text	通信流当前的状态。
query_id	bigint	通信流对应的debug_query_id编号。
pn_id	integer	通信流所执行查询的plan_node_id编号。
send_smp	integer	通信流所执行查询send端的smpid编号。
recv_smp	integer	通信流所执行查询recv端的smpid编号。
recv_bytes	bigint	通信流接收的数据总量（单位：Byte）。
time	bigint	通信流当前生命周期使用时长（单位：毫秒）。
speed	bigint	通信流的平均接收速率（单位：Byte/s）。
quota	bigint	通信流当前的通信配额值（单位：Byte）。
buff_usize	bigint	通信流当前缓存的数据大小（单位：Byte）。

13.2.11.5 COMM_SEND_STREAM

COMM_SEND_STREAM展示单个DN上所有的TCP代理通信库发送流状态。

表 13-143 COMM_SEND_STREAM 字段

名称	类型	描述
node_name	text	节点名称。
local_tid	bigint	使用此通信流的线程ID。
remote_name	text	连接对端节点名称。
remote_tid	bigint	连接对端线程ID。
idx	integer	通信对端DN在本DN内的标识编号。
sid	integer	通信流在物理连接中的标识编号。
tcp_sock	integer	通信流所使用的tcp通信socket。
state	text	通信流当前的状态。
query_id	bigint	通信流对应的debug_query_id编号。
pn_id	integer	通信流所执行查询的plan_node_id编号。
send_smp	integer	通信流所执行查询send端的smpid编号。
recv_smp	integer	通信流所执行查询recv端的smpid编号。
send_bytes	bigint	通信流发送的数据总量（单位：Byte）。
time	bigint	通信流当前生命周期使用时长（单位：ms）。
speed	bigint	通信流的平均发送速率（单位：Byte/s）。
quota	bigint	通信流当前的通信配额值（单位：Byte）。
wait_quota	bigint	通信流等待quota值产生的额外时间开销（单位：毫秒）。

13.2.11.6 GLOBAL_COMM_SEND_STREAM

GLOBAL_COMM_SEND_STREAM视图展示所有DN上所有的TCP代理通信库发送流状态。

表 13-144 GLOBAL_COMM_SEND_STREAM 字段

名称	类型	描述
node_name	text	节点名称。
local_tid	bigint	使用此通信流的线程ID。
remote_name	text	连接对端节点名称。
remote_tid	bigint	连接对端线程ID。

名称	类型	描述
idx	integer	通信对端DN在本DN内的标识编号。
sid	integer	通信流在物理连接中的标识编号。
tcp_sock	integer	通信流所使用的tcp通信socket。
state	text	通信流当前的状态。
query_id	bigint	通信流对应的debug_query_id编号。
pn_id	integer	通信流所执行查询的plan_node_id编号。
send_smp	integer	通信流所执行查询send端的smpid编号。
recv_smp	integer	通信流所执行查询recv端的smpid编号。
send_bytes	bigint	通信流发送的数据总量（单位：Byte）。
time	bigint	通信流当前生命周期使用时长（单位：毫秒）。
speed	bigint	通信流的平均发送速率（单位：Byte/s）。
quota	bigint	通信流当前的通信配额值（单位：Byte）。
wait_quota	bigint	通信流等待quota值产生的额外时间开销（单位：毫秒）。

13.2.11.7 COMM_STATUS

COMM_STATUS视图展示单个DN的TCP代理通信库状态。

表 13-145 COMM_STATUS 字段

名称	类型	描述
node_name	text	节点名称。
rxpck_rate	integer	节点通信库接收速率，单位Byte/s。
txpck_rate	integer	节点通信库发送速率，单位Byte/s。
rxkbyte_rate	bigint	bigint节点通信库接收速率，单位KByte/s。
txkbyte_rate	bigint	bigint节点通信库发送速率，单位KByte/s。
buffer	bigint	cmailbox的buffer大小。
memkbyte_libcomm	bigint	libcomm进程通信内存大小，单位Byte。
memkbyte_libpq	bigint	libpq进程通信内存大小，单位Byte。
used_pm	integer	postmaster线程实时使用率。

名称	类型	描述
used_sflow	integer	gs_sender_flow_controller线程实时使用率。
used_rflow	integer	gs_receiver_flow_controller线程实时使用率。
used_rloop	integer	多个gs_receivers_loop线程中高的实时使用率。
stream	integer	当前使用的逻辑连接总数。

13.2.11.8 GLOBAL_COMM_STATUS

GLOBAL_COMM_STATUS视图展示所有DN的TCP代理通信库状态。

表 13-146 GLOBAL_COMM_STATUS 字段

名称	类型	描述
node_name	text	节点名称。
rxpck_rate	integer	节点通信库接收速率，单位Byte/s。
txpck_rate	integer	节点通信库发送速率，单位Byte/s。
rxkbyte_rate	bigint	bigint节点通信库接收速率，单位KByte/s。
txkbyte_rate	bigint	bigint节点通信库发送速率，单位KByte/s。
buffer	bigint	cmailbox的buffer大小。
memkbyte_libcomm	bigint	libcomm进程通信内存大小，单位Byte。
memkbyte_libpq	bigint	libpq进程通信内存大小，单位Byte。
used_pm	integer	postmaster线程实时使用率。
used_sflow	integer	gs_sender_flow_controller线程实时使用率。
used_rflow	integer	gs_receiver_flow_controller线程实时使用率。
used_rloop	integer	多个gs_receivers_loop线程中高的实时使用率。
stream	integer	当前使用的逻辑连接总数。

13.2.12 Utility

13.2.12.1 REPLICATION_STAT

REPLICATION_STAT用于描述日志同步状态信息，如发起端发送日志位置、收端接收日志位置等。

表 13-147 REPLICATION_STAT 字段

名称	类型	描述
pid	bigint	线程的PID。
usesysid	oid	用户系统ID。
usename	name	用户名。
application_name	text	程序名称。
client_addr	inet	客户端地址。
client_hostname	text	客户端名。
client_port	integer	客户端端口。
backend_start	timestamp with time zone	程序启动时间。
state	text	日志复制的状态： • 追赶状态。 • 一致的流状态。
sender_sent_location	text	发送端发送日志位置。
receiver_write_location	text	接收端write日志位置。
receiver_flush_location	text	接收端flush日志位置。
receiver_replay_location	text	接收端replay日志位置。
sync_priority	integer	同步复制的优先级（0表示异步）。
sync_state	text	同步状态： • 异步复制。 • 同步复制。 • 潜在同步者。

13.2.12.2 GLOBAL_REPLICATION_STAT

GLOBAL_REPLICATION_STAT视图用于获得各节点描述日志同步状态信息，如发起端发送日志位置、收端接收日志位置等。

表 13-148 GLOBAL_REPLICATION_STAT 字段

名称	类型	描述
node_name	name	节点名称。
pid	bigint	线程的PID。

名称	类型	描述
usesysid	oid	用户系统ID。
usename	name	用户名。
application_name	text	程序名称。
client_addr	inet	客户端地址。
client_hostname	text	客户端名。
client_port	integer	客户端端口。
backend_start	timestamp with time zone	程序启动时间。
state	text	日志复制的状态： ● 追赶状态。 ● 一致的流状态。
sender_sent_location	text	发送端发送日志位置。
receiver_write_location	text	接收端write日志位置。
receiver_flush_location	text	接收端flush日志位置。
receiver_replay_location	text	接收端replay日志位置。
sync_priority	integer	同步复制的优先级（0表示异步）。
sync_state	text	同步状态： ● 异步复制。 ● 同步复制。 ● 潜在同步者。

13.2.12.3 REPLICATION_SLOTS

REPLICATION_SLOTS视图用于查看复制槽的信息。

表 13-149 REPLICATION_SLOTS 字段

名称	类型	描述
slot_name	text	复制槽的名称。
plugin	text	逻辑复制槽对应的输出插件名称。

名称	类型	描述
slot_type	text	复制槽的类型。 ● physical：物理复制槽。 ● logical：逻辑复制槽。
datoid	oid	复制槽所在的数据库OID。
database	name	复制槽所在的数据库名称。
active	boolean	复制槽是否为激活状态。 ● t（true）：表示是。 ● f（false）：表示不是。
xmin	xid	复制槽需要数据库保留的最旧事务。VACUUM不能移除被其后续事务删除的元组。
catalog_xmin	xid	复制槽需要数据库保留的影响系统表的最旧事务。VACUUM不能移除被其后续事务删除的系统表元组。
restart_lsn	text	复制槽需要的最早xLog的物理位置。
dummy_standby	boolean	实验室特性。

13.2.12.4 GLOBAL_REPLICATION_SLOTS

GLOBAL_REPLICATION_SLOTS视图用于查看集群各节点的复制槽的信息。

表 13-150 GLOBAL_REPLICATION_SLOTS 字段

名称	类型	描述
node_name	name	节点名称。
slot_name	text	复制槽的名称。
plugin	text	逻辑复制槽对应的输出插件名称。
slot_type	text	复制槽的类型。 ● physical：物理复制槽。 ● logical：逻辑复制槽。
datoid	oid	复制槽所在的数据库OID。
database	name	复制槽所在的数据库名称。
active	boolean	复制槽是否为激活状态。 ● t（true）：表示是。 ● f（false）：表示不是。

名称	类型	描述
x_min	xid	复制槽需要数据库保留的最旧事务。VACUUM不能移除被其后续事务删除的元组。
catalog_xmin	xid	复制槽需要数据库保留的影响系统表的最旧事务。VACUUM不能移除被其后续事务删除的系统表元组。
restart_lsn	text	复制槽需要的最早xLog的物理位置。
dummy_standby	boolean	实验室特性。

13.2.12.5 BGWRITER_STAT

BGWRITER_STAT视图显示关于后端写进程活动的统计信息。

表 13-151 BGWRITER_STAT 字段

名称	类型	描述
checkpoints_timed	bigint	执行的定期检查点数。
checkpoints_req	bigint	执行的需求检查点数。
checkpoint_write_time	double precision	在检查点处理过程中执行写入操作所花费的总时间，单位为毫秒。
checkpoint_sync_time	double precision	在检查点处理过程中执行同步（sync）操作所花费的总时间，单位为毫秒。
buffers_checkpoint	bigint	检查点写缓冲区的数量。
buffers_clean	bigint	BGWRITER线程写缓冲区的数量。
maxwritten_clean	bigint	BGWRITER线程停止清理扫描过程的次数（由于单次扫描刷页数量过多导致）。
buffers_backend	bigint	后端直接写缓冲区的数量。
buffers_backend_fsync	bigint	后端线程自行执行fsync的次数（通常情况下fsync调用由BGWRITER线程执行）。
buffers_alloc	bigint	分配的缓冲区数量。
stats_reset	timestamp with time zone	本行统计信息上次被重置的时间。

13.2.12.6 GLOBAL_BGWRITER_STAT

GLOBAL_BGWRITER_STAT视图显示所有实例关于后端写线程活动的统计信息。

表 13-152 GLOBAL_BGWRITER_STAT 字段

名称	类型	描述
node_name	name	实例名称。
checkpoints_timed	bigint	执行的定期检查点数。
checkpoints_req	bigint	执行的需求检查点数。
checkpoint_write_time	double precision	在检查点处理过程中执行写入操作所花费的总时间，单位为毫秒。
checkpoint_sync_time	double precision	在检查点处理过程中执行同步（sync）操作所花费的总时间，单位为毫秒。
buffers_checkpoint	bigint	检查点写缓冲区的数量。
buffers_clean	bigint	BGWRITER线程写缓冲区的数量。
maxwritten_clean	bigint	BGWRITER线程停止清理扫描过程的次数（由于单次扫描刷页数量过多导致）。
buffers_backend	bigint	后端直接写缓冲区的数量。
buffers_backend_fsync	bigint	后端线程自行执行fsync的次数（通常情况下fsync调用由BGWRITER线程执行）。
buffers_alloc	bigint	分配的缓冲区数量。
stats_reset	timestamp with time zone	本行统计信息上次被重置的时间。

13.2.12.7 POOLER_STATUS

POOLER_STATUS视图用于查询本地CN 的pooler中的缓存连接状态。

表 13-153 POOLER_STATUS 字段

名称	类型	描述
database	text	数据库名称。
user_name	text	用户名。
tid	bigint	非线程池逻辑下为连接CN的线程id，线程池逻辑下为连接CN的sessionid。

名称	类型	描述
node_oid	bigint	连接的实例节点OID。
node_name	name	连接的实例节点名称。
in_use	boolean	连接是否正被使用： ● t（true）：表示连接正在使用。 ● f（false）：表示连接没有使用。
node_port	integer	连接的节点端口。
fdsock	bigint	端口文件描述符。
remote_pid	bigint	连接的远端节点线程号。
session_params	text	会话参数。
used_count	bigint	该连接的复用次数。
idx	bigint	连接的实例节点逻辑连接id。
streamid	bigint	每个逻辑连接对应的流标识id。

13.2.12.8 GLOBAL_COMM_CHECK_CONNECTION_STATUS

GLOBAL_COMM_CHECK_CONNECTION_STATUS视图用于显示所有CN和所有活跃节点（CN和主DN）的连接情况，权限控制继承DBE_PERF schema。

表 13-154 GLOBAL_COMM_CHECK_CONNECTION_STATUS 字段

名称	类型	描述
node_name	name	实例名称。
remote_name	name	对端实例名称。
remote_host	text	对端实例的IP。
remote_port	bigint	对端实例的PORT。
is_connected	boolean	检测当前实例与对端实例之间的网络情况： ● t（true）：表示实例之间网络正常。 ● f（false）：表示实例之间网络异常。

名称	类型	描述
no_error_occur	boolean	当前实例与对端实例的pooler建连结果： ● t（true）：表示pooler连接正常。 ● f（false）：表示pooler连接异常。

13.2.12.9 GLOBAL_CKPT_STATUS

GLOBAL_CKPT_STATUS视图用于显示整个集群所有实例的检查点信息和各类日志刷页情况。

表 13-155 GLOBAL_CKPT_STATUS 字段

名称	类型	描述
node_name	text	实例名称。
ckpt_redo_point	text	当前实例的检查点。
ckpt_clog_flush_num	bigint	从启动到当前时间clog刷盘页面数。
ckpt_csnlog_flush_num	bigint	从启动到当前时间csnlog刷盘页面数。
ckpt_multixact_flush_num	bigint	从启动到当前时间multixact刷盘页面数。
ckpt_predicate_flush_num	bigint	从启动到当前时间predicate刷盘页面数。
ckpt_twophase_flush_num	bigint	从启动到当前时间twophase刷盘页面数。

13.2.12.10 GLOBAL_DOUBLE_WRITE_STATUS

GLOBAL_DOUBLE_WRITE_STATUS视图显示整个集群所有实例的双写文件的情况。

表 13-156 GLOBAL_DOUBLE_WRITE_STATUS 字段

名称	类型	描述
node_name	text	实例名称。
curr_dwn	bigint	当前双写文件的序列号。
curr_start_page	bigint	当前双写文件恢复起始页面。
file_trunc_num	bigint	当前双写文件复用的次数。

名称	类型	描述
file_reset_num	bigint	当前双写文件写满后发生重置的次数。
total_writes	bigint	当前双写文件总的I/O次数。
low_threshold_writes	bigint	低效率写双写文件的I/O次数。
high_threshold_writes	bigint	高效率写双写文件的I/O次数。
total_pages	bigint	当前刷页到双写文件区的总的页面个数。
low_threshold_pages	bigint	低效率刷页的页面个数。
high_threshold_pages	bigint	高效率刷页的页面个数。
file_id	bigint	当前双写文件的id号。

13.2.12.11 GLOBAL_PAGEWRITER_STATUS

GLOBAL_PAGEWRITER_STATUS视图显示整个集群所有实例的刷页信息和检查点信息。

表 13-157 GLOBAL_PAGEWRITER_STATUS 字段

名称	类型	描述
node_name	text	实例名称。
pgwr_actual_flush_total_num	bigint	从启动到当前时间总计刷脏页数量。
pgwr_last_flush_num	integer	上一批刷脏页数量。
remain_dirty_page_num	bigint	当前预计还剩余多少脏页。
queue_head_page_rec_lsn	text	当前实例的脏页队列第一个脏页的recovery_lsn。
queue_rec_lsn	text	当前实例的脏页队列的recovery_lsn。
current_xlog_insert_lsn	text	当前实例xLog写入的位置。
ckpt_redo_point	text	当前实例的检查点。

13.2.12.12 GLOBAL_POOLER_STATUS

GLOBAL_POOLER_STATUS视图用于查询全局CN 的pooler中的缓存连接状态。

表 13-158 GLOBAL_POOLER_STATUS 字段

名称	类型	描述
source_node_name	name	源节点名称。
database	text	数据库名称。
user_name	text	用户名。
tid	bigint	非线程池逻辑下为连接CN的线程id，线程池逻辑下为连接CN的sessionid。
node_oid	bigint	连接的实例节点OID。
node_name	name	连接的实例节点名称。
in_use	boolean	连接是否正被使用： ● t（true）：表示连接正在使用。 ● f（false）：表示连接没有使用。
fdsock	bigint	端口文件描述符。
remote_pid	bigint	连接的远端节点线程号。
session_params	text	会话参数。

13.2.12.13 GLOBAL_RECORD_RESET_TIME

GLOBAL_RECORD_RESET_TIME用于获取集群中各节点的“重置（重启，主备倒换，数据库删除）时间”的统计信息。

表 13-159 GLOBAL_RECORD_RESET_TIME 字段

名称	类型	描述
node_name	text	节点名称。
reset_time	timestamp with time zone	重置时间点。

13.2.12.14 GLOBAL_REDO_STATUS

GLOBAL_REDO_STATUS视图显示整个集群所有实例的日志回放情况。

表 13-160 GLOBAL_REDO_STATUS 字段

名称	类型	描述
node_name	text	实例名称。
redo_start_ptr	bigint	当前实例日志回放的起始点。
redo_start_time	bigint	当前实例日志回放的起始UTC时间。
redo_done_time	bigint	当前实例日志回放的结束UTC时间。
curr_time	bigint	当前实例的当前UTC时间。
min_recovery_point	bigint	当前实例日志完成回放后可对外提供服务的最小一致性点。
read_ptr	bigint	当前实例日志的读取位置。
last_replayed_read_ptr	bigint	当前实例的日志回放位置。
recovery_done_ptr	bigint	当前实例启动完成时的回放位置。
read_xlog_io_counter	bigint	当前实例读取回放日志的I/O次数计数。
read_xlog_io_total_dur	bigint	当前实例读取回放日志的I/O总用时。
read_data_io_counter	bigint	当前实例日志回放过程中，读取数据页面的I/O次数计数。
read_data_io_total_dur	bigint	当前实例日志回放过程中，读取数据页面的I/O总用时。
write_data_io_counter	bigint	当前实例日志回放过程中，写数据页面的I/O次数计数。
write_data_io_total_dur	bigint	当前实例日志回放过程中，写数据页面的I/O总用时。
process_pending_counter	bigint	当前实例日志回放过程中，日志分发线程的同步次数计数。
process_pending_total_dur	bigint	当前实例日志回放过程中，日志分发线程的同步总用时。
apply_counter	bigint	当前实例日志回放过程中，回放线程的同步次数计数。
apply_total_dur	bigint	当前实例日志回放过程中，回放线程的同步总用时。
speed	bigint	当前实例日志回放速率，每回放256MB日志该值更新一次，单位byte/s。 在集群环境下，建议使用cm_ctl query -rv命令来获取更精确的备机回放速度。

名称	类型	描述
local_max_ptr	bigint	当前实例启动成功后本地收到的回放日志的最大值。
primary_flush_ptr	bigint	主机落盘日志的位置。
worker_info	text	当前实例回放线程信息，若没有开并行回放则该值为空。

13.2.12.15 GLOBAL_RECOVERY_STATUS

GLOBAL_RECOVERY_STATUS视图显示关于主机和备机的日志流控信息。

表 13-161 GLOBAL_RECOVERY_STATUS 字段

名称	类型	描述
node_name	text	节点的名称，包含主机和备机。
standby_node_name	text	备机节点的名称。
source_ip	text	主机的IP地址。
source_port	integer	主机的端口号。
dest_ip	text	备机的IP地址。
dest_port	integer	备机的端口号。
current_rto	bigint	备机当前的日志流控时间，单位秒。
target_rto	bigint	备机通过GUC参数设置的预期流控时间，单位秒。
current_sleep_time	bigint	为了达到此RTO预期，主机所需要执行的睡眠时间，单位微秒。

13.2.12.16 CLASS_VITAL_INFO

CLASS_VITAL_INFO视图用于做WDR时校验相同的表或者索引的Oid是否一致。

表 13-162 CLASS_VITAL_INFO 字段

名称	类型	描述
relid	oid	表的oid。
schemaname	name	schema名称。
relname	name	表名。

名称	类型	描述
relkind	"char"	表示对象类型，取值范围如下： ● r：表示普通表。 ● t：表示toast表。 ● i：表示索引。

13.2.12.17 USER_LOGIN

USER_LOGIN用来记录用户登录和退出次数的相关信息。

表 13-163 USER_LOGIN 字段

名称	类型	描述
node_name	text	节点名称。
user_name	text	用户名称。
user_id	integer	用户oid(同pg_authid中的oid字段)。
login_counter	bigint	登录次数。
logout_counter	bigint	退出次数。

13.2.12.18 SUMMARY_USER_LOGIN

SUMMARY_USER_LOGIN用来记录所有CN节点上用户登录和退出次数的相关信息。

表 13-164 SUMMARY_USER_LOGIN 字段

名称	类型	描述
node_name	text	节点名称。
user_name	text	用户名称。
user_id	integer	用户oid(同pg_authid中的oid字段)。
login_counter	bigint	登录次数。
logout_counter	bigint	退出次数。

13.2.12.19 GLOBAL_GET_BGWRITER_STATUS

GLOBAL_GET_BGWRITER_STATUS视图显示整个集群所有实例bgwriter线程刷页信息、候选buffer链中页面个数和buffer淘汰信息。

表 13-165 GLOBAL_GET_BGWRITER_STATUS 字段

名称	类型	描述
node_name	text	实例名称。
bgwr_actual_flush_total_num	bigint	从启动到当前时间bgwriter线程总计刷脏页数量。
bgwr_last_flush_num	integer	bgwriter线程上一批刷脏页数量。
candidate_slots	integer	当前候选buffer链中页面个数。
get_buffer_from_list	bigint	buffer淘汰从候选buffer链中获取页面的次数。
get_buf_clock_sweep	bigint	buffer淘汰从原淘汰方案中获取页面的次数。

13.2.12.20 GLOBAL_SINGLE_FLUSH_DW_STATUS

GLOBAL_SINGLE_FLUSH_DW_STATUS视图显示整个集群所有实例单页面淘汰双写文件信息。显示内容中，/ 前是第一个版本双写文件刷页情况，/ 后是第二个版本双写文件刷页情况。

表 13-166 GLOBAL_SINGLE_FLUSH_DW_STATUS 字段

名称	类型	描述
node_name	text	实例名称。
curr_dwn	text	当前双写文件的序列号。
curr_start_page	text	当前双写文件start位置。
total_writes	text	当前双写文件总计写数据页面个数。
file_trunc_num	text	当前双写文件复用的次数。
file_reset_num	text	当前双写文件写满后发生重置的次数。

13.2.12.21 GLOBAL_CANDIDATE_STATUS

GLOBAL_CANDIDATE_STATUS视图显示整个数据库所有实例候选buffer个数、buffer淘汰信息。

表 13-167 GLOBAL_GET_BGWRITER_STATUS 字段

名称	类型	描述
node_name	text	节点名称。

名称	类型	描述
candidate_slots	integer	当前Normal Buffer Pool候选buffer链中页面个数。
get_buf_from_list	bigint	Normal Buffer Pool，buffer淘汰从候选buffer链中获取页面的次数。
get_buf_clock_sweep	bigint	Normal Buffer Pool，buffer淘汰从原淘汰方案中获取页面的次数。
seg_candidate_slots	integer	当前Segment Buffer Pool候选buffer链中页面个数。
seg_get_buf_from_list	bigint	Segment Buffer Pool，buffer淘汰从候选buffer链中获取页面的次数。
seg_get_buf_clock_sweep	bigint	Segment Buffer Pool，buffer淘汰从原淘汰方案中获取页面的次数。

13.2.13 Lock

13.2.13.1 LOCKS

LOCKS视图用于查看各打开事务所持有的锁信息。

表 13-168 LOCKS 字段

名称	类型	描述
locktype	text	被锁定对象的类型：relation，extend，page，tuple，transactionid，virtualxid，object，userlock，advisory。
database	oid	被锁定对象所在数据库的OID： ● 如果被锁定的对象是共享对象，则OID为0。 ● 如果是一个事务ID，则为NULL。
relation	oid	关系的OID，如果锁定的对象不是关系，也不是关系的一部分，则为NULL。
page	integer	关系内部的页面编号，如果对象不是关系页或者不是行页，则为NULL。
tuple	smallint	页面里边的行编号，如果对象不是行，则为NULL。
bucket	integer	哈希桶号。
virtualxid	text	事务的虚拟ID，如果对象不是一个虚拟事务ID，则为NULL。

名称	类型	描述
transactionid	xid	事务的ID，如果对象不是一个事务ID，则为NULL。
classid	oid	包含该对象的系统表的OID，如果对象不是普通的数据库对象，则为NULL。
objid	oid	对象在其系统表内的OID，如果对象不是普通数据库对象，则为NULL。
objsubid	smallint	对于表的一个字段，这是字段编号；对于其他对象类型，这个字段是零；如果这个对象不是普通数据库对象，则为NULL。
virtualtransaction	text	持有此锁或者在等待此锁的事务的虚拟ID。
pid	bigint	持有或者等待这个锁的服务器线程的逻辑ID。如果锁是被一个预备事务持有的，则为NULL。
sessionid	bigint	持有或者等待这个锁的会话ID。如果锁是被一个预备事务持有的，则为NULL。
mode	text	这个线程持有的或者是期望的锁模式。
granted	boolean	• 如果锁是持有锁，则为TRUE。 • 如果锁是等待锁，则为FALSE。
fastpath	boolean	如果通过fast-path获得锁，则为TRUE；如果通过主要的锁表获得，则为FALSE。
locktag	text	会话等待锁信息，可通过locktag_decode()函数解析。
global_sessionid	text	全局会话ID。

13.2.13.2 GLOBAL_LOCKS

GLOBAL_LOCKS视图用于查看各节点各打开事务所持有的锁信息。

表 13-169 GLOBAL_LOCKS 字段

名称	类型	描述
node_name	name	节点名称。
locktype	text	被锁定对象的类型：relation，extend，page，tuple，transactionid，virtualxid，object，userlock，advisory。

名称	类型	描述
database	oid	被锁定对象所在数据库的OID： ● 如果被锁定的对象是共享对象，则OID为0。 ● 如果是一个事务ID，则为NULL。
relation	oid	关系的OID，如果锁定的对象不是关系，也不是关系的一部分，则为NULL。
page	integer	关系内部的页面编号，如果对象不是关系页或者不是行页，则为NULL。
tuple	smallint	页面里边的行编号，如果对象不是行，则为NULL。
bucket	integer	哈希桶号。
virtualxid	text	事务的虚拟ID，如果对象不是一个虚拟事务ID，则为NULL。
transactionid	xid	事务的ID，如果对象不是一个事务ID，则为NULL。
classid	oid	包含该对象的系统表的OID，如果对象不是普通的数据库对象，则为NULL。
objid	oid	对象在其系统表内的OID，如果对象不是普通数据库对象，则为NULL。
objsubid	smallint	对于表的一个字段，这是字段编号；对于其他对象类型，这个字段是零；如果这个对象不是普通数据库对象，则为NULL。
virtualtransaction	text	持有此锁或者在等待此锁的事务的虚拟ID。
pid	bigint	持有或者等待这个锁的服务器线程的逻辑ID。如果锁是被一个预备事务持有的，则为NULL。
sessionid	bigint	持有或者等待这个锁的会话ID。如果锁是被一个预备事务持有的，则为NULL。
global_sessionid	text	全局会话ID。
mode	text	这个线程持有的或者是期望的锁模式。
granted	boolean	● 如果锁是持有锁，则为TRUE。 ● 如果锁是等待锁，则为FALSE。
fastpath	boolean	如果通过fast-path获得锁，则为TRUE；如果通过主要的锁表获得，则为FALSE。
locktag	text	会话等待锁信息，可通过locktag_decode()函数解析。

13.2.14 Wait Events

13.2.14.1 WAIT_EVENTS

WAIT_EVENTS显示当前节点wait event的相关统计信息。或从视图wait_event_info中查看系统中所有的事件列表。关于每种事务锁对业务的影响程度，请参考**LOCK**语法小节的详细描述。

表 13-170 WAIT_EVENTS 字段

名称	类型	描述
nodename	text	节点名称。
type	text	event类型。
event	text	event名称。
wait	bigint	等待次数。
failed_wait	bigint	失败的等待次数。
total_wait_time	bigint	总等待时间（单位：微秒）。
avg_wait_time	bigint	平均等待时间（单位：微秒）。
max_wait_time	bigint	最大等待时间（单位：微秒）。
min_wait_time	bigint	最小等待时间（单位：微秒）。
last_updated	timestamp with time zone	最后一次更新该事件的时间。

13.2.14.2 GLOBAL_WAIT_EVENTS

GLOBAL_WAIT_EVENTS视图显示各节点wait event的相关统计信息。查询视图必须具有sysadmin权限或者monitor admin权限。

表 13-171 GLOBAL_WAIT_EVENTS 字段

名称	类型	描述
nodename	text	节点名称。
type	text	event类型。
event	text	event名称。
wait	bigint	等待次数。
failed_wait	bigint	失败的等待次数。

名称	类型	描述
total_wait_time	bigint	总等待时间（单位：微秒）。
avg_wait_time	bigint	平均等待时间（单位：微秒）。
max_wait_time	bigint	最大等待时间（单位：微秒）。
min_wait_time	bigint	最小等待时间（单位：微秒）。
last_updated	timestamp with time zone	最后一次更新该事件的时间。

13.2.15 Configuration

13.2.15.1 CONFIG_SETTINGS

CONFIG_SETTINGS视图显示数据库运行时参数的相关信息。

表 13-172 CONFIG_SETTINGS 字段

名称	类型	描述
name	text	参数名称。
setting	text	参数当前值。
unit	text	参数的隐式结构。
category	text	参数的逻辑组。
short_desc	text	参数的简单描述。
extra_desc	text	参数的详细描述。
context	text	设置参数值的上下文，包括internal，postmaster，sighup，backend，superuser，user。
vartype	text	参数类型，包括bool，enum，integer，real，string。
source	text	参数的赋值方式。
min_val	text	参数最大值。如果参数类型不是数值型，那么该字段值为null。
max_val	text	参数最小值。如果参数类型不是数值型，那么该字段值为null。
enumvals	text[]	enum类型参数合法值。如果参数类型不是enum型，那么该字段值为null。

名称	类型	描述
boot_val	text	数据库启动时参数默认值。
reset_val	text	数据库重置时参数默认值。
sourcefile	text	设置参数值的配置文件。如果参数不是通过配置文件赋值，那么该字段值为null。
sourceline	integer	设置参数值的配置文件的行号。如果参数不是通过配置文件赋值，那么该字段值为null。

13.2.15.2 GLOBAL_CONFIG_SETTINGS

GLOBAL_CONFIG_SETTINGS显示各节点数据库运行时参数的相关信息。

表 13-173 GLOBAL_CONFIG_SETTINGS 的字段

名称	类型	描述
node_name	text	节点名称。
name	text	参数名称。
setting	text	参数当前值。
unit	text	参数的隐式结构。
category	text	参数的逻辑组。
short_desc	text	参数的简单描述。
extra_desc	text	参数的详细描述。
context	text	设置参数值的上下文，包括internal，postmaster，sighup，backend，superuser，user。
vartype	text	参数类型，包括bool，enum，integer，real，string。
source	text	参数的赋值方式。
min_val	text	参数最大值。如果参数类型不是数值型，那么该字段值为null。
max_val	text	参数最小值。如果参数类型不是数值型，那么该字段值为null。
enumvals	text[]	enum类型参数合法值。如果参数类型不是enum型，那么该字段值为null。
boot_val	text	数据库启动时参数默认值。
reset_val	text	数据库重置时参数默认值。

名称	类型	描述
sourcefile	text	设置参数值的配置文件。如果参数不是通过配置文件赋值，那么该字段值为null。
sourceline	integer	设置参数值的配置文件的行号。如果参数不是通过配置文件赋值，那么该字段值为null。

13.2.16 Operator

13.2.16.1 OPERATOR_HISTORY_TABLE

OPERATOR_HISTORY_TABLE系统视图显示执行作业结束后的算子相关的记录。此数据是从内核中转储到系统视图中的数据。当设置GUC参数enable_resource_record为on时，系统会定时（周期为3分钟）将GS_WLM_OPERATOR_HISTORY中的记录导入此系统表，开启此功能会占用系统存储空间并对性能有一定影响，不建议用户使用。

表 13-174 OPERATOR_HISTORY_TABLE 的字段

名称	类型	描述
queryid	bigint	语句执行使用的内部query_id。
pid	bigint	后端线程id。
plan_node_id	integer	查询对应的执行计划的plan node id。
plan_node_name	text	对应于plan_node_id的算子的名称。
start_time	timestamp with time zone	该算子处理第一条数据的开始时间。
duration	bigint	该算子到结束时候总的执行时间(ms)。
query_dop	integer	当前算子执行时的并行度。
estimated_rows	bigint	优化器估算的行数信息。
tuple_processed	bigint	当前算子返回的元素个数。
min_peak_memory	integer	当前算子在所有DN上的最小内存峰值(MB)。
max_peak_memory	integer	当前算子在所有DN上的最大内存峰值(MB)。
average_peak_memory	integer	当前算子在所有DN上的平均内存峰值(MB)。
memory_skew_percent	integer	当前算子在各DN间的内存使用倾斜率。

名称	类型	描述
min_spill_size	integer	若发生下盘，所有DN上下盘的最小数据量(MB)，默认为0。
max_spill_size	integer	若发生下盘，所有DN上下盘的最大数据量(MB)，默认为0。
average_spill_size	integer	若发生下盘，所有DN上下盘的平均数据量(MB)，默认为0。
spill_skew_percent	integer	若发生下盘，DN间下盘倾斜率。
min_cpu_time	bigint	该算子在所有DN上的最小执行时间(ms)。
max_cpu_time	bigint	该算子在所有DN上的最大执行时间(ms)。
total_cpu_time	bigint	该算子在所有DN上的总执行时间(ms)。
cpu_skew_percent	integer	DN间执行时间的倾斜率。
warning	text	主要显示如下几类告警信息： ● Sort/SetOp/HashAgg/HashJoin spill ● Spill file size large than 256MB ● Broadcast size large than 100MB ● Early spill ● Spill times is greater than 3 ● Spill on memory adaptive ● Hash table conflict

13.2.16.2 OPERATOR_HISTORY

OPERATOR_HISTORY视图显示的是当前用户当前CN上执行作业结束后的算子的相关记录。内核中的数据会定时被清理，清理周期为3分钟。当GUC参数enable_resource_record为on时，视图中的记录每隔3分钟被转储到系统表GS_WLM_OPERATOR_INFO中一次，同时视图中的记录被删除；当GUC参数enable_resource_record为off时，记录在视图中的存留时间达到超期时间（超期时间为3分钟）后会被删除。记录的数据同GS_WLM_OPERATOR_INFO的字段。

13.2.16.3 OPERATOR_RUNTIME

OPERATOR_RUNTIME视图显示当前用户正在执行的作业的算子相关信息。

表 13-175 OPERATOR_RUNTIME 的字段

名称	类型	描述
queryid	bigint	语句执行使用的内部query_id。

名称	类型	描述
pid	bigint	后端线程id。
plan_node_id	integer	查询对应的执行计划的plan node id。
plan_node_name	text	对应于plan_node_id的算子的名称。
start_time	timestamp with time zone	该算子处理第一条数据的开始时间。
duration	bigint	该算子到结束时候总的执行时间(ms)。
status	text	当前算子的执行状态，包括finished和running。
query_dop	integer	当前算子执行时的并行度。
estimated_rows	bigint	优化器估算的行数信息。
tuple_processed	bigint	当前算子返回的元素个数。
min_peak_memory	integer	当前算子在所有DN上的最小内存峰值(MB)。
max_peak_memory	integer	当前算子在所有DN上的最大内存峰值(MB)。
average_peak_memory	integer	当前算子在所有DN上的平均内存峰值(MB)。
memory_skew_percent	integer	当前算子在各DN间的内存使用倾斜率。
min_spill_size	integer	若发生下盘，所有DN上下盘的最小数据量(MB)，默认为0。
max_spill_size	integer	若发生下盘，所有DN上下盘的最大数据量(MB)，默认为0。
average_spill_size	integer	若发生下盘，所有DN上下盘的平均数据量(MB)，默认为0。
spill_skew_percent	integer	若发生下盘，DN间下盘倾斜率。
min_cpu_time	bigint	该算子在所有DN上的最小执行时间(ms)。
max_cpu_time	bigint	该算子在所有DN上的最大执行时间(ms)。
total_cpu_time	bigint	该算子在所有DN上的总执行时间(ms)。
cpu_skew_percent	integer	DN间执行时间的倾斜率。

名称	类型	描述
warning	text	主要显示如下几类告警信息： ● Sort/SetOp/HashAgg/HashJoin spill ● Spill file size large than 256MB ● Broadcast size large than 100MB ● Early spill ● Spill times is greater than 3 ● Spill on memory adaptive ● Hash table conflict

13.2.16.4 GLOBAL_OPERATOR_HISTORY

GLOBAL_OPERATOR_HISTORY系统视图显示的是当前用户在所有CN节点上执行作业结束后的算子的相关记录。

表 13-176 GLOBAL_OPERATOR_HISTORY 的字段

名称	类型	描述
queryid	bigint	语句执行使用的内部query_id。
pid	bigint	后端线程id。
plan_node_id	integer	查询对应的执行计划的plan node id。
plan_node_name	text	对应于plan_node_id的算子的名称。
start_time	timestamp with time zone	该算子处理第一条数据的开始时间。
duration	bigint	该算子到结束时候总的执行时间(ms)。
query_dop	integer	当前算子执行时的并行度。
estimated_rows	bigint	优化器估算的行数信息。
tuple_processed	bigint	当前算子返回的元素个数。
min_peak_memory	integer	当前算子在所有DN上的最小内存峰值(MB)。
max_peak_memory	integer	当前算子在所有DN上的最大内存峰值(MB)。
average_peak_memory	integer	当前算子在所有DN上的平均内存峰值(MB)。
memory_skew_percent	integer	当前算子在各DN间的内存使用倾斜率。

名称	类型	描述
min_spill_size	integer	若发生下盘，所有DN上下盘的最小数据量(MB)，默认为0。
max_spill_size	integer	若发生下盘，所有DN上下盘的最大数据量(MB)，默认为0。
average_spill_size	integer	若发生下盘，所有DN上下盘的平均数据量(MB)，默认为0。
spill_skew_percent	integer	若发生下盘，DN间下盘倾斜率。
min_cpu_time	bigint	该算子在所有DN上的最小执行时间(ms)。
max_cpu_time	bigint	该算子在所有DN上的最大执行时间(ms)。
total_cpu_time	bigint	该算子在所有DN上的总执行时间(ms)。
cpu_skew_percent	integer	DN间执行时间的倾斜率。
warning	text	主要显示如下几类告警信息： 1. Sort/SetOp/HashAgg/HashJoin spill 2. Spill file size large than 256MB 3. Broadcast size large than 100MB 4. Early spill 5. Spill times is greater than 3 6. Spill on memory adaptive 7. Hash table conflict

13.2.16.5 GLOBAL_OPERATOR_HISTORY_TABLE

GLOBAL_OPERATOR_HISTORY_TABLE视图显示所有CN节点执行作业结束后的算子相关的记录。此数据是从内核中转储到系统表GS_WLM_OPERATOR_INFO中的数据。当设置GUC参数enable_resource_record为on时，系统会定时（周期为3分钟）将GS_WLM_OPERATOR_HISTORY中的记录导入系统表GS_WLM_OPERATOR_INFO。该视图是查询所有CN节点系统表GS_WLM_OPERATOR_INFO的汇聚视图。表字段同**表13-176**。

13.2.16.6 GLOBAL_OPERATOR_RUNTIME

GLOBAL_OPERATOR_RUNTIME视图显示当前用户在所有CN节点上正在执行的作业的算子相关信息。

表 13-177 GLOBAL_OPERATOR_RUNTIME 的字段

名称	类型	描述
queryid	bigint	语句执行使用的内部query_id。

名称	类型	描述
pid	bigint	后端线程id。
plan_node_id	integer	查询对应的执行计划的plan node id。
plan_node_name	text	对应于plan_node_id的算子的名称。
start_time	timestamp with time zone	该算子处理第一条数据的开始时间。
duration	bigint	该算子到结束时候总的执行时间(ms)。
status	text	当前算子的执行状态，包括finished和running。
query_dop	integer	当前算子执行时的并行度。
estimated_rows	bigint	优化器估算的行数信息。
tuple_processed	bigint	当前算子返回的元素个数。
min_peak_memory	integer	当前算子在所有DN上的最小内存峰值(MB)。
max_peak_memory	integer	当前算子在所有DN上的最大内存峰值(MB)。
average_peak_memory	integer	当前算子在所有DN上的平均内存峰值(MB)。
memory_skew_percent	integer	当前算子在各DN间的内存使用倾斜率。
min_spill_size	integer	若发生下盘，所有DN上下盘的最小数据量(MB)，默认为0。
max_spill_size	integer	若发生下盘，所有DN上下盘的最大数据量(MB)，默认为0。
average_spill_size	integer	若发生下盘，所有DN上下盘的平均数据量(MB)，默认为0。
spill_skew_percent	integer	若发生下盘，DN间下盘倾斜率。
min_cpu_time	bigint	该算子在所有DN上的最小执行时间(ms)。
max_cpu_time	bigint	该算子在所有DN上的最大执行时间(ms)。
total_cpu_time	bigint	该算子在所有DN上的总执行时间(ms)。
cpu_skew_percent	integer	DN间执行时间的倾斜率。

名称	类型	描述
warning	text	主要显示如下几类告警信息： • Sort/SetOp/HashAgg/HashJoin spill • Spill file size large than 256MB • Broadcast size large than 100MB • Early spill • Spill times is greater than 3 • Spill on memory adaptive • Hash table conflict

13.2.17 Global Plancache

13.2.17.1 LOCAL_PLANCACHE_STATUS

LOCAL_PLANCACHE_STATUS视图显示当前节点的GPC全局计划缓存状态信息。

表 13-178 LOCAL_PLANCACHE_STATUS 字段

名称	类型	描述
nodename	text	所属节点名称。
query	text	查询语句text。
refcount	integer	被引用次数。
valid	bool	是否合法。
databaseid	oid	所属数据库id。
schema_name	text	所属schema。
params_num	integer	参数数量。
func_id	oid	该plancache所在存储过程oid，如果不属于存储过程则为0。

13.2.17.2 GLOBAL_PLANCACHE_STATUS

GLOBAL_PLANCACHE_STATUS视图显示全部节点上GPC全局计划缓存的状态信息，具体的字段见**LOCAL_PLANCACHE_STATUS**。

13.2.17.3 LOCAL_PREPARE_STATEMENT_STATUS（废弃）

LOCAL_PREPARE_STATEMENT_STATUS视图显示当前节点的GPC全局计划缓存状态对应的prepare statement信息。

表 13-179 LOCAL_PREPARE_STATEMENT_STATUS 字段

名称	类型	描述
nodename	text	所属节点名称。
cn_sess_id	bigint	其来自的cn的sessionid。
cn_node_id	integer	其来自的cn的node_id。
cn_time_line	integer	其来自的cn的重启次数。
statement_name	text	其statement name。
refcount	integer	其对应的plancache的被引用次数。
is_shared	bool	其对应plancache是否共享。
query	text	对应的query语句。

13.2.17.4 GLOBAL_PREPARE_STATEMENT_STATUS（废弃）

GLOBAL_PREPARE_STATEMENT_STATUS视图显示全部节点的GPC全局计划缓存状态对应的prepare statement信息。具体字段见**LOCAL_PREPARE_STATEMENT_STATUS（废弃）**

13.2.18 RTO & RPO

13.2.18.1 global_rto_status

global_rto_status视图显示关于主机和备机的日志流控信息（本节点除外、DN上不可使用）。

表 13-180 global_rto_status 字段

参数	类型	描述
node_name	text	节点的名称，包含主机和备机。
rto_info	text	流控的信息，包含了备机当前的日志流控时间（单位：秒），备机通过GUC参数设置的预期流控时间（单位：秒），为了达到这个预期主机所需要的睡眠时间（单位：微秒）。

13.2.18.2 global_streaming_hadr_rto_and_rpo_stat

global_streaming_hadr_rto_and_rpo_stat视图显示流式容灾的主集群和备集群日志流控信息（只可在主集群的CN使用，DN以及备集群上均不可获取到统计信息）。

表 13-181 global_streaming_hadr_rto_and_rpo_stat 参数说明

参数	类型	描述
hadr_sender_node_name	text	节点的名称，包含主集群和备集群首备。
hadr_receiver_node_name	text	备集群首备名称。
current_rto	int	流控的信息，当前主备集群的日志rto时间（单位：秒）。
target_rto	int	流控的信息，目标主备集群间的rto时间（单位：秒）。
current_rpo	int	流控的信息，当前主备集群的日志rpo时间（单位：秒）。
target_rpo	int	流控的信息，目标主备集群间的rpo时间（单位：秒）。
rto_sleep_time	int	RTO流控信息，为了达到目标RTO，预期主机walsender所需要的睡眠时间（单位：微秒）。
rpo_sleep_time	int	RPO流控信息，为了达到目标RPO，预期主机xlogInsert所需要的睡眠时间（单位：微秒）。

14 逻辑复制

14.1 逻辑解码

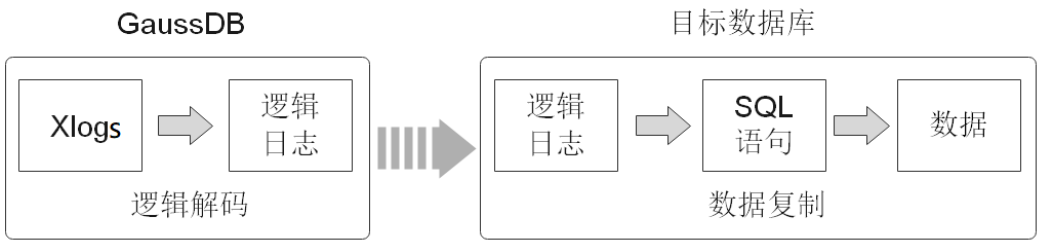
14.1.1 逻辑解码概述

GaussDB对数据复制能力的支持情况如下：

- 支持通过数据迁移工具定期向异构数据库（如Oracle等）进行数据同步，不具备实时数据复制能力。不足以支撑与异构数据库间并网运行实时数据同步的诉求。
- 备集群不支持异形部署，即不支持CN、DN数量和实例部署形态与原集群不一致。备集群恢复期间不支持读写操作，且复制时延相对较高。

基于上述两点，GaussDB提供了逻辑解码功能，通过反解xLog的方式生成逻辑日志。目标数据库解析逻辑日志以实时进行数据复制。具体如图14-1所示。逻辑复制降低了对目标数据库的形态限制，支持异构数据库、同构异形数据库对数据的同步，支持目标库进行数据同步期间的数据可读写，数据同步时延低。

图 14-1 逻辑复制



逻辑复制由两部分组成：逻辑解码和数据复制。逻辑解码会输出以事务为单位组织的逻辑日志。业务或数据库中间件将会对逻辑日志进行解析并最终实现数据复制。GaussDB当前只提供逻辑解码功能，因此本章节只涉及逻辑解码的说明。

功能描述

逻辑解码为逻辑复制提供事务解码的基础能力，GaussDB使用SQL函数接口进行逻辑解码。此方法调用方便，不需使用工具，对接外部工具接口也比较清晰，不需要额外适配。

由于逻辑日志是以事务为单位的，在事务提交后才能输出，且逻辑解码是由用户驱动的；因此为了防止事务开始时的xLog被系统回收，或所需的事务信息被VACUUM回收，GaussDB新增了逻辑复制槽，用于阻塞xLog的回收。

一个逻辑复制槽表示一个更改流，这些更改可以在其它集群上以它们在原集群上产生的顺序重新执行。每个逻辑复制槽都由其对应逻辑日志的获取者维护。

前提条件

- 逻辑日志目前从DN中抽取，如果进行逻辑复制，应使用SSL连接，需要保证相应DN上的GUC参数ssl设置为on。

  说明

  为避免安全风险，请保证启用SSL连接。

- 设置GUC参数wal_level=logical。
- 设置GUC参数max_replication_slots>=每个DN所需的物理流复制槽数、备份槽数、逻辑复制槽数之和。

  说明

  关于逻辑复制槽数，请按如下规则考虑。

  - 一个逻辑复制槽只能解码一个Database的修改，如果需要解码多个Database，则需要创建多个逻辑复制槽。
  - 如果需要多路逻辑复制同步给多个目标数据库，在源端数据库需要创建多个逻辑复制槽，每个逻辑复制槽对应一条逻辑复制链路。

- 用户需要通过DN端口连接数据库，才可以直接使用SQL函数接口进行逻辑解码操作。如果使用CN端口连接数据库，则需要通过EXECUTE DIRECT ON (datanode_name) 'statement'语句来执行SQL函数。
- 仅限初始用户和拥有REPLICATION权限的用户进行操作。三权分立关闭时数据库管理员可进行逻辑复制操作，三权分立开启时不允许数据库管理员进行逻辑复制操作。
- 并行解码需要配置充足的内存资源，内存占用主要分为如下四部分：
  - TOAST元组拼接哈希表：并发事务数量 * 元组平均大小，当前版本不受落盘逻辑管控，处理TOAST元组时可能占用大量内存。
  - 队列内存：队列长度 * 解码并行度 * 元组平均大小 * 2 (有读取日志队列和解码结果队列两种)，通过队列长度、解码并行度控制。
  - 并行解码日志发送重排：并发事务数量 * 事务修改元组数量 * 元组平均大小，可通过解码配置选项max-txn-in-memory及max-reorderbuffer-in-memory管控上限。
  - 事务元信息内存：并发事务数量 * 每个事务元信息大小（约300字节）。

注意事项

- 逻辑解码不支持DDL。
- 在执行特定的DDL语句（如普通表truncate或分区表exchange）时，可能造成解码数据丢失。

- 不支持数据页复制的DML解码。
- 不支持解码分布式事务，当前机制为从DN解码，无法保证分布式事务一致性解码。
- 当执行DDL语句（如alter table）后，该DDL语句前尚未解码的物理日志可能会丢失。
- 使用逻辑解码功能时，禁止进行集群在线扩容。
- 单条元组大小不超过1GB，考虑解码结果可能大于插入数据，因此建议单条元组大小不超过500MB。
- GaussDB支持解码的数据类型为：INTEGER、BIGINT、SMALLINT、TINYINT、SERIAL、SMALLSERIAL、BIGSERIAL、FLOAT、DOUBLE PRECISION、DATE、TIME[WITHOUT TIME ZONE]、TIMESTAMP[WITHOUT TIME ZONE]、CHAR(n)、VARCHAR(n)、TEXT。
- 如果需要ssl连接需要保证前置条件guc参数ssl设置为on。
- 逻辑复制槽名称必须小于64个字符，仅支持小写字母、数字以及_?-.字符，且不支持“.”或“..”单独作为复制槽名称。
- 当逻辑复制槽所在数据库被删除后，这些复制槽变为不可用状态，需要用户手动删除。
- 不支持interval partition表复制。
- 对多库的解码需要分别在库内创建流复制槽并开始解码，每个库的解码都需要单独扫一遍日志。
- 不支持强切，强切后需要重新全量导出数据。
- 在事务中执行DDL语句后，该DDL语句与之后的语句不会被解码。
- 如需进行备机解码，需在对应主机上设置GUC参数enable_slot_log = on。
- 备机解码时，switchover和failover时可能出现解码数据变多，需用户手动过滤。Quorum协议下，switchover和failover选择升主的备机，需要与当前主机日志同步。
- 不允许主备，多个备机同时使用同一个复制槽解码，否则会产生数据不一致。
- 只支持主机创建删除复制槽。
- 数据库故障重启或逻辑复制进程重启后，解码数据可能存在重复，用户需自己过滤。
- 计算机内核故障后，解码可能存在乱码，需手动或自动过滤。
- 当前备机逻辑解码，不支持开启极致RTO。
- 请确保在创建逻辑复制槽过程中长事务未启动，启动长事务会阻塞逻辑复制槽的创建。
- 为解析某个astore表的UPDATE和DELETE语句，需为此表配置REPLICA IDENTITY属性，在此表无主键时需要配置为FULL，配置方式参照**▪REPLICA IDENTITY { DEFA...**字段。
- 禁止在使用逻辑复制槽时在其他节点对该复制槽进行操作，删除复制槽的操作需在该复制槽停止解码后执行。
- 基于目标库可能需要源库的系统状态信息考虑，逻辑解码仅自动过滤模式'pg_catalog'和'pg_toast'下OID小于16384的系统表的逻辑日志。若目标库不需要复制其他相关系统表的内容，逻辑日志回放过程中需要对相关系统表进行过滤。
- 在开启逻辑复制的场景下，如需创建包含系统列的主键索引，必须将该表的REPLICA IDENTITY属性设置为FULL或是使用USING INDEX指定不包含系统列的、唯一的、非局部的、不可延迟的、仅包括标记为NOT NULL的列的索引。

- 在逻辑复制槽使用完毕后，需及时删除，否则会阻塞xLog日志回收。
- 若一个事务的子事务过多导致落盘文件过多，退出解码时需执行SQL函数pg_terminate_backend(*逻辑解码的walsender线程id*)来手动停止解码，而且退出时延增加约为1分钟/30万个子事务。因此在开启逻辑解码时，若一个事务的子事务数量达到5万时，会打印一条WARNING日志。
- 当同一事务产生大量需要落盘的子事务时，同时打开的文件句柄可能会超限，需将GUC参数max_files_per_process配置成大于子事务数量上限的两倍。

SQL 函数解码性能

在Benchmarksql-5.0的100warehouse场景下，采用pg_logical_slot_get_changes时：

- 单次解码数据量4K行（对应约5MB～10MB日志），解码性能0.3MB/s～0.5 MB/s。
- 单次解码数据量32K行（对应约40MB～80MB日志），解码性能3MB/s～5MB/s。
- 单次解码数据量256K行（对应约320MB～640MB日志），解码性能3MB/s～5MB/s。
- 单次解码数据量再增大，解码性能无明显提升。

如果采用pg_logical_slot_peek_changes + pg_replication_slot_advance方式，解码性能相比采用pg_logical_slot_get_changes时要下降30%～50%。

14.1.2 逻辑解码选项

通用选项（串行解码和并行解码均可配置，但可能无效，请参考相关选项详细说明）

- include-xids：

  解码出的data列是否包含xid信息。

  取值范围：0或1，默认值为1。

  - 0：设为0时，解码出的data列不包含xid信息。
  - 1：设为1时，解码出的data列包含xid信息。

- skip-empty-xacts：

  解码时是否忽略空事务信息。

  取值范围：0或1，默认值为0。

  - 0：设为0时，解码时不忽略空事务信息。
  - 1：设为1时，解码时会忽略空事务信息。

- include-timestamp：

  解码信息是否包含commit时间戳。

  取值范围：0或1，默认值为0。

  - 0：设为0时，解码信息不包含commit时间戳。
  - 1：设为1时，解码信息包含commit时间戳。

- only-local：

  是否仅解码本地日志。

  取值范围：0或1，默认值为1。

  - 0：设为0时，解码非本地日志和本地日志。

  - 1：设为1时，仅解码本地日志。

- white-table-list：

  白名单参数，包含需要进行解码的schema和表名。

  取值范围：包含白名单中表名的字符串，不同的表以','为分隔符进行隔离；使用'*'来模糊匹配所有情况；schema名和表名间以'.'分隔，不允许存在任意空白符。例如：

  ```
  select * from pg_logical_slot_peek_changes('slot1', NULL, 4096, 'white-table-list',
  'public.t1,public.t2,*.t3,my_schema.*');
  ```

- max-txn-in-memory：

  内存管控参数，单位为MB，单个事务占用内存大于该值即进行落盘。

  取值范围：0~100的整型，默认值为0，即不开启此种管控。

- max-reorderbuffer-in-memory

  内存管控参数，单位为GB，拼接-发送线程中正在拼接的事务总内存（包含缓存）大于该值则对当前解码事务进行落盘。

  取值范围：0~100的整型，默认值为0，即不开启此种管控。

- include-user：

  事务的BEGIN逻辑日志是否输出事务的用户名。事务的用户名特指授权用户——执行事务对应会话的登录用户，它在事务的整个执行过程中不会发生变化。

  取值范围：0或1，默认值为0。

  - 0：设为0时，事物的BEGIN逻辑日志不输出事务的用户名。
  - 1：设为1时，事物的BEGIN逻辑日志输出事务的用户名。

- exclude-userids：

  黑名单用户的OID参数。

  取值范围：字符串类型，指定黑名单用户的OID，多个OID通过','分隔，不校验用户OID是否存在。

- exclude-users：

  黑名单用户的名称列表。

  取值范围：字符串类型，指定黑名单用户名，通过','分隔，不校验用户名是否存在。

- dynamic-resolution：

  是否动态解析黑名单用户名。

  取值范围：0或1，默认值为1。

  - 0：设为0时，当解码观测到黑名单exclude-users中用户不存在时将会报错并退出逻辑解码。
  - 1：设为1时，当解码观测到黑名单exclude-users中用户不存在时继续解码。

- standby-connection：

  仅流式解码设置，是否仅限制备机解码。

  取值范围：bool型，默认值为false。

  - true：设为true时，仅允许连接备机解码，连接主机解码时会报错退出。
  - false：设为false时，不做限制，允许连接主机或备机解码。

- sender-timeout：

  仅流式解码设置，内核与客户端的心跳超时阈值。当该时间段内没有收到客户端任何消息，逻辑解码将主动停止，并断开和客户端的连接。单位为毫秒（ms）。

取值范围：0~2147483647的int型，默认值取决于GUC参数logical_sender_timeout的配置值。

- parallel-decode-num：

  仅流式解码设置有效，并行解码的Decoder线程数量；系统函数调用场景下此选项无效，仅校验取值范围。

  取值范围：取1表示按照原有的串行逻辑进行解码，取其余值即为开启并行解码，默认值为1。

须知

当parallel-decode-num不配置（即为默认值1）或显式配置为1时，下述“并行解码”中的选项不可配置。

串行解码

- force-binary：

  是否以二进制格式输出解码结果，针对不同场景呈现不同行为。

  - 针对系统函数pg_logical_slot_get_binary_changes和pg_logical_slot_peek_binary_changes：

    取值范围：bool型，默认值为false。此值无实际意义，均以二进制格式输出解码结果。
  - 针对系统函数pg_logical_slot_get_changes、pg_logical_slot_peek_changes和pg_logical_get_area_changes：

    取值范围：仅取false值的bool型。以文本格式输出解码结果。
  - 针对流式解码：

    取值范围：bool型，默认值为false。此值无实际意义，均以文本格式输出解码结果。

并行解码

- 以下配置选项仅限流式解码设置。
  - decode-style：

    指定解码格式。

    取值范围：char型的字符'j'、't'或'b'，分别代表json格式，text格式及二进制格式。默认值为'b'即二进制格式解码。

    对于json格式和text格式解码，开启批量发送选项时的解码结果中，每条解码语句的前4字节组成的uint32代表该条语句总字节数（不包含该uint32类型占用的4字节，0代表本批次解码结束），8字节uint64代表相应lsn（begin对应first_lsn，commit对应end_lsn，其他场景对应该条语句的lsn）。

说明

二进制格式编码规则如下所示：

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

- sending-batch：

  指定是否批量发送。

  取值范围：0或1的int型，默认值为0。

  - 0：设为0时，表示逐条发送解码结果。
  - 1：设为1时，表示解码结果累积到达1MB则批量发送解码结果。

开启批量发送的场景中，当解码格式为'j'或't'时，在原来的每条解码语句之前会附加一个uint32类型，表示本条解码结果长度（长度不包含当前的uint32类型），以及一个uint64类型，表示当前解码结果对应的lsn。

- parallel-queue-size：

  指定并行逻辑解码线程间进行交互的队列长度。

  取值范围：2~1024的int型，且必须为2的整数幂，默认值为128。

  队列长度和解码过程的内存使用量正相关。

14.1.3 使用 SQL 函数接口进行逻辑解码

GaussDB可以通过调用SQL函数，进行创建、删除、推进逻辑复制槽，获取解码后的事务日志。

操作步骤

步骤1 以具有REPLICATION权限的用户登录GaussDB集群任一主机。

步骤2 使用如下命令通过DN端口连接数据库。

```
gsql -U user1 -d gaussdb -p 40000 -r
```

其中，user1为用户名，gaussdb为需要连接的数据库名称，40000为数据库DN端口号，用户可根据实际情况替换。复制槽是建立在DN上的，因此需要通过DN端口连接数据库。

步骤3 创建名称为slot1的逻辑复制槽。

```
openGauss=# SELECT * FROM pg_create_logical_replication_slot('slot1', 'mppdb_decoding');
slotname | xlog_position
----------+---------------
slot1    | 0/601C150
(1 row)
```

步骤4 在数据库中创建表t，并向表t中插入数据。

```
openGauss=# CREATE TABLE t(a int PRIMARY KEY, b int);
openGauss=# INSERT INTO t VALUES(3,3);
```

步骤5 读取复制槽slot1解码结果，解码条数为4096。

```
openGauss=# SELECT * FROM pg_logical_slot_peek_changes('slot1', NULL, 4096);
location  |  xid  | data
-----------+-------
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------
-------------------------------------------
 0/601C188 | 1010023 | BEGIN 1010023
 0/601ED60 | 1010023 | COMMIT 1010023 CSN 1010022
 0/601ED60 | 1010024 | BEGIN 1010024
 0/601ED60 | 1010024 | {"table_name":"public.t","op_type":"INSERT","columns_name":
["a","b"],"columns_type":["integer","integer"],"columns_val":["3","3"],"old_keys_name":[],"old_keys_type":
[],"old_keys_val":[]}
 0/601EED8 | 1010024 | COMMIT 1010024 CSN 1010023
(5 rows)
```

步骤6 删除逻辑复制槽slot1。

```
openGauss=# SELECT * FROM pg_drop_replication_slot('slot1');
 pg_drop_replication_slot
--------------------------

(1 row)
```

----结束

14.1.4 使用逻辑复制工具复制数据

目前支持GaussDB逻辑复制的工具有SDR和DRS。复制工具从GaussDB抽取逻辑日志后到对端数据库回放。对于使用JDBC连接数据库的复制工具，具体代码请参考**示例：逻辑复制代码示例**。

15 GTM 模式

为适应不同的并发和一致性要求，GaussDB提供了三种不同的GTM模式，GTM，GTM-Lite，GTM-Free。三种模式之间的主要区别是中心事务管理节点GTM的压力和事务处理流程不同，其中GTM模式下，中心事务处理节点的压力最大，比较容易成为性能和并发瓶颈；在GTM-Lite模式下，中心事务处理节点的压力得到减轻，事务处理流程进一步优化，GTM的性能和并发瓶颈得到减轻，在保证一致性的情况下，事务处理能力得到更大限度的提升；在GTM-Free模式下，中心事务管理节点不再参与事务管理，消除了GTM单点瓶颈，可达到更高的事务处理性能，但是在一致性方面，支持所有事务运行完，保证读的外部一致性，不支持分布式事务强一致性读，不支持insert into select * from等依赖于查询结果的事务一致性。

当前版本暂不支持三种模式之间的相互切换，建议使用安装时默认的GTM模式，支持升级前后GTM模式不变化。

相关的GUC参数包括enable_gtm_free和gtm_option，可通过gsql执行show语句查询当前GTM模式：

```
SHOW enable_gtm_free;
SHOW gtm_option;
```

具体模式判断方法如下。

- GTM模式：enable_gtm_free=off 且 gtm_option=0；
- GTM-Lite模式：enable_gtm_free=off 且 gtm_option=1;
- GTM-Free模式：enable_gtm_free=on 或 gtm_option=2。

GTM-lite模式和GTM模式下，GaussDB支持分布式事务强一致和完整的语法功能。GTM-FREE模式下，由于采用了分布式事务最终一致的执行和并发控制机制，因此约束了部分语法的使用场景和使用方式。如果业务确有使用被限制语法的需求，那么在明确了解最终一致性行为的基础之上，需要对业务进行一定程度的改造。主要涉及的约束语法和改造建议如下：

1. 总体原则：所有用户表必须指定分布键（DISTRIBUTE BY），且选择合理：
   - 考虑数据分布均匀。
   - 尽量选择查询中的关联条件作为分布键，保证关联查询不会引起DN节点之间的数据流动。
   - 考虑将表的主键作为分布键。
2. SELECT：
   - 表查询时，WHERE条件应包含所有分布键字段等值查询条件。

- 避免在SELECT目标列中使用子查询，可能导致计划无法下推到DN执行，影响执行性能。

3. DML:

   默认不支持跨节点事务，如果所执行的DML语句包含跨节点事务，会报错处理，具体分为两种场景：

   a. 如果用户语句在数据库内部被拆分成多条独立语句执行，会报错：INSERT/UPDATE/DELETE/MERGE contains multiple remote queries under GTM-free mode Unsupport DML two phase commit under gtm free mode. modify your SQL to generate light-proxy or fast-query-shipping plan。这时，需要修改语句，来单节点执行。

      举例：

      insert into t select * from b where b.c = xx;

      假设t表和b表的分布键不同，且上述where条件只会过滤出一条数据。在不打开enable_stream_operator的情况下， 上面的查询在数据内部会被拆分成两条独立语句串行执行：首先执行select * from b where b.c = xx从某个DN节点抽取到目标记录；然后再执行insert into t语句，将抽取的目标记录下发到另一个节点DN节点完成插入。在gtm-free模式下，这样的语句执行方式会返回上述报错。类似的，create table as select * from、带子查询的delete/join/insert等语句，也可能会出现类似报错。

      业务改造方案：在业务执行之前，需要加上set enable_stream_operator=on命令，打开流算子，使得业务语句可以被整体下推执行。

   b. 如果同一个用户语句在数据库内部涉及多节点执行，会报错：Your SQL needs more than one datanode to be involved in。这时，建议对语句进行修改，使得能够单节点执行。

      举例：

      insert into t values(3,3),(1,1);

      假设(3,3)和(1,1)被分布在不同的DN节点上，那么上述语句在数据库内部的执行过程会涉及两个DN节点。在gtm-free模式下，这样的语句执行方式会返回上述报错。

      业务改造方式：对于上述语句，如果业务确有需要在多个节点上执行，需要在语句中添加一个hint来避免报错，如下：

      insert /*+ multinode */ into t values(3,3),(1,1);

      类似的，对delete和update语句也有类似约束，一般建议用户在delete和update语句的where条件中加上分布键等值过滤条件。

4. 建议开发阶段在jdbc连接串内设置application_type=perfect_sharding_type，这样所有跨节点读写操作的SQL都会报错，用来提示开发人员尽早优化语句。

16 物化视图

物化视图实际上就是一种特殊的物理表，物化视图是相对普通视图而言的。普通视图是虚拟表，应用的局限性较大，任何对视图的查询实际上都是转换为对SQL语句的查询，性能并没有实际上提高。而物化视图实际上就是存储SQL所执行语句的结果，起到缓存的效果。

16.1 全量物化视图

16.1.1 概述

全量物化视图仅支持对创建好的物化视图做全量更新，而不支持做增量更新。创建全量物化视图语法和CREATE TABLE AS语法一致，不支持对全量物化视图指定NodeGroup创建。

16.1.2 使用

语法格式

- 创建全量物化视图
  ```
  CREATE MATERIALIZED VIEW [ view_name ] AS { query_block };
  ```
- 全量刷新物化视图
  ```
  REFRESH MATERIALIZED VIEW [ view_name ];
  ```
- 删除物化视图
  ```
  DROP MATERIALIZED VIEW [ view_name ];
  ```
- 查询物化视图
  ```
  SELECT * FROM [ view_name ];
  ```

示例

```
-- 准备数据
CREATE TABLE t1(c1 int, c2 int);
INSERT INTO t1 VALUES(1, 1);
INSERT INTO t1 VALUES(2, 2);

-- 创建全量物化视图
openGauss=# CREATE MATERIALIZED VIEW mv AS select count(*) from t1;
CREATE MATERIALIZED VIEW

-- 查询物化视图结果
```



```
openGauss=# SELECT * FROM mv;
 count
-------
     2
(1 row)

-- 再次向物化视图中基表插入数据
openGauss=# INSERT INTO t1 VALUES(3, 3);

-- 对全量物化视图做全量刷新
openGauss=# REFRESH MATERIALIZED VIEW mv;
REFRESH MATERIALIZED VIEW

-- 查询物化视图结果
openGauss=# SELECT * FROM mv;
 count
-------
     3
(1 row)

-- 删除物化视图
openGauss=# DROP MATERIALIZED VIEW mv;
DROP MATERIALIZED VIEW
```

16.1.3 支持和约束

支持场景

- 大体上，全量物化视图所支持的查询范围与CREATE TABLE AS语句一致。
- 创建全量物化视图可以指定分布列。
- 可以在全量物化视图上创建索引。
- 支持analyze、explain。

不支持场景

- 全量物化视图不支持NodeGroup。
- 不可对物化视图做增删改操作，只支持查询语句。

约束

- 创建全量物化视图所使用的基表必须在所有DN上有定义，基表所属nodegroup必须为installation group。
- 全量物化视图的刷新、删除过程中会给基表加高级别锁，若物化视图的定义涉及多张表，需要注意业务逻辑，避免死锁产生。

16.2 增量物化视图

16.2.1 概述

增量物化视图顾名思义就是可以对物化视图增量刷新，需要用户手动执行语句完成对物化视图在一段时间内的增量数据进行刷新。与全量创建物化视图不同在于目前增量物化视图所支持场景较小，目前物化视图创建语句仅支持基表扫描语句或者UNION ALL语句。

16.2.2 使用

语法格式

- 创建增量物化视图
  ```
  CREATE INCREMENTAL MATERIALIZED VIEW [ view_name ] AS { query_block };
  ```
- 全量刷新物化视图
  ```
  REFRESH MATERIALIZED VIEW [ view_name ];
  ```
- 增量刷新物化视图
  ```
  REFRESH INCREMENTAL MATERIALIZED VIEW [ view_name ];
  ```
- 删除物化视图
  ```
  DROP MATERIALIZED VIEW [ view_name ];
  ```
- 查询物化视图
  ```
  SELECT * FROM [ view_name ];
  ```

示例

```
-- 准备数据
CREATE TABLE t1(c1 int, c2 int);
INSERT INTO t1 VALUES(1, 1);
INSERT INTO t1 VALUES(2, 2);

-- 创建增量物化视图
openGauss=# CREATE INCREMENTAL MATERIALIZED VIEW mv AS SELECT * FROM t1;
CREATE MATERIALIZED VIEW

-- 插入数据
openGauss=# INSERT INTO t1 VALUES(3, 3);
INSERT 0 1

-- 增量刷新物化视图
openGauss=# REFRESH INCREMENTAL MATERIALIZED VIEW mv;
REFRESH MATERIALIZED VIEW

-- 查询物化视图结果
openGauss=# SELECT * FROM mv;
 c1 | c2
----+----
  1 |  1
  2 |  2
  3 |  3
(3 rows)

-- 插入数据
openGauss=# INSERT INTO t1 VALUES(4, 4);
INSERT 0 1

-- 全量刷新物化视图
openGauss=# REFRESH MATERIALIZED VIEW mv;
REFRESH MATERIALIZED VIEW

-- 查询物化视图结果
openGauss=# select * from mv;
 c1 | c2
----+----
  1 |  1
  2 |  2
  3 |  3
  4 |  4
(4 rows)

-- 删除物化视图
openGauss=# DROP MATERIALIZED VIEW mv;
DROP MATERIALIZED VIEW
```


16.2.3 支持和约束

支持场景

- 单表查询语句
- 多个单表查询的UNION ALL
- 在物化视图上创建索引
- 对物化视图做Analyze操作
- 增量物化视图会继承基表NodeGroup创建（检查各个基表是否在同一个NodeGroup，并基于这个NodeGroup进行创建）。

不支持场景

- 物化视图中不支持带Stream计划，多表join连接计划以及subquery计划。
- 除少部分ALTER操作外，不支持对物化视图中基表做绝大多数DDL操作。
- 创建物化视图不可指定物化视图分布列。
- 不可对物化视图做增删改操作，只支持查询语句。
- 不支持用临时表/hashbucket/unlog/分区表创建物化视图，只支持hash分布表。
- 不支持物化视图嵌套创建（物化视图上创建物化视图）。
- 仅支持行存表。
- 不支持UNLOGGED类型的物化视图，不支持WITH语法。

约束

- 物化视图定义如果为UNION ALL，则其中每个子查询需使用不同的基表，且各基表分布列相同。物化视图的分布列会自动推导且与各基表相同。
- 物化视图定义的列必须包含基表的所有分布列。
- 增量物化视图的创建、全量刷新、删除过程中会给基表加高级别锁，若物化视图的定义为UNION ALL，需要注意业务逻辑，避免死锁产生。

17 错误日志信息参考

17.1 内核错误信息

ERRMSG: "unsupported syntax: ENCRYPTED WITH in this operation"

SQLSTATE: 42601

CAUSE: "client encryption feature is not supported this operation."

ACTION: "Check client encryption feature whether supported this operation."

ERRMSG: "invalid grant operation"

SQLSTATE: 0LP01

CAUSE: "Grant options cannnot be granted to public."

ACTION: "Grant grant options to roles."

ERRMSG: "unrecognized object kind: %d"

SQLSTATE: XX004

CAUSE: "The object type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "unrecognized GrantStmt.targtype: %d"

SQLSTATE: XX004

CAUSE: "The target type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported target types."

ERRMSG: "invalid grant operation"

SQLSTATE: 0LP01

CAUSE: "Grant to public operation is forbidden in security mode."

ACTION: "Don't grant to public in security mode."

ERRMSG: "unrecognized object type"

SQLSTATE: XX004

CAUSE: "The object type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "invalid grant/revoke operation"

SQLSTATE: 0LP01

CAUSE: "Column privileges are only valid for relations in GRANT/REVOKE."

ACTION: "Use the column privileges only for relations."

ERRMSG: "invalid AccessPriv node"

SQLSTATE: 0LP01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "unrecognized GrantStmt.objtype: %d"

SQLSTATE: XX004

CAUSE: "The object type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "undefined client master key"

SQLSTATE: 42705

CAUSE: "The client master key does not exist."

ACTION: "Check whether the client master key exists."

ERRMSG: "undefined column encryption key"

SQLSTATE: 42705

CAUSE: "The column encryption key does not exist."

ACTION: "Check whether the column encryption key exists."

ERRMSG: "large object %u does not exist"

SQLSTATE: 42704

CAUSE: "The large object does not exist."

ACTION: "Check whether the large object exists."

ERRMSG: "redundant options"

SQLSTATE: 42601

CAUSE: "The syntax 'schemas' is redundant in ALTER DEFAULT PRIVILEGES statement."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax."

ERRMSG: "redundant options"

SQLSTATE: 42601

CAUSE: "The syntax 'roles' is redundant in ALTER DEFAULT PRIVILEGES statement."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax."

ERRMSG: "option '%s' not recognized"

SQLSTATE: 42601

CAUSE: "The option in ALTER DEFAULT PRIVILEGES statement is not supported."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax."

ERRMSG: "unrecognized GrantStmt.objtype: %d"

SQLSTATE: XX004

CAUSE: "The object type is not supported for ALTER DEFAULT PRIVILEGES."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax to obtain the supported object types."

ERRMSG: "invalid alter default privileges operation"

SQLSTATE: 0LP01

CAUSE: "Default privileges cannot be set for columns."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax."

ERRMSG: "unrecognized objtype: %d"

SQLSTATE: XX004

CAUSE: "The object type is not supported for default privileges."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax to obtain the supported object types."

ERRMSG: "could not find tuple for default ACL %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "unexpected default ACL type: %d"

SQLSTATE: 0LP01

CAUSE: "The object type is not supported for default privilege."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax to obtain the supported object types."

ERRMSG: "invalid object id"

SQLSTATE: 0LP01

CAUSE: "The object type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "undefined column"

SQLSTATE: 42703

CAUSE: "The column of the relation does not exist."

ACTION: "Check whether the column exists."

ERRMSG: "column number out of range"

SQLSTATE: 0LP01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for attribute %d of relation %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for relation %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "unsupported object type"

SQLSTATE: 42809

CAUSE: "Index type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "unsupported object type"

SQLSTATE: 42809

CAUSE: "Composite type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "wrong object type"

SQLSTATE: 42809

CAUSE: "GRANT/REVOKE SEQUENCE only support sequence objects."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "invalid privilege type USAGE for table"

SQLSTATE: 0LP01

CAUSE: "GRANT/REVOKE TABLE do not support USAGE privilege."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported privilege types for tables."

ERRMSG: "invalid privilege type %s for column"

SQLSTATE: 0LP01

CAUSE: "The privilege type is not supported for column object."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported privilege types for column object."

ERRMSG: "cache lookup failed for database %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for foreign-data wrapper %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for foreign server %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for function %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for language %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "Grant/revoke on untrusted languages if forbidden."

SQLSTATE: 0LP01

CAUSE: "Grant/revoke on untrusted languages if forbidden."

ACTION: "Support grant/revoke on trusted C languages"

ERRMSG: "cache lookup failed for large object %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for namespace %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "Role %s has not privilege to grant/revoke node group %s."

SQLSTATE: 42501

CAUSE: "Role has not privilege to grant/revoke node group."

ACTION: "Must have sysadmin privilege."

ERRMSG: "Can not grant CREATE privilege on node group %u to role %u in node group %u."

SQLSTATE: 42501

CAUSE: "Role has not privilege to grant CREATE privilege node group."

ACTION: "Must have sysadmin privilege."

ERRMSG: "cache lookup failed for tablespace %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for type %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cannot set privileges of array types"

SQLSTATE: 0LP01

CAUSE: "Cannot set privileges of array types."

ACTION: "Set the privileges of the element type instead."

ERRMSG: "wrong object type"

SQLSTATE: 42809

CAUSE: "GRANT/REVOKE DOMAIN only support domain objects."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "cache lookup failed for client master key %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for column encryption key %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for directory %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "unrecognized privilege type '%s'"

SQLSTATE: 42601

CAUSE: "The privilege type is not supported."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported privilege types."

ERRMSG: "unrecognized privilege: %d"

SQLSTATE: XX004

CAUSE: "The privilege type is not supported."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported privilege types."

ERRMSG: "unrecognized AclResult"

SQLSTATE: XX004

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "permission denied for column '%s' of relation '%s'"

SQLSTATE: 42501

CAUSE: "Insufficient privileges for the column."

ACTION: "Select the system tables to get the acl of the column."

ERRMSG: "role with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "unrecognized objkind: %d"

SQLSTATE: XX004

CAUSE: "The object type is not supported for privilege check."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "attribute %d of relation with OID %u does not exist"

SQLSTATE: 42703

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "the column has been dropped"

SQLSTATE: 42703

CAUSE: "The column does not exist."

ACTION: "Check whether the column exists."

ERRMSG: "relation with OID %u does not exist"

SQLSTATE: 42P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "invalid group"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "database with OID %u does not exist"

SQLSTATE: 3D000

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "directory with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "function with OID %u does not exist"

SQLSTATE: 42883

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "client master key with OID %u does not exist"

SQLSTATE: 42705

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "language with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "large object %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "schema with OID %u does not exist"

SQLSTATE: 3F001

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "node group with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "tablespace with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "foreign-data wrapper with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "foreign server with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "type with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "operator with OID %u does not exist"

SQLSTATE: 42883

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "column encryption key with OID %u does not exist"

SQLSTATE: 42705

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "operator class with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "operator family with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "text search dictionary with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "text search configuration with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "collation with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "conversion with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "synonym with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "package can not create the same name with schema."

SQLSTATE: 22023

CAUSE: "Package name conflict"

ACTION: "Please rename package name"

ERRMSG: "type is not exists %s."

SQLSTATE: 22023

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "This input type is not supported for tdigest_in()"

SQLSTATE: 0A000

CAUSE: "input type is not supported"

ACTION: "Check tdigest_in syntax to obtain the supported privilege types"

ERRMSG: "Failed to apply for memory"

SQLSTATE: 53200

CAUSE: "palloc failed"

ACTION: "Check memory"

ERRMSG: "Failed to get tde info from relation '%s'."

SQLSTATE: XX005

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "SPI_connect failed: %s"

SQLSTATE: SP001

CAUSE: "System error."

ACTION: "Analyze the error message before the error"

ERRMSG: "permission denied for terminate snapshot thread"

SQLSTATE: 42501

CAUSE: "The user does not have system admin privilege"

ACTION: "Grant system admin to user"

ERRMSG: "terminate snapshot thread failed"

SQLSTATE: OP001

CAUSE: "Execution failed due to: %s"

ACTION: "check if snapshot thread exists"

ERRMSG: "terminate snapshot thread failed"

SQLSTATE: OP001

CAUSE: "restart wdr snapshot thread timeoutor The thread did not respond to the kill signal"

ACTION: "Check the wdr snapshot thread is restarted"

ERRMSG: "set lockwait_timeout failed"
SQLSTATE: XX000
CAUSE: "System error."
ACTION: "Contact engineer to support."

ERRMSG: "permission denied for create WDR Snapshot"
SQLSTATE: 42501
CAUSE: "The user does not have system admin privilege"
ACTION: "Grant system admin to user"

ERRMSG: "WDR snapshot request can not be accepted, please retry later"
SQLSTATE: OP001
CAUSE: "wdr snapshot thread does not exist"
ACTION: "Check if wdr snapshot thread exists"

ERRMSG: "Cannot respond to WDR snapshot request"
SQLSTATE: OP001
CAUSE: "Execution failed due to: %s"
ACTION: "Check if wdr snapshot thread exists"

ERRMSG: "query(%s) can not get datum values"
SQLSTATE: 22000
CAUSE: "System error."
ACTION: "Check whether the query can be executed"

ERRMSG: "create sequence failed"
SQLSTATE: 22000
CAUSE: "System error."
ACTION: "Check if sequence can be created"

ERRMSG: "update snapshot end time stamp filled"
SQLSTATE: 22000
CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "query can not get datum values"
SQLSTATE: 22000
CAUSE: "System error."
ACTION: "Check whether the query can be executed"

ERRMSG: "SPI_connect failed: %s"
SQLSTATE: XX000
CAUSE: "System error."
ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "query(%s) execute failed"
SQLSTATE: 22000
CAUSE: "System error."
ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "clean table of snap_%s is failed"
SQLSTATE: 22000
CAUSE: "System error."
ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "analyze table failed"
SQLSTATE: 22000
CAUSE: "System error."
ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "insert into tables_snap_timestamp start time stamp is failed"
SQLSTATE: 22000
CAUSE: "System error."
ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "insert data failed"
SQLSTATE: 22000
CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful and check whether the query can be executed"

ERRMSG: "update tables_snap_timestamp end time stamp is failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "clean snapshot id %lu is failed in snapshot table"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful and check whether the query can be executed"

ERRMSG: "clean snapshot failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "can not create snapshot stat table"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "create WDR snapshot data table failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "insert into tables_snap_timestamp start time stamp failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "insert into snap_%s is failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "update tables_snap_timestamp end time stamp failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "create index failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "analyze table, connection failed: %s"

SQLSTATE: XX000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "snapshot thread SPI_connect failed: %s"

SQLSTATE: XX000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "Distributed key column can't be transformed"

SQLSTATE: 42P10

CAUSE: "There is a risk of violating uniqueness when transforming distribution columns."

ACTION: "Change transform column."

ERRMSG: "cannot convert %s to %s"

SQLSTATE: 42804

CAUSE: "There is no conversion path in pg_cast."

ACTION: "Rewrite or cast the expression."

ERRMSG: "create matview on TDE table failed"

SQLSTATE: 0A000

CAUSE: "create materialized views is not supported on TDE table"

ACTION: "check CREATE syntax about create the materialized views"

ERRMSG: "schema name can not same as package"

SQLSTATE: 22023

CAUSE: "schema name conflict"

ACTION: "rename schema name"

ERRMSG: "Unrecognized commandType when checking read-only attribute."

SQLSTATE: XX004

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Fail to generate subquery plan."

SQLSTATE: XX005

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Unrecognized node type when processing qual condition."

SQLSTATE: XX004

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Unrecognized node type when processing const parameters."

SQLSTATE: XX004

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "SELECT FOR UPDATE/SHARE is not allowed with UNION/INTERSECT/EXCEPT"

SQLSTATE: 0A000

CAUSE: "SQL uses unsupported feature."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "GROUP BY cannot be implemented."

SQLSTATE: 0A000

CAUSE: "GROUP BY uses unsupported datatypes."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "Unrecognized node type when extracting index."

SQLSTATE: XX004

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Ordering operator cannot be identified."

SQLSTATE: 42883

CAUSE: "Grouping set columns must be able to sort their inputs."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "DISTINCT cannot be implemented."

SQLSTATE: 0A000

CAUSE: "DISTINCT uses unsupported datatypes."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "Failed to locate grouping columns."

SQLSTATE: 55000

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Resjunk output columns are not implemented."

SQLSTATE: 20000

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "PARTITION BY cannot be implemented."

SQLSTATE: 0A000

CAUSE: "PARTITION BY uses unsupported datatypes."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "ORDER BY cannot be implemented."

SQLSTATE: 0A000

CAUSE: "ORDER BY uses unsupported datatypes."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "Failed to deconstruct sort operators into partitioning/ordering operators."

SQLSTATE: D0011

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Pool size should not be zero"

SQLSTATE: 22012

CAUSE: "Compute pool configuration file contains error."

ACTION: "Please check the value of 'pl' in cp_client.conf."

ERRMSG: "Failed to get the runtime info from the compute pool."

SQLSTATE: 22004

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Version is not compatible between local cluster and the compute pool."

SQLSTATE: XX008

CAUSE: "Compute pool is not installed appropriately."

ACTION: "Configure compute pool according to manual."

ERRMSG: "No optional index path is found."

SQLSTATE: 01000

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "MERGE INTO on replicated table does not yet support using distributed tables."

SQLSTATE: 0A000

CAUSE: "SQL uses unsupported feature."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "Fail to find ForeignScan node!"

SQLSTATE: P0002

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "sql advisor don't support none table, temp table, system table."

SQLSTATE: 42601

CAUSE: "sql advisor don't support none table, temp table, system table."

ACTION: "check query component"

ERRMSG: "Invalid autonomous transaction return datatypes"

SQLSTATE: P0000

CAUSE: "PL/SQL uses unsupported feature."

ACTION: "Contact Huawei Engineer."

ERRMSG: "new row for relation '%s' violates check constraint '%s'"

SQLSTATE: 23514

CAUSE: "some rows copy failed"

ACTION: "check table defination"

ERRMSG: "new row for relation '%s' violates check constraint '%s'"

SQLSTATE: 23514

CAUSE: "some rows copy failed"

ACTION: "set client_min_messages = info for more details"

ERRMSG: "get gauss home path is NULL"

SQLSTATE: XX005

CAUSE: "gauss home path not set"

ACTION: "check if $GAUSSHOME is exist"

ERRMSG: "unable to open kms_iam_info.json file"

SQLSTATE: 58P03

CAUSE: "file not exist or broken"

ACTION: "check the kms_iam_info.json file"

ERRMSG: "can not get password plaintext"

SQLSTATE: XX005

CAUSE: "file not exist or broken"

ACTION: "check the password cipher rand file"

ERRMSG: "IAM info json key is NULL"

SQLSTATE: XX005

CAUSE: "IAM info value error"

ACTION: "check tde_config kms_iam_info.json file"

ERRMSG: "get internal password is NULL"

SQLSTATE: XX005

CAUSE: "cipher rand file missing"

ACTION: "check password cipher rand file"

ERRMSG: "KMS info json key is NULL"

SQLSTATE: XX005

CAUSE: "KMS info value error"

ACTION: "check tde_config kms_iam_info.json file"

ERRMSG: "unable to get json file"

SQLSTATE: 58P03

CAUSE: "parse json file failed"

ACTION: "check the kms_iam_info.json file format"

ERRMSG: "get JSON tree is NULL"

SQLSTATE: XX005

CAUSE: "get KMS JSON tree failed"

ACTION: "check input prarmeter or config.ini file"

ERRMSG: "failed to get json tree"

SQLSTATE: XX005

CAUSE: "config.ini json tree error"

ACTION: "check input prarmeter or config.ini file"

ERRMSG: "failed to set the value of json tree"

SQLSTATE: XX005

CAUSE: "config.ini json tree error"

ACTION: "check input prarmeter or config.ini file"

ERRMSG: "http request failed"

SQLSTATE: XX005

CAUSE: "http request error"

ACTION: "check KMS or IAM connect or config parameter"

ERRMSG: "get iam token or iam agency token is NULL"

SQLSTATE: XX005

CAUSE: "connect IAM failed"

ACTION: "check if your env can connect with IAM server"

ERRMSG: "KMS dek json key is NULL"

SQLSTATE: XX005

CAUSE: "KMS return value error"

ACTION: "check KMS config paramenter"

ERRMSG: "get kms dek is NULL"

SQLSTATE: XX005

CAUSE: "connect KMS failed"

ACTION: "check if your env can connect with KMS server"

ERRMSG: "get http header is NULL"

SQLSTATE: XX005

CAUSE: "http request failed"

ACTION: "check IAM config parameter"

ERRMSG: "create KMS dek failed"

SQLSTATE: XX005

CAUSE: "KMS error"

ACTION: "check KMS connect or config parameter"

ERRMSG: "get KMS dek failed"

SQLSTATE: XX005

CAUSE: "KMS error"

ACTION: "check KMS connect or config parameter"

ERRMSG: "get KMS DEK is NULL"

SQLSTATE: XX005

CAUSE: "get KMS dek_plaintext failed"

ACTION: "check KMS network or cipher is right"

ERRMSG: "create matview with TDE failed"

SQLSTATE: 0A000

CAUSE: "TDE feature is not supported for Create materialized views"

ACTION: "check CREATE syntax about create the materialized views"

ERRMSG: "failed to add item to the index page"

SQLSTATE: XX002

CAUSE: "System error."

ACTION: "Check WARNINGS for the details."

ERRMSG: "index row size %lu exceeds maximum %lu for index '%s'"

SQLSTATE: 54000

CAUSE: "Values larger than 1/3 of a buffer page cannot be indexed."

ACTION: "Consider a function index of an MD5 hash of the value, or use full text indexing."

ERRMSG: "fail to insert a tuple to an orderd index, the ordered tuple list is corrupted"

SQLSTATE: XX002

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "Tag field is too long."

SQLSTATE: 54000

CAUSE: "Tag buffer overflow."

ACTION: "Shorten tag Key."

17.2 CM 错误信息

ERRMSG: "Fail to access the cluster static config file."

SQLSTATE: c3000

CAUSE: "The cluster static config file is not generated or is manually deleted."

ACTION: "Please check the cluster static config file."

ERRMSG: "Fail to open the cluster static file."

SQLSTATE: c3000

CAUSE: "The cluster static config file is not generated or is manually deleted."

ACTION: "Please check the cluster static config file."

ERRMSG: "Fail to read the cluster static file."

SQLSTATE: c3001

CAUSE: "The cluster static file permission is insufficient."

ACTION: "Please check the cluster static config file."

ERRMSG: "Failed to read the static config file."

SQLSTATE: c1000

CAUSE: "out of memeory."

ACTION: "Please check the system memory and try again."

ERRMSG: "Could not find the current node in the cluster by the node id %u."

SQLSTATE: c3002

CAUSE: "The static config file probably contained content error."

ACTION: "Please check static config file."

ERRMSG: "Failed to open the logic config file."

SQLSTATE: c3000

CAUSE: "The logic config file is not generated or is manually deleted."

ACTION: "Please check the cluster static config file."

ERRMSG: "Fail to read the logic static config file."

SQLSTATE: c3001

CAUSE: "The logic static config file permission is insufficient."

ACTION: "Please check the logic static config file."

ERRMSG: "Failed to open or read the static config file."

SQLSTATE: c1000

CAUSE: "out of memeory."

ACTION: "Please check the system memory and try again."

ERRMSG: "Failed to open the log file '%s'."

SQLSTATE: c3000

CAUSE: "Log file not found."

ACTION: "Please check the log file."

ERRMSG: "Failed to open the log file '%s'."

SQLSTATE: c3000

CAUSE: "The log file permission is insufficient."

ACTION: "please check the log file."

ERRMSG: "Failed to open the dynamic config file '%s'."

SQLSTATE: c3000

CAUSE: "The dynamic config file permission is insufficient."

ACTION: "Please check the dynamic config file."

ERRMSG: "Failed to malloc memory, size = %lu."

SQLSTATE: c1000

CAUSE: "out of memeory."

ACTION: "Please check the system memory and try again."

ERRMSG: "unrecognized AZ name '%s'."

SQLSTATE: c3000

CAUSE: "The parameter(%s) entered by the user is incorrect."

ACTION: "Please check the parameter entered by the user and try again."

ERRMSG: "unrecognized minorityAz name '%s'."

SQLSTATE: c3000

CAUSE: "The parameter(%s) entered by the user is incorrect."

ACTION: "Please check the parameter entered by the user and try again."

ERRMSG: "Get GAUSSHOME failed."

SQLSTATE: c3000

CAUSE: "The environment variable('GAUSSHOME') is incorrectly configured."

ACTION: "Please check the environment variable('GAUSSHOME')."

ERRMSG: "Get current user name failed."

SQLSTATE: c3000

CAUSE: "N/A"

ACTION: "Please check the environment."

ERRMSG: "-B option must be specified."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-T option must be specified.\n"

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "can't stop one node or instance with -m normal."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "can't stop one node or instance with -m resume."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "can't stop one availability zone with -m resume."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "log level or cm server arbitration mode must be specified."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "log level or cm server arbitration mode need not be specified."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-R is needed."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-D is needed."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-n and -R are needed."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-n and -D are needed."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "no operation specified."
SQLSTATE: c3000
CAUSE: "The cmdline entered by the user is incorrect."
ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "no cm directory specified."
SQLSTATE: c3000
CAUSE: "The cmdline entered by the user is incorrect."
ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "Please check the usage of switchover."
SQLSTATE: c3000
CAUSE: "The cmdline entered by the user is incorrect."
ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-n and -z cannot be specified at the same time."
SQLSTATE: c3000
CAUSE: "The cmdline entered by the user is incorrect."
ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-m cannot be specified at the same time with -n or -z."
SQLSTATE: c3000
CAUSE: "%s: The cmdline entered by the user is incorrect."
ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-n node(%d) is invalid."
SQLSTATE: c3000
CAUSE: "The cmdline entered by the user is incorrect."
ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-n node is needed."
SQLSTATE: c3000
CAUSE: "The cmdline entered by the user is incorrect."
ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "%s: -C is needed."
SQLSTATE: c3000
CAUSE: "The cmdline entered by the user is incorrect."
ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-z value must be 'ALL' when query mppdb cluster."
SQLSTATE: c3000
CAUSE: "%s: The cmdline entered by the user is incorrect."
ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-v is needed."
SQLSTATE: c3000
CAUSE: "%s: The cmdline entered by the user is incorrect."
ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-C is needed."
SQLSTATE: c3000
CAUSE: "%s: The cmdline entered by the user is incorrect."
ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-Cv is needed."
SQLSTATE: c3000
CAUSE: "%s: The cmdline entered by the user is incorrect."
ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-L value must be 'ALL' when query logic cluster."
SQLSTATE: c3000
CAUSE: "%s: The cmdline entered by the user is incorrect."
ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "unrecognized LC name '%s'."
SQLSTATE: c3000
CAUSE: "%s: The cmdline entered by the user is incorrect."
ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-n is needed."
SQLSTATE: c3000
CAUSE: "%s: The cmdline entered by the user is incorrect."
ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "There is no '%s' information in cluster."
SQLSTATE: c3000
CAUSE: "%s: The cmdline entered by the user is incorrect."
ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-D path is too long.\n"
SQLSTATE: c3000
CAUSE: "%s: The cmdline entered by the user is incorrect."
ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-D path is invalid."
SQLSTATE: c3000
CAUSE: "%s: The cmdline entered by the user is incorrect."
ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-n node(%s) is invalid."
SQLSTATE: c3000
CAUSE: "%s: The cmdline entered by the user is incorrect."
ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-R only support when the cluster is single-inst."
SQLSTATE: c3000
CAUSE: "%s: The cmdline entered by the user is incorrect."
ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-t time is invalid."
SQLSTATE: c3000
CAUSE: "%s: The cmdline entered by the user is incorrect."
ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-votenum is invalid."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "unrecognized build mode."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "unrecognized build mode '%s'."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "too many command-line arguments (first is '%s')."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "unrecognized operation mode '%s'."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "no cm directory specified."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "Failed to malloc memory."

SQLSTATE: c1000

CAUSE: "out of memeory."

ACTION: "Please check the system memory and try again."

ERRMSG: "Failed to open etcd: %s."

SQLSTATE: c4000

CAUSE: "Etcd is abnoraml."

ACTION: "Please check the Cluster Status and try again."

ERRMSG: "[PATCH-ERROR] hotpatch command or path set error."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "no standby datanode in single node cluster."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "restart logic cluster failed."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "restart logic cluster failed"

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "The option parameter is not specified."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

18 配置运行参数

18.1 查看参数当前取值

GaussDB安装后，有一套默认的运行参数，为了使GaussDB与业务的配合度更高，用户需要根据业务场景和数据量的大小进行GUC参数调整。

操作步骤

步骤1 使用如下命令连接数据库，具体操作请参考**通过gsql连接实例**章节。

```
gsql -d postgres -p 8000
```

postgres为需要连接的数据库名称，8000为CN的端口号。

连接成功后，系统显示类似如下信息：

```
gsql((GaussDB Kernel VxxxRxxxCxx build f521c606) compiled at 2021-09-16 14:55:22 commit 2935 last mr 6385 release)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

openGauss=#
```

步骤2 查看数据库运行参数当前取值。

- 方法一：使用SHOW命令。
  - 使用如下命令查看单个参数：
    ```
    openGauss=# SHOW server_version;
    ```
    server_version显示数据库版本信息的参数。
  - 使用如下命令查看所有参数：
    ```
    openGauss=# SHOW ALL;
    ```
- 方法二：使用pg_settings视图。
  - 使用如下命令查看单个参数：
    ```
    openGauss=# SELECT * FROM pg_settings WHERE NAME='server_version';
    ```
  - 使用如下命令查看所有参数：
    ```
    openGauss=# SELECT * FROM pg_settings;
    ```

----结束

示例

查看客户端的字符编码类型。

```
openGauss=# SHOW client_encoding;
 client_encoding
-----------------
 UTF8
(1 row)
```

18.2 重设参数

GaussDB支持在管理控制台修改部分参数，建议在管理控制台上修改指定参数，如果需要修改的参数在管理控制台无法修改，请提前评估风险后再联系客服进行修改。

背景信息

GaussDB提供了多种修改GUC参数的方法，用户可以方便地针对数据库、用户、会话进行设置。

- 参数名称不区分大小写。
- 参数取值有整型、浮点型、字符串、布尔型和枚举型五类。
  - 布尔值可以是（on，off）、（true，false）、（yes，no）或者（1，0），且不区分大小写。
  - 枚举类型的取值是在系统表pg_settings的enumvals字段取值定义的。
- 对于有单位的参数，在设置时请指定单位，否则将使用默认的单位。
  - 参数的默认单位在系统表pg_settings的unit字段定义的。
  - 内存单位有：KB（千字节）、MB（兆字节）和GB（吉字节）。
  - 时间单位：ms（毫秒）、s（秒）、min（分钟）、h（小时）和d（天）。
- CN和DN参数可以同时进行设置，其他类型的参数不能同时进行设置。

具体参数说明请参见**GUC参数说明**。

GUC 参数设置

GaussDB提供了六类GUC参数，具体分类和设置方式请参考**表18-1**：

表 18-1 GUC 参数分类

参数类型	说明	设置方式
INTERNAL	固定参数，在创建数据库的时候确定，用户无法修改，只能通过show语法或者pg_settings视图进行查看。	无
POSTMASTER	数据库服务端参数，在数据库启动时确定，可以通过配置文件指定。	支持**表18-2**中的方式一。
SIGHUP	数据库全局参数，可在数据库启动时设置或者在数据库启动后，发送指令重新加载。	支持**表18-2**中的方式一、方式二。

参数类型	说明	设置方式
BACKEND	会话连接参数。在创建会话连接时指定，连接建立后无法修改。连接断掉后参数失效。内部使用参数，不推荐用户设置。	支持表18-2中的方式一、方式二。 **说明** 设置该参数后，下一次建立会话连接时生效。
SUSET	数据库管理员参数。可在数据库启动时、数据库启动后或者数据库管理员通过SQL进行设置。	支持表18-2中的方式一、方式二或由数据库管理员通过方式三设置。
USERSET	普通用户参数。可被任何用户在任何时刻设置。	支持表18-2中的方式一、方式二或方式三设置。 **说明** 设置USERSET类型的参数时，ALTER DATABASE设置的参数值优先级高于gs_guc设置。如果想要gs_guc设置的参数值生效，则需要执行“alter database *xxx* reset *xxx*”进行重置。

GaussDB提供了三种方式来修改GUC参数，具体操作请参考表18-2：

表 18-2 GUC 参数设置方式

序号	设置方法
方式一	1. 登录管理控制台。 2. 在“实例管理”页面，选择指定的实例，单击实例名称，进入实例基本信息页面。 3. 在左侧导航栏单击“参数修改”，进入参数修改页面，在该页面修改参数。 如果需要修改的参数在管理该控制台无法修改，请提前评估风险后再联系客服进行修改。 4. 重启数据库使参数生效。 **说明** 重启数据库集群操作会导致用户执行操作中断，请在操作之前规划好合适的执行窗口。
方式二	1. 登录管理控制台。 2. 在“实例管理”页面，选择指定的实例，单击实例名称，进入实例基本信息页面。 3. 在左侧导航栏单击“参数修改”，进入参数修改页面，在该页面修改参数。 如果需要修改的参数在管理该控制台无法修改，请提前评估风险后再联系客服进行修改。

序号	设置方法
方式三	修改会话级别的参数。 • 设置会话级别的参数 openGauss=# **SET** *paraname* **TO** *value;* 修改本次会话中的取值。退出会话后，设置将失效。

⚠ 注意

- 使用方式一和方式二设置参数时，若所设参数不属于当前环境，数据库会提示参数不在支持范围内的相关信息。
- 使用方式三设置参数时，若参数值为int整型，则会将整数前导零过滤掉，例如SET *paraname* TO *008192*与SET *paraname* TO *8192*效果相同。

18.3 GUC 参数说明

18.3.1 GUC 使用说明

数据库提供了许多运行参数，配置这些参数可以影响数据库系统的行为。在修改这些参数时请确保用户理解了这些参数对数据库的影响，否则可能会导致无法预料的结果。

注意事项

- 参数中如果取值范围为字符串，此字符串应遵循操作系统的路径和文件名命名规则。
- 取值范围最大值为INT_MAX的参数，此选项最大值跟所在的操作系统有关。其值为数据类型INT的最大值，值为2147483647。
- 取值范围最大值为DBL_MAX的参数，此选项最大值跟所在的操作系统有关。其值为数据类型FLOAT的最大值。

18.3.2 文件位置

数据库安装后会自动生成三个配置文件（postgresql.conf、pg_hba.conf和pg_ident.conf），并统一存放在数据目录（data）下。用户可以使用本节介绍的方法修改配置文件的名称和存放路径。

修改任意一个配置文件的存放目录时，postgresql.conf里的data_directory参数必须设置为实际数据目录（data）。

须知

考虑到配置文件修改一旦出错对数据库的影响很大，不建议安装后再修改本节的配置文件。

data_directory

参数说明：设置GaussDB的数据目录（data目录），仅sysadmin用户可以访问。此参数可以通过如下方式指定。

- 在安装GaussDB时指定。
- 该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串，长度大于0

默认值：安装时指定，如果在安装时不指定，则默认不初始化数据库。

config_file

参数说明：设置主服务器配置文件名称（postgresql.conf）。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串，长度大于0

默认值：postgresql.conf（实际安装可能带有绝对目录）

hba_file

参数说明：设置基于主机认证（HBA）的配置文件（pg_hba.conf）。此参数只能在配置文件postgresql.conf中指定，仅sysadmin用户可以访问。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

默认值：pg_hba.conf（实际安装可能带有绝对目录）

ident_file

参数说明：设置用于客户端认证的配置文件的名称（pg_ident.conf），仅sysadmin用户可以访问。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

默认值：pg_ident.conf（实际安装可能带有绝对目录）

external_pid_file

参数说明：声明可被服务器管理程序使用的额外PID文件，仅sysadmin用户可以访问。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

这个参数只能在数据库服务重新启动后生效。

取值范围：字符串

默认值：空

18.3.3 连接和认证

18.3.3.1 连接设置

介绍设置客户端和服务器连接方式相关的参数。

listen_addresses

参数说明：声明服务器侦听客户端的TCP/IP地址。

该参数指定GaussDB服务器使用哪些IP地址进行侦听，如IPV4。服务器主机上可能存在多个网卡，每个网卡可以绑定多个IP地址，该参数就是控制GaussDB绑定在哪个或者哪几个IP地址上。而客户端则可以通过该参数中指定的IP地址来连接GaussDB或者给GaussDB发送请求。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：

- 主机名或IP地址，多个值之间用英文逗号分隔。
- “*”或“0.0.0.0”表示侦听所有IP地址。配置侦听所有IP地址存在安全风险，不推荐用户使用。
- 置空则服务器不会侦听任何IP地址，这种情况下，只有UNIX域套接字可以用于连接数据库。

默认值：

集群安装好后，根据public_cloud.conf配置文件中不同实例的IP地址配置不同默认值。CN的默认参数值为：listen_addresses = 'localhost,mgr.net网卡对应的IP地址,data.net网卡对应的IP地址,virtual.net网卡对应的IP地址'；DN的默认参数值为：listen_addresses = 'data.net网卡对应的IP'。

说明

localhost表示只允许进行本地“回环”连接。

public_cloud.conf文件保存的网卡信息，包括：mgr.net（管理网卡）、data.net（数据网卡）、virtual.net（虚拟网卡）。

local_bind_address

参数说明：声明当前节点连接集群其他节点绑定的本地IP地址。

该参数属于POSTMASTER类型参数。

说明

该参数由安装时的配置文件指定，请勿轻易修改，否则修改后会影响数据库正常通信。

默认值：

集群安装好后，根据public_cloud.conf配置文件中不同实例的IP地址配置不同默认值。CN/DN的默认参数值为：local_bind_address = 'data.net网卡对应的IP地址'。

说明

public_cloud.conf文件保存的网卡信息，包括：mgr.net（管理网卡）、data.net（数据网卡）、virtual.net（虚拟网卡）。

port

参数说明：GaussDB服务侦听的TCP端口号。

说明

该参数由安装时的配置文件指定，请勿轻易修改，否则修改后会影响数据库正常通信。

取值范围：整型，1～65535

说明

- 设置端口号时，请设置一个未被占用的端口号。设置多个实例的端口号，不可冲突。
- 1~1023为操作系统保留端口号，请不要使用。
- 通过配置文件安装集群时，配置文件中的端口号需要注意通信矩阵预留端口。如：DN还需保留dataPortBase+1作为内部工具使用端口，保留dataPortBase+6作为流引擎（由于规格变更，当前版本已经不再支持本特性，请不要使用）消息队列通信端口等。故集群安装阶段，port最大值为：CN可设置65532，DN可设置65529，GTM可设置65534，同时需要保证端口号不冲突。

默认值：5432（实际值由安装时的配置文件指定）

max_connections

参数说明：允许和数据库连接的最大并发连接数。此参数会影响集群的并发能力。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型。最小值为10（要大于max_wal_senders），理论最大值为262143，实际最大值为动态值，计算公式为“262143 - job_queue_processes - autovacuum_max_workers - max_inner_tool_connections - AUXILIARY_BACKENDS - AV_LAUNCHER_PROCS - min(max(newValue/4,64),1024)”，**job_queue_processes**、**autovacuum_max_workers**和**max_inner_tool_connections**的值取决于对应GUC参数的设置，AUXILIARY_BACKENDS为预留辅助线程数固定为20，AV_LAUNCHER_PROCS为预留autovacuum的launcher线程数固定为2，min(max(newValue/4,64),1024)公式中newValue为新设置的值。

在不同实例的内存规格下，强制该参数取值范围如下：

表 18-3 独立部署模式下，不同实例的内存规格的参数取值范围

内存规格	CN参数取值范围	DN参数取值范围
< 32GB	[10, 100]	[10, 100]
[32GB, 64GB)	[10, 200]	[10, 200]
[64GB, 128GB)	[10, 1000]	[10, 2500]
[128GB, 256GB)	[10, 2000]	[10, 6000]

内存规格	CN参数取值范围	DN参数取值范围
[256GB, 480GB)	[10, 4000]	[10, 12000]
>= 480GB	[10, 8000]	[10, 24000]

表 18-4 金融版（标准型）模式下，不同实例的内存规格的参数取值范围

内存规格	CN参数取值范围	DN参数取值范围
< 64GB	[10, 100]	[10, 100]
[64GB, 128GB)	[10, 200]	[10, 1000]
[128GB, 256GB)	[10, 500]	[10, 2000]
[256GB, 480GB)	[10, 1000]	[10, 4000]
[480GB, 512GB)	[10, 2250]	[10, 9000]
[512GB, 576GB)	[10, 2500]	[10, 11000]
[576GB, 640GB)	[10, 3000]	[10, 12000]
[640GB, 768GB)	[10, 3500]	[10, 14000]
[768GB, 1024GB)	[10, 4000]	[10, 16000]
[1024GB, 1536GB)	[10, 6000]	[10, 21000]
>= 1536GB	[10, 10000]	[10, 33000]

表 18-5 企业版模式下，不同实例的内存规格的参数取值范围

内存规格	CN参数取值范围	DN参数取值范围
< 64GB	[10, 100]	[10, 100]
[64GB, 128GB)	[10, 200]	[10, 900]
[128GB, 256GB)	[10, 350]	[10, 1500]
[256GB, 480GB)	[10, 900]	[10, 3500]
[480GB, 512GB)	[10, 1800]	[10, 7000]
[512GB, 576GB)	[10, 2000]	[10, 7500]
[576GB, 640GB)	[10, 2000]	[10, 8500]
[640GB, 768GB)	[10, 2500]	[10, 10000]
[768GB, 1024GB)	[10, 3000]	[10, 11000]
[1024GB, 1536GB)	[10, 4000]	[10, 15000]

内存规格	CN参数取值范围	DN参数取值范围
>= 1536GB	[10, 7500]	[10, 24000]

表 18-6 金融版（数据计算型）模式下，不同实例的内存规格的参数取值范围

内存规格	CN参数取值范围	DN参数取值范围
< 256GB	[10, 100]	[10, 100]
[256GB, 512GB)	[10, 200]	[10, 1000]
[512GB, 576GB)	[10, 500]	[10, 4000]
[576GB, 768GB)	[10, 1000]	[10, 5000]
[768GB, 1024GB)	[10, 2500]	[10, 8000]
[1024GB, 1536GB)	[10, 4000]	[10, 12000]
>= 1536GB	[10, 5000]	[10, 18000]

默认值：

- 独立部署：

  CN：8000（60核CPU/480G内存）；4000（32核CPU/256G内存）；2000（16核CPU/128G内存）；1000（8核CPU/64G内存）；100（4核CPU/32G内存，4核CPU/16G内存）

  DN：24000（60核CPU/480G内存）；12000（32核CPU/256G内存）；6000（16核CPU/128G内存）；2500（8核CPU/64G内存）；100（4核CPU/32G内存，4核CPU/16G内存）

- 金融版（标准型）：

  CN：6000（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存）；4000（96核CPU/768G内存）；3500（80核CPU/640G内存）；3000（72核CPU/576G内存）；2500（64核CPU/512G内存）；2250（60核CPU/480G内存）；1000（32核CPU/256G内存）；500（16核CPU/128G内存）；200（8核CPU/64G内存）

  DN：21000（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存）；16000（96核CPU/768G内存）；14000（80核CPU/640G内存）；12000（72核CPU/576G内存）；11000（64核CPU/512G内存）；9000（60核CPU/480G内存）；4000（32核CPU/256G内存）；2000（16核CPU/128G内存）；1000（8核CPU/64G内存）

- 企业版：

  CN：4000（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存）；3000（96核CPU/768G内存）；2500（80核CPU/640G内存）；2000（80核CPU/512G内存，72核CPU/576G内存，64核CPU/512G内存）；1800（60核CPU/480G内存）；900（32核CPU/256G内存）；350（16核CPU/128G内存）；200（8核CPU/64G内存）

  DN：15000（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存）；11000（96核CPU/768G内存）；10000（80核CPU/640G内存）；8500

（72核CPU/576G内存）；7500（80核CPU/512G内存，64核CPU/512G内存）；7000（60核CPU/480G内存）；3500（32核CPU/256G内存）；1500（16核CPU/128G内存）；900（8核CPU/64G内存）

- 金融版（数据计算型）：

  CN：2500（96核CPU/768G内存）；1000（72核CPU/576G内存）；500（64核CPU/512G内存）；200（32核CPU/256G内存）

  DN：8000（96核CPU/768G内存）；5000（72核CPU/576G内存）；4000（64核CPU/512G内存）；1000（32核CPU/256G内存）

配置不当时影响：

若配置max_connections过大，超过计算公式所描述的最大动态值，会出现节点拉起失败问题，报错提示“invalid value for parameter "max_connections"”；或在拉起时申请内存失败，报错提示“Cannot allocate memory”；

若未按照对外出口规格配置仅调大max_connections参数值，未同比例调整内存参数。业务压力大时，容易出现内存不足，报错提示“memory is temporarily unavailable”；

混合部署场景下，若CN的max_connections参数值大于（DN的max_connections/CN个数）时，客户端总压力超过DN端max_connections值时，可能会出现CN节点连接DN失败情况，报错提示“pooler... Too many clients already”。

说明

- 对于管理员用户的连接数限制会略超过max_connections设置，目的是为了让管理员在连接被普通用户占满后仍可以连接上数据库，再超过一定范围（sysadmin_reserved_connections参数）后才会报错。即管理员用户的最大连接数等于max_connections + sysadmin_reserved_connections。
- 对于普通用户来说，由于内部作业也会使用一些链接，因此会略小于max_connections，具体值取决于内部链接个数。
- 开启线程池后，stream线程数的上限为max_connections设置值。若stream线程数量达到上限时，将产生“Exceed stream thread pool limitation...”报错，此时可以通过调整max_connections参数值将stream线程数上限调高。由于该参数属于POSTMASTER类型，故设置时可以综合业务情况进行合理预估：stream线程数总数 = 业务并发数 * 每一个并发执行的语句所消耗的stream线程数量（可通过执行计划查看）。

max_inner_tool_connections

参数说明：允许和数据库连接的工具的最大并发连接数。此参数会影响GaussDB的工具连接并发能力。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，最小值为1，最大值为MIN(262143, max_connections)，max_connections的计算方法见上文。

默认值：50。如果该默认值超过内核支持的最大值（在执行gs_initdb的时候判断），系统会提示错误。

设置建议：

数据库主节点中此参数建议保持默认值。

sysadmin_reserved_connections

参数说明：为管理员用户预留的最少连接数，不建议设置过大。该参数和max_connections参数配合使用，管理员用户的最大连接数等于max_connections + sysadmin_reserved_connections。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，最小值为0，最大值为MIN(262143, max_connections)，max_connections的计算方法见上文。

默认值：3

注意：当启用线程池功能时，若线程池占满将形成处理瓶颈，导致管理员预留连接无法正常建立；作为逃生手段，此时可使用gsql通过主端口+1端口号连入，清理无用会话，即可正常连入。

unix_socket_directory

参数说明：设置GaussDB服务器侦听客户端连接的UNIX域套接字目录，仅sysadmin用户可以访问。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

该参数的长度限制于操作系统的长度，Linux系统下，套接字路径名（套接字目录与套接字文件名拼接而成）长度不得超过107bytes，目录最长不得超过92bytes。超过该限制将会导致UNIX-domain socket path "xxx" is too long的问题，影响进程正常拉起，若误设置出错，可以通过检索cm_agent路径下system_call日志定界。

取值范围：字符串

默认值：空字符串（实际值由安装时配置文件中tmpMppdbPath指定）

unix_socket_group

参数说明：设置UNIX域套接字的所属组（套接字的所属用户总是启动服务器的用户）。可以与选项**unix_socket_permissions**一起用于对套接字进行访问控制。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串，其中空字符串表示当前用户的缺省组。

默认值：空字符串

unix_socket_permissions

参数说明：设置UNIX域套接字的访问权限。

UNIX域套接字使用普通的UNIX文件系统权限集。这个参数的值应该是数值的格式（chmod和umask命令可接受的格式）。如果使用自定义的八进制格式，数字必须以0开头。

建议设置为0770（只有当前连接数据库的用户和同组的人可以访问）或者0700（只有当前连接数据库的用户自己可以访问，同组或者其他人都没有权限）。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：0000-0777

默认值：0700

说明

在Linux中，文档具有十个属性，其中第一个属性为文档类型，后面九个为权限属性，分别为Owner，Group及Others这三个组别的read、write、execute属性。

文档的权限属性分别简写为r，w，x，这九个属性三个为一组，也可以使用数字来表示文档的权限，对照表如下：

r：4

w: 2

x：1

-：0

同一组（owner/group/others）的三个属性是累加的。

例如，-rwxrwx---表示这个文档的权限为：

owner = rwx = 4+2+1 = 7

group = rwx = 4+2+1 = 7

others = --- = 0+0+0 = 0

所以其权限为0770。

application_name

参数说明：当前连接请求当中，所使用的客户端名称。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

在备机请求主机进行日志复制时，如果该参数非空串，那么会被用来作为备机在主机上的流复制槽名字。此时，如果该参数长度超过61个字节，那么流复制槽名字只会截取使用前61个字节的字符。

取值范围：字符串，实际查询结果取决于查询所用的客户端或用户设置。

默认值：空字符串

connection_info

参数说明：连接数据库的驱动类型、驱动版本号、当前驱动的部署路径和进程属主用户。

该参数属于USERSET类型参数，属于运维类参数，不建议用户设置。

取值范围：字符串。

默认值：空字符串。

说明

- 空字符串，表示当前连接数据库的驱动不支持自动设置connection_info参数或应用程序未设置。
- 驱动连接数据库的时候自行拼接的connection_info参数格式如下：

  ```
  {"driver_name":"ODBC","driver_version": "(GaussDB VxxxRxxxCxx build 290d125f) compiled at 2020-05-08 02:59:43 commit 2143 last mr 131 release","driver_path":"/usr/local/lib/psqlodbcw.so","os_user":"omm"}
  ```

  默认显示driver_name和driver_version，driver_path和os_user的显示由用户控制（参见《开发指南》中“应用程序开发教程 > 基于JDBC开发 > 连接数据库”章节和“应用程序开发教程 > 基于ODBC开发 > Linux下配置数据源”章节）。

backend_version

参数说明：用于CN和CN，CN和DN建立连接时同步连接的版本号，该参数涉及版本号，用户不可以随意设置。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0-100000。

check_disconnect_query

参数说明：当客户端异常断连（如JDBC触发socketTimeout、libpq触发rwtimeout且关闭连接、运行业务过程中客户端进程终止等）后，该参数控制GaussDB服务端语句是否终止执行。

参数类型：布尔型

参数单位：无

取值范围：

- on：表示当客户端异常断连后，GaussDB服务端终止运行对应的语句。
- off：表示当客户端异常断连后，GaussDB服务端不会终止运行对应的语句。

默认值：on

设置方式：该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

设置建议：推荐使用默认值。

注意

若设置此参数后执行升级，在升级到其他版本前需检查目标版本是否支持该参数。如果不支持，则在升级前需从配置文件中删除该参数。

18.3.3.2 安全和认证（postgresql.conf）

介绍设置客户端和服务器的安全认证方式的相关参数。

authentication_timeout

参数说明：完成客户端认证的最长时间。如果一个客户端没有在这段时间里完成与服务器端的认证，则服务器自动中断与客户端的连接，这样就避免了出问题的客户端无限制地占用连接数。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，最小值为1，最大值为600，最小单位为s。

默认值：1min

auth_iteration_count

参数说明：认证加密信息生成过程中使用的迭代次数。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，2048-134217728。

默认值：10000

须知

迭代次数设置过小会降低口令存储的安全性，设置过大会导致认证、用户创建等涉及口令加密的场景性能劣化，请根据实际硬件条件合理设置迭代次数，推荐采用默认迭代次数。

session_authorization

参数说明：当前会话的用户标识。

该参数属于USERSET类型参数，只能通过SET SESSION AUTHORIZATION语法设置，不支持直接设置。

取值范围：字符串。

默认值：NULL

session_timeout

参数说明：表明与服务器建立连接后，不进行任何操作的最长时间。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0-86400，最小单位为s，0表示关闭超时设置。

默认值：1800

须知

GaussDB gsql客户端中有自动重连机制，所以针对初始化用户本地连接，超时后gsql表现的现象为断开后重连。

ssl

参数说明：启用SSL连接。请在使用这个选项之前阅读**通过gsql连接实例**章节。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示启用SSL连接。
- off表示不启用SSL连接。

须知

GaussDB目前支持SSL的场景为客户端连接CN场景，该参数目前建议只在CN中开启，DN默认值为off。开启此参数需要同时确保**ssl_cert_file**、**ssl_key_file**和**ssl_ca_file**参数配置正确，不正确的配置可能会导致集群无法正常启动。

默认值：on（CN实例）；off（DN实例）

comm_ssl

参数说明：启用主DN之间SSL连接。请在使用这个选项之前阅读**通过gsql连接实例**章节。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示主DN之间启用SSL连接。
- off表示主DN之间不启用SSL连接。

须知

- 该参数目前建议只在DN中开启，CN默认值为off。
- 开启此参数需要同时确保**ssl_cert_file**、**ssl_key_file**和**ssl_ca_file**参数配置正确，不正确的配置可能会导致集群无法正常启动。

默认值：off

require_ssl

参数说明：设置服务器端是否强制要求SSL连接，该参数只有当参数**ssl**为on时才有效。请在使用这个选项之前阅读**通过gsql连接实例**章节。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示服务器端强制要求SSL连接。
- off表示服务器端对是否通过SSL连接不作强制要求。

须知

GaussDB目前支持SSL的场景为客户端连接CN场景，该参数目前建议只在CN中开启。

默认值：off

ssl_ciphers

参数说明：指定SSL支持的加密算法列表，仅sysadmin用户可以访问。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串，如果指定多个加密算法，加密算法之间需要以分号分隔。

须知

ssl_ciphers设置错误会导致集群不能正常启动。

默认值：ALL

ssl_renegotiation_limit

参数说明：指定在会话密钥重新协商之前，通过SSL加密通道可以传输的流量。这个重新协商流量限制机制可以减少攻击者针对大量数据使用密码分析法破解密钥的几率，但是也带来较大的性能损失。流量是指发送和接受的流量总和。使用SSL重协商机制可能引入其他风险，因此已禁用SSL重协商机制，为保持版本兼容保留此参数，修改参数配置不再起作用。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，最小值为0，最大值为2147483647。单位为KB。其中0表示禁用重新协商机制。

默认值：0

ssl_cert_file

参数说明：指定包含SSL服务器证书的文件的名称，其相对路径是相对于数据目录的。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

默认值：server.crt

ssl_key_file

参数说明：指定包含SSL私钥的文件名称，其相对路径是相对于数据目录的。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

默认值：server.key

ssl_ca_file

参数说明：指定包含CA信息的根证书名称，其相对路径是相对于数据目录的。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串，其中空字符串表示没有CA文件被加载，不进行客户端证书验证。

默认值：cacert.pem

ssl_crl_file

参数说明：证书吊销列表，如果客户端证书在该列表中，则当前客户端证书被视为无效证书，其相对路径是相对于数据目录的。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串，空字符串表示没有吊销列表。

默认值：空字符串

ssl_cert_notify_time

参数说明：SSL服务器证书到期前提醒的天数。建立连接初始化ssl证书时，若当前时间距离证书到期时间小于设定值，则在日志中打印过期提醒。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，最小值为7，最大值为180，单位为天。

默认值：90

krb_server_keyfile

参数说明：指定Kerberos服务主配置文件的位置，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

默认值：空字符串

krb_srvname

参数说明：设置Kerberos服务名。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

默认值：postgres

krb_caseins_users

参数说明：设置Kerberos用户名是否大小写敏感。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示大小写不敏感
- off表示大小写敏感

默认值：off

modify_initial_password

参数说明：当GaussDB安装成功后，数据库中仅存在一个初始用户（UID为10的用户）。客户通过该账户初次登录数据库进行操作时，该参数决定是否要对该初始账户的密码进行修改。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

如果安装过程中未指定初始用户密码，则安装后初始用户密码默认为空，执行其他操作前需要先通过gsql客户端设置初始用户的密码。此参数功能不再生效，保留此参数仅为兼容升级场景。

取值范围：布尔型

- on表示集群安装成功后初始用户首次登录操作前需要修改初始密码。
- off表示集群安装成功后初始用户无需修改初始密码即可进行操作。

默认值：off

password_policy

参数说明：在使用CREATE ROLE/USER或者ALTER ROLE/USER命令创建或者修改GaussDB账户时，该参数决定是否进行密码复杂度检查。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

从安全性考虑，请勿关闭密码复杂度策略。

取值范围：0、1

- 0表示不采用密码复杂度校验策略。
- 1表示采用默认密码复杂度校验策略。

默认值：1

password_reuse_time

参数说明：在使用ALTER USER或者ALTER ROLE修改用户密码时，该参数指定是否对新密码进行可重用天数检查。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

修改密码时会检查配置参数**password_reuse_time**和**password_reuse_max**。

- 当**password_reuse_time**和**password_reuse_max**都为正数时，只要满足其中任一个，即可认为密码可以重用。
- 当**password_reuse_time**为0时，表示不限制密码重用天数，仅限制密码重用次数。
- 当**password_reuse_max**为0时，表示不限制密码重用次数，仅限制密码重用天数。
- 当**password_reuse_time**和**password_reuse_max**都为0时，表示不对密码重用进行限制。

取值范围：浮点型，最小值为0，最大值为3650，单位为天。

- 0表示不检查密码可重用的天数。
- 正数表示新密码不能为该值指定的天数内使用过的密码。

password_reuse_max

参数说明：在使用ALTER USER或者ALTER ROLE修改用户密码时，该参数指定是否对新密码进行可重用次数检查，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

修改密码时会检查配置参数**password_reuse_time**和**password_reuse_max**。

- 当**password_reuse_time**和**password_reuse_max**都为正数时，只要满足其中任一个，即可认为密码可以重用。
- 当**password_reuse_time**为0时，表示不限制密码重用天数，仅限制密码重用次数。
- 当**password_reuse_max**为0时，表示不限制密码重用次数，仅限制密码重用天数。
- 当**password_reuse_time**和**password_reuse_max**都为0时，表示不对密码重用进行限制。

取值范围：整型，最小值为0，最大值为1000。

- 0表示不检查密码可重用次数。
- 正整数表示新密码不能为该值指定的次数内使用过的密码。

默认值：0

password_lock_time

参数说明：该参数指定账户被锁定后自动解锁的时间。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

password_lock_time和**failed_login_attempts**必须都为正数时锁定和解锁功能才能生效。

取值范围：浮点型，最小值为0，最大值为365，单位为天。整数部分表示天数，小数部分可以换算成时、分、秒，如：password_lock_time=1.5，表示1天零12小时。

- 0表示密码验证失败时，自动锁定功能不生效。
- 正数表示账户被锁定后，当锁定时间超过password_lock_time设定的值时，账户将会被自行解锁。

默认值：1

failed_login_attempts

参数说明：在任意时候，如果输入密码错误的次数达到failed_login_attempts则当前账户被锁定，password_lock_time秒后被自动解锁，仅sysadmin用户可以访问。例如，登录时输入密码失败，ALTER USER时修改密码失败等。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

failed_login_attempts和**password_lock_time**必须都为正数时锁定和解锁功能才能生效。

取值范围：整型，最小值为0，最大值为1000。

- 0表示自动锁定功能不生效。
- 正整数表示当错误密码次数达到failed_login_attempts设定的值时，当前账户将被锁定。

默认值：10

password_encryption_type

参数说明：该字段决定采用何种加密方式对用户密码进行加密存储。修改此参数的配置不会自动触发已有用户密码加密方式的修改，只会影响新创建用户或修改用户密码操作。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：0、1、2、3

- 0表示采用md5方式对密码加密。
- 1表示采用sha256和md5两种方式分别对密码加密。
- 2表示采用sha256方式对密码加密。
- 3表示采用sm3方式对密码加密。

须知

MD5加密算法安全性低，存在安全风险，不建议使用。

默认值：2

password_min_length

参数说明：该字段决定账户密码的最小长度，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，6~999个字符。

默认值：8

password_max_length

参数说明：该字段决定账户密码的最大长度，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，6~999个字符。

默认值：32

password_min_uppercase

参数说明：该字段决定账户密码中至少需要包含大写字母个数，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~999

- 0表示没有限制。
- 1~999表示创建账户所指定的密码中至少需要包含大写字母个数。

默认值：0

password_min_lowercase

参数说明：该字段决定账户密码中至少需要包含小写字母的个数，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~999

- 0表示没有限制。
- 1~999表示创建账户所指定的密码中至少需要包含小写字母个数。

默认值：0

password_min_digital

参数说明：该字段决定账户密码中至少需要包含数字的个数，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~999

- 0表示没有限制。
- 1~999表示创建账户所指定的密码中至少需要包含数字个数。

默认值：0

password_min_special

参数说明：该字段决定账户密码中至少需要包含特殊字符个数，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~999

- 0表示没有限制。
- 1~999表示创建账户所指定的密码中至少需要包含特殊字符个数。

默认值：0

password_effect_time

参数说明：该字段决定账户密码的有效时间。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：浮点型，最小值为0，最大值为999，单位为天。

- 0表示不开启有效期限制功能。
- 1~999表示创建账户所指定的密码有效期，临近或超过有效期系统会提示用户修改密码。

默认值：0

password_notify_time

参数说明：该字段决定账户密码到期前提醒的天数。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，最小值为0，最大值为999，单位为天。

- 0表示不开启提醒功能。
- 1~999表示账户密码到期前提醒的天数。

默认值：7

18.3.3.3 通信库参数

本节介绍通信库相关的参数设置及取值范围等内容。

tcp_keepalives_idle

参数说明：在支持TCP_KEEPIDLE套接字选项的系统上，设置发送活跃信号的间隔秒数。不设置发送保持活跃信号，连接就会处于闲置状态。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

- 如果操作系统不支持TCP_KEEPIDLE选项 ，这个参数的值必须为0。
- 在通过UNIX域套接字进行的连接的操作系统上，这个参数将被忽略。

取值范围：0-3600，单位为s。

默认值：1min

tcp_keepalives_interval

参数说明：在支持TCP_KEEPINTVL套接字选项的操作系统上，以秒数声明在重新传输之间等待响应的时间。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：0-180，单位为s。

默认值：30

须知

- 如果操作系统不支持TCP_KEEPINTVL选项，这个参数的值必须为0。
- 在通过UNIX域套接字进行的连接的操作系统上，这个参数将被忽略。

tcp_keepalives_count

参数说明：在支持TCP_KEEPCNT套接字选项的操作系统上，设置GaussDB服务端在断开与客户端连接之前可以等待的保持活跃信号个数。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

- 如果操作系统不支持TCP_KEEPCNT选项，这个参数的值必须为0。
- 在通过UNIX域套接字进行连接的操作系统上，这个参数将被忽略。

取值范围：0-100，其中0表示GaussDB未收到客户端反馈的保持活跃信号则立即断开连接。

默认值：20

tcp_user_timeout

参数说明：在支持TCP_USER_TIMEOUT套接字选项的操作系统上，设置GaussDB在发送数据时，指定传输的数据在TCP连接被强制关闭之前可以保持未确认状态的最大时长。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

- 如果操作系统不支持TCP_USER_TIMEOUT选项，这个参数的值将不生效，默认为0。
- 在通过UNIX域套接字进行连接的操作系统上，这个参数将被忽略。

取值范围：0-3600000，单位为ms。其中0表示跟随操作系统设置。

默认值：0

注意，不同操作系统内核下，这个参数生效结果将不同：

- aarch64 EulerOS（Linux内核版本：4.19），超时时间即为该参数设置值。
- x86 Euler2.5（Linux内核版本：3.10），超时时间不是该参数设置值，而是不同区间的最大值，即超时时间取值为：tcp_user_timeout设置值所处“Linux TCP重传总耗时”区间的上限最大值。例如：tcp_user_timeout=40000时，重传总耗时为51秒。

表 18-7 x86 Euler2.5（Linux 内核版本：3.10）tcp_user_timeout 参数取值示意

Linux TCP重传次数	Linux TCP重传总耗时区间（秒）	tcp_user_timeout设置举例（毫秒）	实际Linux TCP重传总耗时（秒）
1	(0.2,0.6]	400	0.6
2	(0.6,1.4]	1000	1.4
3	(1.4,3]	2000	3
4	(3,6.2]	4000	6.2
5	(6.2,12.6]	10000	12.6
6	(12.6,25.4]	20000	25.4
7	(25.4,51]	40000	51
8	(51,102.2]	80000	102.2
9	(102.2,204.6]	150000	204.6
10	(204.6,324.6]	260000	324.6
11	(324.6,444.6]	400000	444.6

注：TCP每次重传耗时随重传次数指数增加，当TCP一次重传到达120秒后，后续每次重传都将耗时120秒不再变化。

comm_tcp_mode

参数说明：通信库使用TCP协议建立数据通道的切换开关，重启集群生效。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型，CN设置为on表示使用TCP模式连接DN，DN设置为on表示DN间使用TCP代理通信。

默认值：on

comm_control_port

参数说明：TCP代理通信库使用的TCP协议侦听端口。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

集群部署时会自动分配此端口号，请不要轻易修改此参数，如端口号配置不正确会导致数据库通信失败。

取值范围：整型，最小值为0，最大值为65535。

默认值：25111（实际值为guc参数port值+3，取决于用户配置）

comm_max_datanode

参数说明：TCP代理通信库支持的最大DN数。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，最小值为1，最大值为8192。

默认值：单个节点支持的最大主DN数量为默认值。

推荐值：256

comm_max_stream

参数说明：TCP代理通信库支持的最大并发数据流数。该参数必须大于业务并发数*每并发平均stream算子数*query_dop的平方。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，最小值为1，最大值为60000。

默认值：1024

须知

- 不建议该参数值设置过大，因为comm_max_stream会占用内存（占用内存=256byte*comm_max_stream*comm_max_datanode），若并发数据流过大，查询较为复杂及smp过大都会导致内存不足。
- 如果comm_max_stream参数值较小，进程内存充足，可以适当将comm_max_stream值调大。

comm_max_receiver

参数说明：TCP代理通信库内部接收线程数量。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，最小值为1，最大值为50。

默认值：4

comm_quota_size

参数说明：TCP代理通信库最大可连续发送包总大小。使用1GE网卡时，建议取较小值，推荐设置为20KB~40KB。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，最小值为0，最大值为2048000，默认单位为KB。

默认值：1MB

comm_usable_memory

参数说明：单个DN内TCP代理通信库缓存最大可使用内存。

须知

此参数需根据环境内存及部署方式具体配置，过大会造成OOM，过小会降低TCP代理通信库或SCTP通信库性能。（当前版本SCTP不再使用）

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，最小值为100*1024，最大值为1073741823，默认单位为KB。

默认值：4000MB

comm_memory_pool

参数说明：单个DN内TCP代理通信库可使用内存池资源的容量大小。

须知

此参数需根据实际业务情况做调整，若通信库使用内存小，可设置该参数数值较小，反之设置数值较大。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，最小值为100*1024，最大值为1073741823，默认单位为KB。

默认值：2000MB

comm_memory_pool_percent

参数说明：单个DN内TCP代理通信库可使用内存池资源的百分比，用于自适应负载预留通信库通信消耗的内存大小。

须知

此参数需根据实际业务情况做调整，若通信库使用内存小，可设置该参数数值较小，反之设置数值较大。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，最小值为0，最大值为100。

默认值：0

comm_client_bind

参数说明：通信库客户端发起连接时是否使用bind绑定指定IP。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示绑定指定IP。
- off表示不绑定指定IP。

须知

如果集群某一节点存在多个IP处于同一通信网段时，需设置为on。此时将绑定本地listen_addresses指定的IP发起通信，随机端口号不能重复使用，集群并发数量会受到可用随机端口号数量的限制。

默认值：off

comm_no_delay

参数说明：是否使用通信库连接的NO_DELAY属性。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

须知

如果集群出现因每秒接收数据包过多导致的丢包时，需设置为off，以便小包合并成大包发送，减少数据包总数。

默认值：off

comm_debug_mode

参数说明：TCP代理通信库debug模式开关，该参数设置是否打印通信层详细日志。

须知

设置为on时，打印日志量较大，会增加额外的overhead并降低数据库性能，仅在调试时打开。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示打印通信库详细debug日志。
- off表示不打印通信库详细debug日志。

默认值：off

comm_ackchk_time

参数说明：无数据包接收情况下，该参数设置通信库服务端主动ACK触发时长。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，最小值为0，最大值为20000，单位为毫秒。取值为0表示关闭此功能。

默认值：2000（2s）

comm_timer_mode

参数说明：TCP代理通信库timer模式开关，该参数设置是否打印通信层各阶段时间桩。

须知

设置为on时，打印日志量较大，会增加额外的overhead并降低数据库性能，仅在调试时打开。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示打印通信库详细时间桩日志。

- off表示不打印通信库详细时间桩日志。

默认值：off

comm_stat_mode

参数说明：TCP代理通信库stat模式开关，该参数设置是否打印通信层的统计信息。

须知

设置为on时，打印日志量较大，会增加额外的overhead并降低数据库性能，仅在调试时打开。

该参数属于USERSET类型参数，请参考表18-1中对应设置方法进行设置。

取值范围：布尔型

- on表示打印通信库统计信息日志。
- off表示不打印通信库统计信息日志。

默认值：off

enable_stateless_pooler_reuse

参数说明：pooler连接池复用切换开关，开启后可对已有的空闲TCP连接进行复用，重启集群生效。

该参数属于POSTMASTER类型参数，请参考表18-1中对应设置方法进行设置。

取值范围：布尔型

- 设置为on/true表示使用pooler复用模式。
- 设置为off/false表示关闭pooler复用模式。

须知

CN和DN需要同步设置。如果CN设置enable_stateless_pooler_reuse为off，DN设置enable_stateless_pooler_reuse为on会导致集群不能正常通信，因此必须对该参数做CN和DN全局相同的配置，重启集群生效。

默认值：on

comm_cn_dn_logic_conn

参数说明：CN和DN间逻辑连接特性开关，重启集群生效。

该参数属于POSTMASTER类型参数，请参考表18-1中对应设置方法进行设置。

取值范围：布尔型

- 设置为on/true表示CN和DN之间连接为逻辑连接，使用libcomm组件。
- 设置为off/false表示CN和DN之间连接为物理连接，使用libpq组件。

须知

不再提供CN和DN之间的逻辑连接支持，为了保持兼容，提供此参数的接口，但此参数会在设置过程中强制改为off。

默认值：off

COMM_IPC

参数说明：通信性能问题定位开关，该参数设置是否打印通信各个节点的报文收发情况。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。**取值范围**：布尔型

- 设置为on/true表示打开报文收发统计日志。
- 设置为off/false表示关闭报文收发统计日志

须知

set logging_module='on(COMM_IPC)'; --打开

set logging_module='off(COMM_IPC)'; --关闭

show logging_module; --查看设置结果。

设置为on时，打印日志量较大，会增加额外的overhead并降低数据库性能，仅在调试时打开，打开后及时关闭。

默认值：off

COMM_PARAM

参数说明：通信性能问题定位开关，该参数设置是否打印节点通信过程中session参数设置情况。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- 设置为on/true表示打开连接的session参数设置日志。
- 设置为off/false表示关闭连接的session参数设置日志。

须知

set logging_module='on(COMM_PARAM)'; --打开

set logging_module='off(COMM_PARAM)'; --关闭

show logging_module; --查看设置结果

设置为on时，打印日志量较大，会增加额外的overhead并降低数据库性能，仅在调试时打开，打开后及时关闭。

默认值：off

18.3.4 资源消耗

18.3.4.1 内存

介绍与内存相关的参数设置。

须知

这些参数只能在数据库服务重新启动后生效，local_syscache_threshold除外。

memorypool_enable

参数说明：设置是否允许使用内存池。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许使用内存池。
- off表示不允许使用内存池。

默认值：off

memorypool_size

参数说明：设置内存池大小。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，128*1024～1073741823，单位为KB。

默认值：512MB

enable_memory_limit

参数说明：启用逻辑内存管理模块。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示启用逻辑内存管理模块。
- off表示不启用逻辑内存管理模块。

默认值：on

⚠ 注意

- 若max_process_memory-shared_buffers-cstore_buffers-元数据少于2G，GaussDB强制把enable_memory_limit设置为off。其中元数据是GaussDB内部使用的内存，和部分并发参数，如max_connections，thread_pool_attr，max_prepared_transactions等参数相关。
- 当该值为off时，不对数据库使用的内存做限制，在大并发或者复杂查询时，使用内存过多，可能导致操作系统OOM问题。

max_process_memory

参数说明：设置一个数据库节点可用的最大物理内存。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，2097152~2147483647，单位为KB。

默认值：

独立部署：360GB（60核CPU/480G内存）；192GB（32核CPU/256G内存）；96GB（16核CPU/128G内存）；40GB（8核CPU/64G内存）；20GB（4核CPU/32G内存）；10GB（4核CPU/16G内存）

金融版（标准型）：

CN：200GB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存）；160GB（96核CPU/768G内存）；130GB（80核CPU/640G内存）；120GB（72核CPU/576G内存）；100GB（64核CPU/512G内存，60核CPU/480G内存）；50GB（32核CPU/256G内存）；20GB（16核CPU/128G内存）；10GB（8核CPU/64G内存）

DN：350GB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存）；260GB（96核CPU/768G内存）；220GB（80核CPU/640G内存）；200GB（72核CPU/576G内存）；180GB（64核CPU/512G内存）；160GB（60核CPU/480G内存）；80GB（32核CPU/256G内存）；40GB（16核CPU/128G内存）；20GB（8核CPU/64G内存）

企业版：

CN：150GB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存）；110GB（96核CPU/768G内存）；90GB（80核CPU/640G内存）；80GB（72核CPU/576G内存）；75GB（80核CPU/512G内存，64核CPU/512G内存）；70GB（60核CPU/480G内存）；35GB（32核CPU/256G内存）；15GB（16核CPU/128G内存）；9GB（8核CPU/64G内存）

DN：250GB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存）；190GB（96核CPU/768G内存）；160GB（80核CPU/640G内存）；140GB（72核CPU/576G内存）；125GB（80核CPU/512G内存，64核CPU/512G内存）；120GB（60核CPU/480G内存）；60GB（32核CPU/256G内存）；25GB（16核CPU/128G内存）；15GB（8核CPU/64G内存）

金融版（数据计算型）：

CN：100GB（96核CPU/768G内存）；60GB（72核CPU/576G内存，64核CPU/512G内存）；20GB（32核CPU/256G内存）

DN：150GB（96核CPU/768G内存）；110GB（72核CPU/576G内存）；100GB（64核CPU/512G内存）；40GB（32核CPU/256G内存）

设置建议：

DN上该数值需要根据系统物理内存及单节点部署主DN个数决定的。计算公式如下：（物理内存大小 – vm.min_free_kbytes）* 0.7 / (n+主DN个数)。该参数目的是尽可能保证系统的可靠性，不会因数据库内存膨胀导致节点OOM。这个公式中提到vm.min_free_kbytes，其含义是预留操作系统内存供内核使用，通常用作操作系统内核中通信收发内存分配，至少为5%内存。即，max_process_memory=物理内存*0.665/(n+主DN个数)，其中，当集群规模小于256时，n=1；当集群规模大于256且小于512时，n=2；当集群规模超过512时，n=3。

CN上该数值内存可设置与DN数值一样。

RAM：集群规划时分配给集群的最大使用内存，实际为服务器的物理内存。

⚠ 注意

当该值设置不合理，即大于服务器物理内存，可能导致操作系统OOM问题。

local_syscache_threshold

参数说明：系统表cache在单个session缓存的大小。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

- 如果enable_global_plancache已打开，为保证GPC生效，local_syscache_threshold设置值小于16MB时不会生效，最小为16MB。
- 如果enable_global_syscache和enable_thread_pool打开，该参数描述的是当前线程和绑定到当前线程上的session缓存的总大小。

取值范围：整型，1*1024～512*1024，单位为KB。

默认值：

- 独立部署：16MB
- 金融版（标准型）：

  32MB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存）；16MB（72核CPU/576G内存，64核CPU/512G内存，60核CPU/480G内存，32核CPU/256G内存，16核CPU/128G内存，8核CPU/64G内存）
- 企业版：

  32MB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，80核CPU/512G内存，72核CPU/576G内存，64核CPU/512G内存）；16MB（60核CPU/480G内存，32核CPU/256G内存，16核CPU/128G内存，8核CPU/64G内存）
- 金融版（数据计算型）：16MB

enable_memory_context_control

参数说明：启用检查内存上下文是否超过给定限制的功能。仅适用于DEBUG版本。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示启用最大内存上下文限制检查功能。
- off表示关闭最大内存上下文限制检查功能。

默认值：off

uncontrolled_memory_context

参数说明：启用检查内存上下文是否超过给定限制的功能时，设置不受此功能约束。仅适用于DEBUG版本。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

查询时会在参数值的最前面添加标题含义字符串“MemoryContext white list:”"。

取值范围：字符串

默认值：空

shared_buffers

参数说明：设置GaussDB使用的共享内存大小。增加此参数的值会使GaussDB比系统默认设置需要更多的System V共享内存。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，16 ~ 1073741823，单位为8KB。

改变BLCKSZ的值会改变最小值。

默认值：

独立部署：

CN：4GB（60核CPU/480G内存）；2GB（32核CPU/256G内存，16核CPU/128G内存）；1GB（8核CPU/64G内存）；512MB（4核CPU/32G内存）；256MB（4核CPU/16G内存）

DN：140GB（60核CPU/480G内存）；76GB（32核CPU/256G内存）；40GB（16核CPU/128G内存）；16GB（8核CPU/64G内存）；8GB（4核CPU/32G内存）；4GB（4核CPU/16G内存）

金融版（标准型）：

CN：2GB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，72核CPU/576G内存，64核CPU/512G内存，60核CPU/480G内存）；1GB（32核CPU/256G内存，16核CPU/128G内存）；512MB（8核CPU/64G内存）

DN：140GB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存）；100GB（96核CPU/768G内存）；80GB（80核CPU/640G内存，72核CPU/576G内存）；70GB（64核CPU/512G内存）；60GB（60核CPU/480G内存）；30GB（32核CPU/256G内存）；16GB（16核CPU/128G内存）；8GB（8核CPU/64G内存）

企业版：

CN：2GB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，80核CPU/512G内存，72核CPU/576G内

存，64核CPU/512G内存，60核CPU/480G内存）；1GB（32核CPU/256G内存，16核CPU/128G内存）；512MB（8核CPU/64G内存）

DN：100GB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存）；76GB（96核CPU/768G内存）；64GB（80核CPU/640G内存）；56GB（72核CPU/576G内存）；50GB（80核CPU/512G内存，64核CPU/512G内存）；48GB（60核CPU/480G内存）；24GB（32核CPU/256G内存）；10GB（16核CPU/128G内存）；6GB（8核CPU/64G内存）

金融版（数据计算型）：

CN：2GB（96核CPU/768G内存）；1GB（72核CPU/576G内存，64核CPU/512G内存）；512MB（32核CPU/256G内存）

DN：50GB（96核CPU/768G内存）；40GB（72核CPU/576G内存）；30GB（64核CPU/512G内存）；10GB（32核CPU/256G内存）

设置建议：

1. 由于GaussDB大部分查询下推，建议DN中此参数设置比CN大。
2. 建议设置shared_buffers值为内存的40%以内。
3. 如果设置较大的shared_buffers需要同时增加checkpoint_segments的值，因为写入大量新增、修改数据需要消耗更多的时间周期。
4. 如果调整shared_buffers参数之后，导致进程重启失败，请参考启动失败的报错信息，采用以下解决方案之一：
   a. 对应调整操作系统kernel.shmall、kernel.shmmax、kernel.shmmin参数，调整方式请参考《安装指南》的“安装前准备 > 修改操作系统配置 > 配置操作系统其他参数”章节。
   b. 执行free -g观察操作系统可用内存和swap空间是否足够，如果内存明显不足，请手动停止其他比较占用内存的用户程序。
   c. 避免设置明显不合理（过大或过小）的shared_buffers值。

segment_buffers

参数说明：设置GaussDB段页式元数据页的内存大小。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，16 ~ 1073741823，单位为8KB。

segment_buffers需要设置为BLCKSZ的整数倍，BLCKSZ目前设置为8KB，即segment_buffers需要设置为8KB整数倍。改变BLCKSZ的值会改变最小值。

默认值：8MB

设置建议：

- segment_buffers用来缓存段页式段头的内容，属于关键元数据信息，为了提高性能建议常用的表的段头都能缓存在buffer中，不被置换出去。建议按照表的个数（包括索引和toast表）* 分区数 * 3 + 128 来设置。乘以3是因为每个表（分区）会有一些额外的元数据段，一般一个表有3个段。最后+128因为段页式表空间管理需要一定数量的buffer。
- 该参数设置过小会导致首次创建段页式表时耗时较久，因此请按照建议进行设置。

bulk_write_ring_size

参数说明：数据并行导入使用的环形缓冲区大小。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，16384 ~ 2147483647，单位为KB。

默认值：2GB

设置建议：建议导入压力大的场景中增加DN中此参数配置。

standby_shared_buffers_fraction

参数说明：备实例所在服务器使用shared_buffers内存缓冲区大小的比例。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：双精度浮点类型，0.1~1.0

默认值：1

temp_buffers

参数说明：设置每个数据库会话使用的LOCAL临时缓冲区的大小。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

在每个会话的第一次使用临时表之前可以改变temp_buffers的值，之后的设置将是无效的。

一个会话将按照temp_buffers给出的限制，根据需要分配临时缓冲区。如果在一个并不需要大量临时缓冲区的会话里设置一个大的数值，其开销只是一个缓冲区描述符的大小。当缓冲区被使用，就会额外消耗8192字节。

取值范围：整型，100~1073741823，单位为8KB。

默认值：1MB

max_prepared_transactions

参数说明：设置可以同时处于"预备"状态的事务的最大数目。增加此参数的值会使GaussDB比系统默认设置需要更多的System V共享内存。

当GaussDB部署为主备双机时，在备机上此参数的设置必须要高于或等于主机上的，否则无法在备机上进行查询操作。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~262143。

默认值：

- 独立部署：

  1200（60核CPU/480G内存，32核CPU/256G内存）；800（16核CPU/128G内存）；400（8核CPU/64G内存）；300（4核CPU/32G内存）；200（4核CPU/16G内存）
- 金融版（标准型）：

CN：900（128核CPU/1024G内存，104核CPU/1024G内存）；800（96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，72核CPU/576G内存，64核CPU/512G内存，60核CPU/480G内存）：400（32核CPU/256G内存，16核CPU/128G内存）；200（8核CPU/64G内存）

DN：4200（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，72核CPU/576G内存，64核CPU/512G内存，60核CPU/480G内存）：2200（32核CPU/256G内存）；1200（16核CPU/128G内存）；800（8核CPU/64G内存）

- 企业版：

  CN：900（128核CPU/1024G内存，104核CPU/1024G内存）；800（96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，80核CPU/512G内存，72核CPU/576G内存，64核CPU/512G内存，60核CPU/480G内存）；400（32核CPU/256G内存，16核CPU/128G内存）；200（8核CPU/64G内存）

  DN：1800（128核CPU/1024G内存，104核CPU/1024G内存）；1200（96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，80核CPU/512G内存，72核CPU/576G内存，64核CPU/512G内存，60核CPU/480G内存）：800（32核CPU/256G内存）；400（16核CPU/128G内存，8核CPU/64G内存）

- 金融版（数据计算型）：

  CN：800（96核CPU/768G内存）；400（72核CPU/576G内存，64核CPU/512G内存）；200（32核CPU/256G内存）

  DN：2400（96核CPU/768G内存）；1200（72核CPU/576G内存）；800（64核CPU/512G内存）；400（32核CPU/256G内存）

说明

为避免在准备步骤失败，线程池模式下此参数的值应大于thread_pool_attr中工作线程个数，非线程池模式下此参数的值不能小于max_connections。

work_mem

参数说明：设置内部排序操作和Hash表在开始写入临时磁盘文件之前使用的内存大小。ORDER BY，DISTINCT和merge joins都要用到排序操作。Hash表在散列连接、散列为基础的聚集、散列为基础的IN子查询处理中都要用到。

对于复杂的查询，可能会同时并发运行好几个排序或者散列操作，每个都可以使用此参数所声明的内存量，不足时会使用临时文件。同样，好几个正在运行的会话可能会同时进行排序操作。因此使用的总内存可能是work_mem的好几倍。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，64~2147483647，单位为KB。

默认值：

- 独立部署：

  128MB（60核CPU/480G内存，32核CPU/256G内存，16核CPU/128G内存）；64MB（8核CPU/64G内存）；32MB（4核CPU/32G内存）；16MB（4核CPU/16G内存）

- 金融版（标准型）：

  CN：128MB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，72核CPU/576G内存，64核CPU/512G内存，60核CPU/480G内存，32核CPU/256G内存，16核CPU/128G内存）；64MB（8核CPU/64G内存）

DN：256MB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存）；128MB（80核CPU/640G内存，72核CPU/576G内存，64核CPU/512G内存，60核CPU/480G内存，32核CPU/256G内存，16核CPU/128G内存）；64MB（8核CPU/64G内存）

- 企业版：

  128MB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，80核CPU/512G内存，72核CPU/576G内存，64核CPU/512G内存，60核CPU/480G内存，32核CPU/256G内存，16核CPU/128G内存）；64MB（8核CPU/64G内存）

- 金融版（数据计算型）：

  128MB（96核CPU/768G内存，72核CPU/576G内存，64核CPU/512G内存）；64MB（32核CPU/256G内存）

须知

设置建议：

依据查询特点和并发来确定，一旦work_mem限定的物理内存不够，算子运算数据将写入临时表空间，带来5-10倍的性能下降，查询响应时间从秒级下降到分钟级。

- 对于串行无并发的复杂查询场景，平均每个查询有5-10关联操作，建议work_mem=50%内存/10。
- 对于串行无并发的简单查询场景，平均每个查询有2-5个关联操作，建议work_mem=50%内存/5。
- 对于并发场景，建议work_mem=串行下的work_mem/物理并发数。
- 对于BitmapScan的哈希表也会受到work_mem的限制，但不会被严格管控下盘。完全Lossify的情况下，哈希表每占用1MB的内存，对应一次BitmapHeapScan的16GB的页面，达到work_mem上限后，会按此比例随数据访问量线性增长。

query_mem

参数说明：设置执行作业所使用的内存。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：0，或大于32MB的整型，默认单位为KB。

默认值：0

须知

- 如果设置的query_mem值大于0，在生成执行计划时，优化器会将作业的估算内存调整为该值。
- 如果设置值为负数或小于32MB，将设置为默认值0，此时优化器不会根据该值调整作业的估算内存。

query_max_mem

参数说明：设置执行作业所能够使用的最大内存。

该参数属于USERSET类型参数，请参考表18-1中对应设置方法进行设置。

取值范围：0，或大于32M的整型，默认单位为KB。

默认值：0

须知

- 如果设置的query_max_mem值大于0，当作业执行时所使用内存超过该值时，将报错退出。
- 如果设置值为负数或小于32M，将设置为默认值0，此时不会根据该值限制作业的内存使用。

maintenance_work_mem

参数说明：设置在维护性操作（比如VACUUM、CREATE INDEX等）中可使用的最大的内存。该参数的设置会影响VACUUM、VACUUM FULL、CLUSTER、CREATE INDEX的执行效率。

该参数属于USERSET类型参数，请参考表18-1中对应设置方法进行设置。

取值范围：整型，1024~2147483647，单位为KB。

默认值：

- 独立部署：

  CN：1GB（60核CPU/480G内存）；512MB（32核CPU/256G内存）；256MB（16核CPU/128G内存）；128MB（8核CPU/64G内存）；64MB（4核CPU/32G内存）；32MB（4核CPU/16G内存）

  DN：2GB（60核CPU/480G内存）；1GB（32核CPU/256G内存）；512MB（16核CPU/128G内存）；256MB（8核CPU/64G内存）；128MB（4核CPU/32G内存）；64MB（4核CPU/16G内存）
- 金融版（标准型）：

  CN：1GB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存）；512MB（72核CPU/576G内存，64核CPU/512G内存）；256MB（60核CPU/480G内存，32核CPU/256G内存，16核CPU/128G内存，8核CPU/64G内存）

  DN：2GB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，72核CPU/576G内存，64核CPU/512G内存，60核CPU/480G内存）；1GB（32核CPU/256G内存）；512MB（16核CPU/128G内存）；256MB（8核CPU/64G内存）
- 企业版：

  CN：1GB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，80核CPU/512G内存）；512MB（72核CPU/576G内存，64核CPU/512G内存）；256MB（60核CPU/480G内存，32核CPU/256G内存，16核CPU/128G内存，8核CPU/64G内存）

  DN：2GB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，80核CPU/512G内存，72核CPU/576G内存，64核CPU/512G内存，60核CPU/480G内存）；1GB（32核CPU/256G内存）；512MB（16核CPU/128G内存）；256MB（8核CPU/64G内存）

- 金融版（数据计算型）：

  CN：1GB（96核CPU/768G内存）；256MB（72核CPU/576G内存，64核CPU/512G内存）；128MB（32核CPU/256G内存）

  DN：2GB（96核CPU/768G内存）；1GB（72核CPU/576G内存，64核CPU/512G内存）；512MB（32核CPU/256G内存）

须知

设置建议：

- 建议设置此参数的值大于**work_mem**，可以改进清理和恢复数据库转储的速度。因为在一个数据库会话里，任意时刻只有一个维护性操作可以执行，并且在执行维护性操作时不会有太多的会话。
- 当**自动清理**线程运行时，**autovacuum_max_workers**倍数的内存将会被分配，所以此时设置maintenance_work_mem的值应该不小于**work_mem**。
- 如果进行大数据量的cluster等，可以在session中调大该值。

max_stack_depth

参数说明：设置GaussDB执行堆栈的最大安全深度。需要这个安全界限是因为在服务器里，并非所有程序都检查了堆栈深度，只是在可能递规的过程，比如表达式计算这样的过程里面才进行检查。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，100~2147483647，单位为KB。

默认值：

- （ulimit -s的设置）- 640 KB的值大于等于2MB时，此参数的默认值为2MB。
- （ulimit -s的设置）- 640 KB的值小于2MB时，此参数的默认值为（ulimit -s的设置）- 640 KB。

须知

设置原则：

- 数据库需要预留640KB堆栈深度，因此此参数可设置的最大值等于操作系统内核允许的最大值（就是ulimit -s的设置）- 640KB。
- 数据库未运行前设置的该参数值大于（ulimit -s的设置）- 640 KB时会导致数据库启动失败；数据库运行阶段设置该参数值大于（ulimit -s的设置）- 640 KB时该值不生效。
- 若（ulimit -s的设置）-640KB小于此参数取值范围的最小值时会导致数据库启动失败。
- 如果设置此参数的值大于实际的内核限制，则一个正在运行的递归函数可能会导致一个独立的服务器进程崩溃。
- 因为并非所有的操作都能够检测，所以建议用户在此设置一个明确的值。
- 默认值最大为2MB，这个值相对比较小，不容易导致系统崩溃。

bulk_read_ring_size

参数说明：并行导出，使用的环形缓冲区大小。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，256~2147483647，单位为KB。

默认值：16MB

enable_early_free

参数说明：控制是否可以实现算子内存的提前释放。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示支持算子内存提前释放。
- off表示不支持算子内存提前释放。

默认值：on

memory_trace_level

参数说明：动态内存使用超过最大动态内存的90%后，记录内存申请信息的管控等级。该参数仅在GUC参数use_workload_manager和enable_memory_limit打开时生效。该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：枚举型

- none：表示不记录内存申请信息。
- level1：动态内存使用超过最大动态内存的90%后，会记录以下信息，并将记录的内存信息保存在$GAUSSLOG/mem_log目录下。
  - 全局内存概况。
  - instance，session，thread三种类型的所有内存上下文中内存占用前20的内存上下文的内存使用情况。
  - 每个内存上下文的totalsize、freesize字段。
- level2：动态内存使用超过最大动态内存的90%后，会记录以下信息，并将记录的内存信息保存在$GAUSSLOG/mem_log目录下。
  - 全局内存概况。
  - instance，session，thread三种类型的所有内存上下文中内存占用前20的内存上下文的内存使用情况。
  - 每个内存上下文的totalsize，freesize字段。
  - 每个内存上下文上所有内存申请的详细信息，包含申请内存所在的文件，行号和大小。

默认值：level1

须知

- 该参数设置为level2后，会记录每个内存上下文的内存申请详情（file，line，size字段），会对性能影响较大，需慎重设置。
- 记录的内存快照信息可以通过系统函数查询，具体请参见《开发指南》中“SQL参考 > 函数和操作符 > 统计信息函数”章节中“gs_get_history_memory_detail(cstring)”。
- use_workload_manager参数关闭的情况下，如果打开bypass_workload_manager，则该参数也会生效，但是因为bypass_workload_manager是SIGHUP类型，reload方式设置后需要重启数据库才会使得当前功能生效。
- 记录的内存上下文是经过将同一类型所有重名的内存上下文进行汇总之后得到的。

resilience_memory_reject_percent

参数说明：用于控制内存过载逃生的动态内存占用百分比。该参数仅在GUC参数use_workload_manager和enable_memory_limit打开时生效。该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串，长度大于0。

该参数分为recover_memory_percent,、overload_memory_percent 2部分，这2个部分的具体含义如下：

- recover_memory_percent：内存从过载状态恢复正常状态的动态内存使用占最大动态内存的百分比，当动态内存使用小于最大动态内存乘以该值对应的百分比后，停止过载逃生并放开新连接接入，取值为0~100，设置为多少表示百分之多少。
- overload_memory_percent：内存过载时动态内存使用占最大动态内存的百分比，当动态内存使用大于最大动态内存乘以该值对应的百分比后，表示当前内存已经过载，触发过载逃生kill会话并禁止新连接接入，取值为0~100，设置为多少表示百分之多少。

默认值：'0,0'，表示关闭内存过载逃生功能。

示例：

```
resilience_memory_reject_percent = '70,90'
```

表示内存使用超过最大内存上限的90%后禁止新连接接入并kill堆积的会话，kill会话过程中内存恢复到最大内存的70%以下时停止kill会话并允许新连接接入。

须知

- 最大动态内存和已使用的动态内存可以通过pv_total_memory_detail视图查询获得，最大动态内存：max_dynamic_memory，已使用的动态内存：dynamic_used_memory。
- 该参数如果设置的百分比过小，则会频繁触发内存过载逃生流程，会使正在执行的会话被强制退出，新连接短时间接入失败，需要根据实际内存使用情况慎重设置。
- use_workload_manager参数关闭的情况下，如果打开bypass_workload_manager，则该参数也会生效，但是因为bypass_workload_manager是SIGHUP类型，reload方式设置后需要重启数据库才会使得当前功能生效。
- recover_memory_percent和overload_memory_percent的值可以同时为0，除此之外，recover_memory_percent的值必须要小于overload_memory_percent的值，否则会设置不生效。

18.3.4.2 磁盘空间

介绍与磁盘空间相关的参数，用于限制临时文件所占用的磁盘空间。

sql_use_spacelimit

参数说明：限制单个SQL在单个DN上，触发落盘操作时，落盘文件的空间大小，管控的空间包括普通表、临时表以及中间结果集落盘占用的空间，对初始用户不生效。

该参数属于USERSET类型参数，请参考**表 GUC参数分类**中对应设置方法进行设置

取值范围：整型，-1~2147483647，单位为KB。其中-1表示没有限制。

默认值：-1

temp_file_limit

参数说明：限制一个会话中，触发下盘操作时，下盘文件占用的空间大小。例如一次会话中，排序和哈希表使用的临时文件，或者游标占用的临时文件。

此设置为会话级别的下盘文件控制。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

SQL查询执行时使用的临时表空间不在此限制。

取值范围：整型，-1~2147483647，单位为KB。其中-1表示没有限制。

默认值：-1

18.3.4.3 内核资源使用

介绍与操作系统内核相关的参数，这些参数是否生效依赖于操作系统的设置。

max_files_per_process

参数说明：设置每个服务器进程允许同时打开的最大文件数目。如果操作系统内核强制一个合理的数目，则不需要设置。

但是在一些平台上（特别是大多数BSD系统），内核允许独立进程打开比系统真正可以支持的数目大得多的文件数。如果用户发现有的“Too many open files”这样的失败现象，请尝试缩小这个设置。通常情况下需要满足，系统FD（file descriptor）数量>=最大并发数*当前物理机主DN个数*max_files_per_process*3。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，25~2147483647

默认值：1024

shared_preload_libraries

参数说明：此参数用于声明一个或者多个在服务器启动的时候预先装载的共享库，多个库名称之间用逗号分隔，仅sysadmin用户可以访问。比如'$libdir/mylib'会在加载标准库目录中的库文件之前预先加载mylib.so（某些平台上可能是mylib.sl）库文件。

可以用这个方法预先装载GaussDB的存储过程库，通常是使用'$libdir/plXXX'语法。XXX只能是pgsql，perl，tcl，python之一。

通过预先装载一个共享库并在需要的时候初始化它，可以避免第一次使用这个库的加载时间。但是启动每个服务器进程的时间可能会增加，即使进程从来没有使用过这些库。因此建议对那些将被大多数会话使用的库才使用这个选项。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

- 如果被声明的库不存在，GaussDB服务将会启动失败。
- 每一个支持GaussDB的库都有一个特殊的标记用于保证兼容性。因此，不支持GaussDB的库不能用这种方法加载。

取值范围：字符串

默认值：security_plugin

18.3.4.4 基于开销的清理延迟

这个特性的目的是允许管理员减少VACUUM和ANALYZE语句在并发活动的数据库上的I/O影响。比如，像VACUUM和ANALYZE这样并不需要迅速完成的维护语句，减少其对系统执行其它数据库操作的干扰。基于开销的清理延迟为管理员提供了一个实现这个目的手段。

须知

有些清理操作会持有关键的锁，这些操作应该尽快结束并释放锁。所以GaussDB的机制是，在这类操作过程中，基于开销的清理延迟不会发生作用。为了避免在这种情况下的长延时，实际的开销限制取下面两者之间的较大值：

- vacuum_cost_delay*accumulated_balance/vacuum_cost_limit
- vacuum_cost_delay*4

背景信息

在ANALYZE | ANALYSE和VACUUM语句执行过程中，系统维护一个内部的记数器，跟踪所执行的各种I/O操作的近似开销。如果积累的开销达到了vacuum_cost_limit声明的限制，则执行这个操作的进程将睡眠vacuum_cost_delay指定的时间。然后它会重置记数器然后继续执行。

这个特性是缺省关闭的。要想打开它，把vacuum_cost_delay变量设置为一个非零值。

vacuum_cost_delay

参数说明：指定开销超过vacuum_cost_limit的值时，进程睡眠的时间。

要注意在许多系统上，睡眠的有效分辨率是10毫秒。因此把vacuum_cost_delay设置为一个不是10的整数倍的数值与将它设置为下一个10的整数倍作用相同。

此参数一般设置较小，常见的设置是10或20毫秒。调整此特性资源占用率时，建议调整其他参数，而不是此参数。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~100，正数值表示打开基于开销的清理延迟特性；0表示关闭基于开销的清理延迟特性。

默认值：

vacuum_cost_page_hit

参数说明：清理一个在共享缓存里找到的缓冲区的预计开销。表示锁住缓冲池、查找共享的Hash表、扫描页面内容的开销。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~10000。

默认值：1

vacuum_cost_page_miss

参数说明：清理一个要从磁盘上读取的缓冲区的预计开销。表示锁住缓冲池、查找共享Hash表、从磁盘读取需要的数据块、扫描它的内容的开销。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~10000。

默认值：10

vacuum_cost_page_dirty

参数说明：清理修改一个原先是干净的块的预计开销。表示把一个脏的磁盘块再次刷新到磁盘上的额外开销。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~10000

默认值：20

vacuum_cost_limit

参数说明：设置清理进程休眠的开销限制。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1~10000。

默认值：1000

18.3.4.5 后端写进程

介绍后端写（background writer）进程的参数配置。后端写进程的功能就是把共享缓冲区中的脏数据（指共享缓冲区中新增或者修改的内容）写入到磁盘。目的是让数据库进程在进行用户查询时可以很少或者几乎不等待写动作的发生（写动作由后端写进程完成）。

此机制同样也减少了检查点造成的性能下降。后端写进程将持续的把脏页面刷新到磁盘上，所以在检查点到来的时候，只有几个页面需要刷新到磁盘上。但是这样还是增加了I/O的总净负荷，因为以前的检查点间隔里，一个重复弄脏的页面可能只会冲刷一次，而同一个间隔里，后端写进程可能会写好几次。在大多数情况下，连续的低负荷要比周期性的尖峰负荷好，但是在本节讨论的参数可以用于按实际需要调节其行为。

bgwriter_delay

参数说明：设置后端写进程写"脏"共享缓冲区之间的时间间隔。每一次，后端写进程都会为一些脏的缓冲区发出写操作，全量checkpoint模式用bgwriter_lru_maxpages参数控制每次写的量，然后休眠bgwriter_delay毫秒后才再次启动；增量checkpoint模式下，根据设定candidate_buf_percent_target计算目标空闲缓冲页面个数，不足时每隔bgwriter_delay毫秒刷一批页面下盘，刷页个数根据目标差距百分比计算，会根据max_io_capacity限制最大数量。

在许多系统上，休眠延时的有效分辨率是10毫秒。因此，设置一个不是10的倍数的数值与把它设置为下一个10的倍数是一样的效果。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，10~10000，单位为毫秒。

默认值：2s

设置建议：在数据写压力比较大的场景中可以尝试减小该值以降低checkpoint的压力。

candidate_buf_percent_target

参数说明：设置用于增量检查点打开时，候选buffer链中可用buffer数目占据shared_buffer内存缓冲区百分比的期望值，当前候选链中的数目少于目标值时，bgwriter线程会启动将满足条件的脏页刷盘。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：双精度浮点类型，0.1～0.85

默认值：0.3

bgwriter_lru_maxpages

参数说明：设置后端写进程每次可写入磁盘的“脏”缓存区的个数。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0～1000

说明

此参数设置为0表示禁用后端写功能，禁用后端写功能不会对checkpoints产生影响。

默认值：100

bgwriter_lru_multiplier

参数说明：通过与已使用缓存区数目的乘积评估下次服务器需要的缓存区数目。

写“脏”缓存区到磁盘的数目取决于服务器最近几次使用的缓存区数目。最近的buffers数目的平均值乘以bgwriter_lru_multiplier是为了评估下次服务器进程需要的buffers数目。在有足够多的干净的、可用的缓存区之前，后端写进程会一直写“脏”缓存区的（每次写的缓存区数目不会超过bgwriter_lru_maxpages的值）。

设置bgwriter_lru_multiplier的值为1.0表示一种“实时”策略，其作用是精准预测下次写“脏”缓冲区的数目。设置为较大的值可以应对突然的需求高峰，而较小的值则可以让服务器进程执行更多的写操作。

设置较小的bgwriter_lru_maxpages和bgwriter_lru_multiplier会减小后端写进程导致的额外I/O开销，但是服务器进程必须自己发出写操作，增加了对查询的响应时间。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：浮点型，0~10。

默认值：2

pagewriter_thread_num

参数说明：设置用于增量检查点打开后后台刷页的线程数，主要是按照脏页置脏的顺序刷盘，用于推进recovery点。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1～16

默认值：4

dirty_page_percent_max

参数说明：设置用于增量检查点打开后脏页数量占shared_buffers的百分比。达到这个设定值时，后台刷页线程将以设置的max_io_capacity计算出的最大值刷脏页。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：浮点型，0.1～1

默认值：0.9

pagewriter_sleep

参数说明：设置用于增量检查点打开后，pagewriter线程每隔pagewriter_sleep的时间刷一批脏页下盘。当脏页占据shared_buffers的比例达到dirty_page_percent_max时，每批页面数量以设定的max_io_capacity计算出的值刷页，其余情况每批页面数量按比例相对减少。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0～3600000，单位为毫秒。

默认值：2000ms（2s）

max_io_capacity

参数说明：设置后端写进程批量刷页每秒的I/O上限，需要根据具体业务场景和机器磁盘I/O能力进行设置，要求RTO很短时间或者数据量比共享内存大多倍的情况，业务访问数据量又是随机访问时，该值不宜过小。设置较小的max_io_capacity会减小后端写进程刷页个数，如果业务触发页面淘汰多时，该值设置小会影响业务。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，30720~10485760，单位是KB。

默认值：500MB/s

enable_consider_usecount

参数说明：设置backend线程在页面置换时是否考虑页面热度，建议大容量场景下开启此参数。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on/true表示考虑页面热度。
- off/false表示不考虑页面热度。

默认值：off

dw_file_num

参数说明：设置批量双写文件的数量，该值与pagewriter_thread_num有关，不会大于pagewriter_thread_num，如果设置过大，内部会纠正为pagewriter_thread_num。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1～16

默认值：1

dw_file_size

参数说明：设置每个批量双写文件的大小。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，32～256

默认值：256

18.3.4.6 异步 I/O

checkpoint_flush_after

参数说明：设置checkpointer线程刷页个数超过设定的阈值时，告知操作系统开始将操作系统缓存中的页面异步刷盘。GaussDB中，磁盘页大小为8KB。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~256（0表示关闭异步刷盘功能），单位页面（8K）。例如，取值32，表示checkpointer线程连续写32个磁盘页，即32*8=256KB磁盘空间后会进行异步刷盘。

默认值：256KB（即32个页面）

bgwriter_flush_after

参数说明：设置background writer线程刷页个数超过设定的阈值时，告知操作系统开始将操作系统缓存中的页面异步刷盘。GaussDB中，磁盘页大小为8KB。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~256（0表示关闭异步刷盘功能），单位页面（8K）。例如，取值64，表示background writer线程连续写64个磁盘页，即64*8=512KB磁盘空间后会进行异步刷盘。

默认值：512KB（即64个页面）

backend_flush_after

参数说明：设置backend线程刷页个数超过设定的阈值时，告知操作系统开始将操作系统缓存中的页面异步刷盘。GaussDB中，磁盘页大小为8KB。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~256（0表示关闭异步刷盘功能）。例如，取值64，表示backend线程连续写64个磁盘页，即64*8=512KB磁盘空间后会进行异步刷盘。

默认值：0

18.3.5 并行导入

GaussDB提供了并行导入功能，以快速、高效地完成大量数据导入。介绍GaussDB并行导入的相关参数。

raise_errors_if_no_files

参数说明：导入时是否区分“导入文件记录数为空”和“导入文件不存在”。raise_errors_if_no_files=TRUE，则“导入文件不存在”的时候，GaussDB将抛出“文件不存在的”错误。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示导入时区分“导入文件记录数为空”和“导入文件不存在”。
- off表示导入时不区分“导入文件记录数为空”和“导入文件不存在”。

默认值：off

gds_debug_mod

参数说明：为了增强对Gauss Data Service（以下简称GDS）相关问题的分析定位能力，可以通过此参数选择是否开启GDS的debug功能。参数开启后，将在集群节点对应的日志中输出GDS每次收发的包裹类型、命令交互的对端以及其他交互相关的细节信息，方便记录Gaussdb端状态机的状态跳转，以及目前所处的状态信息。此参数打开会输出额外日志，增加日志I/O开销，进而影响性能和日志的信息有效性，因此请仅在定位GDS问题时开启。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：

- on表示开启GDS debug功能。
- off表示不开启GDS debug功能。

默认值：off

safe_data_path

参数说明：设置初始用户以外的路径前缀限制，目前包括copy和高级包路径限制（不支持参数路径结尾处有"/"，不支持使用的路径中有".."）。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串（不超过4096个字符）

默认值：NULL

enable_copy_server_files

参数说明：是否开启copy服务器端文件的权限。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启copy服务端文件的权限。
- off表示不开启copy服务端文件的权限。

默认值：off

须知

当参数enable_copy_server_files关闭时，只允许初始用户执行COPY FROM FILENAME或COPY TO FILENAME命令，当参数enable_copy_server_files打开，允许具有SYSADMIN权限的用户或继承了内置角色gs_role_copy_files权限的用户执行。

18.3.6 预写式日志

18.3.6.1 设置

wal_level

参数说明：设置写入WAL信息量的级别，不能为空或被注释掉。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

- 如果需要启用WAL日志归档和主备机的数据流复制，必须将此参数设置为archive、hot_standby或者logical。
- 如果该参数设置为archive或minimal，则hot_standby必须设置为off，因为分布式环境下该参数不支持设置hot_standby为off，因此不建议该参数设置为archive或minimal，否则数据库将无法启动。

取值范围：枚举类型

- minimal

  优点：一些重要操作（包括创建表、创建索引、簇操作和表的复制）都能安全的跳过，这样就可以使操作变得更快。

  缺点：WAL仅提供从数据库服务器崩溃或者紧急关闭状态恢复时所需要的基本信息，无法用WAL归档日志恢复数据。

- archive

  这个参数增加了WAL归档需要的日志信息，从而可以支持数据库的归档恢复。

- hot_standby
  - 这个参数进一步增加了在备机上运行的SQL查询的信息，这个参数只能在数据库服务重新启动后生效。
  - 为了在备机上开启只读查询，wal_level必须在主机上设置成hot_standby，并且备机必须打开hot_standby参数。hot_standby和archive级别之间的性能只有微小的差异，如果它们的设置对产品的性能影响有明显差异，欢迎反馈。

- logical

  设置为logical后才可以进行逻辑日志解析，设置后xLog日志中会额外记录主键信息。

默认值：hot_standby

fsync

参数说明：设置GaussDB服务器是否使用fsync()系统函数（请参见wal_sync_method）确保数据的更新及时写入物理磁盘中。

该参数属于SIGHUP类型参数，请参考表18-1中对应设置方法进行设置。

须知

- 使用fsync()系统函数可以保证在操作系统或者硬件崩溃的情况下将数据恢复到一个已知的状态。
- 如果将此参数关闭，可能会在系统崩溃时无法恢复原来的数据，导致数据库不可用。

取值范围：布尔型

- on表示使用fsync()系统函数。
- off表示不使用fsync()系统函数。

默认值：on

synchronous_commit

参数说明：设置当前事务的同步方式。

该参数属于USERSET类型参数，请参考表18-1中对应设置方法进行设置。

通常情况下，一个事务产生的日志的同步顺序如下：

1. 主机将日志内容写入本地内存。
2. 主机将本地内存中的日志写入本地文件系统。
3. 主机将本地文件系统中的日志内容刷盘。
4. 主机将日志内容发送给备机。
5. 备机接收到日志内容，存入备机内存。
6. 备机将备机内存中的日志写入备机文件系统。
7. 备机将备机文件系统中的日志内容刷盘。
8. 备机回放日志，完成对数据文件的增量更新。

取值范围：枚举类型

- on（true, yes, 1）：表示主机事务提交需要等待备机将对应日志刷新到磁盘。
- off（false, no, 0）：表示主机事务提交无需等待主机自身将对应日志刷新到磁盘，通常也称为异步提交。
- local：表示主机事务提交需要等待主机自身将对应日志刷新到磁盘，通常也称为本地提交。
- remote_write：表示主机事务提交需要等待备机将对应日志写到文件系统（无需刷新到磁盘）。
- remote_receive：表示主机事务提交需要等待备机接收到对应日志数据（无需写入文件系统）。

- remote_apply：表示主机事务提交需要等待备机完成对应日志的回放操作。
- true：同on。
- false：同off。
- yes：同on。
- no：同off。
- 1：同on。
- 0：同off。
- 2：同remote_apply。

默认值：on

wal_sync_method

参数说明：设置向磁盘强制更新WAL数据的方法。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

如果将**fsync**关闭，这个参数的设置就没有意义，因为所有数据更新都不会强制写入磁盘。

取值范围：枚举类型

- open_datasync表示用带O_DSYNC选项的open()打开“WAL”文件。
- fdatasync表示每次提交的时候都调用fdatasync()（支持suse10和suse11）。
- fsync_writethrough表示每次提交的时候调用fsync()强制把缓冲区任何数据写入磁盘。

  说明

  由于历史原因，Windows平台支持将wal_sync_method设置为fsync_writethrough。在windows平台上fsync_writethrough和fsync等效。
- fsync表示每次提交的时候调用fsync()（支持suse10和suse11）。
- open_sync表示用带O_SYNC选项的open()写“WAL”文件（支持suse10和suse11）。

  说明

  不是所有的平台都支持以上参数。

默认值：fdatasync

full_page_writes

参数说明：设置GaussDB服务器在检查点之后对页面的第一次修改时，是否将每个磁盘页面的全部内容写到WAL日志中。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

- 设置这个参数是因为在操作系统崩溃过程中可能磁盘页面只写入了一部分内容，从而导致在同一个页面中包含新旧数据的混合。在崩溃后的恢复期间，由于在WAL日志中存储的行变化信息不够完整，因此无法完全恢复该页。把完整的页面影像保存下来就可以保证页面被正确还原，代价是增加了写入WAL日志的数据量。
- 关闭此参数，在系统崩溃的时候，可能无法恢复原来的数据。如果服务器硬件的特质（比如电池供电的磁盘控制器）可以减小部分页面的写入风险，或者文件系统特性支持（比如ReiserFS 4），并且清楚知道写入风险在一个可以接受的范畴，可以关闭这个参数。

取值范围：布尔型

- on表示启用此特性。
- off表示关闭此特性。

默认值：on

wal_log_hints

参数说明：设置在检查点之后对页面的第一次修改为页面上元组hint bits的修改时，是否将整个页面的全部内容写到WAL日志中。不推荐用户修改此设置。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示整个页面全部内容写到WAL日志中。
- off表示整个页面内容不会写到WAL日志中。

默认值：on

wal_buffers

参数说明：设置用于存放WAL数据的共享内存空间的XLOG_BLCKSZ数，XLOG_BLCKSZ的大小默认为8KB。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：-1~2^{18}，最小值为-1，最大值为262144，单位为8KB。

- 如果设置为-1，表示wal_buffers的大小随着参数shared_buffers自动调整，默认为shared_buffers的1/32。当该值小于8时，会被强制设置为8；当该值大于2048时，会被强制设置为2048。
- 如果设置为其他值，当小于4时，会被强制设置为4。
- 独立部署：1GB（60核CPU/480G内存，32核CPU/256G内存）；512MB（16核CPU/128G内存）；256MB（8核CPU/64G内存）；128MB（4核CPU/32G内存）；64MB（4核CPU/16G内存）
- 金融版（标准型）：

  CN：512MB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存）；256MB（72核CPU/576G内存，64核CPU/512G内存，60核CPU/480G内存）；128MB（32核CPU/256G内存，16核CPU/128G内存）；64MB（8核CPU/64G内存）

DN：2GB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存）；1GB（80核CPU/640G内存，72核CPU/576G内存，64核CPU/512G内存，60核CPU/480G内存，32核CPU/256G内存），512MB（16核CPU/128G内存）；256MB（8核CPU/64G内存）

- 企业版：

  CN：512MB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，80核CPU/512G内存）；256MB（72核CPU/576G内存，64核CPU/512G内存，60核CPU/480G内存）；128MB（32核CPU/256G内存，16核CPU/128G内存）；64MB（8核CPU/64G内存）

  DN：1GB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，80核CPU/512G内存，72核CPU/576G内存，64核CPU/512G内存，60核CPU/480G内存，32核CPU/256G内存）；512MB（16核CPU/128G内存）；256MB（8核CPU/64G内存）

- 金融版（数据计算型）：

  CN：256MB（96核CPU/768G内存）；125MB（72核CPU/576G内存，64核CPU/512G内存）；64MB（32核CPU/256G内存）

  DN：1GB（96核CPU/768G内存，72核CPU/576G内存，64核CPU/512G内存）；512MB（32核CPU/256G内存）

设置建议：每次事务提交时，WAL缓冲区的内容都写入到磁盘中，因此设置为很大的值不会带来明显的性能提升。如果将它设置成几百兆，就可以在有很多即时事务提交的服务器上提高写入磁盘的性能。根据经验来说，默认值可以满足大多数的情况。

wal_writer_delay

参数说明：WalWriter进程的写间隔时间。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

如果时间过长可能造成WAL缓冲区的内存不足，时间过短会引起WAL不断写入，增加磁盘I/O负担。

取值范围：整型， 1～10000（毫秒）

默认值：200ms

commit_delay

参数说明：表示一个已经提交的数据在WAL缓冲区中存放的时间。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

- 设置为非 0 值时事务执行commit后不会立即写入WAL中，而仍存放在WAL缓冲区中，等待WalWriter进程周期性写入磁盘。
- 如果系统负载很高，在延迟时间内，其他事务可能已经准备好提交。但如果没有事务准备提交，这个延迟就是在浪费时间。

取值范围：整型， 0～100000（微秒），其中0表示无延迟。

默认值：0

commit_siblings

参数说明：当一个事务发出提交请求时，如果数据库中正在执行的事务数量大于此参数的值，则该事务将等待一段时间（**commit_delay**的值），否则该事务则直接写入WAL。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型， 0～1000

默认值：5

wal_block_size

参数说明：说明WAL日志段文件中日志页面的大小。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：整型，单位为Byte。

默认值：8192

wal_segment_size

参数说明：说明WAL日志段文件的大小。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：整型，单位为8KB。

默认值：16MB（2048 * 8KB）

force_promote

参考说明：备机强切功能开关。

备机强切在集群故障状态下，以丢失部分数据为代价换取集群尽可能快的恢复服务；是集群状态为不可用时的一个逃生方法，不建议频繁触发。如果操作者不清楚备机强切后丢失数据对业务的影响，请勿使用本功能。

使用时需要分别在DN和cmserver开启并重启集群生效，备机强切功能请参考《故障处理》的“应急处理 > 备机强切”章节。

取值范围：整型，0或1，0表示关闭，1表示开启。

默认值：0

wal_file_init_num

参数说明：设置WAL writer辅助线程一次创建xLog段文件的数量。

该参数属于POSTMASTER类型参数，参考**表18-2**中对应设置方法进行设置。

取值范围：整型，0~1000000

默认值：10

wal_debug

参数说明：允许输出wal相关的调试信息。仅在编译时开启WAL_DEBUG编译宏时可用。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔类型

默认值：false

walwriter_sleep_threshold

参考说明：xLog刷新器进入睡眠之前的空闲xLog刷新次数。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1~50000

默认值：500

wal_flush_timeout

参数说明：遍历WalInsertStatusEntryTbl的超时时间。xLog刷盘自适应控制的刷盘I/O遍历WalInsertStatusEntryTbl等待的最大时间。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

须知

如果时间过长可能造成xLog刷盘频率降低，降低xLog处理性能。

取值范围：整型， 0 ~ 90000000（微秒）

默认值：2us

wal_flush_delay

参数说明：遍历WalInsertStatusEntryTbl时，遇到WAL_NOT_COPIED状态entry时等待的时间间隔。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型， 0 ~ 90000000（微秒）

默认值：1us

18.3.6.2 检查点

checkpoint_segments

参数说明：设置**checkpoint_timeout**周期内所保留的最少WAL日志段文件数量。每个日志文件大小为16MB。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1~2147483646

提升此参数可加快大数据的导入速度，但需要结合**checkpoint_timeout**、**shared_buffers**这两个参数统一考虑。这个参数同时影响WAL日志段文件复用数量，通常情况下pg_xlog文件夹下最大的复用文件个数为2倍的checkpoint_segments个，复用的文件被改名为后续即将使用的WAL日志段文件，不会被真正删除。

默认值：1024

checkpoint_timeout

参数说明：设置自动WAL检查点之间的最长时间。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，30～3600（秒）

在提升**checkpoint_segments**以加快大数据导入的场景也需将此参数调大，同时这两个参数提升会加大**shared_buffers**的负担，需要综合考虑。

默认值：15min

checkpoint_completion_target

参数说明：指定检查点完成的目标。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：双精度浮点类型，0.0～1.0

默认值：0.5

说明

默认值0.5表示：每个checkpoint需要在checkpoints间隔时间的50%内完成。

checkpoint_warning

参数说明：如果由于填充检查点段文件导致检查点发生的时间间隔接近这个参数表示的秒数，就向服务器日志发送一个建议增加**checkpoint_segments**值的消息。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~2147483647（秒），其中0表示关闭警告。

默认值：5min

推荐值：5min

checkpoint_wait_timeout

参数说明：设置请求检查点等待checkpointer线程启动的最长时间。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，2～3600（秒）

默认值：1min

enable_incremental_checkpoint

参数说明：增量检查点开关。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

默认值：on

enable_double_write

参数说明：双写开关，增量检查点开关打开时，不再使用full_page_writes防止半页写问题，而是依赖双写特性保护。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

默认值：on

incremental_checkpoint_timeout

参数说明：增量检查点开关打开之后，设置自动WAL检查点之间的最长时间。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1~3600（秒）

默认值：1min

enable_xlog_prune

参数说明：设置在任一备机断联时，主机是否根据xLog日志的大小超过参数max_size_for_xlog_prune的值而回收日志。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- 设置为on时，如果任一备机断联时，主机回收日志。
- 设置为off时，如果任一备机断联时，主机不回收日志。

默认值：on

max_size_for_xlog_prune

参数说明：在enable_xlog_prune打开时生效，工作机制如下：

1. 如果replconninfo系列guc参数配置的所有备机都连着主机，那么该参数实际不起作用。
2. 如果replconninfo系列guc参数配置的备机至少有一个没有连着主机，那么该参数生效：当主机历史日志数量大于该参数值，会强制回收。例外：在同步提交模式下（即synchronous_commit参数非local或off时），如果还存在连着的备机，那么主机会考虑保留满足多数派备机中最小日志接受位置的日志，这种情况下，保留的日志可能会多余max_size_for_xlog_prune参数值。
3. 如果有任何一个备机正在build，那么该参数不会生效，主机日志会全量保留，防止build操作由于日志回收重复失败。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0～2147483647，单位为KB

默认值：256GB

max_redo_log_size

参数说明：备DN表示当前回放的最新检查点位置和当前日志回放位置之间日志量的期望值，主DN表示恢复点到当前最新日志之间日志量的期望值，关注RTO的情况下，这个值建议不宜过大。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，163840～2147483647，单位为KB

默认值：1048576，单位KB

18.3.6.3 日志回放

recovery_time_target

参数说明：设置recovery_time_target秒能够让备机完成日志写入和回放。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~3600（秒）

0是指不开启日志流控，1~3600是指备机能够在recovery_time_target时间内完成日志的写入和回放，可以保证主机与备机切换时能够在recovery_time_target秒完成日志写入和回放，保证备机能够快速升主机。recovery_time_target设置时间过小会影响主机的性能，设置过大会失去流控效果。

默认值：60

recovery_max_workers

参数说明：设置最大并行回放线程个数。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~20

默认值：4

recovery_parallelism

参数说明：查询实际回放线程个数，该参数为只读参数，无法修改。

该参数属于POSTMASTER类型参数，受recovery_max_workers以及recovery_parse_workers参数影响，任意一值大于0时，recover_parallelism将被重新计算。

取值范围：整型，1~2147483647

默认值：1

recovery_parse_workers

参数说明：是极致RTO特性中ParseRedoRecord线程的数量。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1~16

仅在开启极致RTO情况下可以设置recovery_parse_workers为>1。需要配合recovery_redo_workers使用。若同时开启recovery_parse_workers和recovery_max_workers，以开启极致RTO的recovery_parse_workers为准，并行回放特性失效。因极致RTO配置要求，仅在参数**replication_type**设置成1时可以设置recovery_parse_workers为>1。同时，开启极致RTO需保证**wal_receiver_buffer_size**的值大于等于32MB（推荐64MB）。

默认值：1

说明

- 从V500R001C00版本升级到V500R001C10及其后续版本后，建议设置该参数为2，并重启DN。
- 打开极致RTO后，备机会额外启动recovery_parse_workers * (recovery_redo_workers + 2) + 5个线程，占用更多的CPU、内存和I/O资源。混合部署场景可能对主机性能会有影响。
- 从本版本开始，极致RTO不再自带流控，流控统一由**recovery_time_target**参数来控制。

recovery_redo_workers

参数说明：是极致RTO特性中每个ParseRedoRecord线程对应的PageRedoWorker数量。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1~8

需要配合recovery_parse_workers使用。在配合recovery_parse_workers使用时，只有recovery_parse_workers大于1，recovery_redo_workers参数才生效。

默认值：1

说明

从V500R001C00版本升级到V500R001C10及其后续版本后，建议根据环境的CPU个数进行参数设置，并重启DN。CPU个数小于16个，建议设置成2；大于16小于32个，建议设置成4；大于32个，建议设置成8。

enable_page_lsn_check

参数说明：数据页lsn检查开关。回放时，检查数据页当前的lsn是否是期望的lsn。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

默认值：on

redo_bind_cpu_attr

参数说明：用于控制回放线程的绑核操作，仅sysadmin用户可以访问。该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串，长度大于0

可选择的配置方式有：1. 'nobind' ，线程不做绑核；2. 'nodebind: 1, 2'，利用NUMA组1,2中的CPU core进行绑核；3. 'cpubind: 0-30'，利用0-30号CPU core进行绑核。该参数不区分大小写。

默认值：'nobind'

说明

本参数主要用于arm环境下的绑核操作。推荐将所有的回放线程绑定到一个numa组内，性能会更好，针对混合部署的场景，推荐将同一个机器上的不同节点的回放线程绑定到不同的numa组。

18.3.6.4 归档

archive_timeout

参数说明：表示归档周期。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

- 超过该参数设定的时间时强制切换WAL段。
- 由于强制切换而提早关闭的归档文件仍然与完整的归档文件长度相同。因此，将archive_timeout设为很小的值将导致占用巨大的归档存储空间，建议将archive_timeout设置为60秒。

取值范围：整型，0 ~ 1073741823，单位为秒，其中0表示禁用该功能。

默认值：0

archive_interval

参数说明：表示归档间隔时间。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

- 超过该参数设定的时间时强制归档日志文件。
- 由于归档有I/O操作，不可过于频繁地归档，也不能设置较大影响PITR的RPO建议使用默认值。

取值范围：整型，1 ~ 1000，单位为秒。

默认值：1

time_to_target_rpo

参数说明：双集群异地灾备模式下，设置主集群发生异常发生时到已归档到OBS的恢复点所允许的time_to_target_rpo秒。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~3600（秒）

双集群异地灾备模式下，主集群日志将被归档到OBS。0是指不开启日志流控，1~3600是指设置主集群发生异常发生时到已归档到OBS的恢复点所允许的time_to_target_rpo秒，保证主集群因灾难崩溃时，最多可能丢失的数据的时长在允许范围内。time_to_target_rpo设置时间过小会影响主机的性能，设置过大会失去流控效果。

默认值：10

18.3.7 双机复制

18.3.7.1 发送端服务器

max_wal_senders

参数说明：指定事务日志发送进程的并发连接最大数量。不可大于等于**max_connections**。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

wal_level必须设置为archive、hot_standby或者logical以允许备机的连接。

取值范围：整型，0 ~ 1024（建议取值范围：8 ~ 100）

说明

只有当使用单DN实例无主备场景下才可以设置0。

默认值：

设置建议：每个备机与主机的日志复制连接均会占用一个walsender线程，因此此参数务必大于等于DN数量，否则会导致备机无法连接主机。当有逻辑复制需求时，每个日

志抽取线程会占用一个walsender线程，有逻辑复制时需要设置max_wal_senders大于备机+逻辑复制抽取线程的数量。

wal_keep_segments

参数说明：“pg_xlog”目录下保留事务日志文件的最小数目。备机通过获取主机此处的日志进行流复制。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，2 ~ INT_MAX

默认值：128

设置建议：

- 当服务器开启日志归档或者从检查点恢复时，保留的日志文件数量可能大于wal_keep_segments设定的值。
- 如果此参数设置过小，则在备机请求事务日志时，此事务日志可能已经被产生的新事务日志覆盖，导致请求失败，主备关系断开。
- 当双机为异步传输时，以COPY方式连续导入4G以上数据需要增大wal_keep_segments配置。以T6000单板为例，如果导入数据量为50G，建议调整参数为1000。您可以在导入完成并且日志同步正常后，动态恢复此参数设置。

wal_sender_timeout

参数说明：设置本端等待事务日志接收端接收日志的最大等待时间。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

- 如果主机数据较大，重建操作需要增大此参数的值，主机数据在500GB时，此参数的参考值为600s。
- 此值不能大于wal_receiver_timeout或数据库重建时的超时参数。

取值范围：整型，0 ~ 2147483647，单位为毫秒（ms）。

默认值：6s

max_replication_slots

参数说明：设置主机端的日志复制slot个数。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0 ~ 1024（建议取值范围：8 ~ 100）

默认值：20

设置建议：

须知

当使用双机复制、备份恢复、逻辑解码时，该参数值建议设为：当前物理流复制槽数+备份槽数+所需的逻辑复制槽数。

如果实际设置值比上述建议值小，可能造成这些功能不可用或异常。

物理流复制槽提供了一种自动化的方法来确保主DN在所有备DN收到xLog之前，xLog不会被移除。也就是说物理流复制槽用于支撑集群HA。集群所需要的物理流复制槽数为：一组DN中，备和与主DN之间的比例。例如，假设集群的DN高可用方案为1主、2备，则所需物理流复制槽数为2。又例如，假设集群的DN高可用方案为1主3备，则所需物理流复制槽数为3。

备份槽：记录备份执行过程中的一些复制信息，全量备份和增量备份各自对应单独的备份槽，共2个。

关于逻辑复制槽数，请按如下规则考虑。

- 一个逻辑复制槽只能解码一个Database的修改，如果需要解码多个Database，则需要创建多个逻辑复制槽。
- 如果需要多路逻辑复制同步给多个目标数据库，在源端数据库需要创建多个逻辑复制槽，每个逻辑复制槽对应一条逻辑复制链路。

max_keep_log_seg

参数说明：流控参数，逻辑复制在DN本地会解析物理日志转换成逻辑日志，当未被解析的物理日志文件数量大于该参数时会触发限流。此参数为0表示关闭限流功能。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0 ~ 2147483647。

默认值：0

enable_wal_shipping_compression

参数说明：在流式容灾模式下设置启动跨集群日志压缩功能。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

- 该参数仅作用于流式容灾中跨集群传输的一对walsender与walreceiver中，在主集群上配置。

取值范围：布尔型

- true 表示打开流式容灾跨集群日志压缩
- false 表示关闭流式容灾跨集群日志压缩

默认值：false

repl_auth_mode

参数说明：设置主备复制和备机重建的验证模式。

该参数属于SIGHUP类型参数，请参考表18-1中对应设置方法进行设置。

须知

- 如果主机上开启了UUID验证功能、且配置了非空字符串的repl_uuid验证码，那么备机也需要开启UUID验证功能、且配置相同的repl_uuid验证码，否则主备日志复制和备机重建请求将被主机拒绝。
- 该参数支持SIGHUP动态加载新值。修改之后不影响已建连的主备连接，对后续主备复制请求和主备重建请求生效。
- 支持Quorum、DCF协议下的备机重建验证；支持Quorum协议下的主备复制验证；不支持DCF协议下的主备复制验证。
- 不支持跨集群主、备之间的认证，包括Dorado主备集群和容灾主备集群。
- UUID验证功能主要为了防止主、备误连导致的数据串扰和污染，不是用于安全目的。
- 该参数不支持主、备间自动同步。

取值范围：枚举类型

- off：表示关闭UUID验证功能。
- default：表示关闭UUID验证功能。
- uuid：表示开启UUID验证功能。

默认值：default

repl_uuid

参数说明：设置用于主备UUID验证的UUID码。

该参数属于SIGHUP类型参数，请参考表18-1中对应设置方法进行设置。

须知

- 如果主机上开启了UUID验证功能、且配置了非空字符串的repl_uuid验证码，那么备机也需要开启UUID验证功能、且配置相同的repl_uuid验证码，否则主备日志复制和备机重建请求将被主机拒绝。
- 该参数支持SIGHUP动态加载新值。修改之后，不影响已建连的主备连接，对后续主备复制请求和主备重建请求生效。
- 支持Quorum、DCF协议下的备机重建验证；支持Quorum协议下的主备复制验证；不支持DCF协议下的主备复制验证。
- 不支持跨集群主、备之间的认证，包括Dorado主备集群和容灾主备集群。
- UUID验证功能主要为了防止主、备误连导致的数据串扰和污染，不是用于安全目的。
- 该参数不支持主、备间自动同步。

取值范围：字符串类型。长度0~63个字符，字母和数字的组合，大小写不敏感，内部统一转换为小写存储。空字符串表示不启用UUID验证功能。

默认值：空字符串

replconninfo1

参数说明：设置本端侦听和鉴权的第一个节点信息。集群安装成功后自动配置无需手动修改。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第一个节点信息。

默认值：DN侦听的第一个连接信息。

示例：

```
replconninfo1 = 'localhost=127.0.0.1 localport=XXXX localheartbeatport=XXXX localservice=XXXX
remotehost=127.0.0.1 remoteport=XXXX remoteheartbeatport=XXXX remoteservice=XXXX'
```

replconninfo2

参数说明：设置本端侦听和鉴权的第二个节点信息。集群安装成功后自动配置无需手动修改。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第二个节点信息。

默认值：DN侦听的第二个连接信息。

示例：

```
replconninfo2 = 'localhost=127.0.0.1 localport=XXXX localheartbeatport=XXXX localservice=XXXX
remotehost=127.0.0.1 remoteport=XXXX remoteheartbeatport=XXXX remoteservice=XXXX'
```

replconninfo3

参数说明：设置本端侦听和鉴权的第三个节点信息。集群安装成功后自动配置无需手动修改。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第三个节点信息。

默认值：DN侦听的第三个连接信息。

示例：

```
replconninfo3 = 'localhost=127.0.0.1 localport=XXXX localheartbeatport=XXXX localservice=XXXX
remotehost=127.0.0.1 remoteport=XXXX remoteheartbeatport=XXXX remoteservice=XXXX'
```

replconninfo4

参数说明：设置本端侦听和鉴权的第四个节点信息。集群安装成功后自动配置无需手动修改。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第四个节点信息。

默认值：DN侦听的第四个连接信息。

示例：


```
replconninfo4 = 'localhost=127.0.0.1 localport=XXXX localheartbeatport=XXXX localservice=XXXX
remotehost=127.0.0.1 remoteport=XXXX remoteheartbeatport=XXXX remoteservice=XXXX'
```

replconninfo5

参数说明：设置本端侦听和鉴权的第五个节点信息。集群安装成功后自动配置无需手动修改。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第五个节点信息。

默认值：DN侦听的第五个连接信息。

示例：

```
replconninfo5 = 'localhost=127.0.0.1 localport=XXXX localheartbeatport=XXXX localservice=XXXX
remotehost=127.0.0.1 remoteport=XXXX remoteheartbeatport=XXXX remoteservice=XXXX'
```

replconninfo6

参数说明：设置本端侦听和鉴权的第六个节点信息。集群安装成功后自动配置无需手动修改。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第六个节点信息。

默认值：DN侦听的第六个连接信息。

示例：

```
replconninfo6 = 'localhost=127.0.0.1 localport=XXXX localheartbeatport=XXXX localservice=XXXX
remotehost=127.0.0.1 remoteport=XXXX remoteheartbeatport=XXXX remoteservice=XXXX'
```

replconninfo7

参数说明：设置本端侦听和鉴权的第七个节点信息。集群安装成功后自动配置无需手动修改。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第七个节点信息。

默认值：DN侦听的第七个连接信息。

示例：

```
replconninfo7 = 'localhost=127.0.0.1 localport=XXXX localheartbeatport=XXXX localservice=XXXX
remotehost=127.0.0.1 remoteport=XXXX remoteheartbeatport=XXXX remoteservice=XXXX'
```

18.3.7.2 主服务器

synchronous_standby_names

参数说明：潜在同步复制的备机名称列表，每个名称用逗号分隔。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

- 当前连接的同步备机是列表中的第一个名称。如果当前同步备机失去连接，则它会立即更换下一个优先级更高的备机，并将此备机的名称放入列表中。
- 备机名称可以通过设置环境变量PGAPPNAME指定。

取值范围：字符串。当取值为*，表示匹配任意提供同步复制的备机名称。支持按如下格式配置：

- ANY *num_sync* (*standby_name* [, ...])
- [FIRST] *num_sync* (*standby_name* [, ...])
- *standby_name* [, ...]

说明

- 其中*num_sync*是事务提交前所需要等待回复的同步备的数量，*standby_name*是备机的名称，FIRST以及ANY指定从所列服务器中选取同步复制的备机的策略。
- ANY N (dn_instanceId1, dn_instanceId2,...)表示在括号内任选N个主机名称作为同步复制的备机名称列表。例如，ANY 1(dn_instanceId1, dn_instanceId2)表示在dn_instanceId1和dn_instanceId2中任选一个作为同步复制的备机名称。
- FIRST N (dn_instanceId1, dn_instanceId2,...)表示在括号内按出现顺序的先后作为优先级选择前N个主机名称作为同步复制的备机名称列表。例如，FIRST 1 (dn_instanceId1, dn_instanceId2)表示选择dn_instanceId1作为同步复制的备机名称。
- dn_instanceId1, dn_instanceId2,...和FIRST 1 (dn_instanceId1, dn_instanceId2,...)具有的含义相同。

若使用gs_guc工具设置该参数，需要如下设置：

```
gs_guc reload -Z datanode -N @NODE_NAME@ -D @DN_PATH@ -c "synchronous_standby_names='ANY NODE 1(dn_instanceId1, dn_instanceId2)'";
```

或者：

```
gs_guc reload -Z datanode -N @NODE_NAME@ -D @DN_PATH@ -c "synchronous_standby_names='ANY 1(AZ1, AZ2)'";
```

默认值： *

most_available_sync

参数说明：指定在备机同步失败时，是否阻塞主机。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示在备机同步失败时，不阻塞主机。
- off表示在备机同步失败时，阻塞主机。

默认值：off

enable_stream_replication

参数说明：控制主备是否进行数据和日志同步。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

- 此参数属于性能测试参数，用于测试带有DN备机和不带DN备机的性能参数。关闭参数后，不能进行切换、故障等异常场景测试。
- 此参数属于受控参数，不建议正常业务场景下关闭此参数。

取值范围：布尔型

- on表示打开主备同步。
- off表示关闭主备同步。

默认值：on

enable_mix_replication

参数说明：控制主备之间WAL日志及数据复制的方式。

该参数属于INTERNAL类型参数，默认值为off，不允许外部修改。

须知

此参数目前不允许正常业务场景下改变其值，即默认关闭WAL日志、数据页混合复制模式。

取值范围：布尔型

- on表示打开WAL日志、数据页混合复制模式。
- off表示关闭WAL日志、数据页混合复制模式。

默认值：off

vacuum_defer_cleanup_age

参数说明：指定VACUUM使用的事务数，VACUUM会延迟清除无效的行存表记录，延迟的事务个数通过vacuum_defer_cleanup_age进行设置。即VACUUM和VACUUM FULL操作不会立即清理刚刚被删除元组。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0～1000000，值为0表示不延迟。

默认值：0

data_replicate_buffer_size

参数说明：发送端与接收端传递数据页时，队列占用内存的大小。此参数会影响主备之间复制的缓冲大小。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，4096~1072693248，单位为KB。

默认值：128MB（即131072KB）

walsender_max_send_size

参数说明：设置主机端日志或数据发送缓冲区的大小。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，8~1048575，单位为KB。

默认值：8MB（即8192KB）

enable_data_replicate

参数说明：当数据库在数据导入行存表时，主机与备机的数据同步方式可以进行选择。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示导入数据行存表时主备数据采用数据页的方式进行同步。当replication_type参数为1时，不允许设置为on，如果此时用guc工具设置成on，会强制改为off。
- off表示导入数据行存表时主备数据采用日志（xLog）方式进行同步。

默认值：off

ha_module_debug

参数说明：用于查看数据复制时具体数据块的复制状态日志。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示日志中将打印数据复制时每个数据块的状态。
- off表示日志中不打印数据复制时每个数据块的状态。

默认值：off

catchup2normal_wait_time

参数说明：单同步备机情况下，控制备机数据追赶（catchup）阻塞主机的最长时间。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，范围-1~10000，单位为毫秒。

- -1表示主机阻塞直到备机数据追赶完成。
- 0表示备机数据追赶时始终不阻塞主机。
- 其余值表示备机数据追赶时阻塞主机的最长时间。例如，取值5000，表示当备机数据追赶完成时间还剩5s时，阻塞主机等待其完成。

默认值：-1

hadr_recovery_time_target

参数说明：在流式容灾模式下设置hadr_recovery_time_target能够让备数据库实例完成日志写入和回放。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~3600（秒）

0是指不开启日志流控，1~3600是指备机能够在hadr_recovery_time_target时间内完成日志的写入和回放，可以保证主数据库实例与备数据库实例切换时能够在hadr_recovery_time_target秒完成日志写入和回放，保证备数据库实例能够快速升主。hadr_recovery_time_target设置时间过小会影响主机的性能，设置过大会失去流控效果。

默认值：60（金融版（数据计算型））

hadr_recovery_point_target

参数说明：在流式容灾模式下设置hadr_recovery_point_target能够让备数据库实例完成日志刷盘的rpo时间。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~3600（秒）

0是指不开启日志流控，1~3600是指备机能够在hadr_recovery_point_target时间内完成日志的刷盘，可以保证主数据库实例与备数据库实例切换时日志差距能够在hadr_recovery_point_target秒内，保障备数据库实例升主日志量。hadr_recovery_point_target设置时间过小会影响主机的性能，设置过大会失去流控效果。

默认值：10（金融版（数据计算型））

hadr_super_user_record_path

参数说明：该参数为流式异地容灾参数，表示备集群中hadr_disaster用户的加密文件存放路径。该参数属于SIGHUP类型参数，请参考**表18-2**中方式对应设置方法进行设置。

修改建议：由流式容灾密码传递工具自动设置，不需要用户手动添加。

取值范围：字符串

默认值：NULL

check_sync_standby

参数说明：打开备机检查开关，主备场景下配置了正确的synchronous_standby_names参数后，当同步备故障时，主机写业务直接报错写失败。该参数属于USERSET类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：on/off

- on表示开启备机检查。
- off表示关闭备机检查。

默认值：off

说明

- 该参数不支持在job work和自治事务中同步，有可能导致检查不生效。
- 若指定用户或session中未设置备机检查，开启强同步提交模式下备机故障，执行一个表的写操作会导致另一个用户或session中的同一个表的查询hang，此时需要备机恢复或者手动终止hang住的客户端。
- 不支持非写操作中触发写日志的场景中（vacuum analyze，gs_clean等）开启备机检查开关。若备机不满足同步备配置，则该场景会导致业务hang，需要手动终止。

18.3.7.3 备服务器

hot_standby

参数说明：设置是否允许备机在恢复到minrecovery点后接受连接和查询。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

- 如果此参数设置为on，**wal_level**级别必须设置为hot_standby或以上，否则将导致数据库无法启动。
- 在分布式环境中，因为会对双机其他一些功能产生影响，hot_standby参数不能设置成off。
- 如果hot_standby参数曾经被关闭，且wal_level参数曾被设置低于hot_standby等级，那么，再次打开hot_standby参数之前，为了确保主备环境下备机上待回放的日志都可以支持备机查询功能，需要进行如下操作：
  1. 将主、备的wal_level参数调整到hot_standby等级或以上，并重启实例生效。
  2. 在主机上执行checkpoint操作，并通过查询pg_stat_get_wal_senders()系统函数，确认各个备机的receiver_replay_location追上主机当前的sender_flush_location，保证wal_level的调整同步到备机并生效，且备机不需要再回放之前低等级的日志。
  3. 将主、备的hot_standby参数打开（设为on），并重启实例生效。

取值范围：布尔型

- on表示允许备机在恢复到minrecovery点后接受连接和查询。
- off表示不允许备机在恢复到minrecovery点后接受连接和查询。

默认值：on

max_standby_archive_delay

参数说明：当开启双机热备模式时，如果备机正处理归档WAL日志数据，这时进行查询就会产生冲突，此参数就是设置备机取消查询之前所等待的时间。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

-1表示允许备机一直等待冲突的查询完成。

取值范围：整型，范围：-1~2147483647，单位为毫秒。

默认值：3s（即3000ms）

max_standby_streaming_delay

参数说明：当开启双机热备模式时，如果备机正通过流复制接收WAL日志数据，这时进行查询就会产生冲突，这个参数就是设置备机取消查询之前所等待的时间。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

-1表示允许备机一直等待冲突的查询完成。

取值范围：整型（毫秒），范围：-1~2147483647

默认值：3s（即3000ms）

wal_receiver_status_interval

参数说明：设置WAL日志接收进程的状态通知给主机的最大时间间隔。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

当该参数设置为0时，表示关闭备机向主机反馈日志接收位置等信息，可能会导致主机事务提交阻塞、switchover操作失败等异常现象。正常业务场景，不建议将该参数设置为0。

取值范围：整型，0 ~ 2147483，单位为秒。

默认值：5s

hot_standby_feedback

参数说明：设置是否允许将备机上执行查询的结果反馈给主机，这可以避免查询冲突。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许将备机上执行查询的结果反馈给主机。
- off表示不允许将备机上执行查询的结果反馈给主机。

默认值：off

wal_receiver_timeout

参数说明：设置从主机接收数据的最大等待时间。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型， 0 ~ 2147483647，单位为毫秒。

默认值：6s（即6000ms）

wal_receiver_connect_timeout

参数说明：设置连接主机的最大等待超时时间。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型， 0 ~ 2147483，单位为秒。

默认值：2s

wal_receiver_connect_retries

参数说明：设置连接主机的最大尝试次数。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型， 1~ 2147483647。

默认值：1

wal_receiver_buffer_size

参数说明：备机接收xLog存放到内存缓冲区的大小。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，4096~1047552，单位为KB。

默认值：64MB（即65536KB）

primary_slotname

参数说明：设置备机对应主机的slot name，用于主备校验，与wal日志删除机制。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符型

默认值：空字符串

enable_redo_atomic_operation

参数说明：开启并行回放，回放线程更新本线程的LSN值时，使用原子操作进行更新，还是使用spinlock加锁进行更新。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用原子操作进行更新。
- off表示使用spinlock加锁进行更新。

默认值：on

18.3.8 查询规划

介绍查询优化器方法配置、开销常量、规划算法以及一些配置参数。

说明

- 优化器中涉及的两个参数：
  - INT_MAX数据类型INT的最大值，其值为2147483647。
  - DBL_MAX数据类型FLOAT的最大值。
- 全局设置查询规划相关参数除了客户业务外也会对数据库自身运维和监控业务造成影响，如WDR报告生成、扩容、重分布、数据导入导出等。

18.3.8.1 优化器方法配置

这些配置参数提供了影响查询优化器选择查询规划的原始方法。如果优化器为特定的查询选择的缺省规划并不是最优的，可以通过使用这些配置参数强制优化器选择一个不同的规划来临时解决这个问题。更好的方法包括调节优化器开销常量、手动运行ANALYZE、增加配置参数default_statistics_target的值或使用ALTER TABLE SET STATISTICS为指定列增加收集的统计信息。

enable_bitmapscan

参数说明：控制优化器对位图扫描规划类型的使用。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

force_bitmapand

参数说明：控制优化器强制使用bitmapand规划类型的使用。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：off

enable_hashagg

参数说明：控制优化器对Hash聚集规划类型的使用。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。

- off表示不使用。

默认值：on

enable_hashjoin

参数说明：控制优化器对Hash连接规划类型的使用。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

enable_indexscan

参数说明：控制优化器对索引扫描规划类型的使用。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

enable_indexonlyscan

参数说明：控制优化器对仅索引扫描规划类型的使用。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

enable_material

参数说明：控制优化器对实体化的使用。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

enable_mergejoin

参数说明：控制优化器对融合连接规划类型的使用。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：off

enable_nestloop

参数说明：控制优化器对内表全表扫描嵌套循环连接规划类型的使用。关闭这个变量就会让优化器在存在其他方法的时候优先选择其他方法。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：off

enable_index_nestloop

参数说明：控制优化器对内表参数化索引扫描嵌套循环连接规划类型的使用。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

enable_seqscan

参数说明：控制优化器对顺序扫描规划类型的使用。关闭这个变量会让优化器在存在其他方法的时候优先选择其他方法。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

enable_sort

参数说明：控制优化器使用的排序步骤。关闭这个变量可以让优化器在存在其他方法的时候优先选择其他方法。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

enable_tidscan

参数说明：控制优化器对TID扫描规划类型的使用。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

enable_kill_query

参数说明：CASCADE模式删除用户时，会删除此用户拥有的所有对象。此参数标识是否允许在删除用户的时候，取消锁定此用户所属对象的query。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许取消锁定。
- off表示不允许取消锁定。

默认值：off

enable_stream_concurrent_update

参数说明：控制优化器在并发更新场景下对stream的使用，该参数受限于**enable_stream_operator**参数。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许优化器对update语句生成stream计划。
- off表示优化器对update语句仅能生成非stream计划。

默认值：on

enable_stream_operator

参数说明：控制优化器对stream的使用。当enable_stream_operator参数关闭时，会有大量关于计划不能下推的日志记录到日志文件中。如果用户不需要这些日志内容，建议用户在enable_stream_operator参数关闭时，也同时关闭enable_unshipping_log参数。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：

- 独立部署：off
- 金融版（标准型）：on
- 企业版：on
- 金融版（数据计算型）：on

enable_stream_recursive

参数说明：控制是否将with-recursive关联查询下推DN分布式执行。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示支持使用with-recursive关联查询下推DN分布式执行。
- off表示不支持使用with_recursive下推。

默认值：on

max_recursive_times

参数说明：控制with recursive的最大迭代次数。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0～2147483647。

默认值：200

enable_broadcast

参数说明：控制优化器对stream代价估算时对broadcast分布方式的使用。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

enable_change_hjcost

参数说明：控制优化器在Hash Join代价估算路径选择时，是否使用将内表运行时代价排除在Hash Join节点运行时代价外的估算方式。如果使用，则有利于选择条数少，但运行代价大的表做内表。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：off

best_agg_plan

参数说明：对于stream下的Agg操作，优化器会生成三种计划：

1. hashagg+gather(redistribute)+hashagg。
2. redistribute+hashagg(+gather)。
3. hashagg+redistribute+hashagg(+gather)。

本参数用于控制优化器生成哪种hashagg的计划。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：0，1，2，3

- 取值为1时，强制生成第一种计划。
- 取值为2时，如果group by列可以重分布，强制生成第二种计划，否则生成第一种计划。
- 取值为3时，如果group by列可以重分布，强制生成第三种计划，否则生成第一种计划。
- 取值为0时，优化器会根据以上三种计划的估算cost选择最优的一种计划生成。

默认值：0

agg_redistribute_enhancement

参数说明：当进行Agg操作时，如果包含多个group by列且均不为分布列，进行重分布时会选择某一group by列进行重分布。本参数控制选择重分布列的策略。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示会选择估算distinct值最多的一个可重分布列作为重分布列。
- off表示会选择第一个可重分布列为重分布列。

默认值：off

enable_absolute_tablespace

参数说明：控制表空间是否可以使用绝对路径。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示可以使用绝对路径。
- off表示不可以使用绝对路径。

默认值：on

enable_valuepartition_pruning

参数说明：是否对DFS分区表进行静态/动态优化。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示对DFS分区表进行静态/动态优化。
- off表示不对DFS分区表进行静态/动态优化。

默认值：on

expected_computing_nodegroup

参数说明：标识选定的计算Node Group模式或目标计算Node Group。Node Group目前为内部用机制，用户无需设置。

共4种计算Node Group模式，用于关联操作和聚集操作时选定计算Node Group。在每一种模式中，优化器有针对性地选定几个候选计算Node Group，然后根据代价，从中为当前算子挑选最佳计算Node Group。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

- optimal：候选计算Node Group列表包含算子操作对象所在的Node Group和由当前用户具有COMPUTE权限的所有Node Group包含的所有DN构成的Node Group
- query：候选计算Node Group列表包含算子操作对象所在的Node Group和由当前查询涉及的所有基表所在Node Group包含的所有DN构成的Node Group
- Node Group名（**enable_nodegroup_debug**被设置为off）：候选计算Node Group列表包含算子操作对象所在的Node Group和该指定的Node Group
- Node Group名（**enable_nodegroup_debug**被设置为on）：候选计算Node Group为指定的Node Group

默认值：query

enable_nodegroup_debug

参数说明：控制优化器在多Node Group环境下，是否使用强制弹性计算。Node Group目前为内部用机制，用户无需设置。

该参数只在**expected_computing_nodegroup**被设置为具体Node Group时生效。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示强制将计算弹性到expected_computing_nodegroup所指定的Node Group进行计算。
- off表示不强制使用某个Node Group进行计算。

默认值：off

stream_multiple

参数说明：设置优化器计算Stream算子的开销时的加权。

在原代价模型的基础上，最终Stream代价将被乘以此加权参数。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

此参数仅对Redistribute和Broadcast类型的Stream有效。

取值范围：浮点型，0～DBL_MAX。

默认值：1

qrw_inlist2join_optmode

参数说明：控制是否使用inlist-to-join查询重写。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

- disable：关闭inlist2join查询重写。
- cost_base：基于代价的inlist2join查询重写。
- rule_base：基于规则的inlist2join查询重写，即强制使用inlist2join查询重写。
- 任意正整数：inlist2join查询重写阈值，即in子句内的list内元素个数大于该阈值，进行inlist2join查询重写。

默认值：cost_base

skew_option

参数说明：控制是否使用优化策略。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：枚举类型

- off：关闭策略。
- normal：采用激进策略。对于不确定是否出现倾斜的场景，认为存在倾斜，并进行相应优化。
- lazy：采用保守策略。对于不确定是否出现倾斜场景，认为不存在倾斜，不进行优化。

默认值：normal

cost_weight_index

参数说明：设置index_scan的代价权重。

- 设置为1表示不调整；
- 设置小于1时，index_scan的代价会降低，更容易被优化器选中；
- 设置大于1时，index_scan的代价会增加。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：浮点型，1e-10~1e+10。

默认值：1

default_limit_rows

参数说明：设置生成genericplan的缺省limit估算行数。此参数设置为正数时意为直接将设置的值作为估算limit的行数；为正数小数时，自动取整；为负数时代表使用百分比的形式设置默认的估算值，负数转换为默认百分比，即-5代表5%。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：浮点型，-100~DBL_MAX。

默认值：-10

enforce_a_behavior

参数说明：控制正则表达式的规则匹配模式。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示正则表达式采用A格式的匹配规则。
- off表示正则表达式采用POSIX格式的匹配规则。

默认值：on

check_implicit_conversions

参数说明：控制检查查询中有隐式类型转换的索引列是否会生成候选索引路径。关于该参数的使用场景请参见《开发指南》中“SQL调优指南 > 检查隐式转换的性能问题”章节。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示检查查询中有隐式类型转换的索引列是否会生成候选索引路径。
- off表示不进行相关检查。

默认值：off

须知

将该参数设置为on时，需要同时将参数enable_fast_query_shipping设置为off，检查索引列的隐式数据类型转换的识别机制才会生效。

18.3.8.2 优化器开销常量

介绍优化器开销常量。这里描述的开销可以按照任意标准度量。只关心其相对值，因此以相同的系数缩放它们将不会对优化器的选择产生任何影响。缺省时，它们以抓取顺序页的开销为基本单位。也就是说将seq_page_cost设为1.0，同时其他开销参数以它为基准设置。也可以使用其他基准，比如以毫秒计的实际执行时间。

seq_page_cost

参数说明：设置优化器计算一次顺序磁盘页面抓取的开销。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：浮点型，0～DBL_MAX。

默认值：1

random_page_cost

参数说明：设置优化器计算一次非顺序抓取磁盘页面的开销。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

虽然服务器允许将random_page_cost设置的比seq_page_cost小，但是物理上实际不受影响。如果所有数据库都位于随机访问内存中时，两者设置为相等很合理。因为在此种情况下，非顺序抓取页并没有副作用。同样，在缓冲率很高的数据库上，应该相对于CPU参数同时降低这两个值，因为获取内存中的页要比通常情况下开销小很多。

取值范围：浮点型，0～DBL_MAX。

默认值：4

说明

- 对于特别表空间中的表和索引，可以通过设置同名的表空间的参数来覆盖这个值。
- 相对于seq_page_cost，减少这个值将导致系统更倾向于使用索引扫描，而增加这个值使得索引扫描开销比较高。可以通过同时增加或减少这两个值来调整磁盘I/O相对于CPU的开销。

cpu_tuple_cost

参数说明：设置优化器计算在一次查询中处理每一行数据的开销。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：浮点型，0～DBL_MAX。

默认值：0.01

cpu_index_tuple_cost

参数说明：设置优化器计算在一次索引扫描中处理每条索引的开销。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：浮点型，0～DBL_MAX。

默认值：0.005

cpu_operator_cost

参数说明：设置优化器计算一次查询中执行一个操作符或函数的开销。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：浮点型，0～DBL_MAX。

默认值：0.0025

effective_cache_size

参数说明：设置优化器在一次单一的查询中可用的磁盘缓冲区的有效大小。

设置这个参数，要考虑GaussDB的共享缓冲区以及内核的磁盘缓冲区，还要考虑预计的在不同表之间的并发查询数目，因为它们将共享可用的空间。

这个参数对GaussDB实际运行时分配的共享内存大小没有影响，它只用于计划生成阶段的估算。该数值是用磁盘页来计算的，通常每个页面是8192字节。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1～2147483647，单位为8KB。

默认值：

独立部署：

CN：2GB（60核CPU/480G内存）；1GB（32核CPU/256G内存，16核CPU/128G内存）；512MB（8核CPU/64G内存）；256MB（4核CPU/32G内存）；128MB（4核CPU/16G内存）

DN：70GB（60核CPU/480G内存）；38GB（32核CPU/256G内存）；20GB（16核CPU/128G内存）；8GB（8核CPU/64G内存）；4GB（4核CPU/32G内存）；2GB（4核CPU/16G内存）

金融版（标准型）：

CN：1GB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，72核CPU/576G内存，64核CPU/512G内存，60核CPU/480G内存）；512MB（32核CPU/256G内存，16核CPU/128G内存）；256MB（8核CPU/64G内存）

DN：70GB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存）；50GB（96核CPU/768G内存）；40GB（80核CPU/640G内存，72核CPU/576G内存）；35GB（64核CPU/512G内存）；30GB（60核CPU/480G内存）；15GB（32核CPU/256G内存），8GB（16核CPU/128G内存）；4GB（8核CPU/64G内存）

企业版：

CN：1GB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，80核CPU/512G内存，72核CPU/576G内存，64核CPU/512G内存，60核CPU/480G内存）；512MB（32核CPU/256G内存，16核CPU/128G内存）；256MB（8核CPU/64G内存）

DN：50GB（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存）；38GB（96核CPU/768G内存）；32GB（80核CPU/640G内存）；28GB（72核CPU/576G内存）；25GB（80核CPU/512G内存，64核CPU/512G内存）；24GB（60核CPU/480G内存）；12GB（32核CPU/256G内存），5GB（16核CPU/128G内存）；3GB（8核CPU/64G内存）

金融版（数据计算型）：

CN：1GB（96核CPU/768G内存）；512MB（72核CPU/576G内存，64核CPU/512G内存）；256MB（32核CPU/256G内存）

DN：25GB（96核CPU/768G内存）；20GB（72核CPU/576G内存）；15GB（64核CPU/512G内存）；5GB（32核CPU/256G内存）

设置建议：

较大的数值使优化器倾向于选择索引扫描，较小的数值使优化器倾向于选择全表扫描。一般情况下，可以设为shared_buffers大小的1/2，较为激进地，可以设为shared_buffers大小的3/4。

allocate_mem_cost

参数说明：设置优化器计算Hash Join创建Hash表开辟内存空间所需的开销，供Hash join估算不准时调优使用。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：浮点型，0～DBL_MAX。

默认值：0

18.3.8.3 基因查询优化器

介绍基因查询优化器相关的参数。基因查询优化器（GEQO）是一种启发式的查询规划算法。这个算法减少了对复杂查询规划的时间，而且生成规划的开销有时也小于正常的详尽的查询算法。

geqo

参数说明：控制基因查询优化的使用。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

通常情况下在执行过程中不要关闭，geqo_threshold变量提供了更精细的控制GEQO的方法。

若该参数通过执行gs_guc reload修改时，如果当前节点上的某个session的连接不是来自于客户端，而是来自于该节点所属集群上的其他节点，那么执行gs_guc reload后该参数在该session上不会立即生效，需与连接节点断开连接后重新连接才会生效。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

geqo_threshold

参数说明：如果执行语句的数量超过设计的FROM的项数，则会使用基因查询优化来执行查询。

该参数属于USERSET类型参数，请参考表18-1中对应设置方法进行设置。

须知

- 对于简单的查询，通常用详尽搜索方法，当涉及多个表的查询的时候，用GEQO可以更好的管理查询。
- 一个FULL OUTER JOIN构造仅作为一个FROM项。

取值范围：整型，2～2147483647。

默认值：12

geqo_effort

参数说明：控制GEQO在规划时间和规划质量之间的平衡。

该参数属于USERSET类型参数，请参考表18-1中对应设置方法进行设置。

须知

geqo_effort实际上并没有直接做任何事情，只是用于计算其他影响GEQO的变量的默认值。如果愿意，可以手工设置其他参数。

取值范围：整型，1～10。

须知

比默认值大的数值增加了查询规划的时间，但是也增加了选中有效查询的几率。

默认值：5

geqo_pool_size

参数说明：控制GEQO使用池的大小，也就是基因全体中的个体数量。

该参数属于USERSET类型参数，请参考表18-1中对应设置方法进行设置。

取值范围：整型，0～2147483647。

须知

至少是2，且有用的值一般在100到1000之间。设置为0，表示使用系统自适应方式，GaussDB会基于geqo_effort和表的个数选取合适的值。

默认值：0

geqo_generations

参数说明：控制GEQO使用的算法的迭代次数。

该参数属于USERSET类型参数，请参考表18-1中对应设置方法进行设置。

取值范围：整型，0～2147483647。

须知

必须至少是1，且有用的值介于100和1000之间。如果设置为0，则基于geqo_pool_size选取合适的值。

默认值：0

geqo_selection_bias

参数说明：控制GEQO的选择性偏好，即就是一个种群中的选择性压力。

该参数属于USERSET类型参数，请参考表18-1中对应设置方法进行设置。

取值范围：浮点型，1.5～2.0。

默认值：2

geqo_seed

参数说明：控制GEQO使用的随机数生产器的初始化值，用来从顺序连接在一起的查询空间中查找随机路径。

该参数属于USERSET类型参数，请参考表18-1中对应设置方法进行设置。

取值范围：浮点型，0.0～1.0。

须知

不同的值会改变搜索的连接路径，从而影响了所找路径的优劣。

默认值：0

18.3.8.4 其他优化器选项

enable_fast_query_shipping

参数说明：控制查询优化器是否使用分布式框架。

该参数属于USERSET类型参数，请参考表18-1中对应设置方法进行设置。

取值范围：布尔型

- on表示执行计划在CN和DN上各自生成。
- off表示使用分布式框架，即执行计划在CN上生成，然后发送到DN中执行。

默认值：on

enable_trigger_shipping

参数说明：控制触发器场景是否允许将触发器下推到DN执行。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许将触发器下推到DN执行。
- off表示不允许将触发器下推到DN执行，在CN执行。

默认值：on

enable_remotejoin

参数说明：设置是否允许连接操作计划下推到DN执行。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许连接操作计划下推到DN执行。
- off表示不允许连接操作计划下推到DN执行。

默认值：on

enable_remotegroup

参数说明：设置是否允许group by与aggregates执行计划下推到DN执行。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许group by与aggregates执行计划下推到DN执行。
- off表示不允许group by与aggregates执行计划下推到DN执行。

默认值：on

enable_remotelimit

参数说明：设置是否允许LIMIT子句执行计划下推到DN执行。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许LIMIT子句执行计划下推到DN执行。
- off表示不允许LIMIT子句执行计划下推到DN执行。

默认值：on

enable_remotesort

参数说明：设置是否允许ORDER BY子句操作计划下推到DN执行。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许ORDER BY子句操作计划下推到DN执行。
- off表示不允许ORDER BY子句操作计划下推到DN执行。

默认值：on

enable_csqual_pushdown

参数说明：进行查询时，是否要将过滤条件下推，进行Rough Check。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示进行查询时，要将过滤条件下推，进行Rough Check。
- off表示进行查询时，不要将过滤条件下推，进行Rough Check。

默认值：on

explain_dna_file

参数说明：指定**explain_perf_mode**为run，导出的csv信息的目标文件。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

这个参数的取值必须是绝对路径加上.csv格式的文件名。

取值范围：字符串

默认值：空

analysis_options

参数说明：通过开启对应选项中所对应的功能选项使用相应的定位功能，包括数据校验，性能统计等，参见取值范围中的选项说明。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

- HASH_CONFLICT表示在DN进程的pg_log目录中的log日志中显示hash表的统计信息，包括hash表大小，hash链长，hash冲突情况。
- STREAM_DATA_CHECK表示对网络传输前后的数据进行CRC校验。

默认值：ALL,on(),off(HASH_CONFLICT,STREAM_DATA_CHECK)，不开启任何定位功能。

explain_perf_mode

参数说明：此参数用来指定explain的显示格式。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：normal、pretty、summary、run

- normal：代表使用默认的打印格式。
- pretty：代表使用GaussDB改进后的新显示格式。新的格式层次清晰，计划包含了plan node id，性能分析简单直接。
- summary：是在pretty的基础上增加了对打印信息的分析。
- run：在summary的基础上，将统计的信息输出到csv格式的文件中，以便于进一步分析。

默认值：pretty

说明

pretty模式当前只支持包含stream算子的计划，不支持下发语句到DN节点的计划。因此显示格式会受enable_stream_operator参数影响，当enable_stream_operator设置为off时无法生成包含stream算子的计划。

cost_param

参数说明：该参数用于控制在特定的客户场景中，使用不同的估算方法使得估算值与真实值更接近。此参数可以同时控制多种方法，与某一方法对应的位做与操作，不为0表示该方法被选择。

- 当cost_param & 1不为0，表示对于求不等值连接选择率时选择一种改良机制，此方法在自连接（两个相同的表之间连接）的估算中更加准确。当前版本已弃用cost_param & 1不为0时的路径，默认选择更优的估算公式。
- 当cost_param & 2不为0，表示求多个过滤条件（Filter）的选择率时，选择最小的作为总的选择率，而非两者乘积，此方法在过滤条件的列之间关联性较强时估算更加准确。
- 当cost_param & 4不为0，表示在进行stream节点估算时，选用调试模型，该模型不推荐用户使用。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0～2147483647

默认值：0

enable_partitionwise

参数说明：分区表连接操作是否选择智能算法。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示选择智能算法。
- off表示不选择智能算法。

默认值：off

enable_fast_numeric

参数说明：标识是否开启Numeric类型数据运算优化。Numeric数据运算是较为耗时的操作之一，通过将Numeric转化为int64/int128类型，提高Numeric运算的性能。

该参数属于SUSET类型参数，请参考表18-1中对应设置方法进行设置。

取值范围：布尔型

- on/true表示开启Numeric优化。
- off/false表示关闭Numeric优化。

默认值：on

rewrite_rule

参数说明：标识已开启的可选查询重写规则。有部分查询重写规则是可选的，开启它们并不能总是对查询效率有提升效果。在特定的客户场景中，通过此GUC参数对查询重写规则进行设置，使得查询效率最优。

此参数可以控制查询重写规则的组合，比如有多个重写规则：rule1、rule2、rule3、rule4。可以设置：

```
set rewrite_rule=rule1;          --启用查询重写规则rule1
set rewrite_rule=rule2,rule3;    --启用查询重写规则rule2和rule3
set rewrite_rule=none;           --关闭所有可选查询重写规则
```

该参数属于USERSET类型参数，请参考表18-1中对应设置方法进行设置。

取值范围：字符串

- none：不使用任何可选查询重写规则。
- lazyagg：使用Lazy Agg查询重写规则（消除子查询中的聚集运算）。
- magicset：使用Magic Set查询重写规则（将带有聚集算子的子查询提前和主查询进行关联，减少子链接的重复扫描）。
- partialpush：使用Partial Push查询重写规则（对于不可下推的语句，下推部分子查询到DN执行，剩余不下推的部分在CN执行）。
- uniquecheck：使用Unique Check查询重写规则（提升目标列中无agg的子查询语句，在执行时检查返回行数是否为1行）。
- disablerep：使用Disable Replicate查询重写规则（由于复制表提升之后可能劣化，开启此规则之后，禁止复制表的子查询提升）。
- intargetlist：使用In Target List查询重写规则（提升目标列中的子查询）。
- predpushnormal：使用Predicate Push查询重写规则（下推谓词条件到子查询中，可能会添加BROADCAST算子来支持分布式执行）。
- predpushforce：使用Predicate Push查询重写规则（下推谓词条件到子查询中，尽可能的利用索引加速）。
- predpush：在predpushnormal和predpushforce中根据代价选择最优计划。
- disable_pullup_expr_sublink：禁止优化器将expr_sublink类型的子连接提升，关于sublink的分类和提升原理详见《开发指南》中“SQL调优指南 > 典型SQL调优点 > 子查询调优”章节。
- disable_pullup_not_in_sublink：禁止优化器对not in相关的子链接进行提升，关于sublink的分类和提升原理详见《开发指南》中“SQL调优指南 > 典型SQL调优点 > 子查询调优”章节。
- enable_sublink_pullup_rownum：允许SQL语句中有ROWNUM伪列的情况下优化器进行子链接提升。

默认值：magicset

enable_pbe_optimization

参数说明：设置优化器是否对以PBE（Parse Bind Execute）形式执行的语句进行查询计划的优化。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型。

- on表示优化器将优化PBE语句的查询计划。
- off表示不使用优化。

默认值：on

enable_light_proxy

参数说明：设置优化器是否对简单查询在CN上优化执行，应用端和内核端字符集不匹配时，该参数不生效，建议建库时将字符集设为UTF8。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型。

- on表示优化器将优化CN上简单查询的执行。
- off表示不使用优化。

默认值：on

enable_global_plancache

参数说明：设置是否对PBE查询的执行计划进行缓存共享，开启该功能可以节省高并发下CN和DN上的内存使用。且该参数必须在CN和DN上一致，否则会出现CN下发给DN的报文不匹配从而报错等问题。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

在打开enable_global_plancache的情况下，为保证GPC生效，默认local_syscache_threshold不小于16MB。即如当前local_syscache_threshold小于16MB，则设置为16MB，如大于16MB，则不改变。

取值范围：布尔型。

- on表示对PBE查询的执行计划进行缓存共享。
- off表示不共享。

默认值：off

gpc_clean_timeout

参数说明：开启enable_global_plancache的情况下，如果共享计划列表里的计划超过gpc_clean_timeout的时间没有被使用，则会被清理掉。本参数用于控制没有使用的共享计划的保留时间。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，300~86400，单位为秒

默认值：1800，即30min

enable_gpc_grayrelease_mode

参数说明：分布式集群下，开启GPC需要重启集群。如果需要在不重启集群的情况下开启GPC，需要用enable_gpc_grayrelease_mode控制滚动开启GPC或关闭GPC。

分布式集群上操作方式。

开启GPC：

1、在所有DN节点开启enable_gpc_grayrelease_mode。

2、在所有CN节点开启enable_gpc_grayrelease_mode。

3、开启GPC参数，由于GPC是POSTMASTER参数，需要先reload参数，之后轮询kill节点，使重新拉起的节点上GPC生效。

关闭GPC：

1、首先确定enable_gpc_grayrelease_mode 是on的状态，reload关闭GPC参数，然后轮询kill节点，使重新拉起的节点上GPC生效。

2、在所有CN节点关闭enable_gpc_grayrelease_mode。

3、在所有DN节点关闭enable_gpc_grayrelease_mode。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型。

- on
- off

默认值：off

enable_opfusion

参数说明：控制是否对简单查询进行查询优化。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

该开关是优化DN的查询性能，可以设置max_datanode_for_plan查看一个查询DN的执行计划，如果DN的执行计划中带有[Bypass]标识则代表该查询在该DN可以查询优化。

简单查询限制如下：

- 只支持indexscan和indexonlyscan，且全部WHERE语句的过滤条件都在索引上。
- 只支持单表增删改查，不支持join、using。
- 只支持行存表，不支持分区表，表不支持有触发器。
- 不支持active sql、QPS等信息统计特性。
- 不支持正在扩容和缩容的表。
- 不支持查询或者修改系统列。

- 只支持简单SELECT语句，例如：
  ```
  SELECT c3 FROM t1 WHERE c1 = ? and c2 =10;
  ```
  仅可以查询目标表的列，c1和c2列为索引列，后边可以是常量或者参数，可以使用 for update。
- 只支持简单INSERT语句，例如：
  ```
  INSERT INTO t1 VALUES (?,10,?);
  ```
  仅支持一个VALUES，VALUES里面的类型可以是常量和参数，不支持returning。
- 只支持简单DELETE语句，例如：
  ```
  DELETE FROM t1 WHERE c1 = ? and c2 = 10;
  ```
  c1和c2列为索引列，后边可以是常量或者参数。
- 只支持简单UPDATE语句，例如：
  ```
  UPDATE t1 SET c3 = c3+? WHERE c1 = ? and c2 = 10;
  ```
  c3列修改的值可以是常量和参数，也可以是一个简单的表达式，c1和c2列为索引列，后边可以是常量或者参数。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：on

sql_beta_feature

参数说明：标识开启的可选SQL引擎Beta特性，其中包括对行数估算、查询等价估算等优化。开启它们可以对特定的场景进行优化，但也可能会导致部分没有被测试覆盖的场景发生性能劣化。在特定的客户场景中，通过此GUC参数对查询重写规则进行设置，使得查询效率最优。

此参数可以控制SQL引擎Beta特性的组合，比如有多个Beta特性：feature1、feature2、feature3、feature4。可以设置：

```
set sql_beta_feature=feature1;          --启用SQL引擎Beta特性feature1
set sql_beta_feature=feature2,feature3;    --启用SQL引擎Beta特性feature2和feature3
set sql_beta_feature=none;          --关闭所有可选SQL引擎Beta特性
```

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

- none：不使用任何Beta优化器特性。
- sel_semi_poisson：使用泊松分布对等值的半连接和反连接选择率进行校准。
- sel_expr_instr：使用字符串匹配的行数估算方法对instr(col, 'const') > 0, = 0, = 1进行更准确的估算。
- param_path_gen：生成更多可能的参数化路径。
- rand_cost_opt：对小数据量表的随机读取代价进行优化。
- param_path_opt：利用表的膨胀系数优化索引analyze信息。
- page_est_opt：优化对表索引analyze信息的relpages估算。
- no_unique_index_first：关闭主键索引扫描路径优先的优化。
- join_sel_with_cast_func：估算join行数的时候支持类型转换函数。
- canonical_pathkey：正则化pathkey生成置前（pathkey：标记数据有序性键值的集合）。

- index_cost_with_leaf_pages_only：估算索引代价时考虑索引叶子节点。
- a_style_coerce：开启Decode类型转换规则兼容O，详见《开发指南》中“SQL参考 > 类型转换 > UNION，CASE和相关构造”章节的“对于case，在ORA兼容模式下的处理”。
- plpgsql_stream_fetchall：在存储过程中for loop或cursor上执行的sql走stream场景下，开启获取所有tuple结果。
- predpush_same_level：开启predpush hint控制同层参数化路径的功能。
- disable_bitmap_cost_with_lossy_pages：关闭bitmap路径代价中对lossy pages代价的计算。

默认值:
"sel_semi_poisson,sel_expr_instr,rand_cost_opt,param_path_opt,page_est_opt"

table_skewness_warning_threshold

参数说明：设置用于表倾斜告警的阈值。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：浮点型，0～1

默认值：1

table_skewness_warning_rows

参数说明：设置用于表倾斜告警的行数。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0～2147483647

默认值：100000

enable_global_stats

参数说明：标识当前统计信息模式，区别采用全局统计信息收集模式还是单节点统计信息收集模式，默认创建为采用全局统计信息模式。当关闭该参数时，则默认收集集群第一个节点的统计信息，此时可能会影响生成查询计划的质量，但信息收集性能较优，建议谨慎设置。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on/true表示全局统计信息。
- off/false表示单DN统计信息。

默认值：on

default_statistics_target

参数说明：为没有用ALTER TABLE SET STATISTICS设置字段目标的表设置缺省统计目标。影响收集统计信息时的采样行数。

此参数设置为正数时，代表统计信息直方图预期桶的数量，统计信息采样行数为default_statistics_target * 300；为负数时，代表使用百分比的形式设置统计目标，负

数转换为对应的百分比，即-5代表5%，采样行数为总行数 * 5%。此参数仅影响统计信息的目标采样行数，实际采样行数还会受到内存参数**work_mem**的限制。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，-100～10000。

须知

- 比默认值大的正数数值增加了ANALYZE所需的时间，但是可能会改善优化器的估计质量。
- 调整此参数可能存在性能劣化的风险，如果某个查询劣化，可以考虑
  1. 恢复默认的统计信息。
  2. 使用plan hint来调整到之前的查询计划。（详细参见《开发指南》中“SQL调优指南 > 使用Plan Hint 进行调优”章节）
- 当此guc参数设置为负数时，如果计算的采样样本数大于等于总数据量的2%，且用户表的数据量小于1600000时，ANALYZE所需时间相比guc参数为默认值的时间会有所增加。
- 当此guc参数设置为负数时，则autoanalyze不生效。

默认值：100

constraint_exclusion

参数说明：控制查询优化器使用表约束查询的优化。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：枚举类型

- on/true/yes/1表示检查所有表的约束。
- off/false/no/0表示不检查约束。
- partition表示只检查继承的子表和UNION ALL子查询。

须知

当constraint_exclusion为on，优化器用查询条件和表的CHECK约束比较，并且在查询条件和约束冲突的时候忽略对表的扫描。

默认值：partition

说明

目前， constraint_exclusion缺省被打开，通常用来实现表分区。为所有的表打开它时，对于简单的查询强加了额外的规划，并且对简单查询没有什么好处。如果不用分区表，可以关掉它。

cursor_tuple_fraction

参数说明：优化器估计游标获取行数在总行数中的占比。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：浮点型，0.0～1.0。

须知

比默认值小的值与使用 “fast start” 为游标规划的值相偏离，从而使得前几行恢复的很快而抓取全部的行需要很长的时间。比默认值大的值加大了总的估计的时间。在最大的值1.0处，像正常的查询一样规划游标，只考虑总的估计时间和传送第一行的时间。

默认值：0.1

from_collapse_limit

参数说明：根据生成的FROM列表的项数来判断优化器是否将把子查询合并到上层查询，如果FROM列表项个数小于等于该参数值，优化器会将子查询合并到上层查询。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1～2147483647。

须知

比默认值小的数值将降低规划时间，但是可能生成差的执行计划。

默认值：8

join_collapse_limit

参数说明：根据得出的列表项数来判断优化器是否执行把除FULL JOIN之外的JOIN构造重写到FROM列表中。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1～2147483647。

须知

- 设置为1会避免任何JOIN重排。这样就使得查询中指定的连接顺序就是实际的连接顺序。查询优化器并不是总能选取最优的连接顺序，高级用户可以选择暂时把这个变量设置为1，然后指定它们需要的连接顺序。
- 比默认值小的数值减少规划时间但也降低了执行计划的质量。

默认值：8

plan_mode_seed

参数说明：该参数为调测参数，目前仅支持OPTIMIZE_PLAN和RANDOM_PLAN两种。其中：OPTIMIZE_PLAN表示通过动态规划算法进行代价估算的最优plan，参数值设置为0；RANDOM_PLAN表示随机生成的plan；如果设置为-1，表示用户不指定随机数的种子标识符seed值，由优化器随机生成[1, 2147483647]范围整型值的随机数，

并根据随机数生成随机的执行计划；如果用户指定guc参数值为[1, 2147483647]范围的整型值，表示指定的生成随机数的种子标识符seed，优化器需要根据seed值生成随机的执行计划。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，-1~ 2147483647

默认值：0

须知

- 当该参数设置为随机执行计划模式时，优化器会生成不同的随机执行计划，该执行计划可能不是最优计划。因此在随机计划模式下，会对查询性能产生影响，所以建议在升级、扩容、缩容等正常业务操作或运维过程中将该参数保持为默认值0。
- 当该参数不为0时，查询指定的plan hint不会生效。

enable_random_datanode

参数说明：表示是否允许开启复制表DN随机查找功能，复制表在每个DN存放一份完整数据，随机选取可以缓解节点压力。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许开启复制表DN随机查找功能。
- off表示不允许开启复制表DN随机查找功能。

默认值：on

hashagg_table_size

参数说明：用于设置执行HASH JOIN操作时HASH表的大小。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0～1073741823。

默认值：0

enable_bloom_filter

参数说明：标识是否允许使用BloomFilter优化。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许使用BloomFilter优化。
- off表示不允许使用BloomFilter优化。

默认值：on

enable_extrapolation_stats

参数说明：标识对于日期类型是否允许基于历史统计信息使用推理估算的逻辑。使用该逻辑对于未及时收集统计信息的表可以增大估算准确的可能性，但也存在错误推理导致估算过大的可能性，需要对于日期类型数据定期插入的场景开启此开关。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许基于历史统计信息使用推理估算的逻辑。
- off表示不允许基于历史统计信息使用推理估算的逻辑。

默认值：off

autoanalyze

参数说明：标识是否允许在生成计划的时候，对于没有统计信息的表进行统计信息自动收集。对于临时表，不支持autoanalyze，如果需要收集统计信息，用户需手动执行analyze操作。如果在auto analyze某个表的过程中数据库发生异常，当数据库正常运行之后再执行语句有可能仍提示需要收集此表的统计信息。此时需要用户对该表手动执行一次analyze操作，以同步统计信息数据。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许自动进行统计信息收集。
- off表示不允许自动进行统计信息收集。

默认值：off

说明

该参数与autovacuum线程下的autoanalyze无关。

query_dop

参数说明：用户自定义的查询并行度。该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，-64-64

[1,64]：打开固定SMP功能，系统会使用固定并行度。

0：打开SMP自适应功能，系统会根据资源情况和计划特征动态选取最优并行度。

[-64,-1]：打开SMP自适应功能，并限制自适应选取的最大并行度。

说明

- 在开启并行查询后，请保证系统CPU、内存、网络、I/O等资源充足，以达到最佳效果。
- 为了避免用户设置不合理的过大值造成性能劣化，系统会计算出该DN可用最大CPU核数，并以此来作为query_dop的上限。如果用户设置query_dop超过4并且同时超过该上限，那么系统会重置query_dop为该上限值。

默认值：1

enable_analyze_check

参数说明: 标识是否允许在生成计划的时候，对于在pg_class中显示reltuples和relpages均为0的表，检查该表是否曾进行过统计信息收集。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许检查。
- off表示不允许检查。

默认值：off

enable_sonic_hashagg

参数说明：标识是否依据规则约束使用基于面向列的hash表设计的Hash Agg算子。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示在满足约束条件时使用基于面向列的hash表设计的Hash Agg算子。
- off表示不使用面向列的hash表设计的Hash Agg算子。

说明

- 在开启enable_sonic_hashagg，且查询达到约束条件使用基于面向列的hash表设计的Hash Agg算子时，查询对应的Hash Agg算子内存使用通常可获得精简。但对于代码生成技术可获得显著性能提升的场景，对应的算子查询性能可能会出现劣化。
- 开启enable_sonic_hashagg，且查询达到约束条件使用基于面向列的hash表设计的Hash Agg算子时，在Explain Analyze/Performance的执行计划和执行信息中，算子显示为“Sonic Hash Aggregation”，而未达到该约束条件时，算子名称将显示为“Hash Aggregation”，Explain详解请参见《开发指南》中“SQL调优指南 > SQL执行计划介绍 > 详解”章节。

默认值：on

enable_sonic_hashjoin

参数说明：标识是否依据规则约束使用基于面向列的hash表设计的Hash Join算子。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示在满足约束条件时使用基于面向列的hash表设计的Hash Join算子。
- off表示不使用面向列的hash表设计的Hash Join算子。

说明

- 当前开关仅适用于Inner Join的场景。
- 在开启enable_sonic_hashjoin，查询对应的Hash Inner算子内存使用通常可获得精简。但对于代码生成技术可获得显著性能提升的场景，对应的算子查询性能可能会出现劣化。
- 开启enable_sonic_hashjoin，且查询达到约束条件使用基于面向列的hash表设计的Hash Join算子时，在Explain Analyze/Performance的执行计划和执行信息中，算子显示为“Sonic Hash Join”，而未达到该约束条件时，算子名称将显示为“Hash Join”，Explain详解请参见《开发指南》中“SQL调优指南 > SQL执行计划介绍 > 详解”章节。

默认值：on

enable_sonic_optspill

参数说明：标识是否对面向列的hash表设计的Hash Join算子进行下盘文件数优化。该参数打开时，在Hash Join算子下盘文件较多的时候，下盘文件数不会显著增加。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示优化面向列的hash表设计的Hash Join算子的下盘文件数。
- off表示不优化面向列的hash表设计的Hash Join算子的下盘文件数。

默认值：on

plan_cache_mode

参数说明：标识在prepare语句中，选择生成执行计划的策略。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：枚举类型

- auto表示按照默认的方式选择custom plan或者generic plan。
- force_generic_plan表示强制走generic plan（软解析）。generic plan是指对于preapre语句生成计划，该计划策略会在执行execute语句的时候把参数bind到plan中，然后执行计划。这种方案的优点是每次执行可以省去重复的优化器开销；缺点是当bind参数字段上数据存在倾斜时该计划可能不是最优的，部分bind参数场景下执行性能较差。
- force_custom_plan表示强制走custom plan（硬解析）。custom plan是指对于prepare语句，在执行execute的时候，把execute语句中的参数嵌套到语句之后生成的计划。custom plan会根据execute语句中具体的参数生成计划，这种方案的优点是每次都按照具体的参数生成优选计划，执行性能比较好；缺点是每次执行前都需要重新生成计划，存在大量的重复的优化器开销。

说明

此参数只对prepare语句生效，一般用在prepare语句中参数化字段存在比较严重的数据倾斜的场景下。

默认值：auto

enable_router

参数说明：是否打开手动设置下推节点功能。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用。
- off表示不使用。

默认值：off

router

参数说明：用于控制router功能的详细属性，仅在打开enable_router和enable_light_proxy后生效。该参数会根据表的hash分布列，计算给定的分布列在哪个DN上，设置router后将支持的sql下推到该DN上执行。如果设置错了router，可能导致数据存入错误的dn，导致不可预测的问题，需谨慎使用。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

该参数分为两部分，'schema_name.table_name,"distribute_keys"'，其具体含义如下：

- schema_name.table_name：表示schema名和表名，如不显示设置schema_name，则默认为current_schema。
- distribute_keys：分布表的所有分布列值，用逗号间隔开，且分布列值的顺序必须和表中分布列顺序一致。

默认值：空

enable_auto_explain

参数说明：控制是否开启自动打印执行计划。该参数是用来定位慢存储过程或慢查询，只对当前连接的CN有效。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型，on表示开启，off表示关闭。

默认值：off

auto_explain_level

参数说明：控制自动打印执行计划的日志等级。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：枚举型，log或notice，log表示在日志中打印执行计划，notice表示以提示知的形式打印出计划。

默认值：log

auto_explain_log_min_duration

参数说明：控制自动打印执行计划的耗时阈值，整体耗时大于auto_explain_log_min_duration的执行计划才会被打印。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~2147483647，单位为毫秒。

- 设置为0，所有执行过的执行计划都会输出。
- 设置为3000，单次语句执行耗时超过3000毫秒后所有执行的执行计划会输出。

默认值：0

max_datanode_for_plan

参数说明：生成FQS计划时设置显示DN上执行计划的个数。显示DN上计划的个数由集群中的DN数和该参数值的更小者决定。

对于PBE执行的语句，当前只能显示内核prepare预编译方式生成的计划，不支持显示JDBC预编译方式生成的计划。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0 ~ 8192

默认值：0

session_sequence_cache

参数说明：在当前会话下，一次性交互申请的sequence数值，会话结束会自动丢弃未用完的值。用户在使用sequence大批量导入数据的时候可以通过调大该参数，提高插入速度，增加高并发性能；用户在高并发单条插入数据的时候，将该参数设置为1，减少了sequence的跳变。如对连续性有强要求，需要在创建sequence的时候指定需要的cache，该参数如果大于cache指定的值会自动失效。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1 ~ 2147483647。

默认值：10

说明

默认值为10是高并发场景下，兼顾单条插入和批量插入的性能会有比较好的表现。

18.3.9 错误报告和日志

18.3.9.1 记录日志的位置

log_destination

参数说明：GaussDB支持多种方法记录服务器日志，log_destination的取值为一个逗号分隔开的列表（如log_destination="stderr,csvlog"）。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

有效值为stderr、csvlog、syslog、eventlog。

- 取值为stderr，表示日志打印到屏幕。
- 取值为csvlog，表示日志的输出格式为“逗号分隔值”即CSV（Comma Separated Value）格式。使用csvlog记录日志的前提是将**logging_collector**设置为on，请参见**使用CSV格式写日志**。
- 取值为syslog，表示通过操作系统的syslog记录日志。GaussDB使用syslog的LOCAL0 ~ LOCAL7记录日志，请参见**syslog_facility**。使用syslog记录日志需在操作系统后台服务配置文件中添加代码：

```
local0.*  /var/log/postgresql
```


默认值：stderr

logging_collector

参数说明：控制开启后端日志收集进程logger进行日志收集。该进程捕获发送到stderr或csvlog的日志消息并写入日志文件。

这种记录日志的方法比将日志记录到syslog更加有效，因为某些类型的消息在syslog的输出中无法显示。例如动态链接库加载失败消息和脚本产生的错误消息。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

将服务器日志发送到stderr时可以不使用logging_collector参数，此时日志消息会被发送到服务器的stderr指向的空间。这种方法的缺点是日志回滚困难，只适用于较小的日志容量。

取值范围：布尔型

- on表示开启日志收集功能。
- off表示关闭日志收集功能。

默认值：on

log_directory

参数说明：logging_collector设置为on时，log_directory决定存放服务器日志文件的目录。它可以是绝对路径，或者是相对路径（相对于数据目录的路径）。log_directory支持动态修改，可以通过gs_guc reload实现，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

- 当配置文件中log_directory的值为非法路径时，会导致集群无法重新启动。
- 通过gs_guc reload动态修改log_directory时，当指定路径为合法路径时，日志输出到新的路径下。当指定路径为非法路径时，日志输出到上一次合法的日志输出路径下而不影响数据库正常运行。此时即使指定的log_directory的值非法，也会写入到配置文件中。
- 在沙箱环境，路径中不可以包含/var/chroot，例如log的绝对路径是/var/chroot/var/lib/log/Ruby/pg_log/cn_log，则只需要设置为/var/lib/log/Ruby/pg_log/cn_log。

说明

- 合法路径：用户对此路径有读写权限。
- 非法路径：用户对此路径无读写权限。

取值范围：字符串

默认值：安装时指定

log_filename

参数说明：logging_collector设置为on时，log_filename决定服务器运行日志文件的名称。通常日志文件名是按照strftime模式生成，因此可以用系统时间定义日志文件名，用%转义字符实现，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

- 建议使用%转义字符定义日志文件名称，否则难以对日志文件进行有效的管理。
- 当log_destination设为csvlog时，系统会生成附加了时间戳的日志文件名，文件格式为csv格式，例如“server_log.1093827753.csv”。

取值范围：字符串

默认值：postgresql-%Y-%m-%d_%H%M%S.log

log_file_mode

参数说明：logging_collector设置为on时，log_file_mode设置服务器日志文件的权限。在Windows系统下，此选项无效。通常log_file_mode的取值是能够被chmod和umask系统调用接受的数字。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

- 使用此选项前请设置log_directory，将日志存储到数据目录之外的地方。
- 因日志文件可能含有敏感数据，故不能将其设为对外可读。

取值范围：整型，0000～0777（8进制计数，转化为十进制 0 ~ 511）。

说明

- 0600表示只允许服务器管理员读写日志文件。
- 0640表示允许管理员所在用户组成员只能读日志文件。

默认值：0600

log_truncate_on_rotation

参数说明：logging_collector设置为on时，log_truncate_on_rotation设置日志消息的写入方式。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

示例如下：

假设日志需要保留7天，每天生成一个日志文件，日志文件名设置为server_log.Mon、server_log.Tue等。第二周的周二生成的日志消息会覆盖写入到server_log.Tue。设置方法：将log_filename设置为server_log.%a ，log_truncate_on_rotation设置为on，log_rotation_age设置为1440，即日志有效时间为1天。

取值范围： 布尔型

- on表示GaussDB以覆盖写入的方式写服务器日志消息。
- off表示GaussDB将日志消息附加到同名的现有日志文件上。

默认值：off

log_rotation_age

参数说明：logging_collector设置为on时，log_rotation_age决定创建一个新日志文件的时间间隔。当现在的时间减去上次创建一个服务器日志的时间超过了log_rotation_age的值时，将生成一个新的日志文件。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0 ~ 35791394，单位为min。其中0表示关闭基于时间的新日志文件的创建。

默认值：1d（即1440min）

log_rotation_size

参数说明：logging_collector设置为on时，log_rotation_size决定服务器日志文件的最大容量。当日志消息的总量超过日志文件容量时，服务器将生成一个新的日志文件。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0 ~ 2097151，单位为KB。

0表示关闭基于容量的新日志文件的创建。

默认值：20MB

syslog_facility

参数说明：log_destination设置为syslog时，syslog_facility配置使用syslog记录日志的“设备”。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：枚举类型，有效值有local0、local1、local2、local3、local4、local5、local6、local7。

默认值：local0

syslog_ident

参数说明：**log_destination**设置为syslog时，syslog_ident设置在syslog日志中GaussDB日志消息的标识。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

默认值：postgres

event_source

参数说明：log_destination设置为eventlog时，event_source设置在日志中GaussDB日志消息的标识。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

默认值：PostgreSQL

18.3.9.2 记录日志的时间

client_min_messages

参数说明：控制发送到客户端的消息级别。每个级别都包含排在它后面的所有级别中的信息。级别越低，发送给客户端的消息就越少。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

当client_min_messages和**log_min_messages**取相同值时，其值所代表的级别不同。

取值范围：枚举类型，有效值有debug、debug5、debug4、debug3、debug2、debug1、info、log、notice、warning、error、fatal、panic，其中debug和debug2等效。参数的详细信息请参见**表18-8**。在实际设置过程中，如果设置的级别大于error，为fatal或panic，系统会默认将级别转为error。

默认值：notice

log_min_messages

参数说明：控制写到服务器日志文件中的消息级别。每个级别都包含排在它后面的所有级别中的信息。级别越低，服务器运行日志中记录的消息就越少。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

当**client_min_messages**和log_min_messages取相同值log时所代表的消息级别不同。

取值范围：枚举类型，有效值有debug、debug5、debug4、debug3、debug2、debug1、info、log、notice、warning、error、fatal、panic，其中debug和debug2等效。参数的详细信息请参见**表18-8**。

默认值：warning

log_min_error_statement

参数说明：控制在服务器日志中记录错误的SQL语句。

该参数属于SUSET类型参数，请参考表18-1中对应设置方法进行设置。

取值范围：枚举类型，有效值有debug、debug5、debug4、debug3、debug2、debug1、info、log、notice、warning、error、fatal、panic。参数的详细信息请参见表18-8。

说明

- 设置为error ，表示导致错误、日志消息、致命错误、panic的语句都将被记录。
- 设置为panic，表示关闭此特性。

默认值：error

log_min_duration_statement

参数说明：当某条语句的持续时间大于或者等于特定的毫秒数时，log_min_duration_statement参数用于控制记录每条完成语句的持续时间。

设置log_min_duration_statement可以很方便地跟踪需要优化的查询语句。对于使用扩展查询协议的客户端，语法分析、绑定、执行每一步所花时间被独立记录。

该参数属于SUSET类型参数，请参考表18-1中对应设置方法进行设置。

须知

当此选项与log_statement同时使用时，已经被log_statement记录的语句文本不会被重复记录。在没有使用syslog情况下，推荐使用log_line_prefix记录PID或会话ID，方便将当前语句消息连接到最后的持续时间消息。

取值范围：整型，-1 ~ 2147483647，单位为毫秒。

- 设置为250，所有运行时间不短于250ms的SQL语句都会被记录。
- 设置为0，输出所有语句的持续时间。
- 设置为-1，关闭此功能。

默认值：3s（即3000ms）

backtrace_min_messages

参数说明：控制当产生该设置参数级别相等或更高级别的信息时，会打印函数的堆栈信息到服务器日志文件中。

该参数属于SUSET类型参数，请参考表18-1中对应设置方法进行设置。

须知

该参数作为客户现场问题定位手段使用，且由于频繁的打印函数栈会对系统的开销及稳定性有一定的影响，因此如果需要进行问题定位时，建议避免将backtrace_min_messages的值设置为fatal及panic以外的级别。

取值范围：枚举类型

有效值有debug、debug5、debug4、debug3、debug2、debug1、info、log、notice、warning、error、fatal、panic。参数的详细信息请参见表18-8。

默认值：panic

表18-8解释GaussDB中使用的消息安全级别。当日志输出到syslog或者eventlog时，GaussDB进行如表中的转换。

表 18-8 信息严重程度分类

信息严重程度类型	详细说明	系统日志	事件日志
debug[1-5]	报告详细调试信息。	DEBUG	INFORMATION
log	报告对数据库管理员有用的信息， 比如检查点操作统计信息。	INFO	INFORMATION
info	报告用户可能需求的信息， 比如在VACUUM VERBOSE过程中的信息。	INFO	INFORMATION
notice	报告可能对用户有帮助的信息， 比如， 长标识符的截断， 作为主键一部分创建的索引等。	NOTICE	INFORMATION
warning	报告警告信息， 比如在事务块范围之外的COMMIT。	NOTICE	WARNING
error	报告导致当前命令退出的错误。	WARNING	ERROR
fatal	报告导致当前会话终止的原因。	ERR	ERROR
panic	报告导致整个数据库被关闭的原因。	CRIT	ERROR

18.3.9.3 记录日志的内容

debug_print_parse

参数说明：用于控制打印解析树结果。

该参数属于SIGHUP类型参数，请参考表18-1中对应设置方法进行设置。

取值范围：布尔型

- on表示开启打印结果的功能。
- off表示关闭打印结果的功能。

默认值：off

debug_print_rewritten

参数说明：用于控制打印查询重写结果。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启打印结果的功能。
- off表示关闭打印结果的功能。

默认值：off

debug_print_plan

参数说明：用于设置是否将查询的执行计划打印到日志中。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启打印结果的功能。
- off表示关闭打印结果的功能。

默认值：off

须知

- 只有当日志的级别为log及以上时，debug_print_parse、debug_print_rewritten和debug_print_plan的调试信息才会输出。当这些选项打开时，调试信息只会记录在服务器的日志中，而不会输出到客户端的日志中。通过设置**client_min_messages**和**log_min_messages**参数可以改变日志级别。
- 在打开debug_print_plan开关的情况下需尽量避免调用gs_encrypt_aes128及gs_decrypt_aes128函数，避免敏感参数信息在日志中泄露的风险。同时建议用户在打开debug_print_plan开关生成的日志中对gs_encrypt_aes128及gs_decrypt_aes128函数的参数信息进行过滤后再提供给外部维护人员定位，日志使用完成后请及时删除。

debug_pretty_print

参数说明：设置此选项对debug_print_parse、debug_print_rewritten和debug_print_plan产生的日志进行缩进，会生成易读但比设置为off时更长的输出格式。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示进行缩进。
- off表示不进行缩进。

默认值：on

log_checkpoints

参数说明：控制在服务器日志中记录检查点和重启点的信息。打开此参数时，服务器日志消息包含涉及检查点和重启点的统计量，其中包含需要写的缓存区的数量及写入所花费的时间等。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示打开此参数时，服务器日志消息包含涉及检查点和重启点的统计量。
- off表示关闭此参数时，服务器日志消息包含不涉及检查点和重启点的统计量。

默认值：off

log_connections

参数说明：控制记录客户端的连接请求信息。

该参数属于BACKEND类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

有些客户端程序（例如gsql），在判断是否需要口令的时候会尝试连接两次，因此日志消息中重复的“connection receive”（收到连接请求）并不意味着一定是问题。

取值范围：布尔型

- on表示记录信息。
- off表示不记录信息。

默认值：off

log_disconnections

参数说明：控制记录客户端结束连接信息。

该参数属于BACKEND类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示记录信息。
- off表示不记录信息。

默认值：off

log_duration

参数说明：控制记录每个已完成SQL语句的执行时间。对使用扩展查询协议的客户端、会记录语法分析、绑定和执行每一步所花费的时间。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- 设置为off，该选项与**log_min_duration_statement**的不同之处在于log_min_duration_statement强制记录查询文本。
- 设置为on并且log_min_duration_statement大于零，记录所有持续时间，但是仅记录超过阈值的语句。这可用于在高负载情况下搜集统计信息。

默认值：off

log_error_verbosity

参数说明：控制服务器日志中每条记录的消息写入的详细度。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：枚举类型

- terse输出不包括DETAIL、HINT、QUERY及CONTEXT错误信息的记录。
- verbose输出包括SQLSTATE错误代码 、源代码文件名、函数名及产生错误所在的行号。
- default输出包括DETAIL、HINT、QUERY及CONTEXT错误信息的记录，不包括SQLSTATE错误代码 、源代码文件名、函数名及产生错误所在的行号。

默认值：default

log_hostname

参数说明：默认状态下，连接消息日志只显示正在连接主机的IP地址。打开此选项同时可以记录主机名。由于解析主机名可能需要一定的时间，可能影响数据库的性能。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示可以同时记录主机名。
- off表示不可以同时记录主机名。

默认值：off

log_line_prefix

参数说明：控制每条日志信息的前缀格式。日志前缀类似于printf风格的字符串，在日志的每行开头输出。用以%为开头的“转义字符”代替**表18-9**中的状态信息。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

表 18-9 转义字符表

转义字符	效果
%a	应用程序名称。
%u	用户名。
%d	数据库名。
%r	远端主机名或者IP地址以及远端端口，在不启动log_hostname时显示IP地址及远端端口。

转义字符	效果
%h	远端主机名或者IP地址，在不启动log_hostname时只显示IP地址。
%p	线程ID。
%t	时间戳（没有毫秒，Windows上没有时区）。
%m	带毫秒的时间戳。
%n	表示指定错误从哪个节点上报的。
%i	命令标签：会话当前执行的命令类型。
%e	SQLSTATE错误码。
%c	会话ID，详见说明。
%l	每个会话的日志编号，从1开始。
%s	会话启动时间。
%v	虚拟事务ID（backendID/ localXID）
%x	事务ID（0表示没有分配事务ID）。
%q	不产生任何输出。如果当前线程是后端线程，忽略这个转义序列，继续处理后面的转义序列；如果当前线程不是后端线程，忽略这个转义序列和它后面的所有转义序列。
%S	会话ID。
%T	TraceID。
%%	字符%。

说明

转义字符%c打印一个唯一的会话ID，由两个4字节的十六进制数组成，通过字符“.”分开。这两个十六进制数分别表示进程的启动时间及进程编号，所以%c也可以看作是保存打印这些名目的途径的空间。比如，从pg_stat_activity中产生会话ID，可以用下面的查询：

```
SELECT to_hex(EXTRACT(EPOCH FROM backend_start)::integer) || '.' ||
       to_hex(pid)
FROM pg_stat_activity;
```

- 当log_line_prefix设置为非空值时，请保证最后一个字符是一个空格，以此来直观地与后续的日志行进行区分，也可以使用一个标点符号。
- Syslog生成自己的时间戳及进程ID信息，所以当登录日志时，不需要包含这些转义字符。

取值范围：字符串

默认值：'%m %n %u %d %h %p %S %x %a '

说明

%m %n %u %d %h %p %S %x %a 表示会话开始时间戳、错误上报节点、用户名、数据库名、远程主机名或IP、线程ID、会话ID、事务ID、应用名。

log_lock_waits

参数说明：当一个会话的等待获得一个锁的时间超过**deadlock_timeout**的值时，此选项控制在数据库日志中记录此消息。这对于决定锁等待是否会产生一个坏的行为是非常有用的。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示记录此信息。
- off表示不记录此信息。

默认值：off

log_statement

参数说明：控制记录SQL语句。对于使用扩展查询协议的客户端，记录接收到执行消息的事件和绑定参数的值（内置单引号要双写）。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

即使log_statement设置为all，包含简单语法错误的语句也不会被记录，因为仅在完成基本的语法分析并确定了语句类型之后才记录日志。在使用扩展查询协议的情况下，在执行阶段之前（语法分析或规划阶段）同样不会记录。将log_min_error_statement设为ERROR或更低才能记录这些语句。

取值范围：枚举类型

- none表示不记录语句。
- ddl表示记录所有的数据定义语句，比如CREATE、ALTER和DROP语句。
- mod表示记录所有DDL语句，还包括数据修改语句INSERT、UPDATE、DELETE、TRUNCATE和COPY FROM 。
- all表示记录所有语句，PREPARE、EXECUTE和EXPLAIN ANALYZE语句也同样被记录。

默认值：none

log_temp_files

参数说明：控制记录临时文件的删除信息。临时文件可以用来排序、哈希及临时查询结果。当一个临时文件被删除时，将会产生一条日志消息。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，-1 ~ 2147483647，单位KB。

- 正整数表示只记录比log_temp_files设定值大的临时文件的删除信息。
- 0表示记录所有的临时文件的删除信息。
- -1表示不记录任何临时文件的删除信息。

默认值：-1

log_timezone

参数说明：设置服务器写日志文件时使用的时区。与**TimeZone**不同，这个值是数据库范围的，针对所有连接到本数据库的会话生效。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串，可查询视图PG_TIMEZONE_NAMES获得，具体可参见《开发指南》中“系统表和系统视图 > 系统视图 > PG_TIMEZONE_NAMES”章节。

默认值：根据OS时区设置

说明

gs_initdb进行相应系统环境设置时会对默认值进行修改。

logging_module

参数说明：用于设置或者显示模块日志在服务端的可输出性。该参数属于会话级参数，不建议通过gs_guc工具来设置。

该参数属于USERSET类型参数，设置请参考**表18-1**中对应设置的方法进行设置。

取值范围：字符串

默认值：所有模块日志在服务端是不输出的，可由SHOW logging_module查看：

```
ALL,on(),off(GUC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,OBS,EXECUTOR,VEC_EXECUTOR,STREAM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PARQUET,PLANHINT,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,INCRE_CKPT,DBL_WRT,RTO,HEARTBEAT,COMM_IPC,COMM_PARAM,ENCODING_CHECK)
```

设置方法：首先，可以通过SHOW logging_module来查看哪些模块是支持可控制的。例如，查询输出结果为：

```
openGauss=# show logging_module;
logging_module
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
ALL,on(),off(GUC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,OBS,EXECUTOR,VEC_EXECUTOR,STREAM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PARQUET,PLANHINT,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,INCRE_CKPT,DBL_WRT,RTO,HEARTBEAT,COMM_IPC,COMM_PARAM,ENCODING_CHECK)
(1 row)
```

支持可控制的模块使用大写来标识，特殊标识ALL用于对所有模块日志进行设置。可以使用on/off来控制模块日志的输出。设置SSL模块日志为可输出，使用如下命令：

```
openGauss=# set logging_module='on(SSL)';
SET
openGauss=# show
logging_module;
       logging_module
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
ALL,on(SSL),off(GUC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,GDS,TBLSPC,WLM,OBS,EXECUTOR,VEC_EXECUTOR,STREAM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PARQUET,PLANHINT,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,INCRE_CKPT,DBL_WRT,RTO,HEARTBEAT,COMM_IPC,COMM_PARAM,ENCODING_CHECK)
(1 row)
```


可以看到模块SSL的日志输出被打开。

ALL标识是相当于一个快捷操作，即对所有模块的日志可输出进行开启或关闭。

```
openGauss=# set logging_module='off(ALL)';
SET
openGauss=# show
logging_module;
         logging_module
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------
ALL,on(),off(GUC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,OBS,EXECUTOR,VEC_EXECU
TOR,STREAM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,UDF,COOP_ANA
LYZE,WLMCP,ACCELERATE,PARQUET,PLANHINT,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQ
L,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,INCRE_CKPT,DBL_WRT,RTO,HEARTBEAT,COMM_IPC,CO
MM_PARAM,ENCODING_CHECK)
(1 row)

openGauss=# set logging_module='on(ALL)';
SET
openGauss=# show
logging_module;
        logging_module
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------
ALL,on(GUC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,OBS,EXECUTOR,VEC_EXECUTOR,
STREAM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,UDF,COOP_ANALYZE
,WLMCP,ACCELERATE,PARQUET,PLANHINT,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TE
XTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,INCRE_CKPT,DBL_WRT,RTO,HEARTBEAT,COMM_IPC,COMM_
PARAM,ENCODING_CHECK),off()
(1 row)
```

依赖关系：该参数依赖于log_min_level参数的设置

enable_unshipping_log

参数说明：用于控制是否打印语句不下推的日志，主要用于帮助用户定位不下推语句可能导致的性能问题。当enable_stream_operator参数关闭时，如果这个参数设置为on，会有大量关于计划不能下推的日志记录到日志文件中。如果用户不需要这些日志内容，建议用户在enable_stream_operator参数关闭时，也同时关闭enable_unshipping_log参数。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示打印日志。
- off表示不打印日志。

默认值：off

opfusion_debug_mode

参数说明：用于调试简单查询是否进行查询优化。设置成log级别可以在DN的执行计划中看到没有查询优化的具体原因。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：枚举类型

- off表示不打开该功能。

- log表示打开该功能，可以在DN的执行计划中看到没有查询优化的具体原因。

须知

- 需要设置参数max_datanode_for_plan才能看到DN的执行计划。
- 提供在log中显示语句没有查询优化的具体原因，需要将参数设置成log级别，log_min_messages设置成debug4级别，logging_module设置'on(OPFUSION)'，注意log内容可能会比较多，尽可能在调优期间执行少量作业使用。

默认值： off

enable_debug_vacuum

参数说明： 允许输出一些与VACUUM相关的日志，便于定位VACUUM相关问题。开发人员专用，不建议普通用户使用。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围： 布尔型

- on/true表示开启此日志开关。
- off/false表示关闭此日志开关。

默认值： off

18.3.9.4 使用 CSV 格式写日志

前提条件

- **log_destination**的值设置为csvlog。
- **logging_collector**的值设置为on。

csvlog 定义

以“逗号分隔值” 即CSV（Comma Separated Value）的形式发出日志。

以下是简单的用来存储CSV形式日志输出的表定义：

```
CREATE TABLE postgres_log
(
log_time timestamp(3) with time zone,
node_name text,
user_name text,
database_name text,
process_id bigint,
connection_from text,
"session_id" text,
session_line_num bigint,
command_tag text,
session_start_time timestamp with time zone,
virtual_transaction_id text,
transaction_id bigint,
query_id bigint,
module text,
error_severity text,
sql_state_code text,
message text,
detail text,
```



```
hint text,
internal_query text,
internal_query_pos integer,
context text,
query text,
query_pos integer,
location text,
application_name text
);
```

详细说明请参见**表18-10**。

表 18-10 csvlog 字段含义表

字段名	字段含义	字段名	字段含义
log_time	毫秒级的时间戳	module	日志所属模块
node_name	节点名称	error_severity	ERRORSTATE代码
user_name	用户名	sql_state_code	SQLSTATE代码
database_name	数据库名	message	错误消息
process_id	进程ID	detail	详细错误消息
connection_from	客户主机：端口号	hint	提示
session_id	会话ID	internal_query	内部查询（查询那些导致错误的信息，如果有的话）
session_line_num	每个会话的行数	internal_query_pos	内部查询指针
command_tag	命令标签	context	环境
session_start_time	会话开始时间	query	错误发生位置的字符统计
virtual_transaction_id	常规事务	query_pos	错误发生位置指针
transaction_id	事务ID	location	在GaussDB源代码中错误的位置（如果**log_error_verbosity**的值设为verbose ）
query_id	查询ID	application_name	应用名称

使用COPY FROM命令将日志文件导入这个表：

```
COPY postgres_log FROM '/opt/data/pg_log/logfile.csv' WITH csv;
```


说明

此处的日志名“logfile.csv”要换成实际生成的日志的名称。

简化输入

简化输入到CSV日志文件，可以通过如下操作：

- 设置**log_filename**和**log_rotation_age**，为日志文件提供一个一致的、可预测的命名方案。通过日志文件名，预测一个独立的日志文件完成并进入准备导入状态的时间。
- 将**log_rotation_size**设为0来终止基于尺寸的日志回滚，因为基于尺寸的日志回滚让预测日志文件名变得非常的困难。
- 将**log_truncate_on_rotation**设为on以便区分在同一日志文件中旧的日志数据和新的日志数据。

18.3.10 告警检测

在集群运行的过程中，会对数据库中的错误场景进行检测，便于用户及早感知到数据库集群的错误。告警写入的system_alarm日志可以在$GAUSSLOG/cm或$GAUSSLOG/pg_log/gtm路径下查看。

enable_alarm

参数说明：允许打开告警检测线程，检测数据库中可能的错误场景。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许打开告警检测线程。
- off表示不允许打开告警检测线程。

默认值：on

说明

该参数生效范围节点仅为CN、DN。

connection_alarm_rate

参数说明：允许和数据库连接的最大并发连接数的比率限制。数据库连接的最大并发连接数为**max_connections*** connection_alarm_rate。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：浮点型，0.0~1.0

默认值：0.9

alarm_report_interval

参数说明：指定告警上报的时间间隔。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，单位为秒。

默认值：10

alarm_component

参数说明：在对告警做上报时，会进行告警抑制，即同一个实例的同一个告警项在alarm_report_interval（默认值为10s）内不做重复上报。在这种情况下设置用于处理告警内容的告警组件的位置，仅sysadmin用户可以访问。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串。

- 若前置脚本gs_preinstall中的--alarm-type参数设置为5时，表示未对接第三方组件，告警写入system_alarm日志，此时GUC参数alarm_component的取值为：/opt/huawei/snas/bin/snas_cm_cmd。
- 若前置脚本gs_preinstall中的--alarm-type参数设置为1时，表示对接第三方组件，此时GUC参数alarm_component的值为第三方组件的可执行程序的绝对路径。

默认值：/opt/huawei/snas/bin/snas_cm_cmd

18.3.11 运行时统计

18.3.11.1 查询和索引统计收集器

查询和索引统计收集器负责收集数据库系统运行中的统计数据，如在一个表和索引上进行了多少次插入与更新操作、磁盘块的数量和元组的数量、每个表上最近一次执行清理和分析操作的时间等。可以通过查询系统视图pg_stats和pg_statistic查看统计数据。下面的参数设置服务器范围内的统计收集特性。

track_activities

参数说明：控制收集每个会话中当前正在执行命令的统计数据。对于存储过程，打开该参数后，可以通过pg_stat_activity视图看到存储过程内正在执行的perform语句、调用存储过程语句、存储过程内的SQL语句、OPEN CURSOR语句。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启收集功能。
- off表示关闭收集功能。

默认值：on

track_counts

参数说明：控制收集数据库活动的统计数据。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启收集功能。
- off表示关闭收集功能。

说明

在AutoVacuum自动清理线程中选择清理的数据库时，需要数据库的统计数据，故默认值设为on。

默认值：on

track_io_timing

参数说明：控制收集数据库I/O调用时序的统计数据。I/O时序统计数据可以在pg_stat_database中查询。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启收集功能，开启时，收集器会在重复地去查询当前时间的操作系统，这可能会引起某些平台的重大开销，故默认值设置为off。
- off表示关闭收集功能。

默认值：off

track_functions

参数说明：控制收集函数的调用次数和调用耗时的统计数据。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

当SQL语言函数设置为调用查询的“内联”函数时，不管是否设置此选项，这些SQL语言函数无法被追踪到。

取值范围：枚举类型

- pl表示只追踪过程语言函数。
- all表示追踪SQL。
- none表示关闭函数追踪功能。

默认值：none

track_activity_query_size

参数说明：设置用于跟踪每一个活动会话的当前正在执行命令的字节数。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，100～102400

默认值：1024

track_procedure_sql

参数说明：该参数控制pg_stat_activity系统表中的query列是否同时打印该存储过程中正在执行的SQL语句。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示调用存储过程时，pg_stat_activity的query列会同时打印存储过程正在执行的语句。
- off表示调用存储过程时，pg_stat_activity的query列只打印存储过程调用语句。

默认值：on

update_process_title

参数说明：控制收集因每次服务器接收到一个新的SQL语句时而产生的进程名称更新的统计数据。

进程名称可以通过ps命令进行查看，在Windows下通过任务管理器查看。

该参数属于INTERNAL类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启收集功能。
- off表示关闭收集功能。

默认值：off

stats_temp_directory

参数说明：设置存储临时统计数据的目录，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

将其设置为一个基于RAM的文件系统目录会减少实际的I/O开销并可以提升其性能。

取值范围：字符串

默认值：pg_stat_tmp

track_thread_wait_status_interval

参数说明：用来定期收集thread状态信息的时间间隔。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0 ~ 1440，单位为min。

默认值：30min

enable_save_datachanged_timestamp

参数说明：确定是否收集insert/update/delete, exchange/truncate/drop partition操作对表数据改动的时间。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许收集相关操作对表数据改动的时间。
- off表示禁止收集相关操作对表数据改动的时间。

默认值：on

track_sql_count

参数说明：控制对每个会话中当前正在执行的SELECT、INSERT、UPDATE、DELETE、MERGE INTO语句进行计数的统计数据。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启计数功能。
- off表示关闭计数功能。

默认值：on

说明

当参数关闭时，查询视图的结果为0行。

18.3.11.2 性能统计

在数据库运行过程中，会涉及到锁的访问、磁盘I/O操作、无效消息的处理，这些操作都可能是数据库的性能瓶颈，通过GaussDB提供的性能统计方法，可以方便定位性能问题。

输出性能统计日志

参数说明：对每条查询，以下4个选项控制在服务器日志里记录相应模块的性能统计数据，具体含义如下：

- log_parser_stats控制在服务器日志里记录解析器的性能统计数据。
- log_planner_stats控制在服务器日志里记录查询优化器的性能统计数据。
- log_executor_stats控制在服务器日志里记录执行器的性能统计数据。
- log_statement_stats控制在服务器日志里记录整个语句的性能统计数据。

这些参数只能辅助管理员进行粗略分析，类似Linux中的操作系统工具getrusage()。

这些参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

- log_statement_stats记录总的语句统计数据，而其他的只记录针对每个模块的统计数据。
- log_statement_stats不能和其他任何针对每个模块统计的选项一起打开。

取值范围：布尔型

- on表示开启记录性能统计数据的功能。
- off表示关闭记录性能统计数据的功能。

默认值：off

18.3.11.3 热点 key 统计

分布式架构下，如果应用短时间内集中访问某一节点，会导致该节点资源使用过高，从而影响数据库正常运行。GaussDB提供的热点key快速检测功能可以用来快速定位是否有热点key以及热点key的分布来帮助定位问题。

enable_hotkeys_collection

参数说明：开关打开后，自动对数据库内的被访问的键值进行统计。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

说明

如果是通过gs_guc set 方式设置参数，需要重启数据库使得GUC参数生效，重启数据库时会清理热点key信息。

当GUC参数关闭时，调用热点key查询结果将会返回空，并且提示GUC参数关闭。但是开关关闭时，热点key清理接口仍可以正常使用。

取值范围：布尔型

- on表示开启计数功能。
- off表示关闭计数功能。

默认值：off

18.3.12 自动清理

系统自动清理线程（autovacuum）自动执行VACUUM和ANALYZE命令，回收被标识为删除状态的记录空间，并更新表的统计数据。

autovacuum

参数说明：控制数据库自动清理线程（autovacuum）的启动。自动清理线程运行的前提是将**track_counts**设置为on。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

说明

- 如果希望系统在故障恢复后，具备自动清理两阶段事务的功能，请将autovacuum设置为on;
- 当设置autovacuum为on，**autovacuum_max_workers**为0时，表示系统不会自动进行autovacuum，只会在故障恢复后，自动清理两阶段事务；
- 当设置autovacuum为on，**autovacuum_max_workers**大于0时，表示系统不仅在故障恢复后，自动清理两阶段事务，并且还可以自动清理线程。

须知

即使此参数设置为off，当事务ID回绕即将发生时，数据库也会自动启动自动清理线程。对于create/drop database发生异常时，可能有的节点提交或回滚，有的节点未提交（prepared状态），此时系统不能自动修复，需要手动修复，修复步骤：

1. 使用gs_clean工具（-N参数）查询出异常两阶段事务的xid以及处于prepared的节点；
2. 登录事务处于prepared状态的节点，系统管理员连接一个可用的数据库（如postgres），执行语句set xc_maintenance_mode = on；
3. 根据事务全局状态提交或者回滚此两阶段事务（如提交语句；回滚语句）。

取值范围：布尔型

- on表示开启数据库自动清理线程。
- off表示关闭数据库自动清理线程。

默认值：on

autovacuum_mode

参数说明：该参数仅在autovacuum设置为on的场景下生效，它控制autoanalyze或autovacuum的打开情况。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：枚举类型

- analyze表示只做autoanalyze。
- vacuum表示只做autovacuum。
- mix表示autoanalyze和autovacuum都做。
- none表示二者都不做。

默认值：mix

autoanalyze_timeout

参数说明：设置autoanalyze的超时时间。在对某张表做autoanalyze时，如果该表的analyze时长超过了autoanalyze_timeout，则自动取消该表此次analyze。

这里的时间检查不能保证完全精确，原则上要保证各个CN上统计信息一致，因此在CN间同步信息时，即便超时也不会被打断。这导致实际的执行时间有可能超过用户定义的时间。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，单位是s，0~2147483，0表示不超时。

默认值：5min（即300s）

autovacuum_io_limits

参数说明：控制autovacuum进程每秒触发I/O的上限。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，-1～1073741823。其中-1表示不控制，而是使用系统默认控制组。

默认值：-1

log_autovacuum_min_duration

参数说明：当自动清理的执行时间大于或者等于某个特定的值时，向服务器日志中记录自动清理执行的每一步操作。设置此选项有助于追踪自动清理的行为。

举例如下：将log_autovacuum_min_duration设置为250ms，表示记录所有运行大于或者等于250ms的自动清理命令的相关信息。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，最小值为-1，最大值为2147483647，单位为毫秒。

- 当参数设置为0时，表示所有的自动清理操作都记录到日志中。
- 当参数设置为-1时，表示所有的自动清理操作都不记录到日志中。
- 当参数设置为大于0时，当由于锁冲突的存在导致一个自动清理操作被跳过，记录一条消息。

默认值：-1

autovacuum_max_workers

参数说明：设置能同时运行的自动清理线程的最大数量，该参数的取值上限与GUC参数max_connections和job_queue_processes大小有关。

该参数属于POSTMASTER类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，最小值为0（表示不会自动进行autovacuum），理论最大值为262143，实际最大值为动态值，计算公式为“262143 - max_inner_tool_connections - max_connections - job_queue_processes - 辅助线程数 – autovacuum的launcher线程数 - 1”，其中辅助线程数和autovacuum的launcher线程数由两个宏来指定，当前版本的默认值分别为20和2。

默认值：3

autovacuum_naptime

参数说明：设置两次自动清理操作的时间间隔。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，单位为s，最小值为1，最大值为2147483。

默认值：10min（即600s）

autovacuum_vacuum_threshold

参数说明：设置触发VACUUM的阈值。当表上被删除或更新的记录数超过设定的阈值时就会对这个表执行VACUUM操作。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，最小值为0，最大值为2147483647。

默认值：50

autovacuum_analyze_threshold

参数说明：设置触发ANALYZE操作的阈值。当表上被删除、插入或更新的记录数超过设定的阈值时才会对这个表执行ANALYZE操作。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，最小值为0，最大值为2147483647。

默认值：50

autovacuum_vacuum_scale_factor

参数说明：设置触发一个VACUUM时增加到autovacuum_vacuum_threshold的表大小的缩放系数。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：浮点型，0.0～100.0

默认值：0.2

autovacuum_analyze_scale_factor

参数说明：设置触发一个ANALYZE时增加到autovacuum_analyze_threshold的表大小的缩放系数。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：浮点型，0.0～100.0

默认值：0.1

autovacuum_freeze_max_age

参数说明：设置事务内的最大时间，使得表的pg_class.relfrozenxid字段在VACUUM操作执行之前被写入。

- VACUUM也可以删除pg_clog/子目录中的旧文件。
- 即使自动清理线程被禁止，系统也会调用自动清理线程来防止循环重复。

该参数属于POSTMASTER类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：**长**整型，100 000～576 460 752 303 423 487

默认值：4000000000

autovacuum_vacuum_cost_delay

参数说明：设置在自动VACUUM操作里使用的开销延迟数值。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，-1～100，单位为毫秒（ms）。其中-1表示使用常规的vacuum_cost_delay。

默认值：20ms

autovacuum_vacuum_cost_limit

参数说明：设置在自动VACUUM操作里使用的开销限制数值。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，-1～10000。其中-1表示使用常规的vacuum_cost_limit。

默认值：-1

twophase_clean_workers

参数说明：该参数用来控制内核调度gs_clean工具的并发清理数。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，1～10

默认值：3

defer_csn_cleanup_time

参数说明：用来指定本地回收时间间隔，单位为毫秒（ms）。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，0~2147483647。

默认值：5s（即5000ms）

18.3.13 客户端连接缺省设置

18.3.13.1 语句行为

介绍SQL语句执行过程的相关默认参数。

search_path

参数说明：当一个被引用对象没有指定模式时，此参数设置模式搜索顺序。它的值由一个或多个模式名构成，不同的模式名用逗号隔开。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

- 当前会话如果存放临时表的模式时，可以使用别名pg_temp将它列在搜索路径中，如'pg_temp，public' 。 存放临时表的模式始终会作为第一个被搜索的对象，排在pg_catalog和search_path中所有模式的前面，即具有第一搜索优先级。

建议用户不要在search_path中显示设置pg_temp。如果在search_path中指定了pg_temp，但不是在最前面，系统会提示设置无效，pg_temp仍被优先搜索。通过使用别名pg_temp，系统只会在存放临时表的模式中搜索表、视图和数据类型这样的数据库对象，不会在里面搜索函数或运算符这样的数据库对象。

- 系统表所在的模式pg_catalog，总是排在search_path中指定的所有模式前面被搜索，即具有第二搜索优先级（pg_temp具有第一搜索优先级）。建议用户不要在search_path中显式设置pg_catalog。如果在search_path中指定了pg_catalog，但不是在最前面，系统会提示设置无效，pg_catalog仍被第二优先搜索。
- 当没有指定一个特定模式而创建一个对象时，它们被放置到以search_path为命名的第一个有效模式中。当搜索路径为空时，会报错误。
- 通过SQL函数current_schema可以检测当前搜索路径的有效值。这和检测search_path的值不尽相同，因为current_schema显示search_path中首位有效的模式名称。

取值范围：字符串

说明

- 设置为"$user"，public时，支持共享数据库（没有用户具有私有模式和所有共享使用public），用户私有模式和这些功能的组合使用。可以通过改变默认搜索路径来获得其他效果，无论是全局化的还是私有化的。
- 设置为空串（''）的时候，系统会自动转换成一对双引号。
- 设置的内容中包含双引号，系统会认为是不安全字符，会将每个双引号转换成一对双引号。

默认值："$user",public

说明

$user表示与当前会话用户名同名的模式名，如果这样的模式不存在，$user将被忽略。

current_schema

参数说明：此参数设置当前的模式。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

默认值："$user",public

说明

- $user表示与当前会话用户名同名的模式名，如果这样的模式不存在，$user将被忽略。
- GaussDB开发过程中如需要获取Schema，请使用search_path对应的值，因为Schema是search_path决定的。为了兼容性ORA数据库，current_schema目的只是作为修改search_path的值使用。

default_tablespace

参数说明：当CREATE命令没有明确声明表空间时，所创建对象(表和索引等)的缺省表空间。

- 值是一个表空间的名称或者一个表示使用当前数据库缺省表空间的空字符串。若指定的是一个非默认表空间，用户必须具有它的CREATE权限，否则尝试创建会失败。

- 临时表不使用此参数，可以用**temp_tablespaces**代替。
- 创建数据库时不使用此参数。默认情况下，一个新的数据库从模板数据库继承表空间配置。

  该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串，其中空表示使用默认表空间。

默认值：空

default_storage_nodegroup

参数说明：此参数设置当前的默认建表所在的Node Group，目前只适用普通表。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

- 值为“installation”表示建表会默认建在安装的Node Group上。
- 值为其他字符串表示建表会默认建在设置的Node Group上。

取值范围：字符串

默认值：installation

temp_tablespaces

参数说明：当一个CREATE命令没有明确指定一个表空间时，temp_tablespaces指定了创建临时对象（临时表和临时表的索引）所在的表空间。在这些表空间中创建临时文件用来做大型数据的排序工作。

其值是一系列表空间名的列表。如果列表中有多个表空间时，每次临时对象的创建，GaussDB会在列表中随机选择一个表空间；如果在事务中，连续创建的临时对象被放置在列表里连续的表空间中。如果选择的列表中的元素是一个空串，GaussDB将自动将当前的数据库设为默认的表空间。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串。空字符串表示所有的临时对象仅在当前数据库默认的表空间中创建，请参见**default_tablespace**。

默认值：空

check_function_bodies

参数说明：设置是否在CREATE FUNCTION执行过程中进行函数体字符串的合法性验证。为了避免产生问题（比如避免从转储中恢复函数定义时向前引用的问题），偶尔会禁用验证。开启后主要验证存储过程中PL/SQL的词语法问题，包括数据类型、语句和表达式等，对于其中出现的SQL则在Create阶段不做检查而采用了运行时检查的方式。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示在CREATE FUNCTION执行过程中进行函数体字符串的合法性验证。
- off表示在CREATE FUNCTION执行过程中不进行函数体字符串的合法性验证。

默认值：on

default_transaction_isolation

参数说明：设置默认的事务隔离级别。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

说明

当前版本暂不支持设置默认的事务隔离级别，默认为read committed，请勿自行修改。

取值范围：枚举类型

- read uncommitted表示隔离级别是读未提交。
- read committed表示事务读已提交。
- repeatable read表示事务可重复读。
- serializable，GaussDB目前功能上不支持此隔离级别，等价于repeatable read。

默认值：read committed

default_transaction_read_only

参数说明：设置每个新创建事务是否是只读状态。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

注意

该参数设为on后只读，无法执行dml和写事务。

取值范围：布尔型

- on表示只读状态。
- off表示非只读状态。

默认值：off

default_transaction_deferrable

参数说明：控制每个新事务的默认延迟状态。只读事务或者那些比序列化更加低的隔离级别的事务除外。

GaussDB不支持可串行化的隔离级别，因此，该参数无实际意义。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示默认延迟。
- off表示默认不延迟。

默认值：off

session_replication_role

参数说明：控制当前会话与复制相关的触发器和规则的行为。

该参数属于SUSET类型参数，请参考表18-1中对应设置方法进行设置。

须知

设置此参数会丢弃之前任何缓存的执行计划。

取值范围：枚举类型

- origin表示从当前会话中复制插入、删除、更新等操作。
- replica表示从其他地方复制插入、删除、更新等操作到当前会话。
- local表示函数执行复制时会检测当前登录数据库的角色并采取相应的操作。

默认值：origin

statement_timeout

参数说明：当语句执行时间超过该参数设置的时间（从服务器收到命令时开始计时）时，该语句将会报错并退出执行。

该参数属于USERSET类型参数，请参考表18-1中对应设置方法进行设置。默认值0代表该参数不生效。

取值范围：整型，0 ~ 2147483647，单位为毫秒。

默认值：0

vacuum_freeze_min_age

参数说明：指定VACUUM在扫描一个表时用于判断是否用FrozenXID替换记录的xmin字段(在同一个事务中)。

该参数属于USERSET类型参数，请参考表18-1中对应设置方法进行设置。

取值范围：整型，0～576 460 752 303 423 487

说明

尽管随时可以将此参数设为上述取值范围之间的任意值，但是，VACUUM将默认其有效值范围限制在autovacuum_freeze_max_age的50%以内。

默认值：2000000000

vacuum_freeze_table_age

参数说明：指定VACUUM对全表的扫描冻结元组的时间。如果当前事务号与表pg_class.relfrozenxid64字段的差值已经大于参数指定的时间时，VACUUM对全表进行扫描。

该参数属于USERSET类型参数，请参考表18-1中对应设置方法进行设置。

取值范围：整型，0～576 460 752 303 423 487

说明

尽管随时可以将此参数设为上述取值范围之间的值，但是，VACUUM将默认其有效值范围限制在**autovacuum_freeze_max_age**的95%以内。定期的手动VACUUM可以在对该表的反重叠自动清理启动之前运行。

默认值：4000000000

bytea_output

参数说明：设置bytea类型值的输出格式。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：枚举类型

- hex：将二进制数据编码为每字节2位十六进制数字。
- escape：传统化的PostgreSQL格式。采用以ASCII字符序列表示二进制串的方法，同时将那些无法表示成ASCII字符的二进制串转换成特殊的转义序列。

默认值：hex

xmlbinary

参数说明：设置二进制值是如何在XML中进行编码的。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

说明

此参数目前不支持XML类型数据。

取值范围：枚举类型

- base64
- hex

默认值：base64

xmloption

参数说明：当XML和字符串值之间进行转换时，设置document或content是否是隐含的。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

说明

此参数目前不支持XML类型数据。

取值范围：枚举类型

- document：表示HTML格式的文档。
- content：普通的字符串。

默认值：content

max_compile_functions

参数说明：设置服务器存储的函数编译结果的最大数量。存储过多的函数和存储过程的编译结果可能占用很大内存。将此参数设置为一个合理的值，有助于减少内存占用，提升系统性能。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1 ~ 2147483647。

默认值：1000

18.3.13.2 区域和格式化

介绍时间格式设置的相关参数。

DateStyle

参数说明：设置日期和时间值的显示格式，以及有歧义的输入值的解析规则。

这个变量包含两个独立的加载部分：输出格式声明（ISO、Postgres、SQL、German）和输入输出的年/月/日顺序（DMY、MDY、YMD）。这两个可以独立设置或者一起设置。关键字Euro和European等价于DMY；关键字US、NonEuro、NonEuropean等价于MDY 。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

默认值：'ISO, MDY'

说明

gs_initdb会将这个参数初始化成与**lc_time**一致的值。

若该参数通过执行gs_guc reload修改时，如果当前节点上的某个session的连接不是来自于客户端，而是来自于该节点所属集群上的其他节点，那么执行gs_guc reload后该参数在该session上不会立即生效，需与连接节点断开连接后重新连接才会生效。

设置建议：优先推荐使用ISO格式。Postgres、SQL和German均采用字母缩写的形式来表示时区，例如“EST、WST、CST”等。这些缩写可同时指代不同的时区，比如CST可同时代表美国中部时间(Central Standard Time (USA) UT-6:00)、澳大利亚中部时间(Central Standard Time (Australia) UT+9:30)、中国标准时间(China Standard Time UT+8:00)。这种情况下在时区转化时可能会得不到正确的结果，从而引发其他问题。

IntervalStyle

参数说明：设置区间值的显示格式。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：枚举类型

- sql_standard表示产生与SQL标准规定匹配的输出。
- postgres表示产生与PostgreSQL 8.4版本相匹配的输出，当**DateStyle**参数被设为ISO时。

- postgres_verbose表示产生与PostgreSQL 8.4版本相匹配的输出，当**DateStyle**参数被设为non_ISO时。
- iso_8601表示产生与在ISO 8601中定义的“格式与代号”相匹配的输出。
- oracle表示产生于Oracle中与numtodsinterval函数相匹配的输出结果，详细请参考《开发指南》中“SQL参考 > 函数和操作符 > 时间和日期处理函数和操作符”章节的“numtodsinterval”。

须知

IntervalStyle参数也会影响不明确的间隔输入的说明。

若该参数通过执行gs_guc reload修改时，如果当前节点上的某个session的连接不是来自于客户端，而是来自于该节点所属集群上的其他节点，那么执行gs_guc reload后该参数在该session上不会立即生效，需与连接节点断开连接后重新连接才会生效。

默认值：postgres

TimeZone

参数说明：设置显示和解释时间类型数值时使用的时区。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串，可查询视图PG_TIMEZONE_NAMES获得，具体可参见《开发指南》中“系统表和系统视图 > 系统视图 > PG_TIMEZONE_NAMES”章节。

默认值：

说明

gs_initdb将设置一个与其系统环境一致的时区值。

若该参数通过执行gs_guc reload修改时，如果当前节点上的某个session的连接不是来自于客户端，而是来自于该节点所属集群上的其他节点，那么执行gs_guc reload后该参数在该session上不会立即生效，需与连接节点断开连接后重新连接才会生效。

timezone_abbreviations

参数说明：设置服务器接受的时区缩写值。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串，可查询视图pg_timezone_names获得。

默认值：Default

说明

Default表示通用时区的缩写。但也有其他诸如 'Australia' 和 'India' 等用来定义特定的安装。

extra_float_digits

参数说明：这个参数为浮点数值调整显示的数据位数，浮点类型包括float4、float8 以及几何数据类型。参数值加在标准的数据位数上（FLT_DIG或DBL_DIG中合适的）。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，-15～3

说明

- 设置为3，表示包括部分关键的数据位。这个功能对转储那些需要精确恢复的浮点数据特别有用。
- 设置为负数，表示消除不需要的数据位。

默认值：0

client_encoding

参数说明：设置客户端的字符编码类型。

请根据前端业务的情况确定。尽量客户端编码和服务器端编码一致，提高效率。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：兼容PostgreSQL所有的字符编码类型。其中UTF8表示使用数据库的字符编码类型。

说明

- 使用命令locale -a查看当前系统支持的区域和相应的编码格式，并可以选择进行设置。
- 默认情况下，gs_initdb会根据当前的系统环境初始化此参数，通过locale命令可以查看当前的配置环境。
- 参数建议保持默认值，不建议通过gs_guc工具或其他方式直接在postgresql.conf文件中设置client_encoding参数，即使设置也不会生效，以保证集群内部通信编码格式一致。

默认值：UTF8

推荐值：SQL_ASCII/UTF8

lc_messages

参数说明：设置信息显示的语言。

- 可接受的值是与系统相关的。
- 在一些系统上，这个区域范畴并不存在，不过仍然允许设置这个变量，只是不会有任何效果。同样，也有可能是所期望的语言的翻译信息不存在。在这种情况下，用户仍然能看到英文信息。

  该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

说明

- 使用命令locale -a查看当前系统支持的区域和相应的编码格式，并可以选择进行设置。
- 默认情况下，gs_initdb会根据当前的系统环境初始化此参数，通过locale命令可以查看当前的配置环境。

默认值：C

lc_monetary

参数说明：设置货币值的显示格式，影响to_char之类的函数的输出。可接受的值是系统相关的。

该参数属于USERSET类型参数，请参考表18-1中对应设置方法进行设置。

取值范围：字符串

说明

- 使用命令locale -a查看当前系统支持的区域和相应的编码格式，并可以选择进行设置。
- 默认情况下，gs_initdb会根据当前的系统环境初始化此参数，通过locale命令可以查看当前的配置环境。
- 若该参数通过执行gs_guc reload修改时，如果当前节点上的某个session的连接不是来自于客户端，而是来自于该节点所属集群上的其他节点，那么执行gs_guc reload后该参数在该session上不会立即生效，需与连接节点断开连接后重新连接才会生效。

默认值：C

lc_numeric

参数说明：设置数值的显示格式，影响to_char之类的函数的输出。可接受的值是系统相关的。

该参数属于USERSET类型参数，请参考表18-1中对应设置方法进行设置。

取值范围：字符串

说明

- 使用命令locale -a查看当前系统支持的区域和相应的编码格式，并可以选择进行设置。
- 默认情况下，gs_initdb会根据当前的系统环境初始化此参数，通过locale命令可以查看当前的配置环境。
- 若该参数通过执行gs_guc reload修改时，如果当前节点上的某个session的连接不是来自于客户端，而是来自于该节点所属集群上的其他节点，那么执行gs_guc reload后该参数在该session上不会立即生效，需与连接节点断开连接后重新连接才会生效。

默认值：C

lc_time

参数说明：设置时间和区域的显示格式，影响to_char之类的函数的输出。可接受的值是系统相关的。

该参数属于USERSET类型参数，请参考表18-1中对应设置方法进行设置。

取值范围：字符串

说明

- 使用命令locale -a查看当前系统支持的区域和相应的编码格式，并可以选择进行设置。
- 默认情况下，gs_initdb会根据当前的系统环境初始化此参数，通过locale命令可以查看当前的配置环境。
- 若该参数通过执行gs_guc reload修改时，如果当前节点上的某个session的连接不是来自于客户端，而是来自于该节点所属集群上的其他节点，那么执行gs_guc reload后该参数在该session上不会立即生效，需与连接节点断开连接后重新连接才会生效。

默认值：C

18.3.13.3 其他缺省

主要介绍数据库系统默认的库加载参数。

dynamic_library_path

参数说明：设置数据查找动态加载的共享库文件的路径。当需要打开一个可以动态装载的模块并且在CREATE FUNCTION或LOAD命令里面声明的名称没有目录部分时，系统将搜索这个目录以查找声明的文件，仅sysadmin用户可以访问。

用于dynamic_library_path的数值必须是一个冒号分隔（Windows下是分号分隔）的绝对路径列表。当一个路径名称以特殊变量$libdir为开头时，会替换为GaussDB发布提供的模块安装路径。例如：

```
dynamic_library_path = '/usr/local/lib/postgresql:/opt/testgs/lib:$libdir'
```

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

说明

设置为空字符串，表示关闭自动路径搜索。

默认值：$libdir

local_preload_libraries

参数说明：指定一个或多个共享库，它们在开始连接前预先加载。多个加载库之间用逗号分隔，除了双引号，所有的库名都转换为小写。

- 并非只有系统管理员才能更改此选项，因此只能加载安装的标准库目录下plugins子目录中的库文件，数据库管理员有责任确保该目录中的库都是安全的。local_preload_libraries中指定的项可以明确含有该目录，例如$libdir/plugins/mylib；也可以仅指定库的名称，例如mylib（等价于$libdir/plugins/mylib）。
- 与shared_preload_libraries不同，在会话开始之前加载模块与在会话中使用到该模块的时候临时加载相比并不具有性能优势。相反，这个特性的目的是为了调试或者测量在特定会话中不明确使用LOAD加载的库。例如针对某个用户将该参数设为ALTER USER SET来进行调试。
- 当指定的库未找到时，连接会失败。
- 每一个支持GaussDB的库都有一个“magic block”用于确保兼容性，因此不支持GaussDB的库不能通过这个方法加载。

该参数属于BACKEND类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

默认值：空

18.3.14 锁管理

在GaussDB中，并发执行的事务由于竞争资源会导致死锁。本节介绍的参数主要管理事务锁的机制。

deadlock_timeout

参数说明：设置死锁超时检测时间，以毫秒为单位。当申请的锁超过设定值时，系统会检查是否产生了死锁。该参数仅针对常规锁生效。

- 死锁的检查代价是比较高的，服务器不会在每次等待锁的时候都运行这个过程。在系统运行过程中死锁是不经常出现的，因此在检查死锁前只需等待一个相对较

短的时间。增加这个值就减少了无用的死锁检查浪费的时间，但是会减慢真正的死锁错误报告的速度。在一个负载过重的服务器上，用户可能需要增大它。这个值的设置应该超过事务持续时间，这样就可以减少在锁释放之前就开始死锁检查的问题。

- 如果要通过设置**log_lock_waits**来将查询执行过程中的锁等待耗时信息写入日志，请确保**log_lock_waits**的设置值小于deadlock_timeout的设置值（或默认值）。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1~2147483647，单位为毫秒（ms）。

默认值：1s

lockwait_timeout

参数说明：控制单个锁的最长等待时间。当申请的锁等待时间超过设定值时，系统会报错。该参数仅针对常规锁生效。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0 ~ 2147483647，单位为毫秒（ms）。

默认值：20min

update_lockwait_timeout

参数说明：允许并发更新参数开启情况下，该参数控制并发更新同一行时单个锁的最长等待时间。当申请的锁等待时间超过设定值时，系统会报错。该参数仅针对常规锁生效。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0 ~ 2147483647，单位为毫秒（ms）。

默认值：120000（2min）

max_locks_per_transaction

参数说明：控制每个事务能够得到对象锁的平均数量。

参数类型：整型

参数单位：无

取值范围：10 ~ 2147483647

默认值：256

设置方式：该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

设置建议：

- 共享锁的哈希表的大小是以假设任意时刻最多只有max_locks_per_transaction*(max_connections+max_prepared_transactions) 个独立的对象需要被锁住为基础进行计算的，进而推导出max_locks_per_transaction >= 业务事务的并发数 * 业务的每个事务加对象锁的

数目/(max_connections+max_prepared_transactions)，计算结果不超过参数取值范围的上限。

- 不超过设定数量的多个对象可以在任意时刻同时被锁定。当在一个事务里面修改很多不同的表时，可能需要提高这个参数值。
- 增大这个参数可能导致GaussDB请求更多的System V共享内存，有可能超过操作系统的缺省配置，导致数据库启动失败。
- 当运行备机时，请将此参数设置不小于主机上的值，否则在备机上查询操作不被允许。

max_pred_locks_per_transaction

参数说明：控制每个事务允许断定锁的最大数量，是一个平均值。

- 共享的断定锁表的大小是以假设任意时刻最多只有max_pred_locks_per_transaction*(max_connections +max_prepared_transactions) 个独立的对象需要被锁住为基础进行计算的。不超过设定数量的多个对象可以在任一时刻同时被锁定。当在一个事务里面修改很多不同的表时，可能需要提高这个默认数值。只能在服务器启动的时候设置。
- 增大这个参数可能导致GaussDB请求更多的System V共享内存，有可能超过操作系统的缺省配置。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，10 ~ 2147483647

默认值：64

gs_clean_timeout

参数说明：控制Coordinator周期性调用gs_clean工具的时间，是一个平均值。

- GaussDB数据库中事务处理使用的是两阶段提交的方法，当有两阶段事务残留时，该事务通常会拿着表级锁，导致其它连接无法加锁，此时需要调用gs_clean工具对集群中两阶段事务进行清理，gs_clean_timeout是控制Coordinator周期性调用gs_clean的时间。
- 增大这个参数可能导致GaussDB周期性调用gs_clean工具的时间延长，导致两阶段事务清理时间延长。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0 ~ 2147483，单位为秒（s）。

默认值：1min

partition_lock_upgrade_timeout

参数说明：在执行某些查询语句的过程中，会需要将分区表上的锁级别由允许读的ExclusiveLock级别升级到读写阻塞的AccessExclusiveLock级别。如果此时已经存在并发的读事务，那么该锁升级操作将阻塞等待。partition_lock_upgrade_timeout为尝试锁升级的等待超时时间。

- 在分区表上进行MERGE PARTITION和CLUSTER PARTITION操作时，都利用了临时表进行数据重排和文件交换，为了最大程度提高分区上的操作并发度，在数据重排阶段给相关分区加锁ExclusiveLock，在文件交换阶段加锁AccessExclusiveLock。

- 常规加锁方式是等待加锁，直到加锁成功，或者等待时间超过**lockwait_timeout**发生超时失败。
- 在分区表上进行MERGE PARTITION或CLUSTER PARTITION操作时，进入文件交换阶段需要申请加锁AccessExclusiveLock，加锁方式是尝试性加锁，加锁成功了则立即返回，不成功则等待50ms后继续下次尝试，加锁超时时间使用会话级设置参数partition_lock_upgrade_timeout。
- 特殊值：若partition_lock_upgrade_timeout取值-1，表示无限等待，即不停的尝试锁升级，直到加锁成功。

  该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，最小值-1，最大值3000，单位为秒（s）。

默认值：1800

fault_mon_timeout

参数说明：轻量级死锁检测周期。该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，最小值0，最大值1440，单位为分钟（min）

默认值：5min

enable_online_ddl_waitlock

参数说明：控制DDL是否会阻塞等待pg_advisory_lock/pgxc_lock_for_backup等集群锁。主要用于OM在线操作场景，不建议用户设置。

该参数属于SIGHUP类型参数，参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启。
- off表示关闭。

默认值：off

xloginsert_locks

参数说明：控制用于并发写预写式日志锁的个数。主要用于提高写预写式日志的效率。

该参数属于POSTMASTER类型参数，参考**表18-1**中对应设置方法进行设置。

取值范围：整型，最小值1，最大值1000

默认值：16

num_internal_lock_partitions

参数说明：控制内部轻量级锁分区的个数。主要用于各类场景的性能调优。内容以关键字和数字的KV方式组织，各个不同类型锁之间以逗号隔开。先后顺序对设置结果不影响，例如“CLOG_PART=256,CSNLOG_PART=512”等同于“CSNLOG_PART=512,CLOG_PART=256”。重复设置同一关键字时，以最后一次设置为准，例如“CLOG_PART=256,CLOG_PART=2”，设置的结果为CLOG_PART=2。当没有设置关键字时，则为默认值，各类锁的使用描述和最大、最小、默认值如下。

- CLOG_PART：CLOG文件控制器的个数，增大该值可以提高CLOG日志写入效率，提升事务提交性能，但是会增大内存使用；减小该值会减少相应内存使用，但可能使得CLOG日志写入冲突变大，影响性能。最小值为1，最大值为256。
- CSNLOG_PART：CSNLOG文件控制器的个数，增大该值可以提高CSNLOG日志写入效率，提升事务提交性能，但是会增大内存使用；减小该值会减少相应内存使用，但可能使得CSNLOG日志写入冲突变大，影响性能。最小值为1，最大值为512。
- LOG2_LOCKTABLE_PART：常规锁表锁分区个数的2对数，增大该值可以提升正常流程常规锁获取锁的并行度，但是可能增加锁转移和锁消除时的耗时，对于等待事件在LockMgrLock时，可以调大该锁增加性能。最小值4，即锁分区数为16；最大值为16，即锁分区数为65536。
- TWOPHASE_PART：两阶段事务锁的分区数，调大该值可以提高两阶段事务提交的并发数。最小值为1，最大值为64。
- FASTPATH_PART：每个线程可以不通过主锁表拿锁的最大锁个数，对于分区表读取、更新、插入、删除操作且等待事件在LockMgrLock时，可以通过调大该值避免获取LockMgrLock提升性能，建议调整数量大于等于分区数*（1+本地索引数量）+全局索引数量+10，调大该值会额外增加内存。最小值为20，最大值为10000。

该参数属于POSTMASTER类型参数，参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

默认值：

- CLOG_PART：256
- CSNLOG_PART：512
- LOG2_LOCKTABLE_PART：4
- TWOPHASE_PART：1
- FASTPATH_PART：20

18.3.15 版本和平台兼容性

18.3.15.1 历史版本兼容性

GaussDB介绍数据库的向下兼容性和对外兼容性特性的参数控制。数据库系统的向后兼容性能够为对旧版本的数据库应用提供支持。本节介绍的参数主要控制数据库的向后兼容性。

array_nulls

参数说明：控制数组输入解析器是否将未用引用的NULL识别为数组的一个NULL元素。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许向数组中输入空元素。
- off表示向下兼容旧式模式。仍然能够创建包含NULL值的数组。

默认值：on

backslash_quote

参数说明：控制字符串文本中的单引号是否能够用\'表示。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

在字符串文本符合SQL标准的情况下，\没有任何其他含义。这个参数影响的是如何处理不符合标准的字符串文本，包括明确的字符串转义语法是（E'...'）。

取值范围：枚举类型

- on表示一直允许使用\'表示。
- off表示拒绝使用\'表示。
- safe_encoding表示仅在客户端字符集编码不会在多字节字符末尾包含\的ASCII值时允许。

默认值：safe_encoding

default_with_oids

参数说明：在没有声明WITH OIDS和WITHOUT OIDS的情况下，这个选项控制在新创建的表中CREATE TABLE和CREATE TABLE AS是否包含一个OID字段。它还决定SELECT INTO创建的表里面是否包含OID 。

不推荐在用户表中使用OID，故默认设置为off。需要带有OID字段的表应该在创建时声明WITH OIDS 。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示在新创建的表中CREATE TABLE和CREATE TABLE AS可以包含一个OID字段。
- off表示在新创建的表中CREATE TABLE和CREATE TABLE AS不可以包含一个OID字段。

默认值：off

escape_string_warning

参数说明：警告在普通字符串中直接使用反斜杠转义。

- 如果需要使用反斜杠作为转义，可以调整为使用转义字符串语法(E'...')来做转义，因为在每个SQL标准中，普通字符串的默认行为现在将反斜杠作为一个普通字符。
- 这个变量可以帮助定位需要改变的代码。
- 使用E转义会导致部分场景下日志记录不全。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

默认值：on

lo_compat_privileges

参数说明：控制是否启动对大对象权限检查的向后兼容模式。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

on表示当读取或修改大对象时禁用权限检查，与PostgreSQL 9.0以前的版本兼容。

off表示启用大对象的权限检查。

默认值：off

quote_all_identifiers

参数说明：当数据库生成SQL时，此选项强制引用所有的标识符（包括非关键字）。这将影响到EXPLAIN的输出及函数的结果，例如pg_get_viewdef。详细说明请参见gs_dump的--quote-all-identifiers选项。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示打开强制引用。
- off表示关闭强制引用。

默认值：off

sql_inheritance

参数说明：控制继承语义。用来控制继承表的访问策略，off表示各种命令不能访问子表，即默认使用ONLY关键字。这是为了兼容7.1之前版本而设置的。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示可以访问子表。
- off表示不访问子表。

默认值：on

standard_conforming_strings

参数说明：控制普通字符串文本（'...'）中是否按照SQL标准把反斜扛当普通文本。

- 应用程序通过检查这个参数可以判断字符串文本的处理方式。
- 建议明确使用转义字符串语法（E'...'）来转义字符。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示打开控制功能。

- off表示关闭控制功能。

默认值：on

synchronize_seqscans

参数说明：控制启动同步的顺序扫描。在大约相同的时间内并行扫描读取相同的数据块，共享I/O负载。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示扫描可能从表的中间开始，然后选择"环绕"方式来覆盖所有的行，为了与已经在进行中的扫描活动同步。这可能会造成没有用ORDER BY子句的查询得到行排序造成不可预测的后果。
- off表示确保顺序扫描是从表头开始的。

默认值：on

enable_beta_features

参数说明：控制开启某些非正式发布的特性，仅用于POC验证，例如GDS表关联操作。这些特性属于延伸特性，建议客户谨慎开启，在某些功能场景下可能存在问题。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启这些功能受限的特性，保持前向兼容。但某些场景可能存在功能上的问题。
- off表示禁止使用这些特性。

默认值：off

18.3.15.2 平台和客户端兼容性

很多平台都使用数据库系统，数据库系统的对外兼容性给平台提供了很大的方便。

transform_null_equals

参数说明：控制表达式expr = NULL（或NULL = expr）当做expr IS NULL处理。如果expr得出NULL值则返回真，否则返回假。

- 正确的SQL标准兼容的expr = NULL总是返回NULL（未知）。
- Microsoft Access里的过滤表单生成的查询使用expr = NULL来测试空值。打开这个选项，可以使用该接口来访问数据库。

该参数属于USERSET类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：布尔型

- on表示控制表达式expr = NULL（或NULL = expr）当做expr IS NULL处理。
- off表示不控制，即expr = NULL总是返回NULL（未知）。

默认值：off

说明

新用户经常在涉及NULL的表达式上语义混淆，故默认值设为off。

support_extended_features

参数说明：控制是否支持数据库的扩展特性。

该参数属于POSTMASTER类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：布尔型

- on表示支持数据库的扩展特性。
- off表示不支持数据库的扩展特性。

默认值：off

lastval_supported

参数说明：控制是否可以使用lastval函数。

该参数属于POSTMASTER类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：布尔型

- on表示支持lastval函数，同时nextval函数不支持下推。
- off表示不支持lastval函数，同时nextval函数可以下推。

默认值：off

sql_compatibility

参数说明：控制数据库的SQL语法和语句行为同哪一个主流数据库兼容。该参数属于INTERNAL类型参数，用户无法修改，只能查看。

取值范围：枚举型

- ORA表示同oracle兼容。
- TD表示同Teradata兼容。
- MYSQL表示同MySQL兼容。
- PG表示同PostgreSQL兼容。

默认值：MYSQL

须知

- 该参数只能在执行CREATE DATABASE命令创建数据库的时候设置。
- 在数据库中，该参数只能是确定的一个值，要么始终设置为ORA，要么始终设置为TD，请勿任意改动，否则会导致数据库行为不一致。

behavior_compat_options

参数说明：数据库兼容性行为配置项，该参数的值由若干个配置项用逗号隔开构成。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

默认值：""

说明

- 当前只支持**表18-11**。
- 配置多个兼容性配置项时，相邻配置项用逗号隔开，例如：set behavior_compat_options='end_month_calculate,display_leading_zero';

表 18-11 兼容性配置项

兼容性配置项	兼容性行为控制
display_leading_zero	浮点数显示配置项。控制数值类型中char、character、nchar、varchar、character varying、varchar2、nvarchar2、text、clob等所有字符串类型和float4、float8、numeric等任意精度类型的小数点前零显示。并且length计算数字长度同步显示。 • 不设置此配置项时，对于-1~0和0~1之间的小数，不显示小数点前的0。比如： openGauss=# select 0.1231243 as a, 0.1231243::numeric as b,0.1231243::integer(10,3) as c, length(0.1242343) as d; a | b | c | d ----------+----------+------+--- .1231243 | .1231243 | .123 | 8 (1 row) • 设置此配置项时，对于-1~0和0~1之间的小数，显示小数点前的0。比如： openGauss=# select 0.1231243 as a, 0.1231243::numeric as b,0.1231243::integer(10,3) as c, length(0.1242343) as d; a | b | c | d -----------+-----------+-------+--- 0.1231243 | 0.1231243 | 0.123 | 9 (1 row)

兼容性配置项	兼容性行为控制
end_month_calculate	add_months函数计算逻辑配置项。 假定函数add_months的两个参数分别为param1和param2，param1的月份和param2的和为月份result。 ● 不设置此配置项时，如果param1的日期（Day字段）为月末，并且param1的日期（Day字段）比result月份的月末日期小，计算结果中的日期字段（Day字段）和param1的日期字段保持一致。比如， openGauss=# select add_months('2018-02-28',3) from *sys_dummy*; add_months --------------------- 2018-05-28 00:00:00 (1 row) ● 设置此配置项时，如果param1的日期（Day字段）为月末，并且param1的日期（Day字段）比result月份的月末日期比小，计算结果中的日期字段（Day字段）和result的月末日期保持一致。比如， openGauss=# select add_months('2018-02-28',3) from *sys_dummy*; add_months --------------------- 2018-05-31 00:00:00 (1 row)
compat_analyze_sample	analyze采样行为配置项。 设置此配置项时，会优化analyze的采样行为，主要体现在analyze时全局采样会更精确的控制在3万条左右，更好的控制analyze时Coordinator端的内存消耗，保证analyze性能的稳定性。
bind_schema_tablespace	绑定模式与同名表空间配置项。 如果存在与模式名sche_name相同的表空间名，那么如果设置search_path为sche_name，default_tablespace也会同步切换到sche_name。
bind_procedure_searchpath	未指定模式名的存储过程中的数据库对象的搜索路径配置项。 在存储过程中如果不显示指定模式名，会优先在存储过程所属的模式下搜索。 如果找不到，则有两种情况： ● 若不设置此参数，报错退出。 ● 若设置此参数，按照search_path中指定的顺序继续搜索。如果还是找不到，报错退出。

兼容性配置项	兼容性行为控制

correct_to_number	控制to_number()结果兼容性的配置项。
	若不设置此配置项，则to_number()函数结果默认与ORA数据库保持一致。
	`openGauss=# select '' AS to_number_14, to_number('34,50','999,99');`
	`ERROR:  invalid data.`
	`CONTEXT:  referenced column: to_number`
	若设置此配置项，则to_number()函数结果与pg11保持一致。
	`openGauss=# select '' AS to_number_14, to_number('34,50','999,99');`
	` to_number_14 | to_number`
	`---------------+-----------`
	`               |      3450`
	`(1 row)`

unbind_divide_bound	控制对整数除法的结果进行范围校验。
	若不设置此配置项，则会对除法结果做范围校验，例如，INT_MIN/(-1)会因为输出结果大于INT_MAX而报越界错误。
	`openGauss=# select (-2147483648)::int4 / (-1)::int4;`
	`ERROR:  integer out of range`
	若设置此配置项，则不需要对除法结果做范围校验，例如，INT_MIN/(-1)可以得到输出结果为INT_MAX+1。
	`openGauss=# select (-2147483648)::int4 / (-1)::int4;`
	`  ?column?`
	`------------`
	` 2147483648`
	`(1 row)`

convert_string_digit_to_numeric	控制是否将表中以字符串形式表示的numeric常量和数字类型做比较时统一都转换为numeric类型再进行比较。
	`openGauss=# create table test1 (c1 int, c2 varchar);`
	`openGauss=# insert into test1 values (2, '1.1');`
	`openGauss=# set behavior_compat_options='';`
	`openGauss=# select * from test1 where c2 > 1;`
	`ERROR:  invalid input syntax for type bigint: "1.1"`
	
	`openGauss=# set behavior_compat_options='convert_string_digit_to_numeric';`
	`openGauss=# select * from test1 where c2 > 1;`
	` c1 | c2`
	`----+-----`
	`  2 | 1.1`
	`(1 row)`

return_null_string	控制函数lpad()和rpad()结果为空字符串''的显示配置项。
	● 不设置此配置项时，空字符串显示为NULL。
	`openGauss=# select length(lpad('123',0,'*')) from *sys_dummy*;`
	`length`
	`--------`
	
	`(1 row)`
	● 设置此配置项时，空字符串显示为''。
	`openGauss=# select length(lpad('123',0,'*')) from *sys_dummy*;`
	`length`
	`--------`
	`0`
	`(1 row)`

兼容性配置项	兼容性行为控制
compat_concat_variadic	控制函数concat()和concat_ws()对variadic类型结果兼容性的配置项。由于MY数据库无variadic类型，所以该选项对MY数据库无影响。 若不设置此配置项，当concat函数参数为variadic类型时，默认ORA数据库和TD数据库兼容模式下结果相同，且与ORA数据库保持一致。 openGauss=# select concat(variadic NULL::int[]) is NULL; ?column? ---------- t (1 row) 若设置此配置项，当concat函数参数为variadic类型时，保留ORA数据库和TD数据库兼容模式下不同的结果形式。 --ORA数据库下： openGauss=# select concat(variadic NULL::int[]) is NULL; ?column? ---------- t (1 row) --TD数据库下： openGauss=# select concat(variadic NULL::int[]) is NULL; ?column? ---------- f (1 row)
merge_update_multi	控制在使用MERGE INTO ... WHEN MATCHED THEN UPDATE（参见《开发指南》中“SQL参考 > SQL语法 > MERGE INTO”章节）和INSERT ... ON DUPLICATE KEY UPDATE（参见《开发指南》中“SQL参考 > SQL语法 > INSERT”章节）时，当目标表中一条目标数据与多条源数据冲突时UPDATE行为。 若设置此配置项，当存在上述场景时，该冲突行将会多次执行UPDATE；否则（默认）报错，即MERGE或INSERT操作失败。
plstmt_implicit_savepoint	控制存储过程中更新语句的执行是否拥有独立的子事务。 若设置此配置项，存储过程中每条更新语句前开启隐式保存点，EXCEPTION块中默认回退到最近的保存点，从而保证只回退失败语句的修改。该选项是为了兼容O数据库的EXCEPTION行为。

兼容性配置项	兼容性行为控制
hide_tailing_zero	numeric显示配置项。不设置此项时，numeric按照指定精度显示；设置此项时，所有输出numeric的场景均隐藏小数点后的末尾0，包括显示指定format精度情况。 例如： openGauss=# set behavior_compat_options='hide_tailing_zero'; openGauss=# select cast(123.123 as numeric(15,10)) as a, to_char(cast(123.123 as numeric(15,10)), '999D999999'); a | to_char ---------+---------- 123.123 | 123.123 (1 row) openGauss=# set behavior_compat_options=''; openGauss=# select cast(123.123 as numeric(15,10)) as a, to_char(cast(123.123 as numeric(15,10)), '999D999999'); a | to_char ----------------+------------- 123.1230000000 | 123.123000 (1 row)
plsql_security_definer	开启此参数后，创建存储过程时默认为定义者权限。

char_coerce_compat	控制char(n)类型向其它变长字符串类型转换时的行为。不设置该参数时，char(n)类型转换其它变长字符串类型时会省略尾部的空格，设置该参数时，转换时不再省略尾部的空格，并且在转换时如果char(n)类型的长度超过其它变长字符串类型时将会报错。该参数仅在sql_compatibility参数的值为ORA时生效，并且开启该参数后无论是隐式转换、显式转换还是通过调用text(bpchar)函数转换类型都不再省略尾部空格。
	openGauss=# set behavior_compat_options='';
	openGauss=# create table tab_1(col1 varchar(3));
	openGauss=# create table tab_2(col2 char(3));
	openGauss=# insert into tab_2 values(' ');
	openGauss=# insert into tab_1 select col2 from tab_2;
	openGauss=# select * from tab_1 where col1 is null;
	col1
	------ 	
	
	(1 row)
	openGauss=# select * from tab_1 where col1=' ';
	col1
	------ 	
	(0 rows)
	openGauss=# delete from tab_1;
	openGauss=# set behavior_compat_options = 'char_coerce_compat';
	openGauss=# insert into tab_1 select col2 from tab_2;
	openGauss=# select * from tab_1 where col1 is null;
	col1
	------ 	
	(0 rows)
	openGauss=# select * from tab_1 where col1=' ';
	col1
	------ 	
	
	(1 row)

兼容性配置项	兼容性行为控制
truncate_numeric_tail_zero	numeric显示配置项。不设置此项时，numeric按照默认精度显示；设置此项时，除去to_char(numeric, format)这种显示设置精度的情况，所有输出numeric的场景均会隐藏小数点后的末尾0。例如： openGauss=#set behavior_compat_options='truncate_numeric_tail_zero'; openGauss=#select cast(123.123 as numeric(15,10)) as a, to_char(cast(123.123 as numeric(15,10)), '999D999999'); a | to_char ---------+------------- 123.123 | 123.123000 (1 row) openGauss=#set behavior_compat_options=''; openGauss=#select cast(123.123 as numeric(15,10)) as a, to_char(cast(123.123 as numeric(15,10)), '999D999999'); a | to_char ----------------+------------- 123.1230000000 | 123.123000 (1 row)
array_count_compat	控制array.count函数，参数开启时，函数返回0，否则返回null。
aformat_regexp_match	控制正则表达式函数的匹配行为。 设置此项，且sql_compatibility参数的值为ORA或MYSQL时，正则表达式的 flags 参数支持的选项含义有变更： 1. 默认不能匹配 '\n' 字符。 2. flags 中包含n选项时， . 能够匹配 '\n' 字符。 3. regexp_replace(source, pattern replacement) 函数替换所有匹配的子串。 4. regexp_replace(source, pattern, replacement, flags) 在flags值为'' 或者null时，返回值为null。 否则，正则表达式的 flags 参数支持的选项含义： 1. 默认能匹配 '\n' 字符。 2. flags 中的 n 选项表示按照多行模式匹配。 3. regexp_replace(source, pattern replacement) 函数仅替换第一个匹配到的子串。 4. regexp_replace(source, pattern, replacement, flags) 在flags值为'' 或者null时，返回值为替换后的字符串。
disable_emptystr2null	开启此参数后，关闭所有字符类型默认将空串转换为null功能。包括text、clob、blob、raw、bytea、varchar、nvarchar2、bpchar、char、name、byteawithoutorderwithqualcol、byteawithoutordercol类型。该参数为逃生参数，非必要用户不要自行设置。

兼容性配置项	兼容性行为控制
proc_uncheck_default_param	函数调用时不检查默认参数省略情况配置项。 ● 不设置此配置项时，调用带有默认参数的函数时，入参从左往右排入函数，如果有非默认参数的入参缺失则会报错。比如： openGauss=# create or replace function test(f1 int, f2 int default 20, f3 int, f4 int default 40, f5 int default 50) return int openGauss-# as openGauss$# begin openGauss$# raise info 'f1:%',f1; openGauss$# raise info 'f2:%',f2; openGauss$# raise info 'f3:%',f3; openGauss$# raise info 'f4:%',f4; openGauss$# raise info 'f5:%',f5; openGauss$# return 1; openGauss$# end; openGauss$# / CREATE FUNCTION openGauss=# select test(1,2); ERROR: function test(integer, integer) does not exist LINE 1: select test(1,2); ^ HINT: No function matches the given name and argument types. You might need to add explicit type casts. CONTEXT: referenced column: test ● 设置此配置项时，调用带有默认参数的函数时，入参从左往右排入函数，允许缺省默认参数个入参，如果有非默认参数的入参缺失，则会用错位的默认值填充该参数。比如： openGauss=# create or replace function test(f1 int, f2 int default 20, f3 int, f4 int default 40, f5 int default 50) return int openGauss-# as openGauss$# begin openGauss$# raise info 'f1:%',f1; openGauss$# raise info 'f2:%',f2; openGauss$# raise info 'f3:%',f3; openGauss$# raise info 'f4:%',f4; openGauss$# raise info 'f5:%',f5; openGauss$# return 1; openGauss$# end; openGauss$# / CREATE FUNCTION openGauss=# select test(1,2); INFO: f1:1 CONTEXT: referenced column: test INFO: f2:2 CONTEXT: referenced column: test INFO: f3:20 CONTEXT: referenced column: test INFO: f4:40 CONTEXT: referenced column: test INFO: f5:50 CONTEXT: referenced column: test test ------ 1 (1 row) 如上，f3被错误的默认值填充。 **警告** 该场景下，非默认参数会被错位的默认值填充。

兼容性配置项	兼容性行为控制

plsql_rollback_keep_user	控制在PL/SQL中rollback和rollback to savepoint是否回退当前用户。当开启此参数时，PL/SQL中rollback将不会修改当前用户。
	示例：
	openGauss=# CREATE USER plsql_rollback1 password 'huawei@123';
	openGauss=# CREATE USER plsql_rollback2 password 'huawei@123';
	openGauss=# GRANT plsql_rollback1 to plsql_rollback2;
	openGauss=# CREATE OR REPLACE procedure plsql_rollback1.p1 () authid definer
	openGauss-# as
	openGauss$# va int;
	openGauss$# begin
	openGauss$# raise info 'current usr:%', current_user;
	openGauss$# rollback;
	openGauss$# raise info 'current usr:%', current_user;
	openGauss$# end;
	openGauss$# /
	CREATE PROCEDURE
	openGauss=# SET session AUTHORIZATION plsql_rollback2 PASSWORD 'huawei@123';
	SET
	openGauss=> SET behavior_compat_options = 'plsql_rollback_keep_user';
	SET
	openGauss=> CALL plsql_rollback1.p1 ();
	INFO: current usr:plsql_rollback1
	INFO: current usr:plsql_rollback1
	p1
	---- 	
	
	(1 row)
	注意
	该参数仅在数据库兼容性为ORA下有效。

兼容性配置项	兼容性行为控制

dynamic_sql_check	开启参数后，对模板SQL中的模板参数个数和using子句中变量个数进行对比，不一致则报错。
	openGauss=# create table tb_t1(col1 varchar2,col2 varchar2,col3 varchar2);
	CREATE TABLE
	gaussdb=# insert into tb_t1 select '1','1','2';
	INSERT 0 1
	openGauss=# create or replace procedure proc_test()as
	openGauss$# v_a varchar2;
	openGauss$# v_b varchar2;
	openGauss$# v_cnt int;
	openGauss$# v_sql varchar2;
	openGauss$# begin
	openGauss$# v_a = '1';
	openGauss$# v_b = '2';
	openGauss$# v_sql = 'select count(1) from tb_t1 where col1 = :va and col2 = :va and col3 = :vb;';
	openGauss$# execute immediate v_sql into v_cnt using v_a,v_a,v_b;
	openGauss$# raise info 'v_cnt:%',v_cnt;
	openGauss$# end;
	openGauss$# /
	CREATE PROCEDURE
	openGauss=# call proc_test();
	INFO: v_cnt:0
	proc_test
	---------- 	
	(1 row)
	openGauss=# set behavior_compat_options='dynamic_sql_check';
	SET
	openGauss=# call proc_test();
	ERROR: argnum not match in Dynamic SQL, using args num : 3 , actual sql args num : 2
	CONTEXT: SQL statement "select count(1) from tb_t1 where col1 = :va and col2 = :va and col3 = :vb;"
	PL/SQL function proc_test() line 10 at EXECUTE statemen
	注意
	不推荐同时使用dynamic_sql_check和dynamic_sql_compat选项，开启dynamic_sql_compat选项后，dynamic_sql_check选项将不再生效。

a_format_version

参数说明：数据库平台兼容性行为配置项，该参数的值为字符串枚举值。

该参数属于USERSET类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：字符串

默认值：""

说明

兼容性配置项时设置字符串，例如：set a_format_version='10c';

表 18-12 兼容性配置项

兼容性配置项	兼容性行为控制
10c	A平台兼容版本。

a_format_dev_version

参数说明：数据库平台迭代小版本兼容性行为配置项，该参数的值为字符串枚举值。

该参数属于USERSET类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：字符串

默认值：""

说明

兼容性配置项时设置字符串，例如：set a_format_dev_version='s1';

表 18-13 兼容性配置项

兼容性配置项	兼容性行为控制
s1	开启参数后cast支持 text转int四舍五入。

plpgsql.variable_conflict

参数说明：设置同名的存储过程变量和表的列的使用优先级。

该参数属于USERSET类型参数，仅支持**表18-2**中对应设置方法3进行设置。

取值范围：字符串

- error表示遇到存储过程变量和表的列名同名则编译报错。
- use_variable表示存储过程变量和表的列名同名则优先使用变量。
- use_column表示存储过程变量和表的列名同名则优先使用列名。

默认值：error

td_compatible_truncation

参数说明：控制是否开启与Teradata数据库相应兼容的特征。该参数在用户连接上与TD兼容的数据库时，可以将参数设置成为on（即超长字符串自动截断功能启用），该功能启用后，在后续的insert语句中，对目标表中char和varchar类型的列插入超长字符串时，会按照目标表中相应列定义的最大长度对超长字符串进行自动截断。保证数据都能插入目标表中，而不是报错。

说明

如果向字符集为字节类型编码（SQL_ASCII，LATIN1等）的数据库中插入多字节字符数据（如汉字等），且字符数据跨越截断位置，这种情况下，按照字节长度自动截断，自动截断后会在尾部产生非预期结果。如果用户有对于截断结果正确性的要求，建议用户采用UTF8等能够按照字符截断的输入字符集作为数据库的编码集。

该参数属于USERSET类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：布尔型

- on表示启动超长字符串自动截断功能。
- off表示停止超长字符串自动截断功能。

默认值：off

nls_timestamp_format

参数说明：设置时间戳默认格式。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

默认值：DD-Mon-YYYY HH:MI:SS.FF AM

max_function_args

参数说明：函数参数最大个数。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：整型

默认值：8192

convert_string_to_digit

参数说明：设置隐式转换优先级，是否优先将字符串转为数字。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示优先将字符串转为数字。
- off表示不优先将字符串转为数字。

默认值：on

须知

该参数调整会修改内部数据类型转换规则，导致不可预期的行为，请谨慎调操作。

18.3.16 容错性

当数据库系统发生错误时，以下参数控制服务器处理错误的方式。

exit_on_error

参数说明：打开该开关，ERROR级别报错会升级为PANIC报错，从而可以产生core堆栈。主要用于问题定位和业务测试。

该参数属于USERSET类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：布尔型

- on表示ERROR级别报错会升级为PANIC报错。
- off表示不会对ERROR级别报错进行升级。

默认值：off

restart_after_crash

参数说明：设置为on，后端进程崩溃时，GaussDB将自动重新初始化此后端进程。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：布尔型

- on表示能够最大限度地提高数据库的可用性。

  在某些情况（比如当采用管理工具（例如xCAT）管理GaussDB时），能够最大限度地提高数据库的可用性。
- off表示能够使得管理工具在后端进程崩溃时获取控制权并采取适当的措施进行处理。

默认值：on

omit_encoding_error

参数说明：设置为on，数据库的客户端字符集编码为UTF-8时，出现的字符编码转换错误将打印在日志中，有转换错误的被转换字符会被忽略，以"?"代替。

该参数属于USERSET类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：布尔型

- on表示有转换错误的字符将被忽略，以"?"代替，打印错误信息到日志中。
- off表示有转换错误的字符不能被转换，打印错误信息到终端。

默认值：off

说明

若该参数通过执行gs_guc reload修改时，如果当前节点上的某个session的连接不是来自于客户端，而是来自于该节点所属集群上的其他节点，那么执行gs_guc reload后该参数在该session上不会立即生效，需与连接节点断开连接后重新连接才会生效。

cn_send_buffer_size

参数说明：指定CN端数据发送数据缓存区的大小。

该参数属于POSTMASTER类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，8~128，单位为KB。

默认值：8KB

max_cn_temp_file_size

参数说明：指定SQL语句出错自动重试功能中CN端使用临时文件的最大值，设定为0表示不使用临时文件。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，0~10485760， 单位为KB。

默认值：5GB

retry_ecode_list

参数说明：指定SQL语句出错自动重试功能支持的错误类型列表。

该参数属于USERSET类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：字符串

默认值：YY001 YY002 YY003 YY004 YY005 YY006 YY007 YY008 YY009 YY010 YY011 YY012 YY013 YY014 YY015 53200 08006 08000 57P01 XX003 XX009 YY016

data_sync_retry

参数说明：控制当fsync到磁盘失败后是否继续运行数据库。由于在某些操作系统的场景下，fsync失败后重试阶段即使再次fsync失败也不会报错，从而导致数据丢失。

该参数属于POSTMASTER类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：布尔型

- on表示当fsync同步到磁盘失败后采取重试机制，数据库继续运行。
- off表示当fsync同步到磁盘失败后直接报panic，停止数据库。

默认值：off

remote_read_mode

参数说明：远程读功能开关。读取主机上的页面失败时可以从备机上读取对应的页面。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：枚举类型

- off表示关闭远程读功能。
- non_authentication表示开启远程读功能，但不进行证书认证。
- authentication表示开启远程读功能，但要进行证书认证。

默认值：authentication

18.3.17 连接池参数

当使用连接池访问数据库时，在系统运行过程中，数据库连接是被当作对象存储在内存中的，当用户需要访问数据库时，并非建立一个新的连接，而是从连接池中取出一个已建立的空闲连接来使用。用户使用完毕后，数据库并非将连接关闭，而是将连接放回连接池中，以供下一个请求访问使用。

pooler_port

参数说明：cm_agent、cm_ctl等内部工具运维管理端口，初始化用户或系统管理员通过客户端连接数据库所使用端口。

该参数属于POSTMASTER类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：CN或DN实例的GUC参数"port"值加1。

默认值：CN或DN实例的GUC参数"port"默认值加1，CN实例该参数默认值为8001，DN实例该参数默认值为40001。

pooler_maximum_idle_time

参数说明：Pooler链接自动清理功能使用，当链接池中链接空闲时间超过所设置值时，会触发自动清理机制，清理各节点的空闲链接数到minimum_pool_size。

该参数属于USERSET类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，最小值为0，最大值为2147483647，最小单位为秒

默认值：10min（即600秒）

minimum_pool_size

参数说明：Pooler链接自动清理功能使用，自动清理后各pooler链接池对应节点的链接数最小剩余量，当参数设置为0时，可以关闭pooler链接自动清理功能。

该参数属于USERSET类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，最小值为1，最大值为65535

默认值：50

max_pool_size

参数说明：CN的连接池与其它某个CN/DN的最大连接数，当集群规模有变化时，如增加节点、减少节点，可能需要调整该参数。

参数类型：整型

参数单位：无

取值范围：1~65535

默认值：

- 独立部署：

  32768（60核CPU/480G内存）；16384（32核CPU/256G内存）；8192（16核CPU/128G内存）；4096（8核CPU/64G内存）；2048（4核CPU/32G内存）；1000（4核CPU/16G内存）
- 金融版（标准型）：

  32000（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存）；16000（72核CPU/576G内存，64核CPU/512G内存，60核CPU/480G内存）；8000（32核CPU/256G内存）；4000（16核CPU/128G内存）；2000（8核CPU/64G内存）
- 企业版：

  CN：5000（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存）；4000（96核CPU/768G内存）；3000（80核CPU/640G内存）；2048（80核CPU/512G内存，72核CPU/576G内存，64核CPU/512G内存，60核CPU/480G内存，32核CPU/256G内存）；1024（16核CPU/128G内存，8核CPU/64G内存）

  DN：20000（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存）；15000（96核CPU/768G内存，80核CPU/640G内存）；10000（80核CPU/512G内存，72核CPU/576G内存，64核CPU/512G内存，60核CPU/480G内存）；5000（32核CPU/256G内存）；2000（16核CPU/128G内存）；1024（8核CPU/64G内存）

- 金融版（数据计算型）：

  CN：3000（96核CPU/768G内存）；2048（72核CPU/576G内存，64核CPU/512G内存）；1024（32核CPU/256G内存）

  DN：2048（96核CPU/768G内存，72核CPU/576G内存，64核CPU/512G内存）；1024（32核CPU/256G内存）

设置方式：该参数属于POSTMASTER类型参数，请参考**表18-2**中对应设置方法进行设置。

设置建议：按照不同实例规格下的默认值进行设置，该参数值需要大于max_connections，调整时需预留内部线程所消耗的连接。当业务并发高时，会消耗连接池中CN至其他CN/DN的连接，如果该参数配置过小，连接数达到上限时会产生报错，导致业务失败。由于CN启动时，会根据参数值提前申请内存，所以当该参数值变大，系统会消耗更多内存资源，但总体来说对CN内存影响不大。

persistent_datanode_connections

参数说明：会话是否会释放获得的连接。

该参数属于BACKEND类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：布尔型

- off表示会释放获得连接。
- on表示不会释放获得连接。

须知

打开此开关后，会存在会话持有连接但并未运行查询的情况，导致其他查询申请不到连接报错。出现此问题时，需约束会话数量小于等于max_active_statements。

默认值：off

max_coordinators

参数说明：集群中CN的最大数目，当集群规模有变化时，如增加节点、减少节点，可能需要调整该参数。扩容时，请确保此参数值大于需要扩容到的集群中CN的个数，否则会导致扩容失败。缩容时，若此参数值偏大，CN启动时会消耗更多的内存资源，但总体来说对CN内存影响不大。

参数类型：整型

参数单位：无

取值范围：2~1024

默认值：128

设置方式：该参数属于POSTMASTER类型参数，不建议修改此参数，若需修改，请参考**表18-2**中对应设置方法进行设置。

设置建议：按照集群的实际规格进行设置，当该参数小于集群当前CN个数，会导致节点创建失败。由于CN启动时，会根据参数值提前申请内存，所以当该参数值变大，系统会消耗更多内存资源，但总体来说对CN内存影响不大。

max_datanodes

参数说明：集群中DN的最大数目，当集群规模有变化时，如增加节点、减少节点，可能需要调整该参数。扩容时，请确保此参数值大于需要扩容到的集群中DN总分片数，否则会导致扩容失败。缩容时，若此参数值偏大，CN启动时会消耗更多的内存资源，但总体来说对CN内存影响不大。

参数类型：整型

参数单位：无

取值范围：2~65535

默认值：256

设置方式：该参数属于POSTMASTER类型参数，不建议修改此参数，若需修改，请参考**表18-2**中对应设置方法进行设置。

设置建议：按照集群的实际规格进行设置，当该参数小于集群当前DN个数，会导致节点创建失败。由于CN启动时，会根据参数值提前申请内存，所以当该参数值变大，系统会消耗更多内存资源，但总体来说对CN内存影响不大。

cache_connection

参数说明：是否回收连接池的连接。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：布尔型

- on表示回收连接池的连接。
- off表示不回收连接池的连接。

默认值：on

enable_force_reuse_connections

参数说明：会话是否强制重用新的连接。

该参数属于BACKEND类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：布尔型

- on表示强制使用新连接。
- off表示使用现有连接。

默认值：off

pooler_connect_max_loops

参数说明：pooler建链重试功能使用，主备切换场景增强建链稳定性，若节点间pooler建链失败会跟备机重试建连，此时若恰好备机升主机成功，则可以在重试阶段建链成功。该参数可以设置重试总轮数，增强建链稳定性。当参数设置为0时，可以关闭重试功能，业务只跟主机建链而不跟备机重试。

该参数属于USERSET类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，最小值为0，最大值为20。

默认值：1

pooler_connect_interval_time

参数说明：pooler建链重试功能使用，当参数pooler_connect_max_loops设置大于1时，该参数可以控制不同重试轮数之间执行时间间隔。参数设置方面，建议略大于当前集群主备切换恢复时间即。

该参数属于USERSET类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，最小值为0，最大值为7200，最小单位为秒。

默认值：15s

pooler_timeout

参数说明：CN连接池中的连接与其它CN/DN通讯时的超时时间。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，最小值为0，最大值为7200，最小单位为秒。

默认值：10min

pooler_connect_timeout

参数说明：CN连接池与集群中其他CN/DN建立连接时的超时时间。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，最小值为0，最大值为7200，最小单位为秒。

默认值：1min

pooler_cancel_timeout

参数说明：CN连接池在错误处理时Cancel某连接的超时时间。如果在子事务或存储过程异常捕获的过程中发生该类超时，那么包含子事务或存储过程的整个事务将发生回滚。在此基础上，在子事务或存储过程异常捕获的过程中，如果错误源自COPY FROM操作中源数据与目标表表结构的不一致，则只要该参数值不为0，就总会触发超时报错。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，最小值为0，最大值为7200，最小单位为秒。其中0时（一般不建议）表示关闭此开关，不做超时限制。

默认值：15s

18.3.18 集群事务

介绍集群事务隔离、事务只读、最大prepared事务数、集群维护模式目的参数设置及取值范围等内容。

transaction_isolation

参数说明：设置当前事务的隔离级别。

该参数属于USERSET类型参数，请参考表18-2中对应设置方法进行设置。

取值范围：字符串，只识别以下字符串，大小写空格敏感：

- serializable：GaussDB中等价于REPEATABLE READ。
- read committed：只能读取已提交的事务的数据（缺省），不能读取到未提交的数据。
- repeatable read：仅能读取事务开始之前提交的数据，不能读取未提交的数据以及在事务执行期间由其它并发事务提交的修改。
- read uncommitted：读未提交，可以读取任何时刻的数据。
- default：设置为defualt_transaction_isolation所设隔离级别。

默认值：read committed

transaction_read_only

参数说明：设置当前事务是只读事务。

该参数属于USERSET类型参数，请参考表18-2中对应设置方法进行设置。

取值范围：布尔型

- on表示设置当前事务为只读事务。
- off表示该事务可以是非只读事务。

默认值：off

xc_maintenance_mode

参数说明：设置系统进入维护模式。

该参数属于SUSET类型参数，仅支持表18-2中的方式三进行设置。

取值范围：布尔型

- on表示该功能启用。
- off表示该功能被禁用。

须知

谨慎打开这个开关，避免引起集群数据不一致。

默认值：off

allow_concurrent_tuple_update

参数说明：设置是否允许并发更新。

该参数属于USERSET类型参数，请参考表18-2中对应设置方法进行设置。

取值范围：布尔型

- on表示该功能启用。

- off表示该功能被禁用。

默认值：on

gtm_host

参数说明：主GTM进程所在的IP地址。仅sysadmin用户可见。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：字符串。

默认值：主GTM的IP地址

gtm_port

参数说明：主GTM进程的侦听端口。仅sysadmin用户可见。

该参数属于POSTMASTER类型参数。

说明

该参数由安装时的配置文件指定，请勿轻易修改，否则修改后会影响数据库正常通信。

取值范围：整型，最小值为1，最大值为65535。

默认值：安装时指定。

gtm_host1

参数说明：备GTM进程所在的IP地址。仅sysadmin用户可见。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：字符串。

默认值：备GTM的IP地址

gtm_port1

参数说明：备GTM进程的侦听端口。仅sysadmin用户可见。

该参数属于POSTMASTER类型参数。

说明

该参数由安装时的配置文件指定，请勿轻易修改，否则修改后会影响数据库正常通信。

取值范围：整型，最小值为1，最大值为65535。

默认值：如有备1，安装指定，否则为6665。

pgxc_node_name

参数说明：指定节点名称。

该参数属于POSTMASTER类型参数，请参考**表18-2**中对应设置方法进行设置。

在备机请求主机进行日志复制时，如果application_name参数没有被设置，那么pgxc_node_name参数会被用来作为备机在主机上的流复制槽名字。该流复制槽的命

名方式为 "该参数值_备机ip_备机port"。其中，备机ip和备机port取自replconninfo参数中指定的备机ip和端口号。该流复制槽最大长度为61个字符，如果拼接后的字符串超过该长度，则会使用截断后的pgxc_node_name进行拼接，以保证流复制槽名字长度小于等于61个字符。

⚠ 注意

此参数修改后会导致连接集群失败，不建议进行修改。

取值范围：字符串。

默认值：当前节点名称。

gtm_backup_barrier

参数说明：指定是否为GTM启动点创建还原点。

该参数属于SUSET类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：布尔型

- on表示创建还原点。
- off表示不创建还原点。

默认值：off

gtm_conn_check_interval

参数说明：设置CN检查本地线程与主GTM连接是否正常时间。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型， 0 ~ 2147483，单位为秒。

默认值：10s

transaction_deferrable

参数说明：指定是否允许一个只读串行事务延迟执行，使其不会执行失败。该参数设置为on时，当一个只读事务发现读取的元组正在被其他事务修改，则延迟该只读事务直到其他事务修改完成。该参数为预留参数，该版本不生效。与该参数类似的还有一个**default_transaction_deferrable**，设置它来指定一个事务是否允许延迟。

该参数属于USERSET类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许执行。
- off表示不允许执行。

默认值：off

enable_show_any_tuples

参数说明：该参数只有在只读事务中可用，用于分析。当这个参数被置为on/true时，表中元组的所有版本都会可见。

该参数属于USERSET类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：布尔型

- on/true表示表中元组的所有版本都会可见。
- off/false表示表中元组的所有版本都不可见。

默认值：off

gtm_connect_timeout

参数说明：控制GTM连接超时时间，如果GTM的连接时间超过此参数设置的值，会超时返回。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，最小值为1，最大值为2147483647，单位为秒。

默认值：2s

gtm_connect_retries

参数说明：控制GTM连接重试的次数。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，最小值为1，最大值为2147483647。

默认值：30

gtm_rw_timeout

参数说明：控制GTM反馈超时时间，如果GTM的反馈时间超过此参数设置的值，也就是等待时间超过了此参数值，会超时返回。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，最小值为1，最大值为2147483647，单位为秒。

默认值：1min

enable_redistribute

参数说明：节点不匹配时是否重新分配。

该参数属于SUSET类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：布尔型

- on表示节点不匹配时重新分配。
- off表示节点不匹配时不重新分配。

默认值：off

replication_type

参数说明：标记当前HA模式是一主多备模式或单主机模式。

该参数属于POSTMASTER类型参数，请参考**表18-2**中对应设置方法进行设置。

该参数是CM部署时的用到的参数，用户不能自己去设置参数值。

取值范围：0~2

- 0 表示实验室特性，详情请参见《实验室特性说明》。
- 1 表示使用一主多备模式，全场景覆盖，推荐使用。
- 2 表示使用单主机模式，此模式无法扩展备机。

默认值：1

enable_gtm_free

参数说明：大并发场景下同一时刻存在活跃事务较多，GTM下发的快照变大且快照请求变多的情况下，瓶颈卡在GTM与CN通讯的网络上。为消除该瓶颈，引入GTM-FREE模式。取消CN和GTM的交互，取消CN下发GTM获取的事务信息给DN。CN只向各个DN发送query，各个DN由本地产生快照及xid等信息，开启该参数支持分布式事务读最终一致性，即分布式事务只有写外部一致性，不具有读外部一致性。

该参数属于POSTMASTER类型参数，请参考**表18-2**中对应设置方法进行设置。

⚠ 注意

业务使用GTM-Free模式时，建议将application_type设置成perfect_sharding_type，以便及时发现可能导致数据不一致的SQL语句。否则，系统不会拦截可能导致数据不一致的语句，造成数据不一致。

取值范围：布尔型

- on表示开启GTM-FREE模式，集群状态为读最终一致性。
- off表示非GTM-FREE模式。

默认值：off

enable_twophase_commit

参数说明：当前云数据库主要解决SDS替换问题，采用模式为GTM Free，为防止业务滥用导致不可靠问题，提供guc参数开关enable_twophase_commit禁用分布式写事务，该参数属于USERSET类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启GTM-FREE模式下，允许业务进行分布式两阶段写事务。
- off表示开启GTM-FREE模式下，禁止业务进行分布式两阶段写事务。

默认值：on

application_type

参数说明：此参数仅在enable_gtm_free为on时有效。此参数用来说明用户的业务类型。该参数属于USERSET类型参数，请参考**表18-2**中对应设置方法进行设置。此参数不允许使用gs_guc设置，只允许以下列两种方式设置：

1. 使用gsql等客户端在session级别设置。

2. 使用jdbc连接数据库时，给连接字符串指定ApplicationType参数。

取值范围：枚举类型

- not_perfect_sharding_type表示跨节点的业务。取此值时，允许执行跨节点的语句。
- perfect_sharding_type表示单节点的业务。取此值时，如果SQL语句需要多个节点参与，会直接报错。对应的SQL语句会同时打印到系统日志中。
  - 取此值时，使用/*+ multinode */ hint可以显示允许SQL语句在多个节点执行。multinode hint可以加到select、insert、update、delete、merge关键字之后。

gtm_host2

参数说明：如有2号GTM，参数值为2号GTM进程所在的主机名或IP地址。仅sysadmin用户可见。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：字符串。

默认值：如有备2，为GTM的IP地址，否则为“ ”。

gtm_host3

参数说明：如有3号GTM，参数值为3号GTM进程所在的主机名或IP地址。仅sysadmin用户可见。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：字符串。

默认值：如有备3，为GTM的IP地址，否则为“ ”。

gtm_host4

参数说明：如有4号GTM，参数值为4号GTM进程所在的主机名或IP地址。仅sysadmin用户可见。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：字符串。

默认值：如有备4，为GTM的IP地址，否则为“ ”。

gtm_host5

参数说明：如有5号GTM，参数值为5号GTM进程所在的主机名或IP地址。仅sysadmin用户可见。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：字符串。

默认值：如有备5，为GTM的IP地址，否则为“ ”。

gtm_host6

参数说明：如有6号GTM，参数值为6号GTM进程所在的主机名或IP地址。仅sysadmin用户可见。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：字符串。

默认值：如有备6，为GTM的IP地址，否则为“”。

gtm_host7

参数说明：如有7号GTM，参数值为7号GTM进程所在的主机名或IP地址。仅sysadmin用户可见。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：字符串。

默认值：如有备7，为GTM的IP地址，否则为“”。

gtm_port2

参数说明：如有2号GTM，参数值为2号GTM进程的侦听端口。仅sysadmin用户可见。

该参数属于POSTMASTER类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，最小值为1，最大值为65535。

默认值：如有备2，安装指定，否则为6666。

gtm_port3

参数说明：如有3号GTM，参数值为3号GTM进程的侦听端口。仅sysadmin用户可见。

该参数属于POSTMASTER类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，最小值为1，最大值为65535。

默认值：如有备3，安装指定，否则为6666。

gtm_port4

参数说明：如有4号GTM，参数值为4号GTM进程的侦听端口。仅sysadmin用户可见。

该参数属于POSTMASTER类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，最小值为1，最大值为65535。

默认值：如有备4，安装指定，否则为6666。

gtm_port5

参数说明：如有5号GTM，参数值为5号GTM进程的侦听端口。仅sysadmin用户可见。

该参数属于POSTMASTER类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，最小值为1，最大值为65535。

默认值：如有备5，安装指定，否则为6666。

gtm_port6

参数说明：如有6号GTM，参数值为6号GTM进程的侦听端口。仅sysadmin用户可见。

该参数属于POSTMASTER类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，最小值为1，最大值为65535。

默认值：如有备6，安装指定，否则为6666。

gtm_port7

参数说明：如有7号GTM，参数值为7号GTM进程的侦听端口。仅sysadmin用户可见。

该参数属于POSTMASTER类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：整型，最小值为1，最大值为65535。

默认值：如有备7，安装指定，否则为6666。

enable_defer_calculate_snapshot

参数说明：延迟计算快照的xmin和oldestxmin，执行1000个事务或者间隔1s才触发计算，设置为on时可以在高负载场景下减少计算快照的开销，但是会导致oldestxmin推进较慢，影响垃圾元组回收，设置为off时xmin和oldestxmin可以实时推进，但是会增加计算快照时的开销。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：布尔型。

- on表示延迟计算快照xmin和oldestxmin。
- off表示实时计算快照xmin和oldestxmin。

默认值：on。

18.3.19 双集群复制参数

enable_roach_standby_cluster

参数说明：设置双集群中备集群的各个实例为只读模式，仅sysadmin用户可以访问。

该参数属于POSTMASTER类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：布尔型

- on表示备集群开启只读模式。
- off表示备集群关闭只读模式。此情况下，备集群可读可写。

默认值：off

enable_slot_log

参数说明：是否开启逻辑复制槽主备同步特性。

该参数属于USERSET类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启逻辑复制槽主备同步特性。
- off表示不开启逻辑复制槽主备同步特性。

默认值：on

max_changes_in_memory

参数说明：逻辑解码时单条事务在内存中缓存的DML语句数量上限。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1~2147483647

默认值：4096

max_cached_tuplebufs

参数说明：逻辑解码时总元组信息在内存中缓存的数量上限。建议设置为max_changes_in_memory的两倍以上。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1~2147483647

默认值：8192

logical_decode_options_default

参数说明：指定逻辑解码启动时未指定解码选项的全局默认值。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

当前支持指定的逻辑解码选项包括：parallel-decode-num, parallel-queue-size, max-txn-in-memory, max-reorderbuffer-in-memory, exclude-users。选项的意义请参考《开发指南》中“应用程序开发教程 > 基于JDBC开发 > 示例：逻辑复制代码示例”章节。

取值范围：通过逗号分隔的key=value字符串，例如：'parallel-decode-num=4,parallel-queue-size=128,exclude-users=userA'。其中空字符串表示采用程序硬编码的默认值。

默认值：""

须知

该参数SIGHUP生效并不会影响已经启动的逻辑解码流程；后续逻辑解码启动将使用该参数设置的选项作为其默认配置，并优先使用启动命令中指定选项的设置。

这里exclude-users选项和逻辑解码启动选项存在差异，不允许指定多个黑名单用户。

logical_sender_timeout

参数说明：设置本端等待逻辑日志接收端接收日志的最大等待时间。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0 ~ 2147483647，单位为毫秒（ms）。

默认值：30s

RepOriginId

参数说明：该参数是一个会话级别的GUC参数，在双向逻辑复制的场景下，为避免数据循环复制，需要设置为一个非0的值。

该参数属于USERSET类型参数，请参考**表18-1**中方式三对应设置方法进行设置。

取值范围：整型，0~2147483647

默认值：0

hadr_max_size_for_xlog_receiver

参数说明：该参数为异地容灾参数，表示灾备集群中实例获取obs端日志和本地回放日志的最大允许差距，若差距大于此值时停止获取obs端日志。

该参数属于SIGHUP类型参数，请参考**表18-1**中方式对应设置方法进行设置。

修改建议：该参数的取值应和本地磁盘大小相关，建议设置为磁盘大小的50%。

取值范围：整型，0~2147483647

默认值：256GB

auto_csn_barrier

参数说明：流式容灾的主集群是否开启barrier打点功能。

该参数属于SIGHUP类型参数，请参考表**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启。
- off表示关闭。

默认值：off

stream_cluster_run_mode

参数说明：流式容灾双集群容灾场景标识CN/DN节点属于主集群还是备集群。单集群使用默认值主集群。

该参数属于POSTMASTER类型参数，请参考表**表18-1**中对应设置方法进行设置。

取值范围：枚举类型

- cluster_primary表示节点是主集群的节点。
- cluster_standby表示节点是备集群的节点。

默认值：cluster_primary

18.3.20 开发人员选项

allow_system_table_mods

参数说明：设置是否允许修改系统表的结构或系统自带模式名称。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许修改系统表的结构或系统自带模式名称。
- off表示不允许修改系统表的结构或系统自带模式名称。

默认值：off

说明

不建议修改该参数默认值，若设置为on，可能导致系统表损坏，甚至数据库无法启动。

allow_create_sysobject

参数说明：设置是否允许在系统模式下创建或修改函数、存储过程、同义词等对象。此处的系统模式指数据库初始后自带的模式，但不包含public模式。系统模式的oid通常小于16384。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许初始用户和系统管理员在系统模式下创建或修改函数、存储过程、同义词等对象。其他用户是否允许创建这些对象请参考对应模式的权限要求。
- off表示禁止所有用户在系统模式下创建或修改函数、存储过程、同义词等对象。

默认值：on

debug_assertions

参数说明：控制打开各种断言检查。能够协助调试，当遇到奇怪的问题或者崩溃，请把此参数打开，因为它能暴露编程的错误。要使用这个参数，必须在编译GaussDB的时候定义宏USE_ASSERT_CHECKING（通过configure选项 --enable-cassert完成）。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示打开断言检查。
- off表示不打开断言检查。

说明

当启用断言选项编译GaussDB时，debug_assertions缺省值为on 。

默认值：off

ignore_checksum_failure

参数说明：设置此参数为打开会导致系统忽略失败（但仍然会告警），继续执行可能导致崩溃，传播或保存损坏数据，无法从远程节点恢复数据及其他严重问题。不建议用户修改设置。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示忽略数据校验错误。
- off表示数据校验错误正常报错。

默认值：off

ignore_system_indexes

参数说明：读取系统表时忽略系统索引（但是修改系统表时依然同时修改索引）。

该参数属于BACKEND类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

这个参数在从系统索引被破坏的表中恢复数据的时候非常有用。

取值范围：布尔型

- on表示忽略系统索引。
- off表示不忽略系统索引。

默认值：off

post_auth_delay

参数说明：在认证成功后，延迟指定时间，启动服务器连接。允许调试器附加到启动进程上。

该参数属于BACKEND类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，最小值为0，最大值为2147，单位为秒。

默认值：0

说明

此参数只用于调试和问题定位，为避免影响正常业务运行，生产环境下请确保参数值为默认值0。参数设置为非0时可能会因认证延迟时间过长导致集群状态异常。

pre_auth_delay

参数说明：启动服务器连接后，延迟指定时间，进行认证。允许调试器附加到认证过程上。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，取值范围为0～60，单位为秒。

默认值：0

说明

此参数只用于调试和问题定位，为避免影响正常业务运行，生产环境下请确保参数值为默认值0。参数设置为非0时可能会因认证延迟时间过长导致集群状态异常。

trace_notify

参数说明：为LISTEN和NOTIFY命令生成大量调试输出。**client_min_messages**或**log_min_messages**级别必须是DEBUG1或者更低时，才能把这些输出分别发送到客户端或者服务器日志。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示打开输出功能。
- off表示关闭输出功能。

默认值：off

trace_recovery_messages

参数说明：启用恢复相关调试输出的日志录，否则将不会被记录。该参数允许覆盖正常设置的**log_min_messages**，但是仅限于特定的消息，这是为了在调试备机中使用。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：枚举类型，有效值有debug5、debug4、debug3、debug2、debug1、log，取值的详细信息请参见**log_min_messages**。

默认值：log

说明

- 默认值log表示不影响记录决策。
- 除默认值外，其他值会导致优先级更高的恢复相关调试信息被记录，因为它们有log优先权。对于常见的log_min_messages设置，这会导致无条件地将它们记录到服务器日志上。

trace_sort

参数说明：控制是否在日志中打印排序操作中的资源使用相关信息。这个选项只有在编译GaussDB的时候定义了TRACE_SORT宏的时候才可用，不过目前TRACE_SORT是由缺省定义的。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示打开控制功能。
- off表示关闭控制功能。

默认值：off

zero_damaged_pages

参数说明：控制检测导致GaussDB报告错误的损坏的页头，终止当前事务。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- 设置为on时，会导致系统报告一个警告，把损坏的页面填充为零然后继续处理。这种行为会破坏数据，也就是所有在已经损坏页面上的行记录。但是它允许绕开坏页面然后从表中尚存的未损坏页面上继续检索数据行。因此它在因为硬件或者软件错误导致的崩溃中进行恢复是很有用的。通常不应该把它设置为on，除非不需要从崩溃的页面中恢复数据。
- 设置为off时，系统不会将损坏页面填充零。

默认值：off

string_hash_compatible

参数说明：该参数用来说明char类型和varchar/text类型的hash值计算方式是否相同，以此来判断进行分布列从char类型到相同值的varchar/text类型转换，数据分布变化时，是否需要进行重分布。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示计算方式相同，不需要进行重分布。
- off表示计算方式不同，需要进行重分布。

说明

计算方式的不同主要体现在字符串计算hash值时传入的字节长度上。（如果为char，则会忽略字符串后面空格的长度，如果为text或varchar，则会保留字符串后面空格的长度。）hash值的计算会影响到查询的计算结果，因此此参数一旦设置后，在整个数据库使用过程中不能再对其进行修改，以避免查询错误。

默认值：off

remotetype

参数说明：设置远程连接类型。

该参数属于BACKEND类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：枚举类型，有效值有application，coordinator，datanode，gtm，gtmproxy，internaltool，gtmtool。

默认值：application

max_user_defined_exception

参数说明：异常最大个数。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，当前只能取固定值1000

默认值：1000

enable_compress_spill

参数说明：标识是否开启下盘压缩功能。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on/true表示开启下盘优化。
- off/false表示关闭下盘优化。

默认值：on

enable_parallel_ddl

参数说明：控制多CN对同一数据库对象是否能安全的并发执行DDL操作。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示可以安全的并发执行DDL操作，不会出现分布式死锁。
- off表示不能安全的并发执行DDL操作，可能会出现分布式死锁。

默认值：on

support_batch_bind

参数说明：控制是否允许通过JDBC、ODBC、Libpq等接口批量绑定和执行PBE形式的语句。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示使用批量绑定和执行。
- off表示不使用批量绑定和执行。

默认值：on

numa_distribute_mode

参数说明：用于控制部分共享数据和线程在NUMA节点间分布的属性。用于大型多NUMA节点的ARM服务器性能调优，一般不用设置。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串，当前有效取值为'none', 'all'。

- none：表示不启用本特性。
- all：表示将部分共享数据和线程分布到不同的NUMA节点下，减少远端访存次数，提高性能。目前仅适用于拥有多个NUMA节点的ARM服务器，并且要求全部NUMA节点都可用于数据库进程，不支持仅选择一部分NUMA节点。

说明

当前版本x86架构下不支持numa_distribute_mode设置为all。

默认值：'none'

log_pagewriter

参数说明：设置用于增量检查点打开后，显示线程的刷页信息以及增量检查点的详细信息，信息比较多，不建议设置为true。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

默认值：off

advance_xlog_file_num

参数说明：用于控制在后台周期性地提前初始化xLog文件的数目。该参数是为了避免事务提交时执行xLog文件初始化影响性能，但仅在超重负载时才可能出现，因此一般不用配置。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~1000000（0表示不提前初始化）。例如，取值10，表示后台线程会周期性地根据当前xLog写入位置提前初始化10个xLog文件。

默认值：0

comm_sender_buffer_size

参数说明：用于设置Stream计划中CN与DN之间，DN与DN之间每次交互的BUFFER大小，在一些情况下不同的取值会对Stream性能产生影响，重置后需要重启集群生效，单位KB。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1~1024。

默认值：8

default_index_kind

参数说明：控制创建索引的默认行为。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，当前只能取固定值0、1、2。

- 0：表示对分布式部署方式不开启全局分区索引功能。
- 1：表示默认创建局部索引。
- 2：表示默认创建全局索引。

默认值：2

⚠ 注意

不建议修改该参数默认值，若随意修改该参数，可能会影响索引有效性。

18.3.21 审计

18.3.21.1 审计开关

audit_enabled

参数说明：控制审计进程的开启和关闭。审计进程开启后，将从管道读取后台进程写入的审计信息，并写入审计文件。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示启动审计功能。
- off表示关闭审计功能。

默认值：on

audit_directory

参数说明：审计文件的存储目录。一个相对于数据目录data的路径，可自行指定，仅sysadmin用户可以访问。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

默认值：pg_audit。如果使用om工具部署集群，则审计日志路径为“$GAUSSLOG/pg_audit/实例名称”。

须知

- 不同的CN或DN实例需要设置不同的审计文件存储目录，否则会导致审计日志异常。
- 当配置文件中audit_directory的值为非法路径时，会导致审计功能无法使用。

说明

- 合法路径：用户对此路径有读写权限。
- 非法路径：用户对此路径无读写权限。

audit_data_format

参数说明：审计日志文件的格式。当前仅支持二进制格式，仅sysadmin用户可以访问。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

默认值：binary

audit_rotation_interval

参数说明：指定创建一个新审计日志文件的时间间隔。当现在的时间减去上次创建一个审计日志的时间超过了此参数值时，服务器将生成一个新的审计日志文件。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1~35791394，单位为min。

默认值：1d

须知

请不要随意调整此参数，否则可能会导致audit_resource_policy无法生效，如果需要控制审计日志的存储空间和时间，请使用**audit_resource_policy**、**audit_space_limit**和**audit_file_remain_time**参数进行控制。

audit_rotation_size

参数说明：指定审计日志文件的最大容量。当审计日志消息的总量超过此参数值时，服务器将生成一个新的审计日志文件。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1024~1048576，单位为KB。

默认值：10MB

须知

- 请不要随意调整此参数，否则可能会导致audit_resource_policy无法生效，如果需要控制审计日志的存储空间和时间，请使用audit_resource_policy、audit_space_limit和audit_file_remain_time参数进行控制。
- 审计日志文件中记录的单条日志占用空间大小超过此参数值时会被作为无效日志文件。

audit_resource_policy

参数说明：控制审计日志的保存策略，以空间还是时间限制为优先策略。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示采用空间优先策略，最多存储**audit_space_limit**大小的日志。
- off表示采用时间优先策略，最少存储**audit_file_remain_time**长度时间的日志。

默认值：on

audit_file_remain_time

参数说明：表示需记录审计日志的最短时间要求，该参数在**audit_resource_policy**为off时生效。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~730，单位为day，0表示无时间限制。

默认值：90

audit_space_limit

参数说明：审计文件占用的磁盘空间总量。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1024KB~1024GB，单位为KB。

默认值：1GB

须知

- 此参数的生效范围是pg_audit目录下的单个进程实例文件夹。即默认情况下，每一个CN、DN目录审计文件占用磁盘空间总量是1GB。
- 多审计线程场景下，审计文件占用的磁盘空间最小值是audit_thread_num与audit_rotation_size的乘积，请保证audit_space_limit的值大于audit_thread_num与audit_rotation_size的乘积。

audit_file_remain_threshold

参数说明：审计目录下审计文件个数的最大值。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，100~1048576

默认值：1048576

须知

- 请尽量保证此参数为1048576，并不要随意调整此参数，否则可能会导致audit_resource_policy无法生效，如果需要控制审计日志的存储空间和时间，请使用audit_resource_policy、audit_space_limit和audit_file_remain_time参数进行控制。
- 多审计线程场景下不建议调整此参数，请保证此参数不小于审计线程个数audit_thread_num，否则会导致审计功能无法正常使用与数据库异常。

audit_thread_num

参数说明：审计线程的个数

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1~48

默认值：1

须知

当audit_dml_state开关打开且在高性能场景下，建议增大此参数保证审计消息可以被及时处理和记录。

18.3.21.2 用户和权限审计

audit_login_logout

参数说明：这个参数决定是否审计GaussDB用户的登录（包括登录成功和登录失败）、注销。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~7。

- 0表示关闭用户登录、注销审计功能。
- 1表示只审计用户登录成功。
- 2表示只审计用户登录失败。
- 3表示只审计用户登录成功和失败。
- 4表示只审计用户注销。
- 5表示只审计用户注销和登录成功。
- 6表示只审计用户注销和登录失败。
- 7表示审计用户登录成功、失败和注销。

默认值：7

说明

DN不会对来自CN的登录、注销行为进行审计。

audit_database_process

参数说明：该参数决定是否对GaussDB的启动、停止、切换和恢复进行审计。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0、1。

- 0表示关闭GaussDB启动、停止、恢复和切换审计功能。
- 1表示开启GaussDB启动、停止、恢复和切换审计功能。

默认值：1

audit_user_locked

参数说明：该参数决定是否审计GaussDB用户的锁定和解锁。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0、1。

- 0表示关闭用户锁定和解锁审计功能。
- 1表示开启审计用户锁定和解锁功能。

默认值：1

audit_user_violation

参数说明：该参数决定是否审计用户的越权访问操作。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0、1。

- 0表示关闭用户越权操作审计功能。
- 1表示开启用户越权操作审计功能。

默认值：0

audit_grant_revoke

参数说明：该参数决定是否审计GaussDB用户权限授予和回收的操作。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0、1。

- 0表示关闭审计用户权限授予和回收功能。
- 1表示开启审计用户权限授予和回收功能。

默认值：1

18.3.21.3 操作审计

audit_system_object

参数说明：该参数决定是否对GaussDB数据库对象的CREATE、DROP、ALTER操作进行审计。GaussDB数据库对象包括DATABASE、USER、schema、TABLE等。通过修改该配置参数的值，可以只审计需要的数据库对象的操作,在主备强制选主场景建议audit_system_object取最大值，所有DDL对象全部审计。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0～134217727

- 0代表关闭GaussDB数据库对象的CREATE、DROP、ALTER操作审计功能。
- 非0代表只审计GaussDB的某类或者某些数据库对象的CREATE、DROP、ALTER操作。

取值说明：

该参数的值由27个二进制位的组合求出，这27个二进制位分别代表GaussDB的27类数据库对象。如果对应的二进制位取值为0，表示不审计对应的数据库对象的CREATE、DROP、ALTER操作；取值为1，表示审计对应的数据库对象的CREATE、DROP、ALTER操作。这27个二进制位代表的具体审计内容请参见**表18-14**。

默认值：67121159，表示对DATABASE、SCHEMA、USER、DATA SOURCE、NODE GROUP这几种数据库对象的DDL操作进行审计。

表 18-14 audit_system_object 取值含义说明

二进制位	含义	取值说明
第0位	是否审计DATABASE对象的CREATE、DROP、ALTER操作。	● 0表示不审计该对象的CREATE、DROP、ALTER操作； ● 1表示审计该对象的CREATE、DROP、ALTER操作。
第1位	是否审计SCHEMA对象的CREATE、DROP、ALTER操作。	● 0表示不审计该对象的CREATE、DROP、ALTER操作； ● 1表示审计该对象的CREATE、DROP、ALTER操作。
第2位	是否审计USER和USER MAPPING对象的CREATE、DROP、ALTER操作。	● 0表示不审计该对象的CREATE、DROP、ALTER操作； ● 1表示审计该对象的CREATE、DROP、ALTER操作。
第3位	是否审计TABLE对象的CREATE、DROP、ALTER、TRUNCATE操作。	● 0表示不审计该对象的CREATE、DROP、ALTER、TRUNCATE操作； ● 1表示审计该对象的CREATE、DROP、ALTER、TRUNCATE操作。
第4位	是否审计INDEX对象的CREATE、DROP、ALTER操作。	● 0表示不审计该对象的CREATE、DROP、ALTER操作； ● 1表示审计该对象的CREATE、DROP、ALTER操作。
第5位	是否审计VIEW/MATVIEW对象的CREATE、DROP操作。	● 0表示不审计该对象的CREATE、DROP操作； ● 1表示审计该对象的CREATE、DROP操作。
第6位	是否审计TRIGGER对象的CREATE、DROP、ALTER操作。	● 0表示不审计该对象的CREATE、DROP、ALTER操作； ● 1表示审计该对象的CREATE、DROP、ALTER操作。
第7位	是否审计PROCEDURE/FUNCTION对象的CREATE、DROP、ALTER操作。	● 0表示不审计该对象的CREATE、DROP、ALTER操作； ● 1表示审计该对象的CREATE、DROP、ALTER操作。
第8位	是否审计TABLESPACE对象的CREATE、DROP、ALTER操作。	● 0表示不审计该对象的CREATE、DROP、ALTER操作； ● 1表示审计该对象的CREATE、DROP、ALTER操作。

二进制位	含义	取值说明
第9位	是否审计RESOURCE POOL对象（ 详见《实验室特性说明》的“资源负载管理 ”章节内容）的CREATE、DROP、ALTER操作。	● 0表示不审计该对象的CREATE、DROP、ALTER操作； ● 1表示审计该对象的CREATE、DROP、ALTER操作。
第10位	是否审计WORKLOAD对象（ 详见《实验室特性说明》的“资源负载管理 ”章节内容）的CREATE、DROP、ALTER操作。	● 0表示不审计该对象的CREATE、DROP、ALTER操作； ● 1表示审计该对象的CREATE、DROP、ALTER操作。
第11位	保留	-
第12位	是否审计DATA SOURCE对象的CRAETE、DROP、ALTER操作。不建议用户使用。	● 0表示不审计该对象的CREATE、DROP、ALTER操作； ● 1表示审计该对象的CREATE、DROP、ALTER操作。
第13位	是否审计NODE GROUP对象的CREATE、DROP操作。	● 0表示不审计该对象的CREATE、DROP操作； ● 1表示审计该对象的CREATE、DROP操作。
第14位	是否审计ROW LEVEL SECURITY对象的CREATE、DROP、ALTER操作。	● 0表示不审计该对象的CREATE、DROP、ALTER操作； ● 1表示审计该对象的CREATE、DROP、ALTER操作。
第15位	是否审计TYPE对象的CREATE、DROP、ALTER操作。	● 0表示不审计TYPE对象的CREATE、DROP、ALTER操作； ● 1表示审计TYPE对象的CREATE、DROP、ALTER操作。
第16位	是否审计TEXT SEARCH对象（CONFIGURATION和DICTIONARY）的CREATE、DROP、ALTER操作。	● 0表示不审计TEXT SEARCH对象的CREATE、DROP、ALTER操作； ● 1表示审计TEXT SEARCH对象的CREATE、DROP、ALTER操作。
第17位	是否审计DIRECTORY对象的CREATE、DROP、ALTER操作。	● 0表示不审计DIRECTORY对象的CREATE、DROP、ALTER操作； ● 1表示审计DIRECTORY对象的CREATE、DROP、ALTER操作。
第18位	是否审计SYNONYM对象的CREATE、DROP、ALTER操作。	● 0表示不审计SYNONYM对象的CREATE、DROP、ALTER操作； ● 1表示审计SYNONYM对象的CREATE、DROP、ALTER操作。

二进制位	含义	取值说明
第19位	是否审计SEQUENCE对象的CREATE、DROP、ALTER操作。	• 0表示不审计SEQUENCE对象的CREATE、DROP、ALTER操作； • 1表示审计SEQUENCE对象的CREATE、DROP、ALTER操作。
第20位	保留	-
第21位	是否审计PACKAGE对象的CREATE、DROP、ALTER操作（分布式场景暂不支持PACKAGE）。	• 0表示不审计PACKAGE对象的CREATE、DROP、ALTER操作； • 1表示审计PACKAGE对象的CREATE、DROP、ALTER操作。
第22位	保留	-
第23位	保留	-
第24位	是否审计对gs_global_config全局对象的ALTER、DROP操作。	• 0表示不审计对系统表gs_global_config全局对象的ALTER、DROP操作； • 1表示审计对系统表gs_global_config全局对象的ALTER、DROP操作。
第25位	保留	-
第26位	保留	-

audit_dml_state

参数说明：这个参数决定是否对具体表的INSERT、UPDATE、DELETE操作进行审计。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0、1。

- 0表示关闭具体表的DML操作（SELECT除外）审计功能。
- 1表示开启具体表的DML操作（SELECT除外）审计功能。

默认值：0

audit_dml_state_select

参数说明：这个参数决定是否对SELECT操作进行审计。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0、1。

- 0表示关闭SELECT操作审计功能。
- 1表示开启SELECT审计操作功能。

默认值：0

audit_function_exec

参数说明：这个参数决定在执行存储过程、匿名块或自定义函数（不包括系统自带函数）时是否记录审计信息。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0、1。

- 0表示关闭过程或函数执行的审计功能。
- 1表示开启过程或函数执行的审计功能。

默认值：0

audit_copy_exec

参数说明：这个参数决定是否对COPY操作进行审计。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0、1。

- 0表示关闭COPY审计功能。
- 1表示开启COPY审计功能。

默认值：1

audit_set_parameter

参数说明：这个参数决定是否对SET操作进行审计。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0、1。

- 0表示关闭SET审计功能。
- 1表示开启SET审计功能。

默认值：0

audit_xid_info

参数说明：这个参数决定是否在审计日志字段detail_info中记录SQL语句的事务ID。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0、1。

- 0表示关闭审计日志记录事务ID功能。
- 1表示开启审计日志记录事务ID功能。

默认值：0

须知

如果开启此开关，审计日志中detail_info信息则以xid开始，例如：

```
detail_info: xid=14619 , create table t1(id int);
```

对于不存在事务ID的审计行为，记录xid=NA。

enableSeparationOfDuty

参数说明：是否开启三权分立选项。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启三权分立。
- off表示不开启三权分立。

默认值：off

enable_nonsysadmin_execute_direct

参数说明：是否允许非系统管理员和非监控管理员执行EXECUTE DIRECT ON语句。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示允许任意用户执行EXECUTE DIRECT ON语句。
- off表示只允许系统管理员和监控管理员执行EXECUTE DIRECT ON语句。

默认值：off

enable_access_server_directory

参数说明：是否开启非初始用户创建、修改和删除DIRECTORY的权限。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启非初始用户创建、修改和删除DIRECTORY的权限。
- off表示不开启非初始用户创建、修改和删除DIRECTORY的权限。

默认值：off

须知

用户在使用高级包UTL_FILE访问服务器端文件时，要求必须拥有所指定的DIRECTORY对象的权限。

出于安全考虑，默认情况下，只有初始用户才能够创建、修改、删除DIRECTORY对象。

如果开启了enable_access_server_directory，具有SYSADMIN权限的用户和继承了内置角色gs_role_directory_create权限的用户可以创建directory对象；具有SYSADMIN权限的用户、directory对象的属主、被授予了该directory的DROP权限的用户或者继承了内置角色gs_role_directory_drop权限的用户可以删除directory对象；具有SYSADMIN权限的用户和directory对象的属主可以修改directory对象的所有者，且要求该用户是新属主的成员。

18.3.22 事务监控

通过设置事务超时预警，可以监控自动回滚的事务并定位其中的语句问题，并且也可以监控执行时间过长的语句。

transaction_sync_naptime

参数说明：为保证数据一致性，当本地事务与GTM上snapshot中状态不一样时会阻塞其他事务的运行，需要等待本地节点上事务状态与GTM状态一致后再运行。当CN上等待时长超过transaction_sync_naptime时会主动触发gs_clean进行清理，缩短不一致时的阻塞时长。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0 ~ 2147483，单位为秒（s）。

默认值：30s

说明

若该值设为0，则不会在阻塞达到时长时主动调用gs_clean进行清理，而是靠**gs_clean_timeout**间隔来调用gs_clean，默认是5分钟。

transaction_sync_timeout

参数说明：为保证数据一致性，当本地事务与GTM上snapshot中状态不一样时会阻塞其他事务的运行，需要等待本地节点上事务状态与GTM状态一致后再运行。当CN上等待时长超过transaction_sync_timeout时会报错，回滚事务，避免由于sync lock等其他情况长时间进程停止响应造成对系统的阻塞。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0 ~ 2147483，单位为秒（s）。

默认值：10min

说明

- 若该值设为0，则不会在阻塞超时时报错，回滚事务。
- 该值必须大于gs_clean_timeout，避免DN上由于还未被gs_clean清理的残留事务阻塞超时引起的不必要的事务回滚。

18.3.23 CM 相关参数

CM相关参数的修改对GaussDB的运行机制有影响，建议由GaussDB的工程师协助修改。修改CM相关参数的方法，请参考**表18-2**中方式一进行设置。

cm_agent相关参数可通过cm_agent数据目录下的cm_agent.conf文件查看，cm_server相关参数可通过cm_server数据目录下的cm_server.conf文件查看。

18.3.23.1 cm_agent 参数

log_dir

参数说明：log_dir决定存放cm_agent日志文件的目录。可以是绝对路径，或者是相对路径（相对于cm_agent数据目录的路径）。

取值范围：字符串。修改后需要重启cm_agent才能生效。参数修改请参考**表18-2**进行设置。

默认值：“log”，表示在cm_agent数据目录下生成cm_agent日志。

log_file_size

参数说明：控制日志文件的大小。当日志文件达到指定大小时，则重新创建一个日志文件记录日志信息。

取值范围：整型，取值范围0~2047，单位为MB。修改后可以reload生效，参数修改请参考**表18-2**进行设置。

默认值：16MB

log_min_messages

参数说明：控制写到cm_agent日志文件中的消息级别。每个级别都包含排在它后面的所有级别中的信息。级别越低，服务器运行日志中记录的消息就越少。

取值范围：枚举类型，有效值有debug5、debug1、warning、error、log、fatal。修改后可以reload生效，参数修改请参考**表18-2**进行设置。

默认值：warning

incremental_build

参数说明：控制重建备DN模式是否为增量。打开这个开关，则增量重建备DN；否则，全量重建备DN。

取值范围：布尔型，有效值有on、off。修改后可以reload生效，参数修改请参考**表18-2**进行设置。

默认值：on

alarm_component

参数说明：设置用于处理告警内容的告警组件的位置。

取值范围：字符串。修改后可以reload生效，参数修改请参考**表18-2**进行设置。

- 若前置脚本gs_preinstall中的--alarm-type参数设置为5时，表示未对接第三方组件，告警写入system_alarm日志，此时GUC参数alarm_component的取值为：/opt/huawei/snas/bin/snas_cm_cmd。
- 若前置脚本gs_preinstall中的--alarm-type参数设置为1时，表示对接第三方组件，此时GUC参数alarm_component的值为第三方组件的可执行程序的绝对路径。

默认值：/opt/huawei/snas/bin/snas_cm_cmd

alarm_report_interval

参数说明：指定告警上报的时间间隔。参数修改请参考**表18-2**进行设置。

取值范围：非负整型，单位为秒。

默认值：1

alarm_report_max_count

参数说明：指定告警上报的最大次数。参数修改请参考**表18-2**进行设置。

取值范围：非负整型。

默认值：1

agent_report_interval

参数说明：cm_agent上报实例状态的时间间隔。

取值范围：整型，单位为秒。参数修改请参考**表18-2**进行设置。

默认值：1

agent_phony_dead_check_interval

参数说明：cm_agent检测CN/DN/GTM进程是否僵死的时间间隔。

取值范围：整型，单位为秒。参数修改请参考**表18-2**进行设置。

默认值：10

agent_check_interval

参数说明：cm_agent查询DN、CN、GTM等实例状态的时间间隔。

取值范围：整型，单位为秒。参数修改请参考**表18-2**进行设置。

默认值：2

agent_heartbeat_timeout

参数说明：cm_agent连接cm_server心跳超时时间。

取值范围：整型，2~2^{31} - 1，单位为秒。参数修改请参考**表18-2**进行设置。

默认值：8

agent_connect_timeout

参数说明：cm_agent连接cm_server超时时间。

取值范围：整型，单位为秒。参数修改请参考**表18-2**进行设置。

默认值：1

agent_connect_retries

参数说明：cm_agent连接cm_server尝试次数。

取值范围：整型。参数修改请参考**表18-2**进行设置。

默认值：15

agent_kill_instance_timeout

参数说明：当cm_agent在无法连接cm_server主节点后，发起一次终止本节点上所有实例的操作之前，所需等待的时间间隔。

取值范围：整型。参数修改请参考**表18-2**进行设置。

默认值：0，不发起终止本节点上所有实例的操作。

enable_gtm_phony_dead_check

参数说明：gtm僵死检查的开关。

取值范围：整型，1表示允许僵死检查，0表示不允许。参数修改请参考**表18-2**进行设置。

默认值：1

security_mode

参数说明：控制是否以安全模式启动CN、DN。打开这个开关，则以安全模式启动CN、DN；否则，以非安全模式启动CN、DN。

取值范围：布尔型，有效值有on、off。参数修改请参考**表18-2**进行设置。

默认值：off

upgrade_from

参数说明：就地升级过程中使用，用于标示升级前集群的内部版本号，此参数禁止手动修改。

取值范围：非负整型。参数修改请参考**表18-2**进行设置。

建议取值区间为[0,V]，V：安装包的版本号。

默认值：0

process_cpu_affinity

参数说明：控制是否以绑核优化模式启动主DN进程。配置该参数为0，则不进行绑核优化；否则，进行绑核优化，且物理CPU片数为2^n个。集群、cm_agent重启生效。仅支持ARM。参数修改请参考**表18-2**进行设置。

取值范围：整型，0~2。

默认值：0

enable_xc_maintenance_mode

参数说明：在集群为只读模式下，控制是否可以修改pgxc_node系统表。

取值范围：布尔型。修改后需要重启cm_agent才能生效。参数修改请参考**表18-2**进行设置。

- on表示开启可以修改pgxc_node系统表功能。
- off表示关闭可以修改pgxc_node系统表功能。

默认值：on

log_threshold_check_interval

参数说明：日志压缩和清除的时间间隔，每1800秒压缩和清理一次。

取值范围：整型，0~2^{31} - 1，单位为秒。参数修改请参考**表18-2**进行设置。

默认值：1800

dilatation_shard_count_for_disk_capacity_alarm

参数说明：扩容场景下，设置新增的扩容分片数，用于上报磁盘容量告警时的阈值计算。

说明

该分片数请与实际扩容分片数设置为一致。

取值范围：整型，0~2^{31} - 1，单位为个。该参数设置为0，表示关闭磁盘扩容告警上报；该参数设置为大于0，表示开启磁盘扩容告警上报，且告警上报的阈值根据此参数设置的分片数量进行计算。参数修改请参考**表18-2**进行设置。

默认值：1

log_max_size

参数说明：控制日志最大存储值。

取值范围：整型，0~2^{31} - 1，单位为MB。参数修改请参考**表18-2**进行设置。

默认值：10240

log_max_count

参数说明：硬盘上可存储的最多日志数量。

取值范围：整型，0~10000，单位为个。参数修改请参考**表18-2**进行设置。

默认值：10000

log_saved_days

参数说明：日志保存的天数。

取值范围：整型，0~1000，单位为天。参数修改请参考**表18-2**进行设置。

默认值：90

enable_log_compress

参数说明：控制压缩日志功能。

取值范围：布尔型。参数修改请参考**表18-2**进行设置。

- on表示允许压缩日志。
- off表示不允许压缩日志。

默认值：on

enable_cn_auto_repair

参数说明：CN自动修复开关。

取值范围：布尔型。参数修改请参考**表18-2**进行设置。

- on表示开启CN自动修复，即CN被剔除后，agent会尝试自动修复并加回CN。
- off表示不开启CN自动修复。

默认值：on

agent_backup_open

参数说明：灾备集群设置，开启后CM按照灾备集群模式运行。

取值范围：整型，0~1。修改后需要重启cm_agent才能生效。参数修改请参考**表18-2**进行设置。

- 0表示关闭。
- 1表示开启。

默认值：0

enable_e2e_rto

参数说明：端到端RTO开关，开启后僵死检测周期及网络检测超时时间将缩短，CM可以达到端到端RTO指标（单实例故障RTO<=10s，叠加故障RTO<=30s）。

取值范围：整型，0~1。1表示开启，0表示关闭。参数修改请参考**表18-2**进行设置。

默认值：

独立部署：1

金融版（标准型）、企业版、金融版（数据计算型）：0

enable_dcf

参数说明：DCF模式开关。

取值范围：布尔型。修改后需要重启cm_agent才能生效。参数修改请参考**表18-2**进行设置。

- 0表示关闭。
- 1表示开启。

默认值：off

unix_socket_directory

参数说明：UNIX套接字的目录位置。

取值范围：字符串。参数修改请参考**表18-2**进行设置。

默认值：''

disaster_recovery_type

参数说明：主备集群灾备关系的类型。

取值范围：整型，0~2。参数修改请参考**表18-2**进行设置。

- 0表示未搭建灾备关系。
- 1表示搭建了obs灾备关系。
- 2表示搭建了流式灾备关系

默认值：0

environment_threshold

参数说明：agent所监控的物理环境和节点状态信息的阈值，超过阈值会打印日志。具体分别表示为内存使用率阈值，cpu占用率阈值，磁盘使用率阈值，实例的内存使用率阈值，实例的线程池使用率阈值。

取值范围：字符串，（0，0，0，0，0），阈值范围为[0,100]，单位为%，0表示关闭检测。参数修改请参考**表18-2**进行设置。

默认值：（0，0，0，0，0）

18.3.23.2 cm_server 参数

log_dir

参数说明：log_dir决定存放cm_server日志文件的目录。它可以是绝对路径，或者是相对路径（相对于cm_server数据目录的路径）。

取值范围：字符串。修改后需要重启cm_server才能生效。参数修改请参考**表18-2**进行设置。

默认值：“log”，表示在cm_server数据目录下生成cm_server日志。

log_file_size

参数说明：控制日志文件的大小。当日志文件达到指定大小时，则重新创建一个日志文件记录日志信息。

取值范围：整型，取值范围0~2047，单位为MB。参数修改请参考**表18-2**进行设置。

默认值：16MB

log_min_messages

参数说明：控制写到cm_server日志文件中的消息级别。每个级别都包含排在它后面的所有级别中的信息。级别越低，服务器运行日志中记录的消息就越少。

取值范围：枚举类型，有效值有debug5、debug1、log、warning、error、fatal。参数修改请参考**表18-2**进行设置。

默认值：warning

thread_count

参数说明：cm_server线程池的线程数。该配置值如果大于集群节点数与处理cm_ctl请求的线程数（集群节点数小于32默认1个线程否则4个线程）之和，实际生效值为集群节点数与处理cm_ctl请求的线程数之和。

取值范围：整型，2~1000。修改后需要重启cm_server才能生效。参数修改请参考**表18-2**进行设置。

默认值：1000

alarm_component

参数说明：设置用于处理告警内容的告警组件的位置。

取值范围：字符串。参数修改请参考**表18-2**进行设置。

- 若前置脚本gs_preinstall中的--alarm-type参数设置为5时，表示未对接第三方组件，告警写入system_alarm日志，此时GUC参数alarm_component的取值为：/opt/huawei/snas/bin/snas_cm_cmd。
- 若前置脚本gs_preinstall中的--alarm-type参数设置为1时，表示对接第三方组件，此时GUC参数alarm_component的值为第三方组件的可执行程序的绝对路径。

默认值：/opt/huawei/snas/bin/snas_cm_cmd

instance_failover_delay_timeout

参数说明：cm_server检测到主机宕机，failover备机的延迟时间。

取值范围：整型，单位为秒。参数修改请参考**表18-2**进行设置。

默认值：0

instance_heartbeat_timeout

参数说明：实例心跳超时时间。

取值范围：整型，单位为秒。参数修改请参考**表18-2**进行设置。

默认值：6

coordinator_heartbeat_timeout

参数说明：CN故障自动剔除心跳超时时间。设置后立即生效，不需要重启cm_server。该参数设置为0，则CN故障后不会自动剔除。

取值范围：整型，单位为秒。参数修改请参考**表18-2**进行设置。

默认值：25

cmserver_ha_connect_timeout

参数说明：cm_server主备连接超时时间。

取值范围：整型，单位为秒。参数修改请参考**表18-2**进行设置。

默认值：2

cmserver_ha_heartbeat_timeout

参数说明：cm_server主备心跳超时时间。

取值范围：整型，单位为秒。参数修改请参考**表18-2**进行设置。

默认值：6

phony_dead_effective_time

参数说明：用于CN/DN/GTM进程的僵死检测，当检测到的僵死次数大于该参数值，认为进程僵死，将进程重启。

取值范围：整型，单位为次数。参数修改请参考**表18-2**进行设置。

默认值：5

enable_transaction_read_only

参数说明：控制是否打开cm_server磁盘自动阈值检测功能，该功能打开后，当磁盘使用率大于datastorage_threshold_value_check值时，cm_server会自动将数据库设置为只读模式。

取值范围：布尔型，有效值有on，off，true，false，yes，no，1，0。参数修改请参考**表18-2**进行设置。

默认值：on

datastorage_threshold_check_interval

参数说明：检测磁盘占用的时间间隔。间隔用户指定时间，检测一次磁盘占用。

取值范围：整型，单位为秒。参数修改请参考**表18-2**进行设置。

默认值：10

datastorage_threshold_value_check

参数说明：设置数据库只读模式的磁盘占用阈值，当数据目录所在磁盘占用超过这个阈值，自动将数据库设置为只读模式。调整该参数时，建议同步调整dn的max_size_for_xlog_retention参数，避免因备份操作触发集群只读阈值。

取值范围：整型，1 ~ 99，表示百分比。参数修改请参考**表18-2**进行设置。

默认值：85

max_datastorage_threshold_check

参数说明：设置磁盘使用率的最大检测间隔时间。当用户手动修改只读模式参数后，会自动在指定间隔时间后开启磁盘满检测操作。

取值范围：整型，单位为秒。参数修改请参考**表18-2**进行设置。

默认值：43200

cmserver_ha_status_interval

参数说明：cm_server主备同步状态信息间隔时间。

取值范围：整型，单位为秒。参数修改请参考**表18-2**进行设置。

默认值：1

cmserver_self_vote_timeout

参数说明：cm_server自仲裁超时时间。

取值范围：整型，单位为秒。参数修改请参考**表18-2**行设置。

默认值：6

alarm_report_interval

参数说明：指定告警上报的时间间隔。

取值范围：非负整型，单位为秒。参数修改请参考**表18-2**进行设置。

默认值：3

alarm_report_max_count

参数说明：指定告警上报的最大次数。

取值范围：非负整型。参数修改请参考**表18-2**进行设置。

默认值：1

enable_az_auto_switchover

参数说明：AZ自动切换开关，若打开，则表示允许cm_server自动切换AZ。否则当发生dn故障等情况时，即使当前AZ已经不再可用，也不会自动切换到其它AZ上，除非手动执行切换命令。

取值范围：非负整型，0或1，0表示开关关闭，1表示开关打开。参数修改请参考**表18-2**进行设置。

默认值：1

instance_keep_heartbeat_timeout

参数说明：cm_agent会定期检测实例状态并上报给cm_server，若实例状态长时间无法成功检测，累积次数超出该数值，则cm_server将下发命令给agent重启该实例。

取值范围：整型，单位为秒。参数修改请参考**表18-2**进行设置。

默认值：40

az_switchover_threshold

参数说明：若一个AZ内DN分片的故障率（故障的dn分片数 / 总dn分片数 * 100%）超过该数值，则会触发AZ自动切换。

取值范围：整型，0~100。参数修改请参考**表18-2**进行设置。

默认值：100

az_check_and_arbitrate_interval

参数说明：当某个AZ状态不正常时，会触发AZ自动切换，该参数是检测AZ状态的时间间隔。

取值范围：整型，单位为秒。参数修改请参考**表18-2**进行设置。

默认值：2

az_connect_check_interval

参数说明：定时检测AZ间的网络连接，该参数表示连续两次检测之间的间隔时间。

取值范围：整型，单位为秒。参数修改请参考**表18-2**进行设置。

默认值：60

az_connect_check_delay_time

参数说明：每次检测AZ间的网络连接时有多次重试，该参数表示两次重试之间的延迟时间。

取值范围：整型，单位为秒。参数修改请参考**表18-2**进行设置。

默认值：150

cmserver_demote_delay_on_etcd_fault

参数说明：因为etcd不健康而导致cm_server从主降为备的时间间隔。

取值范围：整型，单位为秒。参数修改请参考**表18-2**进行设置。

默认值：8

instance_phony_dead_restart_interval

参数说明：当cn/dn/gtm实例僵死时，会被cm_agent重启，相同的实例连续因僵死被杀时，其间隔时间不能小于该参数数值，否则cm_agent不会下发命令。

取值范围：整型，1800~2^{31} - 1，单位为秒。参数修改请参考**表18-1**进行设置。

默认值：21600

cm_auth_method

参数说明：CM模块端口认证方式，trust表示未配置端口认证，gss表示采用kerberos端口认证。必须注意的是：只有当kerberos服务端和客户端成功安装后才能修改为gss，否则CM模块无法正常通信，将影响集群状态

取值范围：枚举类型，有效值有trust, gss。参数修改请参考**表18-1**进行设置。

默认值：trust

cm_krb_server_keyfile

参数说明：kerberos服务端key文件所在位置，需要配置为绝对路径。该文件通常为${GAUSSHOME}/kerberos路径下，以keytab格式结尾，文件名与集群运行所在用户名相同。与上述cm_auth_method参数是配对的，当cm_auth_method参数修改为gss时，该参数也必须配置为正确路径，否则将影响集群状态

取值范围：字符串类型，参数修改请参考**表18-1**进行设置。

默认值：${GAUSSHOME}/kerberos/{UserName}.keytab，默认值无法生效，仅作为提示

cm_server_arbitrate_delay_base_time_out

参数说明：cm_server仲裁延迟基础时长。cm_server主断连后，仲裁启动计时开始，经过仲裁延迟时长后，将选出新的cm_server主。其中仲裁延迟时长由仲裁延迟基础时长、节点index（server ID序号）和增量时长共同决定。公式为：仲裁延迟时长=仲裁延迟基础时长+节点index*仲裁延迟增量时长参数

取值范围：整型，index>0，单位为秒。参数修改请参考**表18-1**进行设置。

默认值：10

cm_server_arbitrate_delay_incrememtal_time_out

参数说明：cm_server仲裁延迟增量时长。cm_server主断连后，仲裁启动计时开始，经过仲裁延迟时长后，将选出新的cm_server主。其中仲裁延迟时长由仲裁延迟基础时长、节点index（server ID序号）和增量时长共同决定。公式为：仲裁延迟时长=仲裁延迟基础时长+节点index*仲裁延迟增量时长参数

取值范围：整型，index>0，单位为秒。参数修改请参考**表18-2**进行设置。

默认值：3

force_promote

参数说明：cm_server是否打开强起逻辑（指集群状态为Unknown的时候以丢失部分数据为代价保证集群基本功能可用）的开关。0代表功能关闭，1代表功能开启。该参数同时适用于cn和dn。

取值范围：整型，0~1。参数修改请参考**表18-2**进行设置。

默认值：0

switch_rto

参数说明：cm_server强起逻辑等待时延。在force_promote被置为1时，当集群的某一分片处于无主状态开始计时，等待该延迟时间后开始执行强起逻辑。

取值范围：整型，0~2147483647，单位为秒。最小生效值为60，若设置参数值小于此值实际生效值为最小生效值。参数修改请参考**表18-2**进行设置。

默认值：600

backup_open

参数说明：灾备集群设置，开启后CM按照灾备集群模式运行

取值范围：整型，0~1。修改后需要重启cm_server才能生效。非灾备集群不能开启该参数。参数修改请参考**表18-2**进行设置。

- 0表示关闭。
- 1表示开启。

默认值：0

enable_e2e_rto

参数说明：端到端RTO开关，开启后僵死检测周期及网络检测超时时间将缩短，CM可以达到端到端RTO指标（单实例故障RTO<=10s，叠加故障RTO<=30s）。

取值范围：整型，0~1。1表示开启，0表示关闭。参数修改请参考**表18-2**进行设置。

默认值：

独立部署：1

金融版（标准型）、企业版、金融版（数据计算型）：0

cluster_starting_aribt_delay

参数说明：cm_server在集群启动阶段，等待DN静态主升主的时间。

取值范围：整型，单位为秒。参数修改请参考**表18-2**进行设置。

默认值：180

enable_dcf

参数说明：DCF模式开关。

取值范围：布尔型。修改后需要重启cm_server才能生效。参数修改请参考**表18-2**进行设置。

- 0表示关闭。
- 1表示开启。

默认值：off

ddb_type

参数说明：etcd，dcc模式切换开关。

取值范围：整型。0：etcd；1：dcc。修改后需要重启cm_server才能生效。参数修改请参考**表18-2**进行设置。

默认值：0

enable_ssl

参数说明：ssl证书开关。

取值范围：布尔型。打开后使用ssl证书加密通信。修改后需要重启才能生效。参数修改请参考**表18-2**进行设置。

默认值：

- on表示启用ssl。
- off表示不启用ssl。
- **默认值**：off

须知

出于安全性考虑，建议不要关闭该配置。关闭后cm将**不使用**加密通信，所有信息明文传播，可能带来窃听、篡改、冒充等安全风险。

ssl_cert_expire_alert_threshold

参数说明：ssl证书过期告警时间。

取值范围：整型，单位为天。证书过期时间少于该时间时，上报证书即将过期告警。修改后需要重启才能生效。参数修改请参考**表18-2**进行设置。

默认值：90

ssl_cert_expire_check_interval

参数说明：ssl证书过期检测周期。

取值范围：整型，单位为秒。修改后需要重启才能生效。参数修改请参考**表18-2**进行设置。

默认值：86400

ddb_log_level

参数说明：设置ddb日志级别。

关闭日志：“NONE”，NONE表示关闭日志打印，不能与以下日志级别混合使用。

开启日志：“RUN_ERR|RUN_WAR|RUN_INF|DEBUG_ERR|DEBUG_WAR|DEBUG_INF|TRACE|PROFILE|OPER”日志级别可以从上述字符串中选取字符串并使用竖线组合使用，不能配置空串。

取值范围：字符串，RUN_ERR|RUN_WAR|RUN_INF|DEBUG_ERR|DEBUG_WAR|DEBUG_INF|TRACE|PROFILE|OPER。参数修改请参考**表18-2**进行设置。

默认值：RUN_ERR|RUN_WAR|DEBUG_ERR|OPER|RUN_INF|PROFILE

ddb_log_backup_file_count

参数说明：最大保存日志文件个数。

取值范围：整型，[1, 100]。参数修改请参考**表18-2**进行设置。

默认值：10

ddb_max_log_file_size

参数说明：单条日志最大字节数。

取值范围：字符串，[1MB, 1000MB]。参数修改请参考**表18-2**进行设置。

默认值：10MB

ddb_log_suppress_enable

参数说明：是否开启日志抑制功能。

取值范围：整型，0：关闭；1：开启。参数修改请参考**表18-2**进行设置。

默认值：1

ddb_election_timeout

参数说明：dcc选举超时时间。

取值范围：整型，[1, 600]，单位：秒。参数修改请参考**表18-2**进行设置。

默认值：3

delay_arbitrate_timeout

参数说明：设置等待跟主DN同AZ节点redo回放后升主的时间。

取值范围：整型，[0, 21474836]，单位：秒。参数修改请参考**表18-2**进行设置。

默认值：0

install_type

参数说明：容灾集群相关的设置，用来区别集群的类型。

取值范围：整型，0~2。修改后需要重启cm_server才能生效。非灾备集群不能开启该参数。参数修改请参考**表18-1**进行设置。

默认值：0

- 0表示未搭建容灾关系的集群。
- 1表示基于dorado的集群。
- 2表示基于流式的集群。

18.3.24 GTM 相关参数

GTM相关参数可以在gtm.conf文件中进行设置，或通过gs_guc进行设置。

nodename

参数说明：主GTM或备GTM名称。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串，要符合标识符命名规范（具体可参见《开发指南》中“SQL参考 > 关键字”章节的“标识符命名规范”）。

默认值：NULL

port

参数说明：主GTM或备GTM侦听的主机端口号。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0～2147483647，建议设置为1024 ~ 65535。

默认值：6666

log_file

参数说明：日志文件名。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串，要符合标识符命名规范（具体可参见《开发指南》中“SQL参考 > 关键字”章节的“标识符命名规范”）。

默认值：gtm-%Y-%m-%d_%H%M%S.log

active_host

参数说明：目标GTM的地址。即在主GTM上时为备GTM的地址，在备GTM上时为主GTM的地址。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串，要符合标识符命名规范（具体可参见《开发指南》中“SQL参考 > 关键字”章节的“标识符命名规范”）。

默认值：NULL

local_host

参数说明：HA本地地址，根据集群配置文件进行设置，不需要手动设置。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串，要符合标识符命名规范（具体可参见《开发指南》中“SQL参考 > 关键字”章节的“标识符命名规范”）。

默认值：NULL

active_port

参数说明：目标GTM的服务器端口号。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0～2147483647，建议设置为1024 ~ 65535。

说明

该参数由安装时的配置文件指定，请勿轻易修改，否则修改后会影响数据库正常通信。

默认值：0

local_port

参数说明：HA本地端口。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0～2147483647，建议设置为1024 ~ 65535。

说明

该参数由安装时的配置文件指定，请勿轻易修改，否则修改后会影响数据库正常通信。

默认值：0

standby_connection_timeout

参数说明：设置GTM主备之间的超时时间。此参数控制GTM主备机之间的超时设置，增大可以增加GTM主备之间的网络容错能力，但是也会增加故障场景下GTM主备断连的检测时长。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，5～2147483647（秒）。

默认值：5

keepalives_count

参数说明：在支持TCP_KEEPCNT套接字选项的操作系统上，设置GTM服务端在断开与客户端连接之前可以等待的保持活跃信号个数。该参数仅在备GTM上生效。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0～2147483647。

默认值：0，表示GTM未收到客户端反馈的保持活跃信号则立即断开连接。

keepalives_idle

参数说明：发送活跃信号的间隔秒数。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0～2147483647（秒）。

默认值：0

keepalives_interval

参数说明：在支持TCP_KEEPINTVL套接字选项的操作系统上，以秒数声明在重新传输之间等待响应的时间。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0～2147483647（秒）。

默认值：0

synchronous_backup

参数说明：指定是否以同步方式备份到GTM备机。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：off/on/auto。

- on表示开启同步方式。
- off表示关闭同步方式。
- auto表示自动同步方式。

默认值：auto

query_memory_limit

参数说明：设置查询可以使用的内存百分比限制。该参数仅适用于默认资源组。对于其它的资源组，没有查询内存限制影响。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：0.0~1.0。

默认值：0.25

wlm_max_mem

参数说明：设置GTM执行时可使用的最大内存。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，512MB～2147483647。

默认值：2048

config_file

参数说明：GTM配置文件名，仅sysadmin用户可以访问。

取值范围：字符串。请参考**表18-1**中对应设置方法进行设置。

默认值：gtm.conf

data_dir

参数说明：GTM数据文件目录。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串。

默认值：NULL

listen_addresses

参数说明：声明服务器侦听客户端的TCP/IP地址。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：

- 主机名或IP地址，多个值之间用英文逗号分隔。
- 星号（*）表示所有IP地址。
- 置空则服务器不会侦听任何IP地址，这种情况下，只有UNIX域套接字可以用于连接数据库。

默认值：*

log_directory

参数说明：当**logging_collector**设置为on时，log_directory决定存放服务器日志文件的目录。 它可以是绝对路径，或者是相对路径（相对于数据目录的路径）。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

- 当配置文件中log_directory的值为非法路径（即用户对此路径无读写权限）时，会导致集群无法重新启动。
- 修改log_directory时，当指定路径为合法路径（即用户对此路径有读写权限）时，日志输出到新的路径下。当指定路径为非法路径时，日志输出到上一次的合法日志输出路径下而不影响数据库正常运行。此时即使指定的log_directory的值非法，也会写入到配置文件中。

取值范围：字符串

默认值：“gtm_log”，表示在数据目录下的“gtm_log/”目录下生成服务器日志。

log_min_messages

参数说明：控制写到服务器日志文件中的消息级别。每个级别都包含排在它后面的所有级别中的信息。级别越低，服务器运行日志中记录的消息就越少。

须知

当client_min_messages和log_min_messages取值相同时，其值所代表的级别不同。

取值范围：枚举类型，有效值有debug、debug5、debug4、debug3、debug2、debug1、info、log、notice、warning、error、fatal、panic。参数的详细信息请参见**表18-8**。

默认值：warning

alarm_component

参数说明：在对告警做上报时，会进行告警抑制，即同一个实例的同一个告警项在**alarm_report_interval**（默认值为10s）内不做重复上报。在这种情况下设置用于处理告警内容的告警组件的位置。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

- 若前置脚本gs_preinstall中的--alarm-type参数设置为5时，表示未对接第三方组件，告警写入system_alarm日志，此时GUC参数alarm_component的取值为：/opt/huawei/snas/bin/snas_cm_cmd。
- 若前置脚本gs_preinstall中的--alarm-type参数设置为1时，表示对接第三方组件，此时GUC参数alarm_component的值为第三方组件的可执行程序的绝对路径。

默认值：/opt/huawei/snas/bin/snas_cm_cmd

alarm_report_interval

参数说明：指定告警上报的时间间隔。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：非负整型，单位为秒。

默认值：10

standby_only

参数说明：是否强制同步信息到备机，在一主多备模式下只强制同步到ETCD。

取值范围：整型，取值为0或1，请参考**表18-1**中对应设置方法进行设置。

- 0表示不强制同步信息到备机。
- 1表示强制同步信息到备机。

默认值：0

gtm_max_trans

参数说明：设置gtm最大可接收连接数，不建议用户修改该参数。如果需要改动，此参数不能小于最大连接数加100。

取值范围：整型，256~200000，请参考**表18-1**中对应设置方法进行设置。

默认值：8192

enable_connect_control

参数说明：设置gtm开启连接控制，检测连接IP是否来自集群内部。

取值范围：布尔型，请参考**表18-1**中对应设置方法进行设置。

- true：检测连接IP是否来自集群内部，非集群内部的IP连接则拒绝访问。
- false：不检测连接IP是否来自集群内部。

默认值：true

gtm_authentication_type

参数说明：GTM模块端口认证方式，trust表示未配置端口认证，gss表示采用kerberos端口认证。必须注意的是：只有当kerberos服务端和客户端成功安装后才能修改为gss，否则GTM模块无法正常通信，将影响集群状态。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：枚举类型，有效值有trust, gss。

默认值：trust

gtm_krb_server_keyfile

参数说明：kerberos服务端key文件所在位置，需要配置为绝对路径。该文件通常为${GAUSSHOME}/kerberos路径下，以keytab格式结尾，文件名与集群运行所在用户名相同。与上述gtm_authentication_type参数是配对的，当gtm_authentication_type参数修改为gss时，该参数也必须配置为正确路径，否则将影响集群状态。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串类型

默认值：''

gtm_option

参数说明：GTM模式选项，用来指定选用的GTM模式，需要在GTM，CN，DN所有实例上配置，配置的值要一致，共三种模式：GTM模式，GTM-Lite模式，GTM-Free模式。其中GTM模式和GTM-Lite模式要在enable_gtm_free参数设置为off的情况下生效，当前版本暂不支持安装好的集群进行不同GTM模式之间的切换。

取值范围：整型，0~2，0表示GTM模式，1表示GTM-Lite模式，2表示GTM-Free模式。请参考**表18-1**中对应设置方法进行设置。

默认值：1

csn_sync_interval

参数说明：用来指定GTM主备之间同步CSN的时间间隔，单位为秒（s）。

取值范围：整型，1~2147483647之间。请参考**表18-1**中对应设置方法进行设置。

默认值：1

restore_duration

参数说明：用来指定gtm上xid或csn的回复间隔（个数）。

取值范围：整型，1000000~2147483647之间。请参考**表18-1**中对应设置方法进行设置。

默认值：1000000

gtm_enable_threadpool

参数说明：用来指定GTM是否开启GTM线程池功能，设置后需要重启才能生效。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型。

默认值：true

gtm_num_threads

参数说明：当gtm_enable_threadpool线程池功能开启时，用来控制线程池工作线程的个数。

该数值与gtm_max_trans大小相关，不应该超过（gtm_max_trans - 1 - 辅助线程数），其中辅助线程数当前版本为2。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~16384之间。

默认值：1024

18.3.25 升级参数

IsInplaceUpgrade

参数说明：是否在升级的过程中。该参数属于升级参数，用户无法修改，仅sysadmin用户可以访问。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示在升级过程中。
- off表示不在升级过程中。

默认值：off

inplace_upgrade_next_system_object_oids

参数说明：就地升级过程中，新增系统对象的OID。该参数属于升级参数，用户无法修改。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

默认值：空

upgrade_mode

参数说明：升级模式。该参数属于升级参数，不建议用户自己修改。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：整型，0~2147483647

- 0表示不在升级过程中或者就地升级和灰度升级的小版本升级过程中。
- 1表示在就地升级大版本升级过程中（执行升级命令，过了检查阶段生效）。
- 2表示在灰度升级大版本升级过程中（执行升级命令，过了检查阶段生效）。

默认值：0

说明

用户执行完新包的前置命令，切回集群用户，source环境变量后，通过gs_upgradectl -t chose-strategy命令查询是大版本升级还是小版本升级。

返回Upgrade strategy: large-binary-upgrade 代表大版本升级。

返回Upgrade strategy: small-binary-upgrade 代表小版本升级。

18.3.26 其它选项

server_version

参数说明：报告服务器版本号(字符串形式)。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。该参数继承自PostgreSQL内核，表示当前数据库内核兼容PostgreSQL对应的server_version版本，无实际含义，为保持北向对外工具接口的生态兼容性(工具连接时查询)，保留该参数。该参数不建议使用，可通过函数opengauss_version()获取内核版本信息。

取值范围：字符串

默认值：9.2.4

server_version_num

参数说明：报告服务器版本号(整数形式)。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。该参数继承自PostgreSQL内核，表示当前数据库内核兼容PostgreSQL对应的server_version_num版本，无实际含义，为保持北向对外工具接口的生态兼容性(工具连接时查询)，保留该参数。

取值范围：整型

默认值：90204

block_size

参数说明：报告当前数据库所使用的块大小。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。该参数继承自PostgreSQL内核，表示当前数据库内核兼容PostgreSQL对应的server_version_num版本，无实际含义，为保持北向对外工具接口的生态兼容性，保留该参数。

默认值：8192

segment_size

参数说明：报告当前数据库所使用的段文件大小。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

默认值：1GB

max_index_keys

参数说明：报告当前数据库能够支持的索引键值的最大数目。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

默认值：32

integer_datetimes

参数说明：报告是否支持64位整数形式的日期和时间格式。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：布尔型

- on表示支持。
- off表示不支持。

默认值：on

enable_cluster_resize

参数说明：对于sql语句中涉及多个表，并且属于不同group，打开此开关可以支持此语句执行计划下推来提高性能。

该参数属于SUSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示支持此语句执行计划下推来提高性能。
- off表示不支持此语句执行计划下推来提高性能。

默认值：off

说明

此参数用于内部运维场景，请勿随意开启。

lc_collate

参数说明：报告当前数据库的字符串排序区域设置。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

默认值：依赖于集群安装部署时的配置

lc_ctype

参数说明：报告当前数据库的字母类别区域设置。如：哪些字符属于字母，它对应的大写形式是什么。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

默认值：依赖于集群安装部署时的配置

max_identifier_length

参数说明：报告当前系统允许的标识符最大长度。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：整型

默认值：63

server_encoding

参数说明：报告当前数据库的服务端编码字符集。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

默认值：在创建数据库的时候决定的。

enable_upgrade_merge_lock_mode

参数说明：当该参数设置为on时，通过提升deltamerge内部实现的锁级别，避免和update/delete并发操作时的报错。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on，提升deltamerge内部实现的锁级别，并发执行deltamerge和update/delete操作时，一个操作先执行，另一个操作被阻塞，在前一个操作完成后，后一个操作再执行。
- off，在对表的delta table的同一行并发执行deltamerge和update/delete操作时，后一个对同一行数据更新的操作会报错退出。

默认值：off

transparent_encrypt_kms_region

参数说明：它存储的是整个集群的部署区域，内容要求不可出现RFC3986标准外的字符，最大长度2047字节。该参数当前版本只适用于DWS场景。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串

默认值：空

datanode_heartbeat_interval

参数说明：设置心跳线程间心跳消息发送时间间隔，建议值不超过wal_receiver_timeout / 2。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1000～60000（毫秒）

默认值：1s

dfs_partition_directory_length

参数说明：在HDFS文件系统上，构造HDFS VALUE分区表的分区目录时，目录名长度的上限值。

该参数属于USERSET类型参数，请参考表**表18-1**中对应设置方法进行设置。

取值范围：92-7999

默认值：512

max_concurrent_autonomous_transactions

参数说明：自治事务最大连接数，同一时间自治事务执行的最大并发数。当设置为0时，将无法执行自治事务。

该参数属于POSTMASTER类型参数，请参考表**表18-1**中对应设置方法进行设置。

取值范围：0-1024

默认值：10

mot_config_file

该参数在分布式不可用。

enable_gpi_auto_update

参数说明：控制在分区DDL命令中是否默认更新Global索引。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示默认更新Global索引，此时分区DDL无论带不带UPDATE GLOBAL INDEX子句，都会更新Global索引。

- off表示默认不更新Global索引，此时只有当分区DDL带UPDATE GLOBAL INDEX子句，才会更新Global索引。

默认值：off

18.3.27 等待事件

enable_instr_track_wait

参数说明：是否开启等待事件信息实时收集功能。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on：表示打开等待事件信息收集功能。
- off：表示关闭等待事件信息收集功能。

默认值：on

18.3.28 Query

instr_unique_sql_count

参数说明：控制系统中unique sql信息实时收集功能。配置为0表示不启用unique sql信息收集功能。

该值由大变小将会清空系统中原有的数据重新统计(备机不支持此能力)；从小变大不受影响。

当系统中产生的unique sql信息大于instr_unique_sql_count时，系统产生的unique sql信息不被统计。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~2147483647

默认值：200000

instr_unique_sql_track_type

参数说明：unique sql记录SQL方式。

该参数属于INTERNAL类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：枚举类型

top：只记录顶层SQL。

默认值：top

unique_sql_retention_time

参数说明：清理unique sql哈希表的间隔，默认为30分钟

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1~3650

默认值：30min

enable_instr_rt_percentile

参数说明：是否开启计算系统中80%和95%的SQL响应时间的功能

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on：表示打开sql响应时间信息计算功能。
- off：表示关闭sql响应时间信息计算功能。

默认值：on

percentile

参数说明：sql响应时间百分比信息，后台计算线程根据设置的值计算相应的百分比信息。

该参数属于INTERNAL类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串。

默认值："80，95"

instr_rt_percentile_interval

参数说明：sql响应时间信息计算间隔，sql响应时间信息计算功能打开后，后台计算线程每隔设置的时间进行一次计算。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0～3600（秒）。

默认值：10s

enable_instr_cpu_timer

参数说明：是否捕获sql执行的cpu时间消耗 。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on：表示捕获sql执行的cpu时间消耗。
- off：表示不捕获sql执行的cpu时间消耗。

默认值：on

enable_slow_query_log（废弃）

参数说明：是否将慢查询信息写到日志文件中，在该版本中已废弃。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on：表示需要将慢查询信息写到日志文件中。
- off：表示不需要将慢查询信息写到日志文件中。

默认值：on

query_log_file（废弃）

参数说明：GUC参数enable_slow_query_log设置为ON，表示需要将慢查询记录写进日志文件中，query_log_file决定服务器慢查询日志文件的名称，仅sysadmin用户可以访问。通常日志文件名是按照strftime模式生成，因此可以用系统时间定义日志文件名，用%转义字符实现，在该版本中已废弃。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

建议使用%转义字符定义日志文件名称，否则难以对日志文件进行有效的管理。

取值范围：字符串

默认值：slow_query_log-%Y-%m-%d_%H%M%S.log

query_log_directory（废弃）

参数说明：enable_slow_query_log设置为on时，query_log_directory决定存放服务器慢查询日志文件的目录，仅sysadmin用户可以访问。它可以是绝对路径，或者是相对路径（相对于数据目录的路径），在该版本中已废弃。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

当配置文件中query_log_directory的值为非法路径时，会导致集群无法重新启动。

说明

合法路径：用户对此路径有读写权限
非法路径：用户对此路径无读写权限

取值范围：字符串

默认值：安装时指定。

asp_log_directory

参数说明：asp_flush_mode设置为all或者file时，asp_log_directory决定存放服务器asp日志文件的目录。它可以是绝对路径，或者是相对路径（相对于数据目录的路径），仅sysadmin用户可以访问。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

须知

当配置文件中asp_log_directory的值为非法路径时，会导致集群无法重新启动。

说明

- 合法路径：用户对此路径有读写权限。
- 非法路径：用户对此路径无读写权限。

取值范围：字符串

默认值：安装时指定。

perf_directory

参数说明：perf_directory决定性能视图打点任务输出文件的目录，仅sysadmin用户可以访问。它可以是绝对路径，或者是相对路径（相对于数据目录的路径）。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

说明

- 合法路径：用户对此路径有读写权限。
- 非法路径：用户对此路径无读写权限。

取值范围：字符串

默认值：安装时指定。

enable_stmt_track

参数说明：控制是否启用Full /Slow SQL特性。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on：表示开启Full /Slow SQL捕获
- off：表示关闭Full /Slow SQL捕获

默认值：on

track_stmt_parameter

参数说明：开启track_stmt_parameter后，在statement_history中记录的执行语句不再进行归一化操作，可以显示完整SQL语句信息，辅助DBA进行问题定位；其中对于简单查询，显示完整语句信息；对于PBE语句，显示完整语句信息的同时，追加每个变量数值信息，格式为“query string；parameters:$1=value1,$2=value2,...”，该参数提供的目的是为用户呈现全量SQL信息，不受track_activity_query_size参数控制。对于PBE类型语句且走SQL Bypass逻辑时，参数直接下发到DN，故在CN查询statement_history无法获取完整语句数信息，同时由于DN无Query字符串信息，故在DN查询statement_history中也无法获取完整语句信息。

该参数属于SIGHUP类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：布尔型

- on：表示开启显示完整SQL语句信息的功能。
- off：表示关闭显示完整SQL语句信息的功能。

默认值：off

track_stmt_session_slot

参数说明：设置一个session缓存的最大的全量/慢SQL的数量，超过这个数量，新的语句执行将不会被跟踪，直到落盘线程将缓存语句落盘，留出空闲的空间。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0 ~ 2147483647

默认值：1000

track_stmt_details_size

参数说明：设置单语句可以收集的最大的执行事件的大小(byte)。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0 ~ 100000000

默认值：4096

track_stmt_retention_time

参数说明：组合参数，控制全量/慢SQL记录的保留时间。以60秒为周期读取该参数，并执行清理超过保留时间的记录，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符型

该参数分为两部分，形式为'full sql retention time, slow sql retention time'：

- full sql retention time为全量SQL保留时间，取值范围为0 ~ 86400。
- slow sql retention time为慢SQL的保留时间，取值范围为0 ~ 604800。

默认值：3600,604800

track_stmt_stat_level

参数说明：控制语句执行跟踪的级别。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置，不区分英文字母大小写，如果打开full sql功能会影响性能，并可能会占用大量的磁盘空间。

取值范围：字符型

该参数分为两部分，形式为'full sql stat level, slow sql stat level'：

- 第一部分为全量SQL跟踪级别，取值范围为OFF、L0、L1、L2。
- 第二部分为慢SQL的跟踪级别，取值范围为OFF、L0、L1、L2。

说明

若全量SQL跟踪级别值为非OFF时，当前SQL跟踪级别值为全量SQL和慢SQL的较高级别（L2 > L1 >L0）。

默认值：OFF,L0

18.3.29 系统性能快照

enable_wdr_snapshot

参数说明：是否开启数据库监控快照功能。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on：打开数据库监控快照功能。
- off：关闭数据库监控快照功能。

默认值：on

wdr_snapshot_retention_days

参数说明：系统中数据库监控快照数据的保留天数，超过设置的值之后，系统每隔wdr_snapshot_interval时间间隔，清理snapshot_id最小的快照数据。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1～8，单位为天。

默认值：8

wdr_snapshot_query_timeout

参数说明：系统执行数据库监控快照操作时，设置快照操作相关的sql语句的执行超时时间。如果语句超过设置的时间没有执行完并返回结果，则本次快照操作失败。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，100～2147483647（秒）。

默认值：100s

wdr_snapshot_interval

参数说明：后台线程Snapshot自动对数据库监控数据执行快照操作的时间间隔。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，10～60（分钟）。

默认值：1h

enable_asp

参数说明：是否开启活跃会话信息active session profile。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on：打开active session profile功能。
- off：关闭active session profile功能。

默认值：on

asp_sample_num

参数说明：LOCAL_ACTIVE_SESSION视图最大的样本个数，仅sysadmin用户可以访问。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，10000～100000。

默认值：100000

asp_sample_interval

参数说明：每次采样的间隔。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1~10(秒)。

默认值：1s

asp_flush_rate

参数说明：当内存中样本个数达到asp_sample_num时，会按一定比例把内存中样本刷新到磁盘上，asp_flush_rate为刷新比例。该参数为10时表示按10：1进行刷新。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1~10。

默认值：10

asp_flush_mode

参数说明：ASP刷新到磁盘上的方式分为写文件和写系统表，当为‘file’时，默认写文件，为‘table’时写系统表，为‘all’时，即写文件也写系统表，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串，‘table’、‘file’、‘all’。

默认值：‘table’

asp_retention_days

参数说明：当ASP样本写到系统表时，该参数表示保留的最大天数。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1~7，单位为天。

默认值：2

asp_log_filename

参数说明：当ASP写文件时，该参数设置文件名的格式，仅sysadmin用户可以访问。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串。

默认值："asp-%Y-%m-%d_%H%M%S.log"

18.3.30 安全配置

enable_security_policy

参数说明：安全策略开关，控制统一审计和数据动态脱敏策略是否生效。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型。

on：安全策略开关打开。

off：安全策略开关关闭。

默认值：off

use_elastic_search

参数说明：使能统一审计发送日志至Elastic Search系统, enable_security_policy打开且本参数打开后，统一审计日志会通过http(https)传递至Elastic Search系统（默认使用https安全协议）。此参数打开后需要保证elastic_search_ip_addr对应的es服务可正常连通，否则进程启动失败。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型。

on：使能统一审计日志发送至Elastic Search。

off：关闭统一审计日志发送至Elastic Search。

默认值：off

elastic_search_ip_addr

参数说明：Elastic Search系统IP地址 。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串。

默认值：'https:127.0.0.1'

is_sysadmin

参数说明：表示当前用户是否是初始用户。

该参数属于INTERNAL类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：布尔型。

on表示是初始用户。

off表示不是初始用户。

默认值：off

18.3.31 HyperLogLog

hll_default_log2m

参数说明：该参数可以指定hll数据结构桶的个数。桶的个数会影响hll计算distinct值的精度，桶的个数越多，误差越小。误差范围为：$[-1.04/2^{log2m*1/2},+1.04/2^{log2m*1/2}]$。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，10~16。

默认值：14

hll_default_log2explicit

参数说明：该参数可以用来设置从Explicit模式到Sparse模式的默认阈值大小。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~12。0表示跳过Explicit模式，取1-12表示在基数到达$2^{hll_default_log2explicit}$时切换模式。

默认值：10

hll_default_log2sparse

参数说明：该参数可以用来设置从Sparse模式到Full模式的默认阈值大小。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，0~14。0表示跳过Explicit模式，取1-14表示在基数到达$2^{hll_default_log2sparse}$时切换模式。

默认值：12

hll_duplicate_check

参数说明：该参数可以用来指定是否默认开启duplicatecheck。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：0，1。0表示默认关闭，1表示默认开启

默认值：0

hll_default_regwidth（废弃）

参数说明：该参数可以指定hll数据结构每个桶的位数，该值越大，hll所占内存越高。hll_default_regwidth和hll_default_log2m可以决定当前hll能够计算的最大distinct value。当前regwidth设为固定值，该参数不再使用。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，1~5。

默认值：5

hll_default_expthresh（废弃）

参数说明：该参数可以用来设置从Explicit模式到Sparse模式的默认阈值大小。当前已经使用参数hll_default_log2explicit替代类似功能。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，-1~7。-1表示自动模式，0表示跳过Explicit模式，取1-7表示在基数到达$2^{hll_default_expthresh}$时切换模式。

默认值：-1

hll_default_sparseon（废弃）

参数说明：该参数可用来指定是否默认开启Sparse模式。当前已经使用参数hll_default_log2sparse替代类似功能，hll_default_log2sparse设置为0时关闭Sparse模式。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：0，1。0表示默认关闭，1表示默认开启。

默认值：1

hll_max_sparse（废弃）

参数说明：该参数可以用来指定max_sparse的大小。当前已经使用参数hll_default_log2sparse替代类似功能。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，-1~2147483647

默认值：-1

enable_compress_hll（废弃）

参数说明：该参数可以用来指定是否对hll开启内存优化模式。目前hll内存已经进行了优化设计，该参数不再使用。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on/true表示对hll开启内存优化模式。
- off/false表示不开启内存优化模式。

默认值：off

18.3.32 用户自定义函数

udf_memory_limit

参数说明：控制每个CN、DN执行UDF时可用的最大物理内存量。本参数当前版本不生效，请使用FencedUDFMemoryLimit和UDFWorkerMemHardLimit参数控制fenced udf worker虚存。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：整型，200*1024～2147483647，单位为KB。

默认值：200MB

FencedUDFMemoryLimit

参数说明：控制每个fenced udf worker进程使用的虚拟内存。

该参数属于USERSET类型参数，请参考**表18-1**中对应设置方法进行设置。

设置建议：详细内容请联系管理员处理。

取值范围：整数，0 ~ 2147483647，单位为KB，设置可带单位（KB，MB，GB）。其中0表示不做内存控制。

默认值：0

UDFWorkerMemHardLimit

参数说明：控制fencedUDFMemoryLimit的最大值。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

设置建议：详细内容请联系管理员处理。

取值范围：整数，0 ~ 2147483647，单位为KB，设置时可带单位（KB，MB，GB）。

默认值：1GB

18.3.33 定时任务

job_queue_processes

参数说明：表示系统可以并发执行的job数目。该参数为postmaster级别，通过gs_guc设置，需要重启gaussdb才能生效。

该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：0～1000

功能：

- 当job_queue_processes设置为0值，表示不启用定时任务功能，任何job都不会被执行（因为开启定时任务的功能会对系统的性能有影响，有些局点可能不需要定时任务的功能，可以通过设置为0不启用定时任务功能）。

- 当job_queue_processes为大于0时，表示启用定时任务功能且系统能够并发处理的最大任务数。

启用定时任务功能后，job_scheduler线程会在定时时间间隔轮询pg_job系统表，系统设置定时任务检查周期默认为1s。

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

注：如果同一时间内并行的job数很多，过小的参数值会导致job等待。而过大的参数值则消耗更多的系统资源，建议设置此参数为100，用户可以根据系统资源情况合理调整。

默认值：10

enable_prevent_job_task_startup

参数说明：控制是否启动job线程。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示不能启动job线程。
- off表示可以启动job线程。

默认值：off

18.3.34 线程池

enable_thread_pool

参数说明：控制是否使用线程池功能。该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启线程池功能。
- off表示不开启线程池功能。

默认值：on

thread_pool_attr

参数说明：用于控制线程池功能的详细属性，该参数仅在enable_thread_pool打开后生效，仅sysadmin用户可以访问。该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串，长度大于0

该参数分为3个部分，'thread_num, group_num, cpubind_info'，这3个部分的具体含义如下：

- thread_num：线程池中的初始线程总数，可以动态扩充，取值范围是0~4096。其中0的含义是数据库根据系统CPU core的数量来自动配置线程池的线程数，如果参数值大于0，线程池中的线程数等于thread_num。线程池大小推荐根据硬件配置设置，计算公式如下：thread_num = CPU核数*3~5，thread_num最大值为4096。
- group_num：线程池中的线程分组个数，取值范围是0~64。其中0的含义是数据库根据系统NUMA组的个数来自动配置线程池的线程分组个数，如果参数值大于0，线程池中的线程组个数等于group_num。
- cpubind_info：线程池是否绑核的配置参数。可选择的配置方式有：1. '(nobind)'，线程不做绑核；2. '(allbind)'，利用当前系统所有能查询到的CPU core做线程绑核；3. '(nodebind: 1, 2)'，利用NUMA组1,2中的CPU core进行绑核；4. '(cpubind: 0-30)'，利用0-30号CPU core进行绑核；5. '(numabind: 0-30)'，在NUMA组内利用0-30号CPU core进行绑核。该参数不区分大小写。

默认值：

- 独立部署：'1024,2,(nobind)'（60核CPU/480G内存，32核CPU/256G内存）；'512,2,(nobind)'（16核CPU/128G内存）；'256,2,(nobind)'（8核CPU/64G内存）；'128,2,(nobind)'（4核CPU/32G内存）；'64,2,(nobind)'（4核CPU/16G内存）
- 金融版（标准型）：

  CN：'768,2,(nobind)'（128核CPU/1024G内存，104核CPU/1024G内存）；'684,2,(nobind)'（96核CPU/1024G内存，96核CPU/768G内存）；'512,2,(nobind)'（80核CPU/640G内存，72核CPU/576G内存，64核CPU/512G内存，60核CPU/480G内存）；'256,2,(nobind)'（32核CPU/256G内存），'128,2,(nobind)'（16核CPU/128G内存）；'64,2,(nobind)'（8核CPU/64G内存）

  DN：'4096,2,(nobind)'（128核CPU/1024G内存，104核CPU/1024G内存，96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，72核CPU/576G内存，64核CPU/512G内存，60核CPU/480G内存）；'2048,2,(nobind)'（32核CPU/256G内存），'1024,2,(nobind)'（16核CPU/128G内存）；'512,2,(nobind)'（8核CPU/64G内存）
- 企业版：

  CN：'768,2,(nobind)'（128核CPU/1024G内存，104核CPU/1024G内存）；'512,2,(nobind)'（96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，80核CPU/512G内存，72核CPU/576G内存，64核CPU/512G内存，60核CPU/480G内存）；'256,2,(nobind)'（32核CPU/256G内存）；'128,2,(nobind)'（16核CPU/128G内存）；'64,2,(nobind)'（8核CPU/64G内存）

  DN：'1536,2,(nobind)'（128核CPU/1024G内存，104核CPU/1024G内存）；'1024,2,(nobind)'（96核CPU/1024G内存，96核CPU/768G内存，80核CPU/640G内存，80核CPU/512G内存，72核CPU/576G内存，64核CPU/512G内存，60核CPU/480G内存）；'512,2,(nobind)'（32核CPU/256G内存）；'256,2,(nobind)'（16核CPU/128G内存）；'128,2,(nobind)'（8核CPU/64G内存）
- 金融版（数据计算型）：

  CN：2GB（96核CPU/768G内存）；'256,2,(nobind)'（72核CPU/576G内存）；'128,2,(nobind)'64核CPU/512G内存）；'64,2,(nobind)'（32核CPU/256G内存）

  DN：'2048,2,(nobind)'（96核CPU/768G内存）；'1024,2,(nobind)'（72核CPU/576G内存）；'512,2,(nobind)'（64核CPU/512G内存）；'256,2,(nobind)'（32核CPU/256G内存）

thread_pool_stream_attr

参数说明：用于控制stream线程池功能的详细属性，stream线程只在DN生效，该参数仅在enable_thread_pool打开后生效，仅sysadmin用户可以访问。该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串，长度大于0

该参数分为4个部分，'stream_thread_num, stream_proc_ratio ,group_num ,cpubind_info'，这4个部分的具体含义如下：

- stream_thread_num：stream线程池中的线程总数，取值范围是0~4096。其中0的含义是数据库根据系统CPU core的数量来自动配置线程池的线程数，如果参数值大于0，线程池中的线程数等于stream_thread_num。线程池大小推荐根据硬件配置设置，计算公式如下：stream_thread_num = CPU核数*3~5，stream_thread_num最大值为4096。
- stream_proc_ratio：预留给stream线程的proc数量比例，浮点类型，默认为0.2，预留proc计算方式为：stream_proc_ratio * stream_thread_num。
- group_num：线程池中的线程分组个数，取值范围是0~64。其中0的含义是数据库根据系统NUMA组的个数来自动配置线程池的线程分组个数，如果参数值大于0，线程池中的线程组个数等于group_num。thread_pool_stream_attr的group_num需与thread_pool_attr的group_num配置和使用保持一致，若设置为不同值，以thread_pool_attr的group_num为准。
- cpubind_info：线程池是否绑核的配置参数。可选择的配置方式有：1. '(nobind)' ，线程不做绑核；2. '(allbind)'，利用当前系统所有能查询到的CPU core做线程绑核；3. '(nodebind: 1, 2)'，利用NUMA组1,2中的CPU core进行绑核；4. '(cpubind: 0-30)'，利用0-30号CPU core进行绑核；5. '(numabind: 0-30)'，在NUMA组内利用0-30号CPU core进行绑核。该参数不区分大小写。thread_pool_stream_attr的cpubind_info需与thread_pool_attr的cpubind_info配置和使用保持一致，若设置为不同值，以thread_pool_attr的cpubind_info为准。

默认值：

stream_thread_num：16

stream_proc_ratio：0.2

group_num、cpubind_info：参见**thread_pool_attr**。

resilience_threadpool_reject_cond

参数说明：用于控制线程池过载逃生的堆积会话数占比。该参数仅在GUC参数enable_thread_pool和use_workload_manager打开时生效。该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：字符串，长度大于0

该参数分为recover_threadpool_percent、overload_threadpool_percent 2部分，这2个部分的具体含义如下：

- recover_threadpool_percent：线程池恢复正常状态的接入会话占线程池初始设置线程数的百分比，当已经接入的会话数小于线程池初始设置数乘以该值对应的百分比后，停止过载逃生并放开新连接接入，取值为0~INT_MAX，设置为多少表示百分之多少。
- overload_threadpool_percent：线程池过载时的接入会话占线程池初始设置线程数的百分比，当已经接入的会话数大于线程池初始设置数乘以该值对应的百分比

后，表示当前线程池已经过载，触发过载逃生kill会话并禁止新连接接入，取值为0~INT_MAX，设置为多少表示百分之多少。

默认值：'0,0'，表示关闭线程池逃生功能。

示例：

```
resilience_threadpool_reject_cond = '100,200'
```

表示已经堆积的会话数超过线程池初始设置的线程数的200%后禁止新连接接入并kill堆积的会话，kill会话过程中会话数恢复到线程池初始设置的线程数的100%以下时停止kill会话并允许新连接接入。

须知

- 已经堆积的会话数可以通过查询pg_stat_activity视图有多少条数据获得，需要过滤少量后台线程；线程池设置的初试线程池线程数目可以通过查询thread_pool_attr参数获得。
- 该参数如果设置的百分比过小，则会频繁触发线程池过载逃生流程，会使正在执行的会话被强制退出，新连接短时间接入失败，需要根据实际线程池使用情况慎重设置。
- use_workload_manager参数关闭的情况下，如果打开bypass_workload_manager，则该参数也会生效，但是因为bypass_workload_manager是SIGHUP类型，reload方式设置后需要重启数据库才会使得当前功能生效。
- recover_threadpool_percent和overload_threadpool_percent的值可以同时为0，除此之外，recover_threadpool_percent的值必须要小于overload_threadpool_percent，否则会设置不生效。

18.3.35 备份恢复

operation_mode

参数说明：系统进入备份恢复模式。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示在备份恢复过程中。
- off表示不在备份恢复过程中。

默认值：off

enable_cbm_tracking

参数说明：当使用roach执行集群的全量和增量备份时需要开启此参数，如果关闭会导致备份失败。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示追踪功能开启。
- off表示追踪功能关闭。

默认值：off

max_size_for_xlog_retention

参数说明：用于控制何时触发备份复制槽的强制推进，以避免由于备份操作过程中日志无法回收导致磁盘满、集群只读等影响。该参数实际设置值建议比cm_server组件的datastorage_threshold_value_check略小一点，以避免集群进入只读状态。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：-100 ~ 2147483647

- 0表示关闭该功能。
- 负值表示按磁盘阈值触发，如-80，表示当磁盘阈值超过80%，且日志回收是由于备份操作被阻塞，那么会触发备份复制槽的强制推进。
- 正值表示按日志积压大小触发，如32，表示当备份复制槽落后当前检查点redo位置超过32段日志大小（每段日志大小为16MB），且日志回收是由于备份操作被阻塞，那么会触发备份复制槽的强制推进。

默认值：0

18.3.36 AI 特性

enable_hypo_index

参数说明：该参数控制数据库的优化器进行EXPLAIN时是否考虑创建的虚拟索引。通过对特定的查询语句执行explain，用户可根据优化器给出的执行计划评估该索引是否能够提升该查询语句的执行效率。

该参数属于USERSET类型参数，请参考**表18-2**中对应设置方法进行设置。

取值范围：布尔型

- on表示在进行EXPLAIN时创建虚拟索引。
- off表示在进行EXPLAIN时不创建虚拟索引。

默认值：off

18.3.37 Global SysCache 参数

enable_global_syscache

参数说明：控制是否使用全局系统缓存功能。该参数属于POSTMASTER类型参数，请参考**表18-1**中对应设置方法进行设置。

取值范围：布尔型

- on表示开启全局系统缓存功能。
- off表示不开启全局系统缓存功能。

默认值：on

推荐结合线程池参数使用。打开该参数后，如果需要访问备机，建议设置备机wal_level级别为hot_standby以上。

global_syscache_threshold

参数说明：全局系统缓存内存最大占用大小。

该参数属于SIGHUP类型参数，请参考**表18-1**中对应设置方法进行设置。

需要打开enable_global_syscache参数。

取值范围：整型，16384 ~ 1073741824，单位为kB。

默认值：163840

推荐计算公式：热点DB个数和线程个数的最小值乘以每个DB分配的内存大小，即global_syscache_threshold = min(count(hot dbs)，count(threads)) * memofdb。

热点DB数即访问较为频繁的数据库，线程数在线程池模式下取线程池线程个数和后台线程个数之和，非线程池模式不需要计算这个值，直接使用热点DB数。

memofdb即平均每个db应该分配的内存，每个DB的底噪内存是2M，平均每增加一个表或者索引，增加11k内存。

如果设置的值过小，会导致内存频繁淘汰，内存存在大量碎片无法回收，导致内存控制失效。

18.3.38 预留参数

说明

下列参数为预留参数，该版本不生效。

acce_min_datasize_per_thread

cstore_insert_mode

dfs_partition_directory_length

enable_fstream

enable_hdfs_predicate_pushdown

enable_orc_cache

schedule_splits_threshold

enable_constraint_optimization

enable_hadoop_env

enable_hypo_index

undo_space_limit_size

undo_limit_size_per_transaction

undo_zone_count

ustore_attr

enable_ustore

walwriter_cpu_bind

废弃函数

enable_adio_debug

enable_adio_function

enable_fast_allocate

prefetch_quantity

backwrite_quantity

cstore_prefetch_quantity

cstore_backwrite_quantity

cstore_backwrite_max_threshold

fast_extend_file_size

effective_io_concurrency

