数据仓库服务（DWS）

用户开发指南

天翼云科技有限公司

目　录

1 欢迎

1.1 文档面向的读者对象

数据库开发指南重点面向数据库的设计者、应用程序开发人员或 DBA，提供设计、构建、查询和维护数据仓库所需的信息。

作为数据库管理员和应用程序开发人员，至少需要了解以下知识：

- 操作系统知识。这是一切的基础。
- SQL 语法。这是操作数据库的必备能力。

声明

GaussDB(DWS)的作者们在进行文档写作时努力基于商用角度，从使用场景和任务完成角度给出内容指引。即使这样，文档中依然可能存在对 Postgres 内容的引用和参考。对于这类内容，遵从如下的 Postgres Copyright：

1.2 阅读指引

对于首次接触 GaussDB(DWS)的用户，建议先阅读以下部分：

- 介绍 GaussDB(DWS)服务的特点、功能和适用场景。
- GaussDB(DWS)入门包含一个示例，引导您完成创建数据仓库集群、创建数据库表、上传数据和测试查询这一过程。

如果打算或正在将应用程序从其他数据仓库向 GaussDB(DWS)迁移，您可能想知道 GaussDB(DWS)在实施方式上有什么区别。

GaussDB(DWS)进行数据库应用程序开发过程中，下表将帮您找到对应的信息。

如果要..	查阅建议
快速开始使用 GaussDB(DWS)。	首先，按照《数据仓库服务用户指南》的“入门”中的步骤快速部署集群、连接到数据库并尝试进行一些查询。 准备好构建数据库后，将数据加载到表中并编写查询内容以操作数据仓库中的数据后，可以回到《数据仓库服务数据库开发指南》。
了解 GaussDB(DWS)数据仓库的内部架构。	如果您想要更全面地了解 GaussDB(DWS)服务，请转到 GaussDB(DWS)产品首页。
了解如何设计表以实现良好性能。	7 开发设计建议介绍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(DWS)的分布式处理架构，输出更高效的业务 SQL 代码。 对业务的执行效率不满意，期望通过调优加快业务执行的情况下，可以参考 11 优化查询性能进行调优。性能调优是一项复杂的工程，有些时候无法系统性地说明和解释，而是依赖于 DBA 的经验判断。尽管如此，11 优化查询性能一节还是期望能尽量系统性的对性能调优方法加以说明，方便应用开发人员和刚接触 GaussDB(DWS)的 DBA 参考。
加载数据。	3.2 导入数据介绍数据入库 GaussDB(DWS)的方法和途径。
管理用户、组和数据库安全。	6 管理数据库安全涵盖数据库安全主题。
监控和优化系统性能。	14 系统表和系统视图详细介绍您可以从中查询数据库状态并监控查询内容与流程的系统表和视图。 您还应该查阅《数据仓库服务用户指南》中的“监控集群”了解如何使用 GaussDB(DWS)管理控制台检查系统运行状况、监控指标。

1.3 文档表达约定

举例约定

内容	说明
dbadmin	表示创建集群时指定的运行和维护 GaussDB(DWS)的用户。
8000	表示 GaussDB(DWS)监听客户端连接请求的端口号。

手册中的 SQL 示例是基于 TPC-DS 模型开发的，如果需要运行手册中的示例，请先参考官网说明（http://www.tpc.org/tpcds/），安装 TPC-DS benchmark。

SQL 语法文本格式约定

为了方便对语法使用的理解，在文档中对 SQL 语法文本按如下格式进行表述。

格式	意义
大写	语法关键字（语句中保持不变、必须照输的部分）采用大写表示。
小写	参数（语句中必须由实际值进行替代的部分）采用小写表示。
[ ]	表示用“[ ]”括起来的部分是可选的。
...	表示前面的元素可重复出现。
[x | y | ...]	表示从两个或多个选项中选取一个或者不选。
{ x | y | ... }	表示从两个或多个选项中选取一个。
[x | y | ...] [...]	表示可选多个参数或者不选，如果选择多个参数，则参数之间用空格分隔。
[x | y | ...] [,...]	表示可选多个参数或者不选，如果选择多个参数，则参数之间用逗号分隔。
{ x | y | ... } [...]	表示可选多个参数，至少选一个，如果选择多个参数，则参数之间以空格分隔。
{ x | y | ... } [,...]	表示可选多个参数，至少选一个，如果选择多个参数，则参数之间用逗号分隔。

1.4 前置条件

使用本指南前，需要完成以下任务。

- 创建 GaussDB(DWS)集群。
- 安装 SQL 客户端。
- 将 SQL 客户端连接到集群的默认数据库。

关于上述任务的详细指导，请参见《数据仓库服务用户指南》的“入门”。

2 系统概述

2.1 高可靠事务处理

GaussDB(DWS)提供集群事务管理功能，此功能是集群 HA、集群故障切换的基础，负责保证集群所有节点间事务的 ACID 特性，保证故障可恢复，以及恢复后满足数据的 ACID（Atomicity，Consistency，Isolation，Durability）特性，并负责节点的并发控制。

故障恢复

为了在集群出现故障时尽可能地不中断服务，GaussDB(DWS)提供了高可靠机制。通过保护关键用户程序对外不间断提供服务，把因为硬件、软件和人为造成的故障对业务的影响程度降到最低，以保证业务的持续性。

- 硬件级高可靠：磁盘 Raid、交换机堆叠及网卡 bond、不间断电源 UPS（Uninterruptible Power Supply）。
- 软件级高可靠：GaussDB(DWS)集群 CN、GTM、DN 等全方位 HA。

事务管理

- 支持事务块，用户可以通过 start transaction 命令显式启动一个事务块。
- 支持单语句事务，用户不显式启动事务，则单条语句就是一个事务。
- 分布式事务管理。支持全局事务信息管理，包括 gxid、snapshot、timestamp 的管理，分布式事务状态管理，gxid 溢出的处理。
- 分布式事务支持 ACID 特性。
- 支持分布式死锁预防，保证在出现死锁时自动解锁或者预防死锁。

2.2 查询高性能

GaussDB(DWS)通过如下功能来努力实现查询的高性能。

全并行的数据查询处理

GaussDB(DWS)是采用 Shared-nothing 架构的 MPP 系统，其由众多拥有独立且互不共享 CPU、内存、存储等系统资源的逻辑节点组成。在这样的系统架构中，业务数据被

分散存储在多个节点上，数据分析任务被推送到数据所在位置就近执行，并行地完成大规模的数据处理工作，实现对数据处理的快速响应。

GaussDB(DWS)后台还通过算子并行执行、指令在寄存器并行执行、及 LLVM 动态编译剪枝冗余的条件逻辑判断，助力数据查询性能提升。

行列混合存储

GaussDB(DWS)支持行存储和列存储两种存储模型，用户可以根据应用场景，建表的时候选择行存储还是列存储表。

行列混合存储引擎可以同时为用户提供更优的数据压缩比（列存）、更好的索引性能（列存）、更好的点更新和点查询（行存）性能。

列存下的数据压缩

对于非活跃的早期数据可以通过压缩来减少空间占用，降低采购和运维成本。

GaussDB(DWS)列存储压缩支持 Delta Value Encoding、Dictionary、RLE 、LZ4、ZLIB 等压缩算法，且能够根据数据特征自适应的选择压缩算法，平均压缩比 7:1。压缩数据可直接访问，对业务透明，极大缩短历史数据访问的准备时间。

2.3 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。创建数据库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。

实例

实例在 GaussDB(DWS)中是运行在内存中的一组数据库进程，一个实例可以管理一个或多个数据库，这些数据库组成一个集簇。集簇是存储磁盘上的一个区域，这个区域在安装时初始化并由一个目录组成，所有数据都存储在这个目录中，这个目录被称为数据目录，使用 initdb 创建。理论上来说一个服务器上可以在不同的端口启动多个实例，但是 GaussDB(DWS)一次只能管理一个实例，启动和停止都是依赖于具体的数据目录。以后由于兼容的需要不排除引入实例名这个概念的可能。

表空间

在 GaussDB(DWS)中，表空间是一个目录，可以存在多个，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(DWS)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(DWS)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(DWS)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(DWS)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(DWS)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(DWS)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(DWS)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(DWS)没有回滚段的概念，但是为了定期清除历史版本数据 GaussDB(DWS)引入了一个 VACUUM 进程。一般情况下用户不用关注它，除非要做性能调优。此外，GaussDB(DWS)是自动提交事务。

3 数据迁移

3.1 迁移数据到 GaussDB(DWS)

GaussDB(DWS)提供了灵活的数据入库方式，可以将多种数据源的数据导入到GaussDB(DWS)中。各导入方式具有不同的特点，如表 3-1 所示，用户可以根据其特点自行选择。建议用户配合云数据迁移（Cloud Data Migration，简称 CDM）和数据湖工厂（Data Lake Factory，简称 DLF）一起使用，CDM 用于批量数据迁移，DIS 用于流数据接入，DLF 可以对整个 ETL 过程进行编排调度，同时提供可视化的开发环境。

说明

- DRS、CDM、OBS、MRS 为云服务。
- GDS、DSC、gs_restore、gs_dump 为内部工具。

表3-1 数据导入方式说明

数据导入方式	数据源	说明	优势
3.2.1 从 OBS 并行导入数据	OBS	支持将存储在 OBS 上的 TXT、CSV、ORC 及 CARBONDATA 格式的数据并行导入到 GaussDB(DWS)，支持导入后查询数据，也支持远程读 OBS 上的数据。 GaussDB(DWS)优先推荐的导入方式。	并行拉取方式，性能好，横向扩展。
3.2.2 使用 GDS 从远端服务器导入数据	Servers（即远端服务器）	使用 GaussDB(DWS)提供的 GDS 工具，利用多 DN 并行的方式，将数据从远端服务器导入到 GaussDB(DWS)。这种方式导入效率高，适用于大批量数据入库。	
3.2.3 从 MRS 导入数据到集群	MRS（HDFS）	配置一个 GaussDB(DWS)集群连接到一个 MRS 集群，然后将数据从 MRS 的 HDFS 中读	并行拉取方式，性能好，

数据导入方式	数据源	说明	优势
		取到 GaussDB(DWS)。	横向扩展。
3.2.4 从 GaussDB(DWS)集群导入数据到新集群	-	支持两个 GaussDB(DWS)集群之间的数据互访互通。通过 Foreign Table 方式实现跨 DWS 集群的数据访问和导入。	适用于多套 DWS 集群之间的数据同步。
3.2.5 基于 GDS 的跨集群互联互通	-	通过 GDS 进行数据中转，实现多个集群之间的数据同步。	适用于多套 DWS 集群之间的数据同步。
3.2.6 使用 gsql 元命令\COPY 导入数据	本地文件	与直接使用 SQL 语句 COPY 不同，该命令读取/写入的文件只能是 gsql 客户端所在机器上的本地文件。	操作简单，适用于小批量数据入库。
3.2.7 使用 COPY FROM STDIN 导入数据	其他文件或数据库	使用 Java 语言开发应用程序时，通过调用 JDBC 驱动的 CopyManager 接口，从文件或其他数据库向 GaussDB(DWS)写入数据。	从其他数据库直接写入 GaussDB(DWS)的方式，具有业务数据无需落地成文件的优势。
3.3.1 使用 CDM 迁移数据到 GaussDB(DWS)	数据库、NoSQL、文件系统、大数据平台	CDM 提供同构/异构数据源之间批量数据迁移的功能，帮助您实现从多种类型的数据源迁移数据到 GaussDB(DWS)。CDM 在迁移数据到 GaussDB(DWS)时，采用的是 Copy 方式和 GDS 并行导入方式。	数据源丰富，操作简单。
3.3.2 使用 DSC 工具迁移 SQL 脚本	数据库、NoSQL、文件系统、大数据平台	请参考第三方 ETL 工具的相关文档。 GaussDB(DWS)提供了 DSC 工具，可以将 Teradata/Oracle 脚本迁移到 GaussDB(DWS)。	通过 OBS 中转，数据源丰富，数据转换能力强。
3.4.1 使用 gs_dump 和 gs_dumpall 命令导出元数据	• 纯文本格式 • 自定义归档格式 • 目录归档格式 • tar 归档格式	gs_dump 支持导出单个数据库或其内的对象，而 gs_dumpall 支持导出集群中所有数据库或各库的公共全局对象。 通过导入工具将导出的元数据信息导入至需要的数据库，可以完成数据库信息的迁移。	适用于元数据迁移。

数据导入方式	数据源	说明	优势
3.4.2 使用 gs_restore 导入数据	sql/tmp/tar 文件格式	在数据库迁移场景下，支持使用 gs_restore 工具将事先使用 gs_dump 工具导出的文件格式，重新导入 GaussDB(DWS) 集群，实现表定义、数据库对象定义等元数据的导入。导入数据主要包括以下内容： • 所有数据库对象定义。 • 单个数据库对象定义。 • 单个 schema 定义。 • 单张表定义。	

3.2 导入数据

3.2.1 从 OBS 并行导入数据

3.2.1.1 关于 OBS 并行导入

对象存储服务 OBS（Object Storage Service）是云上提供的一个基于对象的海量存储服务，为客户提供安全、高可靠、低成本的数据存储能力。OBS 为用户提供了超大存储容量的能力，适合存放任意类型的文件。

数据仓库服务 GaussDB(DWS)使用 OBS 作为集群数据与外部数据互相转化的平台，实现安全、高可靠和低成本的存储需求。

GaussDB(DWS)支持将 OBS 上 TXT、CSV、ORC、CARBONDATA 以及 JSON 格式的数据导入到集群进行查询，也支持远程读 OBS 上的数据。因此对于经常查询的热数据建议直接导入 GaussDB(DWS)后再做查询。偶尔查询的冷数据可以存储在 OBS 上直接远程读以节省成本。

目前，导入数据有两种方式：

- 方式一：无需用户创建 server，使用默认 server 创建外表，支持 TXT、CSV 格式的数据，参见 3.2.1.2 从 OBS 导入 CSV,TXT 数据。
- 方式二：需用户创建 server，使用该 server 创建外表，支持 ORC、CARBONDATA、TXT、CSV 以及 JSON 格式的数据，参见 3.2.1.3 从 OBS 导入 ORC，CARBONDATA 数据。

须知

- OBS 导入导出数据时，不支持中文路径。
- OBS 导入导出数据时，暂不支持跨 Region 进行 OBS 数据导入导出，必须确保 OBS 和 DWS 集群在同一个 Region 中。

概述

在数据迁移、ETL（Extract-Transform-Load）过程中，需要向 GaussDB(DWS)并行导入海量数据，使用普通方式会耗费大量的时间。GaussDB(DWS)提供了 OBS（Object Storage Service）及外表接口，通过 OBS 外表设置的导入 URL 路径、导入数据格式等信息来识别数据源文件，利用多 DN（Datanode）并行的方式，实现了数据的快速并行导入。

优势：

- CN 只负责任务的规划及下发，把数据导入的工作交给了 DN，释放了 CN 的资源，使其有能力处理外部请求。
- 通过让各个 DN 都参与数据导入，充分利用各个设备的计算能力及网络带宽。
- 支持导入过程中对数据做预处理。
- 支持在导入过程中，针对数据格式错误设置导入容错性，并可在导入结束后根据错误信息定位错误数据。

劣势：

需要创建 OBS 外表，并且要在 OBS 服务器上存放导入数据。

适用场景：

高并发、大数据量导入。

相关概念

- **数据源文件**：存储有数据的 TEXT、CSV、ORC、CARBONDATA、JSON 文件。文件中保存的是待并行导入数据库的数据。
- **OBS**：对象存储服务，是一种可存储文档、图片、影音视频等非结构化数据的云存储服务。向 GaussDB(DWS)并行导入数据时，数据对象放置在 OBS 服务器上。
- **桶（Bucket）**：对 OBS 中的一个存储空间的形象称呼，是存储对象的容器。
  - 对象存储是一种非常扁平化的存储方式，桶中存储的对象都在同一个逻辑层级，去除了文件系统中的多层级树形目录结构。
  - 在 OBS 中，桶名必须是全局唯一的且不能修改，即用户创建的桶不能与自己已创建的其他桶名称相同，也不能与其他用户创建的桶名称相同。每个桶在创建时都会生成默认的桶 ACL（Access Control List），桶 ACL 列表的每项包含了对被授权用户授予什么样的权限，如读取权限、写入权限、完全控制权限等。用户只有对桶有相应的权限，才可以对桶进行操作，如创建、删除、显示、设置桶 ACL 等。

- 一个用户最多可创建 100 个桶，但每个桶中存放的总数据容量和对象/文件数量没有限制。

- **对象**：是存储在 OBS 中的基本数据单位。用户上传的数据以对象的形式存储在 OBS 的桶中。对象的属性包括名称 Key，Metadata，Data。

  通常，我们将对象等同于文件来进行管理，但是由于 OBS 是一种对象存储服务，并没有文件系统中的文件和文件夹概念。为了使用户更方便进行管理数据，OBS 提供了一种方式模拟文件夹。通过在对象的名称中增加“/”，如 tpcds1000/stock.csv，tpcds1000 可以等同于文件夹，stock.csv 就可以等同于文件名，而对象名称（key）仍然是 tpcds1000/stock.csv、对象的内容就是 stock.csv 数据文件的内容。

- **Key**：对象的名称（键），为经过 UTF-8 编码的长度大于 0 且不超过 1024 的字符序列，一个桶里的每个对象必须拥有唯一的对象键值。用户可使用桶名+对象名来存储和获取对应的对象。

- **Metadata**：对象元数据，用来描述对象的信息。元数据又可分为系统元数据和用户元数据。这些元数据以键值对（Key-value）的形式随 http 头域一起上传到 OBS 系统。

  - 系统元数据由 OBS 系统产生，在处理对象数据时使用。系统元数据包括：Date, Content-length, last-modify, Content-MD5 等。
  - 用户元数据由用户上传对象时指定，是用户自己对对象的一些描述信息。

- **Data**：对象的数据内容，OBS 对于数据的内容是无感知的，即认为对象内的数据为无状态的二进制数据。

- **数据库普通表**：数据库中的普通表，数据源文件中的数据最终并行导入到这些表中存储，包括行存表、列存表。

- **外表**：用于识别数据源文件中的数据。外表中保存了数据源文件的位置、文件格式、编码格式、数据间的分隔符等信息。

导入数据原理

OBS 导入原理如图 3-1 所示，CN 负责任务的规划及下发，它是按文件给每个 DN 节点分配任务的。

分配算法如下：

例如，图 3-1 中，总共有 4 个节点 DN0~DN3， OBS 上有 6 个文件 t1.data.0~t1.data.5，那么分配方式如下：

t1.data.0 -> DN0

t1.data.1 -> DN1

t1.data.2 -> DN2

t1.data.3 -> DN3

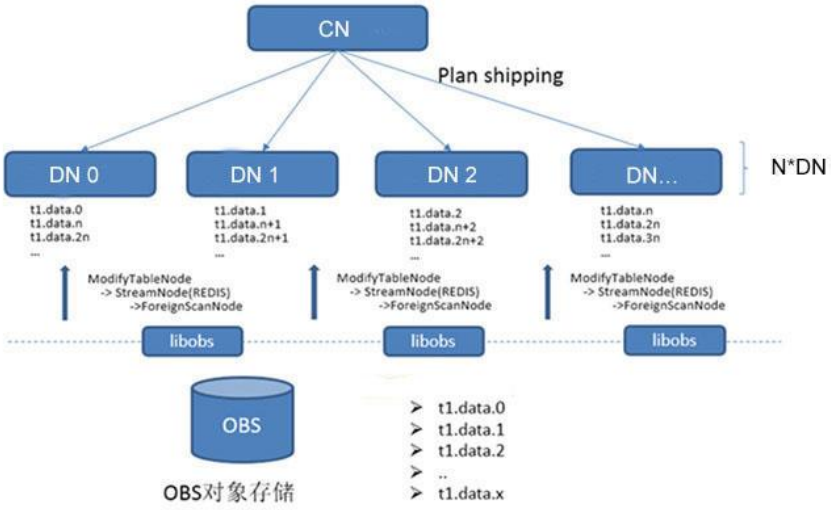
t1.data.4 -> DN0

t1.data.5 -> DN1

其中 DN0 和 DN1 上分配了两个文件，其他 DN 分配了 1 个文件。

如果 OBS 上文件大小都相同时，OBS 上的文件数与 DN 节点数的比例为 1:1 时导入性能为最好，因为每个 DN 分配的任务都相同。因此建议将数据文件存储到 OBS 前，尽可能均匀地将文件切分成多个，文件的数量以 DN 的整数倍更适合。

图3-1 通过 OBS 外表并行导入数据



导入流程图

图3-2 并行导入流程

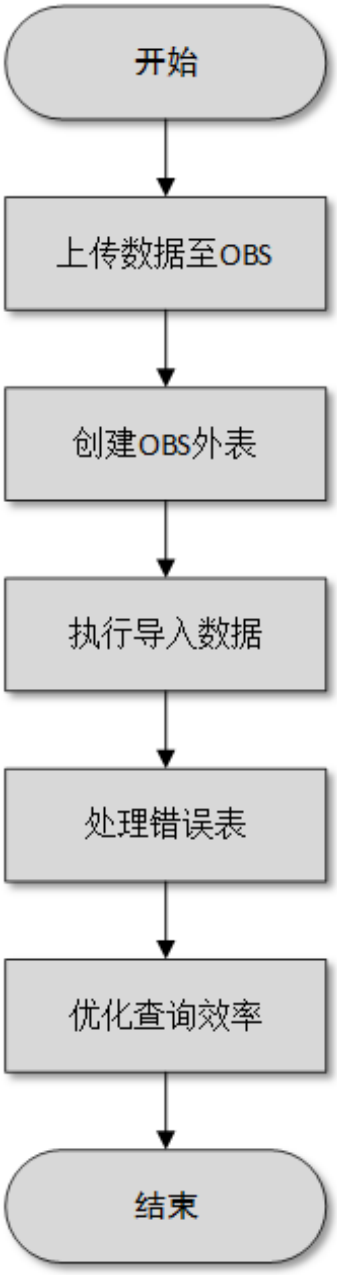


表3-2 流程说明

流程	说明	子任务
上传数据至OBS。	在 OBS 服务器上规划存储路径，并上传数据文件。 详细请参见 3.2.1.2.2 上传数据	-

流程	说明	子任务
	到 OBS。	
创建 OBS 外表。	创建外表用于识别 OBS 服务器上的数据源文件。在 OBS 外表中保存了数据源在 OBS 服务器上的桶名、对象名，文件格式、存放位置、编码格式、数据间的分隔符等信息。 详细请参见 3.2.1.2.3 创建 OBS 外表。	-
执行导入数据。	在创建好外表后，通过 INSERT 语句，将数据快速、高效地导入到目标表中。 详细请参见 3.2.1.2.4 执行导入数据。	-
处理错误表。	在数据并行导入发生错误时，请根据错误信息，3.2.1.2.5 处理错误表，以保证导入数据的完整性。	-
优化查询效率。	导入数据后，通过 ANALYZE 语句生成表统计信息。ANALYZE 语句会将统计结果自动存储在系统表 PG_STATISTIC 中。执行计划生成器会使用这些统计数据，以生成最有效的查询执行计划。	-

3.2.1.2 从 OBS 导入 CSV,TXT 数据

3.2.1.2.1 创建访问密钥（AK 和 SK）

在本示例中，将导入 OBS 数据到 GaussDB(DWS)集群数据库中。云平台用户通过客户端或 API、SDK 等方式访问 OBS 时，需要通过 AK/SK 认证方式进行认证鉴权。因此，当您需要通过客户端或 JDBC/ODBC 应用程序等方式连接 GaussDB(DWS)数据库访问 OBS 时，必须先获取访问密钥（AK 和 SK）。

- Access Key Id（AK）：访问密钥 ID。与私有访问密钥关联的唯一标识符；访问密钥 ID 和私有访问密钥一起使用，对请求进行加密签名。
- Secret Access Key（SK）：与访问密钥 ID 结合使用的密钥，对请求进行加密签名，可标识发送方，并防止请求被修改。

创建访问密钥（AK 和 SK）

在创建访问密钥前，请确保登录控制台的帐号已通过实名认证。

通过管理控制台创建访问密钥（AK 和 SK）操作步骤如下：

步骤 1 登录 GaussDB(DWS) 管理控制台。

步骤 2 单击右上角用户名，在下拉菜单中单击“我的凭证”。

步骤 3 选择“管理访问密钥”页签。

如果在访问密钥列表中已经有访问密钥了，可以直接使用现有的访问密钥。但是，在访问密钥列表中只能查看到访问密钥 ID（即 Access Key ID），只有在新增访问密钥时，用户才可以下载到含有 Access Key ID 和 Secret Access Key 的密钥文件。如果您没有该密钥文件，可以单击“新增访问密钥”重新创建。

说明

- 每个用户最多可创建两个有效的访问密钥，如果当前已存在 2 个访问密钥，只能先删除现有的访问密钥，然后再重新创建。删除时，需要输入当前用户的登录密码、邮箱或手机短信的验证码，验证通过才能成功删除。
- 为了账号安全性，建议您定期更换并妥善保存访问密钥。

步骤 4 单击“新增访问密钥”。

步骤 5 在弹出的对话框中，输入登录密码和对应验证码，然后单击“确定”。

说明

- 用户如果未绑定邮箱和手机，则只需输入登录密码。
- 用户如果同时绑定了邮箱和手机，可以选择其中一种方式进行验证。

步骤 6 在弹出的“下载确认”提示框中，单击“确定”后，密钥会直接保存到浏览器默认的下载文件夹中。

说明

- 为防止访问密钥泄露，建议您将其保存到安全的位置。
- 如果用户在此提示框中单击“取消”，则不会下载密钥，后续也将无法重新下载。如果需要使用访问密钥，可以重新创建新的访问密钥。

步骤 7 打开下载下来的“credentials.csv”文件即可获取到访问密钥（AK 和 SK）。

----结束

注意事项

当用户发现访问密钥被异常使用（包括丢失，泄露等情况），或不再使用访问密钥时，建议在访问密钥列表中立即删除密钥或者通知管理员重置相关密钥。

删除访问密钥时，需要输入登录密码和邮箱或者手机验证码进行验证。

说明

删除的访问密钥将永久删除且无法恢复。

3.2.1.2.2 上传数据到 OBS

操作场景

从 OBS 导入数据到集群之前，需要提前准备数据源文件，并将数据源文件上传到 OBS。如果您的数据文件已经在 OBS 上了，则只需完成上传数据到 OBS 中的步骤 2~步骤 3。

准备数据文件

准备需要上传到 OBS 的数据源文件。GaussDB(DWS)只支持 CSV、TEXT、ORC 和 CARBONDATA 格式的数据源文件。

如果用户数据无法以 CSV 格式保存，可以选择以文本类型保存为其他任意格式后缀的文件。

说明

根据导入数据原理，当数据源文件的数据量较大时，将数据文件存储到 OBS 前，尽可能均匀地将文件切分成多个，文件数量为 DataNode 的整数倍时，导入性能更好。

假设您已将 3 个 CSV 数据文件存储在 OBS 上，其原始数据分别如下：

- 数据文件“product_info.0”

  示例数据如下所示：

```
100,XHDK-A-1293-#fJ3,2017-09-01,A,2017 Autumn New Shirt Women,red,M,328,2017-09-04,715,good!
205,KDKE-B-9947-#kL5,2017-09-01,A,2017 Autumn New Knitwear Women,pink,L,584,2017-09-05,406,very good!
300,JODL-X-1937-#pV7,2017-09-01,A,2017 autumn new T-shirt men,red,XL,1245,2017-09-03,502,Bad.
310,QQPX-R-3956-#aD8,2017-09-02,B,2017 autumn new jacket women,red,L,411,2017-09-05,436,It's really super nice.
150,ABEF-C-1820-#mC6,2017-09-03,B,2017 Autumn New Jeans Women,blue,M,1223,2017-09-06,1200,The seller's packaging is exquisite.
```

- 数据文件“product_info.1”

  示例数据如下所示：

```
200,BCQP-E-2365-#qE4,2017-09-04,B,2017 autumn new casual pants men,black,L,997,2017-09-10,301,The clothes are of good quality.
250,EABE-D-1476-#oB1,2017-09-10,A,2017 autumn new dress women,black,S,841,2017-09-15,299,Follow the store for a long time.
108,CDXK-F-1527-#pL2,2017-09-11,A,2017 autumn new dress women,red,M,85,2017-09-14,22,It's really amazing to buy.
450,MMCE-H-4728-#nP9,2017-09-11,A,2017 autumn new jacket women,white,M,114,2017-09-14,22,Open the package and the clothes have no odor.
260,OCDA-G-2817-#bD3,2017-09-12,B,2017 autumn new woolen coat women,red,L,2004,2017-09-15,826,Very favorite clothes.
```

- 数据文件“product_info.2”

示例数据如下所示：

```
980,"ZKDS-J",2017-09-13,"B","2017 Women's Cotton Clothing","red","M",112,,,
98,"FKQB-I",2017-09-15,"B","2017 new shoes men","red","M",4345,2017-09-18,5473
50,"DMQY-K",2017-09-21,"A","2017 pants men","red","37",28,2017-09-
25,58,"good","good","good"
80,"GKLW-l",2017-09-22,"A","2017 Jeans Men","red","39",58,2017-09-25,72,"Very
comfortable."
30,"HWEC-L",2017-09-23,"A","2017 shoes women","red","M",403,2017-09-
26,607,"good!"
40,"IQPD-M",2017-09-24,"B","2017 new pants Women","red","M",35,2017-09-
27,52,"very good."
50,"LPEC-N",2017-09-25,"B","2017 dress Women","red","M",29,2017-09-28,47,"not
good at all."
60,"NQAB-O",2017-09-26,"B","2017 jacket women","red","S",69,2017-09-29,70,"It's
beautiful."
70,"HWNB-P",2017-09-27,"B","2017 jacket women","red","L",30,2017-09-30,55,"I
like it so much"
80,"JKHU-Q",2017-09-29,"C","2017 T-shirt","red","M",90,2017-10-02,82,"very
good."
```

上传数据到 OBS

步骤 1 上传数据到 OBS。

将待导入的数据源文件存储在 OBS 桶中。

1. 登录 OBS 管理控制台。

   单击“服务列表”，选择“对象存储服务”，打开 OBS 管理控制台页面。

2. 创建桶。

   如何创建 OBS 桶，具体请参见《对象存储服务用户指南》中的“控制台指南 > 管理桶 > 创建桶”章节。

   例如，创建以下两个桶：“mybucket”和“mybucket02”。

3. 新建文件夹。

   具体请参见《对象存储服务用户指南》中的“控制台指南 > 管理对象 > 新建文件夹”章节。

   例如：

   - 在已创建的 OBS 桶“mybucket”中新建一个文件夹“input_data”。
   - 在已创建的 OBS 桶“mybucket02”中新建一个文件夹“input_data”。

4. 上传文件。

   具体请参见《对象存储服务用户指南》中的“控制台指南 > 管理对象 >上传文件”章节。

   例如：

   - 将以下数据文件上传到 OBS 桶“mybucket”的“input_data”目录中。

     ```
     product_info.0
     product_info.1
     ```

   - 将以下数据文件上传到 OBS 桶“mybucket02”的“input_data”目录中。

     ```
     product_info.2
     ```

步骤 2 获取数据源文件的 OBS 路径。

数据源文件在上传到 OBS 桶之后，会生成全局唯一的访问路径。数据源文件的 OBS 路径用于创建外表时 location 参数设置。

location 参数中 OBS 文件的路径由“obs://”、桶名和文件路径组成，即为：obs://<bucket_name>/<file_path>/

例如，在本例中，location 参数中数据文件的 OBS 路径分别为：

```
obs://mybucket/input_data/product_info.0
obs://mybucket/input_data/product_info.1
obs://mybucket02/input_data/product_info.2
```

步骤 3 为导入用户设置 OBS 桶的读取权限。

在从 OBS 导入数据到集群时，执行导入操作的用户需要取得数据源文件所在 OBS 桶的读取权限。通过配置桶的 ACL 权限，可以将读取权限授予指定的用户帐号。

具体请参见《对象存储服务用户指南》中的“控制台指南 > 权限控制 > 配置桶 ACL”章节。

----结束

3.2.1.2.3 创建 OBS 外表

操作步骤

步骤 1 根据 3.2.1.2.2 上传数据到 OBS 中规划的路径，由此确定创建外表时使用的参数 **loaction** 的值。

步骤 2 用户获取 OBS 访问协议对应的 AK 值和 SK 值。获取访问密钥的具体步骤，请参见本文中的“导入数据 > 从 OBS 并行导入数据 > 创建访问密钥（AK 和 SK）”章节。

步骤 3 梳理待导入数据的格式信息，确定创建外表时使用的数据格式参数的值。需要收集的主要数据源格式信息如下：

- **format**：外表中数据源文件的格式。OBS 外表导入支持 CSV、TEXT 格式。缺省值为 TEXT。
- **header**：指定导出数据文件是否包含标题行，header 只能用于 CSV 格式的文件中。
- **delimiter**：指定数据文件行数据的字段分隔符，不指定则使用默认分隔符。
- 外表可以识别的更多参数，详细使用请参见数据格式参数 。

步骤 4 **规划并行导入容错性**，以控制导入过程中处理错误的方式。

- **fill_missing_fields**：数据入库时，数据源文件中某行的最后一个字段缺失时，请选择是直接将字段设为 Null，还是在错误表中报错提示。
- **ignore_extra_data**：数据源文件中的字段比外表定义列数多时，请选择是忽略多出的列，还是在错误表中报错提示。
- **per node reject limit**：本次数据导入过程中每个 DN 实例上允许出现的数据格式错误的数量。如果有一个 DN 实例上录入错误表中的错误数量超过设定值时，本次

导入失败，报错退出。可以选择不做限制，也可以根据所能容忍的错误数量选择一个上限值。

- **compatible_illegal_chars**：导入时遇到非法字符，选择如何处理。是将非法字符按照转换规则转换后入库，还是报错中止导入。

  非法字符容错转换规则如下：

  - 对于'\0'，容错后转换为空格。
  - 对于其他非法字符，容错后转换为问号。
  - 对非法字符进行容错转换时，如遇 NULL、DELIMITER、QUOTE、ESCAPE 也设置成了空格或问号，GaussDB(DWS)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。
- **error_table_name**：用于记录数据格式错误信息的错误表表名。并行导入结束后查询此错误信息表，能够获取详细的错误信息。
- 更多参数，详细使用请参见容错性参数。

步骤 5 根据前面步骤确定的参数，**创建 OBS 外表**。外表的创建语法以及详细使用，请参考 CREATE FOREIGN TABLE (OBS 导入导出)。

----结束

示例

在 GaussDB(DWS)数据库中，创建一个外表。参数信息如下所示：

- **数据格式参数访问秘钥（AK 和 SK）**
  - 用户获取 OBS 访问协议对应的 AK 值（access_key）。
  - 用户获取 OBS 访问协议对应的 SK 值 （secret_access_key）。

  说明

  请根据用户实际获取的 access_key 和 secret_access_key 的密钥替换示例中的对应内容。
- **设置数据格式参数**
  - **数据源文件格式（format）**为 CSV。
  - **编码格式（encoding）**为 UTF-8。
  - **是否使用加密（encrypt），**默认为 'off'。
  - **字段分隔符（delimiter）**为','。
  - **引号字符（quote）**使用默认值双引号。
  - **null（数据文件中空值的表示）**为“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）**为 false，当数据文件第一行不是标题行（即表头），不需要设置。

  说明

  OBS 导出数据时不支持该参数为 ture，使用缺省值 false。
- **设置导入时的容错性参数**

- **PER NODE REJECT LIMIT '*value*'** 为 unlimited，即接受导入过程中所有数据格式错误。
- **LOG INTO *error_table_name*** 指定为 product_info_err，将数据导入过程中出现的数据格式错误信息写入表 product_info_err。
- **fill_missing_fields** 为 true，即当数据加载时，若数据源文件中一行数据的最后一个字段缺失，则把最后一个字段的值设置为 NULL，不报错。
- **ignore_extra_data** 为 true，当数据加载时，若数据源文件比外表定义列数多，则忽略行尾多出来的列，不报错。

根据以上信息，创建的外表如下所示：

```
DROP FOREIGN TABLE product_info_ext;

CREATE FOREIGN TABLE product_info_ext
(
   product_price             integer       not null,
   product_id                char(30)      not null,
   product_time              date          ,
   product_level             char(10)      ,
   product_name              varchar(200)  ,
   product_type1             varchar(20)   ,
   product_type2             char(10)      ,
   product_monthly_sales_cnt   integer       ,
   product_comment_time        date          ,
   product_comment_num         integer       ,
   product_comment_content     varchar(200)
)
SERVER gsmpp_server
OPTIONS(

LOCATION 'obs://mybucket/input_data/product_info |
obs://mybucket02/input_data/product_info',
FORMAT 'CSV' ,
DELIMITER ',',
encoding 'utf8',
header 'false',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
fill_missing_fields 'true',
ignore_extra_data 'true'
)
READ ONLY
LOG INTO product_info_err
PER NODE REJECT LIMIT 'unlimited';
```

返回如下信息表示创建成功：

```
CREATE FOREIGN TABLE
```

3.2.1.2.4 执行导入数据

背景信息

在执行数据导入前，您可以参考以下优秀实践方法进行合理的设计部署，最大化的使用系统资源，以提高数据导入性能。

- OBS 的数据导入性能，多数场景受限于网络的并发访问速率，因此在 OBS 服务器上最好部署多个桶，使用多桶并发导入，提高 DN 数据传输利用率。
- 并发导入场景，与单表导入相似，至少应保证 I/O 性能大于网络最大速率。
- 配置 GUC 参数“raise_errors_if_no_files”、“partition_mem_batch”和“partition_max_cache_size”，设置导入时是否区分“导入文件记录数为空”和“导入文件不存在”、导入时的缓存个数以及数据缓存区大小。
- 若导入表存在索引，在数据导入过程中，将增量更新索引信息，影响数据导入性能。建议在执行数据导入前，先删除相关表的索引。在数据导入完成后，再重新创建索引。

操作步骤

步骤 1 在 GaussDB(DWS)数据库中，创建目标表，用于存储从 OBS 导入的数据。建表语法请参考 CREATE TABLE。

目标表的表结构和 OBS 上将要导入的数据源文件的字段要保持一一对应，即字段个数、字段类型要一致。并且，目标表和创建的外表的表结构也要保持一致，字段名称可以不一样。

步骤 2 （可选）若导入表存在索引，在数据导入过程中，将增量更新索引信息，影响数据导入性能。建议在执行数据导入前，先删除相关表的索引。在数据导入完成后，再重新创建索引。

步骤 3 执行数据导入。

```
INSERT INTO [目标表名] SELECT * FROM [foreign table 表名];
```

- 若出现以下类似信息，说明数据导入成功。请查询错误信息表，查看是否存在数据格式错误，详细操作请参见 3.2.1.2.5 处理错误表。

  ```
  INSERT 0 20
  ```

- 若出现数据加载错误，请参见 3.2.1.2.5 处理错误表，并重新执行数据导入。

----结束

示例

创建一个名为 product_info 的表，示例如下：

```
DROP TABLE IF EXISTS product info;
CREATE TABLE product info
(
    product_price             integer       not null,
    product_id                char(30)      not null,
    product_time              date          ,
```

```
   product_level             char(10)      ,
   product_name              varchar(200)   ,
   product_type1             varchar(20)    ,
   product_type2             char(10)      ,
   product_monthly_sales_cnt    integer       ,
   product_comment_time        date          ,
   product_comment_num         integer       ,
   product_comment_content     varchar(200)
)
with (
orientation = column,
compression=middle
)
DISTRIBUTE BY HASH (product_id);
```

执行以下命令将外表 *product_info*_ext 的数据导入到目标表 *product_info* 中：

```
INSERT INTO product_info SELECT * FROM product_info_ext;
```

3.2.1.2.5 处理错误表

操作场景

当数据导入发生错误时，请根据本文指引信息进行处理。

查询错误信息

数据导入过程中发生的错误，一般分为数据格式错误和非数据格式错误。

- 数据格式错误

  在创建外表时，通过设置参数“LOG INTO error_table_name”，将数据导入过程中出现的数据格式错误信息写入指定的错误信息表 error_table_name 中。您可以通过以下 SQL，查询详细错误信息。

  ```
  SELECT * FROM error_table_name;
  ```

  错误信息表结构如表 3-3 所示。

表3-3 错误信息表

列名称	类型	描述
nodeid	integer	报错节点编号。
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。
rownum	bigint	在数据源文件中，出现数据格式错误的行号。
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。

- 非数据格式错误

  对于非数据格式错误，一旦发生将导致整个数据导入失败。您可以根据执行数据导入过程中，界面提示的错误信息，帮助定位问题，处理错误表。

处理数据导入错误

根据获取的错误信息，请对照下表，处理数据导入错误。

表3-4 处理数据导入错误

错误信息	原因	解决办法
missing data for column "r_reason_desc"	1. 数据源文件中的列数比外表定义的列数少。 2. 对于 TEXT 格式的数据源文件，由于转义字符（\）导致 delimiter（分隔符）错位或者 quote（引号字符）错位造成的错误。 **示例：**目标表存在 3 列字段，导入的数据如下所示。由于存在转义字符“\”，分隔符“|”被转义为第二个字段的字段值，导致第三个字段值缺失。 `BE|Belgium\\|1`	1. 由于列数少导致的报错，选择下列办法解决： • 在数据源文件中，增加列“r_reason_desc”的字段值。 • 在创建外表时，将参数“fill_missing_fields”设置为“on”。即当导入过程中，若数据源文件中一行数据的最后一个字段缺失，则把最后一个字段的值设置为 NULL，不报错。 2. 对由于转义字符导致的错误，需检查报错的行中是否含有转义字符（\）。若存在，建议在创建外表时，将参数“noescaping”（是否不对'\'和后面的字符进行转义）设置为 true。
extra data after last expected column	数据源文件中的列数比外表定义的列数多。	• 在数据源文件中，删除多余的字段值。 • 在创建外表时，将参数“ignore_extra_data”设置为“on”。即在导入过程中，若数据源文件比外表定义的列数多，则忽略行尾多出来的列。
invalid input syntax for type numeric: "a"	数据类型错误。	在数据源文件中，修改输入字段的数据类型。根据此错误信息，请将输入的数据类型修改为 numeric。

错误信息	原因	解决办法
null value in column "staff_id" violates not-null constraint	非空约束。	在数据源文件中，增加非空字段信息。根据此错误信息，请增加“staff_id”列的值。
duplicate key value violates unique constraint "reg_id_pk"	唯一约束。	• 删除数据源文件中重复的行。 • 通过设置关键字“DISTINCT”，从 SELECT 结果集中删除重复的行，保证导入的每一行都是唯一的。 `INSERT INTO reasons SELECT DISTINCT * FROM foreign_tpcds_reasons;`
value too long for type character varying(16)	字段值长度超过限制。	在数据源文件中，修改字段值长度。根据此错误信息，字段值长度限制为 VARCHAR2(16)。

3.2.1.3 从 OBS 导入 ORC，CARBONDATA 数据

3.2.1.3.1 OBS 上的数据准备

操作场景

使用 SQL on OBS 功能查询 OBS 数据之前：

1. 假设您已将 ORC 数据存储在 OBS 上。

   例如，在使用 Hive 或 Spark 等组件时创建了 ORC 表，其表数据已经存储在 OBS 上的场景。

   假设有 2 个 ORC 数据文件“product_info.0”和“product_info.1”，其原始数据如原始数据所示，都已经存储在 OBS 桶“mybucket”的“demo.db/product_info_orc/”目录中。
2. 如果您的数据文件已经在 OBS 上了，请执行获取源数据的 OBS 路径并设置读取权限中的步骤。

说明

本小节以导入 ORC 格式为例，CARBONDATA 数据的导入方法与 ORC 格式相似。

原始数据

假设您已将 2 个 ORC 数据文件存储在 OBS 上，其原始数据分别如下：

- 数据文件“product_info.0”

示例数据如下所示：

```
100,XHDK-A-1293-#fJ3,2017-09-01,A,2017 Autumn New Shirt Women,red,M,328,2017-09-04,715,good!
205,KDKE-B-9947-#kL5,2017-09-01,A,2017 Autumn New Knitwear Women,pink,L,584,2017-09-05,406,very good!
300,JODL-X-1937-#pV7,2017-09-01,A,2017 autumn new T-shirt men,red,XL,1245,2017-09-03,502,Bad.
310,QQPX-R-3956-#aD8,2017-09-02,B,2017 autumn new jacket women,red,L,411,2017-09-05,436,It's really super nice.
150,ABEF-C-1820-#mC6,2017-09-03,B,2017 Autumn New Jeans Women,blue,M,1223,2017-09-06,1200,The seller's packaging is exquisite.
```

- 数据文件“product_info.1”

  示例数据如下所示：

```
200,BCQP-E-2365-#qE4,2017-09-04,B,2017 autumn new casual pants men,black,L,997,2017-09-10,301,The clothes are of good quality.
250,EABE-D-1476-#oB1,2017-09-10,A,2017 autumn new dress women,black,S,841,2017-09-15,299,Follow the store for a long time.
108,CDXK-F-1527-#pL2,2017-09-11,A,2017 autumn new dress women,red,M,85,2017-09-14,22,It's really amazing to buy.
450,MMCE-H-4728-#nP9,2017-09-11,A,2017 autumn new jacket women,white,M,114,2017-09-14,22,Open the package and the clothes have no odor.
260,OCDA-G-2817-#bD3,2017-09-12,B,2017 autumn new woolen coat women,red,L,2004,2017-09-15,826,Very favorite clothes.
```

获取源数据的 OBS 路径并设置读取权限

步骤 1 登录 OBS 管理控制台。

单击“服务列表”，选择“对象存储服务”，打开 OBS 管理控制台页面。

步骤 2 获取数据源文件的 OBS 路径。

数据源文件在上传到 OBS 桶之后，会生成全局唯一的访问路径。在创建外表时需要指定数据源文件的 OBS 路径。

如何查看 OBS 路径，请参见《对象存储服务用户指南》中的“控制台指南 > 管理对象 > 通过对象 URL 访问对象”章节。

例如，在本例中，查看到数据文件的 OBS 路径分别为：

```
https://obs.xxx.com/mybucket/demo.db/product_info_orc/product_info.0
https://obs.xxx.com/mybucket/demo.db/product_info_orc/product_info.1
```

步骤 3 为用户设置 OBS 桶的读取权限。

在使用 SQL on OBS 功能时，执行该功能的用户需要取得数据源文件所在 OBS 桶的读取权限。通过配置桶的 ACL 权限，可以将读取权限授予指定的用户帐号。

具体请参见《对象存储服务用户指南》中的“控制台指南 > 权限控制 > 配置桶 ACL”章节。

----结束

3.2.1.3.2 创建外部服务器

创建外部服务器，用于定义 OBS 服务器的信息，供外表调用。创建外部服务器的详细语法，请参见 CREATE SERVER。

（可选）新建用户及数据库并授予外表权限

如果您将使用普通用户在自定义数据库中创建外部服务器和外表，由于普通用户没有外表权限无法创建，所以，您必须参照以下步骤新建用户和数据库，并授予该用户外表权限。

以下示例，是新建一个普通用户 dbuser 并创建一个数据库 mydatabase，然后使用管理员用户授予 dbuser 外表权限。

步骤 1 使用数据库管理员通过 GaussDB(DWS)提供的数据库客户端连接默认数据库 gaussdb。

例如，使用 gsql 客户端的用户执行下面命令连接数据库：

```
gsql -d gaussdb -h 192.168.2.30 -U dbadmin -p 8000 -W password -r
```

步骤 2 新建一个普通用户，并用它创建一个数据库。

新建一个具有创建数据库权限的用户 dbuser：

```
CREATE USER dbuser WITH CREATEDB PASSWORD 'password';
```

切换为新建的用户：

```
SET ROLE dbuser PASSWORD 'password';
```

执行以下命令创建数据库：

```
CREATE DATABASE mydatabase;
```

查询数据库：

```
SELECT * FROM pg_database;
```

返回结果中有 **mydatabase** 的信息表示创建成功：

```
datname   | datdba | encoding | datcollate | datctype | datistemplate | datallowconn
| datconnlimit | datlastsysoid | datfrozenxid | dattablespace | datcompatibility |
datacl

------------+--------+----------+------------+----------+---------------+----------
----+--------------+---------------+--------------+---------------+----------------
--+---------------------------------------
--------------
 template1  |    10 |       0 | C         | C       | t           | t           |
-1 |        14146 |        1351 |         1663 | ORA           |
{=c/Ruby,Ruby=CTc/Ruby}
 template0  |    10 |       0 | C         | C       | t           | f           |
-1 |        14146 |        1350 |         1663 | ORA           |
{=c/Ruby,Ruby=CTc/Ruby}
 gaussdb    |    10 |       0 | C         | C       | f           | t           |
-1 |        14146 |        1352 |         1663 | ORA           |
{=Tc/Ruby,Ruby=CTc/Ruby,chaojun=C/Ruby,hu
obinru=C/Ruby}
```

```
 mydatabase |  17000 |       0 | C         | C       | f           | t          |
-1 |       14146 |       1351 |        1663 | ORA           |
(4 rows)
```

步骤 3 使用管理员用户给普通用户赋予创建外部服务器的权限和使用外表的权限。

使用数据库管理员用户通过 GaussDB(DWS)提供的数据库客户端连接新建的数据库。

例如，使用 gsql 客户端的用户可以直接使用如下语句切换为管理员用户去连接新建的数据库：

```
\c mydatabase dbadmin;
```

根据提示输入管理员用户密码。

说明

注意，必须先使用管理员用户连接到**将要创建外部服务器和使用外表的数据库**，再对普通用户进行授权。

默认只有系统管理员才可以创建外部服务器，普通用户需要授权才可以创建，执行以下命令授权：

```
GRANT ALL ON SCHEMA public TO dbuser;
GRANT ALL ON FOREIGN DATA WRAPPER dfs_fdw TO dbuser;
```

其中 fdw_name 的名字可以是 hdfs_fdw 或者 dfs_fdw，dbuser 为创建 SERVER 的用户名。

执行以下命令赋予用户使用外表的权限。

```
ALTER USER dbuser USEFT;
```

查看用户：

```
SELECT r.rolname, r.rolsuper, r.rolinherit,
  r.rolcreaterole, r.rolcreatedb, r.rolcanlogin,
  r.rolconnlimit, r.rolvalidbegin, r.rolvaliduntil,
  ARRAY(SELECT b.rolname
       FROM pg_catalog.pg_auth_members m
       JOIN pg_catalog.pg_roles b ON (m.roleid = b.oid)
       WHERE m.member = r.oid) as memberof
, r.rolreplication
, r.rolauditadmin
, r.rolsystemadmin
, r.roluseft
FROM pg_catalog.pg_roles r
ORDER BY 1;
```

返回结果中 dbuser 的信息中包含了 UseFT 权限，表示授权成功：

```
rolname  | rolsuper | rolinherit | rolcreaterole | rolcreatedb | rolcanlogin |
rolconnlimit | rolvalidbegin | rolvaliduntil | memberof | rolreplication |
rolauditadmin | rolsystemadmin | roluseft
-----------+----------+------------+---------------+-------------+-------------+---
-----------+---------------+---------------+----------+----------------+-----------
----+----------------+----------
 dbuser    | f        | t          | f             | t           | t           |          -1 |
```

```
|            | {}      | f           | f           | f           | t
 lily     | f       | t         | f           | f          | t          |          -1 |
|            | {}      | f           | f           | f           | f
 Ruby      | t       | t         | t           | t          | t          |          -1 |
|            | {}      | t           | t           | t           | t
```

----结束

创建外部服务器

步骤 1 使用即将创建外部服务器的用户去连接其对应的数据库。

在本示例中，将使用（可选）新建用户及数据库并授予外表权限中创建的普通用户 dbuser 连接其创建的数据库 mydatabase 。用户需要通过 GaussDB(DWS)提供的数据库客户端连接数据库。

例如，使用 gsql 客户端的用户可以通过以下两种方法中的一种进行连接：

- 如果已经登录了 gsql 客户端，可以执行以下命令切换数据库和用户：

  ```
  \c mydatabase dbuser;
  ```

  根据提示输入密码。
- 如果尚未登录 gsql 客户端，或者已经登录了 gsql 客户端执行\q 退出 gsql 后，执行以下命令重新进行连接：

  ```
  gsql -d mydatabase -h 192.168.2.30 -U dbuser -p 8000 -r
  ```

  根据提示输入密码。

步骤 2 创建外部服务器。

创建外部服务器的详细语法，请参见 CREATE SERVER。

例如，执行以下命令创建外部服务器'obs_server' ：

```
CREATE SERVER obs_server FOREIGN DATA WRAPPER dfs_fdw
OPTIONS (
  address 'obs.xxx.com' ,
  ACCESS KEY 'access_key_value_to_be_replaced',
  SECRET ACCESS KEY 'secret_access_key_value_to_be_replaced',
  encrypt 'on',
  type 'obs'
);
```

以下为必选参数的说明：

- **外部服务器名称**

  允许用户自定义名字。

  在本例中指定为“obs_server” 。
- **FOREIGN DATA WRAPPER**

  fdw_name 的名字可以是 hdfs_fdw 或者 dfs_fdw，它在数据库中已经存在。
- **OPTIONS 参数**
  - **address**

    指定 OBS 服务的终端节点。

address 的获取方法如下：

i. 先通过 3.2.1.3.1 OBS 上的数据准备中的 2 获取 OBS 路径。

ii. 在 OBS 上查看到的 OBS 路径，为 OBS 服务终端节点（Endpoint）：obs.*xxx*.*xxx*.com。

- **访问密钥（AK 和 SK）**（必选）

  GaussDB(DWS)需要通过访问密钥（AK 和 SK）访问 OBS，因此，必须先获取访问密钥。

  - “access_key”（必选）：表示用户的 AK 信息。
  - “secret_access_key”（必选）：表示用户的 SK 信息。

  获取访问密钥的具体步骤，请参见 3.2.1.2.1 创建访问密钥（AK 和 SK）。

- **type**

  取值为'obs'，表示 dfs_fdw 连接的是 OBS。

步骤 3 查看外部服务器：

```
SELECT * FROM pg_foreign_server WHERE srvname='obs_server';
```

返回结果如下所示，表示已经创建成功：

```
  srvname   | srvowner | srvfdw | srvtype | srvversion | srvacl |
srvoptions

------------+----------+--------+---------+------------+--------+------------------
-------------------------------------------------------------------------------------
-------------------------------------------------------------------------
-----------
 obs_server |    24661 |  13686 |         |            |        |
{address=xxx.xxx.x.xxx,access_key=xxxxxxxxxxxxxxxxxxxx,type=obs,secret_access_key=x
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}
(1 row)
```

----结束

3.2.1.3.3 创建外表

当完成 3.2.1.3.2 创建外部服务器后，在 GaussDB(DWS)数据库中创建一个 OBS 外表，用来访问存储在 OBS 上的数据。OBS 外表是只读的，只能用于查询操作，可直接使用 SELECT 查询其数据。

创建外表

创建外表的语法格式如下，详细的描述请参见 CREATE FOREIGN TABLE (SQL on Hadoop or OBS)。

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name
( [ { column_name type_name
    [ { [CONSTRAINT constraint_name] NULL |
    [CONSTRAINT constraint_name] NOT NULL |
      column_constraint [...]} ] |
      table_constraint [, ...]} [, ...] ] )
    SERVER dfs_server
```

```
OPTIONS ( { option_name ' value ' } [, ...] )
DISTRIBUTE BY {ROUNDROBIN | REPLICATION}
[ PARTITION BY ( column_name ) [ AUTOMAPPED ] ] ;
```

例如，创建一个名为"*product_info_ext_obs*"的外表，对语法中的参数按如下描述进行设置：

- **table_name**

  外表的表名。

- **表字段定义**
  - **column_name**：外表中的字段名。
  - **type_name**：字段的数据类型。

  多个字段用“,”隔开。

  外表的字段个数和字段类型，需要与 OBS 上保存的数据完全一致。

- **SERVER dfs_server**

  外表的外部服务器名称，这个 server 必须存在。外表通过设置外部服务器连接 OBS 读取数据。

  此处应填写为参照 3.2.1.3.2 创建外部服务器创建的外部服务器名称。

- **OPTIONS 参数**

  用于指定外表数据的各类参数，关键参数如下所示。
  - “format”：表示对应的 OBS 服务上的文件格式，支持“orc”、“carbondata”格式。
  - “foldername”：必选参数。数据源文件的 OBS 路径，此处仅需要填写“*/桶名/文件夹目录层级/*”。

    可以先通过 3.2.1.3.1 OBS 上的数据准备中的 2 获取数据源文件的完整的 OBS 路径，该路径为 OBS 服务的终端节点（Endpoint）。
  - “totalrows”：可选参数。该参数不是导入的总行数。由于 OBS 上文件可能很多，做 analyze 可能会很慢，通过“totalrows”参数，让用户来设置一个预估的值，使优化器能通过这个值做大小表的估计。一般预估值与实际值的数量级差不多时，查询效率较高。
  - **“encoding”**：外表中数据源文件的编码格式名称，缺省为 utf8。对于 OBS 外表此参数为必选项。

- **DISTRIBUTE BY**：

  这个子句是必须的，对于 OBS 外表，当前只支持 **ROUNDROBIN** 分布方式。

  表示外表在从数据源读取数据时，GaussDB(DWS)集群每一个节点随机读取一部分数据，并组成完整数据。

- **语法中的其他参数**

  其他参数均为可选参数，用户可以根据自己的需求进行设置，在本例中不需要设置。

根据以上信息，创建外表命令如下所示：

建立不包含分区列的 OBS 外表，表关联的外部服务器为 obs_server，表对应的 OBS 服务上的文件格式为‘orc’，OBS 上的数据存储路径为'/mybucket/data/'。

```
DROP FOREIGN TABLE IF EXISTS product_info_ext_obs;
CREATE FOREIGN TABLE product_info_ext_obs
(
   product_price             integer       not null,
   product_id               char(30)      not null,
   product_time              date          ,
   product_level             char(10)      ,
   product_name              varchar(200)  ,
   product_type1             varchar(20)   ,
   product_type2             char(10)      ,
   product_monthly_sales_cnt   integer       ,
   product_comment_time       date          ,
   product_comment_num        integer       ,
   product_comment_content     varchar(200)
) SERVER obs_server
OPTIONS (
format 'orc',
foldername '/mybucket/demo.db/product_info_orc/',
encoding 'utf8',
totalrows '10'
)
DISTRIBUTE BY ROUNDROBIN;
```

建立包含分区列的 OBS 外表，product_info_ext_obs 外表使用 product_manufacturer 字段作为分区键，obs/mybucket/demo.db/product_info_orc/路径下有如下分区目录：

分区目录 1：product_manufacturer=10001

分区目录 2：product_manufacturer=10010

分区目录 3：product_manufacturer=10086

...

```
DROP FOREIGN TABLE IF EXISTS product_info_ext_obs;
CREATE FOREIGN TABLE product_info_ext_obs
(
   product_price             integer       not null,
   product_id               char(30)      not null,
   product_time              date          ,
   product_level             char(10)      ,
   product_name              varchar(200)  ,
   product_type1             varchar(20)   ,
   product_type2             char(10)      ,
   product_monthly_sales_cnt   integer       ,
   product_comment_time       date          ,
   product_comment_num        integer       ,
   product_comment_content     varchar(200)  ,
   product_manufacturer    integer
) SERVER obs_server
OPTIONS (
format 'orc',
foldername '/mybucket/demo.db/product_info_orc/',
encoding 'utf8',
totalrows '10'
)
```

```
DISTRIBUTE BY ROUNDROBIN
PARTITION BY (product_manufacturer) AUTOMAPPED;
```

3.2.1.3.4 通过外表查询 OBS 上的数据

直接查询外表查看 OBS 上的数据

如果数据量较少，可直接使用 SELECT 查询外表，即可查看到 OBS 上的数据。

步骤 1 执行以下命令，则可以从外表查询数据。

```
SELECT * FROM product_info_ext_obs;
```

查询结果显示与原始数据显示相同，则表示导入成功。查询结果的结尾将显示以下信息：

```
(10 rows)
```

通过外表查询到数据后，用户可以将数据插入数据库的普通表。

----结束

导入数据后查询数据

步骤 1 在 GaussDB(DWS)数据库中，创建导入数据的目标表，用于存储导入的数据。

该表的表结构必须与 3.2.1.3.3 创建外表中创建的外表的表结构保持一致，即字段个数、字段类型要完全一致。

例如，创建一个名为 product_info 的表，示例如下：

```
DROP TABLE IF EXISTS product_info;

CREATE TABLE product_info
(
    product_price             integer       not null,
    product_id                char(30)      not null,
    product_time              date          ,
    product_level             char(10)      ,
    product_name              varchar(200)  ,
    product_type1             varchar(20)   ,
    product_type2             char(10)      ,
    product_monthly_sales_cnt   integer       ,
    product_comment_time        date          ,
    product_comment_num         integer       ,
    product_comment_content     varchar(200)
)
with (
orientation = column,
compression=middle
)
DISTRIBUTE BY HASH (product_id);
```

步骤 2 执行“INSERT INTO .. SELECT ..”命令从外表导入数据到目标表。

示例：

```
INSERT INTO product_info SELECT * FROM product_info_ext_obs;
```

若出现以下类似信息，说明数据导入成功。

```
INSERT 0 10
```

步骤 3 执行 SELECT 命令，查看从 OBS 导入到 GaussDB(DWS)中的数据。

```
SELECT * FROM product_info;
```

查询结果显示如原始数据中所示的数据，表示导入成功。查询结果的结尾将显示以下信息：

```
(10 rows)
```

----结束

3.2.1.3.5 清除资源

当完成本教程的示例后，如果您不再需要使用本示例中创建的资源，您可以删除这些资源，以免资源浪费或占用您的配额。步骤如下：

1. 删除外表和目标表
2. 删除创建的外部服务器
3. 删除数据库及其所属的用户

   如果您执行了（可选）新建用户及数据库并授予外表权限中的步骤，请删除数据库及其所属的用户。

删除外表和目标表

步骤 1 （可选）如果执行了导入数据后查询数据，请执行以下命令，删除目标表。

```
DROP TABLE product_info;
```

当结果显示为如下信息，则表示删除成功。

```
DROP TABLE
```

步骤 2 执行以下命令，删除外表。

```
DROP FOREIGN TABLE product_info_ext_obs;
```

当结果显示为如下信息，则表示删除成功。

```
DROP FOREIGN TABLE
```

----结束

删除创建的外部服务器

步骤 1 使用创建外部服务器的用户连接到外部服务器所在的数据库。

在本示例中，使用的是普通用户 dbuser 在数据库 mydatabase 中创建了一个外部服务器 。用户需要通过 GaussDB(DWS)提供的数据库客户端连接数据库。例如，使用 gsql 客户端的用户，可以通过以下两种方法中的一种进行连接：

- 如果已经登录了 gsql 客户端，可以执行以下命令进行切换：

  ```
  \c mydatabase dbuser;
  ```

  根据提示输入密码。
- 如果已经登录了 gsql 客户端，您也可以执行**\q** 退出 gsql 后，再执行以下命令重新进行连接：

  ```
  gsql -d mydatabase -h 192.168.2.30 -U dbuser -p 8000 -r
  ```

  根据提示输入密码。

步骤 2 删除创建的外部服务器。

执行以下命令进行删除，详细语法请参见 DROP SERVER。

```
DROP SERVER obs_server;
```

返回以下信息表示删除成功：

```
DROP SERVER
```

查看外部服务器：

```
SELECT * FROM pg_foreign_server WHERE srvname='obs_server';
```

返回结果如下所示，表示已经删除成功：

```
 srvname | srvowner | srvfdw | srvtype | srvversion | srvacl | srvoptions
---------+----------+--------+---------+------------+--------+------------
(0 rows)
```

----结束

删除数据库及其所属的用户

如果您执行了（可选）新建用户及数据库并授予外表权限中的步骤，请参照以下步骤删除数据库及其所属的用户。

步骤 1 删除自定义数据库。

通过 GaussDB(DWS)提供的数据库客户端连接默认数据库 gaussdb。

如果已经登录了 gsql 客户端，可以直接执行如下命令进行切换：

先切换到默认数据库：

```
\c gaussdb
```

根据界面提示输入密码。

执行以下命令，删除自定义数据库：

```
DROP DATABASE mydatabase;
```

返回以下信息表示删除成功：

```
DROP DATABASE
```

步骤 2 使用管理员用户，删除本示例中创建的普通用户。

使用数据库管理员用户通过 GaussDB(DWS)提供的数据库客户端连接数据库。

如果已经登录了 gsgl 客户端，可以直接执行如下命令进行切换：

```
\c gaussdb dbadmin
```

执行以下命令回收创建外部服务器的权限：

```
REVOKE ALL ON FOREIGN DATA WRAPPER dfs_fdw FROM dbuser;
```

其中 FOREIGN DATA WRAPPER 的名字只能是 dfs_fdw，dbuser 为创建 SERVER 的用户名。

执行以下命令删除用户：

```
DROP USER dbuser;
```

可使用**\du** 命令查询用户，确认用户是否已经删除。

----结束

3.2.1.3.6 支持的数据类型

目前大数据领域，主流文件格式为 ORC。GaussDB(DWS)主要支持 ORC 文件格式。用户利用 HIVE 将数据导出存储为 ORC 文件格式，使用 GaussDB(DWS)通过只读外表对 ORC 文件内的数据进行查询分析，因此，需要在 ORC 文件格式支持的数据类型与 GaussDB(DWS)自身支持数据类型间进行匹配，匹配状况如表 3-5 所示。同理，GaussDB(DWS)可通过只写外表将数据导出存储为 ORC 文件格式，使用 HIVE 读取 ORC 文件内容，相互之间也需要类型匹配，匹配状况如表 3-6 所示：

表3-5 ORC 格式的只读外表与 HIVE 数据类型匹配关系

类型名称	GaussDB(DWS)外表支持类型	HIVE 建表类型
1 字节整数	TINYINT（不推荐）	TINYINT
	SMALLINT（推荐）	TINYINT
2 字节整数	SMALLINT	SMALLINT
4 字节整数	INTEGER	INT
8 字节整数	BIGINT	BIGINT
单精度浮点数	FLOAT4 (REAL)	FLOAT
双精度浮点型	FLOAT8(DOUBLE PRECISION)	DOUBLE
科学数据类型	DECIMAL[p (,s)] 最大支持 38 位精度	DECIMAL 最大支持 38 位（HIVE 0.11）
日期类型	DATE	DATE
时间类型	TIMESTAMP	TIMESTAMP

类型名称	GaussDB(DWS)外表支持类型	HIVE 建表类型
BOOLEAN 类型	BOOLEAN	BOOLEAN
Char 类型	CHAR(n)	CHAR (n)
VarChar 类型	VARCHAR(n)	VARCHAR (n)
字符串(文本大对象)	TEXT(CLOB)	STRING

表3-6 ORC 格式的只写外表与 HIVE 数据类型匹配关系

类型名称	GaussDB(DWS)内表支持类型（数据源表）	GaussDB(DWS)只写外表对应的类型	HIVE 建表类型
1 字节整数	TINYINT	TINYINT（不推荐）	SMALLINT
		SMALLINT（推荐）	SMALLINT
2 字节整数	SMALLINT	SMALLINT	SMALLINT
4 字节整数	INTEGER、BINARY_INTEGER	INTEGER	INT
8 字节整数	BIGINT	BIGINT	BIGINT
单精度浮点数	FLOAT4 、REAL	FLOAT4、REAL	FLOAT
双精度浮点型	DOUBLE PRECISION、FLOAT8、BINARY_DOUBLE	DOUBLE PRECISION、FLOAT8、BINARY_DOUBLE	DOUBLE
科学数据类型	DECIMAL、NUMERIC	DECIMAL[p (,s)] 最大支持 38 位精度	precision <=38 时，DECIMAL, precision > 38 时， STRING
日期类型	DATE	TIMESTAMP[(p)] [WITHOUT TIME ZONE]	TIMESTAMP
时间类型	TIME [(p)] [WITHOUT TIME ZONE]、TIME [(p)] [WITH TIME ZONE]	TEXT	STRING

类型名称	GaussDB(DWS)内表支持类型（数据源表）	GaussDB(DWS)只写外表对应的类型	HIVE 建表类型
	TIMESTAMP[(p)] [WITHOUT TIME ZONE]、TIMESTAMP[(p)][WITH TIME ZONE]、SMALLDATETIME	TIMESTAMP[(p)] [WITHOUT TIME ZONE]	TIMESTAMP
	INTERVAL DAY (l) TO SECOND (p)、INTERVAL [FIELDS] [(p)]	VARCHAR(n)	VARCHAR(n)
BOOLEAN 类型	BOOLEAN	BOOLEAN	BOOLEAN
Char 类型	CHAR(n)、CHARACTER(n)、NCHAR(n)	CHAR(n)、CHARACTER(n)、NCHAR(n)	n<=255 时，CHAR(n), n>255 时，STRING
VarChar 类型	VARCHAR(n)、CHARACTER VARYING(n)、VARCHAR2(n)、	VARCHAR(n)	n<=65535 时，VARCHAR(n)，n>65535 时，STRING
	NVARCHAR2(n)	TEXT	STRING
字符串(文本大对象)	TEXT、CLOB	TEXT、CLOB	STRING
货币类型	MONEY	NUMERIC	BIGINT

须知

1. GaussDB(DWS)外表支持 NULL 定义，HIVE 数据表支持并采用相对应的 NULL 定义。
2. HIVE 数据表中的 TINYINT 的取值范围为[-128,127]，而 GaussDB(DWS) 的 TINYINT 的取值范围为[0,255]，因此，HIVE 表中的 TINYINT 类型在建 GaussDB(DWS)只读外表时最好采用 SMALLINT 类型，如果使用 TINYINT 有可能存在读取值与实际值不一致的情况。同样，GaussDB(DWS)的 TINYINT 类型在导出时，只写外表和 HIVE 的建表类型也最好采用 SMALLINT 类型。
3. GaussDB(DWS)外表的日期和时间类型，不支持时区定义，HIVE 不支持时区定义。
4. HIVE 中 date 类型只有日期，没有时间，GaussDB(DWS)的 date 类型包含日期和时间。
5. GaussDB(DWS)支持 ORC 的压缩格式，包括 ZLIB，SNAPPY，LZ4 及 NONE 压缩方式。
6. 其中 FLOAT4 格式本身存在不精准问题，求和等操作在不同环境下可能产生不同的结果，在高精度要求场景下建议使用 DECIMAL 类型代替。
7. 兼容 Teradata 数据库模式下，外表不支持 DATE 类型。

3.2.2 使用 GDS 从远端服务器导入数据

3.2.2.1 关于 GDS 并行导入

INSERT 和 COPY 方式执行数据导入时，是一个串行执行的过程，导入性能低，因此适用于小数据量的导入。对于大数据量的导入，GaussDB(DWS)支持使用 GDS 工具通过外表并行导入数据到集群。

当前版本的 GDS 已经支持从管道文件导入数据库，该功能使 GDS 的导入更加灵活多变。

- 当 GDS 用户的本地磁盘空间不足时：
  - 可直接将 hdfs 上的数据写入到管道文件而不需要占用额外的磁盘空间。
- 当用户导入前需要清洗数据时：
  - 用户可以根据自己的需求编写程序，将需要处理的数据流式实时的写入管道文件，完成导入的数据清洗工作。

说明

- 当前版本暂不支持 SSL 模式下 GDS 导入，请勿以 SSL 方式使用 GDS。
- 本章涉及的所有管道文件都是指 linux 上的命名管道。

概述

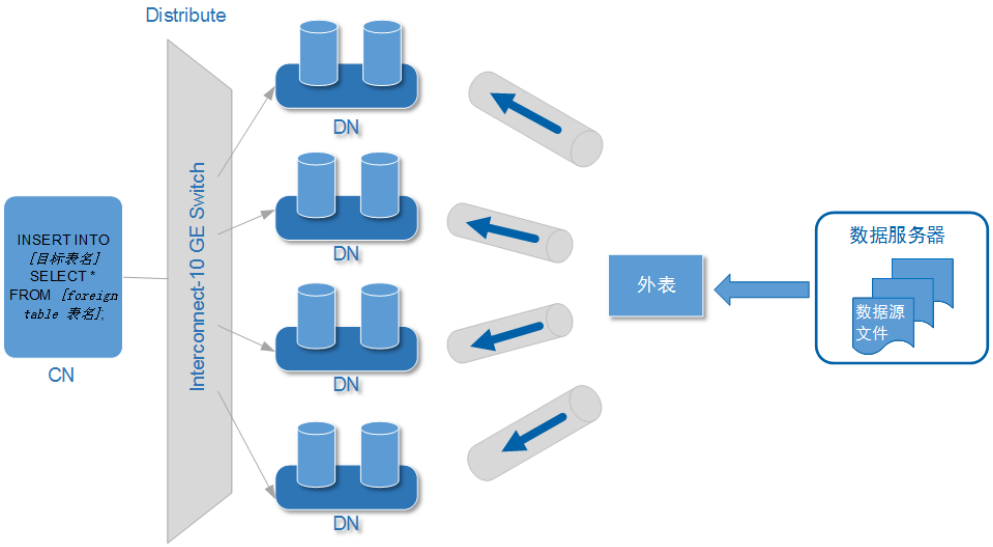
并行导入将存储在服务器普通文件系统中的数据导入到 GaussDB(DWS)数据库中。暂时不支持将存储在 HDFS 文件系统上的数据导入 GaussDB(DWS)。

并行导入功能通过外表设置的导入策略、导入数据格式等信息来识别数据源文件，利用多 DN 并行的方式，将数据从数据源文件导入到数据库中，从而提高整体导入性能。如图 3-3 所示：

- CN 只负责任务的规划及下发，把数据导入的工作交给了 DN，释放了 CN 的资源，使其有能力处理其他外部请求。
- 所有 DN 都参与数据导入，这样可以充分利用各设备的计算能力及网络带宽，提升导入效率。

外表灵活的 OPTION 设置，有利于在数据入库前对数据做预处理，例如非法字符替换、容错处理等。

图3-3 数据并行导入示意图



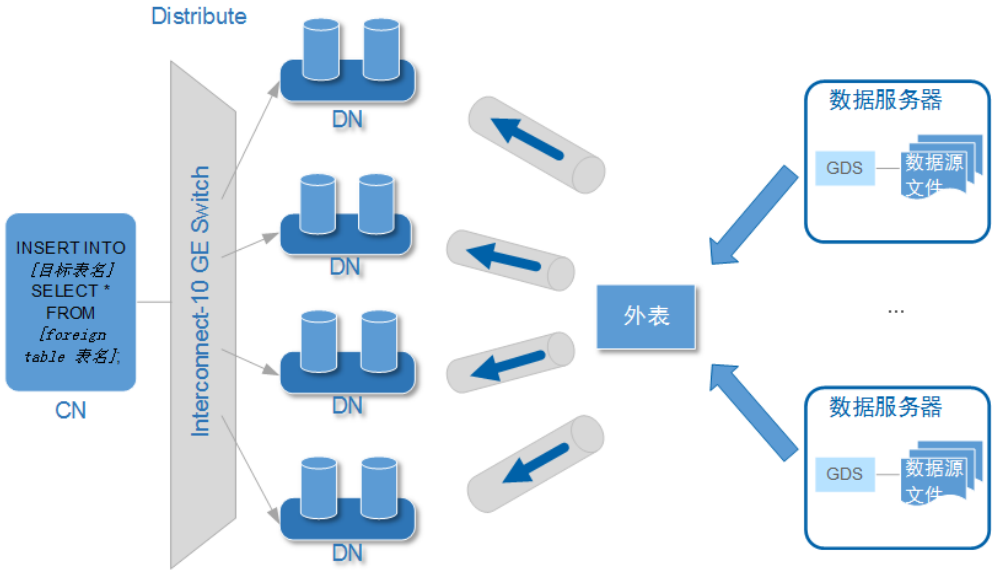
上图中所涉及的相关概念说明如下：

- **CN（Coordinator）**：GaussDB(DWS)协调节点。在导入场景下，接收到应用或客户端的导入 SQL 指令后，负责任务的规划及下发到 DN。
- **DN（Datanode）**：GaussDB(DWS)数据节点。接收 CN 下发的导入任务，将数据源文件中的数据通过外表写入数据库目标表中。
- **数据源文件**：存有数据的文件。文件中保存的是待导入数据库的数据。
- **数据服务器**：数据源文件所在的服务器称为数据服务器。基于安全考虑，建议数据服务器和 GaussDB(DWS)集群处于同一内网。
- **外表 Foreign Table**：用于识别数据源文件的位置、文件格式、存放位置、编码格式、数据间的分隔符等信息。是关联数据文件与数据库实表（目标表）的对象。
- **目标表**：数据库中的实表。数据源文件中的数据最终导入到这些表中存储，包括行存表和列存表。

GDS 并发导入

- 数据量大，数据存储在多个服务器上时，在每个数据服务器上安装配置、启动 GDS 后，各服务器上的数据可以并行入库。如图 3-4 所示。

  图3-4 多数据服务器并行导入



须知

GDS 进程数目不能超过 DN 数目。如果超过，会出现一个 DN 连接多个 GDS 进程的情形，可能会导致部分 GDS 异常运行。

- 数据存储在一台数据服务器上时，如果 GaussDB(DWS)及数据服务器上的 I/O 资源均还有可利用空间时，可以采用 GDS 多线程来支持并发导入。

  GDS 是根据导入事务并发数来决定服务运行线程数的。也就是说即使启动 GDS 时设置了多线程，也并不会加速单个导入事务。未做过人为事务处理时，一条 INSERT 语句就是一个导入事务。

  综上，多线程的使用场景如下：

  - 多表并发导入时，采用多线程充分利用资源及提升并发导入效率。
  - 对数据量大的某一事实表的导入进行提速。

    将该事实表对应的数据拆分为多个数据文件，通过多外表同时入库的方式实现多线程并发导入。注意需确保每个外表所能读取的数据文件不重复。

导入流程

图3-5 GDS 并行导入流程

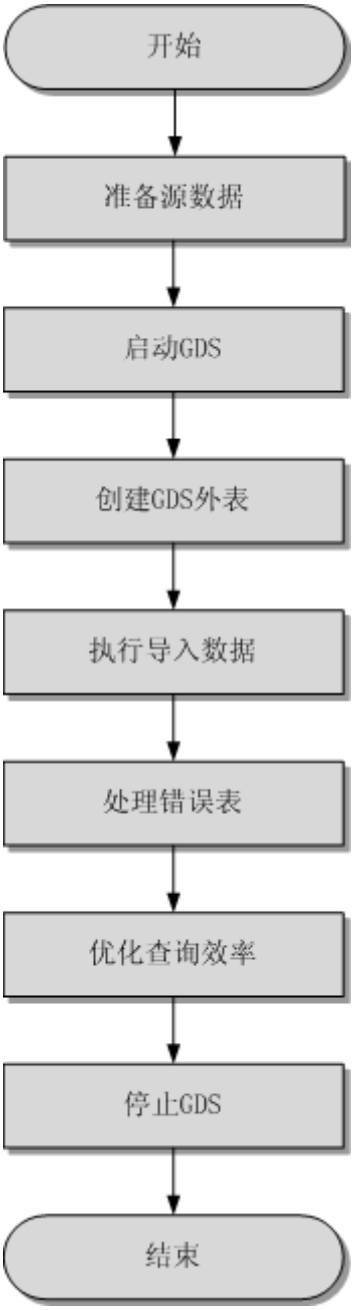


表3-7 流程说明

流程	说明
准备源数据。	准备需要导入数据库的源数据文件，并上传至数据服务器。 详细内容请参见 3.2.2.2 准备源数据。

流程	说明
启动 GDS。	在数据服务器上安装配置并启动 GDS。 详细内容请参见 3.2.2.3 安装配置和启动 GDS。
创建外表。	创建外表用于识别数据源文件中的数据。外表中保存了数据源文件的位置、文件格式、存放位置、编码格式、数据间的分隔符等信息。 详细内容请参见 3.2.2.4 创建 GDS 外表。
执行导入数据。	在创建好外表后，通过 INSERT 语句，将数据快速、高效地导入到目标表中。详细内容请参见 3.2.2.5 执行导入数据。
处理错误表。	在数据并行导入发生错误时，请根据具体的错误信息进行处理，以保证导入数据的完整性。 详细内容请参见 3.2.2.6 处理错误表。
优化查询效率。	导入数据后，通过 ANALYZE 语句生成表统计信息。ANALYZE 语句会将统计结果自动存储在系统表 PG_STATISTIC 中。执行计划生成器会使用这些统计数据，以生成最有效的查询执行计划。
停止 GDS	待数据导入完成后，登录每台数据服务器，分别停止 GDS。 GDS 的停止请参见 3.2.2.7 停止 GDS。

3.2.2.2 准备源数据

操作场景

通常在将数据导入数据库前，即将入库的数据已经在相关主机上了。这种保存着待入库数据的服务器为数据服务器。此时，只需检测以确认数据服务器和 GaussDB(DWS) 集群能够正常通信，并查看和记录数据在数据服务器上的存放目录备用。

如果待入库数据还没有就绪，则请先参考如下步骤，将数据上传到数据服务器上。

操作步骤

步骤 1 以 root 用户登录数据服务器。

步骤 2 创建数据文件存放目录“/input_data”。

```
mkdir -p /input_data
```

步骤 3 将数据源文件上传至上一步所创建的目录中。

GDS 并行导入支持 CSV、TEXT 格式的数据导入。请确保数据源文件符合格式要求。

----结束

3.2.2.3 安装配置和启动 GDS

操作场景

GaussDB(DWS)提供了数据服务工具 GDS 来帮助分发待导入的用户数据及实现数据的高速导入。GDS 需部署到数据服务器上。

数据量大，数据存储在多个服务器上时，在每个数据服务器上安装配置、启动 GDS 后，各服务器上的数据可以并行入库。GDS 在各台数据服务器上的安装配置和启动方法相同，本节以一台服务器为例进行说明。

背景信息

1. GDS 支持在如下的操作系统中安装：

   鲲鹏平台：

   - Community Enterprise Operating System 7.6。
   - EulerOS 2.0 SP8。
   - Red Hat Enterprise Linux Server release 7.5。
   - 中标麒麟 7.5/7.6。

   x86 平台：

   - SUSE Linux Enterprise Server 10 SP4 x86_64。
   - SUSE Linux Enterprise Server 11 SP1/SP2/SP3/SP4 x86_64。
   - SUSE Linux Enterprise Server 12 SP0/SP1/SP2/SP3/SP5 x86_64。
   - Red Hat Enterprise Linux Server release 6.4/6.5/6.6/6.7/6.8/6.9/7.0/7.1/7.2/7.3/7.4/7.5 x86_64。
   - Community Enterprise Operating System 6.4/6.5/6.6/6.7/6.8/6.9/7.0/7.1/7.2/7.3/7.4 x86_64。
   - EulerOS 2.5 x86_64。

2. GDS 的版本需与集群版本保持一致（如：GDS V100R008C00 版本与 DWS 1.3.X 版本配套），否则可能会出现导入导出失败或导入导出进程停止响应等情况。

   因此请勿使用历史版本的 GDS 进行导入。数据库版本升级后，请按照操作步骤中的办法下载 GaussDB(DWS)软件包解压缩自带的 GDS 进行安装配置和启动。在导入导出开始时，GaussDB(DWS)也会进行两端的版本一致性检测，不一致时会打屏显示报错信息并终止对应操作。

   GDS 的版本号的查看办法为：在 GDS 工具的解压目录下执行如下命令。

   ```
   gds -V
   ```

   数据库版本的查看办法为：连接数据库后，执行如下 SQL 命令查看。

   ```
   SELECT version();
   ```

操作步骤

步骤 1 在使用 GDS 导入/导出数据前，请先参考《数据仓库服务用户指南》的“教程：使用 GDS 导入数据 > 步骤 1：准备 ECS 作为 GDS 服务器”中的步骤：“准备弹性云主机作为 GDS 服务器”、“下载 GDS 工具包和 SSL 证书”。

步骤 2 以 root 用户登录待安装 GDS 的数据服务器，创建存放 GDS 工具包的目录。

```
mkdir -p /opt/bin/dws
```

步骤 3 将 GDS 工具包上传至上一步所创建的目录中。

以上传 SUSE Linux 版本的工具包为例 ，将 GDS 工具包“dws_client_8.1.x_suse_x64.zip”上传至上一步所创建的目录中。

步骤 4 （可选）如果使用 SSL 加密传输，请一并上传 SSL 证书至步骤 2 所创建的目录下。

步骤 5 在工具包所在目录下，解压工具包。

```
cd /opt/bin/dws
unzip dws_client_8.1.x_suse_x64.zip
```

步骤 6 创建 GDS 专有用户及其所属的用户组。此用户用于启动 GDS 及读取源数据。

```
groupadd gdsgrp
useradd -g gdsgrp gds_user
```

步骤 7 分别修改工具包和数据源文件目录属主为 GDS 专有用户。

```
chown -R gds_user:gdsgrp /opt/bin/dws/gds
chown -R gds_user:gdsgrp /input_data
```

步骤 8 切换到 gds_user 用户。

```
su - gds_user
```

若当前集群版本为 8.0.x 及以前版本，请跳过步骤 9，直接执行步骤 10。

若当前集群版本为 8.1.x 版本，则正常执行以下步骤。

步骤 9 执行环境依赖脚本。（仅 8.1.x 版本适用）

```
cd /opt/bin/dws/gds/bin
source gds_env
```

步骤 10 启动 GDS 服务。

GDS 是绿色软件，解压后启动即可。GDS 启动方式有两种。

方式一：直接使用“gds”命令，在命令项中设置启动参数。

方式二：将启动参数写进配置文件“gds.conf”后， 使用“gds_ctl.py”命令启动。

对于集中一次性导入的场景推荐使用第一种方式。对于需要隔段时间再次导入的场景，推荐使用第二种方式以配置文件的形式提升启动效率。

- 方式一：直接使用“gds”命令，启动 GDS。
  - 非 SSL 模式传输数据的情况下，启动 GDS。

    ```
    gds -d dir -p ip:port -H address_string -l log_file -D -t worker_num
    ```

    示例：

    ```
    /opt/bin/dws/gds/bin/gds -d /input_data/ -p 192.168.0.90:5000 -H
    10.10.0.1/24 -l /opt/bin/dws/gds/gds_log.txt -D -t 2
    ```

  - 使用 SSL 加密方式传输数据的情况下，启动 GDS。

```
gds -d dir -p ip:port -H address_string -l log_file -D
-t worker_num --enable-ssl --ssl-dir Cert_file
```

示例：

以步骤 4 中 SSL 证书以上传至/opt/bin 为例，命令如下。

```
/opt/bin/dws/gds/bin/gds -d /input data/ -p 192.168.0.90:5000 -H
10.10.0.1/24 -l /opt/bin/dws/gds/gds log.txt -D --enable-ssl --ssl-dir
/opt/bin/
```

命令中的斜体部分请根据实际替换。

- **-d** *dir*：保存有待导入数据的数据文件所在目录。本教程中为“/input_data/”。
- **-p** *ip:port*：GDS 监听 IP 和监听端口。默认值为：127.0.0.1，需要替换为能跟 GaussDB(DWS)通信的万兆网 IP。监听端口的取值范围：1024~65535。默认值为：8098。本教程配置为：192.168.0.90:5000。
- **-H** *address_string*：允许哪些主机连接和使用 GDS 服务。参数需为 CIDR 格式。此参数配置的目的是允许 GaussDB(DWS)集群可以访问 GDS 服务进行数据导入。所以请保证所配置的网段包含 GaussDB(DWS)集群各主机。
- **-l** *log_file*：存放 GDS 的日志文件路径及文件名。本教程为“/opt/bin/dws/gds/gds_log.txt”。
- **-D**：后台运行 GDS。仅支持 Linux 操作系统下使用。
- **-t** *worker_num*：设置 GDS 并发线程数。GaussDB(DWS)及数据服务器上的 I/O 资源均充足时，可以加大并发线程数。

  GDS 是根据导入事务并发数来决定服务运行线程数的。也就是说即使启动 GDS 时设置了多线程，也并不会加速单个导入事务。未做过人为事务处理时，一条 INSERT 语句就是一个导入事务。
- **--enable-ssl**：启用 SSL 加密方式传输数据。
- **--ssl-dir** *Cert_file*：SSL 证书所在目录。需与步骤 4 中的证书保存目录保持一致。
- 关于更多参数的设置信息请参考《数据仓库服务工具指南》中的“GDS 并行数据加载工具 > gds 命令简介”。

- 方式二：将启动参数写进配置文件“gds.conf”后， 使用“gds_ctl.py”命令启动。

  a. 使用如下命令，进入 GDS 工具包的“config”目录下，配置“gds.conf”文件。“gds.conf”配置详细信息请参考表 3-8。

  ```
  vim /opt/bin/dws/gds/config/gds.conf
  ```

  示例：

  配置“gds.conf”文件如下：

  ```
  <?xml version="1.0"?>
  <config>
  <gds name="gds1" ip="192.168.0.90" port="5000" data_dir="/input_data/"
  err_dir="/err" data_seg="100MB" err_seg="100MB"
  log_file="/log/gds_log.txt" host="10.10.0.1/24" daemon='true'
  recursive="true" parallel="32"></gds>
  </config>
  ```

  配置文件信息如下：

  - 数据服务器所在 IP 为 192.168.0.90，GDS 监听端口为 5000。

- 数据文件存放在“/input_data/”目录下。
- 错误日志文件存放在“/err”目录下。该目录需要拥有 GDS 读写权限的用户自行创建。
- 单个数据文件大小为 100MB。
- 每个错误日志大小为 100MB。
- 日志保存在“/log/gds_log.txt”文件中。该目录需要拥有 GDS 读写权限的用户自行创建。
- 只允许 IP 为 10.10.0.*的节点进行连接。
- GDS 进程以后台方式运行。
- 递归数据文件目录。
- 指定并发导入工作线程数目为 2。

b. 执行如下命令启动 GDS 并确认 GDS 是否启动成功。

```
python3 gds_ctl.py start
```

示例：

```
cd /opt/bin/dws/gds/bin
python3 gds_ctl.py start
Start GDS gds1               [OK]
gds [options]:
 -d dir          Set data directory.
 -p port         Set GDS listening port.
   ip:port       Set GDS listening ip address and port.
 -l log file      Set log file.
 -H secure ip range
                Set secure IP checklist in CIDR notation. Required for GDS
to start.
 -e dir          Set error log directory.
 -E size         Set size of per error log segment.(0 < size < 1TB)
 -S size         Set size of data segment.(1MB < size < 100TB)
 -t worker_num    Set number of worker thread in multi-thread mode, the
upper limit is 32. If without setting, the default value is 1.
 -s status_file   Enable GDS status report.
 -D             Run the GDS as a daemon process.
 -r             Read the working directory recursively.
 -h             Display usage.
```

----结束

gds.conf 参数说明

表3-8 gds.conf 配置说明

属性	说明	取值范围
name	标识名。	-
ip	监听 ip 地址。	IP 需为合法 IP 地址。 IP 的默认值：127.0.0.1

属性	说明	取值范围
port	监听端口号。	取值范围：1024~65535，正整数。 默认值：8098。
data_dir	数据文件目录。	-
err_dir	错误日志文件目录。	默认值：数据文件目录
log_file	日志文件路径。	-
host	设置允许连接到 GDS 的主机 IP 地址（参数为 CIDR 格式，仅支持 linux 系统）。	-
recursive	是否递归数据文件目录。	取值范围： • true：递归 。 • false：不递归。 默认值：false。
daemon	是否以 DAEMON（后台）模式运行。	取值范围： • true：以 DAEMON 模式运行。 • false：不以 DAEMON 模式运行。 默认值：false。
parallel	导入工作线程并发数目。	取值范围：0~32，正整数。 默认值：1。

3.2.2.4 创建 GDS 外表

外表中配置了数据源格式信息、GDS 服务的访问信息，从而 GaussDB(DWS)最终可以通过外表将数据服务器上的数据引流进数据库实表中。

操作步骤

步骤 1 收集数据源格式信息、GDS 服务的访问信息。

需要收集的主要数据源格式信息如下：

- **format**：GDS 外表导入支持 CSV、TEXT 和 FIXED 格式。请确认存放在数据服务器上待入库数据的格式。例如，假设待入库的数据为 CSV 格式。
- **header（仅支持 CSV，FIXED 格式）**：确认数据文件是否包含标题行。
- **delimiter**：确认数据文件中，字段间的分隔符。例如，假设是以英文逗号分隔的。
- **encoding**：数据源文件的数据编码格式。例如，假设为 UTF-8。

- **eol**：确认数据文件中，行间的换行符。例如，默认的换行符，如 0x0D0A、0X0A，或者自定义的换行符，如字符串!@#。该参数仅支持 TEXT 格式导入。
- 外表可识别的其他更多格式信息请参见数据格式参数。

需要收集的 GDS 服务的访问信息如下：

location：GDS 服务的访问地址。例如以 3.2.2.3 安装配置和启动 GDS 中的 GDS 信息为例。非 SSL 模式下的 location 为：**gsfs**://*192.168.0.90:5000//input_data/* 。SSL 模式下的 location 为：**gsfss**://*192.168.0.90:5000//input_data/* 。其中，“192.168.0.90:5000”为 GDS 服务的 IP 及端口号；“input_data”为 GDS 服务管理的数据源文件所在的路径。请根据实际情况替换。

步骤 2 依据数据源文件中的数据情况，设计导入容错机制。

GaussDB(DWS)支持如下的数据容错性处理，相当于数据入库前对数据做初步的简单清洗。

- **fill_missing_fields**：数据入库时，数据源文件中某行的最后一个字段缺失时，请选择是直接将字段设为 Null，还是在错误表中报错提示。
- **ignore_extra_data**：数据源文件中的字段比外表定义列数多时，请选择是忽略多出的列，还是在错误表中报错提示。
- **per node reject_limit**：本次数据导入过程中每个 DN 实例上允许出现的数据格式错误的数量。如果有一个 DN 实例上录入错误表中的错误数量超过设定值时，本次导入失败，报错退出。可以选择不做限制，也可以根据所能容忍的错误数量选择一个上限值。
- **compatible_illegal_chars**：导入时遇到非法字符，选择如何处理。是将非法字符按照转换规则转换后入库，还是报错中止导入。

  非法字符容错转换规则如下：

  - 对于'\0'，容错后转换为空格。
  - 对于其他非法字符，容错后转换为问号。
  - 对非法字符进行容错转换时，如遇 NULL、DELIMITER、QUOTE、ESCAPE 也设置成了空格或问号，GaussDB(DWS)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。
- **error_table_name**：用于记录数据格式错误信息的错误表表名。并行导入结束后查询此错误信息表，能够获取详细的错误信息。
- **remote log 'name'**：数据导入过程中的数据格式错误信息是否同时在 GDS 服务器上以文件方式保存。name 为错误数据文件的文件名前缀。
- 关于容错性参数的更多信息请参考容错性参数。

步骤 3 使用 gsql 或 Data Studio 连接数据库后，根据前面步骤所收集和规划的信息参数，创建 GDS 外表。

示例如下：

```
CREATE FOREIGN TABLE foreign_tpcds_reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
```

```
  r_reason_desc char(100)
)
 SERVER gsmpp_server
 OPTIONS
(
LOCATION 'gsfs://192.168.0.90:5000/input_data |
gsfs://192.168.0.91:5000/input_data',
FORMAT 'CSV' ,
DELIMITER ',',
ENCODING 'utf8',
HEADER 'false',
FILL_MISSING_FIELDS 'true',
IGNORE_EXTRA_DATA 'true'
)
LOG INTO product_info_err
PER NODE REJECT LIMIT 'unlimited';
```

示例中的各项说明如下：

- 外表字段需与数据库中即将存储数据的事实表保持一致。
- 对于 GDS 导入，SERVER gsmpp_server 请保持不变。
- location 参数请根据步骤 1 中收集的 GDS 服务访问信息修改。注意 GDS 使用 SSL 加密传输时，需要将“gsfs”替换为“gsfss”。
- FORMAT、DELIMITER、ENCODING、HEADER 请根据步骤 1 中收集的数据源格式信息填写。
- FILL_MISSING_FIELDS、IGNORE_EXTRA_DATA、LOG INTO 及 PER NODE REJECT LIMIT 请根据步骤 2 中设计的导入容错机制填写。注意 LOG INTO 是指将数据格式错误录入哪个错误表，即其取值为错误表表名。

CREATE FOREIGN TABLE 语法的更多信息，请参考 CREATE FOREIGN TABLE (GDS 导入导出)。

----结束

任务示例

除了以下示例，更多外表创建的示例请参考 3.2.2.8 GDS 导入示例。

- **示例 1：**创建 GDS 外表 foreign_tpcds_reasons，数据格式为 CSV。

```
CREATE FOREIGN TABLE foreign_tpcds_reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
)
 SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/* |
gsfs://192.168.0.91:5000/*', FORMAT 'CSV',MODE 'Normal', ENCODING 'utf8',
DELIMITER E'\x08', QUOTE E'\x1b', NULL '');
```

- **示例 2：**创建 GDS 导入外表 foreign_tpcds_reasons_SSL，使用 SSL 加密传输的模式传输，数据格式为 CSV。

```
CREATE FOREIGN TABLE foreign_tpcds_reasons_SSL
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
)
 SERVER gsmpp_server OPTIONS (location 'gsfss://192.168.0.90:5000/* |
gsfss://192.168.0.91:5000/*', FORMAT 'CSV',MODE 'Normal', ENCODING 'utf8',
DELIMITER E'\x08', QUOTE E'\x1b', NULL '');
```

- 示例 3：创建 GDS 外表 foreign_tpcds_reasons，数据格式为 TEXT。

```
CREATE FOREIGN TABLE foreign_tpcds_reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
) SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/* |
gsfs://192.168.0.91:5000/*', FORMAT 'TEXT', delimiter E'\x08',  null
'',reject_limit '2',EOL '0x0D') WITH err_foreign_tpcds_reasons;
```

3.2.2.5 执行导入数据

完成 GDS 的安装部署及外表创建后，本节介绍如何在 GaussDB(DWS)数据库中创建事实表并将数据导入事实表中。

对于记录数超过千万条的表，建议在执行全量数据导入前，先导入部分数据，以 3.6.2 查看数据倾斜状态，避免导入大量数据后发现数据倾斜，调整成本高。

前提条件

GDS 服务器和 GaussDB(DWS)集群之间网络可以互通。

- 需要创建一个弹性云主机作为 GDS 服务器。
- 创建的弹性云主机与 GaussDB(DWS)集群应处于同一区域、同一虚拟私有云和子网。

操作步骤

步骤 1 在 GaussDB(DWS)中创建目标表，用于存储导入的数据。建表语句请参见 CREATE TABLE。

步骤 2 （可选）若导入表存在索引，在数据导入过程中，将增量更新索引信息，影响数据导入性能。建议在执行数据导入前，先删除相关表的索引，但是如果不能保证数据唯一性不建议删除唯一索引。在数据导入完成后，再重新创建索引。

1. 假定在导入表“product_info”上的“product_id”字段上存在普通索引“product_idx”。在执行数据导入前，请先删除相关索引。

```
DROP INDEX product_idx;
```

2. 在数据导入完成后，重建索引。

```
CREATE INDEX product_idx ON product_info(product_id);
```

步骤 3 执行数据导入。

```
INSERT INTO [目标表名] SELECT * FROM [foreign table 表名];
```

- 若出现以下类似信息，说明数据导入成功。请查询错误信息表，查看是否存在数据格式错误，详细操作请参见 3.2.2.6 处理错误表。

  ```
  INSERT 0 9
  ```

- 若出现数据加载错误，请参见 3.2.2.6 处理错误表，并重新执行数据导入。

说明

- 若执行过程中出现数据加载错误，则数据全部导入失败，没有数据导入至目标表中。
- 编写批处理任务脚本，实现并发批量导入数据。并发量视机器资源使用情况而定。可通过几个表测试，监控资源利用率，根据结果提高或减少并发量。常用资源监控命令有：内存和 CPU 监控 top 命令，IO 监控命令 iostat，网络监控命令 sar 等。相关案例请参见。
- 在资源许可的情况下，多台 GDS 服务器并发导入会很大程度上提高数据导入效率。相关案例请参见多数据服务器并行导入。
- 对于高并发的 GDS 导入场景，为了保持 GDS 和 DN 间的数据连接稳定，可以将 GDS 服务器环境和 DN 所在环境的 TCP Keepalive 检测时间增长（推荐增长至 5 分钟）。调整集群环境的 TCP Keepalive 参数会影响故障检测的响应时间。

----结束

任务示例

1. 创建一个名为 reasons 的目标表。

   ```
   CREATE TABLE reasons
   (
     r_reason_sk   integer  not null,
     r_reason_id   char(16) not null,
     r_reason_desc char(100)
   )
   DISTRIBUTE BY HASH (r_reason_sk);
   ```

2. 在执行数据导入前，先删除相关表的索引。

   假定在导入表“reasons”上的“r_reason_id”字段上存在普通索引“reasons_idx”。在执行数据导入前，请先删除相关索引。

   ```
   DROP INDEX reasons_idx;
   ```

3. 将数据源文件中的数据通过外表“foreign_tpcds_reasons”导入到表“reasons”中。

   ```
   INSERT INTO reasons SELECT * FROM foreign_tpcds_reasons ;
   ```

4. 在数据导入完成后，再重新创建索引。

   ```
   CREATE INDEX reasons_idx ON reasons(r_reasons_id);
   ```

3.2.2.6 处理错误表

操作场景

当数据导入发生错误时，请根据本文指引信息进行处理。

查询错误信息

数据导入过程中发生的错误，一般分为数据格式错误和非数据格式错误。

- 数据格式错误

  在创建外表时，通过设置参数“LOG INTO error_table_name”，将数据导入过程中出现的数据格式错误信息写入指定的错误信息表 error_table_name 中。您可以通过以下 SQL，查询详细错误信息。

  ```
  SELECT * FROM error_table_name;
  ```

  错误信息表结构如表 3-9 所示。

表3-9 错误信息表

列名称	类型	描述
nodeid	integer	报错节点编号。
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。 当 GDS 导入时，同时会包括对应 GDS 服务端的 IP 地址端口信息。
rownum	bigint	在数据源文件中，出现数据格式错误的行号。
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。

- 非数据格式错误

  对于非数据格式错误，一旦发生将导致整个数据导入失败。您可以根据执行数据导入过程中，界面提示的错误信息，帮助定位问题，处理错误表。

处理数据导入错误

根据获取的错误信息，请对照下表，处理数据导入错误。

表3-10 处理数据导入错误

错误信息	原因	解决办法
missing data for column "r_reason_desc"	1. 数据源文件中的列数比外表定义的列数少。 2. 对于 TEXT 格式的数据源文件，由于转义字符（\）导致 delimiter（分隔符）错位或者 quote（引号字符）	1. 由于列数少导致的报错，选择下列办法解决： • 在数据源文件中，增加列“r_reason_desc”的字段值。 • 在创建外表时，将参数

错误信息	原因	解决办法
	错位造成的错误。 **示例：**目标表存在 3 列字段，导入的数据如下所示。由于存在转义字符“\”，分隔符“|”被转义为第二个字段的字段值，导致第三个字段值缺失。 `BE|Belgium\\|1`	“fill_missing_fields”设置为“on”。即当导入过程中，若数据源文件中一行数据的最后一个字段缺失，则把最后一个字段的值设置为 NULL，不报错。 2. 对由于转义字符导致的错误，需检查报错的行中是否含有转义字符（\）。若存在，建议在创建外表时，将参数“noescaping”（是否不对'\'和后面的字符进行转义）设置为 true。
extra data after last expected column	数据源文件中的列数比外表定义的列数多。	• 在数据源文件中，删除多余的字段值。 • 在创建外表时，将参数“ignore_extra_data”设置为“on”。即在导入过程中，若数据源文件比外表定义的列数多，则忽略行尾多出来的列。
invalid input syntax for type numeric: "a"	数据类型错误。	在数据源文件中，修改输入字段的数据类型。根据此错误信息，请将输入的数据类型修改为 numeric。
null value in column "staff_id" violates not-null constraint	非空约束。	在数据源文件中，增加非空字段信息。根据此错误信息，请增加“staff_id”列的值。
duplicate key value violates unique constraint "reg_id_pk"	唯一约束。	• 删除数据源文件中重复的行。 • 通过设置关键字“DISTINCT”，从 SELECT 结果集中删除重复的行，保证导入的每一行都是唯一的。 `INSERT INTO reasons SELECT DISTINCT * FROM foreign_tpcds_reasons;`
value too long for type character varying(16)	字段值长度超过限制。	在数据源文件中，修改字段值长度。根据此错误信息，字段值长度限制为

错误信息	原因	解决办法
		VARCHAR2(16)。

3.2.2.7 停止 GDS

操作场景

待导入数据成功后，停止 GDS。

操作步骤

步骤 1 以 gds_user 用户登录安装 GDS 的数据服务器。

步骤 2 请根据启动 GDS 的方式，选择停止 GDS 的方式。

- 若用户使用“gds”命令启动 GDS，请使用以下方式停止 GDS。
  a. 执行如下命令，查询 GDS 进程号。

  ```
  ps -ef|grep gds
  ```

  示例：其中 GDS 进程号为 128954。

  ```
  ps -ef|grep gds
  gds_user 128954     1  0 15:03 ?       00:00:00 gds -d /input_data/ -p
  192.168.0.90:5000 -l /log/gds_log.txt -D
  gds_user 129003 118723  0 15:04 pts/0    00:00:00 grep gds
  ```

  b. 使用“kill”命令，停止 GDS。其中 128954 为上一步骤中查询出的 GDS 进程号。

  ```
  kill -9 128954
  ```

----结束

3.2.2.8 GDS 导入示例

多数据服务器并行导入

规划数据服务器与集群处于同一内网，数据服务器 IP 为 192.168.0.90 和 192.168.0.91。数据源文件格式为 CSV。

1. 创建导入的目标表 tpcds.reasons。

```
CREATE TABLE tpcds.reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
);
```

2. 以 root 用户登录每台 GDS 数据服务器，在两台数据服务器上，分别创建数据文件存放目录“/input_data”。以下以 IP 为 192.168.0.90 的数据服务器为例进行操作，剩余服务器上的操作与它一致。

```
mkdir -p /input_data
```

3. （可选）创建用户及其所属的用户组。此用户用于启动 GDS。若该类用户及所属用户组已存在，可跳过此步骤。

```
groupadd gdsgrp
useradd -g gdsgrp gds_user
```

4. 将数据源文件均匀分发至相应数据服务器的“/input_data”目录中。
5. 修改每台数据服务器上数据文件及数据文件目录“/input_data”的属主为 gds_user。以下以 IP 为 192.168.0.90 的数据服务器为例，进行操作。

```
chown -R gds_user:gdsgrp /input_data
```

6. 以 gds_user 用户登录每台数据服务器上分别启动 GDS。

   其中 GDS 安装路径为“/opt/bin/dws/gds”，数据文件存放在“/input_data/”目录下，数据服务器所在 IP 为 192.168.0.90 和 192.168.0.91，GDS 监听端口为 5000，以后台方式运行。

   在 IP 为 192.168.0.90 的数据服务器上启动 GDS。

```
/opt/bin/dws/gds/bin/gds -d /input_data -p 192.168.0.90:5000 -H 10.10.0.1/24 -D
```

   在 IP 为 192.168.0.91 的数据服务器上启动 GDS。

```
/opt/bin/dws/gds/bin/gds -d /input_data -p 192.168.0.91:5000 -H 10.10.0.1/24 -D
```

7. 创建外表 tpcds.foreign_tpcds_reasons 用于接收数据服务器上的数据。

   其中设置**导入模式信息**如下所示：

   - 导入模式为 Normal 模式。
   - 由于启动 GDS 时，设置的数据源文件存放目录为“/input_data”，GDS 监听端口为 5000，所以设置参数“location”为“gsfs://192.168.0.90:5000/* | gsfs://192.168.0.91:5000/*”。

   设置**数据格式信息**是根据导出时设置的详细数据格式参数信息指定的，参数设置如下所示：

   - 数据源文件格式（format）为 CSV。
   - 编码格式（encoding）为 UTF-8。
   - 字段分隔符（delimiter）为 E'\x08'。
   - 引号字符（quote）为 E'\x1b'。
   - 数据文件中空值（null）为没有引号的空字符串。
   - 逃逸字符（escape）默认和 quote 相同。
   - 数据文件是否包含标题行（header）为默认值 false，即导入时数据文件第一行被识别为数据。

   设置**导入容错性**如下所示：

   - 允许出现的数据格式错误个数（PER NODE REJECT LIMIT 'value'）为 unlimited，即接受导入过程中所有数据格式错误。
   - 将数据导入过程中出现的数据格式错误信息（LOG INTO error_table_name）写入表 err_tpcds_reasons。

   根据以上信息，创建的外表如下所示：

```
CREATE FOREIGN TABLE tpcds.foreign_tpcds_reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
)
SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/* |
gsfs://192.168.0.91:5000/*', format 'CSV',mode 'Normal', encoding 'utf8',
delimiter E'\x08', quote E'\x1b', null '', fill_missing_fields 'false') LOG
INTO err_tpcds_reasons PER NODE REJECT LIMIT 'unlimited';
```

8. 通过外表 tpcds.foreign_tpcds_reasons，将数据导入目标表 tpcds.reasons。

```
INSERT INTO tpcds.reasons SELECT * FROM tpcds.foreign_tpcds_reasons;
```

9. 查询错误信息表 err_tpcds_reasons，处理数据导入错误。详细请参见 3.2.2.6 处理错误表。

```
SELECT * FROM err_tpcds_reasons;
```

10. 待数据导入完成后，以 gds_user 用户登录每台数据服务器，分别停止 GDS。

    以下以 IP 为 192.168.0.90 的数据服务器为例，停止 GDS。其中 GDS 进程号为 128954。

```
ps -ef|grep gds
gds_user 128954     1  0 15:03 ?       00:00:00 gds -d /input_data -p
192.168.0.90:5000 -D
gds_user 129003 118723  0 15:04 pts/0    00:00:00 grep gds
kill -9 128954
```

多线程导入

规划数据服务器与集群处于同一内网，数据服务器 IP 为 192.168.0.90，导入的数据源文件格式为 CSV，同时导入 2 个目标表。

1. 在数据库中创建导入的目标表 tpcds.reasons1 和 tpcds.reasons2。

```
CREATE TABLE tpcds.reasons1
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
) ;
CREATE TABLE tpcds.reasons2
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
) ;
```

2. 以 root 用户登录 GDS 数据服务器，创建数据文件存放目录“/input_data”，以及子目录“/input_data/import1/”和“/input_data/import2/”。

```
mkdir -p /input_data
```

3. 将目标表 tpcds.reasons1 的数据源文件存放在数据服务器“/input_data/import1/”目录下，将目标表 tpcds.reasons2 的数据源文件存放在目录“/input_data/import2/”下。

4. （可选）创建用户及其所属的用户组。此用户用于启动 GDS。若该用户及所属用户组已存在，可跳过此步骤。

```
groupadd gdsgrp
useradd -g gdsgrp gds_user
```

5. 修改数据服务器上数据文件及数据文件目录“/input_data”的属主为 gds_user。

```
chown -R gds_user:gdsgrp /input_data
```

6. 以 gds_user 用户登录数据服务器上启动 GDS。

   其中 GDS 安装路径为“/gds”，数据文件存放在“/input_data/”目录下，数据服务器所在 IP 为 192.168.0.90，GDS 监听端口为 5000，以后台方式运行，设定并发度为 2，并设定递归文件目录。

```
/gds/gds -d /input_data -p 192.168.0.90:5000 -H 10.10.0.1/24  -D -t 2 -r
```

7. 在数据库中创建外表 tpcds.foreign_tpcds_reasons1 和 tpcds.foreign_tpcds_reasons2 用于接收数据服务器上的数据。

   以下以外表 tpcds.foreign_tpcds_reasons1 为例，讲解设置的导入外表参数信息。

   其中设置的**导入模式信息**如下所示：

   - 导入模式为 Normal 模式。
   - 由于启动 GDS 时，设置的数据源文件存放目录为“/input_data/”，GDS 监听端口为 5000，实际存放数据源文件目录为“/input_data/import1/”，所以设置参数“location”为“gsfs://192.168.0.90:5000/import1/*”。

   设置的**数据格式信息**是根据导出时设置的详细数据格式参数信息指定的，参数设置如下所示：

   - 数据源文件格式（format）为 CSV。
   - 编码格式（encoding）为 UTF-8。
   - 字段分隔符（delimiter）为 E'\x08'。
   - 引号字符（quote）为 E'\x1b'。
   - 数据文件中空值（null）为没有引号的空字符串。
   - 逃逸字符（escape）默认和 quote 相同。
   - 数据文件是否包含标题行（header）为默认值 false，即导入时数据文件第一行被识别为数据。

   设置的**导入容错性**如下所示：

   - 允许出现的数据格式错误个数（PER NODE REJECT LIMIT 'value'）为 unlimited，即接受导入过程中所有数据格式错误。
   - 将数据导入过程中出现的数据格式错误信息（LOG INTO error_table_name）写入表 err_tpcds_reasons1。
   - 当数据源文件中一行的最后一个字段缺失（fill_missing_fields）时，自动设置为 NULL。

   根据以上信息，创建的外表 tpcds.foreign_tpcds_reasons1 如下所示：

```
CREATE FOREIGN TABLE tpcds.foreign tpcds reasons1
(
  r reason sk integer not null,
  r reason id char(16) not null,
  r_reason_desc char(100)
```

```
) SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/import1/*',
format 'CSV',mode 'Normal', encoding 'utf8', delimiter E'\x08', quote E'\x1b',
null '',fill_missing_fields 'on')LOG INTO err_tpcds_reasons1 PER NODE REJECT
LIMIT 'unlimited';
```

参考以上设置，创建的外表 tpcds.foreign_tpcds_reasons2 如下所示：

```
CREATE FOREIGN TABLE tpcds.foreign tpcds reasons2
(
  r reason sk integer not null,
  r reason id char(16) not null,
  r_reason_desc char(100)
) SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/import2/*',
format 'CSV',mode 'Normal', encoding 'utf8', delimiter E'\x08', quote E'\x1b',
null '',fill_missing_fields 'on')LOG INTO err_tpcds_reasons2 PER NODE REJECT
LIMIT 'unlimited';
```

8. 通过外表 tpcds.foreign_tpcds_reasons1 和 tpcds.foreign_tpcds_reasons2 将数据分别导入 tpcds.reasons1 和 tpcds.reasons2。

```
INSERT INTO tpcds.reasons1 SELECT * FROM tpcds.foreign_tpcds_reasons1;
INSERT INTO tpcds.reasons2 SELECT * FROM tpcds.foreign_tpcds_reasons2;
```

9. 查询错误信息表 err_tpcds_reasons1 和 err_tpcds_reasons2，处理数据导入错误。详细请参见 3.2.2.6 处理错误表。

```
SELECT * FROM err_tpcds_reasons1;
SELECT * FROM err_tpcds_reasons2;
```

10. 待数据导入完成后，以 gds_user 用户登录数据服务器，停止 GDS。

    其中 GDS 进程号为 128954。

```
ps -ef|grep gds
gds_user 128954     1  0 15:03 ?       00:00:00 gds -d /input_data -p
192.168.0.90:5000 -D -t 2 -r
gds_user 129003 118723  0 15:04 pts/0    00:00:00 grep gds
kill -9 128954
```

单个管道文件导入

步骤 1 启动 GDS。

```
gds -d /***/gds_data/ -D -p 192.168.0.1:7789 -l /***/gds_log/aa.log -H 0/0 -t 10 -D
```

如果需要设置管道文件的超时时间，则使用--pipe-timeout 参数设置。

步骤 2 执行数据导入。

1. 登录数据库创建内表。

```
CREATE TABLE test_pipe_1( id integer not null, sex text not null, name  text );
```

2. 创建只读外表。

```
CREATE FOREIGN TABLE foreign_test_pipe_tr( like test_pipe ) SERVER gsmpp_server
OPTIONS (LOCATION 'gsfs://192.168.0.1:7789/foreign_test_pipe.pipe', FORMAT
'text', DELIMITER ',',  NULL '', EOL '0x0a' ,file_type 'pipe',auto_create_pipe
'false');
```

3. 执行导入语句，此时语句会阻塞。

```
INSERT INTO test_pipe_1 select * from foreign_test_pipe_tr;
```

步骤 3 通过 GDS 管道文件导入数据。

1. 登录 GDS 服务器进入 GDS 数据目录。

```
cd /***/gds_data/
```

2. 创建管道文件，如果 auto_create_pipe 设置为 true 跳过此步骤。

```
mkfifo foreign_test_pipe.pipe;
```

说明

管道文件创建完成后，每执行完一次操作，业务会被自动清理。如果还需要执行其他业务，请参考该步骤重新创建管道文件。

3. 向管道文件中写入数据。

```
cat postgres_public_foreign_test_pipe_tw.txt > foreign_test_pipe.pipe
```

4. 若需要读取压缩文件到管道文件，执行：

```
gzip -d < out.gz > foreign_test_pipe.pipe
```

5. 若需要读取 hdfs 文件到管道文件，执行：

```
hdfs dfs -cat - /user/hive/***/test_pipe.txt > foreign_test_pipe.pipe
```

步骤 4 查看导入语句返回的结果：

```
INSERT INTO test_pipe_1 select * from foreign_test_pipe_tr;
INSERT 0 4
SELECT * FROM test_pipe_1;
id | sex |     name
----+-----+----------------
3 | 2   | 11111111111111
1 | 2   | 11111111111111
2 | 2   | 11111111111111
4 | 2   | 11111111111111
(4 rows)
```

----结束

多进程管道文件导入

GDS 支持多进程管道文件导入， 即启动一个外表对应多个 GDS。

以本地文件的导入为例：

步骤 1 启动多个 GDS，如果已经启动跳过此步骤。

```
gds -d /***/gds_data/ -D -p 192.168.0.1:7789 -l /***/gds_log/aa.log -H 0/0 -t 10 -D
gds -d /***/gds_data_1/ -D -p 192.168.0.1:7790 -l /***/gds_log_1/aa.log -H 0/0 -t
10 -D
```

如果需要设置管道文件的超时时间，则使用--pipe-timeout 参数设置。

步骤 2 执行数据导入。

1. 登录数据库创建内表。

```
CREATE TABLE test_pipe( id integer not null, sex text not null, name  text );
```

2. 创建只读外表。

```
CREATE FOREIGN TABLE foreign_test_pipe_tr( like test_pipe ) SERVER gsmpp_server
OPTIONS (LOCATION
'gsfs://192.168.0.1:7789/foreign_test_pipe.pipe|gsfs://192.168.0.1:7790/foreign
_test_pipe.pipe', FORMAT 'text', DELIMITER ',', NULL '', EOL '0x0a' , file_type
'pipe', auto_create_pipe 'false');
```

3. 导入语句，此时语句会阻塞。

```
INSERT INTO test_pipe_1 select * from foreign_test_pipe_tr;
```

步骤 3 通过 GDS 管道文件导入数据。

1. 登录 GDS，分别进入 GDS 数据目录。

```
cd /***/gds_data/
cd /***/gds_data_1/
```

2. 创建管道文件，如果 auto_create_pipe 设置为 true 跳过此步骤。

```
mkfifo foreign_test_pipe.pipe;
```

3. 分别读取管道文件并写入新文件：

```
cat postgres_public_foreign_test_pipe_tw.txt > foreign_test_pipe.pipe
```

步骤 4 查看导入语句返回的结果：

```
INSERT INTO test_pipe_1 select * from foreign_test_pipe_tr;
INSERT 0 4
SELECT * FROM test_pipe_1;
id | sex |     name
----+-----+----------------
3 | 2   | 11111111111111
1 | 2   | 11111111111111
2 | 2   | 11111111111111
4 | 2   | 11111111111111
(4 rows)
```

----结束

集群间不落地数据导入

步骤 1 启动 GDS。（如果已经启动跳过此步骤）

```
gds -d /***/gds_data/ -D -p GDS_IP:GDS_PORT -l /***/gds_log/aa.log -H 0/0 -t 10 -D
```

如果需要设置管道文件的超时时间，则使用--pipe-timeout 参数设置。

步骤 2 源数据库数据导出。

1. 登录目标数据库创建内表，并写入数据。

```
CREATE TABLE test_pipe( id integer not null, sex text not null, name  text );
INSERT INTO test_pipe values(1,2,'11111111111111');
INSERT INTO test_pipe values(2,2,'11111111111111');
INSERT INTO test_pipe values(3,2,'11111111111111');
INSERT INTO test_pipe values(4,2,'11111111111111');
INSERT INTO test_pipe values(5,2,'11111111111111');
```

2. 创建只写外表。

```
CREATE FOREIGN TABLE foreign_test_pipe( id integer not null, age text not null,
name  text ) SERVER gsmpp_server OPTIONS (LOCATION 'gsfs://GDS_IP:GDS_PORT/',
FORMAT 'text', DELIMITER ',', NULL '', EOL '0x0a' ,file_type 'pipe') WRITE ONLY;
```

3. 导入语句，此时语句会阻塞。

```
INSERT INTO foreign_test_pipe SELECT * FROM test_pipe;
```

步骤 3 目标集群导入数据。

1. 创建内表。

```
CREATE TABLE test_pipe (id integer not null, sex text not null, name text);
```

2. 创建只读外表。

```
CREATE FOREIGN TABLE foreign_test_pipe(like test_pipe) SERVER gsmpp_server
OPTIONS (LOCATION 'gsfs://GDS_IP:GDS_PORT/', FORMAT 'text', DELIMITER ',', NULL
'', EOL '0x0a' , file_type 'pipe', auto_create_pipe 'false');
```

3. 执行导入语句：

```
INSERT INTO test_pipe SELECT * FROM foreign_test_pipe;
```

步骤 4 查看目标集群导入语句返回的结果：

```
SELECT * FROM test_pipe;
 id | sex |     name
----+-----+----------------
  3 | 2   | 11111111111111
  6 | 2   | 11111111111111
  7 | 2   | 11111111111111
  1 | 2   | 11111111111111
  2 | 2   | 11111111111111
  4 | 2   | 11111111111111
  5 | 2   | 11111111111111
  8 | 2   | 11111111111111
  9 | 2   | 11111111111111
(9 rows)
```

----结束

说明

GDS 默认导出或者导入的管道文件命名规则为：“数据库名_模式名_外表名.pipe”，因此默认需要目标集群与源集群的数据库名及模式名保持一致。如果数据库或模式不一致，则可以在 location 的 url 中指定相同的管道文件。

示例：

- 只写外表指定管道名。

  ```
  CREATE FOREIGN TABLE foreign_test_pipe(id integer not null, age text not
  null, name  text) SERVER gsmpp_server OPTIONS (LOCATION
  'gsfs://GDS_IP:GDS_PORT/foreign_test_pipe.pipe', FORMAT 'text', DELIMITER
  ',',  NULL '', EOL '0x0a' ,file_type 'pipe') WRITE ONLY;
  ```

- 只读外表指定管道名。

  ```
  CREATE FOREIGN TABLE foreign_test_pipe(like test_pipe) SERVER gsmpp_server
  OPTIONS (LOCATION 'gsfs://GDS_IP:GDS_PORT/foreign_test_pipe.pipe', FORMAT
  ```

```
'text', DELIMITER ',',  NULL '', EOL '0x0a' ,file_type
'pipe',auto_create_pipe 'false');
```

3.2.3 从 MRS 导入数据到集群

3.2.3.1 从 MRS 导入数据概述

MapReduce 服务（MapReduce Service，简称 MRS）是一个基于开源 Hadoop 生态环境而运行的大数据集群，对外提供大容量数据的存储和分析能力，可解决用户的数据存储和处理需求。具体信息可参考《MapReduce 服务用户指南》。

用户可以将海量业务数据，存储在 MRS 的分析集群，即使用 Hive/Spark 组件保存。Hive/Spark 的数据文件则保存在 HDFS 中。GaussDB(DWS)支持在相同网络中，配置一个 GaussDB(DWS)集群连接到一个 MRS 集群，然后将数据从 HDFS 中的文件读取到 GaussDB(DWS)。

须知

确保 MRS 跟 DWS 网络互联互通，主要分以下几种场景：

场景一：MRS 与 DWS 在同一个区域、同一个 VPC 下，默认网络互通。

场景二：MRS 与 DWS 在同一个区域，不同 VPC 下，需要建立 VPC 对等连接，参见《虚拟私有云用户指南》的“对接连接简介”。

场景三：MRS 与 DWS 不在一个区域，需要通过“云连接(CC)”打通网络，请参见对应服务的用户指南。

场景四：MRS 属于云下场景，需要通过“云专线(DC)”或“虚拟专用网络(VPN)”打通网络，请参见对应服务的用户指南。

从 MRS 导入数据到集群的流程

1. 3.2.3.2 MRS 集群上的数据准备
2. （可选）3.2.3.3 手动创建外部服务器
3. 3.2.3.4 创建外表
4. 3.2.3.5 执行数据导入
5. 3.2.3.6 清除资源

3.2.3.2 MRS 集群上的数据准备

从 MRS 导入数据到 GaussDB(DWS)集群之前，假设您已经完成了以下准备工作：

1. 已创建 MRS 集群。
2. 在 MRS 集群上创建了 Hive/Spark ORC 表，且表数据已经存储到该表对应的 HDFS 路径上。

如果您已经完成上述准备，则可以跳过本章节。

为方便起见，以在 MRS 集群上创建 Hive ORC 表作为示例，完成上述准备工作。在 MRS 集群上创建 Spark ORC 表的大致流程和 SQL 语法，同 Hive 类似，在本文中不再展开描述。

数据文件

假设有数据文件 product_info.txt，示例数据如下所示：

```
100,XHDK-A-1293-#fJ3,2017-09-01,A,2017 Autumn New Shirt Women,red,M,328,2017-09-04,715,good
205,KDKE-B-9947-#kL5,2017-09-01,A,2017 Autumn New Knitwear Women,pink,L,584,2017-09-05,406,very good!
300,JODL-X-1937-#pV7,2017-09-01,A,2017 autumn new T-shirt men,red,XL,1245,2017-09-03,502,Bad.
310,QQPX-R-3956-#aD8,2017-09-02,B,2017 autumn new jacket women,red,L,411,2017-09-05,436,It's really super nice
150,ABEF-C-1820-#mC6,2017-09-03,B,2017 Autumn New Jeans Women,blue,M,1223,2017-09-06,1200,The seller's packaging is exquisite
200,BCQP-E-2365-#qE4,2017-09-04,B,2017 autumn new casual pants men,black,L,997,2017-09-10,301,The clothes are of good quality.
250,EABE-D-1476-#oB1,2017-09-10,A,2017 autumn new dress women,black,S,841,2017-09-15,299,Follow the store for a long time.
108,CDXK-F-1527-#pL2,2017-09-11,A,2017 autumn new dress women,red,M,85,2017-09-14,22,It's really amazing to buy
450,MMCE-H-4728-#nP9,2017-09-11,A,2017 autumn new jacket women,white,M,114,2017-09-14,22,Open the package and the clothes have no odor
260,OCDA-G-2817-#bD3,2017-09-12,B,2017 autumn new woolen coat women,red,L,2004,2017-09-15,826,Very favorite clothes
980,ZKDS-J-5490-#cW4,2017-09-13,B,2017 Autumn New Women's Cotton Clothing,red,M,112,2017-09-16,219,The clothes are small
98,FKQB-I-2564-#dA5,2017-09-15,B,2017 autumn new shoes men,green,M,4345,2017-09-18,5473,The clothes are thick and it's better this winter.
150,DMQY-K-6579-#eS6,2017-09-21,A,2017 autumn new underwear men,yellow,37,2840,2017-09-25,5831,This price is very cost effective
200,GKLW-l-2897-#wQ7,2017-09-22,A,2017 Autumn New Jeans Men,blue,39,5879,2017-09-25,7200,The clothes are very comfortable to wear
300,HWEC-L-2531-#xP8,2017-09-23,A,2017 autumn new shoes women,brown,M,403,2017-09-26,607,good
100,IQPD-M-3214-#yQ1,2017-09-24,B,2017 Autumn New Wide Leg Pants Women,black,M,3045,2017-09-27,5021,very good.
350,LPEC-N-4572-#zX2,2017-09-25,B,2017 Autumn New Underwear Women,red,M,239,2017-09-28,407,The seller's service is very good
110,NQAB-O-3768-#sM3,2017-09-26,B,2017 autumn new underwear women,red,S,6089,2017-09-29,7021,The color is very good
210,HWNB-P-7879-#tN4,2017-09-27,B,2017 autumn new underwear women,red,L,3201,2017-09-30,4059,I like it very much and the quality is good.
230,JKHU-Q-8865-#uO5,2017-09-29,C,2017 Autumn New Clothes with Chiffon Shirt,black,M,2056,2017-10-02,3842,very good
```

在 MRS 集群上创建 Hive ORC 表

1. 创建了 MRS 集群。

   具体操作请参见《MapReduce 服务用户指南》的“创建集群 > 创建自定义集群”章节。

2. 下载客户端。
   a. 在 MRS 集群页面，单击集群名称进入“概览”，单击“前往 Manager”，如果提示绑定公网 IP，请先绑定公网 IP。
   b. 输入 MRS Manager 的用户名 admin 和密码，密码为创建 MRS 集群时输入的 admin 密码。
   c. 登录成功后，选择“服务管理 > 下载客户端”，“客户端类型”选择“仅配置文件”，“下载路径”选择“服务器端”。单击“确定”。

3. 登录 MRS 集群的 Hive 客户端。
   a. 登录 Master 节点。

      具体操作，请参见《MapReduce 服务用户指南》中的“远程操作指南 > 登录 Master 节点”章节。
   b. 执行以下命令切换用户。

      ```
      sudo su - omm
      ```
   c. 执行以下命令切换到客户端目录：

      ```
      cd /opt/client
      ```
   d. 执行以下命令配置环境变量：

      ```
      source bigdata_env
      ```
   e. 如果当前集群已启用 Kerberos 认证，执行以下命令认证当前用户，当前用户需要具有创建 Hive 表的权限，具体操作请参见《MapReduce 服务用户指南》的“创建角色”章节。配置拥有对应权限的角色，具体操作请参见《MapReduce 服务用户指南》的“创建角色”章节。为用户绑定对应角色。如果当前集群未启用 Kerberos 认证，则无需执行此命令。

      ```
      kinit MRS 集群用户
      ```

      例如，kinit hiveuser。
   f. 执行以下命令启动 Hive 客户端：

```
beeline
```

4. 在 Hive 中创建数据库 demo。

   执行以下命令创建数据库：

```
CREATE DATABASE demo;
```

5. 在数据库 demo 中新建了一个 Hive TEXTFILE 类型的表 product_info，并将数据文件（product_info.txt）导入到该表对应的 HDFS 路径中。

   执行以下命令切换到 demo 数据库：

```
USE demo;
```

   执行以下命令，创建表 product_info，表字段按照数据文件中的数据进行定义：

```
DROP TABLE product_info;

CREATE TABLE product_info
(
    product_price             int          ,
    product_id               char(30)      ,
    product_time              date         ,
    product_level             char(10)      ,
    product_name             varchar(200)   ,
    product_type1             varchar(20)   ,
    product_type2             char(10)      ,
    product monthly sales cnt   int          ,
    product comment time       date         ,
    product comment num        int       ,
    product comment content     varchar(200)
)
row format delimited fields terminated by ','
stored as TEXTFILE;
```

   有关导入数据到 MRS 集群的操作，请参见《MapReduce 服务用户指南》中的“集群操作指导 > 管理现有集群 > 管理数据文件”章节。

6. 在数据库 demo 中创建了一个 Hive ORC 表 product_info_orc。

   执行以下命令，创建 Hive ORC 表 product_info_orc，表字段与上一步创建的表 product_info 完全一致：

```
DROP TABLE product_info_orc;

CREATE TABLE product_info_orc
(
    product_price             int          ,
    product_id               char(30)      ,
    product_time              date         ,
    product_level             char(10)      ,
    product_name             varchar(200)   ,
    product_type1             varchar(20)   ,
    product_type2             char(10)      ,
    product_monthly_sales_cnt   int          ,
    product_comment_time       date         ,
    product_comment_num        int          ,
    product_comment_content     varchar(200)
)
row format delimited fields terminated by ','
stored as orc;
```

7. 将 product_info 表的数据插入到 Hive ORC 表 product_info_orc 中。

```
insert into product_info_orc select * from product_info;
```

查询表 product_info_orc：

```
select * from product_info_orc;
```

如果查询到如数据文件所示的数据，表示已经成功将数据插入到 ORC 表。

3.2.3.3 手动创建外部服务器

创建外表语法（CREATE FOREIGN TABLE (SQL on Hadoop or OBS)）中，需指定一个与 MRS 数据源连接相关联的外部服务器。

当您通过 GaussDB(DWS)管理控制台创建 MRS 数据源连接时，数据库管理员 dbadmin 会在默认数据库 postgres 中自动创建一个外部服务器。因此，如果您希望在默认数据库 postgres 中创建外表读取 MRS 数据，可以跳过本章节。

如果您希望使用普通用户在自定义数据库中创建外表读取 MRS 数据，必须先在自定义数据库中手动创建一个外部服务器。本章节将为您介绍，如何使用普通用户在自定义数据库中创建外部服务器。步骤如下：

1. 请确保 GaussDB(DWS)集群已创建 MRS 数据源连接。

   具体操作请参见《数据仓库服务用户指南》的“管理 MRS 数据源 > 创建 MRS 数据源连接”章节。
2. 创建用户和数据库并授予外表权限
3. 手动创建外部服务器

说明

需要注意的是，当您不再需要从该 MRS 数据源读取数据时，通过 GaussDB(DWS)管理控制台删除 MRS 数据源，仅会删除在默认数据库 postgres 中自动创建的外部服务器，手动创建的外部服务器需要您手动删除，具体操作请参见删除手动创建的外部服务器中的描述。

创建用户和数据库并授予外表权限

以下示例，是新建一个普通用户 dbuser 并创建一个数据库 mydatabase，然后使用管理员用户授予 dbuser 外表权限。

步骤 1 使用数据库管理员通过 GaussDB(DWS)提供的数据库客户端连接默认数据库 gaussdb

例如，使用 gsql 客户端的用户通过如下语句连接数据库：

```
gsql -d gaussdb -h 192.168.2.30 -U dbadmin -p 8000 -W password -r
```

步骤 2 新建一个普通用户，并用它创建一个数据库。

新建一个具有创建数据库权限的用户 dbuser：

```
CREATE USER dbuser WITH CREATEDB PASSWORD 'password';
```

切换为新建的用户：

```
SET ROLE dbuser PASSWORD 'password';
```

执行以下命令创建数据库：

```
CREATE DATABASE mydatabase;
```

查询数据库：

```
SELECT * FROM pg_database;
```

返回结果中有 **mydatabase** 的信息表示创建成功：

```
datname   | datdba | encoding | datcollate | datctype | datistemplate | datallowconn
| datconnlimit | datlastsysoid | datfrozenxid | dattablespace | datcompatibility |
datacl

------------+--------+----------+------------+----------+---------------+----------
----+--------------+---------------+--------------+---------------+----------------
--+-------------------------------------
--------------
 template1  |    10 |       0 | C         | C       | t           | t          |
-1 |        14146 |        1351 |         1663 | ORA           |
{=c/Ruby,Ruby=CTc/Ruby}
 template0  |    10 |       0 | C         | C       | t           | f          |
-1 |        14146 |        1350 |         1663 | ORA           |
{=c/Ruby,Ruby=CTc/Ruby}
 gaussdb   |    10 |       0 | C         | C       | f           | t          |
-1 |        14146 |        1352 |         1663 | ORA           |
{=Tc/Ruby,Ruby=CTc/Ruby,chaojun=C/Ruby,hu
obinru=C/Ruby}
 mydatabase | 17000 |       0 | C         | C       | f           | t          |
-1 |        14146 |        1351 |         1663 | ORA           |
(4 rows)
```

步骤 3 使用管理员用户给普通用户赋予创建外部服务器的权限和使用外表的权限。

使用数据库管理员用户通过数据库客户端连接新建的数据库。

例如，使用 gsql 客户端的用户可以直接使用如下语句切换为管理员用户去连接新建的数据库：

```
\c mydatabase dbadmin;
```

根据提示输入用户密码。

说明

注意，必须先使用管理员用户连接到**将要创建外部服务器和使用外表的数据库**，再对普通用户进行授权。

默认只有系统管理员才可以创建外部服务器，普通用户需要授权才可以创建，执行以下命令授权：

```
GRANT ALL ON FOREIGN DATA WRAPPER hdfs_fdw TO dbuser;
```

其中 FOREIGN DATA WRAPPER 的名字只能是 hdfs_fdw，dbuser 为创建 SERVER 的用户名。

执行以下命令赋予用户使用外表的权限。

```
ALTER USER dbuser USEFT;
```

查看用户：

```
SELECT r.rolname, r.rolsuper, r.rolinherit,
  r.rolcreaterole, r.rolcreatedb, r.rolcanlogin,
  r.rolconnlimit, r.rolvalidbegin, r.rolvaliduntil,
  ARRAY(SELECT b.rolname
       FROM pg_catalog.pg_auth_members m
       JOIN pg_catalog.pg_roles b ON (m.roleid = b.oid)
       WHERE m.member = r.oid) as memberof
, r.rolreplication
, r.rolauditadmin
, r.rolsystemadmin
, r.roluseft
FROM pg_catalog.pg_roles r
ORDER BY 1;
```

返回结果中，dbuser 的信息中包含了 UseFT 权限，表示授权成功：

```
 rolname  | rolsuper | rolinherit | rolcreaterole | rolcreatedb | rolcanlogin |
rolconnlimit | rolvalidbegin | rolvaliduntil | memberof | rolreplication |
rolauditadmin | rolsystemadmin | roluseft
-----------+----------+------------+---------------+-------------+-------------+---
-----------+---------------+---------------+----------+----------------+-----------
----+----------------+----------
 dbuser    | f        | t          | f             | t           | t           |          -1 |
|             | {}       | f             | f             | f             | t
 lily      | f        | t          | f             | f           | t           |          -1 |
|             | {}       | f             | f             | f             | f
 Ruby      | t        | t          | t             | t           | t           |          -1 |
|             | {}       | t             | t             | t             | t
```

----结束

手动创建外部服务器

步骤 1 使用数据库管理员通过 GaussDB(DWS)提供的数据库客户端连接默认数据库 postgres。

例如：通过 gsql 客户端登录数据库的用户可以使用以下两种方法中的一种进行连接：

可以通过以下两种方法中的一种进行连接：

- 如果已经登录了 gsql 客户端，可以执行以下命令切换数据库和用户：

  ```
  \c postgres dbadmin;
  ```

  根据提示输入密码。

- 如果尚未登录 gsql 客户端，或者已经登录了 gsql 客户端执行**\q** 退出 gsql 后，执行以下命令重新进行连接：

  ```
  gsql -d postgres -h 192.168.2.30 -U dbadmin -p 8000 -W password -r
  ```

步骤 2 执行以下命令查询自动创建的外部服务器的信息。

```
SELECT * FROM pg_foreign_server;
```

返回结果如：

```
                srvname                    | srvowner | srvfdw | srvtype |
srvversion | srvacl |                                                 srvoptions
-------------------------------------------+----------+--------+---------+--
----------+--------+-------------------------------------------------------------------------
-------------------------------------------------------
 gsmpp_server                              |       10 |  13673 |         |          |
|
 gsmpp_errorinfo_server                    |       10 |  13678 |         |
|       |
 hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca |    16476 |  13685 |         |
|       |
{"address=192.168.1.245:25000,192.168.1.218:25000",hdfscfgpath=/MRS/8f79ada0-d998-
4026-9020-80d6de2692ca,type=hdfs}
(3 rows)
```

查询结果中，每一行代表一个外部服务器的信息。与 MRS 数据源连接相关联的外部服务器包含以下信息：

- srvname 值包含“hdfs_server”字样以及 MRS 集群的 ID，此 ID 与 MRS 管理控制台的集群列表 MRS ID 相同。
- srvoptions 字段中的 address 参数为 MRS 集群的主备节点的 IP 地址及端口。

您可以根据上述信息找到您所要的外部服务器，并记录下它的 srvname 和 srvoptions 的值。

步骤 3 切换为即将创建外部服务器的用户去连接其对应的数据库。

在本示例中，执行以下命令，使用创建用户和数据库并授予外表权限中创建的普通用户 dbuser 连接其创建的数据库 mydatabase 。

```
\c mydatabase dbuser;
```

步骤 4 创建外部服务器。

创建外部服务器的详细语法，请参见 CREATE SERVER。示例如下：

```
CREATE SERVER hdfs_server_8f79ada0_d998_4026_9020_80d6de2692cahdfs_server FOREIGN
DATA WRAPPER HDFS_FDW
OPTIONS
(
address '192.168.1.245:25000,192.168.1.218:25000',
hdfscfgpath '/MRS/8f79ada0-d998-4026-9020-80d6de2692ca',
type 'hdfs'
);
```

以下为必选参数的说明：

- 外部服务器名称

  允许用户自定义名字。

  在本例中，我们指定为前面的步骤步骤 2 中记录下来的 srvname 字段的值，如 '*hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca*'。

  不同的数据库之间资源是隔离的，因此在不同的数据库中外部服务器名称可以相同。
- FOREIGN DATA WRAPPER

只能指定为 HDFS_FDW，它在数据库中已经存在。

- OPTIONS 参数

  以下参数请分别指定为步骤步骤 2 中记录下来的 srvoptions 中的参数值。

  - address

    指定 HDFS 集群的主备节点所在的 IP 地址以及端口。

  - hdfscfgpath

    指定 HDFS 集群配置文件路径。该参数仅支持 type 为 HDFS 时设置。只能设置一个路径。

  - type

    取值为'hdfs'，表示 HDFS_FDW 连接的是 HDFS。

步骤 5 查看外部服务器。

```
SELECT * FROM pg_foreign_server WHERE
srvname='hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca';
```

返回结果如下所示，表示已经创建成功：

```
                 srvname                  | srvowner | srvfdw | srvtype |
srvversion | srvacl |                                                srvoptions
--------------------------------------------------+----------+--------+---------+--
----------+--------+-----------------------------------------------------------------
-----------------------------------------------------
 hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca |    16476 |  13685 |         |
|        |
{"address=192.168.1.245:25000,192.168.1.218:25000",hdfscfgpath=/MRS/8f79ada0-d998-
4026-9020-80d6de2692ca,type=hdfs}
(1 row)
```

----结束

3.2.3.4 创建外表

在 GaussDB(DWS)数据库中创建一个 Hadoop 外表，用来访问存储在 MRS HDFS 文件系统上的 Hadoop 结构化数据。Hadoop 外表是只读的，只能用于查询操作，可直接使用 SELECT 查询其数据。

前提条件

- 已创建 MRS 集群，并将数据导入 Hive/Spark 数据库中的 ORC 表。

  请参见 3.2.3.2 MRS 集群上的数据准备。

- GaussDB(DWS)集群已创建 MRS 数据源连接。

  具体操作请参见《数据仓库服务用户指南》的“管理 MRS 数据源 > 创建 MRS 数据源连接”章节。

获取 MRS 数据源的 HDFS 路径

有两种方法可以查看：

- **方法一：**

对于 Hive 数据，可以登录 MRS 的 Hive 客户端（参见 2），执行以下命令查看表的详细信息，并记录下 location 参数中的数据存储路径。

```
use <database_name>;
desc formatted <table_name>;
```

例如，返回结果中 location 参数值为“hdfs://hacluster/user/hive/warehouse/demo.db/product_info_orc/”，则记录 HDFS 路径为“/user/hive/warehouse/demo.db/product_info_orc/”。

- **方法二：**

  按以下步骤获取 HDFS 路径。

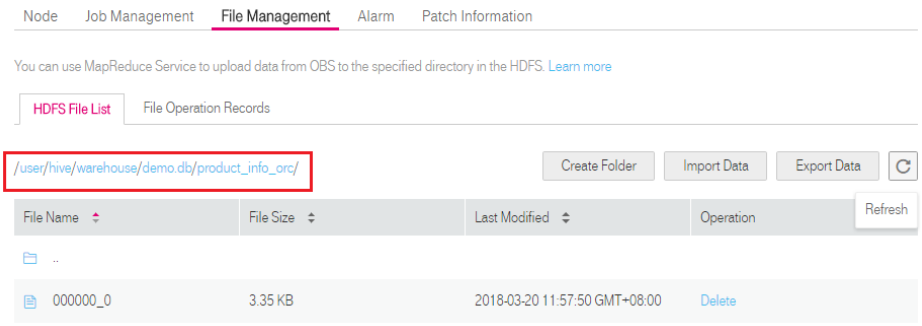
  a. 登录 MRS 管理控制台。

  b. 选择“集群列表 > 现有集群”，单击要查看的集群名称，进入集群基本信息页面。

  c. 单击“文件管理”，选择“HDFS 文件列表”。

  d. 进入您要导入到 GaussDB(DWS)集群的数据的存储目录，并记录其路径。

  图3-6 在 MRS 上查看数据存储路径



获取 MRS 数据源连接的外部服务器信息

步骤 1 使用创建外部服务器的用户去连接其对应的数据库。

是否使用普通用户在自定义数据库中创建外表，请根据需求进行选择：

- **是**

  a. 请先确保，您已按照 3.2.3.3 手动创建外部服务器章节中的步骤，创建了普通用户 dbuser 和它的数据库 mydatabase，并在 mydatabase 中手动创建了一个外部服务器。

  b. 使用用户 dbuser 通过 GaussDB(DWS)提供的数据库客户端连接数据库 mydatabase。

  如果已经使用 gsql 客户端连接至数据库，可以直接执行如下命令进行用户和数据库切换：

  ```
  \c mydatabase dbuser;
  ```

  根据界面提示输入密码。

- **否**

当您通过 GaussDB(DWS)管理控制台创建 MRS 数据源连接时，数据库管理员 dbadmin 会在默认数据库 postgres 中自动创建一个外部服务器。因此，如果使用数据库管理员 dbadmin 在默认数据库 postgres 中创建外表，需要通过 GaussDB(DWS)提供的数据库客户端工具连接数据库。例如，使用 gsql 客户端的用户通过如下命令连接数据库：

```
gsql -d postgres -h 192.168.2.30 -U dbadmin -p 8000 -W password -r
```

步骤 2 执行以下命令，查看已创建的 MRS 数据源连接的外部服务器信息。

```
SELECT * FROM pg_foreign_server;
```

说明

也可以执行\desc+命令查看外部服务器信息。

返回结果如：

```
                srvname                    | srvowner | srvfdw | srvtype |
srvversion | srvacl |                                        srvoptions
-------------------------------------------------+----------+--------+---------+--
----------+--------+-----------------------------------------------------------------------
-----------------------------------------------------
 gsmpp_server                                   |       10 |  13673 |         |
|
 gsmpp_errorinfo_server                         |       10 |  13678 |         |
|        |
 hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca |    16476 |  13685 |         |
|        |
{"address=192.168.1.245:25000,192.168.1.218:25000",hdfscfgpath=/MRS/8f79ada0-d998-
4026-9020-80d6de2692ca,type=hdfs}
(3 rows)
```

查询结果中，每一行代表一个外部服务器的信息。与 MRS 数据源连接相关联的外部服务器包含以下信息：

- srvname 值包含“hdfs_server”字样以及 MRS 集群的 ID，此 ID 与 MRS 管理控制台的集群列表 MRS ID 相同。
- srvoptions 字段中的 address 参数为 MRS 集群的主备节点的 IP 地址及端口。

您可以根据上述信息找到您所要的外部服务器，并记录下它的 srvname 和 srvoptions 的值。

----结束

创建外表

当完成获取 MRS 数据源连接的外部服务器信息和获取 MRS 数据源的 HDFS 路径后，就可以创建一个外表，用于读取 MRS 数据源数据。

创建外表的语法格式如下，详细的描述请参见（CREATE FOREIGN TABLE　(SQL on Hadoop or OBS)）。

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table name
( [ { column name type name
    [ { [CONSTRAINT constraint_name] NULL |
```

```
  [CONSTRAINT constraint_name] NOT NULL |
   column_constraint [...]} ] |
   table_constraint [, ...]} [, ...] ] )
SERVER dfs_server
OPTIONS ( { option_name ' value ' } [, ...] )
DISTRIBUTE BY {ROUNDROBIN | REPLICATION}
[ PARTITION BY ( column_name ) [ AUTOMAPPED ] ] ;
```

例如，创建一个名为"*foreign_product_info*"的外表，对语法中的参数按如下描述进行设置：

- **table_name**

  必选。外表的表名。

- **表字段定义**
  - **column_name**：外表中的字段名。
  - **type_name**：字段的数据类型。

  多个字段用“,”隔开。

  外表的字段个数和字段类型，需要与 MRS 上保存的数据完全一致。定义字段的数据类型之前，您必须先了解数据类型转换说明。

- **SERVER dfs_server**

  外表的外部服务器名称，这个 server 必须存在。外表通过设置外部服务器，从而关联 MRS 数据源连接并从 MRS 集群读取数据。

  此处应填写为通过获取 MRS 数据源连接的外部服务器信息查询到的“srvname”字段的值。

- **OPTIONS 参数**

  用于指定外表数据的各类参数，关键参数如下所示。
  - **format**：必选参数。取值只支持“orc”。表示数据源文件的格式，只支持 Hive 的 ORC 数据文件。
  - **foldername**：必选参数。表示数据在 HDFS 的存储目录或数据文件路径。

    如果是启用了 Kerberos 认证的 MRS 分析集群，请确保 MRS 数据源连接的 MRS 用户，拥有此目录的读取权限。

    请按照获取 MRS 数据源的 HDFS 路径中的步骤获取 HDFS 路径，该路径作为 **foldername** 的参数值。
  - **encoding**：可选参数。外表中数据源文件的编码格式名称，缺省为 utf8。
  - **DISTRIBUTE BY**

    表示外表的数据读取方式。有以下两种方式供选择，在本例中选择 ROUNDROBIN。
    - **ROUNDROBIN**：表示外表在从数据源读取数据时，GaussDB(DWS)集群每一个节点读取随机一部分数据，并组成完整数据。
    - **REPLICATION**：表示外表在从数据源读取数据时，GaussDB(DWS)集群每一个节点都读取一份完整数据。
  - **语法中的其他参数**

    其他参数均为可选参数，用户可以根据自己的需求进行设置，在本例中不需要设置。

根据以上信息，创建外表命令如下所示：

```
DROP FOREIGN TABLE IF EXISTS foreign_product_info;

CREATE FOREIGN TABLE foreign product info
(
   product price             integer       ,
   product id               char(30)      ,
   product time             date          ,
   product level            char(10)      ,
   product_name             varchar(200)   ,
   product_type1            varchar(20)    ,
   product_type2            char(10)      ,
   product_monthly_sales_cnt    integer       ,
   product_comment_time        date          ,
   product_comment_num         integer       ,
   product_comment_content     varchar(200)
) SERVER hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca
OPTIONS (
format 'orc',
encoding 'utf8',
foldername '/user/hive/warehouse/demo.db/product_info_orc/'
)
DISTRIBUTE BY ROUNDROBIN;
```

数据类型转换说明

当前用户导入到 Hive/Spark 的数据在 HDFS 存储为 ORC 文件格式，GaussDB(DWS)实际读取 HDFS 中的 ORC 文件，并对文件内的数据进行查询分析。

由于 Hive/Spark 支持的数据类型与 GaussDB(DWS)自身支持的数据类型存在差异，在创建外表定义表字段时，您需要了解这两者之间数据类型的对应关系，具体如表 3-11 所示：

表3-11 数据类型匹配表

类型名称	GaussDB(DWS)的 HDFS/OBS 外表支持的字段类型	Hive 表字段类型	Spark 表字段类型
2 字节整数	SMALLINT	SMALLINT	SMALLINT
4 字节整数	INTEGER	INT	INT
8 字节整数	BIGINT	BIGINT	BIGINT
单精度浮点数	FLOAT4 (REAL)	FLOAT	FLOAT
双精度浮点型	FLOAT8(DOUBLE PRECISION)	DOUBLE	FLOAT
科学数据类型	DECIMAL[p (,s)] 最大支持 38 位精度	DECIMAL 最大支持 38 位（Hive 0.11）	DECIMAL

类型名称	GaussDB(DWS)的HDFS/OBS 外表支持的字段类型	Hive 表字段类型	Spark 表字段类型
日期类型	DATE	DATE	DATE
时间类型	TIMESTAMP	TIMESTAMP	TIMESTAMP
Boolean 类型	BOOLEAN	BOOLEAN	BOOLEAN
Char 类型	CHAR(n)	CHAR (n)	STRING
VarChar 类型	VARCHAR(n)	VARCHAR (n)	VARCHAR (n)
字符串	TEXT(CLOB)	STRING	STRING

3.2.3.5 执行数据导入

直接查询外表查看 MRS 数据源的数据

如果数据量较少，可直接使用 SELECT 查询外表，即可查看到 MRS 数据源的数据。

步骤 1 执行以下命令，则可以从外表查询数据。

```
SELECT * FROM foreign_product_info;
```

查询结果显示如数据文件中所示的数据，表示导入成功。查询结果的结尾将显示以下信息：

```
(20 rows)
```

通过外表查询到数据后，用户可以将数据插入数据库的普通表。

----结束

导入数据后查询数据

也可以将 MRS 数据导入 GaussDB(DWS)后，再查询数据。

步骤 1 在 GaussDB(DWS)数据库中，创建导入数据的目标表，用于存储导入的数据。

该表的表结构必须与 3.2.3.4 创建外表中创建的外表的表结构保持一致，即字段个数、字段类型要完全一致。

例如，创建一个名为 product_info 的表，示例如下：

```
DROP TABLE IF EXISTS product info;
CREATE TABLE product info
(
   product_price             integer       ,
   product_id                char(30)      ,
   product_time              date          ,
   product_level             char(10)      ,
   product_name              varchar(200)   ,
```

```
   product_type1             varchar(20)    ,
   product_type2             char(10)      ,
   product_monthly_sales_cnt   integer       ,
   product_comment_time        date          ,
   product_comment_num         integer       ,
   product_comment_content     varchar(200)
)
with (
orientation = column,
compression=middle
)
DISTRIBUTE BY HASH (product_id);
```

步骤 2 执行“INSERT INTO .. SELECT ..”命令从外表导入数据到目标表。

示例：

```
INSERT INTO product_info SELECT * FROM foreign_product_info;
```

若出现以下类似信息，说明数据导入成功。

```
INSERT 0 20
```

步骤 3 执行 SELECT 命令，查看从 MRS 导入到 GaussDB(DWS)中的数据。

```
SELECT * FROM product_info;
```

查询结果显示如数据文件中所示的数据，表示导入成功。查询结果的结尾将显示以下信息：

```
(20 rows)
```

----结束

3.2.3.6 清除资源

当完成本教程的示例后，如果您不再需要使用本示例中创建的资源，您可以删除这些资源，以免资源浪费或占用您的配额。

删除外表和目标表

步骤 1 （可选）如果执行了导入数据后查询数据，请执行以下命令，删除目标表。

```
DROP TABLE product_info;
```

步骤 2 执行以下命令，删除外表。

```
DROP FOREIGN TABLE foreign_product_info;
```

----结束

删除手动创建的外部服务器

如果执行了 3.2.3.3 手动创建外部服务器，请按照以下步骤删除外部服务器、数据库和用户。

步骤 1 使用创建外部服务器的用户通过 GaussDB(DWS)提供的数据库客户端连接到外部服务器所在的数据库。

例如，使用 gsql 客户端的用户可以通过以下两种方法中的一种进行连接：

- 如果已经登录了 gsql 客户端，可以执行以下命令进行切换：

  ```
  \c mydatabase dbuser;
  ```

  根据提示输入密码。
- 如果已经登录了 gsql 客户端，您也可以执行**\q** 退出 gsql 后，再执行以下命令重新进行连接：

  ```
  gsql -d mydatabase -h 192.168.2.30 -U dbuser -p 8000 -r
  ```

  根据提示输入密码。

步骤 2 删除手动创建的外部服务器。

执行以下命令进行删除，详细语法请参见 DROP SERVER：

```
DROP SERVER hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca;
```

返回以下信息表示删除成功：

```
DROP SERVER
```

查看外部服务器：

```
SELECT * FROM pg_foreign_server WHERE
srvname='hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca';
```

返回结果如下所示，表示已经删除成功：

```
 srvname | srvowner | srvfdw | srvtype | srvversion | srvacl | srvoptions
---------+----------+--------+---------+------------+--------+------------
(0 rows)
```

步骤 3 删除自定义数据库。

通过 GaussDB(DWS)提供的数据库客户端连接默认数据库 postgres。

如果已经登录了 gsql 客户端，可以直接执行如下命令进行切换：

```
\c postgres
```

根据界面提示输入密码。

执行以下命令，删除自定义数据库：

```
DROP DATABASE mydatabase;
```

返回以下信息表示删除成功：

```
DROP DATABASE
```

步骤 4 使用管理员用户，删除本示例中创建的普通用户。

使用数据库管理员用户通过 GaussDB(DWS)提供的数据库客户端连接数据库。

如果已经登录了 gsql 客户端，可以直接执行如下命令进行切换：

```
\c postgres dbadmin
```

执行以下命令回收创建外部服务器的权限：

```
REVOKE ALL ON FOREIGN DATA WRAPPER hdfs_fdw FROM dbuser;
```

其中 FOREIGN DATA WRAPPER 的名字只能是 hdfs_fdw，dbuser 为创建 SERVER 的用户名。

执行以下命令删除用户：

```
DROP USER dbuser;
```

可使用**\du** 命令查询用户，确认用户是否已经删除。

----结束

3.2.3.7 错误处理

如下错误信息，表示 GaussDB(DWS)期望读取 ORC 数据文件，但实际却是*.txt 类型的数据文件。请先创建 Hive ORC 类型的表，并将数据存储到该 Hive ORC 表中。

```
ERROR:  dn_6009_6010: Error occurs while creating an orc reader for file
/user/hive/warehouse/products_info.txt, detail can be found in dn log of
dn_6009_6010.
```

3.2.4 从 GaussDB(DWS)集群导入数据到新集群

功能描述

通过在集群中创建 Foreign Table 的方式，实现在多个集群之间的关联查询和用来导入数据。

使用场景

- 将数据从一个 GaussDB(DWS)集群导入到另外一个 GaussDB(DWS)集群中。
- 多个集群之间的关联查询。

注意事项

- 两个集群必须在同一个 Region、一个 AZ 内且 VPC 网络互通。
- 创建的外表与其对应的远端表的列名和类型名要完全一致，且远端表的类型为行存表、列存表、哈希表或者复制表。
- 如果关联的表在另外一个集群是复制表或者存在数据倾斜，性能可能会很差。
- 使用期间，两个集群的状态应为“Normal”。
- 使用期间，禁止对远端集群的源数据表做 ddl 修改和增、删、改操作，否则可能导致查询结果不一致。
- 两个集群都需要具备基于 Foreign Table 的 SQL on other GaussDB 数据处理功能。
- 建议配置 LVS，如未配置，推荐使用多个 CN 作为 server 的地址，禁止将多个集群的 CN 地址写在一起。
- 请尽可能保证两端数据库的编码相同，否则可能出现报错或者收到的数据为乱码。

- 如果远端表已经做过统计信息收集，可以对外表执行 analyze 以获得更优的执行计划。
- 仅支持 8.0.0 及以上版本。

操作步骤

步骤 1　创建 server。

```
CREATE SERVER server_remote FOREIGN DATA WRAPPER GC_FDW OPTIONS
   (address '10.180.157.231:8000,10.180.157.130:8000' ,
  dbname 'gaussdb',
  username 'xyz',
  password 'xxxxxx'
);
```

说明

- server_remote 为 server 名字，供外表使用。
- address 为远端集群 CN 的地址和端口号，如配置 LVS，推荐只填写一个 LVS 地址，如未配置，推荐使用多个 CN 作为 server 的地址。
- dbname 为远端集群的数据库名。
- username 为连接远端集群使用的用户名，注意该用户不能为系统管理员。
- password 为连接远端集群使用的用户名的密码。

步骤 2　创建外表。

```
CREATE FOREIGN TABLE region
(
    R_REGIONKEY INT4,
    R_NAME TEXT,
    R_COMMENT TEXT
)
SERVER
    server_remote
OPTIONS
(
    schema_name 'test',
    table_name 'region',
    encoding 'gbk'
);
```

说明

- 外表的列不允许带任何约束。
- 外表的列名和列的类型要与远端集群对应的表的列名和列的类型完全一致。
- schema_name 为远端集群对应的表所在的 schema，如果该 option 省略，则 schema_name 预设该外表所在的 schema。
- table_name 为远端集群对应的表所在的表名，如果该 option 省略，则 table_name 预设该外表的表名。
- encoding 为远端集群的编码，如果该 option 省略，则编码使用远端集群数据库的默认编码。

步骤 3 查看建立的外表。

```
\d+ region

                        Foreign table "public.region"
   Column    |  Type   | Modifiers | FDW Options | Storage  | Stats target |
Description
-------------+---------+-----------+-------------+----------+--------------+-------
------
 r_regionkey | integer |           |             | plain    |              |
 r_name      | text    |           |             | extended |              |
 r_comment   | text    |           |             | extended |              |
Server: server_remote
FDW Options: (schema_name 'test', table_name 'region', encoding 'gbk')
FDW permition: read only
Has OIDs: no
Distribute By: ROUND ROBIN
Location Nodes: ALL DATANODES
```

步骤 4 查看建立的 server。

```
\des+ server_remote

List of foreign servers
    Name     |  Owner  | Foreign-data wrapper | Access privileges | Type | Version |
               FDW Options
| Description
---------------+---------+----------------------+-------------------+------+-------
--+-------------------------------------------------------------------
-------------------------------------------------------------------------------------
------------------------------+-------------
 server_remote | dbadmin | gc_fdw               |                   |      |         |
(address '10.180.157.231:8000,10.180.157.130:8000', dbname 'gaussdb'
, username 'xyz', password 'xxxxxx') |
(1 row)
```

步骤 5 使用外表进行导入数据或者关联查询。

- 导入数据。

  ```
  CREATE TABLE local_region
  (
      R_REGIONKEY INT4,
      R_NAME TEXT,
      R_COMMENT TEXT
  );
  INSERT INTO local_region SELECT * FROM region;
  ```

  说明

  - 如遇到报错连接失败，请检查 server 的信息确认两个集群是否已经相互连通。
  - 如遇到报错表不存在，请检查外表的 option 信息是否正确。
  - 如遇到报错列信息不匹配，请检查外表的列信息是否与远端集群对应表的列信息是否一致。
  - 如遇到报错版本不一致，请升级低版本的集群在继续使用。
  - 如遇到乱码，请检查数据源的实际编码方式，并重新创建外表指定正确的编码。

- 关联查询。

```
SELECT * FROM region, local_region WHERE local_region.R_NAME = region.R_NAME;
```

说明

- 外表可以当做一个本地表来使用，执行复杂的作业。
- 如果远端集群已经有统计信息，请对该外表执行 analyze 以获得更优的执行计划。
- 如果本地集群的 DN 数量比远端集群的 DN 数量少，本地集群需要使用 SMP 来获得更佳的性能。

步骤 6 删除外表。

```
DROP FOREIGN TABLE region;
```

----结束

3.2.5 基于 GDS 的跨集群互联互通

功能描述

在“基于 Foreign Table 的数据处理”的基础上，通过 GDS 进行数据中转，实现多个集群之间的数据同步。

使用场景

将数据从一个集群同步到另外一个集群，支持全量数据同步、过滤条件数据同步。

注意事项

- 创建的互联互通外表与其对应的远端表的列名和类型名要完全一致，且远端表的类型为行存表或列存表。
- 执行同步语句时，要确保本地集群、远端集群的待同步表已存在。
- 使用期间，两个集群的状态应为 Normal。
- 两个集群都需要具备基于 GDS 的跨集群互联互通功能。
- 建议两端集群的数据库编码保持一致，否则可能出现报错或者收到的数据为乱码。
- 两端集群所指定的数据库兼容类型要保持一致，否则可能报错或乱码。
- 确保执行数据同步的相关用户对待同步表有相应的访问权限。
- 互联互通外表只能用于跨集群数据同步场景，其他场景可能出错或无效。
- 互联互通外表不支持复杂的列上表达式，不支持复杂语法，包括 join、排序、游标、with、集合等。
- 不下推的 SQL 语句无法使用本特性进行数据同步，否则会报错。
- 不支持 EXPLAIN 计划、逻辑集群。
- 当本地集群同步数据到远端集群时，只支持内表查询。
- Foreign Server 的 syncsrv 选项指定的 GDS 不支持 SSL 模式。
- 数据同步结束时只校验数据行数，不校验数据内容。

- 业务最大并发数不能大于 GDS 启动参数-t 的一半，同时也不能大于 max_active_statements，否则可能会导致业务超时失败。

使用前准备

- 配置两个集群互连。
- 规划部署 GDS 服务器，确保所有的 GDS 服务器可以和上面配置的两个集群所有节点网络连通。部署 GDS 请参考 3.2.2.3 安装配置和启动 GDS。

操作步骤

假设远端集群的待同步表名称是 tbl_remote，用于数据同步的用户是 user_remote，该用户须对表 tbl_remote 有访问权限；假设本地集群的待同步表名称是 tbl_local。

步骤 1 创建 server。

```
CREATE SERVER server_remote FOREIGN DATA WRAPPER GC_FDW OPTIONS(
  address '100.185.178.207:8109',
  dbname 'db_remote',
  username 'user_remote',
  password 'xxxxxxxx',
  syncsrv 'gsfs://100.185.178.129:8789|gsfs://100.185.178.129:8790'
);
```

- server_remote 为 server 名称，供互联互通外表使用。
- address 为远端集群 CN 的 IP 地址和端口，仅允许填写一个地址。
- dbname 为远端集群的数据库名。
- username 为连接远端集群使用的用户名，注意该用户不能为系统管理员。
- password 为连接远端集群使用的用户名的密码。
- syncsrv 为 GDS Server 的 IP 地址和端口，如果有多个地址使用|分割，与 GDS 外表的 location 类似。

说明

GaussDB(DWS)会对 syncsrv 所设置的 GDS 地址进行网络连接测试：

- 只能判断本地执行集群与 GDS 的网络情况，无法判断远端集群与 GDS 的网络情况，需要注意报错提示。
- 在移除不可用 GDS 后，从中选择不会导致业务 hang 的、数目适当的 GDS 进行数据同步。

步骤 2 创建互联互通外表。

```
CREATE FOREIGN TABLE ft_tbl(
  col_1    type_name,
  col_2    type_name,
  …
) SERVER server_remote OPTIONS (
  schema_name 'schema_remote',
  table_name 'tbl_remote',
  encoding 'utf8'
);
```

- schema_name 为远端集群表所属 schema，如果该 option 缺省，则 schema_name 预设为该外表所在的 schema。
- table_name 为远端集群表名，如果该 option 缺省，则 table_name 预设为该外表的表名。
- encoding 为远端集群的编码，如果该 option 缺省，则编码使用本地集群数据库的默认编码。

说明

- 选项 schema_name、table_name 大小写敏感，必须与远端 schema、table 的名字大小写保持一致。
- 互联互通外表的列不允许带任何约束。
- 互联互通外表的列名、列类型必须与远端集群的表 tbl_remote 的列名和列类型完全一致。
- SERVER 须设置为步骤 1 中新建的 server，必须包含 syncsrv 属性。

步骤 3　使用互联互通外表进行数据同步。

- 本地集群是目标集群时，发起数据同步业务：

  全列全量数据同步：

  ```
  INSERT INTO tbl_local SELECT * FROM ft_tbl;
  ```

  全列过滤条件数据同步：

  ```
  INSERT INTO tbl_local SELECT * FROM ft_tbl WHERE col_2 = XX;
  ```

  部分列全量数据同步：

  ```
  INSERT INTO tbl_local (col_1) SELECT col_1 FROM ft_tbl;
  ```

  部分列过滤条件数据同步：

  ```
  INSERT INTO tbl_local (col_1) SELECT col_1 FROM ft_tbl WHERE col_2 = XX;
  ```

- 本地集群是源集群时，发起数据同步业务：

  单表数据同步：

  ```
  INSERT INTO ft_tbl SELECT * FROM tbl_local;
  ```

  join 结果集数据同步：

  ```
  INSERT INTO ft_tbl SELECT * FROM tbl_local1 join tbl_local2 ON XXX;
  ```

说明

- 如遇到报错连接失败，请检查 server 的信息确认两个集群是否已经相互连通。
- 如遇到报错 GDS 连接失败，请检查 syncsrv 指定的 GDS Server 是否都已经启动，且与两个集群所有节点可以网络连通。
- 如遇到报错表不存在，请检查外表的 option 信息是否正确。
- 如遇到报错列不存在，请检查外表的列名是否与源表一致。
- 如遇到报错列重复定义，请检查是否相应列名超长，若超长建议使用 AS 别名精简。
- 如遇到报错无法解析列类型，请检查语句中是否有列上表达式。
- 如遇到报错列信息不匹配，请检查外表的列信息是否与远端集群对应表的列信息是否一致。
- 如遇到报错语法不支持，请检查是否使用了 Join、distinct、排序等复杂用法。

- 如遇到乱码，请检查两端数据库的实际编码是否一致。
- 当本地集群是源集群时，存在极小的概率出现数据成功同步到远端集群，但是本地集群返回执行失败的情况，针对这种情况建议校验同步数据记录数。
- 当本地集群是源集群时，通过事务块、子事务等控制的数据同步，需要总事务提交后才能查询到数据同步结果。

步骤 4 删除互联互通外表。

```
DROP FOREIGN TABLE ft_tbl;
```

----结束

3.2.6 使用 gsql 元命令\COPY 导入数据

GaussDB(DWS)的 gsql 工具提供了元命令\copy 进行数据导入。

\copy 命令

\copy 命令格式以及说明参见表 3-12。

表3-12 \copy 元命令说明

语法	说明
\copy { table [(column_list)] | (query) } { from | to } { filename | stdin | stdout | pstdin | pstdout } [with] [binary] [oids] [delimiter [as] 'character'] [null [as] 'string'] [csv [header] [quote [as] 'character'] [escape [as] 'character'] [force quote column_list | *] [force not null column_list]]	在任何 gsql 客户端登录数据库成功后，可以使用该命令进行数据的导入/导出。但是与 SQL 的 COPY 命令不同，该命令读取/写入的文件是本地文件，而非数据库服务器端文件；所以，要操作的文件的可访问性、权限等，都是受限于本地用户的权限。 说明 \COPY 只适合小批量，格式良好的数据导入，容错能力较差。导入数据应优先选择 GDS 或 COPY。

参数说明

- table
  表的名字（可以有模式修饰）。
  取值范围：已存在的表名。
- column_list
  可选的待拷贝字段列表。
  取值范围：任意字段。如果没有声明字段列表，将使用所有字段。
- query

其结果将被拷贝。

取值范围：一个必须用圆括弧包围的 SELECT 或 VALUES 命令。

- filename

  文件的绝对路径。执行 copy 命令的用户必须有此路径的写权限。

- stdin

  声明输入是来自标准输入。

- stdout

  声明输出打印到标准输出。

- pstdin

  声明输入是来自 gsql 的标准输入。

- pstout
- 声明输出打印到 gsql 的标准输出。
- 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。

- oid

  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。

默认值：false

- delimiter [as] 'character'

  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： \.abcdefghijklmnopqrstuvwxyz0123456789。
- 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 E'\x07'，E'\x08'，E'\x1b'等。

取值范围：支持多字符分隔符，但分隔符不能超过 10 个字节。

默认值：

  - 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 [as] 'string'

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。

默认值：

- 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，则数据文本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。fileheader 是指定导出数据包含标题行的定义文件。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。

  默认值：false

- quote [as] 'character'

  CSV 格式文件下的引号字符。

  默认值：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 E'\x07'，E'\x08'，E'\x1b'等。

- escape [as] '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：双引号。当与 quote 值相同时，会被替换为'\0'。

- force quote column_list | *

  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_list

  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。

示例

创建目标表 copy_example。

```
create table copy_example
(
```

```
    col_1 integer,
    col_2 text,
    col_3 varchar(12),
    col_4 date,
    col_5 time
);
```

- 示例一：从 stdin 拷贝数据到目标表 copy_example。

  ```
  \copy copy_example from stdin csv;
  ```

  出现>>符号提示时，输入数据，输入\.时结束。

  ```
  Enter data to be copied followed by a newline.
  End with a backslash and a period on a line by itself.
  >> 1,"iamtext","iamvarchar",2006-07-07,12:00:00
  >> \.
  ```

- 示例二：在本地目录'/local/data/'下有 example.csv 文件，包含 header 行，使用'|'作为 delimiter，使用双引号作为 quote。内容如下：

  iamheader

  1|"iamtext"|"iamvarchar"|2006-07-07|12:00:00

  2|"iamtext"|"iamvarchar"|2022-07-07|19:00:02

  从本地文件 example.csv 导入数据到目标表 copy_example，header 选项为'on'，自动忽略第一行。quote 默认为双引号，因此可以不用指定。

  ```
  \copy copy_example from  '/local/data/example.csv' with(header 'on', format 'csv', delimiter '|', date_format 'yyyy-mm-dd',  time_format 'hh24:mi:ss');
  ```

- 示例三：在本地目录'/local/data/'下有 example.csv 文件，使用','作为 delimiter，使用双引号作为 quote，其中第一行缺少最后一个字段，第二行多一个字段。内容如下

  1,"iamtext","iamvarchar",2006-07-07

  2,"iamtext","iamvarchar",2022-07-07,19:00:02,12:00:00

  从本地文件 example.csv 导入数据到目标表 copy_example，delimiter 默认为','，因此可以不用指定，由于指定了容错参数 IGNORE_EXTRA_DATA 和 FILL_MISSING_FIELD，缺少的字段会用 NULL 替换，多出的字段被忽略。

  ```
  \copy copy_example from  '/local/data/example.csv' with( format 'csv', date_format 'yyyy-mm-dd',  time_format 'hh24:mi:ss',  IGNORE_EXTRA_DATA 'true', FILL_MISSING_FIELD 'true');
  ```

- 示例四：将 copy_example 表的内容导出到 stdout，格式为 csv，使用双引号作为 quote，第四列和第五列强制使用 quote 包围；

  ```
  \copy copy_example to stdout CSV quote as '"' force quote col_4,col_5;
  ```

3.2.7 使用 COPY FROM STDIN 导入数据

3.2.7.1 关于 COPY FROM STDIN 导入数据

这种方式适合数据写入量不太大， 并发度不太高的场景。

用户可以使用以下方式通过 COPY FROM STDIN 语句直接向 GaussDB(DWS)写入数据。

- 通过键盘输入向 GaussDB(DWS)写入数据。

- 通过 JDBC 驱动的 CopyManager 接口从文件或者数据库向 GaussDB(DWS)写入数据。此方法支持 COPY 语法中 copy option 的所有参数。

3.2.7.2 CopyManager 类简介

CopyManager 是 GaussDB(DWS) JDBC 驱动中提供的一个 API 接口类，用于批量向 GaussDB(DWS)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3-13 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导入数据。	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导入数据。	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导入数据。	SQLException,IOException
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导入数据。	SQLException,IOException

返回值	方法	描述	throws
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

3.2.7.3 示例：通过本地文件导入导出数据

在使用 JAVA 语言基于 GaussDB(DWS)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(DWS) jdbc 驱动。

```
//以下用例以 gsjdbc4.jar 为例。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://10.180.155.74:8000/gaussdb"); //数据库 URL
     String username = new String("jack");          //用户名
     String password = new String("********");       //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) {
 // TODO Auto-generated catch block
 e.printStackTrace();
  } catch (IOException e) {
 // TODO Auto-generated catch block
 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) {
 // TODO Auto-generated catch block
      e.printStackTrace();
 } catch (IOException e) {
 // TODO Auto-generated catch block
 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) {
 // TODO Auto-generated catch block
 e.printStackTrace();
  } catch (IOException e) {
 // TODO Auto-generated catch block
 e.printStackTrace();
 }
    }

 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();
          }
       }
    }
 }
```

```
  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();
           }
        }
     }
  }
}
```

3.2.7.4 示例：从 MySQL 向 GaussDB(DWS)进行数据迁移

下面示例演示如何通过 CopyManager 从 mysql 向 GaussDB(DWS)进行数据迁移的过程。

```
//以下用例以 gsjdbc4.jar 为例。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://10.180.155.74:8000/gaussdb"); //数
据库 URL
      String user = new String("jack");          //mppdb 用户名
      String pass = new String("********");           //mppdb 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();
            Connection conn =
DriverManager.getConnection("jdbc:mysql://10.119.179.227:3306/jack?useSSL=false&all
owPublicKeyRetrieval=true", "jack", "********");
            Statement stmt = conn.createStatement();
            rs = stmt.executeQuery("select * from migration table");
        } catch (SQLException e) {
```

```
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return rs;
    }
}
```

3.3 整库迁移

3.3.1 使用 CDM 迁移数据到 GaussDB(DWS)

使用云数据迁移服务（Cloud Data Migration，简称 CDM），可以将其他数据源（例如 MySQL）的数据迁移到 GaussDB(DWS) 集群的数据库中。

使用 CDM 迁移数据到 GaussDB(DWS) 的典型场景，请参见《云数据迁移服务用户指南》中的如下章节：

- 入门：该入门场景为使用 CDM 迁移本地 MySQL 数据库到 GaussDB(DWS)

3.3.2 使用 DSC 工具迁移 SQL 脚本

DSC（Database Schema Convertor）是一款运行在 Linux 或 Windows 操作系统上的命令行工具，致力于向客户提供简单、快速、可靠的应用程序 SQL 脚本迁移服务，通过内置的语法迁移逻辑解析源数据库应用程序 SQL 脚本，并迁移为适用于 GaussDB(DWS) 数据库的应用程序 SQL 脚本。DSC 不需要连接数据库，可在离线模式下实现零停机迁移。在 GaussDB(DWS) 中通过执行迁移后的 SQL 脚本即可恢复数据库，从而实现线下数据库轻松上云。

DSC 支持迁移 Teradata、Oracle、Netezza、MySQL 和 DB2 数据库的 SQL 脚本。

下载 DSC SQL 语法迁移工具

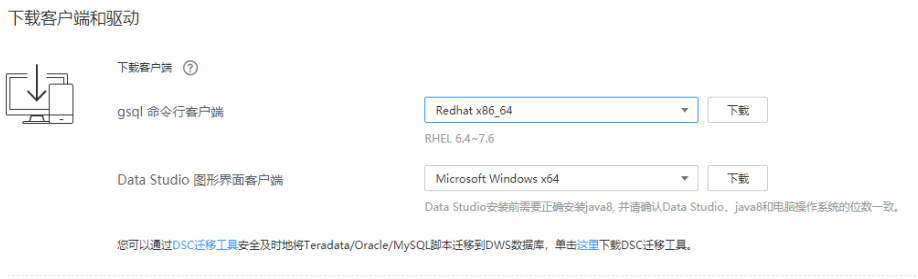
步骤 1 登录 GaussDB(DWS) 管理控制台。

步骤 2 在左侧导航栏中，单击“连接管理”。

步骤 3 在“下载客户端和驱动”区域，单击“这里”即可下载 DSC 迁移工具。

如果同时拥有不同版本的集群，系统会弹出对话框，提示您选择“集群版本”然后下载与集群版本相对应的客户端。在“集群管理”页面的集群列表中，单击指定集群的名称，再选择“基本信息”页签，可查看集群版本。

图3-7 下载工具



步骤 4 下载到本机后，使用 WinSCP 工具，将 DSC 工具上传到一个需安装工具的 Linux 主机上。

执行上传操作的用户需要对 Linux 主机的目标存放目录有完全控制权限。

----结束

DSC SQL 语法迁移工具操作指导

详细指导请参见《数据仓库服务工具指南》中的“DSC SQL 语法迁移工具”。

3.4 元数据迁移

3.4.1 使用 gs_dump 和 gs_dumpall 命令导出元数据

3.4.1.1 概述

GaussDB(DWS)提供的 gs_dump 和 gs_dumpall 工具，能够帮助用户导出需要的数据库对象或其相关信息。通过导入工具将导出的元数据信息导入至需要的数据库，可以完成数据库信息的迁移。gs_dump 支持导出单个数据库或其内的对象，而 gs_dumpall 支持导出集群中所有数据库或各库的公共全局对象。详细的使用场景见表 3-14。

表3-14 适用场景

适用场景	支持的导出粒度	支持的导出格式	配套的导入方法
导出单个数据库	数据库级导出。 • 导出全量信息。 使用导出的全量信息可以创建一个与当前库相同的数据库，且库中数据也与当前库相同。 • 仅导出库中所有对象的定义，包含库定义、函数定义、模式定义、表定义、索引定义和存	• 纯文本格式 • 自定义归档格式 • 目录归档格式 • tar 归档	• 纯文本格式数据文件导入请参见 3.2.6 使用 gsql 元命令 \COPY 导入数据。 • 自定义归档格式、目录归档格式和 tar 归档格式数据文件导入请参见 3.4.2

适用场景	支持的导出粒度	支持的导出格式	配套的导入方法
	储过程定义等。 使用导出的对象定义，可以快速创建一个相同的数据库，但是库中并无原数据库的数据。 • 仅导出数据。	格式	使用 gs_restore 导入数据。
	模式级导出。 • 导出模式的全量信息。 • 仅导出模式中数据。 • 仅导出对象的定义，包含表定义、存储过程定义和索引定义等。		
	表级导出。 • 导出表的全量信息。 • 仅导出表中数据。 • 仅导出表的定义。		
导出所有数据库	数据库级导出。 • 导出全量信息。 使用导出的全量信息可以创建与当前集群相同的一个集群，拥有相同数据库和公共全局对象，且库中数据也与当前各库相同。 • 仅导出各数据库中的对象定义，包含表空间、库定义、函数定义、模式定义、表定义、索引定义和存储过程定义等。 使用导出的对象定义，可以快速创建与当前集群相同的一个集群，拥有相同的数据库和表空间，但是库中并无原数据库的数据。 • 仅导出数据。	纯文本格式	数据文件导入请参见 3.2.6 使用 gsql 元命令 \COPY 导入数据。
	各库公共全局对象导出 • 仅导出表空间信息。 • 仅导出角色信息。 • 导出角色与表空间。		

gs_dump 和 gs_dumpall 通过-U 指定执行导出的用户帐户。如果当前使用的帐户不具备导出所要求的权限时，会无法导出数据。此时，可在导出命令中设置--role 参数来指定具备权限的角色。在执行命令后，gs_dump 和 gs_dumpall 会使用--role 参数指定的角色，完成导出动作。可使用该功能的场景请参见表 3-14，详细操作请参见 3.4.1.4 无权限角色导出数据。

gs_dump 和 gs_dumpall 通过对导出的数据文件加密，导入时对加密的数据文件进行解密，可以防止数据信息泄露，为数据库的安全提供保证。

gs_dump 和 gs_dumpall 工具在进行数据导出时，其他用户可以访问集群数据库（读或写）。

gs_dump 和 gs_dumpall 工具支持导出完整一致的数据。例如，T1 时刻启动 gs_dump 导出 A 数据库，或者启动 gs_dumpall 导出整个集群数据库，那么导出数据结果将会是 T1 时刻 A 数据库或者该集群数据库的数据状态，T1 时刻之后对 A 数据库或集群数据库的修改不会被导出。

gs_dump 和 gs_dumpall 工具是通过“gsql 命令行客户端”软件包解压缩获取。

注意事项

- 禁止修改导出的文件和内容，否则可能无法恢复成功。
- 为了保证数据一致性和完整性，导出工具会对需要转储的表设置共享锁。如果表在别的事务中设置了共享锁，gs_dump 和 gs_dumpall 会等待锁释放后锁定表。如果无法在指定时间内锁定某个表，转储会失败。用户可以通过指定--lock-wait-timeout 选项，自定义等待锁超时时间。
- 由于 gs_dumpall 读取所有数据库中的表，因此必须以数据库集群管理员身份进行连接，才能导出完整文件。在使用 gsql 执行脚本文件导入时，同样需要管理员权限，以便添加用户和组，以及创建数据库。
- 由于 GaussDB(DWS)数据库所有视图的定义都默认带有表名或别名的前缀（即 tab.col 的形式），因此可能与原始定义不符，导致在极少的场景下会发生重建视图字段对应基表不准确而报错的情况。为避免此情况建议在导出视图定义时，设置 guc 参数 behavior_compat_options='compat_display_ref_table'，使导出定义与原始语句一致。

3.4.1.2 导出单个数据库

3.4.1.2.1 导出数据库

GaussDB(DWS)支持使用 gs_dump 工具导出某个数据库级的内容，包含数据库的数据和所有对象定义。可根据需要自定义导出如下信息：

- 导出数据库全量信息，包含数据和所有对象定义。

  使用导出的全量信息可以创建一个与当前库相同的数据库，且库中数据也与当前库相同。
- 仅导出所有对象定义，包括：库定义、函数定义、模式定义、表定义、索引定义和存储过程定义等。

  使用导出的对象定义，可以快速创建一个相同的数据库，但是库中并无原数据库的数据。
- 仅导出数据，不包含所有对象定义。

操作步骤

步骤 1 准备 ECS 作为 gsql 客户端主机，具体操作请参见《数据仓库服务用户指南》中的“准备 ECS 作为 gsql 客户端主机”。

步骤 2 请参见《数据仓库服务用户指南》中的“下载客户端”下载 gsql 客户端，并使用 SSH 文件传输工具（例如 WinSCP 工具），将客户端工具上传到一个待安装 gsql 的 Linux 主机上。

执行上传 gsql 操作的用户需要对客户端主机的目标存放目录有完全控制权限。

步骤 3 执行以下命令解压客户端工具。

```
cd <客户端存放路径>
unzip dws_client_8.1.x_redhat_x64.zip
```

其中：

- <客户端存放路径>：请替换为实际的客户端存放路径。
- dws_client_8.1.x_redhat_x86.zip：这是“RedHat x86”对应的客户端工具包名称，请替换为实际下载的包名。

步骤 4 执行以下命令配置客户端。

```
source gsql_env.sh
```

提示以下信息表示客户端已配置成功

```
All things done.
```

步骤 5 使用 gs_dump 导出 gaussdb 数据库。

```
gs_dump -W password -U jack -f /home//backup/postgres_backup.tar -p 8000 gaussdb -h 10.10.10.100 -F t
```

表3-15 常用参数说明

参数	参数说明	举例
-U	连接数据库的用户名，如果未填写则表示当前已连接的数据库用户。	-U jack
-W	指定用户连接的密码。 • 如果主机的认证策略是 trust，则不会对数据库管理员进行密码验证，即无需输入-W 选项； • 如果没有-W 选项，并且不是数据库管理员，会提示用户输入密码。	-W *password*，此处密码需要用户自定义。
-f	将导出文件发送至指定目录文件夹。如果这里省略，则使用标准输出。	-f /home//backup/*postgres*_backup.tar
-p	指定服务器所监听的 TCP 端口或本	-p 8000

参数	参数说明	举例
	地 Unix 域套接字后缀，以确保连接。	
-h	“集群地址”如果通过公网地址连接，请指定为集群“公网访问地址”或“公网访问域名”，如果通过内网地址连接，请指定为集群“内网访问地址”或“内网访问域名”。	-h 10.10.10.100
dbname	需要导出的数据库名称	gaussdb
-F	选择导出文件格式。-F 参数值如下： • p：纯文本格式 • c：自定义归档 • d：目录归档格式 • t：tar 归档格式	-F t

其他参数说明请参见《工具指南》中“gs_dump”章节。

----结束

示例

示例一：执行 gs_dump，导出 gaussdb 数据库全量信息，并对导出文件进行压缩，导出文件格式为 sql 文本格式。

```
gs dump -W password -U jack -f /home//backup/postgres backup.sql -p 8000  -h
10.10.10.100 gaussdb -Z 8 -F p
gs_dump[port=''][gaussdb][2017-07-21 15:36:13]: dump database gaussdb successfully
gs_dump[port=''][gaussdb][2017-07-21 15:36:13]: total time: 3793  ms
```

示例二：执行 gs_dump，仅导出 gaussdb 数据库中的数据，不包含数据库对象定义，导出文件格式为自定义归档格式。

```
gs_dump -W Password -U jack -f /home//backup/postgres_data_backup.dmp -p 8000 -h
10.10.10.100 gaussdb -a -F c
gs_dump[port=''][gaussdb][2017-07-21 15:36:13]: dump database gaussdb successfully
gs_dump[port=''][gaussdb][2017-07-21 15:36:13]: total time: 3793  ms
```

示例三：执行 gs_dump，仅导出 gaussdb 数据库所有对象的定义，导出文件格式为 sql 文本格式。

```
--导出前，表 nation 有数据
select n_nationkey,n_name,n_regionkey from nation limit 3;
 n_nationkey |         n_name          | n_regionkey
-------------+-------------------------+-------------
           0 | ALGERIA                 |           0
```

```
         3 | CANADA                    |          1
        11 | IRAQ                      |          4
(3 rows)

gs_dump -W password -U jack -f /home//backup/postgres_def_backup.sql -p 8000 -h
10.10.10.100 gaussdb -s -F p
gs_dump[port=''][gaussdb][2017-07-20 15:04:14]: dump database gaussdb successfully
gs_dump[port=''][gaussdb][2017-07-20 15:04:14]: total time: 472 ms
```

示例四：执行 gs_dump，仅导出 gaussdb 数据库的所有对象的定义，导出文件格式为文本格式，并对导出文件进行加密。

```
gs dump -W password -U jack -f /home//backup/postgres def backup.sql -p 8000 -h
10.10.10.100 gaussdb --with-encryption AES128 --with-key 1234567812345678 -s -F p
gs dump[port=''][gaussdb][2018-11-14 11:25:18]: dump database gaussdb successfully
gs_dump[port=''][gaussdb][2018-11-14 11:25:18]: total time: 1161  ms
```

3.4.1.2.2 导出模式

GaussDB(DWS)目前支持使用 gs_dump 工具导出模式级的内容，包含模式的数据和定义。用户可通过灵活的自定义方式导出模式内容，不仅支持选定一个模式或多个模式的导出，还支持排除一个模式或者多个模式的导出。可根据需要自定义导出如下信息：

- 导出模式全量信息，包含数据和对象定义。
- 仅导出数据，即模式包含表中的数据，不包含对象定义。
- 仅导出模式对象定义，包括：表定义、存储过程定义和索引定义等。

操作步骤

步骤 1 准备 ECS 作为 gsql 客户端主机，具体操作请参见《数据仓库服务用户指南》中的“准备 ECS 作为 gsql 客户端主机”。

步骤 2 请参见《数据仓库服务用户指南》中的“下载客户端”下载 gsql 客户端，并使用 SSH 文件传输工具（例如 WinSCP 工具），将客户端工具上传到一个待安装 gsql 的 Linux 主机上。

执行上传 gsql 操作的用户需要对客户端主机的目标存放目录有完全控制权限。

步骤 3 执行以下命令解压客户端工具。

```
cd <客户端存放路径>
unzip dws_client_8.1.x_redhat_x64.zip
```

其中：

- <客户端存放路径>：请替换为实际的客户端存放路径。
- dws_client_8.1.x_redhat_x86.zip：这是“RedHat x86”对应的客户端工具包名称，请替换为实际下载的包名。

步骤 4 执行以下命令配置客户端。

```
source gsql_env.sh
```

提示以下信息表示客户端已配置成功

```
All things done.
```

步骤 5　使用 gs_dump 同时导出 hr 和 public 模式。

```
gs_dump -W Password -U jack -f /home//backup/MPPDB_schema_backup -p 8000 -h
10.10.10.100 human_resource -n hr -F d
```

表3-16 常用参数说明

参数	参数说明	举例
-U	连接数据库的用户名，如果未填写则表示当前已连接的数据库用户。	-U jack
-W	指定用户连接的密码。 • 如果主机的认证策略是 trust，则不会对数据库管理员进行密码验证，即无需输入-W 选项。 • 如果没有-W 选项，并且不是数据库管理员，会提示用户输入密码。	-W *password*，此处密码需要用户自定义。
-f	将导出文件发送至指定目录文件夹。如果这里省略，则使用标准输出。	-f /home//backup/MPPDB_schema_backup
-p	指定服务器所监听的 TCP 端口或本地 Unix 域套接字后缀，以确保连接。	-p 8000
-h	“集群地址”如果通过公网地址连接，请指定为集群“公网访问地址”或“公网访问域名”，如果通过内网地址连接，请指定为集群“内网访问地址”或“内网访问域名”。	-h 10.10.10.100
dbname	需要导出的数据库名称	human_resource
-n	只导出与模式名称匹配的模式，此选项包括模式本身和所有它包含的对象。 • 单个模式：-n *schemaname* • 多个模式：多次输入-n *schemaname*	• 单个模式：-n hr • 多个模式：-n hr -n public
-F	选择导出文件格式。-F 参数值如下： • p：纯文本格式 • c：自定义归档 • d：目录归档格式	-F d

参数	参数说明	举例
	• t：tar 归档格式	

其他参数说明请参见《工具指南》中“gs_dump”章节。

----结束

示例

示例一：执行 gs_dump，导出 hr 模式全量信息，并对导出文件进行压缩，导出文件格式为文本格式。

```
gs_dump -W password -U jack -f /home//backup/MPPDB_schema_backup.sql -p 8000 -h
10.10.10.100 human_resource -n hr -Z 6 -F p
gs_dump[port=''][human_resource][2017-07-21 16:05:55]: dump database human_resource
successfully
gs_dump[port=''][human_resource][2017-07-21 16:05:55]: total time: 2425  ms
```

示例二：执行 gs_dump，仅导出 hr 模式的数据，导出文件格式为 tar 归档格式。

```
gs_dump -W password -U jack -f /home//backup/MPPDB_schema_data_backup.tar -p 8000 -
h 10.10.10.100 human_resource -n hr -a -F t
gs_dump[port=''][human_resource][2018-11-14 15:07:16]: dump database human_resource
successfully
gs_dump[port=''][human_resource][2018-11-14 15:07:16]: total time: 1865  ms
```

示例三：执行 gs_dump，仅导出 hr 模式的定义，导出文件格式为目录归档格式。

```
gs_dump -W password -U jack -f /home//backup/MPPDB_schema_def_backup -p 8000 -h
10.10.10.100 human_resource -n hr -s -F d
gs_dump[port=''][human_resource][2018-11-14 15:11:34]: dump database human_resource
successfully
gs_dump[port=''][human_resource][2018-11-14 15:11:34]: total time: 1652  ms
```

示例四：执行 gs_dump，导出 human_resource 数据库时，排除 hr 模式，导出文件格式为自定义归档格式。

```
gs_dump -W password -U jack -f /home//backup/MPPDB_schema_backup.dmp -p 8000 -h
10.10.10.100 human_resource -N hr -F c
gs_dump[port=''][human_resource][2017-07-21 16:06:31]: dump database human_resource
successfully
gs_dump[port=''][human_resource][2017-07-21 16:06:31]: total time: 2522  ms
```

示例五：执行 gs_dump，同时导出 hr 和 public 模式，且仅导出模式定义，并对导出文件进行加密，导出文件格式为 tar 归档格式。

```
gs_dump -W password -U jack -f /home//backup/MPPDB_schema_backup1.tar -p 8000 -h
10.10.10.100 human_resource -n hr -n public -s --with-encryption AES128 --with-key
1234567812345678 -F t
gs_dump[port=''][human_resource][2017-07-21 16:07:16]: dump database human_resource
successfully
gs_dump[port=''][human_resource][2017-07-21 16:07:16]: total time: 2132  ms
```

示例六：执行 gs_dump，导出 human_resource 数据库时，排除 hr 和 public 模式，导出文件格式为自定义归档格式。

```
gs_dump -W password -U jack -f /home//backup/MPPDB_schema_backup2.dmp -p 8000 -h
10.10.10.100 human_resource -N hr -N public -F c
gs_dump[port=''][human_resource][2017-07-21 16:07:55]: dump database human_resource
successfully
gs_dump[port=''][human_resource][2017-07-21 16:07:55]: total time: 2296  ms
```

示例七：执行 gs_dump，导出 public 模式下所有表（视图、序列和外表）和 hr 模式中 staffs 表，包含数据和表定义，导出文件格式为自定义归档格式。

```
gs_dump -W password -U jack -f /home//backup/MPPDB_backup3.dmp -p 8000 -h
10.10.10.100 human_resource -t public.* -t hr.staffs -F c
gs_dump[port=''][human_resource][2018-12-13 09:40:24]: dump database human_resource
successfully
gs_dump[port=''][human_resource][2018-12-13 09:40:24]: total time: 896  ms
```

3.4.1.2.3 导出表

GaussDB(DWS)支持使用 gs_dump 工具导出表级的内容，包含表定义和表数据。视图、序列和外表属于特殊的表。用户可通过灵活的自定义方式导出表内容，不仅支持选定一个表或多个表的导出，还支持排除一个表或者多个表的导出。可根据需要自定义导出如下信息：

- 导出表全量信息，包含表数据和表定义。
- 仅导出数据，不包含表定义。
- 仅导出表定义。

操作步骤

步骤 1 准备 ECS 作为 gsql 客户端主机，具体操作请参见《数据仓库服务用户指南》中的“准备 ECS 作为 gsql 客户端主机”。

步骤 2 请参见《数据仓库服务用户指南》中的“下载客户端”下载 gsql 客户端，并使用 SSH 文件传输工具（例如 WinSCP 工具），将客户端工具上传到一个待安装 gsql 的 Linux 主机上。

执行上传 gsql 操作的用户需要对客户端主机的目标存放目录有完全控制权限。

步骤 3 执行以下命令解压客户端工具。

```
cd <客户端存放路径>
unzip dws_client_8.1.x_redhat_x64.zip
```

其中：

- <客户端存放路径>：请替换为实际的客户端存放路径。
- dws_client_8.1.x_redhat_x86.zip：这是“RedHat x86”对应的客户端工具包名称，请替换为实际下载的包名。

步骤 4 执行以下命令配置客户端。

```
source gsql_env.sh
```

提示以下信息表示客户端已配置成功

```
All things done.
```

步骤 5 使用 gs_dump 同时导出指定表 hr.staffs 和 hr.employments。

```
gs_dump -W password -U jack -f /home//backup/MPPDB table backup -p 8000 -h
10.10.10.100 human_resource -t hr.staffs -F d
```

表3-17 常用参数说明

参数	参数说明	举例
-U	连接数据库的用户名，如果未填写则表示当前已连接的数据库用户。	-U jack
-W	指定用户连接的密码。 • 如果主机的认证策略是 trust，则不会对数据库管理员进行密码验证，即无需输入-W 选项。 • 如果没有-W 选项，并且不是数据库管理员，会提示用户输入密码。	-W *password*
-f	将导出文件发送至指定目录文件夹。如果这里省略，则使用标准输出。	-f /home//backup/MPPDB_table_backup
-p	指定服务器所监听的 TCP 端口或本地 Unix 域套接字后缀，以确保连接。	-p 8000
-h	“集群地址”如果通过公网地址连接，请指定为集群“公网访问地址”或“公网访问域名”，如果通过内网地址连接，请指定为集群“内网访问地址”或“内网访问域名”。	-h 10.10.10.100
dbname	需要导出的数据库名称	human_resource
-t	指定导出的表（或视图、序列、外表），可以使用多个-t 选项来选择多个表，也可以使用通配符指定多个表对象。当使用通配符指定多个表对象时，注意给 pattern 打引号，防止 shell 扩展通配符。 • 单个表：-t *schema.table* • 多个表：多次输入-t *schema.table*	• 单个表：-t hr.staffs • 多个表：-t hr.staffs -t hr.employments
-F	选择导出文件格式。-F 参数值如下：	-F d

参数	参数说明	举例
	• p：纯文本格式 • c：自定义归档 • d：目录归档格式 • t：tar 归档格式	

其他参数说明请参见《工具指南》中“gs_dump”章节。

----结束

示例

示例一：执行 gs_dump，导出表 hr.staffs 的定义和数据，并对导出文件进行压缩，导出文件格式为文本格式。

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_backup.sql -p 8000 -h
10.10.10.100 human_resource -t hr.staffs -Z 6 -F p
gs_dump[port=''][human_resource][2017-07-21 17:05:10]: dump database human_resource
successfully
gs_dump[port=''][human_resource][2017-07-21 17:05:10]: total time: 3116  ms
```

示例二：执行 gs_dump，只导出表 hr.staffs 的数据，导出文件格式为 tar 归档格式。

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_data_backup.tar -p 8000 -h
10.10.10.100 human_resource -t hr.staffs -a -F t
gs_dump[port=''][human_resource][2017-07-21 17:04:26]: dump database human_resource
successfully
gs_dump[port=''][human_resource][2017-07-21 17:04:26]: total time: 2570  ms
```

示例三：执行 gs_dump，导出表 hr.staffs 的定义，导出文件格式为目录归档格式。

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_def_backup -p 8000 -h
10.10.10.100 human_resource -t hr.staffs -s -F d
gs_dump[port=''][human_resource][2017-07-21 17:03:09]: dump database human_resource
successfully
gs_dump[port=''][human_resource][2017-07-21 17:03:09]: total time: 2297  ms
```

示例四：执行 gs_dump，不导出表 hr.staffs，导出文件格式为自定义归档格式。

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_backup4.dmp -p 8000 -h
10.10.10.100 human_resource -T hr.staffs -F c
gs_dump[port=''][human_resource][2017-07-21 17:14:11]: dump database human_resource
successfully
gs_dump[port=''][human_resource][2017-07-21 17:14:11]: total time: 2450  ms
```

示例五：执行 gs_dump，同时导出两个表 hr.staffs 和 hr.employments，导出文件格式为文本格式。

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_backup1.sql -p 8000 -h
10.10.10.100 human_resource -t hr.staffs -t hr.employments -F p
gs_dump[port=''][human_resource][2017-07-21 17:19:42]: dump database human_resource
```

```
successfully
gs_dump[port=''][human_resource][2017-07-21 17:19:42]: total time: 2414  ms
```

示例六：执行 gs_dump，导出时，排除两个表 hr.staffs 和 hr.employments，导出文件格式为文本格式。

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_backup2.sql -p 8000 -h
10.10.10.100 human_resource -T hr.staffs -T hr.employments -F p
gs_dump[port=''][human_resource][2017-07-21 17:21:02]: dump database human_resource
successfully
gs_dump[port=''][human_resource][2017-07-21 17:21:02]: total time: 3165  ms
```

示例七：执行 gs_dump，导出表 hr.staffs 的定义和数据，只导出表 hr.employments 的定义，导出文件格式为 tar 归档格式。

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_backup3.tar -p 8000 -h
10.10.10.100 human_resource -t hr.staffs -t hr.employments --exclude-table-data
hr.employments -F t
gs_dump[port=''][human_resource][2018-11-14 11:32:02]: dump database human_resource
successfully
gs_dump[port=''][human_resource][2018-11-14 11:32:02]: total time: 1645  ms
```

示例八：执行 gs_dump，导出表 hr.staffs 的定义和数据，并对导出文件进行加密，导出文件格式为文本格式。

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_backup4.sql -p 8000 -h
10.10.10.100 human_resource -t hr.staffs --with-encryption AES128 --with-key
1212121212121212 -F p
gs_dump[port=''][human_resource][2018-11-14 11:35:30]: dump database human_resource
successfully
gs_dump[port=''][human_resource][2018-11-14 11:35:30]: total time: 6708  ms
```

示例九：执行 gs_dump，导出 public 模式下所有表（包括视图、序列和外表）和 hr 模式中 staffs 表，包含数据和表定义，导出文件格式为自定义归档格式。

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_backup5.dmp -p 8000 -h
10.10.10.100 human_resource -t public.* -t hr.staffs -F c
gs_dump[port=''][human_resource][2018-12-13 09:40:24]: dump database human_resource
successfully
gs_dump[port=''][human_resource][2018-12-13 09:40:24]: total time: 896  ms
```

示例十： 执行 gs_dump，仅导出依赖于 t1 模式下的 test1 表对象的视图信息，导出文件格式为目录归档格式。

```
gs_dump -W password -U jack -f /home//backup/MPPDB_view_backup6 -p 8000 -h
10.10.10.100 human_resource -t t1.test1 --include-depend-objs --exclude-self -F d
gs_dump[port=''][jack][2018-11-14 17:21:18]: dump database human_resource
successfully
gs_dump[port=''][jack][2018-11-14 17:21:23]: total time: 4239  ms
```

3.4.1.3 导出所有数据库

3.4.1.3.1 导出所有数据库

GaussDB(DWS)支持使用 gs_dumpall 工具导出所有数据库的全量信息，包含集群中每个数据库信息和公共的全局对象信息。可根据需要自定义导出如下信息：

- 导出所有数据库全量信息，包含集群中每个数据库信息和公共的全局对象信息（包含角色和表空间信息）。

  使用导出的全量信息可以创建与当前集群相同的一个集群，拥有相同数据库和公共全局对象，且库中数据也与当前各库相同。
- 仅导出数据，即导出每个数据库中的数据，且不包含所有对象定义和公共的全局对象信息。
- 仅导出所有对象定义，包括：表空间、库定义、函数定义、模式定义、表定义、索引定义和存储过程定义等。

  使用导出的对象定义，可以快速创建与当前集群相同的一个集群，拥有相同的数据库和表空间，但是库中并无原数据库的数据。

操作步骤

步骤 1 准备 ECS 作为 gsql 客户端主机，具体操作请参见《数据仓库服务用户指南》中的“准备 ECS 作为 gsql 客户端主机”。

步骤 2 请参见《数据仓库服务用户指南》中的“下载客户端”下载 gsql 客户端，并使用 SSH 文件传输工具（例如 WinSCP 工具），将客户端工具上传到一个待安装 gsql 的 Linux 主机上。

执行上传 gsql 操作的用户需要对客户端主机的目标存放目录有完全控制权限。

步骤 3 执行以下命令解压客户端工具。

```
cd <客户端存放路径>
unzip dws_client_8.1.x_redhat_x64.zip
```

其中：

- <客户端存放路径>：请替换为实际的客户端存放路径。
- dws_client_8.1.x_redhat_x86.zip：这是“RedHat x86”对应的客户端工具包名称，请替换为实际下载的包名。

步骤 4 执行以下命令配置客户端。

```
source gsql_env.sh
```

提示以下信息表示客户端已配置成功

```
All things done.
```

步骤 5 使用 gs_dumpall 一次导出所有数据库信息。

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_backup.sql -p 8000
-h 10.10.10.100
```

表3-18 常用参数说明

参数	参数说明	举例
-U	连接数据库的用户名，需要是集群管理员用户。	-U dbadmin

参数	参数说明	举例
-W	指定用户连接的密码。 • 如果主机的认证策略是 trust，则不会对数据库管理员进行密码验证，即无需输入-W 选项； • 如果没有-W 选项，并且不是数据库管理员，会提示用户输入密码。	-W *password*，此处密码需要用户自定义。
-f	将导出文件发送至指定目录文件夹。如果这里省略，则使用标准输出。	-f /home/dbadmin/backup/MPPDB_backup.sql
-p	指定服务器所监听的 TCP 端口或本地 Unix 域套接字后缀，以确保连接。	-p 8000
-h	“集群地址”如果通过公网地址连接，请指定为集群“公网访问地址”或“公网访问域名”，如果通过内网地址连接，请指定为集群“内网访问地址”或“内网访问域名”。	-h 10.10.10.100

其他参数说明请参见《工具指南》中“gs_dumpall”章节。

----结束

示例

示例一：执行 gs_dumpall，导出所有数据库全量信息（dbadmin 用户为管理员用户），导出文件为文本格式。执行命令后，会有很长的打印信息，最终出现 total time 即代表执行成功。示例中将不体现中间的打印信息。

```
gs dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB backup.sql -p 8000
-h 10.10.10.100
gs dumpall[port=''][2017-07-21 15:57:31]: dumpall operation successful
gs_dumpall[port=''][2017-07-21 15:57:31]: total time: 9627  ms
```

示例二：执行 gs_dumpall，仅导出所有数据库定义（dbadmin 用户为管理员用户），导出文件为文本格式。执行命令后，会有很长的打印信息，最终出现 total time 即代表执行成功。示例中将不体现中间的打印信息。

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_backup.sql -p 8000
-h 10.10.10.100 -s
gs_dumpall[port=''][2018-11-14 11:28:14]: dumpall operation successful
gs_dumpall[port=''][2018-11-14 11:28:14]: total time: 4147  ms
```

示例三：执行 gs_dumpall，仅导出所有数据库中数据，并对导出文件进行加密，导出文件为文本格式。执行命令后，会有很长的打印信息，最终出现 total time 即代表执行成功。示例中将不体现中间的打印信息。

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_backup.sql -p 8000
-h 10.10.10.100 -a --with-encryption AES128 --with-key 1234567812345678
gs_dumpall[port=''][2018-11-14 11:32:26]: dumpall operation successful
gs_dumpall[port=''][2018-11-14 11:23:26]: total time: 4147  ms
```

3.4.1.3.2 导出全局对象

GaussDB(DWS)支持使用 gs_dumpall 工具导出所有数据库公共的全局对象，包含数据库用户和组，表空间及属性（例如：适用于数据库整体的访问权限）信息。

操作步骤

步骤 1 准备 ECS 作为 gsql 客户端主机，具体操作请参见《数据仓库服务用户指南》中的“准备 ECS 作为 gsql 客户端主机”。

步骤 2 请参见《数据仓库服务用户指南》中的“下载客户端”下载 gsql 客户端，并使用 SSH 文件传输工具（例如 WinSCP 工具），将客户端工具上传到一个待安装 gsql 的 Linux 主机上。

执行上传 gsql 操作的用户需要对客户端主机的目标存放目录有完全控制权限。

步骤 3 执行以下命令解压客户端工具。

```
cd <客户端存放路径>
unzip dws_client_8.1.x_redhat_x64.zip
```

其中：

- <客户端存放路径>：请替换为实际的客户端存放路径。
- dws_client_8.1.x_redhat_x86.zip：这是“RedHat x86”对应的客户端工具包名称，请替换为实际下载的包名。

步骤 4 执行以下命令配置客户端。

```
source gsql_env.sh
```

提示以下信息表示客户端已配置成功

```
All things done.
```

步骤 5 使用 gs_dumpall 导出表空间对象信息。

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_tablespace.sql -p
8000 -h 10.10.10.100 -t
```

表3-19 常用参数说明

参数	参数说明	举例
-U	连接数据库的用户名，需要是集群管理员用户。	-U dbadmin

参数	参数说明	举例
-W	指定用户连接的密码。 • 如果主机的认证策略是 trust，则不会对数据库管理员进行密码验证，即无需输入-W 选项； • 如果没有-W 选项，并且不是数据库管理员，会提示用户输入密码。	--W *password*，此处密码需要用户自定义。
-f	将导出文件发送至指定目录文件夹。如果这里省略，则使用标准输出。	-f /home//backup/*MPPDB_tablespace*.sql
-p	指定服务器所监听的 TCP 端口或本地 Unix 域套接字后缀，以确保连接。	-p 8000
-h	“集群地址”如果通过公网地址连接，请指定为集群“公网访问地址”或“公网访问域名”，如果通过内网地址连接，请指定为集群“内网访问地址”或“内网访问域名”。	-h 10.10.10.100
-t	或者--tablespaces-only，只转储表空间，不转储数据库或角色。	-

其他参数说明请参见《工具指南》中“gs_dumpall”章节。

----结束

示例

示例一：执行 gs_dumpall，导出所有数据库的公共全局表空间信息和用户信息（dbadmin 用户为管理员用户），导出文件为文本格式。

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_globals.sql -p 8000
-h 10.10.10.100 -g
gs_dumpall[port=''][2018-11-14 19:06:24]: dumpall operation successful
gs_dumpall[port=''][2018-11-14 19:06:24]: total time: 1150  ms
```

示例二： 执行 gs_dumpall，导出所有数据库的公共全局表空间信息（dbadmin 用户为管理员用户），并对导出文件进行加密，导出文件为文本格式。

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_tablespace.sql -p
8000 -h 10.10.10.100 -t --with-encryption AES128 --with-key 1212121212121212
gs_dumpall[port=''][2018-11-14 19:00:58]: dumpall operation successful
gs_dumpall[port=''][2018-11-14 19:00:58]: total time: 186  ms
```

示例三：执行 gs_dumpall，导出所有数据库的公共全局用户信息（dbadmin 用户为管理员用户），导出文件为文本格式。

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_user.sql -p 8000 -h
10.10.10.100 -r
gs_dumpall[port=''][2018-11-14 19:03:18]: dumpall operation successful
gs_dumpall[port=''][2018-11-14 19:03:18]: total time: 162  ms
```

3.4.1.4 无权限角色导出数据

gs_dump 和 gs_dumpall 通过-U 指定执行导出的用户帐户。如果当前使用的帐户不具备导出所要求的权限时，会无法导出数据。此时，可在导出命令中设置--role 参数来指定具备权限的角色。在执行命令后，gs_dump 和 gs_dumpall 会使用--role 参数指定的角色，完成导出动作。

操作步骤

步骤 1 准备 ECS 作为 gsql 客户端主机，具体操作请参见《数据仓库服务用户指南》中的“准备 ECS 作为 gsql 客户端主机”。

步骤 2 请参见《数据仓库服务用户指南》中的“下载客户端”下载 gsql 客户端，并使用 SSH 文件传输工具（例如 WinSCP 工具），将客户端工具上传到一个待安装 gsql 的 Linux 主机上。

执行上传 gsql 操作的用户需要对客户端主机的目标存放目录有完全控制权限。

步骤 3 执行以下命令解压客户端工具。

```
cd <客户端存放路径>
unzip dws_client_8.1.x_redhat_x64.zip
```

其中：

- <客户端存放路径>：请替换为实际的客户端存放路径。
- dws_client_8.1.x_redhat_x86.zip：这是“RedHat x86”对应的客户端工具包名称，请替换为实际下载的包名。

步骤 4 执行以下命令配置客户端。

```
source gsql_env.sh
```

提示以下信息表示客户端已配置成功

```
All things done.
```

步骤 5 使用 gs_dump 导出 human_resource 数据库数据。

用户 jack 不具备导出数据库 human_resource 的权限，而角色 role1 具备该权限，要实现导出数据库 human_resource，可以在导出命令中设置--role 角色为 role1，使用 role1 的权限，完成导出目的。导出文件格式为 tar 归档格式。

```
gs_dump -U jack -W password -f /home//backup/MPPDB_backup.tar -p 8000 -h
10.10.10.100 human_resource --role role1 --rolepassword password -F t
```

表3-20 常用参数说明

参数	参数说明	举例 dbadmin
-U	连接数据库的用户名。	-U jack
-W	指定用户连接的密码。 • 如果主机的认证策略是 trust，则不会对数据库管理员进行密码验证，即无需输入-W 选项。 • 如果没有-W 选项，并且不是数据库管理员，会提示用户输入密码。	-W *password*，此处密码需要用户自定义。
-f	将导出文件发送至指定目录文件夹。如果这里省略，则使用标准输出。	-f /home//backup/MPPDB_backup.tar
-p	指定服务器所监听的 TCP 端口或本地 Unix 域套接字后缀，以确保连接。	-p 8000
-h	“集群地址”如果通过公网地址连接，请指定为集群“公网访问地址”或“公网访问域名”，如果通过内网地址连接，请指定为集群“内网访问地址”或“内网访问域名”。	-h 10.10.10.100
dbname	需要导出的数据库名称	human_resource
--role	指定导出使用的角色名。选择该选项，会使导出工具连接数据库后，发起一个 SET ROLE 角色名命令。当所授权用户（由-U 指定）没有导出工具要求的权限时，该选项会起到作用，即切换到具备相应权限的角色。	-r role1
--rolepassword	指定具体角色用户的角色密码。	--rolepassword password
-F	选择导出文件格式。-F 参数值如下： • p：纯文本格式 • c：自定义归档 • d：目录归档格式 • t：tar 归档格式	-F t

其他参数说明请参见《工具指南》中“gs_dump”或“gs_dumpall”章节。

----结束

示例

示例一：执行 gs_dump 导出数据，用户 jack 不具备导出数据库 human_resource 的权限，而角色 role1 具备该权限，要实现导出数据库 human_resource，可以在导出命令中设置--role 角色为 role1，使用 role1 的权限，完成导出目的。导出文件格式为 tar 归档格式。

```
human resource=# CREATE USER jack IDENTIFIED BY "password";

gs_dump -U jack -W password -f /home//backup/MPPDB_backup11.tar -p 8000 -h
10.10.10.100 human_resource --role role1 --rolepassword password -F t
gs_dump[port='8000'][human_resource][2017-07-21 16:21:10]: dump database
human_resource successfully
gs_dump[port='8000'][human_resource][2017-07-21 16:21:10]: total time: 4239  ms
```

示例二：执行 gs_dump 导出数据，用户 jack 不具备导出模式 public 的权限，而角色 role1 具备该权限，要实现导出模式 public，可以在导出命令中设置--role 角色为 role1，使用 role1 的权限，完成导出目的。导出文件格式为 tar 归档格式。

```
human_resource=# CREATE USER jack IDENTIFIED BY "1234@abc";

gs_dump -U jack -W password -f /home//backup/MPPDB_backup12.tar -p 8000 -h
10.10.10.100 human_resource -n public --role role1 --rolepassword password -F t
gs_dump[port='8000'][human_resource][2017-07-21 16:21:10]: dump database
human_resource successfully
gs_dump[port='8000'][human_resource][2017-07-21 16:21:10]: total time: 3278  ms
```

示例三：执行 gs_dumpall 导出数据，用户 jack 不具备导出所有数据库的权限，而角色 role1 具备该权限，要实现导出所有数据库，可以在导出命令中设置--role 角色为 role1，使用 role1 的权限，完成导出目的。导出文件格式为文本归档格式。

```
human_resource=# CREATE USER jack IDENTIFIED BY "password";

gs_dumpall -U jack -W password -f /home//backup/MPPDB_backup.sql -p 8000 -h
10.10.10.100 --role role1 --rolepassword password
gs_dumpall[port='8000'][human_resource][2018-11-14 17:26:18]: dumpall operation
successful
gs_dumpall[port='8000'][human_resource][2018-11-14 17:26:18]: total time: 6437  ms
```

3.4.2 使用 gs_restore 导入数据

操作场景

gs_restore 是 GaussDB(DWS)提供的与 gs_dump 配套的导入工具。通过该工具，可将 gs_dump 导出的文件导入至数据库。gs_restore 支持导入的文件格式包含自定义归档格式、目录归档格式和 tar 归档格式。

gs_restore 具备如下两种功能。

- 导入至数据库

如果指定了数据库，则数据将被导入到指定的数据库中。其中，并行导入必须指定连接数据库的密码。

- 导入至脚本文件

  如果未指定导入数据库，则创建包含重建数据库所需的 SQL 语句脚本，并将其写入至文件或者标准输出。该脚本文件等效于 gs_dump 导出的纯文本格式文件。

gs_restore 工具在导入时，允许用户选择需要导入的内容，并支持在数据导入前对等待导入的内容进行排序。

操作步骤

说明

gs_restore 默认是以追加的方式进行数据导入。为避免多次导入造成数据异常，在进行导入时，建议使用"-e"和"-c" 参数，即导入前删除已存在于待导入数据库中的数据库对象，同时当出现导入错误时，忽略当前错误，继续执行导入任务，并在导入后会显示相应的错误信息。

步骤 1 以 root 用户登录到服务器，执行如下命令进入数据存放路径。

```
cd /opt/bin
```

步骤 2 使用 gs_restore 命令，从 postgres 整个数据库内容的导出文件中，将数据库的所有对象的定义导入到 backupdb。

```
gs_restore -W password -U jack /home//backup/MPPDB_backup.tar -p 8000 -h
10.10.10.100 -d backupdb -s -e -c
```

表3-21 常用参数说明

参数	参数说明	举例
-U	连接数据库的用户名。	-U jack
-W	指定用户连接的密码。 • 如果主机的认证策略是 trust，则不会对数据库管理员进行密码验证，即无需输入-W 选项； • 如果没有-W 选项，并且不是数据库管理员，会提示用户输入密码。	-W *password*，此处密码需要用户自定义。
-d	连接数据库 dbname，并直接将数据导入到该数据库中。	-d backupdb
-p	指定服务器所监听的 TCP 端口或本地 Unix 域套接字后缀，以确保连接。	-p 8000
-h	“集群地址”如果通过公网地址连接，请指定为集群“公网访问地址”或“公网访问域名”，如果通过内网地址连接，请指定为集群	-h 10.10.10.100

参数	参数说明	举例
	“内网访问地址”或“内网访问域名”。	
-e	当发送 SQL 语句到数据库时如果出现错误，退出当前出现错误的任务，并执行其他导入任务。即默认状态下会忽略错误任务并继续执行导入，且在导入后会显示一系列错误信息。	-
-c	在重新创建数据库对象前，清理（删除）已存在于将要导入的数据库中的数据库对象。	-
-s	只导入模式定义，不导入数据。当前的序列值也不会被导入。	-

其他参数说明请参见《工具参考》中“服务端工具>gs_restore”章节。

----结束

示例

示例一：执行 gs_restore，导入指定 MPPDB_backup.dmp 文件（自定义归档格式）中 postgres 数据库的数据和对象定义。

```
gs_restore -W password backup/MPPDB_backup.dmp -p 8000 -h 10.10.10.100 -d backupdb
gs_restore[2017-07-21 19:16:26]: restore operation successfu
gs_restore: total time: 13053  ms
```

示例二：执行 gs_restore，导入指定 MPPDB_backup.tar 文件（tar 归档格式）中 postgres 数据库的数据和对象定义。

```
gs_restore backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100 -d backupdb
gs_restore[2017-07-21 19:21:32]: restore operation successful
gs_restore[2017-07-21 19:21:32]: total time: 21203  ms
```

示例三：执行 gs_restore，导入指定 MPPDB_backup 目录文件（目录归档格式）中 postgres 数据库的数据和对象定义。

```
gs_restore backup/MPPDB_backup -p 8000 -h 10.10.10.100 -d backupdb
gs_restore[2017-07-21 19:26:46]: restore operation successful
gs_restore[2017-07-21 19:26:46]: total time: 21003  ms
```

示例四：执行 gs_restore，将 postgres 数据库的所有对象的定义导入至 backupdb 数据库。导入前，postgres 存在完整的定义和数据，导入后，backupdb 数据库只存在所有对象定义，表没有数据。

```
gs_restore -W password /home//backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100 -d
backupdb -s -e -c
```

```
gs_restore[2017-07-21 19:46:27]: restore operation successful
gs_restore[2017-07-21 19:46:27]: total time: 32993  ms
```

示例五：执行 gs_restore，导入 MPPDB_backup.dmp 文件中 PUBLIC 模式的所有定义和数据。在导入时会先删除已经存在的对象，如果原对象存在跨模式的依赖则需手工强制干预。

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -h 10.10.10.100 -d backupdb -e -c -n
PUBLIC
gs_restore: [archiver (db)] Error while PROCESSING TOC:
gs_restore: [archiver (db)] Error from TOC entry 313; 1259 337399 TABLE table1
gaussdba
gs_restore: [archiver (db)] could not execute query: ERROR:  cannot drop table
table1 because other objects depend on it
DETAIL:  view t1.v1 depends on table table1
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
Command was: DROP TABLE public.table1;
```

手工删除依赖，导入完成后再重新创建。

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -h 10.10.10.100 -d backupdb -e -c -n
PUBLIC
gs_restore[2017-07-21 19:52:26]: restore operation successful
gs_restore[2017-07-21 19:52:26]: total time: 2203  ms
```

示例六：执行 gs_restore，导入 MPPDB_backup.dmp 文件中 PUBLIC 模式下表 hr.staffs 的定义。在导入之前，hr.staffs 表不存在。

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -h 10.10.10.100 -d backupdb -e -c -s -n
PUBLIC -t hr.staffs
gs_restore[2017-07-21 19:56:29]: restore operation successful
gs_restore[2017-07-21 19:56:29]: total time: 21000  ms
```

示例七：执行 gs_restore，导入 MPPDB_backup.dmp 文件中 PUBLIC 模式下表 hr.staffs 的数据。在导入之前，hr.staffs 表不存在数据。

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -h 10.10.10.100 -d backupdb -e -a -n
PUBLIC -t hr.staffs
gs_restore[2017-07-21 20:12:32]: restore operation successful
gs_restore[2017-07-21 20:12:32]: total time: 20203  ms
```

示例八：执行 gs_restore，导入指定表 hr.staffs 的定义。在导入之前，hr.staffs 表的数据是存在的。

```
human_resource=# select * from hr.staffs;
 staff_id | first_name  | last_name  | email   |    phone_number    |     hire_date
| employment_id |  salary  | commission_pct | manager_id | section_id
----------+-------------+------------+----------+--------------------+------------
---------+---------------+----------+----------------+------------+------------
     200 | Jennifer    | Whalen     | JWHALEN  | 515.123.4444       | 1987-09-17
00:00:00 | AD_ASST       |  4400.00 |              |       101 |        10
     201 | Michael     | Hartstein   | MHARTSTE | 515.123.5555       | 1996-02-17
00:00:00 | MK_MAN        | 13000.00 |              |       100 |        20

gsql -d human_resource -p 8000
gsql ((GaussDB 8.1.3 build 39137c2d) compiled at 2022-04-01 15:43:11 commit 3629
last mr 5138 release)
```

```
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

human_resource=# drop table hr.staffs CASCADE;
NOTICE:  drop cascades to view hr.staff_details_view

gs_restore -W password /home//backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100-d
human_resource -n hr -t staffs -s -e
restore operation successful
total time: 904  ms

human_resource=# select * from hr.staffs;
 staff_id | first_name | last_name | email | phone_number | hire_date |
employment_id | salary | commission_pct | manager_id | section_id
----------+------------+-----------+-------+--------------+-----------+------------
---+--------+----------------+------------+------------
(0 rows)
```

示例九：执行 gs_restore，导入 staffs 和 areas 两个指定表的定义和数据。在导入之前，staffs 和 areas 表不存在。

```
human_resource=# \d
                          List of relations
 Schema |       Name       | Type  |  Owner   |           Storage
--------+------------------+-------+----------+-------------------------------
 hr     | employment_history | table |  | {orientation=row,compression=no}
 hr     | employments       | table |  | {orientation=row,compression=no}
 hr     | places           | table |  | {orientation=row,compression=no}
 hr     | sections         | table |  | {orientation=row,compression=no}
 hr     | states           | table |  | {orientation=row,compression=no}
(5 rows)

gs_restore -W password /home/mppdb/backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100
-d human_resource -n hr -t staffs -n hr -t areas
restore operation successful
total time: 724  ms

human_resource=# \d
                          List of relations
 Schema |       Name       | Type  |  Owner   |           Storage
--------+------------------+-------+----------+-------------------------------
 hr     | areas            | table |  | {orientation=row,compression=no}
 hr     | employment_history | table |  | {orientation=row,compression=no}
 hr     | employments       | table |  | {orientation=row,compression=no}
 hr     | places           | table |  | {orientation=row,compression=no}
 hr     | sections         | table |  | {orientation=row,compression=no}
 hr     | staffs           | table |  | {orientation=row,compression=no}
 hr     | states           | table |  | {orientation=row,compression=no}
(7 rows)

human_resource=# select * from hr.areas;
 area_id |      area_name
---------+------------------------
       4 | Iron
       1 | Wood
       2 | Lake
```

```
      3 | Desert
(4 rows)
```

示例十：执行 gs_restore，导入 hr 的模式，包含模式下的所有对象定义和数据。

```
gs_restore -W password /home//backup/MPPDB_backup1.sql -p 8000 -h 10.10.10.100 -d
backupdb -n hr -e -c
restore operation successful
total time: 702  ms
```

示例十一：执行 gs_restore，同时导入 hr 和 hr1 两个模式，仅导入模式下的所有对象定义。

```
gs_restore -W password /home//backup/MPPDB_backup2.dmp -p 8000 -h 10.10.10.100 -d
backupdb -n hr -n hr1 -s
restore operation successful
total time: 665  ms
```

示例十二：执行 gs_restore，将 human_resource 数据库导出文件进行解密并导入至 backupdb 数据库中。

```
create database backupdb;


gs_restore /home//backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100 -d backupdb --
with-key=1234567812345678
restore operation successful
total time: 23472  ms

gsql -d backupdb -p 8000 -r
gsql ((GaussDB 8.1.3 build 39137c2d) compiled at 2022-04-01 15:43:11 commit 3629
last mr 5138 release)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

backupdb=# select * from hr.areas;
 area_id |      area_name
---------+------------------------
      4 | Iron
      1 | Wood
      2 | Lake
      3 | Desert
(4 rows)
```

示例十三：用户 user1 不具备将导出文件中数据导入至数据库 backupdb 的权限，而角色 role1 具备该权限，要实现将文件数据导入数据库 backupdb，可以在导出命令中设置 --role 角色为 role1，使用 role1 的权限，完成导出目的。

```
human resource=# CREATE USER user1 IDENTIFIED BY 'password';

gs_restore -U user1 -W password /home//backup/MPPDB_backup.tar -p 8000 -h
10.10.10.100 -d backupdb --role role1 --rolepassword password
restore operation successful
total time: 554  ms

gsql -d backupdb -p 8000 -r
gsql ((GaussDB 8.1.3 build 39137c2d) compiled at 2022-04-01 15:43:11 commit 3629
```

```
last mr 5138 release)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

backupdb=# select * from hr.areas;
 area_id |      area_name
---------+------------------------
       4 | Iron
       1 | Wood
       2 | Lake
       3 | Desert
(4 rows)
```

3.5 导出数据

3.5.1 导出数据到 OBS

3.5.1.1 关于 OBS 并行导出

概述

GaussDB(DWS)数据库支持通过 OBS 外表并行导出数据：通过 OBS 外表设置的导出模式、导出数据格式等信息来指定导出的数据文件，利用多 DN 并行的方式，将数据从 GaussDB(DWS)数据库导出到外部，存放在 OBS 对象存储服务器上，从而提高整体导出性能。

- CN 只负责任务的规划及下发，把数据导出的工作交给了 DN，释放了 CN 的资源，使其有能力处理外部请求。
- 通过让各个 DN 都参与数据导出，充分利用各个设备的计算能力及网络带宽。
- 支持多个 OBS 服务并发导出，导出的桶和对象的路径必须不同并且不能为空。
- 选择 OBS 服务器与集群节点处于联网状态，导出速率会受网络带宽影响。
- 支持数据文件格式：TEXT、CSV。单行数据大小需<1GB。
- ORC 格式的数据仅 8.1.0 版本以后支持。

相关概念

- **数据源文件**：存储有数据的 TEXT、CSV 文件。
- **OBS**：对象存储服务，是一种可存储文档、图片、影音视频等非结构化数据的云存储服务。从 GaussDB(DWS)并行导出数据时，数据对象放置在 OBS 服务器上。
- **桶（Bucket）**：对 OBS 中的一个存储空间的形象称呼，是存储对象的容器。
  - 对象存储是一种非常扁平化的存储方式，桶中存储的对象都在同一个逻辑层级，去除了文件系统中的多层级树形目录结构。
  - 在 OBS 中，桶名必须是全局唯一的且不能修改，即用户创建的桶不能与自己已创建的其他桶名称相同，也不能与其他用户创建的桶名称相同。每个桶在创建时都会生成默认的桶 ACL（Access Control List），桶 ACL 列表的每项包含了对被授权用户授予什么样的权限，如读取权限、写入权限、完全控制权

限等。用户只有对桶有相应的权限，才可以对桶进行操作，如创建、删除、显示、设置桶 ACL 等。

- 一个用户最多可创建 100 个桶，但每个桶中存放的总数据容量和对象/文件数量没有限制。

- **对象**：是存储在 OBS 中的基本数据单位。用户上传的数据以对象的形式存储在 OBS 的桶中。对象的属性包括名称 Key，Metadata，Data。

  通常，将对象等同于文件来进行管理，但是由于 OBS 是一种对象存储服务，并没有文件系统中的文件和文件夹概念。为了使用户更方便进行管理数据，OBS 提供了一种方式模拟文件夹。通过在对象的名称中增加“/”，如 tpcds1000/stock.csv，tpcds1000 可以等同于文件夹，stock.csv 就可以等同于文件名，而对象名称（key）仍然是 tpcds1000/stock.csv、对象的内容就是 stock.csv 数据文件的内容。

- **Key**：对象的名称（键），为经过 UTF-8 编码的长度大于 0 且不超过 1024 的字符序列，一个桶里的每个对象必须拥有唯一的对象键值。用户可使用桶名+对象名来存储和获取对应的对象。

- **Metadata**：对象元数据，用来描述对象的信息。元数据又可分为系统元数据和用户元数据。这些元数据以键值对（Key-value）的形式随 http 头域一起上传到 OBS 系统。

  - 系统元数据由 OBS 系统产生，在处理对象数据时使用。系统元数据包括：Date, Content-length, last-modify, Content-MD5 等。
  - 用户元数据由用户上传对象时指定，是用户自己对对象的一些描述信息。

- **Data**：对象的数据内容，OBS 对于数据的内容是无感知的，即认为对象内的数据为无状态的二进制数据。

- **外表**：用于识别数据源文件中的数据。外表中保存了数据源文件的位置、文件格式、存放位置、编码格式、数据间的分隔符等信息。

相关原理

下面分别从以下两类表介绍从集群导出数据到 OBS 的原理。

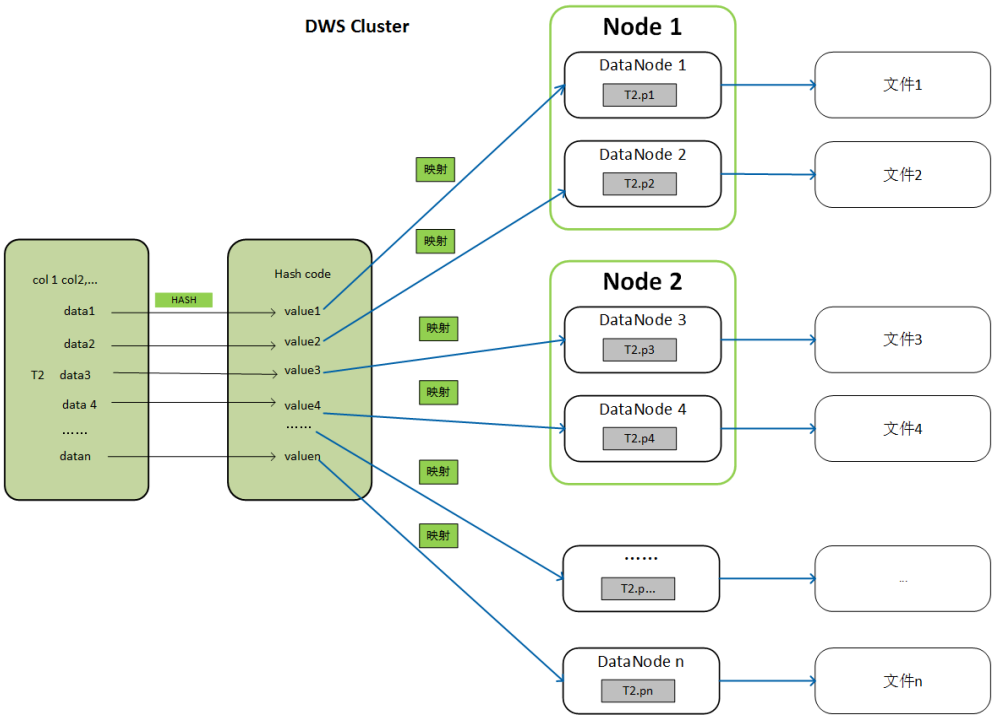
- Hash 分布表：在建表语句中指定了 DISTRIBUTE BY HASH (Column_Name)的表。

  对于 Hash 分布表而言，在存储表数据时，采用的是散列（Hash）方式的存储原理，如图 3-8 所示，图中以将表（T2）导出到 OBS 为例。

  在存储表数据时，将表（T2）中指定的 Hash 字段（col2）进行 Hash 运算后，生成相应的 Hash 值（value），根据 DN 与 Hash 值的映射关系，将该元组分发到相应的 DN 上进行存储。

  在导出数据到 OBS 时，每一个存储了导出表的（T2）数据的 DN 会直接向 OBS 导出属于自己的数据文件。多个节点将并行导出原始数据。

图3-8 Hash 分布原理



- Replication 表：在建表语句中指定了 DISTRIBUTE BY REPLICATION 的表。

  Replication 表在 GaussDB(DWS)集群的每个节点上都会存储一份完整的表数据。因此，在导出数据到 OBS 时，GaussDB(DWS)只会随机选择一个 DN 节点向 OBS 导出数据。

导出文件的命名规则

GaussDB(DWS)向 OBS 导出数据的文件命名规则如下：

- 从 DN 节点导出数据时，以 segment 的格式存储在 OBS 服务中，文件命名规则为“**表名称_节点名称_segment.n**”。这里的“n”是从 0 开始按照自然数 0、1、2、3 递增。

  例如，表 t1 在 datanode3 里面的数据导出成文件“t1_datanode3_segment.0”、“t1_datanode3_segment.1”等等，以此类推。

  对于来自不同集群或不同数据库的数据，建议用户可以将数据导出到不同的 OBS 桶或者同一个 OBS 桶的不同路径下。

- 每个 segment 可以存储的最大数据为 1GB，并且不能切断元组。如果 segment 超过 1GB，超过 1GB 的数据会作为第二个 segment 进行存储。

  例如：

  datanode3 节点将表（t1）导出到 OBS 时，一个 segment 里面已经存储了 100 条元组，大小是 1023MB，如果再插入一条 5MB 的元组，大小就变成 1028MB 了，此时会以 1023MB 生成一个“t1_datanode3_segment.0”保存到 OBS 服务中，新插入的第 101 条元组作为下一个“t1_datanode3_segment.1”保存到 OBS 服务中。

- 导出 Hash 分布表时，每个 DataNode 节点生成的 segment 数量和集群的 DataNode 节点数无关，而是取决于每个 DataNode 节点上存储的数据量。按照 Hash 方式存储在各个 DataNode 节点上的数据分布不一定均匀。

  例如，一个有 6 个 DataNode 节点的集群，DataNode1 到 DataNode6 分别有 1.5GB、0.7GB、0.6GB、0.8GB、0.4GB、0.5GB 的数据，则导出时会生成 7 个 OBS segment 文件，其中 DataNode1 会生成 1GB 和 0.5GB 两个 segment 文件。

导出流程

图3-9 并行导出流程

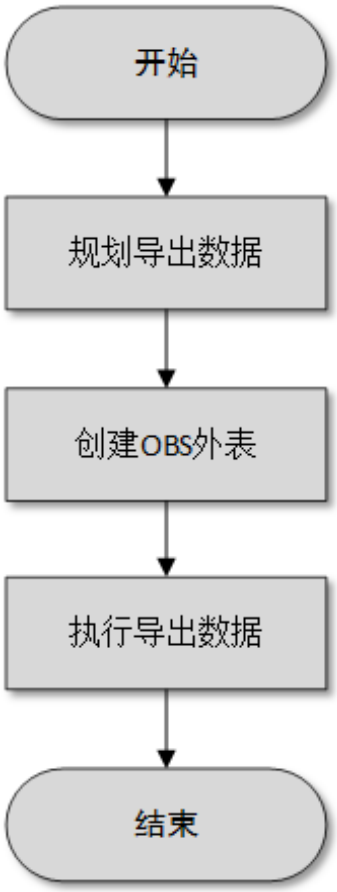


表3-22 流程说明

流程	说明	子任务
规划导出数据	创建 OBS 桶，并在桶中创建导出后的数据文件的存放目录。 详细请参见 3.5.1.2.1 规划导出数据。	-
创建 OBS 外	创建外表用于帮助 OBS 指定的	-

流程	说明	子任务
表。	待导出数据文件。外表中保存了数据源文件导出后的位置、文件格式、编码格式、数据间的分隔符等信息。 详细内容请参见 3.5.1.2.2 创建 OBS 外表。	
执行导出数据。	在创建好外表后，通过 INSERT 语句，将数据快速、高效地导出到数据文件中。 详细内容请参见 3.5.1.2.3 执行导出。	-

3.5.1.2 导出 CSV,TXT 数据到 OBS

3.5.1.2.1 规划导出数据

操作场景

在 OBS 上规划导出数据存放的位置。

规划 OBS 存储位置和文件

导出数据需要指定数据在 OBS 中的存储路径（需指定到目录），导出的数据可以按 CSV 解析格式保存到文件中。系统还支持 TEXT 类型的解析格式，将数据导出保存便于导入不同的应用程序。

导出路径的目标目录中不能存在任何文件。

规划 OBS 桶权限

在导出数据时，执行导出操作的用户需要具备以下条件：

- 已开通 OBS 服务。
- 具备数据导出路径所在的 OBS 桶的写入权限。

  通过配置桶的 ACL 权限，可以将写入权限授予指定的用户帐号。

  具体操作请参见根据规划准备 OBS 存储位置和 OBS 桶的写权限。

规划导出数据和外表

提前在数据库的表中准备好待导出的数据，且单行数据大小需要小于 1GB。根据导出数据，规划匹配用户数据的外表，外表的字段、字段类型以及长度等属性需要能够对应用户数据。

根据规划准备 OBS 存储位置和 OBS 桶的写权限

步骤 1 创建 OBS 桶，并在 OBS 桶中新建文件夹作为导出数据的存放目录。

1. 登录 OBS 管理控制台。

   单击“服务列表”，选择“对象存储服务”，打开 OBS 管理控制台页面。

2. 创建桶。

   如何创建 OBS 桶，具体请参见《对象存储服务用户指南》中的“控制台指南 > 管理桶 > 创建桶”章节。

   例如，创建桶：“mybucket”。

3. 新建文件夹。

   在 OBS 桶中，创建导出数据的存放目录。

   具体请参见《对象存储服务用户指南》中的“控制台指南 > 管理对象 > 新建文件夹”章节。

   例如，在已创建的 OBS 桶“mybucket”中新建一个文件夹“output_data”。

步骤 2 获取新建文件夹的 OBS 路径。

在创建外表时需要指定导出数据文件的 OBS 存放目录的路径，用于创建外表时 location 参数设置。

location 参数中 OBS 文件夹的路径由“obs://”、桶名和文件路径组成，即为：obs://<bucket_name>/<file_path>/

例如，在本例中，location 参数中 OBS 文件夹路径为：

```
obs://mybucket/output_data/
```

说明

执行数据导出的时候，导出数据文件的 OBS 存放目录的路径必须为空。

步骤 3 为导出用户设置 OBS 桶的写权限。

在导出数据时，执行导出操作的用户需要具备数据导出路径所在的 OBS 桶的写入权限。通过配置桶的 ACL 权限，可以将写入权限授予指定的用户帐号。

具体请参见《对象存储服务用户指南》中的“控制台指南 > 权限控制 > 配置桶 ACL”章节。

----结束

3.5.1.2.2 创建 OBS 外表

操作步骤

步骤 1 根据 3.5.1.2.1 规划导出数据中规划的路径，由此确定创建外表时使用的参数 **location** 的值。

步骤 2 用户获取 OBS 访问协议对应的 AK 值和 SK 值。

获取访问密钥，请登录管理控制台，单击右上角的用户名并选择菜单“我的凭证”，然后在左侧导航树单击“管理访问密钥”。在访问密钥页面，可以查看已有的访问密钥ID（即 AK），如果要同时获取 AK 和 SK，可以单击“新增访问密钥”创建并下载访问密钥。

步骤 3 梳理待导出数据的格式信息，确定创建外表时使用的数据格式参数的值。详细使用请参见数据格式参数。

步骤 4 根据前面步骤确定的参数，**创建 OBS 外表**。外表的创建语法以及详细使用，请参考 CREATE FOREIGN TABLE (OBS 导入导出)。

----结束

示例一

例如，在 GaussDB(DWS)数据库中，创建一个 format 参数为 text 的只写外表，用于导出 text 文件。设置的参数信息如下所示：

- **location**

  在 3.5.1.2.1 规划导出数据中，通过获取数据源文件的 OBS 路径，我们已经获取到数据源文件的 OBS 路径。

  因此，设置参数“location”为：

  ```
  location 'obs://mybucket/output_data/',
  ```

- **访问秘钥（AK 和 SK）**
  - 用户获取 OBS 访问协议对应的 AK 值（access_key）。
  - 用户获取 OBS 访问协议对应的 SK 值 （secret_access_key）。

  说明

  用户在创建用户时已经获取了 access_key 和 secret_access_key 的密钥，请根据实际密钥替换示例中的内容。

- **设置数据格式参数**
  - **数据源文件格式（format）**为 TEXT。
  - **编码格式（encoding）**为 UTF-8。
  - **是否使用加密（encrypt）**，默认为 'off'。
  - **字段分隔符（delimiter）**为'|'。

根据以上信息，创建的外表如下所示：

```
DROP FOREIGN TABLE IF EXISTS product info output ext1;
CREATE FOREIGN TABLE product info output ext1
(
 c bigint bigint,
 c_char char(30),
 c_varchar varchar(30),
 c_nvarchar2 nvarchar2(30) ,
 c_data date,
 c_time time ,
 c_test varchar(30))
 server gsmpp_server
```

```
options (
LOCATION 'obs://mybucket/output_data/',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
format 'text',
delimiter '|',
encoding 'utf-8',
encrypt 'on'
 )
WRITE ONLY;
```

返回如下信息表示创建成功：

```
CREATE FOREIGN TABLE
```

示例二

例如，在 GaussDB(DWS)数据库中，创建一个 format 参数为 CSV 的只写外表，用于导出 CSV 文件。设置的参数信息如下所示：

- **location**

  在 3.5.1.2.1 规划导出数据中，通过获取数据源文件的 OBS 路径，已经获取到数据源文件的 OBS 路径。

  因此，设置参数“location”为：

  ```
  location 'obs://mybucket/output_data/',
  ```

- **访问秘钥（AK 和 SK）**
  - 用户获取 OBS 访问协议对应的 AK 值（access_key）。
  - 用户获取 OBS 访问协议对应的 SK 值 （secret_access_key）。

  说明

  用户在创建用户时已经获取了 access_key 和 secret_access_key 的密钥，请根据实际密钥替换示例中的对应内容。

- **设置数据格式参数**
  - **数据源文件格式（format）**为 CSV。
  - **编码格式（encoding）**为 UTF-8。
  - **是否使用加密（encrypt），**默认为 'off'。
  - **字段分隔符（delimiter）**为','。
  - **header（指定导出数据文件是否包含标题行）**

    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。

    OBS 导出数据时不支持该参数为 ture，使用缺省值 false，不需要设置，表示导出的数据文件第一行不是标题行（即表头）。

根据以上信息，创建的外表如下所示：

```
DROP FOREIGN TABLE IF EXISTS product_info_output_ext2;
CREATE FOREIGN TABLE product_info_output_ext2
(
    product_price             integer       not null,
```

```
   product_id                char(30)      not null,
   product_time              date         ,
   product_level             char(10)      ,
   product_name              varchar(200)   ,
   product_type1             varchar(20)    ,
   product_type2             char(10)      ,
   product_monthly_sales_cnt    integer       ,
   product_comment_time       date          ,
   product_comment_num        integer       ,
   product_comment_content     varchar(200)
)
SERVER gsmpp_server
OPTIONS(
location 'obs://mybucket/output_data/',
FORMAT 'CSV' ,
DELIMITER ',',
encoding 'utf8',
header 'false',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
)
WRITE ONLY ;
```

返回如下信息表示创建成功：

```
CREATE FOREIGN TABLE
```

3.5.1.2.3 执行导出

操作步骤

步骤 1 执行数据导出。

```
INSERT INTO  [foreign table 表名] SELECT * FROM [源表名];
```

----结束

执行导出数据示例

- **示例 1**：将表 product_info_output 的数据通过外表 product_info_output_ext 导出到数据文件中。

  ```
  INSERT INTO product_info_output_ext SELECT * FROM product_info_output;
  ```

  若出现以下类似信息，说明数据导出成功。

  ```
  INSERT 0 10
  ```

- **示例 2**：通过条件过滤（WHERE *product_price>500*），向数据文件中导出部分数据。

  ```
  INSERT INTO product_info_output_ext SELECT * FROM product_info_output WHERE
  product_price>500;
  ```

  说明

  - 在需要执行多次数据导出时，导出数据的存放路径必须为空，否则将导出失败。

- 对于特殊的数据类型如 RAW 类型，在导出之后是一个二进制文本，导入工具无法识别。需使用 RAWTOHEX()函数将其转换为 16 进制文本导出。

3.5.1.2.4 示例

单表导出操作步骤

通过创建外表，将数据库中的单表导出至 OBS 的两个桶中。

步骤 1 用户通过管理控制台登录到 OBS 数据服务器。在 OBS 数据服务器上，分别创建数据文件存放的两个桶“/input-data1”“/input-data2”，并创建每个桶下面的 data 目录“/input-data1/data”“/input-data2/data”。

步骤 2 在 GaussDB(DWS)数据库上，创建外表 tpcds.customer_address_ext1 和 tpcds.customer_address_ext2 用于 OBS 数据服务器接收数据库导出数据。

OBS 与集群处于同一区域，需要导出的表为 GaussDB(DWS)示例表 tpcds.customer_address。

其中设置的**导出信息**如下所示：

- 由于 OBS 数据服务器上的数据源文件存放目录为“/input-data1/data/ ”和/input-data2/data/ ，所以设置 tpcds.customer_address_ext1 参数“location”为“obs://input-data1/data/ ”，设置 tpcds.customer_address_ext2 参数“location”为“ obs://input-data2/data/”。

设置的**数据格式信息**是根据表从数据库导出时需要的详细数据格式参数信息指定的，参数设置如下所示：

- 数据源文件格式（format）为 CSV。
- 编码格式（encoding）为 UTF-8。
- 字段分隔符（delimiter）为 E'\x08'。
- 是否使用加密（encrypt），默认为 'off'。
- 用户获取 OBS 访问协议对应的 AK 值（access_key ）。（必选）
- 用户获取 OBS 访问协议对应的 SK 值（secret_access_key）。（必选）

说明

用户在创建用户时已经获取了 access_key 和 secret_access_key 的密钥，请根据实际密钥替换示例中的内容。

根据以上信息，创建的外表如下所示：

```
CREATE FOREIGN TABLE tpcds.customer_address_ext1
(
ca_address_sk           integer                  ,
ca_address_id           char(16)                 ,
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)
)
SERVER gsmpp_server
OPTIONS(LOCATION 'obs://input-data1/data/',
FORMAT 'CSV',
ENCODING 'utf8',
DELIMITER E'\x08',
ENCRYPT 'off',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
)Write Only;
CREATE FOREIGN TABLE tpcds.customer_address_ext2
(
ca_address_sk          integer                  ,
ca_address_id          char(16)                 ,
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)
)
SERVER gsmpp_server
OPTIONS(LOCATION 'obs://input-data2/data/',
FORMAT 'CSV',
ENCODING 'utf8',
DELIMITER E'\x08',
ENCRYPT 'off',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
)Write Only;
```

步骤 3 在 GaussDB(DWS)数据库上，将数据表 tpcds.customer_address 并发导出到外表 tpcds.customer_address_ext1 和 tpcds.customer_address_ext2 中。

```
INSERT INTO tpcds.customer_address_ext1 SELECT * FROM tpcds.customer_address;
INSERT INTO tpcds.customer_address_ext2 SELECT * FROM tpcds.customer_address;
```

说明

OBS 外表在设计上禁止往非空的路径下导出文件，但是在并发场景下会出现同一路径导出文件的情况，此时会发生异常。

异常场景：假如用户使用同一张表的数据并发导出到同一个 OBS 的外表，在一条 SQL 语句执行在 OBS 服务器上没有生成文件时，另一条 SQL 语句也执行导出，最终执行结果为两条 SQL 语句均执行成功，产生数据覆盖现象，建议用户在执行 OBS 外表导出任务时，不要往同一 OBS 外表并发导出。

----结束

多表并发导出操作步骤

通过创建的两个外表，将数据库中的两个表分别导出至 OBS 的桶中。

步骤 1 用户通过管理控制台登录到 OBS 数据服务器。在 OBS 数据服务器上，分别创建数据文件存放的两个桶“/input-data1”“/input-data2”，并创建每个桶下面的 data 目录“/input-data1/data”“/input-data2/data”。

步骤 2 在 GaussDB(DWS)数据库上，创建外表 tpcds.customer_address_ext1 和 tpcds.customer_address_ext2 分别用于 OBS 服务器接收导出的数据。

规划 OBS 与集群处于同一区域，需要导出的表为已存在的表 tpcds.customer_address 和 tpcds.customer_demographics。

其中设置的**导出信息**如下所示：

- 由于 OBS 服务器上的数据源文件存放目录为“/input-data1/data/ ”和/input-data2/data/ ，所以设置 tpcds.customer_address_ext1 参数“location”为“obs://input-data1/data/ ”，设置 tpcds.customer_address_ext2 参数“location”为“ obs://input-data2/data/”。

设置的**数据格式信息**是根据表从 GaussDB(DWS)中导出时需要的详细数据格式参数信息指定的，参数设置如下所示：

- 数据源文件格式（format）为 CSV。
- 编码格式（encoding）为 UTF-8。
- 字段分隔符（delimiter）为 E'\x08'。
- 是否使用加密（encrypt），默认为 'off'。
- 用户获取 OBS 访问协议对应的 AK 值（access_key ）。（必选）
- 用户获取 OBS 访问协议对应的 SK 值（secret_access_key）。（必选）

说明

用户在创建用户是已经获取了 access_key 和 secret_access_key 的密钥，请根据实际密钥替换示例中的内容。

根据以上信息，创建的外表如下所示：

```
CREATE FOREIGN TABLE tpcds.customer address ext1
(
ca address sk           integer             ,
ca address id           char(16)            ,
ca_street_number         char(10)                    ,
ca_street_name          varchar(60)                  ,
ca_street_type          char(15)                    ,
```

```
ca_suite_number         char(10)                  ,
ca_city                varvchar(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)
)
SERVER gsmpp_server
OPTIONS(LOCATION 'obs://input-data1/data/',
FORMAT 'CSV',
ENCODING 'utf8',
DELIMITER E'\x08',
ENCRYPT 'off',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
)Write Only;
CREATE FOREIGN TABLE tpcds.customer_address_ext2
(
ca_address_sk          integer            ,
ca_address_id          char(16)           ,
ca_address_name        varchar(20)         ,
ca_address_code        integer            ,
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)
)
SERVER gsmpp_server
OPTIONS(LOCATION 'obs://input_data2/data/',
FORMAT 'CSV',
ENCODING 'utf8',
DELIMITER E'\x08',
QUOTE E'\x1b',
ENCRYPT 'off',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
)Write Only;
```

步骤 3　在 GaussDB(DWS)数据库上，将数据表 tpcds.customer_address 和 tpcds.warehouse 并发导出到外表 tpcds.customer_address_ext1 和 tpcds.customer_address_ext2 中。

```
INSERT INTO tpcds.customer_address_ext1 SELECT * FROM tpcds.customer_address;
INSERT INTO tpcds.customer_address_ext2 SELECT * FROM tpcds.warehouse;
```

----结束

3.5.1.3 导出 ORC 数据到 OBS

3.5.1.3.1 规划导出数据

OBS 导出数据准备：请参见 3.5.1.2.1 规划导出数据完成 OBS 导出数据准备。

OBS 导出支持的数据类型请参见表 3-6。

HDFS 导出数据准备：HDFS 导出准备即配置 MRS，具体信息可参考《MapReduce 服务用户指南》。

3.5.1.3.2 创建外部服务器

OBS 创建外部服务器请参见 3.2.1.3.2 创建外部服务器。

HDFS 创建外部服务器请参见 3.2.3.3 手动创建外部服务器。

3.5.1.3.3 创建外表

当完成 3.5.1.3.2 创建外部服务器后，在 GaussDB(DWS)数据库中创建一个 OBS/HDFS 只写外表，用来访问存储在 OBS/HDFS 上的数据。此外表是只写的，只能用于导出操作。

创建外表的语法格式如下，详细的描述请参见 CREATE FOREIGN TABLE (SQL on Hadoop or OBS)。

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name
( [ { column_name type_name
   [ { [CONSTRAINT constraint_name] NULL |
   [CONSTRAINT constraint_name] NOT NULL |
     column_constraint [...]} ] |
     table_constraint [, ...]} [, ...] ] )
   SERVER dfs_server
   OPTIONS ( { option_name ' value ' } [, ...] )
   [ {WRITE ONLY }]
   DISTRIBUTE BY {ROUNDROBIN | REPLICATION}
   [ PARTITION BY ( column_name ) [ AUTOMAPPED ] ] ;
```

例如，创建一个名为"*product_info_ext_obs*"的外表，对语法中的参数按如下描述进行设置：

- **table_name**

  外表的表名。

- **表字段定义**
  - **column_name**：外表中的字段名。
  - **type_name**：字段的数据类型。

  多个字段用“,”隔开。

- **SERVER dfs_server**

  外表的外部服务器名称，这个 server 必须存在。外表通过设置外部服务器连接 OBS/HDFS 读取数据。

  此处应填写为参照 9.2.3 创建外部服务器创建的外部服务器名称。

- **OPTIONS 参数**

用于指定外表数据的各类参数，关键参数如下所示。

- “format”：表示导出的数据文件格式，支持“orc”格式。
- “foldername”：必选参数。外表中数据源文件目录。OBS：数据源文件的 OBS 路径，此处仅需要填写“*/桶名/文件夹目录层级/*”。HDFS：HDFS 文件系统上的路径。此选项对 WRITE ONLY 外表为必选项。
- **“encoding”**：外表中数据源文件的编码格式名称，缺省为 utf8。
- filesize

  指定 WRITE ONLY 外表的文件大小。此选项为可选项，不指定该选项默认分布式文件系统配置中文件大小的配置值。此语法仅对 WRITE ONLY 的外表有效。

  取值范围：[1, 1024]的整数。

说明

filesize 参数只对 ORC 格式的 WRITE ONLY 的 HDFS 外表有效。

- compression

  指定 ORC 格式文件的压缩方式，此选项为可选项。此语法仅对 WRITE ONLY 的外表有效。

  取值范围：zlib，snappy，lz4。缺省值为 snappy。
- version

  指定 ORC 格式的版本号，此选项为可选项。此语法仅对 WRITE ONLY 的外表有效。

  取值范围：目前仅支持 0.12。缺省值为 0.12。
- dataencoding

  在数据库编码与数据表的数据编码不一致时，该参数用于指定导出数据表的数据编码。比如数据库编码为 Latin-1，而导出的数据表中的数据为 UTF-8 编码。此选项为可选项，如果不指定该选项，默认采用数据库编码。此语法仅对 HDFS 的 WRITE ONLY 外表有效。

  取值范围：该数据库编码支持转换的数据编码。

说明

dataencoding 参数只对 ORC 格式的 WRITE ONLY 的 HDFS 外表有效。

- **语法中的其他参数**

  其他参数均为可选参数，用户可以根据自己的需求进行设置，在本例中不需要设置。

  根据以上信息，创建外表命令如下所示：

```
DROP FOREIGN TABLE IF EXISTS product_info_ext_obs;

---建立不包含分区列的 OBS 外表，表关联的外部服务器为 obs_server，表对应的 OBS 服务上的文件格式为'orc'，OBS 上的数据存储路径为'/mybucket/data/'。

CREATE FOREIGN TABLE product_info_ext_obs
(
    product_price             integer       ,
    product_id                char(30)      ,
    product_time              date          ,
```

```
   product_level             char(10)      ,
   product_name              varchar(200)   ,
   product_type1             varchar(20)    ,
   product_type2             char(10)      ,
   product_monthly_sales_cnt    integer       ,
   product_comment_time        date          ,
   product_comment_num         integer       ,
   product_comment_content     varchar(200)
) SERVER obs_server
OPTIONS (
format 'orc',
foldername '/mybucket/demo.db/product_info_orc/',
  compression 'snappy',
   version '0.12'
) Write Only;
```

3.5.1.3.4 执行导出

操作步骤

步骤 1 执行数据导出。

```
INSERT INTO  [foreign table 表名] SELECT * FROM [源表名];
```

----结束

执行导出数据示例

- **示例 1**：将表 product_info_output 的数据通过外表 product_info_output_ext 导出到数据文件中。

  ```
  INSERT INTO product_info_output_ext SELECT * FROM product_info_output;
  ```

  若出现以下类似信息，说明数据导出成功。

  ```
  INSERT 0 10
  ```

- **示例 2**：通过条件过滤（WHERE *product_price>500*），向数据文件中导出部分数据。

  ```
  INSERT INTO product_info_output_ext SELECT * FROM product_info_output WHERE
  product_price>500;
  ```

说明

对于特殊的数据类型如 RAW 类型，在导出之后是一个二进制文本，导入工具无法识别。需使用 RAWTOHEX()函数将其转换为 16 进制文本导出。

3.5.2 导出 ORC 数据到 MRS

3.5.2.1 导出 ORC 数据概述

GaussDB(DWS)数据库支持通过 HDFS 外表导出 ORC 格式数据至 MRS，通过外表设置的导出模式、导出数据格式等信息来指定导出的数据文件，利用多 DN 并行的方式，将数据从 GaussDB(DWS)数据库导出到外部，存放在 HDFS 文件系统上，从而提高整体导出性能。

- CN 只负责任务的规划及下发，把数据导出的工作交给了 DN，释放了 CN 的资源，使其有能力处理外部请求。
- 通过让各个 DN 都参与数据导出，充分利用各个设备的计算能力及网络带宽。
- 支持多个 hdfs server 并发导出，导出的路径可以为空，命名规则需与导出文件一致。
- 选择 MRS 服务与集群节点处于联网状态，导出速率会受网络带宽影响。
- 支持数据文件格式：ORC。

导出文件命名规则

GaussDB(DWS)导出 ORC 数据的文件命名规则如下：

1. 导出至 MRS（HDFS）：从 DN 节点导出数据时，以 segment 的格式存储在 HDFS 中，文件命名规则为“**mpp_数据库名_模式名_表名称_节点名称_n.orc**”。这里的“n”是从 0 开始按照自然数 0、1、2、3 递增。
2. 对于来自不同集群或不同数据库的数据，建议用户可以将数据导出到不同路径下。ORC 格式文件大小最大为 128M，Stripe 大小最大为 64M。
3. 导出完成后会生成_SUCCESS 标记文件。

3.5.2.2 规划导出数据

MRS 导出支持的数据类型请参见表 3-6。

HDFS 导出数据准备：HDFS 导出准备即配置 MRS，具体信息可参考《MapReduce 服务用户指南》。

3.5.2.3 创建外部服务器

HDFS 创建外部服务器请参见 3.2.3.3 手动创建外部服务器。

3.5.2.4 创建外表

当完成 3.5.2.3 创建外部服务器后，在 GaussDB(DWS)数据库中创建一个 HDFS 只写外表，用来访问存储在 HDFS 上的数据。此外表是只写的，只能用于导出操作。

创建外表的语法格式如下，详细的描述请参见 CREATE FOREIGN TABLE (SQL on Hadoop or OBS)。

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name
( [ { column_name type_name
   [ { [CONSTRAINT constraint_name] NULL |
   [CONSTRAINT constraint_name] NOT NULL |
     column_constraint [...]} ] |
     table_constraint [, ...]} [, ...] ] )
   SERVER dfs_server
   OPTIONS ( { option_name ' value ' } [, ...] )
   [ {WRITE ONLY }]
   DISTRIBUTE BY {ROUNDROBIN | REPLICATION}
   [ PARTITION BY ( column_name ) [ AUTOMAPPED ] ] ;
```

例如，创建一个名为"*product_info_ext_obs*"的外表，对语法中的参数按如下描述进行设置：

- **table_name**

  外表的表名。

- **表字段定义**
  - **column_name**：外表中的字段名。
  - **type_name**：字段的数据类型。

  多个字段用“,”隔开。

- **SERVER dfs_server**

  外表的外部服务器名称，这个 server 必须存在。外表通过设置外部服务器连接 OBS/HDFS 读取数据。

  此处应参考 3.5.2.3 创建外部服务器中创建的外部服务器名称填写。

- **OPTIONS 参数**

  用于指定外表数据的各类参数，关键参数如下所示。
  - “format”：表示导出的数据文件格式，支持“orc”格式。
  - “foldername”：外表中数据源文件目录，即表数据目录在 HDFS 文件系统上对应的文件目录。此选项对 WRITE ONLY 外表为必选项，对 READ ONLY 外表为可选项。
  - **“encoding”**：外表中数据源文件的编码格式名称，缺省为 utf8。
  - filesize

    指定 WRITE ONLY 外表的文件大小。此选项为可选项，不指定该选项默认分布式文件系统配置中文件大小的配置值。此语法仅对 WRITE ONLY 的外表有效。

    取值范围：[1, 1024]的整数。

说明

filesize 参数只对 ORC 格式的 WRITE ONLY 的 HDFS 外表有效。

  - compression

    指定 ORC 格式文件的压缩方式，此选项为可选项。此语法仅对 WRITE ONLY 的外表有效。

    取值范围：zlib，snappy，lz4。缺省值为 snappy。
  - version

    指定 ORC 格式的版本号，此选项为可选项。此语法仅对 WRITE ONLY 的外表有效。

    取值范围：目前仅支持 0.12。缺省值为 0.12。
  - dataencoding

    在数据库编码与数据表的数据编码不一致时，该参数用于指定导出数据表的数据编码。比如数据库编码为 Latin-1，而导出的数据表中的数据为 UTF-8 编码。此选项为可选项，如果不指定该选项，默认采用数据库编码。此语法仅对 HDFS 的 WRITE ONLY 外表有效。

    取值范围：该数据库编码支持转换的数据编码。

说明

dataencoding 参数只对 ORC 格式的 WRITE ONLY 的 HDFS 外表有效。

- **语法中的其他参数**

  其他参数均为可选参数，用户可以根据自己的需求进行设置，在本例中不需要设置。详细的描述请参见 CREATE FOREIGN TABLE　(SQL on Hadoop or OBS)。

  根据以上信息，创建外表命令如下所示：

```
DROP FOREIGN TABLE IF EXISTS product_info_ext_obs;

---建立不包含分区列的 HDFS 外表，表关联的外部服务器为 hdfs_server，表对应的 HDFS 服务上的文件格式为'orc'，HDFS 上的数据存储路径为'/user/hive/warehouse/product_info_orc/'。

CREATE FOREIGN TABLE product_info_ext_obs
(
    product_price                integer       ,
    product_id                   char(30)      ,
    product_time                 date          ,
    product_level                char(10)      ,
    product_name                 varchar(200)  ,
    product_type1                varchar(20)   ,
    product_type2                char(10)      ,
    product_monthly_sales_cnt    integer       ,
    product_comment_time         date          ,
    product_comment_num          integer       ,
    product_comment_content      varchar(200)
) SERVER obs_server
OPTIONS (
format 'orc',
foldername '/user/hive/warehouse/product info orc/',
   compression 'snappy',
    version '0.12'
) Write Only;
```

3.5.2.5 执行导出

操作步骤

步骤 1　执行数据导出。

```
INSERT INTO  [foreign table 表名] SELECT * FROM [源表名];
```

----结束

执行导出数据示例

- **示例 1**：将表 product_info_output 的数据通过外表 product_info_output_ext 导出到数据文件中。

```
INSERT INTO product_info_output_ext SELECT * FROM product_info_output;
```

若出现以下类似信息，说明数据导出成功。

```
INSERT 0 10
```

- **示例 2**：通过条件过滤（WHERE *product_price>500*），向数据文件中导出部分数据。

```
INSERT INTO product_info_output_ext SELECT * FROM product_info_output WHERE product_price>500;
```

说明

对于特殊的数据类型如 RAW 类型，在导出之后是一个二进制文本，导入工具无法识别。需使用 RAWTOHEX()函数将其转换为 16 进制文本导出。

3.5.3 使用 GDS 导出数据到远端服务器

3.5.3.1 关于 GDS 并行导出

使用 GDS 工具将数据从数据库导出到普通文件系统中，适用于高并发、大量数据导出的场景。

当前版本的 GDS 支持从数据库导出到管道文件，该功能使 GDS 的导出更加灵活多变。

- 当 GDS 用户的本地磁盘空间不足时：
  - 通过管道文件将从 GDS 导出的数据进行压缩减少磁盘空间。
  - 通过管道直接将导出来的数据放到 hdfs 服务器上。
- 当用户导出前需要清洗数据时：
  - 用户可以根据自己的需求编写程序，将需要处理的流式数据实时从管道中读取内容，完成导出的数据清洗工作。

说明

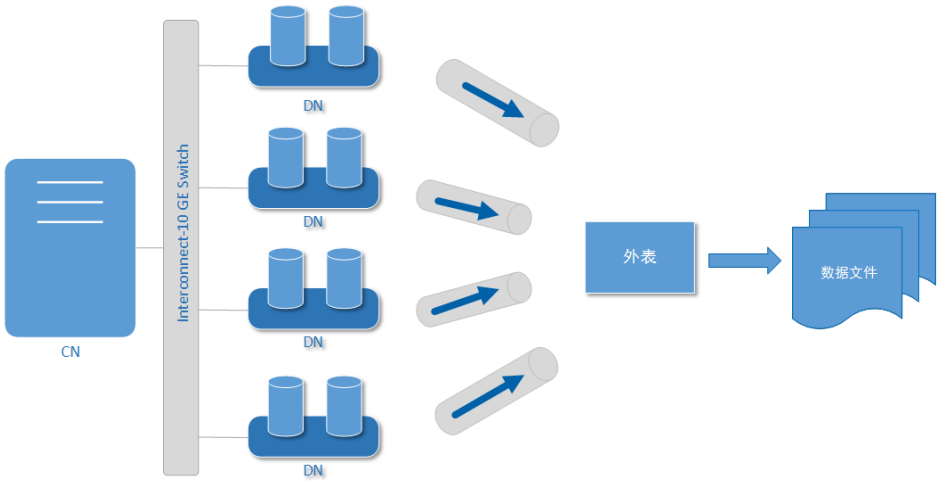
- 当前版本暂不支持 SSL 模式下 GDS 导出，请勿以 SSL 方式使用 GDS。
- 本章涉及的所有管道文件都是指 linux 上的命名管道。

概述

通过外表导出数据：通过 GDS 外表设置的导出模式、导出数据格式等信息来指定待导出的数据文件，利用多 DN 并行的方式，将数据从数据库导出到数据文件中，从而提高整体导出性能。不支持直接导出文件到 HDFS 文件系统。

- CN 只负责任务的规划及下发，把数据导出的工作交给了 DN，释放了 CN 的资源，使其有能力处理外部请求。
- 通过让各个 DN 都参与数据导出，充分利用各个设备的计算能力及网络带宽。

图3-10 通过外表导出数据



相关概念

- **数据文件**：存储有数据的 TEXT、CSV 或 FIXED 文件。文件中保存的是从 GaussDB(DWS)数据库导出的数据。
- **外表**：用于规划导出数据文件的数据文件格式、存放位置、编码格式等信息。
- **GDS**：数据服务工具。在导出数据时，需要将此工具部署到数据文件所在的服务器上，使 DN 可以通过该工具导出数据。
- **表**：数据库中的表，包括行存表和列存表。数据文件中的数据从这些表中导出。
- **Remote 导出模式**：将集群中的业务数据导出到集群之外的主机上。

导出模式

GaussDB(DWS)支持的导出模式有 Remote 模式。

- **Remote 模式**：将集群中的业务数据导出到集群之外的主机上。
  - 支持多个 GDS 服务并发导出，但 1 个 GDS 在同一时刻，只能为 1 个集群提供导出服务。
  - 配置与集群节点处于统一内网的 GDS 服务，导出速率受网络带宽影响，推荐的网络配置为 10GE。
  - 支持数据文件格式：TEXT、CSV。单行数据大小需<1GB。

导出流程

图3-11 并行导出流程

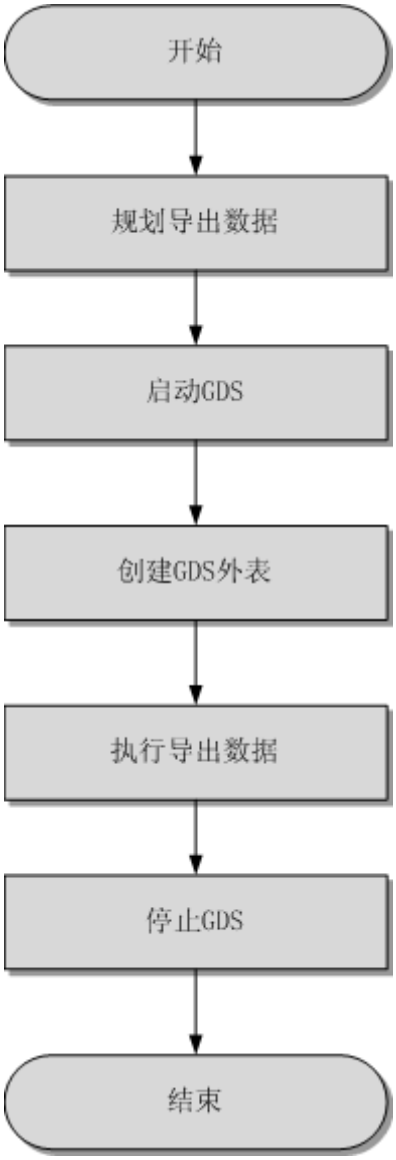


表3-23 流程说明

流程	说明	子任务
规划导出数据。	根据所选模式，准备需要导出的数据并规划导出路径。 详细内容请参见 3.5.3.2 规划导出数据	-
启动GDS。	若规划的导出模式为 Remote 模式，需在数据服务器上安装配置并启动GDS。	-

流程	说明	子任务
	详细内容请参见 3.5.3.3 安装配置和启动 GDS。	
创建外表。	创建外表用于帮助 GDS 指定导出的数据文件。外表中保存了导出数据文件的位置、文件格式、编码格式、数据间的分隔符等信息。 详细内容请参见 3.5.3.4 创建 GDS 外表。	-
执行导出数据。	在创建好外表后，通过 INSERT 语句，将数据快速、高效地导出到数据文件中。 详细内容请参见 3.5.3.5 执行导出数据。	-
停止 GDS。	数据导出完成后，停止 GDS。 详细请参见 3.5.3.6 停止 GDS。	-

3.5.3.2 规划导出数据

操作场景

使用 GDS 从集群导出到数据之前，要提前准备需要导出的数据，并规划导出的路径。

规划导出路径

- **Remote** 模式

步骤 1 以 root 用户登录 GDS 数据服务器，创建导出的数据文件存放目录“/output_data”。

```
mkdir -p /output_data
```

步骤 2 （可选）创建用户及所属的用户组。此用户为启动 GDS 的用户，该用户需要拥有导出数据文件存放目录的写权限。

```
groupadd gdsgrp
useradd -g gdsgrp gdsuser
```

若出现以下提示，说明数据库用户及所属用户组已存在，可跳过本步骤。

```
useradd: Account 'gdsuser' already exists.
groupadd: Group 'gdsgrp' already exists.
```

步骤 3 修改数据文件目录属主为 gdsuser。

```
chown -R gdsuser:gdsgrp /output_data
```

----结束

3.5.3.3 安装配置和启动 GDS

GDS 是 GaussDB(DWS)提供的数据服务工具，通过和外表机制的配合，实现数据的高速导出。

详细内容请参见 3.2.2.3 安装配置和启动 GDS。

3.5.3.4 创建 GDS 外表

操作步骤

步骤 1 根据 3.5.3.2 规划导出数据中规划的路径确定外表参数 **location** 的值。

- **Remote** 模式

  请通过 URL 方式设置参数“location”，用于指定导出的数据文件存放路径。

  - 不需要指定文件名。
  - 当有多个路径时，只有第一个路径有效。

  示例：

  GDS 数据服务器 IP 为 192.168.0.90，假定启动 GDS 时设置的监听端口为 5000，设置的导出后文件存放目录为“/output_data/”。

  根据以上情况，在创建外表时，指定参数“location”为“gsfs://192.168.0.90:5000/”。

  说明

  - location 可以指定子目录如"gsfs://192.168.0.90:5000/2019/11/"实现同一张表根据日期导出到不同目录下。
  - 现有版本在执行**导出**任务的时候会判断"/output_data/2019/11" 目录是否存在，不存在则创建。导出时会将文件写入此目录下， 这样用户在创建或修改外表后就不需要再去手动执行“mkdir -p /output_data/2019/11”。

步骤 2 梳理待导出数据的格式信息，确定创建外表时使用的数据格式参数的值。格式参数详细介绍，请参见数据格式参数。

步骤 3 根据前面步骤确定的参数，**创建 GDS 外表**。外表的创建语法以及详细使用，请参考 CREATE FOREIGN TABLE (GDS 导入导出)。

----结束

示例

- **示例：**创建 GDS 导出外表 foreign_tpcds_reasons，待导出数据格式为 CSV，用于接收数据服务器上的数据。

  其中设置的**导出模式**信息如下所示：

  规划数据服务器与集群处于同一内网，数据服务器 IP 为 192.168.0.90，待导出的数据文件格式为 CSV，选择并行导出模式为 Remote 模式。

  假定启动 GDS 时，规划导出的数据文件存放目录为“/output_data/”，GDS 监听端口为 5000，所以设置参数“location”为“gsfs://192.168.0.90:5000/”。

设置导出的**数据格式信息**，参数设置如下所示：

- 导出数据文件格式（format）为 CSV。
- 编码格式（encoding）为 UTF-8。
- 字段分隔符（delimiter）为 E'\x08'。
- 引号字符（quote）为 E'\x1b'。
- 数据文件中空值（null）为没有引号的空字符串。
- 逃逸字符（escape）默认和 quote 相同。
- 数据文件是否包含标题行（header）为默认值 false，即导出时数据文件第一行被识别为数据。
- 导出数据文件换行符样式（EOL）为 0X0A。

创建的外表如下所示：

```
CREATE FOREIGN TABLE foreign_tpcds_reasons
(
  r_reason_sk    integer       not null,
  r_reason_id    char(16)      not null,
  r_reason_desc  char(100)
)
SERVER gsmpp_server
OPTIONS (LOCATION 'gsfs://192.168.0.90:5000/',
FORMAT 'CSV',
DELIMITER E'\x08',
QUOTE E'\x1b',
NULL '',
EOL '0x0a'
)
WRITE ONLY;
```

3.5.3.5 执行导出数据

前提条件

需要确保每一个 CN 和 DN 所在服务器到 GDS 服务器的 IP 和端口是互通的。

操作步骤

步骤 1 执行数据导出。

```
INSERT INTO [foreign table 表名] SELECT * FROM [源表名];
```

说明

编写批处理任务脚本，实现并发批量导出数据。并发量视机器资源使用情况而定。可通过几个表测试，监控资源利用率，根据结果提高或减少并发量。常用资源监控命令有：内存和 CPU 监控 top 命令，IO 监控命令 iostat，网络监控命令 sar 等。相关案例请参见多线程导出。

----结束

任务示例

- **示例 1**：将表 reason 的数据通过外表 foreign_tpcds_reasons 导出到数据文件中。

```
INSERT INTO foreign_tpcds_reasons SELECT * FROM tpcds.reason;
```

- **示例 2**：通过条件过滤（r_reason_sk =1），向数据文件中导出部分数据。

```
INSERT INTO foreign_tpcds_reasons SELECT * FROM tpcds.reason WHERE
r_reason_sk=1;
```

- **示例 3**：对于特殊的数据类型如 RAW 类型，在导出之后是一个二进制文本，导入工具无法识别。需使用 RAWTOHEX()函数将其转换为 16 进制文本导出。

```
INSERT INTO foreign_tpcds_reasons SELECT RAWTOHEX(c) FROM tpcds.reason;
```

3.5.3.6 停止 GDS

GDS 是 GaussDB(DWS)提供的数据服务工具，通过和外表机制的配合，实现数据的高速导出。

详细内容请参见 3.2.2.7 停止 GDS。

3.5.3.7 GDS 导出示例

Remote 模式导出

规划数据服务器与集群处于同一内网，数据服务器 IP 为 192.168.0.90，导出数据文件格式为 CSV，所以规划的并行导出模式为 Remote 模式。

Remote 模式并行导出数据操作示例如下所示：

1. 以 root 用户登录 GDS 数据服务器，创建数据文件存放目录“/output_data”，启动 gds_user 用户及所属的用户组。

```
mkdir -p /output_data
```

2. （可选）创建用户及其所属的用户组。此用户用于启动 GDS。若该类用户及所属用户组已存在，可跳过此步骤。

```
groupadd gdsgrp
useradd -g gdsgrp gds_user
```

3. 修改数据服务器上数据文件目录“/output_data”的属主为 gds_user。

```
chown -R gds_user:gdsgrp /output_data
```

4. 以 gds_user 用户登录数据服务器上分别启动 GDS。

   其中 GDS 安装路径为“/opt/bin/dws/gds”，导出数据文件存放在“/output_data/”目录下，数据服务器所在 IP 为 192.168.0.90，GDS 监听端口为 5000，以后台方式运行。

```
/opt/bin/dws/gds/bin/gds -d /output_data -p 192.168.0.90:5000 -H 10.10.0.1/24 -
D
```

5. 在数据库中创建外表 foreign_tpcds_reasons 用于接收数据服务器上的数据。

   其中设置的**导出模式信息**如下所示：

   - 由于启动 GDS 时，设置的导出数据文件存放目录为“/output_data/”，GDS 监听端口为 5000。创建的导出数据文件存放目录为“/output_data/”。所以设置参数“location”为“gsfs://192.168.0.90:5000/”。

设置导出的**数据文件格式**信息如下所示：

- 数据文件格式（format）为 CSV。
- 编码格式（encoding）为 UTF-8。
- 字段分隔符（delimiter）为 E'\x08'。
- 引号字符（quote）为 E'\x1b'。
- 数据文件中空值（null）为没有引号的空字符串。
- 逃逸字符（escape）默认和 quote 相同。
- 数据文件是否包含标题行（header）为默认值 false，即导出时数据文件第一行被识别为数据。

根据以上信息，创建的外表如下所示：

```
CREATE FOREIGN TABLE foreign_tpcds_reasons
(
  r_reason_sk    integer       not null,
  r_reason_id    char(16)      not null,
  r_reason_desc  char(100)
)
SERVER gsmpp_server
OPTIONS
(
LOCATION 'gsfs://192.168.0.90:5000/',
FORMAT 'CSV',
ENCODING 'utf8',
DELIMITER E'\x08',
QUOTE E'\x1b',
NULL ''
)
WRITE ONLY;
```

6. 在数据库上，通过外表 foreign_tpcds_reasons，将数据导出到数据文件中。

```
INSERT INTO foreign_tpcds_reasons SELECT * FROM tpcds.reason;
```

7. 待数据导出完成后，以 gds_user 用户登录数据服务器，停止 GDS。

   其中 GDS 进程号为 128954。

```
ps -ef|grep gds
gds_user 128954     1  0 15:03 ?       00:00:00 gds -d /output_data -p
192.168.0.90:5000 -D
gds_user 129003 118723  0 15:04 pts/0    00:00:00 grep gds
kill -9 128954
```

多线程导出

规划数据服务器与集群处于同一内网，数据服务器 IP 为 192.168.0.90，导出的数据文件格式为 CSV，同时导出 2 个目标表，所以规划使用 Remote 模式进行多线程导出。

Remote 模式多线程导出数据操作示例如下所示：

1. 以 root 用户登录 GDS 数据服务器，创建导出数据文件存放目录“/output_data”，数据库用户及所属的用户组。

```
mkdir -p /output_data
groupadd gdsgrp
useradd -g gdsgrp gds_user
```

2. 修改数据服务器上数据文件目录“/output_data”的属主为 gds_user。

```
chown -R gds_user:gdsgrp /output_data
```

3. 以 gds_user 用户登录数据服务器上启动 GDS。

   其中 GDS 安装路径为“/opt/bin/dws/gds”，导出数据文件存放在“/output_data/”目录下，数据服务器所在 IP 为 192.168.0.90，GDS 监听端口为 5000，以后台方式运行，设定并发度为 2。

```
/opt/bin/dws/gds/bin/gds -d /output_data -p 192.168.0.90:5000 -H 10.10.0.1/24 -D -t 2
```

4. 在 GaussDB(DWS)上，创建外表 foreign_tpcds_reasons1 和 foreign_tpcds_reasons2 用于接收数据服务器上的数据。

   - 其中设置的**导出模式信息**如下所示：
     - 由于启动 GDS 时，设置的导出数据文件存放目录为“/output_data/”，GDS 监听端口为 5000。创建的导出数据文件存放目录为“/output_data/”。所以设置参数“location”为“gsfs://192.168.0.90:5000/”。
   - 设置导出的数据**文件格式信息**如下所示：
     - 数据文件格式（format）为 CSV。
     - 编码格式（encoding）为 UTF-8。
     - 字段分隔符（delimiter）为 E'\x08'。
     - 引号字符（quote）为 E'\x1b'。
     - 数据文件中空值（null）为没有引号的空字符串。
     - 逃逸字符（escape）默认和 quote 相同。
     - 数据文件是否包含标题行（header）为默认值 false，即导出时数据文件第一行被识别为数据。

   根据以上信息，创建的外表 foreign_tpcds_reasons1 如下所示：

```
CREATE FOREIGN TABLE foreign_tpcds_reasons1
(
  r_reason_sk   integer    not null,
  r_reason_id   char(16)   not null,
  r_reason_desc  char(100)
)
SERVER gsmpp_server
OPTIONS
(
LOCATION 'gsfs://192.168.0.90:5000/',
FORMAT 'CSV',
ENCODING 'utf8',
DELIMITER E'\x08',
QUOTE E'\x1b',
NULL ''
)
WRITE ONLY;
```

   参考以上设置，创建的外表 foreign_tpcds_reasons2 如下所示：

```
CREATE FOREIGN TABLE foreign_tpcds_reasons2
(
  r_reason_sk    integer     not null,
  r_reason_id    char(16)    not null,
  r_reason_desc  char(100)
)
SERVER gsmpp_server
OPTIONS
(
LOCATION 'gsfs://192.168.0.90:5000/',
FORMAT 'CSV',
DELIMITER E'\x08',
QUOTE E'\x1b',
NULL ''
)
WRITE ONLY;
```

5. 在数据库中通过外表 foreign_tpcds_reasons1 和 foreign_tpcds_reasons2，将表 reasons1 和 reasons2 中的数据导出到目录“/output_data”中。

```
INSERT INTO foreign_tpcds_reasons1 SELECT * FROM tpcds.reason;
INSERT INTO foreign_tpcds_reasons2 SELECT * FROM tpcds.reason;
```

6. 待数据导出完成后，以 gds_user 用户登录数据服务器，停止 GDS。

   其中 GDS 进程号为 128954。

```
ps -ef|grep gds
gds_user 128954     1  0 15:03 ?       00:00:00 gds -d /output_data -p
192.168.0.90:5000 -D -t 2
gds_user 129003 118723  0 15:04 pts/0    00:00:00 grep gds
kill -9 128954
```

单个管道文件导出

步骤 1 启动 GDS。

```
gds -d /***/gds_data/ -D -p 192.168.0.1:7789 -l /***/gds_log/aa.log -H 0/0 -t 10 -D
```

如果需要设置管道文件的超时时间，则使用--pipe-timeout 参数设置。

步骤 2 执行数据导出。

1. 登录数据库创建内表，并写入数据。

```
CREATE TABLE test_pipe( id integer not null, sex text not null, name text ) ;


INSERT INTO test_pipe values(1,2,'11111111111111');
INSERT INTO test_pipe values(2,2,'11111111111111');
INSERT INTO test_pipe values(3,2,'11111111111111');
INSERT INTO test_pipe values(4,2,'11111111111111');
INSERT 0 1
```

2. 创建只写外表。

```
CREATE FOREIGN TABLE foreign_test_pipe_tw( id integer not null, age text not
null, name  text ) SERVER gsmpp_server OPTIONS (LOCATION
'gsfs://192.168.0.1:7789/', FORMAT 'text', DELIMITER ',',  NULL '', EOL
'0x0a' ,file_type 'pipe', auto_create_pipe 'false') WRITE ONLY;
```

3. 导出语句，此时语句会阻塞。

```
INSERT INTO foreign_test_pipe_tw select * from test_pipe;
```

步骤 3 通过管道文件将数据从 GDS 导出。

1. 登录 GDS，进入 GDS 数据目录。

```
cd /***/gds_data/
```

2. 创建管道文件，如果 auto_create_pipe 设置为 true 跳过此步骤。

```
mkfifo postgres_public_foreign_test_pipe_tw.pipe
```

说明

管道文件创建完成后，每执行完一次操作，业务会被自动清理。如果还需要执行其他业务，请参考该步骤重新创建管道文件。

3. 读取管道文件并写入新文件。

```
cat postgres_public_foreign_test_pipe_tw.pipe >
postgres_public_foreign_test_pipe_tw.txt
```

4. 若需要对导出的文件进行压缩执行：

```
gzip -9 -c < postgres_public_foreign_test_pipe_tw.pipe  > out.gz
```

5. 若需要将管道文件的内容导出到 hdfs 服务器：

```
cat postgres_public_foreign_test_pipe_tw.pipe  | hdfs dfs -put -
/user/hive/***/test_pipe.txt
```

步骤 4 验证导出的数据。

1. 查看文件是否导出正确。

```
cat postgres_public_foreign_test_pipe_tw.txt
3,2,11111111111111
1,2,11111111111111
2,2,11111111111111
4,2,11111111111111
```

2. 查看压缩后的文件。

```
vim out.gz
3,2,11111111111111
1,2,11111111111111
2,2,11111111111111
4,2,11111111111111
```

3. 查看导出到 hdfs 服务器上的数据。

```
hdfs dfs -cat /user/hive/***/test_pipe.txt
3,2,11111111111111
1,2,11111111111111
2,2,11111111111111
4,2,11111111111111
```

----结束

多进程管道文件导出

GDS 也支持多进程管道文件导入导出， 即启动一个外表对应多个 GDS。

以本地文件的导出为例：

步骤 1 启动多个 GDS。

```
gds -d /***/gds_data/ -D -p 192.168.0.1:7789 -l /***/gds_log/aa.log -H 0/0 -t 10 -D
gds -d /***/gds_data_1/ -D -p 192.168.0.1:7790 -l /***/gds_log/aa.log -H 0/0 -t 10
-D
```

如果需要设置管道文件的超时时间，则使用--pipe-timeout 参数设置。

步骤 2 执行数据导出。

1. 登录数据库创建内表。

```
CREATE TABLE test_pipe (id integer not null, sex text not null, name  text);
```

2. 写入数据。

```
INSERT INTO test_pipe values(1,2,'11111111111111');
INSERT INTO test_pipe values(2,2,'11111111111111');
INSERT INTO test_pipe values(3,2,'11111111111111');
INSERT INTO test_pipe values(4,2,'11111111111111');
```

3. 创建只写外表。

```
CREATE FOREIGN TABLE foreign test pipe tw( id integer not null, age text not
null, name  text ) SERVER gsmpp server OPTIONS (LOCATION
'gsfs://192.168.0.1:7789/|gsfs://192.168.0.1:7790/', FORMAT 'text', DELIMITER
',',  NULL '', EOL '0x0a' ,file_type 'pipe', auto_create_pipe 'false') WRITE
ONLY;
```

4. 导出语句，此时语句会阻塞。

```
INSERT INTO foreign_test_pipe_tw select * from test_pipe;
```

步骤 3 通过管道文件将数据从 GDS 导出。

1. 登录 GDS，分别进入 GDS 数据目录。

```
cd /***/gds_data/
cd /***/gds_data_1/
```

2. 创建管道文件，如果 auto_create_pipe 设置为 true 跳过此步骤。

```
mkfifo postgres_public_foreign_test_pipe_tw.pipe
```

3. 分别读取管道文件并写入新文件。

```
cat postgres_public_foreign_test_pipe_tw.pipe >
postgres_public_foreign_test_pipe_tw.txt
```

步骤 4 验证导出的数据。

```
cat /***/gds_data/postgres_public_foreign_test_pipe_tw.txt
3,2,11111111111111
cat /***/gds_data_1/postgres_public_foreign_test_pipe_tw.txt
1,2,11111111111111
2,2,11111111111111
4,2,11111111111111
```

----结束

3.6 其他操作

3.6.1 GDS 管道文件常见问题

注意事项

- GDS 支持并发导入导出，gds -t 参数用于设置 gds 的工作线程池大小，控制并发场景下同时工作的工作线程数且不会加速单个 sql 任务。gds -t 缺省值为 8，上限值为 200。在使用管道功能进行导入导出时，-t 参数应不低于业务并发数。如果是双集群互联互通场景，-t 参数应不低于业务并发数的两倍。
- 由于管道“读取即删除”的特点，需确保导入或导出过程中除 GDS 程序外无其他程序读取管道文件，避免导入过程中数据丢失或者任务报错及导出的文件内容混乱。
- 不支持对具有相同 location 的外表并发导入导出，即 GDS 的多个线程同时读取管道文件或者同时写入管道文件。
- GDS 的单个导入导出任务只识别一个管道文件，因此不要对 GDS 外表设置带有通配符({}[]?)的 location 地址。如：

```
CREATE FOREIGN TABLE foreign_test_pipe_tr( like test_pipe ) SERVER gsmpp_server
OPTIONS (LOCATION 'gsfs://192.168.0.1:7789/foreign_test_*', FORMAT 'text',
DELIMITER ',',  NULL '', EOL '0x0a' ,file_type 'pipe',auto_create_pipe 'false');
```

- GDS 启用 - r 递归参数时只识别一个管道文件，即 GDS 只会识别当前数据目录下的一个管道文件而不会递归寻找，因此-r 参数在管道文件导入导出场景下不生效。
- 管道文件的导入导出不支持 CN Retry，因为 GDS 无法控制对端用户和程序操作管道的行为。
- GDS 导入时默认对端程序超过 1 小时未向管道中写入数据导入任务将会超时报错。
- GDS 导出时默认对端程序超过 1 小时未从管道中读数据导出任务将会超时报错。
- 确保 GDS 版本和内核版本都已经支持管道文件导入导出功能。
- 当外表参数 auto_create_pipe 设置为 true 时，GDS 自动创建管道文件可能存在延迟，因此操作管道文件时建议先判断自动创建的管道文件是否存在，且是否为管道文件类型。
- GDS 管道文件的导入导出任务结束后会自动删除管道文件，但是手动终止任务时，管道文件的删除会有延迟，直到到达超时时间后才会被删除。

常见问题和定位方法：

- 问题 1："/***/postgres_public_foreign_test_pipe_tr.pipe" must be named pipe.

  定位方法：GDS 的外表 file_type 类型为 pipe 但是操作的文件却是一个普通文件类型。应该排查 postgres_public_foreign_test_pipe_tr.pipe 是否是为管道文件。
- 问题 2：could not open pipe "/***/postgres_public_foreign_test_pipe_tw.pipe" cause by Permission denied.

  定位方法：GDS 没有权限打开管道文件。
- 问题 3：could not open source file /*****/postgres_public_foreign_test_pipe_tw.pipe because timeout 300s for WRITING.

定位方法：GDS 导出时打开管道文件超时，一般由于 auto_create_pipe 为 false 时候，管道文件在 300 秒内未被创建，或者创建了但是 300 秒内没有程序读取该管道文件。

- 问题 4：could not open source file /*****/postgres_public_foreign_test_pipe_tw.pipe because timeout 300s for READING.

  定位方法：GDS 导出时打开管道文件超时，一般由于 auto_create_pipe 为 false 时候，管道文件在 300 秒内未被创建或者创建了但是 300 秒之内没有程序写入该管道文件。

- 问题 5：could not poll writing source pipe file "/****/postgres_public_foreign_test_pipe_tw.pipe" timeout 300s.

  定位方法：GDS 导出时超过 300 秒未等到管道上的写事件，一般由于该管道文件超过 300 秒没有被读取。

- 问题 6：could not poll reading source pipe file "/****/postgres_public_foreign_test_pipe_tw.pipe" timeout 300s.

  定位方法：GDS 导入时超过 300 秒未等到管道上的读事件，一般由于该管道文件超过 300 秒没有被写入。

- 问题 7：could not open pipe file "/***/postgres_public_foreign_test_pipe_tw.pipe" for "WRITING" with error No such device or address.

  定位方法：表示当前"/***/postgres_public_foreign_test_pipe_tw.pipe"管道文件没有程序正在读取导致 GDS 无法以写的方式打开管道文件。

3.6.2 查看数据倾斜状态

操作场景

数据倾斜会造成查询表性能下降。对于记录数超过千万条的表，建议在执行全量数据导入前，先导入部分数据，以进行数据倾斜检查和调整分布列，避免导入大量数据后发现数据倾斜，调整成本高。

背景信息

GaussDB(DWS)是采用 Shared-nothing 架构的 MPP（Massive Parallel Processor，大规模并发处理）系统，采用水平分布的方式，将业务数据表的元组按合适的分布策略分散存储在所有的 DN。

当前产品支持复制（Replication）、散列（Hash）和轮询（Roundrobin）三种用户表分布策略。

- Replication 方式：在每一个 DN 上存储一份全量表数据。对于数据量比较小的表建议采取 Replication 分布策略。
- Hash 方式：采用这种分布方式，需要为用户表指定一个分布列（distribute key）。当插入一条记录时，系统会根据分布列的值进行 hash 运算后，将数据存储在对应的 DN 中。对于数据量比较大的表建议采取 Hash 分布策略。
- Roundrobin 方式：表的每一行被轮番地发送给各个 DN，因此数据会被均匀地分布在各个 DN 中。对于数据量比较大的表，如果 Hash 分布找不到一个合适的分布列，建议采用 Roundrobin 分布策略。

对于 Hash 分布策略，如果分布列选择不当，可能导致数据倾斜。因此在采用 Hash 分布策略之后会对用户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。一般情况下分布列都是选择键值重复度小，数据分布比较均匀的列。

操作步骤

步骤 1 分析数据源特征，选择若干个键值重复度小，数据分布比较均匀的备选分布列。

步骤 2 从步骤 1 中选择一个备选分布列创建目标表。

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE [ IF NOT
EXISTS ] table_name
    ({ column_name data_type [ compress_mode ] [ COLLATE collation ]
[ column_constraint [ ... ] ]
    | table_constraint    | LIKE source_table [ like_option [...] ] }
    [, ... ])    [ WITH ( {storage_parameter = value} [, ... ] ) ]
    [ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
    [ COMPRESS | NOCOMPRESS ]    [ TABLESPACE tablespace_name ]
    [ DISTRIBUTE BY { REPLICATION
                  | ROUNDROBIN
                  | { HASH ( column_name [,...] ) } } ];
```

步骤 3 参照前面章节中的办法向目标表中导入小批量数据。

对于单个数据源文件，在导入时，可通过均匀切割，导入部分切割后的数据源文件来验证数据倾斜性。

步骤 4 检验数据倾斜性。命令中的 table_name ，请填入实际的目标表名。

```
SELECT a.count,b.node_name FROM (SELECT count(*) AS count,xc_node_id FROM
table name GROUP BY xc_node_id) a, pgxc_node b WHERE a.xc_node_id=b.node_id ORDER
BY a.count desc;
```

步骤 5 若各 DN 上数据分布差小于 10%，表明数据分布均衡，选择的分布列合适。请清理已导入小批量数据，导入全量数据，以完成数据迁移。

若各 DN 上数据分布差大于等于 10%，表明数据分布倾斜，请从步骤 1 的备选分布列中删除该列，删除目标表，并重复步骤 2 、步骤 3 、步骤 4 和步骤 5。

说明

此处的数据分布差表示实际查询到 DN 上的数据量与 DN 平均数据量的差异。

步骤 6 （可选）如果上述步骤不能选出适合的分布列，需要从备选分布列选择多个列的组合作为分布列来完成数据迁移。

----结束

示例

对目标表 staffs 选择合适的分布列。

1. 分析表 staffs 的数据源特征，选择数据重复度低且分布均匀的备选分布列 staff_ID、FIRST_NAME 和 LAST_NAME。
2. 先选择 staff_ID 作为分布列，创建目标表 staffs。

```
CREATE TABLE staffs
(
  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(2,2),
  MANAGER_ID    NUMBER(6),
  section_ID    NUMBER(4)
)
DISTRIBUTE BY hash(staff_ID);
```

3. 向目标表 staffs 中导入部分数据。

   根据以下查询所得，集群环境中主 DN 数为 8 个，则建议导入的记录数为 80000 条。

```
SELECT count(*) FROM pgxc_node where node_type='D';
 count
-------
     8
(1 row)
```

4. 校验以 staff_ID 为分布列的目标表 staffs 的数据倾斜性。

```
SELECT a.count,b.node_name FROM (select count(*) as count,xc_node_id FROM
staffs GROUP BY xc_node_id) a, pgxc_node b WHERE a.xc_node_id=b.node_id ORDER
BY a.count desc;
count | node_name
------+-----------
11010 | datanode4
10000 | datanode3
12001 | datanode2
 8995 | datanode1
10000 | datanode5
 7999 | datanode6
 9995 | datanode7
10000 | datanode8
(8 rows)
```

5. 根据上一步骤查询所得，各 DN 上数据分布差大于 10%，数据分布倾斜。所以从步骤 1 的备选分布列中删除该列，并删除目标表 staffs。

```
DROP TABLE staffs;
```

6. 尝试选择 staff_ID、FIRST_NAME 和 LAST_NAME 的组合作为分布列，创建目标表 staffs。

```
CREATE TABLE staffs
(
  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(2,2),
  MANAGER_ID     NUMBER(6),
  section_ID     NUMBER(4)
)
DISTRIBUTE BY hash(staff_ID,FIRST_NAME,LAST_NAME);
```

7. 校验以 staff_ID、FIRST_NAME 和 LAST_NAME 的组合为分布列的目标表 staffs 的数据倾斜性。

```
SELECT a.count,b.node_name FROM (select count(*) as count,xc_node_id FROM
staffs GROUP BY xc_node_id) a, pgxc_node b WHERE a.xc_node_id=b.node_id ORDER
BY a.count desc;
count | node_name
------+-----------
10010 | datanode4
10000 | datanode3
10001 | datanode2
 9995 | datanode1
10000 | datanode5
 9999 | datanode6
 9995 | datanode7
10000 | datanode8
(8 rows)
```

8. 根据上一步骤查询所得，各 DN 上数据分布差小于 10%，数据分布均衡，选择的分布列合适。
9. 清理已导入小批量数据。

```
TRUNCATE TABLE staffs;
```

10. 导入全量数据，以完成数据迁移。

3.6.3 分析表

执行计划生成器需要使用表的统计信息，以生成最有效的查询执行计划，提高查询性能。因此数据导入完成后，建议执行 ANALYZE 语句生成最新的表统计信息。统计结果存储在系统表 PG_STATISTIC 中。

分析表

ANALYZE 支持的表类型有行/列存表、HDFS 表、ORC/CARBONDATA 格式的 OBS 外表。ANALYZE 同时也支持对本地表的指定列进行信息统计。下面以表的 ANALYZE 为例，更多关于 ANALYZE 的信息，请参见 ANALYZE | ANALYSE。

步骤 1 更新表统计信息。

以表 product_info 为例，ANALYZE 命令如下：

```
ANALYZE product_info;
```

----结束

表自动分析

GaussDB(DWS)提供了三种场景下表的自动分析。

- 当查询中存在“统计信息完全缺失”或“修改量达到 analyze 阈值”的表，且执行计划不采取 FQS (Fast Query Shipping)执行时，则通过 GUC 参数 autoanalyze 控制此场景下表统计信息的自动收集。此时，查询语句会等待统计信息收集成功后，生成更优的执行计划，再执行原查询语句。
- 当 autovacuum 设置为 on 时，系统会定时启动 autovacuum 线程，对“修改量达到 analyze 阈值”的表在后台自动进行统计信息收集。

表3-24 表自动分析

触发方式	触发条件	触发频率	控制参数	备注
同步	统计信息完全缺失	查询时	autoanalyze	truncate 主表时会清空统计信息。
同步	数据修改量达到 analyze 阈值	查询时	autoanalyze	先触发 analyze，后选择最优计划。
异步	数据修改量达到 analyze 阈值	autovacuum 线程轮询检查	autovacuum_mode, autovacuum_naptime	2s 等锁超时, 5min 执行超时。

须知

- autoanalyze 只支持默认采样方式，不支持百分比采样方式。
- 多列统计信息仅支持百分比采样，因此 autoanalyze 不收集多列统计信息。
- 查询过程因表的“统计信息完全缺失”和“修改量达到 analyze 阈值”而自动触发 autoanalyze 的场景，当前不支持对外表触发 autoanalyze，不支持对带有 ON COMMIT [DELETE ROWS | DROP]选项的临时表触发 autoanalyze。
- 修改量达到 analyze 阈值是指：表的修改量超过 autovacuum_analyze_threshold + autovacuum_analyze_scale_factor * reltuples，其中 reltuples 是 pg_class 中记录的表的估算行数。
- 基于定时启动的 autovacuum 线程触发的 autoanalyze，仅支持行存表和列存表，不支持外表、HDFS 表、OBS 外表、临时表、unlogged 表和 toast 表。
- 查询时触发 analyze 会对分区表的所有分区加四级锁，直到查询所在事务提交后才会放锁。四级锁不堵塞增删改查，但会堵塞分区的修改操作，比如分区的 truncate，可以通过将 object_mtime_record_mode 设置为 disable_partition，实现提前释放分区锁。
- autovacuum 自动清理功能的生效还依赖于下面两个 GUC 参数：
- track_counts 参数需要设置为 on，开启收集收据库统计数据功能。
- autovacuum_max_workers 参数需要大于 0，该参数表示能同时运行的自动清理线程的最大数量。

4 冷热数据管理

冷热数据简介

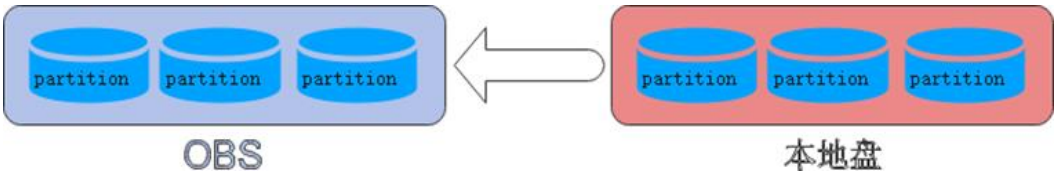
海量大数据场景下，随着业务和数据量的不断增长，数据存储与消耗的资源也日益增长。根据业务系统中用户对不同时期数据的不同使用需求，对膨胀的数据本身进行“冷热”分级管理，不仅可以提高数据分析性能还能降低业务成本。

例如，在网络流量分析系统中，用户可能对最近一个月内安全事件和网络访问情况感兴趣，而很少关注几个月前的数据。针对这样的一些场景，可以将数据按照时间分为：热数据、冷数据。

冷热数据主要从数据访问频率、更新频率进行划分。

- Hot（热数据）：访问、更新频率较高，未来被调用的概率较高的数据，对访问的响应时间要求很高的数据。
- Cold（冷数据）：不允许更新或更新频率比较低，访问频率比较低，对访问的响应时间要求不高的数据。

用户可以定义冷热管理表，将符合规则的冷数据切换至 OBS 上进行存储，可以按照分区自动进行冷热数据的判断和迁移。



冷热数据迁移

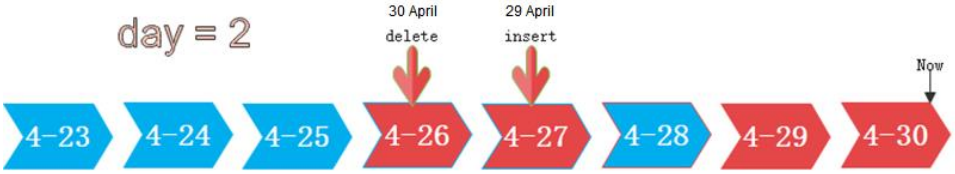
GaussDB(DWS)列存数据写入时，数据首先进入热分区进行存储，分区数据较多后，可通过手动或自动的方式，将符合冷数据规则的数据切换至 OBS 上进行存储。在数据切换至 OBS 上后，其元数据、Desc 表信息以及索引信息仍在本地进行存储，保证了读取的性能。

冷热切换策略

目前冷热切换的策略名称支持 LMT（last modify time）和 HPN（hot partition number），LMT 指按分区的最后更新时间切换，HPN 指保留热分区的个数切换。

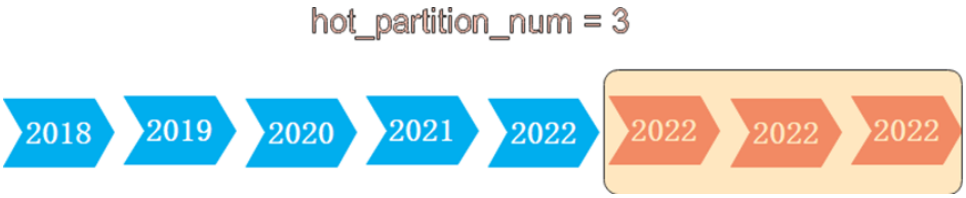
- LMT：表示切换[day]时间前修改的热分区数据为冷分区，将该数据迁至 OBS 表空间中。其中[day]为整型，范围[0, 36500]，单位为天。

  如下图中，设置 day 为 2，即在冷热切换时，根据分区数据的最晚修改时间，保留 2 日内所修改的分区为热分区，其余数据为冷分区数据。假设当前时间为 4 月 30 日，4 月 30 日对[4-26]分区进行了 delete 操作，4 月 29 日对[4-27]分区进行了 insert 操作，故在冷热切换时，保留[4-26][4-27][4-29][4-30]四个分区为热分区。



- HPN：表示保留 HPN 个有数据的分区为热分区。分区顺序按照分区的 Sequence ID 来确定，分区的 Sequence ID 是根据分区边界值的大小，内置生成的序号，此序号不对外呈现。对于 RANGE 分区，分区的边界值越大，分区对应的 Sequence ID 越大；对于 LIST 分区，分区边界枚举值中的最大值越大，分区对应的 Sequence ID 越大。在冷热切换时，需要将数据迁移至 OBS 表空间中。其中 HPN 为整型，范围为[0,1600]。其中 HPN 为 0 时，表示不保留热分区，在进行冷热切换时，将所有有数据的分区都转为冷分区并存储在 OBS 上。

  如下图中，设置 HPN 为 3，即在冷热切换时，保留最新的 3 个有数据的分区为热分区数据，其余分区均切为冷分区。



注意事项

1. 冷热数据管理支持功能：
   - 支持对冷热表的 insert、copy、delete、update、select 等表相关的 DML 操作。
   - 支持对冷热表的权限管理等 DCL 操作。
   - 支持对冷热表进行 analyze、vacuum、merge into 等操作和一些分区的操作。
   - 支持从普通列存分区表升级为冷热数据表。
   - 支持带有冷热数据管理表的升级、扩容、缩容和重分布。
2. 冷热数据管理的使用约束限制：
   - 目前冷热表只支持列存 2.0 版本的分区表，外表不支持冷热分区。
   - 仅支持从热数据切换为冷数据，不支持从冷数据切换为热数据。对于已经切冷分区再次插入数据，数据直接会进入 OBS，不会改变分区的冷热属性。
   - 对于同一分区在同一 DN 只会存在冷或热的一种情况，对于同一分区在不同 DN 可能存在部分 DN 为热数据，部分 DN 为冷数据。
   - 对于同时存在冷热分区的表，查询时会变慢，因为冷数据存储在 OBS 上，读写速度和时延都比在本地查询要慢。
   - 只支持修改冷热表的冷热切换策略，不支持修改冷热表的冷数据的表空间。

- 冷热表的分区操作约束：
  - 不支持对冷分区的数据进行 exchange 操作。
  - Merge partition 分区只支持热分区和热分区合并、冷分区和冷分区合并，不支持冷热分区合并。
  - ADD/Merge/Split Partition 等分区操作不支持指定表空间为 OBS 表空间。
  - 不支持创建时指定和修改冷热表分区的表空间。
- 冷热切换不是只要满足条件就立刻进行冷热数据切换，依赖用户手动调用切换命令，或者通过调度器调用切换命令后才真正进行数据切换。目前自动调度时间为每日 0 点，可进行修改。
- 目前冷热切换规则只支持 LMT 和 HPN 两种。
- 冷热数据表不支持物理细粒度备份和恢复，由于物理备份时只备份热数据，在备份恢复前后 OBS 上冷数据为同一份，不支持 truncate 和 drop table 等涉及删除文件操作语句的备份恢复操作。

使用示例

1. 创建列存冷热数据管理表，指定热数据有效期 LMT 为 100 天。

```
CREATE TABLE lifecycle_table(i int, val text) WITH (ORIENTATION = COLUMN,
storage_policy = 'LMT:100')
PARTITION BY RANGE (i)
(
PARTITION P1 VALUES LESS THAN(5),
PARTITION P2 VALUES LESS THAN(10),
PARTITION P3 VALUES LESS THAN(15),
PARTITION P8 VALUES LESS THAN(MAXVALUE)
)ENABLE ROW MOVEMENT;
```

2. 切换冷数据至 OBS 表空间。
   - 自动切换：每日 0 点调度框架自动触发，无需关注切换情况；

     可自定义自动切换时间：根据业务情况调整自动触发时间，修改为每天早晨 6 点 30 分:；

     ```
     select * from pg_obs_cold_refresh_time('lifecycle_table', '06:30:00');
     ```
   - 手动切换

     执行如下操作手动切换单表：

     ```
     alter table lifecycle_table refresh storage;
     ```

     执行如下操作批量切换所有冷热表：

     ```
     select pg_catalog.pg_refresh_storage();
     ```
3. 查看冷热表数据分布情况。

   查看单表数据分布情况：

   ```
   select * from pg_catalog.pg_lifecycle_table_data_distribute('lifecycle_table');
   ```

   查看所有冷热表数据分布情况：

   ```
   select * from pg_catalog.pg_lifecycle_node_data_distribute();
   ```

5 Oracle、Teradata 和 MySQL 语法兼容性差异

GaussDB(DWS)支持 Oracle、Teradata 和 MySQL 三种兼容模式，分别兼容 Oracle、Teradata 和 MySQL 语法，不同兼容模式下的语法行为有一些差异。

表5-1 兼容项差异

兼容项	Oracle 兼容	Teradata 兼容	MySQL 兼容
空串	只有 null	区分空串和 null	区分空串和 null
空串转数字	null	转换为 0	转换为 0
超长字符自动截断	不支持	支持(GUC 参数 td_compatible_truncation 设置为 ON)	不支持
null 拼接	非 null 对象与 null 拼接后返回非 null 对象。 例如，'abc'||null 返回'abc'。	GUC 参数 behavior_compat_options 增加 strict_text_concat_td 选项后，兼容 TD 行为，null 类型拼接后返回 null。 例如，'abc'||null 返回 null。	兼容 MySQL 行为，null 类型拼接后返回 null。 例如，'abc'||null 返回 null。
char(n)类型拼接	char(n)类型做拼接时移除右侧空格和占位。 例如，cast('a' as char(3))||'b'返回'ab'。	GUC 参数 behavior_compat_options 增加 bpchar_text_without_rtrim 选项后，char(n)类型做拼接时，保留空格，并补足空格至指定的 n 长度。 当前不支持“比较字符串时忽略尾部空	移除右侧空格和占位

兼容项	Oracle 兼容	Teradata 兼容	MySQL 兼容
		格”，拼接后结果如果存在尾部空格，进行比较时会对空格敏感。 例如，cast('a' as char(3))||'b'返回'a b'。	
concat(str1,str2)	返回所有非 null 字符串的连接	返回所有非 null 字符串的连接	入参中存在 null 时，返回结果为 null。
left 和 right 负数处理	返回除最后/前|n|个字符以外的所有字符	返回除最后/前|n|个字符以外的所有字符	返回空串
lpad(string text, length int [, fill text]) rpad(string text, length int [, fill text])	通过填充字符 fill(缺省为空格)，把 string 填充到 length 长度，如果 string 已经比 length 长则将其尾部截断。如果 fill 为空串或者 length 为负数则返回 null。	如果 fill 为空串且 string 长度小于 length 时，返回原字符串，如果 length 为负数则返回空串。	如果 fill 为空串且 string 长度小于 length 时，返回空串，如果 length 为负数则返回 null。
substr(str, s[, n])	s = 0 时，返回前 n 个字符	s = 0 时，返回前 n 个字符。	s = 0 时，返回空串。
substring(str, s[, n]) substring(str [from s] [for n])	s = 0 时，返回前 n - 1 个字符 s < 0 时，返回前 s + n - 1 个字符 n < 0 时，报错。	s = 0 时，返回前 n - 1 个字符。 s < 0 时，返回前 s + n - 1 个字符。 n < 0 时，报错。	s = 0 时，返回空串。 s < 0 时，倒数第|s|个字符位置开始截取 n 个字符。 n < 0 时，返回空串。
trim、ltrim、rtrim、btrim(string[,characters])	从字符串 string 的指定位置删除只包含 characters 中字符（缺省为空格）的最长的字符串。	从字符串 string 的指定位置删除只包含 characters 中字符（缺省为空格）的最长的字符串。	从字符串 string 的指定位置删除等于 characters 的字符串（缺省为空格）。
log(x)	以 10 为底的对数	以 10 为底的对数	自然对数
mod(x, 0)	除数为 0 时返回 x	除数为 0 时返回 x	除数为 0 时报错
数据类型 date	date 会转为 timestamp，包含年月日时分秒	只有年月	只有年月

兼容项	Oracle 兼容	Teradata 兼容	MySQL 兼容
to_char(date)	入参最大值仅支持 timestamp 类型的最大值，不支持 date 类型的最大值；返回值类型为 timestamp	入参最大值仅支持 timestamp 类型的最大值，不支持 date 类型的最大值；返回值类型为 date，且格式为 'YYYY/MM/DD'(GUC 参数 convert_empty_str_to_null_td 打开)。	入参最大值仅支持 timestamp 类型的最大值，不支持 date 类型的最大值；返回值类型为 date。
to_date, to_timestamp 和 to_number 空串处理	返回 null	返回 null(GUC 参数 convert_empty_str_to_null_td 打开)	to_date 和 to_timestamp 返回 null，to_number 中参数为空串时，返回 0。
last_day 和 next_day 返回类型	timestamp 类型	timestamp 类型	date 类型
add_months 返回类型	timestamp 类型	timestamp 类型	入参为 date 类型，返回 date 类型。 入参为 timestamp 类型，返回 timestamp 类型。 入参为 timestamptz 类型，返回 timestamptz 类型。
CURRENT_TIME CURRENT_TIME(p)	获取当前事务的时间，返回值类型为 timetz	获取当前事务的时间，返回值类型为 timetz。	获取当前语句执行时的时间，返回值类型为 time。
CURRENT_TIMESTAMP CURRENT_TIMESTAMP(p)	获取当前语句执行时的时间，返回值类型为 timestamptz	获取当前语句执行时的时间，返回值类型为 timestamptz。	获取当前语句执行时的时间，返回值类型为 timestamp。
LOCALTIME LOCALTIME(p)	获取当前事务的时间，返回值类型为 time	获取当前事务的时间，返回值类型为 time。	获取当前语句执行时的时间，返回值类型为 time。
LOCALTIMESTAMP LOCALTIMESTAMP(p)	获取当前事务的时间，返回值类型为 timestamp	获取当前事务的时间，返回值类型为 timestamp。	获取当前语句执行时的时间，返回值类型为 timestamp。
SYSDATE	获取当前语句执行	获取当前语句执行时	获取当前系统的时

兼容项	Oracle 兼容	Teradata 兼容	MySQL 兼容
SYSDATE(p)	时的时间，返回值类型为 timestamp(0)	的时间，返回值类型为 timestamp(0)。	间，返回值类型为 timestamp(0)。
now()	获取当前事务时间，返回值类型为 timestamptz	获取当前事务时间，返回值类型为 timestamptz。	获取语句执行的时间，返回值类型为 timestamptz。
操作符'^'	幂运算	幂运算	异或
表达式 greatest、least	返回所有非 null 入参的比较结果	返回所有非 null 入参的比较结果	入参中存在 null 时，返回结果为 null。
表达式 case、coalesce、if、ifnull 入参类型不同	报错	兼容 TD 行为，支持数字和字符串之间的类型转换，比如 coalesce 参数输入 int 和 varchar 类型，解析成 varchar 类型。	兼容 MySQL 行为，支持其他类型和字符串之间的类型转换，比如 coalesce 参数输入 date、int 和 varchar 类型，解析成 varchar 类型。

6 管理数据库安全

6.1 管理用户及权限

6.1.1 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。集群安装后的默认情况下，未开启 6.1.3 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(DWS)支持以下的权限：SELECT、INSERT、UPDATE、DELETE、TRUNCATE、REFERENCES、CREATE、CONNECT、EXECUTE、USAGE 和 ANALYZE|ANALYSE。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、GRANT 和 REVOKE）是隐式的，无法授予或撤销。即只要拥有对象就可以执行对象所有者的这些隐式权限。对象所有者可以撤销自己的普通权限，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 14 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。

6.1.2 系统管理员

系统管理员是指具有 SYSADMIN 属性的帐户。集群安装后，默认情况下系统管理员具有与对象所有者相同的权限。

在启动 GaussDB(DWS)集群时创建的用户 dbadmin 是系统管理员。

要创建新的数据库管理员，请以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。

```
CREATE USER sysadmin WITH SYSADMIN password 'password';
```

或者

```
ALTER USER joe SYSADMIN;
```

ALTER USER 时，要求用户已存在。

6.1.3 三权分立

6.1.1 默认权限机制和 6.1.2 系统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，将系统管理员的权限分立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，并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。

三权分立后，系统管理员只会对自己作为所有者的对象有权限。

三权分立的设置办法请参考《数据仓库服务用户指南》中的“管理集群 > 设置集群安全配置 > 设置三权分立”章节。

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 6-1 和表 6-2。

表6-1 默认的用户权限

对象名称	系统管理员	安全管理员	审计管理员	普通用户
表空间	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。		
表	对所有表有所有的权限。	仅对自己的表有所有的权限，对其他用户的表无权限。		
索引	可以在所有的表上建立索引。	仅可以在自己的表上建立索引。		
模式	对所有模式有所有的权限。	仅对自己的模式有所有的权限，对其他用户的模式无权限。		
函数	对所有的函数有所有的权限。	仅对自己的函数有所有的权限，对其他用户放在 public 这个公共模式下的函数有调用的权限，对其他用户放在其他模式下的函数无权限。		
自定义	对所有的视图有所有	仅对自己的视图有所有的权限，对其他用户的视图		

对象名称	系统管理员	安全管理员	审计管理员	普通用户
视图	的权限。	无权限。		
系统表和系统视图	可以查看所有系统表和视图。	只可以查看部分系统表和视图。详细请参见 14 系统表和系统视图。		

表6-2 三权分立较非三权分立权限变化说明

对象名称	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	无变化。		
表	权限缩小。 只对自己的表有所有权限，对其他用户放在属于各自模式下的表无权限。	无变化。		
索引	权限缩小。 只可以在自己的表上建立索引。	无变化。		
模式	权限缩小。 只对自己的模式有所有的权限，对其他用户的模式无权限。	无变化。		
函数	权限缩小。 只对自己的函数有所有的权限，对其他用户放在属于各自模式下的函数无权限。	无变化。		
自定义视图	权限缩小。 只对自己的视图及其他用户放在 public 模式下的视图有所有的权限，对其他用户放在属于各自模式下的视图无权限。	无变化。		
系统表和系统视图	无变化。	无变化。	无变化。	无权查看任何系统表和视图。

6.1.4 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 6.1.3 三权分立下，GaussDB(DWS)用户帐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，用户帐户只能由安全管理员创建。

在用户登录 GaussDB(DWS)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以控制谁可以访问哪个对象。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。

  ```
  CREATE USER joe WITH CREATEDB PASSWORD 'password';
  ```

- 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。
- 要更改用户帐户（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 14.3.159 PG_USER：

  ```
  SELECT * FROM pg_user;
  ```

- 要查看用户属性，请查询系统表 14.2.13 PG_AUTHID：

  ```
  SELECT * FROM pg_authid;
  ```

私有用户

对于有多个业务部门，各部门间使用不同的数据库用户进行业务操作，同时有一个同级的数据库维护部门使用数据库管理员进行维护操作的场景下，业务部门可能希望在未经授权的情况下，管理员用户只能对各部门的数据进行控制操作（DROP、ALTER、TRUNCATE），但是不能进行访问操作（INSERT、DELETE、UPDATE、SELECT、COPY）。即针对管理员用户，表对象的控制权和访问权要能够分离，提高普通用户数据安全性。

6.1.3 三权分立情况下，管理员对其他用户放在属于各自模式下的表无权限。但是，这种无权限包含了无控制权限，因此不能满足上面的诉求。为此，GaussDB(DWS)提供了私有用户方案。即在非三权分立模式下，创建具有 INDEPENDENT 属性的私有用户。

```
CREATE USER user_independent WITH INDEPENDENT IDENTIFIED BY "password";
```

针对该用户的对象，数据库管理员在未经其授权前，只能进行控制操作（DROP、ALTER、TRUNCATE），无权进行 INSERT、DELETE、SELECT、UPDATE、COPY、GRANT、REVOKE、ALTER OWNER 操作。

6.1.5 角色

角色是一组权限的集合。

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所需数据的差异权限。在角色级别授予或撤销权限时，这些权限更改会对角色下的所有成员生效。

GaussDB(DWS)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 语法。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。

预置角色

GaussDB(DWS)提供了一组预置角色，以 gs_role_开头命名，提供对特定的、通常需要高权限的操作的访问，可以将这些角色授权予数据库中的其他用户或角色，使这些用户能够访问或使用特定的信息和功能。请谨慎使用预置角色，以确保预置角色权限的安全使用。

预置角色允许的权限范围可参考下表：

表6-3 预置角色允许的权限范围

角色	权限描述
gs_role_signal_backend	具有调用函数 pg_cancel_backend、pg_terminate_backend、pg_terminate_query、pg_cancel_query、pgxc_terminate_query、pgxc_cancel_query 来取消或终止其他会话的权限，但不能操作属于初始用户的会话。
gs_role_read_all_stats	读取系统状态视图并且使用与扩展相关的各种统计信息，包括有些通常只对系统管理员可见的信息。包括： 资源管理类： • pgxc_wlm_operator_history • pgxc_wlm_operator_info • pgxc_wlm_operator_statistics • pgxc_wlm_session_info • pgxc_wlm_session_history • pgxc_wlm_session_statistics • pgxc_wlm_workload_records • pgxc_workload_sql_count • pgxc_workload_sql_elapse_time • pgxc_workload_transaction 状态信息类： • pgxc_stat_activity • pgxc_get_table_skewness • table_distribution • pgxc_total_memory_detail • pgxc_os_run_info

角色	权限描述
	• pg_nodes_memory • pgxc_instance_time • pgxc_redo_stat
gs_role_analyze_any	具有系统级 ANALYZE 权限类似系统管理员用户，跳过 schema 权限检查，对所有的表可以执行 ANALYZE。
gs_role_vacuum_any	具有系统级 VACUUM 权限类似系统管理员用户，跳过 schema 权限检查，对所有的表可以执行 VACUUM。

预置角色的使用约束：

- 以 gs_role_开头的角色名作为数据库的预置角色保留字，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。
- 禁止对预置角色执行 ALTER 和 DROP 操作。
- 预置角色默认没有 LOGIN 权限，不设置预置登录密码。
- gsql 元命令\du 和\dg 不显示预置角色的相关信息，但若指定了 PATTERN（用来指定要被显示的对象名称）则预置角色信息会显示。
- 三权分立关闭时，系统管理员和具有预置角色 ADMIN OPTION 权限的用户有权对预置角色执行 GRANT/REVOKE 管理；三权分立打开时，安全管理员（具有 CREATEROLE 属性）和具有预置角色 ADMIN OPTION 权限的用户有权对预置角色执行 GRANT/REVOKE 管理。例如：

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

查看所有角色

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 6.1.3 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帐户。

6.1.6 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 Schema，且所有用户都拥有此 Schema 的权限。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。

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。

在当前数据库中创建用户时，系统会在当前数据库中为新用户创建一个同名 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。任何用户都可以创建 Schema。

```
CREATE SCHEMA schema_name;
```

- 要更改 Schema 名称或者所有者，请使用 ALTER SCHEMA。只有 Schema 所有者可以更改 Schema。

```
ALTER SCHEMA schema_name RENAME TO new_name;
ALTER SCHEMA schema_name OWNER TO new_owner;
```

- 要删除 Schema 及其对象，请使用 DROP SCHEMA。只有 Schema 所有者可以删除 Schema。

```
DROP SCHEMA schema_name;
```

- 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。

```
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。

```
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。

```
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，则将该对象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。

```
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。

```
SET SEARCH_PATH TO myschema, public;
SET
```

6.1.7 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字，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。

```
GRANT USAGE ON SCHEMA tpcds TO joe;
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：

```
CREATE ROLE lily WITH CREATEDB PASSWORD '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。

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

  c. 将角色的权限赋予用户。

```
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。

6.1.8 设置安全策略

6.1.8.1 设置帐户安全策略

背景信息

GaussDB(DWS)为帐户提供了自动锁定和解锁帐户、手动锁定和解锁异常帐户和删除不再使用的帐户等一系列的安全措施，保证数据安全。

自动锁定和解锁帐户

- 为了保证帐户安全，如果连接数据库时输入密码次数超过 10 次，系统将自动锁定该帐户。
- 帐户被锁定时间超过 1 天后，自动解锁。

手动锁定和解锁帐户

若管理员发现某帐户被盗、非法访问等异常情况，可手动锁定该帐户。

当管理员认为帐户恢复正常后，可手动解锁该帐户。

以手动锁定和解锁用户 joe 为例，用户的创建请参见 6.1.4 用户，命令格式如下：

- 手动锁定

```
ALTER USER joe ACCOUNT LOCK;
```

- 手动解锁

```
ALTER USER joe ACCOUNT UNLOCK;
```

删除不再使用的帐户

当确认帐户不再使用，管理员可以删除帐户。该操作不可恢复。

当删除的用户正处于活动状态时，此会话状态不会立马断开，用户在会话状态断开后才会被完全删除。

以删除帐户 joe 为例，命令格式如下：

```
DROP USER joe CASCADE;
```

6.1.8.2 设置帐号有效期

注意事项

创建新用户时，需要限制用户的操作期限（有效开始时间和有效结束时间）。

不在有效操作期内的用户需要重新设定帐号的有效操作期。

操作步骤

步骤 1 创建用户并制定用户的有效开始时间和有效结束时间。

```
CREATE USER joe WITH PASSWORD '********' VALID BEGIN '2015-10-10 08:00:00' VALID
UNTIL '2016-10-10 08:00:00';
```

显示如下信息表示创建用户成功。

```
CREATE USER
```

步骤 2 用户已不在有效使用期内，需要重新设定帐号的有效期，这包括有效开始时间和有效结束时间。

```
ALTER USER joe WITH VALID BEGIN '2016-11-10 08:00:00' VALID UNTIL '2017-11-10
08:00:00';
```

显示如下信息表示重新设定成功。

```
ALTER USER
```

----结束

说明

若在“CREATE ROLE”或“ALTER ROLE”语法中不指定“VALID BEGIN”，表示不对用户的开始操作时间做限定；若不指定“VALID UNTIL”，表示不对用户的结束操作时间做限定；若两者均不指定，表示该用户一直有效。

6.1.8.3 设置用户密码

用户密码存储在系统表 pg_authid 中，为防止用户密码泄露，GaussDB(DWS)对用户密码进行加密存储。

- 密码复杂度

  帐户密码的复杂度要求如下：

  - 包含大写字母（A-Z）的个数为 0~999，包含小写字母（a-z）的个数为 0~999，包含数字（0-9）的个数为 0~999，包含特殊字符的个数为 0~999（特殊字符的列表请参见表 6-4）。

  说明

  帐户密码至少包含上述四类字符中的三类。

  - 密码的最小长度 6，默认值为 8。
  - 密码的最大长度 999，默认值为 32。
  - 不能和用户名、用户名倒写相同。
  - 不能和当前密码、当前密码的倒写相同。
- 密码重用

  用户修改密码时，只有持续未使用天数超过 60 天的历史密码才可以被重新使用。
- 密码有效期限

  数据库用户的密码都有密码有效期（默认值为 90 天），当达到密码到期提醒天数（7 天）时，系统会在用户登录数据库时提示用户修改密码。

说明

考虑到数据库使用特殊性及业务连续性，密码过期后用户还可以登录数据库，但是每次登录都会提示修改密码，直至修改为止。

- 密码修改
  - 在安装数据库时，会新建一个和初始化用户重名的操作系统用户，为了保证帐户安全，请定期修改操作系统用户的密码。

    以修改用户 user1 密码为例，命令格式如下：

    ```
    passwd user1
    ```

    根据提示信息完成修改密码操作。
  - 建议系统管理员和普通用户都要定期修改自己的帐户密码，避免帐户密码被非法窃取。

    以修改用户 user1 密码为例，使用管理员用户连接数据库并执行如下命令：

    ```
    ALTER USER user1 IDENTIFIED BY "新密码" REPLACE "原密码";
    ```

说明

密码要符合规则，否则会执行失败。

- 管理员可以修改自己的或者其他帐户的密码。通过修改其他帐户的密码，解决用户密码遗失所造成无法登录的问题。

  以修改用户 joe 帐户密码为例，命令格式如下：

  ```
  ALTER USER joe IDENTIFIED BY 'password';
  ```

说明

- 系统管理员之间不允许互相修改对方密码。
- 系统管理员可以修改普通用户密码且不需要用户原密码。
- 系统管理员可以修改自己的密码但需要管理员原密码。

- 密码验证

  设置当前会话的用户和角色时，需要验证密码。如果输入密码与用户的存储密码不一致，则会报错。

  以设置用户 joe 为例，命令格式如下：

  ```
  SET ROLE joe PASSWORD 'password';
  ```

  显示如下命令表示设置成功：

  ```
  SET ROLE
  ```

表6-4 特殊字符

编号	字符	编号	字符	编号	字符	编号	字符
1	~	9	*	17	|	25	<
2	!	10	(	18	[	26	.
3	@	11	)	19	{	27	>
4	#	12	-	20	}	28	/

编号	字符	编号	字符	编号	字符	编号	字符
5	$	13	_	21	]	29	?
6	%	14	=	22	;	-	-
7	^	15	+	23	:	-	-
8	&	16	\	24	,	-	-

6.2 敏感数据管理

6.2.1 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
CREATE ROLE alice PASSWORD 'password';
CREATE ROLE bob PASSWORD 'password';
CREATE ROLE peter PASSWORD 'password';

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

--向数据表插入数据
INSERT INTO all_data VALUES(1, 'alice', 'alice data');
INSERT INTO all_data VALUES(2, 'bob', 'bob data');
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
GRANT SELECT ON all_data TO alice, bob, peter;

--打开行访问控制策略开关
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
CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role =
```

```
CURRENT_USER);

--查看表详细信息
 \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, compression=no, enable_rowsecurity=true

--切换至用户 alice，执行 SQL"SELECT * FROM all_data"
SET ROLE alice PASSWORD 'password';
SELECT * FROM all_data;
 id | role  |    data
----+-------+------------
  1 | alice | alice data
(1 row)

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)

--切换至用户 peter，执行 SQL"SELECT * FROM .all_data"
SET ROLE peter PASSWORD 'password';
SELECT * FROM all_data;
 id | role  |    data
----+-------+------------
  3 | peter | peter data
(1 row)

 EXPLAIN(COSTS OFF) SELECT * FROM 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)
```

6.2.2 数据脱敏

GaussDB(DWS)提供列级别的数据脱敏(Dynamic Data Masking)功能。针对某些敏感信息（如身份证号、手机号、银行卡号等），通过应用脱敏函数进行原始数据的变形改写，实现敏感隐私数据的可靠保护，从而增强产品在数据安全和隐私保护方面的能力。

- 用户可以在指定表对象创建脱敏策略，并限定策略生效范围，且支持按角色匹配脱敏规则。详细内容请参见《SQL 语法参考》中“CREATE REDACTION POLICY”章节。
- 提供脱敏策略的修改语法，包括修改脱敏策略生效表达式、重命名脱敏策略，以及新增、修改、删除脱敏列信息。详细内容请参见《SQL 语法参考》中“ALTER REDACTION POLICY”章节。
- 删除脱敏策略，可以一键式删除脱敏策略在表的所有列字段上的脱敏函数信息。详细内容请参见《SQL 语法参考》中“DROP REDACTION POLICY”章节。
- GaussDB(DWS)提供 MASK_NONE、MASK_FULL、MASK_PARTIAL 三种内置脱敏函数，也支持使用 C 语言、PL/PGSQL 语言创建的用户自定义脱敏函数。详细规格请参见《SQL 语法参考》中“数据脱敏函数”章节。
- 脱敏策略信息存储在系统表 14.2.49 PG_REDACTION_POLICY，脱敏列信息存储在系统表 14.2.48 PG_REDACTION_COLUMN。
- 用户可以通过系统视图 14.3.232 REDACTION_POLICIES 和 14.3.231 REDACTION_COLUMNS，更加方便地查看脱敏策略及脱敏列信息。

说明

- 通常，用户可以执行 SELECT 语句查看敏感信息的脱敏效果。如果语句有如下特征，则可能存在故意套取敏感数据的可能性，造成语句执行报错。
- GROUP BY 子句引用与目标列一样含有脱敏列的 Target Entry 作为分组。
- DISTINCT 作用在输出的脱敏列上。
- 带有 CTE 的语句。
- 涉及集合操作。
- 子查询的目标列不是基表的脱敏列，而是基表脱敏列的表达式或者函数调用。
- 支持脱敏数据的 COPY TO 或者 GDS 导出功能。由于脱敏数据的不可逆性，针对脱敏数据的二次运算无任何实际意义。
- 禁止 UPDATE、MERGE INTO、DELETE 语句的目标列涉及脱敏列。
- UPSERT 语句允许通过 EXCLUDED 更新插入数据。如果引用脱敏列更新基表数据，存在误改数据的可能，执行会报错。

示例

以员工表 emp，管理员用户 alice 以及普通用户 matu、july 为例，简要介绍数据脱敏过程。其中，用户 alice 是表 emp 的属主，表 emp 包含员工的姓名、手机号、邮箱、银行卡号、薪资等隐私数据。

1. 创建用户 alice、matu 和 july。

```
CREATE ROLE alice PASSWORD 'password';
CREATE ROLE matu PASSWORD 'password';
CREATE ROLE july PASSWORD 'password';
```

2. 用户 alice 创建表 emp 并插入三条员工信息。

```
CREATE TABLE emp(id int, name varchar(20), phone no varchar(11), card no number,
card string varchar(19), email text, salary numeric(100, 4), birthday date);

INSERT INTO emp VALUES(1, 'anny', '13420002340', 1234123412341234, '1234-1234-
1234-1234', 'smithWu@163.com', 10000.00, '1999-10-02');
INSERT INTO emp VALUES(2, 'bob', '18299023211', 3456345634563456, '3456-3456-
3456-3456', '66allen_mm@qq.com', 9999.99, '1989-12-12');
INSERT INTO emp VALUES(3, 'cici', '15512231233', NULL, NULL,
'jonesishere@sina.com', NULL, '1992-11-06');
```

3. 用户 alice 将表 emp 的读取权限授予用户 matu、july。

```
GRANT SELECT ON emp TO matu, july;
```

4. 仅用户 alice 可查看所有员工信息，matu 和 july 对员工所有银行卡号和薪资数据不可见，于是，对表 emp 创建脱敏策略，分布为字段 card_no、card_string 和 salary 绑定脱敏函数。

```
CREATE REDACTION POLICY mask_emp ON emp WHEN (current_user IN ('matu', 'july'))
 ADD COLUMN card_no WITH mask_full(card_no),
 ADD COLUMN card_string WITH mask_partial(card_string,
'VVVVFVVVVFVVVVFVVVV','VVVV-VVVV-VVVV-VVVV','#',1,12),
 ADD COLUMN salary WITH mask_partial(salary, '9', 1, length(salary) - 2);
```

5. 切换到用户 matu 和 july，查看员工表 emp。

```
SET ROLE matu PASSWORD 'password';
SELECT * FROM emp;
 id | name |  phone_no   | card_no |    card_string    |       email        |
salary   |     birthday
----+------+-------------+---------+-------------------+--------------------+----------
-+------------+---------------------
  1 | anny | 13420002340 |       0 | ####-####-####-1234 | smithWu@163.com     |
99999.9990 | 1999-10-02 00:00:00
  2 | bob  | 18299023211 |       0 | ####-####-####-3456 | 66allen_mm@qq.com   |
9999.9990 | 1989-12-12 00:00:00
  3 | cici | 15512231233 |         |                     | jonesishere@sina.com |
| 1992-11-06 00:00:00
(3 rows)

SET ROLE july PASSWORD 'password';
SELECT * FROM emp;
 id | name |  phone_no   | card_no |    card_string    |       email        |
salary   |     birthday
----+------+-------------+---------+-------------------+--------------------+----------
-+------------+---------------------
  1 | anny | 13420002340 |       0 | ####-####-####-1234 | smithWu@163.com     |
99999.9990 | 1999-10-02 00:00:00
  2 | bob  | 18299023211 |       0 | ####-####-####-3456 | 66allen_mm@qq.com   |
9999.9990 | 1989-12-12 00:00:00
  3 | cici | 15512231233 |         |                     | jonesishere@sina.com |
| 1992-11-06 00:00:00
(3 rows)
```

6. 用户 matu 也享有了员工所有信息的查看权限，只有 july 不可见，修改策略生效范围。

```
ALTER REDACTION POLICY mask_emp ON emp WHEN(current_user = 'july');
```

7. 切换到用户 matu 和 july，重新查看员工表 emp。

```
SET ROLE matu PASSWORD 'password';
SELECT * FROM emp;
 id | name |  phone_no   |    card_no     |    card_string     |       email
|   salary   |     birthday
----+------+-------------+------------------+--------------------+------------
----------+------------+---------------------
  1 | anny | 13420002340 | 1234123412341234 | 1234-1234-1234-1234 |
smithWu@163.com     | 10000.0000 | 1999-10-02 00:00:00
  2 | bob  | 18299023211 | 3456345634563456 | 3456-3456-3456-3456 |
66allen_mm@qq.com    |  9999.9900 | 1989-12-12 00:00:00
  3 | cici | 15512231233 |                  |                     |
jonesishere@sina.com |           | 1992-11-06 00:00:00
(3 rows)

SET ROLE july PASSWORD 'password';
SELECT * FROM emp;
 id | name |  phone_no   | card_no |    card_string     |       email        |
salary   |     birthday
----+------+-------------+---------+--------------------+--------------------
-+------------+---------------------
  1 | anny | 13420002340 |       0 | ####-####-####-1234 | smithWu@163.com     |
99999.9990 | 1999-10-02 00:00:00
  2 | bob  | 18299023211 |       0 | ####-####-####-3456 | 66allen_mm@qq.com    |
9999.9990 | 1989-12-12 00:00:00
  3 | cici | 15512231233 |         |                     | jonesishere@sina.com |
| 1992-11-06 00:00:00
(3 rows)
```

8. 员工信息 phone_no、email 和 birthday 也是隐私数据，更新脱敏策略 mask_emp，新增三个脱敏列。

```
ALTER REDACTION POLICY mask_emp ON emp ADD COLUMN phone_no WITH
mask_partial(phone_no, '*', 4);
ALTER REDACTION POLICY mask_emp ON emp ADD COLUMN email WITH mask_partial(email,
'*', 1, position('@' in email));
ALTER REDACTION POLICY mask_emp ON emp ADD COLUMN birthday WITH
mask_full(birthday);
```

9. 切换到用户 july，查看表 emp 数据。

```
SET ROLE july PASSWORD 'password';
SELECT * FROM emp;
 id | name |  phone_no   | card_no |    card_string     |       email        |
salary   |     birthday
----+------+-------------+---------+--------------------+--------------------
-+------------+---------------------
  1 | anny | 134******** |       0 | ####-####-####-1234 | ********163.com     |
99999.9990 | 1970-01-01 00:00:00
  2 | bob  | 182******** |       0 | ####-####-####-3456 | ***********qq.com    |
9999.9990 | 1970-01-01 00:00:00
  3 | cici | 155******** |         |                     | ************sina.com |
```

```
| 1970-01-01 00:00:00
(3 rows)
```

10. 查询视图 redaction_policies 和 redaction_columns，查看当前脱敏策略 mask_emp 的详细信息。

```
SELECT * FROM redaction_policies;
 object_schema | object_owner | object_name | policy_name |          expression
| enable | policy_description
---------------+--------------+-------------+-------------+--------------------
---------------+--------+--------------------
 public        | alice        | emp         | mask_emp    | ("current_user"() =
'july'::name) | t      |
(1 row)

SELECT object_name, column_name, function_info FROM redaction_columns;
 object_name | column_name |                                        function_info
-------------+-------------+-----------------------------------------------------
-------------------------------------------------
 emp         | card_no     | mask_full(card_no)
 emp         | card_string | mask_partial(card_string,
'VVVVFVVVVFVVVVFVVVV'::text, 'VVVV-VVVV-VVVV-VVVV'::text, '#'::text, 1, 12)
 emp         | email       | mask_partial(email, '*'::text, 1, "position"(email,
'@'::text))
 emp         | salary      | mask_partial(salary, '9'::text, 1,
(length((salary)::text) - 2))
 emp         | birthday    | mask_full(birthday)
 emp         | phone_no    | mask_partial(phone_no, '*'::text, 4)
(6 rows)
```

11. 新增一列 salary_info，若需要将文本类型的薪资信息统一脱敏成“*.*”，可以创建自定义脱敏函数实现。此处采用 PL/PGSQL 语言定义脱敏函数 mask_regexp_salary，创建脱敏列时，只需自定义脱敏的函数名和参数列表，详细内容可参考 12 用户自定义函数。

```
ALTER TABLE emp ADD COLUMN salary_info TEXT;
UPDATE emp SET salary_info = salary::text;

CREATE FUNCTION mask_regexp_salary(salary_info text) RETURNS text AS
$$
 SELECT regexp_replace($1, '[0-9]+','*','g');
$$
LANGUAGE SQL
STRICT SHIPPABLE;

ALTER REDACTION POLICY mask_emp ON emp ADD COLUMN salary_info WITH
mask_regexp_salary(salary_info);

SET ROLE july PASSWORD 'password';
SELECT id, name, salary_info FROM emp;
 id | name | salary_info
----+------+-------------
  1 | anny | *.*
  2 | bob  | *.*
  3 | cici |
(3 rows)
```

12. 无需为表 emp 设置敏感策略，删除脱敏策略 mask_emp。

```
DROP REDACTION POLICY mask_emp ON emp;
```

6.2.3 使用函数加解密

GaussDB(DWS)支持使用以下函数对字符串进行加解密。

- gs_encrypt(encryptstr, keystr, cryptotype, cryptomode, hashmethod)

  描述：采用 cryptotype 和 cryptomode 组成的加密算法以及 hashmethod 指定的 HMAC 算法，以 keystr 为密钥对 encryptstr 字符串进行加密，返回加密后的字符串。支持的 cryptotype：aes128, aes192, aes256, sm4。支持的 cryptomode：cbc。支持的 hashmethod：sha256, sha384, sha512, sm3。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。keystr 的长度范围与加密算法相关，为 1~KeyLen 字节。当 cryptotype 为 aes128 和 sm4 时，KeyLen 为 16，aes192 时 KeyLen 为 24，aes256 时 KeyLen 为 32。

  返回值类型：text

  返回值长度：至少为 4*[(maclen + 56)/3] 字节，不超过 4*[(Len + maclen + 56)/3] 字节，其中 Len 为加密前数据长度（单位为字节），maclen 为 HMAC 值的长度，当 hashmethod 为 sha256 和 sm3 时 maclen 为 32，sha384 时 maclen 为 48，sha512 时 maclen 为 64。即当 hashmethod 为 sha256 和 sm3 时，返回值长度至少为 120 字节，不超过 4*[(Len + 88)/3] 字节；当 hashmethod 为 sha384 时，返回值长度至少为 140 字节，不超过 4*[(Len + 104)/3] 字节；当 hashmethod 为 sha512 时，返回值长度至少为 160 字节，不超过 4*[(Len + 120)/3] 字节；

  示例：

  ```
  SELECT gs_encrypt('GaussDB(DWS)', '1234', 'aes128', 'cbc', 'sha256');
                                                  gs_encrypt
  ------------------------------------------------------------------------------
  ------------------------------------------

  AAAAAAAAAACcFjDcCSbop7D87sOa2nxTFrkE9RJQGK34ypgrOPsFJIqggI8tl+eMDcQYT3po98wPCC7
  VBfhv7mdBy7IVnzdrp0rdMrD6/zTl8w0v9/s2OA==
  (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 工具不会将该函数记录入执行历史。即无法在 gsql 里通过上下翻页功能找到该函数的执行历史。
  - 在同一张数据表中，加密函数 ge_encrypt 与 gs_encrypt_aes128 不要混合使用。

- gs_decrypt(decryptstr, keystr，cryptotype, cryptomode, hashmethod)

  描述：采用 cryptotype 和 cryptomode 组成的加密算法以及 hashmethod 指定的 HMAC 算法，以 keystr 为密钥对 decryptstr 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 一致才能正常解密。keystr 不得为空。

  返回值类型：text

  示例：

  ```
  SELECT
  gs_decrypt('AAAAAAAAAACcFjDcCSbop7D87sOa2nxTFrkE9RJQGK34ypgrOPsFJIqggI8tl+eMDcQ
  YT3po98wPCC7VBfhv7mdBy7IVnzdrp0rdMrD6/zTl8w0v9/s2OA==', '1234', 'aes128', 'cbc',
  ```

```
'sha256');
 gs_decrypt
--------------
 GaussDB(DWS)
(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 工具不会将该函数记录入执行历史。即无法在 gsql 里通过上下翻页功能找到该函数的执行历史。
- 此函数需要结合 gs_encrypt 加密函数共同使用，且加密算法和 HMAC 算法要保证一致。

- gs_encrypt_aes128(encryptstr,keystr)

  描述：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 1~16 字节。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：

```
SELECT gs_encrypt_aes128('MPPDB','1234');

                          gs_encrypt_aes128
------------------------------------------------------------------------------
------
gwditQLQG8NhFw4OuoKhhQJoXojhFlYkjeG0aYdSCtLCnIUgkNwvYI04KbuhmcGZp8jWizBdR1vU9Cs
pjuzI0lbz12A=
(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 工具不会将该函数记录入执行历史。即无法在 gsql 里通过上下翻页功能找到该函数的执行历史。
- 在同一张数据表中，加密函数 gs_encrypt_aes128 与 ge_encrypt 不要混合使用。

- gs_decrypt_aes128(decryptstr,keystr)

  描述：以 keystr 为密钥对 decryptstr 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 一致才能正常解密。keystr 不得为空。

  返回值类型：text

  示例：

```
SELECT
gs decrypt aes128('gwditQLQG8NhFw4OuoKhhQJoXojhFlYkjeG0aYdSCtLCnIUgkNwvYI04Kbuh
mcGZp8jWizBdR1vU9CspjuzI0lbz12A=','1234');
 gs decrypt aes128
-------------------
 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 工具不会将该函数记录入执行历史。即无法在 gsql 里通过上下翻页功能找到该函数的执行历史。
- 此函数需要结合 gs_encrypt_aes128 加密函数共同使用。

- gs_hash(hashstr, hashmethod)

  描述：以 hashmethod 算法对 hashstr 字符串进行信息摘要，返回信息摘要字符串。支持的 hashmethod：sha256, sha384, sha512, sm3。

  返回值类型：text

  返回值长度：sha256 和 sm3 返回 64 字节，sha384 返回 96 字节，sha512 返回 128 字节。

  示例：

```
SELECT gs_hash('GaussDB(DWS)', 'sha256');
                                    gs_hash
--------------------------------------------------------------------------------
------------------

e59069daa6541ae20af7c747662702c731b26b8abd7a788f4d15611aa0db608efdbb5587ba90789
a983f85dd51766609
(1 row)
```

7 开发设计建议

7.1 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(DWS)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。

7.2 数据库对象命名

数据库对象命名需要满足约束：长度不超过 63 个字符，以字母或下划线开头，中间字符可以是字母、数字、下划线、$、#。

- 【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(DWS)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。
  - 数据库对象名称由字母、数字和下划线组成，并且不能由数字开头。建议使用多个单词组成，以下划线分割。

- 数据库对象名称最好能够望文知意，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。
  - 外表以“f_+后缀”命名。

7.3 数据库对象设计

7.3.1 Database 和 Schema 设计

GaussDB(DWS)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

- 【建议】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 gaussdb 数据库。
- 【建议】一个集群内，用户自定义的 Database 数量建议不超过 3 个。
- 【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(DWS)支持 Oracle、Teradata 和 MySQL 三种兼容模式，分别兼容 Oracle、Teradata 和 MySQL 语法，不同兼容模式下的语法行为可能有一些差异。详细内容可参考 5 Oracle、Teradata 和 MySQL 语法兼容性差异。
- 【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

- 【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。

7.3.2 表设计

GaussDB(DWS)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，减小不必要的数据 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 7-1。

表7-1 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。
列存	• 统计分析类查询 (关联、分组操作较多的场景)。 • 即席查询（查询条件不确定，行存表扫描难以使用索引）。

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7-2 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。
Roundrobin	表的每一行被轮番地发送给各个 DN，因此数据会被均匀地分布在各个 DN 中。	数据量较大的事实表，且使用 Hash 分布时找不到合适的分布列。

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)
);
```

选择分布键

Hash 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 Hash 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 字段做为分布键。
- 【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。

7.3.3 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，下表列出了 GaussDB(DWS)中常见的字符串类型：

表7-3 常用字符串类型

名称	描述	最大存储空间
CHAR(n)	定长字符串，n 描述了存储的字节长度，如果输入的字符串字节格式小于 n，那么后面会自动用空字符补齐至 n 个字节。	10MB
CHARACTER(n)	定长字符串，n 描述了存储的字节长度，如果输入的字符串字节格式小于 n，那么后面会自动用空字符补齐至 n	10MB

名称	描述	最大存储空间
	个字节。	
NCHAR(n)	定长字符串，n 描述了存储的字节长度，如果输入的字符串字节格式小于 n，那么后面会自动用空字符补齐至 n 个字节。	10MB
BPCHAR(n)	定长字符串，n 描述了存储的字节长度，如果输入的字符串字节格式小于 n，那么后面会自动用空字符补齐至 n 个字节。	10MB
VARCHAR(n)	变长字符串，n 描述了可以存储的最大字节长度。	10MB
CHARACTER VARYING(n)	变长字符串，n 描述了可以存储的最大字节长度；此数据类型和 VARCHAR(n)是同一数据类型的不同表达形式。	10MB
VARCHAR2(n)	变长字符串，n 描述了可以存储的最大字节长度，此数据类型是为兼容 Oracle 类型新增的，行为和 VARCHAR(n)一致。	10MB
NVARCHAR2(n)	变长字符串，n 描述了可以存储的最大字符长度。	10MB
TEXT	不限长度(不超过 1GB-8203 字节)变长字符串。	1GB-8203 字节

7.3.4 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。

局部聚簇

Partial Cluster Key（局部聚簇，简称 PCK）是列存表的一种局部聚簇技术，在 GaussDB(DWS)中，使用 PCK 可以通过 min/max 稀疏索引实现事实表快速过滤扫描。PCK 的选取遵循以下原则：

- 【关注】一张表上只能建立一个 PCK，一个 PCK 可以包含多列，但是一般不建议超过 2 列。
- 【建议】在查询中的简单表达式过滤条件上创建 PCK。这种过滤条件一般形如 col op const，其中 col 为列名，op 为操作符 =、>、>=、<=、<，const 为常量值。
- 【建议】在满足上面条件的前提下，选择 distinct 值比较少的列上建 PCK。

唯一约束

- 【关注】行存表与列存表都支持唯一约束。
- 【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】行存表与列存表都支持主键约束。
- 【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，而列存表不支持。
- 【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”。

7.3.5 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。

7.4 JDBC 配置

目前，GaussDB(DWS)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(DWS)时，JDBC 向 GaussDB(DWS)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", "2" },
{ "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，应该保证下面两个时区设置一致：
  - JDBC 客户端所在主机的时区。
  - GaussDB(DWS)集群所在主机的时区。

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(DWS)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 SQL 的编写习惯。

【建议】在应用程序完成作业任务之后，应当及时断开和 GaussDB(DWS)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(DWS) JDBC 驱动的 copyManger 接口进行微批导入。

CopyManager 的使用方法请参见 8.3.12.14 CopyManager。

7.5 SQL 编写

DDL

- 【建议】在 GaussDB(DWS)中，建议 DDL（建表、comments 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(DWS)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（列存）表时，临时表和非日志表也推荐创建为行存（列存）表，可以避免行列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】在 ORACLE 兼容模式下，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需要新建兼容性为 TD 的数据库。

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。
- 【关注】通过常量来过滤数据时，会根据常量的数据类型和匹配列的数据类型来查找用于这两种数据类型计算的函数，如果找不到对应的函数，则会相应的进行隐式数据类型转化，然后再根据转化后的数据类型查找用于转化后的数据类型计算的函数。

```
SELECT * FROM test WHERE timestamp_col = 20000101;
```

上述例子中，假设 timestamp_col 是 timestamp 类型，则会先查找支持 timestamp 类型和 int 类型（常量数字认为是 int 类型）“等于”运算的函数，如果找不到，则把 timestamp_col 和常量数字隐式类型转化成 text 类型来计算。

- 【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，括号内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(DWS)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 AND
inline=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 (va11), (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 T1,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。

7.6 自定义外部函数(pgSQL/Java)使用

- 【关注】Java UDF 可以实现一些 java 逻辑计算，禁止在 Java UDF 中封装业务。
- 【关注】禁止在 Java 函数中使用任何方式连接数据库，包括但不限于 JDBC。
- 【关注】只能选择下表中的数据类型，不支持自定义类型、复杂数据类型（Java Array 类及派生类）等：
- 【关注】不支持 UDAF（用户定义聚合函数），UDTF（用户自定义表生成函数）。

表7-4 PL/Java 默认数据类型映射关系

GaussDB(DWS)	**Java**
BOOLEAN	boolean
"char"	byte
bytea	byte[]
SMALLINT	short
INTEGER	int
BIGINT	long
FLOAT4	float
FLOAT8	double
CHAR	java.lang.String
VARCHAR	java.lang.String
TEXT	java.lang.String
name	java.lang.String
DATE	java.sql.Timestamp
TIME	java.sql.Time (stored value treated as local time)
TIMETZ	java.sql.Time
TIMESTAMP	java.sql.Timestamp
TIMESTAMPTZ	java.sql.Timestamp

7.7 PL/pgSQL 使用

总体开发原则

1. 应完全按照设计文档进行开发。

2. 程序模块应做到高内聚低耦合。
3. 应有正确、全面的故障对策。
4. 程序编写应做到结构合理，条理清晰。
5. 程序名称命名应按照统一的命名规则进行命名。
6. 应充分考虑程序的运行效率，包括程序的执行效率和数据库的查询、存储效率，在保证应用的同时应使用效率高的处理方法。
7. 程序注释应详细、正确、规范。
8. 除非应用特别需要控制 commit 和 rollback 的提交时机，否则应在存储过程结束时执行显式的 commit 或者 rollback 操作。
9. 程序处理应支持 7*24 小时；对于中断，应用程序应提供安全、简单的断点再续处理。
10. 应提供标准、简单的应用输出，为应用维护人员提供明确的进度显示、错误描述和运行结果；为业务人员提供明确、直观的报表、凭证输出。

程序编写原则

1. 在 PL/PGSQL 中的 SQL 语句宜使用绑定变量。
2. 在 PL/PGSQL 中的 SQl 语句宜使用 RETURNING 字句。
3. 存储过程使用原则：
   a. 对于单个存储过程中 Varchar 或者 Varchar2 类型输出参数个数不应超过 50 个。
   b. 不应使用 long 类型作为输入或输出参数。
   c. 对于大小超过 10MB 的字符串类型输出，应使用 CLOB 类型。
4. 变量声明原则：
   a. 变量声明时，如果含义和应用表某字段含义或某变量相同时，应使用%TYPE 声明。
   b. 记录声明时，如果含义和某应用表行数据相同时，应使用%ROWTYPE 声明。
   c. 变量声明每行应只包含一条语句。
   d. 不应声明 LONG 类型的变量。
5. 游标使用类型：
   a. 显式游标使用后应关闭。
   b. 游标变量使用后应关闭，若游标变量需要传递数据给调用的应用程序，应在应用程序中进行游标关闭处理；若游标变量仅在存储过程中使用，应显式关闭游标。
   c. 在使用 DBMS_SQL.CLOSE_CURSOR 关闭游标前，应使用 DBMS_SQL.IS_OPEN 判断游标是否已打开。
6. 集合使用原则：
   a. 引用集合中的元素时宜使用 FORALL 语句，不宜使用 FOR 循环语句。
7. 动态语句使用原则：
   a. 联机系统的交易程序不宜使用动态 SQL。

b. PL/PGSQL 中要实现 DDL 语句和系统控制命令，可使用动态 SQL。

c. 宜尽量使用变量绑定。

8. 拼装 SQL 的使用原则：

a. 拼装 SQL 宜使用绑定变量。

b. 拼装 SQL 语句的条件如果有外部输入源，应对输入条件进行字符检查，防止攻击。

c. 在 PL/PGSQL 脚本中，单行代码的长度，不宜超过 2499 字符。

9. Trigger 使用原则：

a. Trigger 可用于实现差量数据日志等于业务处理无关的可用性设计场景。

b. 不应使用 Trigger 实现业务处理相关功能。

异常处理原则

任何在 PL/pgSQL 函数中发生的错误会中止该函数的执行，而且实际上会中止其周围的事务。你可以使用一个带有 EXCEPTION 子句的 BEGIN 块俘获错误并且从中恢复。

1. 在使用 PL/PGSQL 块中，如果使用了不能返回确定结果的 SQL 语句，宜在 EXCEPTION 中对程序可能出现的异常进行处理，避免出现未处理的出错被传递到外层块，导致程序逻辑错误。
2. 对于系统已经定义了的异常，可以直接使用。DWS 暂不支持自定义异常。
3. 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，非必要场景不应使用 EXCEPTION。

书写规范

1. 变量命名规则：

a. 过程、函数的输入参数格式宜为：IN_参数名，参数名宜使用大写。

b. 过程、函数的输出参数格式宜为：OUT_参数名，参数名宜使用大写。

c. 过程、函数得输入输出参数格式宜为：IO_参数名，参数名宜使用大写。

d. 过程、函数得程序中用到的变量宜由 v_变量名组成，变量名宜使用小写。

e. 将查询语句做成字符串拼接时，where 语句的拼接变量名宜统一为 v_where，select 语句的拼接变量名宜为 v_select。

f. 记录（RECORD）的类型（TYPE）命名宜由 T+变量名组成，名称宜使用大写。

g. 游标命名宜由 CUR+变量名组成，名称宜使用大写。

h. 引用游标（REF CURSOR）的命名宜由 REF+变量名组成，名称宜使用大写。

2. 变量类型定义：

a. 变量类型声明时，如果其含义和应用表某字段含义相同时，应使用%TYPE 声明。

b. 记录类型声明时，如果其含义和某应用表行数据相同时，应使用%ROWTYPE 声明。

3. 注释规范：

a. 注释应该是有意义的，而不应是重述代码。

b. 注释应简洁、易懂，以中文为主。为了表达准确，名词或操作上也可以使用英文。

c. 应在每个存储过程、函数得开始加入注释，内容应包括：本程序的简要功能描述、编写者、编写日期、程序版本号信息和程序变更信息，而且各存储过程开头注释应保持统一格式。

d. 应在输入输出参数的旁边添加注释，注明次变量的意义。

e. 每个块或大分支的开始宜添加注释，描述块的简要功能，若使用算法，宜添加注释简单描述算法的目的和结果。

4. 变量声明格式：

   每行应只包含一条语句，如同时需要赋初始值，应在同一行书写。

5. 大小写规范：

   除了变量名，应一律使用大写。

6. 缩进规范：

   创建存储过程语句中，同一层的 CREATE、AS/IS、BEGIN、END 这几个关键字应位于同一列，其他部分依次缩进。

7. 语句详述：

   a. 变量定义语句。每行应只包含一条语句。

   b. 同一层的 IF、ELSEIF、ELSE 和 END 关键字应开始于同一列，执行语句缩进。

   c. CASE 和 END 关键字应位于同一列，WHEN 和 ELSE 关键字应缩进。

   d. 同一层的 LOOP 和 END LOOP 关键字应位于同一列，层内语句或嵌套应依次缩进。

8 教程：使用 JDBC 或 ODBC 开发

8.1 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。

8.2 驱动下载

请参见《数据仓库服务用户指南》的“**下载 JDBC 或 ODBC 驱动**”章节。

8.3 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(DWS)库提供了对 JDBC 4.0 特性的支持，需要使用 JDK1.6 及以上版本编译程序代码，不支持 JDBC 桥接 ODBC 方式。

8.3.1 JDBC 包与驱动类

JDBC 包

从管理控制台下载包名为 dws_8.1.x_jdbc_driver.zip，具体下载方法请参见 8.2 驱动下载。

解压后的 JDBC 的驱动 jar 包：

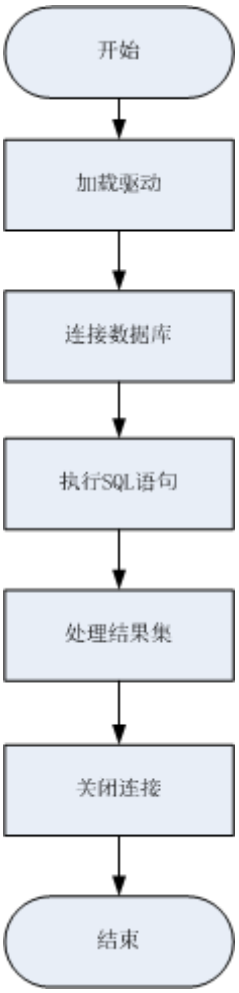
gsjdbc4.jar：与 PostgreSQL 保持兼容的驱动包，其中类名、类结构与 PostgreSQL 驱动完全一致，曾经运行于 PostgreSQL 的应用程序可以直接移植到当前系统使用。

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）。

8.3.2 开发流程

图8-1 采用 JDBC 开发应用程序的流程



8.3.3 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

8.3.4 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。

说明

如果您使用的是开源的 JDBC 驱动程序，应确保数据库参数 password_encryption_type 取值设置为 1，如果参数值不为 1，可能会出现连接失败，典型的报错信息比如：“none of the server's SASL authentication mechanisms are supported”，参见以下操作：

1. 将参数修改为 1，修改方法参见《用户指南》的“修改数据库参数”章节。
2. 新建一个数据库用户用于连接，或者重置准备使用的数据库用户的密码。
   - 如果您使用的是管理员账号，参见《用户指南》的“重置密码”章节。
   - 如果是普通用户，可以先通过其他客户端工具（例如 Data Studio）连接数据库后，使用 ALTER USER 语句来修改密码。
3. 再尝试连接数据库。

需要执行以上操作的原因：

- 调整参数的原因：当前 MD5 算法已被证实可以人工碰撞，已严禁将之用于密码校验算法。GaussDB(DWS) 采用默认安全设计，默认禁止 MD5 算法的密码校验，而 PostgreSQL 的开源 libpq 通信协议恰恰使用的是 MD5 算法。所以需要调整一下密码算法参数 password_encryption_type，打开 MD5 算法。
- 修改密码的原因：GaussDB(DWS) 中是不会存储您的密码原文的，而是存储的密码 HASH 摘要（默认是 SHA256 摘要），在密码校验时该摘要会与客户端发来的密码摘要进行比对（中间会有加盐操作）。故当您只是单纯调整了密码算法策略时，数据库是无法还原您的密码进而再生成 MD5 的摘要值的，必须要求您手动修改一次密码或者创建一个新用户，这时新的密码将会采用您设置的 HASH 算法进行摘要存储，用于下次连接认证。

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8-1 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 • jdbc:postgresql://host:port/database

参数	描述
	• jdbc:postgresql://host:port[,host:port][...]/database 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。 连接 GaussDB(DWS)的机器与 GaussDB(DWS)不在同一网段时，host 指定的 IP 地址应为 Manager 界面上所设的 mppdb.coo.cooListenIp2（应用访问 IP）的取值。 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。 • port 为数据库服务器端口。缺省情况下，会尝试连接到 localhost 的 8000 端口的 database。 • 支持多 ip 端口配置形式，jdbc 自动实现了负载均衡，多 ip 端口配置形式是采取随机访问+failover 的方式，这个过程系统会自动忽略不可达 IP。 以","隔开，例如 jdbc:postgresql://10.10.0.13:8000,10.10.0.14:8000/database
info	数据库连接属性。常用的属性如下： • 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。 • ssl：Boolean 类型。表示是否使用 SSL 连接。 • loggerLevel：string 类型。为 LogStream 或 LogWriter 设置记录进 DriverManager 当前值的日志信息量。目前支持"OFF"、"DEBUG"和"TRACE"。 值为"DEBUG"时，表示只打印 DEBUG 级别以上的日志，将记录非常少的信息。值等于 TRACE 时，表示打印 DEBUG 和 TRACE 级别的日志，将产生详细的日志信息。默认值为 OFF，表示不打印日志。 • prepareThreshold：integer 类型。用于确定在转换为服务器端的预备语句之前，要求执行方法 PreparedStatement 的次数。缺省值是 5。 • batchMode : boolean 类型，用于确定是否使用 batch 模式连接。 • fetchsize ： integer 类型，用于设置数据库链接所创建 statement 的默认 fetchsize。 • ApplicationName：string 类型。应用名称，在不做设置时，缺省值为 PostgreSQL JDBC Driver。 • allowReadOnly:boolean 类型，用于设置 connection 是否允许设置 readonly 模式，默认为 false，若该参数不被设置为 true，则执行 connection.setReadOnly 不生效。 • blobMode:string 类型，用于设置 setBinaryStream 方法为不同的数据类型赋值，设置为 on 时表示为 blob 数据类型赋值，设置为 off 时表示为

参数	描述
	bytea 数据类型赋值，默认为 on。 • connectionExtraInfo：Boolean 类型。表示驱动是否上报当前驱动的部署路径、进程属主用户到数据库。 说明 取值范围：true 或 false，默认值为 true。设置 connectionExtraInfo 为 true，JDBC 驱动会将当前驱动的部署路径、进程属主用户上报到数据库中，记录在 connection_info 参数（参见 connection_info）里；同时可以在 14.3.124 PG_STAT_ACTIVITY 和 14.3.195 PGXC_STAT_ACTIVITY 中查询到。
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 GetConnection(String username, String passwd)
    {
        //驱动类。
        String driver = "org.postgresql.Driver";
        //数据库连接描述符。
        String sourceURL =
"jdbc:postgresql://10.10.0.13:8000/postgres?currentSchema=test";
        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;
};
```

8.3.5 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。

```
Statement stmt = con.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。

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);
```

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(DWS)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。

```
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 in out 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(DWS)数据库的 JDBC 驱动程序不支持此方法。
- GaussDB(DWS)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。

----结束

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。

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。

```
PreparedStatement pstmt = con.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

8.3.6 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 8-2 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8-2 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 8-3 所示。

表8-3 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。

方法	描述
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。
relative(int)	向前或者向后移动参数指定的行。

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 8-4 所示。

表8-4 获取结果集光标的位置

方法	描述
isFirst()	是否在一行。
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 8-5 所示，其他方法请参考 JDK 官方文档。

表8-5 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。

8.3.7 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 方法即可。如：**conn.close()**

8.3.8 示例：常用操作

示例 1

在完成以下示例前，需要先创建存储过程。

```
create or replace procedure testproc
(
   psv_in1 in integer,
   psv_in2 in integer,
   psv_inout in out integer
)
as
begin
   psv_inout := psv_in1 + psv_in2 + psv_inout;
end;
/
```

此示例将演示如何基于 GaussDB(DWS)提供的 JDBC 接口开发应用程序。

```
//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。

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

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://localhost:/gaussdb";
    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 {

     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。
    Connection conn = GetConnection("tester", "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);

    //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。

```
// 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.");
}
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.");
}
rs.close();

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

8.3.9 示例：重新执行应用 SQL

当主 DN 故障且 40s 未恢复时，GaussDB(DWS)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。

```
//以下用例以 gsjdbc4.jar 为例。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://10.131.72.136:8000/gaussdb";
     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();

      // add ctrl+c handler
      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。
   Connection conn = GetConnection("testuser", "test@123");

   //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();
   }

 }

}
```

8.3.10 示例：通过本地文件导入导出数据

在使用 JAVA 语言基于 GaussDB(DWS)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(DWS) jdbc 驱动。

```
//以下用例以 gsjdbc4.jar 为例。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://10.180.155.74:8000/gaussdb"); //数
据库 URL
     String username = new String("jack");          //用户名
     String password = new String("********");       //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) {
 // TODO Auto-generated catch block
 e.printStackTrace();
  } catch (IOException e) {
 // TODO Auto-generated catch block
 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) {
 // TODO Auto-generated catch block
       e.printStackTrace();
} catch (IOException e) {
 // TODO Auto-generated catch block
 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) {
  // TODO Auto-generated catch block
  e.printStackTrace();
   } catch (IOException e) {
  // TODO Auto-generated catch block
  e.printStackTrace();
 }
    }

  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();
            }
        }
    }
 }

  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();
             }
         }
     }
  }
}
```

8.3.11 示例：从 MySQL 向 GaussDB(DWS)进行数据迁移

下面示例演示如何通过 CopyManager 从 mysql 向 GaussDB(DWS)进行数据迁移的过程。

```
//以下用例以 gsjdbc4.jar 为例。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://10.180.155.74:8000/gaussdb"); //数据库 URL
      String user = new String("jack");          //mppdb 用户名
      String pass = new String("*********");           //mppdb 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();
         Connection conn =
DriverManager.getConnection("jdbc:mysql://10.119.179.227:3306/jack?useSSL=false&all
owPublicKeyRetrieval=true", "jack", "********");
         Statement stmt = conn.createStatement();
         rs = stmt.executeQuery("select * from migration_table");
      } catch (SQLException e) {
         e.printStackTrace();
      } catch (Exception e) {
         e.printStackTrace();
      }
      return rs;
   }
}
```

8.3.12 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。

8.3.12.1 java.sql.Connection

java.sql.Connection 是数据库连接接口。

表8-6 对 java.sql.Connection 接口的支持情况

方法名	返回值类型	支持 JDBC 4
close()	void	Yes
commit()	void	Yes

方法名	返回值类型	支持 JDBC 4
createStatement()	Statement	Yes
getAutoCommit()	boolean	Yes
getClientInfo()	Properties	Yes
getClientInfo(String name)	String	Yes
getTransactionIsolation()	int	Yes
isClosed()	boolean	Yes
isReadOnly()	boolean	Yes
prepareStatement(String sql)	PreparedStatement	Yes
rollback()	void	Yes
setAutoCommit(boolean autoCommit)	void	Yes
setClientInfo(Properties properties)	void	Yes
setClientInfo(String name,String value)	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。

8.3.12.2 java.sql.CallableStatement

java.sql.CallableStatement 是存储过程执行接口。

表8-7 对 java.sql.CallableStatement 的支持情况

方法名	返回值类型	支持 JDBC 4
registerOutParameter(int parameterIndex, int type)	void	Yes
wasNull()	boolean	Yes
getString(int parameterIndex)	String	Yes
getBoolean(int parameterIndex)	boolean	Yes
getByte(int parameterIndex)	byte	Yes
getShort(int parameterIndex)	short	Yes

方法名	返回值类型	支持 JDBC 4
getInt(int parameterIndex)	int	Yes
getLong(int parameterIndex)	long	Yes
getFloat(int parameterIndex)	float	Yes
getDouble(int parameterIndex)	double	Yes
getBigDecimal(int parameterIndex)	BigDecimal	Yes
getBytes(int parameterIndex)	byte[]	Yes
getDate(int parameterIndex)	Date	Yes
getTime(int parameterIndex)	Time	Yes
getTimestamp(int parameterIndex)	Timestamp	Yes
getObject(int parameterIndex)	Object	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 。

8.3.12.3 java.sql.DatabaseMetaData

java.sql.DatabaseMetaData 是数据库对象定义接口。

表8-8 对 java.sql.DatabaseMetaData 的支持情况

方法名	返回值类型	支持 JDBC 4
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

方法名	返回值类型	支持 JDBC 4
getUserName()	String	Yes
isReadOnly()	boolean	Yes
nullsAreSortedHigh()	boolean	Yes
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,	boolean	Yes

方法名	返回值类型	支持 JDBC 4
int toType)		
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
supportsSavepoints()	boolean	Yes

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

8.3.12.4 java.sql.Driver

java.sql.Driver 是数据库驱动接口。

表8-9 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

8.3.12.5 java.sql.PreparedStatement

java.sql.PreparedStatement 是预处理语句接口。

表8-10 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
getMetaData()	ResultSetMetaData	Yes
setBoolean(int parameterIndex, boolean x)	void	Yes

方法名	返回值类型	支持 JDBC 4
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
setNString(int parameterIndex, String value)	void	Yes
setShort(int parameterIndex, short x)	void	Yes
setString(int parameterIndex, String x)	void	Yes
addBatch()	void	Yes
executeBatch()	int[]	Yes
clearBatch()	void	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。

8.3.12.6 java.sql.ResultSet

java.sql.ResultSet 是执行结果集接口。

表8-11 对 java.sql.ResultSet 的支持情况

方法名	返回值类型	支持 JDBC 4
findColumn(String columnLabel)	int	Yes
getBigDecimal(int columnIndex)	BigDecimal	Yes
getBigDecimal(String columnLabel)	BigDecimal	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
getDate(int columnIndex)	Date	Yes
getDate(String columnLabel)	Date	Yes
getDouble(int columnIndex)	double	Yes
getDouble(String columnLabel)	double	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
getShort(int columnIndex)	short	Yes
getShort(String columnLabel)	short	Yes
getString(int columnIndex)	String	Yes
getString(String	String	Yes

方法名	返回值类型	支持 JDBC 4
columnLabel)		
getTime(int columnIndex)	Time	Yes
getTime(String columnLabel)	Time	Yes
getTimestamp(int columnIndex)	Timestamp	Yes
getTimestamp(String columnLabel)	Timestamp	Yes
isAfterLast()	boolean	Yes
isBeforeFirst()	boolean	Yes
isFirst()	boolean	Yes
next()	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。

8.3.12.7 java.sql.ResultSetMetaData

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

表8-12 对 java.sql.ResultSetMetaData 的支持情况

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

8.3.12.8 java.sql.Statement

java.sql.Statement 是 SQL 语句接口。

表8-13 对 java.sql.Statement 的支持情况

方法名	返回值类型	支持 JDBC 4
close()	void	Yes

方法名	返回值类型	支持 JDBC 4
execute(String sql)	boolean	Yes
executeQuery(String sql)	ResultSet	Yes
executeUpdate(String sql)	int	Yes
getConnection()	Connection	Yes
getResultSet()	ResultSet	Yes
getQueryTimeout()	int	Yes
getUpdateCount()	int	Yes
isClosed()	boolean	Yes
setQueryTimeout(int seconds)	void	Yes
setFetchSize(int rows)	void	Yes
cancel()	void	Yes

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。

8.3.12.9 javax.sql.ConnectionPoolDataSource

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

表8-14 对 javax.sql.ConnectionPoolDataSource 的支持情况

方法名	返回值类型	支持 JDBC 4
getLoginTimeout()	int	Yes
getLogWriter()	PrintWriter	Yes
getPooledConnection()	PooledConnection	Yes
getPooledConnection(String user,String password)	PooledConnection	Yes
setLoginTimeout(int seconds)	void	Yes
setLogWriter(PrintWriter out)	void	Yes

8.3.12.10 javax.sql.DataSource

javax.sql.DataSource 是数据源接口。

表8-15 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

8.3.12.11 javax.sql.PooledConnection

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

表8-16 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
addStatementEventListener (StatementEventListener listener)	void	Yes
removeStatementEventListener (StatementEventListener listener)	void	Yes

8.3.12.12 javax.naming.Context

javax.naming.Context 是连接配置的上下文接口。

表8-17 对 javax.naming.Context 的支持情况

方法名	返回值类型	支持 JDBC 4

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

8.3.12.13 javax.naming.spi.InitialContextFactory

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

表8-18 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

8.3.12.14 CopyManager

CopyManager 是 GaussDB(DWS) JDBC 驱动中提供的一个 API 接口类，用于批量向 GaussDB(DWS)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8-19 CopyManager 常用方法

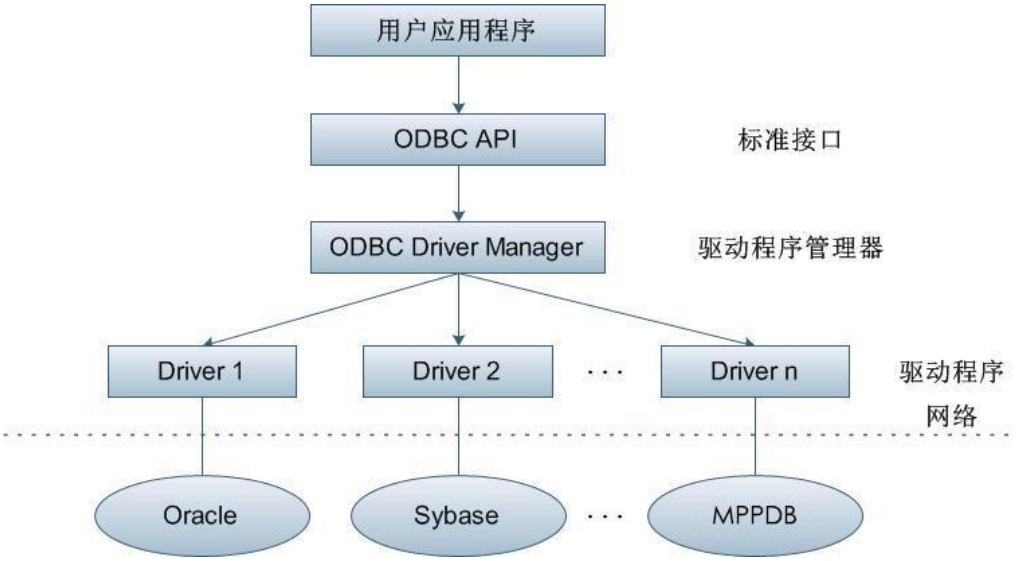
返回值	方法	描述	throws
CopyIn	copyIn(String sql)	-	SQLException
long	copyIn(String sql, InputStream from)	使用 COPY FROM STDIN 从 InputStream 中快速向数据库中的表加载数据。	SQLException,IOException
long	copyIn(String sql, InputStream from, int bufferSize)	使用 COPY FROM STDIN 从 InputStream 中快速向数据库中的表加载数据。	SQLException,IOException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

8.4 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 8-2。

图8-2 ODBC 系统机构



GaussDB(DWS)目前在以下环境中提供对 ODBC3.5 的支持。

表8-20 ODBC 支持平台

操作系统	平台
SUSE Linux Enterprise Server 11 SP1/SP2/SP3/SP4 SUSE Linux Enterprise Server 12 及 SP1/SP2/SP3/SP5	x86_64 位
Red Hat Enterprise Linux 6.4/6.5/6.6/6.7/6.8/6.9/7.0/7.1/7.2/7.3/7.4/7.5	x86_64 位
Red Hat Enterprise Linux 7.5	ARM64 位
CentOS 6.4/6.5/6.6/6.7/6.8/6.9/7.0/7.1/7.2/7.3/7.4	x86_64 位
CentOS 7.6	ARM64 位
EulerOS 2.0 SP2/SP3	x86_64 位
EulerOS 2.0 SP8	ARM64 位
中标麒麟 7.5/7.6	ARM64 位

操作系统	平台
Oracle Linux R7U4	x86_64 位
Windows 7	32 位
Windows 7	64 位
Windows Server 2008	32 位
Windows Server 2008	64 位

以上操作系统平台是指 ODBC 程序所在的操作系统平台，可以与数据库部署的操作系统平台不同。

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。

8.4.1 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 dws_8.1.x_odbc_driver_for_xxx_xxx.zip。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-2.3.0 的安装包中获得。

Windows 下的 ODBC 包

从发布包中获取，包名为 dws_8.1.x_odbc_driver_for_windows.zip。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。

8.4.2 Linux 下配置数据源

将 GaussDB(DWS)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：http://sourceforge.net/projects/unixodbc/files/unixODBC/2.3.0/unixODBC-2.3.0.tar.gz/download

步骤 2 目前不支持 unixODBC-2.2.1 版本。以 unixODBC-2.3.0 版本为例，在客户端执行如下命令安装 unixODBC。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。

```
tar zxvf unixODBC-2.3.0.tar.gz
cd unixODBC-2.3.0
#修改 configure 文件(如果不存在，那么请修改 configure.ac)，找到 LIB_VERSION
#将它的值修改为"1:0:0"，这样将编译出*.so.1 的动态库，与 psqlodbcw.so 的依赖关系相同。
vim configure

./configure --enable-gui=no #如果要在 TaiShan 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
```

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。

步骤 3 替换客户端 GaussDB(DWS)驱动程序。

将 dws_8.1.x_odbc_driver_for_xxx_xxx.zip 解压，在“/dws_8.1.x_odbc_driver_for_xxx_xxx/odbc/lib”目录下得到“psqlodbcw.la”和“psqlodbcw.so”两个文件。

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 8-21 所示。

表8-21 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=/xxx/odbc/lib/psqlodbcw.so
setup	驱动安装路径，与 Driver64 中动态库的路径一致。	setup=/xxx/odbc/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。

   ```
   [MPPODBC]
   Driver=GaussMPP
   Servername=10.10.0.13（数据库 Server IP）
   Database=gaussdb （数据库名）
   ```

```
Username=dbadmin （数据库用户名）
Password= （数据库用户密码）
Port=8000 （数据库监听端口）
Sslmode=allow
```

odbc.ini 文件配置参数说明如表 8-22 所示。

表8-22 odbc.ini 文件配置参数

参数	描述	示例
[DSN]	数据源的名称。	[MPPODBC]
Driver	驱动名，对应 odbcinst.ini 中的 DriverName。	Driver=DRIVER_N
Servername	服务器的 IP 地址。	Servername=10.145.130.26
Database	要连接的数据库的名称。	Database=gaussdb
Username	数据库用户名称。	Username=dbadmin
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。
Port	服务器的端口号。	Port=8000
Sslmode	开启 SSL 模式	Sslmode=allow
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
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。	UseBatchProtocol=1

参数	描述	示例
	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。	
ConnectionExtraInfo	GUC 参数 connection_info（参见 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 1。当设置为 0 时，ODBC 驱动会将当前驱动的名称、驱动版本上报到数据库中；当设置为 1 时，ODBC 驱动会将当前驱动的名称、部署路径、进程属主用户上报到数据库中，记录在 connection_info 参数（参见 connection_info）里；同时可以在 14.3.124 PG_STAT_ACTIVITY 和 14.3.195 PGXC_STAT_ACTIVITY 中查询到。
ForExtensionConnector	ETL 工具性能优化参数，可进行内存优化，降低对端的 CN 内存占用，避免因 CN 内存使用过多导致系统不稳定。 可选值 0 或者 1，默认为 0，表示不开启优化项。 请勿在数据库系统之外的其他业务中配置此参数，以免影响业务的正确性。	ForExtensionConnector=1
KeepDisallowPremature	当 UseDeclareFetch=1 时，应用程序调用 SQLPrepare 后调用 SQLNumResultCols、SQLDescribeCol 或 SQLColAttribute 获取结果集列信息时，SQL 语句中的游标是否具有 with hold 属性。 可选值 0 或者 1，0 表示具有 with hold 属性，1 表示不具有 with hold 属性，默认为 0。	KeepDisallowPremature=1 说明 • UseServerSidePrepare=1 时，KeepDisallowPremature 参数不生效，使用时需要指定 UseServerSidePrepare 为 0，例如：UseDeclareFetch=1

参数	描述	示例
		KeepDisallowPremature=1 UseServerSidePrepare=0

其中关于 sslmode 的选项的允许值，具体信息见下表：

表8-23 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器签发的证书。
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。GaussDB(DWS)不支持此模式。

步骤 5 SSL 模式

如果需要使用 SSL 证书连接，那么请将 GaussDB(DWS)安装包中的 SSLCERT 的证书包解压，在 shell 环境下，执行“source sslcert_env.sh”，即在**当前会话**完成证书的默认位置的部署。

或者手动声明如下环境变量，同时保证 client.key*系列文件为 600 权限：

```
export PGSSLCERT="/YOUR/PATH/OF/client.crt" #请修改该路径到 client.crt 的绝对路径
export PGSSLKEY="/YOUR/PATH/OF/client.key"  #请修改该路径到 client.key 的绝对路径
```

同时将数据源中的 Sslmode 选项调整至“verify-ca”。

步骤 6 将客户端所在主机的 IP 网段加入 GaussDB(DWS)的安全组规则，确保客户端主机与 GaussDB(DWS)网络互通。

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

执行 isql -v GaussODBC(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                                  |
|                                       |
+---------------------------------------+
SQL>
```

- 若显示 ERROR 信息，则表明配置错误。请检查上述配置是否正确。

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：'ls'一下错误信息中的路径，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：ldd 一下错误信息中的路径，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；如果是缺少其他库，请将 ODBC 驱动包中的 lib 目录添加到 LD_LIBRARY_PATH 中。

- [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。

  - 服务器监听不正确

    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，确保数据库监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，请确认一下相关的设置。

- [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 8-23。

- 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 CA 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。
- 早期版本（V100R002C80SPC300 之前的版本）的数据库只存储了 SHA256 格式的哈希，并未存储 MD5 的哈希，所以无法使用 MD5 做用户口令校验。
- 新版本（V100R002C80SPC300 及之后版本）的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。

要解决该问题，可以更新用户口令（参见《SQL 语法参考》中“ALTER USER”章节）；或者新建一个用户（参见《SQL 语法参考》中“CREATE USER”章节），赋于同等权限，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。

8.4.3 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(DWS)驱动程序

将 GaussDB-8.1.3-Windows-Odbc.tar.gz 解压后，根据需要，点击 psqlodbc.msi（32 位）或者 psqlodbc_x64.msi（64 位）进行驱动安装。

步骤 2 打开驱动管理器

在配置数据源时，请使用对应的驱动管理器（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** 位操作系统上进行 **32** 位程序开发，安装 **32** 位驱动程序后，使用 **32** 位的驱动管理器：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。

- **64** 位操作系统上进行 **64** 位程序开发，安装 **64** 位驱动程序后，使用 **64** 位的驱动管理器：C:\Windows**System32**\odbcad32.exe

  请勿直接使用控制面板->管理工具->数据源(ODBC)。

- 32 位操作系统请使用：C:\Windows\System32\odbcad32.exe

  或者点击计算机->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（如果是 64 位驱动，将会有 64 位标识），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

如果需要使用 SSL 证书连接，那么请将 GaussDB(DWS)安装包中的 SSLCERT 的证书包解压，双击"sslcert_env.bat"文件，即可完成证书的默认位置的部署。

须知

该 sslcert_env.bat 为了保证证书环境的纯净，在%APPDATA%\postgresql 目录存在时，会提示是否需要移除相关目录。如果有需要，请备份该目录中的文件。

或者手动将 client.crt、client.key、client.key.cipher、client.key.rand 文件放至%APPDATA%\postgresql(该目录需手动建立)目录下，并且将文件名中的 client 改为 postgres，例如 client.key 修改为 postgres.key；将 cacert.pem 文件放至%APPDATA%\postgresql 目录，并更名为 root.crt。

同时将步骤 2 中的设置窗口的“SSL Mode”选项调整至“verify-ca”。

表8-24 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器签发的证书。
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。 说明 不支持此模式。

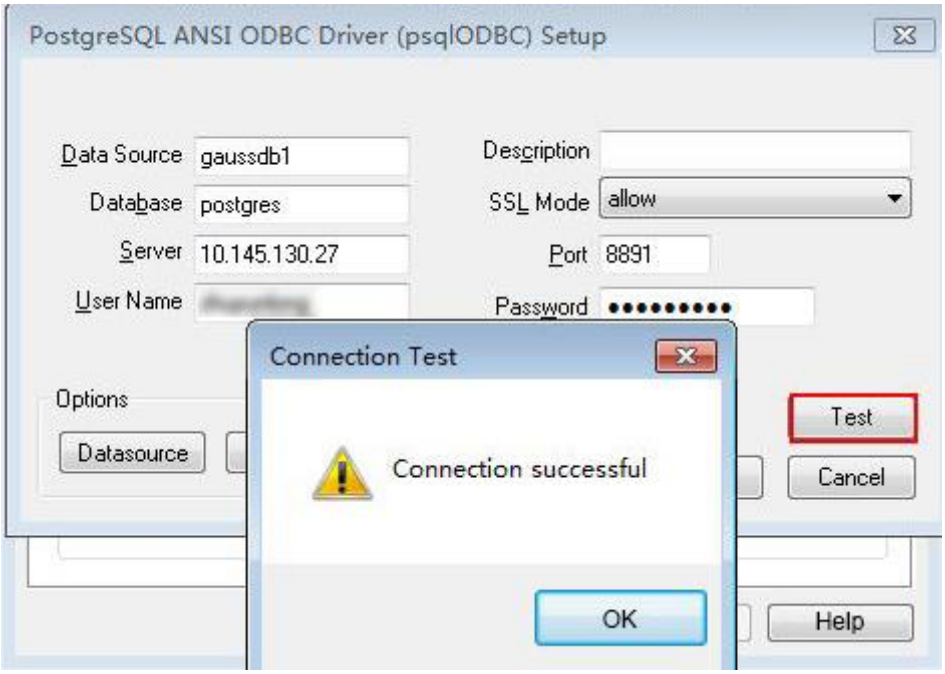
步骤 5 将客户端所在主机的 IP 网段加入 GaussDB(DWS)的安全组规则，确保客户端主机与 GaussDB(DWS)网络互通。

----结束

测试数据源配置

点击 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。

常见问题处理

- Server common name "xxxx" does not match host name "xxxxx"

  此问题的原因是使用了 SSL 加密的“verify-full”选项，这时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。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 CA 证书。

- 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。

  - 服务器监听不正确

    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，确保数据库监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，请确认一下相关的设置。

- 在指定的 DSN 中，驱动程序和应用程序之间的体系结构不匹配

  此问题可能的原因：在 64 位程序中使用了 32 位驱动，或者相反。

  C:\Windows\SysWOW64\odbcad32.exe：这是 32 位 ODBC 驱动管理器。

  C:\Windows\System32\odbcad32.exe：这是 64 位 ODBC 驱动管理器。

- 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 8-24。

- 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。
  - 早期版本（V100R002C80SPC300 之前的版本）的数据库只存储了 SHA256 格式的哈希，并未存储 MD5 的哈希，所以无法使用 MD5 做用户口令校验。
  - 新版本（V100R002C80SPC300 及之后版本）的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。

  要解决该问题，可以更新用户口令（参见《SQL 语法参考》中“ALTER USER”章节）；或者新建一个用户（参见《SQL 语法参考》中“CREATE USER”章节），赋于同等权限，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。

8.4.4 ODBC 开发示例

常用功能示例代码

```
// 此示例演示如何通过 ODBC 方式获取 GaussDB(DWS)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;
SQLLEN       V_StrLen_or_IndPtr;
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);
    }
    // 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，请根据实际情况修改。
    // 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 id from testtable",SQL_NTS);
    // 12. 获取结果集某一列的属性

SQLColAttribute(V_OD_hstmt,1,SQL_DESC_TYPE_NAME,typename,sizeof(typename),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 StrLen or IndPtr);
    // 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
*************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
#include <string.h>

#include "util.c"

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);
    CHECK_ERROR(retcode, "SQLAllocHandle(SQL_HANDLE_STMT)",
              hstmt, SQL_HANDLE_STMT);

    // Prepare Statement
    retcode = SQLPrepare(hstmt, (SQLCHAR*) sql, SQL_NTS);
    sprintf((char*)loginfo, "SQLPrepare log: %s", (char*)sql);
    CHECK_ERROR(retcode, loginfo, hstmt, SQL_HANDLE_STMT);

    retcode = SQLExecute(hstmt);
    sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);
    CHECK_ERROR(retcode, loginfo, hstmt, SQL_HANDLE_STMT);

    retcode = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
    sprintf((char*)loginfo, "SQLFreeHandle stmt log: %s", (char*)sql);
```

```
    CHECK_ERROR(retcode, loginfo, hstmt, SQL_HANDLE_STMT);
}

int main ()
{
    SQLHENV  henv  = SQL_NULL_HENV;
    SQLHDBC  hdbc  = SQL_NULL_HDBC;
    int     batchCount = 1000;
    SQLLEN   rowsCount = 0;
    int     ignoreCount = 0;

    SQLRETURN   retcode;
    SQLCHAR    dsn[1024] = {'\0'};
    SQLCHAR    loginfo[2048];

    // 交互获取数据源名称
    getStr("Please input your DSN", (char*)dsn, sizeof(dsn), 'N');
    // 交互获取批量绑定的数据量
    getInt("batchCount", &batchCount, 'N', 1);
    do
    {
        // 交互获取批量绑定的数据中，不要入库的数据量
        getInt("ignoreCount", &ignoreCount, 'N', 1);
        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);
    CHECK_ERROR(retcode, "SQLAllocHandle(SQL_HANDLE_ENV)",
              henv, SQL_HANDLE_ENV);

    // Set ODBC Verion
    retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
                                (SQLPOINTER*)SQL_OV_ODBC3, 0);
    CHECK_ERROR(retcode, "SQLSetEnvAttr(SQL_ATTR_ODBC_VERSION)",
              henv, SQL_HANDLE_ENV);

    // Allocate Connection
    retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
    CHECK_ERROR(retcode, "SQLAllocHandle(SQL_HANDLE_DBC)",
              henv, SQL_HANDLE_DBC);

    // Set Login Timeout
    retcode = SQLSetConnectAttr(hdbc, SQL_LOGIN_TIMEOUT, (SQLPOINTER)5, 0);
    CHECK_ERROR(retcode, "SQLSetConnectAttr(SQL_LOGIN_TIMEOUT)",
              hdbc, SQL_HANDLE_DBC);

    // Set Auto Commit
    retcode = SQLSetConnectAttr(hdbc, SQL ATTR AUTOCOMMIT,
                                (SQLPOINTER)(1), 0);
    CHECK ERROR(retcode, "SQLSetConnectAttr(SQL ATTR AUTOCOMMIT)",
              hdbc, SQL_HANDLE_DBC);
```

```
    // Connect to DSN
    sprintf(loginfo, "SQLConnect(DSN:%s)", dsn);
    retcode = SQLConnect(hdbc, (SQLCHAR*) dsn, SQL_NTS,
                         (SQLCHAR*) NULL, 0, NULL, 0);
    CHECK_ERROR(retcode, loginfo, hdbc, SQL_HANDLE_DBC);

    // 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;
        int         i;
        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 (int 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);
        CHECK_ERROR(retcode, "SQLAllocHandle(SQL_HANDLE_STMT)",
                  hstmtinesrt, SQL_HANDLE_STMT);

        // 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);
        CHECK_ERROR(retcode, loginfo, hstmtinesrt, SQL_HANDLE_STMT);

        retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAMSET_SIZE,
(SQLPOINTER)batchCount, sizeof(batchCount));
        CHECK_ERROR(retcode, "SQLSetStmtAttr", hstmtinesrt, SQL_HANDLE_STMT);

        retcode = SQLBindParameter(hstmtinesrt, 1, SQL_PARAM_INPUT, SQL_C_SLONG,
SQL_INTEGER, sizeof(ids[0]), 0,&(ids[0]), 0, bufLenIds);
        CHECK_ERROR(retcode, "SQLBindParameter for id", hstmtinesrt,
SQL_HANDLE_STMT);

        retcode = SQLBindParameter(hstmtinesrt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_CHAR, 50, 50, cols, 50, bufLenCols);
        CHECK_ERROR(retcode, "SQLBindParameter for cols", hstmtinesrt,
SQL_HANDLE_STMT);

        retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAMS_PROCESSED_PTR,
(SQLPOINTER)&process, sizeof(process));
        CHECK_ERROR(retcode, "SQLSetStmtAttr for SQL_ATTR_PARAMS_PROCESSED_PTR",
hstmtinesrt, SQL_HANDLE_STMT);

        retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAM_STATUS_PTR,
(SQLPOINTER)statusptr, sizeof(statusptr[0]) * batchCount);
        CHECK_ERROR(retcode, "SQLSetStmtAttr for SQL_ATTR_PARAM_STATUS_PTR",
hstmtinesrt, SQL_HANDLE_STMT);

        retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAM_OPERATION_PTR,
(SQLPOINTER)operptr, sizeof(operptr[0]) * batchCount);
        CHECK_ERROR(retcode, "SQLSetStmtAttr for SQL_ATTR_PARAM_OPERATION_PTR",
hstmtinesrt, SQL_HANDLE_STMT);

        retcode = SQLExecute(hstmtinesrt);
        sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);
        CHECK_ERROR(retcode, loginfo, hstmtinesrt, SQL_HANDLE_STMT);

        retcode = SQLRowCount(hstmtinesrt, &rowsCount);
        CHECK_ERROR(retcode, "SQLRowCount execution", hstmtinesrt, SQL_HANDLE_STMT);

        if (rowsCount != (batchCount - ignoreCount))
        {
            sprintf(loginfo, "(batchCount - ignoreCount)(%d) != rowsCount(%d)",
(batchCount - ignoreCount), rowsCount);
            CHECK ERROR(SQL ERROR, loginfo, NULL, SQL HANDLE STMT);
        }
        else
```

```
      {
         sprintf(loginfo, "(batchCount - ignoreCount)(%d) == rowsCount(%d)",
(batchCount - ignoreCount), rowsCount);
         CHECK_ERROR(SQL_SUCCESS, loginfo, NULL, SQL_HANDLE_STMT);
      }

      if (rowsCount != process)
      {
         sprintf(loginfo, "process(%d) != rowsCount(%d)", process, rowsCount);
         CHECK_ERROR(SQL_ERROR, loginfo, NULL, SQL_HANDLE_STMT);
      }
      else
      {
         sprintf(loginfo, "process(%d) == rowsCount(%d)", process, rowsCount);
         CHECK_ERROR(SQL_SUCCESS, loginfo, NULL, SQL_HANDLE_STMT);
      }

      for (int 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]);
               CHECK_ERROR(SQL_ERROR, loginfo, NULL, SQL_HANDLE_STMT);
            }
         }
         else if (statusptr[i] != SQL_PARAM_SUCCESS)
         {
            sprintf(loginfo, "statusptr[%d](%d) != SQL_PARAM_SUCCESS", i,
statusptr[i]);
            CHECK_ERROR(SQL_ERROR, loginfo, NULL, SQL_HANDLE_STMT);
         }
      }

      retcode = SQLFreeHandle(SQL_HANDLE_STMT, hstmtinesrt);
      sprintf((char*)loginfo, "SQLFreeHandle hstmtinesrt");
      CHECK_ERROR(retcode, loginfo, hstmtinesrt, SQL_HANDLE_STMT);
   }


exit:
   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;
}
```

8.4.5 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。

8.4.5.1 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 8.4.5.3 SQLAllocHandle。

8.4.5.2 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 8.4.5.3 SQLAllocHandle。

8.4.5.3 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8-25 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

关键字	参数说明
	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 8.4.5.19 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：8.4.4 ODBC 开发示例

8.4.5.4 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 8.4.5.3 SQLAllocHandle。

8.4.5.5 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,
```

```
    SQLLEN       BufferLength,
    SQLLEN       *StrLen_or_IndPtr);
```

参数

表8-26 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 8.4.5.19 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：8.4.4 ODBC 开发示例

8.4.5.6 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,
                        SQLULEN       ColumnSize,
                        SQLSMALLINT    DecimalDigits,
                        SQLPOINTER     ParameterValuePtr,
                        SQLLEN        BufferLength,
                        SQLLEN        *StrLen_or_IndPtr);
```

参数

表8-27 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 SQLBindCol 返回 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 时，通过调用 8.4.5.19 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：8.4.4 ODBC 开发示例

8.4.5.7 SQLColAttribute

功能描述

返回结果集中一列的描述符信息。

原型

```
SQLRETURN SQLColAttribute(SQLHSTMT       StatementHandle,
                          SQLUSMALLINT    ColumnNumber,
                          SQLUSMALLINT    FieldIdentifier,
                          SQLPOINTER     CharacterAtrributePtr,
                          SQLSMALLINT    BufferLength,
                          SQLSMALLINT    *StringLengthPtr,
                          SQLPOINTER     NumericAttributePtr);
```

参数

表8-28 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。
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 8.4.5.19 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：8.4.4 ODBC 开发示例

8.4.5.8 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8-29 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。
NameLength1	ServerName 的长度。
UserName	数据源中数据库用户名。
NameLength2	UserName 的长度。

关键字	参数说明
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 8.4.5.19 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：8.4.4 ODBC 开发示例

8.4.5.9 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8-30 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 8.4.5.19 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：8.4.4 ODBC 开发示例

8.4.5.10 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8-31 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 8.4.5.19 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：8.4.4 ODBC 开发示例

8.4.5.11 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8-32 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 8.4.5.19 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：8.4.4 ODBC 开发示例

8.4.5.12 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8-33 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 8.4.5.19 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：8.4.4 ODBC 开发示例

8.4.5.13 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 8.4.5.15 SQLFreeHandle。

8.4.5.14 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 8.4.5.15 SQLFreeHandle。

8.4.5.15 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8-34 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：8.4.4 ODBC 开发示例

8.4.5.16 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 8.4.5.15 SQLFreeHandle。

8.4.5.17 SQLPrepare

功能描述

准备一个将要进行的 SQL 语句。

原型

```
SQLRETURN SQLPrepare(SQLHSTMT     StatementHandle,
                     SQLCHAR      *StatementText,
                     SQLINTEGER    TextLength);
```

参数

表8-35 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 8.4.5.19 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：8.4.4 ODBC 开发示例

8.4.5.18 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8-36 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 SQLFetch 函数返回 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 时，通过调用 8.4.5.19 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：8.4.4 ODBC 开发示例

8.4.5.19 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8-37 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

关键字	参数说明
	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。
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8-38 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

SQLSATATE	错误	描述
		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：8.4.4 ODBC 开发示例

8.4.5.20 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8-39 SQLSetConnectAttr 参数

关键字	参数说明
StatementtHandle	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 8.4.5.19 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：8.4.4 ODBC 开发示例

8.4.5.21 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8-40 SQLSetEnvAttr 参数

关键字	参数说明
EnviromentHandle	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 8.4.5.19 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：8.4.4 ODBC 开发示例

8.4.5.22 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8-41 SQLSetStmtAttr 参数

关键字	参数说明
StatementtHandle	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 8.4.5.19 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：8.4.4 ODBC 开发示例

9 PostGIS Extension

9.1 PostGIS 概述

说明

- PostGIS Extension 依赖的第三方软件需要用户进行单独安装，用户如需使用 PostGIS 功能，请提交工单或联系技术支持人员提交申请。
- 如果用户在使用中出现“ERROR：EXTENSION is not yet supported.”这种报错，则表示没有安装 PostGIS 软件包，请联系技术支持进行获取。

GaussDB(DWS)提供 PostGIS Extension（版本为 PostGIS-2.4.2）。PostGIS Extension 是 PostgreSQL 的空间数据库扩展，提供如下空间信息服务功能：空间对象、空间索引、空间操作函数和空间操作符。PostGIS Extension 完全遵循 OpenGIS 规范。

GaussDB(DWS)中 PostGIS Extension 依赖第三方开源软件如下。

- Geos 3.6.2
- Proj 4.9.2
- Json 0.12.1
- Libxml2 2.7.1
- Gdal 1.11.0

9.2 PostGIS 使用

说明

- PostGIS Extension 依赖的第三方软件需要用户进行单独安装，用户如需使用 PostGIS 功能，请提交工单或联系技术支持人员提交申请。
- 如果用户在使用中出现“ERROR：EXTENSION is not yet supported.”这种报错，则表示没有安装 PostGIS 软件包，请联系技术支持进行获取。

创建 Extension

创建 PostGIS Extension 可直接使用 CREATE EXTENSION 命令进行创建：

```
CREATE EXTENSION postgis;
```

使用 Extension

PostGIS Extension 函数调用格式为：

```
SELECT GisFunction (Param1, Param2,......);
```

其中 GisFunction 为函数名，Param1、Param2 等为函数参数名。下列 SQL 语句展示 PostGIS 的简单使用，对于各函数的具体使用，请参考《PostGIS-2.4.2 用户手册》。

示例 1：几何表的创建。

```
CREATE TABLE cities ( id integer, city_name varchar(50) );
SELECT AddGeometryColumn('cities', 'position', 4326, 'POINT', 2);
```

示例 2：几何数据的插入。

```
INSERT INTO cities (id, position, city_name) VALUES (1,ST_GeomFromText('POINT(-9.5
23)',4326),'CityA');
INSERT INTO cities (id, position, city_name) VALUES (2,ST_GeomFromText('POINT(-10.6
40.3)',4326),'CityB');
INSERT INTO cities (id, position, city_name) VALUES (3,ST_GeomFromText('POINT(20.8
30.3)',4326), 'CityC');
```

示例 3：计算三个城市间任意两个城市距离。

```
SELECT p1.city_name,p2.city_name,ST_Distance(p1.position,p2.position) FROM cities
AS p1, cities AS p2 WHERE p1.id > p2.id;
```

删除 Extension

在 GaussDB(DWS)中删除 PostGIS Extension 的方法如下所示：

```
DROP EXTENSION postgis [CASCADE];
```

说明

如果 Extension 被其它对象依赖（如创建的几何表），需要加入 CASCADE（级联）关键字，删除所有依赖对象。

9.3 PostGIS 支持和限制

支持数据类型

GaussDB(DWS)的 PostGIS Extension 支持如下数据类型:

- box2d
- box3d
- geometry_dump
- geometry
- geography
- raster

说明

创建 Postgis 和使用 Postgis 非同一用户时，请设置如下 GUC 参数：

```
SET behavior_compat_options = 'bind_procedure_searchpath';
```

支持的操作符和函数列表

表9-1 **PostGIS 支持的操作符和函数列表**

函数分类	包含函数
Management Functions	AddGeometryColumn、DropGeometryColumn、DropGeometryTable、PostGIS_Full_Version、PostGIS_GEOS_Version、PostGIS_Liblwgeom_Version、PostGIS_Lib_Build_Date、PostGIS_Lib_Version、PostGIS_PROJ_Version、PostGIS_Scripts_Build_Date、PostGIS_Scripts_Installed、PostGIS_Version、PostGIS_LibXML_Version、PostGIS_Scripts_Released、Populate_Geometry_Columns 、UpdateGeometrySRID
Geometry Constructors	ST_BdPolyFromText 、ST_BdMPolyFromText 、ST_Box2dFromGeoHash、ST_GeogFromText、ST_GeographyFromText、ST_GeogFromWKB、ST_GeomCollFromText、ST_GeomFromEWKB、ST_GeomFromEWKT、ST_GeometryFromText、ST_GeomFromGeoHash、ST_GeomFromGML、ST_GeomFromGeoJSON、ST_GeomFromKML、ST_GMLToSQL、ST_GeomFromText 、ST_GeomFromWKB、ST_LineFromMultiPoint、ST_LineFromText、ST_LineFromWKB、ST_LinestringFromWKB、ST_MakeBox2D、ST_3DMakeBox、ST_MakeEnvelope、ST_MakePolygon、ST_MakePoint、ST_MakePointM、ST_MLineFromText、ST_MPointFromText、ST_MPolyFromText、ST_Point、ST_PointFromGeoHash、ST_PointFromText、ST_PointFromWKB、ST_Polygon、ST_PolygonFromText、ST_WKBToSQL、ST_WKTToSQL
Geometry Accessors	GeometryType、ST_Boundary、ST_CoordDim、ST_Dimension、ST_EndPoint、ST_Envelope、ST_ExteriorRing、ST_GeometryN、ST_GeometryType、ST_InteriorRingN、ST_IsClosed、ST_IsCollection、ST_IsEmpty、ST_IsRing、ST_IsSimple、ST_IsValid、ST_IsValidReason、ST_IsValidDetail、ST_M、ST_NDims、ST_NPoints、ST_NRings、ST_NumGeometries、ST_NumInteriorRings、ST_NumInteriorRing、ST_NumPatches、ST_NumPoints、ST_PatchN、ST_PointN、ST_SRID、ST_StartPoint、ST_Summary、ST_X、ST_XMax、ST_XMin、ST_Y、ST_YMax、ST_YMin、ST_Z、ST_ZMax、ST_Zmflag、ST_ZMin

函数分类	包含函数
Geometry Editors	ST_AddPoint、ST_Affine、ST_Force2D、ST_Force3D、ST_Force3DZ、ST_Force3DM、ST_Force4D、ST_ForceCollection、ST_ForceSFS、ST_ForceRHR、ST_LineMerge、ST_CollectionExtract、ST_CollectionHomogenize、ST_Multi、ST_RemovePoint、ST_Reverse、ST_Rotate、ST_RotateX、ST_RotateY、ST_RotateZ、ST_Scale、ST_Segmentize、ST_SetPoint、ST_SetSRID、ST_SnapToGrid、ST_Snap、ST_Transform、ST_Translate、ST_TransScale
Geometry Outputs	ST_AsBinary、ST_AsEWKB、ST_AsEWKT、ST_AsGeoJSON、ST_AsGML、ST_AsHEXEWKB、ST_AsKML、ST_AsLatLonText 、ST_AsSVG、ST_AsText、ST_AsX3D、ST_GeoHash
Operators	&&、&&&、&<、&<|、&>、<<、<<|、=、>>、@ 、|&> 、|>>、~、~=、<->、<#>
Spatial Relationships and Measurements	ST_3DClosestPoint、ST_3DDistance、ST_3DDWithin、ST_3DDFullyWithin、ST_3DIntersects、ST_3DLongestLine、ST_3DMaxDistance、ST_3DShortestLine、ST_Area、ST_Azimuth、ST_Centroid、ST_ClosestPoint、ST_Contains、ST_ContainsProperly、ST_Covers、ST_CoveredBy、ST_Crosses、ST_LineCrossingDirection、ST_Disjoint、ST_Distance、ST_HausdorffDistance、ST_MaxDistance、ST_DistanceSphere、ST_DistanceSpheroid、ST_DFullyWithin、ST_DWithin、ST_Equals、ST_HasArc、ST_Intersects、ST_Length、ST_Length2D、ST_3DLength、ST_Length_Spheroid、ST_Length2D_Spheroid、ST_3DLength_Spheroid、ST_LongestLine、ST_OrderingEquals、ST_Overlaps、ST_Perimeter、ST_Perimeter2D、ST_3DPerimeter、ST_PointOnSurface、ST_Project、ST_Relate、ST_RelateMatch、ST_ShortestLine、ST_Touches、ST_Within
Geometry Processing	ST_Buffer、ST_BuildArea、ST_Collect、ST_ConcaveHull、ST_ConvexHull、ST_CurveToLine、ST_DelaunayTriangles、ST_Difference、ST_Dump、ST_DumpPoints、ST_DumpRings、ST_FlipCoordinates、ST_Intersection、ST_LineToCurve、ST_MakeValid、ST_MemUnion、ST_MinimumBoundingCircle、ST_Polygonize、ST_Node、ST_OffsetCurve、ST_RemoveRepeatedPoints、ST_SharedPaths、ST_Shift_Longitude、ST_Simplify、ST_SimplifyPreserveTopology、ST_Split、ST_SymDifference、ST_Union、ST_UnaryUnion
Linear Referencing	ST_LineInterpolatePoint、ST_LineLocatePoint、ST_LineSubstring、ST_LocateAlong、ST_LocateBetween、ST_LocateBetweenElevations、ST_InterpolatePoint、

函数分类	包含函数
	ST_AddMeasure
Miscellaneous Functions	ST_Accum、Box2D、Box3D、ST_Expand、ST_Extent、ST_3Dextent、Find_SRID、ST_MemSize
Exceptional Functions	PostGIS_AddBBox、PostGIS_DropBBox、PostGIS_HasBBox
Raster Management Functions	AddRasterConstraints、DropRasterConstraints、AddOverviewConstraints、DropOverviewConstraints、PostGIS_GDAL_Version、PostGIS_Raster_Lib_Build_Date、PostGIS_Raster_Lib_Version、ST_GDALDrivers、UpdateRasterSRID
Raster Constructors	ST_AddBand、ST_AsRaster、ST_Band、ST_MakeEmptyRaster、ST_Tile、ST_FromGDALRaster
Raster Accessors	ST_GeoReference、ST_Height、ST_IsEmpty、ST_MetaData、ST_NumBands、ST_PixelHeight、ST_PixelWidth、ST_ScaleX、ST_ScaleY、ST_RasterToWorldCoord、ST_RasterToWorldCoordX、ST_RasterToWorldCoordY、ST_Rotation、ST_SkewX、ST_SkewY、ST_SRID、ST_Summary、ST_UpperLeftX、ST_UpperLeftY、ST_Width、ST_WorldToRasterCoord、ST_WorldToRasterCoordX、ST_WorldToRasterCoordY
Raster Band Accessors	ST_BandMetaData、ST_BandNoDataValue、ST_BandIsNoData、ST_BandPath、ST_BandPixelType、ST_HasNoBand
Raster Pixel Accessors and Setters	ST_PixelAsPolygon、ST_PixelAsPolygons、ST_PixelAsPoint、ST_PixelAsPoints、ST_PixelAsCentroid、ST_PixelAsCentroids、ST_Value、ST_NearestValue、ST_Neighborhood、ST_SetValue、ST_SetValues、ST_DumpValues、ST_PixelOfValue
Raster Editors	ST_SetGeoReference、ST_SetRotation、ST_SetScale、ST_SetSkew、ST_SetSRID、ST_SetUpperLeft、ST_Resample、ST_Rescale、ST_Reskew、ST_SnapToGrid、ST_Resize、ST_Transform
Raster Band Editors	ST_SetBandNoDataValue、ST_SetBandIsNoData
Raster Band Statistics and Analytics	ST_Count、ST_CountAgg、ST_Histogram、ST_Quantile、ST_SummaryStats、ST_SummaryStatsAgg、ST_ValueCount
Raster Outputs	ST_AsBinary、ST_AsGDALRaster、ST_AsJPEG、ST_AsPNG、ST_AsTIFF
Raster Processing	ST_Clip、ST_ColorMap、ST_Intersection、ST_MapAlgebra、ST_Reclass、ST_Union、ST_Distinct4ma、ST_InvDistWeight4ma、ST_Max4ma、ST_Mean4ma、

函数分类	包含函数
	ST_Min4ma、ST_MinDist4ma、ST_Range4ma、ST_StdDev4ma、ST_Sum4ma、ST_Aspect、ST_HillShade、ST_Roughness、ST_Slope、ST_TPI、ST_TRI、Box3D、ST_ConvexHull、ST_DumpAsPolygons、ST_Envelope、ST_MinConvexHull、ST_Polygon、ST_Contains、ST_ContainsProperly、ST_Covers、ST_CoveredBy、ST_Disjoint、ST_Intersects、ST_Overlaps、ST_Touches、ST_SameAlignment、ST_NotSameAlignmentReason、ST_Within、ST_DWithin、ST_DFullyWithin
Raster Operators	&&、&<、&>、=、@、~=、~

空间索引

GaussDB(DWS)数据库的 PostGIS Extension 支持 GIST (Generalized Search Tree) 空间索引（分区表除外）。相比于 B-tree 索引，GIST 索引适应于任意类型的非常规数据结构，可有效提高几何和地理数据信息的检索效率。

使用如下命令创建 GIST 索引：

```
CREATE INDEX indexname ON tablename USING GIST ( geometryfield );
```

扩展限制

- 只支持行存表。
- 只支持 Oracle 兼容格式数据库。
- 不支持拓扑对象管理模块 Topology。
- 不支持 BRIN 索引。
- spatial_ref_sys 表在扩容期间只支持查询操作。

9.4 OPEN SOURCE SOFTWARE NOTICE (For PostGIS)

This document contains open source software notice for the product. And this document is confidential information of copyright holder. Recipient shall protect it in due care and shall not disseminate it without permission.

Warranty Disclaimer

This document is provided “as is” without any warranty whatsoever, including the accuracy or comprehensiveness. Copyright holder of this document may change the contents of this document at any time without prior notice, and copyright holder disclaims any liability in relation to recipient’s use of this document.

Open source software is provided by the author “as is” and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall the author be liable for any direct,

indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of data or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of open source software, even if advised of the possibility of such damage.

Copyright Notice And License Texts

Software: postgis-2.4.2

Copyright notice:

"Copyright (C) 1996-2015 Free Software Foundation, Inc.

Copyright (C) 1989, 1991 Free Software Foundation, Inc.,

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Copyright 2008 Kevin Neufeld

Copyright (c) 2009 Walter Bruce Sinclair

Copyright 2006-2013 Stephen Woodbridge.

Copyright (c) 2008 Walter Bruce Sinclair

Copyright (c) 2012 TJ Holowaychuk <tj@vision-media.ca>

Copyright (c) 2008, by Attractive Chaos <attractivechaos@aol.co.uk>

Copyright (c) 2001-2012 Walter Bruce Sinclair

Copyright (c) 2010 Walter Bruce Sinclair

Copyright 2006 Stephen Woodbridge

Copyright 2006-2010 Stephen Woodbridge.

Copyright (c) 2006-2014 Stephen Woodbridge.

Copyright (c) 2017, Even Rouault <even.rouault at spatialys.com>

Copyright (C) 2004-2015 Sandro Santilli <strk@kbt.io>

Copyright (C) 2008-2011 Paul Ramsey <pramsey@cleverelephant.ca>

Copyright (C) 2008 Mark Cave-Ayland <mark.cave-ayland@siriusit.co.uk>

Copyright 2015 Nicklas Avén <nicklas.aven@jordogskog.no>

Copyright 2008 Paul Ramsey

Copyright (C) 2012 Sandro Santilli <strk@kbt.io>

Copyright 2012 Sandro Santilli <strk@kbt.io>

Copyright (C) 2014 Sandro Santilli <strk@kbt.io>

Copyright 2013 Olivier Courtin <olivier.courtin@oslandia.com>

Copyright 2009 Paul Ramsey <pramsey@cleverelephant.ca>

Copyright 2008 Paul Ramsey <pramsey@cleverelephant.ca>

Copyright 2011 Sandro Santilli <strk@kbt.io>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.?

GNU GENERAL PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program

(independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>

Copyright (C) <year> <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author

Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.

This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program `Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Software:Geos

Copyright notice：

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refractions Research Inc.

Copyright (C) 2013 Sandro Santilli <strk@keybit.net>

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason (www.grinninglizard.com)

License：LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refractions Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refractions Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refractions Research Inc.

Copyright (C) 2005-2007 Refractions Research Inc.

Copyright (C) 2007 Refractions Research Inc.

Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>

Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Mateusz Loskot

Copyright (C) 2005-2009 Refractions Research Inc.

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refractions Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refractions Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refractions Research Inc.

Copyright (C) 2005-2007 Refractions Research Inc.

Copyright (C) 2007 Refractions Research Inc.

Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>

Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Mateusz Loskot

Copyright (C) 2005-2009 Refractions Research Inc.

Copyright (C) 2001-2009 Vivid Solutions Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Wu Yongwei

Copyright (C) 2012 Excensus LLC.

Copyright (C) 1996-2015 Free Software Foundation, Inc.

Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>

Copyright (C) 2007-2010 Safe Software Inc.

Copyright (C) 2010 Safe Software Inc.

Copyright (C) 2006 Refractions Research

Copyright 2004 Sean Gillies, sgillies@frii.com

Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>

Copyright (C) 2015 Nyall Dawson <nyall dot dawson at gmail dot com>

Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason (www.grinninglizard.com)

Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason (www.grinninglizard.com)

License：LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refractions Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refractions Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refractions Research Inc.

Copyright (C) 2005-2007 Refractions Research Inc.

Copyright (C) 2007 Refractions Research Inc.

Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>

Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Mateusz Loskot

Copyright (C) 2005-2009 Refractions Research Inc.

Copyright (C) 2001-2009 Vivid Solutions Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Wu Yongwei

Copyright (C) 2012 Excensus LLC.

Copyright (C) 1996-2015 Free Software Foundation, Inc.

Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>

Copyright (C) 2007-2010 Safe Software Inc.

Copyright (C) 2010 Safe Software Inc.

Copyright (C) 2006 Refractions Research

Copyright 2004 Sean Gillies, sgillies@frii.com

Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>

Copyright (C) 2015 Nyall Dawson <nyall dot dawson at gmail dot com>

Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason (www.grinninglizard.com)

Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason (www.grinninglizard.com)

License：LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refractions Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason (www.grinninglizard.com)

License：LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refractions Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refractions Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refractions Research Inc.

Copyright (C) 2005-2007 Refractions Research Inc.

Copyright (C) 2007 Refractions Research Inc.

Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>

Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Mateusz Loskot

Copyright (C) 2005-2009 Refractions Research Inc.

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refractions Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refractions Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refractions Research Inc.

Copyright (C) 2005-2007 Refractions Research Inc.

Copyright (C) 2007 Refractions Research Inc.

Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>

Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Mateusz Loskot

Copyright (C) 2005-2009 Refractions Research Inc.

Copyright (C) 2001-2009 Vivid Solutions Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Wu Yongwei

Copyright (C) 2012 Excensus LLC.

Copyright (C) 1996-2015 Free Software Foundation, Inc.

Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>

Copyright (C) 2007-2010 Safe Software Inc.

Copyright (C) 2010 Safe Software Inc.

Copyright (C) 2006 Refractions Research

Copyright 2004 Sean Gillies, sgillies@frii.com

Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>

Copyright (C) 2015 Nyall Dawson <nyall dot dawson at gmail dot com>

Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason (www.grinninglizard.com)

Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason (www.grinninglizard.com)

License：LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refractions Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refractions Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refractions Research Inc.

Copyright (C) 2005-2007 Refractions Research Inc.

Copyright (C) 2007 Refractions Research Inc.

Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>

Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Mateusz Loskot

Copyright (C) 2005-2009 Refractions Research Inc.

Copyright (C) 2001-2009 Vivid Solutions Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Wu Yongwei

Copyright (C) 2012 Excensus LLC.

Copyright (C) 1996-2015 Free Software Foundation, Inc.

Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>

Copyright (C) 2007-2010 Safe Software Inc.

Copyright (C) 2010 Safe Software Inc.

Copyright (C) 2006 Refractions Research

Copyright 2004 Sean Gillies, sgillies@frii.com

Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>

Copyright (C) 2015 Nyall Dawson <nyall dot dawson at gmail dot com>

Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason (www.grinninglizard.com)

Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason (www.grinninglizard.com)

License：LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refractions Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason (www.grinninglizard.com)

License：LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refractions Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refractions Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refractions Research Inc.

Copyright (C) 2005-2007 Refractions Research Inc.

Copyright (C) 2007 Refractions Research Inc.

Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>

Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Mateusz Loskot

Copyright (C) 2005-2009 Refractions Research Inc.

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refractions Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refractions Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refractions Research Inc.

Copyright (C) 2005-2007 Refractions Research Inc.

Copyright (C) 2007 Refractions Research Inc.

Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>

Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Mateusz Loskot

Copyright (C) 2005-2009 Refractions Research Inc.

Copyright (C) 2001-2009 Vivid Solutions Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Wu Yongwei

Copyright (C) 2012 Excensus LLC.

Copyright (C) 1996-2015 Free Software Foundation, Inc.

Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>

Copyright (C) 2007-2010 Safe Software Inc.

License：LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public

Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and

is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in

non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The

former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under

copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an

appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the

Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1

above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an

application does not supply such function or table, the facility still operates, and performs whatever part of

its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in

themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or

collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany

it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to

distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a

work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License.

Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object

file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6,

whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work

under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system,

rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception,

the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on

which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot

use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or

modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein.

You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the

integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot

impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time.

Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is

copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status

of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING

RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the library's name and a brief idea of what it does.>

Copyright (C) <year> <name of author>

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU

Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library `Frob' (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990

Ty Coon, President of Vice

That's all there is to it!

Software: JSON-C

Copyright notice:

Copyright (c) 2004, 2005 Metaparadigm Pte. Ltd.

Copyright (c) 2009-2012 Eric Haszlakiewicz

Copyright (c) 2004, 2005 Metaparadigm Pte Ltd

Copyright (c) 2009 Hewlett-Packard Development Company, L.P.

Copyright 2011, John Resig

Copyright 2011, The Dojo Foundation

Copyright (c) 2012 Eric Haszlakiewicz

Copyright (c) 2009-2012 Hewlett-Packard Development Company, L.P.

Copyright (c) 2008-2009 Yahoo! Inc. All rights reserved.

Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2003, 2004, 2005, 2006,

2007, 2008, 2009, 2010, 2011 Free Software Foundation, Inc.

Copyright (c) 2013 Metaparadigm Pte. Ltd.

License: MIT License

Copyright (c) 2009-2012 Eric Haszlakiewicz

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

See Copyright for the status of this software

"

License: MIT License

Please see above

10 资源监控

GaussDB(DWS)为用户提供了多维度的资源监控视图。可以查看作业的实时资源记录和历史资源记录。

10.1 用户资源监控

背景信息

在多租户管理的框架下，用户可以实时查询所有用户资源（包括内存，CPU 核数，存储空间、临时空间、算子落盘空间和 IO）实时使用情况，也可以查询用户资源的历史使用情况。

说明

- 用户实时资源相关视图/函数为：14.3.158 PG_TOTAL_USER_RESOURCE_INFO、GS_WLM_USER_RESOURCE_INFO；用户历史资源相关表为：14.2.6 GS_WLM_USER_RESOURCE_HISTORY。
- 用户监控可以同时监控快慢车道所有作业的 CPU、IO 和内存使用情况，不再受限于仅监控慢车道作业；
- 当前快车道作业内存和 CPU 不受控，在快车道运行作业占用资源较多情况下，可能出现已用资源大于资源限制的情况；
- DN 监控视图中，IO、内存和 CPU 显示的是本 DN 上资源池资源使用和资源限制信息；
- CN 监控视图中，IO、内存和 CPU 显示的是集群内所有 DN 资源池资源使用和资源限制的累积和；
- DN 每隔 5s 更新一次监控信息，CN 每隔 5s 从 DN 收集一次用户监控信息，因为各实例单独更新/收集用户监控信息，因此各实例监控信息更新时间可能不一致；
- 辅助线程中每隔 30s 自动调用持久化函数，持久化用户监控数据，正常情况下不需要用户单独调用持久化函数持久化用户监控数据；
- 当用户数量较多，集群规模较大时，查询此类实时视图，因 CN/DN 间实时通信开销，会有一定的网络延时；

- 初始管理用户不进行资源监控。

操作步骤

- 查询所有用户的资源限额和资源实时使用情况。

```
SELECT * FROM PG_TOTAL_USER_RESOURCE_INFO;
```

得到的结果视图如下：

```
username       | used memory | total memory | used cpu | total cpu | used space
| total space | used temp space | total temp space | used spill space |
total_spill_space | read_kbytes | write_kbytes | read_counts | write_counts |
read_speed | write_speed
----------------------+-------------+--------------+----------+-----------+---
---------+-------------+-----------------+------------------+------------------
+-------------------+-------------+--------------+-------------+--------------
+------------+-------------
perfadm              |          0 |           0 |       0 |        0 |         0 |
-1 |            0 |            -1 |             0 |             -1 |          0
|          0 |          0 |          0 |         0 |          0
usern                |          0 |       17250 |       0 |       48 |         0 |
-1 |            0 |            -1 |             0 |             -1 |          0
|          0 |          0 |          0 |         0 |          0
userg                |         34 |       15525 |   23.53 |       48 |         0 |
-1 |            0 |            -1 |       814955731 |             -1 |
6111952 |     1145864 |     763994 |      143233 |     42678 |       8001
userg1               |         34 |       13972 |   23.53 |       48 |         0 |
-1 |            0 |            -1 |       814972419 |             -1 |
6111952 |     1145864 |     763994 |      143233 |     42710 |       8007
(4 rows)
```

其中，IO 资源监控字段(read_kbytes、write_kbytes、read_counts、write_counts、read_speed 和 write_speed)需要在 GUC 参数 enable_user_metric_persistent 开启时才有监控数据。

所查各字段说明详见 14.3.158 PG_TOTAL_USER_RESOURCE_INFO 。

- 查询具体某个用户的资源限额和资源实时使用情况。

```
SELECT * FROM GS_WLM_USER_RESOURCE_INFO('username');
```

查询结果如下：

```
userid | used_memory | total_memory | used_cpu | total_cpu | used_space |
total_space | used_temp_space | total_temp_space | used_spill_space |
total_spill_space | read_kbytes | write_kbytes | read_counts | write_counts |
read_speed | write_speed
--------+-------------+--------------+----------+-----------+------------+-----
--------+-----------------+------------------+------------------+--------------
-----+-------------+--------------+-------------+--------------+------------+--
-----------
16407 |          18 |       1655 |       6 |        19 |         13787176 |
-1 |            0 |            -1 |             0 |             -1 |          0
|          0 |          0 |          0 |         0 |          0
(1 row)
```

- 查询所有用户的资源限额和资源历史使用情况。

```
SELECT * FROM GS_WLM_USER_RESOURCE_HISTORY;
```

查询结果如下：

```
username       |          timestamp          | used_memory | total_memory |
used_cpu | total_cpu | used_space | total_space | used_temp_space |
total_temp_space | used_spill_space | total_spill_space | read_kbytes |
write_kbytes | read_counts | write_counts | read_speed  | write_speed
-----------------------+-------------------------------+-------------+---------
-----+----------+-----------+------------+-------------+-----------------+-----
-------------+-----------------+-------------------+-------------+-----------
--+-------------+--------------+-------------+-------------
usern               | 2020-01-08 22:56:06.456855+08 |          0 |       17250 |
0 |       48 |         0 |         -1 |             0 |             -1 |
88349078 |              -1 |      45680 |          34 |       5710 |           8 |
320 |          0
userg               | 2020-01-08 22:56:06.458659+08 |          0 |       15525 |
33.48 |       48 |         0 |         -1 |             0 |             -1 |
110169581 |              -1 |      17648 |          23 |       2206 |           5
|        123 |          0
userg1              | 2020-01-08 22:56:06.460252+08 |          0 |       13972 |
33.48 |       48 |         0 |         -1 |             0 |             -1 |
136106277 |              -1 |      17648 |          23 |       2206 |           5
|        123 |          0
```

对于系统表 14.2.6 GS_WLM_USER_RESOURCE_HISTORY，仅当 GUC 参数 enable_user_metric_persistent 开启时，才会定期将视图 14.3.158 PG_TOTAL_USER_RESOURCE_INFO 中的数据保存到历史表中。

所查各字段说明详见 14.2.6 GS_WLM_USER_RESOURCE_HISTORY。

10.2 资源池资源监控

简介

多租户管理框架下，用户关联资源池执行查询，用户执行查询占用的资源将汇总至关联资源池上，通过资源池监控视图用户可以直观的查询到所有资源池的实时资源使用情况，同时也可以通过资源池监控历史表查询资源池资源的历史使用情况。

资源池监控数据每 5s 更新一次，但是因为 CN 和 DN 时间差，实际监控数据更新时间可能会大于 5s，正常不会超过 10s。资源池监控数据每 30s 持久化一次，资源池监控和用户监控逻辑基本一致，因此共用 GUC 参数控制持久化和老化，使用 GUC 参数 enable_user_metric_persistent 控制是否进行资源池监控数据持久化，使用 GUC 参数 user_metric_retention_time 控制资源池监控数据老化。

资源池监控的资源包含：快慢车道作业运行和排队信息，CPU、内存以及逻辑 IO 资源监控信息。涉及的监控视图和历史表如下所示：

1. 资源池实时运行信息监控视图(单 CN)：14.3.59 GS_RESPOOL_RUNTIME_INFO；
2. 资源池实时运行信息监控视图(所有 CN)：14.3.188 PGXC_RESPOOL_RUNTIME_INFO；
3. 资源池实时资源监控视图(单 CN)：14.3.60 GS_RESPOOL_RESOURCE_INFO；
4. 资源池实时资源监控视图(所有 CN)：14.3.189 PGXC_RESPOOL_RESOURCE_INFO；
5. 资源池历史资源监控表(单 CN)：14.2.2 GS_RESPOOL_RESOURCE_HISTORY；

6. 资源池历史资源监控视图(所有 CN)：14.3.190 PGXC_RESPOOL_RESOURCE_HISTORY；

说明

- 资源池监控可以同时监控快慢车道所有作业的 CPU、IO 和内存使用情况，不再受限于仅监控慢车道作业；
- 当前快车道作业内存和 CPU 不受控，在快车道运行作业占用资源较多情况下，可能出现已用资源大于资源限制的情况；
- DN 资源池监控视图中，IO、内存和 CPU 显示的是本 DN 上资源池资源使用和资源限制信息；
- CN 资源池监控视图中，IO、内存和 CPU 显示的是集群内所有 DN 资源池资源使用和资源限制的累积和；
- DN 每隔 5s 更新一次资源池监控信息，CN 每隔 5s 从 DN 收集一次资源池监控信息，因为各实例单独更新/收集资源池监控信息，因此各实例监控信息更新时间可能不一致；
- 辅助线程中每隔 30s 自动调用持久化函数，持久化资源池监控数据，正常情况下不需要用户单独调用持久化函数持久化资源池监控数据。

操作步骤

- 查询资源池的作业实时运行情况。

```
SELECT * FROM GS_RESPOOL_RUNTIME_INFO;
```

得到的结果视图如下：

```
 nodegroup |    rpname    | ref_count | fast_run | fast_wait | slow_run |
slow_wait
-----------+--------------+-----------+----------+-----------+----------+------
-----
 vc1       | p2           |        10 |        0 |         0 |        0 |
 vc2       | p3           |        10 |        5 |         5 |        0 |
 vc2       | p4           |         0 |        0 |         0 |        0 |
 vc1       | default_pool |         0 |        0 |         0 |        0 |
 vc2       | default_pool |         0 |        0 |         0 |        0 |
 vc1       | p1           |        20 |        5 |         5 |        3 |
(6 rows)
```

其中：

a. ref_count 为引用当前资源池信息的作业数，语句从进入管控到结束一直占用该计数；

b. fast_run 和 slow_run 为负载管理记账信息，只有管控(fast_limit/slow_limit 大于 0)时该值才有效；

c. 该视图仅在 CN 上有效，持久化信息保存在 GS_RESPOOL_RESOURCE_HISTORY 中；

d. 各字段说明详见 14.3.59 GS_RESPOOL_RUNTIME_INFO。

- 查询资源池的资源限额和资源实时使用情况。

```
SELECT * FROM GS_RESPOOL_RESOURCE_INFO;
```

得到的结果视图如下：

```
nodegroup |    rpname    |      cgroup       | ref_count | fast_run | fast_wait |
fast_limit | slow_run | slow_wait | slow_limit | used_cpu | cpu_limit |
used_mem | estimate_mem | mem_limit |read_kbytes | write_kbytes | read_counts |
write_counts | read_speed | write_speed
-----------+--------------+--------------------+-----------+----------+-------
----+------------+----------+-----------+------------+----------+-----------+--
--------+--------------+-----------+------------+--------------+-------------
+--------------+------------+-------------
 vc1       | p2           | DefaultClass:Rush   |       10 |        0 |         0 |
-1 |        0 |         0 |        10 |     9.97 |       48 |       20 |           0 |
11555 |         8 |        2880 |          1 |         360 |         1 |         589
 vc2       | p3           | DefaultClass:Rush   |       10 |        5 |         5 |
5 |        0 |         0 |        10 |     4.98 |       48 |       11 |           0 |
11555 |         0 |         848 |          0 |         106 |         0 |         173
 vc2       | p4           | DefaultClass:Rush   |        0 |        0 |         0 |
-1 |        0 |         0 |        10 |        0 |       48 |        0 |           0 |
11555 |         0 |           0 |          0 |           0 |         0 |           0
 vc1       | default_pool | DefaultClass:Medium |        0 |        0 |         0 |
-1 |        0 |         0 |        -1 |        0 |       48 |        0 |           0 |
11555 |         0 |           0 |          0 |           0 |         0 |           0
 vc2       | default_pool | DefaultClass:Medium |        0 |        0 |         0 |
-1 |        0 |         0 |        -1 |        0 |       48 |        0 |           0 |
11555 |         0 |           0 |          0 |           0 |         0 |           0
 vc1       | p1           | DefaultClass:Rush   |       20 |        5 |         5 |
5 |        3 |         7 |         3 |     7.98 |       48 |       16 |         768 |
11555 |         8 |        2656 |          1 |         332 |         1 |         543
(6 rows)
```

a. 该视图在 CN 和 DN 上均有效，DN 上 CPU、内存和 IO 为本 DN 资源消耗情况，CN 上 CPU、内存和 IO 为集群内所有 DN 上资源消耗的累加和；

b. estimate_mem 仅在动态负载管理情况下 CN 上有效，显示资源池估算内存记账情况；

c. IO 监控信息仅在 enable_logical_io_statistics 开启时才会记录；

d. 各字段说明详见 14.3.60 GS_RESPOOL_RESOURCE_INFO。

- 查询资源池的资源限额和资源历史使用情况。

```
SELECT * FROM GS_RESPOOL_RESOURCE_HISTORY ORDER BY timestamp DESC;
```

得到的结果视图如下：

```
timestamp          | nodegroup   |    rpname    |      cgroup       | ref_count |
fast_run | fast_wait | fast_limit | slow_run | slow_wait | slow_limit |
used_cpu | cpu_limit | used_mem | estimate_mem | mem_limit | read_kbytes |
write_kbytes | read_counts | write_counts | read_speed | write_speed
-------------------------------+-------------+--------------+-------------------
----+-----------+----------+----------+-----------+----------+-----------+---
--------+----------+-----------+----------+-----------+-------------+--------
-----+--------------+-------------+--------------+------------+-------------
 2022-03-04 09:41:57.53739+08  | vc1         | p2           | DefaultClass:Rush
|       10 |       0 |        0 |        -1 |       0 |        0 |        10 |
9.97 |       48 |   20 |          0 |    11555 |          0 |        2320 |
0 |         290 |         0 |        474
 2022-03-04 09:41:57.53739+08  | vc1         | p1           | DefaultClass:Rush
|       20 |       5 |        5 |         5 |       3 |        7 |         3 |
7.98 |       48 |   16 |         768 |    11555 |          0 |        1896 |
```

```
0 |        237 |         0 |        387
 2022-03-04 09:41:57.53739+08  | vc2         | default_pool |
DefaultClass:Medium |        0 |       0 |        0 |        -1 |       0 |
0 |        -1 |       0 |       48 |    0 |          0 |    11555 |          0 |
0 |          0 |          0 |         0 |          0
 2022-03-04 09:41:57.53739+08  | vc1         | default_pool |
DefaultClass:Medium |        0 |       0 |        0 |        -1 |       0 |
0 |        -1 |       0 |       48 |    0 |          0 |    11555 |          0 |
0 |          0 |          0 |         0 |          0
 2022-03-04 09:41:57.53739+08  | vc2         | p4          | DefaultClass:Rush
|        0 |       0 |        0 |        -1 |       0 |        0 |        10 |
0 |       48 |    0 |           0 |    11555 |          0 |          0 |          0
|          0 |         0 |          0
 2022-03-04 09:41:57.53739+08  | vc2         | p3          | DefaultClass:Rush
|       10 |       5 |        5 |         5 |       0 |        0 |        10 |
4.99 |       48 |   11 |          0 |    11555 |          0 |         880 |
0 |         110 |         0 |        180
 2022-03-04 09:41:27.335234+08 | vc2         | p3          | DefaultClass:Rush
|       10 |       5 |        5 |         5 |       0 |        0 |        10 |
4.98 |       48 |   11 |          0 |    11555 |          0 |         856 |
0 |         107 |         0 |        175
```

a. 该监控信息来自资源池监控历史表，enable_user_metric_persistent 开启时每 30 秒记录一次；

b. 该表数据保存时间由 GUC 参数 user_metric_retention_time 控制；

c. 各字段说明详见 14.2.2 GS_RESPOOL_RESOURCE_HISTORY。

10.3 内存资源监控

内存监控

GaussDB(DWS)提供了监控整个集群内存使用状态的视图：

查询 pgxc_total_memory_detail 视图，必须具有 sysadmin 权限。

```
SELECT * FROM pgxc_total_memory_detail;
```

如果查询该视图时出现以下错误，请开启内存管理功能。

```
SELECT * FROM pgxc total memory detail;
ERROR:  unsupported view for memory protection feature is disabled.
CONTEXT:  PL/pgSQL function pgxc_total_memory_detail() line 12 at FOR over EXECUTE
statement
```

开启内存管理功能用户可通过 GaussDB(DWS) 控制台设置 enable_memory_limit 和 max_process_memory 参数，方法如下：

1. 登录 GaussDB(DWS) 管理控制台。
2. 在左侧导航栏中，单击“集群管理”。
3. 在集群列表中找到所需要的集群，单击集群名称，进入集群“基本信息”页面。
4. 单击“参数修改”页签，修改参数“enable_memory_limit”的值为 **on**，然后单击“保存”。

5. 修改参数“max_process_memory”的值为合适的值，修改建议请参见max_process_memory，然后单击“保存”。
6. 在“修改预览”窗口，确认修改无误后，单击“保存”。修改完成后需要重启集群，参数才会生效。

共享内存监控

用户可以通过视图 pg_shared_memory_detail 查询共享内存上下文信息：

```
SELECT * FROM pg_shared_memory_detail;
         contextname          | level |           parent            | totalsize |
freesize | usedsize
------------------------------+-------+-----------------------------+-------
---+----------+----------
 ProcessMemory                |     0 |                             |     24576 |
9840 |    14736
 Workload manager memory context |     1 | ProcessMemory               |   2105400 |
7304 |  2098096
 wlm collector hash table       |     2 | Workload manager memory context |      8192
|     3736 |     4456
 Resource pool hash table       |     2 | Workload manager memory context |     24576
|    15968 |     8608
 wlm cgroup hash table          |     2 | Workload manager memory context |     24576
|    15968 |     8608
(5 rows)
```

该视图列举了内存使用的上下文名称、级别、上级内存上下文、共享内存总大小等。

另外，在数据库中，GUC 参数“memory_tracking_mode”用来设置内存信息统计的模式，共支持四种模式：

- none，不启动内存统计功能。
- normal，仅做内存实时统计，不生成文件。
- executor，生成统计文件，包含执行层使用过的所有已分配内存的上下文信息。

  当为 executor 模式时，将在 DN 进程的 pg_log 目录下生成 cvs 格式文件，命名方式为：memory_track_<DN 名称>_query_<queryid>.csv。作业执行时，执行器 postgres 线程和所有 stream 线程执行的算子信息，都将输入该文件。

  文件内容根据以下面内容组成实例如下：

```
0, 0, ExecutorState, 0, PortalHeapMemory, 0, 40K, 602K, 23
1, 3, CStoreScan_29360131_25, 0, ExecutorState, 1, 265K, 554K, 23
2, 128, cstore scan per scan memory context, 1, CStoreScan_29360131_25, 2, 24K,
24K, 23
3, 127, cstore scan memory context, 1, CStoreScan_29360131_25, 2, 264K, 264K,
23
4, 7, InitPartitionMapTmpMemoryContext, 1, CStoreScan_29360131_25, 2, 31K, 31K,
23
5, 2, VecPartIterator_29360131_24, 0, ExecutorState, 1, 16K, 16K, 23
0, 0, ExecutorState, 0, PortalHeapMemory, 0, 24K, 1163K, 20
1, 3, CStoreScan_29360131_22, 0, ExecutorState, 1, 390K, 1122K, 20
2, 20, cstore scan per scan memory context, 1, CStoreScan_29360131_22, 2, 476K,
476K, 20
3, 19, cstore scan memory context, 1, CStoreScan_29360131_22, 2, 264K, 264K, 20
```

```
4, 7, InitPartitionMapTmpMemoryContext, 1, CStoreScan_29360131_22, 2, 23K, 23K,
20
5, 2, VecPartIterator_29360131_21, 0, ExecutorState, 1, 16K, 16K, 20
```

其中各字段分别为：输出顺序号、线程内分配内存上下文的顺序号、当前内存上下文的名称、父内存上下文的输出顺序号、父内存上下文的名称、内存上下文树形层次级别号、当前内存上下文使用的内存峰值、当前内存上下文及其所有子内存上下文使用的内存峰值、当前线程所在 query 的 plannodeid。

在本例中，记录“1, 3, CStoreScan_29360131_22, 0, ExecutorState, 1, 390K, 1122K, 20”和 Explain Analyze 的对应关系如下：

- CstoreScan_29360131_22 代表 CstoreScan 算子。
- 1122K 代表 CstoreScan 算子的 PeakMemory。

- fullexec，生成文件包含执行层申请过的所有内存上下文信息。

  当设置为 fullexec 模式时，输出信息和 executor 模式相同，但可能增加部分内存上下文分配信息，因为所有申请内存（无论是否申请成功）相关的信息都会被打印出来。由于仅记录内存申请信息，故记录中内存上下文使用的峰值均为 0。

10.4 实例资源监控

GaussDB(DWS)提供了监控 CN、DN 实例资源使用状态（包括内存，CPU，磁盘 IO，进程物理 IO 和进程逻辑 IO）的系统表及监控整个集群资源使用状态的系统表。

关于系统表 GS_WLM_INSTANCE_HISTORY 的详细介绍，请参考 14.2.3 GS_WLM_INSTANCE_HISTORY。

说明

系统表 GS_WLM_INSTANCE_HISTORY 中的数据分布在对应的实例中，CN 实例监控数据保存在 CN 实例中，DN 实例监控数据保存在 DN 实例中；DN 实例由于有备机，当主 DN 实例异常时，该 DN 实例的监控数据能够从备机恢复；但 CN 实例无备机，当某 CN 实例异常再恢复时，该 CN 实例的监控数据会丢失。

操作步骤

- 查询当前实例最近的资源使用情况。

```
SELECT * FROM GS_WLM_INSTANCE_HISTORY ORDER BY TIMESTAMP DESC;
```

查询结果如下：

```
instancename |         timestamp         | used_cpu | free_mem | used_mem |
io_await | io_util  | disk_read | disk_write | process_read | process_write |
logical_read | logical_write | read_counts | write_counts
--------------+-------------------------------+----------+----------+----------
+----------+----------+-----------+------------+--------------+---------------
+--------------+---------------+-------------+--------------
dn_6015_6016 | 2022-01-10 17:29:17.329495+08 |        0 |    14570 |     8982 |
662.923 |  99.9601 |    697666 |    93655.5 |      183104 |        30082 |
285659 |        30079 |      357717 |       37667
dn_6015_6016 | 2022-01-10 17:29:07.312049+08 |        0 |    14578 |     8974 |
883.102 |  99.9801 |    756228 |    81417.4 |      189722 |        30786 |
```

```
285681 |        30780 |      358103 |       38584
dn_6015_6016 | 2022-01-10 17:28:57.284472+08 |       0 |    14583 |     8969 |
727.135 |  99.9801 |    648581 |    88799.6 |      177120 |        31176 |
252161 |        31175 |      316085 |       39079
dn_6015_6016 | 2022-01-10 17:28:47.256613+08 |       0 |    14591 |     8961 |
679.534 |   100.08 |    655360 |    169962 |      179404 |        30424 |
242002 |        30422 |      303351 |       38136
```

- 查询当前实例某一段时间内的资源使用情况。

```
SELECT * FROM GS_WLM_INSTANCE_HISTORY WHERE TIMESTAMP > '2022-01-10' AND
TIMESTAMP < '2020-01-11' ORDER BY TIMESTAMP DESC;
```

查询结果如下：

```
instancename |          timestamp          | used_cpu | free_mem | used_mem |
io_await | io_util  | disk_read | disk_write | process_read | process_write |
logical_read | logical_write | read_counts | write_counts
--------------+-------------------------------+----------+----------+----------
+----------+----------+-----------+------------+--------------+---------------
+--------------+---------------+-------------+--------------
dn_6015_6016 | 2022-01-10 17:29:17.329495+08 |       0 |    14570 |     8982 |
662.923 |  99.9601 |    697666 |    93655.5 |      183104 |        30082 |
285659 |        30079 |      357717 |       37667
dn_6015_6016 | 2022-01-10 17:29:07.312049+08 |       0 |    14578 |     8974 |
883.102 |  99.9801 |    756228 |    81417.4 |      189722 |        30786 |
285681 |        30780 |      358103 |       38584
dn_6015_6016 | 2022-01-10 17:28:57.284472+08 |       0 |    14583 |     8969 |
727.135 |  99.9801 |    648581 |    88799.6 |      177120 |        31176 |
252161 |        31175 |      316085 |       39079
dn_6015_6016 | 2022-01-10 17:28:47.256613+08 |       0 |    14591 |     8961 |
679.534 |   100.08 |    655360 |    169962 |      179404 |        30424 |
242002 |        30422 |      303351 |       38136
```

- 查询集群最近的资源使用情况，可以在 CN 节点上调用 pgxc_get_wlm_current_instance_info 存储过程函数。

```
SELECT * FROM pgxc_get_wlm_current_instance_info('ALL');
```

查询结果如下：

```
instancename |          timestamp          | used_cpu | free_mem | used_mem |
io_await | io_util | disk_read | disk_write | process_read | process_write |
logical_read | logical_write | read_counts | write_counts
--------------+-------------------------------+----------+----------+----------
+----------+---------+-----------+------------+--------------+---------------+-
-------------+---------------+-------------+--------------
coordinator2 | 2020-01-14 21:58:29.290894+08 |       0 |    12010 |      278 |
16.0445 | 7.19561 |   184.431 |    27959.3 |          0 |          10 |
0 |           0 |         0 |          0
coordinator3 | 2020-01-14 21:58:27.567655+08 |       0 |    12000 |      288
|  .964557 | 3.40659 |   332.468 |    3375.02 |         26 |          13 |
0 |           0 |         0 |          0
datanode1    | 2020-01-14 21:58:23.900321+08 |       0 |    11899 |      389 |
1.17296 |    3.25 |     329.6 |     2870.4 |         28 |           8 |
13 |           3 |        18 |          6
datanode2    | 2020-01-14 21:58:32.832989+08 |       0 |    11904 |      384 |
17.948 | 8.52148 |   214.186 |    25894.1 |         28 |          10 |
13 |           3 |        18 |          6
datanode3    | 2020-01-14 21:58:24.826694+08 |       0 |    11894 |      394 |
```

```
1.16088 |    3.15 |      328 |    2868.8 |         25 |          10 |          13
|           3 |         18 |           6
coordinator1 | 2020-01-14 21:58:33.367649+08 |       0 |    11988 |     300 |
9.53286 |   10.05 |     43.2 |     55232 |          0 |           0 |           0
|           0 |          0 |           0
coordinator1 | 2020-01-14 21:58:23.216645+08 |       0 |    11988 |     300 |
1.17085 | 3.21182 |   324.729 |    2831.13 |          8 |          13 |
0 |           0 |          0 |           0
(7 rows)
```

- 查询集群历史的资源使用情况，可以在 CN 节点上调用 pgxc_get_wlm_current_instance_info 存储过程函数。

```
SELECT * FROM pgxc_get_wlm_history_instance_info('ALL', '2020-01-14 21:00:00',
'2020-01-14 22:00:00', 3);
```

查询结果如下：

```
instancename |          timestamp          | used_cpu | free_mem | used_mem |
io_await |  io_util  | disk_read | disk_write | process_read | process_write |
logical_read | logical_write | read_counts | write_counts
--------------+-------------------------------+----------+----------+----------
+----------+-----------+-----------+-----------+------------+--------------+---------------
+--------------+---------------+-------------+--------------
coordinator2 | 2020-01-14 21:50:49.778902+08 |       0 |    12020 |     268
|  .127371 |   .789211 |    15.984 |    3994.41 |          0 |           0 |
0 |           0 |          0 |           0
coordinator2 | 2020-01-14 21:53:49.043646+08 |       0 |    12018 |     270 |
30.2902 |   8.65404 |    276.77 |    16741.8 |          3 |           1 |
0 |           0 |          0 |           0
coordinator2 | 2020-01-14 21:57:09.202654+08 |       0 |    12018 |     270
|   .16051 |   .979021 |   59.9401 |      5596 |          0 |           0 |
0 |           0 |          0 |           0
coordinator3 | 2020-01-14 21:38:48.948646+08 |       0 |    12012 |     276
| .0769231 | .00999001 |         0 |    35.1648 |          0 |           1 |
0 |           0 |          0 |           0
coordinator3 | 2020-01-14 21:40:29.061178+08 |       0 |    12012 |     276
|  .118421 |  .0199601 |         0 |    970.858 |          0 |           0 |
0 |           0 |          0 |           0
coordinator3 | 2020-01-14 21:50:19.612777+08 |       0 |    12010 |     278 |
24.411 |   11.7665 |   8.78244 |    44641.1 |          0 |           0 |
0 |           0 |          0 |           0
datanode1    | 2020-01-14 21:49:42.758649+08 |       0 |    11909 |     379
|  .798776 |     8.02 |     51.2 |    20924.8 |          0 |           0 |
0 |           0 |          0 |           0
datanode1    | 2020-01-14 21:49:52.760188+08 |       0 |    11909 |     379 |
23.8972 |     14.1 |        0 |     74760 |          0 |           0 |
0 |           0 |          0 |           0
datanode1    | 2020-01-14 21:50:22.769226+08 |       0 |    11909 |     379 |
39.5868 |      7.4 |        0 |    19760.8 |          0 |           0 |
0 |           0 |          0 |           0
datanode2    | 2020-01-14 21:58:02.826185+08 |       0 |    11905 |     383
|  .351648 |      .32 |     20.8 |     504.8 |          0 |           0 |
0 |           0 |          0 |           0
datanode2    | 2020-01-14 21:56:42.80793+08  |       0 |    11906 |     382
|  .559748 |      .04 |        0 |     326.4 |          0 |           0 |
0 |           0 |          0 |           0
```

```
datanode2    | 2020-01-14 21:45:21.632407+08 |       0 |    11901 |     387 |
12.1313 |   4.55544 |    3.1968 |    45177.2 |          0 |           0 |
0 |           0 |          0 |           0
datanode3    | 2020-01-14 21:58:14.823317+08 |       0 |    11898 |     390
|  .378205 |       .99 |       48 |    23353.6 |          0 |           0 |
0 |           0 |          0 |           0
datanode3    | 2020-01-14 21:47:50.665028+08 |       0 |    11901 |     387 |
1.07494 |     1.19 |        0 |    15506.4 |          0 |           0 |
0 |           0 |          0 |           0
datanode3    | 2020-01-14 21:51:21.720117+08 |       0 |    11903 |     385 |
10.2795 |     3.11 |        0 |    11031.2 |          0 |           0 |
0 |           0 |          0 |           0
coordinator1 | 2020-01-14 21:42:59.121945+08 |       0 |    12020 |     268
| .0857143 |  .0699301 |        0 |    6579.02 |          0 |           0 |
0 |           0 |          0 |           0
coordinator1 | 2020-01-14 21:41:49.042646+08 |       0 |    12020 |     268 |
20.9039 |   11.3786 |   6042.76 |    57903.7 |          0 |           0 |
0 |           0 |          0 |           0
coordinator1 | 2020-01-14 21:41:09.007652+08 |       0 |    12020 |     268
| .0446429 |    .03996 |        0 |    1109.29 |          0 |           0 |
0 |           0 |          0 |           0
(18 rows)
```

10.5 实时 TopSQL

系统提供了不同级别的资源监控实时视图用来查询实时 TopSQL。资源监控实时视图记录了查询作业运行时的资源使用情况(包括内存、下盘、CPU 时间等)以及性能告警信息。

实时视图具体的对外接口如下表所示：

表10-1 资源监控实时视图

视图级别	节点范围	查询视图
query 级别/perf 级别	当前 CN	14.3.80 GS_WLM_SESSION_STATISTICS
	所有 CN	14.3.215 PGXC_WLM_SESSION_STATISTICS
operator 级别	当前 CN	14.3.77 GS_WLM_OPERATOR_STATISTICS
	所有 CN	14.3.212 PGXC_WLM_OPERATOR_STATISTICS

说明

- 视图级别取决于资源监控的等级，即参数 resource_track_level 的配置。

- perf 和 operator 级别会影响 14.3.80 GS_WLM_SESSION_STATISTICS/14.3.213 PGXC_WLM_SESSION_INFO 中的 query_plan 和 warning 字段的取值，详细内容参见 11.3.4.1 SQL 自诊断。
- 对外接口通过不同的前缀(gs 与 pgxc)来区分单 CN 查询视图以及集群级别查询视图。普通用户仅支持登录到集群的某个 CN 查询以 gs 为前缀的视图。
- 查询此类实时视图时，因需要获取作业运行实时资源使用情况，会有一定的网络延时。
- 实例故障时，实时 TopSQL 视图有可能记录不全。
- 实时 TopSQL 中能够记录的 SQL 语句的规格是：
- 不记录特殊数据定义语句，如：SET、RESET、SHOW、ALTER SESSION SET、SET CONSTRAINTS 语句；
- 记录数据定义语句，例如：执行 CREATE、ALTER、DROP、GRANT、REVOKE 和 VACUUM 语句；
- 记录数据操作语句，例如：
- 执行 SELECT、INSERT、UPDATE 和 DELETE 语句。
- 执行 explain analyze 和 explain performance 场景。
- 使用 query 级别/perf 级别视图。
- 记录函数与存储过程的调用入口语句，当 GUC 参数 enable_track_record_subsql 开启的情况下，可记录存储过程的部分内部语句(declare 定义语句除外)，仅会记录其中下发到 DN 执行的内部语句，未下发到 DN 执行的内部语句会被过滤掉；
- 记录匿名块语句，当 GUC 参数 enable_track_record_subsql 开启的情况下，可记录匿名块中的部分内部语句，仅会记录其中下发到 DN 执行的内部语句，未下发到 DN 执行的内部语句会被过滤掉；
- 记录游标语句，当游标并非从缓存中读取数据，而确实触发语句下发到 DN 上执行的条件下，该游标语句会被记录，并且会进行语句、执行计划增强，但当游标从缓存中读取数据时，不进行记录；当游标语句在匿名块或者函数中使用时，当游标从 DN 上读取较多数据但不完全使用时，因当前架构限制，无法记录该游标在 DN 上的监控信息。对于 With Hold 游标，该语法执行逻辑特殊，会在事务提交阶段执行实际查询动作，当语句在该阶段执行报错时，作业的 aborted 状态无法反馈到 TopSQL 历史表中。
- 重分布过程中的作业不统计；
- JDBC 执行的带占位符语句，通常会补齐参数内容，但如果参数和原语句合起来长度超过 64KB，则不记录参数，或者如果是轻量化语句，直接下发到 DN 上执行，不记录参数。

前提条件

- GUC 参数 enable_resource_track 为 on （默认为 on）。
- GUC 参数 resource_track_level 为 query、perf 或 operator（默认为 query）。
- 监控作业的类型为：
  - 优化器估算的执行代价大于或等于 resource_track_cost 取值的作业。
- Cgroups 功能正常加载，可通过 gs_cgroup -P 查看控制组信息。

- GUC 参数 enable_track_record_subsql 控制是否记录存储过程、匿名块内部语句。

在上述条件中，enable_resource_track 为系统级参数，用于设置是否开启资源监控功能。resource_track_level 为 session 级参数，可以对某个 session 的资源监控级别进行灵活设置。这两个参数的设置方法如下表：

表10-2 设置资源监控信息统计级别

enable_resource_track	resource_track_level	query 级别信息	算子级别信息
on(default)	none	不统计	不统计
	query(default)	统计	不统计
	perf	统计	不统计
	operator	统计	统计
off	none/query/operator	不统计	不统计

操作步骤

步骤 1 通过视图 gs_session_cpu_statistics 查询实时 CPU 信息。

```
SELECT * FROM gs_session_cpu_statistics;
```

步骤 2 通过视图 gs_session_memory_statistics 查询实时 memory 信息。

```
SELECT * FROM gs_session_memory_statistics;
```

步骤 3 通过视图 gs_wlm_session_statistics 查询当前 CN 的实时资源。

```
SELECT * FROM gs_wlm_session_statistics;
```

步骤 4 通过视图 pgxc_wlm_session_statistics 查询所有 CN 的实时资源。

```
SELECT * FROM pgxc_wlm_session_statistics;
```

步骤 5 通过视图 gs_wlm_operator_statistics 查询当前 CN 作业算子执行实时资源信息。

```
SELECT * FROM gs_wlm_operator_statistics;
```

步骤 6 通过视图 pgxc_wlm_operator_statistics 查询所有 CN 作业算子执行实时资源信息。

```
SELECT * FROM pgxc_wlm_operator_statistics;
```

步骤 7 通过视图 pg_session_wlmstat 查询当前用户执行作业正在运行时的负载管理信息。

```
SELECT * FROM pg_session_wlmstat;
```

步骤 8 通过视图 pgxc_wlm_workload_records（动态负载功能开启，即 enable_dynamic_workload 为 on 时该视图有效）查询当前用户在每个 CN 上作业执行时的状态信息。

```
SELECT * FROM pgxc_wlm_workload_records;
```

----结束

10.6 历史 TopSQL

系统提供了资源监控历史视图用例查询历史 TopSQL。资源监控历史视图记录了查询作业运行结束时的资源使用情况(包括内存、下盘、CPU 时间等)和运行状态信息(包括报错、终止、异常等)以及性能告警信息。但对于由于 FATAL、PANIC 错误导致查询异常结束时，状态信息列只显示 aborted，无法记录详细异常信息。特别的，对于查询解析，优化阶段的状态信息则无法监控。

历史视图具体的对外接口如下表所示：

视图级别	节点范围	查询视图	
query 级别/perf 级别	当前 CN	历史（Database Manager 接口）	14.3.79 GS_WLM_SESSION_HISTORY
		历史（内部转储接口）	14.3.78 GS_WLM_SESSION_INFO
	所有 CN	历史（Database Manager 接口）	14.3.214 PGXC_WLM_SESSION_HISTORY
		历史（内部转储接口）	14.3.213 PGXC_WLM_SESSION_INFO
operator 级别	当前 CN	历史（Database Manager 接口）	14.3.76 GS_WLM_OPERATOR_HISTORY
		历史（内部转储接口）	14.3.75 GS_WLM_OPERARROR_INFO
	所有 CN	历史（Database Manager 接口）	14.3.210 PGXC_WLM_OPERATOR_HISTORY
		历史（内部转储接口）	14.3.211 PGXC_WLM_OPERATOR_INFO

说明

- 视图级别取决于资源监控的等级，即参数 resource_track_level 的配置。
- perf 和 operator 级别会影响 14.3.80 GS_WLM_SESSION_STATISTICS/14.3.213 PGXC_WLM_SESSION_INFO 中的 query_plan 和 warning 字段的取值，详细内容参见 11.3.4.1 SQL 自诊断。

- 对外接口通过不同的前缀(gs 与 pgxc)来区分单 CN 查询视图以及集群级别查询视图。普通用户仅支持登录到集群的某个 CN 查询以 gs 为前缀的视图。
- 实例故障时，历史 TopSQL 视图有可能记录不全。
- 在某些异常的情况下，历史 TopSQL 中的状态信息列可能会显示为 unknown，其记录的监控信息会导致不准确。
- 历史 TopSQL 能够记录的 SQL 语句的规格与实时 TopSQL 能够记录的 SQL 语句的规格一致。请参考实时 TopSQL 中能够记录的 SQL 语句的规格。
- 历史 TopSQL 只有在 GUC 参数 enable_resource_record 开启时才会记录数据。
- 查询历史 TopSQL Query 以及算子级别数据时，仅能通过 postgres 数据库进行访问。
- 历史 TopSQL 侧重于查询性能的定位定界辅助分析，不作为审计功能使用，不记录语法分析报错类语句。

前提条件

- GUC 参数 enable_resource_track 为 on （默认为 on）。
- GUC 参数 resource_track_level 为 query、perf 或 operator（默认为 query）。设置方法详见表 10-2。
- GUC 参数 enable_resource_record 为 on（默认为 on）。
- GUC 参数 resource_track_duration 小于作业执行时间（默认为 60s）。
- GUC 参数 enable_track_record_subsql 控制是否记录存储过程、匿名块内部语句（默认为 off）。
- 监控作业类型为：
  - 资源监控实时视图（参见表 10-1）中记录的作业结束时的执行时间大于或等于 resource_track_duration 的作业。
- Cgroups 功能正常加载，可通过 gs_cgroup -P 查看控制组信息。

操作步骤

步骤 1 通过视图 gs_wlm_session_history 查询当前 CN 最近执行作业结束后的负载记录。

```
SELECT * FROM gs_wlm_session_history;
```

步骤 2 通过视图 pgxc_wlm_session_history 查询所有 CN 最近执行作业结束后的负载记录。

```
SELECT * FROM pgxc_wlm_session_history;
```

步骤 3 通过数据表 gs_wlm_session_info 查询当前 CN 作业执行结束后的负载记录。要查到历史记录，必须保证 enable_resource_record 为 on。

```
SELECT * FROM gs_wlm_session_info;
```

- 消耗内存最多的 10 个 Query（可指定查询时间段）

```
SELECT * FROM gs_wlm_session_info order by max_peak_memory desc limit 10;
SELECT * FROM gs_wlm_session_info WHERE start_time >= '2022-05-15 21:00:00' and
finish_time <='2022-05-15 23:30:00'order by max_peak_memory desc limit 10;
```

- 消耗 CPU 最多的 10 个 Query

```
SELECT * FROM gs_wlm_session_info order by total_cpu_time desc limit 10;
SELECT * FROM gs_wlm_session_info WHERE start_time >= '2022-05-15 21:00:00' and
finish_time <='2022-05-15 23:30:00'order by total_cpu_time desc limit 10;
```

步骤 4 通过视图 pgxc_wlm_session_info 查询所有 CN 的作业执行结束后的负载记录。要查到历史记录，必须保证 enable_resource_record 为 on。

```
SELECT * FROM pgxc_wlm_session_info;
```

- 查询所有 CN 消耗时间最多的 10 个 query（可指定查询时间段）

```
SELECT * FROM pgxc_wlm_session_info order by duration desc limit 10;
SELECT * FROM pgxc_wlm_session_info WHERE start_time >= '2022-05-15 21:00:00' and
finish_time <='2022-05-15 23:30:00'order by nodename,max_peak_memory desc limit 10;
```

步骤 5 通过函数 pgxc_get_wlm_session_info_bytime 对视图 pgxc_wlm_session_info 进行筛选查询，要查到历史记录，必须保证 enable_resource_record 为 on。在统计数据量很大的场景中，建议使用该函数进行查询。

说明

GaussDB(DWS)集群默认使用时区为 UTC 时间，与系统时间存在 8h 时差，请确保数据库时间与系统时间一致后进行以下查询。

- 查询所有 CN 上开始时间介于“2019-09-10 15:30:00”和“2019-09-10 15:35:00”之间的 query，每个 CN 最多返回 10 条记录

```
SELECT * FROM pgxc_get_wlm_session_info_bytime('start_time', '2019-09-10 15:30:00',
'2019-09-10 15:35:00', 10);
```

- 查询所有 CN 上结束时间介于“2019-09-10 15:30:00”和“2019-09-10 15:35:00”之间的 query，每个 CN 最多返回 10 条记录

```
SELECT * FROM pgxc_get_wlm_session_info_bytime('finish_time', '2019-09-10 15:30:00',
'2019-09-10 15:35:00', 10);
```

步骤 6 通过视图 gs_wlm_operator_history 查询当前 CN 作业算子最近执行资源信息。要查到记录，必须保证 resource_track_level 为 operator。

```
SELECT * FROM gs_wlm_operator_history;
```

步骤 7 通过视图 pgxc_wlm_operator_history 查询所有 CN 作业算子最近执行资源信息。要查到记录，必须保证 resource_track_level 为 operator。

```
SELECT * FROM pgxc_wlm_operator_history;
```

步骤 8 通过数据表 gs_wlm_operator_info 查询当前 CN 作业算子历史执行资源信息。要查到记录，必须保证 resource_track_level 为 operator 和 enable_resource_record 为 on。

```
SELECT * FROM gs_wlm_operator_info;
```

步骤 9 通过视图 pgxc_wlm_operator_info 查询所有 CN 作业算子历史执行资源信息。要查到记录，必须保证 resource_track_level 为 operator 和 enable_resource_record 为 on。

```
SELECT * FROM pgxc_wlm_operator_info;
```

----结束

说明

- 对于上述的视图信息，由于预设内存的限制，内存中能够保留的数据记录数量有限，实时查询在结束后会导入到历史相关的视图中。关于记录上限，对于 query 级别视图，当新的需要记录的查询超过内存约束记录数上限时，则当前查询无法记录，下条查询重新进行规则判断；在每个 CN 上，记 query 级别历史视图的内存占用（默认 100MB）可通过 14.3.156 PG_TOTAL_MEMORY_DETAIL 视图进行查询。
- 对于算子级别视图，当需要记录的查询的 plan_node 数量加上当前内存中已有的记录数量超过内存约束记录数上限时，则当前查询的所有算子节点不记录，下条查询重新按照算子规则判定。在每个 CN 上，记算子级别视图在内存中可记录的最大实时和历史记录数分别为 max_oper_realt_num(当前系统值为 56987)，max_oper_hist_num(113975)；记当前用户业务系统的平均每个查询的节点数为 num_plan_node，则在每个 CN 上，实时视图允许客户执行的最大并发数: num_realt_active = max_oper_realt_num/num_plan_node；历史视图允许客户执行的最大并发数：num_hist_active = max_oper_hist_num/(180/run_time)/num_plan_node。
- 如果并发过高，避免需要记录的查询数量超过 query 级别视图和算子级别视图的值，可以通过参数 session_history_memory 修改历史查询视图的内存，内存增大和查询数量成正比。

10.7 TopSQL 查询示例

本章节以查询 TPC-DS 样例数据的作业为例，演示如何查看 10.5 实时 TopSQL 和 10.6 历史 TopSQL。

配置集群参数

查询 TopSQL 资源监控信息之前，需要先配置相关的 GUC 参数，以便能查询到作业的资源监控历史信息或归档信息。步骤如下：

1. 登录 GaussDB(DWS)管理控制台。
2. 在“集群管理”页面，找到所需要的集群，单击集群名称，进入集群详情页面。
3. 单击“参数修改”标签页，可以看到当前集群的参数值。
4. 修改参数 resource_track_duration 值为合适的值，单击“保存”按钮进行保存。

说明

enable_resource_record 开关打开后，会引起存储空间膨胀及轻微性能影响，不用时请关闭。

5. 返回集群管理页面，单击右上角的刷新按钮，等待集群参数配置完成。

TopSQL 查询示例

本示例以 TPC-DS 样例数据为例。

步骤 1 打开 SQL 客户端工具，连接到您的数据库。

步骤 2 使用 explain 语句查询所要执行的 SQL 语句的预估代价，从而可以确定该 SQL 语句是否会进行资源监控。

默认执行代价大于 resource_track_cost 的查询才会进行资源监控，用户才可以查询到相关的资源监控信息。

例如，执行如下语句查询该 SQL 语句的预估执行代价：

```
SET CURRENT_SCHEMA = tpcds;
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 = 'TN'
AND ctr1.ctr_customer_sk = c_customer_sk
ORDER BY c_customer_id
limit 100;
```

查询结果如下所示，第一行的 E-costs 列的值即为当前语句的预估代价。

图10-1 explain 查询结果

```
id |                                operation                                | E-rows | E-width | E-costs
---+-------------------------------------------------------------------------+--------+---------+---------
 1 | ->  Row Adapter                                                         |      6 |      20 | 153.06
 2 |    ->  Vector Limit                                                     |      6 |      20 | 153.06
 3 |       ->  Vector Streaming (type: GATHER)                               |      6 |      20 | 153.06
 4 |          ->  Vector Limit                                               |      6 |      20 | 152.84
 5 |             ->  Vector Sort                                             |      6 |      20 | 152.84
 6 |                ->  Vector Hash Join (7,26)                              |      6 |      20 | 152.83
 7 |                   ->  Vector Streaming(type: REDISTRIBUTE)              |      6 |       4 | 134.57
 8 |                      ->  Vector Hash Join (9,18)                        |      6 |       4 | 134.46
 9 |                         ->  Vector Hash Join (10,11)                    |      1 |      44 | 97.33
10 |                            ->  CStore Scan on store                     |      1 |       4 | 60.23
11 |                            ->  Vector Subquery Scan on ctr1             |      6 |      40 | 37.07
12 |                               ->  Vector Hash Aggregate                 |      6 |      54 | 37.06
13 |                                  ->  Vector Streaming(type: REDISTRIBUTE)|      6 |      22 | 37.04
14 |                                     ->  Vector Hash Join (15,17)        |      6 |      22 | 37.00
15 |                                        ->  Vector Streaming(type: BROADCAST)|      6 |       4 | 18.74
16 |                                           ->  CStore Scan on date_dim   |      1 |       4 | 18.06
17 |                                        ->  CStore Scan on store_returns |     60 |      26 | 18.02
18 |                         ->  Vector Hash Aggregate                       |      6 |      68 | 37.09
19 |                            ->  Vector Subquery Scan on ctr2             |      6 |      36 | 37.07
20 |                               ->  Vector Hash Aggregate                 |      6 |      54 | 37.06
21 |                                  ->  Vector Streaming(type: REDISTRIBUTE)|      6 |      22 | 37.04
22 |                                     ->  Vector Hash Join (23,25)        |      6 |      22 | 37.00
23 |                                        ->  Vector Streaming(type: BROADCAST)|      6 |       4 | 18.74
24 |                                           ->  CStore Scan on date_dim   |      1 |       4 | 18.06
25 |                                        ->  CStore Scan on store_returns |     60 |      26 | 18.02
26 |                   ->  CStore Scan on customer                           |     60 |      24 | 18.02
(26 rows)
```

本示例为了演示 TopSQL 的资源监控功能，需要将 resource_track_cost 参数设置为比 explain 查询结果中的预估代价小的一个值，例如 100，设置方法请参见 resource_track_cost。

说明

在完成本示例后，仍然要将 resource_track_cost 设置为原始的默认值 100000 或者一个比较合理的值，否则参数值太小会影响数据库性能。

步骤 3 执行 SQL 语句。

```
SET CURRENT_SCHEMA = tpcds;
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 = 'TN'
AND ctr1.ctr_customer_sk = c_customer_sk
ORDER BY c_customer_id
limit 100;
```

步骤 4 在 SQL 语句执行期间，查询该条 SQL 语句在当前 CN 上的 Memory 峰值的实时信息。

```
SELECT query,max_peak_memory,average_peak_memory,memory_skew_percent FROM
gs_wlm_session_statistics ORDER BY start_time DESC;
```

含义：查询 **query 级别**的 SQL 语句的 Memory 峰值**实时信息**（语句在所有 DN 上的每秒最大 Memory 峰值，在所有 DN 上的每秒平均 Memory 峰值，在 DN 间的 Memory 倾斜率）

实时 TopSQL 资源监控信息的更多查询示例，请参见 10.5 实时 TopSQL。

步骤 5 等待步骤 3 中的 SQL 执行完成，然后查询该语句执行期间的资源监控历史信息。

```
select query,start_time,finish_time,duration,status from gs_wlm_session_history
order by start_time desc;
```

含义：查询 **query 级别**的 SQL 语句执行期间的**历史信息**（语句执行的开始时间，结束时间，实际执行时间，执行状态），时间单位为 ms。

历史 TopSQL 资源监控信息的更多查询示例，请参见 10.6 历史 TopSQL。

步骤 6 等待步骤 3 中的 SQL 执行结束的三分钟后，在 info 视图中查询该语句的资源监控历史信息。

如果设置参数 enable_resource_record 为“on”，且步骤 3 中 SQL 的执行时间不小于 resource_track_duration 所设置的值，该条语句的历史信息将会在三分钟后被归档到 gs_wlm_session_info 视图中。

对于 info 视图，只支持在连接 postgres 数据库时查询。因此，请切换为连接 postgres 数据库后，再执行以下语句进行查询：

```
select query,start_time,finish_time,duration,status from gs_wlm_session_info order
by start_time desc;
```

----结束

11 优化查询性能

11.1 优化查询性能概述

SQL 调优的唯一目的是“资源利用最大化”，即 CPU、内存、磁盘 IO、网络 IO 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。

本章主要讲述了查询的分析和改进方法，并且为用户提供了一些常见案例以及错误处理办法。

11.2 分析查询

11.2.1 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 11-1 和表 11-1 所示。其中，红色字体部分为 DBA 可以介入实施调优的环节。

图11-1 SQL 引擎执行查询类 SQL 语句的流程

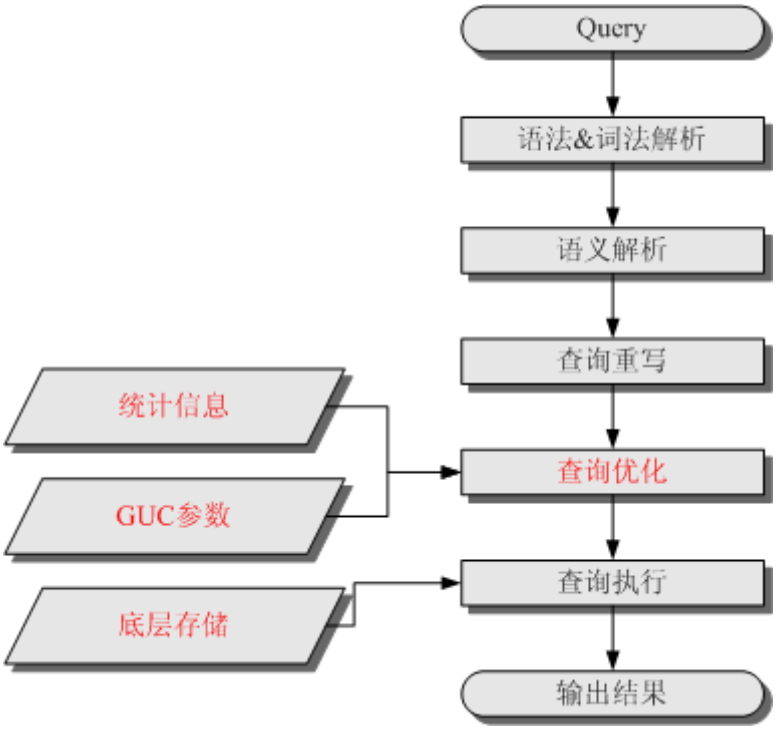


表11-1 SQL 引擎执行查询类 SQL 语句的步骤说明

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。详见调优手段之底层存储。

调优手段之统计信息

GaussDB(DWS)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。

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(DWS)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。

调优手段之底层存储

GaussDB(DWS)的表支持行存表、列存表，底层存储方式的选择严格依赖于客户的具体业务场景。一般来说计算型业务查询场景(以关联、聚合操作为主)建议使用列存表；点查询、大批量 UPDATE/DELETE 业务场景适合行存表。

对于每种存储方式还有对应的存储层优化手段，这部分会在后续的调优章节深入介绍。

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，可以大幅度的提升 SQL 语句的性能。

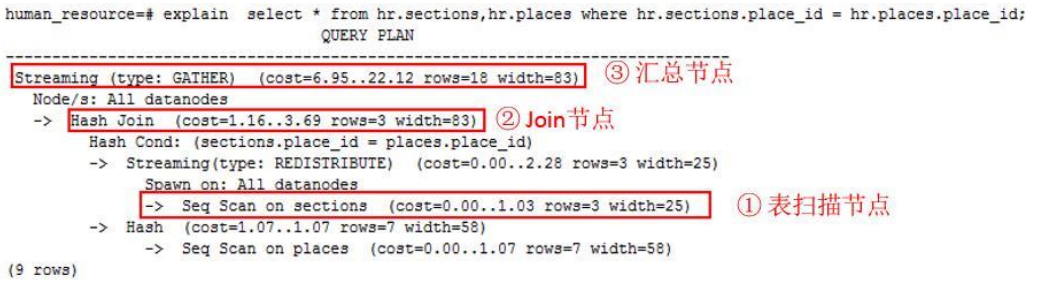
这种调优场景对 DBA 的要求比较高，需要对客户业务有足够的了解，同时也需要扎实的 SQL 语句基本功，后续会介绍几个常见的 SQL 改写场景。

11.2.2 SQL 执行计划概述

SQL 执行计划是一个节点树，显示 GaussDB(DWS)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 11-2 所示。

图11-2 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(DWS)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 11-2 中即为此显示格式。
- pretty：代表使用 GaussDB(DWS)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 11-3。
- 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11-3 pretty 格式执行计划示例

```
postgres=# 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的花费更多的时间。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。

EXPLAIN PERFORMANCE 轻量化执行方式与 EXPLAIN PERFORMANCE 保持一致，在原来的基础上减少了性能分析的时间，执行时间与 SQL 执行时间的差异显著减少。

11.2.3 SQL 执行计划详解

如 11.2.2 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：

```
postgres=# 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)
```

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。

  Vector 前缀的算子是指向量化执行引擎算子，一般出现含有列存表的 Query 中。

  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：CStore Scan on dwcjk

   表扫描算子，用 CStore Scan 的方式扫描表 dwcjk。这一层的作用是把表 dwcjk 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：Vector Hash Aggregate

   聚合算子，作用是把下层计算输送上来的算子做聚合操作(group by)。
3. 第三层：Vector Streaming (type: GATHER)

   Shuffle 算子，此处 GATHER 类型的 Shuffle 算子作用是把数据从 DN 汇聚到 CN。
4. 第四层：Row Adapter

   存储格式转化算子，主要作用是把内存中列式格式数据转为行式数据，以便客户端展示。

需要注意的是最顶层算子为 Data Node Scan 时，需要设置 enable_fast_query_shipping 为 off 才能看到具体的执行计划，如下面这个计划：

```
postgres=# set enable_fast_query_shipping=on;
SET
postgres=# set explain_perf_mode=pretty;
SET
postgres=# explain select cjxh, count(1) from dwcjk group by cjxh;
                                QUERY PLAN
--------------------------------------------------------------------------------
  id |                 operation                  | E-rows | E-width | E-costs
 ----+--------------------------------------------+--------+---------+---------
   1 | ->  Data Node Scan on "__REMOTE_FQS_QUERY__" |      0 |       0 | 0.00
(3 rows)
```

设置 enable_fast_query_shipping 参数之后，执行计划显示如下：

```
postgres=# set enable_fast_query_shipping=off;
SET
postgres=# explain select cjxh, count(1) from dwcjk group by cjxh;
                                QUERY PLAN
--------------------------------------------------------------------------------
  id |              operation                | E-rows | E-width | E-costs
 ----+---------------------------------------+--------+---------+---------
   1 | ->  Row Adapter                       |     20 |      12 | 19.18
   2 |    ->  Vector Streaming (type: GATHER) |     20 |      12 | 19.18
   3 |       ->  Vector Sonic Hash Aggregate  |     20 |      12 | 13.18
   4 |          ->  CStore Scan on dwcjk      |     20 |       4 | 13.01
(6 rows)
```

执行计划中的关键字说明：

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。

     索引扫描可以分为以下几类，他们之间的差异在于索引的排序机制。

     - 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，该方式表的数据行是以索引顺序抓取的，这样就令读取它们的开销更大，但是这里的行少得可怜，因此对行位置的额外排序并不值得。最常见的就是看到这种规划类型只抓取一行，以及那些要求 ORDER BY 条件匹配索引顺序的查询。因为那时候没有多余的排序步骤是必要的以满足 ORDER BY。
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

  归并连接或融合连接，是先将关联表的关联列各自做排序，然后从各自的排序表中抽取数据，到另一个排序表中做匹配。

  因为 Merge join 需要做更多的排序，所以消耗的资源更多，因此通常情况下执行性能差于 Hash Join。 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 Merge Join 的性能优于 Hash Join。

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。

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 t_ddw_f10_op_cust_asset_mon b1
WHERE b1.year_mth < substr('20200722',1 ,6 )
AND b1.year_mth + 1 >= cast(substr('20200722',1 ,6 ) AS int);
```

执行 EXPLAIN PERFORMANCE 输出为：

```
QUERY PLAN
```

```
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 id |                             operation                              |        A-time         |  A-rows  |  E-rows  | E-distinct | Peak Memory  | E-memory | A-width | E-width |  E-costs
 ----+------------------------------------------------------------------------------+-----------------------+----------+----------+------------+--------------+----------+---------+---------+-----------
   1 | ->  Aggregate                                                    | 12026.054 |        1 |        1 |            | 32KB         |          |         |       8 | 593656.42
   2 |    ->  Streaming (type: GATHER)                                  | 12025.975             |        4 |        4 |            | 136KB        |          |         | 8 | 593656.42
   3 |       ->  Aggregate                                               | [11230.413, 12013.476] |        4 |        4 |            | [24KB, 24KB] | 1MB      |         |       8 | 593646.42
   4 |          ->  Partition Iterator                                   | [10245.695, 10925.896] | 16384000 | 32752850 |            | [16KB, 16KB] | 1MB      |         |       0 | 573175.88
   5 |             ->  Partitioned Seq Scan on public.t_ddw_f10_op_cust_asset_mon b1 | [9733.684, 10368.781]  | 16384000 | 32752850 |            | [32KB, 32KB] | 1MB      |         |       0 | 573175.88

                                  SQL Diagnostic Information
 ---------------------------------------------------------------------------------------------------
 Partitioned table unprunable Qual
        table public.t_ddw_f10_op_cust_asset_mon b1:
        left side of expression "((year_mth + 1) >= 202007)" is an expression or occurs type conversion

                   Predicate Information (identified by plan id)
 ----------------------------------------------------------------------------------
   4 --Partition Iterator
        Iterations: 6
   5 --Partitioned Seq Scan on public.t_ddw_f10_op_cust_asset_mon b1
        Filter: ((b1.year_mth < 202007::bigint) AND ((b1.year_mth + 1) >= 202007))
        Rows Removed by Filter: 81920000
        Partitions Selected by Static Prune: 1..6

                Memory Information (identified by plan id)
 ------------------------------------------------------------------------
 Coordinator Query Peak Memory:
        Query Peak Memory: 1MB
 DataNode Query Peak Memory
        dn_6001_6002 Query Peak Memory: 1MB
        dn_6003_6004 Query Peak Memory: 1MB
        dn_6005_6006 Query Peak Memory: 1MB
        dn_6007_6008 Query Peak Memory: 1MB
   1 --Aggregate
        Peak Memory: 32KB, Estimate Memory: 64MB
   2 --Streaming (type: GATHER)
        Peak Memory: 136KB, Estimate Memory: 64MB
   3 --Aggregate
```

```
       dn_6001_6002 Peak Memory: 24KB, Estimate Memory: 1024KB
       dn_6003_6004 Peak Memory: 24KB, Estimate Memory: 1024KB
       dn_6005_6006 Peak Memory: 24KB, Estimate Memory: 1024KB
       dn_6007_6008 Peak Memory: 24KB, Estimate Memory: 1024KB
       dn_6001_6002 Stream Send time: 0.000; Data Serialize time: 0.029
       dn_6003_6004 Stream Send time: 0.000; Data Serialize time: 0.018
       dn_6005_6006 Stream Send time: 0.000; Data Serialize time: 0.015
       dn_6007_6008 Stream Send time: 0.000; Data Serialize time: 0.033
  4 --Partition Iterator
       dn_6001_6002 Peak Memory: 16KB, Estimate Memory: 1024KB
       dn_6003_6004 Peak Memory: 16KB, Estimate Memory: 1024KB
       dn_6005_6006 Peak Memory: 16KB, Estimate Memory: 1024KB
       dn_6007_6008 Peak Memory: 16KB, Estimate Memory: 1024KB
  5 --Partitioned Seq Scan on public.t_ddw_f10_op_cust_asset_mon b1
       dn_6001_6002 Peak Memory: 32KB, Estimate Memory: 1024KB
       dn_6003_6004 Peak Memory: 32KB, Estimate Memory: 1024KB
       dn_6005_6006 Peak Memory: 32KB, Estimate Memory: 1024KB
       dn_6007_6008 Peak Memory: 32KB, Estimate Memory: 1024KB

         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 --Partition Iterator
       Output: year_mth, c
  5 --Partitioned Seq Scan on public.t_ddw_f10_op_cust_asset_mon b1
       Output: year_mth, c
       Distribute Key: c

                      Datanode Information (identified by plan id)
 ------------------------------------------------------------------------------------
----------------
  1 --Aggregate
       (actual time=12026.054..12026.054 rows=1 loops=1) (projection time=0.016)
       (Buffers: 0)
       (CPU: ex c/r=59037, ex row=4, ex cyc=236150, inc cyc=36078149400)
  2 --Streaming (type: GATHER)
       (actual time=11241.158..12025.975 rows=4 loops=1)
       (Buffers: 0)
       (CPU: ex c/r=9019478312, ex row=4, ex cyc=36077913250, inc cyc=36077913250)
  3 --Aggregate
       dn_6001_6002 (actual time=12013.476..12013.476 rows=1 loops=1) (projection
time=290.644)
       dn_6003_6004 (actual time=11230.413..11230.413 rows=1 loops=1) (projection
time=260.508)
       dn 6005 6006 (actual time=11679.178..11679.178 rows=1 loops=1) (projection
time=281.826)
       dn 6007 6008 (actual time=11435.836..11435.836 rows=1 loops=1) (projection
time=253.765)
       dn_6001_6002 (Buffers: shared hit=5 read=117787)
```

```
       dn_6003_6004 (Buffers: shared hit=5 read=109732)
       dn_6005_6006 (Buffers: shared hit=5 read=114434)
       dn_6007_6008 (Buffers: shared hit=5 read=113991)
       dn_6001_6002 (CPU: ex c/r=833, ex row=4341760, ex cyc=3616944880, inc
cyc=36040410726)
       dn_6003_6004 (CPU: ex c/r=832, ex row=3932160, ex cyc=3275098310, inc
cyc=33691198040)
       dn_6005_6006 (CPU: ex c/r=822, ex row=4243456, ex cyc=3492305592, inc
cyc=35037518242)
       dn_6007_6008 (CPU: ex c/r=819, ex row=3866624, ex cyc=3170445826, inc
cyc=34307470566)
  4 --Partition Iterator
       dn_6001_6002 (actual time=8310.416..10925.896 rows=4341760 loops=1)
       dn_6003_6004 (actual time=7834.320..10245.695 rows=3932160 loops=1)
       dn_6005_6006 (actual time=8079.610..10632.024 rows=4243456 loops=1)
       dn_6007_6008 (actual time=8161.853..10485.530 rows=3866624 loops=1)
       dn_6001_6002 (CPU: ex c/r=386, ex row=4341760, ex cyc=1679143266, inc
cyc=32423465846)
       dn_6003_6004 (CPU: ex c/r=394, ex row=3932160, ex cyc=1550699376, inc
cyc=30416099730)
       dn_6005_6006 (CPU: ex c/r=385, ex row=4243456, ex cyc=1637108520, inc
cyc=31545212650)
       dn_6007_6008 (CPU: ex c/r=386, ex row=3866624, ex cyc=1495553894, inc
cyc=31137024740)
  5 --Partitioned Seq Scan on public.t_ddw_f10_op_cust_asset_mon b1
       dn_6001_6002 (actual time=8310.131..10368.781 rows=4341760 loops=6) (filter
time=4967.859)
       dn_6003_6004 (actual time=7834.019..9733.684 rows=3932160 loops=6) (filter
time=4718.095)
       dn_6005_6006 (actual time=8079.320..10086.164 rows=4243456 loops=6) (filter
time=4815.813)
       dn_6007_6008 (actual time=8161.518..9988.215 rows=3866624 loops=6) (filter
time=4785.132)
       dn_6001_6002 (Buffers: shared hit=5 read=117787)
       dn_6003_6004 (Buffers: shared hit=5 read=109732)
       dn_6005_6006 (Buffers: shared hit=5 read=114434)
       dn_6007_6008 (Buffers: shared hit=5 read=113991)
       dn_6001_6002 (CPU: ex c/r=1210, ex row=25395200, ex cyc=30744322580, inc
cyc=30744322580)
       dn_6003_6004 (CPU: ex c/r=1220, ex row=23658496, ex cyc=28865400354, inc
cyc=28865400354)
       dn_6005_6006 (CPU: ex c/r=1212, ex row=24674304, ex cyc=29908104130, inc
cyc=29908104130)
       dn_6007_6008 (CPU: ex c/r=1206, ex row=24576000, ex cyc=29641470846, inc
cyc=29641470846)

                  User Define Profiling
 ---------------------------------------------------------------
 Plan Node id: 2  Track name: coordinator get datanode connection
      cn_5001 (time=6.694 total_calls=1 loops=1)

                     ====== Query Summary =====
 -------------------------------------------------------------------------------
 Datanode executor start time [dn 6007 6008, dn 6005 6006]: [0.873 ms,1.036 ms]
 Datanode executor end time [dn_6005_6006, dn_6007_6008]: [0.569 ms,1.768 ms]
```

```
Remote query poll time: 12017.871 ms, Deserialze time: 0.000 ms
System available mem: 262144KB
Query Max mem: 8388608KB
Query estimated mem: 32768KB
Coordinator executor start time: 0.424 ms
Coordinator executor run time: 12026.201 ms
Coordinator executor end time: 0.302 ms
Planner runtime: 0.658 ms
Query Id: 76561193665298453
Total runtime: 12027.060 ms
(136 rows)
```

图中显示执行信息分为以下 8 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 以及部分 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. SQL Diagnostic Information：

   11.3.4.1 SQL 自诊断信息。优化和执行过程中识别到的性能优化点，当对 DML 语句进行带 VERBOSE 属性的 EXPLAIN（EXPLAIN PERFORMANCE 内置自带 VERBOSE 属性）时，SQL 自诊断信息也会输出，以辅助性能问题定位。
3. 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。
4. 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] ）。其中 sort 算子不会显示具体的下盘文件数，仅在显示排序方法时显示 Disk。
5. 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。
6. DataNode Information (identified by plan id)：

这一部分会将各个算子的执行时间（若包含过滤及投影也会显示对应的执行时间）、CPU、buffer 的使用情况全部打印出来。

7. 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。

8. ====== 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，他们偏差越大，我们越可以认为优化器生成的计划的越不可信，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。一般来说，两个相邻的算子，上层算子的执行时间包含下层算子的执行时间，但如果上层算子为 stream 算子，由于各线程不存在驱动关系，上层算子执行时间可能小于下层算子的执行时间，即不存在包含关系。
- 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。

11.2.4 查询最耗性能的 SQL

系统中有些 SQL 语句运行了很长时间还没有结束，这些语句会消耗很多的系统性能，请根据本章内容查询长时间运行的 SQL 语句。

操作步骤

步骤 1 查询系统中长时间运行的查询语句。

```
SELECT current_timestamp - query_start AS runtime, datname, usename, query FROM
pg_stat_activity where state != 'idle' ORDER BY 1 desc;
```

查询后会按执行时间从长到短顺序返回查询语句列表，第一条结果就是当前系统中执行时间最长的查询语句。返回结果中包含了系统调用的 SQL 语句和用户执行 SQL 语句，请根据实际找到用户执行时间长的语句。

若当前系统较为繁忙，可以通过限制 current_timestamp - query_start 大于某一阈值来查看执行时间超过此阈值的查询语句。

```
SELECT query FROM pg_stat_activity WHERE current_timestamp - query_start > interval
'1 days';
```

步骤 2 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 3 查看正在运行的查询语句。

以查看视图 pg_stat_activity 为例：

```
SELECT datname, usename, state FROM pg_stat_activity;
 datname  | usename | state  |
----------+---------+--------+
 postgres |   omm   | idle   |
 postgres |   omm   | active |
(2 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 FROM pg_stat_activity WHERE state != 'idle';
```

步骤 4 分析长时间运行的查询语句状态。

- 若查询语句处于正常状态，则等待其执行完毕。
- 若查询语句阻塞，则通过如下命令查看当前处于阻塞状态的查询语句：

  ```
  SELECT datname, usename, state, query FROM pg_stat_activity WHERE waiting = true;
  ```

  查询结果中包含了当前被阻塞的查询语句，该查询语句所请求的锁资源可能被其他会话持有，正在等待持有会话释放锁资源。

说明

只有当查询阻塞在系统内部锁资源时，waiting 字段才显示为 true。尽管等待锁资源是数据库系统最常见的阻塞行为，但是在某些场景下查询也会阻塞在等待其他系统资源上，例如写文件、定时器等。但是这种情况的查询阻塞，不会在视图 pg_stat_activity 中体现。

----结束

11.2.5 分析作业是否被阻塞

数据库系统运行时，在某些业务场景下查询语句会被阻塞，导致语句运行时间过长，可以强制结束有问题的会话。

操作步骤

步骤 1 查看阻塞的查询语句及阻塞查询的表、模式信息。

```
SELECT w.query as waiting_query,
w.pid as w_pid,
w.usename as w_user,
l.query as locking_query,
l.pid as l_pid,
l.usename as l_user,
```

```
t.schemaname || '.' || t.relname as tablename
from pg_stat_activity w join pg_locks l1 on w.pid = l1.pid
and not l1.granted join pg_locks l2 on l1.relation = l2.relation
and l2.granted join pg_stat_activity l on l2.pid = l.pid join pg_stat_user_tables t
on l1.relation = t.relid
where w.waiting;
```

该查询返回线程 ID、用户信息、查询状态，以及导致阻塞的表、模式信息。

步骤 2 使用如下命令结束相应的会话。其中，139834762094352 为线程 ID。

```
SELECT PG_TERMINATE_BACKEND(139834762094352);
```

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，表示用户正在尝试结束当前会话，此时仅会重连会话，而不是结束会话。

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

说明

gsql 客户端使用 PG_TERMINATE_BACKEND 函数终止本会话后台线程时，客户端不会退出而是自动重连。

----结束

11.3 改进查询

11.3.1 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 11.3.2 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。

步骤 3 11.3.3 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 11.3.4 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。11.3.5 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。

步骤 6 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。详细请参见 11.3.7 使用 Plan Hint 进行调优。

步骤 7 为了保证数据库性能的持续优质，建议 11.3.8 例行维护表和 11.3.9 例行重建索引。

步骤 8 （可选）GaussDB(DWS)支持在资源富足的情况下，通过算子并行来提升性能。详细请参见 11.3.10.5 SMP 手动调优建议。

----结束

11.3.2 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。

对于表中多个列有相关性且查询中有同时基于这些列的条件或分组操作的情况，可尝试收集多列统计信息，以便查询优化器可以更准确地估算行数，并生成更有效的执行计划。

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
```

使用以下命令进行多列统计信息相关操作。

```
ANALYZE tablename ((column_1, column_2));                    --收集 tablename 表的
column_1、column_2 列的多列统计信息
```

```
ALTER TABLE tablename ADD STATISTICS ((column_1, column_2));    --添加 tablename 表的
column_1、column_2 列的多列统计信息声明
ANALYZE tablename;                                              --收集单列统计信息，并收集已声明
的多列统计信息

ALTER TABLE tablename DELETE STATISTICS ((column_1, column_2)); --删除 tablename 表的
column_1、column_2 列的多列统计信息或其声明
```

须知

在使用 ALTER TABLE tablename ADD STATISTICS 语句添加了多列统计信息声明后，系统并不会立刻收集多列统计信息，而是在下次对该表或全库进行 ANALYZE 时，进行多列统计信息的收集。

如果想直接收集多列统计信息，请使用 ANALYZE 命令进行收集。

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。

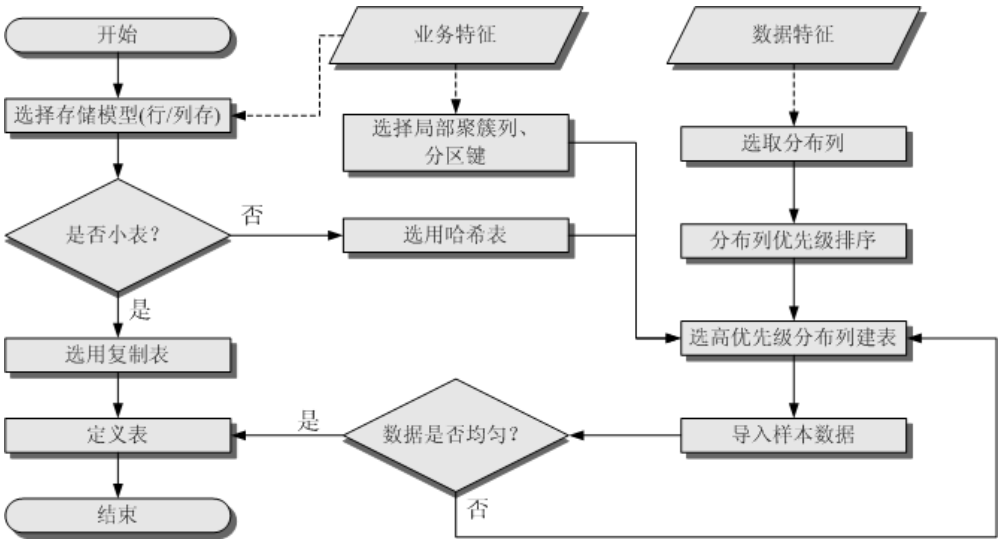
11.3.3 审视和修改表定义

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 IO**。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 11-4 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11-4 表定义流程



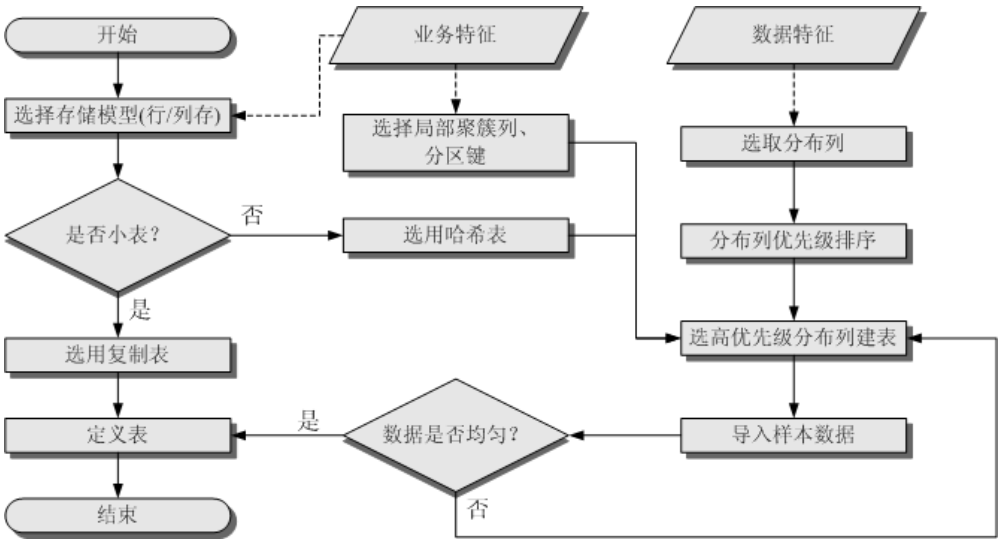
11.3.3.1 审视和修改表定义概述

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 IO**。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 11-5 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11-5 表定义流程



11.3.3.2 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。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。
列存	统计分析类查询 (group , join 多的场景)。

11.3.3.3 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 上都有此表的全量数据，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 IO 资源，大大提升表的读/写速度。一般情况下大表定义为 Hash 表。

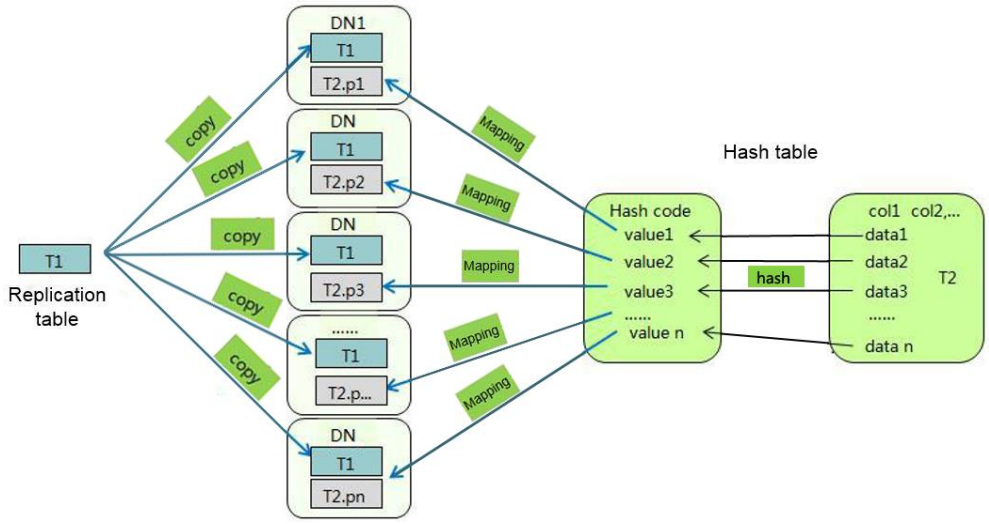
轮询（Roundrobin）表将表的每一行被轮番地发送给各个 DN，因此数据会被均匀地分布在各个 DN 中。这种存储方式的优点是保证了数据不会发生倾斜，从而提高了集群

的空间利用率；缺点是无法像 Hash 表一样进行 DN 本地化优化，查询性能通常不如 Hash 表。一般在大表无法找到合适的分布列时，定义为 Roundrobin 表，若大表能够找到合适的分布列，优先选择性能更好的 Hash 分布。

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。
Roundrobin	表的每一行被轮番地发送给各个 DN，因此数据会被均匀地分布在各个 DN 中。	数据量较大的事实表，且使用 Hash 分布时找不到合适的分布列。

如图 11-6 所示，复制表如图中的表 T1，哈希表如图中的表 T2。

图11-6 复制表和哈希表



11.3.3.4 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**。

4. 一般不建议用户新增一列专门用作分布列，尤其不建议用户新增一列，然后用 SEQUENCE 的值来填充做为分布列。因为 SEQUENCE 可能会带来性能瓶颈和不必要的维护成本。

GaussDB(DWS)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。

11.3.3.5 使用局部聚簇

局部聚簇（Partial Cluster Key）是列存下的一种技术。这种技术可以通过 min/max 稀疏索引较快的实现基表扫描的 filter 过滤。Partial Cluster Key 可以指定多列，但是一般不建议超过 2 列。Partial Cluster Key 的选取原则：

1. 受基表中的简单表达式约束。这种约束一般形如 col op const，其中 col 为列名，op 为操作符 =、>、>=、<=、<，const 为常量值。
2. 尽量采用选择度比较高(过滤掉更多数据)的简单表达式中的列。
3. 尽量把选择度比较低的约束 col 放在 Partial Cluster Key 中的前面。
4. 尽量把枚举类型的列放在 Partial Cluster Key 中的前面。

11.3.3.6 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(DWS)支持的分区表为范围分区表和列表分区表。

11.3.3.7 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。比如某客户场景中对列存表进行点查

询，filter 条件在一个 numeric 列上，执行时间为 10+s；修改 numeric 为 int 类型之后，执行时间缩短为 1.8s 左右。

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 11.2.3 SQL 执行计划详解文件的大小，提升 IO 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。

11.3.4 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。

11.3.4.1 SQL 自诊断

用户在执行 INSERT/UPDATE/DELETE/SELECT/MERGE INTO 或者 CREATE TABLE AS 语句时，可能会遇到性能问题。产品内置集成了性能自动诊断功能，并把相关的诊断信息保存到 10.5 实时 TopSQL 中，当配置参数 enable_resource_track 为 on 的时候，这些诊断信息会转存到 10.6 历史 TopSQL 中。通过查询 14.3.80 GS_WLM_SESSION_STATISTICS，14.3.79 GS_WLM_SESSION_HISTORY，14.2.5 GS_WLM_SESSION_INFO 视图的 warning 字段可以获得对应的性能诊断信息，为性能调优提供参考。

SQL 自诊断的告警类型与 resource_track_level 的设置有关系。当 resource_track_level 设置为 query 时，可以诊断多列/单列统计信息未收集、分区未剪枝告警和 SQL 不下推的告警等非执行态的诊断信息；resource_track_level 设置为 perf 或 operator 时，可以诊断所有的告警场景。

SQL 自诊断的诊断范围与 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。

执行 EXPLAIN PERFORMANCE 或者 EXPLAIN VERBOSE 的时候，除缺乏多列统计信息之外的 SQL 自诊断信息也会输出，具体请参考 11.2.3 SQL 执行计划详解。

告警场景

目前支持对以下 9 种导致性能问题的场景上报告警。

- 多列/单列统计信息未收集

  如果存在单列或者多列统计信息未收集，则上报相关告警。对于这种告警，建议的优化方案是对相关表进行 ANALYZE，可参考 11.3.2 更新统计信息和 11.3.4.4 统计信息调优。

  需要特别注意的是，如果查询语句中的 OBS 外表和 HDFS 外表未收集统计信息也会上报统计信息未收集的告警，因为 OBS 外表和 HDFS 外表的 analyze 的性能比较差，一般不建议对其进行 ANALYZE，但是建议使用 ALTER FOREIGN TABLE 的语法修改外表的 totalrows 属性，从而修正外表的行数估算。

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)
```

多列统计信息未收集:

```
Statistic Not Collect
      schema_test.t3((c1,c2))
```

单列和多列统计信息未收集：

```
Statistic Not Collect
      schema_test.t4(c1)
      schema_test.t5((c1,c2))
```

- 分区不剪枝（该功能 8.1.2 及以上版本支持）

  分区表查询时，往往会期望通过分区键上的约束条件进行分区剪枝，从而提升分区表查询性能，但有时候会因为约束条件书写不当，导致分区表没有剪枝，出现查询性能问题，具体请参见 11.4.14 案例：改写 SQL 排除剪枝干扰。

- 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。调优方法可参考案例 11.3.4.2 语句下推调优。

  导致不下推的因素主要有以下两点：

  - 函数导致的不下推

    诊断信息中会给出具体的函数名称。函数下推是由函数的 shippable 属性决定，具体可参见 CREATE FUNCTION 语法。

  - 语法导致的不下推

    诊断信息中会提示导致不下推的具体语法，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
        "enable_stream_operator" is off

  SQL is not plan-shipping
        "Distinct On" can not be shipped

  SQL is not plan-shipping
        "v_test_unshipping_log" is VIEW that will be treated as Record type
  can't be shipped
  ```

- HashJoin 中大表做内表

  如果在表连接过程中使用了 Hashjoin(可通过 14.3.79 GS_WLM_SESSION_HISTORY 的 query_plan 字段查看)，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。针对这种场景，需要调整 HashJoin 内外表顺序，具体调优方法参考 11.3.7.2 Join 顺序的 Hint。

告警信息示例：

```
Execute diagnostic information
      PlanNode[7] Large Table is INNER in HashJoin “Vector Hash Aggregate”
```

其中，7 为 query_plan 字段文本中编号为 7 的算子。

- 大表等值连接使用 Nestloop

  如果在表连接过程中使用了 nestloop(可通过 14.3.79 GS_WLM_SESSION_HISTORY 的 query_plan 字段查看)，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。针对这种场景，需要调整表关联方式，禁止当前内外表之间使用 NestLoop 的关联方式，具体调优方法参考 11.3.7.3 Join 方式的 Hint。

  告警信息示例：

  ```
  Execute diagnostic information
        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。针对这种场景，需要禁止 BroadCast 下层算子的做 Broadcast 动作，具体调优方法参考 11.3.7.5 Stream 方式的 Hint。

  告警信息示例：

  ```
  Execute diagnostic information
        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。该告警一般分为存储层倾斜和计算层倾斜，具体调优方法参考 11.3.4.6 数据倾斜调优。

  告警信息示例：

  ```
  Execute diagnostic information
       PlanNode[6] DataSkew:"Seq Scan", min_dn_tuples:0, max_dn_tuples:524288
  ```

- 索引不合理

  在基表扫描时，满足下述条件则上报相关告警：

  - 对于行存表：
    - 使用索引扫描，输出行数/扫描行数>1/1000 且输出行数>1 万行
    - 使用顺序扫描，输出行数/扫描行数<1/1000 且输出行数<=1 万行、扫描行数>1 万行
  - 对于列存表：
    - 使用索引扫描，输出行数/扫描行数>1/10000 且输出行数>100
    - 使用顺序扫描，输出行数/扫描行数<1/10000 且输出行数<=100、扫描行数>1 万行

  调优方法可参考 11.3.4.5 算子级调优，也可参考案例 11.4.2 案例：建立合适的索引和 11.4.16 案例：使用 partial cluster key。

  告警信息示例：

  ```
  Execute diagnostic information
        PlanNode[4] Indexscan is not properly used:"Index Only Scan",
  ```

```
output:524288, filtered:0, rate:1.00000
      PlanNode[5] Indexscan is ought to be used:"Seq Scan", output:1,
filtered:524288, rate:0.00000
```

需要注意的是，这个诊断结果只是针对当前 SQL 的建议，是否创建索引要结合整体业务综合分析，对于高频的过滤条件才建议创建索引。

- 估算不准

  如果优化器的估算行数和实际行数中的较大值平均每 DN 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。针对这种场景，可以参照 11.3.7.4 行数的 Hint 修正行数估算，让优化器在正确的行数基础上重新规划执行计划。

  告警信息示例：

```
Execute diagnostic information
      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。

11.3.4.2 语句下推调优

语句下推介绍

目前，GaussDB(DWS)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。

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：

```
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。

```
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);
CREATE TABLE test_stream(a int,b float); --float 不支持重分布
CREATE TABLE sal_emp ( c1 integer[] ) DISTRIBUTE BY replication;
```

- 不支持 returning 语句下推

```
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)
```

- count(distinct expr)中的字段不支持重分布，则不支持下推

```
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

```
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)
```

- Fulljoin 的 join 列如果不支持重分布，则不支持下推

```
explain select * from test_stream t1 full join test_stream t2 on t1.a=t2.b;
                                    QUERY PLAN
------------------------------------------------------------------ Hash Full
Join  (cost=0.38..0.82 rows=30 width=24)
   Hash Cond: ((t1.a)::double precision = t2.b)
   ->  Data Node Scan on test_stream "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00
rows=30 width=12)
       Node/s: All datanodes
```

```
  ->  Hash  (cost=0.00..0.00 rows=30 width=12)
        ->  Data Node Scan on test_stream "_REMOTE_TABLE_QUERY_"
(cost=0.00..0.00 rows=30 width=12)
              Node/s: All datanodes
(7 rows)
```

- 不支持数组表达式下推

```
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、HDFS 表的查询场景	LOG: SQL can't be shipped, reason: RecursiveUnion contains HDFS Table or ForeignScan is not shippable（LOG 为 CN 日志中打印的不下推原因，下同） 外表、HDFS 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	`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;`
		LOG: SQL can't be shipped, reason: With-Recursive does not contain "ALL" to bind recursive & none-recursive branches
		UNION 不带 ALL，需要去重。

4	`WITH RECURSIVE x(id) AS`	LOG: SQL can't be shipped, reason:

序号	场景	不下推原因
	(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;	With-Recursive contains system table is not shippable 基表中有系统表。
5	WITH RECURSIVE t(n) AS (VALUES (1) UNION ALL SELECT n+1 FROM t WHERE n < 100) SELECT sum(n) FROM t;	LOG: SQL can't be shipped, reason: With-Recursive contains only values rte is not shippable 基表扫描只有 VALUES 子句，仅在 CN 上即可完成执行。
6	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;	LOG: SQL can't be shipped, reason: With-Recursive recursive term correlated only is not shippable 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。
7	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	LOG: SQL can't be shipped, reason: With-Recursive contains conflict distribution in none-recursive(Replicate) recursive(Hash) 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。

序号	场景	不下推原因
	) SELECT * FROM t_result t;	
8	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;	LOG: SQL can't be shipped, reason: Recursive CTE references recursive CTE "cte" 多层 Recursive 嵌套，即 recursive 的递归部分又嵌套另一个 recursive 查询。

不支持下推的函数

首先介绍函数的易变性。在 GaussDB(DWS)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 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，比如我们要写一个函数，获取商品的打折情况，需要一个计算折扣的函数，我们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 on "__REMOTE_FQS_QUERY__"  (cost=0.00..0.00 rows=0 width=0)
   Output: (func_percent_1(store_sales.ss_sales_price, store_sales.ss_list_price))
   Node/s: All datanodes
   Remote query: SELECT public.func_percent_1(ss_sales_price, ss_list_price) AS func_percent_1 FROM public.store_sales
(4 rows)
```

可见函数 func_percent_1 被下推到 DN 执行，提升了执行效率（TPCDS 1000X，3CN18DN，查询效率提升 100 倍以上）。

实例分析 2：使排序下推

请参考 11.4.4 案例：使排序下推。

11.3.4.3 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(DWS)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(DWS)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 IN,<,>,=操作符
  - all_sublink：对应 op ALL(select…)语句，其中 OP 可以是 IN,<,>,=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 OLAP、HTAP 场景中常用的 sublink 为 exist_sublink、any_sublink，在 GaussDB(DWS)的优化引擎中对其应用场景做了优化（子链接提升），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。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：

```
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 AND-ed 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：

```
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(DWS)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：

```
select t1.c1,t1.c2
from t1
where t1.c1 in (
    select c2
    from t2
    where t2.c1 = t1.c1
);
```

```
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
(11 rows)
```

- **目前 GaussDB(DWS)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。

```
select t1.c1,t1.c2
from t1
where t1.c1 in (
    select c2
    from t2
    where t2.c1 = 1
);
```

```
QUERY PLAN
-------------------------------------------------------
Streaming (type: GATHER)
  Node/s: All datanodes
  ->  Nested Loop Semi Join
        Join Filter: (t1.c1 = t2.c2)
        ->  Seq Scan on t1
        ->  Materialize
              ->  Streaming(type: REDISTRIBUTE)
                    Spawn on: datanode1
                    ->  Seq Scan on t2
                          Filter: (c1 = 1)
(10 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。

```
select t1.c1,t1.c2
from t1
where exists (
    select c2
    from t2
    where t2.c1 = t1.c1
);
```

```
QUERY PLAN
-----------------------------------
Streaming (type: GATHER)
  Node/s: All datanodes
  ->  Hash Semi Join
        Hash Cond: (t1.c1 = t2.c1)
        ->  Seq Scan on t1
        ->  Hash
              ->  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 OR-ed 连接的 EXIST 相关子查询 OR 字句的提升过程：

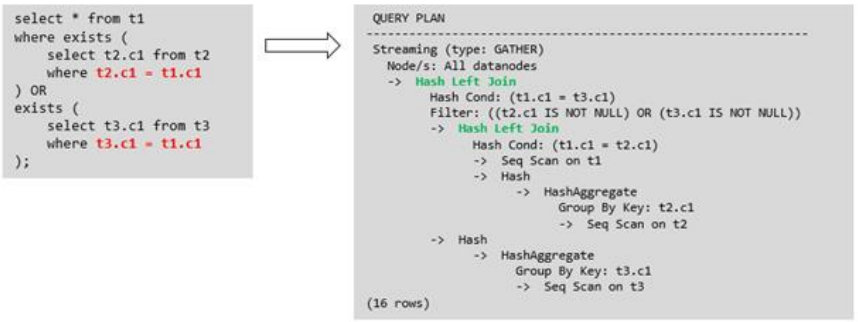
i. 提取 where 条件中，or 子句中的 opExpr。为：t1.a = (select avg(a) from t3 where t1.b = t3.b)

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 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)
left join (select t4.c from t4 group by t4.c) where t3.b is not null
or t4.c is not null;
```



- **目前 GaussDB(DWS)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。

  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：

```
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：

```
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 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。

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 t2.c2 from t2
)
select ssq.c2, t1.c2
from t1 left join ssq on t1.c1 = ssq.c2
where t1.c2 > 10;
```

改写后的执行计划为：

```
              QUERY PLAN
-----------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Hash Right Join
         Hash Cond: (t2.c2 = t1.c1)
         ->  Streaming(type: REDISTRIBUTE)
               Spawn on: All datanodes
               ->  Seq Scan on t2
```

```
       ->  Hash
            ->  Seq Scan on t1
                 Filter: (c2 > 10)
(10 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(DWS)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_node_id + t1.ctid 进行 rowid 关联，但是 xc_node_id 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 replicate 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，可以将 sub_table 修改为一个 relication 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。

```
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：

```
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。

11.3.4.4 统计信息调优

统计信息调优介绍

GaussDB(DWS)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 NULL3
, 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
) with (orientation = column, COMPRESSION = MIDDLE) 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
)with (orientation = column, COMPRESSION = MIDDLE) 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,l_commitdate,l_orderkey, l_suppkey),
   public.orders(o_orderstatus,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是当前执行的 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,
   l_commitdate,l_orderkey,
   .l_suppkey), public.orders(o_orderstatus,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 可以解决由于为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 11.4.5 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 11-7 所示。该查询实际耗时约 12 秒。

图11-7 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。

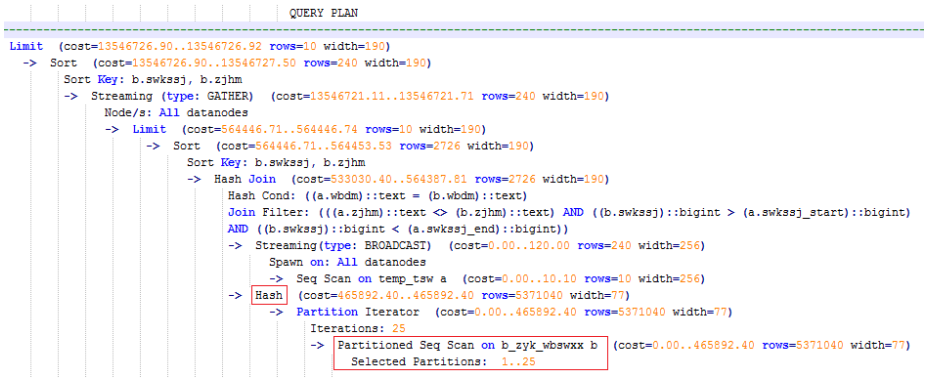
2. 考虑使用 unlogged table 保存目标人的上网信息，且在插入时处理上网开始时间和终止时间，以避免后续进行时间加减。

```
//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 11-8 所示。

图11-8 应用 unlogged table 案例（二）



3. 分析上述执行计划，发现执行了 Hash Join，对大表 b_zyk_wbswxx 建立了 Hash Table。由于该表数据量大，创建过程耗时较长。

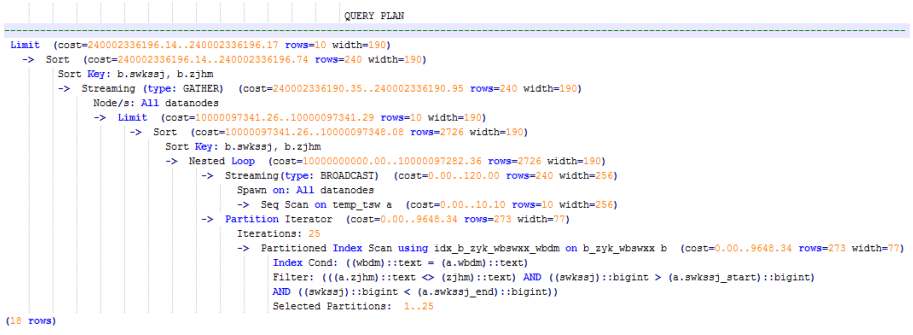
   由于 temp_tsw 中仅包含几百条记录，且 temp_tsw 和 b_zyk_wbswxx 均通过 wbdm（网吧代码）执行等值连接。因此，如果 Join 方式改为 Nest Loop Join，则扫描节点可以实现 Index Scan，性能预计将会提升。

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 11-9 所示。查询耗时约 3 秒。

图11-9 应用 unlogged table 案例（三）



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

通过 analyze 收集全局统计信息，通常会改善查询性能。

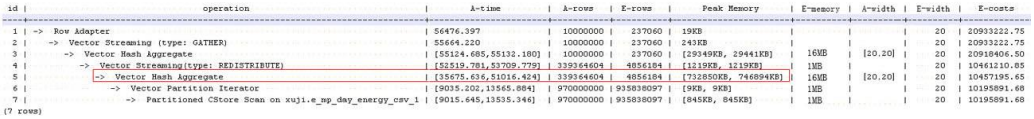
如果遇到性能问题：可以使用 plan hint 来调整到之前的查询计划，详情请参见 11.3.7 使用 Plan Hint 进行调优。

11.3.4.5 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	[9035.202,13565.884]	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。

```
 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] |

 Predicate Information (identified by plan id)
-----------------------------------------------
   2 --Seq Scan on store_sales
         Filter: (ss_sold_date_sk = 2450944)
         Rows Removed by Filter: 4968936
 create index idx on store_sales_row(ss_sold_date_sk);
CREATE INDEX
 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] |
```

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。

```
postgres=#  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)


postgres=#  set enable_nestloop=off;
SET
postgres=#  set enable_mergejoin=off;
SET
postgres=# 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。

```
postgres=#  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)


postgres=#  set enable_sort=off;
SET
postgres=#  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)
```

11.3.4.6 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(DWS)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(DWS)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：

```
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：

```
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，改为 inv_item_sk 作为分布列，则倾斜情况分布如下：

```
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 14.3.175 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。

```
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]

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，GaussDB(DWS)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 11.3.7.8 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_REDISTRIBUTE_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]

  5 --Vector 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。

如果执行计划中 Stream 算子类型显示为 HYBRID，则表示对不同的倾斜数据所应用的 stream 方式不同，例如以下计划：

```
EXPLAIN (nodes OFF, costs OFF) SELECT COUNT(*) FROM skew scol s, skew scol1 s1
WHERE s.b = s1.c;
QUERY PLAN
------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
----
id |                                                                  operation
----+-------------------------------------------------------------------------------
------------------------------------------------------------------------------------
--
1 | ->  Aggregate
2 |    ->  Streaming (type: GATHER)
3 |       ->  Aggregate
4 |          ->  Hash Join (5,7)
5 |             ->  Streaming(type: HYBRID)
6 |                ->  Seq Scan on skew_scol s
7 |             ->  Hash
8 |                ->  Streaming(type: HYBRID)
9 |                   ->  Seq Scan on skew_scol1 s1

Predicate Information (identified by plan id)
------------------------------------------------------------------------------------
------------------------------------------------------------
4 --Hash Join (5,7)
Hash Cond: (s.b = s1.c)
Skew Join Optimized by Statistic
5 --Streaming(type: HYBRID)
Skew Filter: (b = 1)
Skew Filter: (b = 0)
8 --Streaming(type: HYBRID)
Skew Filter: (c = 0)
Skew Filter: (c = 1)
```

数据 1 在 skew_scol 表上有倾斜，对倾斜数据做 roundrobin，非倾斜数据做 redistribute。

数据 0 在 skew_scol 表上是非倾斜侧，对倾斜数据做 broadcast，非倾斜数据做 redistribute。

由此可以看到两个 stream 类型分别是 PART REDISTRIBUTE PART ROUNDROBIN 和 PART REDISTRIBUTE PART BROADCAST，这里标记 stream 类型为 HYBRID 类型。

- 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 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，

从全局来看，每个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

   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]

 Predicate Information (identified by plan id)
-----------------------------------------------
   3 --HashAggregate
       Skew Agg Optimized by Statistic

   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。

11.3.5 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11-10 not exists 执行计划

```
 id |                operation
----+-----------------------------------------
  1 | ->  Streaming (type: GATHER)
  2 |     ->  Hash Right Anti Join (3, 5)
  3 |        ->  Streaming(type: REDISTRIBUTE)
  4 |           ->  Seq Scan on t2
  5 |        ->  Hash
  6 |           ->  Seq Scan on t1

Predicate Information (identified by plan id)
---------------------------------------------
  2 --Hash Right Anti Join (3, 5)
        Hash Cond: (t2.d2 = t1.c1)
(13 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(DWS)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。

11.3.6 SQL 调优关键参数调整

本节将介绍影响 GaussDB(DWS) SQL 调优性能的关键 CN 配置参数，配置方法参见 16.2 设置 GUC 参数。

表11-2 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;` 此参数默认设置为“on”，但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 性能较差，实际调优中可以选择关闭。
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

参数/参考值	描述
	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。
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特定组合的重写规则。

11.3.7 使用 Plan Hint 进行调优

11.3.7.1 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，指定配置参数的值等多个手段来进行执行计划的调优，以提升查询的性能。

功能描述

Plan Hint 仅支持在 SELECT 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。

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 Agg 重分布列 Hint。仅 8.1.3.100 及以上集群版本支持。
- 指定配置参数值的 Hint，仅支持部分配置参数，详见 11.3.7.9 配置参数的 hint。

注意事项

- 不支持 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
;
```

```
 id |                                operation                                |   E-rows   | E-memory | E-width |  E-costs
----+-------------------------------------------------------------------------+------------+----------+---------+------------
  1 | ->  Row Adapter                                                         |          6 |          |     273 | 3401632.49
  2 |    ->  Vector Streaming (type: GATHER)                                  |          6 |          |     273 | 3401632.49
  3 |       ->  Vector Hash Aggregate                                         |          6 | 16MB     |     273 | 3401630.82
  4 |          ->  Vector Streaming(type: REDISTRIBUTE)                       |          6 | 1MB      |     169 | 3401630.78
  5 |             ->  Vector Hash Join (6,21)                                 |          6 | 16MB     |     169 | 3401630.42
  6 |                ->  Vector Hash Join (7,20)                              |          7 | 43MB     |     173 | 3400343.15
  7 |                   ->  Vector Streaming(type: REDISTRIBUTE)              |          7 | 1MB      |     123 | 3395775.64
  8 |                      ->  Vector Hash Join (9,19)                        |          7 | 27MB     |     123 | 3395775.48
  9 |                         ->  Vector Streaming(type: REDISTRIBUTE)        |          7 | 1MB      |     123 | 3386294.72
 10 |                            ->  Vector Hash Join (11,18)                 |          7 | 16MB     |     123 | 3386294.56
 11 |                               ->  Vector Hash Join (12,14)              |          7 | 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 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)
```

11.3.7.2 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 join，t2 做内表；然后再和 t1 join，t2, t3 的 join 表做内表；然后再依次跟 t4, t5 做 join，t4, 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 11.3.7.10 Hint 的错误、冲突及告警的说明。

11.3.7.3 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)
```

11.3.7.4 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 11.3.7.3 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）。

11.3.7.5 Stream 方式的 Hint

功能描述

指明 stream 使用的方法，可以为 broadcast 和 redistribute 以及指定 AGG 重分布的分布键。

说明

指定 Agg 重分布列 Hint，仅 8.1.3.100 及以上集群版本支持。

语法格式

```
[no] broadcast | redistribute(table_list) | redistribute ((*) (columns))
```

参数说明

- **no** 表示 hint 的 stream 方式不使用，当指定 AGG 分布列的 hint 时指定 no 关键字无效。
- **table_list** 为进行 stream 操作的单表或多表 join 结果集，见参数说明。
- 当指定分布列的 hint 时，*为固定写法，不支持指定表名。
- **columns** 指定 group by 子句中的一个或者多个列，没有 group by 子句也可以指定 distinct 子句中的列。

说明

- 指定的分布列，需要用 group by 或 distinct 中的列序号来表示，不可以指定列名。
- 对于多层的查询，可以在每层指定对应层的分布列 hint，只在当前层生效。
- 指定了分布列，如果优化器估算后发现不需要重分布，则指定的分布列无效。

建议

- 通常优化器会根据统计信息选择一组不倾斜的分布键进行数据重分布。当默认选择的分布键有倾斜时，可以手动指定重分布的列，避免数据倾斜。
- 在选择分布键的时候，通常要根据数据分布特征选取一组 distinct 值比较高的列做为分布列，这样可以保证重分布后，数据均匀的分布到各个 DN。
- 在编写好 hint 后，可以通过 explain verbose+SQL 打印执行计划，查看指定的分布键是否有效，如果指定的分布键无效会有 warning 提示。

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)
```

- 指定 Agg 重分布的分布列。

```
explain (verbose on, costs off, nodes off)
select /*+ redistribute ((*) (2 3)) */ a1, b1, c1, count(c1)  from t1 group by
a1, b1, c1 having  count(c1) > 10 and sum(d1) > 100
```

通过下面的示例可以看到指定的 group by 列的后两列做为分布键。

```
                              QUERY PLAN
-------------------------------------------------------------------
  id |                 operation
 ----+-------------------------------------------
   1 | ->  Streaming (type: GATHER)
   2 |     ->  HashAggregate
   3 |        ->  Streaming(type: REDISTRIBUTE)
   4 |           ->  Seq Scan on public.t1

        Predicate Information (identified by plan id)
 -------------------------------------------------------------
   2 --HashAggregate
         Filter: ((count(t1.c1) > 10) AND (sum(t1.d1) > 100))

 Targetlist Information (identified by plan id)
 ----------------------------------------------
   1 --Streaming (type: GATHER)
         Output: a1, b1, c1, (count(c1))
   2 --HashAggregate
         Output: a1, b1, c1, count(c1)
         Group By Key: t1.a1, t1.b1, t1.c1
   3 --Streaming(type: REDISTRIBUTE)
         Output: a1, b1, c1, d1
         Distribute Key: b1, c1
   4 --Seq Scan on public.t1
         Output: a1, b1, c1, d1

    ====== Query Summary =====
 ---------------------------------
 System available mem: 24862720KB
 Query Max mem: 24862720KB
 Query estimated mem: 3138KB
(30 rows)
```

- 当语句中不包含 group by 子句时，指定 distinct 列作为重分布列。

```
explain (verbose on, costs off, nodes off)
select /*+ redistribute ((*) (3 1)) */ distinct a1, b1, c1 from t1;
```

```
                    QUERY PLAN
-------------------------------------------------
  id |                 operation
 ----+-----------------------------------------
   1 | ->  Streaming (type: GATHER)
   2 |     ->  HashAggregate
   3 |        ->  Streaming(type: REDISTRIBUTE)
   4 |           ->  Seq Scan on public.t1

 Targetlist Information (identified by plan id)
 -------------------------------------------------
   1 --Streaming (type: GATHER)
         Output: a1, b1, c1
   2 --HashAggregate
         Output: a1, b1, c1
         Group By Key: t1.a1, t1.b1, t1.c1
   3 --Streaming(type: REDISTRIBUTE)
         Output: a1, b1, c1
         Distribute Key: c1, a1
   4 --Seq Scan on public.t1
         Output: a1, b1, c1

    ====== Query Summary =====
 ---------------------------------
 System available mem: 24862720KB
 Query Max mem: 24862720KB
 Query estimated mem: 3136KB
(25 rows)
```

11.3.7.6 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 支持在行列存表、hdfs 内外表、obs 表、子查询表上指定。对于 hdfs 内表，由主表和 delta 表组成，delta 表对用户不可见，故 hint 仅作用在主表上。

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)
```

11.3.7.7 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)
```

11.3.7.8 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;
```

11.3.7.9 配置参数的 hint

功能描述

指明计划生成时配置参数的值，又称作 guc hint。

语法格式

```
set [global](guc_name guc_value)
```

参数说明

- **global** 表示 hint 设置的配置参数在语句级别生效，不加 global 表示 hint 设置的配置参数在子查询级别生效，即仅在 hint 所在的子查询中生效，在该语句的其它子查询中不生效。

- **guc_name** 表示 hint 指定的配置参数的名称。
- **guc_value** 表示 hint 指定的配置参数的值。

说明

- 如果 hint 设置的配置参数在语句级别生效，则该 hint 必须写在顶层查询中，而不能写在子查询中。对于 UNION、INTERSECT、EXCEPT 和 MINUS 语句，可以将在语句级别的 guc hint 写在参与集合运算的任意一个 SELECT 子句上，该 guc hint 设置的配置参数会在参与集合运算的每个 SELECT 子句上生效。
- 子查询提升时，该子查询上的所有 guc hint 会被丢弃。
- 如果一个配置参数既被语句级别的 guc hint 设置，又被子查询级别的 guc hint 设置，则子查询级别的 guc hint 在对应的子查询中生效，语句级别的 guc hint 在语句的其它子查询中生效。

guc hint 当前仅支持部分配置参数，并且有些配置参数不支持在子查询级别设置，只能在语句级别设置，以下为支持的参数列表：

表11-3 guc hint 支持的配置参数

配置参数名	是否支持在子查询级别设置
agg_redistribute_enhancement	是
best_agg_plan	是
cost_model_version	否
cost_param	否
enable_bitmapscan	是
enable_broadcast	是
enable_extrapolation_stats	是
enable_fast_query_shipping	否
enable_force_vector_engine	否
enable_hashagg	是
enable_hashjoin	是
enable_index_nestloop	是
enable_indexscan	是
enable_join_pseudoconst	是
enable_nestloop	是
enable_nodegroup_debug	否
enable_partition_dynamic_pruning	是

配置参数名	是否支持在子查询级别设置
enable_sort	是
enable_vector_engine	否
expected_computing_nodegroup	否
force_bitmapand	是
from_collapse_limit	是
join_collapse_limit	是
join_num_distinct	是
qrw_inlist2join_optmode	是
qual_num_distinct	是
query_dop	否
rewrite_rule	否
skew_option	是

示例

对示例中原语句使用如下 hint：

```
explain
select /*+ set global(query_dop 0) */ i_product_name product_name
...
```

该 hint 表示：在整个语句的计划生成过程中，将配置参数 query_dop 设置为 0，即打开 SMP 自适应功能。生成的计划如下图所示：

```
 id |                                      operation                                       | E-rows  | E-memory | E-width | E-costs
----+--------------------------------------------------------------------------------------+---------+----------+---------+---------
  1 | ->  Row Adapter                                                                      |       1 |          |     230 | 19595.89
  2 |    ->  Vector Sonic Hash Aggregate                                                   |       1 |          |     230 | 19595.89
  3 |       ->  Vector Streaming (type: GATHER)                                            |       3 |          |     230 | 19595.89
  4 |          ->  Vector Sonic Hash Aggregate                                             |       3 | 16MB     |     230 | 19595.66
  5 |             ->  Vector Nest Loop (6,28)                                              |       3 | 1MB      |     126 | 19595.62
  6 |                ->  Vector Nest Loop (7,27)                                           |       3 | 1MB      |     130 | 19291.57
  7 |                   ->  Vector Streaming(type: LOCAL GATHER dop: 1/2)                  |       3 | 4MB      |     118 | 19279.41
  8 |                      ->  Vector Nest Loop (9,24)                                     |       3 | 1MB      |     118 | 19279.38
  9 |                         ->  Vector Streaming(type: SPLIT REDISTRIBUTE dop: 2/2)      |       3 | 4MB      |      82 | 18117.66
 10 |                            ->  Vector Nest Loop (11,21)                              |       3 | 1MB      |      82 | 18117.61
 11 |                               ->  Vector Streaming(type: SPLIT REDISTRIBUTE dop: 2/2)|       3 | 4MB      |      82 | 16195.20
 12 |                                  ->  Vector Sonic Hash Join (13,15)                 |       3 | 16MB     |      82 | 16195.15
 13 |                                     ->  Vector Partition Iterator                   |  287514 | 1MB      |      12 | 1110.42
 14 |                                        ->  Partitioned CStore Scan on store_returns |  287514 | 1MB      |      12 | 1110.42
 15 |                                     ->  Vector Streaming(type: LOCAL BROADCAST dop: 2/2)|    2764 | 4MB      |      94 | 14718.42
 16 |                                        ->  Vector Sonic Hash Join (17,19)           |    1382 | 16MB     |      94 | 14699.69
 17 |                                           ->  Vector Partition Iterator             | 2880404 | 1MB      |      39 | 11541.07
 18 |                                              ->  Partitioned CStore Scan on store_sales| 2880404 | 1MB      |      39 | 11541.07
 19 |                                           ->  Vector Streaming(type: LOCAL BROADCAST dop: 2/2)|      16 | 4MB      |      55 | 1947.12
 20 |                                              ->  CStore Scan on item                 |       8 | 1MB      |      55 | 1947.00
 21 |                               ->  Vector Materialize                                 |  100000 | 16MB     |       8 | 1797.41
 22 |                                  ->  Vector Streaming(type: LOCAL REDISTRIBUTE dop: 2/2)|  100000 | 4MB      |       8 | 1714.07
 23 |                                     ->  CStore Scan on customer                     |  100000 | 1MB      |       8 | 703.67
 24 |                            ->  Vector Materialize                                   |   50000 | 16MB     |      44 | 1099.22
 25 |                               ->  Vector Streaming(type: LOCAL REDISTRIBUTE dop: 2/2)|   50000 | 4MB      |      44 | 1057.55
 26 |                                  ->  CStore Scan on customer_address ad2           |   50000 | 1MB      |      44 | 552.33
 27 |                      ->  CStore Scan on store                                       |      36 | 1MB      |      20 | 12.01
 28 |                   ->  CStore Scan on promotion                                      |     900 | 1MB      |       4 | 300.30
(28 rows)
```

11.3.7.10 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。
  - 对于 AGG 重分布列的 hint，hint 未被使用的可能原因如下：
    - 指定的分布键包含不支持重分布的数据类型。
    - 执行计划中不需要重分布。
    - 执行的分布键的序号有误。
    - 对于使用 grouping sets 子句和 cube 子句的 AP 函数，window agg 中的分布键，不支持 hint。

说明

指定 Agg 重分布列 Hint，仅 8.1.3.100 及以上集群版本支持。

  - 如果子链接未被提升，则 blockname hint 不会被使用。
  - 对于 skew hint，hint 未被使用的可能原因如下：
    - 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。
  - 对于 guc hint，hint 未被使用的可能原因如下：
    - 配置参数不存在。
    - 配置参数不支持 guc hint。
    - 配置参数的值无效。
    - 语句级别的 guc hint 没有被写在顶层查询中。
    - 子查询级别的 guc hint 设置的配置参数不支持在子查询级别设置。
    - guc hint 所在的子查询被提升。

11.3.7.11 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
    );
```

11.3.8 例行维护表

为了保证数据库的有效运行，数据库必须在插入/删除操作后，基于客户场景，定期做 VACUUM FULL 和 ANALYZE，更新统计信息，以便获得更优的性能。

相关概念

使用 VACUUM、VACUUM FULL 和 ANALYZE 命令定期对每个表进行维护，主要有以下原因：

- VACUUM FULL 可回收已更新或已删除的数据所占据的磁盘空间，同时将小数据文件合并。
- VACUUM 对每个表维护了一个可视化映射来跟踪包含对别的活动事务可见的数组的页。一个普通的索引扫描首先通过可视化映射来获取对应的数组，来检查是否

对当前事务可见。若无法获取，再通过堆数组抓取的方式来检查。因此更新表的可视化映射，可加速唯一索引扫描。

- VACUUM 可避免执行的事务数超过数据库阈值时，事务 ID 重叠造成的原有数据丢失。
- ANALYZE 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，生成最有效的执行计划。

操作步骤

步骤 1 使用 VACUUM 或 VACUUM FULL 命令，进行磁盘空间回收。

- **VACUUM：**

  对表执行 VACUUM 操作

  ```
  VACUUM customer;
  ```

  可以与数据库操作命令并行运行。（执行期间，可正常使用的语句：SELECT、INSERT、UPDATE 和 DELETE。不可正常使用的语句：ALTER TABLE）。

  对表分区执行 VACUUM 操作

  ```
  VACUUM customer_par PARTITION ( P1 );
  ```

- **VACUUM FULL：**

  ```
  VACUUM FULL customer;
  ```

  需要向正在执行的表增加排他锁，且需要停止其他所有数据库操作。

  在进行磁盘空间回收时，用户可以使用如下命令查询集群中最早事务对应 session，再根据需要结束最早执行的长事务，从而更加高效的利用磁盘空间。

  a. 使用命令从 GTM 上查询 oldestxmin

  ```
  select * from pgxc_gtm_snapshot_status();
  ```

  b. 从 CN 上查询对应的 session 的 pid，此处 xmin 为上一步的 oldestxmin。

  ```
  select * from pgxc_running_xacts() where xmin=1400202010;
  ```

步骤 2 使用 ANALYZE 语句更新统计信息。

```
ANALYZE customer;
```

使用 ANALYZE VERBOSE 语句更新统计信息，并输出表的相关信息。

```
ANALYZE VERBOSE customer;
```

也可以同时执行 VACUUM ANALYZE 命令进行查询优化。

```
VACUUM ANALYZE customer;
```

说明

VACUUM 和 ANALYZE 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，建议通过“vacuum_cost_delay”参数设置。

步骤 3 删除表

```
DROP TABLE customer;
DROP TABLE customer_par;
DROP TABLE part;
```

----结束

维护建议

- 定期对部分大表做 VACUUM FULL，在性能下降后为全库做 VACUUM FULL，目前暂定每月做一次 VACUUM FULL。
- 定期对系统表做 VACUUM FULL，主要是 PG_ATTRIBUTE。
- 启用系统自动清理进程（AUTOVACUUM）自动执行 VACUUM 和 ANALYZE，回收被标识为删除状态的记录空间，并更新表的统计数据。

11.3.9 例行重建索引

背景信息

数据库经过多次删除操作后，索引页面上的索引键将被删除，造成索引膨胀。例行重建索引，可有效的提高查询效率。

数据库支持的索引类型包含 B-tree 索引、GIN 索引和 PSORT 索引。

- 对于 B-tree 索引，例行重建索引可有效的提高查询效率。
  - 如果数据发生大量删除后，索引页面上的索引键将被删除，导致索引页面数量的减少，造成索引膨胀。重建索引可回收浪费的空间。
  - 新建的索引中逻辑结构相邻的页面，通常在物理结构中也是相邻的，所以一个新建的索引比更新了多次的索引访问速度要快。
- 对于非 B-tree 索引，不建议例行重建。

重建索引

重建索引有以下两种方式：

- 先删除索引（DROP INDEX），再创建索引（CREATE INDEX）。

  在删除索引过程中，会在父表上增加一个短暂的排他锁，阻止相关读写操作。在创建索引过程中，会锁住写操作但是不会锁住读操作，此时读操作只能使用顺序扫描。
- 使用 REINDEX 语句重建索引。
  - 使用 REINDEX TABLE 语句重建索引，会在重建过程中增加排他锁，阻止相关读写操作。
  - 使用 REINDEX INTERNAL TABLE 语句重建 desc 表（包括）的索引，会在重建过程中增加排他锁，阻止相关读写操作。

操作步骤

假定在导入表“areaS”上的“area_id”字段上存在普通索引“areaS_idx”。重建索引有以下两种方式：

- 先删除索引（DROP INDEX），再创建索引（CREATE INDEX）

  a. 删除索引。

```
DROP INDEX areaS_idx;
```

b. 创建索引。

```
CREATE INDEX areaS_idx ON areaS (area_id);
```

- 使用 REINDEX 重建索引。
  - 使用 REINDEX TABLE 语句重建索引。

```
REINDEX TABLE areaS;
```

  - 使用 REINDEX INTERNAL TABLE 重建 desc 表（包括）的索引。

```
REINDEX INTERNAL TABLE areaS;
```

11.3.10 配置 SMP

介绍 SMP 模块的使用限制与适用场景，并给出 SMP 设置指南。

11.3.10.1 SMP 适用场景与限制

背景信息

SMP 特性通过算子并行来提升性能，同时会占用更多的系统资源，包括 CPU、内存、网络、I/O 等等。本质上 SMP 是一种以资源换取时间的方式，在合适的场景以及资源充足的情况下，能够起到较好的性能提升效果；但是如果在不合适的场景下，或者资源不足的情况下，反而可能引起性能的劣化。同时，生成 SMP 需要考虑更多的候选计划，会导致生成时间较长，相比串行场景也会引起性能的劣化。

适用场景

- 支持并行的算子

  计划中存在以下算子支持并行：

  a. Scan：支持行存普通表和行存分区表顺序扫描、列存普通表和列存分区表顺序扫描、HDFS 内外表顺序扫描；支持 GDS 数据导入的外表扫描并行。以上均不支持复制表。

  b. Join：HashJoin、NestLoop

  c. Agg：HashAgg、SortAgg、PlainAgg、WindowAgg(只支持 partition by，不支持 order by)。

  d. Stream：Redistribute、Broadcast

  e. 其他：Result、Subqueryscan、Unique、Material、Setop、Append、VectoRow、RowToVec

- SMP 特有算子

  为了实现并行，新增了并行线程间的数据交换 Stream 算子供 SMP 特性使用。这些新增的算子可以看做 Stream 算子的子类。

  a. Local Gather：实现 DN 内部并行线程的数据汇总

  b. Local Redistribute：在 DN 内部各线程之间，按照分布键进行数据重分布

  c. Local Broadcast：将数据广播到 DN 内部的每个线程

  d. Local RoundRobin：在 DN 内部各线程之间实现数据轮询分发

  e. Split Redistribute：在集群跨 DN 的并行线程之间实现数据重分布

f. Split Broadcast：将数据广播到集群所有 DN 的并行线程

上述新增算子可以分为 Local 与非 Local 两类，Local 类算子实现了 DN 内部并行线程间的数据交换，而非 Local 类算子实现了跨 DN 的并行线程间的数据交换。

- 示例说明

以 TPCH Q1 的并行计划为例：

```
 id |                                operation                                 |
----+--------------------------------------------------------------------------+
  1 | ->  Row Adapter                                                          |
  2 |    ->  Vector Streaming (type: GATHER)                                   |
  3 |       ->  Vector Sort                                                    |
  4 |          ->  Vector Streaming(type: LOCAL GATHER dop: 1/4)               |
  5 |             ->  Vector Hash Aggregate                                    |
  6 |                ->  Vector Streaming(type: SPLIT REDISTRIBUTE dop: 4/4)   |
  7 |                   ->  Vector Hash Aggregate                              |
  8 |                      ->  Vector Append(9, 10)                            |
  9 |                         ->  Dfs Scan on lineitem                         |
 10 |                         ->  Vector Adapter                               |
 11 |                            ->  Seq Scan on pg_delta_1423863972 lineitem  |
(11 rows)
```

在这个计划中，实现了 Hdfs Scan 以及 HashAgg 算子的并行，并且新增了 Local Gather 和 Split Redistribute 数据交换算子。

其中 6 号算子为 Split Redistribute 算子，上面标有的“dop: 4/4”表明 Split Redistribute 的发送端和接收端线程的并行度均为 4。4 号算子为 Local Gather，上面标有“dop: 1/4”，该算子的发送端线程并行度为 4，而接收端线程并行度为 1，即下层的 5 号 Hash Aggregate 算子按照 4 并行度执行，而上层的 1~3 号算子按照串行执行，4 号算子实现了 DN 内并行线程的数据汇总。

通过计划 Stream 算子上标明的 dop 信息即可看出各个算子的并行情况。

非适用场景

1. 生成计划时间占比很高的短查询场景。
2. 不支持 CN 上的算子并行。
3. 不支持不能下推的查询并行执行。
4. 不支持子查询 subplan 的并行，以及包含子查询的算子并行。

11.3.10.2 资源对 SMP 性能的影响

SMP 架构是一种利用富余资源来换取时间的方案，计划并行之后必定会引起资源消耗的增加，包括 CPU、内存、I/O 和网络带宽等资源的消耗都会出现明显的增长，而且随着并行度的增大，资源消耗也随之增大。当上述资源成为瓶颈的情况下，SMP 无法提升性能，反而可能导致集群整体性能的劣化。SMP 支持自适应特性，该特性会根据当前资源和查询特征，动态选取最优的并行度。下面对各种资源对 SMP 性能的影响情况分别进行说明：

- **CPU 资源**

在一般客户场景中，系统 CPU 利用率不高的情况下，利用 SMP 并行架构能够更充分地利用系统 CPU 资源，提升系统性能。但当数据库服务器的 CPU 核数较少，CPU 利用率已经比较高的情况下，如果打开 SMP 并行，不仅性能提升不明显，反而可能因为多线程间的资源竞争而导致性能劣化。

- **内存资源**

查询并行后会导致内存使用量的增长，但每个算子使用内存上限仍受到work_mem 等参数的限制。假设 work_mem 为 4GB，并行度为 2，那么每个并行线程所分到的内存上限为 2GB。在 work_mem 较小或者系统内存不充裕的情况下，使用 SMP 并行后，可能出现数据下盘，导致查询性能劣化的问题。

- **网络带宽资源**

  为了实现查询并行执行，会新增并行线程间的数据交换算子。对于 Local 类 Stream 算子，所需要进行数据交换的线程在同一个 DN 内，通过内存交换，不会增加网络负担。而非 Local 类算子，需要通过网络进行数据交换，因此会加重网络负担。当网络资源成为瓶颈的情况下，并行可能会导致一定程度的劣化。

- **I/O 资源**

  要实现并行扫描必定会增加 I/O 的资源消耗，因此只有在 I/O 资源充足的情况下，并行扫描才能够提高扫描性能。

11.3.10.3 其他因素对 SMP 性能的影响

除了资源因素外，还有一些因素也会对 SMP 并行性能造成影响。例如分区表中分区数据不均，以及系统并发度等因素。

- **数据倾斜对 SMP 性能的影响**

  当数据中存在严重数据倾斜时，并行效果较差。例如某表 join 列上某个值的数据量远大于其他值，开启并行后，根据 join 列的值对该表数据做 hash 重分布，使得某个并行线程的数据量远多于其他线程，造成长尾问题，导致并行后效果差。

- **系统并发度对 SMP 性能的影响**

  SMP 特性会增加资源的使用，而在高并发场景下资源剩余较少。所以，如果在高并发场景下，开启 SMP 并行，会导致各查询之间严重的资源竞争问题。一旦出现了资源竞争的现象，无论是 CPU、I/O、内存或者网络资源，都会导致整体性能的下降。因此在高并发场景下，开启 SMP 往往不能达到性能提升的效果，甚至可能引起性能劣化。

11.3.10.4 SMP 相关参数配置建议

自当前版本开始，SMP 自适应开启，新安装的集群 query_dop 默认值设置为 0，并完成 SMP 相关参数的调整。为保持前向兼容，旧集群升级后的 query_dop 值与升级前保持一致。

对于升级集群，如果要设置 query_dop=0 并开启 SMP 并行，需同步调整以下相关参数值，以获取更佳的 dop 选择：

- comm_usable_memory

  当系统内存较大时，max_process_memory 设置较大，可适当调大该值，建议设置为 max_process_memory 的 5%，默认值为 4GB。

- comm_max_stream

  设置建议值为：comm_max_stream=Min(dop_limit * dop_limit * 20 * 2, max_process_memory(字节数) * 0.025 / 总 DN 数 / 260)，且该值在 comm_max_stream 取值范围内。

- max_connections

  设置建议值为：max_connections=dop_limit * 20 * 6, + 24，且该值在 max_connections 取值范围内。

⚠ 注意

公式中的 dop_limit 为集群中每个 DN 对应的 CPU 数，计算公式为：dop_limit = 单机器的 CPU 逻辑核数 / 单机器的 DN 数。

11.3.10.5 SMP 手动调优建议

如果想手动进行 SMP 调优，需要熟练掌握 11.3.10.4 SMP 相关参数配置建议，并了解本节内容。

使用限制

系统的 CPU、内存、I/O 和网络带宽等资源充足。SMP 架构是一种利用富余资源来换取时间的方案，计划并行之后必定会引起资源消耗的增加，当上述资源成为瓶颈的情况下，SMP 无法提升性能，反而可能导致性能的劣化。同时，SMP 计划的生成时间较串行要长。因此，在短查询为主的 TP 类业务中，或者出现资源瓶颈的情况下，建议关闭 SMP，即设置 query_dop=1。

配置步骤

1. 观察当前系统负载情况，如果系统资源充足（资源利用率小于 50%），执行步骤 2；否则退出。
2. 设置 query_dop=1（默认值），利用 explain 打出执行计划，观察计划是否符合 11.3.10.1 SMP 适用场景与限制小节中的适用场景。如果符合，进入下一步。
3. 设置 query_dop=-value，在考虑资源情况和计划特征基础上，限制 dop 选取的范围为[1,value]。
4. 设置 query_dop=value，不考虑资源情况和计划特征，强制选取 dop 为 1 或 value。
5. 在符合条件的查询语句执行前设置合适的 query_dop 值，在语句执行结束后关闭 query_dop。例如，

```
SET query_dop = 0;
SELECT COUNT(*) FROM t1 GROUP BY a;
......
SET query_dop = 1;
```

说明

- 资源许可的情况下，并行度越高，性能提升效果越好。
- SMP 并行度支持会话级设置，推荐客户在执行符合要求的查询前，打开 smp，执行结束后，关闭 smp。以免在业务峰值时，对业务造成冲击。
- SMP 自适应（query_dop<=0）依赖资源管理，如果资源管理禁用（use_workload_manager 为 off），那么只会产生 1 或 2 并行度的计划。

11.4 实际调优案例

11.4.1 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 11-11 所示。

图11-11 选择合适的分布列案例（一）

```
postgres=> 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：

```
ALTER TABLE t1  DISTRIBUTE BY HASH (a);
ALTER TABLE t2  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 11-12 所示。

图11-12 选择合适的分布列案例（二）

```
postgres=> 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)
```

11.4.2 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：

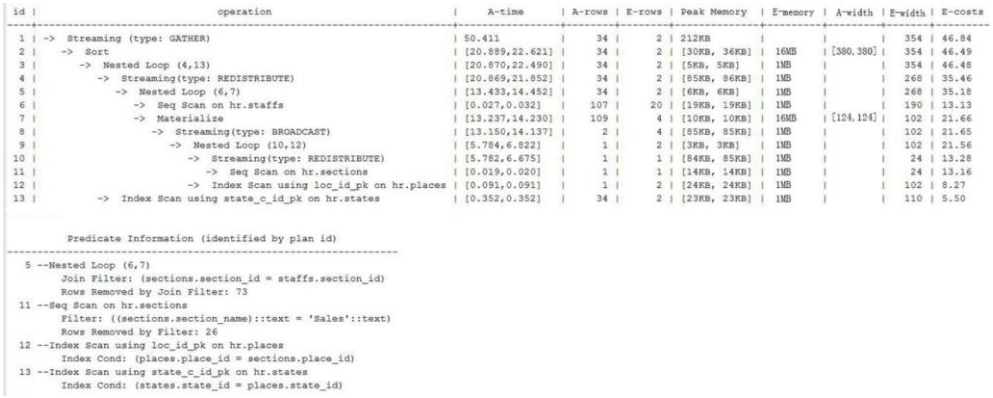
```
SELECT staff_id,first_name,last_name,employment_id,state_name,city
FROM staffs,sections,states,places
WHERE sections.section_name='Sales'
AND staffs.section_id = sections.section_id
AND sections.place_id = places.place_id
AND places.state_id = states.state_id
ORDER BY staff_id;
```

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                       |     A-time      | A-rows | E-rows | Peak Memory  | E-memory | A-width | E-width | E-costs
---+-------------------------------------------------------+-----------------+--------+--------+--------------+----------+---------+---------+---------
 1 | ->  Streaming (type: GATHER)                          | 69.801          |     34 |      2 |212KB         |          |         |     354 | 53.67
 2 |    ->  Sort                                           | [21.508,23.283] |     34 |      2 |[30KB, 36KB]  | 16MB     | [380,380] |     354 | 53.32
 3 |       ->  Hash Join (4,5)                             | [21.483,23.153] |     34 |      2 |[7KB, 7KB]    | 1MB      |         |     354 | 53.31
 4 |          ->  Seq Scan on hr.states                    | [0.007,0.022]   |     17 |     20 |[12KB, 12KB]  | 1MB      |         |     110 | 13.13
 5 |          ->  Hash                                     | [21.026,22.663] |     34 |      2 |[262KB, 294KB]| 16MB     | [284,284] |     268 | 40.08
 6 |             ->  Streaming(type: REDISTRIBUTE)         | [21.024,22.458] |     34 |      2 |[85KB, 86KB]  | 1MB      |         |     268 | 40.08
 7 |                ->  Hash Join (8,9)                    | [13.814,14.827] |     34 |      2 |[8KB, 8KB]    | 1MB      |         |     268 | 39.80
 8 |                   ->  Seq Scan on hr.staffs           | [0.035,0.043]   |    107 |     20 |[19KB, 19KB]  | 1MB      |         |     190 | 13.13
 9 |                   ->  Hash                            | [13.361,14.348] |      2 |      4 |[292KB, 292KB]| 16MB     | [124,124] |     102 | 26.57
10 |                      ->  Streaming(type: BROADCAST)   | [13.291,14.279] |      2 |      4 |[85KB, 85KB]  | 1MB      |         |     102 | 26.57
11 |                         ->  Hash Join (12,13)         | [6.359,7.446]   |      1 |      2 |[6KB, 6KB]    | 1MB      |         |     102 | 26.48
12 |                            ->  Seq Scan on hr.places  | [0.008,0.018]   |     15 |     20 |[14KB, 14KB]  | 1MB      |         |     102 | 13.13
13 |                            ->  Hash                   | [5.999,7.077]   |      1 |      1 |[259KB, 291KB]| 16MB     | [32,32] |      24 | 13.28
14 |                               ->  Streaming(type: REDISTRIBUTE) | [5.999,6.958]   |      1 |      1 |[84KB, 85KB]  | 1MB      |         |      24 | 13.28
15 |                                  ->  Seq Scan on hr.sections | [0.021,0.022]   |      1 |      1 |[14KB, 14KB]  | 1MB      |         |      24 | 13.16


        Predicate Information (identified by plan id)
-------------------------------------------------------------------
  3 --Hash Join (4,5)
        Hash Cond: (states.state_id = places.state_id)
  7 --Hash Join (8,9)
        Hash Cond: (staffs.section_id = sections.section_id)
 11 --Hash Join (12,13)
        Hash Cond: (places.place_id = sections.place_id)
 15 --Seq Scan on hr.sections
        Filter: ((sections.section_name)::text = 'Sales'::text)
        Rows Removed by Filter: 26
```

建议在 places.place_id 和 states.state_id 列上建立 2 个索引，执行计划如下：



```
 id |                                operation                                |      A-time       | A-rows | E-rows |  Peak Memory  | E-memory | A-width | E-width | E-costs
----+-------------------------------------------------------------------------+-------------------+--------+--------+---------------+----------+---------+---------+---------
  1 | ->  Streaming (type: GATHER)                                            | 50.411            |     34 |      2 | 212KB         |          |         |     354 |   46.84
  2 |    ->  Sort                                                             | [20.889,22.621]   |     34 |      2 | [30KB, 36KB]  | 16MB     | [380,380] |     354 |   46.49
  3 |       ->  Nested Loop (4,13)                                            | [20.870,22.490]   |     34 |      2 | [5KB, 5KB]    | 1MB      |         |     354 |   46.48
  4 |          ->  Streaming(type: REDISTRIBUTE)                              | [20.869,21.852]   |     34 |      2 | [85KB, 86KB]  | 1MB      |         |     268 |   35.46
  5 |             ->  Nested Loop (6,7)                                       | [13.433,14.452]   |     34 |      2 | [6KB, 6KB]    | 1MB      |         |     268 |   35.18
  6 |                ->  Seq Scan on hr.staffs                                | [0.027,0.032]     |    107 |     20 | [19KB, 19KB]  | 1MB      |         |     190 |   13.13
  7 |                ->  Materialize                                          | [13.237,14.230]   |    109 |      4 | [10KB, 10KB]  | 16MB     | [124,124] |     102 |   21.66
  8 |                   ->  Streaming(type: BROADCAST)                        | [13.150,14.137]   |      2 |      4 | [85KB, 85KB]  | 1MB      |         |     102 |   21.65
  9 |                      ->  Nested Loop (10,12)                            | [5.784,6.822]     |      1 |      2 | [3KB, 3KB]    | 1MB      |         |     102 |   21.56
 10 |                         ->  Streaming(type: REDISTRIBUTE)               | [5.782,6.675]     |      1 |      1 | [84KB, 85KB]  | 1MB      |         |      24 |   13.28
 11 |                            ->  Seq Scan on hr.sections                  | [0.019,0.020]     |      1 |      1 | [14KB, 14KB]  | 1MB      |         |      24 |   13.16
 12 |                         ->  Index Scan using loc_id_pk on hr.places     | [0.091,0.091]     |      1 |      2 | [24KB, 24KB]  | 1MB      |         |     102 |    8.27
 13 |          ->  Index Scan using state_c_id_pk on hr.states                | [0.352,0.352]     |     34 |      2 | [23KB, 23KB]  | 1MB      |         |     110 |    5.50


        Predicate Information (identified by plan id)
-------------------------------------------------------------
  5 --Nested Loop (6,7)
       Join Filter: (sections.section_id = staffs.section_id)
       Rows Removed by Join Filter: 73
 11 --Seq Scan on hr.sections
       Filter: ((sections.section_name)::text = 'Sales'::text)
       Rows Removed by Filter: 26
 12 --Index Scan using loc_id_pk on hr.places
       Index Cond: (places.place_id = sections.place_id)
 13 --Index Scan using state_c_id_pk on hr.states
       Index Cond: (states.state_id = places.state_id)
```

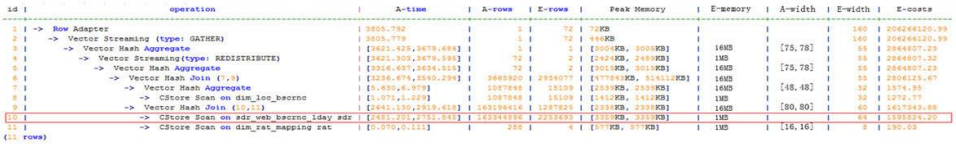
11.4.3 案例：增加 JOIN 列非空条件

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 11-13 所示。

图11-13 增加 JOIN 列非空条件（一）



优化分析

1. 分析执行计划图 11-13 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。

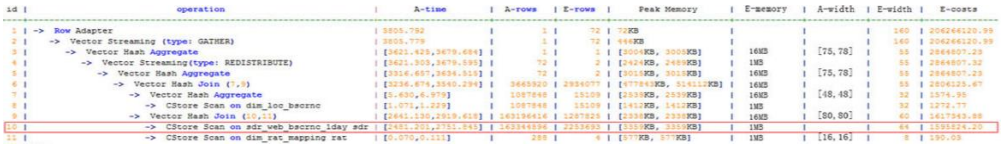
   建议在语句中手动添加 JOIN 列的非空判断，修改后的语句如下所示。

```
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 11-14 所示。

图11-14 增加 JOIN 列非空条件（二）



11.4.4 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：

```
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：

```
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 7s，优化效果明显。

11.4.5 案例：设置 cost_param 对查询性能优化

现象描述

cost_param 的 bit0(set cost_param=1)值为 1 时，表示对于求!=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。V300R002C00 版本开始已弃用 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
```

```
) with (orientation = column, COMPRESSION = MIDDLE) 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
)with (orientation = column, COMPRESSION = MIDDLE) 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
```

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

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_NATIONKEYINT NOT NULL
, N_NAMECHAR(25) NOT NULL
, N_REGIONKEYINT NOT NULL
, N_COMMENTVARCHAR(152)
) distribute by replication;
CREATE TABLE SUPPLIER
(
S_SUPPKEYBIGINT NOT NULL
, S_NAMECHAR(25) NOT NULL
, S_ADDRESSVARCHAR(40) NOT NULL
, S_NATIONKEYINT NOT NULL
, S_PHONECHAR(15) NOT NULL
, S_ACCTBALDECIMAL(15,2) NOT NULL
, S_COMMENTVARCHAR(101) NOT NULL
) distribute by hash(S_SUPPKEY);
CREATE TABLE PARTSUPP
(
PS_PARTKEYBIGINT NOT NULL
, PS_SUPPKEYBIGINT NOT NULL
, PS_AVAILQTYBIGINT NOT NULL
, PS_SUPPLYCOSTDECIMAL(15,2)NOT NULL
, PS_COMMENTVARCHAR(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
```

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

11.4.6 案例：调整分布键

现象描述 1

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

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]，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
```

优化分析 1

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 信息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。

现象描述 2

除了分析 EXPLAIN ANALYZE 语句，也可以通过如下 SQL 语句直接查询表中数据在各个 DN 上的分布情况，以表 t 为例：

```
SELECT COUNT(*) FROM t GROUP BY(xc_node_id) ORDER BY 1;
```

上述语句返回的结果，就是表 t 的数据在各个 DN 上的行数，若某些 DN 的行数明显高于其它 DN，则说明存在存储倾斜的情况。

优化分析 2

一般情况下，存储倾斜的现象只发生在 hash 分布表中，当发生该现象后，可以按照如下步骤解决：

1. 重新选择分布列。

   在选择分布列的时候，应该选择值相对均匀的列作为分布列，即该列上不会存在某个值出现频率非常高。

   在满足上述条件的情况下，还应该考虑到业务的性质，尽量选取经常作 join 列或 group by 列的列作为分布列。

2. 若按照上述原则，找不到一个合适的分布列，则考虑修改分布方式为 roundrobin。

   roundrobin 分布会将表的数据均匀地分布在各个 DN 上，因此可以解决倾斜的问题。

修改分布列和修改分布方式，都是通过 ALTER TABLE 语句实现的，具体请参考 ALTER TABLE 的语法。

11.4.7 案例：调整局部聚簇键

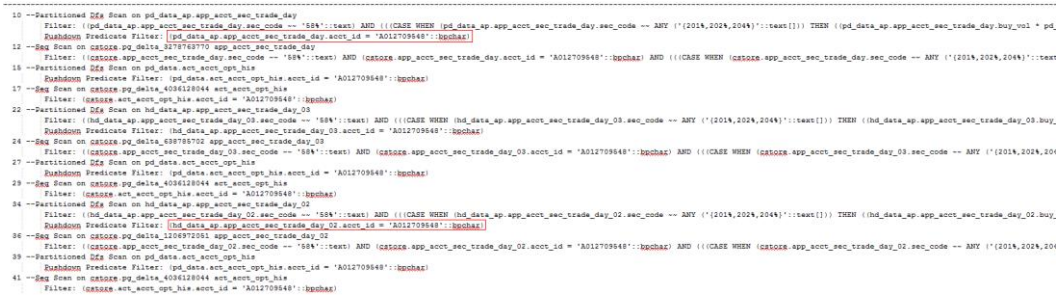
现象描述

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	[14348.466,14348.464]		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			132 | 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)	[10112.589,10112.589]		0	2 |[480KB, 480KB]	1MB			136 | 649136.79
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 |[2964KB, 2964KB]	1MB			57 | 649051.65
11	-> Vector Adapter	[0.005,0.005]		0	1 |[114KB, 114KB]	1MB			212 | 33.92
12	-> Seg 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	-> Seg 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	-> Seg 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	-> Seg 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 | 432151.58
35	-> Vector Adapter	[0.005,0.005]		0	1 |[114KB, 114KB]	1MB			212 | 33.92
36	-> Seg 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	-> Seg Scan on cstore.pg_delta_4036128044 act_acct_opt_his	[0,0]		0	1 |[0, 0]	1MB			11 | 2.21

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。



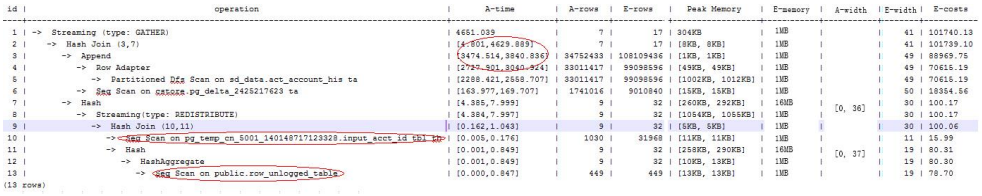
试着给两个表的 acct_id 列增加局部聚簇键，然后对两张表执行 VACUUM FULL，使局部聚簇生效。调整后性能得到提升。

11.4.8 案例：调整中间表存储方式

现象描述

在 GaussDB(DWS)中行存表天然的使用行执行引擎，列存表天然的使用列执行引擎。如果一个 SQL 语句涉及的表既有行存表又有列存表，系统会自动选择行执行引擎。由于列执行引擎的性能(除 indexscan 相关的算子)比行执行引擎性能要好很多，因此一般建议使用列存表。特别是对一些中间结果集转储的表，一定要分析清楚，使用合适的表存储类型。

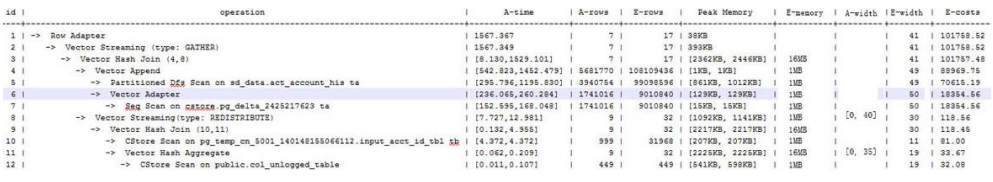
某局点测试过程遇到如下的执行计划，客户希望将性能提升至 3s 内返回结果。



优化分析

经过分析发现计划走了行引擎。根本原因是：临时计划表 input_acct_id_tbl 和中间结果转储表 row_unlogged_table 使用了行存表。

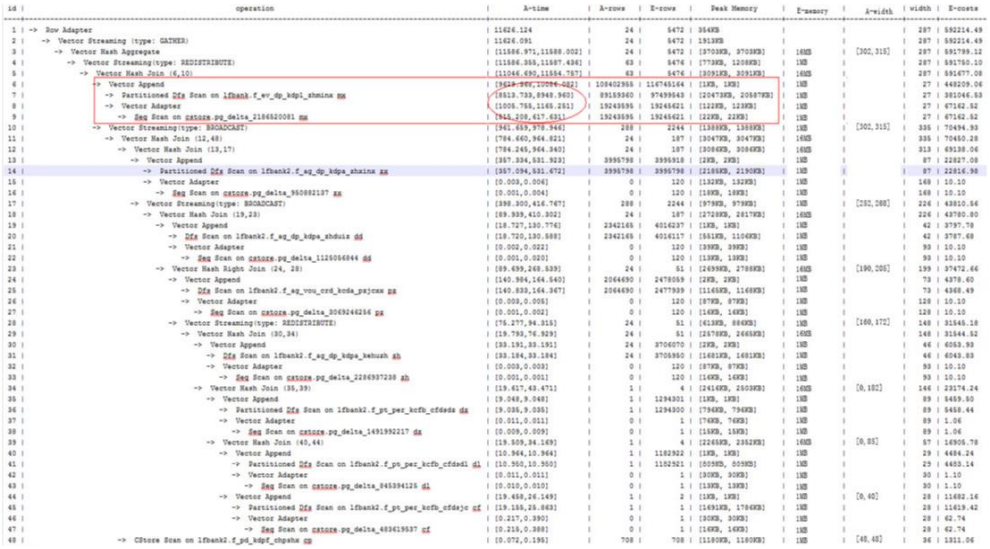
修改这两个表为列存表之后，性能提升至 1.6s。



11.4.9 案例：调整局部聚簇列

现象描述

某局点测试过程中出现如下计划，客户要求将性能提升至 3s 内返回。

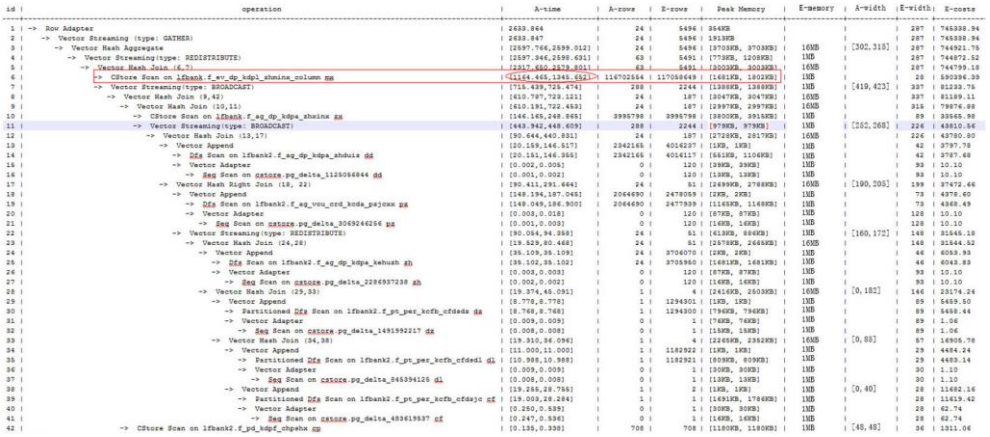


优化分析

分析发现上述计划的性能瓶颈点为 lfbank.f_ev_dp_kdpl_zhminx 的 scan。进一步分析此表的 Scan 条件如下：

```
                                                  Predicate Information (identified by plan id)
--------------------------------------------------------------------------------------------------------------------------------------------------------
  5 --Vector Hash Join (6,10)
      Hash Cond: (((mx.zhanghao)::text = (dd.zhanghao)::text) AND ((mx.farendma)::text = (cf.farendma)::text))
  7 --Partitioned Dfs Scan on lfbank.f_ev_dp_kdpl_zhminx mx
      Pushdown Predicate Filter: ((mx.yezdminc)::text = 'DPLDGBAL'::text)
  9 --Seq Scan on cstore.pg_delta_21865ZU0U1 mx
      Filter: ((mx.yezdminc)::text = 'DPLDGBAL'::text)
      Rows Removed by Filter: 489
```

尝试把 lfbank.f_ev_dp_kdpl_zhmin 表修改为列存表，然后在 yezdminc 列上建 PCK（局部聚簇），并设置 PARTIAL_CLUSTER_ROWS=100000000。执行计划优化为：



说明

- 此方法实际是靠牺牲数据导入时的性能来提升业务查询性能。

- 此方法导致局部排序的元组数增加，需要增大 psort_work_mem 来提高排序效率。

11.4.10 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。

```
postgres=# 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

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。

```
postgres=# 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

11.4.11 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：

```
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(DWS)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

```
set best_agg_plan to 1;
SET
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)
set best_agg_plan to 2;
SET
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)
set best_agg_plan to 3;
SET
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。

11.4.12 案例：改写 SQL 消除子查询（案例 1）

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：

```
postgres=# explain select 1,(select count(*)
postgres(#           from customer_address_001 a4
postgres(#          where a4.ca_address_sk = a.ca_address_sk
postgres(#         ) 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。

11.4.13 案例：改写 SQL 消除子查询（案例 2）

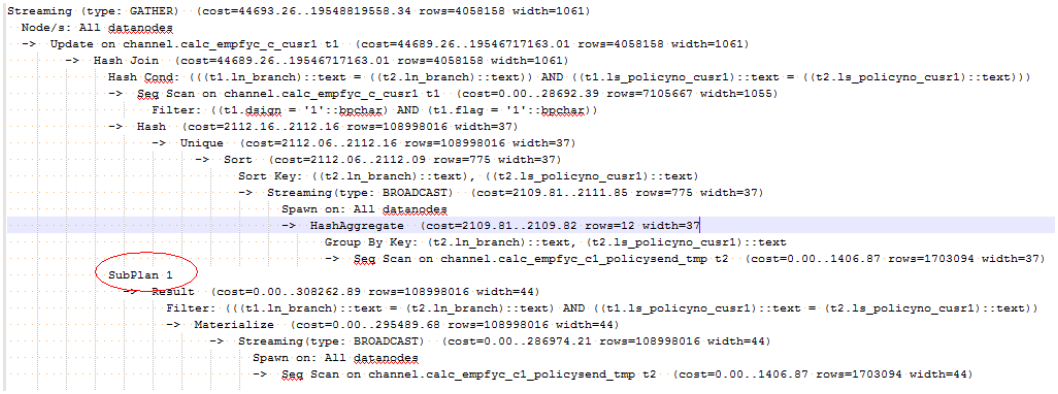
现象描述

某局点客户反馈如下 SQL 语句的执行时间超过 1 天未结束：

```
UPDATE calc empfyc c cusr1 t1
SET ln rec count =
 (
   SELECT CASE WHEN current date - ln process date + 1 <= 12 THEN 0 ELSE
t2.ln_rec_count END
```

```
    FROM calc_empfyc_c1_policysend_tmp t2
    WHERE t1.ln_branch = t2.ln_branch AND t1.ls_policyno_cusr1 =
t2.ls_policyno_cusr1
)
WHERE dsign = '1'
AND flag = '1'
AND EXISTS
    (SELECT 1
    FROM calc_empfyc_c1_policysend_tmp t2
    WHERE t1.ln_branch = t2.ln_branch AND t1.ls_policyno_cusr1 =
t2.ls_policyno_cusr1
    );
```

对应的执行计划如下：



优化说明

很明显，执行计划中存在 SubPlan，并且 SubPlan 中的运算相当重，即此 SubPlan 是一个明确的性能瓶颈点。

根据 SQL 语意等价改写 SQL 消除 SubPlan 如下：

```
UPDATE calc_empfyc_c_cusr1 t1
SET ln_rec_count = CASE WHEN current_date - ln_process_date + 1 <= 12 THEN 0 ELSE
t2.ln_rec_count END
FROM calc_empfyc_c1_policysend_tmp t2
WHERE
t1.dsign = '1' AND t1.flag = '1'
AND t1.ln_branch = t2.ln_branch AND t1.ls_policyno_cusr1 = t2.ls_policyno_cusr1;
```

改写之后 SQL 语句在 50S 内执行完成

11.4.14 案例：改写 SQL 排除剪枝干扰

现象描述

t_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整数。

测试 SQL 如下：

```
SELECT
   count(1)
FROM t_ddw_f10_op_cust_asset_mon b1
WHERE b1.year_mth  < substr('20200722',1 ,6 )
AND b1.year_mth + 1 >= substr('20200722',1 ,6 )
;
```

测试结果显示此 SQL 的表 Scan 耗时长达 10s，查询 SQL 语句的执行计划如下

```
EXPLAIN (ANALYZE ON, VERBOSE ON)
SELECT
   count(1)
FROM t_ddw_f10_op_cust_asset_mon b1
WHERE b1.year_mth < substr('20200722',1 ,6 )
AND b1.year_mth + 1 >= cast(substr('20200722',1 ,6 ) AS int);

QUERY PLAN
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
--------------------------------------
 id |                              operation                                |       A-
time        | A-rows  | E-rows  | E-distinct | Peak Memory  | E-memory | A-width |
E-width |  E-costs
 ----+-------------------------------------------------------------------------------
--+----------------------+----------+----------+------------+--------------+------
----+---------+---------+-----------
   1 | ->  Aggregate                                                          | 10662.260
|       1 |       1 |          | 32KB        |         |        |      8 | 593656.42
   2 |    ->  Streaming (type: GATHER)                                          |
10662.172           |       4 |       4 |          | 136KB       |         |        |
8 | 593656.42
   3 |      ->  Aggregate                                                       | [9692.785,
10656.068] |       4 |       4 |          | [24KB, 24KB] | 1MB     |         |      8
| 593646.42
   4 |         ->  Partition Iterator                                           |
[8787.198, 9629.138]  | 16384000 | 32752850 |          | [16KB, 16KB] | 1MB     |
|      0 | 573175.88
   5 |           ->  Partitioned Seq Scan on public.t_ddw_f10_op_cust_asset_mon b1 |
[8365.655, 9152.115]  | 16384000 | 32752850 |          | [32KB, 32KB] | 1MB     |
|      0 | 573175.88

                          SQL Diagnostic Information
 ------------------------------------------------------------------------------------
---------
 Partitioned table unprunable Qual
       table public.t_ddw_f10_op_cust_asset_mon b1:
       left side of expression "(((year_mth)::text)::bigint >= 202007)" is an
expression or occurs type conversion

               Predicate Information (identified by plan id)
 ------------------------------------------------------------------------------------
   4 --Partition Iterator
        Iterations: 6
   5 --Partitioned Seq Scan on public.t_ddw_f10_op_cust_asset_mon b1
        Filter: ((b1.year_mth < 202007::bigint) AND ((b1.year_mth + 1) >= 202007))
```

```
       Rows Removed by Filter: 81920000
       Partitions Selected by Static Prune: 1..6
```

优化分析

分析语句的执行计划，查看执行计划中的 SQL 自诊断信息，发现如下诊断信息：

```
                          SQL Diagnostic Information
 ------------------------------------------------------------------------------------
--------
 Partitioned table unprunable Qual
       table public.t_ddw_f10_op_cust_asset_mon b1:
       left side of expression "((year_mth + 1) > 202008)" invokes function-
call/type-conversion
```

Filter 条件中存在表达式(year_mth + 1) > 202008，这种**表达式一侧不是单纯的分区键、而是包含分区键的表达式的 Filter 条件是不能用来剪枝的**，因而导致查询语句扫描了几乎整个分区表的数据。

跟原始 SQL 语句对比，可以确定表达式 '(year_mth + 1) > 202008' 是从表达式 'b1.year_mth + 1 > substr('20200822',1 ,6)' 衍生而来，按照诊断信息把修改 SQL 语句为如下方式

```
SELECT
   count(1)
FROM t_ddw_f10_op_cust_asset_mon b1
WHERE b1.year_mth <= substr('20200822',1 ,6 )
AND b1.year_mth > cast(substr('20200822',1 ,6 ) AS int) - 1;
```

改写之后，SQL 语句的执行信息如下，可以看到不剪枝告警已经消除，剪枝后需要扫描分区数为 1，执行时间从 10s 提升至 3s。

```
EXPLAIN (analyze ON, verbose ON)
SELECT
   count(1)
FROM t_ddw_f10_op_cust_asset_mon b1
WHERE b1.year_mth < substr('20200722',1 ,6 )
AND b1.year_mth >= cast(substr('20200722',1 ,6 ) AS int) - 1;

QUERY PLAN
------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----------------------------------
 id |                               operation                                |      A-
time       | A-rows  | E-rows  | E-distinct | Peak Memory  | E-memory | A-width |
E-width |  E-costs
 ----+------------------------------------------------------------------------------
--+----------------------+----------+----------+------------+--------------+-------
---+---------+---------+-----------
   1 | ->  Aggregate                                                          | 3009.796
|       1 |       1 |          | 32KB        |         |         |      8 | 501541.70
   2 |    ->  Streaming (type: GATHER)                                          |
3009.718           |       4 |       4 |          | 136KB       |         |         |
8 | 501541.70
   3 |      ->  Aggregate                                                       | [2675.509,
```

```
3003.298] |       4 |       4 |          | [24KB, 24KB] | 1MB     |        |      8 |
501531.70
  4 |        ->  Partition Iterator                                          |
[1820.725, 2053.836] | 16384000 | 16380697 |          | [16KB, 16KB] | 1MB     |
|      0 | 491293.75
  5 |           ->  Partitioned Seq Scan on public.t_ddw_f10_op_cust_asset_mon b1 |
[1420.972, 1590.083] | 16384000 | 16380697 |          | [16KB, 16KB] | 1MB     |
|      0 | 491293.75

             Predicate Information (identified by plan id)
 ------------------------------------------------------------------------------
  4 --Partition Iterator
       Iterations: 1
  5 --Partitioned Seq Scan on public.t_ddw_f10_op_cust_asset_mon b1
       Filter: ((b1.year_mth < 202007::bigint) AND (b1.year_mth >= 202006))
       Partitions Selected by Static Prune: 6
```

11.4.15 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

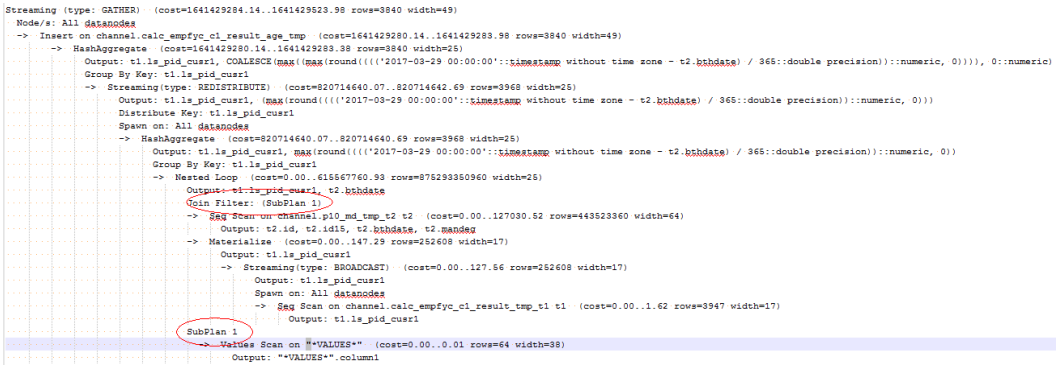
```
select
count(1)
from calc empfyc c1 result tmp t1
where ls_pid_cusr1 in any('20120405', '20130405');
```

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

id	operation	A-time	A-rows	E-rows	Peak Memory	E-memory	A-width
1	-> Streaming (type: GATHER)	6737.281	0	192	292KB		
2	-> Insert on channel.calc_empfyc_c1_result_age_tmp	[4665.024,4990.666]	0	192	[1108KB, 1108KB]	1MB	
3	-> HashAggregate	[4664.996,4990.641]	0	192	[12KB, 12KB]	16MB	
4	-> Streaming(type: REDISTRIBUTE)	[4664.991,4990.637]	0	3392	[2090KB, 2090KB]	1MB	
5	-> HashAggregate	[3416.939,4958.348]	0	3392	[14KB, 14KB]	16MB	
6	-> Append	[3416.936,4958.340]	0	4011	[1KB, 1KB]	1MB	
7	-> Hash Join (8,9)	[2011.226,3080.697]	0	3947	[6KB, 6KB]	1MB	
8	-> Seq Scan on channel.p10_md_tmp_t2 t2	[803.782,1238.984]	443525717	443523360	[12KB, 12KB]	1MB	
9	-> Hash	[4.357,328.979]	252608	252608	[482KB, 482KB]	16MB	[35,39]
10	-> Streaming(type: BROADCAST)	[2.345,326.320]	252608	252608	[2090KB, 2090KB]	1MB	
11	-> Seq Scan on channel.calc_empfyc_c1_result_tmp_t1 t1	[0.011,0.030]	3947	3947	[11KB, 11KB]	1MB	
12	-> Hash Join (13,14)	[1376.258,2066.110]	0	64	[5KB, 5KB]	1MB	
13	-> Seq Scan on channel.p10_md_tmp_t2 t2	[777.552,1388.499]	443525717	443523360	[12KB, 12KB]	1MB	
14	-> Hash	[2.812,4.217]	252608	252608	[482KB, 482KB]	16MB	[28,27]
15	-> Streaming(type: BROADCAST)	[1.276,1.868]	252608	252608	[2090KB, 2090KB]	1MB	
16	-> Seq Scan on channel.calc_empfyc_c1_result_tmp_t1 t1	[0.010,0.033]	3947	3947	[11KB, 11KB]	1MB	

(16 rows)

优化后，从超过 1 个小时未返回结果优化到 7s 返回结果。

11.4.16 案例：使用 partial cluster key

列存表可以选取某一列或几列设置为 partial cluster key(column_name[, ...])。在导入数据时，按设置的列进行局部排序（默认每 70 个 CU 即 420 万行排序一次），生成的 CU 会聚集在一起，即 CU 的 min,max 会在一个较小的区间内。当查询时，where 条件含有这些列时，可产生良好的过滤效果。

1. 使用 partial cluster key。

```
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
)
with (orientation = column)
distribute by hash(L_ORDERKEY);

select
sum(l extendedprice * l discount) as revenue
from
lineitem
where
l shipdate >= '1994-01-01'::date
and l_shipdate < '1994-01-01'::date + interval '1 year'
and l_discount between 0.06 - 0.01 and 0.06 + 0.01
and l_quantity < 24;
```

where 条件中 l_shipdate 和 l_quantity 的 distinct 值数量较少且可以做 min max 过滤，修改表定义如下：

```
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
, partial cluster key(l_shipdate, l_quantity)
)
with (orientation = column)
distribute by hash(L_ORDERKEY);
```

重新导入数据后执行查询，对比使用前后执行时间：

图11-15 未使用 partial cluster key

```
 id |                operation                |      A-time       | A-rows | E-rows |  Peak Memory   | A-width | E-width |  E-costs
----+-----------------------------------------+-------------------+--------+--------+----------------+---------+---------+----------
  1 | ->  Row Adapter                         | 1653.156          |      1 |      1 | 12KB           |         |      44 | 205803.90
  2 |   ->  Vector Aggregate                  | 1653.146          |      1 |      1 | 184KB          |         |      44 | 205803.90
  3 |     ->  Vector Streaming (type: GATHER) | 1653.070          |      1 |      1 | 174KB          |         |      44 | 205803.90
  4 |       ->  Vector Aggregate              | [1481.497,1481.497] |      1 |      1 | [225KB, 225KB] |         |      44 | 205803.84
  5 |         ->  CStore Scan on public.lineitem | [1405.004,1405.004] | 114160 | 111485 | [792KB, 792KB] |         |      12 | 205246.40
(5 rows)
```

图11-16 未使用 partial cluster key 后 CU 加载情况

```
  5 --CStore Scan on public.lineitem
        datanode1 (actual time=40.623..1405.004 rows=114160 loops=1)
        datanode1 (RoughCheck CU: CUNone: 0, CUSome: 101)
        datanode1 (LLVM Optimized)
        datanode1 (Buffers: shared hit=18385 read=23)
        datanode1 (CPU: ex c/r=31917, ex cyc=3643646206, inc cyc=3643646206)
```

图11-17 使用 partial cluster key

```
 id |                operation                |      A-time       | A-rows | E-rows |  Peak Memory   | A-width | E-width |  E-costs
----+-----------------------------------------+-------------------+--------+--------+----------------+---------+---------+----------
  1 | ->  Row Adapter                         | 459.539           |      1 |      1 | 12KB           |         |      44 | 205693.85
  2 |   ->  Vector Aggregate                  | 459.528           |      1 |      1 | 184KB          |         |      44 | 205693.85
  3 |     ->  Vector Streaming (type: GATHER) | 459.452           |      1 |      1 | 174KB          |         |      44 | 205693.85
  4 |       ->  Vector Aggregate              | [285.177,285.177] |      1 |      1 | [225KB, 225KB] |         |      44 | 205693.79
  5 |         ->  CStore Scan on public.lineitem | [249.757,249.757] | 114160 |  89475 | [792KB, 792KB] |         |      12 | 205246.40
(5 rows)
```

图11-18 使用 partial cluster key 后 CU 加载情况

```
  5 --CStore Scan on public.lineitem
        datanode1 (actual time=23.017..249.757 rows=114160 loops=1)
        datanode1 (RoughCheck CU: CUNone: 84, CUSome: 17)
        datanode1 (LLVM Optimized)
        datanode1 (Buffers: shared hit=2853 read=23)
        datanode1 (CPU: ex c/r=5673, ex cyc=647656146, inc cyc=647656146)
```

使用 partial cluster key 后，5-- CStore Scan on public.lineitem 的时间减少了 1.2s，得益于有 84 个 CU 被过滤掉了。

2. 选取 partial cluster key 列。

- 列存表支持创建 partial cluster key 的类型 character varying(n), varchar(n), character(n), char(n), text, nvarchar2, timestamp with time zone, timestamp without time zone, date, time without time zone, time with time zone。
- 数据的 distinct 值数量较少，这样能产生较好的过滤效果。
- 出现在查询 where 条件中，优先选取能过滤大量数据的列。
- partial cluster key 中设置多个列时，是先按第一个列排序，当第一个列值相同时，使用第二列比较，后续列依次类推。推荐不要超出 3 个列。

3. 添加 partial cluster key 后，优化导入性能。

   由于添加了 partial cluster key，在导入时会增加排序计算，会对导入性能产生影响。当排序可以完全在内存中进行时影响较小，如果无法在内存中完成排序时，会下盘写临时文件，这时就会产生较大的影响。

   排序使用的内存通过 GUC 参数 psort_work_mem 来设置，可以设置较大的值来使用更大的内存进行排序。

   排序的数据量是通过表的存储参数 PARTIAL_CLUSTER_ROWS 来设置，降低这个数值，可减少一次排序的数据量。这个参数通常与存储参数 MAX_BATCHROW 配置使用。PARTIAL_CLUSTER_ROWS 设置值必须是 MAX_BATCHROW 的整数倍，MAX_BATCHROW 是设置单个 CU 中数据的最大行数。

11.5 SQL 执行 troubleshooting

11.5.1 分析查询效率异常降低的问题

通常在几十毫秒内完成的查询，有时会突然需要几秒的时间完成；而通常需要几秒完成的查询，有时需要半小时才能完成。如何分析这种查询效率异常降低的问题呢？

处理步骤

通过下列的操作步骤，可以分析出查询效率异常降低的原因。

步骤 1 使用 analyze 命令分析数据库。

analyze 命令更新所有表中数据大小以及属性等相关统计信息，该命令较为轻量级，可以经常执行。如果此命令执行后性能恢复或者有所提升，则表明 autovacuum 未能很好的完成它的工作，有待进一步分析。

步骤 2 检查查询语句是否返回了多余的数据信息。

例如，如果查询语句先查询一个表中所有的记录，而只用到结果中的前 10 条记录。对于包含 50 条记录的表，查询起来是很快的；但是，当表中包含的记录达到 50000，查询效率将会有所下降。

若业务应用中存在只需要部分数据信息，但是查询语句却是返回所有信息的情况，建议修改查询语句，增加 LIMIT 子句来限制返回的记录数。这样至少使数据库优化器有了一定的优化空间，一定程度上会提升查询效率。

步骤 3 检查查询语句单独运行时是否仍然较慢。

尝试在数据库没有其他查询或查询较少的时候运行查询语句，并观察运行效率。如果效率较高，则说明可能是由于之前运行数据库系统的主机负载过大导致查询低效。此外，还可能是由于执行计划比较低效，但是由于主机硬件较快使得查询效率较高。

步骤 4 检查重复相同查询语句的执行效率。

查询效率低的一个重要原因是查询所需信息没有缓存在内存中，这可能是由于内存资源紧张，缓存信息被其他查询处理覆盖。

重复执行相同的查询语句，如果后续执行的查询语句效率提升，则可能是由于上述原因导致。

----结束

11.5.2 DROP TABLE 失败

问题现象

DROP TABLE 失败的两种现象：

- 在 gsql 客户端下使用\dt+命令查看数据库中无表 table_name；CREATE TABLE table_name 时报 table_name 已经存在的错误，使用 DROP TABLE table_name 失败，报不存在该表的错误，导致无法再次创建 table_name 表。
- 在 gsql 客户端下使用\dt+命令查看数据库中有表 table_name；使用 DROP TABLE table_name 失败，报不存在该表的错误，导致无法再次创建 table_name 表。

原因分析

导致该错误的原因有的节点上有该张表，有的节点上无该张表。

处理方法

当遇到上面两种现象时，使用 DROP TABLE table_name 失败，请再次使用 DROP TABLE if exists table_name 命令，使得 DROP 表可以成功。

11.5.3 不同用户查询同表显示数据不同

问题现象

2 个用户登录相同数据库 human_resource，分别执行的查询语句如下：select count(*) from areas，查询同一张表 areas 时，查询结果却不一致。

原因分析

请先判断同名的表是否确实是同一张表。在关系型数据库中，确定一张表通常需要 3 个因素：database，schema，table。从问题现象描述看，database，table 已经确定，分别是 human_resource、areas。接着，需要检查 schema。使用 dbadmin，user01 分别登录发现，search_path 依次是 public 和"$user"。dbadmin 作为集群管理员，默认不会创建 dbadmin 同名的 schema，即不指定 schema 的情况下所有表都会建在 public 下。而对于

普通用户如 user01，则会在创建用户时，默认创建同名的 schema，即不指定 schema 时表都会创建在 user01 的 schema 下。最终确定该案例发生时，确实因为 2 个用户之间交错对表进行操作，导致了同名不同表的情况。

处理方法

在操作表时加上 schema 引用，格式：schema.table。

11.5.4 业务运行时整数转换错误

问题现象

在转换整数时报出以下错误：

```
Invalid input syntax for integer: "13."
```

原因分析

有些数据类型不能转换成目标数据类型。

处理办法

逐步缩小 SQL 范围来确定。

11.5.5 SQL 语句出错自动重试

GaussDB(DWS)支持在 SQL 语句执行出错时的自动重试功能（下文简称 CN Retry）。对于来自 gsql 客户端、JDBC、ODBC 驱动的 SQL 语句，在 SQL 语句执行失败时，CN 端能够自动识别语句执行过程中的报错，并重新下发任务进行自动重试。

该功能的限制和约束如下：

- 功能范围限制：
  - 仅能提高故障发生时 SQL 语句执行成功率，不能保证 100%的执行成功。
  - CN Retry 默认开启，开启后 temp 表会记录日志，关闭 CN Retry 后，temp 表即不会记录日志，因此不能在使用 temp 表时反复打开/关闭 CN Retry 开关，否则主备切换后再 CN Retry 会造成数据不一致。
  - CN Retry 默认开启，开启后新创建的 unlogged 表会忽视 unlogged 关键字，建成普通表。关闭 CN Retry 后，unlogged 表不会记录日志，因此不能在使用 unlogged 表时反复打开/关闭 CN Retry 开关，否则主备切换后再 CN Retry 会造成数据不一致。
  - 在使用 gds 进行数据导出时，支持 CN Retry。现有机制导出时会对重复文件进行检测并删除相同的文件，因此建议不要对相同的外表重复导出数据，除非确定数据目录中相同文件名的文件需要删除。
- 错误类型约束：

  SQL 语句出错时能够被识别和重试的错误，仅限在错误类型列表（请参考表 11-4）中定义的错误。
- 语句类型约束：

支持单语句 CN Retry、存储过程、函数、匿名块。不支持事务块中的语句。

- 存储过程语句约束：
  - 包含 EXCEPTION 的存储过程，如果在执行过程中（包含语句块执行和 EXCEPTION 中的语句执行）错误被抛出，可以 retry，且系统内部错误发生时，retry 会先于 exception 被执行，而如果报错被 EXCEPTION 捕获则不能 retry。
  - 不支持使用全局变量的 package。
  - 不支持 DBMS_JOB。
  - 不支持 UTL_FILE。
  - 如果存储过程中有输出打印信息（如 dbms_output.put_line 或 raise info 等），在发生 retry 时会重复输出已打印的消息，并会在重复消息前输出“Notice：Retry triggered, some message may be duplicated.”加以提示。
- 集群状态约束：
  - 仅支持 DN、GTM 实例故障。
  - CN Retry 有次数限制，如果在 CN Retry 达到最大尝试次数（最大次数由 max_query_retry_times 控制）之前，集群状态无法从故障状态恢复到正常状态，CN Retry 不能保证执行成功。
  - 扩容时不支持 CN Retry。
- 数据导入约束：
  - 不支持 COPY FROM STDIN 语句。
  - 不支持 gsql \copy from 元命令。
  - 不支持 JDBC CopyManager copyIn 导入数据。

CN Retry 支持的错误类型列表和对应的错误码信息见表 11-4， 可以通过 GUC 参数 retry_ecode_list 设置 CN Retry 支持的错误类型列表，但不建议用户直接修改该参数，如有修改需求请联系技术工程师协助处理。

表11-4 CN Retry 支持的错误类型列表

错误类型	错误码	备注
对端连接重置（CONNECTION_RESET_BY_PEER）	YY001	TCP 通信错误：Connection reset by peer（CN 和 DN 间通信）
对端流重置（STREAM_CONNECTION_RESET_BY_PEER）	YY002	TCP 通信错误：Stream connection reset by peer（DN 和 DN 间通信）
锁等待超时（LOCK_WAIT_TIMEOUT）	YY003	锁超时，Lock wait timeout
连接超时（CONNECTION_TIMED_OUT）	YY004	TCP 通信错误，Connection timed out
查询设置错误	YY00	SET 命令发送失败，Set query

错误类型	错误码	备注
（SET_QUERY_ERROR）	5	
超出逻辑内存（OUT_OF_LOGICAL_MEMORY）	YY006	内存申请失败，Out of logical memory
通信库内存分配（SCTP_MEMORY_ALLOC）	YY007	SCTP 通信错误，Memory allocate error
无通信库缓存数据（SCTP_NO_DATA_IN_BUFFER）	YY008	SCTP 通信错误，SCTP no data in buffer
通信库释放内存关闭（SCTP_RELEASE_MEMORY_CLOSE）	YY009	SCTP 通信错误，Release memory close
SCTP、TCP 断开（SCTP_TCP_DISCONNECT）	YY010	SCTP 通信错误，TCP disconnect
通信库断开（SCTP_DISCONNECT）	YY011	SCTP 通信错误，SCTP disconnect
通信库远程关闭（SCTP_REMOTE_CLOSE）	YY012	SCTP 通信错误，Stream closed by remote
等待未知通信库通信（SCTP_WAIT_POLL_UNKNOW）	YY013	等待未知通信库通信，SCTP wait poll unknow
无效快照（SNAPSHOT_INVALID）	YY014	快照非法，Snapshot invalid
通讯接收信息错误（ERRCODE_CONNECTION_RECEIVE_WRONG）	YY015	连接获取错误，Connection receive wrong
内存耗尽（OUT_OF_MEMORY）	53200	内存耗尽，Out of memory
连接失败（CONNECTION_FAILURE）	08006	GTM 出错，Connection failure
连接异常（CONNECTION_EXCEPTION）	08000	连接出现错误，和 DN 的通讯失败，Connection exception
管理员关闭系统（ADMIN_SHUTDOWN）	57P01	管理员关闭系统，Admin shutdown
关闭远程流接口（STREAM_REMOTE_CLOSE_SOCKET）	XX003	关闭远程套接字，Stream remote close socket
重复查询编号（ERRCODE_STREAM_DUPLICATE	XX009	重复查询，Duplicate query id

错误类型	错误码	备注
_QUERY_ID）		
stream 查询并发更新同一行（ERRCODE_STREAM_CONCURRENT_UPDATE）	YY016	stream 查询并发更新同一行，Stream concurrent update
LLVM 内存分配错误（ERRCODE_LLVM_BAD_ALLOC_ERROR ）	CG003	内存分配错误， Allocate error
LLVM 致命错误（ERRCODE_LLVM_FATAL_ERROR）	CG004	致命错误，Fatal error
HashJoin 临时文件读取错误(ERRCODE_HASHJOIN_TEMP_FILE_ERROR)	F0011	临时文件读取错误，File error
分区个数发生变化(ERRCODE_PARTITION_NUM_CHANGED)	45003	在扫描 LIST 分区表时，发现此时的分区个数和优化阶段的分区个数不一致，一般出现在查询和 ADD/DROP 分区并发时。（此错误类型仅 8.1.3 及以上集群版本支持）

开启 CN Retry 功能需要设置如下 GUC 参数：

- 必选的 GUC 参数（CN 和 DN 都需设置）
  max_query_retry_times

⚠ 注意

CN Retry 功能开启时会为临时表数据记录日志，为保证数据一致性，在使用临时表时不能切换 CN Retry 开关状态，保持使用临时表的会话中 CN Retry 开关始终处于打开状态或者关闭状态。

- 可选的 GUC 参数
  cn_send_buffer_size
  max_cn_temp_file_size

11.6 常见性能参数调优设计

在使用数据库的时候，为了提高集群的性能，有多种方式去调优，从硬件配置到软件驱动升级，再到数据库的内部参数调整。本章节旨在介绍一些常用参数以及推荐配置。

1. query_dop ：用户自定义的查询并行度

   SMP 架构是一种利用富余资源来换取时间的方案，计划并行之后必定会引起资源消耗的增加，包括 CPU、内存、I/O 和网络带宽等资源的消耗都会出现明显的增长，而且随着并行度的增大，资源消耗也随之增大。

   - 当资源成为瓶颈的情况下，SMP 无法提升性能，反而可能导致性能的劣化。在出现资源瓶颈的情况下，建议关闭 SMP。
   - 当资源许可的情况下，并行度越高，性能提升效果越好。

   SMP 并行度支持会话级设置，推荐在执行符合要求的查询前，打开 SMP，执行结束后，关闭 SMP。以免在业务峰值时，对业务造成冲击。

   query_dop 的默认值是 1， 可通过 set query_dop=10，在会话中打开 SMP。

2. enable_dynamic_workload：动态负载管理

   动态负载管理指数据库内部根据用户负载情况，自动对复杂查询进行队列控制，不再需要手动设置参数，做到系统参数免调优。

   该参数默认打开，需要注意以下几点。

   - 集群有一个 CN 会作为中心协调节点（CCN），用于收集和调度作业执行，该节点可以通过 gs_om -t status --detail 查询到，Central Coordinator State 会显示其状态。当 CCN 不存在时，作业不再受动态负载管理控制。
   - 简单查询作业（估算值<32MB）、非 DML（即非 INSERT、UPDATE、DELETE 和 SELECT）语句，不走自适应负载，需要通过 max_active_statements 来进行单 CN 的上限控制。
   - 默认 work_mem 数值为 64MB，在自适应负载特性下，该数值不能变大，否则会引起内存不受控（例如未做 Analyze 的语句）。
   - 以下场景或语句由于内存使用特殊性和不确定性，可能导致大并发场景内存不受控，如果遇到需要降低并发数。
     - 单条元组占用内存过大的场景，例如，基表包含超过 MB 级别的宽列。
     - 完全下推语句的查询。
     - 需要在 CN 上耗费大量内存的语句，例如，不能下推的语句，withhold cursor 场景。
     - 由于计划生成不当导致 hashjoin 算子建立的 hash 表重复值过多，占用大量内存。
     - 包含 UDF 的场景，且 UDF 中使用大量内存的场景。

   该参数可配合 query_dop=0 使用，当 query_dop 设置为 0（自适应），系统会根据资源情况和计划特征，动态为每个查询选取[1,8]之间的最优的并行度。
   enable_dynamic_workload 参数会动态分配内存。

3. max_active_statements

   设置全局的最大并发数量。此参数只应用到 CN，且针对一个 CN 上的执行作业。

需根据系统资源（如 CPU 资源、IO 资源和内存资源）情况，调整此数值大小，使得系统支持最大限度的并发作业，且防止并发执行作业过多，引起系统崩溃。

- 当取值-1 或者 0 时，不限制全局并发数。
- 在点查询的场景下，参数建议设置为 100。
- 在分析类查询的场景下，参数的值设置为 CPU 的核数除以 DN 个数，一般可以设置 5~8 个。

4. session_timeout

   缺省情况下，客户端连接数据库后处于空闲状态时会根据参数的默认值自动断开连接。

   建议设成 0，表示关闭超时设置，以防止超时断连。

5. 影响数据库内存的五大参数:

   max_process_memory、shared_buffers、cstore_buffers、work_mem 和 maintenance_work_mem。

   - max_process_memory

     max_process_memory 是逻辑内存管理参数，主要功能是控制单个 CN/DN 上可用内存的最大峰值。

     计算公式：max_process_memory=物理内存*0.665/(1+主 DN 个数)。

   - shared_buffers

     设置 DWS 使用的共享内存大小。增加此参数的值会使 DWS 比系统默认设置需要更多的 System V 共享内存。

     建议设置 shared_buffers 值为内存的 40%以内。主要用于行存表 scan。计算公式：shared_buffers=(单服务器内存/单服务器 DN 个数)*0.4*0.25

   - cstore_buffers

     设置列存和 OBS、HDFS 外表列存格式（orc、parquet、carbondata）所使用的共享缓冲区的大小。

     计算公式可参考 shared_buffers。

   - work_mem

     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。

     计算公式：

     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。

   - maintenance_work_mem

maintenance_work_mem 用来设置维护性操作（比如 VACUUM、CREATE INDEX、ALTER TABLE ADD FOREIGN KEY 等）中可使用的最大的内存。

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。

当自动清理进程运行时，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。

6. bulk_write_ring_size

   数据并行导入使用的环形缓冲区大小。

   该参数主要影响入库性能，建议导入压力大的场景增加 DN 上的该参数配置。

7. 连接两大参数:

   max_connections 和 max_prepared_transactions

   - max_connections

     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。

     设置建议：

     CN 中此参数建议保持默认值。DN 中此参数建议设置为 CN 的个数乘以 CN 中此参数的值。

     增大这个参数可能导致 GaussDB(DWS)要求更多的 System V 共享内存或者信号量，可能超过操作系统缺省配置的最大值。这种情况下，请酌情对数值加以调整。

   - max_prepared_transactions

     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(DWS)比系统默认设置需要更多的 System V 共享内存。

须知

max_connections 取值的设置受 max_prepared_transactions 的影响，在设置 max_connections 之前，应确保 max_prepared_transactions 的值大于或等于 max_connections 的值，这样可确保每个会话都有一个等待中的预备事务。

8. checkpoint_completion_target

   指定检查点完成的目标。

   含义是每个 checkpoint 需要在 checkpoints 间隔时间的 50%内完成。

   默认值为 0.5，为提高性能可改成 0.9。

9. data_replicate_buffer_size

   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。

   默认值为 128MB，若服务器内存为 256G，可适当增大到 512MB。

10. wal_receiver_buffer_size

    备机与从备接收 Xlog 存放到内存缓冲区的大小。

默认值为 64MB，若服务器内存为 256G，可适当增大到 128MB。

12 用户自定义函数

> **说明**
>
> 实时数仓（单机部署）暂不支持用户自定义函数。

12.1 PL/Java 语言函数

使用 GaussDB(DWS)数据库的 PL/Java 函数，用户可以使用自己喜欢的 Java IDE 编写 Java 方法，并将包含这些方法的 jar 文件安装到 GaussDB(DWS)数据库中，然后使用该方法。GaussDB(DWS) PL/Java 基于开源 PL/Java 1.5.5 开发，所使用的 JRE 版本为 1.8.0_322。

使用限制

Java UDF 可以实现一些 java 逻辑计算，强烈建议不要在 Java UDF 中封装业务

- 强烈建议不要在 Java 函数中使用任何方式连接数据库，包括但不限于 JDBC。
- 暂不支持的数据类型：除表 12-1 内容之外的数据类型，包括自定义类型，复杂数据类型（Java Array 类及派生类）。
- 暂不支持 UDAF，UDTF。

示例

使用 PL/Java 函数时，需要首先将 Java 方法的实现打包为 jar 包并且部署到数据库中，然后使用数据库管理员账号创建函数，考虑兼容性问题，请使用 1.8.0_322 版本的 JRE 进行编译。

步骤 1 **编译 jar 包**。

Java 方法的实现和出包可以借助 IDE 来实现，以下是一个通过命令行来进行编译和出包的简单的示例，通过这个简单示例可以创建出一个包含单个方法的 jar 包文件。

首先，编写一个 Example.java 文件，在此文件中实现子字符串大写转换的方法，本例中类名为 Example，方法名为 upperString，内容如下：

```
public class Example
{
    public static String upperString (String text, int beginIndex, int endIndex)
    {
        return text.substring(beginIndex, endIndex).toUpperCase();
    }
}
```

然后，创建 manifest.txt 清单文件，文件内容如下：

```
Manifest-Version: 1.0
Main-Class: Example
Specification-Title: "Example"
Specification-Version: "1.0"
Created-By: 1.6.0_35-b10-428-11M3811
Build-Date: 08/14/2018 10:09 AM
```

其中，Manifest-Version 定义了 manifest 文件的版本，Main-Class 定义了 jar 文件的入口类，Specification-Title 和 Specification-Version 属于包的扩展属性，Specification-Title 定义了扩展规范的标题，Specification-Version 定义了扩展规范的版本，Created-By 声明了该文件的生成者，Build-Date 声明了该文件构建日期。

最后，编译 java 文件并打包得到 javaudf-example.jar

```
javac Example.java
jar cfm javaudf-example.jar manifest.txt Example.class
```

须知

jar 包的命名规则应符合 JDK 命名要求，如果含有非法字符，在部署或者使用函数时将出错。

步骤 2 **部署 jar 包**。

Jar 包首先需要放置到 OBS 服务器中，放置方法具体请参见*《对象存储服务控制台指南》*的上传文件章节。接着创建访问密钥 AK/SK，获取访问密钥的具体步骤，请参见《数据仓库服务用户指南》中的“创建访问密钥（AK 和 SK）”章节。登录数据库运行 gs_extend_library 函数，将文件导入到 GaussDB(DWS)中：

```
SELECT gs_extend_library('addjar', 'obs://bucket/path/javaudf-example.jar
accesskey=access_key_value_to_be_replaced
secretkey=secret_access_key_value_to_be_replaced  region=region_name
libraryname=example');
```

gs_extend_library 函数如何使用请参见 管理 jar 包和文件。函数中的 AK/SK 值，请用户根据实际获取值替换。region_name 请用户根据实际所在的区域名称替换。

步骤 3 **使用 PL/Java 函数**。

首先，使用拥有 sysadmin 权限的数据库用户（例如：dbadmin）登录数据库并创建 java_upperstring 函数如下：

```
CREATE FUNCTION java_upperstring(VARCHAR, INTEGER, INTEGER)
    RETURNS VARCHAR
```

```
   AS 'Example.upperString'
LANGUAGE JAVA;
```

说明

- 函数 java_upperstring 中定义的数据类型为 GaussDB(DWS)的数据类型。该数据类型需要和步骤 1 中 java 定义的方法 upperString 中数据类型一一对应。GaussDB(DWS)与 Java 数据类型的对应关系，请参见表 12-1。
- AS 子句用于指定该函数所调用的 Java 方法的类名和 static 方法名，格式为“类名.方法名”。该字段需要和步骤 1 中 java 定义的类名和方法名一致。
- 使用 PL/Java 函数时，LANGUAGE 字段应指定为 JAVA。
- CREATE FUNCTION 更多说明，请参见创建函数。

然后，执行 java_upperstring 函数：

```
SELECT java_upperstring('test', 0, 1);
```

得到预期结果为：

```
 java_upperstring
----------------------
 T
(1 row)
```

步骤 4 授权普通用户使用 PL/Java 函数。

创建普通用户，名称为 udf_user。

```
CREATE USER udf_user PASSWORD 'password';
```

授权普通用户 udf_user 对 java_upperstring 函数的使用权限。注意，此处需要把函数所在模式和函数的使用权限同时赋予给用户，用户才可以使用此函数。

```
GRANT ALL PRIVILEGES ON SCHEMA public TO udf_user;
GRANT ALL PRIVILEGES ON FUNCTION java_upperstring(VARCHAR, INTEGER, INTEGER) TO
udf_user;
```

以普通用户 udf_user 登录数据库。

```
SET SESSION SESSION AUTHORIZATION udf_user PASSWORD 'password';
```

执行 java_upperstring 函数：

```
SELECT public.java_upperstring('test', 0, 1);
```

得到预期结果为：

```
 java_upperstring
----------------------
 T
(1 row)
```

步骤 5 删除函数。

如果不再使用该函数可以进行删除：

```
DROP FUNCTION java_upperstring;
```

步骤 6 **卸载 jar 包**。

使用 gs_extend_library 函数卸载 jar 包：

```
SELECT gs_extend_library('rmjar', 'libraryname=example');
```

----结束

SQL 定义与使用

- **管理 jar 包和文件**

  拥有 sysadmin 权限的数据库用户可以使用 gs_extend_library 函数来部署、查看和删除数据库中的 jar 包，函数语法如下：

  ```
  SELECT gs_extend_library('[action]', '[operation]');
  ```

  说明

  - **action** 表明操作动作，值可以为：
  - ls 表示查看数据库中 jar 包，会对各节点的文件进行 MD5 值一致性检查。
  - addjar 表示将 OBS 服务器中的 jar 包部署到数据库中。
  - rmjar 表示将数据库中的 jar 包删除。
  - **operation** 表明操作字符串，格式为：

    obs://[bucket]/[source_filepath] accesskey=[accesskey] secretkey=[secretkey] region=[region] libraryname=[libraryname]
  - bucket: OBS 文件所属桶名，不可缺省。
  - source_filepath: OBS 服务器上的文件路径，仅支持 jar 文件。
  - accesskey: obs 服务获得的 accesskey，不可缺省。
  - secret_key: obs 服务获得的 secretkey，不可缺省。
  - region: 自定义函数 jar 包所存放的 OBS 桶所属的区域，不可缺省。
  - libraryname: 自定义库名，此自定义命名用于 GaussDB(DWS)内对 jar 文件的调用。当 action 为 addjar 和 rmjar 时，该参数不可缺省，当 action 为 ls 时，该参数可以缺省。注意，自定义库名不允许含有/|;&$<>\'{}"()[]~*?!等字符。

- **创建函数**

  PL/Java 函数通过 CREATE FUNCTION 语法创建，并且定义为 LANGUAGE JAVA，且包含 RETURNS 和 AS 子句。

  - CREATE FUNCTION 时指定所创建函数的名称，以及参数类型；
  - RETURNS 子句用于指定该函数的返回类型；
  - AS 子句与用于指定该函数所调用的 Java 方法的类名和 static 方法名，如果需要向 Java 方法传递 NULL 值作为入参，还需要指定该参数类型所对应的 Java 封装类名 （详见 NULL 值处理）。
  - 更多语法说明，请参见 CREATE FUNCTION。

    ```
    CREATE [ OR REPLACE ] FUNCTION function_name
    ( [ { argname [ argmode ] argtype [ { DEFAULT | := | = } expression ]}
    [, …] ])
    [ RETURNS rettype [ DETERMINISTIC ] ]
    ```

```
LANGAUGE JAVA
[
    { IMMUTABLE | STATBLE | VOLATILE }
    | [ NOT ] LEAKPROOF
    | WINDOW
    | { CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT |STRICT }
    | {[ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER |
AUTHID DEFINER | AUTHID CURRENT_USER}
    | { FENCED }
    | COST execution_cost
    | ROWS result_rows
    | SET configuration_parameter { {TO |=} value | FROM CURRENT}
] […]
{
    AS 'class_name.method_name' ( { argtype } [, …] )
}
```

- **使用函数**

  执行时，PL/Java 会根据 jar 包名的字母序列，在所有部署的 jar 包中寻找函数指定的 Java 类，并调用首次找到的类中函数所指定的 Java 方法，并返回调用结果。

- **删除函数**

  PL/Java 函数通过 DROP FUNCTION 语法删除函数。更多语法说明，请参见 DROP FUNCTION。

  ```
  DROP FUNCTION [ IF EXISTS ] function name [ ( [ {[ argmode ] [ argname ]
  argtype} [, ...] ] ) [ CASCADE | RESTRICT ] ];
  ```

  需要注意的是，如果所删除的函数为重载函数（详见重载函数），则删除时需要指明该函数的参数类型，如为非重载函数，则可直接指定函数名进行删除。

- **函数授权**

  非 sysadmin 户无法创建 PL/Java 函数，sysadmin 用户可以赋予其他类型用户使用函数的权限。更多语法说明，请参见 GRANT。

  ```
  GRANT { EXECUTE | 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 ];
  ```

基本数据类型映射关系

表12-1 **PL/Java 默认数据类型映射关系**

GaussDB(DWS)	**Java**
BOOLEAN	boolean
"char"	byte
bytea	byte[]
SMALLINT	short
INTEGER	int

GaussDB(DWS)	**Java**
BIGINT	long
FLOAT4	float
FLOAT8	double
CHAR	java.lang.String
VARCHAR	java.lang.String
TEXT	java.lang.String
name	java.lang.String
DATE	java.sql.Timestamp
TIME	java.sql.Time (stored value treated as local time)
TIMETZ	java.sql.Time
TIMESTAMP	java.sql.Timestamp
TIMESTAMPTZ	java.sql.Timestamp

数组类型处理

GaussDB(DWS)支持基础数组类型的转换，只需要在创建函数时在数据类型后追加 [] 即可，例如：

```
CREATE FUNCTION java_arrayLength(INTEGER[])
   RETURNS INTEGER
   AS 'Example.getArrayLength'
LANGUAGE JAVA;
```

Java 代码类似于：

```
public class Example
{
   public static int getArrayLength(Integer[] intArray)
   {
      return intArray.length;
   }
}
```

那么下面的调用的语句后：

```
SELECT java_arrayLength(ARRAY[1, 2, 3]);
```

得到预期结果应该如下所示：

```
java_arrayLength
---------------------
3
(1 row)
```

NULL 值处理

对于默认与 Java 的简单类型进行映射转换的那些 GaussDB(DWS)数据类型，是无法处理 NULL 值的，如果希望在 Java 方法里能够获得并处理从 GaussDB(DWS)中传入的 NULL 值，可以使用 Java 的封装类，并通过以下方式在 AS 子句中指定该 Java 封装类：

```
CREATE FUNCTION java_countnulls(INTEGER[])
   RETURNS INTEGER
   AS 'Example.countNulls(java.lang.Integer[])'
LANGUAGE JAVA;
```

Java 代码类似于：

```
public class Example
{
   public static int countNulls(Integer[] intArray)
   {
      int nullCount = 0;
      for (int idx = 0; idx < intArray.length; ++idx)
      {
         if (intArray[idx] == null)
         nullCount++;
      }
      return nullCount;
   }
}
```

那么下面的调用的语句后：

```
SELECT java_countNulls(ARRAY[null, 1, null, 2, null]);
```

得到的预期结果应该如下所示：

```
java_countNulls
--------------------
3
(1 row)
```

重载函数

PL/Java 支持重载函数，因此可以创建同名函数，或者调用 Java 代码中的重载方法。步骤如下：

步骤 1 创建重载函数

例如，在 Java 中可以实现两个方法名相同，输入参数类型不同的方法 dummy(int) 和 dummy(String)

```
public class Example
{
   public static int dummy(int value)
   {
      return value*2;
   }
   public static String dummy(String value)
   {
```

```
        return value;
    }
}
```

并在 GaussDB(DWS)中创建两个同名函数分别指定为上述两个方法：

```
CREATE FUNCTION java_dummy(INTEGER)
    RETURNS INTEGER
    AS 'Example.dummy'
LANGUAGE JAVA;

CREATE FUNCTION java_dummy(VARCHAR)
    RETURNS VARCHAR
    AS 'Example.dummy'
LANGUAGE JAVA;
```

步骤 2 调用重载函数

在调用重载函数时，GaussDB(DWS)会根据输入的参数类型去调用匹配该类型的 Java 方法。因此上述两个函数的调用结果如下所示：

```
SELECT java_dummy(5);
 java_dummy
-----------------
          10
(1 row)

SELECT java_dummy('5');
 java_dummy
---------------
5
(1 row)
```

需要注意的是，由于 GaussDB(DWS)对数据类型存在隐式转换的情况，因此建议在调用重载函数时，指定输入参数的类型，例如：

```
SELECT java_dummy(5::varchar);
 java_dummy
----------------
5
(1 row)
```

此时会优先匹配所指定的参数类型，如果不存在指定参数类型的 Java 方法，则会对参数进行隐式转换匹配转换后的参数类型对应的 Java 方法。

```
SELECT java_dummy(5::INTEGER);
 java_dummy
-----------------
10
(1 row)

DROP FUNCTION java_dummy(INTEGER);

SELECT java_dummy(5::INTEGER);
 java_dummy
----------------
```

```
5
(1 row)
```

须知

隐式转换的数据类型包括：

- 可以默认转换为 INTEGER 类型的包括：SMALLINT
- 可以默认转换为 BIGINT 类型的包括：SMALLINT, INTEGER
- 可以默认转换为 BOOL 类型的包括：TINYINT, SMALLINT, INTEGER, BIGINT
- 可以默认转换为 TEXT 类型的包括：CHAR, NAME, BIGINT, INTEGER, SMALLINT, TINYINT, RAW, FLOAT4, FLOAT8, BPCHAR, VARCHAR, NVARCHAR2, DATE, TIMESTAMP, TIMESTAMPTZ, NUMERIC, SMALLDATETIME
- 可以默认转换为 VARCHAR 类型的包括：TEXT, CHAR, BIGINT, INTEGER, SMALLINT, TINYINT, RAW, FLOAT4, FLOAT8, BPCHAR, DATE, NVARCHAR2, TIMESTAMP, NUMERIC, SMALLDATETIME

步骤 3 **删除重载函数**

对于重载函数，删除时需要指定函数的参数类型，否则无法删除。

```
DROP FUNCTION java_dummy(INTEGER);
```

----结束

相关 GUC 参数

- **pljava_vmoptions**

  会话级别的 GUC 参数，该参数用于设置 JVM 的启动参数，例如：

  ```
  SET pljava_vmoptions='-Xmx64m -Xms2m -XX:MaxMetaspaceSize=8m';
  ```

  pljava_vmoptions 可接受的参数包括：

  - JDK8 JVM 启动参数
  - JDK8 JVM 系统属性参数（以 -D 开头，如： -Djava.ext.dirs）。

须知

不建议用户设置任何包含目录的参数，可能会导致不可预期的行为。

  - 用户自定义参数（以 -D 开头，如： -Duser.defined.option）

须知

如果所设置的参数不在上述范围内，视为非法参数，会在调用函数时报错。

```
SET pljava_vmoptions=' illegal.option';
SET
SELECT java_dummy(5::int);
ERROR:  UDF Error:cannot use PL/Java before successfully completing its
setup.Please check if your pljava_vmoption is set correctly,since we do not ignore
illegal parameters.Or check the log for more messages.
```

- **FencedUDFMemoryLimit**

  会话级别的 GUC 参数，用户限制会话发起的单个 Fenced UDF Worker 进程的最大虚拟内存使用量，设置方法如下：

  ```
  SET FencedUDFMemoryLimit='512MB';
  ```

  该参数的取值范围为 (150MB, 1G]，当设置大于 1G 时会立即报错，当设置小于等于 150MB 时，则会在调用函数时报错。

须知

- FencedUDFMemoryLimit 设置为 0，表示不控制 Fenced UDF Worker 的虚拟内存使用量。
- 建议通过设置 udf_memory_limit 控制 Fenced UDF Worker 使用的物理内存量。不建议用户使用 FencedUDFMemoryLimit，尤其在使用 Java UDF 时不建议用户设置此参数。但是如果用户非常清楚设置该参数带来的影响，可以参考下列信息进行设置：
- C UDF worker 启动之后，占用的虚拟内存约为 200MB，占用的物理内存约为 16MB。
- Java UDF worker 启动之后，占用的虚拟内存约为 2.5GB，占用的物理内存约为 50MB。

异常处理

如果在 JVM 中发生异常，PL/Java 的异常处理机制会将异常时 JVM 的堆栈信息输出到客户端。

日志

PL/Java 使用标准的 Java Logger。 因此，用户可以通过如下方式记录日志：

```
Logger.getAnonymousLogger().config( "Time is " + new
Date(System.currentTimeMillis()));
```

初始化的 Java Logger 类会默认设置为 CONFIG 级别，对应为 GaussDB(DWS)的 LOG 级别。Java Logger 类输出的日志消息都会重定向到 GaussDB(DWS)后端，并写入到服务器日志或显示在用户界面上。MPPDB 服务器日志将记录 LOG、WARNING、ERROR 级别的信息，而 SQL 用户界面将显示 WARNING 和 ERROR 级别的日志消息。Java Logger 级别与 GaussDB(DWS)的日志级别对应关系见下表。

表12-2 **PL/Java 日志级别**

java.util.logging.Level	GaussDB(DWS) 日志级别
SERVER	ERROR
WARINING	WARNING
CONFIG	LOG
INFO	INFO
FINE	DEBUG1
FINER	DEBUG2
FINEST	DEBUG3

用户可以通过以下方式更改 Java Logger 的记录级别。例如通过下面的 Java 代码修改 Java Logger 级别为 SEVERE，此时再记录 WARNING 级别的日志时，日志消息（msg）就不会再写入到 GaussDB(DWS)日志中。

```
Logger log = Logger.getAnonymousLogger();
Log.setLevel(Level.SEVERE);
log.log(Level.WARNING, msg);
```

安全问题

在 GaussDB(DWS)中，PL/Java 是一种 untrusted 语言，PL/Java 函数只能由数据库 sysadmin 用户进行创建，通过 GRANT 方式赋予其他用户使用权限（详见函数授权）。

同时 PL/Java 控制用户对文件系统的访问权限，不允许用户在 Java 方法中对大部分的系统文件进行读操作，不允许所有的写、删除和执行操作。

12.2 PL/pgSQL 语言函数

PL/pgSQL 类似于 Oracle 的 PL/SQL，是一种可载入的过程语言。

用 PL/pgSQL 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必须发送每一个查询到数据库服务器、等待它被处理、接收并处理结果、做一些计算，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，所有这些会引起进程间通信并且将带来网络负担。

通过 PL/pgSQL，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/pgSQL 可以使用 SQL 中所有的数据类型、操作符和函数。

应用 PL/pgSQL 创建函数的语法为 CREATE FUNCTION。正如前面所说，PL/pgSQL 类似于 Oracle 的 PL/SQL，是一种可载入的过程语言。其应用方法与 13 存储过程相似，只是存储过程无返回值，函数有返回值。

13 存储过程

13.1 存储过程

商业规则和业务逻辑可以通过程序存储在 GaussDB(DWS)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《SQL 语法参考》中 CREATE PROCEDURE。

12.2 PL/pgSQL 语言函数节所提到的 PL/pgSQL 语言创建的函数与存储过程的应用方法相通。下面各节中，除非特别声明，否则内容通用于存储过程和 PL/pgSQL 语言函数。

13.2 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(DWS)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(DWS)根据数据类型有相应函数对其内容进行操作，例如 GaussDB(DWS)可对数值型数据进行加、减、乘、除操作。

13.3 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(DWS)提供的类型转换函数，例如 CAST 进行数据类型强转。

GaussDB(DWS)数据库常见的隐式类型转换，请参见表 13-1。

须知

GaussDB(DWS)支持的 DATE 的效限范围是：公元前 4713 年到公元 294276 年。

表13-1 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	-

13.4 数组和 record

13.4.1 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 [NOT NULL];
```

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。
- NOT NULL：可选约束，可以约束该数组中的元素均不为 NULL。

说明

- 在 GaussDB(DWS)中，数组会自动增长，访问越界会返回一个 NULL，不会报错。越界写入数组会提示：Subscript outside of limit.
- 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(DWS)会优先选择存储过程中定义的数组类型来声明数组变量。

GaussDB(DWS) 8.1.0 之前版本， 由于数组可以自动增长，系统不会校验数组越界以及数组元素的长度限制。当前版本为了兼容 Oracle 的用法增加了相关约束。如果已经存在越界写入等场景，可通过在 behavior_compat_options 参数中配置 varray_verification，来兼容之前不校验的行为。

示例：

```
--演示在存储过程中对数组声明操作。
CREATE OR REPLACE PROCEDURE array_proc
AS
      TYPE ARRAY_INTEGER IS VARRAY(1024) OF INTEGER;--定义数组类型
      TYPE ARRAY_INTEGER_NOT_NULL IS VARRAY(1024) OF INTEGER NOT NULL;--定义非空数组类
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;
      DBMS_OUTPUT.PUT_LINE(ARRINT.COUNT);
      DBMS_OUTPUT.PUT_LINE(ARRINT(1));
      DBMS_OUTPUT.PUT_LINE(ARRINT(10));
      DBMS_OUTPUT.PUT_LINE(ARRINT(ARRINT.FIRST));
```

```
      DBMS_OUTPUT.PUT_LINE(ARRINT(ARRINT.last));
END;
/

--调用该存储过程。
CALL array_proc();
10
1
10
1
10
--删除存储过程。
DROP PROCEDURE array_proc;
```

ROWTYPE 类型数组的声明及使用

除了上述示例中普通数组以及 not null 数组的声明及使用，数组中同时支持 rowtype 类型数组的声明及使用。

示例：

```
--演示在存储过程中对数组 COUNT 函数的用法。
CREATE TABLE tbl (a int, b int);
INSERT INTO tbl VALUES(1, 2),(2, 3),(3, 4);
CREATE OR REPLACE PROCEDURE array_proc
AS
   CURSOR all_tbl IS SELECT * FROM tbl ORDER BY a;
   TYPE tbl_array_type IS varray(50) OF tbl%rowtype; --定义 rowtype 类型的数组，tbl 是任意表。
   tbl_array tbl_array_type;
   tbl_item tbl%rowtype;
   inx1 int;
BEGIN
   tbl_array := tbl_array_type();
   inx1 := 0;
   FOR tbl_item IN all_tbl LOOP
      inx1 := inx1 + 1;
      tbl_array(inx1) := tbl_item;
   END LOOP;
   WHILE inx1 IS NOT NULL LOOP
      DBMS_OUTPUT.PUT_LINE('tbl_array(inx1).a=' || tbl_array(inx1).a || ' tbl_array(inx1).b=' || tbl_array(inx1).b);
      inx1 := tbl_array.PRIOR(inx1);
   END LOOP;
END;
/
```

执行结果：

```
call array proc();
tbl array(inx1).a=3 tbl array(inx1).b=4
tbl array(inx1).a=2 tbl array(inx1).b=3
tbl_array(inx1).a=1 tbl_array(inx1).b=2
```

数组相关函数使用

在 GaussDB(DWS)中，提供了类似 Oracle 相关的数组函数的支持，可以通过以下函数获取数组的一些属性或者对数组内容进行操作。

COUNT

COUNT 函数可以返回当前数组元素的数量，仅统计初始化过的元素或者经过 EXTEND 函数扩展后的元素。

用法如下：

varray.COUNT 或 *varray*.COUNT()

示例：

```
--演示在存储过程中对数组 COUNT 函数的用法。
CREATE OR REPLACE PROCEDURE test_varray
AS
    TYPE varray type IS VARRAY(20) OF INT;
    v varray varray type;
BEGIN
    v varray := varray type(1, 2, 3);
    DBMS OUTPUT.PUT LINE('v varray.count=' || v varray.count);
    v_varray.extend;
    DBMS_OUTPUT.PUT_LINE('v_varray.count=' || v_varray.count);
END;
/
```

执行结果：

```
call test_varray();
v_varray.count=3
v_varray.count=4
```

FIRST 和 LAST

FIRST 函数可以返回第一个元素的下标。LAST 函数可以返回最后一个元素的下标。

用法如下：

varray.FIRST 或 *varray*.FIRST()

varray.LAST 或 *varray*.LAST()

示例：

```
--演示在存储过程中对数组 FIRST 和 LAST 函数的用法。
CREATE OR REPLACE PROCEDURE test_varray
AS
    TYPE varray_type IS VARRAY(20) OF INT;
    v_varray varray_type;
BEGIN
    v_varray := varray_type(1, 2, 3);
    DBMS_OUTPUT.PUT_LINE('v_varray.first=' || v_varray.first);
    DBMS_OUTPUT.PUT_LINE('v_varray.last=' || v_varray.last);
```

```
END;
/
```

执行结果：

```
call test_varray();
v_varray.first=1
v_varray.last=3
```

EXTEND

说明

EXTEND 函数主要是为了兼容 Oracle 的两种用法。在 GaussDB(DWS)中，数组会自动增长，EXTEND 函数不是必须的。如果是新写的存储过程，完全没有必要使用 EXTEND 函数。

EXTEND 函数可以对数组进行扩展，EXTEND 有两种调用方式。

- 方式一：

  EXTEND 包含一个整型入参，表示数组向后扩展 size 大小的长度，EXTEND 后 COUNT 和 LAST 函数的值也会有相应的变化。

  用法如下：

  varray.EXTEND(size)

  其中 *varray*.EXTEND 这种无参的调用默认会向后扩展 1 位等价于 *varray*.EXTEND(1)

- 方式二：

  EXTEND 包含两个整型入参，第一个参数代表向后扩展 size 大小的长度，第二个参数表示扩展后的数组元素值和之下标为 index 的元素相同。

  用法如下：

  varray.EXTEND(size, index)

示例：

```
--演示在存储过程中对数组 EXTEND 函数的用法。
CREATE OR REPLACE PROCEDURE test_varray
AS
   TYPE varray_type IS VARRAY(20) OF INT;
   v_varray varray_type;
BEGIN
   v_varray := varray_type(1, 2, 3);
   v_varray.extend(3);
   DBMS_OUTPUT.PUT_LINE('v_varray.count=' || v_varray.count);
   v_varray.extend(2,3);
   DBMS_OUTPUT.PUT_LINE('v_varray.count=' || v_varray.count);
   DBMS_OUTPUT.PUT_LINE('v_varray(7)=' || v_varray(7));
   DBMS_OUTPUT.PUT_LINE('v_varray(8)=' || v_varray(7));
END;
/
```

执行结果：

```
call test_varray();
v_varray.count=6
```

```
v_varray.count=8
v_varray(7)=3
v_varray(8)=3
```

NEXT 和 PRIOR

NEXT 函数和 PRIOR 函数主要用于数组的循环遍历中，NEXT 函数会根据入参 index 值，返回下一个数组元素的下标，若已经到达数组下标最大值则返回 NULL。PRIOR 函数会根据入参 index 值，返回上一个数组元素的下标，若已经到达数组下标最小值则返回 NULL。

用法如下：

varray.NEXT(index)

varray.PRIOR(index)

示例：

```
--演示在存储过程中对数组 NEXT 和 PRIOR 函数的用法。
CREATE OR REPLACE PROCEDURE test_varray
AS
   TYPE varray_type IS VARRAY(20) OF INT;
   v_varray varray_type;
   i int;
BEGIN
   v varray := varray type(1, 2, 3);

   i := v varray.COUNT;
   WHILE i IS NOT NULL LOOP
      DBMS_OUTPUT.PUT_LINE('test prior v_varray('||i||')=' || v_varray(i));
      i := v_varray.PRIOR(i);
   END LOOP;

   i := 1;
   WHILE i IS NOT NULL LOOP
      DBMS_OUTPUT.PUT_LINE('test next v_varray('||i||')=' || v_varray(i));
      i := v_varray.NEXT(i);
   END LOOP;
END;
/
```

执行结果：

```
call test_varray();
test prior v_varray(3)=3
test prior v_varray(2)=2
test prior v_varray(1)=1
test next v_varray(1)=1
test next v_varray(2)=2
test next v_varray(3)=3
```

EXISTS

EXISTS 函数可以判断数组下标是否存在。

用法如下：

varray.EXISTS(index)

示例：

```
--演示在存储过程中对数组 EXISTS 函数的用法。
CREATE OR REPLACE PROCEDURE test_varray
AS
    TYPE varray_type IS VARRAY(20) OF INT;
    v_varray varray_type;
BEGIN
    v_varray := varray_type(1, 2, 3);
    IF v_varray.EXISTS(1) THEN
        DBMS_OUTPUT.PUT_LINE('v_varray.EXISTS(1)');
    END IF;
    IF NOT v_varray.EXISTS(10) THEN
        DBMS_OUTPUT.PUT_LINE('NOT v_varray.EXISTS(10)');
    END IF;
END;
/
```

执行结果：

```
call test_varray();
v_varray.EXISTS(1)
NOT v_varray.EXISTS(10)
```

TRIM

TRIM 函数可以从数组尾部删除指定数量的元素。

用法如下：

varray.TRIM(size)

其中 *varray*.TRIM 这种无参的调用会默认入参为 1，等价于 *varray*.TRIM(1)

示例：

```
--演示在存储过程中对数组 TRIM 函数的用法。
CREATE OR REPLACE PROCEDURE test_varray
AS
    TYPE varray_type IS VARRAY(20) OF INT;
    v_varray varray_type;
BEGIN
    v_varray := varray_type(1, 2, 3, 4, 5);
    v_varray.trim(3);
    DBMS_OUTPUT.PUT_LINE('v_varray.count' || v_varray.count);
    v_varray.trim;
    DBMS_OUTPUT.PUT_LINE('v_varray.count:' || v_varray.count);
END;
/
```

执行结果：

```
call test_varray();
v_varray.count:2
v_varray.count:1
```

DELETE

DELETE 函数可以从数组删除数组中的所有元素。

用法如下：

varray.DELETE 或 *varray*.DELETE()

示例：

```
--演示在存储过程中对数组 DELETE 函数的用法。
CREATE OR REPLACE PROCEDURE test_varray
AS
    TYPE varray_type IS VARRAY(20) OF INT;
    v_varray varray_type;
BEGIN
    v_varray := varray_type(1, 2, 3, 4, 5);
    v_varray.delete;
    DBMS_OUTPUT.PUT_LINE('v_varray.count:' || v_varray.count);
END;
/
```

执行结果：

```
call test_varray();
v_varray.count:0
```

LIMIT

LIMIT 函数可以返回数组的最大长度限制。

用法如下：

varray.LIMIT 或 *varray*.LIMIT()

示例：

```
--演示在存储过程中对数组 LIMIT 函数的用法。
CREATE OR REPLACE PROCEDURE test varray
AS
    TYPE varray type IS VARRAY(20) OF INT;
    v varray varray type;
BEGIN
    v varray := varray type(1, 2, 3, 4, 5);
    DBMS OUTPUT.PUT LINE('v varray.limit:' || v varray.limit);
END;
/
```

执行结果：

```
call test_varray();
v_varray.limit:20
```

13.4.2 record

record 类型的变量

创建一个 record 变量的方式：

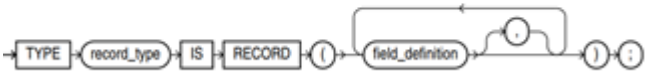
定义一个 record 类型 ，然后使用该类型来声明一个变量。

语法

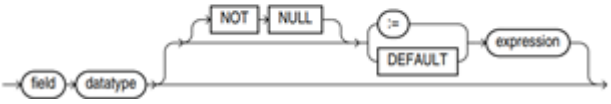
record 类型的语法参见图 13-1。

图13-1 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(DWS)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：

```
CREATE TABLE emp rec
(
    empno          numeric(4,0),
    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)
)
with (orientation = column,compression=middle)
distribute by hash (sal);
\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。
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。
CALL regress_record('abc');
INFO:  employer name: WARD , epno:18
INFO:  employer1 name: WARD , epno: 18
INFO:  employer1 name: <NULL> , epno: <NULL>
INFO:  employer2 name: SCOTT ,epno: 10
--删除存储过程。
DROP PROCEDURE regress_record;
```

13.5 声明语法

13.5.1 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

```
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 13-2。
- 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。

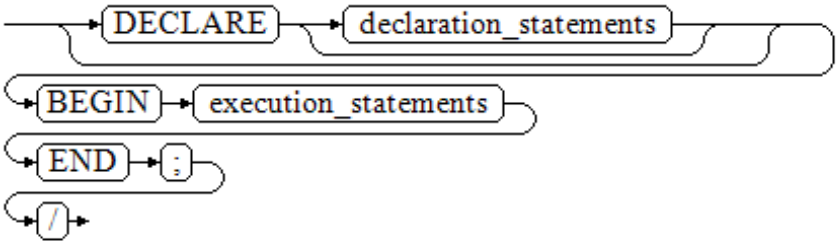
13.5.2 匿名块

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 13-2。

图13-2 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
BEGIN
    NULL;
END;
/

--将信息打印到控制台：
BEGIN
    dbms_output.put_line('hello world!');
END;
/

--将变量内容打印到控制台：
DECLARE
    my_var VARCHAR2(30);
BEGIN
    my_var :='world';
    dbms_output.put_line('hello'||my_var);
END;
/
```

13.5.3 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。

13.6 基本语句

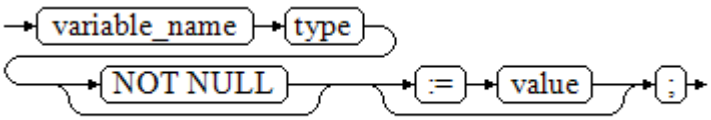
13.6.1 定义变量

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 13-3。

图13-3 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

```
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。假如想定义一个 my_name 变量，它的变量类型与 employee 的 firstname 类型相同，我们可以通过如下定义：

```
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，我们也不需要再次修改 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
```

须知

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

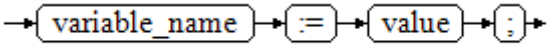
```
DECLARE
    emp_id  INTEGER :=7788;  --定义变量并赋值
    outer_var  INTEGER :=6688;  --定义变量并赋值
BEGIN
    DECLARE
        emp_id INTEGER :=7799;  --定义变量并赋值
        inner_var  INTEGER :=6688;  --定义变量并赋值
    BEGIN
        dbms_output.put_line('inner emp_id ='||emp_id);  --显示值为 7799
        dbms_output.put_line('outer_var ='||outer_var);  --引用外部块的变量
    END;
    dbms_output.put_line('outer emp_id ='||emp_id);  --显示值为 7788
END;
/
```

13.6.2 赋值语句

语法

给变量赋值的语法请参见图 13-4。

图13-4 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。

示例

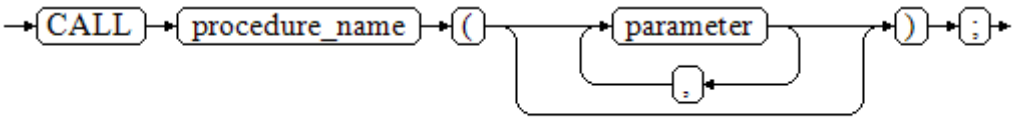
```
DECLARE
    emp_id  INTEGER := 7788;--赋值
BEGIN
    emp_id := 5;--赋值
    emp_id := 5*7784;
END;
/
```

13.6.3 调用语句

语法

调用一个语句的语法请参见图 13-5。

图13-5 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

```
--创建存储过程proc staffs
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 staffs where
section_id = section;
END;
/

--创建存储过程proc_return.
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
dbms_output.put_line(v_sum||'#'||v_num);
RETURN;  --返回语句
END;
/

--调用存储过程proc_return.
CALL proc_return();

--清除存储过程
DROP PROCEDURE proc_staffs;
DROP PROCEDURE proc_return;

--创建函数func_return.
CREATE OR REPLACE FUNCTION func_return returns void
language plpgsql
AS $$
DECLARE
v_num INTEGER := 1;
BEGIN
dbms_output.put_line(v_num);
RETURN;  --返回语句
END $$;


-- 调用函数func_return
CALL func_return();
1

-- 清除函数
DROP FUNCTION func_return;
```

13.7 动态语句

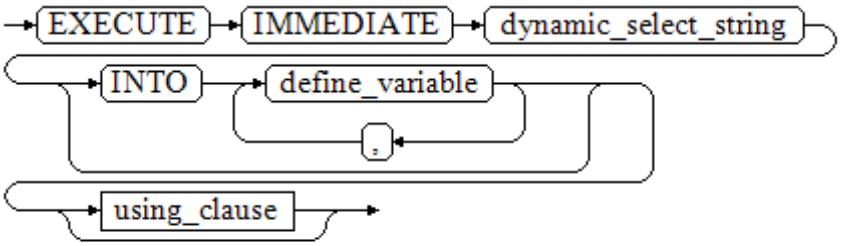
13.7.1 执行动态查询语句

介绍执行动态查询语句。GaussDB(DWS)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

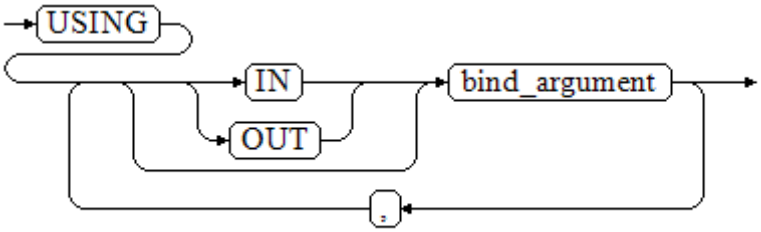
语法图请参见图 13-6。

图13-6 EXECUTE IMMEDIATE dynamic_select_clause::=



using_clause 子句的语法图参见图 13-7。

图13-7 using_clause-1



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

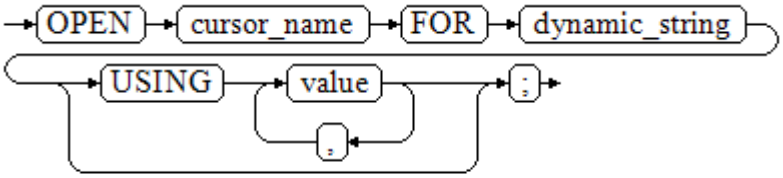
```
--从动态语句检索值（INTO 子句）：
DECLARE
   staff_count  VARCHAR2(20);
BEGIN
   EXECUTE IMMEDIATE 'select count(*) from staffs'
      INTO staff_count;
   dbms_output.put_line(staff_count);
END;
/

--传递并检索值（INTO 子句用在 USING 子句前）：
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 staffs where staff_id = :1'
      INTO first_name, salary
      USING IN staff_id;
   dbms_output.put_line(first_name || ' ' || salary);
END;
/

--调用存储过程
CALL dynamic_proc();

--删除存储过程
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 13-8。

图13-8 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 13.10 游标。

示例

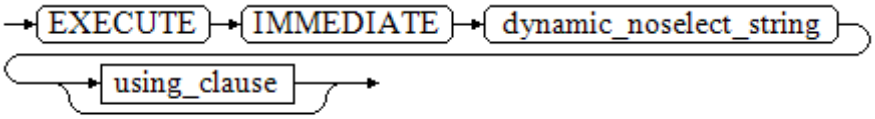
```
DECLARE
    name         VARCHAR2(20);
    phone_number  VARCHAR2(20);
    salary       NUMBER(8,2);
    sqlstr       VARCHAR2(1024);

    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 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
         dbms_output.put_line(name||'#'||phone_number||'#'||salary);
         FETCH my_cur INTO name, phone_number, salary;
    END LOOP;
    CLOSE my_cur;   --关闭游标
END;
/
```

13.7.2 执行动态非查询语句

语法

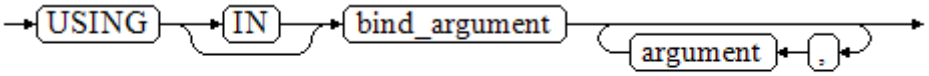
语法请参见图 13-9。

图13-9 noselect::=



using_clause 子句的语法参见图 13-10。

图13-10 using_clause-2



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
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
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
SELECT * FROM sections_t1;

--删除表
DROP TABLE sections_t1;
```

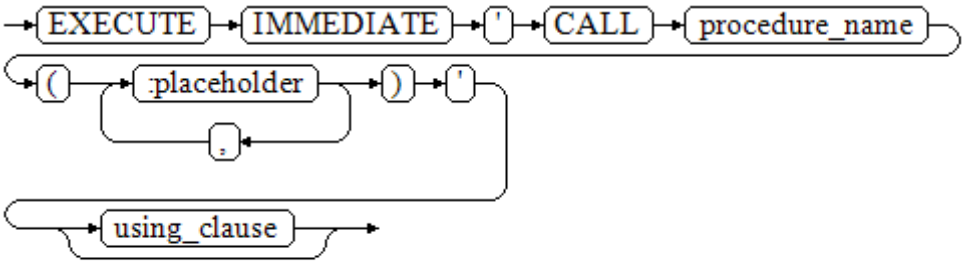
13.7.3 动态调用存储过程

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

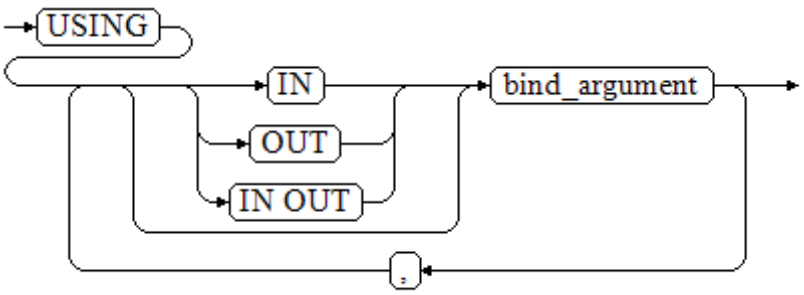
语法请参见图 13-11。

图13-11 call_procedure::=



using_clause 子句的语法参见图 13-12。

图13-12 using_clause-3



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。
CREATE OR REPLACE PROCEDURE proc_add
(
   param1   in   INTEGER,
   param2   out  INTEGER,
   param3   in   INTEGER
)
AS
BEGIN
  param2:= param1 + param3;
END;
/

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;
   dbms_output.put_line('result is: '||to_char(param2));
END;
/

--删除存储过程
DROP PROCEDURE proc_add;
```

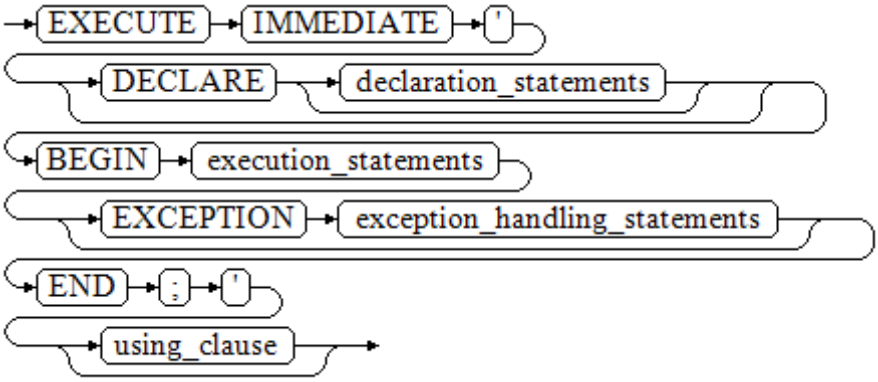
13.7.4 动态调用匿名块

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

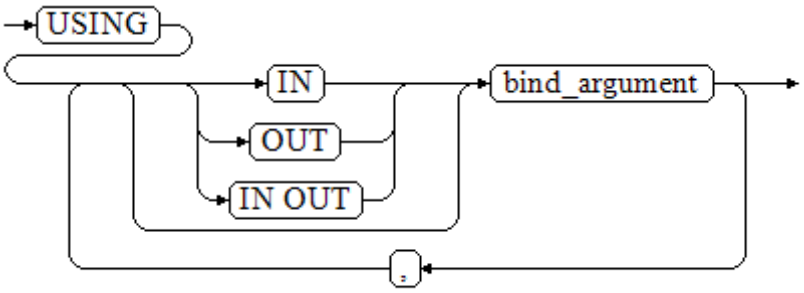
语法请参见图 13-13。

图13-13 call_anonymous_block::=



using_clause 子句的语法参见图 13-14。

图13-14 using_clause-4



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(DWS)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。

示例

```
--创建存储过程 dynamic proc
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 staffs where staff_id= :dno; end;'
       USING OUT first_name, OUT salary, IN staff_id;
   dbms_output.put_line(first_name|| ' ' || salary);
END;
/

--调用存储过程
CALL dynamic_proc();

--删除存储过程
DROP PROCEDURE dynamic_proc;
```

13.8 控制语句

13.8.1 返回语句

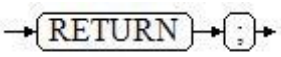
GaussDB(DWS)提供两种方式返回数据：RETURN 或 RETURN NEXT 及 RETURN QUERY。其中，RETURN NEXT 和 RETURN QUERY 只适用于函数，不适用存储过程。

13.8.1.1 RETURN

语法

返回语句的语法请参见图 13-15。

图13-15 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

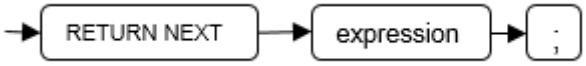
请参见调用语句的示例。

13.8.1.2 RETURN NEXT 及 RETURN QUERY

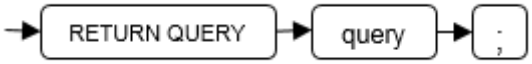
语法

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

```
CREATE TABLE t1(a int);
INSERT INTO t1 VALUES(1),(10);

--RETURN NEXT
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;
call fun_for_return_next();
 a
---
 1
 10
(2 rows)

-- RETURN QUERY
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;
call fun_for_return_next();
 a
---
```

```
1
10
(2 rows)
```

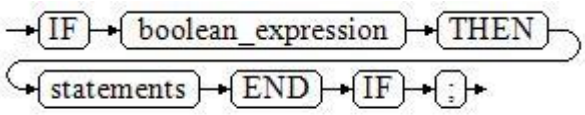
13.8.2 条件语句

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(DWS)有五种形式的 IF：

- IF_THEN

  图13-16 IF_THEN::=

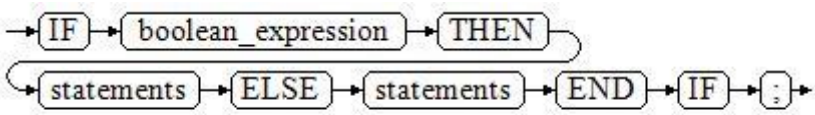
  IF_THEN 语句是 IF 的最简单形式。如果条件为真，statements 将被执行。否则，将忽略它们的结果使该 IF_THEN 语句执行结束。

  示例

  ```
  IF v_user_id <> 0 THEN
     UPDATE users SET email = v_email WHERE user_id = v_user_id;
  END IF;
  ```

- IF_THEN_ELSE

  图13-17 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

  ```
  IF parentid IS NULL OR parentid = ''
  THEN
     RETURN;
  ELSE
     hp_true_filename(parentid);--表示调用存储过程
  END IF;
  ```

- IF_THEN_ELSE IF

  IF 语句可以嵌套，嵌套方式如下：

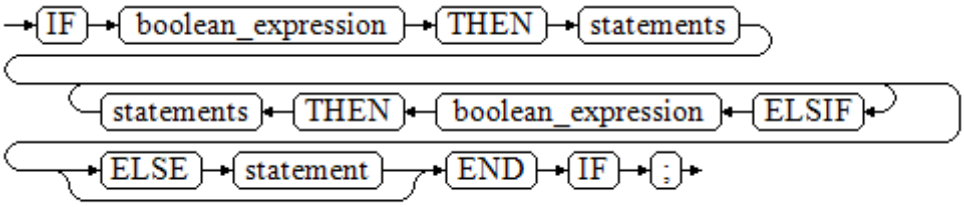
  ```
  IF sex = 'm' THEN
     pretty_sex := 'man';
  ```

```
ELSE
    IF sex = 'f' THEN
       pretty_sex := 'woman';
    END IF;
END IF;
```

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13-18 IF_THEN_ELSIF_ELSE::=

  示例

```
IF number_tmp = 0 THEN
    result := 'zero';
ELSIF number_tmp > 0 THEN
    result := 'positive';
ELSIF number_tmp < 0 THEN
    result := 'negative';
ELSE
    result := 'NULL';
END IF;
```

- 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);
```

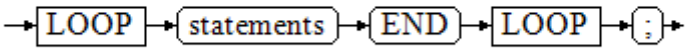
```
--删除存储过程
DROP PROCEDURE proc_control_structure;
```

13.8.3 循环语句

简单 LOOP 语句

语法图

图13-19 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);
```

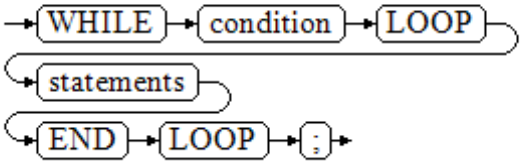
须知

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13-20 while_loop::=



只要条件表达式为真，WHILE 语句就会不停的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。

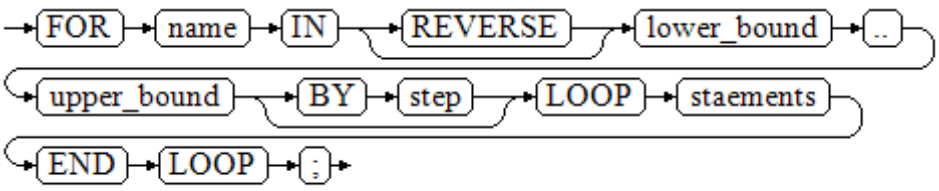
示例

```
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13-21 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。

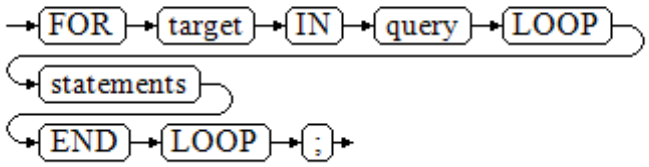
示例

```
--从 0 到 5 进行循环
CREATE OR REPLACE PROCEDURE proc_for_loop()
AS
    BEGIN
    FOR I IN 0..5 LOOP
       DBMS_OUTPUT.PUT_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13-22 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

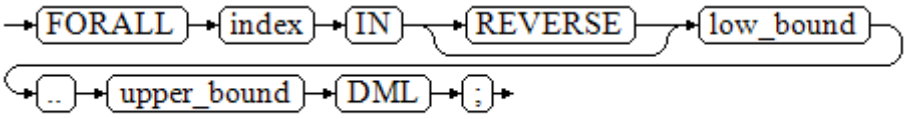
```
--循环输出查询结果。
CREATE OR REPLACE PROCEDURE proc_for_loop_query()
AS
    record VARCHAR2(50);
BEGIN
```

```
    FOR record IN SELECT spcname FROM pg_tablespace LOOP
    dbms_output.put_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13-23 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 hdfs_t1 (
  title NUMBER(6),
  did VARCHAR2(20),
  data_peroid VARCHAR2(25),
  kind VARCHAR2(25),
  interval VARCHAR2(20),
  time DATE,
  isModified VARCHAR2(10)
)
DISTRIBUTE BY hash(did);

INSERT INTO hdfs_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
       insert into hdfs t1(title) values(i);
END;
/
```

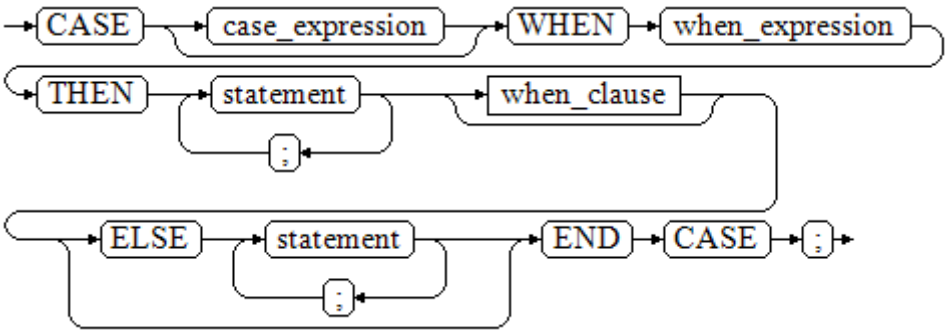
```
--调用函数
CALL proc_forall();

--查询存储过程调用结果
SELECT * FROM hdfs_t1 WHERE title BETWEEN 100 AND 120;

--删除存储过程和表
DROP PROCEDURE proc_forall;
DROP TABLE hdfs_t1;
```

13.8.4 分支语句

语法

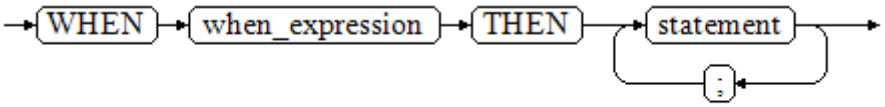
分支语句的语法请参见图 13-24。

图13-24 case_when::=



when_clause 子句的语法图参见图 13-25。

图13-25 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;
```

13.8.5 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;
/
```

13.8.6 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。

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 的名字可以是《数据仓库服务错误码参考》中的 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，我们可能得到唯一 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;
```

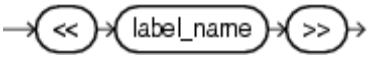
```
    END LOOP;
END;
$$
LANGUAGE plpgsql;

SELECT merge_db(1, 'david');
SELECT merge_db(1, 'dennis');

--删除 FUNCTION 和 TABLE
DROP FUNCTION merge_db;
DROP TABLE db ;
```

13.8.7 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

```
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();
DROP PROCEDURE 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;
/
```

13.9 其他语句

13.9.1 锁操作

GaussDB(DWS)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(DWS)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。

13.9.2 游标操作

GaussDB(DWS)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字。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 13.10.2 显式游标。

13.10 游标

13.10.1 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 13-2。

表13-2 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

13.10.2 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。

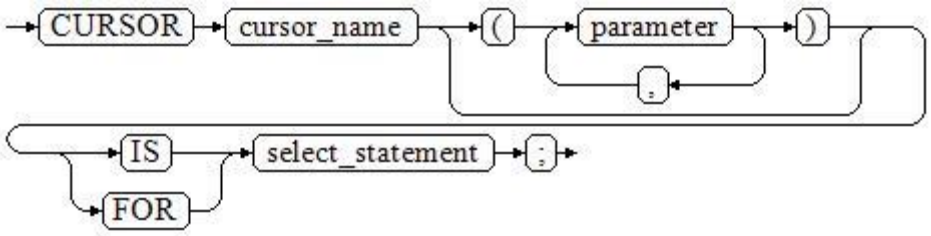
处理步骤

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 13-26。

图13-26 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。

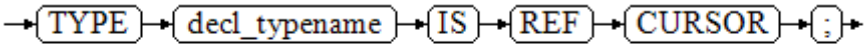
说明

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。

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。

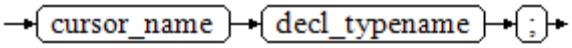
定义动态游标的语法图，请参见图 13-27 和图 13-28。

图13-27 cursor_typename::=



GaussDB(DWS)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。

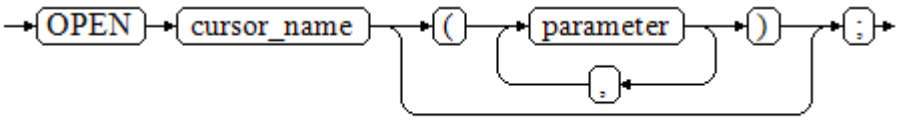
图13-28 dynamic_cursor_define::=



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 13-29。

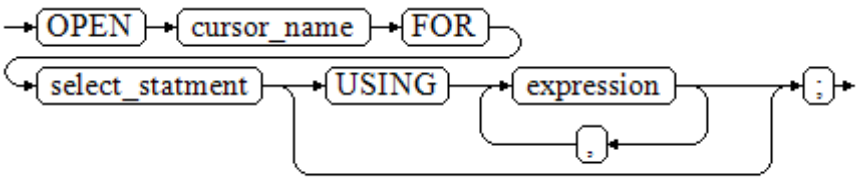
图13-29 open_static_cursor::=



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 13-30。

图13-30 open_dynamic_cursor::=

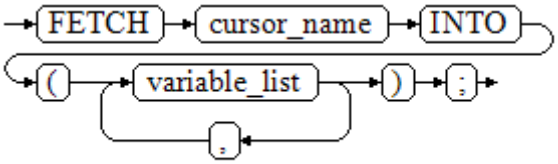


PL/SQL 程序不能用 OPEN 语句重复打开一个游标。

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 13-31。

图13-31 fetch_cursor::=



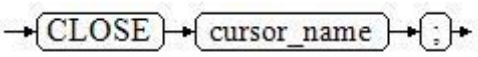
步骤 4 对该记录进行处理。

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 13-32。

图13-32 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

```
--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 sections WHERE section_id <= 50;
    CURSOR C2(sect_id INTEGER) IS
        SELECT section_name, place_id FROM 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;
        DBMS_OUTPUT.PUT_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;
        DBMS_OUTPUT.PUT_LINE(DEPT_NAME||'---'||DEPT_LOC);
    END LOOP;
    CLOSE C2;

    SQL_STR := 'SELECT section_name, place_id FROM 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;
        DBMS_OUTPUT.PUT_LINE(DEPT_NAME||'---'||DEPT_LOC);
    END LOOP;
    CLOSE C3;
END;
/

CALL cursor_proc1();

DROP PROCEDURE cursor_proc1;
--给工资低于 3000 的员工增加工资 500。
CREATE TABLE staffs_t1 AS TABLE 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 staffs_t1;
BEGIN
  OPEN C;
  LOOP
    FETCH C INTO V_EMPNO, V_SAL;
    EXIT WHEN C%NOTFOUND;
    IF V SAL<=3000 THEN
         UPDATE staffs t1 SET salary =salary + 500 WHERE staff id = V EMPNO;
    END IF;
  END LOOP;
  CLOSE C;
```

```
END;
/

CALL cursor_proc2();

--删除存储过程
DROP PROCEDURE cursor_proc2;
DROP TABLE staffs_t1;
--SYS_REFCURSOR 类型做为函数参数
CREATE OR REPLACE PROCEDURE proc_sys_ref(O OUT SYS_REFCURSOR)
IS
C1 SYS_REFCURSOR;
BEGIN
OPEN C1 FOR SELECT section_ID FROM 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;
  DBMS_OUTPUT.PUT_LINE(C1%ROWCOUNT);
  EXIT WHEN C1%NOTFOUND;
END LOOP;
END;
/

--删除存储过程
DROP PROCEDURE proc_sys_ref;
```

13.10.3 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，隐式游标的名字为 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，DROP，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得记录数。
- 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 staffs_t1 AS TABLE staffs;
CREATE TABLE sections_t1 AS TABLE sections;

CREATE OR REPLACE PROCEDURE proc_cursor3()
AS
    DECLARE
    V_DEPTNO NUMBER(4) := 100;
    BEGIN
        DELETE FROM staffs WHERE section_ID = V_DEPTNO;
        --根据游标状态做进一步处理
        IF SQL%NOTFOUND THEN
        DELETE FROM 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 staffs t1;
DROP TABLE sections_t1;
```

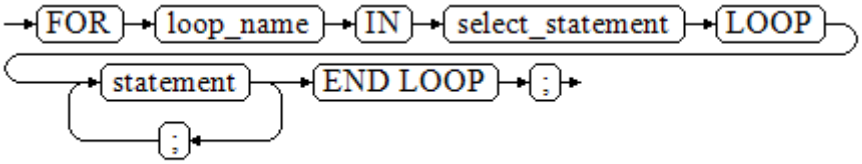
13.10.4 游标循环

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 13-33。

图13-33 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(DWS)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 staffs
    LOOP
       DBMS_OUTPUT.PUT_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
        DBMS_OUTPUT.PUT_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;
```

13.11 高级包

13.11.1 DBMS_LOB

接口介绍

高级功能包 DBMS_LOB 支持的所有接口请参见表 13-3。

表13-3 DBMS_LOB

接口名称	描述
DBMS_LOB.GETLENGTH	获取并返回指定的 LOB 类型对象的长度。
DBMS_LOB.OPEN	打开一个 LOB 返回一个 LOB 的描述符。
DBMS_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。
DBMS_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。
DBMS_LOB.WRITEAPPEND	根据指定长度将 BUFFER 中内容写入到 LOB 的尾部。
DBMS_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。
DBMS_LOB.ERASE	根据指定长度及起始位置偏移删除 BLOB 中的内容。
表 13-11	关闭已经打开的 LOB 描述符。
DBMS_LOB.INSTR	返回一个字符串在 LOB 中第 N 次出现的位置。
DBMS_LOB.COMPARE	比较两个 LOB 或者两个 LOB 的某一部分。
DBMS_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。
DBMS_LOB.TRIM	用于截断指定长度的 LOB，执行完会将 LOB 的长度设置为 newlen 参数指定的长度。
DBMS_LOB.CREATETEMPORARY	创建一个临时的 BLOB 或者 CLOB。

接口名称	描述
DBMS_LOB.APPEND	将原 LOB 的内容拼接到目的 LOB 中。

- DBMS_LOB.GETLENGTH

  存储过程 GETLENGTH 获取并返回指定的 LOB 类型对象的长度。

  DBMS_LOB.GETLENGTH 函数原型为：

```
DBMS_LOB.GETLENGTH (
lob_loc    IN   BLOB)
RETURN INTEGER;

DBMS_LOB.GETLENGTH (
lob_loc    IN   CLOB)
RETURN INTEGER;
```

表13-4 DBMS_LOB.GETLENGTH 接口参数说明

参数	描述
lob_loc	待获取长度的指定的 LOB 类型的对象。

- DBMS_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。

  DBMS_LOB.OPEN 函数原型为：

```
DBMS_LOB.LOB (
lob_loc    INOUT   BLOB,
open_mode IN BINARY_INTEGER);

DBMS_LOB.LOB (
lob_loc  INOUT    CLOB,
open_mode IN BINARY_INTEGER);
```

表13-5 DBMS_LOB.OPEN 接口参数说明

参数	描述
lob_loc	被打开的一个 BLOB 或者 CLOB 描述符。
open_mode IN BINARY_INTEGER	打开的模式，目前支持 DBMS_LOB.LOB_READWRITE

- DBMS_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。

DBMS_LOB.READ 函数原型为：

```
DBMS_LOB.READ (
lob_loc    IN         BLOB,
amount     IN         INTEGER,
offset     IN         INTEGER,
buffer     OUT        RAW);

DBMS_LOB.READ (
lob_loc    IN         CLOB,
amount     IN OUT      INTEGER,
offset     IN         INTEGER,
buffer     OUT        VARCHAR2);
```

表13-6 DBMS_LOB.READ 接口参数说明

参数	说明
lob_loc	待读入的指定的 LOB 类型的对象。
amount	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.”
offset	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。

- DBMS_LOB.WRITE

  存储过程 WRITE 根据指定长度及起始位置偏移将 BUFFER 中内容写入到 LOB 变量中。

  DBMS_LOB.WRITE 函数原型为：

```
DBMS_LOB.WRITE (
lob_loc    IN OUT    BLOB,
amount     IN       INTEGER,
offset     IN       INTEGER,
buffer     IN       RAW);

DBMS_LOB.WRITE (
lob_loc   IN OUT     CLOB,
amount    IN        INTEGER,
offset    IN        INTEGER,
buffer    IN        VARCHAR2);
```

表13-7 DBMS_LOB.WRITE 接口参数说明

参数	说明
lob_loc	待写入的指定的 LOB 类型的对象。

参数	说明
amount	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。
offset	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 或偏移量大于 LOB 类型最大长度，则报错。
buffer	待写入的内容。

- DBMS_LOB.WRITEAPPEND

  存储过程 WRITEAPPEND 根据指定长度将 BUFFER 中内容写入到 LOB 的尾部。

  DBMS_LOB.WRITEAPPEND 函数原型为：

```
DBMS_LOB.WRITEAPPEND (
lob_loc    IN OUT    BLOB,
amount     IN        INTEGER,
buffer     IN        RAW);

DBMS_LOB.WRITEAPPEND (
lob_loc    IN OUT    CLOB,
amount     IN        INTEGER,
buffer     IN        VARCHAR2);
```

表13-8 DBMS_LOB.WRITEAPPEND 接口参数说明

参数	说明
lob_loc	待写入的指定的 LOB 类型的对象。
amount	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。
buffer	待写入的内容。

- DBMS_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。

  DBMS_LOB.COPY 函数原型为：

```
DBMS_LOB.COPY (
dest_lob     IN OUT    BLOB,
src_lob      IN        BLOB,
amount       IN        INTEGER,
```

```
dest_offset   IN        INTEGER  DEFAULT 1,
src_offset    IN        INTEGER  DEFAULT 1);
```

表13-9 DBMS_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。
amount	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_offset	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_offset	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。

- DBMS_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。

  DBMS_LOB.ERASE 函数原型为：

```
DBMS_LOB.ERASE (
lob_loc         IN OUT   BLOB,
amount          IN OUT   INTEGER,
offset          IN       INTEGER DEFAULT 1);
```

表13-10 DBMS_LOB.ERASE 接口参数说明

参数	说明
lob_loc	待删除内容的 BLOB 类型对象。
amount	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。
offset	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。

- DBMS_LOB.CLOSE

  存储过程 CLOSE 根据指定长度及起始位置偏移关闭已经打开的 LOB 内容。

  DBMS_LOB.CLOSE 函数原型为：

```
DBMS_LOB.CLOSE(
src_lob      IN            BLOB);

DBMS_LOB.CLOSE (
src_lob     IN             CLOB);
```

表13-11 DBMS_LOB.CLOSE 接口参数说明

参数	说明
src_loc	待关闭的 LOB 类型对象。

- DBMS_LOB.INSTR

  该函数返回在 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。

  DBMS_LOB.INSTR 函数原型为：

```
DBMS_LOB.INSTR (
lob loc     IN     BLOB,
pattern     IN     RAW,
offset      IN     INTEGER := 1,
nth         IN     INTEGER := 1)
RETURN INTEGER;

DBMS LOB.INSTR (
lob loc    IN     CLOB,
pattern    IN     VARCHAR2 ,
offset     IN     INTEGER := 1,
nth        IN     INTEGER := 1)
RETURN INTEGER;
```

表13-12 DBMS_LOB.INSTR 接口参数说明

参数	说明
lob_loc	要查找的 LOB 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。
offset	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。
nth	模式匹配的次数，最小值为 1。

- DBMS_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 CLOB 比第二个小，返回-1；如果第一个 CLOB 比第二个大，返回 1。
  - 如果 amount，offset_1，offset_2 这几个参数有无效参数返回 NULL，有效的偏移量范围是 1~LOBMAXSIZE。

  DBMS_LOB.READ 函数原型为：

```
DBMS_LOB.COMPARE (
lob_1     IN     BLOB,
lob_2     IN     BLOB,
amount    IN     INTEGER := DBMS_LOB.LOBMAXSIZE,
offset_1  IN     INTEGER := 1,
offset_2  IN     INTEGER := 1)
RETURN INTEGER;

DBMS LOB.COMPARE (
lob 1     IN     CLOB,
lob 2     IN     CLOB,
amount    IN     INTEGER := DBMS LOB.LOBMAXSIZE,
offset 1  IN     INTEGER := 1,
offset_2  IN     INTEGER := 1)
RETURN INTEGER;
```

表13-13 DBMS_LOB.COMPARE 接口参数说明

参数	说明
lob_1	第一个要比较的 LOB 描述符。
lob_2	第二个要比较的 LOB 描述符。
amount	要比较的字符数或者字节数， 最大值为 DBMS_LOB.LOBMAXSIZE。
offset_1	第一个 LOB 描述符的偏移量，初始位置是 1。
offset_2	第二个 LOB 描述符的偏移量，初始位置是 1。

- DBMS_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。

  DBMS_LOB.SUBSTR 函数原型为：

```
DBMS_LOB.SUBSTR (
lob_loc     IN     BLOB,
amount      IN     INTEGER := 32767,
offset      IN     INTEGER := 1)
RETURN RAW;
```

```
DBMS_LOB.SUBSTR (
lob_loc    IN     CLOB,
amount     IN     INTEGER := 32767,
offset     IN     INTEGER := 1)
RETURN VARCHAR2;
```

表13-14 DBMS_LOB.SUBSTR 接口参数说明

参数	说明
lob_loc	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。
offset	要读取的字节数或者字符数量。
buffer	从开始位置偏移的字符数或者字节数量。

- DBMS_LOB.TRIM

  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。

  DBMS_LOB.TRIM 函数原型为：

```
DBMS_LOB.TRIM (
lob_loc     IN OUT     BLOB,
newlen      IN         INTEGER);

DBMS_LOB.TRIM (
lob_loc     IN         OUT CLOB,
newlen      IN         INTEGER);
```

表13-15 DBMS_LOB.TRIM 接口参数说明

参数	说明
lob_loc	待读入的指定 BLOB 类型的对象。
newlen	截断后 LOB 的新长度对于 BLOB 是字节数，对于 CLOB 是字符数。

- DBMS_LOB.CREATETEMPORARY

  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。

  DBMS_LOB.CREATETEMPORARY 函数原型为：

```
DBMS_LOB.CREATETEMPORARY (
lob_loc    IN OUT     BLOB,
cache      IN         BOOLEAN,
dur        IN         INTEGER);

DBMS_LOB.CREATETEMPORARY (
```

```
lob_loc    IN OUT    CLOB,
cache      IN        BOOLEAN,
dur        IN        INTEGER);
```

表13-16 DBMS_LOB.CREATETEMPORARY 接口参数说明

参数	说明
lob_loc	LOB 描述符。
cache	仅用于语法上的兼容。
dur	仅用于语法上的兼容。

- DBMS_LOB.APPEND

  存储过程 READ 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。

  DBMS_LOB.APPEND 函数原型为：

```
DBMS_LOB.APPEND (
dest_lob    IN OUT      BLOB,
src_lob     IN          BLOB);

DBMS_LOB.APPEND (
dest_lob    IN OUT      CLOB,
src_lob     IN          CLOB);
```

表13-17 DBMS_LOB.APPEND 接口参数说明

参数	说明
dest_lob	要写入的 LOB 描述符。
src_lob	读取的 LOB 描述符。

示例

```
--获取字符串的长度
SELECT DBMS_LOB.GETLENGTH('12345678');

DECLARE
myraw  RAW(100);
amount INTEGER :=2;
buffer INTEGER :=1;
begin
DBMS_LOB.READ('123456789012345',amount,buffer,myraw);
dbms_output.put_line(myraw);
end;
/

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 := utl_raw.cast_to_raw(str);
amount := utl_raw.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;

DBMS_LOB.WRITE(dest, amount, 1, source);
DBMS_LOB.WRITEAPPEND(dest, amount, source);

DBMS_LOB.ERASE(dest, a, 1);
DBMS_OUTPUT.PUT_LINE(a);
DBMS_LOB.COPY(copyto, dest, amount, 10, 1);
DBMS_LOB.CLOSE(dest);
RETURN;
END;
/

--删除表
DROP TABLE blob_Table;
DROP TABLE blob_Table_bak;
```

13.11.2 DBMS_RANDOM

接口介绍

高级功能包 DBMS_RANDOM 支持的所有接口请参见表 13-18。

表13-18 DBMS_RANDOM 接口参数说明

接口名称	描述
DBMS_RANDOM.SEED	设置一个随机数的种子。
DBMS_RANDOM.VALUE	生成一个大小介于指定的 low 及 high 之间的随机数。

- DBMS_RANDOM.SEED

  存储过程 SEED 用于设置一个随机数的种子。DBMS_RANDOM.SEED 函数原型为：

```
DBMS_RANDOM.SEED (seed  IN  INTEGER);
```

表13-19 DBMS_RANDOM.SEED 接口参数说明

参数	描述
seed	用于产生一个随机数的种子。

- DBMS_RANDOM.VALUE

  存储过程 VALUE 生成一个大小介于指定的 low 及 high 之间的随机数。DBMS_RANDOM.VALUE 函数原型为：

```
DBMS_RANDOM.VALUE(
low  IN  NUMBER,
high IN  NUMBER)
RETURN NUMBER;
```

表13-20 DBMS_RANDOM.VALUE 接口参数说明

参数	描述
low	指定随机数大小的下边界，生成的随机数大于或等于 low。
high	指定随机数大小的上边界，生成的随机数小于 high。

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。

示例

```
--产生 0 到 1 之间的随机数：
SELECT DBMS_RANDOM.VALUE(0,1);

--对于指定范围内的整数，要加入参数 low 和 high，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(DBMS_RANDOM.VALUE(0,100));
```

13.11.3 DBMS_OUTPUT

接口介绍

高级功能包 DBMS_OUTPUT 支持的所有接口请参见表 13-21。

表13-21 DBMS_OUTPUT

接口名称	描述
表 13-22	输出指定的文本，文本长度不能超过 32767 字节。
表 13-23	将指定的文本输出到指定文本的前面，不添加换行符，文本长度不能超过 32767 字节。
表 13-24	设置输出缓冲区的大小。若不指定，缓冲区最大只能容纳 20000 字节，缓冲区最小可设置为 2000 字节，若设置小于 2000 字节将按 2000 字节处理。

- DBMS_OUTPUT.PUT_LINE

存储过程 PUT_LINE 向消息缓冲区写入一行带有行结束符的文本。
DBMS_OUTPUT.PUT_LINE 函数原型为：

```
DBMS_OUTPUT.PUT_LINE (
item IN VARCHAR2);
```

表13-22 DBMS_OUTPUT.PUT_LINE 接口参数说明

参数	描述
item	写入消息缓冲区的文本。

- DBMS_OUTPUT.PUT

存储过程 PUT 将指定的文本输出到指定文本的前面，不添加换行符。
DBMS_OUTPUT.PUT 函数原型为：

```
DBMS_OUTPUT.PUT (
item IN VARCHAR2);
```

表13-23 DBMS_OUTPUT.PUT 接口参数说明

参数	描述
item	写入指定文本前的文本。

- DBMS_OUTPUT.ENABLE

存储过程 ENABLE 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。DBMS_OUTPUT.ENABLE 函数原型为：

```
DBMS_OUTPUT.ENABLE (
buf IN INTEGER);
```

表13-24 DBMS_OUTPUT.ENABLE 接口参数说明

参数	描述
buf	设置输出缓冲区的大小。

示例

```
BEGIN
    DBMS_OUTPUT.ENABLE(50);
    DBMS_OUTPUT.PUT ('hello, ');
    DBMS_OUTPUT.PUT_LINE('database!');--输出 hello, database!
END;
/
```

13.11.4 UTL_RAW

接口介绍

高级功能包 UTL_RAW 支持的所有接口请参见表 13-25。

表13-25 UTL_RAW

接口名称	描述
表 13-26	将 INTEGER 类型值转换为二进制表示形式（即 RAW 类型）。
表 13-27	将二进制表示形式的整型值（即 RAW 类型）转换为 INTEGER 类型。
UTL_RAW.LENGTH	获取 RAW 类型对象的长度。
UTL_RAW.CAST_TO_RAW	将 VARCHAR2 类型值转化为二进制表示形式（即 RAW 类型）。

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’。

- UTL_RAW.CAST_FROM_BINARY_INTEGER

  存储过程 CAST_FROM_BINARY_INTEGER 将 INTEGER 类型值转换为二进制表示形式（即 RAW 类型）。

  UTL_RAW.CAST_FROM_BINARY_INTEGER 函数原型为：

  ```
  UTL_RAW.CAST_FROM_BINARY_INTEGER (
  n          IN  INTEGER,
  ```

```
endianess   IN  INTEGER)
RETURN RAW;
```

表13-26 UTL_RAW.CAST_FROM_BINARY_INTEGER 接口参数说明

参数	描述
n	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）。

- UTL_RAW.CAST_TO_BINARY_INTEGER

  存储过程 CAST_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。

  UTL_RAW.CAST_TO_BINARY_INTEGER 函数原型为：

```
UTL_RAW.CAST_TO_BINARY_INTEGER (
r         IN  RAW,
endianess  IN  INTEGER)
RETURN BINARY_INTEGER;
```

表13-27 UTL_RAW.CAST_TO_BINARY_INTEGER 接口参数说明

参数	描述
r	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）。

- UTL_RAW.LENGTH

  存储过程 LENGTH 返回 RAW 类型对象的长度。

  UTL_RAW.LENGTH 函数原型为：

```
UTL_RAW.LENGTH(
r      IN RAW)
RETURN INTEGER;
```

表13-28 UTL_RAW.LENGTH 接口参数说明

参数	描述
r	RAW 类型对象

- UTL_RAW.CAST_TO_RAW

  存储过程 CAST_TO_RAW 将 VARCHAR2 类型的对象转换成 RAW 类型。

  UTL_RAW.CAST_TO_RAW 函数原型为：

```
UTL_RAW.CAST_TO_RAW(
c      IN VARCHAR2)
RETURN RAW;
```

表13-29 UTL_RAW.CAST_TO_RAW 接口参数说明

参数	描述
c	待转换的 VARCHAR2 类型对象

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 := utl_raw.cast_to_raw(str);--类型转换
amount := utl_raw.length(source);--获取长度
dbms_output.put_line(amount);
END;
/

--调用存储过程
CALL proc raw();

--删除存储过程
DROP PROCEDURE proc_raw;
```

13.11.5 DBMS_JOB

接口介绍

高级功能包 DBMS_JOB 支持的所有接口请参见表 13-30。

表13-30 DBMS_JOB

接口名称	描述
DBMS_JOB.SUBMIT	提交一个定时任务。作业号由系统自动生成。
DBMS_JOB.ISUBMIT	提交一个定时任务。作业号由用户指定。
DBMS_JOB.REMOVE	通过作业号来删除定时任务。
DBMS_JOB.BROKEN	禁用或者启用定时任务。
DBMS_JOB.CHANG	修改定时任务的属性，包括任务内容、下次执行时间、执行

接口名称	描述
E	间隔。
DBMS_JOB.WHAT	修改定时任务的任务内容属性。
DBMS_JOB.NEXT_DATE	修改定时任务的下次执行时间属性。
DBMS_JOB.INTERVAL	修改定时任务的执行间隔属性。
DBMS_JOB.CHANGE_OWNER	修改定时任务的属主。

- DBMS_JOB.SUBMIT

  存储过程 SUBMIT 提交一个系统提供的定时任务。

  DBMS_JOB.SUBMIT 函数原型为：

```
DMBS_JOB.SUBMIT(
what        IN   TEXT,
next_date    IN   TIMESTAMP DEFAULT sysdate,
job_interval IN   TEXT  DEFAULT 'null',
job         OUT  INTEGER);
```

说明

当创建一个定时任务（DBMS_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（DBMS_JOB）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。

表13-31 DBMS_JOB.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‘DML’，‘匿名块’，‘调用存储过程的语句’或 3 种混合的场景。
next_date	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	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

参数	类型	入参/出参	是否可以为空	描述
				调用 dbms.submit 时，该参数可以省略。

示例：

```
select DBMS_JOB.SUBMIT('call pro_xxx();',
to_date('20180101','yyyymmdd'),'sysdate+1');

select DBMS_JOB.SUBMIT('call pro_xxx();',
to_date('20180101','yyyymmdd'),'sysdate+1.0/24');

CALL DBMS_JOB.SUBMIT('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);
```

- DBMS_JOB.ISUBMIT

  ISUBMIT 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。

  示例：

```
CALL dbms_job.isubmit(101, 'insert_msg_statistic1;', sysdate, 'sysdate+3.0/24');
```

- DBMS_JOB.REMOVE

  存储过程 REMOVE 删除指定的定时任务。

  DBMS_JOB.REMOVE 函数原型为：

```
REMOVE(job  IN  INTEGER);
```

表13-32 DBMS_JOB.REMOVE 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。

示例：

```
CALL dbms_job.remove(101);
```

- DBMS_JOB.BROKEN

  存储过程 BROKEN 禁用或者启用定时任务。

  DBMS_JOB.BROKEN 函数原型为：

```
DMBS_JOB.BROKEN(
job         IN   INTEGER,
broken      IN   BOOLEAN,
next_date   IN   TIMESTAMP  DEFAULT  sysdate);
```

表13-33 DBMS_JOB.BROKEN 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。
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_date	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_date 不为空值，则更新指定 job 的 next_date 值，如果 next_date 为空值，则不更新 next_date 值。该参数可以省略，为默认值。

示例：

```
CALL dbms_job.broken(101, true);
CALL dbms_job.broken(101, false, sysdate);
```

- DBMS_JOB.CHANGE

  存储过程 CHANGE 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。

  DBMS_JOB.CHANGE 函数原型为：

```
DMBS JOB.CHANGE(
job         IN   INTEGER,
what        IN   TEXT,
next_date    IN   TIMESTAMP,
interval     IN   TEXT);
```

表13-34 DBMS_JOB.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_	timest	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

参数	类型	入参/出参	是否可以为空	描述
date	amp			定 job 的 next_date 值。
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 interval 值；如果该参数不为空值，会校验 interval 是否为有效的时间类型或 interval 类型，则更新指定 job 的 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。

示例：

```
CALL dbms_job.change(101, 'call userproc();', sysdate, 'sysdate + 1.0/1440');
CALL dbms_job.change(101, 'insert into tbl_a values(sysdate);', sysdate,
'sysdate + 1.0/1440');
```

- DBMS_JOB.WHAT

  存储过程 WHAT 修改定时任务的任务内容属性。

  DBMS_JOB.WHAT 函数原型为：

```
DMBS_JOB.WHAT(
job             IN     INTEGER,
what            IN     TEXT);
```

表13-35 DBMS_JOB.WHAT 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	否	执行的存储过程调用或者 sql 语句块。

说明

- 当 what 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 what 参数为一个简单的 insert、update 等语句，需要在表前加模式名。

示例：

```
CALL dbms_job.what(101, 'call userproc();');
CALL dbms_job.what(101, 'insert into tbl_a values(sysdate);');
```

- DBMS_JOB.NEXT_DATE

  存储过程 NEXT_DATE 修改定时任务的下次执行时间属性。

  DBMS_JOB.NEXT_DATE 函数原型为：

```
DMBS_JOB.NEXT_DATE(
job         IN    INTEGER,
next_date    IN    TIMESTAMP);
```

表13-36 DBMS_JOB.NEXT_DATE 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。
next_date	timestamp	IN	否	下次运行时间。

说明

如果输入的 next_date 的值小于当前日期值，该 job 会立即执行一次。

示例：

```
CALL dbms_job.next_date(101, sysdate);
```

- DBMS_JOB.INTERVAL

  存储过程 INTERVAL 修改定时任务的执行间隔属性。

  DBMS_JOB.INTERVAL 函数原型为：

```
DMBS_JOB.INTERVAL(
job            IN   INTEGER,
interval        IN   TEXT);
```

表13-37 DBMS_JOB.INTERVAL 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。
interval	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。interval 是否为有效的时间类型或 interval 类型。

示例：

```
CALL dbms_job.interval(101, 'sysdate + 1.0/1440');
```

说明

对于指定 job 正在运行状态（即 job_status 为'r'）时，不允许通过 remove、change、next_date、what、interval 等接口删除或修改 job 的参数信息。

- DBMS_JOB.CHANGE_OWNER

  存储过程 CHANGE_OWNER 修改定时任务的属主。

DBMS_JOB.CHANGE_OWNER 函数原型为：

```
DMBS_JOB.CHANGE_OWNER(
job            IN     INTEGER,
new_owner       IN     NAME);
```

表13-38 DBMS_JOB.CHANGE_OWNER 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。
new_owner	name	IN	否	新的用户名。

示例：

```
CALL dbms_job.change_owner(101, 'alice');
```

约束说明

1. 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。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 dbms_job 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_jobs 表主键的关联操作，如果通过 DML 语句对 pg_jobs 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，若不开启 CN 故障自动迁移功能，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。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。建议开启 CN 故障自动迁移功能，故障 CN 上的作业会迁移至其他 CN 继续调度。
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_jobs 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_jobs 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_jobs 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。

13.11.6 DBMS_SQL

接口介绍

高级功能包 DBMS_SQL 支持的接口请参见表 13-39。

表13-39 DBMS_SQL

接口名称	描述
DBMS_SQL.OPEN_CURSOR	打开一个游标。
DBMS_SQL.CLOSE_CURSOR	关闭一个已打开的游标。
DBMS_SQL.PARSE	向游标传递一组 SQL 语句，目前只支持 SELECT。
DBMS_SQL.EXECUTE	在游标上执行一组动态定义操作。
DBMS_SQL.FETCHE_ROWS	读取游标一行数据。
DBMS_SQL.DEFINE_COLUMN	动态定义一个列。
DBMS_SQL.DEFINE_COLUMN_CHAR	动态定义一个 char 类型的列。
DBMS_SQL.DEFINE_COLUMN_INT	动态定义一个 int 类型的列。
DBMS_SQL.DEFINE_COLUMN_LONG	动态定义一个 long 类型的列。
DBMS_SQL.DEFINE_COLUMN_RAW	动态定义一个 raw 类型的列。
DBMS_SQL.DEFINE_COLUMN_TEXT	动态定义一个 text 类型的列。
DBMS_SQL.DEFINE_COLUMN_UNKNOWN	动态定义一个未知列（类型不识别时入此接口）。
DBMS_SQL.COLUMN_VALUE	读取一个已动态定义的列值。
DBMS_SQL.COLUMN_VALUE_CHAR	读取一个已动态定义的列值（指定 char 类型）。
DBMS_SQL.COLUMN_VALUE_INT	读取一个已动态定义的列值（指定 int 类型）。
DBMS_SQL.COLUMN_VALUE_LONG	读取一个已动态定义的列值（指定 long 类型）。
DBMS_SQL.COLUMN_VALUE_RAW	读取一个已动态定义的列值（指定 raw 类型）。
DBMS_SQL.COLUMN_VALUE_TEXT	读取一个已动态定义的列值（指定 text 类型）。
DBMS_SQL.COLUMN_VALUE_UNKNOWN	读取一个已动态定义的列值（类型不识别时入此接口）。

接口名称	描述
DBMS_SQL.IS_OPEN	检查游标是否已打开。

说明

- 建议使用 dbms_sql.define_column 及 dbms_sql.column_value 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。

- DBMS_SQL.OPEN_CURSOR

  该函数用来打开一个游标，是后续 dbms_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。

  DBMS_SQL.OPEN_CURSOR 函数原型为：

```
DBMS_SQL.OPEN_CURSOR (
)
RETURN INTEGER;
```

- DBMS_SQL.CLOSE_CURSOR

  该函数用来关闭一个游标，是 dbms_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。

  DBMS_SQL.CLOSE_CURSOR 函数原型为：

```
DBMS_SQL.CLOSE_CURSOR (
cursorid    IN INTEGER
)
RETURN INTEGER;
```

表13-40 DBMS_SQL.CLOSE_CURSOR 接口说明

参数名称	描述
cursorid	打算关闭的游标 ID 号

- DBMS_SQL.PARSE

  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。

  DBMS_SQL.PARSE 函数的原型为：

```
DBMS_SQL.PARSE (
cursorid    IN INTEGER,
query_string IN TEXT,
label       IN INTEGER
)
RETURN BOOLEAN;
```

表13-41 DBMS_SQL.PARSE 接口说明

参数名称	描述
cursorid	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

- DBMS_SQL.EXECUTE

  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。目前仅支持 SELECT 查询语句的执行。

  DBMS_SQL.EXECUTE 函数的原型为：

```
DBMS_SQL.EXECUTE(
cursorid    IN INTEGER,
)
RETURN INTEGER;
```

表13-42 DBMS_SQL.EXECUTE 接口说明

参数名称	描述
cursorid	执行查询语句解析的游标 ID

- DBMS_SQL.FETCHE_ROWS

  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。

  DBMS_SQL.FETCHE_ROWS 函数的原型为：

```
DBMS_SQL.FETCHE_ROWS(
cursorid    IN INTEGER,
)
RETURN INTEGER;
```

表13-43 DBMS_SQL.FETCH_ROWS 接口说明

参数名称	描述
curosorid	执行的游标 ID

- DBMS_SQL.DEFINE_COLUMN

  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。

  DBMS_SQL.DEFINE_COLUMN 函数的原型为：

```
DBMS_SQL.DEFINE_COLUMN(
cursorid    IN INTEGER,
position    IN INTEGER,
column_ref   IN ANYELEMENT,
column_size    IN INTEGER default 1024
)
RETURN INTEGER;
```

表13-44 DBMS_SQL.DEFINE_COLUMN 接口说明

参数名称	描述
cursorid	执行的游标 ID
position	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
column_size	定义的列的长度

- DBMS_SQL.DEFINE_COLUMN_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。

  DBMS_SQL.DEFINE_COLUMN_CHAR 函数的原型为：

```
DBMS_SQL.DEFINE_COLUMN_CHAR(
cursorid    IN INTEGER,
position    IN INTEGER,
column      IN TEXT,
column_size    IN INTEGER
)
RETURN INTEGER;
```

表13-45 DBMS_SQL.DEFINE_COLUMN_CHAR 接口说明

参数名称	描述
cursorid	执行的游标 ID
position	动态定义列在查询中的位置
column	需要定义的某类型的参数变量
column_size	动态定义列长度

- DBMS_SQL.DEFINE_COLUMN_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。

DBMS_SQL.DEFINE_COLUMN_INT 函数的原型为：

```
DBMS_SQL.DEFINE_COLUMN_INT(
cursorid    IN INTEGER,
position    IN INTEGER
)
RETURN INTEGER;
```

表13-46 DBMS_SQL.DEFINE_COLUMN_INT 接口说明

参数名称	描述
cursorid	执行的游标 ID
position	动态定义列在查询中的位置

- DBMS_SQL.DEFINE_COLUMN_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。

  DBMS_SQL.DEFINE_COLUMN_LONG 函数的原型为：

```
DBMS_SQL.DEFINE_COLUMN_LONG(
cursorid    IN INTEGER,
position    IN INTEGER
)
RETURN INTEGER;
```

表13-47 DBMS_SQL.DEFINE_COLUMN_LONG 接口说明

参数名称	描述
cursorid	执行的游标 ID
position	动态定义列在查询中的位置

- DBMS_SQL.DEFINE_COLUMN_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。

  DBMS_SQL.DEFINE_COLUMN_RAW 函数的原型为：

```
DBMS_SQL.DEFINE_COLUMN_RAW(
cursorid    IN INTEGER,
position    IN INTEGER,
column      IN BYTEA,
column_size    IN INTEGER
)
RETURN INTEGER;
```

表13-48 DBMS_SQL.DEFINE_COLUMN_RAW 接口说明

参数名称	描述
cursorid	执行的游标 ID
position	动态定义列在查询中的位置
column	RAW 类型的参数变量
column_size	列的长度

- DBMS_SQL.DEFINE_COLUMN_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。

  DBMS_SQL.DEFINE_COLUMN_TEXT 函数的原型为：

```
DBMS_SQL.DEFINE_COLUMN_CHAR(
cursorid    IN INTEGER,
position    IN INTEGER,
max_size    IN INTEGER
)
RETURN INTEGER;
```

表13-49 DBMS_SQL.DEFINE_COLUMN_TEXT 接口说明

参数名称	描述
cursorid	执行的游标 ID
position	动态定义列在查询中的位置
max_size	定义的 TEXT 类型的最大长度

- DBMS_SQL.DEFINE_COLUMN_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。

  DBMS_SQL.DEFINE_COLUMN_UNKNOWN 函数的原型为：

```
DBMS_SQL.DEFINE_COLUMN_CHAR(
cursorid    IN INTEGER,
position    IN INTEGER,
column      IN TEXT
)
RETURN INTEGER;
```

表13-50 DBMS_SQL.DEFINE_COLUMN_UNKNOWN 接口说明

参数名称	描述

参数名称	描述
cursorid	执行的游标 ID
position	动态定义列在查询中的位置
column	动态定义的参数

- DBMS_SQL.COLUMN_VALUE

  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 DBMS_SQL.FETCH_ROWS 获取的数据。

  DBMS_SQL.COLUMN_VALUE 函数的原型为：

```
DBMS_SQL.COLUMN_VALUE(
cursorid              IN    INTEGER,
position              IN    INTEGER,
column_value           INOUT ANYELEMENT
)
RETURN ANYELEMENT;
```

表13-51 DBMS_SQL.COLUMN_VALUE 接口说明

参数名称	描述
cursorid	执行的游标 ID
position	动态定义列在查询中的位置
column_value	定义的列的返回值

- DBMS_SQL.COLUMN_VALUE_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 DBMS_SQL.FETCH_ROWS 获取的数据。

  DBMS_SQL.COLUMN_VALUE_CHAR 函数的原型为：

```
DBMS_SQL.COLUMN_VALUE_CHAR(
cursorid              IN    INTEGER,
position              IN    INTEGER,
column_value           INOUT CHARACTER,
err_num               INOUT NUMERIC default 0,
actual_length          INOUT INTEGER default 1024
)
RETURN RECORD;
```

表13-52 DBMS_SQL.COLUMN_VALUE_CHAR 接口说明

参数名称	描述
cursorid	执行的游标 ID

参数名称	描述
position	动态定义列在查询中的位置
column_value	返回值
err_num	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

- DBMS_SQL.COLUMN_VALUE_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 DBMS_SQL.FETCH_ROWS 获取的数据。DBMS_SQL.COLUMN_VALUE_INT 函数的原型为：

```
DBMS_SQL.COLUMN_VALUE_INT(
cursorid              IN    INTEGER,
position              IN    INTEGER
)
RETURN INTEGER;
```

表13-53 DBMS_SQL.COLUMN_VALUE_INT 接口说明

参数名称	描述
cursorid	执行的游标 ID
position	动态定义列在查询中的位置

- DBMS_SQL.COLUMN_VALUE_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 DBMS_SQL.FETCH_ROWS 获取的数据。

  DBMS_SQL.COLUMN_VALUE_LONG 函数的原型为：

```
DBMS_SQL.COLUMN_VALUE_LONG(
cursorid              IN    INTEGER,
position              IN    INTEGER,
length               IN    INTEGER,
off_set               IN    INTEGER,
column_value          INOUT TEXT,
actual_length         INOUT INTEGER default 1024
)
RETURN RECORD;
```

表13-54 DBMS_SQL.COLUMN_VALUE_LONG 接口说明

参数名称	描述
cursorid	执行的游标 ID

参数名称	描述
position	动态定义列在查询中的位置
length	返回值的长度
off_set	返回值的起始位置
column_value	返回值
actual_length	实际返回值的长度

- DBMS_SQL.COLUMN_VALUE_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 DBMS_SQL.FETCH_ROWS 获取的数据。

  DBMS_SQL.COLUMN_VALUE_RAW 函数的原型为：

```
DBMS_SQL.COLUMN_VALUE_RAW(
cursorid              IN    INTEGER,
position              IN    INTEGER,
column_value           INOUT BYTEA,
err_num               INOUT NUMERIC default 0,
actual_length          INOUT INTEGER default 1024
)
RETURN RECORD;
```

表13-55 DBMS_SQL.COLUMN_VALUE_RAW 接口说明

参数名称	描述
cursorid	执行的游标 ID
position	动态定义列在查询中的位置
column_value	返回的列值
err_num	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

- DBMS_SQL.COLUMN_VALUE_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 DBMS_SQL.FETCH_ROWS 获取的数据。

  DBMS_SQL.COLUMN_VALUE_TEXT 函数的原型为：

```
DBMS_SQL.COLUMN_VALUE_TEXT(
cursorid              IN    INTEGER,
position              IN    INTEGER
```

```
)
RETURN TEXT;
```

表13-56 DBMS_SQL.COLUMN_VALUE_TEXT 接口说明

参数名称	描述
cursorid	执行的游标 ID
position	动态定义列在查询中的位置

- DBMS_SQL.COLUMN_VALUE_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。

  DBMS_SQL.COLUMN_VALUE_UNKNOWN 函数的原型为：

```
DBMS_SQL.COLUMN_VALUE_UNKNOWN(
cursorid              IN    INTEGER,
position              IN    INTEGER,
COLUMN_TYPE            IN    TEXT
)
RETURN TEXT;
```

表13-57 DBMS_SQL.COLUMN_VALUE_UNKNOWN 接口说明

参数名称	描述
cursorid	执行的游标 ID
position	动态定义列在查询中的位置
column_type	返回的参数类型

- DBMS_SQL.IS_OPEN

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。

DBMS_SQL.IS_OPEN 函数的原型为：

```
DBMS_SQL.IS_OPEN(
cursorid              IN    INTEGER
)
RETURN BOOLEAN;
```

表13-58 DBMS_SQL.IS_OPEN 接口说明

参数名称	描述
cursorid	被查询的游标 ID

示例

```
--在存储过程中操作 raw 数据
create or replace procedure pro_dbms_sql_all_02(in raw raw,v in int,v offset int)
as
cursorid 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_dbms_sql_all_tb1_02 ;
create table pro_dbms_sql_all_tb1_02(a int ,b blob);
insert into pro_dbms_sql_all_tb1_02 values(1,HEXTORAW('DEADBEEE'));
insert into pro_dbms_sql_all_tb1_02 values(2,in_raw);
query := 'select * from pro_dbms_sql_all_tb1_02 order by 1';
--打开游标
cursorid := dbms_sql.open_cursor();
--编译游标
dbms_sql.parse(cursorid, query, 1);
--定义列
define_column_ret:= dbms_sql.define_column(cursorid,1,v_id);
define_column_ret_raw:= dbms_sql.define_column_raw(cursorid,2,v_info,10);
--执行
execute_ret := dbms_sql.execute(cursorid);
loop
exit when (dbms_sql.fetch_rows(cursorid) <= 0);
--获取值
dbms_sql.column_value(cursorid,1,v_id);
dbms_sql.column_value_raw(cursorid,2,v_info,v_in,v_offset);
--输出结果
dbms_output.put_line('id:'|| v_id || ' info:' || v_info);
end loop;
--关闭游标
dbms_sql.close_cursor(cursorid);
end;
/
--调用存储过程
call pro_dbms_sql_all_02(HEXTORAW('DEADBEEF'),0,1);

--删除存储过程
DROP PROCEDURE pro_dbms_sql_all_02;
```

13.12 调试

语法

RAISE 有以下五种语法格式：

图13-34 raise_format::=

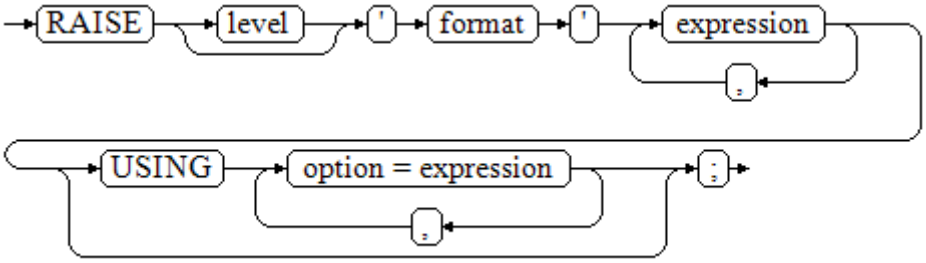


图13-35 raise_condition::=

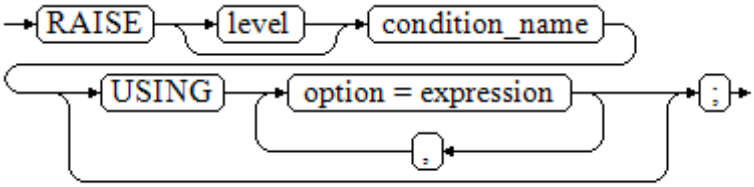


图13-36 raise_sqlstate::=

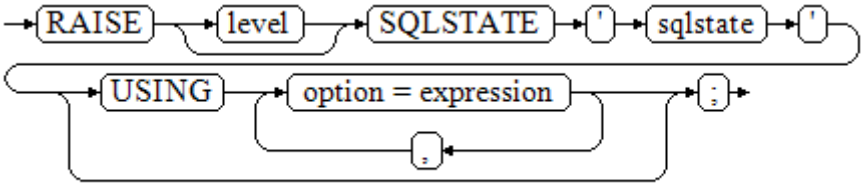


图13-37 raise_option::=

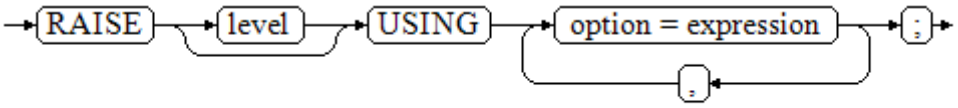
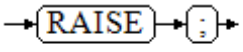


图13-38 raise::=



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。

说明

图 13-38 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
LOCATION:  exec_stmt_raise, pl_exec.cpp:3482
```

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;
```

14 系统表和系统视图

14.1 系统表和系统视图概述

系统表是 GaussDB(DWS)存放结构元数据，是 GaussDB(DWS)数据库系统运行控制信息的来源，也是数据库系统的核心组成部分。系统表包含集群安装信息以及GaussDB(DWS)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。

系统视图提供了查询系统表和访问数据库内部状态的方法。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来解决这个问题。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。

三权分立下，非管理员无权查看系统表和视图。非三权分立下，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。

须知

禁止对系统表或系统视图进行增删改等操作，手动对系统表或系统视图的修改或破坏可能会导致系统信息不一致，造成系统控制异常甚至出现集群不可用。

14.2 系统表

14.2.1 GS_OBSSCANINFO

GS_OBSSCANINFO 系统表定义了在云上加速场景中，使用加速集群时扫描 OBS 数据的运行时信息，每条记录对应一个 query 中单个 OBS 外表的运行时信息。

表14-1 GS_OBSSCANINFO 字段

名字	类型	引用	描述
query_id	bigint	-	查询标识。
user_id	text	-	执行该查询的数据库用户。
table_name	text	-	OBS 外表的表名。
file_type	text	-	底层数据保存的文件格式。
time_stamp	time_stam	-	扫描操作开始的时间。
actual_time	double	-	扫描操作执行时间，单位为秒。
file_scanned	bigint	-	扫描的文件数量。
data_size	double	-	扫描的数据量，单位为字节。
billing_info	text	-	保留字段。

14.2.2 GS_RESPOOL_RESOURCE_HISTORY

GS_RESPOOL_RESOURCE_HISTORY 表记录资源池监控历史信息，CN 和 DN 上均进行记录。

表14-2 GS_RESPOOL_RESOURCE_HISTORY 字段

名称	类型	描述
timestamp	timestamp	资源池监控信息持久化时间
nodegroup	name	资源池所属逻辑集群名称，默认集群显示 "installation"
rpname	name	资源池名称
cgroup	name	资源池关联控制组名称
ref_count	int	资源池引用作业数，作业经过资源池不管是否管控都会计数，仅 CN 上有效
fast_run	int	资源池快车道运行作业数，只在 CN 上有效
fast_wait	int	资源池快车道排队作业数，只在 CN 上有效
fast_limit	int	资源池快车道作业并发限制，只在 CN 上有效
slow_run	int	资源池慢车道运行作业数，只在 CN 上有效
slow_wait	int	资源池慢车道排队作业数，只在 CN 上有效
slow_limit	int	资源池慢车道作业并发限制，只在 CN 上有效

名称	类型	描述
used_cpu	double	资源池 5s 监控周期内使用 CPU 个数平均值，保留小数点后 2 位 • DN：显示当前 DN 上资源池使用的 CPU 个数 • CN：显示所有 DN 上资源池使用 CPU 的累积和
cpu_limit	int	资源池可用 CPU 的上限，CPU 配额管控情况下为 GaussDB 可用 CPU，CPU 限额管控情况下为关联控制组 CPU 可用 CPU • DN：显示当前 DN 上资源池可用 CPU 上限 • CN：显示所有 DN 上资源池可用 CPU 上限的累积和
used_mem	int	资源池使用的内存大小，单位：'MB'’ • DN：显示当前 DN 上资源池使用的内存大小 • CN：显示所有 DN 上资源池使用内存的累积和
estimate_mem	int	当前 CN 上，资源池运行作业的估算内存之和，只在 CN 上有效
mem_limit	int	资源池可用内存上限，单位：'MB' • DN：显示当前 DN 上资源池可用内存上限 • CN：显示所有 DN 上资源池可用内存上限的累积和
read_kbytes	bigint	资源池 5s 监控周期内逻辑读字节数，单位：'KB' • DN：显示当前 DN 上资源池逻辑读字节数 • CN：显示所有 DN 上资源池逻辑读字节的累积和
write_kbytes	bigint	资源池 5s 监控周期内逻辑写字节数，单位：'KB' • DN：显示当前 DN 上资源池逻辑写字节数 • CN：显示所有 DN 上资源池逻辑写字节的累积和
read_counts	bigint	资源池 5s 监控周期内逻辑读次数 • DN：显示当前 DN 上资源池逻辑读次数 • CN：显示所有 DN 上资源池逻辑读次数的累积和
write_counts	bigint	资源池 5s 监控周期内逻辑写次数 • DN：显示当前 DN 上资源池逻辑写次数 • CN：显示所有 DN 上资源池逻辑写次数的累

名称	类型	描述
		积和
read_speed	double	资源池 5s 监控周期内逻辑读速率的平均值 • DN：显示当前 DN 上资源池逻辑读速率 • CN：显示所有 DN 上资源池逻辑读速率的累积和
write_speed	double	资源池 5s 监控周期内逻辑写速率平均值 • DN：显示当前 DN 上资源池逻辑写速率 • CN：显示所有 DN 上资源池逻辑写速率的累积和

14.2.3 GS_WLM_INSTANCE_HISTORY

GS_WLM_INSTANCE_HISTORY 系统表存储与实例(CN 或 DN)相关的资源使用相关信息。该系统表里每条记录都是对应时间点某实例资源使用情况，包括：内存、CPU 核数、磁盘 IO、进程物理 IO 和进程逻辑 IO 信息。

表14-3 GS_WLM_INSTANCE_HISTORY 字段

名称	类型	描述
instancename	text	实例名称。
timestamp	timestamp with time zone	时间戳。
used_cpu	int	实例使用 CPU 所占用的百分比。
free_mem	int	实例未使用的内存大小，单位 MB。
used_mem	int	实例已使用的内存大小，单位 MB。
io_await	real	实例所使用磁盘的 io_wait 值（10 秒均值）。
io_util	real	实例所使用磁盘的 io_util 值（10 秒均值）。
disk_read	real	实例所使用磁盘的读速率（10 秒均值），单位 KB/s。
disk_write	real	实例所使用磁盘的写速率（10 秒均值），单位 KB/s。
process_read	bigint	实例对应进程从磁盘读数据的读速率(不包括从磁盘 pagecache 中读取的字节数，10 秒均值)，单位 KB/s。
process_write	bigint	实例对应进程向磁盘写数据的写速率(不包括向磁盘 pagecache 中写入的字节数，10 秒均值)，单位

名称	类型	描述
		KB/s。
logical_read	bigint	CN 实例：不统计。 DN 实例：该实例在本次统计间隙（10 秒）内逻辑读字节速率，单位 KB/s。
logical_write	bigint	CN 实例：不统计。 DN 实例：该实例在本次统计间隙（10 秒）内逻辑写字节速率，单位 KB/s。
read_counts	bigint	CN 实例：不统计。 DN 实例：该实例在本次统计间隙（10 秒）内逻辑读操作次数之和，单位次。
write_counts	bigint	CN 实例：不统计。 DN 实例：该实例在本次统计间隙（10 秒）内逻辑写操作次数之和，单位次。

14.2.4 GS_WLM_OPERATOR_INFO

GS_WLM_OPERATOR_INFO 系统表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表中的数据。当设置 GUC 参数 enable_resource_record 为 on 时，系统会定时将 14.3.76 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。

说明

- 此系统表的 schema 是 dbms_om。
- 此系统表在 gaussdb 数据库中有分布列，分布列是 queryid，其它数据库中无分布列。
- pg_catalog 下存在 GS_WLM_OPERATOR_INFO 视图。

表14-4 GS_WLM_OPERATOR_INFO 的字段

名称	类型	描述
nodename	text	执行语句的 CN 实例名称。
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。

名称	类型	描述
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。
min_spill_size	integer	若发生下盘，所有下盘 DN 的最小下盘数据量(MB)，默认为 0。
max_spill_size	integer	若发生下盘，所有下盘 DN 的最大下盘数据量(MB)，默认为 0。
average_spill_size	integer	若发生下盘，所有下盘 DN 的平均下盘数据量(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

14.2.5 GS_WLM_SESSION_INFO

GS_WLM_SESSION_INFO 系统表显示所有 CN 执行作业结束后的负载管理记录。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，系统会定时将 14.3.79 GS_WLM_SESSION_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。具体的字段请参考表 14-127。

说明

- 此系统表的 schema 是 dbms_om。
- 此系统表在 gaussdb 数据库中有分布列，分布列是 queryid，其它数据库中无分布列。
- pg_catalog 下存在 GS_WLM_SESSION_INFO 视图。

14.2.6 GS_WLM_USER_RESOURCE_HISTORY

GS_WLM_USER_RESOURCE_HISTORY 系统表存储与用户使用资源相关的信息，仅在 CN 上有效。该系统表的每条记录都是对应时间点某用户的资源使用情况，包括：内存、CPU 核数、存储空间、临时空间、算子落盘空间、逻辑 IO 流量、逻辑 IO 次数和逻辑 IO 速率信息。其中，内存、CPU、IO 相关监控项仅记录用户复杂作业的资源使用情况。

GS_WLM_USER_RESOURCE_HISTORY 系统表的数据来源于 14.3.158 PG_TOTAL_USER_RESOURCE_INFO 视图。

表14-5 GS_WLM_USER_RESOURCE_HISTORY 字段

名称	类型	描述
username	text	用户名。
timestamp	timestamp with time zone	时间戳。
used_memory	int	正在使用的内存大小，单位 MB。
total_memory	int	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	real	正在使用的 CPU 核数。
total_cpu	int	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	已使用的临时存储空间大小，单位 KB。
total_temp	bigint	可使用的临时存储空间大小，单位 KB，值为-1 表示

名称	类型	描述
_space		未限制最大临时存储空间。
used_spill_space	bigint	已使用的算子落盘存储空间大小，单位 KB。
total_spill_space	bigint	可使用的算子落盘存储空间大小，单位 KB，值为-1表示未限制最大算子落盘存储空间。
read_kbytes	bigint	监控周期内，读操作的字节流量，单位 KB。
write_kbytes	bigint	监控周期内，写操作的字节流量，单位 KB。
read_counts	bigint	监控周期内，读操作的次数，单位次。
write_counts	bigint	监控周期内，写操作的次数，单位次。
read_speed	real	监控周期内，读操作的字节速率，单位 KB/s。
write_speed	real	监控周期内，写操作的字节速率，单位 KB/s。

14.2.7 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字、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14-6 PG_AGGREGATE 字段

名字	类型	引用	描述
aggfnoid	regproc	14.2.46 PG_PROC.oid	此聚集函数的 14.2.46 PG_PROC OID。
aggtransfn	regproc	14.2.46 PG_PROC.oid	转换函数。
aggcollectfn	regproc	14.2.46 PG_PROC.oid	收集函数。
aggfinalfn	regproc	14.2.46 PG_PROC.oid	最终处理函数（如果没有则为 0）。
aggsortop	oid	14.2.42 PG_OPERATOR.oid	关联排序操作符（如果没有则为 0）。
aggtranstype	oid	14.2.68 PG_TYPE.oid	此聚集函数的内部转换（状态）数据的数据类型。
agginitval	text	-	转换状态的初始值。这是一个

名字	类型	引用	描述
			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。

14.2.8 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14-7 PG_AM 字段

名字	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择才会显示）。
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。
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。
amcanbackward	boolean	-	访问方式是否支持向后扫描。
amcanunique	boolean	-	访问方式是否支持唯一索引。
amcanmulticol	boolean	-	访问方式是否支持多字段索引。
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。
amsearcharray	boolean	-	访问方式是否支持 ScalarArrayOpExpr 搜索。

名字	类型	引用	描述
amsearchnulls	boolean	-	访问方式是否支持 IS NULL/NOT NULL 搜索。
amstorage	boolean	-	允许索引存储的数据类型与列的数据类型是否不同。
amclusterable	boolean	-	是否允许在一个这种类型的索引上集群。
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。
amkeytype	oid	14.2.68 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	14.2.46 PG_PROC.oid	“插入此行”函数。
ambeginscan	regproc	14.2.46 PG_PROC.oid	“准备索引扫描” 函数。
amgettuple	regproc	14.2.46 PG_PROC.oid	“下一个有效行”函数，如果没有则为 0。
amgetbitmap	regproc	14.2.46 PG_PROC.oid	“抓取所有的有效行” 函数，如果没有则为 0。
amrescan	regproc	14.2.46 PG_PROC.oid	“（重新）开始索引扫描”函数。
amendscan	regproc	14.2.46 PG_PROC.oid	“索引扫描后清理” 函数。
ammarkpos	regproc	14.2.46 PG_PROC.oid	“标记当前扫描位置”函数。
amrestrpos	regproc	14.2.46 PG_PROC.oid	“恢复已标记的扫描位置”函数。
ammerge	regproc	14.2.46 PG_PROC.oid	“归并多个索引对象”函数。
ambuild	regproc	14.2.46 PG_PROC.oid	“建立新索引”函数。
ambuildempty	regproc	14.2.46 PG_PROC.oid	“建立空索引”函数。
ambulkdelete	regproc	14.2.46 PG_PROC.oid	批量删除函数。
amvacuumcleanup	regproc	14.2.46 PG_PROC.oid	VACUUM 后的清理函数。
amcanreturn	regproc	14.2.46	检查是否索引支持唯一索引扫描的

名字	类型	引用	描述
		PG_PROC.oid	函数，如果没有则为 0。
amcostestimate	regproc	14.2.46 PG_PROC.oid	估计一个索引扫描开销的函数。
amoptions	regproc	14.2.46 PG_PROC.oid	用于分析和验证索引的 reloptions 函数。

14.2.9 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14-8 PG_AMOP 字段

名字	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择才会显示）。
amopfamily	oid	14.2.43 PG_OPFAMILY.oid	该项的操作符族。
amoplefttype	oid	14.2.68 PG_TYPE.oid	操作符的左输入类型。
amoprighttype	oid	14.2.68 PG_TYPE.oid	操作符的右输入类型。
amopstrategy	smallint	-	操作符策略数。
amoppurpose	"char"	-	操作符目的，s 为搜索或 o 为排序。
amopopr	oid	14.2.42 PG_OPERATOR.oid	该操作符的 OID。
amopmethod	oid	14.2.8 PG_AM.oid	使用此操作符族的索引访问方法。
amopsortfamily	oid	14.2.43 PG_OPFAMILY.oid	如果是一个排序操作符，则该项会按照 btree 操作符族排序；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。

14.2.10 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14-9 PG_AMPROC 字段

名字	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择才会显示）
amprocfamily	oid	14.2.43 PG_OPFAMILY.oid	该项的操作符族
amproclefttype	oid	14.2.68 PG_TYPE.oid	相关操作符的左输入数据类型
amprocrighttype	oid	14.2.68 PG_TYPE.oid	相关操作符的右输入数据类型
amprocnum	smallint	-	支持过程编号
amproc	regproc	14.2.46 PG_PROC.oid	过程的 OID

amproclefttype 和 amprocrighttype 字段的习惯解释，标识一个特定支持过程所支持的操作符的左右输入类型。对于某些访问方式，匹配支持过程本身的输入数据类型，对其他的则不会匹配。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。

14.2.11 PG_ATTRDEF

PG_ATTRDEF 系统表存储字段的默认值。

表14-10 PG_ATTRDEF 字段

名称	类型	描述
adrelid	oid	该字段所属的表
adnum	smallint	字段编号
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

14.2.12 PG_ATTRIBUTE

PG_ATTRIBUTE 系统表存储关于表字段的信息。

表14-11 PG_ATTRIBUTE 字段

名称	类型	描述
attrelid	oid	该字段所属的表。
attname	name	字段名。
atttypid	oid	字段类型。
attstattarget	integer	控制 ANALYZE 为该字段设置的统计细节的级别。 • 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 pg_type.typlen 的拷贝。
attnum	smallint	字段编号。
attndims	integer	如果该字段是数组，该值表示数组的维数，否则是 0 。
attcacheoff	integer	在磁盘上总是-1 ，但是如果加载入内存中的行描述器中，它可能会被更新为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，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	pg_type.typbyval 字段值的拷贝。
attstorage	"char"	pg_type.typstorage 字段值的拷贝。
attalign	"char"	pg_type.typalign 字段值的拷贝。
attnotnull	boolean	代表一个非空约束。可以改变这个字段来打开或者关闭该约束。
atthasdef	boolean	该字段是否存在缺省值，此时它对应 pg_attrdef 表里实际定义此值的记录。

名称	类型	描述
attisdropped	boolean	该字段是否已经被删除，不再有效。如果被删除，该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	该字段是否局部定义在对象中。一个字段可以同时是局部定义和继承的。
attcmprmode	tinyint	对某一列指定压缩方式。压缩方式包括： • ATT_CMPR_NOCOMPRESS • ATT_CMPR_DELTA • ATT_CMPR_DICTIONARY • ATT_CMPR_PREFIX • ATT_CMPR_NUMSTR
attinhcount	integer	该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。
attoptions	text[]	属性级可选项。
attfdwoptions	text[]	属性级外数据选项。
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	该字段的 kv_type 属性。取值如下： • 0 表示默认值，用于非时序表 • 1 表示维度属性(TSTAG)，仅用于时序表 • 2 表示指标属性(TSFIELD)，仅用于时序表 • 3 时间属性(TSTIME)，仅用于时序表

14.2.13 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14-12 PG_AUTHID 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择才会显示）。
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。
rolinherit	boolean	角色是否自动继承其所属角色的权限。
rolcreaterole	boolean	角色是否可以创建更多角色。
rolcreatedb	boolean	角色是否可以创建数据库。
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。
rolcanlogin	boolean	角色是否可以登录，即该角色是否能够作为初始会话授权标识符。
rolreplication	boolean	角色是一个复制的角色（适配作用，没有实际的功能）。
rolauditadmin	boolean	审计用户。
rolsystemadmin	boolean	管理员用户。
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(可能是加密的)，如果没有口令，则为 NULL。
rolvalidbegin	timestamp with time zone	帐户的有效开始时间，如果没有开始时间，则为 NULL。
rolvaliduntil	timestamp with time zone	帐户的有效结束时间，如果没有结束时间，则为 NULL。
rolrespool	name	用户所能够使用的 resource pool。
roluseft	boolean	角色是否可以操作外表。
rolparentid	oid	用户所在组用户的 OID。
roltabspace	Text	用户永久表存储空间限额。
rolkind	char	特殊用户种类，包括私有用户、逻辑集群管理员、普通用户。
rolnodegroup	oid	用户所关联的 Node Group OID，该 Node Group 必须是逻辑集群。

名称	类型	描述
roltempspace	Text	用户临时表存储空间限额。
rolspillspace	Text	用户算子落盘空间限额。
rolexcpdata	text	保留字段未使用。
rolauthinfo	text	用户采用 LDAP 认证时的额外信息。如果是其他认证模式，则为 NULL。
rolpwdexpire	integer	用户口令过期时间，在口令未过期时，用户自己修改口令。过期后请管理员修改口令。-1 表示没有过期时间限制。
rolpwdtime	timestamp with time zone	口令的创建时间

14.2.14 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14-13 PG_AUTH_HISTORY 字段

名称	类型	描述
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，使用 MD5、SHA256 加密或者不加密

14.2.15 PG_AUTH_MEMBERS

PG_AUTH_MEMBERS 系统表存储显示角色之间的成员关系。

表14-14 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 角色的成员角色 ID
grantor	oid	赋予此成员关系的角色 ID
admin_option	boolean	如果有权限可以把 ROLEID 角色的成员关系赋予其他角色，则为真

14.2.16 PG_CAST

PG_CAST 系统表存储数据类型之间的转化关系。

表14-15 PG_CAST 字段

名称	类型	描述
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。0 表示不需要转化函数。
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 表示只能进行隐式转化。 • a 表示类型间同时支持隐式和显式转化。
castmethod	"char"	转化方法： • f 表示使用 castfunc 字段中指定的函数进行转化。 • b 表示类型间是二进制强制转化，不使用 castfunc。

14.2.17 PG_CLASS

PG_CLASS 系统表存储数据库对象信息及其之间的关系。

表14-16 PG_CLASS 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择才会显示）。
relname	name	表、索引、视图等对象的名称。
relnamespace	oid	包含该关系的命名空间的 OID。
reltype	oid	对应该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）。

名称	类型	描述
relfilenode	oid	该关系在磁盘上的文件的名称，如果没有则为 0。
reltablespace	oid	该关系存储所在的表空间。如果为 0，则使用该数据库的缺省表空间。如果关系无磁盘文件，该字段无意义。
relpages	double precision	以页(大小为 BLCKSZ)为单位的此表在磁盘上的大小，只是优化器使用的一个近似值。
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数。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	与此表关联的 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。
reldeltarelid	oid	Delta 表的 OID。 Delta 表附属于列存表。用于存储数据导入过程中的甩尾数据。
reldeltaidx	oid	Delta 表的索引表 OID。
relcudescrelid	oid	CU 描述表的 OID。 CU 描述表（Desc 表）附属于列存表。用于控制表目录中存储数据的可见性。
relcudescidx	oid	CU 描述表的索引表 OID。
relhasindex	boolean	如果对象是一个表且至少有（或者最近建有）一个索引，则为真。 由 CREATE INDEX 设置，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，会清理 relhasindex 字段，将 relhasindex 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。只有某些系统表（比如 pg_database）是共享的。
relpersistence	"char"	• p 表示永久表。 • u 表示非日志表。 • t 表示临时表。

名称	类型	描述
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。
relnatts	smallint	关系中用户字段数目（除了系统字段以外）。在 pg_attribute 里肯定有相同数目对应行。
relchecks	smallint	表上检查约束的数目，参见 14.2.19 PG_CONSTRAINT。
relhasoids	boolean	如果为关系中每行都生成一个 OID，则为真。
relhaspkey	boolean	如果该表有一个（或曾有）主键，则为真。
relhasrules	boolean	如果表有规则，则为真。是否有规则可参考系统表 14.2.53 PG_REWRITE。
relhastriggers	boolean	如果表有（或曾有）触发器，则为真。参见 14.2.62 PG_TRIGGER。
relhassubclass	boolean	如果表有（或曾有）任何继承的子表，则为真。
relcmprs	tinyint	表示是否启用表的压缩特性。需要特别注意，当且仅当批量插入才会触发压缩，普通的 CRUD 并不能够触发压缩。 • 0 表示其他不支持压缩的表（主要是指系统表，不支持压缩属性的修改操作）。 • 1 表示表数据的压缩特性为 NOCOMPRESS 或者无指定关键字。 • 2 表示表数据的压缩特性为 COMPRESS。
relhasclusterkey	boolean	是否有局部聚簇存储。
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。 • v 表示该表为 HDFS 的 Value 分区表。
relfrozenxid	xid32	该表中所有在此之间的事务 ID 已经被替换为一个固定

名称	类型	描述
		的（"frozen"）事务 ID。该字段用于跟踪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 0（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 表示授予该权限的角色。权限的参数说明请参见表 14-17。

reloptions	text[]	特定的访问方法选项，用"keyword=value"字符串形式表示。
relfrozenxid64	xid	该表中所有在此之前的事务 ID 已经被替换为一个固定的（"frozen"）事务 ID。该字段用于跟踪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 0（InvalidTransactionId）。

表14-17 权限的参数说明

参数	参数说明
r	SELECT（读）
w	UPDATE（写）
a	INSERT（插入）
d	DELETE
D	TRUNCATE
x	REFERENCES
t	TRIGGER
X	EXECUTE
U	USAGE
C	CREATE
c	CONNECT
T	TEMPORARY

参数	参数说明
A	ANALYZE|ANALYSE
L	ALTER
P	DROP
v	VACUUM
arwdDxtA, vLP	ALL PRIVILEGES（用于表）
*	给前面权限的授权选项

应用示例

查看某张表的 oid 及 relfilenode：

```
select oid,relname,relfilenode from pg_class where relname = 'table_name';
```

统计行存表数量：

```
select 'row count:'||count(1) as point from pg_class where relkind = 'r' and oid > 16384 and reloptions::text not like '%column%' and reloptions::text not like '%internal_mask%';
```

统计列存表数量：

```
select 'column count:'||count(1) as point from pg_class where relkind = 'r' and oid > 16384 and reloptions::text like '%column%';
```

14.2.18 PG_COLLATION

PG_COLLATION 系统表描述可用的排序规则，本质上从一个 SQL 名字映射到操作系统本地类别。

表14-18 PG_COLLATION 字段

名字	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择才会显示）
collname	name	-	排序规则名（每个命名空间和编码唯一）
collnamespace	oid	14.2.38 PG_NAMESPACE.oid	包含该排序规则的命名空间的 OID
collowner	oid	14.2.13 PG_AUTHID.oid	排序规则的所有者
collencoding	integer	-	排序规则可用的编码，如果适

名字	类型	引用	描述
			用于任意编码为-1
collcollate	name	-	排序规则对象的 LC_COLLATE
collctype	name	-	排序规则对象的 LC_CTYPE

14.2.19 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14-19 PG_CONSTRAINT 字段

名称	类型	描述
conname	name	约束名称（不一定是唯一的）。
connamespace	oid	包含约束的命名空间的 OID。
contype	"char"	• c = 检查约束 • f = 外键约束 • p = 主键约束 • u = 唯一约束 • t = 触发器约束
condeferrable	boolean	该约束是否可以推迟。
condeferred	boolean	缺省时该约束是否可以推迟。
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。
conrelid	oid	该约束所在的表；如果不是表约束则为 0。
contypid	oid	该约束所在的域；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

名称	类型	描述
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 = 简单（未指定）
conislocal	boolean	是否是为关系创建的本地约束。
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。
consoft	boolean	是否为信息约束(Informational Constraint)。
conopt	boolean	是否使用信息约束优化执行计划。
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，则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，则是其表达式的内部形式。
consrc	text	如果是检查约束，则是表达式的人类可读形式。

须知

- 当被引用的对象改变时，consrc 不能被更新。例如，它不会跟踪字段的重命名。最好还是使用 pg_get_constraintdef()来抽取一个检查约束的定义，而不是依赖这个字段。
- pg_class.relchecks 需要和每个关系在此目录中的检查约束数量保持一致。

14.2.20 PG_CONVERSION

PG_CONVERSION 系统表描述编码转换信息。

表14-20 PG_CONVERSION 字段

名字	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择才会显示）
conname	name	-	转换名（在一个命名空间里唯一）
connamespace	oid	14.2.38 PG_NAMESPACE.oid	包含此转换的命名空间的 OID
conowner	oid	14.2.13 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	14.2.46 PG_PROC.oid	转换过程
condefault	boolean	-	如果为缺省转换则为真

14.2.21 PG_DATABASE

PG_DATABASE 系统表存储关于可用数据库的信息。

表14-21 PG_DATABASE 字段

名称	类型	描述
datname	name	数据库名称。
datdba	oid	数据库所有者，通常为其创建者。
encoding	integer	数据库的字符编码方式。

名称	类型	描述
		pg_encoding_to_char()可以将此编号转换为编码名称。
datcollate	name	数据库使用的排序顺序。
datctype	name	数据库使用的字符分类。
datistemplate	boolean	是否允许作为模板数据库。
datallowconn	boolean	如果为假，则没有用户可以连接到这个数据库。这个字段用于保护 template0 数据库不被更改。
datconnlimit	integer	该数据库上允许的最大并发连接数，-1 表示无限制。
datlastsysoid	oid	数据库里最后一个系统 OID 。
datfrozenxid	xid32	用于跟踪该数据库是否需要为了防止事务 ID 重叠而进行清理。 为保持前向兼容，保留此字段，新增 datfrozenxid64 用于记录此信息。
dattablespace	oid	数据库的缺省表空间。
datcompatibility	name	数据库兼容模式。
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。

应用实例

查看某数据库的所有者、兼容模式及访问权限：

```
SELECT datname, datdba,datcompatibility,datacl from pg_database where
datname='database_name';
```

14.2.22 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14-22 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。

名称	类型	描述
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。

14.2.23 PG_DEFAULT_ACL

PG_DEFAULT_ACL 系统表存储为新建对象设置的初始权限。

表14-23 PG_DEFAULT_ACL 字段

名称	类型	描述
defaclrole	oid	与此权限相关的角色 ID。
defaclnamespace	oid	与此权限相关的命名空间，如果没有，则为 0。
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。
defaclacl	aclitem[]	创建该类型时所拥有的访问权限。

应用示例

查看新建用户 role1 的初始权限：

```
select * from PG DEFAULT ACL;
 defaclrole | defaclnamespace | defaclobjtype |    defaclacl
------------+-----------------+---------------+-----------------
      16820 |         16822 | r             | {role1=r/user1}
```

也可使用如下语句进行转换后更直观的查看：

```
SELECT pg_catalog.pg_get_userbyid(d.defaclrole) AS "Granter",  n.nspname AS
"Schema",  CASE d.defaclobjtype WHEN 'r' THEN 'table' WHEN 'S' THEN 'sequence' WHEN
'f' THEN 'function' WHEN 'T' THEN 'type' END AS "Type",
pg_catalog.array_to_string(d.defaclacl, E', ') AS "Access privileges" FROM
pg_catalog.pg_default_acl d LEFT JOIN pg_catalog.pg_namespace n ON n.oid =
d.defaclnamespace ORDER BY 1, 2, 3;
```

输出结果如下，表示通过用户 user1 授予用户 role1 对模式“user1”有读的权限。

```
 Granter | Schema | Type  | Access privileges
---------+--------+-------+-------------------
```

```
 user1   | user1  | table | role1=r/user1
(1 row)
```

14.2.24 PG_DEPEND

PG_DEPEND 系统表记录数据库对象之间的依赖关系。这些信息允许 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。

另请参考 14.2.55 PG_SHDEPEND，对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提供了相似的功能。

表14-24 PG_DEPEND 字段

名字	类型	引用	描述
classid	oid	14.2.17 PG_CLASS.oid	依赖对象所在系统表的 OID。
objid	oid	任意 OID 属性	指定依赖对象的 OID。
objsubid	integer	-	对于表字段，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，此字段是 0。
refclassid	oid	14.2.17 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，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，此字段是零。
deptype	"char"	-	定义此依赖关系特定语义的代码。

在所有情况下，一个 PG_DEPEND 记录表示被引用对象不能在没有删除依赖对象的情况下被删除。但是其中也有几种由 deptype 定义的情况：

- DEPENDENCY_NORMAL (n)：独立创建的对象之间的一般关系。依赖对象可以在不影响被引用对象的情况下删除。被引用对象只能通过指定 CASCADE 被删除，这种情况下依赖对象也会被删除。例如：一个表字段对其数据类型有一般依赖关系。
- DEPENDENCY_AUTO (a)：依赖对象可以和被引用对象分别删除，且在被引用对象被删除时应自动被删除（不管是 RESTRICT 或 CASCADE 模式）。例如：一个表上的命名约束是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)：依赖对象作为被引用对象过程的一部分创建，且是其内部实现的一部分。 DROP 依赖对象是不会直接允许的（会给户发出一个针对被引用对象的 DROP）。不管是否指定 CASCADE，一个被引用对象的 DROP 将被传播来删除其依赖对象。例如：一个用于强制外键约束的触发器将被设置为内部依赖于其约束的 14.2.19 PG_CONSTRAINT 项。

- DEPENDENCY_EXTENSION (e)：依赖对象作为被依赖对象 extension 的一个成员（请参见 14.2.27 PG_EXTENSION）。依赖对象可以通过在被依赖对象上 DROP EXTENSION 删除。在功能上，这种依赖类型和内部依赖的作用相同，其存在只是为了清晰和简化 gs_dump。
- 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，因此这个对象决不能被删除。这种类型的记录只有在 initdb 的时候创建。有依赖对象的字段都为 0。

应用示例

查询名为 serial1 的数据库对象 sequence 和哪个表有依赖关系。

1. 先通过系统表 PG_CLASS 查询序列名为 serial1 的 oid。

```
SELECT oid FROM pg_class WHERE relname ='serial1';
  oid
-------
 17815
(1 row)
```

2. 使用系统表 PG_DEPEND 根据所查询的序列 serial1 的 oid 获取依赖该序列的对象。

```
SELECT * FROM pg_depend WHERE objid ='17815';
 classid | objid | objsubid | refclassid | refobjid | refobjsubid | deptype
---------+-------+----------+------------+----------+-------------+---------
    1259 | 17815 |        0 |       2615 |     2200 |           0 | n
    1259 | 17815 |        0 |       1259 |    17812 |           1 | a
(2 rows)
```

3. 根据字段 refobjid 获取依赖该序列 serial1 的表的 OID，并查询到表名。其结果表示，序列 serial1 依赖于表 customer_address。

```
SELECT relname FROM pg_class where oid='17812';
     relname
------------------
 customer_address
(1 row)
```

14.2.25 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 14.2.56 PG_SHDESCRIPTION，用于记录整个集群范围内共享对象的注释。

表14-25 PG_DESCRIPTION 字段

名字	类型	引用	描述
objoid	oid	任意 OID 属性	描述所属对象的 OID。
classoid	oid	14.2.17 PG_CLASSoid	对象显示的系统表的 OID。

名字	类型	引用	描述
objsubid	integer	-	对于一个表字段的注释，为字段号（objoid 和 classoid 指向表自身）。对于其它对象类型，为 0。
description	text	-	对该对象描述的任意文本。

14.2.26 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14-26 PG_ENUM 字段

名字	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择才会显示）
enumtypid	oid	14.2.68 PG_TYPE.oid	包含此枚举值的 pg_type 项的 OID
enumsortorder	real	-	此枚举值在其枚举类型中的排序位置
enumlabel	name	-	此枚举值的文本标签

PG_ENUM 行的 OID 值遵循一种特殊规则：OID 的数值被保证按照其枚举类型一样的排序顺序排序。即如果两个偶数 OID 属于同一枚举类型，那么较小的 OID 必然具有较小 enumsortorder 值。奇数 OID 不需要遵循排序顺序。这种规则使得枚举比较例程在很多常见情况下可以避免系统目录查找。创建和修改枚举类型的例程尝试尽可能地为枚举值分配偶数 OID。

当一个枚举类型被创建后，其成员会被分配排序顺序位置 1 到 n。但是后面增加的成员可能会分配负值或者分数值的 enumsortorder。对于这些值的唯一要求是它们必须被正确的排序且在每个枚举类型中保持唯一。

14.2.27 PG_EXTENSION

PG_EXTENSION 系统表存储关于所安装扩展的信息。GaussDB(DWS)默认有十二个扩展，即 PLPGSQL、DIST_FDW、FILE_FDW、HDFS_FDW 、HSTORE、PLDBGAPI、DIMSEARCH、PACKAGES、GC_FDW、UUID-OSSP、LOG_FDW 和 ROACH_API。

表14-27 PG_EXTENSION

名称	类型	描述
extname	name	扩展名

名称	类型	描述
extowner	oid	扩展的所有者
extnamespace	oid	扩展导出对象的命名空间
extrelocatable	boolean	如果扩展能够重定位到其他 schema，则为 true
extversion	text	扩展的版本号
extconfig	oid[]	扩展的配置信息
extcondition	text[]	扩展配置信息的过滤条件

14.2.28 PG_EXTENSION_DATA_SOURCE

PG_EXTENSION_DATA_SOURCE 系统表存储外部数据源对象的信息。一个外部数据源对象（Data Source）包含了外部数据库的一些口令编码等信息，主要配合 Extension Connector 使用。

表14-28 PG_EXTENSION_DATA_SOURCE 字段

名字	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择才会显示）。
srcname	name	-	外部数据源对象的名称。
srcowner	oid	PG_AUTHID.oid	外部数据源对象的所有者。
srctype	text	-	外部数据源对象的类型，缺省为空。
srcversion	text	-	外部数据源对象的版本，缺省为空。
srcacl	aclitem[]	-	访问权限。
srcoptions	text[]	-	外部数据源对象的指定选项，使用“keyword=value”格式的字符串。

14.2.29 PG_FOREIGN_DATA_WRAPPER

PG_FOREIGN_DATA_WRAPPER 系统表存储外部数据封装器定义。一个外部数据封装器是在外部服务器上驻留外部数据的机制，是可以访问的。

表14-29 PG_FOREIGN_DATA_WRAPPER 字段

名字	类型	引用	描述

名字	类型	引用	描述
oid	oid	-	行标识符(隐藏属性，必须明确选择才会显示)。
fdwname	name	-	外部数据封装器名。
fdwowner	oid	14.2.13 PG_AUTHID.oid	外部数据封装器的所有者。
fdwhandler	oid	14.2.46 PG_PROC.oid	引用一个负责为外部数据封装器提供扩展例程的处理函数。如果没有提供处理函数则为 0。
fdwvalidator	oid	14.2.46 PG_PROC.oid	引用一个验证器函数，这个验证器函数负责验证给予外部数据封装器的选项、外部服务器选项和使用外部数据封装器的用户映射的有效性。如果没有提供验证器函数则为 0。
fdwacl	aclitem[]	-	访问权限。
fdwoptions	text[]	-	外部数据封装器指定选项，使用“keyword=value”格式的字符串。

14.2.30 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14-30 PG_FOREIGN_SERVER 字段

名字	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择才会显示）
srvname	name	-	外部服务器名
srvowner	oid	14.2.13 PG_AUTHID.oid	外部服务器的所有者
srvfdw	oid	14.2.29 PG_FOREIGN_DATA_WRAPPER.oid	此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

名字	类型	引用	描述
			“keyword=value”格式的字符串

14.2.31 PG_FOREIGN_TABLE

PG_FOREIGN_TABLE 系统表存储外部表的辅助信息。

表14-31 PG_FOREIGN_TABLE 字段

名称	类型	描述
ftrelid	oid	外部表的 OID
ftserver	oid	外部表的所在服务器的 OID
ftwriteonly	boolean	外部表是否可写
ftoptions	text[]	外部表的可选项

14.2.32 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14-32 PG_INDEX 字段

名称	类型	描述
indexrelid	oid	此索引的 pg_class 项的 OID。
indrelid	oid	使用该索引的表在 pg_class 项的 OID。
indnatts	smallint	索引中的字段数目。
indisunique	boolean	如果为真，为唯一索引。
indisprimary	boolean	如果为真，该索引为该表的主键。该字段为真时，indisunique 也总是为真。
indisexclusion	boolean	如果为真，该索引支持排他约束。
indimmediate	boolean	如果为真，在插入数据时会立即执行唯一性检查。
indisclustered	boolean	如果为真，则该表最后以此索引进行了聚簇。
indisusable	boolean	如果为真，此索引对 INSERT/SELECT 可用。
indisvalid	boolean	如果为真，则此索引可以用于查询。如果为假，则该索引可能不完整，仍然必须在 INSERT/UPDATE 操作时进行更新，但不能安

名称	类型	描述
		全的被用于查询。如果是唯一索引，则唯一属性也不为真。
indcheckxmin	boolean	如果为真，查询不能使用此索引，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可见的不兼容行断开的 HOT 链。
indisready	boolean	如果为真，表示此索引对插入数据可用。如果为假，在插入或修改数据时忽略此索引。
indkey	int2vector	这是一个包含 indnatts 值的数组，这些数组值表示此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。数组中的 0 表示对应的索引属性是一个表字段上的表达式，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	对于索引键中的每个字段，该字段都包含要使用的操作符类的 OID，详见 14.2.41 PG_OPCLASS。
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	部分索引谓词的表达式树（以 nodeToString()的形式表现）。如果不是部分索引，则为空。

14.2.33 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14-33 PG_INHERITS 字段

名字	类型	引用	描述
inhrelid	oid	14.2.17 PG_CLASS.oid	子表的 OID。
inhparent	oid	14.2.17 PG_CLASS.oid	父表的 OID。

名字	类型	引用	描述
inhseqno	integer	-	如果一个子表存在多个直系父表（多重继承），这个数字表示此继承字段的排列顺序。计数从 1 开始。

14.2.34 PG_JOBS

PG_JOBS 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_jobs 系统表中的时间，当任务到期会触发任务的执行。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14-34 PG_JOBS 字段

名字	类型	描述
job_id	integer	作业 ID，主键，是唯一的（有唯一索引）。
what	text	作业内容。
log_user	oid	创建者的 UserID。
priv_user	oid	作业执行者的 UserID。
job_db	oid	标识作业执行的数据库 OID。
job_nsp	oid	标识作业运行时所在的命名空间 OID。
job_node	oid	标识当前作业是在哪个 CN 上创建和执行。
is_broken	boolean	标识当前作业是否为失效状态，当作业连续失败 16 次后，会将 is_broken 自动设置为 true，后续不再执行该作业。
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	下次定时执行任务的时间，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。
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。

名字	类型	描述
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。

14.2.35 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14-35 PG_LANGUAGE 字段

名字	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择才会显示）。
lanname	name	-	语言的名称。
lanowner	oid	14.2.13 PG_AUTHID.oid	语言的所有者。
lanispl	boolean	-	内部语言为假（比如 SQL），用户定义语言为真。目前，gs_dump 仍然使用该字段判断哪些语言需要转储，但可能在将来会被其它机制所取代。
lanpltrusted	boolean	-	如果是可信语言则为真，即系统相信它不会被授予任何正常 SQL 执行环境之外的权限。只有初始用户可以在用非可信的语言创建函数。
lanplcallfoid	oid	14.2.46 PG_PROC.oid	对于非内部语言，这是指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	14.2.46 PG_PROC.oid	此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 0。
lanvalidator	oid	14.2.46 PG_PROC.oid	此字段引用一个语言校验器函数，它负责检查新创建函数的语法和有效性。如果没有提供校验器，则为 0。
lanacl	aclitem[]	-	访问权限。

14.2.36 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14-36 PG_LARGEOBJECT 字段

名字	类型	引用	描述
loid	oid	14.2.37 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 0 开始计算。
data	bytea	-	实际存储在大对象中的数据。这些数据不会超过 LOBLKSIZE 字节，且可能更小。

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。这种实现允许稀疏存储：页面可能丢失，并且可以比 LOBLKSIZE 字节少（即使它们不是对象的最后一页）。大对象中丢失的区域会被读为 0。

14.2.37 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14-37 PG_LARGEOBJECT_METADATA 字段

名字	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择才会显示）
lomowner	oid	14.2.13 PG_AUTHID.oid	大对象的所有者
lomacl	aclitem[]	-	访问权限

14.2.38 PG_NAMESPACE

PG_NAMESPACE 系统表存储命名空间，即存储 schema 相关的信息。

表14-38 PG_NAMESPACE 字段

名称	类型	描述
nspname	name	命名空间的名称。
nspowner	oid	命名空间的所有者。
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。
permspace	bigint	schema 永久表空间限额。
usedspace	bigint	schema 已用永久表空间大小。

14.2.39 PG_OBJECT

PG_OBJECT 系统表存储限定类型对象(object_type 中存在的类型)的创建用户、创建时间、最后修改时间和最后 analyze 时间。

表14-39 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 表示表，包括普通表和临时表 • 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	对象创建时间。
mtime	timestamp with time zone	对象最后修改时间，默认记录修改行为包括 ALTER 操作、COMMENT、GRANT/REVOKE 和 TRUNCATE 操作。 object_mtime_record_mode 参数可以细粒度控制 ALTER、COMMENT、GRANT/REVOKE 和 TRUNCATE 操作是否被记录。
last_analyze_time	timestamp with time	对象进行最后一次 analyze 的时间。

名称	类型	描述
	zone	

须知

- 仅针对用户正常操作行为进行记录，无法记录对象升级以前和 initdb 过程中的行为。
- ctime 和 mtime 的时间记录为本次操作的事务起始时间。
- 由扩容引起的对象修改时间也会被记录。

14.2.40 PG_OBSSCANINFO

PG_OBSSCANINFO 系统表定义了在云上加速场景中，使用加速集群时扫描 OBS 数据的运行时信息，每条记录对应一个 query 中单个 OBS 外表的运行时信息。

表14-40 PG_OBSSCANINFO 字段

名字	类型	引用	描述
query_id	bigint	-	查询标识
user_id	text	-	执行该查询的数据库用户
table_name	text	-	OBS 外表的表名
file_type	text	-	底层数据保存的文件格式
time_stamp	time_stam	-	扫描操作开始的时间
actual_time	double	-	扫描操作执行时间，单位为秒
file_scanned	bigint	-	扫描的文件数量
data_size	double	-	扫描的数据量，单位为字节
billing_info	text	-	保留字段

14.2.41 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14-41 PG_OPCLASS 字段

名字	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择才会显示）。
opcmethod	oid	14.2.8 PG_AM.oid	操作符类所属的索引访问方法。
opcname	name	-	操作符类的名称。
opcnamespace	oid	14.2.38 PG_NAMESPACE.oid	操作符类所属的命名空间。
opcowner	oid	14.2.13 PG_AUTHID.oid	操作符类所有者。
opcfamily	oid	14.2.43 PG_OPFAMILY.oid	包含此操作符类的操作符族。
opcintype	oid	14.2.68 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	14.2.68 PG_TYPE.oid	索引数据的类型，如果和 opcintype 相同则为 0。

一个操作符类的 opcmethod 必须匹配包含它的操作符族的 opfmethod。同样，对于任意给定的 opcmethod 和 opcintype 的组合，不能有超过一个 PG_OPCLASS 行有 opcdefault 为真。

14.2.42 PG_OPERATOR

PG_OPERATOR 系统表存储有关操作符的信息。

表14-42 PG_OPERATOR 字段

名字	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择才会显示）
oprname	name	-	操作符的名称
oprnamespace	oid	14.2.38 PG_NAMESPACE.oid	包含此操作符的命名空间的 OID
oprowner	oid	14.2.13 PG_AUTHID.oid	操作符所有者
oprkind	"char"	-	• b=infix =中缀（两边） • l=前缀（左边）

名字	类型	引用	描述
			• r=后缀（右边）
oprcanmerge	boolean	-	该操作符是否支持合并连接
oprcanhash	boolean	-	该操作符是否支持 Hash 连接
oprleft	oid	14.2.68 PG_TYPE.oid	左操作数的类型
oprright	oid	14.2.68 PG_TYPE.oid	右操作数的类型
oprresult	oid	14.2.68 PG_TYPE.oid	结果类型
oprcom	oid	14.2.42 PG_OPERATOR.oid	此操作符的交换符（如果存在）
oprnegate	oid	14.2.42 PG_OPERATOR.oid	此操作符的反转器（如果存在）
oprcode	regproc	14.2.46 PG_PROC.oid	实现该操作符的函数
oprrest	regproc	14.2.46 PG_PROC.oid	此操作符的约束选择性计算函数
oprjoin	regproc	14.2.46 PG_PROC.oid	此操作符的连接选择性计算函数

14.2.43 PG_OPFAMILY

PG_OPFAMILY 系统表定义操作符族。

每个操作符族是操作符和相关支持例程的集合，这些例程实现了为特定索引访问方法指定的语义。此外，按照访问方法指定的某种方式，一个族内的操作符都是“兼容的”。操作符族允许跨数据类型操作符与索引一起使用，并可以推理使用访问方法语义相关内容。

表14-43 PG_OPFAMILY 字段

名字	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择才会显示）
opfmethod	oid	14.2.8 PG_AM.oid	操作符族使用的索引方法
opfname	name	-	操作符族的名称
opfnamespace	oid	14.2.38 PG_NAMESPACE.oid	操作符族的命名空间
opfowner	oid	14.2.13 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 PG_OPFAMILY，而是在相关的 14.2.9 PG_AMOP，14.2.10 PG_AMPROC 和 14.2.41 PG_OPCLASS 中。

14.2.44 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14-44 PG_PARTITION 字段

名称	类型	描述
relname	name	分区表、分区、分区上 toast 表和分区索引的名称。
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。 'l'：列表分区。
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

名称	类型	描述
		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。
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。
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。

boundexprs	pg_node_tree	分区边界表达式。
		• 对于范围分区来说是分区上边界表达式。
		• 对于列表分区来说是分区边界枚举值集合。
		pg_node_tree 数据类型是不可读的，可用如下表达式 pg_get_expr 把当前字段单翻译为可读信息。
		`SELECT pg_get_expr(boundexprs, 0) FROM pg_partition WHERE relname = 'country_202201';`
		`pg_get_expr`
		`----------`
		`----------`

名称	类型	描述
		ROW(202201, 'city1'::text), ROW(202201, 'city2'::text) (1 row)

14.2.45 PG_PLTEMPLATE

PG_PLTEMPLATE 系统表存储过程语言的“模板”信息。

表14-45 PG_PLTEMPLATE 字段

名称	类型	描述
tmplname	name	该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。
tmpldbacreate	boolean	如果语言是由数据库所有者创建的，则为真。
tmplhandler	text	调用处理器函数的名称。
tmplinline	text	匿名块处理器的名称，如果没有则为 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）。

14.2.46 PG_PROC

PG_PROC 系统表存储函数或过程的信息。

表14-46 PG_PROC 字段

名称	类型	描述
proname	name	函数名。
pronamespace	oid	此函数所在命名空间的 OID。
proowner	oid	函数的所有者。
prolang	oid	实现语言或函数的调用接口。
procost	real	估计执行成本。
prorows	real	结果行估计数。
provariadic	oid	参数元素的数据类型。

名称	类型	描述
protransform	regproc	此函数的简化调用方式。
proisagg	boolean	函数是否为聚集函数。
proiswindow	boolean	函数是否为窗口函数。
prosecdef	boolean	函数是否为一个安全定义器（例如，一个“setuid”函数）。
proleakproof	boolean	函数有无其他影响。如果函数没有对参数进行防泄露处理，则会抛出错误。
proisstrict	boolean	如果任意调用参数为空，函数是否返回空值。这种情况下函数实际上根本不会被调用。非“strict”的函数必须准备处理空值输入。
proretset	boolean	函数是否返回一个集合（即，指定数据类型的多个数值）。
provolatile	"char"	说明该函数的结果是只倚赖于它的输入参数，或者还会被外接因素影响。 • i 表示“不可变的”（immutable）函数，对于相同的输入总是输出相同的结果。 • s 表示稳定的”（stable）函数，对于固定输入其结果在一次扫描里不变。 • v 表示“易变”（volatile）函数，其结果可能在任何时候都变化。
pronargs	smallint	参数个数。
pronargdefaults	smallint	有默认值的参数个数。
prorettype	oid	返回参数类型的 OID。
proargtypes	oidvector	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），因此也表现了函数的调用特征。
proallargtypes	oid[]	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，则这个字段就会为空。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"[]	函数参数模式的数组。 • i 表示 IN 参数 • o 表示 OUT 参数 • b 表示 INOUT 参数 如果所有参数都是 IN 参数，则这个字段为空。注

名称	类型	描述
		意此数组下标对应的是 proallargtypes 的位置，而不是 proargtypes。
proargnames	text[]	函数参数的名字的数组。没有名字的参数在数组里设置为空字符串。如果没有一个参数有名字，这个字段为空。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，或者一个链接符号，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。
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	函数默认值的位置，不局限于能最后几个参数才可有默认值。
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。如果是 fence 执行模式，函数的执行会在重新 fork 的进程中执行。默认值是 fence。
proshippable	boolean	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	该函数是否支持重载，主要针对 Oracle 风格的函数，默认值是 false。

应用示例

查询指定函数的 OID。例如，获取函数 justify_days 的 OID 为 1295。

```
SELECT oid FROM pg_proc where proname ='justify_days';
 oid
```

```
------
 1295
(1 row)
```

查询指定函数是否为聚集函数。例如，查询 justify_days 函数为非聚集函数。

```
SELECT proisagg FROM pg_proc where proname ='justify_days';
 proisagg
----------
 f
(1 row)
```

14.2.47 PG_RANGE

PG_RANGE 系统表存储关于范围类型的信息。

除了 14.2.68 PG_TYPE 里类型的记录。

表14-47 PG_RANGE 字段

名字	类型	引用	描述
rngtypid	oid	14.2.68 PG_TYPE.oid	范围类型的 OID。
rngsubtype	oid	14.2.68 PG_TYPE.oid	该范围类型的元素类型（子类型）的 OID。
rngcollation	oid	14.2.18 PG_COLLATION.oid	用于范围比较的排序规则的 OID，如果没有则为 0。
rngsubopc	oid	14.2.41 PG_OPCLASS.oid	用于范围比较的子类型的操作符类的 OID。
rngcanonical	regproc	14.2.46 PG_PROC.oid	转换范围类型为规范格式的函数的 OID，如果没有则为 0。
rngsubdiff	regproc	14.2.46 PG_PROC.oid	返回两个 double precision 元素值的不同的函数的 OID，如果没有则为 0。

rngsubopc（如果元素类型是可排序的，则加上 rngcollation）决定用于范围类型的排序顺序。当元素类型是时使用 rngcanonical 用于离散类型的元素类型。

14.2.48 PG_REDACTION_COLUMN

PG_REDACTION_COLUMN 系统表存储脱敏列的信息。

表14-48 PG_REDACTION_COLUMN 字段

名称	类型	描述
object_oid	oid	脱敏对象 OID

名称	类型	描述
column_attrno	smallint	脱敏列 attrno
function_type	integer	脱敏类型 说明 保留字段，仅为向前兼容低版本的脱敏列信息，可取值为 0（NONE）、1（FULL）。
function_parameters	text	脱敏类型为 partial 类型时的参数。（保留字段，无实际意义）
regexp_pattern	text	脱敏类型为 regexp 时，格式化字符串。（保留字段，无实际意义）
regexp_replace_string	text	脱敏类型为 regexp 时，替换串。（保留字段，无实际意义）
regexp_position	integer	脱敏类型为 regexp 时，起始替换位置。（保留字段，无实际意义）
regexp_occurrence	integer	脱敏类型为 regexp 时，替换次数。（保留字段，无实际意义）
regexp_match_parameter	text	脱敏类型为 regexp 时，正则控制参数。（保留字段，无实际意义）
column_description	text	脱敏列描述信息
function_expr	pg_node_tree	脱敏函数的内部表现形式。

14.2.49 PG_REDACTION_POLICY

PG_REDACTION_POLICY 系统表提供了脱敏对象的信息。

表14-49 PG_REDACTION_POLICY 字段

名称	类型	描述
object_oid	oid	脱敏对象 OID
policy_name	name	脱敏策略名称
enable	boolean	策略状态（开启、关闭） 说明 enable 的取值： • ture 表示开启 • false 表示关闭

名称	类型	描述
expression	pg_node_tree	策略生效表达式（针对用户）
policy_description	text	策略描述信息

14.2.50 PG_RELFILENODE_SIZE

PG_RELFILENODE_SIZE 系统表提供了文件级空间统计能力，表中的每一条记录则对应着磁盘上相应的物理文件和该文件的文件大小。

表14-50 PG_RELFILENODE_SIZE 字段

名称	类型	描述
databaseid	oid	该物理文件所属 database 对应的 OID。如果是跨库共享系统表，该值为 0。
tablespaceid	oid	该物理文件所属表空间对应的 OID。
relfilenode	oid	该物理文件的物理文件编号。
backendid	integer	创建该物理文件的后台线程号，通常为-1。
type	integer	该物理文件的文件类型。 • 0 为数据类型。 • 1 为 FSM 文件类型。 • 2 为 VM 文件类型。 • 3 为 BCM 文件类型。 • 大于 4 为列存表对应列的数据文件和 BCM 文件大小之和。
filesize	bigint	该物理文件的文件大小，单位 Byte。

14.2.51 PG_RLSPOLICY

PG_RLSPOLICY 系统表提供了行级访问控制策略的信息。

表14-51 PG_RLSPOLICY 字段

名称	类型	描述
polname	name	行访问控制策略名称。
polrelid	oid	行访问控制策略的表 OID。

名称	类型	描述
polcmd	char	行访问控制策略影响的 SQL 操作，包括：*(ALL)、r(SELECT)、w(UPDATE)、d(DELETE)。
polpermissive	boolean	行访问控制策略的类型。 说明 polpermissive 的取值： • ture 表示 PERMISSIVE，表示行访问控制策略是宽容性策略。 • false 表示 RESTRICTIVE，表示行访问控制策略是限制性策略。
polroles	oid[]	行访问控制策略影响的数据库用户 OID。
polqual	pg_node_tree	行访问控制策略的 SQL 条件表达式。

14.2.52 PG_RESOURCE_POOL

PG_RESOURCE_POOL 系统表提供了数据库资源池的信息。

表14-52 PG_RESOURCE_POOL 字段

名称	类型	描述
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 名字。
active_statements	integer	资源池上最大的并发数。
max_dop	integer	最大并发度，保留字段。
memory_limit	name	资源池最大的内存。
parentid	oid	父资源池 OID。
io_limits	integer	保留字段，无实际意义。
io_priority	text	保留字段，无实际意义。
is_foreign	boolean	表示资源池是否用于逻辑集群之外的用户。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。

14.2.53 PG_REWRITE

PG_REWRITE 系统表存储为表和视图定义的重写规则。

表14-53 PG_REWRITE 字段

名称	类型	描述
rulename	name	规则名称
ev_class	oid	使用该规则的表名
ev_attr	smallint	该规则适用的字段（目前总是为 0，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	如果是 INSTEAD 规则，则为真
ev_qual	pg_node_tree	规则条件的表达式树（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）

14.2.54 PG_SECLABEL

PG_SECLABEL 系统表存储数据对象上的安全标签。

14.2.57 PG_SHSECLABEL 的作用类似，只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。

表14-54 PG_SECLABEL 字段

名字	类型	引用	描述
objoid	oid	任意 OID 属性	此安全标签所属的对象的 OID
classoid	oid	14.2.17	此对象的系统目录的 OID

名字	类型	引用	描述
		PG_CLASS.oid	
objsubid	integer	-	出现在此对象中的列的序号
provider	text	-	与此标签相关的标签提供者
label	text	-	应用于此对象的安全标签

14.2.55 PG_SHDEPEND

PG_SHDEPEND 系统表记录数据库对象和共享对象（比如角色）之间的依赖关系。这些信息使得 GaussDB(DWS)可以确保对象在被删除时没有被其他对象引用。

14.2.24 PG_DEPEND 的作用类似，只是它是针对单个数据库中对象之间的依赖。

和大多数其他系统表不同，PG_SHDEPEND 在集群的所有数据库之间共享：每个数据库集群只有一个 PG_SHDEPEND，并非每个数据库一个。

表14-55 PG_SHDEPEND 字段

名字	类型	引用	描述
dbid	oid	14.2.21 PG_DATABASE.oid	依赖对象所在的数据库的 OID ，如果是共享对象，则为 0。
classid	oid	14.2.17 PG_CLASS.oid	依赖对象所在的系统表的 OID。
objid	oid	任意 OID 属性	指定的依赖对象的 OID。
objsubid	integer	-	对于一个表字段，为字段号（objid 和 classid 参考表本身）。对于所有其他对象类型，该字段为 0。
refclassid	oid	14.2.17 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"	-	定义该依赖关系的特定语义的代码见表后说明。
objfile	text	-	用户定义 C 函数库文件路径。

在任何情况下，一条 PG_SHDEPEND 记录就表明被引用的对象不能在未删除依赖对象的前提下被删除。但是其中也有几种依赖类型由 deptype 定义的情况：

- 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)

  在依赖对象的 ACL（访问控制列表，也就是权限列表）中提到被引用的对象（必须是一个角色）。不会为对象的所有者创建 SHARED_DEPENDENCY_ACL，因为所有者将具有 SHARED_DEPENDENCY_OWNER 记录。

- SHARED_DEPENDENCY_PIN (p)

  没有依赖对象。这类记录标识系统自身依赖于被依赖对象，因此这种对象绝对不能被删除。此类型的记录只能被 initdb 创建，依赖对象的字段都为 0。

14.2.56 PG_SHDESCRIPTION

PG_SHDESCRIPTION 系统表存储共享数据库对象的可选注释。可以使用 COMMENT 命令操作注释的内容，使用 psql 的\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。

不同于大多数系统表，PG_SHDESCRIPTION 在集群中所有数据库之间共享：每个数据库集群只有一个 PG_SHDESCRIPTION，而不是每个数据库一个。

表14-56 PG_SHDESCRIPTION 字段

名字	类型	引用	描述
objoid	oid	任意 OID 属性	此描述所属的对象的 OID
classoid	oid	14.2.17 PG_CLASS.oid	此对象所在的系统目录的 OID
description	text	-	作为对该对象描述的任意文本

14.2.57 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 14.3.117 PG_SECLABELS。

14.2.54 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14-57 PG_SHSECLABEL 字段

名字	类型	引用	描述
objoid	oid	任意 OID 属性	此安全标签所属对象的 OID
classoid	oid	14.2.17 PG_CLASS.oid	对象所属系统目录的 OID

名字	类型	引用	描述
provider	text	-	与此标签关联的标签提供者
label	text	-	应用于该对象的安全标签

14.2.58 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。需要有系统管理员权限才可以访问此系统表。

表14-58 PG_STATISTIC 字段

名称	类型	描述
starelid	oid	所描述的字段所属的表或者索引。
starelkind	"char"	所属对象的类型。
staattnum	smallint	所描述的字段在表中的编号，从 1 开始。
stainherit	boolean	是否统计有继承关系的对象。
stanullfrac	real	该字段中为 NULL 的记录的比率。
stawidth	integer	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。 • 大于 0 的值是独立数值的实际数目。 • 小于 0 的值是表中行数的分数的负数（比如，一个字段的数值平均出现概率为两次，则可以表示为 stadistinct=-0.5）。 • 0 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	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

名称	类型	描述
		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。 • 大于 0 的值是独立数值的实际数目。 • 小于 0 的值是表中行数的分数的负数（比如，一个字段的数值平均出现概率为两次，则可以表示为 stadistinct=-0.5）。 • 0 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。

14.2.59 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，包括多列统计数据和表达式统计数据（后续支持）。收集哪些扩展统计数据是由用户指定的。需要有系统管理员权限才可以访问此系统表。

表14-59 PG_STATISTIC_EXT 字段

名称	类型	描述
starelid	oid	所描述的字段所属的表或者索引。
starelkind	"char"	所属对象的类型。
stainherit	boolean	是否统计有继承关系的对象。
stanullfrac	real	该字段中为 NULL 的记录的比率。
stawidth	integer	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。 • 一个小于零的数值是表中行数的分数的负数（比如，一个字段的数值平均出现概率为两次，则可以表示为 stadistinct=-0.5）。 • 0 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。 • 一个小于零的数值是表中行数的分数的负数（比如，一个字段的数值平均出现概率为两次，则可以表示为 stadistinct=-0.5）。

名称	类型	描述
		• 0 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，则为 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。

14.2.60 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14-60 PG_SYNONYM 字段

名称	类型	描述
synname	name	同义词名称。
synnamespace	oid	该同义词所在的命名空间的 OID。
synowner	oid	同义词的所有者，通常是其创建者 OID。
synobjschema	name	关联对象指定的模式名。
synobjname	name	关联对象名。

14.2.61 PG_TABLESPACE

PG_TABLESPACE 系统表存储表空间信息。

表14-61 PG_TABLESPACE 字段

名称	类型	描述
spcname	name	表空间名。
spcowner	oid	表空间的所有者，通常是其创建者。
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。

14.2.62 PG_TRIGGER

PG_TRIGGER 系统表存储触发器信息。

名称	类型	描述
tgrelid	oid	触发器所在表的 OID。
tgname	name	触发器名。
tgfoid	oid	触发器 OID。
tgtype	smallint	触发器类型。
tgenabled	"char"	O 表示触发器在“origin”和“local”模式下触发。 D 表示触发器被禁用。 R 表示触发器在“replica”模式下触发。 A 表示触发器始终触发。
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。
tginitdeferred	boolean	约束触发器是否为 INITIALLY DEFERRED 类型。
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。

名称	类型	描述
tgqual	pg_node_tree	表示触发器的 WHEN 条件，如果没有则为 null。

14.2.63 PG_TS_CONFIG

PG_TS_CONFIG 系统表包含表示文本搜索配置的选项。一个配置指定一个特定的文本搜索解析器和一个用于解析器输出类型的字典列表。

解析器在 PG_TS_CONFIG 记录中显示，但是字典映射的标记是由 14.2.64 PG_TS_CONFIG_MAP 中的辅助记录定义的。

表14-62 PG_TS_CONFIG 字段

名字	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择才会显示）
cfgname	name	-	文本搜索配置名
cfgnamespace	oid	14.2.38 PG_NAMESPACE.oid	此配置所在的命名空间的 OID
cfgowner	oid	14.2.13 PG_AUTHID.oid	配置的所有者
cfgparser	oid	14.2.66 PG_TS_PARSER.oid	此配置的文本搜索解析器的 OID
cfoptions	text[]	-	分词相关配置选项

14.2.64 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种输出符号类型，显示有哪些文本搜索字典可供查询以及以哪种顺序搜索。

表14-63 PG_TS_CONFIG_MAP 字段

名字	类型	引用	描述
mapcfg	oid	14.2.63 PG_TS_CONFIG.oid	拥有此映射记录的 14.2.63 PG_TS_CONFIG 的 OID
maptokentype	integer	-	由配置的解析器发出的一个符号类型
mapseqno	integer	-	查询该项的顺序

名字	类型	引用	描述
mapdict	oid	14.2.65 PG_TS_DICT.oid	查询的文本搜索字典的 OID

14.2.65 PG_TS_DICT

PG_TS_DICT 系统表包含定义文本搜索字典的项。字典取决于文本搜索模板，该模板显示所有需要实现的功能。字典本身提供了用户可设置参数的模板。

即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14-64 PG_TS_DICT 字段

名字	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择才会显示）
dictname	name	-	文本搜索字典名
dictnamespace	oid	14.2.38 PG_NAMESPACE.oid	此字典所在的命名空间的 OID
dictowner	oid	14.2.13 PG_AUTHID.oid	字典的所有者
dicttemplate	oid	14.2.67 PG_TS_TEMPLATE.oid	此字典的文本搜索模板的 OID
dictinitoption	text	-	模板的初始化选项字符串

14.2.66 PG_TS_PARSER

PG_TS_PARSER 系统表包含定义文本解析器的项。解析器负责分割输入文本为词位，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14-65 PG_TS_PARSER 字段

名字	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择才会显示）
prsname	name	-	文本搜索解析器名称
prsnamespace	oid	14.2.38 PG_NAMESPACE.oid	解析器所在的命名空间的 OID

名字	类型	引用	描述
prsstart	regproc	14.2.46 PG_PROC.oid	解析器的启动函数的 OID
prstoken	regproc	14.2.46 PG_PROC.oid	解析器的下一个标记函数的 OID
prsend	regproc	14.2.46 PG_PROC.oid	解析器的关闭函数的 OID
prsheadline	regproc	14.2.46 PG_PROC.oid	解析器的标题函数的 OID
prslextype	regproc	14.2.46 PG_PROC.oid	解析器的 lextype 函数的 OID

14.2.67 PG_TS_TEMPLATE

PG_TS_TEMPLATE 系统表包含定义文本搜索模板的项。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14-66 PG_TS_TEMPLATE 字段

名字	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择才会显示）
tmplname	name	-	文本搜索模板名
tmplnamespace	oid	14.2.38 PG_NAMESPACE.oid	模板所属的命名空间的 OID
tmplinit	regproc	14.2.46 PG_PROC.oid	模板的初始化函数的 OID
tmpllexize	regproc	14.2.46 PG_PROC.oid	模板的 lexize 函数的 OID

14.2.68 PG_TYPE

PG_TYPE 系统表存储数据类型的相关信息。

表14-67 PG_TYPE 字段

名称	类型	描述
typname	name	数据类型名称。
typnamespace	oid	此类型所在的命名空间的 OID。
typowner	oid	此类型的所有者。
typlen	smallint	对于定长类型是该类型内部表现形式的字节数。对于变

名称	类型	描述
		长类型 typlen 为负值。 • -1 表示一种“变长”（有长度字属性的数据）。 • -2 表示一个以 NULL 结尾的 C 字符串。
typbyval	boolean	指定内部传递这个类型的数值时是传值还是传引用。如果该类型的 TYPLEN 不是 1，2，4，8， TYPBYVAL 最好为假。变长类型通常是传引用。即使 TYPLEN 允许传值，TYPBYVAL 也可以为假。
typtype	"char"	• b 表示基础类型。 • c 表示复合类型（比如，一个表的行类型）。 • e 表示枚举类型。 • p 表示伪类型。 参见 typrelid 和 typbasetype。
typcategory	"char"	数据类型的模糊分类，可用于解析器使用的数据转换依据。
typispreferred	boolean	如果为真，则数据符合 TYPCATEGORY 所指定的转换规则时进行转换。
typisdefined	boolean	如果定义了类型，则为真。如果是一种尚未定义的类型的占位符，则为假。如果为假，则除了该类型名称，命名空间和 OID 之外没有可依赖的对象。
typdelim	"char"	分析数组输入时，分隔两个此类型数值的字符。请注意，分隔符是与数组元素数据类型相关联，而不是与数组数据类型相关联。
typrelid	oid	如果是复合类型（请参见 typtype），则此字段指向 pg_class 中定义该表的行。对于独立的复合类型，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。非复合类型为 0。
typelem	oid	如果不为 0，则它标识 pg_type 中的另一行。当前类型可以像一个产生类型为 typelem 的数组来描述。“true”数组类型是变长的（typlen= -1），但是某些定长（typlen > 0）类型也有非零的 typelem（比如 name 和 point）。如果一个定长类型有 typelem ，则其内部形式必须是 typelem 数据类型的某个数目的个数值，不能有其他数据。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）。

名称	类型	描述
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，如果为 0，则不支持。
typmodout	regproc	输出类型修改符函数，如果为 0，则不支持。
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方式。适用于磁盘存储以及该值在数据库中的大多数形式。如果数值是连续存储的，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边界存储。对齐引用是该序列中第一个数据的开头。可能的值包含： • c = char 对齐，即不需要对齐。 • s = short 对齐（在大多数机器上是 2 字节） • i = int 对齐（在大多数机器上是 4 字节） • d = double 对齐（在大多数机器上是 8 字节，但不一定是全部） 须知 对于系统表里使用的类型，在 pg_type 里定义的尺寸和对齐方式要和编译器在表示表行的结构中布局方式保持一致。
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（如果有该关系，请参见 pg_class.reltoastrelid）。 • m：数值可以以内联压缩方式存储。 • x：数值可以以内联压缩方式或者在"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（首先移动 e 和 x 域）。
typenotnull	boolean	表示在某类型上的一个 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。

名称	类型	描述
typtypmod	integer	域使用 typtypmod 记录要应用于其基础类型上的 typmod（如果基础类型不使用 typmod，则为-1）。如果此类型不是域，则为-1。
typndims	integer	如果一个域是数组，则 typndims 是数组维数的数值（即 typbasetype 是一个数组类型；域的 typelem 将匹配基本类型的 typelem）。除了数组类型的域以外的类型为 0。
typcollation	oid	指定类型的排序规则。如果为 0，则表示不支持排序规则。
typdefaultbin	pg_node_tree	如果不为 NULL，则为该类型缺省表达式的 nodeToString() 表现形式。目前这个字段只用于域。
typdefault	text	如果某类型没有相关缺省值，则为 NULL。如果 typdefaultbin 不为 NULL，则 typdefault 必须包含一个 typdefaultbin 代表的缺省表达式的人类可读版本。如果 typdefaultbin 为 NULL 但 typdefault 不为 NULL，typdefault 则是该类型缺省值的外部表现形式，可以将其输入到类型的转换器生成一个常量。
typacl	aclitem[]	访问权限。

14.2.69 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 14.3.160 PG_USER_MAPPINGS 进行查询。

表14-68 PG_USER_MAPPING 字段

名字	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择才会显示）。
umuser	oid	14.2.13 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	14.2.30 PG_FOREIGN_SERVER.oid	包含此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。

14.2.70 PG_USER_STATUS

PG_USER_STATUS 系统表提供了访问数据库用户的状态。需要有系统管理员权限才可以访问此系统表

表14-69 PG_USER_STATUS 字段

名称	类型	描述
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。
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	角色在当前实例上已经使用的永久表存储空间大小。
tempspace	bigint	角色在当前实例上已经使用的临时表存储空间大小。

14.2.71 PG_WORKLOAD_ACTION

PG_WORKLOAD_ACTION 系统表存储 query_band 的信息。

表14-70 PG_WORKLOAD_ACTION 字段

名称	类型	描述
qband	name	query band 键值对
class	name	query band 关联行为类别
object	name	query band 关联行为
action	name	query band 关联行为表现

14.2.72 PGXC_CLASS

PGXC_CLASS 系统表存储每张表的复制或分布信息。

表14-71 PGXC_CLASS 字段

名称	类型	描述
pcrelid	oid	表的 OID
pclocatortype	"char"	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	节点组名称
redistributed	"char"	表已经完成重分布
redis_order	integer	重分布的顺序
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

14.2.73 PGXC_GROUP

PGXC_GROUP 系统表存储节点组信息。

表14-72 PGXC_GROUP 字段

名称	类型	描述
group_name	name	节点组名称
in_redistribution	"char"	是否需要重分布 • 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
group_acl	aclitem[]	访问权限
group_kind	"char"	node group 类型 • i 表示 installation node group • n 表示普通非逻辑集群 node group • v 表示逻辑集群 node group • e 表示弹性集群

14.2.74 PGXC_NODE

PGXC_NODE 系统表存储集群节点信息。

表14-73 PGXC_NODE 字段

名称	类型	描述
node_name	name	节点名称。
node_type	"char"	节点类型。 C：协调节点。 D：数据节点。
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。
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。
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。
node_id	integer	节点标识符。
sctp_port	integer	主节点使用 TCP 代理通信库或 SCTP 通信库的数据通道监听端口。
control_port	integer	主节点使用 TCP 代理通信库或 SCTP 通信库的控制

名称	类型	描述
		通道监听端口。
sctp_port1	integer	备节点使用 TCP 代理通信库或 SCTP 通信库的数据通道监听端口。
control_port1	integer	备节点使用 TCP 代理通信库或 SCTP 通信库的控制通道监听端口。
nodeis_central	boolean	当前节点为中心控制节点。

14.2.75 SNAPSHOT

SNAPSHOT 系统表记录每次创建性能视图快照的起止时间，设置 enable_wdr_snapshot 为 on 后，该表由后台快照线程创建并维护。需要有系统管理员权限才可以访问此系统表。

表14-74 dbms_om.snapshot 字段

名称	类型	描述
snapshot_id	name	快照 ID（此字段为主键和分布键）
start_ts	timestamp with time zone	快照开始时间
end_ts	timestamp with time zone	快照结束时间

须知

- 此系统表的 schema 是 dbms_om。
- 禁止从外部修改或删除此表，否则可能引起视图快照相关功能异常。

14.2.76 TABLES_SNAP_TIMESTAMP

TABLES_SNAP_TIMESTAMP 系统表记录每次对每个性能视图创建快照的起止时间，设置 enable_wdr_snapshot 为 on 后，该表由后台快照线程创建并维护。需要有系统管理员权限才可以访问此系统表。

表14-75 dbms_om.tables_snap_timestamp 字段

名称	类型	描述
snapshot_id	name	快照 ID（此字段为主键和分布键）
db_name	text	视图所属数据库名称

名称	类型	描述
tablename	text	视图名称
start_ts	timestamp with time zone	快照开始时间
end_ts	timestamp with time zone	快照结束时间

须知

- 此系统表的 schema 是 dbms_om。
- 禁止从外部修改或删除此表，否则可能引起视图快照相关功能异常。

14.2.77 性能视图快照系统表

设置 enable_wdr_snapshot 为 on 后，后台快照线程会创建并维护以“SNAP_+视图名称”方式命名的系统表，用以记录各性能视图的快照结果。需要有系统管理员权限才可以访问下列系统表。

- SNAP_14.3.182 PGXC_OS_RUN_INFO
- SNAP_14.3.209 PGXC_WAIT_EVENTS
- SNAP_14.3.178 PGXC_INSTR_UNIQUE_SQL
- SNAP_14.3.196 PGXC_STAT_BAD_BLOCK
- SNAP_14.3.197 PGXC_STAT_BGWRITER
- SNAP_14.3.199 PGXC_STAT_REPLICATION
- SNAP_14.3.187 PGXC_REPLICATION_SLOTS
- SNAP_14.3.193 PGXC_SETTINGS
- SNAP_14.3.177 PGXC_INSTANCE_TIME
- SNAP_14.3.52 GLOBAL_WORKLOAD_TRANSACTION
- SNAP_14.3.217 PGXC_WORKLOAD_SQL_COUNT
- SNAP_14.3.198 PGXC_STAT_DATABASE
- SNAP_14.3.47 GLOBAL_STAT_DATABASE
- SNAP_14.3.185 PGXC_REDO_STAT
- SNAP_14.3.44 GLOBAL_REDO_STAT
- SNAP_14.3.186 PGXC_REL_IOSTAT
- SNAP_14.3.45 GLOBAL_REL_IOSTAT
- SNAP_14.3.204 PGXC_TOTAL_MEMORY_DETAIL
- SNAP_14.3.181 PGXC_NODE_STAT_RESET_TIME
- SNAP_14.3.200 PGXC_SQL_COUNT
- SNAP_14.3.49 GLOBAL_TABLE_STAT
- SNAP_14.3.48 GLOBAL_TABLE_CHANGE_STAT
- SNAP_14.3.43 GLOBAL_COLUMN_TABLE_IO_STAT

- SNAP_14.3.46 GLOBAL_ROW_TABLE_IO_STAT

此类系统表除增加 snapshot_id 字段（bigint 类型）外，其余的字段定义与对应视图相同，且各表的分布键均为 snapshot_id。

例如，SNAP_PGXC_OS_RUN_INFO，用于存储 PGXC_OS_RUN_INFO 视图的快照，其字段新增了 snapshot_id，其余字段含义均与 PGXC_OS_RUN_INFO 视图相同。

须知

- 以上系统表的 schema 都是 dbms_om。
- 禁止从外部修改或删除以上系统表，否则可能引起视图快照相关功能异常。

14.3 系统视图

14.3.1 ALL_ALL_TABLES

ALL_ALL_TABLES 视图存储当前用户所能访问的表或视图。

表14-76 ALL_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

14.3.2 ALL_CONSTRAINTS

ALL_CONSTRAINTS 视图存储当前用户可访问的约束的信息。

表14-77 ALL_CONSTRAINTS 字段

名称	类型	描述
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

名称	类型	描述
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）

14.3.3 ALL_CONS_COLUMNS

ALL_CONS_COLUMNS 视图存储当前用户可访问的约束字段的信息。

表14-78 ALL_CONS_COLUMNS 字段

名称	类型	描述
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

14.3.4 ALL_COL_COMMENTS

ALL_COL_COMMENTS 视图存储当前用户可访问的表和视图中字段的注释信息。

表14-79 ALL_COL_COMMENTS 字段

名称	类型	描述
column_name	character varying(64)	列名
table_name	character varying(64)	表名或视图名
owner	character varying(64)	表或视图的所有者
comments	text	注释

14.3.5 ALL_DEPENDENCIES

ALL_DEPENDENCIES 视图存储了当前用户可访问的函数、高级包之间的依赖关系。

须知

因为相关信息的限制，目前 GaussDB(DWS)中，此表为空表，表内没有任何记录。

表14-80 ALL_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)	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 或 HARD）

14.3.6 ALL_IND_COLUMNS

ALL_IND_COLUMNS 视图存储了当前用户可访问的所有索引的字段信息。

表14-81 ALL_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

14.3.7 ALL_IND_EXPRESSIONS

ALL_IND_EXPRESSIONS 视图存储了当前用户可访问的表达式索引的信息。

表14-82 ALL_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

14.3.8 ALL_INDEXES

ALL_INDEXES 视图存储了当前用户可访问的索引信息。

表14-83 ALL_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	表示索引是否为唯一索引
generated	character varying(1)	表示索引的名称是否为系统生成
partitioned	character(3)	表示索引是否具有分区表的性质

14.3.9 ALL_OBJECTS

ALL_OBJECTS 视图记录了当前用户可访问的数据库对象。

表14-84 ALL_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

名称	类型	描述
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
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 14.2.39 PG_OBJECT 中的记录范围。

14.3.10 ALL_PROCEDURES

ALL_PROCEDURES 视图存储了当前用户可访问的所有存储过程或函数信息。

表14-85 ALL_PROCEDURES 字段

名称	类型	描述
owner	name	对象的所有者
object_name	name	对象名称

14.3.11 ALL_SEQUENCES

ALL_SEQUENCES 视图存储当前用户能够访问的所有序列。

表14-86 ALL_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	bigint	序列最小值
max_value	bigint	序列最大值
increment_by	bigint	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

名称	类型	描述
		• Y 表示是循环序列 • N 表示不是循环序列

14.3.12 ALL_SOURCE

ALL_SOURCE 视图存储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。

表14-87 ALL_SOURCE 字段

名称	类型	描述
owner	name	对象的所有者
name	name	对象名称
type	name	对象的类型
text	text	对象的定义

14.3.13 ALL_SYNONYMS

ALL_SYNONYMS 视图存储了当前用户可访问的所有同义词信息。

表14-88 ALL_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
table_schema_name	text	关联对象所属模式名
table_name	text	关联对象名

14.3.14 ALL_TAB_COLUMNS

ALL_TAB_COLUMNS 视图存储了当前用户可访问的表和视图的列的描述信息。

表14-89 ALL_TAB_COLUMNS 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)	表名或视图名。
column_name	character varying(64)	列名。
data_type	character varying(128)	列的数据类型。
column_id	integer	对象创建或增加列时列的序号。
data_length	integer	列的字节长度。
avg_col_len	numeric	列的平均长度（单位字节）。
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。
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。
char_length	numeric	列的长度（单位字符），只对 varchar，nvarchar2，bpchar，char 类型有效。
schema	character varying(64)	包含该表或视图的命名空间。
kind	text	当前记录所属的种类，如果此列属于表，则此字段显示为 table；如果此列属于视图，则此字段显示为 view。

14.3.15 ALL_TAB_COMMENTS

ALL_TAB_COMMENTS 视图存储当前用户可访问的所有表和视图的注释信息。

表14-90 ALL_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

名称	类型	描述
comments	text	注释

14.3.16 ALL_TABLES

ALL_TABLES 视图存储当前用户可访问的所有表。

表14-91 ALL_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)	表所在的表空间名称
status	character varying(8)	当前记录是否有效
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
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

14.3.17 ALL_USERS

ALL_USERS 视图存储记录数据库中所有用户，但不对用户信息进行详细的描述。

表14-92 ALL_USERS 字段

名称	类型	描述
username	name	用户名
user_id	oid	用户的 OID

14.3.18 ALL_VIEWS

ALL_VIEWS 视图存储了当前用户可访问的所有视图描述信息。

表14-93 ALL_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_length	integer	视图文本长度
text	text	视图文本

14.3.19 DBA_DATA_FILES

DBA_DATA_FILES 视图存储关于数据库文件的描述。需要有系统管理员权限才可以访问。

表14-94 DBA_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

14.3.20 DBA_USERS

DBA_USERS 视图存储关于数据库所有用户名信息。需要有系统管理员权限才可以访问。

表14-95 DBA_USERS 字段

名称	类型	描述
username	character varying(64)	用户名

14.3.21 DBA_COL_COMMENTS

DBA_COL_COMMENTS 视图存储关于数据库中表和视图中字段的注释信息。需要有系统管理员权限才可以访问。

名称	类型	描述
column_name	character varying(64)	列名
table_name	character varying(64)	表名或视图名
owner	character varying(64)	表或视图的所有者

名称	类型	描述
comments	text	注释

14.3.22 DBA_CONSTRAINTS

DBA_CONSTRAINTS 视图存储关于数据库表中约束的信息。需要有系统管理员权限才可以访问。

名称	类型	描述
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)	约束相关的索引名（只针对唯一约束和主键约束）

14.3.23 DBA_CONS_COLUMNS

DBA_CONS_COLUMNS 视图存储关于数据库表中约束字段的信息。需要有系统管理员权限才可以访问。

名称	类型	描述
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

14.3.24 DBA_IND_COLUMNS

DBA_IND_COLUMNS 视图存储关于数据库中所有索引的字段信息。需要有系统管理员权限才可以访问。

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

14.3.25 DBA_IND_EXPRESSIONS

DBA_IND_EXPRESSIONS 视图存储了数据库中的表达式索引的信息。需要有系统管理员权限才可以访问。

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

14.3.26 DBA_IND_PARTITIONS

DBA_IND_PARTITIONS 视图存储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 DBA_IND_PARTITIONS 里都会有一行记录。需要有系统管理员权限才可以访问。

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

名称	类型	描述
schema	character varying(64)	索引分区所属分区表索引的模式
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
index_partition_usable	boolean	索引分区是否可用
high_value	text	索引分区所对应的表分区的边界(范围分区为上边界，列表分区为边界值集合)。 前向兼容的保留字段，8.1.3 集群版本新增 pretty_high_value 用于记录此信息。
pretty_high_value	text	索引分区所对应的表分区的边界(范围分区为上边界，列表分区为边界值集合)。 查询结果为表分区对应边界表达式的即时反编译输出。该字段的输出比 high_value 的信息更详细，根据实际使用场景可输出 collaton、字段数据类型等信息。
def_tablespace_name	name	索引分区的表空间名称

14.3.27 DBA_INDEXES

DBA_INDEXES 视图存储关于数据库下的所有索引信息。需要有系统管理员权限才可以访问。

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	表示索引是否为唯一索引
generated	character varying(1)	表示索引名称是否为系统生成
partitioned	character(3)	表示索引是否具有分区表的性质

14.3.28 DBA_OBJECTS

DBA_OBJECTS 视图存储了数据库中所有数据库对象。需要有系统管理员权限才可以访问。

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
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 14.2.39 PG_OBJECT 中的记录范围。

14.3.29 DBA_PART_INDEXES

DBA_PART_INDEXES 视图存储数据库中所有分区表索引的信息。需要有系统管理员权限才可以访问。

名称	类型	描述
index_owner	character varying(64)	分区表索引的所有者名称
schema	character varying(64)	分区表索引的模式
index_name	character varying(64)	分区表索引的名称
table_name	character varying(64)	分区表索引所属的分区表名称
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）和列表分区（List Partitioning）。
partition_count	bigint	分区表索引的索引分区的个数
def_tablespace_name	name	分区表索引的表空间名称

名称	类型	描述
partitioning_key_count	integer	分区表的分区键个数

14.3.30 DBA_PART_TABLES

DBA_PART_TABLES 视图存储数据中所有分区表的信息。需要有系统管理员权限才可以访问。

名称	类型	描述
table_owner	character varying(64)	分区表的所有者名称
schema	character varying(64)	分区表的模式
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）和列表分区（List Partitioning）。
partition_count	bigint	分区表的分区个数
def_tablespace_name	name	分区表的表空间名称
partitioning_key_count	integer	分区表的分区键个数

14.3.31 DBA_PROCEDURES

DBA_PROCEDURES 视图存储关于数据库下的所有存储过程或函数信息。需要有系统管理员权限才可以访问。

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

14.3.32 DBA_SEQUENCES

DBA_SEQUENCES 视图存储关于数据库下的所有序列信息。需要有系统管理员权限才可以访问。

名称	类型	描述
sequence_owner	character varying(64)	序列的所有者
sequence_name	character varying(64)	序列的名称

14.3.33 DBA_SOURCE

DBA_SOURCE 视图存储关于数据库下的所有存储过程或函数信息，且提供存储过程或函数定义的字段。需要有系统管理员权限才可以访问。

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)	存储过程或函数的名称
text	text	存储过程或函数的定义

14.3.34 DBA_SYNONYMS

DBA_SYNONYMS 视图存储关于数据库下的所有同义词信息。需要有系统管理员权限才可以访问。

表14-96 DBA_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
table_schema_name	text	关联对象所属模式名
table_name	text	关联对象名

14.3.35 DBA_TAB_COLUMNS

DBA_TAB_COLUMNS 视图存储关于表和视图的字段的信息。数据库里每个表的每个字段都在 DBA_TAB_COLUMNS 里有一行。需要有系统管理员权限才可以访问。

名称	类型	描述
owner	character	表或视图的所有者

名称	类型	描述
	varying(64)	
table_name	character varying(64)	表名或视图名
column_name	character varying(64)	列名
data_type	character varying(128)	列的数据类型
column_id	integer	创建表或视图时列的序号
data_length	integer	列的字节长度
comments	text	注释
avg_col_len	numeric	列的平均长度（单位字节）
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
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
char_length	numeric	列的长度（以字符计），只对 varchar，nvarchar2， bpchar，char 类型有效
schema	character varying(64)	包含该表或视图的命名空间。
kind	text	当前记录所属的种类，如果此列属于表，则此字段显示为 table；如果此列属于视图，则此字段显示为 view。

14.3.36 DBA_TAB_COMMENTS

DBA_TAB_COMMENTS 视图存储关于数据库下的所有表和视图的注释信息。需要有系统管理员权限才可以访问。

名称	类型	描述

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

14.3.37 DBA_TAB_PARTITIONS

DBA_TAB_PARTITIONS 视图提供数据库中所有分区的信息。

名称	类型	描述
table_owner	character varying(64)	分区所在表的所有者
schema	character varying(64)	分区表模式
table_name	character varying(64)	表名
partition_name	character varying(64)	分区的名称
high_value	text	范围分区的上边界，或列表分区的边界值集合。 前向兼容的保留字段，8.1.3 集群版本新增 pretty_high_value 用于记录此信息。
pretty_high_value	text	范围分区的上边界，或列表分区的边界值集合。 查询结果为表分区对应边界表达式的即时反编译输出。该字段的输出比 high_value 的信息更详细，根据实际使用场景可输出 collaton、字段数据类型等信息。
tablespace_name	name	分区所在表空间的名称

14.3.38 DBA_TABLES

DBA_TABLES 视图存储关于数据库下的所有表信息。需要有系统管理员权限才可以访问。

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)	表所在的表空间的名称

名称	类型	描述
status	character varying(8)	当前记录是否有效
temporary	character(1)	是否为临时表 • Y 表示是临时表 • N 表示不是临时表
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

14.3.39 DBA_TABLESPACES

DBA_TABLESPACES 视图存储有关可用的表空间的信息。需要有系统管理员权限才可以访问。

表14-97 DBA_TABLESPACES 字段

名称	类型	描述
tablespace_name	character varying(64)	表空间的名称

14.3.40 DBA_TRIGGERS

DBA_TRIGGERS 视图存储关于数据库内的触发器信息。需要有系统管理员权限才可以访问。

名称	类型	描述
trigger_name	character varying(64)	触发器名称
table_name	character varying(64)	定义触发器的表的名称
table_owner	character varying(64)	定义触发器的表的所有者

14.3.41 DBA_VIEWS

DBA_VIEWS 视图存储关于数据库内的视图信息。需要有系统管理员权限才可以访问。

名称	类型	描述

名称	类型	描述
owner	character varying(64)	视图的所有者
view_name	character varying(64)	视图的名称

14.3.42 DUAL

DUAL 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。

表14-98 DUAL 字段

名称	类型	描述
dummy	text	表达式计算结果

14.3.43 GLOBAL_COLUMN_TABLE_IO_STAT

GLOBAL_COLUMN_TABLE_IO_STAT 视图提供当前数据库所有列存表的 IO 统计数据。其字段的名称、类型和顺序与 GS_COLUMN_TABLE_IO_STAT 视图相同，具体的字段请参考 14.3.55 GS_COLUMN_TABLE_IO_STAT。各统计字段为所有节点对应字段之和。

需要有系统管理员权限才可以访问此视图。

14.3.44 GLOBAL_REDO_STAT

GLOBAL_REDO_STAT 视图显示集群中所有节点上 XLOG 重做过程中的统计信息总和。除 avgiotim（表示所有节点平均的重做写入时间）外，其余字段名称和 14.3.230 PV_REDO_STAT 视图相同，但其余字段含义为各节点上 PV_REDO_STAT 视图同名字段的数值之和。

需要有系统管理员权限才可以访问此视图。

14.3.45 GLOBAL_REL_IOSTAT

GLOBAL_REL_IOSTAT 视图显示集群中所有节点上磁盘读写统计信息的总和。其各字段的名称与 14.3.58 GS_REL_IOSTAT 视图相同，但含义为各节点上 GS_REL_IOSTAT 视图同名字段的数值之和。需要有系统管理员权限才可以访问此视图。

14.3.46 GLOBAL_ROW_TABLE_IO_STAT

GLOBAL_ROW_TABLE_IO_STAT 视图提供当前数据库所有行存表的 IO 统计数据。其字段的名称、类型和顺序与 GS_ROW_TABLE_IO_STAT 视图相同，具体的字段请参考 14.3.61 GS_ROW_TABLE_IO_STAT。各统计字段为所有节点对应字段之和。

需要有系统管理员权限才可以访问此视图。

14.3.47 GLOBAL_STAT_DATABASE

GLOBAL_STAT_DATABASE 视图显示集群中所有节点上数据库的状态和统计信息之和。

- CN 上查询 GLOBAL_STAT_DATABASE 视图，返回的结果除 stats_reset 字段（当前 CN 上的状态重置时间）之外，其余字段表示在集群内相关节点上的数值之和。需注意，因 GLOBAL_STAT_DATABASE 视图中各字段的逻辑含义不同，求和的范围也有所不同。
- 在 DN 上查询视图 GLOBAL_STAT_DATABASE，所得结果与表 14-99 相同。

表14-99 GLOBAL_STAT_DATABASE 字段

名称	类型	描述	求和范围
datid	oid	数据库 OID。	-
datname	name	数据库名。	-
numbackends	integer	当前节点上连接到该数据库的后端数。 这是该视图中唯一一个反映目前状态值的列；所有列均返回自上次重置以来的累积值。	CN
xact_commit	bigint	当前节点上该数据库中已经提交的事务数。	CN
xact_rollback	bigint	当前节点上该数据库中已经回滚的事务数。	CN
blks_read	bigint	当前节点上该数据库中读取的磁盘块的数量。	DN
blks_hit	bigint	当前节点上高速缓存中发现的磁盘块的个数，即缓存中命中的块数（只包括 GaussDB(DWS)缓冲区高速缓存，不包括文件系统的缓存）。	DN
tup_returned	bigint	当前节点上该数据库查询返回的行数。	DN
tup_fetched	bigint	当前节点上该数据库查询抓取的行数。	DN
tup_inserted	bigint	当前节点上该数据库插入的行数。	DN
tup_updated	bigint	当前节点上该数据库更新的行数。	DN
tup_deleted	bigint	当前节点上该数据库删除的行数。	DN
conflicts	bigint	当前节点上由于数据库恢复冲突取消的查询数量（只在备用服务器上发生）。请参见 14.3.130 PG_STAT_DATABASE_CONFLICT	CN 和 DN

名称	类型	描述	求和范围
		S 获取更多信息。	
temp_files	bigint	当前节点上该数据库创建的临时文件个数。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），而且不考虑 log_temp_files 设置。	DN
temp_bytes	bigint	当前节点上该数据库写入临时文件的大小。计算所有临时文件，不论为什么创建临时文件，而且不考虑 log_temp_files 设置。	DN
deadlocks	bigint	当前节点上该数据库中发生的死锁数量。	CN 和 DN
blk_read_time	double precision	当前节点上该数据库后端读取数据文件块花费的时间，以毫秒计算。	DN
blk_write_time	double precision	当前节点上该数据库后端写入数据文件块花费的时间，以毫秒计算。	DN
stats_reset	timestamp with time zone	当前节点上该数据库统计重置的时间。	-

14.3.48 GLOBAL_TABLE_CHANGE_STAT

GLOBAL_TABLE_CHANGE_STAT 视图显示当前数据库中所有表格（不包括外表）变更情况。表示次数的各字段为实例启动以来的累计值。

需要有系统管理员权限才可以访问此视图。

表14-100 GLOBAL_TABLE_CHANGE_STAT 字段

名称	类型	描述
schemaname	name	表的命名空间
relname	name	表的名称
last_vacuum	timestamp with time zone	最后一次手动 Vacuum 的时间。
vacuum_count	bigint	手动 Vacuum 的次数。为各 CN 节点上次数之和。
last_autovacuum	timestamp with time zone	最后一次自动 Vacuum 的时间。
autovacuum_count	bigint	自动 Vacuum 的次数。为各 CN 节点上次数之

名称	类型	描述
		和。
last_analyze	timestamp with time zone	最后一次分析（包括手动和自动）的时间。
analyze_count	bigint	分析（包括手动和自动）的次数。由于 analyze 会同时在所有节点上进行，该字段为所有 CN 节点上的最大值。
last_autoanalyze	timestamp with time zone	最后一次自动分析的时间。
autoanalyze_count	bigint	自动分析的次数。为各 CN 节点上次数之和。
last_change	bigint	最后一次修改（INSERT，UPDATE 或 DELETE）的时间。

14.3.49 GLOBAL_TABLE_STAT

GLOBAL_TABLE_STAT 视图显示当前数据库中所有表格（不包括外表）的统计信息。除 live_tuples 和 dead_tuples 为当前实时值外，其余各统计字段为实例启动以来的累计值。

需要有系统管理员权限才可以访问此视图。

表14-101 GLOBAL_TABLE_STAT 字段

名称	类型	描述
schemaname	name	表的命名空间
relname	name	表的名称
distribute_mode	“char”	表的分布方式，与系统表 pgxc_class 中的 pclocatortype 字段含义相同。
seq_scan	bigint	顺序扫描的次数。只统计行存表。如果是分区表，显示各个分区扫描次数的和。
seq_tuple_read	bigint	顺序扫描的行数。只统计行存表。
index_scan	bigint	索引扫描的次数。只统计行存表。
index_tuple_read	bigint	索引扫描的行数。只统计行存表。
tuple_inserted	bigint	插入的行数。如果是复制表，显示各节点最大值；如果是分布表，显示各节点之和。
tuple_updated	bigint	更新的行数。如果是复制表，显示各节点最大值；如果是分布表，显示各节点之和。

名称	类型	描述
tuple_deleted	bigint	删除的行数。如果是复制表，显示各节点最大值；如果是分布表，显示各节点之和。
tuple_hot_updated	bigint	热更新的行数。如果是复制表，显示各节点最大值；如果是分布表，显示各节点之和。
live_tuples	bigint	活元组数量。显示各节点最大值；如果是分布表，显示各节点之和。 只适用行存表。
dead_tuples	bigint	死元组数量。显示各节点最大值；如果是分布表，显示各节点之和。 只适用行存表。

14.3.50 GLOBAL_WORKLOAD_SQL_COUNT

GLOBAL_WORKLOAD_SQL_COUNT 视图显示集群中所有 Workload 控制组内 SQL 语句执行次数的统计信息，包括 SELECT、UPDATE、INSERT、DELETE 语句的执行次数统计，以及 DDL、DML、DCL 类型语句的执行次数统计。

表14-102 GLOBAL_WORKLOAD_SQL_COUNT 字段

名称	类型	描述
workload	name	Workload 控制组名称
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

14.3.51 GLOBAL_WORKLOAD_SQL_ELAPSE_TIME

GLOBAL_WORKLOAD_SQL_ELAPSE_TIME 视图显示集群中所有 Workload 控制组内 SQL 语句执行的响应时间的统计信息，包括 SELECT、UPDATE、INSERT、DELETE 语句的最大、最小、平均、以及总响应时间，单位为微秒。

表14-103 GLOBAL_WORKLOAD_SQL_ELAPSE_TIME 字段

名称	类型	描述
workload	name	Workload 控制组名称
total_select_elapse	bigint	SELECT 总响应时间
max_select_elapse	bigint	SELECT 最大响应时间
min_select_elapse	bigint	SELECT 最小响应时间
avg_select_elapse	bigint	SELECT 平均响应时间
total_update_elapse	bigint	UPDATE 总响应时间
max_update_elapse	bigint	UPDATE 最大响应时间
min_update_elapse	bigint	UPDATE 最小响应时间
avg_update_elapse	bigint	UPDATE 平均响应时间
total_insert_elapse	bigint	INSERT 总响应时间
max_insert_elapse	bigint	INSERT 最大响应时间
min_insert_elapse	bigint	INSERT 最小响应时间
avg_insert_elapse	bigint	INSERT 平均响应时间
total_delete_elapse	bigint	DELETE 总响应时间
max_delete_elapse	bigint	DELETE 最大响应时间
min_delete_elapse	bigint	DELETE 最小响应时间
avg_delete_elapse	bigint	DELETE 平均响应时间

14.3.52 GLOBAL_WORKLOAD_TRANSACTION

GLOBAL_WORKLOAD_TRANSACTION 视图提供集群所有 CN 上 WORKLOAD 控制组相关的事务信息的总和。需要有系统管理员权限才可以访问。该视图仅在资源实时监控功能开启，即 enable_resource_track 为 on 时有效。

表14-104 GLOBAL_WORKLOAD_TRANSACTION 字段

名称	类型	描述
workload	name	WORKLOAD 控制组名称
commit_counter	bigint	各 CN 上提交次数总和
rollback_counter	bigint	各 CN 上回滚次数总和
resp_min	bigint	集群总体最小响应时间

名称	类型	描述
resp_max	bigint	集群总体最大响应时间
resp_avg	bigint	各 CN 上平均响应时间
resp_total	bigint	各 CN 上响应时间总和

14.3.53 GS_ALL_CONTROL_GROUP_INFO

GS_ALL_CONTROL_GROUP_INFO 视图显示数据库内所有的控制组信息。

表14-105 GS_ALL_CONTROL_GROUP_INFO 字段

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
wdlevel	bigint	Workload 控制组层级
cpucores	text	控制组使用的 CPU 核的信息

14.3.54 GS_CLUSTER_RESOURCE_INFO

GS_CLUSTER_RESOURCE_INFO 视图显示的是所有 DN 资源的汇总信息。

表14-106 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

名称	类型	描述
min_io_util	integer	DN 最小 IO 使用率
max_io_util	integer	DN 最大 IO 使用率
used_mem_rate	integer	物理节点最大内存使用率

14.3.55 GS_COLUMN_TABLE_IO_STAT

GS_COLUMN_TABLE_IO_STAT 视图显示当前数据库中所有列存表在当前节点上的 IO 情况。各统计字段为实例启动以来的累计值。

表14-107 GS_COLUMN_TABLE_IO_STAT 字段

名称	类型	描述
schemaname	name	表的命名空间
relname	name	表的名称
heap_read	bigint	堆逻辑读块数
heap_hit	bigint	堆命中块数
idx_read	bigint	索引逻辑读块数
idx_hit	bigint	索引命中块数
cu_read	bigint	Compression Unit 逻辑读个数
cu_hit	bigint	Compression Unit 命中个数
cidx_read	bigint	Compression Unit Index 逻辑读个数
cidx_hit	bigint	Compression Unit Index 命中个数

14.3.56 GS_INSTR_UNIQUE_SQL

Unique SQL 定义

数据库将接收到的每个 SQL 的文本字符串，都进行解析并生成内部解析树，遍历解析树并忽略其中的常数值，以一定的算法计算出来一个整数值作为 Unique SQL ID，用来唯一标识这一类 SQL，Unique SQL ID 相同的一类 SQL 就叫做 Unique SQL。

示例

假如，用户先后输入 SQL:

```
select * from t1 where id = 1;
select * from t1 where id = 2;
```

那么，这两条 SQL 的统计信息会汇聚到同一个 Unique SQL 上：

```
select * from t1 where id = ?;
```

GS_INSTR_UNIQUE_SQL 视图

GS_INSTR_UNIQUE_SQL 视图显示当前节点收集的 Unique SQL 的执行信息，主要包括以下内容：

- Unique SQL ID 以及归一化后的 SQL 文本字符串，归一化后的 SQL 文本如示例中所示。通常对于 DML 语句，在计算 Unique SQL ID 的过程中会忽略常量值。但对于 DDL、DCL 以及设置参数等语句，常量值不可以忽略。
- 执行次数（成功执行的次数），响应时间（数据库内部的 SQL 执行时间，包括最大、最小和总时间）。
- Cache/IO 信息，包含 block 的物理读、逻辑读次数，仅统计执行成功的 SQL 在各 DN 节点上的相关信息。该统计值与查询执行当时所处理的数据量、所使用的内存、是否多次执行、内存管理策略、是否有其他并发查询等因素相关，反映整个查询执行过程中的 buffer 块物理读和逻辑读次数，不同时间执行可能统计值不同。
- 行活动，包含 SELECT 语句的结果集返回行数、更新行、插入行、删除行、顺序扫描行、随机扫描行等信息。除结果集返回行数与该 SELECT 语句的结果集行数一致、且仅在 CN 上记录外，其他行活动信息均在 DN 上记录，且统计数值反应的是整个查询执行过程中的行活动，包括对相关系统表、元数据表、数据表等做必要的扫描和修改，与对应数据量以及相关参数设置相关，即统计数值将会大于等于对实际数据表的扫描和修改。
- 时间分布，包含：DB_TIME/CPU_TIME/EXECUTION_TIME/PARSE_TIME/PLAN_TIME/REWRITE_TIME/PL_EXECUTION_TIME/PL_COMPILATION_TIME/NET_SEND_TIME/DATA_IO_TIME，相关定义见表 14-108。该信息在 CN 和 DN 节点均有统计，视图查询时将汇总展示。
- 软硬解析次数，包含软解析（缓存计划）、硬解析（生成计划）的次数，即如果本次执行的是之前缓存的计划，软解析次数+1，如果本次执行的计划是重新生成的，则硬解析次数+1。该次数在 CN 和 DN 节点上都会统计，视图查询时将汇总展示。

Unique SQL 收集功能存在以下约束：

- 只有执行成功的 SQL 才会显示其详细的统计信息，否则可能只记录 query、node、user 等信息。
- 如果开启 Unique SQL 收集功能，CN 节点将对所有接收到的查询进行统计收集，包括工具和用户的查询等。
- 若一条 SQL 语句内部包含执行多条 SQL 语句、类似存储过程执行等场景，仅会对最外层 SQL 生成一条 Unique SQL，所有子 SQL 的统计信息都会汇总到该 Unique SQL 记录上。
- Unique SQL 的响应时间统计中不完全包含 NET_SEND_TIME 阶段的时间，所以 EXECUTION_TIME 和 elapse_time 等时间之间不存在大小比较关系。

- 对于类似 begin;...;commit;等形式的事务块，当前不支持统计子句的解析时间（parse_time）。

普通用户访问 GS_INSTR_UNIQUE_SQL 视图，只能看到该用户相关的 Unique SQL 信息，管理员用户可以看到当前节点所有的 Unique SQL 信息。CN 和 DN 上均可查询 GS_INSTR_UNIQUE_SQL 视图，DN 上显示的是本节点内的 Unique SQL 统计信息，CN 上显示的是本节点 Unique SQL 完整统计信息，即该 CN 节点会收集其他 CN 和 DN 上对应该 CN 的 Unique SQL 的执行信息，进行汇总展示。通过查询 GS_INSTR_UNIQUE_SQL 视图，能够定位由于消耗不同资源导致的 Top SQL，为集群性能调优和维护提供依据。

GUC 参数 instr_unique_sql_timeout 设置了 Unique SQL 的超时时间，单位是小时。后台线程每隔 1 小时检查一次所有的 Unique SQL，将 last_time 在 instr_unique_sql_timeout 小时之前的 Unique SQL 删除。

表14-108 GS_INSTR_UNIQUE_SQL 字段

名称	类型	描述
node_name	name	接收 SQL 的 CN 节点名称
node_id	integer	节点 ID，等同于 pgxc_node 表中 node_id
user_name	name	用户名称
user_id	oid	用户 ID
unique_sql_id	bigint	归一化的 UNIQUE SQL ID
query	text	归一化的 SQL 文本
n_calls	bigint	成功执行次数
min_elapse_time	bigint	SQL 在数据库内的最小运行时间（单位：微秒）
max_elapse_time	bigint	SQL 在数据库内的最大运行时间（单位：微秒）
total_elapse_time	bigint	SQL 在数据库内的总运行时间（单位：微秒）
n_returned_rows	bigint	行活动-SELECT 语句返回的结果集行数
n_tuples_fetched	bigint	行活动-随机扫描行（列存表/外表不统计）
n_tuples_returned	bigint	行活动-顺序扫描行（列存表/外表不统计）
n_tuples_inserted	bigint	行活动-插入行数
n_tuples_updated	bigint	行活动-更新行数
n_tuples_deleted	bigint	行活动-删除行数

名称	类型	描述
n_blocks_fetched	bigint	buffer 的块访问次数，即物理读/IO
n_blocks_hit	bigint	buffer 的块命中次数，即逻辑读/Cache
n_soft_parse	bigint	软解析次数（缓存计划）
n_hard_parse	bigint	硬解析次数（生成计划）
db_time	bigint	有效的 DB 执行时间，包含等待时间、网络发送时间等，若查询执行涉及到多线程，DB_TIME 是多个线程的 DB_TIME 之和（单位：微秒）
cpu_time	bigint	CPU 的执行时间，不包含 sleep 时间（单位：微秒）
execution_time	bigint	查询执行器内的 SQL 执行时间，DDL 语句、以及某些不经过执行器执行的语句（例如 Copy 语句）不计数（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 过程化语言函数上的执行时间（单位：微秒）
pl_compilation_time	bigint	plpgsql 过程化语言函数上的编译时间（单位：微秒）
net_send_time	bigint	网络时间，包含 CN 向客户端发送数据、DN 向 CN 发送数据等时间（单位：微秒）
data_io_time	bigint	IO 时间，文件 IO 耗时（单位：微秒）
first_time	timestamp with time zone	该 SQL 第一次执行的时间
last_time	timestamp with time zone	该 SQL 上一次执行的时间

14.3.57 GS_NODE_STAT_RESET_TIME

GS_NODE_STAT_RESET_TIME 视图提供当前节点的统计信息重置时间，返回带时区的时间戳，详细含义参考《SQL 语法参考》中“函数和操作符>系统管理函数>其他函数”章节的 get_node_stat_reset_time()函数。

14.3.58 GS_REL_IOSTAT

GS_REL_IOSTAT 视图提供当前节点上磁盘读写的统计信息。当前版本中，每次读/写磁盘只读/写一页，所以读写次数与页数相等。

表14-109 GS_REL_IOSTAT 字段

名称	类型	描述
phyrds	bigint	读磁盘次数
phywrts	bigint	写磁盘次数
phyblkrd	bigint	读磁盘页数
phyblkwrt	bigint	写磁盘页数

14.3.59 GS_RESPOOL_RUNTIME_INFO

GS_RESPOOL_RUNTIME_INFO 视图显示当前 CN 所有资源池作业运行信息。

表14-110 GS_RESPOOL_RUNTIME_INFO 字段

名称	类型	描述
nodegroup	name	资源池所属逻辑集群名称，默认集群显示 "installation"
rpname	name	资源池名称
ref_count	int	资源池引用作业数，作业经过资源池不管是否管控都会计数
fast_run	int	资源池快车道运行作业数
fast_wait	int	资源池快车道排队作业数
slow_run	int	资源池慢车道运行作业数
slow_wait	int	资源池慢车道排队作业数

14.3.60 GS_RESPOOL_RESOURCE_INFO

GS_RESPOOL_RESOURCE_INFO 视图显示 CN 上所有资源池作业运行信息以及当前实例(CN/DN)所有资源池资源使用信息。

说明

DN 上仅显示当前 DN 所属逻辑集群的资源池监控信息。

表14-111 GS_RESPOOL_RESOURCE_INFO 字段

名称	类型	描述
nodegroup	name	资源池所属逻辑集群名称，默认集群显示"installation"
rpname	name	资源池名称
cgroup	name	资源池关联控制组名称
ref_count	int	资源池引用作业数，作业经过资源池不管是否管控都会计数，只在 CN 上有效
fast_run	int	资源池快车道运行作业数，只在 CN 上有效
fast_wait	int	资源池快车道排队作业数，只在 CN 上有效
fast_limit	int	资源池快车道作业并发限制，只在 CN 上有效
slow_run	int	资源池慢车道运行作业数，只在 CN 上有效
slow_wait	int	资源池慢车道排队作业数，只在 CN 上有效
slow_limit	int	资源池慢车道作业并发限制，只在 CN 上有效
used_cpu	double	资源池 5s 监控周期内使用 CPU 个数平均值，保留小数点后 2 位 • DN：显示当前 DN 上资源池使用的 CPU 个数 • CN：显示所有 DN 上资源池使用 CPU 的累积和
cpu_limit	int	资源池可用 CPU 的上限，CPU 配额管控情况下为 GaussDB 可用 CPU，CPU 限额管控情况下为关联控制组 CPU 可用 CPU • DN：显示当前 DN 上资源池可用 CPU 上限 • CN：显示所有 DN 上资源池可用 CPU 上限的累积和
used_mem	int	资源池当前使用的内存大小，单位：'MB'’ • DN：显示当前 DN 上资源池使用的内存大小 • CN：显示所有 DN 上资源池使用内存的累积和
estimate_mem	int	当前 CN 上，资源池运行作业的估算内存之和，只在 CN 上有效
mem_limit	int	资源池可用内存上限，单位：'MB' • DN：显示当前 DN 上资源池可用内存上限 • CN：显示所有 DN 上资源池可用内存上限的累积和

名称	类型	描述
read_kbytes	bigint	资源池 5s 监控周期内逻辑读字节数，单位：'KB' • DN：显示当前 DN 上资源池逻辑读字节数 • CN：显示所有 DN 上资源池逻辑读字节的累积和
write_kbytes	bigint	资源池 5s 监控周期内逻辑写字节数，单位：'KB' • DN：显示当前 DN 上资源池逻辑写字节数 • CN：显示所有 DN 上资源池逻辑写字节的累积和
read_counts	bigint	资源池 5s 监控周期内逻辑读次数 • DN：显示当前 DN 上资源池逻辑读次数 • CN：显示所有 DN 上资源池逻辑读次数的累积和
write_counts	bigint	资源池 5s 监控周期内逻辑写次数 • DN：显示当前 DN 上资源池逻辑写次数 • CN：显示所有 DN 上资源池逻辑写次数的累积和
read_speed	double	资源池 5s 监控周期内逻辑读速率的平均值 • DN：显示当前 DN 上资源池逻辑读速率 • CN：显示所有 DN 上资源池逻辑读速率的累积和
write_speed	double	资源池 5s 监控周期内逻辑写速率平均值 • DN：显示当前 DN 上资源池逻辑写速率 • CN：显示所有 DN 上资源池逻辑写速率的累积和

14.3.61 GS_ROW_TABLE_IO_STAT

GS_ROW_TABLE_IO_STAT 视图显示当前数据库中所有行存表在当前节点上的 IO 情况。各统计字段为实例启动以来的累计值。

表14-112 GS_ROW_TABLE_IO_STAT 字段

名称	类型	描述
schemaname	name	表的命名空间
relname	name	表的名称
heap_read	bigint	堆逻辑读块数

名称	类型	描述
heap_hit	bigint	堆命中块数
idx_read	bigint	索引逻辑读块数
idx_hit	bigint	索引命中块数
toast_read	bigint	TOAST 表逻辑读块数
toast_hit	bigint	TOAST 表命中块数
tidx_read	bigint	TOAST 表 Index 逻辑读个数
tidx_hit	bigint	TOAST 表 Index 命中个数

14.3.62 GS_SESSION_CPU_STATISTICS

GS_SESSION_CPU_STATISTICS 视图显示和当前用户执行复杂作业正在运行时的负载管理 CPU 使用的信息。

表14-113 GS_SESSION_CPU_STATISTICS 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
query	text	正在执行的语句
node_group	text	语句所属用户对应的逻辑集群

14.3.63 GS_SESSION_MEMORY_STATISTICS

GS_SESSION_MEMORY_STATISTICS 视图显示和当前用户执行复杂作业正在运行时的负载管理内存使用的信息。

表14-114 GS_SESSION_MEMORY_STATISTICS 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_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逻辑集群

14.3.64 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，包括执行次数和响应时间（除 MERGE INTO 语句外，统计其他四类语句的最大、最小、平均和总响应时间，单位为微秒），以及 DDL、DML、DCL 类型语句的执行次数。

GS_SQL_COUNT 视图中对 DDL、DML、DCL 类型语句分类与 SQL 语法中略有不同，具体如下：

- CREATE/ALTER/DROP USER，CREATE/ALTER/DROP ROLE 等用户相关语句属于 DCL 类型。
- BEGIN/COMMIT/SET CONSTRAINTS/ROLLBACK/SAVEPOINT/START 等事务相关语句属于 DCL 类型。
- ALTER SYSTEM KILL SESSION 等价于 SELECT pg_terminate_backend()语句，属于 DML 类型。

其余语句的分类与 SQL 语法中定义类似。

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，计数会清零，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；GS_SQL_COUNT 视图涉及的统计信息只在 CN 上统计，且不统计从其他 CN 发送过来的 SQL。在 DN 上查询该视图返回结果为空。

表14-115 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 数量
update_count	bigint	UPDATE 数量
insert_count	bigint	INSERT 数量
delete_count	bigint	DELETE 数量
mergeinto_count	bigint	MERGE INTO 数量
ddl_count	bigint	DDL 数量
dml_count	bigint	DML 数量
dcl_count	bigint	DCL 数量
total_select_elapse	bigint	SELECT 总响应时间
avg_select_elapse	bigint	SELECT 平均响应时间
max_select_elapse	bigint	SELECT 最大响应时间
min_select_elapse	bigint	SELECT 最小响应时间
total_update_elapse	bigint	UPDATE 总响应时间
avg_update_elapse	bigint	UPDATE 平均响应时间
max_update_elapse	bigint	UPDATE 最大响应时间
min_update_elapse	bigint	UPDATE 最小响应时间
total_delete_elapse	bigint	DELETE 总响应时间
avg_delete_elapse	bigint	DELETE 平均响应时间
max_delete_elapse	bigint	DELETE 最大响应时间
min_delete_elapse	bigint	DELETE 最小响应时间
total_insert_elapse	bigint	INSERT 总响应时间
avg_insert_elapse	bigint	INSERT 平均响应时间
max_insert_elapse	bigint	INSERT 最大响应时间

名称	类型	描述
min_insert_elapse	bigint	INSERT 最小响应时间

14.3.65 GS_STAT_DB_CU

GS_STAT_DB_CU 视图查询集群各个节点，每个数据库的 CU 命中情况。可以通过 gs_stat_reset()进行清零。

表14-116 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

14.3.66 GS_STAT_SESSION_CU

GS_STAT_SESSION_CU 视图查询当前集群各个节点，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14-117 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

14.3.67 GS_TABLE_CHANGE_STAT

GS_TABLE_CHANGE_STAT 视图显示当前数据库中所有表格（不包括外表）在当前节点上的变更情况。表示次数的各字段为实例启动以来的累计值。

表14-118 GS_TABLE_CHANGE_STAT 字段

名称	类型	描述
schemaname	name	表的命名空间
relname	name	表的名称
last_vacuum	timestamp with time zone	最后一次手动 vacuum 的时间。
vacuum_count	bigint	手动 Vacuum 的次数。
last_autovacuum	timestamp with time zone	最后一次自动 vacuum 的时间。
autovacuum_count	bigint	自动 vacuum 的次数。
last_analyze	timestamp with time zone	最后一次分析（包括手动和自动）的时间。
analyze_count	bigint	分析（包括手动和自动）的次数。
last_autoanalyze	timestamp with time zone	最后一次自动分析的时间。
autoanalyze_count	bigint	自动分析的次数。
last_change	bigint	最后一次修改（INSERT，UPDATE 或 DELETE）的时间。

14.3.68 GS_TABLE_STAT

GS_TABLE_STAT 视图显示当前数据库中所有表格（不包括外表）在当前节点上的统计信息。除 live_tuples 和 dead_tuples 为当前实时值外，其余各统计字段为实例启动以来的累计值。

表14-119 GS_TABLE_STAT 字段

名称	类型	描述
schemaname	name	表的命名空间
relname	name	表的名称
seq_scan	bigint	顺序扫描的次数。只统计行存表。如果是分区表，显示各个分区扫描次数的和。
seq_tuple_read	bigint	顺序扫描的行数。只统计行存表。
index_scan	bigint	索引扫描的次数。只统计行存表。
index_tuple_read	bigint	索引扫描的行数。只统计行存表。
tuple_inserted	bigint	插入的行数。

名称	类型	描述
tuple_updated	bigint	更新的行数。
tuple_deleted	bigint	删除的行数。
tuple_hot_updated	bigint	热更新的行数。
live_tuples	bigint	活元组数量。CN 上查询该视图，analyze 后显示该表格总的活元组数量，未 analyze 的情况下显示 0。只适用行存表。
dead_tuples	bigint	死元组数量。CN 上查询该视图，analyze 后显示该表格总的死元组数量，未 analyze 的情况下显示 0。只适用行存表。

14.3.69 GS_TOTAL_NODEGROUP_MEMORY_DETAIL

GS_TOTAL_NODEGROUP_MEMORY_DETAIL 视图统计当前数据库逻辑集群使用内存的信息，单位为 MB。

表14-120 GS_TOTAL_NODEGROUP_MEMORY_DETAIL 字段

名称	类型	描述
ngname	text	逻辑集群名称
memorytype	text	内存类型，包括以下几种： • ng_total_memory：该逻辑集群的总内存大小 • ng_used_memory：该逻辑集群的实际使用内存大小 • ng_estimate_memory：该逻辑集群的估算使用内存大小 • ng_foreignrp_memsize：该逻辑集群的外部资源池的总内存大小 • ng_foreignrp_usedsize：该逻辑集群的外部资源池实际使用内存大小 • ng_foreignrp_peaksize：该逻辑集群的外部资源池使用内存的峰值 • ng_foreignrp_mempct：该逻辑集群的外部资源池占该逻辑集群总内存大小的百分比 • ng_foreignrp_estmsize：该逻辑集群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

14.3.70 GS_USER_TRANSACTION

GS_USER_TRANSACTION 视图提供查询单 CN 上用户相关的事务信息。数据库记录每个用户事务提交和回滚的次数及事务提交和回滚的响应时间，单位是微秒。

表14-121 GS_USER_TRANSACTION 字段

名称	类型	描述
username	name	用户名称
commit_counter	bigint	提交次数
rollback_counter	bigint	回滚次数
resp_min	bigint	最小响应时间
resp_max	bigint	最大响应时间
resp_avg	bigint	平均响应时间
resp_total	bigint	响应时间总和

14.3.71 GS_VIEW_DEPENDENCY

GS_VIEW_DEPENDENCY 视图提供查询当前用户可见的所有视图的直接依赖关系。

表14-122 GS_VIEW_DEPENDENCY 字段

名称	类型	描述
objschema	name	视图空间名称
objname	name	视图名称
refobjschema	name	依赖对象的空间名称
refobjname	name	依赖对象的名称
relobjkind	char	依赖对象的类型 • r 表示依赖对象为表 • v 表示依赖对象为视图

14.3.72 GS_VIEW_DEPENDENCY_PATH

GS_VIEW_DEPENDENCY_PATH 视图提供查询当前用户可见的所有视图的直接依赖关系。如果该视图依赖的基础表存在且各级视图依赖关系正常，通过该视图可以查询自基础表开始的各级视图的依赖关系。

表14-123 GS_VIEW_DEPENDENCY_PATH 字段

名称	类型	描述
objschema	name	视图空间名称
objname	name	视图名称
refobjschema	name	依赖对象的空间名称
refobjname	name	依赖对象的名称
path	text	依赖路径

14.3.73 GS_VIEW_INVALID

GS_VIEW_INVALID 视图提供查询当前用户可见的所有不可用的视图。如果该视图依赖的基础表或函数或同义词存在异常，该视图 validtype 列显示为“invalid”。

表14-124 GS_VIEW_INVALID 字段

名称	类型	描述
oid	oid	视图 OID
schemaname	name	视图空间名称
viewname	name	视图名称
viewowner	name	视图的所有者
definition	text	视图定义
validtype	text	视图有效性标识

14.3.74 GS_WAIT_EVENTS

GS_WAIT_EVENTS 视图显示当前节点上各类等待状态和事件的统计信息。

仅在 GUC 参数 enable_track_wait_event 为 on 的情况下，视图中各统计字段的数值才会被累加。若在运行过程中将 enable_track_wait_event 设置为 off，则不再累加统计数值，但已有数值不受影响。enable_track_wait_event 为 off，查询该视图返回 0 行。

表14-125 GS_WAIT_EVENTS 字段

名称	类型	描述
nodename	name	节点名称。
type	text	事件的类型，包括 STATUS，LOCK_EVENT，LWLOCK_EVENT 和

名称	类型	描述
		IO_EVENT 四种类型。
event	text	事件名称，可参考 14.3.151 PG_THREAD_WAIT_STATUS 视图。
wait	bigint	事件发生次数。该字段及以下字段均为进程运行中的累计值。
failed_wait	bigint	等待失败次数。当前版本中只有 LOCK 和 LWLOCK 等锁超时或失败才会使用该字段。
total_wait_time	bigint	该事件总持续时间。
avg_wait_time	bigint	该事件平均持续时间。
max_wait_time	bigint	该事件最大等待时间。
min_wait_time	bigint	该事件最小等待时间。

当前版本中，对于 type='LOCK_EVENT'，'LWLOCK_EVENT'和'IO_EVENT'的事件，GS_WAIT_EVENTS 视图显示范围与 14.3.151 PG_THREAD_WAIT_STATUS 视图对应事件相同。

对于 type='STATUS'的事件 GS_WAIT_EVENTS 包含的等待状态列如下，其详细含义参见 14.3.151 PG_THREAD_WAIT_STATUS 视图。

- acquire lwlock
- acquire lock
- wait io
- wait pooler get conn
- wait pooler abort conn
- wait pooler clean conn
- wait transaction sync
- wait wal sync
- wait data sync
- wait producer ready
- create index
- analyze
- vacuum
- vacuum full
- gtm connect
- gtm begin trans
- gtm commit trans
- gtm rollback trans
- gtm create sequence

- gtm alter sequence
- gtm get sequence val
- gtm set sequence val
- gtm drop sequence
- gtm rename sequence

14.3.75 GS_WLM_OPERARAOR_INFO

本视图显示当前 CN 上已经完成执行的 query 语句中的算子执行信息，此系统视图信息来源于系统表 dbms_om.14.2.4 GS_WLM_OPERATOR_INFO。

14.3.76 GS_WLM_OPERATOR_HISTORY

GS_WLM_OPERATOR_HISTORY 视图显示的是当前用户在当前 CN 上执行作业结束后的算子的相关记录。

此视图用于 Database Manager 从内核中查询数据，内核中的数据会定时被清理。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 14.2.4 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后会被删除。记录的数据同表 14-4。

14.3.77 GS_WLM_OPERATOR_STATISTICS

GS_WLM_OPERATOR_STATISTICS 视图显示当前用户正在执行的作业的算子相关信息。

表14-126 GS_WLM_OPERATOR_STATISTICS 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	该算子直到结束时总的执行时间(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_memo	integer	当前算子在所有 DN 上的最小内存峰值(MB)。

名称	类型	描述
ry		
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，所有下盘 DN 的最小下盘数据量(MB)，默认为 0。
max_spill_size	integer	若发生下盘，所有下盘 DN 的最大下盘数据量(MB)，默认为 0。
average_spill_size	integer	若发生下盘，所有下盘 DN 的平均下盘数据量(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

14.3.78 GS_WLM_SESSION_INFO

本视图显示当前 CN 上已经完成执行的 query 语句的执行信息，此系统视图信息来源于系统表 dbms_om.14.2.5 GS_WLM_SESSION_INFO。

14.3.79 GS_WLM_SESSION_HISTORY

GS_WLM_SESSION_HISTORY 视图显示当前用户在当前 CN 上执行作业结束后的负载管理记录。此视图用于 Database Manager 从 GaussDB(DWS)中查询数据，仅当 GUC 参

数 enable_resource_track 为 on 时，视图会查询 GS_WLM_SESSION_INFO 表中 3 分钟内的数据进行返回。

表14-127 GS_WLM_SESSION_HISTORY 的字段

名称	类型	描述
datid	oid	连接后端的数据库 OID。
dbname	text	连接后端的数据库名称。
schemaname	text	模式名。
nodename	text	语句执行的 CN 名称。
username	text	连接到后端的用户名。
application_name	text	连接到后端的应用名。
client_addr	inet	连接到后端的客户端的 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	客户端用于与后端通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
query_band	text	用于标示作业类型，可通过 GUC 参数 query_band 进行设置，默认为空字符串。
block_time	bigint	语句执行前的阻塞时间，包含语句解析和优化时间，单位 ms。
start_time	timestamp with time zone	语句执行的开始时间。
finish_time	timestamp with time zone	语句执行的结束时间。
duration	bigint	语句实际执行的时间，单位 ms。
estimate_total_time	bigint	语句预估执行时间，单位 ms。
status	text	语句执行结束状态：正常为 finished，异常为 aborted。该处记录的语句状态应为数据库服务端执行状态，当服务器端执行成功，结果集返回时报错，该语句应为 finished。
abort_info	text	语句执行结束状态为 aborted 时显示异常信息。
resource_pool	text	用户使用的资源池。

名称	类型	描述
control_group	text	语句所使用的 Cgroup。
estimate_memory	integer	语句预估使用内存，单位 MB。
min_peak_memory	integer	语句在所有 DN 上的最小内存峰值，单位 MB。
max_peak_memory	integer	语句在所有 DN 上的最大内存峰值，单位 MB。
average_peak_memory	integer	语句执行过程中的内存使用平均值，单位 MB。
memory_skew_percent	integer	语句各 DN 间的内存使用倾斜率。
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。
min_spill_size	integer	若发生下盘，所有下盘 DN 的最小下盘数据量(MB)，默认为 0。
max_spill_size	integer	若发生下盘，所有下盘 DN 的最大下盘数据量(MB)，默认为 0。
average_spill_size	integer	若发生下盘，所有下盘 DN 的平均下盘数据量(MB)，默认为 0。
spill_skew_percent	integer	若发生下盘，DN 间下盘倾斜率。
min_dn_time	bigint	语句在所有 DN 上的最小执行时间，单位 ms。
max_dn_time	bigint	语句在所有 DN 上的最大执行时间，单位 ms。
average_dn_time	bigint	语句在所有 DN 上的平均执行时间，单位 ms。
dntime_skew_percent	integer	语句在各 DN 间的执行时间倾斜率。
min_cpu_time	bigint	语句在所有 DN 上的最小 CPU 时间，单位 ms。
max_cpu_time	bigint	语句在所有 DN 上的最大 CPU 时间，单位 ms。
total_cpu_time	bigint	语句在所有 DN 上的 CPU 总时间，单位 ms。
cpu_skew_percent	integer	语句在 DN 间的 CPU 时间倾斜率。
min_peak_iops	integer	语句在所有 DN 上的每秒最小 IO 峰值（列存单位是次/s，行存单位是万次/s）。

名称	类型	描述
max_peak_iops	integer	语句在所有 DN 上的每秒最大 IO 峰值（列存单位是次/s，行存单位是万次/s）。
average_peak_iops	integer	语句在所有 DN 上的每秒平均 IO 峰值（列存单位是次/s，行存单位是万次/s）。
iops_skew_percent	integer	语句在 DN 间的 IO 倾斜率。
warning	text	主要显示如下几类告警信息以及 SQL 自诊断调优相关告警： 1. Spill file size large than 256MB 2. Broadcast size large than 100MB 3. Early spill 4. Spill times is greater than 3 5. Spill on memory adaptive 6. Hash table conflict
queryid	bigint	语句执行使用的内部 query id。
query	text	执行的语句。
query_plan	text	语句的执行计划。
node_group	text	语句所属用户对应的逻辑集群。
pid	bigint	语句的后端线程的 pid。
lane	text	语句执行时所在的快慢车道。
unique_sql_id	bigint	归一化的 Unique SQL ID。

14.3.80 GS_WLM_SESSION_STATISTICS

GS_WLM_SESSION_STATISTICS 视图显示当前用户在当前 CN 上正在执行的作业的负载管理记录。

表14-128 GS_WLM_SESSION_STATISTICS 的字段

名称	类型	描述
datid	oid	连接后端的数据 OID。
dbname	name	连接后端的数据库名称。
schemaname	text	模式名。
nodename	text	语句执行的 CN 节点名称。
username	name	连接到后端的用户名。

名称	类型	描述
application_name	text	连接到后端的应用名。
client_addr	inet	连接到后端的客户端的 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	客户端用于与后端通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
query_band	text	用于标示作业类型，可通过 GUC 参数 query_band 进行设置，默认为空字符串。
pid	bigint	后端线程 ID。
block_time	bigint	语句执行前的阻塞时间，单位 ms。
start_time	timestamp with time zone	语句执行的开始时间。
duration	bigint	语句已经执行的时间，单位 ms。
estimate_total_time	bigint	语句执行预估总时间，单位 ms。
estimate_left_time	bigint	语句执行预估剩余时间，单位 ms。
enqueue	text	工作负载管理资源状态。
resource_pool	name	用户使用的资源池。
control_group	text	语句所使用的 Cgroup。
estimate_memory	integer	语句预估使用内存，单位 MB。
min_peak_memory	integer	语句在所有 DN 上的最小内存峰值，单位 MB。
max_peak_memory	integer	语句在所有 DN 上的最大内存峰值，单位 MB。
average_peak_memory	integer	语句执行过程中的内存使用平均值，单位 MB。
memory_skew_percent	integer	语句在各 DN 间的内存使用倾斜率。
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。

名称	类型	描述
		[a:b]: 数量为 b 个 DN 中有 a 个 DN 下盘。
min_spill_size	integer	若发生下盘，所有下盘 DN 的最小下盘数据量(MB)，默认为 0。
max_spill_size	integer	若发生下盘，所有下盘 DN 的最大下盘数据量(MB)，默认为 0。
average_spill_size	integer	若发生下盘，所有下盘 DN 的平均下盘数据量(MB)，默认为 0。
spill_skew_percent	integer	若发生下盘，DN 间下盘倾斜率。
min_dn_time	bigint	语句在所有 DN 上的最小执行时间，单位 ms。
max_dn_time	bigint	语句在所有 DN 上的最大执行时间，单位 ms。
average_dn_time	bigint	语句在所有 DN 上的平均执行时间，单位 ms。
dntime_skew_percent	integer	语句在各 DN 间的执行时间倾斜率。
min_cpu_time	bigint	语句在所有 DN 上的最小 CPU 时间，单位 ms。
max_cpu_time	bigint	语句在所有 DN 上的最大 CPU 时间，单位 ms。
total_cpu_time	bigint	语句在所有 DN 上的 CPU 总时间，单位 ms。
cpu_skew_percent	integer	语句在各 DN 间的 CPU 时间倾斜率。
min_peak_iops	integer	语句在所有 DN 上的每秒最小 IO 峰值（列存单位是次/s，行存单位是万次/s）。
max_peak_iops	integer	语句在所有 DN 上的每秒最大 IO 峰值（列存单位是次/s，行存单位是万次/s）。
average_peak_iops	integer	语句在所有 DN 上的每秒平均 IO 峰值（列存单位是次/s，行存单位是万次/s）。
iops_skew_percent	integer	语句在 DN 间的 IO 倾斜率。
warning	text	主要显示如下几类告警信息以及 SQL 自诊断调优相关告警： 1. Spill file size large than 256MB 2. Broadcast size large than 100MB 3. Early spill 4. Spill times is greater than 3 5. Spill on memory adaptive 6. Hash table conflict
queryid	bigint	语句执行使用的内部 query id。
query	text	正在执行的语句。

名称	类型	描述
query_plan	text	语句的执行计划。
node_group	text	语句所属用户对应的逻辑集群。

14.3.81 GS_WLM_SQL_ALLOW

GS_WLM_SQL_ALLOW 视图显示已经设置的资源管理 SQL 白名单，包括两部分内容：系统默认的 SQL 白名单和通过 GUC 参数 wlm_sql_allow_list 设置的 SQL 白名单。

14.3.82 GS_WORKLOAD_SQL_COUNT

GS_WORKLOAD_SQL_COUNT 视图显示当前节点上 Workload 控制组内的 SQL 语句执行次数的统计信息，包括 SELECT、UPDATE、INSERT、DELETE 语句的执行次数统计，以及 DDL、DML、DCL 类型语句的执行次数统计。

表14-129 GS_WORKLOAD_SQL_COUNT 字段

名称	类型	描述
workload	name	Workload 控制组名称
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

14.3.83 GS_WORKLOAD_SQL_ELAPSE_TIME

GS_WORKLOAD_SQL_ELAPSE_TIME 视图显示当前节点上 Workload 控制组内 SQL 语句执行的响应时间的统计信息，包括 SELECT、UPDATE、INSERT、DELETE 语句的最大、最小、平均、以及总响应时间，单位为微秒。

表14-130 GS_WORKLOAD_SQL_ELAPSE_TIME 字段

名称	类型	描述
workload	name	Workload 控制组名称
total_select_elapse	bigint	SELECT 总响应时间

名称	类型	描述
max_select_elapse	bigint	SELECT 最大响应时间
min_select_elapse	bigint	SELECT 最小响应时间
avg_select_elapse	bigint	SELECT 平均响应时间
total_update_elapse	bigint	UPDATE 总响应时间
max_update_elapse	bigint	UPDATE 最大响应时间
min_update_elapse	bigint	UPDATE 最小响应时间
avg_update_elapse	bigint	UPDATE 平均响应时间
total_insert_elapse	bigint	INSERT 总响应时间
max_insert_elapse	bigint	INSERT 最大响应时间
min_insert_elapse	bigint	INSERT 最小响应时间
avg_insert_elapse	bigint	INSERT 平均响应时间
total_delete_elapse	bigint	DELETE 总响应时间
max_delete_elapse	bigint	DELETE 最大响应时间
min_delete_elapse	bigint	DELETE 最小响应时间
avg_delete_elapse	bigint	DELETE 平均响应时间

14.3.84 GS_WORKLOAD_TRANSACTION

GS_WORKLOAD_TRANSACTION 视图提供查询单 CN 上 Workload 控制组相关的事务信息。数据库记录每个 Workload 控制组事务提交和回滚的次数及事务提交和回滚的响应时间，单位是微秒。

表14-131 GS_WORKLOAD_TRANSACTION 字段

名称	类型	描述
workload	name	Workload 控制组名称
commit_counter	bigint	提交次数
rollback_counter	bigint	回滚次数
resp_min	bigint	最小响应时间
resp_max	bigint	最大响应时间
resp_avg	bigint	平均响应时间

名称	类型	描述
resp_total	bigint	响应时间总和

14.3.85 MPP_TABLES

MPP_TABLES 视图显示 PGXC_CLASS 中的表信息。

表14-132 MPP_TABLES 字段

名称	类型	描述
schemaname	name	包含表的模式名。
tablename	name	表名。
tableowner	name	表的所有者。
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。

14.3.86 PG_AVAILABLE_EXTENSION_VERSIONS

PG_AVAILABLE_EXTENSION_VERSIONS 视图显示数据库中某些特性的扩展版本信息。

表14-133 PG_AVAILABLE_EXTENSION_VERSIONS 字段

名称	类型	描述
name	name	扩展名
version	text	版本名
installed	boolean	如果此扩展的版本当前已经安装，则为真
superuser	boolean	如果只允许系统管理员安装此扩展，则为真
relocatable	boolean	如果扩展可以重新加载到另一个模式，则为真
schema	name	扩展必须安装到的模式名，如果部分或全部可重新定位，则为 NULL
requires	name[]	必备扩展的名称，如果没有则为 NULL
comment	text	扩展的控制文件的注释字符串

14.3.87 PG_AVAILABLE_EXTENSIONS

PG_AVAILABLE_EXTENSIONS 视图显示数据库中某些特性的扩展信息。

表14-134 PG_AVAILABLE_EXTENSIONS 字段

名称	类型	描述
name	name	扩展名
default_version	text	缺省版本名，如果没有指定则为 NULL
installed_version	text	当前安装的扩展版本，如果未安装则为 NULL
comment	text	扩展控制文件的注释字符串

14.3.88 PG_BULKLOAD_STATISTICS

在集群任一正常节点上，通过查询 PG_BULKLOAD_STATISTICS 视图可以获取当前登录节点正在进行的导入导出业务执行情况，其中每一个导入/导出业务对应一条记录。需要有系统管理员权限才可以访问此视图

表14-135 PG_BULKLOAD_STATISTICS 字段

名称	类型	描述
node_name	text	节点名称。
db_name	text	数据库名称。
query_id	bigint	查询 ID，对应 debug_query_id。
tid	bigint	当前线程的线程号。
lwtid	integer	当前线程的轻量级线程号。
session_id	bigint	GDS 的会话 ID。
direction	text	业务类型，取值包括：gds to file、gds from file、gds to pipe、gds from pipe、copy from、copy to。
query	text	查询语句。
address	text	当前导入导出外表的 location。
query_start	timestamp with time zone	导入/导出开始时间。
total_bytes	bigint	待处理数据的总大小。 仅 GDS 普通文件导入时，且该行记录来自 CN 节点才会显示，否则为空。

phase	text	当前业务导入导出执行阶段，包括：INITIALIZING、TRANSFER_DATA、RELEASE_RESOURCE。
done_lines	bigint	已传输行数。
done_bytes	bigint	已传输字节数。

14.3.89 PG_COMM_CLIENT_INFO

PG_COMM_CLIENT_INFO 视图存储单个节点客户端连接信息（DN 上查询该视图显示 CN 连接 DN 的信息）。

表14-136 PG_COMM_CLIENT_INFO 字段

名称	类型	描述
node_name	text	当前节点的名称。
app	text	客户端应用名。
tid	bigint	当前线程的线程号。
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，显示-1 时表示当前连接是物理连接。

14.3.90 PG_COMM_DELAY

PG_COMM_DELAY 视图展示单个 DN 的通信库时延状态。

表14-137 PG_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

14.3.91 PG_COMM_STATUS

PG_COMM_STATUS 视图展示单个 DN 的通信库状态。

表14-138 PG_COMM_STATUS 字段

名称	类型	描述
node_name	text	节点名称。
rxpck/s	integer	节点通信库接收速率，单位 Byte/s。
txpck/s	integer	节点通信库发送速率，单位 Byte/s。
rxkB/s	bigint	节点通信库接收速率，单位 KByte/s。
txkB/s	bigint	节点通信库发送速率，单位 KByte/s。
buffer	bigint	cmailbox 的 buffer 大小。
memKB(libcomm)	bigint	libcomm 进程通信内存大小，单位 KByte。
memKB(libpq)	bigint	libpq 进程通信内存大小，单位 KByte。
%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 线程中最高的实时使用率。
stream	integer	当前使用的逻辑连接总数。

14.3.92 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14-139 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：表示逻辑连接接收报文正常。 • HOLD：表示逻辑连接接收报文等待中。 • 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

14.3.93 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14-140 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。
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。

14.3.94 PG_COMM_QUERY_SPEED

PG_COMM_QUERY_SPEED 视图展示单个节点上所有 query 对应的流量信息。

表14-141 PG_COMM_QUERY_SPEED 字段

名称	类型	描述
node_name	text	节点名称。
query_id	bigint	通信流对应的 debug_query_id 编号。
rxkB/s	bigint	查询对应的通信流接收速率，单位 Byte/s。
txkB/s	bigint	查询对应的通信流发送速率，单位 Byte/s。
rxkB	bigint	查询对应的通信流接收数据总量，单位 Byte。
txkB	bigint	查询对应的通信流发送数据总量，单位 Byte。
rxpck/s	bigint	查询对应的通信包接收速率，单位 packages/s。
txpck/s	bigint	查询对应的通信包发送速率，单位 packages/s。
rxpck	bigint	查询对应的通信包接收总量，单位 packages。
txpck	bigint	查询对应的通信包发送总量，单位 packages。

14.3.95 PG_CONTROL_GROUP_CONFIG

PG_CONTROL_GROUP_CONFIG 视图存储系统的控制组配置信息。

表14-142 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

14.3.96 PG_CURSORS

PG_CURSORS 视图列出了当前可用的游标。

表14-143 PG_CURSORS 字段

名称	类型	描述
name	text	游标名。
statement	text	声明改游标时的查询语句。
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

名称	类型	描述
		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。

14.3.97 PG_EXT_STATS

PG_EXT_STATS 视图提供对存储在 14.2.59 PG_STATISTIC_EXT 表里面的扩展统计信息的访问。扩展统计信息目前包括多列统计信息。

表14-144 PG_EXT_STATS 字段

名称	类型	引用	描述
schemaname	name	14.2.38 PG_NAMESPACE.nspname	包含表的模式名。
tablename	name	14.2.17 PG_CLASS.relname	表名。
attname	int2vector	14.2.59 PG_STATISTIC_EXT.stakey	统计信息扩展的多列信息。
inherited	boolean	-	如果为真，则包含继承的子列，否则只是指定表的字段。
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 0，表示字段组合中独立数值的估计数目。 • 如果小于 0，表示独立数值的数目被行数除的负数。 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； 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。比如，-1 表示一个字段组合中独立数值的个数和行数相同。

名称	类型	引用	描述
			• 如果等于 0，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 数据的唯一值的数目。 • 如果大于 0，表示独立数值的实际数目。 • 如果小于 0，表示独立数值的数目被行数除的负数。（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 0，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[]	-	一个最常用数值组合的频率的列表，即每个出现的次数除以行数。如果 most_common_vals 是 NULL，则为 NULL。
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[]	-	一个最常用数值组合的频率的列表，即每个出现的次数除以行数。如果 most_common_vals_null 是 NULL，则为 NULL。

14.3.98 PG_GET_INVALID_BACKENDS

PG_GET_INVALID_BACKENDS 视图提供显示 CN 上连接到当前 DN 备机的后端线程信息。

表14-145 PG_GET_INVALID_BACKENDS 字段

名称	类型	描述
pid	bigint	线程 ID
node_name	text	后端线程中连接的节点信息

名称	类型	描述
dbname	name	当前连接的数据库
backend_start	timestamp with time zone	后端线程启动的时间
query	text	后端线程正在执行的查询语句

14.3.99 PG_GET_SENDERS_CATCHUP_TIME

PG_GET_SENDERS_CATCHUP_TIME 视图显示单个 DN 上当前活跃的主备发送线程的追赶信息。

表14-146 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。
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。
catchup_type	text	catchup 方式为全量还是增量。
catchup_bcm_filename	text	catchup 当前执行的 bcm 文件。
catchup_bcm_finished	integer	catchup 已经操作完成的 bcm 文件数量。
catchup_bcm_total	integer	catchup 总共需要操作的 bcm 文件数量。
catchup_percent	text	catchup 已经操作完成的百分比。
catchup_remaining_time	text	catchup 预估剩余时间。

14.3.100 PG_GROUP

PG_GROUP 视图查看数据库认证角色及角色之间的成员关系。

表14-147 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

14.3.101 PG_INDEXES

PG_INDEXES 视图提供对数据库中每个索引的有用信息的访问。

表14-148 PG_INDEXES 字段

名称	类型	引用	描述
schemaname	name	14.2.38 PG_NAMESPACE.nspname	包含表和索引的模式名
tablename	name	14.2.17 PG_CLASS.relname	此索引所服务的表名
indexname	name	14.2.17 PG_CLASS.relname	索引名
tablespace	name	14.2.61 PG_TABLESPACE.spcname	包含索引的表空间名称
indexdef	text	-	索引定义（一个重建的 CREATE INDEX 命令）

14.3.102 PG_JOB

PG_JOB 视图用于代替之前版本的 PG_JOB 系统表，提供对之前版本的前向兼容。原 PG_JOB 系统表已经变更为 PG_JOBS 系统表，关于 PG_JOBS 系统表的描述详见 14.2.34 PG_JOBS。

表14-149 PG_JOB 字段

名字	类型	描述

名字	类型	描述
job_id	bigint	作业 ID。
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 postgres 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	标识作业执行的数据库名。
node_name	name	标识当前作业是在哪个 CN 上创建和执行。
job_status	text	当前任务的执行状态，取值范围：('r', 's', 'f', 'd')，默认为's'，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。 说明 • 当用户将定时任务关闭（即 job_queue_processes 为 0 时），由于监控 job 执行的线程不会启动，所以 job_status 不会根据 job 的实时状态进行设置，用户不需要关注 job_status。 • 只有当开启定时任务功能（job_queue_processes 为非 0 时），系统才会根据当前 job 的实时状态刷新 job_status 的值。
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	下次定时执行任务的时间，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。
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	上次运行的结束时间，时间精确到毫秒。

名字	类型	描述
	zone	
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	作业运行时所在的命名空间的名称。
what	text	作业内容。

14.3.103 PG_JOB_PROC

PG_JOB_PROC 视图用于代替之前版本的 PG_JOB_PROC 系统表，提供对之前版本的前向兼容。原 PG_JOB_PROC 系统表已经和原 PG_JOB 系统表一同并入当前版本的 PG_JOBS 系统表，关于 PG_JOBS 系统表的描述详见 14.2.34 PG_JOBS。

表14-150 PG_JOB_PROC 字段

名字	类型	描述
job_id	bigint	作业 ID
what	text	作业内容

14.3.104 PG_JOB_SINGLE

PG_JOB_SINGLE 视图用于显示当前节点的作业信息。

表14-151 PG_JOB_SINGLE 字段

名字	类型	描述
job_id	bigint	作业 ID。
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 postgres 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	标识作业执行的数据库名。
node_name	name	标识当前作业是在哪个 CN 上创建和执行。

名字	类型	描述
job_status	text	当前任务的执行状态，取值范围：('r', 's', 'f', 'd')，默认为's'，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。 说明 • 当用户将定时任务关闭（即 job_queue_processes 为 0 时），由于监控 job 执行的线程不会启动，所以 job_status 不会根据 job 的实时状态进行设置，用户不需要关注 job_status。 • 只有当开启定时任务功能（job_queue_processes 为非 0 时），系统才会根据当前 job 的实时状态刷新 job_status 的值。
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	下次定时执行任务的时间，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。
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	作业运行时所在的命名空间的名称。
what	text	作业内容。

14.3.105 PG_LIFECYCLE_DATA_DISTRIBUTE

PG_LIFECYCLE_DATA_DISTRIBUTE 视图查询 OBS 多温表中冷热数据分布情况。

表14-152 PG_LIFECYCLE_DATA_DISTRIBUTE 字段

名称	类型	描述
schemaname	name	模式名
tablename	name	当前表名
nodename	name	节点名
hotpartition	text	该 DN 节点上的热分区
coldpartition	text	该 DN 节点上的冷分区
switchablepartition	text	该 DN 节点上的可切分区
hotdatasize	text	该 DN 节点上的热分区数据大小
colddatasize	text	该 DN 节点上的冷分区数据大小
switchabledatasize	text	该 DN 节点上的可切分区数据大小

14.3.106 PG_LOCKS

PG_LOCKS 视图存储各打开事务所持有的锁信息。

表14-153 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	14.2.21 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。 • 如果是一个事务 ID，则为 NULL。
relation	oid	14.2.17 PG_CLASS.oid	被锁定对象关系的 OID，如果锁定的对象不是关系，也不是关系的一

名称	类型	引用	描述
			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。
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	14.2.17 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，如果对象不是普通数据库对象，则为 NULL。
objsubid	smallint	-	对于表的某个字段对应为字段编号；对于其他对象类型，该字段为 0；如果该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	-	持有此锁或者等待此锁的服务器线程的逻辑 ID。如果锁被一个预备事务持有，则为 NULL。
mode	text	-	此线程持有的或者是期望持有的锁模式。更多有关锁模式的内容请参见《数据仓库服务 SQL 语法参考》中的“LOCK”。
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；如果通过主锁表获得，则为 FALSE。

14.3.107 PG_NODE_ENV

PG_NODE_ENV 视图提供获取当前节点的环境变量信息。

表14-154 PG_NODE_ENV 字段

名称	类型	描述
node_name	text	当前节点名称
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

14.3.108 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14-155 PG_OS_THREADS 字段

名称	类型	描述
node_name	text	当前节点名称
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

14.3.109 PG_POOLER_STATUS

PG_POOLER_STATUS 视图查询 pooler 中的缓存连接状态。该视图只能在 CN 上执行查询，显示本地 CN 的 pooler 模块的连接缓存信息。

表14-156 PG_POOLER_STATUS 字段

名称	类型	描述

名称	类型	描述
database	text	数据库名称
user_name	text	用户名
tid	bigint	连接 CN 的线程 ID
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	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

14.3.110 PG_PREPARED_STATEMENTS

PG_PREPARED_STATEMENTS 视图显示当前会话所有可用的预备语句。

表14-157 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。

14.3.111 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14-158 PG_PREPARED_XACTS 字段

名称	类型	引用	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	14.2.13 PG_AUTHID.rolname	执行该事务的用户名
database	name	14.2.21 PG_DATABASE.datname	执行该事务所在的数据库名

14.3.112 PG_QUERYBAND_ACTION

PG_QUERYBAND_ACTION 视图显示 query_band 关联行为和次序。

表14-159 PG_QUERYBAND_ACTION 字段

名称	类型	描述
qband	text	query_band 键值对
respool_id	oid	query_band 关联资源池 OID
respool	text	query_band 关联资源池名
priority	text	query_band 关联队列内优先级
qborder	integer	query_band 搜索次序

14.3.113 PG_REPLICATION_SLOTS

PG_REPLICATION_SLOTS 视图查看复制节点的信息。

表14-160 PG_REPLICATION_SLOTS 字段

名称	类型	描述
slot_name	text	复制节点的名称

名称	类型	描述
plugin	name	逻辑复制槽对应的输出插件名
slot_type	text	复制节点的类型
datoid	oid	复制节点的数据库 OID
database	name	复制节点的数据库名称
active	boolean	复制节点是否为激活状态
xmin	xid	复制节点事务标识
catalog_xmin	text	逻辑复制槽对应的最早解码事务标识
restart_lsn	text	复制节点的 Xlog 文件信息
dummy_standby	boolean	复制节点是否为假备

14.3.114 PG_ROLES

PG_ROLES 视图提供访问数据库角色的相关信息。

表14-161 PG_ROLES 字段

名称	类型	引用	描述
rolname	name	-	角色名。
rolsuper	boolean	-	该角色是否是拥有最高权限的初始系统管理员。
rolinherit	boolean	-	该角色是否继承角色的权限。
rolcreaterole	boolean	-	该角色是否可以创建其他的角色。
rolcreatedb	boolean	-	该角色是否可以创建数据库。
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。
rolcanlogin	boolean	-	该角色是否可以登录数据库。
rolreplication	boolean	-	该角色是否可以复制。
rolauditadmin	boolean	-	该角色是否为审计管理员。
rolsystemadmin	boolean	-	该角色是否为系统管理员。

名称	类型	引用	描述
rolconnlimit	integer	-	对于可以登录的角色，rolconnlimit 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	-	不是口令，总是********。
rolvalidbegin	timestamp with time zone	-	帐户的有效开始时间；如果没有设置有效开始时间，则为 NULL。
rolvaliduntil	timestamp with time zone	-	帐户的有效结束时间；如果没有设置有效结束时间，则为 NULL。
rolrespool	name	-	用户所能够使用的 resource pool。
rolparentid	oid	14.2.13 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。
roltempspace	text	-	用户临时表存储空间限额。
rolspillspace	text	-	用户算子落盘空间限额。
rolconfig	text[]	-	运行时配置变量的会话缺省。
oid	oid	14.2.13 PG_AUTHID.oid	角色的 ID。
roluseft	boolean	14.2.13 PG_AUTHID.roluseft	角色是否可以操作外表。
nodegroup	name	-	角色所关联的逻辑集群名字，如果没有关联逻辑集群，该值为空。

14.3.115 PG_RULES

PG_RULES 视图提供对查询重写规则的有用信息访问的接口。

表14-162 PG_RULES 字段

名称	类型	描述
schemaname	name	包含表的模式名
tablename	name	规则作用的表名
rulename	name	规则的名称

名称	类型	描述
definition	text	规则定义（一个重新构造的创建命令）

14.3.116 PG_RUNNING_XACTS

PG_RUNNING_XACTS 视图主要功能是显示当前节点运行事务的信息。

表14-163 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 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
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

14.3.117 PG_SECLABELS

PG_SECLABELS 视图提供关于安全标签的信息。

表14-164 PG_SECLABELS 字段

名字	类型	引用	描述
objoid	oid	任意 OID 属性	安全标签所属的对象的 OID。
classoid	oid	14.2.17 PG_CLASS.oid	此对象的系统表的 OID。
objsubid	integer	-	对于某个在表字段上的安全标签，为字段编号（引用表本身的 objoid 和 classoid）。对于所有其他对象类型，该字段为 0。

名字	类型	引用	描述
objtype	text	-	标签出现的对象的类型。
objnamespace	oid	14.2.38 PG_NAMESPACE.oid	对象的命名空间的 OID，如果适用；否则为 NULL。
objname	text	-	标签适用的对象名。
provider	text	14.2.54 PG_SECLABEL.provider	与标签相关的标签提供者。
label	text	14.2.54 PG_SECLABEL.label	应用于此对象的安全标签。

14.3.118 PG_SESSION_WLMSTAT

PG_SESSION_WLMSTAT 视图显示和当前用户执行作业正在运行时的负载管理相关信息。

表14-165 PG_SESSION_WLMSTAT 字段

名称	类型	描述
datid	oid	连接后端的数据库 OID。
datname	name	连接后端的数据库名称。
threadid	bigint	后端线程 ID。
processid	integer	后端线程的 pid。
usesysid	oid	登录后端的用户 OID。
appname	text	连接到后端的应用名。
usename	name	登录到该后端的用户名。
priority	bigint	语句所在 Cgroups 的优先级。
attribute	text	语句的属性： • Ordinary：语句发送到数据库后被解析前的默认属性。 • Simple：简单语句。 • Complicated：复杂语句。 • Internal：数据库内部语句。
block_time	bigint	语句当前为止的 pending 的时间，单位 s。
elapsed_time	bigint	语句当前为止的实际执行时间，单位 s。
total_cpu_time	bigint	语句在上一时间周期内的 DN 上 CPU 使用的总时

名称	类型	描述
		间，单位 s。
cpu_skew_percent	integer	语句在上一时间周期内的 DN 上 CPU 使用的倾斜率。
statement_mem	integer	语句执行所需要的估算内存，预留字段。
active_points	integer	语句占用的资源池并发点数。
dop_value	integer	语句的从资源池中获取的 dop 值。
control_group	text	语句当前所使用的 Cgroups。
status	text	语句当前的状态，包括： • pending：执行前状态。 • running：执行进行状态。 • finished：执行正常结束。（当 enqueue 字段为 StoredProc 或 Transaction 时，仅代表语句中的部分作业已经执行完毕，该状态会持续到该语句完全执行完毕。） • aborted：执行异常终止。 • active：非以上四种状态外的正常状态。 • unknown：未知状态。
enqueue	text	语句当前的排队情况，包括： • Global：全局排队。 • Respool：资源池排队。 • CentralQueue：在中心协调节点(CCN)中排队。 • Transaction：语句处于一个事务块中。 • StoredProc：语句处于一个存储过程中。 • None：未在排队。 • Forced None：事务块语句或存储过程语句由于超出设定的等待时间而强制执行。
resource_pool	name	语句当前所在的资源池。
query	text	该后端的最新查询。如果 state 状态是 active，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。
isplana	bool	逻辑集群模式下，语句当前是否占用其他逻辑集群的资源执行。该值默认为 f，表示不占用其他逻辑集群的资源执行。
node_group	text	语句所属用户对应的逻辑集群。
lane	text	表示语句查询的快慢车道。

名称	类型	描述
		• fast：快车道。 • slow：慢车道。 • none：未管控。

14.3.119 PG_SESSION_IOSTAT

PG_SESSION_IOSTAT 视图 8.1.2 版本中已废弃，为兼容历史版本功能保留该视图，当前版本查询无效。

表14-166 PG_SESSION_IOSTAT 字段

名称	类型	描述
query_id	bigint	作业 ID。
mincurriops	integer	该作业当前 io 在各 DN 中的最小值。
maxcurriops	integer	该作业当前 io 在各 DN 中的最大值。
minpeakiops	integer	在作业运行时，作业 io 峰值中，各 DN 的最小值。
maxpeakiops	integer	在作业运行时，作业 io 峰值中，各 DN 的最大值。
io_limits	integer	该作业所设 GUC 参数 io_limits。
io_priority	text	该作业所设 GUC 参数 io_priority。
query	text	作业。
node_group	text	作业所属用户对应的逻辑集群。

14.3.120 PG_SETTINGS

PG_SETTINGS 视图显示数据库运行时参数的相关信息。

表14-167 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	参数的隐式结构。
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，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。

14.3.121 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。

这个系统表的名字来自于该表是不可读的，因为它包含口令。14.3.159 PG_USER 是一个在 PG_SHADOW 上公开可读的视图，只是把口令域填成了空白。

表14-168 PG_SHADOW 字段

名字	类型	引用	描述
usename	name	14.2.13 PG_AUTHID.rolname	用户名。
usesysid	oid	14.2.13 PG_AUTHID.oid	用户的 ID。
usecreatedb	boolean	-	用户可以创建数据库。
usesuper	boolean	-	用户是系统管理员。
usecatupd	boolean	-	用户可以更新系统表。即使是系统

名字	类型	引用	描述
			管理员，如果此字段不为真，也不能更新系统表。
userepl	boolean	-	用户可以初始化流复制和使系统处于或不处于备份模式。
passwd	text	-	口令（可能是加密的）；如果没有则为 null。参阅 14.2.13 PG_AUTHID 获取加密的口令是如何存储的信息。
valbegin	timestamp with time zone	-	帐户的有效开始时间；如果没有设置有效开始时间，则为 NULL。
valuntil	timestamp with time zone	-	帐户的有效结束时间；如果没有设置有效结束时间，则为 NULL。
respool	name	-	用户使用的资源池。
parent	oid	-	父资源池。
spacelimit	text	-	永久表存储空间限额。
tempspacelimit	text	-	临时表存储空间限额。
spillspacelimit	text	-	算子落盘空间限额。
useconfig	text[ ]	-	运行时配置变量的会话缺省。

14.3.122 PG_SHARED_MEMORY_DETAIL

PG_SHARED_MEMORY_DETAIL 视图查询所有已产生的共享内存上下文的使用信息。

表14-169 PG_SHARED_MEMORY_DETAIL 字段

名字	类型	描述
contextname	text	内存上下文的名字
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

名字	类型	描述
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

14.3.123 PG_STATS

PG_STATS 视图提供对存储在 pg_statistic 表里面的单列统计信息的访问。

表14-170 PG_STATS 字段

名称	类型	引用	描述
schemaname	name	14.2.38 PG_NAMESPACE.nspname	包含表的模式名。
tablename	name	14.2.17 PG_CLASS.relname	表名。
attname	name	14.2.12 PG_ATTRIBUTE.attname	字段名。
inherited	boolean	-	如果为真，则包含继承的子列，否则只是指定表的字段。
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。
n_distinct	real	-	• 如果大于 0，表示字段中独立数值的估计数目。 • 如果小于 0，表示独立数值的数目被行数除的负数。 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； 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。比如，-1 表示一个唯一字段，独立数值的个数和行数相同。
n_dndistinct	real	-	标识 dn1 上字段中非 NULL 数据的唯一值的数目。 • 如果大于 0，表示独立数值的实际数目。 • 如果小于 0，表示独立数值的数

名称	类型	引用	描述
			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 0，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。
most_common_freqs	real[]	-	一个最常用数值的频率的列表，即每个出现的次数除以行数。如果 most_common_vals 是 NULL，则为 NULL。
histogram_bounds	anyarray	-	一个数值的列表，它把字段的数值分成几组大致相同的组。如果在 most_common_vals 里有数值，则在此饼图的计算中省略。如果字段数据类型没有<操作符或者 most_common_vals 列表代表了整个分布性，则此字段为 NULL。
correlation	real	-	统计与字段值的物理行序和逻辑行序有关。它的范围从-1 到+1。在数值接近-1 或者+1 的时候，在字段上的索引扫描将被认为比它接近零的时候开销更少，因为减少了对磁盘的随机访问。如果字段数据类型没有<操作符，则这个字段为 NULL。
most_common_elems	anyarray	-	一个最常用的非空元素的列表。
most_common_elem_freqs	real[]	-	一个最常用元素的频率的列表。
elem_count_histogram	real[]	-	对于独立的非空元素的统计直方图。

14.3.124 PG_STAT_ACTIVITY

PG_STAT_ACTIVITY 视图显示和当前用户查询相关的信息。若有管理员权限或预置角色权限可以显示和所有用户查询相关的信息。

表14-171 PG_STAT_ACTIVITY 字段

名称	类型	描述
datid	oid	用户会话在后端连接到的数据库 OID。
datname	name	用户会话在后端连接到的数据库名称。
pid	bigint	后端线程 ID。
lwtid	integer	轻量级线程 ID。
usesysid	oid	登录该后端的用户 OID。
usename	name	登录该后端的用户名。
application_name	text	连接到该后端的应用名。
client_addr	inet	连接到该后端的客户端的 IP 地址。 如果此字段是 null，则表示通过服务器机器上 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，此字段是通过 client_addr 的反向 DNS 查找得到。此字段只有在启动 log_hostname 且使用 IP 连接时才非空。
client_port	integer	客户端用于与后端通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
backend_start	timestamp with time zone	后端进程启动时间，即客户端连接服务器的时间。
xact_start	timestamp with time zone	当前事务的启动时间，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	状态最后一次改变的时间。
waiting	boolean	如果后端当前正等待锁则为 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。 • waiting in global queue：表示语句在全局排队中。

名称	类型	描述
		• waiting in respool queue：表示语句在资源池排队中。 • waiting in ccn queue：表示作业在 CCN 排队中。 • 空或 no waiting queue：表示语句正在运行。

state	text	后端当前总体状态。可能值是：
		• active：后台正在执行查询。
		• idle：后台正在等待新的客户端命令。
		• idle in transaction：后端在事务中，但事务中没有语句在执行。
		• idle in transaction (aborted)：后端在事务中，但事务中有语句执行失败。
		• fastpath function call：后端正在执行一个 fast-path 函数。
		• disabled：如果后端禁用 track_activities，则报告此状态。
		说明
		普通用户只能查看到自己帐户所对应的会话状态。即其他帐户的 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 dbadmin 的 state 信息为空：
		`SELECT datname, usename, usesysid, state,pid FROM pg_stat_activity;`
		`datname | usename | usesysid | state | pid`
		`----------+----------+----------+----------+-----------------`
		`postgres | dbadmin | 10 | | 139968752121616`
		`postgres | dbadmin | 10 | | 139968903116560`
		`db_tpcds | judy | 16398 | active | 139968391403280`
		`postgres | dbadmin | 10 | | 139968643069712`
		`postgres | dbadmin | 10 | | 139968680818448`
		`postgres | joe | 16390 | | 139968563377936`
		`(6 rows)`

名称	类型	描述
resource_pool	name	用户使用的资源池。
stmt_type	text	语句类型。
query_id	bigint	查询语句的 ID。
query	text	此后端的最新查询。如果 state 状态是 active（活跃的），此字段显示当前正在执行的查询。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（参见 connection_info）。

14.3.125 PG_STAT_ALL_INDEXES

PG_STAT_ALL_INDEXES 视图将包含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，一个位图扫描增加 pg_stat_all_indexes.idx_tup_read 使用索引计数，并且增加 pg_stat_all_tables.idx_tup_fetch 表计数，但不影响 pg_stat_all_indexes.idx_tup_fetch。

表14-172 PG_STAT_ALL_INDEXES 字段

名称	类型	描述
relid	oid	索引的表的 OID
indexrelid	oid	索引的 OID
schemaname	name	索引中模式名
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

14.3.126 PG_STAT_ALL_TABLES

PG_STAT_ALL_TABLES 视图包含当前数据库中每个表每行的信息（包括 TOAST 表），显示访问特定表的统计信息。

表14-173 PG_STAT_ALL_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	此表的模式名。
relname	name	表名。
seq_scan	bigint	在此表上启动的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的有 live 数据行的数目。
idx_scan	bigint	索引扫描的次数。
idx_tup_fetch	bigint	索引扫描抓取的有 live 数据行的数目。
n_tup_ins	bigint	插入的行数。
n_tup_upd	bigint	更新的行数。
n_tup_del	bigint	删除的行数。
n_tup_hot_upd	bigint	HOT 更新的行数（即不需要单独的索引更新）。
n_live_tup	bigint	live 行估计数。
n_dead_tup	bigint	dead 行估计数。
last_vacuum	timestamp with time zone	最后一次手动 vacuum 时间（不计算 VACUUM FULL）。
last_autovacuum	timestamp with time zone	最后一次 autovacuum 时间。
last_analyze	timestamp with time zone	最后一次 analyze 时间。
last_autoanalyze	timestamp with time zone	最后一次 autovacuum 时间。
vacuum_count	bigint	vacuum 次数（不计算 VACUUM FULL）。
autovacuum_count	bigint	autovacuum 次数。
analyze_count	bigint	analyze 次数。
autoanalyze_count	bigint	autoanalyze 次数。
last_data_chang	timestamp with	记录该表最后一次数据发生变化的时间（引起数

名称	类型	描述
ed	time zone	据变化的操作包括 INSERT/UPDATE/DELETE、EXCHANGE/TRUNCATE/DROP partition），该列数据仅在本地 CN 记录。

14.3.127 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14-174 PG_STAT_BAD_BLOCK 字段

名字	类型	描述
nodename	text	节点名称
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
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	最近一次出现时间

14.3.128 PG_STAT_BGWRITER

PG_STAT_BGWRITER 视图显示关于后端写进程活动的统计信息。

表14-175 PG_STAT_BGWRITER 字段

名称	类型	描述
checkpoints_timed	bigint	定期执行的检查点数量。
checkpoints_req	bigint	请求执行的检查点数量。
checkpoint_write_time	double precision	检查点期间将文件写入磁盘花费的时间，以毫秒为单位。
checkpoint_syn	double	检查点期间数据同步到磁盘花费的时间，以毫秒

名称	类型	描述
c_time	precision	为单位。
buffers_checkpoint	bigint	检查点期间写入缓冲区的数量。
buffers_clean	bigint	后端写进程写的缓冲区数量。
maxwritten_clean	bigint	由于写入缓冲区太多，后端写进程停止清理扫描的次数。
buffers_backend	bigint	后端直接写入的缓冲区数量。
buffers_backend_fsync	bigint	后端需要 fsync 的次数。
buffers_alloc	bigint	分配的缓冲区数量。
stats_reset	timestamp with time zone	统计重置的时间。

14.3.129 PG_STAT_DATABASE

PG_STAT_DATABASE 视图显示当前节点上每个数据库的状态和统计信息。

表14-176 PG_STAT_DATABASE 字段

名称	类型	描述
datid	oid	数据库 OID。
datname	name	数据库名。
numbackends	integer	当前节点上连接到该数据库的后端数。 这是该视图中唯一一个反映目前状态值的列；所有列均返回自上次重置以来的累积值。
xact_commit	bigint	当前节点上该数据库中已经提交的事务数。
xact_rollback	bigint	当前节点上该数据库中已经回滚的事务数。
blks_read	bigint	当前节点上该数据库中读取的磁盘块的数量。
blks_hit	bigint	当前节点上高速缓存中发现的磁盘块的个数，即缓存中命中的块数（只包括 GaussDB(DWS)缓冲区高速缓存，不包括文件系统的缓存）。
tup_returned	bigint	当前节点上该数据库查询返回的行数。
tup_fetched	bigint	当前节点上该数据库查询抓取的行数。
tup_inserted	bigint	当前节点上该数据库插入的行数。

名称	类型	描述
tup_updated	bigint	当前节点上该数据库更新的行数。
tup_deleted	bigint	当前节点上该数据库删除的行数。
conflicts	bigint	当前节点上由于数据库恢复冲突取消的查询数量（只在备用服务器上发生）。可参见 14.3.130 PG_STAT_DATABASE_CONFLICTS。
temp_files	bigint	当前节点上该数据库创建的临时文件个数。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），而且不考虑 log_temp_files 设置。
temp_bytes	bigint	当前节点上该数据库写入临时文件的大小。计算所有临时文件，不论为什么创建临时文件，而且不考虑 log_temp_files 设置。
deadlocks	bigint	当前节点上该数据库中发生的死锁数量。
blk_read_time	double precision	当前节点上该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	当前节点上该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	当前节点上该数据库统计重置的时间。

14.3.130 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14-177 PG_STAT_DATABASE_CONFLICTS 字段

名称	类型	描述
datid	oid	数据库 OID
datname	name	数据库名
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

14.3.131 PG_STAT_GET_MEM_MBYTES_RESERVED

PG_STAT_GET_MEM_MBYTES_RESERVED 视图显示线程在内存中保存的当前活动信息。该函数在调用时需要指定线程 ID，线程 ID 的选取请参考 14.3.124 PG_STAT_ACTIVITY 中的 pid，线程 ID 为 0 时表示选取当前线程 ID，例如：

```
SELECT pg_stat_get_mem_mbytes_reserved(0);
```

表14-178 PG_STAT_GET_MEM_MBYTES_RESERVED 信息

名称	描述
ConnectInfo	连接信息
ParctlManager	并发管理信息
GeneralParams	基本参数信息
GeneralParams RPDATA	基本资源池信息
ExceptionManager	异常管理信息
CollectInfo	收集信息
GeneralInfo	基本信息
ParctlState	并发状态信息
CPU INFO	CPU 信息
ControlGroup	控制组信息
IOSTATE	IO 状态信息

14.3.132 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14-179 PG_STAT_USER_FUNCTIONS 字段

名称	类型	描述
funcid	oid	函数 OID
schemaname	name	模式名
funcname	name	函数名
calls	bigint	函数被调用的次数
total_time	double precision	函数的总执行时长

名称	类型	描述
self_time	double precision	当前线程调用函数的总的时长

14.3.133 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14-180 PG_STAT_USER_INDEXES 字段

名称	类型	描述
relid	oid	此索引表的 OID
indexrelid	oid	索引的 OID
schemaname	name	索引中模式名
relname	name	索引的表名
indexrelname	name	索引名
idx_scan	bigint	通过索引扫描的次数
idx_tup_read	bigint	通过索引上扫描返回的索引条目数量
idx_tup_fetch	bigint	索引扫描抓取的有 live 数据行的数目

14.3.134 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14-181 PG_STAT_USER_TABLES 字段

名称	类型	描述
relid	oid	表的 OID
schemaname	name	表的模式名
relname	name	表名
seq_scan	bigint	在此表上表启动的顺序扫描的次数
seq_tup_read	bigint	顺序扫描抓取的有 live 数据行的数目
idx_scan	bigint	索引扫描的次数
idx_tup_fetch	bigint	索引扫描抓取的有 live 数据行的数目

名称	类型	描述
n_tup_ins	bigint	插入的行数
n_tup_upd	bigint	更新的行数
n_tup_del	bigint	删除的行数
n_tup_hot_upd	bigint	HOT 更新的行数（即不需要单独的索引更新）
n_live_tup	bigint	live 行估计数
n_dead_tup	bigint	dead 行估计数
last_vacuum	timestamp with time zone	最后一次手动 vacuum 时间（不计算 VACUUM FULL）
last_autovacuum	timestamp with time zone	最后一次 autovacuum 时间
last_analyze	timestamp with time zone	最后一次 analyze 时间
last_autoanalyze	timestamp with time zone	最后一次 autoanalyze 时间
vacuum_count	bigint	vacuum 的次数（不计算 VACUUM FULL）
autovacuum_count	bigint	autovacuum 的次数
analyze_count	bigint	analyze 的次数
autoanalyze_count	bigint	autoanalyze 的次数

14.3.135 PG_STAT_REPLICATION

PG_STAT_REPLICATION 视图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14-182 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

名称	类型	描述
client_hostname	text	客户端名
client_port	integer	客户端端口号
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，还是一致的流状态）
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，还是潜在同步）

14.3.136 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14-183 PG_STAT_SYS_INDEXES 字段

名称	类型	描述
relid	oid	此索引表的 OID
indexrelid	oid	索引的 OID
schemaname	name	索引中模式名
relname	name	索引的表名
indexrelname	name	索引名
idx_scan	bigint	通过索引扫描的次数
idx_tup_read	bigint	通过索引上扫描返回的索引条目数
idx_tup_fetch	bigint	索引扫描抓取的有 live 数据行的数目

14.3.137 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14-184 PG_STAT_SYS_TABLES 字段

名称	类型	描述
relid	oid	表的 OID
schemaname	name	表的模式名
relname	name	表名
seq_scan	bigint	在此表上表启动的顺序扫描的次数
seq_tup_read	bigint	顺序扫描抓取的有 live 数据行的数目
idx_scan	bigint	索引扫描的次数
idx_tup_fetch	bigint	索引扫描抓取的有 live 数据行的数目
n_tup_ins	bigint	插入的行数
n_tup_upd	bigint	更新的行数
n_tup_del	bigint	删除的行数
n_tup_hot_upd	bigint	HOT 更新的行数（比如没有更新所需的单独索引）
n_live_tup	bigint	live 行估计数
n_dead_tup	bigint	dead 行估计数
last_vacuum	timestamp with time zone	最后一次手动 vacuum 时间（不计算 VACUUM FULL）
last_autovacuum	timestamp with time zone	最后一次 autovacuum 时间
last_analyze	timestamp with time zone	最后一次 analyze 时间
last_autoanalyze	timestamp with time zone	最后一次 autoanalyze 时间
vacuum_count	bigint	vacuum 的次数（不计算 VACUUM FULL）
autovacuum_count	bigint	autovacuum 的次数
analyze_count	bigint	analyze 的次数

名称	类型	描述
autoanalyze_count	bigint	autoanalyze 的次数

14.3.138 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14-185 PG_STAT_XACT_ALL_TABLES 字段

名称	类型	描述
relid	oid	表的 OID
schemaname	name	此表的模式名
relname	name	表名
seq_scan	bigint	在此表上启动的顺序扫描数
seq_tup_read	bigint	顺序扫描抓取的活跃行数
idx_scan	bigint	在此表上启动的索引扫描数
idx_tup_fetch	bigint	索引扫描抓取的活跃行数
n_tup_ins	bigint	插入的行数
n_tup_upd	bigint	更新的行数
n_tup_del	bigint	删除的行数
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）

14.3.139 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14-186 PG_STAT_XACT_SYS_TABLES 字段

名称	类型	描述
relid	oid	表的 OID
schemaname	name	此表的模式名
relname	name	表名
seq_scan	bigint	在此表上启动的顺序扫描数

名称	类型	描述
seq_tup_read	bigint	顺序扫描抓取的活跃行数
idx_scan	bigint	在此表上启动的索引扫描数
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）

14.3.140 PG_STAT_XACT_USER_FUNCTIONS

PG_STAT_XACT_USER_FUNCTIONS 视图包含每个跟踪函数的行，显示关于函数执行的统计。

表14-187 PG_STAT_XACT_USER_FUNCTIONS 字段

名称	类型	描述
funcid	oid	函数 OID
schemaname	name	模式名
funcname	name	函数名
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

14.3.141 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14-188 PG_STAT_XACT_USER_TABLES 字段

名称	类型	描述
relid	oid	表的 OID
schemaname	name	此表的模式名

名称	类型	描述
relname	name	表名
seq_scan	bigint	在该表上启动的顺序扫描数
seq_tup_read	bigint	顺序扫描抓取的活跃行数
idx_scan	bigint	在该表上启动的索引扫描数
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）

14.3.142 PG_STATIO_ALL_INDEXES

PG_STATIO_ALL_INDEXES 视图将包含当前数据库中的每个索引行， 显示特定索引的 I/O 的统计。

表14-189 PG_STATIO_ALL_INDEXES 字段

名称	类型	描述
relid	oid	此索引表的 OID
indexrelid	oid	索引的 OID
schemaname	name	索引中模式名
relname	name	索引的表名
indexrelname	name	索引名
idx_blks_read	bigint	从索引中读取的磁盘块数
idx_blks_hit	bigint	索引缓冲区命中数量

14.3.143 PG_STATIO_ALL_SEQUENCES

PG_STATIO_ALL_SEQUENCES 视图包含当前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14-190 PG_STATIO_ALL_SEQUENCES 字段

名称	类型	描述

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	序列缓冲区命中数量

14.3.144 PG_STATIO_ALL_TABLES

PG_STATIO_ALL_TABLES 视图将包含当前数据库中每个表的一行（包括 TOAST 表）， 显示出特定表 I/O 的统计。

表14-191 PG_STATIO_ALL_TABLES 字段

名称	类型	描述
relid	oid	表 OID
schemaname	name	此表模式名
relname	name	表名
heap_blks_read	bigint	从该表中读取的磁盘块数
heap_blks_hit	bigint	此表缓冲区命中数
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冲区数
toast_blks_read	bigint	此表的 TOAST 表读取的磁盘块数（如果存在）
toast_blks_hit	bigint	此表的 TOAST 表命中缓冲区数（如果存在）
tidx_blks_read	bigint	此表的 TOAST 表索引读取的磁盘块数（如果存在）
tidx_blks_hit	bigint	此表的 TOAST 表索引命中缓冲区数（如果存在）

14.3.145 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 IO 状态信息。

表14-192 PG_STATIO_SYS_INDEXES 字段

名称	类型	描述
relid	oid	此索引表的 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_blks_read	bigint	从索引中读取的磁盘块数
idx_blks_hit	bigint	索引缓冲区命中数量

14.3.146 PG_STATIO_SYS_SEQUENCES

PG_STATIO_SYS_SEQUENCES 视图显示命名空间中所有系统表为序列的 IO 状态信息。

表14-193 PG_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	序列缓冲区命中数量

14.3.147 PG_STATIO_SYS_TABLES

PG_STATIO_SYS_TABLES 视图显示命名空间中所有系统表的 IO 状态信息。

表14-194 PG_STATIO_SYS_TABLES 字段

名称	类型	描述
relid	oid	表 OID
schemaname	name	表模式名
relname	name	表名

名称	类型	描述
heap_blks_read	bigint	从表中读取的磁盘块数
heap_blks_hit	bigint	此表缓冲区命中数
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冲区数
toast_blks_read	bigint	此表的 TOAST 表读取的磁盘块数（如果存在）
toast_blks_hit	bigint	此表的 TOAST 表命中缓冲区数（如果存在）
tidx_blks_read	bigint	此表的 TOAST 表索引读取的磁盘块数（如果存在）
tidx_blks_hit	bigint	此表的 TOAST 表索引命中缓冲区数（如果存在）

14.3.148 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 IO 状态信息。

表14-195 PG_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	索引命中缓冲区数

14.3.149 PG_STATIO_USER_SEQUENCES

PG_STATIO_USER_SEQUENCES 视图显示命名空间中所有用户关系表类型为序列的 IO 状态信息。

表14-196 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	序列中缓存命中数

14.3.150 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 IO 状态信息。

表14-197 PG_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	此表缓冲区命中数
idx_blks_read	bigint	从表中所有索引读取的磁盘块数
idx_blks_hit	bigint	表中所有索引缓冲区命中数
toast_blks_read	bigint	此表的 TOAST 表读取的磁盘块数（如果存在）
toast_blks_hit	bigint	此表的 TOAST 表命中缓冲区数（如果存在）
tidx_blks_read	bigint	此表的 TOAST 表索引读取的磁盘块数（如果存在）
tidx_blks_hit	bigint	此表的 TOAST 表索引命中缓冲区数（如果存在）

14.3.151 PG_THREAD_WAIT_STATUS

通过 PG_THREAD_WAIT_STATUS 视图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。

表14-198 PG_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。
lwtid	integer	当前线程的轻量级线程号。
ptid	integer	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。等待状态的详细信息请参见表 14-199。
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、IO 的信息。否则为空。

wait_status 列的等待状态有以下状态。

表14-199 等待状态列表

wait_status 值	含义
none	未等任意事件。
acquire lock	等待加锁，要么加锁成功，要么加锁等待超时。
acquire lwlock	等待获取轻量级锁。
wait io	等待 IO 完成。
wait cmd	等待完成读取网络通信包。
wait pooler get conn	等待 pooler 完成获取连接。
wait pooler abort conn	等待 pooler 完成终止连接。
wait pooler clean conn	等待 pooler 完成清理连接。
pooler create conn: [nodename], total N	等待 pooler 建立连接，当前正在与 nodename 指定节点建立连接，且仍有 N 个连接等待建立。

wait_status 值	含义
get conn	获取到其他节点的连接。
set cmd: [nodename]	在连接上执行 SET/RESET/TRANSACTION BLOCK LEVEL PARA SET/SESSION LEVEL PARA SET，当前正在 nodename 指定节点上执行。
cancel query	取消某连接上正在执行的 SQL 语句。
stop query	停止某连接上正在执行的查询。
wait node: [nodename](plevel), total N, [phase]	等待接收与某节点的连接上的数据，当前正在等待 nodename 节点 plevel 线程的数据，且仍有 N 个连接的数据待返回。如果状态包含 phase 信息，则可能的阶段状态有： • begin：表示处于事务开始阶段。 • commit：表示处于事务提交阶段。 • rollback：表示处于事务回滚阶段。
wait transaction sync: xid	等待 xid 指定事务同步。
wait wal sync	等待特定 LSN 的 wal log 完成到备机的同步。
wait data sync	等待完成数据页到备机的同步。
wait data sync queue	等待把行存的数据页或列存的 CU 放入同步队列。
flush data: [nodename](plevel), [phase]	等待向网络中 nodename 指定节点的 plevel 对应线程发送数据。如果状态包含 phase 信息，则可能的阶段状态为 wait quota，即当前通信流正在等待 quota 值。
stream get conn: [nodename], total N	初始化 stream flow 时，等待与 nodename 节点的 consumer 对象建立连接，且当前有 N 个待建连对象。
wait producer ready: [nodename](plevel), total N	初始化 stream flow 时，等待每个 producer 都准备好，当前正在等待 nodename 节点 plevel 对应线程的 producer 对象准备好，且仍有 N 个 producer 对象处于等待状态。
synchronize quit	steam plan 结束时，等待 stream 线程组内的线程统一退出。
nodegroup destroy	steam plan 结束时，等待销毁 stream node group。
wait active statement	等待作业执行，正在资源负载管控中。
wait global queue	等待作业执行，正在全局队列排队。

wait_status 值	含义
wait respool queue	等待作业执行，正在资源池上排队。
wait ccn queue	等待作业执行，正在中心协调节点(CCN)中排队。
gtm connect	等待与 GTM 建连。
gtm get gxid	等待从 GTM 获取事务 xid。
gtm get snapshot	等待从 GTM 获取事务快照 snapshot。
gtm begin trans	等待 GTM 开始事务。
gtm commit trans	等待 GTM 提交事务。
gtm rollback trans	等待 GTM 执行事务回滚。
gtm create sequence	等待 GTM 创建 sequence。
gtm alter sequence	等待 GTM 修改 sequence。
gtm get sequence val	等待从 GTM 获取 sequence 的下一个值。
gtm set sequence val	等待 GTM 设置 sequence 的值。
gtm drop sequence	等待 GTM 删除 sequence。
gtm rename sequece	等待 GTM 重命名 sequence。
analyze: [relname], [phase]	当前正在对表 relname 执行 analyze。如果状态包含 phase 信息，则为 autovacuum，表示是数据库自动开启 AutoVacuum 线程执行的 analyze 分析操作。
vacuum: [relname], [phase]	当前正在对表 relname 执行 vacuum。如果状态包含 phase 信息，则为 autovacuum，表示是数据库自动开启 AutoVacuum 线程执行的 vacuum 清理操作。
vacuum full: [relname]	当前正在对表 relname 执行 vacuum full 清理。
create index	当前正在创建索引。
HashJoin - [build hash | write file]	当前是 HashJoin 算子，主要关注耗时的执行阶段。 • build hash：表示当前 HashJoin 算子正在建立哈希表。 • write file：表示当前 HashJoin 算子正在将数据写入磁盘。
HashAgg - [build hash | write file]	当前是 HashAgg 算子，主要关注耗时的执行阶段。

wait_status 值	含义
	• build hash：表示当前 HashAgg 算子正在建立哈希表。 • write file：表示当前 HashAgg 算子正在将数据写入磁盘。
HashSetop - [build hash | write file]	当前是 HashSetop 算子，主要关注耗时的执行阶段。 • build hash：表示当前 HashSetop 算子正在建立哈希表。 • write file：表示当前 HashSetop 算子正在将数据写入磁盘。
Sort | Sort - write file	当前是 Sort 算子做排序，write file 表示 Sort 算子正在将数据写入磁盘。
Material | Material - write file	当前是 Material 算子，write file 表示 Material 算子正在将数据写入磁盘。
wait sync consumer next step	Consumer 接收端同步等待下一轮迭代。
wait sync producer next step	Producer 发送端同步等待下一轮迭代。
wait agent release	正在释放当前 agent(8.1.2 及以上版本支持)。
wait stream task	stream 线程等待被复用(8.1.2 及以上版本支持)。

当 wait_status 为 acquire lwlock、acquire lock 或者 wait io 时，表示有等待事件。正在等待获取 wait_event 列对应类型的轻量级锁、事务锁，或者正在进行 IO。

其中，wait_status 值为 acquire lwlock（轻量级锁）时对应的 wait_event 等待事件类型与描述信息如下。（wait_event 为 extension 时，表示此时的轻量级锁是动态分配的锁，未被监控。）

表14-200 轻量级锁等待事件列表

wait_event 类型	类型描述
ShmemIndexLock	用于保护共享内存中的主索引哈希表。
OidGenLock	用于避免不同线程产生相同的 OID。
XidGenLock	用于避免两个事务获得相同的 xid。
ProcArrayLock	用于避免并发访问或修改 ProcArray 共享数组。
SInvalReadLock	用于避免与清理失效消息并发执行。
SInvalWriteLock	用于避免与其它写失效消息、清理失效消息并发执

wait_event 类型	类型描述
	行。
WALInsertLock	用于避免与其它 WAL 插入操作并发执行。
WALWriteLock	用于避免并发 WAL 写盘。
ControlFileLock	用于避免 pg_control 文件的读写并发、写写并发。
CheckpointLock	用于避免多个 checkpoint 并发执行。
CLogControlLock	用于避免并发访问或者修改 Clog 控制数据结构。
MultiXactGenLock	用于串行分配唯一 MultiXact id。
MultiXactOffsetControlLock	用于避免对 pg_multixact/offset 的写写并发和读写并发。
MultiXactMemberControlLock	用于避免对 pg_multixact/members 的写写并发和读写并发。
RelCacheInitLock	用于失效消息场景对 init 文件进行操作时加锁。
CheckpointerCommLock	用于向 checkpointer 发起文件刷盘请求场景，需要串行的向请求队列插入请求结构。
TwoPhaseStateLock	用于避免并发访问或者修改两阶段信息共享数组。
TablespaceCreateLock	用于确定 tablespace 是否已经存在。
BtreeVacuumLock	用于防止 vacuum 清理 B-tree 中还在使用的页面。
AutovacuumLock	用于串行化访问 autovacuum worker 数组。
AutovacuumScheduleLock	用于串行化分配需要 vacuum 的 table。
SyncScanLock	用于确定 heap 扫描时某个 relfilenode 的起始位置。
NodeTableLock	用于保护存放 CN 和 DN 节点信息的共享结构。
PoolerLock	用于保证两个线程不会同时从连接池里取到相同的连接。
RelationMappingLock	用于等待更新系统表到存储位置之间映射的文件。
AsyncCtlLock	用于避免并发访问或者修改共享通知状态。
AsyncQueueLock	用于避免并发访问或者修改共享通知信息队列。
SerializableXactHashLock	用于避免对于可串行事务共享结构的写写并发和读写并发。
SerializableFinishedListLock	用于避免对于已完成可串行事务共享链表的写写并发和读写并发。
SerializablePredicateLockLis	用于保护对于可串行事务持有的锁链表。

wait_event 类型	类型描述
tLock	
OldSerXidLock	用于保护记录冲突可串行事务的结构。
FileStatLock	用于保护存储统计文件信息的数据结构。
SyncRepLock	用于在主备复制时保护 xlog 同步信息。
DataSyncRepLock	用于在主备复制时保护数据页同步信息。
CStoreColspaceCacheLock	用于保护列存表的 CU 空间分配。
CStoreCUCacheSweepLock	用于列存 CU Cache 循环淘汰。
MetaCacheSweepLock	用于元数据循环淘汰。
DfsConnectorCacheLock	用于保护缓存 HDFS 连接的句柄的全局哈希表。
dummyServerInfoCacheLock	用于保护缓存加速集群连接信息的全局哈希表。
ExtensionConnectorLibLock	用于初始化 ODBC 连接场景，在加载与卸载特定动态库时进行加锁。
SearchServerLibLock	用于 GPU 加速场景初始化加载特定动态库时，对读文件操作进行加锁。
DfsUserLoginLock	用于保护 HDFS 用户信息的全局链表。
DfsSpaceCacheLock	用于控制 HDFS 表导入时文件 ID 单调递增。
LsnXlogChkFileLock	用于串行更新特定结构中记录的主备机的 xlog flush 位置点。
GTMHostInfoLock	用于避免并发访问或者修改 GTM 主机信息。
ReplicationSlotAllocationLock	用于主备复制时保护主机端的流复制槽的分配。
ReplicationSlotControlLock	用于主备复制时避免并发更新流复制槽状态。
ResourcePoolHashLock	用于避免并发访问或者修改资源池哈希表。
WorkloadStatHashLock	用于避免并发访问或者修改包含 CN 侧的 SQL 请求构成的哈希表。
WorkloadIoStatHashLock	用于避免并发访问或者修改用于统计当前 DN 的 IO 信息的哈希表。
WorkloadCGroupHashLock	用于避免并发访问或者修改 Cgroup 信息构成的哈希表。
OBSGetPathLock	用于避免对 obs 路径的写写并发和读写并发。
WorkloadUserInfoLock	用于避免并发访问或修改负载管理的用户信息哈希表。

wait_event 类型	类型描述
WorkloadRecordLock	用于避免并发访问或修改在内存自适应管理时对 CN 收到请求构成的哈希表。
WorkloadIOUtilLock	用于保护记录 iostat，CPU 等负载信息的结构。
WorkloadNodeGroupLock	用于避免并发访问或者修改内存中的 nodegroup 信息构成的哈希表。
JobShmemLock	用于 MPP 兼容 ORACLE 定时任务功能中保护定时读取的全局变量。
OBSRuntimeLock	用于获取环境变量，如 GAUSSHOME。
LLVMDumpIRLock	用于导出动态生成函数所对应的汇编语言。
LLVMParseIRLock	用于在查询开始处从 IR 文件中编译并解析已写好的 IR 函数。
RPNumberLock	用于加速集群的 DN 对正在执行计划的任务线程的计数。
ClusterRPLock	用于加速集群的 CCN 中维护的集群负载数据的并发存取控制。
CriticalCacheBuildLock	用于从共享或者本地缓存初始化文件中加载 cache 的场景。
WaitCountHashLock	用于保护用户语句计数功能场景中的共享结构。
BufMappingLock	用于保护对共享缓冲映射表的操作。
LockMgrLock	用于保护常规锁结构信息。
PredicateLockMgrLock	用于保护可串行事务锁结构信息。
OperatorRealTLock	用于避免并发访问或者修改记录算子级实时数据的全局结构。
OperatorHistLock	用于避免并发访问或者修改记录算子级历史数据的全局结构。
SessionRealTLock	用于避免并发访问或者修改记录 query 级实时数据的全局结构。
SessionHistLock	用于避免并发访问或者修改记录 query 级历史数据的全局结构。
CacheSlotMappingLock	用于保护 CU Cache 全局信息。
BarrierLock	用于保证当前只有一个线程在创建 Barrier。

当 wait_status 值为 wait io 时对应的 wait_event 等待事件类型与描述信息如下。

表14-201 IO 等待事件列表

wait_event 类型	类型描述
BufFileRead	从临时文件中读取数据到指定 buffer。
BufFileWrite	向临时文件中写入指定 buffer 中的内容。
ControlFileRead	读取 pg_control 文件。主要在数据库启动、执行 checkpoint 和主备校验过程中发生。
ControlFileSync	将 pg_control 文件持久化到磁盘。数据库初始化时发生。
ControlFileSyncUpdate	将 pg_control 文件持久化到磁盘。主要在数据库启动、执行 checkpoint 和主备校验过程中发生。
ControlFileWrite	写入 pg_control 文件。数据库初始化时发生。
ControlFileWriteUpdate	更新 pg_control 文件。主要在数据库启动、执行 checkpoint 和主备校验过程中发生。
CopyFileRead	copy 文件时读取文件内容。
CopyFileWrite	copy 文件时写入文件内容。
DataFileExtend	扩展文件时向文件写入内容。
DataFileFlush	将表数据文件持久化到磁盘
DataFileImmediateSync	将表数据文件立即持久化到磁盘。
DataFilePrefetch	异步读取表数据文件。
DataFileRead	同步读取表数据文件。
DataFileSync	将表数据文件的修改持久化到磁盘。
DataFileTruncate	表数据文件 truncate。
DataFileWrite	向表数据文件写入内容。
LockFileAddToDataDirRead	读取"postmaster.pid"文件。
LockFileAddToDataDirSync	将"postmaster.pid"内容持久化到磁盘。
LockFileAddToDataDirWrite	将 pid 信息写到"postmaster.pid"文件。
LockFileCreateRead	读取 LockFile 文件"%s.lock"。
LockFileCreateSync	将 LockFile 文件"%s.lock"内容持久化到磁盘。
LockFileCreateWRITE	将 pid 信息写到 LockFile 文件"%s.lock"。
RelationMapRead	读取系统表到存储位置之间的映射文件

wait_event 类型	类型描述
RelationMapSync	将系统表到存储位置之间的映射文件持久化到磁盘。
RelationMapWrite	写入系统表到存储位置之间的映射文件。
ReplicationSlotRead	读取流复制槽文件。重新启动时发生。
ReplicationSlotRestoreSync	将流复制槽文件持久化到磁盘。重新启动时发生。
ReplicationSlotSync	checkpoint 时将流复制槽临时文件持久化到磁盘。
ReplicationSlotWrite	checkpoint 时写流复制槽临时文件。
SLRUFlushSync	将 pg_clog、pg_subtrans 和 pg_multixact 文件持久化到磁盘。主要在执行 checkpoint 和数据库停机时发生。
SLRURead	读取 pg_clog、pg_subtrans 和 pg_multixact 文件。
SLRUSync	将脏页写入文件 pg_clog、pg_subtrans 和 pg_multixact 并持久化到磁盘。主要在执行 checkpoint 和数据库停机时发生。
SLRUWrite	写入 pg_clog、pg_subtrans 和 pg_multixact 文件。
TimelineHistoryRead	读取 timeline history 文件。在数据库启动时发生。
TimelineHistorySync	将 timeline history 文件持久化到磁盘。在数据库启动时发生。
TimelineHistoryWrite	写入 timeline history 文件。在数据库启动时发生。
TwophaseFileRead	读取 pg_twophase 文件。在两阶段事务提交、两阶段事务恢复时发生。
TwophaseFileSync	将 pg_twophase 文件持久化到磁盘。在两阶段事务提交、两阶段事务恢复时发生。
TwophaseFileWrite	写入 pg_twophase 文件。在两阶段事务提交、两阶段事务恢复时发生。
WALBootstrapSync	将初始化的 WAL 文件持久化到磁盘。在数据库初始化发生。
WALBootstrapWrite	写入初始化的 WAL 文件。在数据库初始化发生。
WALCopyRead	读取已存在的 WAL 文件并进行复制时产生的读操作。在执行归档恢复完后发生。
WALCopySync	将复制的 WAL 文件持久化到磁盘。在执行归档恢复完后发生。
WALCopyWrite	读取已存在 WAL 文件并进行复制时产生的写操作。在执行归档恢复完后发生。
WALInitSync	将新初始化的 WAL 文件持久化磁盘。在日志回收或写

wait_event 类型	类型描述
	日志时发生。
WALInitWrite	将新创建的 WAL 文件初始化为 0。在日志回收或写日志时发生。
WALRead	从 xlog 日志读取数据。两阶段文件 redo 相关的操作产生。
WALSyncMethodAssign	将当前打开的所有 WAL 文件持久化到磁盘。
WALWrite	写入 WAL 文件。

当 wait_status 值为 acquire lock（事务锁）时对应的 wait_event 等待事件类型与描述信息如下。

表14-202 事务锁等待事件列表

wait_event 类型	类型描述
relation	对表加锁
extend	对表扩展空间时加锁
partition	对分区表加锁
partition_seq	对分区表的分区加锁
page	对表页面加锁
tuple	对页面上的 tuple 加锁
transactionid	对事务 ID 加锁
virtualxid	对虚拟事务 ID 加锁
object	加对象锁
cstore_freespace	对列存空闲空间加锁
userlock	加用户锁
advisory	加 advisory 锁

14.3.152 PG_TABLES

PG_TABLES 视图提供了对数据库中每个表访问的有用信息。

表14-203 PG_TABLES 字段

名称	类型	引用	描述

名称	类型	引用	描述
schemaname	name	14.2.38 PG_NAMESPACE.nspname	包含表的模式名。
tablename	name	14.2.17 PG_CLASS.relname	表名。
tableowner	name	pg_get_userbyid(14.2.17 PG_CLASS.relowner)	表的所有者。
tablespace	name	14.2.61 PG_TABLESPACE.spcname	包含表的表空间，默认为NULL。
hasindexes	boolean	14.2.17 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	14.2.17 PG_CLASS.relhasruls	如果表上有规则，则为 TRUE，否则为 FALSE。
hastriggers	boolean	14.2.17 PG_CLASS.RELHASTRIGGERS	如果表上有触发器，则为 TRUE，否则为 FALSE。
tablecreator	name	pg_get_userbyid(14.2.39 PG_OBJECT.creator)	表创建用户，如创建用户已删除，则返回空。
created	timestamp with time zone	14.2.39 PG_OBJECT.ctime	表创建时间。
last_ddl_time	timestamp with time zone	14.2.39 PG_OBJECT.mtime	表最后修改时间。

14.3.153 PG_TDE_INFO

PG_TDE_INFO 视图提供了当前集群加密信息。

表14-204 PG_TDE_INFO 字段

名称	类型	描述
is_encrypt	text	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

应用示例

查看当前集群是否加密/查看当前集群的加密算法：

```
SELECT * FROM PG_TDE_INFO;
 is_encrypt | g_tde_algo  | remain
------------+-------------+--------
 f          | AES-CTR-128 | remain
(1 row)
```

14.3.154 PG_TIMEZONE_ABBREVS

PG_TIMEZONE_ABBREVS 视图提供了输入例程能够识别的所有时区缩写。

表14-205 PG_TIMEZONE_ABBREVS 字段

名称	类型	描述
abbrev	text	时区缩写
utc_offset	interval	相对于 UTC 的偏移量
is_dst	boolean	如果这是一个夏时制时区缩写则为 TRUE，否则为 FALSE

14.3.155 PG_TIMEZONE_NAMES

PG_TIMEZONE_NAMES 视图提供了显示了所有能够被 SET TIMEZONE 识别的时区名及其缩写、UTC 偏移量、是否夏时制。

表14-206 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	如果当前正处于夏令时范围则为 TRUE，否则为 FALSE

14.3.156 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14-207 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，包括以下几种： • max_process_memory：GaussDB(DWS)集群实例所占用的内存大小。 • process_used_memory：GaussDB(DWS)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_cstore_memory：列存所允许使用的最大内存。 • cstore_used_memory：列存已使用的内存大小。 • 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。 • max_topsql_memory：TopSQL 记录历史作业监控信息允许使用的最大内存。 • topsql_used_memory：TopSQL 记录历史作业监控信息已使用的内存大小。 • topsql_peak_memory：TopSQL 记录历史作业监控信息的内存峰值。 • other_used_memory：其他已使用的内存大小。 • gpu_max_dynamic_memory：GPU 内存最大值。 • gpu_dynamic_used_memory：当前 GPU 可用内存和当前临时 GPU 内存之和。 • gpu_dynamic_peak_memory：GPU 内存使用

名称	类型	描述
		的最大内存。 • pooler_conn_memory：pooler 连接占用内存大小。 • pooler_freeconn_memory：pooler 空闲连接占用的内存大小。 • storage_compress_memory：列存压缩和解压缩使用的内存大小。 • udf_reserved_memory：为 UDF Worker 进程预留的内存大小。 • mmap_used_memory：mmap 使用的内存大小。
memorymbytes	integer	内存使用的大小，单位 MB

14.3.157 PG_TOTAL_SCHEMA_INFO

PG_TOTAL_SCHEMA_INFO 视图显示各个数据库下所有 Schema 的存储资源使用情况。此视图在参数 use_workload_manager 为 on 时才有效。

名称	类型	描述
schemaid	oid	Schema 的 OID
schemaname	text	Schema 名称
databaseid	oid	数据库的 OID
databasename	name	数据库名称
usedspace	bigint	该 Schema 已使用的永久表存储空间大小，单位 Byte
permspace	bigint	该 Schema 的永久表存储空间上限值，单位 Byte

14.3.158 PG_TOTAL_USER_RESOURCE_INFO

PG_TOTAL_USER_RESOURCE_INFO 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。此视图在参数 use_workload_manager 为 on 时才有效。

表14-208 PG_TOTAL_USER_RESOURCE_INFO 字段

名称	类型	描述
username	name	用户名。

名称	类型	描述
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（仅统计非默认资源池上复杂作业的 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 表示未限制永久表存储空间。
used_temp_space	bigint	已使用的临时表存储空间大小，单位 KB。
total_temp_space	bigint	可使用的临时表存储空间大小，单位 KB，值为-1 表示未限制临时表存储空间。
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大小，单位 KB，值为-1 表示未限制算子落盘空间。
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

名称	类型	描述
		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)
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)

14.3.159 PG_USER

PG_USER 视图提供了访问数据库用户的信息。

表14-209 PG_USER 字段

名称	类型	描述
usename	name	用户名。
usesysid	oid	此用户的 ID。
usecreatedb	boolean	用户是否可以创建数据库。
usesuper	boolean	用户是否是拥有最高权限的初始系统管理员。
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。
userepl	boolean	用户是否可以复制数据流。
passwd	text	密文存储后的用户口令，始终为********。
valbegin	timestamp with time zone	帐户的有效开始时间；如果没有设置有效开始时间，则为 NULL。
valuntil	timestamp with time zone	帐户的有效结束时间；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。
parentid	oid	父用户 OID
spacelimit	text	永久表存储空间限额。
tempspacelimit	text	临时表存储空间限额。
spillspacelimit	text	算子落盘空间限额。

名称	类型	描述
useconfig	text[]	运行时配置参数的会话缺省。
nodegroup	name	用户关联的逻辑集群名字，如果该用户没有管理逻辑集群，则该字段为空。

14.3.160 PG_USER_MAPPINGS

PG_USER_MAPPINGS 视图提供访问关于用户映射的信息的接口。

这个视图只是一个 14.2.69 PG_USER_MAPPING 的可读部分的视图化表现，如果用户无权使用它则查询此表时，有些选项字段会显示为空。

表14-210 PG_USER_MAPPINGS 字段

名字	类型	引用	描述
umid	oid	14.2.69 PG_USER_MAPPING.oid	用户映射的 OID。
srvid	oid	14.2.30 PG_FOREIGN_SERVER.oid	包含这个映射的外部服务器的 OID。
srvname	name	14.2.30 PG_FOREIGN_SERVER.srvname	外部服务器的名字。
umuser	oid	14.2.13 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字。
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。

14.3.161 PG_VIEWS

PG_VIEWS 视图提供访问数据库中每个视图的有用信息。

表14-211 PG_VIEWS 字段

名称	类型	引用	描述
schemaname	name	14.2.38 PG_NAMESPACE.nspname	包含视图的模式名

名称	类型	引用	描述
viewname	name	14.2.17 PG_CLASS.relname	视图的名称
viewowner	name	14.2.13 PG_AUTHID.Erolname	视图的所有者
definition	text	-	视图的定义

14.3.162 PG_WLM_STATISTICS

PG_WLM_STATISTICS 视图显示作业结束后或已被处理异常后的负载管理相关信息。
该视图 8.1.2 版本中已废弃。

表14-212 PG_WLM_STATISTICS 字段

名称	类型	描述
statement	text	执行了异常处理的语句。
block_time	bigint	语句执行前的阻塞时间。
elapsed_time	bigint	语句的实际执行时间。
total_cpu_time	bigint	语句执行异常处理时 DN 上 CPU 使用的总时间。
qualification_time	bigint	语句检查倾斜率的时间周期。
cpu_skew_percent	integer	语句在执行异常处理时 DN 上 CPU 使用的倾斜率。
control_group	text	语句执行异常处理时所使用的 Cgroups。
status	text	语句执行异常处理后的状态，包括： • pending：执行前预备状态。 • running：执行进行状态。 • finished：执行正常结束。 • abort：执行异常终止。
action	text	语句执行的异常处理动作，包括： • abort：执行终止操作。 • adjust：执行 Cgroups 调整操作，目前只有降级操作。 • finish：正常结束。
queryid	bigint	语句执行所使用的内部 query_id。
threadid	bigint	后端线程 ID。

14.3.163 PGXC_BULKLOAD_PROGRESS

PGXC_BULKLOAD_PROGRESS 显示导入业务的执行进度，仅支持 GDS 普通文件导入业务。需要有系统管理员权限才可以访问此视图

表14-213 PGXC_BULKLOAD_PROGRESS 字段

名称	类型	描述
session_id	bigint	GDS 的会话 ID。
query_id	bigint	查询 ID，对应 debug_query_id。
query	text	查询语句。
progress	text	进度百分比。

14.3.164 PGXC_BULKLOAD_STATISTICS

通过 CN 查看 PGXC_BULKLOAD_STATISTICS 视图，可获取 GDS、COPY、\COPY 等业务执行过程中的实时统计信息。该视图汇总当前集群上各个节点正在执行的导入/导出类业务的实时执行情况，从而可以监控导入导出类业务的实时进度，辅助进行性能问题排查。

PGXC_BULKLOAD_STATISTICS 视图与 PG_BULKLOAD_STATISTICS 视图列定义完全相同。这是由于 PGXC_BULKLOAD_STATISTICS 视图本质是到集群中各个节点上查询 PG_BULKLOAD_STATISTICS 视图汇总的结果。

需要有系统管理员权限才可以访问此视图。

表14-214 PGXC_BULKLOAD_STATISTICS 字段

名称	类型	描述
node_name	text	节点名称。
db_name	text	数据库名称。
query_id	bigint	查询 ID，对应 debug_query_id。
tid	bigint	当前线程的线程号。
lwtid	integer	当前线程的轻量级线程号。
session_id	bigint	GDS 的会话 ID。
direction	text	业务类型，取值包括：gds to file、gds from file、gds to pipe、gds from pipe、copy from、copy to。
query	text	查询语句。

address	text	当前导入导出外表的 location。
query_start	timestamp with time zone	导入/导出开始时间。
total_bytes	bigint	待处理数据的总大小。 仅 GDS 普通文件导入时，且该行记录来自 CN 节点才会显示，否则为空。
phase	text	当前阶段，包括：INITIALIZING、TRANSFER_DATA、RELEASE_RESOURCE。
done_lines	bigint	已传输行数。
done_bytes	bigint	已传输字节数。

14.3.165 PGXC_COLUMN_TABLE_IO_STAT

PGXC_COLUMN_TABLE_IO_STAT 视图提供集群所有 CN 和 DN 节点上当前数据库所有列存表的 IO 统计数据。除在每一行前面增加 name 类型的 nodename 字段外，其余字段的名称、类型和顺序与 GS_COLUMN_TABLE_IO_STAT 视图相同，具体的字段请参考 14.3.55 GS_COLUMN_TABLE_IO_STAT。

需要有系统管理员权限才可以访问此视图。

14.3.166 PGXC_COMM_CLIENT_INFO

PGXC_COMM_CLIENT_INFO 视图存储所有节点客户端连接信息（DN 上查询该视图显示 CN 连接 DN 的信息）。

表14-215 PGXC_COMM_CLIENT_INFO 字段

名称	类型	描述
node_name	text	当前节点的名称。
app	text	客户端应用名。
tid	bigint	当前线程的线程号。
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，展示 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，展示 sid，显示-1 时表示当前连

名称	类型	描述
		接是物理连接。

14.3.167 PGXC_COMM_DELAY

PGXC_COMM_DELAY 视图展示所有 DN 的通信库时延状态。

表14-216 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

14.3.168 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14-217 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。

名称	类型	描述
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：表示逻辑连接接收报文正常。 • HOLD：表示逻辑连接接收报文等待中。 • 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。

14.3.169 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14-218 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。

名称	类型	描述
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。
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。

14.3.170 PGXC_COMM_STATUS

PGXC_COMM_STATUS 视图展示所有 DN 的通信库状态。

表14-219 PGXC_COMM_STATUS 字段

名称	类型	描述
node_name	text	节点名称。
rxpck/s	integer	节点通信库接收速率，单位 Byte/s。
txpck/s	integer	节点通信库发送速率，单位 Byte/s。
rxkB/s	bigint	节点通信库接收速率，单位 KByte/s。
txkB/s	bigint	节点通信库发送速率，单位 KByte/s。
buffer	bigint	cmailbox 的 buffer 大小。
memKB(libcomm)	bigint	libcomm 进程通信内存大小，单位 KByte。

名称	类型	描述
memKB(libpq)	bigint	libpq 进程通信内存大小，单位 KByte。
%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 线程中最高的实时使用率。
stream	integer	当前使用的逻辑连接总数。

14.3.171 PGXC_COMM_QUERY_SPEED

PGXC_COMM_QUERY_SPEED 视图展示所有节点上所有 query 对应的流量信息。

表14-220 PGXC_COMM_QUERY_SPEED 字段

名称	类型	描述
node_name	text	节点名称。
query_id	bigint	通信流对应的 debug_query_id 编号。
rxkB/s	bigint	查询对应的通信流接收速率，单位 Byte/s。
txkB/s	bigint	查询对应的通信流发送速率，单位 Byte/s。
rxkB	bigint	查询对应的通信流接收数据总量，单位 Byte。
txkB	bigint	查询对应的通信流发送数据总量，单位 Byte。
rxpck/s	bigint	查询对应的通信包接收速率，单位 packages/s。
txpck/s	bigint	查询对应的通信包发送速率，单位 packages/s。
rxpck	bigint	查询对应的通信包接收总量，单位 packages。
txpck	bigint	查询对应的通信包发送总量，单位 packages。

14.3.172 PGXC_DEADLOCK

PGXC_DEADLOCK 视图获取导致分布式死锁产生的锁等待信息。

目前，PGXC_DEADLOCK 视图只收集 locktype 为 relation、partition、page、tuple 和 transactionid 的锁等待信息。

表14-221 PGXC_DEADLOCK 字段

名称	类型	描述
locktype	text	被锁定对象的类型。
nodename	name	被锁定对象的节点的名称。
dbname	name	被锁定对象的数据库的名称。如果被锁定对象是事务，则为 NULL。
nspname	name	被锁定对象的命名空间的名称。
relname	name	被锁定对象对应的关系的名称。如果被锁定对象既不是关系，也不是关系的一部分，则为 NULL。
partname	name	被锁定对象对应的分区的名称。如果被锁定对象不是分区，则为 NULL。
page	integer	被锁定对象对应的页面的编号。如果被锁定对象既不是页面，也不是元组，则为 NULL。
tuple	smallint	被锁定对象对应的元组的编号。如果被锁定对象不是元组，则为 NULL。
transactionid	xid	被锁定对象对应的事务的 ID。如果被锁定对象不是事务，则为 NULL。
waitusername	name	等待锁的用户的名称。
waitgxid	xid	等待锁的事务的 ID。
waitxactstart	timestamp with time zone	等待锁的事务的开始时间。
waitqueryid	bigint	等待锁的线程的最新查询 ID。
waitquery	text	等待锁的线程的最新查询语句。
waitpid	bigint	等待锁的线程的 ID。
waitmode	text	等待的锁的级别。
holdusername	name	持有锁的用户的名称。
holdgxid	xid	持有锁的事务的 ID。
holdxactstart	timestamp with time zone	持有锁的事务的开始时间。
holdqueryid	bigint	持有锁的线程的最新查询 ID。
holdquery	text	持有锁的线程的最新查询语句。
holdpid	bigint	持有锁的线程的 ID。

名称	类型	描述
holdmode	text	持有的锁的级别。

14.3.173 PGXC_GET_STAT_ALL_TABLES

PGXC_GET_STAT_ALL_TABLES 视图获取各表的插入、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用户操作该系统表时，再执行 VACUUM FULL。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。

表14-222 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	live 元组的条数
n_dead_tup	numeric	dead 元组的条数
page_dirty_rate	numeric(5,2)	表的脏页率信息(%)

同时 GaussDB(DWS)提供了函数 pgxc_get_stat_dirty_tables(int dirty_percent, int n_tuples)和 pgxc_get_stat_dirty_tables(int dirty_percent, int n_tuples, text schema)可以快速筛选出脏页率大于 dirty_percent，dead 元组数大于 n_tuples，模式名是 schema 的表。详细内容可参考《SQL 语法参考》的“函数和操作符>系统管理函数>其他函数”章节。

应用示例

使用 PGXC_GET_STAT_ALL_TABLES 视图查询数据库内脏页率大于 30%的表：

```
select * from PGXC GET STAT ALL TABLES where dirty page rate>30;
 relid |        relname        | schemaname | n tup ins | n tup upd | n tup del |
n live tup | n dead tup | dirty page rate
-------+-----------------------+------------+-----------+-----------+-----------
+------------+------------+-----------------
  2840 | pg_toast_2619         | pg_toast   |     7415 |        0 |     7415 |
0 |       291 |         88.00
```

```
  9001 | pgxc_class            | pg_catalog |    56331 |        3 |    56285 |
54 |       143 |         72.59
 53860 | reason               | dbadmin    |        9 |       19 |        0 |         9
|        19 |         67.86
  9025 | pg_object            | pg_catalog |   112858 |  1179707 |   112619 |
246 |       429 |         63.56
  9015 | pgxc_node            | pg_catalog |       15 |       24 |        0 |
15 |        24 |         61.54
  2606 | pg_constraint         | pg_catalog |       78 |        0 |       42 |
36 |        42 |         53.85
  1260 | pg_authid            | pg_catalog |        6 |        6 |        0 |
6 |         6 |         50.00
(7 rows)
```

也可以使用 pgxc_get_stat_dirty_tables 函数查询数据库内脏页率大于 10%，脏数据行数大于 1000 行的表：

```
select
a.schemaname,a.relname,pg_size_pretty(pg_table_size(b.oid)),a.dirty_page_rate from
pgxc_get_stat_dirty_tables(10,1000) a,pg_catalog.pg_class b where a.relname =
b.relname order by pg_table_size(b.oid) desc;
 schemaname |   relname    | pg_size_pretty | dirty_page_rate
------------+--------------+----------------+-----------------
 pg_catalog | pg_attribute | 2792 KB        |          12.09
 pg_catalog | pg_class     | 568 KB         |          15.36
 pg_catalog | pg_type      | 368 KB         |          12.17
(3 rows)
```

14.3.174 PGXC_GET_STAT_ALL_PARTITIONS

PGXC_GET_STAT_ALL_PARTITIONS 视图获取各分区表分区的插入、更新、删除以及脏页率信息。

该视图的统计信息依赖于 ANALYZE，为获取最准确的信息请先对分区表进行 ANALYZE。

表14-223 PGXC_GET_STAT_ALL_PARTITIONS 字段

名称	类型	描述
relid	oid	表的 oid。
partid	oid	分区的 oid
schename	name	表 schemaname。
relname	name	表名。
partname	name	分区名。
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。

名称	类型	描述
n_live_tup	numeric	存活元组的条数。
n_dead_tup	numeric	死亡元组的条数。
page_dirty_rate	numeric(5,2)	表的脏页率信息(%)。

应用示例

查询数据库内脏页率大于 30%的分区表：

```
select * from PGXC_GET_STAT_ALL_PARTITIONS where dirty_page_rate>30;
 relid | partid |   schemaname    |     relname      | partname | n_tup_ins |
n_tup_upd | n_tup_del | n_live_tup | n_dead_tup | dirty_page_rate
-------+--------+-----------------+------------------+----------+-----------+----
-------+-----------+------------+------------+-----------------
 58320 |  58626 | schema_subquery | store_hash_par    | p1      |        2 |        0
|        2 |         0 |         2 |         100.00
 58430 |  58706 | schema_subquery | store_hash_par_mor | p4      |        1 |
1 |        1 |         0 |         2 |         100.00
 58320 |  58644 | schema_subquery | store_hash_par    | p1      |        3 |        0
|        3 |         0 |         3 |         100.00
 58430 |  58770 | schema_subquery | store_hash_par_mor | p4      |        1 |
1 |        1 |         0 |         2 |         100.00
 58320 |  58643 | schema_subquery | store_hash_par    | p1      |        2 |        0
|        2 |         0 |         2 |         100.00
 58320 |  58625 | schema_subquery | store_hash_par    | p1      |        2 |        0
|        2 |         0 |         2 |         100.00
 58320 |  58579 | schema subquery | store hash par    | p1      |        2 |        0
|        2 |         0 |         2 |         100.00
 58320 |  58619 | schema subquery | store hash par    | p1      |        3 |        0
|        3 |         0 |         3 |         100.00
 58320 |  58627 | schema subquery | store hash par    | p1      |        4 |        0
|        4 |         0 |         4 |         100.00
 58320 |  58657 | schema_subquery | store_hash_par    | p1      |        3 |        0
|        3 |         0 |         3 |         100.00
(10 rows)
```

14.3.175 PGXC_GET_TABLE_SKEWNESS

PGXC_GET_TABLE_SKEWNESS 视图展示当前库中表的数据分布倾斜情况。需要有系统管理员权限或预置角色 gs_role_read_all_stats 权限才可以访问此视图。

表14-224 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。

名称	类型	描述
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）。

应用示例

查询当前数据库中（库中表个数少于 1W 的场景）所有表的数据倾斜情况：

```
SELECT * FROM pgxc_get_table_skewness ORDER BY totalsize DESC;
 schemaname |       tablename       | totalsize | avgsize | maxratio | minratio |
skewsize | skewratio | skewstddev
------------+-----------------------+-----------+---------+----------+----------
+----------+-----------+------------
 dbadmin    | reason                |    147456 |   49152 |     .333 |     .333 |       0
|     0.000 |         0
 tpcds      | reason_t2             |     73728 |   24576 |     .556 |    0.000 |
40960 |     .556 |     21674
 dbadmin    | reason_bk             |     65536 |   21845 |     .500 |    0.000 |
32768 |     .500 |     18919
 tsearch    | pgweb                 |     49152 |   16384 |     .333 |     .333 |       0
|     0.000 |         0
 dbadmin    | student               |     40960 |   13653 |     .400 |     .200 |
8192 |     .200 |      4730
 tsearch    | ts_zhparser           |     40960 |   13653 |     .400 |     .200 |
8192 |     .200 |      4730
 dbms_om    | gs_wlm_session_info   |     24576 |    8192 |     .333 |     .333 |
0 |     0.000 |         0
 dbms_om    | gs_wlm_ec_operator_info |     24576 |    8192 |     .333 |     .333 |
0 |     0.000 |         0
 dbms_om    | gs_wlm_operator_info  |     24576 |    8192 |     .333 |     .333 |
0 |     0.000 |         0
(9 rows)
```

若数据库中表个数非常多（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;
```

14.3.176 PGXC_GTM_SNAPSHOT_STATUS

PGXC_GTM_SNAPSHOT_STATUS 视图用于查看当前 GTM 上事务信息。

表14-225 PGXC_GTM_SNAPSHOT_STATUS 字段

名称	类型	描述
xmin	xid	仍在运行的最小事务号。
xmax	xid	已完成的事务号最大的事务的下一个事务号。
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。
running_xids	text	当前活跃的事务号。

14.3.177 PGXC_INSTANCE_TIME

PGXC_INSTANCE_TIME 视图显示集群中各节点上进程的运行时间信息及各执行阶段所消耗时间，除新增 node_name（节点名称）字段外，其余字段内容和 14.3.223 PV_INSTANCE_TIME 视图相同。需要有系统管理员权限或预置角色 gs_role_read_all_stats 权限才可以访问此视图。

14.3.178 PGXC_INSTR_UNIQUE_SQL

PGXC_INSTR_UNIQUE_SQL 视图展示集群中所有 CN 节点的 Unique SQL 的完整统计信息。

需要有系统管理员权限才可以访问此视图，具体的字段请参考 14.3.56 GS_INSTR_UNIQUE_SQL。

14.3.179 PGXC_LOCK_CONFLICTS

PGXC_LOCK_CONFLICTS 视图提供集群中有冲突的锁的信息。

当某一个锁正在等待另一个锁，或正在被另一个锁等待，即该锁是有冲突的。

目前，PGXC_LOCK_CONFLICTS 视图只收集 locktype 为 relation、partition、page、tuple 和 transactionid 的锁的信息。

表14-226 PGXC_LOCK_CONFLICTS 字段

名称	类型	描述
locktype	text	被锁定对象的类型。
nodename	name	被锁定对象的节点的名称。
dbname	name	被锁定对象的数据库的名称。如果被锁定对象是事务，则为 NULL。
nspname	name	被锁定对象的命名空间的名称。
relname	name	被锁定对象对应的关系的名称。如果被锁定对象既不是关系，也不是关系的一部分，则为 NULL。
partname	name	被锁定对象对应的分区的名称。如果被锁定对象不是分区，则为 NULL。
page	integer	被锁定对象对应的页面的编号。如果被锁定对象既不是页面，也不是元组，则为 NULL。
tuple	smallint	被锁定对象对应的元组的编号。如果被锁定对象不是元组，则为 NULL。
transactionid	xid	被锁定对象对应的事务的 ID。如果被锁定对象不是事务，则为 NULL。
username	name	申请锁的用户的名称。
gxid	xid	申请锁的事务的 ID。
xactstart	timestamp with time zone	申请锁的事务的开始时间。
queryid	bigint	申请锁的线程的最新查询 ID。
query	text	申请锁的线程的最新查询语句。
pid	bigint	申请锁的线程的 ID。
mode	text	锁的级别。
granted	boolean	• 如果锁已被持有，则为 TRUE。 • 如果锁还在等待其它锁，则为 FALSE。

14.3.180 PGXC_NODE_ENV

PGXC_NODE_ENV 视图提供获取集群中所有节点的环境变量信息。

表14-227 PGXC_NODE_ENV 字段

名称	类型	描述
node_name	text	集群中所有节点的名称
host	text	集群中所有节点的主机名称
process	integer	集群中所有节点的进程号
port	integer	集群中所有节点的端口号
installpath	text	集群中所有节点的安装目录
datapath	text	集群中所有节点的数据目录
log_directory	text	集群中所有节点的日志目录

14.3.181 PGXC_NODE_STAT_RESET_TIME

PGXC_NODE_STAT_RESET_TIME 视图显示集群中各节点的统计信息重置时间，除新增 node_name（节点名称）字段外，其余字段内容和 14.3.57 GS_NODE_STAT_RESET_TIME 视图相同。需要有系统管理员权限才可以访问此视图。

14.3.182 PGXC_OS_RUN_INFO

PGXC_OS_RUN_INFO 视图显示集群中各节点上操作系统运行的状态信息，除新增 node_name（节点名称）字段外，其余字段内容和 14.3.224 PV_OS_RUN_INFO 视图相同。需要有系统管理员权限或预置角色 gs_role_read_all_stats 权限才可以访问此视图。

14.3.183 PGXC_OS_THREADS

PGXC_OS_THREADS 视图提供当前集群中所有正常节点下的线程状态信息。

表14-228 PGXC_OS_THREADS 字段

名称	类型	描述
node_name	text	当前集群中所有正常节点的名称
pid	bigint	当前集群中所有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

名称	类型	描述
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

14.3.184 PGXC_PREPARED_XACTS

PGXC_PREPARED_XACTS 视图显示当前处于 prepared 阶段的两阶段事务。

表14-229 PGXC_PREPARED_XACTS 字段

名字	类型	描述
pgxc_prepared_xact	text	查看当前处于 prepared 阶段的两阶段事务。

14.3.185 PGXC_REDO_STAT

视图 PGXC_REDO_STAT 显示集群中各节点上 XLOG 重做过程中的统计信息，除新增 node_name（节点名称）字段外，其余字段内容和 14.3.230 PV_REDO_STAT 视图相同。需要有系统管理员权限或预置角色 gs_role_read_all_stats 权限才可以访问此视图。

14.3.186 PGXC_REL_IOSTAT

PGXC_REL_IOSTAT 视图显示集群中各节点上磁盘读写的统计信息。除新增 node_name（节点名称）字段外，其余字段内容和 14.3.58 GS_REL_IOSTAT 视图相同。需要有系统管理员权限才可以访问此视图。

14.3.187 PGXC_REPLICATION_SLOTS

PGXC_REPLICATION_SLOTS 视图显示集群中 DN 上的复制信息，除增加表示节点名称的 node_name 字段外，其余字段内容和 14.3.113 PG_REPLICATION_SLOTS 视图相同。需要有系统管理员权限才可以访问此视图。

14.3.188 PGXC_RESPOOL_RUNTIME_INFO

PGXC_RESPOOL_RUNTIME_INFO 视图显示所有 CN 上所有资源池作业运行信息。

表14-230 PGXC_RESPOOL_RUNTIME_INFO 字段

名称	类型	描述
nodename	name	CN 名称
nodegroup	name	资源池所属逻辑集群名称，默认集群显示 "installation"

名称	类型	描述
rpname	name	资源池名称
ref_count	int	资源池引用作业数，作业经过资源池不管是否管控都会计数
fast_run	int	资源池快车道运行作业数
fast_wait	int	资源池快车道排队作业数
slow_run	int	资源池慢车道运行作业数
slow_wait	int	资源池慢车道排队作业数

14.3.189 PGXC_RESPOOL_RESOURCE_INFO

PGXC_RESPOOL_RESOURCE_INFO 视图显示所有实例上资源池实时监控信息。

说明

DN 上仅显示当前 DN 所属逻辑集群的资源池监控信息。

表14-231 PGXC_RESPOOL_RESOURCE_INFO 字段

名称	类型	描述
nodename	name	实例名称，包含 CN 和 DN
nodegroup	name	资源池所属逻辑集群名称，默认集群显示 "installation"
rpname	name	资源池名称
cgroup	name	资源池关联控制组名称
ref_count	int	资源池引用作业数，作业经过资源池不管是否管控都会计数，仅 CN 上有效
fast_run	int	资源池快车道运行作业数，只在 CN 上有效
fast_wait	int	资源池快车道排队作业数，只在 CN 上有效
fast_limit	int	资源池快车道作业并发限制，只在 CN 上有效
slow_run	int	资源池慢车道运行作业数，只在 CN 上有效
slow_wait	int	资源池慢车道排队作业数，只在 CN 上有效
slow_limit	int	资源池慢车道作业并发限制，只在 CN 上有效
used_cpu	double	资源池 5s 监控周期内使用 CPU 个数平均值，保留小数点后 2 位

名称	类型	描述
		• DN：显示当前 DN 上资源池使用的 CPU 个数 • CN：显示所有 DN 上资源池使用 CPU 的累积和
cpu_limit	int	资源池可用 CPU 的上限，CPU 配额管控情况下为 GaussDB 可用 CPU，CPU 限额管控情况下为关联控制组 CPU 可用 CPU • DN：显示当前 DN 上资源池可用 CPU 上限 • CN：显示所有 DN 上资源池可用 CPU 上限的累积和
used_mem	int	资源池当前使用的内存大小，单位：'MB'’ • DN：显示当前 DN 上资源池使用的内存大小 • CN：显示所有 DN 上资源池使用内存的累积和
estimate_mem	int	当前 CN 上，资源池运行作业的估算内存之和，只在 CN 上有效
mem_limit	int	资源池可用内存上限，单位：'MB' • DN：显示当前 DN 上资源池可用内存上限 • CN：显示所有 DN 上资源池可用内存上限的累积和
read_kbytes	bigint	资源池 5s 监控周期内逻辑读字节数，单位：'KB' • DN：显示当前 DN 上资源池逻辑读字节数 • CN：显示所有 DN 上资源池逻辑读字节的累积和
write_kbytes	bigint	资源池 5s 监控周期内逻辑写字节数，单位：'KB' • DN：显示当前 DN 上资源池逻辑写字节数 • CN：显示所有 DN 上资源池逻辑写字节的累积和
read_counts	bigint	资源池 5s 监控周期内逻辑读次数 • DN：显示当前 DN 上资源池逻辑读次数 • CN：显示所有 DN 上资源池逻辑读次数的累积和
write_counts	bigint	资源池 5s 监控周期内逻辑写次数 • DN：显示当前 DN 上资源池逻辑写次数 • CN：显示所有 DN 上资源池逻辑写次数的累积和
read_speed	double	资源池 5s 监控周期内逻辑读速率的平均值

名称	类型	描述
		• DN：显示当前 DN 上资源池逻辑读速率 • CN：显示所有 DN 上资源池逻辑读速率的累积和
write_speed	double	资源池 5s 监控周期内逻辑写速率平均值 • DN：显示当前 DN 上资源池逻辑写速率 • CN：显示所有 DN 上资源池逻辑写速率的累积和

14.3.190 PGXC_RESPOOL_RESOURCE_HISTORY

PGXC_RESPOOL_RESOURCE_HISTORY 用于查询所有实例上资源池监控历史信息。

表14-232 PGXC_RESPOOL_RESOURCE_HISTORY 字段

名称	类型	描述
nodename	name	实例名称，包含 CN 和 DN
timestamp	timestamp	资源池监控信息持久化时间
nodegroup	name	资源池所属逻辑集群名称，默认集群显示 "installation"
rpname	name	资源池名称
cgroup	name	资源池关联控制组名称
ref_count	int	资源池引用作业数，作业经过资源池不管是否管控都会计数，仅 CN 上有效
fast_run	int	资源池快车道运行作业数，只在 CN 上有效
fast_wait	int	资源池快车道排队作业数，只在 CN 上有效
fast_limit	int	资源池快车道作业并发限制，只在 CN 上有效
slow_run	int	资源池慢车道运行作业数，只在 CN 上有效
slow_wait	int	资源池慢车道排队作业数，只在 CN 上有效
slow_limit	int	资源池慢车道作业并发限制，只在 CN 上有效
used_cpu	double	资源池 5s 监控周期内使用 CPU 个数平均值，保留小数点后 2 位 • DN：显示当前 DN 上资源池使用的 CPU 个数 • CN：显示所有 DN 上资源池使用 CPU 的累积和

名称	类型	描述
cpu_limit	int	资源池可用 CPU 的上限，CPU 配额管控情况下为 GaussDB 可用 CPU，CPU 限额管控情况下为关联控制组 CPU 可用 CPU • DN：显示当前 DN 上资源池可用 CPU 上限 • CN：显示所有 DN 上资源池可用 CPU 上限的累积和
used_mem	int	资源池使用的内存大小，单位：'MB'’ • DN：显示当前 DN 上资源池使用的内存大小 • CN：显示所有 DN 上资源池使用内存的累积和
estimate_mem	int	当前 CN 上，资源池运行作业的估算内存之和，只在 CN 上有效
mem_limit	int	资源池可用内存上限，单位：'MB' • DN：显示当前 DN 上资源池可用内存上限 • CN：显示所有 DN 上资源池可用内存上限的累积和
read_kbytes	bigint	资源池 5s 监控周期内逻辑读字节数，单位：'KB' • DN：显示当前 DN 上资源池逻辑读字节数 • CN：显示所有 DN 上资源池逻辑读字节的累积和
write_kbytes	bigint	资源池 5s 监控周期内逻辑写字节数，单位：'KB' • DN：显示当前 DN 上资源池逻辑写字节数 • CN：显示所有 DN 上资源池逻辑写字节的累积和
read_counts	bigint	资源池 5s 监控周期内逻辑读次数 • DN：显示当前 DN 上资源池逻辑读次数 • CN：显示所有 DN 上资源池逻辑读次数的累积和
write_counts	bigint	资源池 5s 监控周期内逻辑写次数 • DN：显示当前 DN 上资源池逻辑写次数 • CN：显示所有 DN 上资源池逻辑写次数的累积和
read_speed	double	资源池 5s 监控周期内逻辑读速率的平均值 • DN：显示当前 DN 上资源池逻辑读速率 • CN：显示所有 DN 上资源池逻辑读速率的累积和

名称	类型	描述
write_speed	double	资源池 5s 监控周期内逻辑写速率平均值 • DN：显示当前 DN 上资源池逻辑写速率 • CN：显示所有 DN 上资源池逻辑写速率的累积和

14.3.191 PGXC_ROW_TABLE_IO_STAT

PGXC_ROW_TABLE_IO_STAT 视图提供集群所有 CN 和 DN 节点上当前数据库所有行存表的 IO 统计数据。除在每一行前面增加 name 类型的 nodename 字段外，其余字段的名称、类型和顺序与 GS_ROW_TABLE_IO_STAT 视图相同，具体的字段请参考 14.3.61 GS_ROW_TABLE_IO_STAT。

需要有系统管理员权限才可以访问此视图。

14.3.192 PGXC_RUNNING_XACTS

PGXC_RUNNING_XACTS 视图主要功能是显示集群中各个节点运行事务的信息，字段内容和 14.3.116 PG_RUNNING_XACTS 相同。

表14-233 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 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。
timeline	bigint	标识数据库重启次数。
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。

14.3.193 PGXC_SETTINGS

PGXC_SETTINGS 视图显示集群中各节点数据库运行时参数的相关信息，除新增 node_name（节点名称）字段外，其余字段内容和 14.3.120 PG_SETTINGS 视图相同。需要有系统管理员权限才可以访问此视图。

14.3.194 PGXC_SESSION_WLMSTAT

PGXC_SESSION_WLMSTAT 视图显示当前集群中各 CN 节点用户执行作业正在运行时的负载管理相关信息。

表14-234 PGXC_SESSION_WLMSTAT 字段

名称	类型	描述
nodename	name	节点名称
datid	oid	连接后端的数据库 OID。
datname	name	连接后端的数据库名称。
threadid	bigint	后端线程 ID。
processid	integer	后端线程的 pid。
usesysid	oid	登录后端的用户 OID。
appname	text	连接到后端的应用名。
usename	name	登录到该后端的用户名。
priority	bigint	语句所在 Cgroups 的优先级。
attribute	text	语句的属性： • Ordinary：语句发送到数据库后被解析前的默认属性。 • Simple：简单语句。 • Complicated：复杂语句。 • Internal：数据库内部语句。
block_time	bigint	语句当前为止的 pending 的时间，单位 s。
elapsed_time	bigint	语句当前为止的实际执行时间，单位 s。
total_cpu_time	bigint	语句在上一时间周期内的 DN 上 CPU 使用的总时间，单位 s。
cpu_skew_percent	integer	语句在上一时间周期内的 DN 上 CPU 使用的倾斜率。
statement_mem	integer	语句执行所需要的估算内存，预留字段。
active_points	integer	语句占用的资源池并发点数。

名称	类型	描述
dop_value	integer	从资源池中获取语句的 dop 值。
control_group	text	语句当前所使用的 Cgroups。
status	text	语句当前的状态，包括： • pending：执行前状态。 • running：执行进行状态。 • finished：执行正常结束。（当 enqueue 字段为 StoredProc 或 Transaction 时，仅代表语句中的部分作业已经执行完毕，该状态会持续到该语句完全执行完毕。） • aborted：执行异常终止。 • active：非以上四种状态外的正常状态。 • unknown：未知状态。
enqueue	text	语句当前的排队情况，包括： • Global：全局排队。 • Respool：资源池排队。 • CentralQueue：在中心协调节点(CCN)中排队。 • Transaction：语句处于一个事务块中。 • StoredProc：语句处于一个存储过程中。 • None：未在排队。 • Forced None：事务块语句或存储过程语句由于超出设定的等待时间而强制执行。
resource_pool	name	语句当前所在的资源池。
query	text	该后端的最新查询。如果 state 状态是 active，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。
isplana	bool	逻辑集群模式下，语句当前是否占用其他逻辑集群的资源执行。该值默认为 f，表示不占用其他逻辑集群的资源执行。
node_group	text	语句所属用户对应的逻辑集群。
lane	text	表示语句查询的快慢车道。 • fast：快车道。 • slow：慢车道。 • none：未管控。

14.3.195 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。

表14-235 PGXC_STAT_ACTIVITY 字段

名称	类型	描述
coorname	text	当前集群下的 CN 名称。
datid	oid	用户会话在后端连接到的数据库 OID。
datname	name	用户会话在后端连接到的数据库名称。
pid	bigint	后端线程 ID。
lwtid	integer	后端线程的轻量级线程号。
usesysid	oid	登录此后端的用户 OID。
usename	name	登录此后端的用户名。
application_name	text	连接到此后端的应用名。
client_addr	inet	连接到此后端的客户端的 IP 地址。 如果此字段是 null，则表示通过服务器机器上 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，此字段是通过 client_addr 的反向 DNS 查找得到。仅在启动 log_hostname 且使用 IP 连接时才非空。
client_port	integer	客户端用于与后端通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
backend_start	timestamp with time zone	后端进程启动时间，即客户端连接服务器的时间。
xact_start	timestamp with time zone	当前事务的启动时间，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	状态最后一次改变的时间。
waiting	boolean	如果后端当前正等待锁则为 true。
enqueue	text	语句当前排队状态。可能值是： • waiting in global queue：表示语句在全

名称	类型	描述
		局排队中。 • waiting in respool queue：表示语句在资源池排队中。 • waiting in ccn queue：表示作业在 CCN 排队中。 • 空或 no waiting queue：表示语句正在运行。

state	text	后端当前总体状态。可能值是：
		• active：后端正在执行一个查询。
		• idle：后端正在等待一个新的客户端命令。
		• idle in transaction：后端在事务中，但事务中没有语句在执行。
		• idle in transaction (aborted)：后端在事务中，但事务中有语句执行失败。
		• fastpath function call：后端正在执行一个 fast-path 函数。
		• disabled：如果后端禁用 track_activities，则报告此状态。
		说明
		只有系统管理员能查看到自己帐户所对应的会话状态。其他帐户的 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 dbadmin 的 state 信息为空：
		`SELECT datname, usename, usesysid, state,pid FROM pgxc_stat_activity;`
		`datname | usename | usesysid | state | pid`
		`----------+---------+----------+-------+-----------------`
		`gaussdb | dbadmin | 10 | | 139968752121616`
		`gaussdb | dbadmin | 10 | | 139968903116560`
		`db_tpcds | judy | 16398 | active | 139968391403280`
		`gaussdb | dbadmin | 10 | | 139968643069712`
		`gaussdb | dbadmin | 10 | | 139968680818448`
		`gaussdb | joe | 16390 | | 139968563377936`
		`(6 rows)`

名称	类型	描述
resource_pool	name	用户使用的资源池。
stmt_type	text	用户语句的语句类型。
query_id	bigint	查询语句的 ID。
query	text	此后端的最新查询。如果 state 状态是 active（活跃的），此字段显示当前正在执行的查询。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（参见 connection_info）。

应用实例

查看当前处于阻塞状态的查询语句。

```
SELECT datname,usename,state,query FROM PGXC_STAT_ACTIVITY WHERE waiting = true;
```

查看快照线程的工作状态。

```
SELECT application_name,backend_start,state_change,state,query FROM
PGXC_STAT_ACTIVITY WHERE application_name='WDRSnapshot';
```

查看正在运行的查询语句。

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

查看指定数据库 postgres 上已使用的会话连接数。其中 1 表示数据库 postgres 上已使用的会话连接数。

```
SELECT COUNT(*) FROM PGXC_STAT_ACTIVITY WHERE DATNAME='postgres';
 count
-------
     1
(1 row)
```

14.3.196 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。

表14-236 PGXC_STAT_BAD_BLOCK 字段

名字	类型	描述
nodename	text	节点名称
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
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

14.3.197 PGXC_STAT_BGWRITER

PGXC_STAT_BGWRITER 视图显示集群中各节点上后端写进程活动的统计信息，除新增 node_name（节点名称）字段外，其余字段内容和 14.3.128 PG_STAT_BGWRITER 视图相同。需要有系统管理员权限才可以访问此视图。

14.3.198 PGXC_STAT_DATABASE

视图 PGXC_STAT_DATABASE 显示集群中各节点上数据库的状态和统计信息，除新增 node_name（节点名称）字段外，其余字段内容和 14.3.129 PG_STAT_DATABASE 视图相同。需要有系统管理员权限才可以访问此视图。

14.3.199 PGXC_STAT_REPLICATION

PGXC_STAT_REPLICATION 视图显示集群中各节点上日志同步的状态信息，除新增 node_name（节点名称）字段外，其余字段内容和 14.3.135 PG_STAT_REPLICATION 视图相同。需要有系统管理员权限才可以访问此视图

14.3.200 PGXC_SQL_COUNT

通过 PGXC_SQL_COUNT 视图，可以实时显示集群中各 CN 节点上 SELECT、INSERT、UPDATE、DELETE、MERGE INTO 五种 SQL、以及 DDL、DML、DCL 语句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。通过对以上几类 SQL 查询进行计数和响应时间统计，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，监测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。

PGXC_SQL_COUNT 视图的字段与 GS_SQL_COUNT 一致，详见表 14-115。

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。

14.3.201 PGXC_TABLE_CHANGE_STAT

PGXC_TABLE_CHANGE_STAT 视图提供集群所有 CN 节点上当前数据库所有表的变更情况。除在每一行前面增加 name 类型的 nodename 字段外，其余字段的名称、类型和顺序与 GS_TABLE_CHANGE_STAT 视图相同，具体的字段请参考 14.3.67 GS_TABLE_CHANGE_STAT。

需要有系统管理员权限才可以访问此视图。

14.3.202 PGXC_TABLE_STAT

PGXC_TABLE_STAT 视图提供集群所有 CN 和 DN 节点上当前数据库所有表的统计信息。除在每一行前面增加 name 类型的 nodename 字段外，其余字段的名称、类型和顺序与 GS_TABLE_STAT 视图相同，具体的字段请参考 14.3.68 GS_TABLE_STAT。

需要有系统管理员权限才可以访问此视图。

14.3.203 PGXC_THREAD_WAIT_STATUS

通过 CN 节点查看 PGXC_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14-237 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。

名称	类型	描述
lwtid	integer	当前线程的轻量级线程号。
ptid	integer	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。等待状态的详细信息请参见表 14-199。
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、IO 的信息。否则是空。

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。

```
 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 | gaussdb | gsql        | 20971544 | 140274089064208 | 22579 |      |
0 |     0 | wait node: datanode4 |
(1 rows)
```

此外，可以查看该语句在全局范围内各个节点上的工作情况。如下所示，每个 DN 上都没有在等待的阻塞资源，因为读取的数据太多而执行较慢。

```
select * from pgxc_thread_wait_status where query_id=20971544;
  node_name   | db_name  | thread_name  | query_id |      tid      | lwtid | ptid  |
tlevel | smpid |     wait_status    |  wait_event
--------------+----------+--------------+----------+-----------------+-------+-----
--+--------+-------+----------------------
 datanode1    | gaussdb | coordinator1 | 20971544 | 139902867994384 | 22735 |      |
0 |     0 | wait node: datanode3 |
 datanode1    | gaussdb | coordinator1 | 20971544 | 139902838634256 | 22970 | 22735
|     5 |     0 | synchronize quit     |
 datanode1    | gaussdb | coordinator1 | 20971544 | 139902607947536 | 22972 | 22735
|     5 |     1 | synchronize quit     |
 datanode2    | gaussdb | coordinator1 | 20971544 | 140632156796688 | 22736 |      |
0 |     0 | wait node: datanode3 |
 datanode2    | gaussdb | coordinator1 | 20971544 | 140632030967568 | 22974 | 22736
|     5 |     0 | synchronize quit     |
 datanode2    | gaussdb | coordinator1 | 20971544 | 140632081299216 | 22975 | 22736
|     5 |     1 | synchronize quit     |
 datanode3    | gaussdb | coordinator1 | 20971544 | 140323627988752 | 22737 |      |
0 |     0 | wait node: datanode3 |
 datanode3    | gaussdb | coordinator1 | 20971544 | 140323523131152 | 22976 | 22737
```

```
|     5 |     0 | net flush data      |
 datanode3    | gaussdb | coordinator1 | 20971544 | 140323548296976 | 22978 | 22737
|     5 |     1 | net flush data
 datanode4    | gaussdb | coordinator1 | 20971544 | 140103024375568 | 22738 |      |
0 |     0 | wait node: datanode3
 datanode4    | gaussdb | coordinator1 | 20971544 | 140102919517968 | 22979 | 22738
|     5 |     0 | synchronize quit    |
 datanode4    | gaussdb | coordinator1 | 20971544 | 140102969849616 | 22980 | 22738
|     5 |     1 | synchronize quit    |
 coordinator1 | gaussdb | gsql        | 20971544 | 140274089064208 | 22579 |      |
0 |     0 | wait node: datanode4  |
(13 rows)
```

14.3.204 PGXC_TOTAL_MEMORY_DETAIL

PGXC_TOTAL_MEMORY_DETAIL 视图显示集群内存使用情况。需要有系统管理员权限或预置角色 gs_role_read_all_stats 权限才可以访问此视图。

表14-238 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(DWS)集群实例所占用的内存大小。 • process_used_memory：GaussDB(DWS)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_cstore_memory：列存所允许使用的最大内存。 • cstore_used_memory：列存已使用的内存大小。 • max_sctpcomm_memory：通信库所允许使用的最大内存。

名称	类型	描述
		• 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。 • gpu_max_dynamic_memory：GPU 内存最大值。 • gpu_dynamic_used_memory：当前 GPU 可用内存和当前临时 GPU 内存之和。 • gpu_dynamic_peak_memory：GPU 内存使用的最大内存。 • pooler_conn_memory：pooler 连接占用内存大小。 • pooler_freeconn_memory：pooler 空闲连接占用的内存大小。 • storage_compress_memory：列存压缩和解压缩使用的内存大小。 • udf_reserved_memory：为 UDF Worker 进程预留的内存大小。 • mmap_used_memory：mmap 使用的内存大小。
memorymbytes	integer	内存使用的大小，单位为 MB。

14.3.205 PGXC_TOTAL_SCHEMA_INFO

PGXC_TOTAL_SCHEMA_INFO 视图提供了集群所有实例上的 Schema 空间信息，便于用户获悉集群各个实例上的 Schema 空间使用情况，仅支持在 CN 节点上查询。

表14-239 PGXC_TOTAL_SCHEMA_INFO 字段

名称	类型	描述
schemaname	text	模式名称
schemaid	oid	模式 OID
databasename	text	数据库名称
databaseid	oid	数据库 OID
nodename	text	实例名称
nodegroup	text	节点组名称

名称	类型	描述
usedspace	bigint	已使用的空间大小
permspace	bigint	空间上限值

14.3.206 PGXC_TOTAL_SCHEMA_INFO_ANALYZE

PGXC_TOTAL_SCHEMA_INFO_ANALYZE 视图提供了集群整体的 Schema 空间信息，包括：集群空间总值、各实例空间平均值、倾斜率、单实例空间最大值、单实例空间最小值以及最大最小空间所在的实例名，便于用户获悉集群整体的 Schema 空间使用情况，仅支持在 CN 节点上查询。

表14-240 PGXC_TOTAL_SCHEMA_INFO_ANALYZE 字段

名称	类型	描述
schemaname	text	模式名称
databasename	text	数据库名称
nodegroup	text	节点组名称
total_value	bigint	该模式的集群空间总值
avg_value	bigint	该模式的各实例空间平均值
skew_percent	integer	倾斜率
extend_info	text	提供信息包括：单实例空间最大值、单实例空间最小值以及最大最小空间所在的实例名

14.3.207 PGXC_USER_TRANSACTION

PGXC_USER_TRANSACTION 视图提供查询所有 CN 上用户相关的事务信息。需要有系统管理员权限才可以访问此视图。该视图仅在资源实时监控功能开启，即 enable_resource_track 为 on 时有效。

表14-241 PGXC_USER_TRANSACTION 字段

名称	类型	描述
node_name	name	节点名称
usename	name	用户名称

名称	类型	描述
commit_counter	bigint	提交次数
rollback_counter	bigint	回滚次数
resp_min	bigint	最小响应时间
resp_max	bigint	最大响应时间
resp_avg	bigint	平均响应时间
resp_total	bigint	响应时间总和

14.3.208 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。

表14-242 PGXC_VARIABLE_INFO 字段

名称	类型	描述
node_name	text	节点名称
nextOid	oid	该节点下一次生成的 OID
nextXid	xid	该节点下一次生成的事务号
oldestXid	xid	该节点最早的事务号。
xidVacLimit	xid	强制 autovacuum 的临界点
oldestXidDB	oid	该节点 datafrozenxid 最小的数据库 OID
lastExtendCSNLogpage	integer	最后一次扩展 csnlog 的页面号
startExtendCSNLogpage	integer	csnlog 扩展的起始页面号
nextCommitSeqNo	integer	该节点下次生成的 csn 号
latestCompletedXid	xid	该节点提交或者回滚后节点上的最新事务号
startupMaxXid	xid	该节点关机前的最后一个事务号

14.3.209 PGXC_WAIT_EVENTS

PGXC_WAIT_EVENTS 视图显示集群中各节点各类等待状态和事件的统计信息，其字段内容和 14.3.74 GS_WAIT_EVENTS 视图相同。需要有系统管理员权限才可以访问此视图。

14.3.210 PGXC_WLM_OPERATOR_HISTORY

PGXC_WLM_OPERATOR_HISTORY 视图显示在所有 CN 上执行作业结束时的算子信息。此视图用于 Database Manager 从数据库中查询数据，数据库中的数据会被定时清理，清理周期为 3 分钟。

需要有系统管理员权限或预置角色 gs_role_read_all_stats 权限才可以访问此视图。具体的字段请参考表 14-4。

14.3.211 PGXC_WLM_OPERATOR_INFO

PGXC_WLM_OPERATOR_INFO 视图显示在所有 CN 上执行作业结束时的算子信息。此视图的数据直接从系统表 14.2.4 GS_WLM_OPERATOR_INFO 获取。

需要有系统管理员权限或预置角色 gs_role_read_all_stats 权限才可以访问此视图。具体的字段请参考表 14-4。

14.3.212 PGXC_WLM_OPERATOR_STATISTICS

PGXC_WLM_OPERATOR_STATISTICS 视图显示在所有 CN 上正在执行作业的算子信息。

需要有系统管理员权限或预置角色 gs_role_read_all_stats 权限才可以访问此视图。具体的字段请参考表 14-126。

14.3.213 PGXC_WLM_SESSION_INFO

PGXC_WLM_SESSION_INFO 视图显示在所有 CN 上执行作业结束后的负载管理记录。此视图的数据直接从系统表 14.2.5 GS_WLM_SESSION_INFO 获取。

具体的字段请参考表 14-127。

14.3.214 PGXC_WLM_SESSION_HISTORY

PGXC_WLM_SESSION_HISTORY 视图显示在所有 CN 上执行作业结束后的负载管理记录。此视图用于 Database Manager 从数据库中查询数据，数据库中的数据会被定时清理，清理周期为 3 分钟，详见 14.3.79 GS_WLM_SESSION_HISTORY 视图介绍。

具体的字段请参考表 14-127。

14.3.215 PGXC_WLM_SESSION_STATISTICS

PGXC_WLM_SESSION_STATISTICS 视图显示在所有 CN 上正在执行的作业的负载管理信息。

具体的字段请参考表 14-128。

14.3.216 PGXC_WLM_WORKLOAD_RECORDS

PGXC_WLM_WORKLOAD_RECORDS 视图显示当前用户在每个 CN 上执行作业时在 CN 上的状态信息。需要有系统管理员权限或预置角色 gs_role_read_all_stats 权限才可以访问此视图。该视图仅在动态负载功能开启，即 enable_dynamic_workload 为 on 时有效。

表14-243 PGXC_WLM_WORKLOAD_RECORDS 字段

名称	类型	描述
node_name	text	作业执行所在的 CN 的 name
thread_id	bigint	后端线程 ID
processid	integer	线程的 lwpid
timestamp	bigint	语句执行的开始时间
username	name	登录到该后端的用户名
memory	integer	语句所需的内存大小
active_points	integer	语句在资源池上消耗的资源点数
max_points	integer	资源在资源池上的最大资源数
priority	integer	作业的优先级
resource_pool	text	作业所在资源池
status	text	作业执行的状态，包括： pending： 阻塞状态 running: 执行状态 finished：结束状态 aborted：终止状态 unkown：未知状态
control_group	text	作业所使用的 Cgroups
enqueue	text	作业的排队信息，包括： GLOBAL：全局排队 RESPOOL: 资源池排队 ACTIVE：不排队
query	text	正在执行的语句

14.3.217 PGXC_WORKLOAD_SQL_COUNT

PGXC_WORKLOAD_SQL_COUNT 视图显示集群中所有 CN 节点上的 Workload 控制组内的 SQL 语句执行次数的统计信息，包括 SELECT、UPDATE、INSERT、DELETE 语句的执行次数统计，以及 DDL、DML、DCL 类型语句的执行次数统计。需要有系统管理员权限或预置角色 gs_role_read_all_stats 权限才可以访问此视图。

表14-244 PGXC_WORKLOAD_SQL_COUNT 字段

名称	类型	描述
node_name	name	节点名称
workload	name	Workload 控制组名称
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

14.3.218 PGXC_WORKLOAD_SQL_ELAPSE_TIME

PGXC_WORKLOAD_SQL_ELAPSE_TIME 视图显示集群中所有 CN 节点上 Workload 控制组内 SQL 语句执行的响应时间的统计信息，包括 SELECT、UPDATE、INSERT、DELETE 语句的最大、最小、平均、以及总响应时间，单位为微秒。需要有系统管理员权限或预置角色 gs_role_read_all_stats 权限才可以访问此视图。

表14-245 PGXC_WORKLOAD_SQL_ELAPSE_TIME 字段

名称	类型	描述
node_name	name	节点名称
workload	name	Workload 控制组名称
total_select_elapse	bigint	SELECT 总响应时间
max_select_elapse	bigint	SELECT 最大响应时间
min_select_elapse	bigint	SELECT 最小响应时间
avg_select_elapse	bigint	SELECT 平均响应时间
total_update_elapse	bigint	UPDATE 总响应时间

名称	类型	描述
max_update_elapse	bigint	UPDATE 最大响应时间
min_update_elapse	bigint	UPDATE 最小响应时间
avg_update_elapse	bigint	UPDATE 平均响应时间
total_insert_elapse	bigint	INSERT 总响应时间
max_insert_elapse	bigint	INSERT 最大响应时间
min_insert_elapse	bigint	INSERT 最小响应时间
avg_insert_elapse	bigint	INSERT 平均响应时间
total_delete_elapse	bigint	DELETE 总响应时间
max_delete_elapse	bigint	DELETE 最大响应时间
min_delete_elapse	bigint	DELETE 最小响应时间
avg_delete_elapse	bigint	DELETE 平均响应时间

14.3.219 PGXC_WORKLOAD_TRANSACTION

PGXC_WORKLOAD_TRANSACTION 视图提供查询所有 CN 上 Workload 控制组相关的事务信息。需要有系统管理员权限或预置角色 gs_role_read_all_stats 权限才可以访问此视图。该视图仅在资源实时监控功能开启，即 enable_resource_track 为 on 时有效。

表14-246 PGXC_WORKLOAD_TRANSACTION 字段

名称	类型	描述
node_name	name	节点名称
workload	name	Workload 控制组名称
commit_counter	bigint	提交次数
rollback_counter	bigint	回滚次数
resp_min	bigint	最小响应时间，单位微秒
resp_max	bigint	最大响应时间，单位微秒
resp_avg	bigint	平均响应时间，单位微秒
resp_total	bigint	响应时间总和，单位微秒

14.3.220 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14-247 PLAN_TABLE 字段

名称	类型	描述
statement_id	varchar2(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	varchar2(30)	计划中算子的操作描述。
options	varchar2(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。
object_type	varchar2(30)	对象类型。
object_owner	name	对象所属 schema，来自于用户定义。
projection	varchar2(4000)	操作输出的列信息。

说明

- object_type 取值范围为 14.2.17 PG_CLASS 中定义的 relkind 类型（TABLE 普通表，INDEX 索引，SEQUENCE 序列，VIEW 视图，FOREIGN TABLE 外表，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。

14.3.221 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14-248 PLAN_TABLE 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。
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	varchar2(30)	用户输入的查询标签。
plan_id	bigint	查询标识。
id	int	计划中的节点编号。
operation	varchar2(30)	操作描述。
options	varchar2(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	varchar2(30)	对象类型。
object_owner	name	对象所属 schema，来自于用户定义。
projection	varchar2(4000)	操作输出的列信息。

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 14.3.220 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 14.3.220 PLAN_TABLE 视图。
- 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。

14.3.222 PV_FILE_STAT

PV_FILE_STAT 视图通过对数据文件 IO 的统计，反映数据的 IO 性能，用以发现 IO 操作异常等性能问题。

表14-249 PV_FILE_STAT 字段

名称	类型	描述

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
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

14.3.223 PV_INSTANCE_TIME

PV_INSTANCE_TIME 视图用于统计进程的运行时间信息及各执行阶段所消耗时间，单位为微秒。

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 ：plpgsql（存储过程）的执行时间.
- PL_COMPILATION_TIME：plpgsql（存储过程）编译时间。
- NET_SEND_TIME：网络上的时间花销。
- DATA_IO_TIME：IO 时间上的花销。

表14-250 PV_INSTANCE_TIME 字段

名称	类型	描述

名称	类型	描述
stat_id	integer	类型编号
stat_name	text	运行时间类型名称
value	bigint	运行时间值

14.3.224 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14-251 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
cumulative	boolean	操作系统运行状态的值是否为累加值

14.3.225 PV_SESSION_MEMORY

PV_SESSION_MEMORY 视图统计 Session 级别的内存使用情况，包含执行作业在数据节点上 Postgres 线程和 Stream 线程分配的所有内存。

表14-252 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。

14.3.226 PV_SESSION_MEMORY_DETAIL

PV_SESSION_MEMORY_DETAIL 统计线程的内存使用情况，以 MemoryContext 节点来统计。

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“/tmp/dumpmem”目录下的“threadid_timestamp.log”文件中。其中 threadid 可通过下表 sessid 中获得。

表14-253 PV_SESSION_MEMORY_DETAIL 字段

名称	类型	描述
sessid	text	线程启动时间+线程标识（字符串信息为 timestamp.threadid）。
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。

应用示例

查询当前节点上所有 MemoryContext 的使用情况。

根据 sessid 定位到该 MemoryContext 是在哪个线程中创建和使用的，依据 totalsize，freesize 及 usedsize 来确认内存的使用情况是否符合预期，预先判断是否可能存在内存泄露的风险。

```
SELECT * FROM PV_SESSION_MEMORY_DETAIL order by totalsize desc;
         sessid         |       sesstype       |             contextname
| level |          parent          | totalsize | freesize | usedsize
-----------------------------+-------------------------+----------------------------
-----------------+-------+------------------------------+-----------+----------+---
-------
 0.139975915622720        | postmaster           | gs_signal
|     1 | TopMemoryContext           |  17209904 |  8081136 |  9128768
 1667462258.139973631031040 | postgres             | SRF multi-call context
|     5 | FunctionScan_139973631031040 |   1725504 |    3168 |  1722336
```

```
 1667461280.139973666686720 | postgres             | CacheMemoryContext
|     1 | TopMemoryContext          |   1472544 |   284456 |  1188088
 1667450443.139973877479168 | postgres             | CacheMemoryContext
|     1 | TopMemoryContext          |   1472544 |   356088 |  1116456
 1667462258.139973631031040 | postgres             | CacheMemoryContext
|     1 | TopMemoryContext          |   1472544 |   128216 |  1344328
 1667461250.139973915236096 | postgres             | CacheMemoryContext
|     1 | TopMemoryContext          |   1472544 |   226352 |  1246192
 1667450439.139974010144512 | WLMarbiter           | CacheMemoryContext
|     1 | TopMemoryContext          |   1472544 |   386736 |  1085808
 1667450439.139974151726848 | WDRSnapshot          | CacheMemoryContext
|     1 | TopMemoryContext          |   1472544 |   159720 |  1312824
 1667450439.139974026925824 | WLMmonitor           | CacheMemoryContext
|     1 | TopMemoryContext          |   1472544 |   297976 |  1174568
 1667451036.139973746386688 | postgres             | CacheMemoryContext
|     1 | TopMemoryContext          |   1472544 |   208064 |  1264480
 1667461250.139973950891776 | postgres             | CacheMemoryContext
|     1 | TopMemoryContext          |   1472544 |   270016 |  1202528
 1667450439.139974076212992 | WLMCalSpaceInfo      | CacheMemoryContext
|     1 | TopMemoryContext          |   1472544 |   393952 |  1078592
 1667450439.139974092994304 | WLMCollectWorker     | CacheMemoryContext
|     1 | TopMemoryContext          |   1472544 |    94848 |  1377696
 1667461254.139973971343104 | postgres             | CacheMemoryContext
|     1 | TopMemoryContext          |   1472544 |   338544 |  1134000
 1667461280.139973822945024 | postgres             | CacheMemoryContext
|     1 | TopMemoryContext          |   1472544 |   284456 |  1188088
 1667450439.139974202070784 | JobScheduler         | CacheMemoryContext
|     1 | TopMemoryContext          |   1472544 |   216728 |  1255816
 1667450454.139973860697856 | postgres             | CacheMemoryContext
|     1 | TopMemoryContext          |   1472544 |   388384 |  1084160
 0.139975915622720        | postmaster           | Postmaster
|     1 | TopMemoryContext          |   1004288 |    88792 |   915496
 1667450439.139974218852096 | AutoVacLauncher      | CacheMemoryContext
|     1 | TopMemoryContext          |    948256 |   183488 |   764768
 1667461250.139973915236096 | postgres             | TempSmallContextGroup
|     0 |                           |    584448 |   148032 |      119
 1667462258.139973631031040 | postgres             | TempSmallContextGroup
|     0 |                           |    579712 |   162128 |      123
```

14.3.227 PV_SESSION_STAT

PV_SESSION_STAT 视图以会话线程或 AutoVacuum 线程为单位，统计会话状态信息。

表14-254 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

名称	类型	描述
value	bigint	统计会话值

14.3.228 PV_SESSION_TIME

PV_SESSION_TIME 视图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，单位为微秒。

表14-255 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	运行时间类型名称
value	bigint	运行时间值

14.3.229 PV_TOTAL_MEMORY_DETAIL

PV_TOTAL_MEMORY_DETAIL 视图统计当前数据库节点使用内存的信息，单位为MB。

表14-256 PV_TOTAL_MEMORY_DETAIL 字段

名称	类型	描述
nodename	text	节点名称。
memorytype	text	内存类型，包括以下几种： • max_process_memory：GaussDB(DWS)集群实例所占用的内存大小。 • process_used_memory：GaussDB(DWS)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。

名称	类型	描述
		• max_cstore_memory：列存所允许使用的最大内存。 • cstore_used_memory：列存已使用的内存大小。 • 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。 • gpu_max_dynamic_memory：GPU 内存最大值。 • gpu_dynamic_used_memory：当前 GPU 可用内存和当前临时 GPU 内存之和。 • gpu_dynamic_peak_memory：GPU 内存使用的最大内存。 • pooler_conn_memory：pooler 连接占用内存大小。 • pooler_freeconn_memory：pooler 空闲连接占用的内存大小。 • storage_compress_memory：列存压缩和解压缩使用的内存大小。 • udf_reserved_memory：为 UDF Worker 进程预留的内存大小。 • mmap_used_memory：mmap 使用的内存大小。
memorymbytes	integer	内存类型分配内存的大小。

14.3.230 PV_REDO_STAT

PV_REDO_STAT 视图提供当前节点上 XLOG 重做过程中的统计信息。

表14-257 PV_REDO_STAT 字段

名称	类型	描述
phywrts	bigint	物理写次数
phyblkwrt	bigint	物理写块数
writetim	bigint	物理写消耗时间
avgiotim	bigint	平均每次写入时间
lstiotim	bigint	上一次写入时间
miniotim	bigint	最小写入时间

名称	类型	描述
maxiowtm	bigint	最大写入时间

14.3.231 REDACTION_COLUMNS

REDACTION_COLUMNS 视图展示当前数据库内所有脱敏列信息。

表14-258 REDACTION_COLUMNS 字段

名称	类型	描述
object_owner	name	脱敏对象 owner
object_name	name	脱敏对象名称
column_name	name	脱敏列名称
function_type	integer	脱敏类型
function_parameters	text	脱敏类型为 partial 类型时的参数（保留字段，无实际意义）
regexp_pattern	text	脱敏类型为 regexp 时，pattern 串（保留字段，无实际意义）
regexp_replace_string	text	脱敏类型为 regexp 时，替换串（保留字段，无实际意义）
regexp_position	integer	脱敏类型为 regexp 时，起始替换位置（保留字段，无实际意义）
regexp_occurrence	integer	脱敏类型为 regexp 时，替换次数（保留字段，无实际意义）
regexp_match_parameter	text	脱敏类型为 regexp 时，正则控制参数（保留字段，无实际意义）
function_info	text	脱敏函数信息
column_description	text	脱敏列描述信息

14.3.232 REDACTION_POLICIES

REDACTION_POLICIES 视图展示当前数据库内所有脱敏对象信息。

表14-259 REDACTION_POLICIES 字段

名称	类型	描述

名称	类型	描述
object_owner	name	脱敏对象 owner
object_name	name	脱敏对象名称
policy_name	name	脱敏策略名称
expression	text	策略生效表达式（针对用户）
enable	boolean	策略状态（开启、关闭）
policy_description	text	策略描述信息

14.3.233 REMOTE_TABLE_STAT

REMOTE_TABLE_STAT 视图提供集群所有 DN 节点上当前数据库所有表的统计信息。除在每一行前面增加 name 类型的 nodename 字段外，其余字段的名称、类型和顺序与 GS_TABLE_STAT 视图相同，具体的字段请参考 14.3.68 GS_TABLE_STAT。

需要有系统管理员权限才可以访问此视图。

14.3.234 USER_COL_COMMENTS

USER_COL_COMMENTS 视图存储当前用户下表和视图的列注释信息。

名称	类型	描述
column_name	character varying(64)	列名
table_name	character varying(64)	表名或视图名
owner	character varying(64)	表或视图的所有者
comments	text	注释

14.3.235 USER_CONSTRAINTS

USER_CONSTRAINTS 视图存储当前用户下表中约束的信息。

名称	类型	描述
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

名称	类型	描述
		• 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）

14.3.236 USER_CONS_COLUMNS

USER_CONS_COLUMNS 视图存储当前用户下表中约束列的信息。

名称	类型	描述
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

14.3.237 USER_INDEXES

USER_INDEXES 视图存储关于本模式下的索引信息。

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	表示索引是否为唯一索引
generated	character varying(1)	表示索引名称是否为系统生成
partitioned	character(3)	表示索引是否具有分区表的性质

14.3.238 USER_IND_COLUMNS

USER_IND_COLUMNS 视图存储当前用户下所有索引的字段信息。

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

14.3.239 USER_IND_EXPRESSIONS

USER_IND_EXPRESSIONS 视图存储当前用户下基于函数的表达式索引的信息。

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

14.3.240 USER_IND_PARTITIONS

USER_IND_PARTITIONS 视图存储当前用户下的索引分区信息。

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
schema	character varying(64)	索引分区所属分区表索引的模式

名称	类型	描述
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
index_partition_usable	boolean	索引分区是否可用
high_value	text	索引分区所对应的表分区的边界(范围分区为上边界，列表分区为边界值集合)。 前向兼容的保留字段，8.1.3 集群版本新增 pretty_high_value 用于记录此信息。
pretty_high_value	text	索引分区所对应的表分区的边界(范围分区为上边界，列表分区为边界值集合)。 查询结果为表分区对应边界表达式的即时反编译输出。该字段的输出比 high_value 的信息更详细，根据实际使用场景可输出 collaton、字段数据类型等信息。
def_tablespace_name	name	索引分区的表空间名称

14.3.241 USER_JOBS

USER_JOBS 视图为当前用户所属定时任务的详细信息。

表14-260 USER_JOBS 字段

名字	类型	描述
job	int4	作业 ID。
log_user	name not null	创建者的 UserName。
priv_user	name not null	作业执行者的 UserName。
dbname	name not null	作业创建数据库名字。
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。

名字	类型	描述
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。
broken	text	任务状态 如果为 Y，不尝试运行此任务。 如果为 N，将尝试执行此任务。
status	char	当前任务的执行状态，取值范围：('r', 's', 'f', 'd')，默认为'r'，取值含义： • r=running • s=successfully finished • f= job failed • d=aborted
interval	text	用来计算下次运行时间的时间表达式，如果为 nul，l 则表示定时任务只执行一次。
failures	smallint	失败计数，作业连续执行失败 16 次，不再继续执行。
what	text	可执行的作业。

14.3.242 USER_OBJECTS

USER_OBJECTS 视图描述了当前用户拥有的数据库对象。

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

名称	类型	描述
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 14.2.39 PG_OBJECT 中的记录范围。

14.3.243 USER_PART_INDEXES

USER_PART_INDEXES 视图存储当前用户下分区表索引的信息。

名称	类型	描述
index_owner	character varying(64)	分区表索引的所有者名称
schema	character varying(64)	分区表索引的模式
index_name	character varying(64)	分区表索引的名称
table_name	character varying(64)	分区表索引所属的分区表名称
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）和列表分区（List Partitioning）。
partition_count	bigint	分区表索引的索引分区的个数
def_tablespace_name	name	分区表索引的表空间名称
partitioning_key_count	integer	分区表的分区键个数

14.3.244 USER_PART_TABLES

USER_PART_TABLES 视图存储当前用户下分区表的信息。

名称	类型	描述
table_owner	character varying(64)	分区表的所有者名称
schema	character varying(64)	分区表的模式
table_name	character varying(64)	分区表的名称

名称	类型	描述
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）和列表分区（List Partitioning）。
partition_count	bigint	分区表的分区个数
def_tablespace_name	name	分区表的表空间名称
partitioning_key_count	integer	分区表的分区键个数

14.3.245 USER_PROCEDURES

USER_PROCEDURES 视图存储关于本模式下的存储过程或函数信息。

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

14.3.246 USER_SEQUENCES

USER_SEQUENCES 视图存储关于本模式下的序列信息。

名称	类型	描述
sequence_owner	character varying(64)	序列的所有者
sequence_name	character varying(64)	序列的名称

14.3.247 USER_SOURCE

USER_SOURCE 视图存储关于本模式下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。

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)	存储过程或函数的名称

名称	类型	描述
text	text	存储过程或函数的定义

14.3.248 USER_SYNONYMS

USER_SYNONYMS 视图存储当前用户可访问的同义词信息。

表14-261 USER_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
table_schema_name	text	关联对象所属模式名
table_name	text	关联对象名

14.3.249 USER_TAB_COLUMNS

USER_TAB_COLUMNS 视图存储当前用户可访问的表和视图字段信息。

名称	类型	描述
owner	character varying(64)	表或视图的所有者
table_name	character varying(64)	表名或视图名
column_name	character varying(64)	列名
data_type	character varying(128)	列的数据类型
column_id	integer	创建表或视图时列的序号
data_length	integer	列的字节长度
comments	text	注释

名称	类型	描述
avg_col_len	numeric	列的平均长度（单位字节）
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
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
char_length	numeric	列的长度（以字符计），只对 varchar，nvarchar2， bpchar，char 类型有效
schema	character varying(64)	包含该表或视图的命名空间。
kind	text	当前记录所属的种类，如果此列属于表，则此字段显示为 table；如果此列属于视图，则此字段显示为 view。

14.3.250 USER_TAB_COMMENTS

USER_TAB_COMMENTS 视图存储当前用户所有表和视图的注释信息。

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

14.3.251 USER_TAB_PARTITIONS

USER_TAB_PARTITIONS 视图存储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在 USER_TAB_PARTITIONS 中都会有一条记录。

名称	类型	描述
table_owner	character varying(64)	分区所在表的所有者
schema	character varying(64)	分区表模式
table_name	character varying(64)	表名

名称	类型	描述
partition_name	character varying(64)	分区的名称
high_value	text	范围分区的上边界，或列表分区的边界值集合。 前向兼容的保留字段，8.1.3 集群版本新增 pretty_high_value 用于记录此信息。
pretty_high_value	text	范围分区的上边界，或列表分区的边界值集合。 查询结果为表分区对应边界表达式的即时反编译输出。该字段的输出比 high_value 的信息更详细，根据实际使用场景可输出 collaton、字段数据类型等信息。
tablespace_name	name	分区所在表空间的名称

14.3.252 USER_TABLES

USER_TABLES 视图存储关于当前模式下的表信息。

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)	表所在的表空间的名称
status	character varying(8)	当前记录是否有效
temporary	character(1)	是否为临时表 • Y 表示是临时表 • N 表示不是临时表
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

14.3.253 USER_TRIGGERS

USER_TRIGGERS 视图存储关于当前用户下的触发器信息。

名称	类型	描述
trigger_name	character varying(64)	触发器名称

名称	类型	描述
table_name	character varying(64)	定义触发器的表的名称
table_owner	character varying(64)	定义触发器的表的所有者

14.3.254 USER_VIEWS

USER_VIEWS 视图存储关于当前模式下的所有视图信息。

名称	类型	描述
owner	character varying(64)	视图的所有者
view_name	character varying(64)	视图的名称

14.3.255 V$SESSION

V$SESSION 视图存储当前会话的所有会话信息。

表14-262 V$SESSION 字段

名称	类型	描述
sid	bigint	当前活动的后台进程的 OID。
serial#	integer	当前活动的后台进程的序号，在 GaussDB(DWS)中为0。
user#	oid	登录此后台进程的用户的 OID。
username	name	登录此后台进程的用户名。

14.3.256 V$SESSION_LONGOPS

V$SESSION_LONGOPS 视图存储当前正在执行的操作的进度。

表14-263 V$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(DWS)中为0。
sofar	integer	目前完成的工作量，在 GaussDB(DWS)中为空。

名称	类型	描述
totalwork	integer	工作总量，在 GaussDB(DWS)中为空。

15 排序规则支持

排序规则(collation)是在字符集中指定数据排序顺序及对数据进行分类的规则。排序规则支持不再受限于数据库的 LC_COLLATE 和 LC_CTYPE 设置创建后就不能更改的约束。

概述

一种可排序数据类型的每一种表达式都有一个排序规则（系统内部的可排序数据类型可以是 text、varchar 和 char 等字符类型。用户定义的基础类型也可以被标记为可排序的，并且在一种可排序数据类型上的域也是可排序的）。如果该表达式是一个列引用，该表达式的排序规则就是列所定义的排序规则。如果该表达式是一个常量，排序规则就是该常量数据类型的默认排序规则。更复杂表达式的排序规则根据其输入的排序规则得来。

排序规则组合原则

- 当表达式的 collation 未指定时，则认为是默认的排序规则 default，它表示数据库的区域设置。表达式的 collation 也可能是不确定的，此时，排序操作和其他不确定的排序规则的操作就会失败。
- 对于函数或操作符调用，其排序规则将通过检查所有参数的 collation 来决定。如果该函数或操作符调用的结果是一种可排序的数据类型，若有外层表达式要用到排序规则，那么该外层的表达式将继承对应函数和操作符所调用结果集的排序规则。
- 表达式的排序规则派生可以是显式或隐式。该区别会影响多个不同的排序规则出现在同一个表达式中时如何对 collation 进行组合。当执行语句使用 COLLATE 子句时，将发生显式派生，否则为隐式派生。当多个排序规则组合时，规则如下：
  - 如果输入表达式中存在显式 COLLATE 派生，则在输入表达式之间的所有显式派生的 COLLATE 必须相同，否则将产生冲突错误。如果存在显式 COLLATE，那它就是排序规则组合的结果。
  - 如果不存在显式 COLLATE，那所有输入表达式必须具有相同的隐式 COLLATE 或默认 COLLATE。如果存在非默认 COLLATE，那它就是排序规则组合的结果。否则，结果是默认 COLLATE。
  - 如果在输入表达式之间存在多个冲突的非默认 COLLATE，则组合被认为是具有不确定排序规则，这并非一种错误。如果被调用的函数或表达式需要用到排序规则，运行时将产生排序规则未知的错误。

- CASE 表达式中，比较行为使用的规则以 WHEN 子句中的 COLLATE 设置为准。
- 显示 COLLATE 的派生仅在当前查询（CTE 或 SUBQUERY）中生效，查询外则降为隐式派生。

排序规则使用建议

- 同一条查询语句中，避免使用多种排序规则，可能导致非预期的结果集。
- 使用 collate 子句指定排序规则时，避免连续使用多个 collate 子句变更排序规则。

大小写不敏感排序规则支持

从集群 8.1.3 版本开始，GaussDB(DWS)增加内置排序规则 case_insensitive，即对字符类型的大小写不敏感行为（如排序、比较、哈希）。

约束条件：

- 支持字符类型：char/character/nchar、varchar/character varying/varchar2/nvarchar2/clob/text。
- 不支持字符类型：“char”和 name。
- 不支持的编码：PG_EUC_JIS_2004、PG_MULE_INTERNAL、PG_LATIN10、PG_WIN874。
- 不支持 CREATE DATABASE 时指定到 LC_COLLATE。
- 不支持正则表达式。
- 不支持字符类型的 record 比较（如 record_eq）。
- 不支持时序表。
- 不支持倾斜优化。
- 不支持 RoughCheck 优化。

示例

```
--语句中显示指定 COLLATE 子句。
SELECT 'a' = 'A', 'a' = 'A' COLLATE case_insensitive;
 ?column? | ?column?
----------+----------
 f        | t
(1 row)
--建表时指定列属性为 case_insensitive。
CREATE TABLE t1 (a text collate case_insensitive);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using round-robin as the
distribution mode by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution
column.
CREATE TABLE
\d t1
          Table "public.t1"
 Column | Type |       Modifiers
--------+------+-------------------------
 a      | text | collate case insensitive

INSERT INTO t1 values('a'),('A'),('b'),('B');
```

```
INSERT 0 4
--建表时指定，查询时无需指定。
SELECT a, a='a' FROM t1;
 a | ?column?
---+----------
 A | t
 B | f
 a | t
 b | f
(4 rows)
SELECT a, count(1) FROM t1 GROUP BY a;
 a | count
---+-------
 a |     2
 B |     2
(2 rows)
--CASE 表达式，以 WHEN 子句中的 COLLATE 设置为准。
SELECT a,case a when 'a' collate case_insensitive then 'case1' when 'b' collate "C"
then 'case2' else 'case3' end from t1;
 a | case
---+-------
 A | case1
 B | case3
 a | case1
 b | case2
(4 rows)
--跨子查询隐式派生。
SELECT * from (SELECT a collate "C" from t1) where a in ('a','b');
 a
---
 a
 b
(2 rows)
SELECT * from t1,(SELECT a collate "C" from t1) t2 where t1.a=t2.a;
ERROR:  could not determine which collation to use for string hashing
HINT:  Use the COLLATE clause to set the collation explicitly.
```

⚠ 注意

- 由于 collate case_insensitive 为不敏感排序，结果集不确定，再使用敏感排序筛选，会有结果集不稳定的问题，因此语句中避免出现敏感排序和不敏感排序混用。
- 使用 collate case_insensitive 指定字符类型行为为大小写不敏感后，性能较使用前会有所下降，因此性能敏感场景需谨慎评估后使用。

16 GUC 参数

16.1 查看 GUC 参数

GaussDB(DWS)的 GUC 参数影响数据库的系统行为，用户可根据业务场景和数据量查看并调整 GUC 参数取值。

- 查看 GUC 参数方式一：集群创建成功后，用户可在 GaussDB(DWS) 管理控制台上查看常用的数据库参数。

- 查看 GUC 参数方式二：成功连接集群后，通过 SQL 命令的方式查看数据库 GUC 参数。
  - 使用 SHOW 命令。

    使用如下命令查看单个参数：

    ```
    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：

    ```
    SHOW ALL;
    ```

  - 使用 pg_settings 视图。

    使用如下命令查看单个参数：

    ```
    SELECT * FROM pg_settings WHERE NAME='server_version';
    ```

    使用如下命令查看所有参数：

    ```
    SELECT * FROM pg_settings;
    ```

16.2 设置 GUC 参数

为确保 GaussDB(DWS)的最优性能，用户可根据业务需求对数据库中的 GUC 参数进行调整。

参数类型和值

- GaussDB(DWS)的 GUC 参数类型分为以下五类：
  - SUSET，数据库管理员参数。设置后立即生效，无需重启集群。若在当前会话中设置该类型参数仅当前会话生效。
  - USERSET，普通用户参数。设置后立即生效，无需重启集群。若在当前会话中设置该类型参数仅当前会话生效。
  - POSTMASTER，数据库服务端参数。设置后需要重启集群才能生效，确认修改后系统会提示集群状态为待重启，建议在非业务高峰期手动重启集群，使参数生效。
  - SIGHUP，数据库全局参数。设置后全局生效，无法会话级生效。
  - BACKEND，数据库全局参数。设置后全局生效，无法会话级生效。
- 所有的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。
  - 枚举类型的取值由系统表 pg_settings 的 enumvals 字段取值所定义。
- 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。
  - 参数的默认单位由系统表 pg_settings 的 unit 字段所定义。
  - 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）。

GUC 参数设置

GUC 参数设置有两种方式：

- 方式一：集群创建成功后，用户可以登录 GaussDB(DWS) 管理控制台，根据实际需要修改集群的数据库参数。 具体操作请参见《数据仓库服务用户指南》中的“修改数据库参数”章节。
- 方式二：成功连接集群后，通过 SQL 命令的方式设置 SUSET 或 USERSET 类型的参数。

  修改指定数据库，用户，会话级别的参数。
  - 设置数据库级别的参数

    ```
    ALTER DATABASE dbname SET paraname TO value;
    ```

    在下次会话中生效。
  - 设置用户级别的参数

    ```
    ALTER USER username SET paraname TO value;
    ```

    在下次会话中生效。

- 设置会话级别的参数

  ```
  SET paraname TO value;
  ```

  修改本次会话中的取值。退出会话后，设置将失效。

操作步骤

设置参数，以设置 explain_perf_mode 参数为例。

步骤 1 查看 explain_perf_mode 参数。

```
SHOW explain perf mode;
 explain perf mode
-------------------
 normal
(1 row)
```

步骤 2 设置 explain_perf_mode 参数。

使用以下任意方式进行设置：

- 设置数据库级别的参数

  ```
  ALTER DATABASE gaussdb SET explain_perf_mode TO pretty;
  ```

  当结果显示为如下信息，则表示设置成功。

  ```
  ALTER DATABASE
  ```

  在下次会话中生效。

- 设置用户级别的参数

  ```
  ALTER USER dbadmin SET explain_perf_mode TO pretty;
  ```

  当结果显示为如下信息，则表示设置成功。

  ```
  ALTER USER
  ```

  在下次会话中生效。

- 设置会话级别的参数

  ```
  SET explain_perf_mode TO pretty;
  ```

  当结果显示为如下信息，则表示设置成功。

  ```
  SET
  ```

步骤 3 检查参数设置的正确性。

```
SHOW explain_perf_mode;
 explain_perf_mode
--------------
 pretty
(1 row)
```

----结束

16.3 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。
- 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。

16.4 连接和认证

16.4.1 连接设置

介绍设置客户端和服务器连接方式相关的参数。

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。

参数类型：POSTMASTER

取值范围：整型。CN 最小值为 1，最大值为 16384；DN 最小值为 1， 最大值为 262143，由于集群内部存在着各种连接，设置时通常达不到最大值，若日志中出现 'invalid value for parameter "max_connections"'，需要调小 DN 的 max_connections 值。

默认值：CN 节点为 800，DN 节点为 5000，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：

CN 中此参数建议保持默认值。DN 中此参数按照如下公式计算：

dop_limit * 20 * 6 + 24，公式中的 dop_limit 为集群中每个 DN 对应的 CPU 数，计算公式为：dop_limit = 单机器的 CPU 逻辑核数 / 单机器的 DN 数。

最小值 5000。

增大这个参数可能导致 GaussDB(DWS)要求更多的 SystemV 共享内存或者信号量，可能超过操作系统缺省配置的最大值。这种情况下，请酌情对数值加以调整。

须知

max_connections 取值的设置受 max_prepared_transactions 的影响，在设置 max_connections 之前，应确保 max_prepared_transactions 的值大于或等于 max_connections 的值，这样可确保每个会话都有一个等待中的预备事务。

sysadmin_reserved_connections

参数说明：为管理员用户预留的最少连接数。

参数类型：POSTMASTER

取值范围：整型，0~262143

默认值：3

application_name

参数说明：连接数据库的客户端程序名称。

参数类型：USERSET

取值范围：字符串。

默认值：gsql

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。（运维类参数，不建议用户设置）

参数类型：USERSET

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 8.1.3 build 39137c2d) compiled at 2022-04-01 15:43:11 commit 3629 last mr 5138 debug","driver_path":"/usr/local/lib/psqlodbcw.so","os_user":"dbadmin"}
```

ODBC，JDBC，gsql 连接默认显示 driver_name 和 driver_version，driver_path，os_user，其他接口连接默认显示 driver_name 和 driver_version，driver_path 和 os_user 的显示由用户控制。

16.4.2 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。

参数类型：SIGHUP

取值范围：整型，1~600，最小单位为秒（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。

参数类型：SIGHUP

取值范围：整型，2048-134217728

默认值：10000

须知

迭代次数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。

session_timeout

参数说明：表明与服务器建立链接后，不进行任何操作的最长时间。

参数类型：USERSET

取值范围：整型，0-86400，最小单位为秒（s），0 表示关闭超时设置。

默认值：10min

须知

- GaussDB(DWS) 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。
- pooler 连接池到其它 CN 和 DN 的连接，不受 session_timeout 参数控制。

ssl

参数说明：启用 SSL 连接。

参数类型：POSTMASTER

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(DWS)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：on

ssl_ciphers

参数说明：指定 SSL 支持的加密算法列表。

参数类型：POSTMASTER

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割。

默认值：ALL

说明

- 配置参数 ssl_ciphers 的默认值为 ALL，表示支持下列所有加密算法。如果对加密算法没有特殊要求，建议用户使用该默认值。
- TLS1_3_RFC_AES_128_GCM_SHA256
- TLS1_3_RFC_AES_256_GCM_SHA384
- TLS1_3_RFC_CHACHA20_POLY1305_SHA256
- TLS1_3_RFC_AES_128_CCM_SHA256
- TLS1_3_RFC_AES_128_CCM_8_SHA256
- SSL 连接认证目前只支持配置标准协议 TLS1.3 的加密算法，TLS1.3 的性能更优、安全性更好。同时也兼容与标准协议 TLS1.2 的客户端之间的 SSL 连接认证。

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。

参数类型：USERSET

说明

参数建议保持默认设置，即禁用重协商机制。不建议通过 gs_guc 工具或其他方式直接在 postgresql.conf 文件中设置 ssl_renegotiation_limit 参数，即使设置也不会生效。

取值范围：整型，0~INT_MAX，单位为 KB。其中 0 表示禁用重新协商机制。

默认值：0

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(DWS)帐户时，该参数决定是否进行密码复杂度检查。

参数类型：SIGHUP

须知

从安全性考虑，请勿关闭密码复杂度策略。

取值范围：整型，0、1

- 0 表示不采用任何密码复杂度策略。
- 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。

参数类型：SIGHUP

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，0~3650，单位为天。

- 0 表示不检查密码可重用天数。
- 正数表示新密码不能为该值指定的天数内使用过的密码。

默认值：60

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。

参数类型：SIGHUP

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，0~1000

- 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：该参数指定帐户被锁定后自动解锁的时间。

参数类型：SIGHUP

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，0~365，单位为天。

- 0 表示密码验证失败时，自动锁定功能不生效。
- 正数表示帐户被锁定后，当锁定时间超过 password_lock_time 设定的值时，帐户将会被自行解锁。

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 则当前帐户被锁定，password_lock_time 秒后被自动解锁。 例如，登录时输入密码失败，ALTER USER 时修改密码失败等。

参数类型：SIGHUP

取值范围：整型，0~1000

- 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，当前帐户将被锁定。

默认值：10

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。
- failed_login_attempts 会与客户端 SSL 连接模式共同决定用户的密码错误次数。当 PGSSLMODE 取值是 allow 或 prefer 时，客户的一次密码连接请求会生成两次连接请求：一次是尝试 SSL 连接，另一次是尝试非 SSL 连接。此时，用户感知到的密码错误次数是 failed_login_attempts 除以 2。

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。

参数类型：SIGHUP

取值范围：整型，0、1、2

- 0 表示采用 md5 方式对密码加密。
- 1 表示采用 sha256 方式对密码加密，兼容 postgres 客户端的 MD5 用户认证方式。
- 2 表示采用 sha256 方式对密码加密。

须知

- md5 为不安全的加密算法，不建议用户使用。
- 如果当前集群为 8.0.0 及以下版本升级到当前版本，该参数的默认值为保持前向兼容和原低版本集群一致。例如，8.0.0 版本的 password_encryption_type 默认值为 1，在 8.0.0 集群升级到 8.1.1 版本后，password_encryption_type 默认值保持向前兼容仍旧是 1。

默认值：2

password_min_length

参数说明：该字段决定帐户密码的最小长度。

参数类型：SIGHUP

取值范围：整型，6~999

默认值：8

password_max_length

参数说明：该字段决定帐户密码的最大长度。

参数类型：SIGHUP

取值范围：整型，6~999

默认值：32

password_min_uppercase

参数说明：该字段决定帐户密码中至少需要包含大写字母个数。

参数类型：SIGHUP

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：该字段决定帐户密码中至少需要包含小写字母的个数。

参数类型：SIGHUP

取值范围：整型，0~999

- 0 表示没有限制。

- 1~999 表示创建帐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：该字段决定帐户密码中至少需要包含数字的个数。

参数类型：SIGHUP

取值范围：整型，0~999

- 0 表示没有限制。
- 1~999 表示创建帐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：该字段决定帐户密码中至少需要包含个数。

参数类型：SIGHUP

取值范围：整型，0~999

- 0 表示没有限制。
- 1~999 表示创建帐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：该字段决定帐户密码的有效时间。

参数类型：SIGHUP

取值范围：浮点型，0~999，单位为天。

- 0 表示不开启有效期限制功能。
- 1~999 表示创建帐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：90

password_notify_time

参数说明：该字段决定帐户密码到期前提醒的天数。

参数类型：SIGHUP

取值范围：整型，0~999，单位为天。

- 0 表示不开启提醒功能。
- 1~999 表示帐户密码到期前提醒的天数。

默认值：7

16.4.3 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。

comm_tcp_mode

参数说明：通信库使用 TCP 或 SCTP 协议建立数据通道的切换开关，重启集群生效。

参数类型：POSTMASTER

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_sctp_port

参数说明：TCP 代理通信库或 SCTP 通信库使用的 TCP 或 SCTP 协议监听端口。

参数类型：POSTMASTER

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，0~65535

默认值：port+本机主 DN 数*2+本 DN 在本机 DN 序号

comm_control_port

参数说明：TCP 代理通信库或 SCTP 通信库使用的 TCP 协议监听端口。

参数类型：POSTMASTER

取值范围：整型，0~65535

默认值：port+本机主 DN 数*2+本 DN 在本机 DN 序号+1

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。

comm_max_datanode

参数说明：TCP 代理通信库或 SCTP 通信库支持的最大 DN 数。

参数类型：USERSET

取值范围：整型，1~8192

默认值：实际 DN 数

comm_max_stream

参数说明：TCP 代理通信库或 SCTP 通信库支持的最大并发数据流数。该参数值必须大于**并发数*每并发平均 stream 算子数*（smp 的平方）**。

参数类型：POSTMASTER

取值范围：整型，1~60000

默认值：通过公式 min(query_dop_limit * query_dop_limit * 2 * 20, max_process_memory(字节) * 0.025 /(最大 CN 数+当前 DN 数) / 260)计算，小于 1024 按照 1024 取值，其中，query_dop_limit = 单个机器 CPU 核数 / 单个机器 DN 数。

说明

- 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），若并发数据流数过大，查询较为复杂及 smp 过大都会导致内存不足。
- 如果 comm_max_datanode 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 代理通信库或 SCTP 通信库内部接收线程数量。

参数类型：POSTMASTER

取值范围：整型，1~50

默认值：4

comm_quota_size

参数说明：TCP 代理通信库或 SCTP 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。

参数类型：USERSET

取值范围：整型，0~102400，默认单位为 KB。0 表示不使用 quota 机制。

默认值：1MB

comm_memory_pool_percent

参数说明：单个 DN 内 TCP 代理通信库或 SCTP 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。

参数类型：POSTMASTER

取值范围：整型，0~100

默认值：0

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。

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。

参数类型：USERSET

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，重启集群生效。

参数类型：USERSET

取值范围：布尔型

默认值：off

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。

comm_debug_mode

参数说明：TCP 代理通信库或 SCTP 通信库 debug 模式开关，该参数设置是否打印通信层详细日志，session 级别生效。

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，打开后及时关闭。

参数类型：USERSET

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。

参数类型：USERSET

取值范围：整型，0~20000，单位为毫秒（ms）。取值为 0 表示关闭此功能。

默认值：2000

comm_timer_mode

参数说明：TCP 代理通信库或 SCTP 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，session 级别生效。

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，打开后及时关闭。

参数类型：USERSET

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 代理通信库或 SCTP 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，session 级别生效。

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，打开后及时关闭。

参数类型：USERSET

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 复用切换开关，重启集群生效。

参数类型：POSTMASTER

取值范围：布尔型

- on/true 表示使用 pooler 复用模式。
- 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：off

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。

参数类型：POSTMASTER

取值范围：布尔型

- on/true 表示 CN 和 DN 之间连接为逻辑链接，使用 libcomm 组件。
- off/false 表示 CN 和 DN 之间连接为物理连接，使用 libpq 组件。

须知

如果 CN 设置 comm_cn_dn_logic_conn 为 off，DN 设置 comm_cn_dn_logic_conn 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：off

16.5 资源消耗

16.5.1 内存

介绍与内存相关的参数设置。

须知

本节涉及的参数仅在数据库服务重新启动后生效。

enable_memory_limit

参数说明：启用逻辑内存管理模块。

参数类型：POSTMASTER

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

须知

- 若 max_process_memory-max_shared_memory-cstore buffers 少于 2G，GaussDB(DWS)强制把 enable_memory_limit 设置为 off。
- max_shared_memory 参数与 shared_buffer、max_connections 以及 max_prepared_transactions 参数强相关，如果 max_shared_memory 过大，可以通过调整这三个参数减小。

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。

参数类型：POSTMASTER

取值范围：整型，2*1024*1024～INT_MAX/2，单位为 KB。

默认值：非从备 DN 节点自动适配，公式为（物理内存大小）* 0.8 / (1+主 DN 个数)，当结果不足 2GB 时，默认取 2GB。从备 DN 默认为 12GB。

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）* 0.8 / (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*0.8/(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。

shared_buffers

参数说明：设置 GaussDB(DWS)使用的共享内存大小。增加此参数的值会使 GaussDB(DWS)比系统默认设置需要更多的 System V 共享内存。

参数类型：POSTMASTER

取值范围：整型，128～INT_MAX，单位为 8KB。

改变 BLCKSZ 的值会改变最小值。

默认值：CN 节点为 DN 节点值的 1/2，DN 节点取公式计算：POWER(2,ROUND(LOG(max_process_memory*1024/18,2),0))。如果操作系统支持的共享内存小于 32MB，则在初始化数据存储区时会自动调整为操作系统支持的最大值。

设置建议：

由于 GaussDB(DWS)大部分查询下推，建议 DN 中此参数设置比 CN 大。

建议设置 shared_buffers 值为内存的 40%以内。行存列存分开对待。行存设大，列存设小。列存：(单服务器内存/单服务器 DN 个数)*0.4*0.25。

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。

bulk_write_ring_size

参数说明：数据并行导入使用的环形缓冲区大小。

参数类型：USERSET

取值范围：整型，16384～INT_MAX，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。

buffer_ring_ratio

参数说明：设置并行导出时使用环形缓冲区的阈值大小。

参数类型：USERSET

取值范围：整型，1～1000

默认值：250

说明

- 默认值表示阈值为 shared_buffers 的 250/1000 即 1/4。
- 最小为 shared_buffers 的 1/1000。
- 最大为 shared_buffers 的大小。

设置建议：导出时出现缓存命中率不符合预期的场景建议在 DN 中设置此参数。

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。

参数类型：USERSET

取值范围：整型，800~INT_MAX/2，单位为 8KB。

默认值：8MB

说明

- 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。

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(DWS)比系统默认设置需要更多的 System V 共享内存。

当 GaussDB(DWS)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。

参数类型：POSTMASTER

取值范围：整型，0~536870911，其中 CN 取值为 800 表示关闭预备事务的特性。

默认值：CN 节点为 800， DN 节点为 800

说明

为避免在准备步骤失败，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。

参数类型：USERSET

取值范围：整型，64~INT_MAX，单位为 KB。

默认值：小规格内存为 512MB， 大规格内存为 2GB（max_process_memory 大于等于 30GB 为大规格内存，否则为小规格内存）。

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。

query_mem

参数说明：设置执行作业所使用的内存。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。

参数类型：USERSET

取值范围：整型，0，或大于 32MB 的整型，默认单位为 KB。如果设置值小于 32MB，系统会自动将该参数设置为默认值 0，此时优化器不会根据该值调整作业的估算内存。

默认值：0

query_max_mem

参数说明：设置执行作业所能够使用的最大内存。如果设置的 query_max_mem 值大于 0，在生成执行计划时，优化器会根据该值来设置算子的可用内存。当作业执行时实际所使用内存超过该值时，将报错退出。

参数类型：USERSET

取值范围：整型，0，或大于 32MB 的整型，单位为 KB。如果设置值小于 32MB，系统会自动将该参数设置为默认值 0，此时优化器不会根据该值限制作业的内存使用。

默认值：0

maintenance_work_mem

参数说明：设置在维护性操作（比如 VACUUM、CREATE INDEX、ALTER TABLE ADD FOREIGN KEY 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。

参数类型：USERSET

取值范围：整型，1024~INT_MAX，单位为 KB。

默认值：小规格内存为 512MB， 大规格内存为 2GB（max_process_memory 大于等于 30GB 为大规格内存，否则为小规格内存）。

设置建议：

- 建议设置此参数的值等于 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 16.14 自动清理进程运行时，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。

psort_work_mem

参数说明：设置列存表在进行局部排序中在开始写入临时磁盘文件之前使用的内存大小。带 partial cluster key 的表、带索引的表插入，创建表索引，删除表和更新表都会用到。

参数类型：USERSET

须知

多个正在运行的会话可能会同时进行表的局部排序操作，因此使用的总内存可能是psort_work_mem 的好几倍。

取值范围：整型，64~INT_MAX，单位为 KB。

默认值：512MB

max_loaded_cudesc

参数说明：设置列存表在做扫描时，每列缓存 cudesc 信息的个数。增大设置会提高查询性能，但也会增加内存占用，特别是当列存表的列非常多时。

参数类型：USERSET

取值范围：整型，100~INT_MAX/2

默认值：1024

须知

max_loaded_cudesc 设置过高时，有可能引起内存分配不足。

cstore_buffers

参数说明：设置列存和 OBS、HDFS 外表列存格式（orc、parquet、carbondata）所使用的共享缓冲区的大小。

参数类型：POSTMASTER

取值范围：整型，16384～INT_MAX，单位为 KB。

默认值：CN 为 32MB，DN 取公式计算：
POWER(2,ROUND(LOG(max_process_memory*1024/18,2),0))

设置建议：

列存表使用 cstore_buffers 设置的共享缓冲区，几乎不用 shared_buffers。因此在列存表为主的场景中，应减少 shared_buffers，增加 cstore_buffers。

OBS、HDFS 外表使用 cstore_buffers 设置 ORC、Parquet、Carbondata 的元数据和数据的缓存，元数据缓存大小为 cstore_buffers 的 1/4，最大不超过 2GB，其余缓存空间为列存数据和外表列存格式数据共享使用。

enable_orc_cache

参数说明：设置是否允许在初始化 cstore_buffers 时，将 1/4 的 cstore_buffers 空间预留，用于缓存 orc 元数据。

参数类型：POSTMASTER

取值范围：布尔型

默认值：

- on 表示开启缓存 orc 元数据，可提升 hdfs 表的查询性能，但是会占用列存 buffer 资源，导致列存性能下降。
- off 表示关闭缓存 orc 元数据。

schedule_splits_threshold

参数说明：设置 HDFS 外表 schedule 阶段能够在内存中存储的最大文件个数，超过该限制时，将把文件列表下盘处理。

参数类型：USERSET

取值范围：整型，1~INT_MAX

默认值：60000

bulk_read_ring_size

参数说明：并行导出，使用的环形缓冲区大小。

参数类型：USERSET

取值范围：整型，256~INT_MAX，单位为 KB。

默认值：16MB

check_cu_size_threshold

参数说明：列存表插入时，如果一个 CU 中已插入的数据量大于该参数时，开始进行行级大小校验，避免生成非压缩态大于 1G 的 CU。

参数类型：USERSET

取值范围：整型，0~1024，单位为 MB。

默认值：1024MB

16.5.2 语句磁盘空间管控

介绍与语句磁盘空间管控相关的参数，用于限制语句磁盘空间使用。

sql_use_spacelimit

参数说明：限制单个 SQL 在单个 DN 上，触发写盘操作时，所有类型写盘文件的总空间大小，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。系统管理员用户也受该参数限制。

参数类型：USERSET

取值范围：整型，-1~INT_MAX，单位为 KB。其中-1 表示没有限制。

默认值：-1

设置建议：建议配置 sql_use_spacelimit 为 DN 所在磁盘空间总容量的 10%（如果单磁盘有两个 DN，则该参数设置为磁盘空间总容量的 5%）。

说明

例如，执行语句中配置参数 sql_use_spacelimit=100，当出现单 DN 写盘超过 100kB 时，DWS 会主动终止该 query 的运行，并提示用户单 DN 写盘量超阈值。

```
insert into user1.t1 select * from user2.t1;
ERROR:  The space used on DN (104 kB) has exceeded the sql use space limit (100 kB).
```

建议处理方式：

- 优化语句，减少语句写盘占用空间。
- 如果磁盘空间充足可以适当调大该参数。

temp_file_limit

参数说明：语句执行过程中触发落盘操作时，限制语句中单个线程落盘文件的总空间大小。例如，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落盘文件控制。

参数类型：SUSET

取值范围：整型，-1~INT_MAX，单位为 KB。其中-1 表示没有限制。

默认值：-1

须知

SQL 查询执行时使用的临时表空间不在此限制。

bi_page_reuse_factor

参数说明：行存表批量插入场景下，主备 DN 使用页复制进行数据同步时，可以复用的旧页面空闲空间的百分比。

参数类型：USERSET

取值范围：整型，0~100，单位为%。其中 0 表示不对页面进行复用，全部申请新页面。

默认值：70

须知

- 不建议将此值设置为 50 以下（0 除外），如果复用页面的空闲空间较小的话，会使主备 DN 间传输过多的旧页面数据，从而导致批量插入性能下降。
- 不建议将此值设置为 90 以上，如果此值设置过高，会导致频繁查询空闲页面，但又无法复用旧页面，得不偿失。

16.5.3 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得文件数。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 。

参数类型：POSTMASTER

取值范围：整型，25~INT_MAX

默认值：1000

16.5.4 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 这样的维护语句并不需要迅速完成， 并且不希望他们严重干扰系统执行其他的数据库操作。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(DWS) 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。

参数类型：USERSET

取值范围：整型，0~100，单位为毫秒（ms）。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：0

须知

- 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，最好是调整其他参数，而不是该参数。

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。他代表锁住缓冲池、查找共享的 Hash 表、扫描页面内容的开销。

参数类型：USERSET

取值范围：整型，0~10000，单位为毫秒（ms）。

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。他代表锁住缓冲池、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。

参数类型：USERSET

取值范围：整型，0~10000，单位为毫秒（ms）。

默认值：2

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。他代表把一个脏的磁盘块再次刷新到磁盘上的额外开销。

参数类型：USERSET

取值范围：整型，0~10000，单位为毫秒（ms）。

默认值：20

vacuum_cost_limit

参数说明：导致清理进程休眠的开销限制。

参数类型：USERSET

取值范围：整型，1~10000，单位为毫秒（ms）。

默认值：200

16.5.5 异步 IO

enable_adio_debug

参数说明：允许维护人员输出一些与 ADIO 相关的日志，便于定位 ADIO 相关问题。开发人员专用，不建议普通用户使用。

参数类型：SUSET

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

enable_fast_allocate

参数说明：磁盘空间快速分配开关。只有在 XFS 文件系统上才能开启该开关。

参数类型：SUSET

取值范围：布尔型

- on/true 表示开启此功能。
- off/false 表示关闭此功能。

默认值：off

prefetch_quantity

参数说明：描述行存储使用 ADIO 预读取 IO 量的大小。

参数类型：USERSET

取值范围：整型，1024～1048576，单位为 8KB。

默认值：32MB

backwrite_quantity

参数说明：描述行存储使用 ADIO 写入 IO 量的大小。

参数类型：USERSET

取值范围：整型，1024～1048576，单位为 8KB。

默认值：8MB

cstore_prefetch_quantity

参数说明：描述列存储使用 ADIO 预取 IO 量的大小。

参数类型：USERSET

取值范围：整型，1024～1048576，单位为 KB。

默认值：32MB

cstore_backwrite_quantity

参数说明：描述列存储使用 ADIO 写入 IO 量的大小。

参数类型：USERSET

取值范围：整型，1024～1048576，单位为 KB。

默认值：8MB

cstore_backwrite_max_threshold

参数说明：描述列存储使用 ADIO 写入数据库可缓存最大的 IO 量。

参数类型：USERSET

取值范围：整型，4096～INT_MAX/2，单位为 KB。

默认值：2GB

fast_extend_file_size

参数说明：描述列存储使用 ADIO 预扩展磁盘的大小。

参数类型：SUSET

取值范围：整型，1024～1048576，单位为 KB。

默认值：8MB

effective_io_concurrency

参数说明：磁盘子系统可以同时有效处理的请求数。对于 RAID 阵列，此参数应该是阵列中驱动器主轴的数量。

参数类型：USERSET

取值范围：整型 ，0~1000

默认值：1

16.6 并行导入

GaussDB(DWS)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(DWS)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(DWS)将抛出“文件不存在的”错误。

参数类型：SUSET

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

partition_mem_batch

参数说明：为了优化对列存分区表的批量插入，在批量插入过程中会对数据进行缓存后再批量写盘。通过 partition_mem_batch 可指定缓存个数。该值设置过大，将消耗较多系统内存资源；设置过小，将降低系统列存分区表批量插入性能。

参数类型：USERSET

取值范围：1~ 65535

默认值：256

partition_max_cache_size

参数说明：为了优化对列存分区表的批量插入，在批量插入过程中会对数据进行缓存后再批量写盘。通过 partition_max_cache_size 可指定数据缓存区大小。该值设置过大，将消耗较多系统内存资源；设置过小，将降低列存分区表批量插入性能。

参数类型：USERSET

取值范围：4096~ INT_MAX / 2，最小单位为 KB。

默认值：2GB

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 IO 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。

参数类型：USERSET

取值范围：布尔型

- on 表示开启 GDS debug 功能。
- off 表示不开启 GDS debug 功能。

默认值：off

enable_delta_store

参数说明：该参数现已废弃。为了保留前向兼容，可以设置成功，但是实际不起任何作用。

开启列存 delta 表功能的相关内容，可参考《SQL 语法参考》的“CREATE TABLE”章节中表级参数 **enable_delta**。

参数类型：POSTMASTER

取值范围：布尔型

- on 表示开启列存 delta 表功能。
- off 表示不开启列存 delta 表功能。

默认值：off

16.7 预写式日志

16.7.1 设置

wal_level

参数说明：设置写入 WAL 信息量的级别。

参数类型：POSTMASTER

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。

默认值：hot_standby

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 或者 hot_standby。
- 如果此参数设置为 archive，hot_standby 必须设置为 off，否则将导致数据库无法启动。

synchronous_commit

参数说明：设置当前事务的同步方式。

参数类型：USERSET

取值范围：枚举类型

- on 表示将备机的同步日志刷新到磁盘。
- off 表示异步提交。
- local 表示为本地提交。
- remote_write 表示要备机的同步日志写到磁盘。
- remote_receive 表示要备机同步日志接收数据。

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。

参数类型：POSTMASTER

取值范围：-1~2^{18}，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，为 shared_buffers 的 1/32，最小值为 8 个 XLOG_BLCKSZ，最大值为 2048 个 XLOG_BLCKSZ。
- 如果设置为其他值，当小于 8 时，会被默认设置为 8；当大于 2048 的时，会被强制设置为 2048。

默认值：256MB

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。

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。

参数类型：USERSET

取值范围：整型， 0～100000（微秒），其中 0 表示无延迟。

默认值：0

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。

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。

参数类型：USERSET

取值范围：整型， 0～1000

默认值：5

enable_xlog_group_insert

参数说明：控制 WAL 日志是否启动 group 的插入方式。仅鲲鹏架构支持。

参数类型：SIGHUP

取值范围：布尔型

- on 表示开启。
- off 表示关闭。

默认值：on

wal_compression

参数说明：控制是否对 FPI 页面进行压缩。

参数类型：USERSET

取值范围：布尔型

- on 表示开启 FPI 压缩。
- off 表示关闭 FPI 压缩。

默认值：on

须知

- 当前压缩算法为 zlib，暂不支持设置为其他压缩算法。
- 对于通过从低版本升级成为当前版本的集群，此参数默认关闭（off）。如果用户需要，可以通过 gs_guc 命令打开 FPI 压缩功能。
- 当前版本若为全新安装版本，此参数默认打开（on）。
- 从低版本升级上来的集群，如果手动开启了此参数，不允许再进行集群回滚操作。

wal_compression_level

参数说明：当打开 wal_compression 参数时，设置 zlib 压缩算法的压缩级别。

参数类型：USERSET

取值范围：整型， 0～9

- 0 表示不压缩。
- 1 表示最低的压缩率。
- 9 表示最高的压缩率。

默认值：9

16.7.2 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。

参数类型：SIGHUP

取值范围：整型，最小值 1

默认值：64

须知

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。

checkpoint_timeout

参数说明：设置自动 WAL 检查点之间的最长时间。

参数类型：SIGHUP

取值范围：整型， 30～3600（秒）

默认值：15min

须知

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。

checkpoint_completion_target

参数说明：指定检查点完成的目标。

参数类型：SIGHUP

取值范围：0.0～1.0，其中默认值 0.5 表示每个 checkpoint 需要在 checkpoints 间隔时间的 50%内完成。

默认值：0.5

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。

参数类型：SIGHUP

取值范围：整型（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。

参数类型：SIGHUP

取值范围：整型， 2～3600（秒）

默认值：1min

16.7.3 归档

archive_mode

参数说明：该参数控制是否将已完成的 WAL 段文件进行归档存储。

参数类型：SIGHUP

取值范围：布尔值

- on 表示进行归档。
- off 表示不进行归档。

默认值：off

须知

当 wal_level 设置成 minimal 时，archive_mode 参数无法使用。

archive_command

参数说明：由管理员设置的用于归档 WAL 日志的命令，建议归档路径为绝对路径。

参数类型：SIGHUP

取值范围：字符串

默认值：(disabled)

须知

- 字符串中任何%p 都被要归档的文件的绝对路径代替，而任何%f 都只被该文件名代替（相对路径都相对于数据目录的）。如果需要在命令里嵌入%字符就必须双写%。
- 这个命令当且仅当成功的时候才返回零。示例如下：

```
   archive_command = 'cp --remove-destination %p /mnt/server/archivedir/%f'
archive_command = 'copy %p /mnt/server/archivedir/%f'  # Windows
```

- --remove-destination 选项作用为：拷贝前如果目标文件已存在，会先删除已存在的目标文件，然后执行拷贝操作。
- 当 archive_mode 为 on, archive_mode 未设置时，系统会在 pg_xlog 目录下新建 backup 目录，并将 wal 日志拷贝压缩至 pg_xlog/backup 目录。

---

max_xlog_backup_size

参数说明：设置 pg_xlog/backup 目录下备份 WAL 日志的大小。

参数类型：SIGHUP

取值范围：整型，1048576 ~ 104857600，单位为 KB。

默认值：2097152

须知

- 当且仅当 archive_mode 开启，archive_command 为 NULL 时，max_xlog_backup_size 参数设置才会生效。
- 每分钟检查 pg_xlog/backup 目录下备份 WAL 日志大小，当超过 max_xlog_backup_size 设置值时，删除最早的备份 wal 日志，直至备份 WAL 日志小于 max_xlog_backup_size * 0.9。

---

archive_timeout

参数说明：表示归档周期。

参数类型：SIGHUP

取值范围：整型，0 ~ INT_MAX，单位为秒。其中 0 表示禁用该功能。

默认值：0

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。

16.8 双机复制

16.8.1 发送端服务器

wal_keep_segments

参数说明：Xlog 日志文件段数量。设置“pg_xlog”目录下保留事务日志文件的最小数目，备机通过获取主机的日志进行流复制。

参数类型：SIGHUP

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。

max_replication_slots

参数说明：设置主机端的日志复制 slot 个数。

参数类型：POSTMASTER

取值范围：整型，0 ~ 262143

默认值：8

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 或从备 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，备加从备的和与主 DN 之间的比例。例如，假设集群的 DN 高可用方案为 1 主、1 备、1 从备，则所需物理流复制槽数为 2。

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_build_io_limit

参数说明：用于限制主机在提供备机重建（build）会话时，一秒时间内所允许磁盘读取的数据流量。

参数类型：SIGHUP

取值范围：整型，0～1048576，单位为 KB。

默认值：0，表示主机对备机 build 无 IO 流控限制。

设置建议：可参考磁盘带宽和作业模型。无限制时或无作业干扰时，全量 build 在性能良好的磁盘（如 SSD 盘）下占磁盘带宽比例较小，磁盘 IO 未达到瓶颈，对业务性能影响较小，不需要设置阈值限制。在普通 10000RPM 转速的 SAS 盘下，如果 build 过程中，发现业务性能明显下降，可对该参数进行设置，当前推荐设置为 20MB。

此设置将直接对 build 的进行速度和完成时间产生影响，不建议设置过低（10MB 以下不建议）。在业务低峰时，建议及时取消限制，恢复 build 的正常速度。

说明

- 该参数可在业务高峰期或主机磁盘 IO 压力较大场景时，通过限制备机 build 的流速阈值以减少对主机业务的影响。待业务高峰期过后，可取消限制或重新设置流速阈值。
- 具体业务场景以及磁盘性能状况，建议选择合适的阈值。

16.8.2 主服务器

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。

参数类型：SIGHUP

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。

参数类型：POSTMASTER

取值范围：整型，4~1023，单位为 MB。

默认值：CN 为 16MB，DN 为 128MB

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。

参数类型：USERSET

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。
- 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：on

enable_incremental_catchup

参数说明：控制主备之间数据追赶（catchup）的方式。

参数类型：SIGHUP

取值范围：布尔类型

- on 表示备机 catchup 时用增量 catchup 方式，即从从备本地数据文件扫描获得主备差异数据文件列表，进行主备之间的 catchup。
- off 表示备机 catchup 时用全量 catchup 方式，即从主机本地所有数据文件扫描获得主备差异数据文件列表，进行主备之间的 catchup。

默认值：on

wait_dummy_time

参数说明：同时控制增量数据追赶（catchup）时，集群主备从按顺序启动时等待从备启动的最长时间以及等待从备发回扫描列表的最长时间。

参数类型：SIGHUP

取值范围：整型，范围 1~INT_MAX，单位为秒。

默认值：300s

⚠ 注意

单位只能设置为秒。

16.9 查询规划

16.9.1 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、增加使用 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。

参数类型：USERSET

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

enable_hashagg

参数说明：控制优化器对 Hash 聚集规划类型的使用。

参数类型：USERSET

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。

参数类型：USERSET

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。

参数类型：USERSET

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。

参数类型：USERSET

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。消除整个实体化是不可能的，但是可以关闭这个变量以防止优化器插入实体节点。

参数类型：USERSET

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。

参数类型：USERSET

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。完全消除嵌套循环连接是不可能的，但是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。

参数类型：USERSET

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。

参数类型：USERSET

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：新安装集群的场景下是 on，如果集群是从 R8C10 版本升级上来的，则保持前向兼容。如果是从 R7C10 或者更早的版本升级上来，则默认值是 off。

enable_seqscan

参数说明：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。

参数类型：USERSET

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。完全消除明确的排序是不可能的，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。

参数类型：USERSET

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。

参数类型：USERSET

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。

参数类型：SUSET

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

enforce_oracle_behavior

参数说明：控制正则表达式的规则匹配模式。

参数类型：USERSET

取值范围：布尔型

- on 表示正则表达式采用 ORACLE 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

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。

参数类型：USERSET

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_ctescan

参数说明：控制 stream 计划是否支持 ctescan。

参数类型：USERSET

取值范围：布尔型

- on 表示 stream 计划下支持 ctescan。
- off 表示 stream 计划下不支持 ctescan。

默认值：off

enable_stream_operator

参数说明：控制优化器对 stream 的使用。

参数类型：USERSET

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：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。

参数类型：USERSET

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。

参数类型：USERSET

取值范围：整型，0～INT_MAX。

默认值：200

enable_vector_engine

参数说明：控制优化器对向量化执行引擎的使用。

参数类型：USERSET

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

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。

参数类型：USERSET

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。

参数类型：USERSET

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：off

enable_fstream

参数说明：控制优化器下发语句时对 stream 的使用，该参数只限定于 HDFS 外表使用。

该参数现在已经废弃。为了保留前向兼容，可以设置成功，但是实际不起任何作用。

参数类型：USERSET

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。

参数类型：USERSET

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。

参数类型：USERSET

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_valuepartition_pruning

参数说明：是否对 DFS 分区表进行静态/动态优化。

参数类型：USERSET

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更合适的计算 Node Group。

参数类型：USERSET

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。
- bind：当前 session 用户是逻辑集群用户时，候选计算 Node Group 为当前用户关联的逻辑集群的 Node Group；当 session 用户不是逻辑集群用户时，候选计算 Node Group 选取规则和参数设置为 query 时的规则一致。
- Node Group 名：
  - enable_nodegroup_debug 为 off 时：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。
  - enable_nodegroup_debug 为 on 时：候选计算 Node Group 为指定的 Node Group。

默认值：bind

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。

参数类型：USERSET

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。

参数类型：USERSET

取值范围：浮点型，0～DBL_MAX。

默认值：1

须知

此参数仅对 Redistribute 和 Broadcast 类型的 Stream 有效。

qrw_inlist2join_optmode

参数说明：控制是否使用 inlist-to-join 查询重写。

参数类型：USERSET

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 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。

参数类型：USERSET

取值范围：字符串

- 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

16.9.2 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。

参数类型：USERSET

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。

参数类型：USERSET

取值范围：浮点型，0～DBL_MAX。

默认值：4

说明

- 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。
- 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。

参数类型：USERSET

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。

参数类型：USERSET

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。

参数类型：USERSET

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，还要考虑 GaussDB(DWS)的共享缓冲区以及内核的磁盘缓冲区。另外，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(DWS)分配的共享内存大小没有影响，它也不会使用内核磁盘缓冲，它只用于估算。数值是用磁盘页来计算的，通常每个页面是 8192 字节。

参数类型：USERSET

取值范围：整型，1～INT_MAX，单位为 8KB。

比默认值高的数值可能会导致使用索引扫描，更低的数值可能会导致选择顺序扫描。

默认值：128MB

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。

参数类型：USERSET

取值范围：浮点型，0～DBL_MAX。

默认值：0

16.9.3 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。

参数类型：USERSET

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

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。

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。

参数类型：USERSET

取值范围：整型，2～INT_MAX。

默认值：12

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。

geqo_effort

参数说明：控制 GEQO 在规划时间和规划质量之间的平衡。

参数类型：USERSET

取值范围：整型，1～10

默认值：5

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。
- geqo_effort 实际上并没有直接作用，只是用于计算其他影响 GEQO 的变量的默认值。如有需求，可以手动设置其他相关参数。

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。

参数类型：USERSET

取值范围：整型，0～INT_MAX

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(DWS)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。

参数类型：USERSET

取值范围：整型，0～INT_MAX

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。

参数类型：USERSET

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。

参数类型：USERSET

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

16.9.4 其他优化器选项

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。此参数设置为正数是代表统计信息的样本数量，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%。采样时，会将 default_statistics_target * 300 作为随机抽样的大小，例如默认值为 100 时，会读取 100* 300 个页面来完成随机抽样。

参数类型：USERSET

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

  - 恢复默认的统计信息。

1. 使用 plan hint 来调整到之前的查询计划。

- 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，autoanalyze 不支持百分比采样，采样过程使用参数默认值。
- 当次 guc 参数设置为正数，用户执行 analyze 需要被授予 ANALYZE 权限。
- 当此 guc 参数设置为负数，即百分比采样时，用户执行 analyze 需要同时被授予 ANALYZE 和 SELECT 权限。

默认值：100

random_function_version

参数说明：控制 analyze 在进行数据采样时选取的 random 函数版本。该参数仅 8.1.2 及以上版本支持。

参数类型：USERSET

取值范围：枚举类型

- 0 表示采用 C 标准库提供的 random 函数。
- 1 表示采用通过每次给不同随机种子封装的 random 函数。

默认值：0

constraint_exclusion

参数说明：控制查询优化器使用表约束查询的优化。

参数类型：USERSET

取值范围：枚举类型

- on 表示检查所有表的约束。
- off 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。

参数类型：USERSET

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。

参数类型：USERSET

取值范围：整型，1～INT_MAX。

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 JOINS 之外的 JOIN 构造重写到 FROM 列表中。

参数类型：USERSET

取值范围：整型，1～INT_MAX。

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。

参数类型：USERSET

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_hdfs_predicate_pushdown

参数说明：标示是否启用谓词下推至原生数据层的功能。

参数类型：SUSET

取值范围：布尔型

- on 表示启用谓词下推至原生数据层的功能。
- off 表示不启用谓词下推至原生数据层的功能。

默认值：on

enable_random_datanode

参数说明：标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。

参数类型：USERSET

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 AGG 操作时 HASH 表的大小。

参数类型：USERSET

取值范围：整型，0～INT_MAX/2。

默认值：0

enable_codegen

参数说明：标识是否允许开启代码生成优化，目前代码生成使用的是 LLVM 优化。

参数类型：USERSET

取值范围：布尔型

- on 表示允许开启代码生成优化。
- off 表示不允许开启代码生成优化。

须知

目前 LLVM 优化仅支持向量化执行引擎特性和 SQL on Hadoop 特性，在其他场景下建议关闭此参数。

默认值：on

codegen_strategy

参数说明：标识在表达式 codegen 化过程中所使用的代码生成优化策略。

参数类型：USERSET

取值范围：枚举类型

- partial 表示当所计算表达式中即使包含部分未被 codegen 化的函数时，仍可借助表达式全 codegen 框架调用 LLVM 动态编译优化策略。
- pure 表示当所计算表达式整体可被 codegen 化时，才考虑调用 LLVM 动态编译优化策略。

须知

在开启代码生成优化会导致查询性能下降的场景下可以设置此参数为 pure，其他场景下建议不改变此参数的默认值 partial。

默认值：partial

enable_codegen_print

参数说明：标识是否允许在 log 日志中打印所生成的 LLVM IR 函数。

参数类型：USERSET

取值范围：布尔型

- on 表示允许在 log 日志中打印 IR 函数。
- off 表示不允许在 log 日志中打印 IR 函数。

默认值：off

codegen_cost_threshold

参数说明：由于 LLVM 编译生成最终的可执行机器码需要一定时间，因此只有当实际执行的代价大于编译生成机器码所需要的代码和优化后的执行代价之和时，利用代码生成才有收益。codegen_cost_threshold 标识代价的阈值，当执行估算代价大于该代价时，使用 LLVM 优化。

参数类型：USERSET

取值范围：整型，0～INT_MAX

默认值：10000

enable_constraint_optimization

参数说明：标识是否允许 HDFS 外表使用信息约束（Informational Constraint）优化执行计划。

参数类型：SUSET

取值范围：布尔型

- on 表示允许使用信息约束优化执行计划。
- off 表示不允许使用信息约束优化执行计划。

默认值：on

enable_bloom_filter

参数说明：标识是否允许使用 BloomFilter 优化。

参数类型：USERSET

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

须知

适用场景：外表侧同线程包含有 HDFS 内外表或列存表的 HASH JOIN 会触发 Bloom Filter。

使用限制：

1. JOIN 类型仅限于 INNER JOIN、SEMI JOIN、RIGHT JOIN、RIGHT SEMI JOIN、RIGHT ANTI JOIN、RIGHT ANTI FULL JOIN。
2. JOIN 内表侧的数据不能超过 5 万行。
3. JOIN 内表侧关联条件：对于 HDFS 内外表不能为表达式；对于列存表可以为表达式，但仅限于非 JOIN 层计算的表达式。
4. JOIN 外表侧关联条件必须为简单列关联。
5. JOIN 内表侧与外表侧关联条件均为简单列关联时，计划层估算必须可以去除 1/3 以上的数据（仅针对 HDFS 内外表）。
6. JOIN 不能包含 null 值关联。
7. JOIN 层未出现下盘。
8. 数据类型：

- HDFS 内外表字段类型支持 SMALLINT、INTEGER、BIGINT、REAL/FLOAT4、DOUBLE PRECISION/FLOAT8、CHAR(n)/CHARACTER(n)/NCHAR(n)、VARCHAR(n)/CHARACTER VARYING(n)、CLOB、TEXT。
- 列存表字段类型支持 SMALLINT、INTEGER、BIGINT、OID、"char"、CHAR(n)/CHARACTER(n)/NCHAR(n)、VARCHAR(n)/CHARACTER VARYING(n)、NVARCHAR2(n)、CLOB、TEXT、DATE、TIME、TIMESTAMP、TIMESTAMPTZ，其中字符类型其排序规则必须指定为"C"。

---

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。

参数类型：USERSET

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“统计信息完全缺失”或“修改量达到 analyze 阈值”的表进行统计信息自动收集，当前不支持对外表触发 autoanalyze，不支持对带有 ON COMMIT [DELETE ROWS|DROP]选项的临时表触发 autoanalyze，如需收集，需用户手动执行 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。

须知

表的修改量达到 analyze 阈值是指：表的修改量超过 autovacuum_analyze_threshold + autovacuum_analyze_scale_factor * reltuples，其中 reltuples 是 pg_class 中记录的表的估算行数。

参数类型：SUSET

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：on

query_dop

参数说明：用户自定义的查询并行度。

参数类型：USERSET

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为每个查询选取[1,8]之间（x86 平台），[1,64]之间（鲲鹏平台）的最优的并行度。

[-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

- 对于短查询为主的 TP 类业务中，如果不能通过 CN 轻量化或下发语句进行业务的调优，则生成 SMP 计划的时间较长，建议设置 query_dop=1。对于 AP 类复杂语句的场景，建议设置 query_dop=0。
- 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，以达到良好效果。
- 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

query_dop_ratio

参数说明：用于当 query_dop 取值为 0 时，设置在系统中给定的最优 dop 基础上，调整 dop 的倍数。即最终 dop=系统设置 dop * query_dop_ratio（最小值为 1，最大值为 64）。当设置为 1 时，表示不调整。

参数类型：USERSET

取值范围：浮点型，0-64

默认值：1

debug_group_dop

参数说明：当 query_dop 取值为 0 时，针对生成的执行计划划分的以 Stream 算子为顶点的 group，均分配统一的 dop 并行度。此参数用于人为指定特定 group 的 dop 进行性能调优，格式为 G1,D1,G2,D2,...，其中：G1,G2 为 group 的 ID，可以从日志中获得，D1,D2 为指定的 dop 值，可以为任意正整数。

参数类型：USERSET

取值范围：字符型

默认值：空

须知

该参数仅供内部调优使用，不允许用户进行设置，建议保持默认值。

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。

参数类型：SUSET

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：on

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。

参数类型：USERSET

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(enable_codegen 打开后获得较大性能提升)，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”。

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。

参数类型：USERSET

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”。

默认值：on

enable_sonic_optspill

参数说明：标识是否优化 sonic 场景下 HashJoin 或者 HashAgg 的下盘文件个数。仅在 enable_sonic_hashjoin 或者 enable_sonic_hashagg 开启情况下生效。

参数类型：USERSET

取值范围：布尔型

- on 表示开启下盘文件数优化。
- off 表示关闭下盘文件数优化。

说明

满足 sonic 条件下的 HashJoin 或者 HashAgg 算子，在关闭此参数（设置为 off）时每列会产生 1 个下盘文件。开启此参数（设置为 on）时如果不同列数据类型相似，只会有 1 个下盘文件（最多 5 个文件)。

默认值：on

expand_hashtable_ratio

参数说明：控制 Hash Agg 和 Hash Join 算子执行过程中 hash 表的大小扩大比例。

参数类型：USERSET

取值范围：浮点型，0, 0.5~10

说明

- 默认值设置为 0 时表示 hash 表大小会根据当前内存进行自适应扩展。
- 默认值设置为 0.5~10 之间时，显式的是指定 hash 表扩大的倍数，通常 hash 表越大性能越好，但会占用更多内存空间，在内存不足场景可能造成数据提前下盘，带来性能劣化。

默认值：0

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。

参数类型：USERSET

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。
- force_custom_plan 表示强制走 custom plan。

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。
- custom plan 是指对于 preapre 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。
- 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。

默认值：auto

wlm_query_accelerate

参数说明：标识在短查询加速打开时，查询是否需要加速。

参数类型：USERSET

取值范围：整型，-1~1

- -1：短查询由快车道管控，长查询由慢车道管控。
- 0：查询不加速，短查询和长查询均由慢车道管控。
- 1：查询加速，短查询和长查询均由快车道管控。

默认值：-1

show_unshippable_warning

参数说明：标识是否将语句不下推的告警打印到客户端。

参数类型：USERSET

取值范围：布尔型

- on：将语句不下推的原因以 WARNING 打印到日志和客户端
- off：仅将语句不下推的原因以 LOG 打印到日志

默认值：off

hashjoin_spill_strategy

参数说明：选择 hashjoin 下盘策略。（该参数 8.1.2 及以上版本支持）

参数类型：USERSET

取值范围：整型，0～4

- 0：当内表较大，并且多次下盘无法分开时，尝试外表是否可以放到数据库可用内存建立哈希表。如果内外表均很大，执行 NestLoop。
- 1：当内表较大，并且多次下盘无法分开时，尝试外表是否可以放到数据库可用内存建立哈希表。如果内外表均很大，强制执行 HashJoin。
- 2：当内表较大，并且多次下盘无法分开时，强制执行 HashJoin。
- 3：当内表较大，并且多次下盘无法分开时，尝试外表是否可以放到数据库可用内存建立哈希表。如果内外表均很大，则报错。
- 4：当内表较大，并且多次下盘无法分开时，则报错。

说明

- 此参数只对向量化 HashJoin 生效。
- 对于数据 distinct 值很少且数据量很大场景，可能出现无法下盘导致使用内存过大产生内存不受控的问题。 取值 0 时通过尝试内外表交换或者 Nestloop 可以避免出现此类内存不受控问题。 执行 Nestloop 可能造成某些场景性能劣化。

- 取值 0 对向量化 Full Join 不生效，行为与取值 1 相同。只尝试外表是否可建立哈希表，不执行 NestLoop

默认值：0

16.10 错误报告和日志

16.10.1 记录日志的位置

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。

参数类型：SIGHUP

取值范围： 布尔型

- on 表示 GaussDB(DWS)以覆盖写入的方式写服务器日志消息。
- off 表示 GaussDB(DWS)将日志消息附加到同名的现有日志文件上。

默认值：off

说明

示例:

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。

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。

参数类型：SIGHUP

取值范围：整型，0 ~ 24d，单位为 min，h，d。其中 0 表示关闭基于时间的新日志文件的创建。

默认值：1d

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。

参数类型：SIGHUP

取值范围：整型，INT_MAX / 1024，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

event_source

参数说明：log_destination 设置为 eventlog 时，event_source 设置在日志中 GaussDB(DWS)日志消息的标识。

参数类型：POSTMASTER

取值范围：字符串

默认值：PostgreSQL

16.10.2 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。

参数类型：USERSET

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 debug5、debug4、debug3、debug2、debug1、info、log、notice、warning、error。参数的详细信息请参见表 16-1。

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。

参数类型：SUSET

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 debug5、debug4、debug3、debug2、debug1、info、log、notice、warning、error、fatal、panic。参数的详细信息请参见表 16-1。

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。

参数类型：SUSET

取值范围：枚举类型，有效值有 debug5、debug4、debug3、debug2、debug1、info、log、notice、warning、error、fatal、panic。参数的详细信息请参见表 16-1。

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。

参数类型：SUSET

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 ~ INT_MAX，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：30min

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。

参数类型：SUSET

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 debug5、debug4、debug3、debug2、debug1、info、log、notice、warning、error、fatal、panic。参数的详细信息请参见表 16-1。

默认值：panic

表 16-1 解释 GaussDB(DWS)中使用的消息安全级别。当日志输出到 syslog 或者 eventlog 时，GaussDB(DWS)进行如表中的转换。

表16-1 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

plog_merge_age

参数说明：该参数用于控制性能日志数据输出的周期。

参数类型：SUSET

须知

该参数以毫秒为单位的，建议在使用过程中设置值为 1000 的整数倍，即设置值以秒为最小单位。该参数所控制的性能日志文件以 prf 为扩展名，文件放置在 $GAUSSLOG/gs_profile/<node_name> 目录下面，其中 node_name 是由 postgres.conf 文件中的 pgxc_node_name 的值，不建议外部使用该参数。

取值范围：0~INT_MAX，单位为毫秒（ms）。

- 当设置为 0 时，当前会话不再输出性能日志数据。
- 当设置为非 0 时，当前会话按照指定的时间周期进行输出性能日志数据。该参数设置得越小，输出的日志数据越多，对性能的负面影响越大。

默认值：3s

profile_logging_module

参数说明：用于设置记录性能日志的类型，使用该参数时需确保 plog_merge_age 参数值非 0。该参数属于会话级参数，不建议通过 gs_guc 工具来设置。仅 8.1.3 及以上集群版本支持。

参数类型：USERSET

取值范围：字符串

默认值：默认打开 OBS,HADOOP,REMOTE_DATANODE，关闭 MD。可由 SHOW profile_logging_module 查看。

设置方法：首先，可以通过 SHOW profile_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：

```
show profile_logging_module;
profile logging module
-----------------------------------------------
ALL,on(OBS,HADOOP,REMOTE_DATANODE),off(MD)(1 row)
```

打开 MD 性能日志，并查看设置结果。其中 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。

```
set profile_logging_module='on(md)';
SET

show profile_logging_module;
profile_logging_module
-----------------------------------------------
ALL,on(MD,OBS,HADOOP,REMOTE_DATANODE),off()(1 row)
```

16.10.3 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。

参数类型：SIGHUP

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。

参数类型：SIGHUP

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：用于控制打印查询执行结果。

参数类型：SIGHUP

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。

参数类型：USERSET

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。

参数类型：SIGHUP

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。

参数类型：BACKEND

须知

- 会话连接参数，不建议用户设置。
- 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。

参数类型：BACKEND

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

说明

会话连接参数，不建议用户设置。

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。

参数类型：SUSET

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：on

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。

参数类型：SUSET

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。

参数类型：SIGHUP

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_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。

参数类型：SUSET

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）。

参数类型：SUSET

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。

参数类型：SUSET

取值范围：整型，-1~INT_MAX，单位为 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。

参数类型：SIGHUP

取值范围：字符串

默认值：PRC

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。

参数类型：USERSET

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。

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：

```
show logging_module;
logging_module
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------
ALL,on(),off(DFS,GUC,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,ANALYZE,CACHE,ADIO,SSL,GDS,TBLSP
C,WLM,SPACE,OBS,EXECUTOR,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,
OPT_SUBPLAN,OPT_SETOP,OPT_CARD,OPT_SKEW,SMP,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PLANH
INT,PARQUET,CARBONDATA,SNAPSHOT,XACT,HANDLE,CLOG,TQUAL,EC,REMOTE,CN_RETRY,PLSQL,TEX
TSEARCH,SEQ,INSTR,COMM_IPC,COMM_PARAM,CSTORE,JOB,STREAMPOOL,STREAM_CTESCAN)
(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：

```
set logging_module='on(SSL)';
SET
show logging_module;
logging_module
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
--------

ALL,on(SSL),off(DFS,GUC,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,ANALYZE,CACHE,ADIO,GDS,TBLSPC
,WLM,SPACE,OBS,EXECUTOR,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,O
PT_SUBPLAN,OPT_SETOP,OPT_CARD,OPT_SKEW,SMP,UDF,COOP_ANALYZE,WLMCP,A
CCELERATE,PLANHINT,PARQUET,CARBONDATA,SNAPSHOT,XACT,HANDLE,CLOG,TQUAL,EC,REMOTE,CN_
RETRY,PLSQL,TEXTSEARCH,SEQ,INSTR,COMM_IPC,COMM_PARAM,CSTORE,JOB,STREAMPOOL,STREAM_C
TESCAN)
(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。

```
set logging_module='off(ALL)';
SET
show logging_module;
```

```
logging_module
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
---------

ALL,on(),off(DFS,GUC,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,ANALYZE,CACHE,ADIO,SSL,GDS,TBLSP
C,WLM,SPACE,OBS,EXECUTOR,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,
OPT_SUBPLAN,OPT_SETOP,OPT_CARD,OPT_SKEW,SMP,UDF,COOP_ANALYZE,WLMCP,
ACCELERATE,PLANHINT,PARQUET,CARBONDATA,SNAPSHOT,XACT,HANDLE,CLOG,TQUAL,EC,REMOTE,CN
_RETRY,PLSQL,TEXTSEARCH,SEQ,INSTR,COMM_IPC,COMM_PARAM,CSTORE,JOB,STREAMPOOL,STREAM_
CTESCAN)
(1 row)

set logging_module='on(ALL)';
SET
show logging_module;
logging_module
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
---------

ALL,on(DFS,GUC,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,ANALYZE,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,
SPACE,OBS,EXECUTOR,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SU
BPLAN,OPT_SETOP,OPT_CARD,OPT_SKEW,SMP,UDF,COOP_ANALYZE,WLMCP,ACCELE
RATE,PLANHINT,PARQUET,CARBONDATA,SNAPSHOT,XACT,HANDLE,CLOG,TQUAL,EC,REMOTE,CN_RETRY
,PLSQL,TEXTSEARCH,SEQ,INSTR,COMM_IPC,COMM_PARAM,CSTORE,JOB,STREAMPOOL,STREAM_CTESCA
N),off()
(1 row)
```

所有模块中，COMM_IPC 必须显式的打开/开闭，执行以下命令都可以将该模块的日志打开：

```
set logging_module='on(ALL)';
SET
set logging_module='on(COMM_IPC)';
SET
```

设置成功后，COMM_IPC 模块日志不会自动关闭，关闭 COMM_IPC 模块的日志，必须手动执行关闭命令，以下两条命令都可以将该模块日志关闭：

```
set logging_module='off(ALL)';
SET
set logging_module='off(COMM_IPC)';
SET
```

依赖关系：该参数依赖于 log_min_messages 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。

参数类型：SUSET

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：on

16.11 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。

alarm_report_interval

参数说明：指定告警上报的时间间隔。

参数类型：SIGHUP

取值范围：非负整型，单位为秒。

默认值：10

16.12 运行时统计

16.12.1 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。

参数类型：SUSET

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。

参数类型：SUSET

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 自动清理进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。

参数类型：SUSET

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。

参数类型：SUSET

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 和 C 语言函数。
- 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。

参数类型：POSTMASTER

取值范围：整型，100～102400

默认值：1024

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。

参数类型：SUSET

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

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。

参数类型：SUSET

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。

参数类型：USERSET

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

instr_unique_sql_count

参数说明：控制是否收集 Unique SQL，以及收集数量限制。

参数类型：SIGHUP

取值范围：整型，0～ INT_MAX

- 值为 0 时，表示不收集 Unique SQL 统计信息；
- 值大于 0 时，在 CN 节点上，将会控制收集的 Unique SQL 数量不超过该设置值。当收集数量达到限制时，不再收集新的 Unique SQL，此时可通过 reload 调大设置值，继续收集新的 Unique SQL。

默认值：0

⚠ 注意

通过 reload 加载新的设置值时，如果新设置值小于原设置值，将会清空对应 CN 节点已收集的 Unique SQL 统计信息。需特别注意该清理操作将由资源管理后台线程完成，若 GUC 参数 use_workload_manager 为 off 时清理操作可能失败，可直接使用函数 reset_instr_unique_sql 进行清理。

instr_unique_sql_timeout

参数说明：控制 Unique SQL 的存在时间。StatCollector 后台线程每小时对所有的 Unique SQL 做一次检查，如果发现某个 Unique SQL 超过 instr_unique_sql_timeout 小时未被执行，则将其删除。（该参数在 8.1.2 及以上版本支持。）

参数类型：SIGHUP

取值范围：整型，0～ INT_MAX，单位为小时。

- 值为 0 时，表示不删除过期的 Unique SQL；
- 值大于 0 时，发现超过 instr_unique_sql_timeout 小时未执行 Unique SQL 则删除。

默认值：24

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 语句是否进行计数统计，对 SELECT、INSERT、UPDATE、DELETE 语句进行响应时间的统计，以及对 DDL、DML、DCL 语句进行计数的统计。

参数类型：SUSET

取值范围：布尔型

- on 表示开启统计功能。
- off 表示关闭统计功能。

默认值：on

说明

- track_sql_count 参数受 track_activities 约束：

- track_activities 开启而 track_sql_count 关闭时，如果查询了 gs_sql_count、pgxc_sql_count、gs_workload_sql_count、pgxc_workload_sql_count、global_workload_sql_count、gs_workload_sql_elapse_time、pgxc_workload_sql_elapse_time、或 global_workload_sql_elapse_time 视图，将会有 LOG 提示 track_sql_count 是关闭的；
- track_activities 和 track_sql_count 同时关闭，那么此时将会有两条 LOG，分别提示 track_activities 是关闭的和 track_sql_count 是关闭的；
- track_activities 关闭而 track_sql_count 开启，此时将仅有 LOG 提示 track_activities 是关闭。
- 当参数关闭时，查询视图的结果为 0 行。

enable_track_wait_event

参数说明：控制是否对各类等待事件的发生次数、失败次数、持续时间、最大、最小和平均等待时间等信息进行统计。

参数类型：SIGHUP

取值范围：布尔型

- on 表示开启统计功能。
- off 表示关闭统计功能。

默认值：off

说明

- enable_track_wait_event 参数受 track_activities 约束，如果 track_activities 关闭，即使开启 enable_track_wait_event 也不启用相关功能。
- track_activities 或 enable_track_wait_event 关闭时，如果查询 get_instr_wait_event 函数、gs_wait_events 视图或 pgxc_wait_events 视图将会提示 GUC 参数关闭，查询结果为 0 行；
- 在集群运行过程中关闭 track_activities 或 enable_track_wait_event，GaussDB(DWS)不再对等待事件相关信息进行统计，但已统计记录的数据不受影响。

enable_wdr_snapshot

参数说明：控制是否启用性能视图快照功能。开启后，GaussDB(DWS)会定期对部分系统性能视图创建快照并持久化保存，并接受手动创建快照请求。

参数类型：SIGHUP

取值范围：布尔型

- on 表示开启快照功能。
- off 表示关闭快照功能。

默认值：off

说明

- 如果 enable_wdr_snapshot 参数关闭，执行 create_wdr_snapshot 函数手动创建视图会提示 GUC 参数未打开。

- 如果在快照的过程中修改 enable_wdr_snapshot 参数，正在创建的快照不受影响，下次定期或手动创建快照时生效。

wdr_snapshot_interval

参数说明：设置自动创建性能视图快照的时间间隔。

参数类型：SIGHUP

取值范围：整型，10～180，单位为分钟。

默认值：60

说明

- 该参数取值应与集群负载相对应，不宜取值过小，建议大于创建一次快照所需时间。
- 如果 wdr_snapshot_interval 小于创建一次快照所需时间，也就是说到了快照的时间，发现上一个快照还没执行完，则跳过本次快照。

wdr_snapshot_retention_days

参数说明：设置性能快照数据保留的最大天数。

参数类型：SIGHUP

取值范围：整型，1～15，单位为天。

默认值：8

说明

- 开启 enable_wdr_snapshot 的情况下，保存 wdr_snapshot_retention_days 天数的快照数据会被自动清除。
- 该参数取值应与可用磁盘空间相对应，取值越大，需要的磁盘空间越大。
- 对该参数的修改不会立即生效，等到下一次自动创建快照时才会清除过期快照数据。

16.12.2 性能统计

在数据库在运行过程中，会涉及到锁的访问、磁盘 IO 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(DWS)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() 。

参数类型：SUSET

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

16.13 负载管理

未对数据库资源做控制时，容易出现并发任务抢占资源导致操作系统过载甚至最终崩溃。操作系统过载时，其响应用户任务的速度会变慢甚至无响应；操作系统崩溃时，整个系统将无法对用户提供任何服务。GaussDB(DWS)的负载管理功能能够基于可用资源的多少均衡数据库的负载，以避免数据库系统过载。

use_workload_manager

参数说明：是否开启资源管理功能。此参数需在 CN 和 DN 同时应用。

参数类型：SIGHUP

取值范围：布尔型

- on 表示打开资源管理。
- off 表示关闭资源管理。

说明

- 当使用 GUC 参数设置来修改参数值时，新参数值只能对更改操作执行后启动的线程生效。此外，对于后台线程以及线程复用执行的新作业，该参数值的改动不会生效。如果希望这类线程即时识别参数变化，可以使用 kill session 或重启节点的方式来实现。
- use_workload_manager 参数由 off 变为 on 状态后，资源管理视图变为可用，并且可以查询 off 状态下统计的存储资源使用情况。若存在些许误差的情况下，需要矫正用户使用的存储资源，可数据库中执行如下命令，在执行该命令的过程中，如果对表中插入数据，可能会出现统计不够准确的情况：

```
select gs_wlm_readjust_user_space(0);
```

默认值：on

enable_control_group

参数说明：是否开启 Cgroups 功能。此参数需在 CN 和 DN 同时应用。

参数类型：SIGHUP

取值范围：布尔型

- on 表示打开 Cgroups 管理。
- off 表示关闭 Cgroups 管理。

默认值：on

说明

当使用 GUC 参数设置来修改参数值时，新参数值只能对更改操作执行后启动的线程生效。此外，对于后台线程以及线程复用执行的新作业，该值的改动不会生效。如果希望这类线程即时识别参数变化，可以使用 kill session 或重启节点的方式来实现。

enable_backend_control

参数说明：是否控制数据库常驻线程到 DefaultBackend 控制组。此参数需在 CN 和 DN 同时应用。

参数类型：POSTMASTER

取值范围：布尔型

- on 表示控制常驻线程到 DefaultBackend 控制组。
- off 表示不控制常驻线程到 DefaultBackend 控制组。

默认值：on

enable_vacuum_control

参数说明：是否控制数据库常驻线程 autoVacuumWorker 到 Vacuum 控制组。此参数需在 CN 和 DN 同时应用。

参数类型：POSTMASTER

取值范围：布尔型

- on 表示控制数据库常驻线程 autoVacuumWorker 到 Vacuum 控制组。
- off 表示不控制数据库常驻线程 autoVacuumWorker 到 Vacuum 控制组。

默认值：on

enable_perm_space

参数说明：是否开启 perm space 功能。此参数需在 CN 和 DN 同时应用。

参数类型：POSTMASTER

取值范围：布尔型

- on 表示打开 perm space 管理。
- off 表示关闭 perm space 管理。

默认值：on

space_once_adjust_num

参数说明：空间管控和空间统计功能中，控制慢速构建与细粒度校准操作中每次处理的文件个数阈值。该参数 8.1.3 及以上集群版本支持。

参数类型：SIGHUP

取值范围：整型，0~INT_MAX

- 0 表示不启用慢速构建和细粒度校准功能。

默认值：300

说明

文件个数阈值影响数据库资源，建议合理设置。

space_readjust_schedule

参数说明：空间管控和空间统计功能中，控制是否触发自动校准以及校准空间误差阈值。该参数 8.1.3 及以上集群版本支持。

参数类型：SIGHUP

取值范围：字符串

- off 表示关闭自动校准功能。
- auto 表示打开自动校准功能，并且触发自动校准的差误阈值为 1GB。
- auto(空间大小+K/M/G)表示打开自动校准功能，并且触发自动校准的误差阈值为自定义的空间大小 KB/MB/GB。例如，auto(200M)表示打开自动校准功能，且触发自动校准的误差阈值为 200MB。

默认值：auto

enable_verify_active_statements

参数说明：在静态自适应负载场景下，是否开启后台校准功能。此参数需在 CN 上应用。

参数类型：SIGHUP

取值范围：布尔型

- on 表示打开后台校准功能。
- off 表示关闭后台校准功能。

默认值：on

max_active_statements

参数说明：设置全局的最大并发数量。此参数只应用到 CN，且针对一个 CN 上的执行作业。

数据库管理员需根据系统资源（如 CPU 资源、IO 资源和内存资源）情况，调整此数值大小，使得系统支持最大限度的并发作业，且防止并发执行作业过多，引起系统崩溃。

参数类型：SIGHUP

取值范围：整型，-1 ～ INT_MAX。设置为-1 和 0 表示对最大并发数不做限制。

默认值：60

parctl_min_cost

参数说明：静态资源管理场景下，复杂作业最小估算代价。用于简单作业和复杂作业划分的阈值，估算代价小于该值的作业为简单作业，估算代价大于等于该值的作业为复杂作业。

参数类型：SIGHUP

取值范围：整型，-1 ～ INT_MAX

- parctl_min_cost 等于-1 时，所有作业都是简单作业；
- 估算代价小于 10 的作业任何场景下都是简单作业。

默认值：100000

cgroup_name

参数说明：设置当前使用的 Cgroups 的名字或者调整当前 group 下排队的优先级。

即如果先设置 cgroup_name，再设置 session_respool，那么 session_respool 关联的控制组起作用，如果再切换 cgroup_name，那么新切换的 cgroup_name 起作用。

切换 cgroup_name 的过程中如果指定到 Workload 控制组级别，数据库不对级别进行验证。级别的范围只要在 1-10 范围内都可以。

参数类型：USERSET

建议尽量不要混合使用 cgroup_name 和 session_respool。

取值范围：字符串

默认值：DefaultClass:Medium

说明

DefaultClass:Medium 表示 DefaultClass 下 Timeshare 控制组中的 Medium 控制组。

cpu_collect_timer

参数说明：设置语句执行时在 DN 上收集 CPU 时间的周期。

数据库管理员需根据系统资源（如 CPU 资源、IO 资源和内存资源）情况，调整此数值大小，使得系统支持较合适的收集周期，太小会影响执行效率，太大会影响异常处理的精确度。

参数类型：SIGHUP

取值范围：整型，1 ～ INT_MAX， 单位为秒。

默认值：30

enable_cgroup_switch

参数说明：是否控制数据库执行语句时根据类型自动切换到 TopWD 组。

参数类型：USERSET

取值范围：布尔型

- on 表示控制数据库执行语句时根据类型自动切换到 TopWD 组。
- off 表示控制数据库执行语句时根据类型不自动切换到 TopWD 组。

默认值：off

memory_tracking_mode

参数说明：设置记录内存信息的模式。

参数类型：USERSET

取值范围：

- none，不启动内存统计功能。
- normal，仅做内存实时统计，不生成文件。
- executor，生成统计文件，包含执行层使用过的所有已分配内存的上下文信息。
- fullexec，生成文件包含执行层申请过的所有内存上下文信息。

默认值：none

memory_detail_tracking

参数说明：设置需要的线程内分配内存上下文的顺序号以及当前线程所在 query 的 plannodeid。

参数类型：USERSET

取值范围：字符型

默认值：空

须知

该参数不允许用户进行设置，建议保持默认值。

enable_resource_track

参数说明：设置是否开启资源实时监控功能。此参数需在 CN 和 DN 同时应用。

参数类型：SIGHUP

取值范围：布尔型

- on 表示打开资源监控。
- off 表示关闭资源监控。

默认值：on

enable_resource_record

参数说明：设置是否开启资源监控记录归档功能。开启时，对于执行结束的记录，会分别被归档到相应的 INFO 视图（14.3.78 GS_WLM_SESSION_INFO 和 14.3.75 GS_WLM_OPERARAOR_INFO）。此参数需在 CN 和 DN 同时应用。

参数类型：SIGHUP

取值范围：布尔型

- on 表示开启资源监控记录归档功能。
- off 表示关闭资源监控记录归档功能。

默认值：on

说明

新建集群默认值为 on，升级场景该参数的默认值为保持前向兼容维持原值。

enable_track_record_subsql

参数说明：设置是否开启子语句记录归档功能。开启时，存储过程、匿名块内部的子语句会被纪录归档到相应的 INFO 表（14.2.5 GS_WLM_SESSION_INFO）。此参数为会话级参数，可在与 CN 的连接会话中设置生效，仅影响该会话连接中的语句；也可在 CN 和 DN 上同时设置，能全局生效。

参数类型：USERSET

取值范围：布尔型

- on 表示开启子语句资源监控记录归档功能。
- off 表示关闭子语句资源监控记录归档功能。

默认值：off

enable_user_metric_persistent

参数说明：设置是否开启用户/资源池历史资源监控转存功能。开启时，对于 14.3.158 PG_TOTAL_USER_RESOURCE_INFO 视图中数据，会定期采样保存到 14.2.6 GS_WLM_USER_RESOURCE_HISTORY 系统表中；对于 14.3.60

GS_RESPOOL_RESOURCE_INFO 视图中数据，会定期采样保存到 14.2.2 GS_RESPOOL_RESOURCE_HISTORY 系统表中。

参数类型：SIGHUP

取值范围：布尔型

- on 表示开启用户/资源池历史资源监控转存功能。
- off 表示关闭用户/资源池历史资源监控转存功能。

默认值：on

user_metric_retention_time

参数说明：设置用户历史资源监控数据的保存天数。该参数仅在 enable_user_metric_persistent 为 on 时有效。

参数类型：SIGHUP

取值范围：整型，0～3650，单位为天。

- 值等于 0 时，用户历史资源监控数据将永久保存。
- 值大于 0 时，用户历史资源监控数据将保存对应天数。

默认值：7

enable_instance_metric_persistent

参数说明：设置是否开启实例资源监控转存功能。开启时，对实例的监控数据会保存到 14.2.3 GS_WLM_INSTANCE_HISTORY 系统表中。

参数类型：SIGHUP

取值范围：布尔型

- on 表示开启实例资源监控转存功能。
- off 表示关闭实例资源监控转存功能。

默认值：on

instance_metric_retention_time

参数说明：设置实例历史资源监控数据的保存天数。该参数仅在 enable_instance_metric_persistent 为 on 时有效。

参数类型：SIGHUP

取值范围：整型，0～3650，单位为天。

- 值等于 0 时，实例历史资源监控数据将永久保存。
- 值大于 0 时，实例历史资源监控数据将保存对应设置天数。

默认值：7

resource_track_level

参数说明：设置当前会话的资源监控的等级。该参数只有当参数 enable_resource_track 为 on 时才有效。

参数类型：USERSET

取值范围：枚举型

- none，不开启资源监控功能。
- query，开启 query 级别资源监控功能，开启此功能会把 SQL 语句的计划信息（类似 explain 输出信息）记录到 top SQL 中。
- perf，开启 perf 级别资源监控功能，开启此功能会把包含实际执行时间和执行行数的计划信息（类似 explain analyze 输出信息）记录到 top SQL 中。
- operator，开启 operator 级别资源监控功能，开启此功能不仅会把包含实际执行时间和执行行数的信息记录到 top SQL 中，还会把算子级别执行信息刷新到 top SQL 中。

默认值：query

resource_track_cost

参数说明：设置对当前会话的语句进行资源监控的最小执行代价。该参数只有当参数 enable_resource_track 为 on 时才有效。

参数类型：USERSET

取值范围：整型，-1～INT_MAX

- 值为-1 时，不进行资源监控。
- 值大于或等于 0 时，对执行代价超过该参数值的语句进行资源监控。

默认值：0

说明

新建集群默认值为 0，升级场景该参数的默认值为保持前向兼容维持原值。

resource_track_duration

参数说明：设置资源监控实时视图（参见表 10-1）中记录的语句执行结束后进行历史信息转存的最小执行时间。当执行完成的作业，其执行时间不小于此参数值时，作业信息会从实时视图（以 statistics 为后缀的视图）转存到相应的历史视图（以 history 为后缀的视图）中。该参数只有当 enable_resource_track 为 on 时才有效。

参数类型：USERSET

取值范围：整型，0～INT_MAX，单位为秒。

- 值为 0 时，资源监控实时视图（表 10-1）中记录的所有语句都进行历史信息归档。
- 值大于 0 时，资源监控实时视图（表 10-1）中记录的语句的执行时间超过这个值就会进行历史信息归档。

默认值：60s

dynamic_memory_quota

参数说明：自适应负载场景下，设置内存控制的比重，即可以使用系统最大可用内存的比例。

参数类型：SIGHUP

取值范围：整型，1～100

默认值：80

disable_memory_protect

参数说明：禁止内存保护功能。当系统内存不足时如果需要查询系统视图，可以先将此参数置为 on，禁止内存保护功能，保证视图可以正常查询。该参数只适用于在系统内存不足时进行系统诊断和调试，正常运行时请保持该参数配置为 off。

参数类型：USERSET

取值范围：布尔型

- on 表示禁止内存保护功能。
- off 表示启动内存保护功能。

默认值：off

query_band

参数说明：用于标示当前会话的作业类型，由用户自定义。

参数类型：USERSET

取值范围：字符型

默认值：空

enable_bbox_dump

参数说明：是否开启黑匣子功能，在系统不配置 core 机制的时候仍可产生 core 文件。此功能需要在 CN 和 DN 同时应用。

参数类型：SIGHUP

取值范围：布尔型

- on 表示打开黑匣子功能。
- off 表示关闭黑匣子功能。

默认值：off

enable_dynamic_workload

参数说明：是否开启动态负载管理功能。

参数类型：POSTMASTER

取值范围：布尔型

- on 表示打开动态负载管理功能。
- off 表示关闭动态负载管理功能。

默认值：on

须知

- 开启内存自适应后，收集统计信息后不再需要使用 work_mem 进行算子内存使用调优，由系统根据当前负载情况，为每个语句生成计划，并估算每个算子的内存使用量和整个语句的内存使用量。系统根据负载情况和整个语句内存使用量进行队列调度，所以多并发场景会出现语句排队的情况。
- 由于优化器行数估算不准现象的存在，会出现语句内存使用量低估或高估的情况。低估时，执行时内存会自动扩展。高估时，会导致系统内存利用不足，排队语句增多，可能导致性能非最优。此时需要识别语句估算内存远大于实际 DN 峰值内存的语句，通过设置 query_max_mem 进行调优，详见 11.3.6 SQL 调优关键参数调整。

bbox_dump_count

参数说明：在 bbox_dump_path 定义的路径下，允许存储的 GaussDB(DWS)所产生 core 文件最大数。超过此数量，旧的 core 文件会被删除。此参数只有当 enable_bbox_dump 为 on 时才生效。

参数类型：USERSET

取值范围：整型，1～20

默认值：8

说明

在并发产生 core 文件时，core 文件的产生个数可能大于 bbox_dump_count。

io_limits

参数说明：该参数 8.1.2 版本中已废弃，为兼容历史版本功能保留该参数，当前版本设置无效。

参数类型：USERSET

取值范围：整型，0～1073741823

默认值：0

io_priority

参数说明：该参数 8.1.2 版本中已废弃，为兼容历史版本功能保留该参数，当前版本设置无效。

参数类型：USERSET

取值范围：枚举型

- None
- Low
- Medium
- High

默认值：None

session_respool

参数说明：当前的 session 关联的 resource pool。

参数类型：USERSET

即如果先设置 cgroup_name，再设置 session_respool，那么 session_respool 关联的控制组起作用，如果再切换 cgroup_name，那么新切换的 cgroup_name 起作用。

切换 cgroup_name 的过程中如果指定到 Workload 控制组级别，数据库不对级别进行验证。级别的范围只要在 1-10 范围内都可以。

建议尽量不要混合使用 cgroup_name 和 session_respool。

取值范围：字符串，通过 create resource pool 所设置的资源池。

默认值：invalid_pool

enable_transaction_parctl

参数说明：是否管控事务块语句和存储过程语句。

参数类型：USERSET

取值范围：布尔型

- on 表示对事务块及存储过程语句进行管控。
- off 表示对事务块及存储过程语句不进行管控。

默认值：on

session_statistics_memory

参数说明：该参数 8.1.2 版本中已废弃，为兼容历史版本功能保留该参数，当前版本设置无效。

参数类型：SIGHUP

取值范围：整型，5MB～max_process_memory 的 50%。

默认值：5MB

session_history_memory

参数说明：设置历史查询视图的内存大小。

参数类型：SIGHUP

取值范围：整型，10MB～max_process_memory 的 50%。

默认值：100MB

topsql_retention_time

参数说明：设置历史 TopSQL 中 gs_wlm_session_info 和 gs_wlm_operator_info 表中数据的保存时间。

参数类型：SIGHUP

取值范围：整型，0~3650，单位为天。

- 值为 0 时，表示数据永久保存。
- 值大于 0 时，表示数据能够保存的对应天数。

默认值：30

注意

设置此 GUC 参数启用数据保存功能前，请先清理 gs_wlm_session_info 和 gs_wlm_operator_info 表中的数据。

transaction_pending_time

参数说明：当 enable_transaction_parctl 为 on 时，事务块语句和存储过程语句排队的最大时间。

参数类型：USERSET

取值范围：整型，-1～INT_MAX，单位为秒。

- 值为-1 或 0：事务块语句和存储过程语句无超时判断，排队至资源满足可执行条件。
- 值大于 0：事务块语句和存储过程语句排队超过所设数值的时间后，无视当前资源情况强制执行。

默认值：0

须知

此参数仅对存储过程及事务块的内部语句有效，即 14.3.118 PG_SESSION_WLMSTAT 中 enqueue 字段显示为 Transaction 或 StoredProc 的语句才会生效。

wlm_sql_allow_list

参数说明：用于指定资源管理 SQL 白名单语句，SQL 白名单语句不受资源管理监控。

参数类型：SIGHUP

取值范围：字符串

默认值：空

须知

- wlm_sql_allow_list 中可指定一条或多条 SQL 白名单语句，指定多条时，通过“;”进行分隔。
- 系统通过前置匹配判断 SQL 语句是否受监控，不区分大小写，例如：wlm_sql_allow_list='SELECT'，则所有 select 语句均不受资源管理监控。
- 识别参数值白名单字符串头部的空格，例如：'SELECT'与' SELECT'的含义是不一致的，' SELECT'只过滤头部带空格的 SELECT 语句。
- 系统默认部分 SQL 语句为白名单语句，默认白名单语句不可修改；可以通过系统视图 gs_wlm_sql_allow 查询默认和已经通过 GUC 设置成功的 SQL 白名单语句。
- 通过 wlm_sql_allow_list 指定的 SQL 语句不可追加，只能通过覆盖的方式设置；若需追加 SQL 语句，需要先查出原先指定的 GUC 值，在原值后面加补上新增的语句，以“;”分隔后重新设置。

16.14 自动清理

系统自动清理进程（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：控制是否启动数据库自动清理进程（autovacuum）。自动清理进程运行的前提是将 track_counts 设置为 on。

参数类型：SIGHUP

取值范围：布尔型

- on 表示开启数据库自动清理进程。
- off 表示关闭数据库自动清理进程。

默认值：on

说明

如系统在故障恢复后，需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进程。

须知

即使此参数设置为 off，数据库也会在需要防止事务 ID 回卷时发起清理进程。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 gaussdb），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）。

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。

参数类型：SIGHUP

取值范围：枚举类型

- analyze 表示只执行 autoanalyze。
- vacuum 表示只执行 autovacuum。
- mix 表示 autoanalyze 和 autovacuum 都执行。
- none 表示二者都不执行。

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。

参数类型：SIGHUP

取值范围：整型，0～2147483，单位为秒（s）。

默认值：5min

autovacuum_io_limits

参数说明：控制 autovacuum 进程每秒触发 IO 的上限。该参数 8.1.2 版本中已废弃，为兼容历史版本功能保留该函数，当前版本设置无效。

参数类型：SIGHUP

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。

参数类型：SIGHUP

例如：将 log_autovacuum_min_duration 设置为 250ms，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。

取值范围：整型，-1~INT_MAX，单位为毫秒（ms）。

- 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。
- 当参数设置为非-1 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。

参数类型：SIGHUP

取值范围：整型，0～128 。其中 0 表示不会自动进行 autovacuum。

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。

参数类型：SIGHUP

取值范围：整型，1～2147483 ，单位为秒（s）。

默认值：60s

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才会对这个表执行 VACUUM 操作。

参数类型：SIGHUP

取值范围：整型，0～INT_MAX

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。

参数类型：SIGHUP

取值范围：整型，0～INT_MAX

默认值：10000

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。

参数类型：SIGHUP

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。

参数类型：SIGHUP

取值范围：浮点型，0.0～100.0

默认值：0.25

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；即使自动清理进程被禁止，系统也会调用自动清理进程来防止循环重复。

参数类型：SIGHUP

取值范围：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。

参数类型：SIGHUP

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：2ms

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。

参数类型：SIGHUP

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

16.15 客户端连接缺省设置

16.15.1 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。

参数类型：USERSET

- 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：设置当前的模式。

参数类型：USERSET

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。

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字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。

参数类型：USERSET

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：设置当前的默认建表所在的 Node Group，目前只适用普通表。

参数类型：USERSET

取值范围：字符串

- installation，表示默认建表在安装的 Node Group 上。
- random_node_group，表示默认建表在随机选择的 NodeGroup 上，该配置 8.1.2 及以上版本支持，仅用于测试环境。
- roach_group，表示默认建表在所有节点上，该值为 roach 工具预留，不能用于其他场景。
- 取值为其他字符串，表示默认建表在设置的 Node Group 上。

默认值：installation

default_colversion

参数说明：设置当前默认创建列存表的存储格式版本。

参数类型：SIGHUP

取值范围：枚举类型

- 1.0 表示列存表的每列以一个单独的文件进行存储，文件名以 relfilenode.C1.0、relfilenode.C2.0、relfilenode.C3.0 等命名。
- 2.0 表示列存表的每列合并存储在一个文件中，文件名以 relfilenode.C1.0 命名。

默认值：2.0

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(DWS)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(DWS)将自动将当前的数据库设为默认的表空间。

参数类型：USERSET

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。

参数类型：USERSET

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。

参数类型：USERSET

取值范围：枚举型

- 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，GaussDB(DWS)不支持 read uncommitted，如果设置了 read uncommitted，实际上使用的是 read committed。
- 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：事务可序列化，GaussDB(DWS)不支持 SERIALIZABLE，如果设置了 serializable，实际上使用的是 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。

参数类型：SIGHUP

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(DWS)不支持可串行化的隔离级别，因此，该参数无实际意义。

参数类型：USERSET

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。

参数类型：USERSET

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。

参数类型：USERSET

取值范围：整型，0～2147483647，单位为毫秒（ms）。

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 替换事务 ID 的中断寿命(在同一个事务中)。

参数类型：USERSET

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 0 到 10 亿之间的任意值，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：5000000000

vacuum_freeze_table_age

参数说明：指定 VACUUM 对全表的扫描冻结元组的时间。如果表的 14.2.17 PG_CLASS.relfrozenxid 字段的值已经达到了参数指定的时间，VACUUM 对全表进行扫描。

参数类型：USERSET

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零到 20 亿之间的值，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 可以在对此表的反重叠自动清理启动之前运行。

默认值：15000000000

bytea_output

参数说明：设置 bytea 类型值的输出格式。

参数类型：USERSET

取值范围：枚举型

- 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。

参数类型：USERSET

取值范围：枚举型

- 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。

参数类型：USERSET

取值范围：枚举型

- 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。

参数类型：POSTMASTER

取值范围：整型，1～INT_MAX。

默认值：1000

gin_pending_list_limit

参数说明：设置当 GIN 索引启用 fastupdate 时，pending list 容量的最大值。当 pending list 的容量大于设置值时，会把 pending list 中数据批量移动到 GIN 索引数据结构中以进行清理。单个 GIN 索引可通过更改索引存储参数覆盖此设置值。

参数类型：USERSET

取值范围：整型，64～INT_MAX，单位为 KB。

默认值：4MB

16.15.2 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 。

参数类型：USERSET

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。

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)等。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。

参数类型：USERSET

取值范围：枚举型

- 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 numtodsinterval。

须知

IntervalStyle 参数也会影响不明确的间隔输入的说明。

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。

参数类型：USERSET

取值范围：字符串，可查询视图 14.3.155 PG_TIMEZONE_NAMES 获得。

默认值：PRC

说明

gs_initdb 将设置一个与其系统环境一致的时区值。

timezone_abbreviations

参数说明：设置服务器接受的时区缩写值。

参数类型：USERSET

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：调整浮点值显示的数据位数，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）。

参数类型：USERSET

取值范围：整型，-15～3

说明

- 设置为 3，表示包括部分有效的数据位。对转储需要精确恢复的浮点数据尤其有用。
- 设置为负数，表示摒弃不需要的数据位。

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。

参数类型：USERSET

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。

参数类型：SUSET

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。

参数类型：USERSET

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_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。

参数类型：USERSET

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_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。

参数类型：USERSET

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

default_text_search_config

参数说明：设置全文检索的配置信息。

如果设置为不存在的文本搜索配置时将会报错。如果 default_text_search_config 对应的文本搜索配置被删除，需要重新设置 default_text_search_config，否则会报设置错误。

- 其被文本搜索函数使用，这些函数并没有一个明确指定的配置。
- 当与环境相匹配的配置文件确定时，gs_initdb 会选择一个与环境相对应的设置来初始化配置文件。

参数类型：USERSET

取值范围：字符串

说明

GaussDB(DWS)支持 pg_catalog.english，pg_catalog.simple 两种配置。

默认值：pg_catalog.english

16.15.3 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 命令里面声明的名字没有目录部分时，系统将搜索这个目录以查找声明的文件。

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。当一个路径名字以特殊变量$libdir 为开头时，会替换为 GaussDB(DWS)发布提供的模块安装路径。例如：

```
dynamic_library_path = '/usr/local/lib/postgresql:/opt/testgs/lib:$libdir'
```

参数类型：SUSET

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

gin_fuzzy_search_limit

参数说明：设置 GIN 索引返回的集合大小的上限。

参数类型：USERSET

取值范围：整型，0～INT_MAX，0 表示没有限制。

默认值：0

16.16 锁管理

在 GaussDB(DWS)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。

- 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。
- 设置 log_lock_waits 时，这个选项也决定了在一个日志消息发出关于锁等待以前要等待的时间。当需要调查锁延迟时，请设置比正常 deadlock_timeout 更小的值。

参数类型：SUSET

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。

参数类型：SUSET

取值范围：整型，0 ~ INT_MAX，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。

参数类型：SUSET

取值范围：整型，0 ~ INT_MAX，单位为毫秒（ms）。

默认值：2min

max_locks_per_transaction

参数说明：控制每个事务能够得到的平均的对象锁的数量。

- 共享的锁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。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。只能在数据库启动的时候设置。
- 增大这个参数可能导致 GaussDB(DWS)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。
- 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会被允许。

参数类型：POSTMASTER

取值范围：整型，10 ~ INT_MAX

默认值：256

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(DWS)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。

参数类型：POSTMASTER

取值范围：整型，10 ~ INT_MAX

默认值：64

partition_lock_upgrade_timeout

参数说明：分区上的锁级别由允许读的 ExclusiveLock 升级到读写阻塞的 AccessExclusiveLock 时，会进行尝试性的锁升级，partition_lock_upgrade_timeout 指示了尝试锁升级的超时时间。

- 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。

参数类型：USERSET

取值范围：整型，-1 ~ 3000，单位为秒（s）。

默认值：1800

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。

参数类型：SIGHUP

取值范围：布尔型

- on 表示开启。
- off 表示关闭。

默认值：off

16.17 版本和平台兼容性

16.17.1 历史版本兼容性

GaussDB(DWS)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。

参数类型：USERSET

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。

参数类型：USERSET

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 。

参数类型：USERSET

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。

参数类型：USERSET

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。

参数类型：SUSET

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。

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。

参数类型：USERSET

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。

参数类型：USERSET

取值范围：布尔型

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。

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。

参数类型：USERSET

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。

参数类型：USERSET

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：控制开启某些功能受限的特性，例如 GDS 表关联操作。这些特性在历史版本未明确禁止，但某些场景功能上存在问题，不建议用户使用。

参数类型：USERSET

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

16.17.2 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。

参数类型：USERSET

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。

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

- 超长字符串自动截断功能不适用于 insert 语句包含外表的场景。
- 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。

参数类型：USERSET

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

16.18 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：控制终止会话。

参数类型：SUSET

取值范围：布尔型

- on 表示任何错误都会终止当前的会话。
- off 表示只有 FATAL 级别的错误才会终止会话。

默认值：off

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。

参数类型：USERSET

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

max_query_retry_times

参数说明：指定 SQL 语句出错自动重试功能的最大重跑次数（目前支持重跑的错误类型为“Connection reset by peer”、“Lock wait timeout”和“Connection timed out”等，设定为 0 时关闭重跑功能。

参数类型：USERSET

取值范围：整型，0~20

默认值：6

cn_send_buffer_size

参数说明：指定 CN 端数据发送数据缓存区的大小。

参数类型：POSTMASTER

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。

参数类型：SIGHUP

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。

参数类型：USERSET

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 CG003 CG004 F0011 45003

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。

参数类型：POSTMASTER

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

16.19 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。

min_pool_size

参数说明：CN 的连接池与其它某个 CN/DN 的最小连接数。

参数类型：POSTMASTER

取值范围：整型，1~65535

默认值：1

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。

参数类型：POSTMASTER

取值范围：整型，1~65535

默认值：CN 为 800， DN 为 5000

persistent_datanode_connections

参数说明：会话是否会释放获得的连接。

参数类型：USERSET

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。

参数类型：POSTMASTER

取值范围：整型，2~40

默认值：40

max_datanodes

参数说明：集群中 DN 的最大数目。

参数类型：POSTMASTER

取值范围：整型，2~65535

默认值：4096

cache_connection

参数说明：是否回收连接池的连接。

参数类型：SIGHUP

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。

参数类型：BACKEND

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

说明

会话连接参数，不建议用户设置。

enable_pooler_parallel

参数说明： CN 的连接池是否可以在并行的模式下进行连接。

参数类型：SIGHUP

取值范围：布尔型

- on 表示 CN 的连接池可以在并行的模式下进行连接。
- off 表示 CN 的连接池不可以在并行的模式下进行连接。

默认值：on

16.20 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。

参数类型：USERSET

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，GaussDB(DWS)不支持 read uncommitted，如果设置了 read uncommitted，实际上使用的是 read committed。
- 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：事务可序列化，GaussDB(DWS)不支持 SERIALIZABLE，如果设置了 serializable，实际上使用的是 repeatable read。

默认值：read committed

transaction_read_only

参数说明：设置当前事务是只读事务。

参数类型：USERSET

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：CN 节点为 off，DN 节点为 on。

xc_maintenance_mode

参数说明：设置系统进入维护模式。

参数类型：SUSET

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：off

须知

谨慎打开这个开关，避免引起集群数据不一致。

allow_concurrent_tuple_update

参数说明：设置是否允许并发更新。

参数类型：USERSET

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_backup_barrier

参数说明：指定是否为 GTM 启动点创建还原点。

参数类型：SUSET

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。

参数类型：SUSET

取值范围：整型， 0 ~ INT_MAX / 1000，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。目前，GaussDB(DWS)暂时未用到这个参数。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。

参数类型：USERSET

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

enforce_two_phase_commit

参数说明：为了兼容历史版本功能保留该参数，当前版本设置无效。

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。

参数类型：USERSET

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_retries

参数说明：控制 GTM 连接重试的次数。

参数类型：SIGHUP

取值范围：int，最小值为 1，最大值为 2147483647。

默认值：30

enable_redistribute

参数说明：节点不匹配时是否重新分配。

参数类型：SUSET

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

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。

对于要求强一致性读的 OLTP 场景或 OLAP 场景，建议不要开启该参数。GaussDB(DWS)不支持该特性，设置后无法生效。

参数类型：SUSET

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

16.21 开发人员选项

enable_light_colupdate

参数说明：控制是否使用列存轻量化 UPDATE。

参数类型：USERSET

取值范围：布尔型

- on 表示开启列存轻量化 UPDATE。
- off 表示关闭列存轻量化 UPDATE。

默认值：off

enable_fast_query_shipping

参数说明：控制查询优化器是否使用分布式框架。

参数类型：USERSET

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。

参数类型：USERSET

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。

参数类型：USERSET

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。

参数类型：USERSET

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。

参数类型：USERSET

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。

参数类型：USERSET

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_join_pseudoconst

参数说明：设置是否允许与伪常数进行 join。伪常数是指 join 两侧的变量都等于同一个常量。

参数类型：USERSET

取值范围：布尔型

- on 表示允许与伪常数进行 join。
- off 表示不允许与伪常数进行 join。

默认值：off

cost_model_version

参数说明：控制应用场景中估算时 cost 使用的模型。该参数的影响范围主要涵盖：表达式 distinct 估算、HashJoin 代价模型、行数估算、重分布时分布键的选择及 Aggregate 的行数估算等。

参数类型：USERSET

取值范围：0、1、2

- 0 表示使用原始的 cost 估算模型。
- 1 表示在 0 的基础上，使用增强的表达式 distinct 估算、HashJoin 代价模型、行数估算、重分布时分布键的选择及 Aggregate 的行数估算。
- 2 表示在 1 的基础上，使用随机性更优的 analyze 采样算法，以提高统计信息准确性。

默认值：1

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(DWS)的时候定义宏 USE_ASSERT_CHECKING（通过 configure 选项 --enable-cassert 完成）。

参数类型：USERSET

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(DWS)时，debug_assertions 缺省值为 on 。

默认值：off

distribute_test_param

参数说明：控制分布式测试框架打桩点是否生效。通常开发人员在进行故障注入测试时会在代码中预埋一些打桩点，使用唯一的名称进行标识，使用此参数可以控制代码中预埋的打桩点是否生效。参数采用逗号分隔的三元组形式，分别指定线程级别、测试桩名称和注入故障的错误级别。

参数类型：USERSET

取值范围：字符串，任一个已预埋的测试桩名称。

默认值：-1, default, default

ignore_checksum_failure

参数说明： 设置读取数据时是否忽略校验信息检查失败（但仍然会告警），继续执行。该参数仅在 enable_crc_check 为 on 时有效。继续执行可能导致崩溃，传播或隐藏损坏数据，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。

参数类型：SUSET

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

default_orientation

参数说明：创建表时，当不指定存储方式时，根据该 GUC 参数的值创建对应类型的表。使用时，各节点值需要保持一致。该参数仅 8.1.3 及以上集群版本支持。

参数类型：SUSET

取值范围：row, column, column enabledelta

- row 表示创建行存表。
- column 表示创建列存表。
- column enabledelta 表示创建开启 delta 表的列存表。

默认值：row

enable_colstore

参数说明：创建表时，当不指定存储方式时，默认创建为列存表。使用时，各节点值需要保持一致。属于测试参数，禁止用户启用。

参数类型：SUSET

取值范围：布尔型

默认值：off

enable_force_vector_engine

参数说明：对于支持向量化的执行器算子，如果其子节点是非向量化的算子，通过设置此参数为 on，强制生成向量化的执行计划。当打开 enable_force_vector_engine 开关时，无论是行存表、列存表或者是行列混存，如果 plantree 中不包含不支持向量化的场景，则强制走向量化执行引擎。

参数类型：USERSET

取值范围：布尔型

默认值：off

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。

参数类型：USERSET

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。

参数类型：USERSET

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：NULL

explain_perf_mode

参数说明：此参数用来指定 explain 的显示格式。

参数类型：USERSET

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(DWS)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

join_num_distinct

参数说明：控制应用场景中 Join 列或表达式的默认 distinct 值。

参数类型：USERSET

取值范围：双精度浮点型，大于或等于-100，客户端显示小数时可能会有截断。

- 值大于 0 时，表示使用该值作为默认 distinct 值。
- 值大于等于-100 且小于 0 时，表示估算默认 distinct 时使用的百分比。
- 值为 0 时，表示使用 200 作为默认 distinct 值。

默认值：-20

qual_num_distinct

参数说明：控制应用场景中过滤列或表达式的默认 distinct 值。

参数类型：USERSET

取值范围：双精度浮点型，大于或等于-100，客户端显示小数时可能会有截断。

- 值大于 0 时，表示使用该值作为默认 distinct 值。
- 值大于等于-100 且小于 0 时，表示估算默认 distinct 时使用的百分比。
- 值为 0 时，表示使用 200 作为默认 distinct 值。

默认值：200

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。

参数类型：USERSET

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。

参数类型：SIGHUP

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(DWS)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。

参数类型：USERSET

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(DWS)报告错误的损坏的页头，中止当前事务。

参数类型：SUSET

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。

说明

- 设置为 on 时，系统报告一个警告，把损坏的页面填充为零然后继续处理。该行为会破坏数据，即被损坏页面上的所有行。但是它允许绕开损坏页面然后从表中存在的未损坏页面上继续检索数据行。因此该参数在硬件或者软件错误导致的数据损坏中进行恢复是有作用的。通常不建议该参数设置为 on，除非不需要从损坏的页面中恢复数据。
- 对于列存表，会将整个 CU 跳过然后继续处理。支持的场景包括 crc 校验失败、magic 校验失败以及读取的 CU 长度错误。

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。

参数类型：POSTMASTER

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

replication_test

参数说明：此参数用于数据复制的内部功能测试。

参数类型：USERSET

取值范围：布尔型

- on 表示启用功能测试。
- off 表示关闭功能测试。

默认值：off

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，V300R002C00 版本开始，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。

当 cost_param & 16 不为 0，表示在计算两个及以上过滤条件或 Join 条件的综合选择率时，采用介于完全相关和完全不相关之间的一种模型，过滤条件较多时倾向于相关性较强的模型。

参数类型：USERSET

取值范围：整型，1～INT_MAX

默认值：16

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。

参数类型：USERSET

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

请谨慎调整该参数，调整该参数会修改内部数据类型转换规则并可能导致不可预期的行为。

nls_timestamp_format

参数说明：设置时间戳默认格式。

参数类型：USERSET

取值范围：字符串

默认值：DD-Mon-YYYY HH:MI:SS.FF AM

enable_partitionwise

参数说明：分区表连接操作是否选择智能算法。

参数类型：USERSET

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_partition_dynamic_pruning

参数说明：分区表扫描是否支持动态剪枝。

参数类型：USERSET

取值范围：布尔型

- on 表示分区表扫描开启动态剪枝。
- off 表示分区表扫描关闭动态剪枝。

默认值：on

max_user_defined_exception

参数说明：异常最大个数，默认值不可更改。

参数类型：USERSET

取值范围：整型

默认值：1000

datanode_strong_sync

参数说明：目前该参数已废弃，无实际意义。

参数类型：USERSET

取值范围：布尔型

- on 表示启用 stream 节点间强同步。
- off 表示不启用 stream 节点间强同步。

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。

参数类型：SIGHUP

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

enable_global_stats

参数说明：标识当前统计信息模式，为便于进行全局统计信息生成计划和单 DN 统计信息计划对比使用，默认创建为全局统计信息模式。属于测试参数，禁止用户启用。

参数类型：SUSET

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

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。

参数类型：USERSET

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

enable_row_fast_numeric

参数说明：标识行存表 numeric 数据落盘的格式。

参数类型：USERSET

取值范围：布尔型

- on/true 表示行存表 numeric 落盘格式为 bigint 格式。
- off/false 表示行存表 numeric 落盘格式为原始格式。

须知

参数值设置为 on 时，建议同步打开 enable_force_vector_engine，可提升大数据集 query 的查询性能。但相比于原始格式，大概率会占用更多磁盘空间。以 TPC-H 测试集为例，大约多占 7%空间（不同环境参考值可能有差异）。

默认值：off

rewrite_rule

参数说明：标识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
```

参数类型：USERSET

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（从主查询中下推条件到子查询）
- uniquecheck：使用 Unique Check 重写规则（允许目标列不含聚集函数的表达式子链接场景提升，需在子链接按关联列聚集后目标列值唯一才能开启，建议专业调优人员使用）。
- disablerep：使用禁止复制表的子链接提升规则（针对复制表禁止子链接提升）。
- projection_pushdown：使用 Projection Pushdown 重写规则（子查询中消除父查询不使用的列）。
- or_conversion：使用 OR 转换重写规则（消除执行效率低下的关联 OR 条件）。
- plain_lazyagg: 使用 Plain Lazy Agg 查询重写规则（消除单子查询中的聚集操作）。该选项仅 8.1.3.100 及以上集群版本支持。

默认值：magicset，or_conversion，projection_pushdown, plain_lazyagg

enable_compress_spill

参数说明：标识是否开启下盘压缩功能。

参数类型：USERSET

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

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。

参数类型：USERSET

取值范围：字符串

- LLVM_COMPILE 表示在 explain performance 显示界面中显示每个线程的 codegen 编译时间。
- 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：off(ALL)，不开启任何定位功能。

resource_track_log

参数说明：控制自诊断的日志级别。目前仅对多列统计信息进行控制。

参数类型：USERSET

取值范围：字符串

- summary：显示简略的诊断信息。
- detail：显示详细的诊断信息。

目前这两个参数值只在显示多列统计信息未收集的告警的情况下有差别，summary 不显示未收集多列统计信息的告警，detail 会显示这类告警。

默认值：summary

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：$[-1.04/2^{log2m*1/2},+1.04/2^{log2m*1/2}]$。

参数类型：USERSET

取值范围：整型，10~16。

默认值：11

hll_default_regwidth

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。具体对应关系可以参见表 16-2。

参数类型：USERSET

取值范围：整型，1~5。

默认值：5

表16-2 hll_default_log2m 和 hll_default_regwidth 与当前能计算的最大 distinct value 值的关系

log2m	regwidth = 1	regwidth = 2	regwidth = 3	regwidth = 4	regwidth = 5
10	7.4e+02	3.0e+03	4.7e+04	1.2e+07	7.9e+11
11	1.5e+03	5.9e+03	9.5e+04	2.4e+07	1.6e+12
12	3.0e+03	1.2e+04	1.9e+05	4.8e+07	3.2e+12
13	5.9e+03	2.4e+04	3.8e+05	9.7e+07	6.3e+12
14	1.2e+04	4.7e+04	7.6e+05	1.9e+08	1.3e+13
15	2.4e+04	9.5e+04	1.5e+06	3.9e+08	2.5e+13

hll_default_expthresh

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。

参数类型：USERSET

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

参数说明：该参数可用来指定是否默认开启 Sparse 模式。

参数类型：USERSET

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

参数说明：该参数可以用来指定 max_sparse 的大小。

参数类型：USERSET

取值范围：整型，-1~INT_MAX

默认值：-1

enable_compress_hll

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。

参数类型：USERSET

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

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。

参数类型：POSTMASTER

取值范围：整型，200*1024～max_process_memory，单位为 KB。

默认值：0.05 * max_process_memory

FencedUDFMemoryLimit

参数说明：控制每个 fenced udf worker 进程使用的虚拟内存。

参数类型：USERSET

设置建议：不建议设置此参数，可用 udf_memory_limit 代替。

取值范围：整数，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。

参数类型：POSTMASTER

设置建议：不建议设置此参数，可用 udf_memory_limit 代替。

取值范围：整数，可带单位（KB，MB，GB）。

默认值：1GB

pljava_vmoptions

参数说明：用户自定义设置 PL/Java 函数所使用的 JVM 虚拟机的启动参数。

参数类型：SUSET

取值范围：字符串， 支持：

- JDK8 JVM 启动参数
- JDK8 JVM 系统属性参数（以 -D 开头如 -Djava.ext.dirs）
- 用户自定义参数（以 -D 开头，如 -Duser.defined.option）

须知

如果用户在 pljava_vmoptions 中设置参数不满足上述取值范围，会在使用 PL/Java 语言函数时报错。

默认值：空

javaudf_disable_feature

参数说明：该参数用于控制 javaudf 行为的细粒度。

参数类型：SIGHUP

取值范围：字符串

- none，不禁用其他细粒度参数中指定的任何行为，当和其他参数一起设置时，none 失效。
- all，禁止执行所有 javaudf 函数，该选项设置后优先级最高。
- extdir，使在第三方路径放置依赖 jar 包的功能失效。
- hadoop，使 hadoop 相关功能失效。
- reflection，javaudf 函数执行过程中禁用反射(ReflectPermission 权限)。
- loadlibrary，javaudf 函数执行过程中禁止加载动态库(loadLibrary 权限)。
- net，javaudf 函数执行过程中禁用网络权限(NetPermission 权限)。
- socket，javaudf 函数执行过程中禁用 socket 套接字(SocketPermission 权限)。
- security，javaudf 函数执行过程中禁止 Security 配置修改(SecurityPermission 权限)。
- classloader，javaudf 函数执行过程中禁止自定义 classLoder(createClassLoader 权限)。
- access_declared_members，javaudf 函数执行过程中禁止读取其他类的内部成员(accessDeclaredMembers 权限)。

默认值：
extdir,hadoop,reflection,loadlibrary,net,socket,security,classloader,access_declared_members

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。

参数类型：USERSET

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。

参数类型：USERSET

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

checkpoint_flush_after

参数说明：设置 checkpointer 线程在连续写多少个磁盘页后会进行异步刷盘操作。GaussDB(DWS)中，磁盘页大小为 8KB。

参数类型：SIGHUP

取值范围：整型，0~256 （0 表示关闭异步刷盘功能）。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：32

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。

参数类型：USERSET

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

show_acce_estimate_detail

参数说明：在 GaussDB(DWS)集群使用加速集群场景下（即 acceleration_with_compute_pool 设置为 on），控制 explain 命令是否显示用于评估执行计划下推到加速集群的评估信息。评估信息一般用于运维人员在维护工作中使用，因此该参数默认关闭，此外为了避免这些信息干扰正常的 explain 信息显示，只有在 explain 命令的 verbose 选项打开的情况下才显示评估信息。

参数类型：USERSET

取值范围：布尔型

- on 表示可以在 explain 命令的输出中显示评估信息。
- off 表示不在 explain 命令的输出中显示评估信息。

默认值：off

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。

参数类型：SIGHUP

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

enable_immediate_interrupt

参数说明：控制是否允许在信号处理函数中立即中断当前语句或会话的执行。

参数类型：SIGHUP

取值范围：布尔型

- on 表示允许信号处理函数中立即中断语句或会话的执行。
- off 表示不允许信号处理函数中立即中断语句或会话的执行。

默认值：off

说明

请谨慎开启为 on，因为允许在信号函数中立即中断语句或会话的执行可能会导致某些关键流程执行被中断，且导致系统内部全局锁无法释放。建议该参数仅在系统调试过程中或规避故障时临时开启。

16.22 审计

16.22.1 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。

参数类型：SIGHUP

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_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。

参数类型：POSTMASTER

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。

参数类型：SIGHUP

取值范围：整型，1~INT_MAX/60，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。

参数类型：SIGHUP

取值范围：整型，1~1024，单位为 MB。

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。

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。

参数类型：SIGHUP

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。如果 audit_file_remain_time 设置过大，导致存储的审计日志所占磁盘空间达到 audit_space_limit，此时将删除最早的审计文件。

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。

参数类型：SIGHUP

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。

参数类型：SIGHUP

取值范围：整型，1024KB~1024GB，单位为 KB。

默认值：1GB

audit_file_remain_threshold

参数说明：审计目录下审计文件个数的最大值。

参数类型：SIGHUP

取值范围：整型，1~1048576

默认值：1048576

须知

请尽量确保此参数取值为 1048576，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。

16.22.2 操作审计

audit_operation_exec

参数说明：该参数决定是否审计 GaussDB(DWS)中各类执行成功的操作，由用户根据实际需求进行配置。

参数类型：SIGHUP

取值范围：字符串

- none：表示未配置审计项，如果同时配置了其他任何审计项，则 none 失效。
- all：表示对所有操作成功的场景进行审计。如果同时配置了其他任何审计项，则覆盖所有其他审计项的配置。注意，即使配置为 all，也不表示对所有的 DDL 操作进行审计，仍然需要结合 audit_system_object，对 DDL 操作的对象级别进行控制。
- login：表示对用户登录成功的场景进行审计。
- logout：表示对用户退出进行审计。
- database_process：表示对数据库启动、停止、切换、恢复操作进行审计。
- user_lock：表示对用户锁定和解锁成功的场景进行审计。
- grant_revoke：表示对用户权限授予和回收成功的场景进行审计。
- ddl：表示对 DDL 操作成功的场景进行审计，因为 DDL 操作由会根据操作对象进行更细粒度控制，仍然沿用审计开关 audit_system_object，即由 audit_system_object 控制对哪些对象的 DDL 操作进行审计（此处不配置 ddl，只要配置了 audit_system_object，审计也会生效）。
- select：表示对 select 操作成功的场景进行审计。
- copy：表示对 copy 操作成功的场景进行审计。
- userfunc：表示对用户自定义函数、存储过程、匿名块操作成功的场景进行审计。
- set：表示对 set 操作成功的场景进行审计。
- transaction：表示对事务操作成功的场景进行审计。
- vacuum：表示对 vacuum 操作成功的场景进行审计。
- analyze：表示对 analyze 操作成功的场景进行审计。
- explain：表示对 explain 操作成功的场景进行审计。
- specialfunc：表示对特殊函数调用操作成功的场景进行审计，特殊函数包括：pg_terminate_backend、pg_cancel_backend。
- insert：表示对 insert 操作成功的场景进行审计。
- update：表示对 update 操作成功的场景进行审计。
- delete：表示对 delete 操作成功的场景进行审计。
- merge：表示对 merge 操作成功的场景进行审计。
- show：表示对 show 操作成功的场景进行审计。
- checkpoint：表示对 checkpoint 操作成功的场景进行审计。
- barrier：表示对 barrier 操作成功的场景进行审计。

- cluster：表示对 cluster 操作成功的场景进行审计。
- comment：表示对 comment 操作成功的场景进行审计。
- cleanconn：表示对 cleanconnection 操作成功的场景进行审计。
- prepare：表示对 PREPARE、EXECUTE、DEALLOCATE 操作成功的场景进行审计。
- constraints：表示对 constraints 操作成功的场景进行审计。
- cursor：表示对游标操作成功的场景进行审计。

默认值：login, logout, database_process, user_lock, grant_revoke, set, transaction, cursor

须知

- 建议 transaction 审计项保留，否则事务内语句都不会被审计。
- 建议 cursor 审计项保留，否则 cursor 内 select 语句不会被审计。
- 需注意 Data Studio 客户端会自动给 select 语句封装 cursor。

audit_operation_error

参数说明：该参数决定是否审计 GaussDB(DWS)中各类执行失败的操作，由用户根据实际需求进行配置。

参数类型：SIGHUP

取值范围：字符串

- none：表示未配置审计项，如果同时配置了其他任何审计项，则 none 失效。
- syn_success：表示同步 audit_operation_exec 的配置，即配置了某操作执行成功场景的审计，则对应的执行失败场景也记入审计。需注意，配置了 syn_success 后，仍可以继续配置其他操作执行失败场景的审计；如果 audit_operation_exec 配置为 all，则所有的失败场景均记入审计；如果 audit_operation_exec 配置为 none，则 syn_success 等同于 none，即未配置审计项。
- parse：表示对用户输入命令解析失败场景进行审计，包含等待命令超时的失败场景。
- login：表示对用户登录失败的场景进行审计。
- user_lock：表示对用户锁定和解锁失败的场景进行审计。
- violation：表示对用户访问存在越权的场景进行审计。
- grant_revoke：表示对用户权限授予和回收失败的场景进行审计。
- ddl：表示对 DDL 操作失败的场景进行审计，因为 DDL 操作由会根据操作对象进行更细粒度控制，仍然需要结合 audit_system_object 的配置情况，所以此处配置 ddl 后，将对 audit_system_object 指定类型的 DDL 失败场景进行审计。
- select：表示对 SELECT 操作失败的场景进行审计。
- copy：表示对 COPY 操作失败的场景进行审计。
- userfunc：表示对用户自定义函数、存储过程、匿名块操作失败的场景进行审计。

- set：表示对 set 操作失败的场景进行审计。
- transaction：表示对事务操作失败的场景进行审计。
- vacuum：表示对 VACUUM 操作失败的场景进行审计。
- analyze：表示对 ANALYZE 操作失败的场景进行审计。
- explain：表示对 EXPLAIN 操作失败的场景进行审计。
- specialfunc：表示对特殊函数调用操作失败的场景进行审计，特殊函数包括：pg_terminate_backend、pg_cancel_backend。
- insert：表示对 INSERT 操作失败的场景进行审计。
- update：表示对 UPDATE 操作失败的场景进行审计。
- delete：表示对 DELETE 操作失败的场景进行审计。
- merge：表示对 MERGE 操作失败的场景进行审计。
- show：表示对 SHOW 操作失败的场景进行审计。
- checkpoint：表示对 CHECKPOINT 操作失败的场景进行审计。
- barrier：表示对 BARRIER 操作失败的场景进行审计。
- cluster：表示对 CLUSTER 操作失败的场景进行审计。
- comment：表示对 COMMENT 操作失败的场景进行审计。
- cleanconn：表示对 CLEANCONNECTION 操作失败的场景进行审计。
- prepare：表示对 PREPARE、EXECUTE、DEALLOCATE 操作失败的场景进行审计。
- constraints：表示对 CONSTRAINTS 操作失败的场景进行审计。
- cursor：表示对游标操作失败的场景进行审计。
- blacklist：表示对黑名单执行失败进行审计。

默认值：login

audit_inner_tool

参数说明：该参数决定是否审计 GaussDB(DWS)中内部维护工具的各类操作。

参数类型：SIGHUP

取值范围：布尔型

- on 表示审计来自内部维护工具的各类操作。
- off 表示不审计来自内部维护工具的各类操作。

默认值：off

audit_system_object

参数说明：该参数决定是否对 GaussDB(DWS)数据库对象的 CREATE、DROP、ALTER 操作进行审计。GaussDB(DWS)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。

参数类型：SIGHUP

取值范围：整型，0～4194303

- 0 代表关闭 GaussDB(DWS)数据库对象的 CREATE、DROP、ALTER 操作审计功能。
- 非 0 代表只审计 GaussDB(DWS)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 22 个二进制位的组合求出，这 22 个二进制位分别代表 GaussDB(DWS)的 22 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 22 个二进制位代表的具体审计内容请参见表 16-3。

默认值：12303

表16-3 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 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 对象的 CREATE、DROP、ALTER	• 0 表示不审计该对象的 CREATE、DROP 操作；

二进制位	含义	取值说明
	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。
第 9 位	是否审计 RESOURCE POOL 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 10 位	是否审计 WORKLOAD 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 11 位	是否审计 SERVER FOR HADOOP 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 12 位	是否审计 DATA SOURCE 对象的 CRAETE、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 13 位	是否审计 NODE GROUP 对象的 CREATE、DROP、ALTER 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。

二进制位	含义	取值说明
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。
第 19 位	是否审计 REDACTION POLICY 对象的 CREATE、DROP、ALTER 操作。	• 0 表示不审计 REDACTION POLICY 对象的 CREATE、DROP、ALTER 操作； • 1 表示审计 REDACTION POLICY 对象的 CREATE、DROP、ALTER 操作。
第 20 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 21 位	是否审计 NODE 对象的 CREATE、DROP、ALTER 操作。	• 0 表示不审计 NODE 对象的 CREATE、DROP、ALTER 操作； • 1 表示审计 NODE 对象的 CREATE、DROP、ALTER 操作。

enableSeparationOfDuty

参数说明：是否开启三权分立选项。

参数类型：POSTMASTER

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_grant_option

参数说明：是否允许安全模式下使用 with grant option 功能。

参数类型：SIGHUP

取值范围：布尔型

- on 表示允许安全模式下使用 with grant option 功能。
- off 表示不允许安全模式下使用 with grant option 功能。

默认值：off

enable_grant_public

参数说明：是否允许安全模式下使用 grant to public 功能。

参数类型：SIGHUP

取值范围：布尔型

- on 表示允许安全模式下使用 grant to public 功能。
- off 表示不允许安全模式下使用 grant to public 功能。

默认值：off

enable_copy_server_files

参数说明：是否开启 copy 服务器端文件的权限。

参数类型：POSTMASTER

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：true

须知

copy from/to file 要求具有系统管理员权限的用户才能使用，但是，在三权分立开启的状态下，系统管理员与初始用户的权限不同，可以通过使用 enable_copy_server_file 控制系统管理员的 copy 权限，避免系统管理员权限升级。

16.23 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。

参数类型：USERSET

取值范围：整型，最小值为 0，单位为秒。

默认值：5s

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。

参数类型：USERSET

取值范围：整型，最小值为 0，单位为秒。

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。

16.24 GTM 相关参数

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。

参数类型：SUSET

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 16-1。

默认值：warning

enable_alarm

参数说明：是否允许打开告警检测线程，检测数据库中可能的错误场景。

参数类型：POSTMASTER

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

16.25 其它选项

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。

参数类型：SUSET

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。

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。

参数类型：USERSET

取值范围：92～7999

默认值：512

enable_hadoop_env

参数说明：设置使用 Hadoop 特性时，是否允许在数据库中创建本地行存表和列存表。GaussDB(DWS)集群中，集群安装好后，该参数默认设为 off。以支持本地行列存储和跨集群访问 Hadoop 特性。不推荐用户调整 enable_hadoop_env 的值。

参数类型：USERSET

取值范围： 布尔型

- on/true，表示使用 Hadoop 特性时，不允许在数据库中创建本地行存表和列存表。
- off/false，表示使用 Hadoop 特性时，可以在数据库中创建本地行存表和列存表。

默认值：off

remote_read_mode

参数说明：设置当开启 enable_crc_check 为 on，主 DN 读取的数据校验失败后是否进行远程读的开关，以及是否采用安全认证方式连接。设置后需要重启集群才能生效。

参数类型：POSTMASTER

取值范围：off，non_authentication，authentication

- off，表示关闭远程读功能。
- non_authentication，表示采用非认证的方式连接备 DN 并获取数据。
- authentication，表示采用认证方式连接备 DN 并获取数据，重启集群前在 $GAUSSHOME/share/sslcert/grpc/目录下必须存在证书，否则无法启动集群。

默认值：non_authentication

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。

参数类型：USERSET

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 HDFS 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

job_queue_processes

参数说明：表示系统可以并发执行的 job 数目。

参数类型：POSTMASTER

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_jobs 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的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

ngram_gram_size

参数说明：ngram 解析器分词的长度。

参数类型：USERSET

取值范围：整型 ，1～4

默认值：2

ngram_grapsymbol_ignore

参数说明：ngram 解析器是否忽略图形化字符。

参数类型：USERSET

取值范围：布尔型

- on 表示忽略图形化字符。
- off 表示不忽略图形化字符。

默认值：off

ngram_punctuation_ignore

参数说明：ngram 解析器是否忽略标点符号。

参数类型：USERSET

取值范围：布尔型

- on 表示忽略标点符号。
- off 表示不忽略标点符号。

默认值：on

zhparser_dict_in_memory

参数说明：Zhparser 解析器是否将字典加载到内存中。

参数类型：POSTMASTER

取值范围：布尔型

- on 表示将字典加载到内存中
- off 表示不将字典加载到内存中

默认值：on

zhparser_multi_duality

参数说明：Zhparser 解析器设定是否将长词内的文字自动以二字分词法聚合。

参数类型：USERSET

取值范围：布尔型

- on 表示将长词内的文字自动以二字分词法聚合。
- off 表示不将长词内的文字自动以二字分词法聚合。

默认值：off

zhparser_multi_short

参数说明：Zhparser 解析器分词执行时是否执行针对长词复合切分。

参数类型：USERSET

取值范围：布尔型

- on 表示执行针对长词复合切分。
- off 表示不执行针对长词复合切分。

默认值：on

zhparser_multi_zall

参数说明：Zhparser 解析器是否将全部单字单独显示。

参数类型：USERSET

取值范围：布尔型

- on 表示将全部单字单独显示。

- off 表示不将全部单字单独显示。

默认值：off

zhparser_multi_zmain

参数说明：Zhparser 解析器是否将重要单字单独显示。

参数类型：USERSET

取值范围：布尔型

- on 表示将重要单字单独显示。
- off 表示不将重要单字单独显示。

默认值：off

zhparser_punctuation_ignore

参数说明：Zhparser 解析器分词结果是否忽略所有的标点等特殊符号（不会忽略\r 和 \n）。

参数类型：USERSET

取值范围：布尔型

- on：忽略所有的标点等特殊符号。
- off：不忽略所有的标点等特殊符号。

默认值：on

zhparser_seg_with_duality

参数说明：Zhparser 解析器是否将闲散文字自动以二字分词法聚合。

参数类型：USERSET

取值范围：布尔型

- on 表示将闲散文字自动以二字分词法聚合。
- off 表示不将闲散文字自动以二字分词法聚合。

默认值：off

acceleration_with_compute_pool

参数说明：在查询包含 OBS 时，通过该参数决定查询是否通过计算资源池进行加速。

参数类型：USERSET

取值范围：布尔型

- on 表示包含有 OBS 的查询在计算资源池可用时，会根据代价评估决定是否通过计算资源池对查询加速。
- off 表示任何查询都不会通过计算资源池进行加速。

默认值：off

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。

参数类型：USERSET

取值范围：字符串

默认值：升级场景下保持前向兼容，即与升级前的集群中该参数的默认值保持一致。新安装集群场景下，该参数默认值为 check_function_conflicts，以防止用户定义错误的函数属性导致严重的问题。

说明

- 当前只支持表 16-4。
- 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';
- 此参数选项中 strict_concat_functions 和 strict_text_concat_td 不能同时设置。

表16-4 兼容性配置项

兼容性配置项	兼容性行为控制	适用兼容模式
display_leading_zero	浮点数显示配置项。 • 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，0.25 显示为.25。 • 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，0.25 显示为 0.25。	ORA TD

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 的月份和为 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，
	`select add_months('2018-02-28',3) from dual;`
	`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
		ORA
		TD

兼容性配置项	兼容性行为控制	适用兼容模式

	如，
	`select add_months('2018-02-28',3) from dual;`
	`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。	ORA TD MySQL
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。	ORA TD MySQL
bind_procedure_searchpath	未指定模式名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。	ORA TD MySQL
correct_to_number	控制 to_number()结果兼容性的配置项。 若设置此配置项，则 to_number()函数结果与 PG11 保持一致，否则默认与 Oracle 保持一致。	ORA

unbind_divide_bound	控制对整数除法的结果进行范围校验。
	• 不设置此配置项时，将对除法结果进行校验，超出范围则报错。例如，示例中 INT_MIN/(-1)会因为超过结果大于 INT_MAX 而报越界错误：
	`SELECT (-2147483648)::int / (-1)::int;`
	`ERROR:  integer out of range`
	• 若设置此配置项，则不需要对除法结果进行范围校验。例如，示例中 INT_MIN/(-1)可以得到输出结果 INT_MAX+1：
	`SELECT (-2147483648)::int / (-1)::int;`
	`  ?column?`
	`------------`
	` 2147483648`
	`(1 row)`
		ORA
		TD

merge_update_	控制 merge into 匹配多行时是否进行 update 操作。	ORA

兼容性配置项	兼容性行为控制	适用兼容模式
multi	若设置此配置项，匹配多行时 update 不报错，否则默认与 Oracle 保持一致，报错。	TD
disable_row_update_multi	控制行存表 update 匹配多行时是否进行 update 操作。 若设置此配置项，匹配多行时 update 报错，否则默认可以进行多行匹配更新。	ORA TD

return_null_string	控制函数 lpad()、rpad()、repeat()、regexp_split_to_table()和 split_part()的结果为空字符串''的显示配置项。
	• 不设置此配置项时，空字符串显示为 NULL。
	`select length(lpad('123',0,'*')) from dual;`
	`length`
	`--------`
	
	`(1 row)`
	• 设置此配置项时，空字符串显示为''。
	`select length(lpad('123',0,'*')) from dual;`
	`length`
	`--------`
	`0`
	`(1 row)`
		ORA

compat_concat_variadic	控制函数 concat()和 concat_ws()对 variadic 类型结果兼容性的配置项。 若设置此配置项，当 concat 函数参数为 variadic 类型时，保留 Oracle 和 Teradata 兼容模式下不同的结果形式；否则默认 Oracle 和 Teradata 兼容模式下结果相同，且与 Oracle 保持一致。	ORA TD
convert_string_digit_to_numeric	控制 CHAR 类型和 INT 类型进行二元 BOOL 运算时类型转换优先级的配置项。 • 不设置此配置项时，类型转换优先级与 PG9.6 一致。 • 设置此配置项时，所有 CHAR 类型和 INT 类型的二元 BOOL 运算均强制转换为 NUMERIC 类型进行计算。 设置此配置项后会被影响的 CHAR 类型包括 BPCHAR、VARCHAR、NVARCHAR2、TEXT 四种类型，会被影响的 INT 类型包括 INT1、INT2、INT4、INT8 四种类型。 注意 此配置项只对二元 BOOL 运算生效，例如，INT2>TEXT、INT4=BPCHAR，非 BOOL 运算不会受到影响，该配置项暂不支持 INT>'1.1'这类 UNKNOWN 类型运算的转换。由于该	ORA TD MySQL

兼容性配置项	兼容性行为控制	适用兼容模式
	配置项开启后，CHAR 类型与 INT 类型的 BOOL 运算会优先转换为 NUMERIC 类型进行计算，因此会影响数据库计算性能，当 JOIN 列为受影响的类型组合时，还会影响执行计划。	

check_function_conflicts	控制是否检查自定义 plpgsql/SQL 函数的属性。
	• 不设置此配置项时，不检查自定义函数的 IMMUTABLE/STABLE/VOLATILE 属性。
	• 设置此配置项时，会检查自定义函数的 IMMUTABLE 属性，如果函数中含有表，或者是有 STABLE/VOLATILE 函数时，在执行时会报错。因为函数中如果有表或者 STABLE/VOLATILE 函数时，与函数定义中的 IMMUTABLE 属性冲突，即这种场景下，函数的行为非 IMMUTABLE。
	例如：设置此参数时，以下场景下会执行报错：
	`CREATE OR replace FUNCTION sql_immutable (INTEGER)`
	`RETURNS INTEGER AS 'SELECT a+$1 from`
	`shipping schema.t4 where a=1;'`
	`LANGUAGE SQL IMMUTABLE`
	`RETURNS NULL`
	`ON NULL INPUT;`
	`select sql immutable(1);`
	`ERROR: IMMUTABLE function cannot contain SQL`
	`statements with relation or Non-IMMUTABLE function.`
	`CONTEXT: SQL function "sql immutable" during`
	`startup`
	`referenced column: sql_immutable`
		ORA
		TD
		MySQL

varray_verification	控制是否校验数组长度以及数组类型长度。用于兼容 GaussDB(DWS) 8.1.0 之前的版本。
	若设置此配置项，不会校验数组长度以及数组类型长度。
	`-- 场景 1`
	`CREATE OR REPLACE PROCEDURE varray verification`
	`AS`
	`TYPE org varray type IS varray(5) OF`
	`VARCHAR2(2);`
	`v org varray org varray type;`
	`BEGIN`
	`v org varray(1) := '111'; --例如赋值已经超过了`
	`VARCHAR2(2)的限制，配置该选项后将和历史版本保持一致不进行校验`
	`END;`
	`/`
	`--场景 2`
	`CREATE OR REPLACE PROCEDURE varray_verification_i3_1`
		ORA
		TD

兼容性配置项	兼容性行为控制	适用兼容模式

	AS
	TYPE org_varray_type IS varray(2) OF NUMBER(2);
	v_org_varray org_varray_type;
	BEGIN
	v_org_varray(3) := 1; --例如赋值已经超过了 varray(2) 的数组长度限制，配置该选项后将和历史版本保持一致不进行校验
	END;
	/	

strict_concat_functions	控制函数 textanycat()和 anytextcat()在参数存在空值时，对返回值兼容性的配置项。此参数不能和 strict_text_concat_td 同时设置。 MySQL 兼容模式下，此参数无影响。 • 不设置此配置项时，函数 textanycat()和 anytextcat()的返回值默认与 Oracle 保持一致。 • 设置此配置项时，若函数 textanycat()和 anytextcat()的参数存在空值，则返回值也为空值，保留与 Oracle 和 Teradata 兼容模式下不同的结果。 例如，不设置此配置项时，函数 textanycat()和 anytextcat()的返回值与 Oracle 保持一致： SELECT textanycat('gauss', cast(NULL as BOOLEAN)); textanycat ------------ gauss (1 row) SELECT 'gauss' || cast(NULL as BOOLEAN); --这种情况下，||运算符会被转换为函数 textanycat ?column? ---------- gauss (1 row) 设置此配置项时，保留与 Oracle 和 Teradata 兼容模式下不同的结果： SELECT textanycat('gauss', cast(NULL as BOOLEAN)); textanycat ------------ (1 row) SELECT 'gauss' || cast(NULL as BOOLEAN); --这种情况下，||运算符会被转换为函数 textanycat ?column? ---------- (1 row)	ORA TD
strict_text_conc	Teradata 兼容模式下，控制函数 textcat()、textanycat()	TD

兼容性配置项	兼容性行为控制	适用兼容模式

at_td	和 anytextcat()在参数存在空值时，对返回值兼容性的配置项。此参数不能和 strict_concat_functions 同时设置。
	• 不设置此配置项时，Teradata 兼容模式下函数 textcat()、textanycat()和 anytextcat()的返回值与 GaussDB(DWS)一致。
	• 设置此配置项时，若 Teradata 兼容模式下函数 textcat()、textanycat()和 anytextcat()的参数存在空值，则返回值为空值。
	例如，不设置此配置项时，函数 textcat()、textanycat()和 anytextcat()的返回值与 GaussDB(DWS)保持一致：
	`td_compatibility_db=# SELECT textcat('abc', NULL);`
	`textcat`
	`---------`
	`abc`
	`(1 row)`
	`td_compatibility_db=# SELECT 'abc' || NULL; --这种情况下，||运算符会被转换为函数 textcat()`
	`?column?`
	`----------`
	`abc`
	`(1 row)`
	设置此配置项时，若函数 textcat()、textanycat()和 anytextcat()的返回值有空值，则返回 NULL：
	`td_compatibility_db=# SELECT textcat('abc', NULL);`
	`textcat`
	`---------`
	
	`(1 row)`
	`td_compatibility_db=# SELECT 'abc' || NULL;`
	`?column?`
	`----------`
	
	`(1 row)`	

compat_display_ref_table	设置视图中列的显示格式。
	• 不设置该选项时默认带前缀，即 tab.col 的格式。
	• 设置该选项时与原始定义一致，原始定义带前缀则显示，否则不显示。
	`SET`
	`behavior_compat_options='compat_display_ref_table';`
	`CREATE OR REPLACE VIEW viewtest2 AS SELECT a.c1, c2,`
	`a.c3, 0 AS c4 FROM viewtest tbl a;`
	`SELECT pg get viewdef('viewtest2');`
	`pg get viewdef`
	`----------`
		ORA
		TD

兼容性配置项	兼容性行为控制	适用兼容模式
	- SELECT a.c1, **c2**, a.c3, 0 AS c4 FROM viewtest_tbl a; (1 row)	
para_support_set_func	列存表中控制函数 COALESCE()、NVL()、GREATEST()、LEAST()入参是否支持多结果集表达式。 • 不设置此配置项时，函数入参包含多结果集表达式时，直接报错不支持。 SELECT COALESCE(regexp_split_to_table(c3,'#'), regexp_split_to_table(c3,'#')) FROM regexp_ext2_tbl1 ORDER BY 1 LIMIT 5; ERROR: set-valued function called in context that cannot accept a set • 设置此配置项时，支持函数入参包含多结果集表达式。 SELECT COALESCE(regexp_split_to_table(c3,'#'), regexp_split_to_table(c3,'#')) FROM regexp_ext2_tbl1 ORDER BY 1 LIMIT 5; coalesce ---------- a a a a a (5 rows)	ORA TD
disable_select_truncate_parallel	控制分区表的 truncate 等 ddl 的锁等级。 • 设置此配置项时，将禁止分区表的不同分区上 truncate 与 DML(如 select)的并发，允许分区表上 select 的 FQS(快速下发)。在 OLTP 场景下分区表上的简单查询较多，并且没有分区表不同分区 truncate 与 DML 并发的需求，可以考虑设置此配置项。 • 不设置此配置项时，分区表上不同分区的 select 与 truncate 可以并发进行，同时关闭分区表的 FQS（快速下发）来避免可能的不一致问题。	ORA TD MySQL
bpchar_text_without_rtrim	Teradata 兼容模式下，设置此参数时，控制 bpchar 到 text 转换保留右侧空格，如果实际长度不足 bpchar 指定的长度，对其进行补空格操作，兼容 Teradata 对 bpchar 字符串的处理风格。 当前不支持“比较字符串时忽略尾部空格”，拼接后结果如果存在尾部空格，进行比较时会对空格敏感。 例如，设置参数时： td_compatibility_db=# select	TD

兼容性配置项	兼容性行为控制	适用兼容模式

	length('a'::char(10)::text);
	length
	-------- 	
	10
	(1 row)
	
	td_compatibility_db=# select
	length('a'||'a'::char(10));
	length
	-------- 	
	11
	(1 row)	

convert_empty_str_to_null_td	Teradata 兼容模式下，设置此参数时，控制 to_date, to_timestamp 和 to_number 类型转换函数处理空串时，返回 null；同时控制 to_char 函数处理 date 类型入参时返回的格式。
	例如：
	未设置此参数时：
	td_compatibility_db=# select to_number('');
	to_number
	----------- 	
	0
	(1 row)
	
	td_compatibility_db=# select to_date('');
	ERROR: the format is not correct
	DETAIL: invalid date length "0", must between 8 and 10.
	CONTEXT: referenced column: to_date
	
	td_compatibility_db=# select to_timestamp('');
	to timestamp
	----------------------- 	
	0001-01-01 00:00:00 BC
	(1 row)
	
	td_compatibility_db=# select to_char(date '2020-11-16');
	to_char
	----------------------- 	
	2020-11-16 00:00:00+08
	(1 row)
	设置此参数，若 to_number, to_date, to_timestamp 函数的参数有空串时：
	td_compatibility_db=# select to_number('');
	to_number
	----------- 	
	
	(1 row)
		TD

兼容性配置项	兼容性行为控制	适用兼容模式

	td_compatibility_db=# select to_date('');
	 to_date
	--------- 	
	
	(1 row)
	
	td_compatibility_db=# select to_timestamp('');
	 to_timestamp
	-------------- 	
	
	(1 row)
	
	td_compatibility_db=# select to_char(date '2020-11-16');
	 to_char
	------------ 	
	 2020/11/16
	(1 row)	

disable_case_specific	控制字符类型匹配时是否忽略大小写。仅在 Teradata 兼容模式下生效。 • 不设置此配置项时，字符类型匹配时，字符的大小写敏感。 • 设置此配置项时，字符类型匹配时，字符的大小写不敏感。 设置此配置项后会影响的字符类型包括 CHAR、TEXT、BPCHAR、VARCHAR、NVARCHAR 五种类型，会被影响的操作符包括<、>、=、>=、<=、!=、<>、!=、like、not like、in、not in 共 12 种操作符以及 case when、decode 表达式。 注意 由于该配置项开启后，字符类型前会增加 UPPER 函数进而会影响估算逻辑，需要使用增强的估算模型。(建议设置：cost_param=16、cost_model_version = 1、join_num_distinct=-20、qual_num_distinct=200)	TD
enable_interval_to_text	控制 interval 到 text 类型的隐式转换功能。 • 设置此选项时，支持 interval 类型到 text 类型的隐式转换。 SELECT TO DATE('20200923', 'yyyymmdd') - TO DATE('20200920', 'yyyymmdd') = '3'::text; ?column? ---------- f (1 row) • 不设置此选项时，不支持 interval 类型到 text 类型	ORA TD MySQL

兼容性配置项	兼容性行为控制	适用兼容模式

	的隐式转换。
	`SELECT TO_DATE('20200923', 'yyyymmdd') - TO_DATE('20200920', 'yyyymmdd') = '3'::text;`
	`?column?`
	`----------`
	`t`
	`(1 row)`	

case_insensitive	MySQL 兼容模式下，设置此参数，控制 locate，strpos，instr 字符串函数入参大小写不敏感。
	目前默认未设置该参数，即入参大小写敏感。
	例如：
	• 未设置此选项时，入参大小写敏感。
	`mysql_compatibility_db=# SELECT LOCATE('sub', 'Substr');`
	` locate`
	`--------`
	`0`
	`(1 row)`
	• 设置此选项时，入参大小写不敏感。
	`mysql_compatibility_db=# SELECT LOCATE('sub', 'Substr');`
	` locate`
	`--------`
	`1`
	`(1 row)`
		MySQL

inherit_not_null_strict_func	控制函数原有的 strict 属性，参数为 1 个的函数可以传递 NOT NULL 属性的行为。即：对于 func(x)，如果 func()为 strict 属性，且 x 包含 NOT NULL 约束，则认为 func(x)也是包含 NOT NULL 约束的。 该兼容配置项在某些优化场景，例如：NOT IN 优化、COUNT(DISTINCT)优化，会有特定的优化效果，但特定场景可能导致结果错误。 目前默认未设置该参数，保证结果正确，但可能导致性能回退，如果出现问题可设置该参数回退到历史版本行为。	ORA TD MySQL
disable_compat_minmax_expr_mysql	MySQL 兼容模式下，控制 greatest/least 表达式对 null 入参的处理方式。 默认兼容 MySQL。可通过设置此参数，回退到历史版本行为。 • 不设置此选项时，兼容 MySQL 行为，入参为 null 时返回 null。 `mysql_compatibility_db=# SELECT greatest(1, 2, null), least(1, 2, null);`	MySQL

兼容性配置项	兼容性行为控制	适用兼容模式
	greatest | least ----------+------- | (1 row) • 设置此选项时，返回非 null 参数中的最大/小值。 mysql_compatibility_db=# SELECT greatest(1, 2, null), least(1, 2, null); greatest | least ----------+------- 2 | 1 (1 row)	

disable_compat_substr_mysql	MySQL 兼容模式下，控制 substr/substring 函数在起始位置 pos <= 0 时的行为。
	默认兼容 MySQL。可通过设置此参数，回退到历史版本行为。
	• 不设置此选项时，兼容 MySQL 行为，即 pos = 0 时返回空串，pos < 0 时从倒数第 |pos| 个位置开始截取字符。
	mysql_compatibility_db=# SELECT substr('helloworld',0);
	substr
	-------- 	
	
	(1 row)
	mysql compatibility db=# SELECT substring('helloworld',0),substring('helloworld',-2,4);
	substring | substring
	-----------+-----------
	| ld
	(1 row)
	• 设置此选项时，pos <= 0 时仍然从左侧开始截取字符。
	mysql_compatibility_db=# SELECT substr('helloworld',0);
	substr
	------------ 	
	helloworld
	(1 row)
	mysql compatibility db=# SELECT substring('helloworld',0),substring('helloworld',-2,4);
	substring | substring
	------------+-----------
	helloworld | h
	(1 row)
		MySQL

disable_compat	MySQL 兼容模式下，控制 trim/ltrim/rtrim 函数对入参	MySQL

兼容性配置项	兼容性行为控制	适用兼容模式
_trim_mysql	的处理方式。 默认兼容 MySQL。可通过设置此参数，回退到历史版本行为。 • 不设置此选项时，兼容 MySQL 行为，匹配完整子串。 `mysql_compatibility_db=# SELECT trim('{}{name}{}','{}'),trim('xyznamezyx','xyz');` ` btrim	btrim` `--------+---------` ` {name}	namezyx` `(1 row)` • 设置此选项时，匹配字符集合中的单个字符。 `mysql_compatibility_db=# SELECT trim('{}{name}{}','{}'),trim('xyznamezyx','xyz');` ` btrim	btrim` `-------+-------` ` name	name` `(1 row)`	
light_object_mtime	控制 pg_object 系统表 mtime 字段是否会记录对象行为的操作。 • 设置此选项时，GRANT/REVOKE/TRUNCATE 操作不被 mtime 记录即不更新 mtime 字段。 • 不设置此选项时（默认行为），ALTER 操作、COMMENT、GRANT/REVOKE 和 TRUNCATE 均会被 mtime 记录即更新 mtime 字段。	ORA TD MySQL

disable_including_all_mysql	MySQL 兼容模式下，控制 CREATE TABLE ... LIKE 语法是否为 INCLUDING_ALL 模式。
	默认不设置此参数，即 MySQL 兼容模式下，CREATE TABLE ... LIKE 语法默认为 INCLUDING_ALL 模式。
	可通过设置此参数，回退到历史版本行为。
	• 不设置此选项，MySQL 兼容模式下，CREATE TABLE ... LIKE 语法为 INCLUDING_ALL 模式。
	`mysql_compatibility_db=# create table mysql_like(id int, name varchar(10), score int) distribute by hash(id) COMMENT 'mysql_like';`
	`CREATE TABLE`
	`mysql_compatibility_db=# create index index_like on mysql_like(name);`
	`CREATE INDEX`
	`mysql_compatibility_db=# \d+ mysql_like;`
	`Table "public.mysql_like"`
	` Column
		Type	Modifiers	Storage	Stats target	Description`	MySQL

兼容性配置项	兼容性行为控制	适用兼容模式
	--------+----------------------+-----------+------- ---+-------------+------------- id	integer		plain | | name	character varying(10)		extended | | score	integer		plain | | Indexes: "index_like" btree (name) TABLESPACE pg_default Has OIDs: no Distribute By: HASH(id) Location Nodes: ALL DATANODES Options: orientation=row, compression=no mysql_compatibility_db=# create table copy_like like mysql_like; CREATE TABLE mysql_compatibility_db=# \d+ copy_like; Table "public.copy_like" Column | Type | Modifiers | Storage | Stats target | Description --------+----------------------+-----------+------- ---+-------------+------------- id	integer		plain | | name	character varying(10)		extended | | score	integer		plain | | Indexes: "copy_like_name_idx" btree (name) TABLESPACE pg_default Has OIDs: no Distribute By: HASH(id) Location Nodes: ALL DATANODES Options: orientation=row, compression=no • 设置此选项，MySQL 兼容模式下，CREATE TABLE ... LIKE 语法为空模式。 mysql_compatibility_db=# set behavior_compat_options = 'disable_including_all_mysql'; SET mysql_compatibility_db=# create table mysql_copy like mysql_like; NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using round-robin as the distribution mode by default. HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column. CREATE TABLE mysql_db=# \d+ mysql_copy;	

兼容性配置项	兼容性行为控制	适用兼容模式
	Table "public.mysql_copy" Column | Type | Modifiers | Storage | Stats target | Description --------+----------------------+-----------+----------+--------------+------------- id | integer | | plain | | name | character varying(10) | | extended | | score | integer | | plain | | Has OIDs: no Distribute By: ROUND ROBIN Location Nodes: ALL DATANODES Options: orientation=row, compression=no	
cte_onetime_inline	控制非 stream 计划是否执行 inline。 • 设置此选项时，非 stream 计划且只被引用一次的 CTE 执行 inline。 • 不设置此选项时，非 stream 计划且只被引用一次的 CTE 不执行 inline。	ORA TD MySQL
skip_first_after_mysql	MySQL 兼容模式下，控制是否在 ALTER TABLE ADD/MODIFY/CHANGE COLUMN 中忽略 FIRST/AFTER colname 语法。 • 设置此选项时，将忽略 FIRST/AFTER colname 语法，执行不报错。 mysql_compatibility_db=# set behavior_compat_options = 'skip_first_after_mysql'; mysql_compatibility_db=# alter table t1 add column b text after a; ALTER TABLE • 不设置此选项时，将不支持 FIRST/AFTER colname 语法，执行报错。 mysql_compatibility_db=# set behavior_compat_options = ''; mysql_compatibility_db=# alter table t1 add column b text after a; ERROR: FIRST/AFTER is not yet supported.	MySQL
enable_division_by_zero_mysql	MySQL 兼容模式下，除数为 0 时，控制除法或取余操作是否报错。（该配置项仅 8.1.3.110 及以上集群版本支持。） • 设置此选项时，除法或取余操作中除数为 0 时，返回 NULL。 compatible_mysql_db=# SET behavior_compat_options = 'mysql_enable_division_by_zero'; SET	MySQL

兼容性配置项	兼容性行为控制	适用兼容模式

	compatible_mysql_db=# select 1/0 as test;
	 test
	---------- 	
	
	(1 row)
	• 不设置此选项时，除法或取余操作中除数为 0 时，执行报错。
	compatible_mysql_db=# select 1/0;
	ERROR: division by zero	

table_skewness_warning_threshold

参数说明：设置用于表倾斜告警的阈值。

参数类型：SUSET

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。

参数类型：SUSET

取值范围：整型，0～INT_MAX

默认值：100000

max_cache_partition_num

参数说明：设置扩容重分布过程中列存节省内存模式的分区数目。如果超过分区数据目，则最早缓存的分区将直接写入列存文件中。

参数类型：SIGHUP

取值范围：整型，最小值为 0，最大值为 32767。

- 0 表示关闭列存节省内存模式。
- 1～32767 表示存分区表最多缓存的分区数目。

默认值：0

说明

该参数用于扩容重分布，合理设置可以缓解列存分区表重分布过程中的内存消耗。但某些分区数据分布非常不均衡的表在重分布完成后，可能会产生较多小 CU。如果出现较多小 CU，需要通过 VACUUM FULL 来合并小 CU。

enable_prevent_job_task_startup

参数说明：设置用于阻止 job 线程的启动。该参数属于系统内部参数，不建议用户修改设置。

参数类型：SIGHUP

取值范围：布尔型

- on 表示阻止启动 job 线程。当 job 周期到来时，不会启动 job 执行线程。
- off 表示允许启动 job 线程。当 job 周期到来时，会启动 job 执行线程，完成 job 中规定的操作。

默认值： off

说明

该参数只需在 CN 上设置。

auto_process_residualfile

参数说明：控制残留文件记录功能的开关。

参数类型：SIGHUP

取值范围：布尔型

- on 表示打开残留文件记录功能。
- off 表示关闭残留文件记录功能。

默认值： off

enable_view_update

参数说明：用于设置是否开启视图更新功能。

参数类型：POSTMASTER

取值范围：布尔型

- on 表示启用视图更新功能。
- off 表示关闭视图更新功能。

默认值： off

view_independent

参数说明：用于设置是否开启视图与表、函数、同义词的解耦功能。基表恢复后目前已支持自动关联重建。

参数类型：SIGHUP

取值范围：布尔型

- on 表示启用视图解耦功能，存在视图依赖的表、函数、同义词及其他视图可以单独删除（临时表及临时视图除外），关联视图保留但不可用。
- off 表示关闭视图解耦功能，存在视图依赖的表、函数、同义词及其他视图不可以单独删除，仅可使用 cascade 级联删除。

默认值： off

bulkload_report_threshold

参数说明：设置导入导出统计信息上报阈值。

参数类型：SIGHUP

取值范围：整型，0～INT_MAX

默认值：50

assign_abort_xid

参数说明：查询时将指定的 xid 判断为需要 abort 的事务。

参数类型：USERSET

取值范围：指定 xid 的字符串

⚠ 注意

此参数只用于用户误删数据(delete 操作)后进行快速恢复。其他场景禁止使用，否则造成事务可见性错误问题。

default_distribution_mode

参数说明：用于设置表的默认分布方式。该参数仅 8.1.2 及以上版本支持。

参数类型：USERSET

取值范围：枚举类型

- roundrobin，创建表不指定分布方式时，按如下规则选取默认分布方式：
  a. 若建表时包含主键/唯一约束，则选取 HASH 分布，分布列为主键/唯一约束对应的列。
  b. 若建表时不包含主键/唯一约束，则选取 ROUNDROBIN 分布。
- hash，创建表不指定分布方式时，按如下规则选取默认分布方式：
  a. 若建表时包含主键/唯一约束，则选取 HASH 分布，分布列为主键/唯一约束对应的列。
  b. 若建表时不包含主键/唯一约束，但存在数据类型支持作分布列的列，则选取 HASH 分布，分布列为第一个数据类型支持作分布列的列。

c. 若建表时不包含主键/唯一约束，也不存在数据类型支持作分布列的列，选取 ROUNDROBIN 分布。

默认值：roundrobin

说明

新建 8.1.2 集群版本默认值为 roundrobin，升级到 8.1.2 集群版本场景该参数的默认值为 hash。

object_mtime_record_mode

参数说明：用于设置 PG_OBJECT 系统表中 mtime 字段的更新行为。

参数类型：SIGHUP

取值范围：字符串

- default，表示默认行为包括 ALTER、COMMENT、GRANT/REVOKE 和 TRUNCATE 操作会更新 mtime 字段。
- none，表示不更新 mtime 字段。
- disable_acl，表示 GRANT/REVOKE 操作不更新 mtime 字段。
- disable_truncate，表示 TRUNCATE 操作不更新 mtime 字段。
- disable_partition，表示分区表相关 ALTER 操作不更新 mtime 字段。

默认值：default

17 术语表

术语	解释
A - E	
ACID	在可靠数据库管理系统（DBMS）中，事务（transaction）所应该具有的四个特性：原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、和持久性（Durability）。
安全环	每一个环都由若干物理机组成，环内的 DN 形成主、备、从备关系，不向环外延伸。也就是说，环内的任何一个节点的主，或者备，或从备，依然在环内。扩容与缩容时，是以环为最小单位进行的。
Bgwriter	数据库启动时创建的一个后台写线程，此线程用于将数据库中脏页面写入到持久性设备（例如磁盘）中。
bit	比特。计算机处理的最小的信息单位。比特用来表示二进制数字 1 或 0，或者一种逻辑条件真或假。在物理上，比特表示一个电路上高或低的电压点或者磁盘上的磁化单程或其它。一个单独的比特位所传达的信息很少有意义的。然而，一个 8 位组却构成了一个字节，可用于表示如一个英文字母，十进制数字，或其它字符等多种类型的信息。
Bloom Filter	布隆过滤器。由 Howard Bloom 在 1970 年提出的二进制向量数据结构，它具有很好的空间和时间效率，被用来检测一个元素是不是集合中的一个成员，这种检测只会对在集合内的数据错判，而不会对不是集合内的数据进行错判，这样每个检测请求返回有“在集合内（可能错误）”和“不在集合内（绝对不在集合内）”两种情况，可见 Bloom filter 是牺牲了正确率换取时间和空间。
CCN	Central Coordinator ，GaussDB(DWS)动态负载管理中心协调节点。负责进行各 CN 中复杂作业是否可以执行的中心判断、排队和调度，以实现动态负载管理。
CIDR	Classless Inter-Domain Routing，无类域间路由 IP 编址方案。CIDR 摒弃传统的基于类（A 类：8，B 类：16，C 类：24）的地址分配方式，允许使用任意长度的地址前缀，有效提高地址空间的利用率。

术语	解释
	CIDR 表示方法：IP 地址/网络 ID 的位数。比如 192.168.23.35/21，其中“21”表示前面地址中的前 21 位代表网络部分，其余位代表主机部分。
Cgroups	Control Groups，控制组（GaussDB(DWS)中也称之为优先级组）。SUSE Linux 和 RedHat 内核提供的一种可以限制、记录、隔离进程组所使用的物理资源的机制。
CLI	Command-line Interface，命令行界面。应用程序和用户交互的一种方式，完全基于文本输入和输出。命令通过键盘或类似装置输入，由程序编译并执行。结果是以文本或图形的方式呈现在终端界面。
CM	Cluster Manager，集群管理模块。管理和监控分布式系统中各个功能单元和物理资源的运行情况，确保整个系统的稳定运行。
CMS	Cluster Management Service，集群管理服务。是用于管理集群状态的部件。
CN	Coordinator，负责数据库系统元数据存储、查询任务的分解和部分执行，以及将 DN 中查询结果汇聚在一起。
CU	Compression Unit，压缩单元。列存表的最小存储单位。
core 文件	当程序出现内存越界、断言失败或者访问非法内存时，操作系统会中止进程，并将当前内存状态导出到 core 文件中，以便进一步分析。 core 文件包含内存转储，支持全二进制和指定端口格式。core 文件名称由字符串 core 以及操作系统进程 ID 组成。 core 文件不依赖于任何平台。
Core Dump	通常在程序异常终止时，核心转储（Core Dump）、内存转储或系统转储用于记录特定时间计算机程序工作内存的状态。实际上，其它关键程序的状态经常在同一时间进行转储，例如处理器寄存器，包括程序指标和栈指针、内存管理信息、其它处理器和操作系统标记及信息。Core Dump 经常用于辅助诊断和纠错计算机程序问题。
DBA	Database Administrator，数据库管理员。指导或执行所有和维护数据库环境相关的操作。
DBLINK	DBLINK 是定义一个数据库到另一个数据库路径的对象，通过它可以查询远程数据库对象。
DBMS	Database Management System，数据库管理系统。数据库管理系统是为了访问数据库中的信息而使用的一个管理系统软件。它包含一组程序使用户可以进入、管理、查询数据库中数据。基于真实数据的位置，可以分为内存数据库管理系统和磁盘数据库管理系统。
DCL	Data Control Language，数据控制语言。
DDL	Data Definition Language，数据定义语言。

术语	解释
DML	Data Manipulation Language，数据操纵语言。
DN	Data Node，和 CN 对应的概念。负责实际执行表数据的存储、查询操作。
ETCD	Editable Text Configuration Daemon，分布式键值存储系统，用于共享配置和服务发现（服务注册和查找）。
ETL	Extract-Transform-Load，描述将数据从来源端经过抽取（extract）、转换（transform）、加载（load）至目的端的过程。
Extension Connector	Extension Connector 是 GaussDB(DWS)提供的功能模块，使用它可以将 SQL 语句发送到集群外部的 Spark，并在当前库中返回执行结果，实现跨集群处理数据。
备份	备份件或者备份过程。指复制并归档计算机数据，当发生数据丢失事件时，可以用该复制并归档的数据来恢复原始数据。
备份和恢复	保护数据库防止由于媒介失效或人为错误造成的数据丢失过程中涉及的一组概念、过程及策略。
备机	GaussDB(DWS)双机方案中的一个节点，用于作为主机的备份，在主机异常时，备机会切换到主机状态，以确保能正常提供数据服务。
崩溃	崩溃（或系统崩溃）指计算机或程序（例如软件应用程序或操作系统）异常终止的事件。出现错误后，通常会自动退出。有时出现恶意程序冻结或挂起直到崩溃上报服务记录崩溃的详细信息。对于操作系统内核关键部分的程序，整个计算机可能瘫痪（可能造成致命的系统错误）。
编码	编码是指用代码来表示各组数据资料，使其成为可利用计算机进行处理和分析的信息。用预先规定的方法将文字、数字或其它对象编成数码，或将信息、数据转换成规定的电脉冲信号。
编码技术	呈现计算机软硬件识别的特定字符集数据的技术。
表	表是由行与列组合成的。每一列被当作是一个字段。每个字段中的值代表一种类型的数据。例如，一个表可能有 3 个字段：姓名、城市和国家。这个表就会有 3 列：一列代表姓名，一列代表城市，一列代表国家。表中的每一行包含 3 个字段的内容，姓名字段包含姓名，城市字段包含城市，国家字段包含国家。
表空间	包含表、索引、大对象、长数据等数据的逻辑存储结构。表空间在物理数据和逻辑数据间提供了抽象的一层，为所有的数据库对象分配存储空间。表空间创建好后，创建数据库对象时可以指定该对象所属的表空间。
并发控制	在多用户环境下同时执行多个事务并保证数据完整性的一个 DBMS 服务。并发控制是 GaussDB(DWS)提供的一种多线程管理机制，用来保证多线程环境下在数据库中执行的操作是安全的和一致的。

术语	解释
查询	向数据库发出的信息请求，包含更新、修改、查询或删除信息的请求。
查询操作符	Query Operator，也称为查询迭代算子（Iterator）或查询节点（Query Tree Node）。一个查询的执行可以分解为一个或多个查询操作符，是构成一个查询执行的最基本单位。常见的查询操作符包括表扫描（Scan），表关联（Join），表聚集（Aggregation）等。
查询片段	每一个查询任务都可以分解成为一个或者多个查询片段。每个查询片段由一个或多个查询操作符构成，可独立在节点上运行。通过数据流操作符与其它查询片段块交换数据。
持久性	数据库事务的 ACID 特性之一。在事务完成以后，该事务对数据库所作的更改便持久的保存在数据库之中，并不会被回滚。
存储过程	存储过程（StoredProcedure）是在大型数据库系统中，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设置参数（如果该存储过程带有参数）来执行它。
操作系统	操作系统 OS（operating system）由引导程序加载到计算中，对计算机中其它程序进行管理。其它程序叫做应用或应用程序。
从备	Secondary，为了保证集群的高可靠性，主、备间无法正常同步数据时，主节点会将日志同步到从备。如果主节点突然故障不可用，备节点会升主，并且升主成功后从从备节点上同步之前异常期间的日志。
大对象	大对象（Blob）在数据库中指使用二进制方式存储的数据。它通常可以用于存储视频、音频和图像等多媒体数据。
动态负载	GaussDB(DWS)动态负载是指基于系统中 CPU、I/O、内存等资源的使用情况，自动调节并发作业的运行数量，避免因为系统资源过载导致业务报错或无响应。
段	数据库中，一段指包含一个或多个区域的数据库中的一部分。区域是数据库的最小范围，由单元调用块组成。一个或多个段组成一个表空间。
F - J	
Failover	指当某个节点出现故障时，自动切换到备节点上的过程。反之，从备节点上切换回来的过程称为 Failback。
FDW	Foreign Data Wrapper，外部数据封装器。是 Postgres 提供的一个 SQL 接口，用于访问远程数据存储中的大数据对象，使 DBA 可以整合来自不相关数据源的数据，将它们存入数据库中的一个公共模型。
Freeze	在事务 ID 耗尽时由 AutoVacuum Worker 进程自动执行的操作。GaussDB(DWS)会把事务 ID 记在行头，在一个事务取得一行时，通

术语	解释
	过比较行头的事务 ID 和事务本身的 ID 判断这行是否可见，而事务 ID 是一个无符号整数，如果事务 ID 耗尽，事务 ID 会跨过整数的界限重新计算，此时原先可见的行就会变成不可见的行，为了避免这个问题，Freeze 操作会将行头的事务标记为一个特殊的事务 ID，标记了这个特殊的事务 ID 的行将对所有事务可见，以此避免事务 ID 耗尽产生的问题。
GDB	GNU 工程调试器，可以监控其它程序运行时的内部情况，或者其它程序要崩溃时发生了什么。GDB 支持如下四种主要操作（使 PDK 功能更加强大），辅助查找缺陷。 • 启动程序，指定可能影响行为的任何因素。 • 特定条件下，停止程序。 • 程序停止时，检查发生了什么。 • 修改程序内容，尝试纠正一个缺陷并继续下一个。
GDS	General Data Service，数据并行加载工具。向 GaussDB(DWS)导入数据时，需要将此工具部署到源数据所在的服务器上，使 DN 可以通过该工具获取数据。
GIN 索引	Generalized Inverted Index，通用倒排索引。作用为处理索引项为组合值的情况，查询时需要通过索引搜索出出现在组合值中的特定元素值。
GNU	GNU 计划，又称革奴计划，是由 RichardStallman 在 1983 年 9 月 27 日公开发起的。它的目标是创建一套完全自由的操作系统。GNU 是“GNU's NotUnix”的递归缩写。Stallman 宣布 GNU 应当发音为 Guh-NOO 以避免与 new 这个单词混淆（注：Gnu 在英文中原意为非洲牛羚，发音与 new 相同）。Unix 是一种广泛使用的商业操作系统的名称。技术上讲，GNU 类似 Unix。但是 GNU 却给了用户自由。
gsql	GaussDB(DWS)交互终端。通过 gsql 能够以交互的方式输入查询，下发查询到 GaussDB(DWS)，然后查看查询结果。或者，也可以从文件中输入。此外，gsql 还提供许多元命令和各种类似 shell 命令，协助脚本编写及自动化各种任务。
GTM	Global Transaction Manager，全局事务管理器。用于管理事务状态的部件。
GUC	Grand Unified Configuration，数据库运行参数。配置这些参数可以影响数据库系统的行为。
HA	高可用性（HighAvailability），通过尽量缩短因日常维护操作（计划）和突发的系统崩溃（非计划）所导致的停机时间，以提高系统和应用的可用性。
HBA	host-based authentication，主机认证。主机鉴权允许主机鉴权部分或全部系统用户。适用于系统所有用户或者使用 Match 指令的子集。该类型鉴权对于管理计算集群以及其它完全同质设备非常有用。总

术语	解释
	之，服务器上的三个文件以及客户端上的一个文件必须修改，为主机鉴权做准备。
HDFS	Hadoop Distributed File System，Apache Hadoop 项目的一个子项目。一个高度容错的分布式文件系统，设计用于在低成本硬件上运行。HDFS 提供高吞吐量应用程序数据访问功能，适合带有大型数据集的应用程序。
服务器	为客户端提供服务的软硬件的组合。单独使用时，指运行服务器操作系统的计算机，也可以指提供服务的软件或者专用硬件。
高级包	GaussDB(DWS)提供的具有一定逻辑和功能的存储过程、函数，这些具备功能的存储过程、函数统称为高级包。
隔离性	数据库事务的 ACID 特性之一。它是指一个事务内部的操作及使用的数据对其它并发事务是隔离的，并发执行的各个事务之间不能互相干扰。
关系型数据库	创建在关系模型基础上的数据库。关系型数据库借助于集合代数等数学概念和方法来处理数据库中的数据。
归档线程	数据库打开归档功能时启动的一个线程，此线程用于将数据库日志归档到指定的路径。
故障接管	功能对等的系统部件对于故障部件的自动替换过程。系统部件包含处理器、服务器、网络、数据库等。
环境变量	定义进程操作环境某一方面的变量。例如，环境变量可以为主目录，命令搜索路径，使用终端或当前时区。
检查点	将数据库内存中某一时刻的数据存到磁盘的机制。GaussDB(DWS)定期将已提交的事务数据和未提交的事务数据存到磁盘，这些数据用来和 Redo 日志一起在数据库重启和崩溃时恢复数据库。
加密	用于传输数据的功能。通过该功能，可以隐藏信息内容，防止非法使用。
节点	将构成 GaussDB(DWS)集群环境的各台服务器（物理机或虚拟机）称为集群节点，简称节点。
纠错	系统自动识别软件和数据流上的错误并自动修正错误的能力，提升系统的稳定性和可靠性。
进程	在单个计算机上执行程序的实例。一个进程由一个或多个线程组成。其它进程不能接入某个进程已占用的线程。
基于时间点恢复	PITR（Point-In-TIme Recovery），基于时间点恢复是GaussDB(DWS)备份恢复的一个特性，是指在备份数据和 WAL 日志正常的情况下，数据可以恢复到指定时间点。
记录	在关系型数据库中，每一条记录对应表中的每一行数据。

术语	解释
集群	集群是由一组服务器和其它资源组成的一个单独的系统，可以实现高可用性。有的情况下，可以实现负载均衡及并行处理。
K - O	
LLVM	LLVM 命名最早源自于底层虚拟机（Low Level Virtual Machine）的缩写。LLVM 是构架编译器（compiler）的框架系统，以 C++编写而成，用于优化以任意程序语言编写的程序的编译时间（compile-time）、链接时间（link-time）、运行时间（run-time）以及空闲时间（idle-time），对开发者保持开放，并兼容已有脚本。 GaussDB(DWS) LLVM 动态编译技术可以为每个查询生成定制化的机器码用于替换原本的通用函数。通过减少实际查询时冗余的条件逻辑判断、虚函数调用并提高数据局域性，从而达到提升查询整体性能的目的。
LVS	Linux Virtual Server，虚拟服务器集群系统，用于负责集群的负载均衡。
MPP	Massive Parallel Processing，大规模并行处理。指利用多个机器构成集群的架构方式，也称为集群（Cluster）系统。
MVCC	Multi-Version Concurrency Control，多版本并发控制。数据库并发控制协议的一种，它的基本算法是一个元组可以有多个版本，不同的查询可以工作在不同的版本上。一个基本的好处是读和写可以不冲突。
NameNode	NameNode 是 Hadoop 系统中的一个中心服务器，负责管理文件系统的名字空间(namespace)以及客户端对文件的访问。
OLAP	Online Analytical Processing，联机分析处理。是数据仓库系统最主要的应用，专门设计用于支持复杂的分析操作，侧重对决策人员和高层管理人员的决策支持，可以根据分析人员的要求快速、灵活地进行大数据量的复杂查询处理，并且以一种直观而易懂的形式将查询结果提供给决策人员，以便准确掌握企业（公司）的经营状况，了解对象的需求，制定正确的方案。
OM	Operations Management，运维管理模块。提供集群日常运维、配置管理的管理接口、工具。
ORC	Optimized Row Columnar，一种广泛使用的 Hadoop 系统结构化数据的文件格式，由 Hadoop HIVE 项目引入。
客户端	连接或请求其它计算机或程序服务的计算机或程序。
空闲空间管理	管理表内空闲空间的机制，通过记录每个表内空闲空间信息，并建立易于查找的数据结构，可以加速对空闲空间进行的操作（例如 INSERT）。
跨集群	GaussDB(DWS)支持通过外表和 Extension Connector 访问当前 GaussDB(DWS)集群外的其他 DBMS 中的数据，这种行为称为跨集

术语	解释
	群。
垃圾元组	是指使用 DELETE 和 UPDATE 语句删除的元组，GaussDB(DWS)在删除元组时只是打个删除标记，由 Vacuum 线程清理这种垃圾元组。
列	字段的等效概念。在数据库中，表由一列或多列组成。
逻辑节点	一个物理节点上可以安装多个逻辑节点。一个逻辑节点是一个数据库实例。
模式	数据库对象集，包括逻辑结构，例如表、视图、序、存储过程、同义名、索引、集群及数据库链接。
模式文件	用于决定数据库结构的 SQL 文件。
P - T	
Page	GaussDB(DWS)数据库关系对象结构中行存的最小存储单元。一个 Page 大小为默认为 8KB。
PostgreSQL	PostgreSQL 是一个开源的关系数据库管理系统(DBMS)，由全球志愿者团队开发。PostgreSQL 不受任何公司或个体所控制，源代码免费使用。
Postgres-XC	一款多节点同步，读写可扩展的 PostgreSQL 集群数据库。
Postmaster	数据库服务启动时启动的一个线程。用于监听来自集群其它节点或客户端的连接请求。 主机上监听到备机连接请求，并接受后，就会创建一个 WAL Sender 线程，用于处理与备机的交互。
RHEL	Red Hat Enterprise Linux，红帽企业 Linux。
REDO 日志	记录对数据库进行操作的日志，这些日志包含重新执行这些操作所需要的信息。当数据库故障时，可以利用 REDO 日志将数据库恢复到故障前的状态。
SCTP	Stream Control Transmission Protocol，流控制传输协议。是 IETF 于 2000 年新定义的一个传输层协议。是提供基于不可靠传输业务的协议之上的可靠的数据报传输协议。SCTP 的设计用于通过 IP 网传输 SCN 窄带信令消息。
Savepoint	保存点。是一种在关系数据库管理系统中实现子事务（也称为嵌套事务）的方法。在一个长事务中，可以把操作过程分成几部分，前面部分执行成功后，可以建一个保存点，若后面的执行失败，则回滚到这个保存点即可，无需回滚整个事务。保存点对于在数据库应用程序中实现复杂错误恢复很有用。如果在多语句事务中发生错误，则应用程序可能能够从错误中恢复（通过回滚到保存点）而无需中止整个事务。
Session	数据库系统在接收到应用程序的连接请求时，为该连接创建的一个

术语	解释
	任务。它被 Session Manager 管理，完成一些初始化任务，执行用户的所有操作。
Shared-nothing architecture	无共享架构是一种分布式计算架构，这种架构中不存在集中共享 CPU、存储的状态，这种架构具有非常强的扩展性。
SLES	SUSE Linux Enterprise Server，由 SUSE 提供的企业级 Linux 操作系统。
SMP	Symmetric Multi-Processing，对称多处理技术，是指在一台计算机上汇集了一组处理器（多 CPU），各 CPU 之间共享内存子系统以及总线结构。操作系统必须支持多任务和多线程处理，以使得 SMP 系统发挥高效的性能。数据库领域的 SMP 并行技术，一般指利用多线程技术实现查询的并行执行，以充分利用 CPU 资源，从而提升查询性能。
SQL	Structure Query Language，结构化查询语言。数据库的标准查询语言。它可以分为数据定义语言（DDL），数据操纵语言（DML）和数据控制语言（DCL）。
SSL	Secure Socket Layer，安全套接层。SSL 是 Netscape 公司率先采用的网络安全协议。它是在传输通信协议（TCP/IP）上实现的一种安全协议，采用公开密钥技术。SSL 广泛支持各种类型的网络，同时提供三种基本的安全服务，它们都使用公开密钥技术。SSL 支持服务通过网络进行通信而不损害安全性。它在客户端和服务器之间创建一个安全连接。然后通过该连接安全地发送任意数据量。
收敛比	交换机下行带宽与上行带宽的比值。收敛比越高，流量收敛程度越大，丢包越严重。
TCP	Transmission Control Protocol，传输控制协议。用于将数据信息分解成信息包，使之经过 IP 协议发送；并对利用 IP 协议接收来的信息包进行校验并将其重新装配成完整的信息。TCP 是面向连接的可靠协议，能够确保信息的无误发送。
trace	一种特殊的日志记录方法，用来记录程序执行的信息。程序员使用该信息进行纠错。另外，根据 trace 日志中信息的类型和内容，有经验的系统管理员或技术支持人员以及软件监控工具诊断软件常见问题。
全备份	备份整个数据库集群。
全量同步	GaussDB(DWS)双机方案中的一种数据同步机制，是指把主机中的所有数据同步给备机。
日志文件	计算机记录自身活动的记录。
事务	数据库管理系统执行过程中的一个逻辑单位，由一个有限的数据库操作序列构成，事务必须满足 ACID 原则。
数据	事实或指令的一种表达形式，适用于人为或自动的通信、解释或处

术语	解释
	理。数据包含常量、变量、阵列和字符串。
数据重分布	用户改变数据的分布方式后，数据表在节点间重新分布的过程。
数据分布	表数据在分布式环境中的分布方式（Distribution），即数据表以何种方式打散存储到各个数据库实例上去。具体的分布方式可以有：散列（Hash）方式，复制方式（Replication）和随机方式（Random）。散列方式根据元组中指定字段的取值算得哈希值，根据节点与哈希值的映射关系获得该元组的目标存储位置。复制方式将元组复制到所有节点上。随机方式将数据随机分布到各节点。
数据分区	数据分区是指在一个数据库实例内部，将表按照划分为多个数据互不重叠的部分（Partition）。具体的分区方式可以有：范围分区（Range），它根据元组中指定字段的取值所处的范围映射到目标存储位置。
数据库	数据库是存储在一起的相关数据的集合，这些数据可以被访问，管理以及更新。同一视图中，数据库可以根据存储内容类型分为以下几类：数目类、全文本类、数字类及图像类。
数据库实例	一个数据库实例是一个 GaussDB(DWS)进程以及它控制的数据库文件。GaussDB(DWS)在一个物理节点上安装多个数据库实例，集群各节点上所安装的 GTM、CM、CN、DN 统称为实例。一个数据库实例也被称为一个逻辑节点。
数据库双机	GaussDB(DWS)提供的高可靠性双机方案。在此方案中，每个 GaussDB(DWS)逻辑节点标识为主机或备机。在同一时间内，只有一个 GaussDB(DWS)被标识为主机。双机初次建立时，主机会对每个备机数据做全量同步，然后做增量同步。双机建立之后的运行过程中，主机能接受数据读和写的操作请求，备机只做日志同步。
数据库文件	保存用户数据和数据库系统内部数据的二进制文件。
数据流操作符	负责查询片段间交换数据的操作符。根据数据流的输入、输出关系，可以细分为聚合流（Gather）、广播流（Broadcast）和重分布流（Redistribution）。聚合流将数据从多个查询片段聚合到一个。广播流将数据从一个查询片段向多个传输。重分布流则将多个查询片段的数据，按照一定规则重组后向多个传输。
数据字典	数据字典是一系列只读的表，用来提供数据库的信息。这些信息包括：数据库设计信息、存储过程信息、用户权限、用户统计数据、数据库进程信息、数据库增长统计数据和数据库性能统计数据。
死锁	为使用同一资源而产生的无法解决的争用状态。
索引	数据库索引，是数据库管理系统中一个排序的数据结构，以协助快速查询、更新数据库表中数据。
统计信息	数据库使用统计信息估算查询代价，以查找代价最小的执行计划，统计信息一般是数据库自动收集的，包括表级信息（元组数、页面数等）和列级信息（列的值域分布直方图）。

术语	解释
停用词	在信息检索中，为节省存储空间和提高搜索效率，在处理自然语言数据（或文本）之前或之后会自动过滤掉某些字或词，这些字或词即被称为 Stop Words（停用词）。
U - Z	
Vacuum	数据库定期启动的清理垃圾元组的线程，根据配置参数可以同时启动多个。
verbose	verbose 选项指定显示在屏幕上的处理信息。
WAL	Write-Ahead Logging，预写日志系统。实现事务日志的标准方法，是指对数据文件（表和索引的载体）持久化修改之前必须先持久化相应的日志。
WAL Receiver	数据库复制时备机创建的一个线程的名称。此线程用于从主机接收数据、命令，并反馈确认信息至主机。一个备机只有一个 WAL Receiver 线程。
WAL Sender	数据库复制过程中，主机接受到备机的连接请求后创建的一个线程的名称。此线程用于发送命令、数据到备机，并从备机接收信息。一个主机可能会有多个 WAL Sender 线程，每一个 WAL Sender 线程对应一个备机的一个连接请求。
WAL Writer	数据库启动时创建的一个写 Redo 日志的线程，用于将内存中的日志写入到持久性设备（如：磁盘）。
WLM	WorkLoad Manager，负载管理。GaussDB(DWS)中系统资源控制和分配的模块。
Xlog	表示事务日志，一个逻辑节点中只有一个，不允许创建多个 Xlog 文件。
xDR	详单。用户面和信令面详单的统称，包括 CDR 和 UFDR、TDR 和 SDR。
网络备份	网络备份为各种平台提供一套完整的、灵活的数据保护方案。平台包含 Microsoft Windows、UNIX 及 Linux。网络备份支持备份、归档、恢复计算机上的文件、文件夹或目录、卷或分区。
物理节点	一个物理机器称为一个物理节点。
系统表	存储数据库元信息的表，元信息包括数据库中的用户表、索引、列、函数和数据类型等。
下推	GaussDB(DWS)是分布式数据库，其可以利用多 DN 分布式并行执行查询计划，即将 CN 中的查询计划下发到各 DN 中并行执行。这种行为称为下推。与将数据抽取到 CN 上执行查询的方式相比，下推可以大幅提升查询性能。
压缩	数据压缩，信源编码，或比特率降低涉及使用相比原来较少比特的编码信息。压缩可以是有损或无损。无损压缩通过识别和消除统计

术语	解释
	冗余降低比特位。无损压缩中没有信息丢失。有损压缩识别并删除次要信息，减少了比特位。减少数据文件大小的方法被普遍称为数据压缩，尽管其正式名称为源编码（数据源的编码，然后将其存储或传输）。
一致性	数据库事务的 ACID 特性之一。在事务开始之前和事务结束以后，数据库的完整性约束没有被破坏。
元数据	用来定义数据的数据。主要是描述数据自身信息，包含源、大小、格式或其它数据特征。数据库字段中，元数据用于理解以及诠释数据仓库的内容。
原子性	数据库事务的 ACID 特性之一。整个事务中的所有操作，要么全部完成，要么全部不完成，不可能停滞在中间某个环节。事务在执行过程中发生错误，会被回滚到事务开始前的状态，就像这个事务从来没有执行过一样。
在线扩容	在线扩容是指 GaussDB(DWS)扩容重分布过程中支持数据持续入库、查询业务不中断。
脏页面	已经被修改且未写入持久性设备的页面。
增量备份	基于上次有效备份之后对文件修改的备份。
增量同步	GaussDB(DWS)双机方案中的一种数据同步机制，是指把主机中数据增量同步给备机，即只同步主备间有差异的数据。
主机	GaussDB(DWS)数据库双机系统中接受数据读写操作的节点，和所有备机一起协同工作。在同一时间内，双机系统中只有一个节点被标识为主机。
主题词	在标引和检索中用以表达文献主题的规范化的词或词组。
转储文件	转储文件是一种特定类型的 trace 文件。转储文件为响应事件过程中一次性输出的诊断数据，trace 文件指诊断数据的连续输出。
资源池	资源池是 GaussDB(DWS)提供的一种资源划分的配置机制。通过将用户绑定到资源池，来限定其所执行作业的优先级及能够利用到的资源。
租户	数据库业务用户在给定的计算资源（cpu，内存和 io）和存储资源下执行业务，通过资源管理和隔离，达成业务的服务等级协定（SLA）。
最小恢复点	最小恢复点是 GaussDB(DWS)提供的数据一致性保障手段之一。最小恢复点特性可以在 GaussDB(DWS)启动时检查出 WAL 日志和持久化到磁盘的数据的不一致性，并提示用户进行处理。

18 修订记录

修改记录

文档版本	发布日期	修改说明
03	2022-11-25	第三次正式发布，适配DWS 8.1.3.110。
02	2022-10-13	第二次正式发布。
01	2022-6-20	第一次正式发布，适配DWS 8.1.1.100。