数据仓库服务

FAQ

文档版本 35

发布日期 2025-08-26

华为云计算技术有限公司

商标声明


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

注意

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

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

目 录

1 Top 问题汇总

近期根据**智能客服**、用户声音和问题反馈情况，梳理了用户在使用DWS产品过程中关注的高频问题，您可以通过本页面尝试找到解决问题的最优方法。

语法使用

表 1-1 语法使用

热度排名	高频问题	页面地址
1	• 查询字符串的bit位数 • 截取子字符串 • 返回结果替换某些字符串 • 返回字符串的前面几个字符 • 过滤头尾部分字符串 • 获取指定字符串的字节数	**字符处理函数和操作符**
2	• 如何创建分区表 • 支持的分区类型	**CREATE TABLE PARTITION**
3	• 查看所有schema • 查看某个schema下所有的表	**Schema**
4	• 增加表字段 • 修改数据类型 • 向表中的列添加NOT NULL约束 • 设置主键 • 修改表属性	**ALTER TABLE**

热度排名	高频问题	页面地址
5	• 日期函数 • pg_sleep()如何使用 • 月份相减 • date类型转换函数	时间、日期处理函数和操作符
6	• 调整分布列 • 把分布列调整到另外一列 • 分布列的数据无法执行update操作，提示Distributed key column can't be updated in current version	如何调整分布列
7	• 分区管理 • 增加或删除分区 • 重命名分区 • 查询某个分区的数据	创建和管理分区表
8	• 查询某个分区的行数 • 合并两个分区	ALTER TABLE PARTITION
9	调用存储过程	CALL
10	• 创建表 • create table like	CREATE TABLE
11	• 授权命令 • grant语法使用 • 将用户权限授权给其他用户 • 将表权限授权给用户 • 将整库权限授权给用户 • 外表权限	GRANT
12	• REPLACE替换函数 • to_timestamp • 转换为指定格式的时间戳 • current_timestamp • to_number	类型转换函数

数据库管理

表 1-2 数据库管理

热度排名	高频问题	页面地址
1	• 查看表定义 • 查看DDL • 查看视图结构 • 查询数据库和表大小	查看表和数据库的信息
2	• 清理表空间 • VACUUM FULL • 表在大量执行增删改后，如何提升查询性能	VACUUM
3	• 如何查看某用户在当前表上是否已有权限 • 查看某用户对某张表是否有某种权限	如何查看DWS某个用户有哪些表的权限?
4	• 查询和终止阻塞语句 • 查询活跃语句 • 终止会话 • 锁等待超时	分析正在执行的SQL
5	• SQL执行计划详解 • 执行计划怎么看 • Nested Loop、Hash Join和Merge Join的差异	SQL执行计划详解
6	• 解除只读 • 数据库进入只读状态	磁盘使用率高&集群只读处理方案
7	• 收集统计信息	ANALYZE | ANALYSE
8	• 优化器配置 • 打开或关闭nestloop • 打开或关闭mergejoin • 影响执行计划的参数	优化器方法配置
9	• 更改数据库时区 • 更改Timezone	数据库时间与系统时间不一致，如何更改数据库默认时区
10	• 怎么查函数定义 • 访问权限查询函数 • 查询视图定义	系统信息函数

热度排名	高频问题	页面地址
11	• 技术指标 • 支持的分区表大小 • 单表最大数据量 • 表支持的最大列数	技术指标
12	• 开发人员选项 • 控制查询优化器是否使用分布式框架	开发人员选项

集群管理

表 1-3 集群管理

热度排名	高频问题	页面地址
1	• 磁盘使用率过高 • 磁盘满 • 集群只读 • 磁盘告警了 • 磁盘阈值怎么设置 • 磁盘告警订阅怎么开启	磁盘使用率高&集群只读处理方案
2	• 查看集群基本信息 • 查看集群公网IP • 查看数据库连接地址 • 集群的规格 • 磁盘基本信息	查看集群详情
3	如何购买集群	创建集群
4	• 集群非均衡怎么办 • 主备切换	集群主备恢复

2 产品咨询

2.1 为什么要使用云数据仓库服务 DWS?

现状和需求

大量的企业经营性数据（订单，库存，原料，付款等）在企业的业务运营系统以及其后台的（事务型）数据库中产生的。

企业的决策者需要及时地对这些数据进行归类分析，从中获得企业运营的各种业务特征，为下一步的经营决策提供数据支撑。

困难

对数据的归类分析往往涉及到对多张数据库表数据的同时访问， 即需要同时锁住多张可能正在被不同事务更新的表单。这对业务繁忙的数据库系统来说可能是一件非常困难的事情 。

- 一方面很难把多张表同时锁住，造成复杂查询的时延增加。
- 另一方面如果锁住了多张表，又会阻挡数据库表单更新的事务，造成业务的延时甚至中断。

解决方案

数据仓库主要适用于企业数据的关联和聚合等分析场景， 并从中发掘出数据背后的商业信息供决策者参考。这里的数据发掘主要指涉及多张表的大范围数据聚合和关联的复杂查询。

使用数据仓库，通过某个数据转换（ETL）的过程，业务运营数据库的数据可以被拷贝到数据仓库中供分析计算使用。同时支持把多个业务运营系统的数据汇集到一个数据仓库中。这样数据可以被更好地关联和分析，从而产生更大的价值。

数据仓库采用了一些和标准的面向事务的数据库（Oracle，SQL Server，MySQL等）不一样的设计，特别是针对数据的聚合性和关联性做了特别的优化，有些时候为了这些优化甚至可能会牺牲掉一些标准数据库的事务或者数据增删改的功能或者性能。因此，数据仓库和数据库的使用场景还是有所不同的。事务型数据库专注于事务处理（企业的业务运营），而数据仓库更擅长于复杂的数据分析。两者各司其职，互不干扰。简单一句话可以理解为，数据库主要负责数据更新，数据仓库主要负责数据分析。

云数据仓库解决方案

传统的数据仓库售价昂贵，设备系统选型，采购周期长，扩容复杂，整体运行成本十分高昂，因此很难为中小企业所采纳。

云数据仓库服务DWS与传统的数据仓库相比，主要有以下特点与显著优势：

- 一款分布式MPP数据仓库云化服务，具备开放，高效，兼容，可扩展，易运维等特点。
- 基于数据仓库产品内核，以云上数据仓库服务的形式将DWS的能力提供给云平台上的企业用户，打造云上云下一致的数据仓库用户体验。

  DWS是具有国产自主知识产权的新一代分布式数据仓库系统。目前已经被广泛地应用在国内外政府，金融，运营商等行业和财富500强企业当中。该产品不仅兼容主流开源Postgres系列数据库，而且特别针对Oracle和Teradata的SQL语法进行了兼容性增强，在很多场合都可以替代国外同类型产品。 数据仓库服务工程师重点设计实现了基于行列混存的数据仓库内核，在支持海量数据快速分析的同时也很好地兼顾了业务运作系统对数据增删改的需求。引入了基于代价的查询优化器，以及当前数据仓库系统所流行的一些黑科技，包括机器码级别的向量计算，算子间和算子内的并行，节点内和节点间并行，使用LLVM优化编译查询计划的本机代码等。这些黑科技极大地提高了数据查询和分析的性能，为用户带来了更好的体验，解决了特定场景当中的业务痛点。
- DWS服务即开即用

  相比以前动辄长达数月的数据仓库选型采购过程，在云上开通使用数据仓库服务只需要数分钟时间，简化了企业用户的购买过程，使用数据仓库的方式，降低使用数据仓库的代价和门槛，让数据仓库实实在在地走进千万家大中小企业， 让数据为企业的发展和决策提供其应有的价值。

2.2 如何选择公有云 DWS 或者公有云 RDS?

公有云DWS和公有云RDS都让您能够在云中运行传统的关系数据库，同时转移数据库管理负载。您可将RDS数据库用于联机事务处理 (OLTP)，报告和分析，对于大量数据的读（一般是复杂的只读类型查询）支持不足。DWS利用多节点的规模和资源并使用各种优化法（**列存，向量引擎，分布式框架等**），专注于联机分析处理（OLAP），为传统数据库对大型数据集的分析及报告工作负荷提供了数量级改善。

当您的数据及查询的复杂性增加时，或者在您要防止报告和分析处理对OLTP工作负荷造成干扰时，DWS可提供横向扩展能力。

您可以根据下表简单判断什么场景更适合用DWS或RDS。

表 2-1 OLTP 和 OLAP 特性比较

特性	OLTP	OLAP
用户	操作人员，低层管理人员	决策人员，高级管理人员
功能	日常操作处理	分析决策
设计	面向应用	面向主题
数据	最新的，细节的，二维的，分离的	历史的，集成的，多维的，统一的

特性	OLTP	OLAP
存取	读/写数十条记录	读上百万条记录
工作范围	简单的读写	复杂的查询
数据库大小	百GB	TB-PB级别

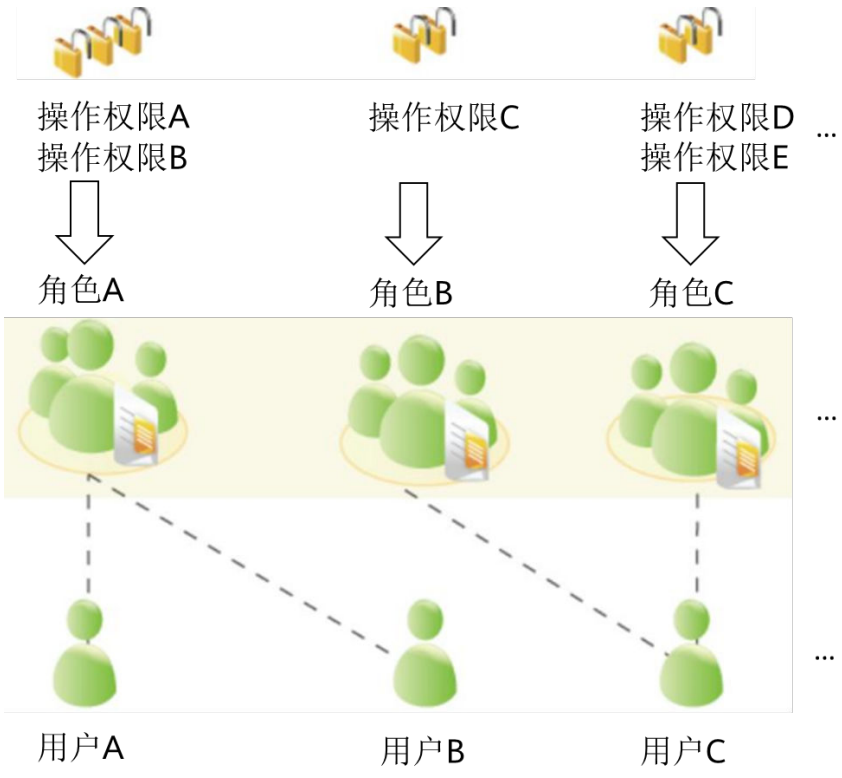
2.3 DWS 用户和角色是什么关系?

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。

- 角色（ROLE）本质上是一组权限的集合，通常情况下使用ROLE来组织权限，使用用户进行权限的管理和业务操作。
- 角色之间的权限可以继承，用户组的所有用户可自动继承对应角色的权限。
- 数据库中USER与ROLE的关系为：USER的权限来自于ROLE。
- 用户组包含了具有相同权限的用户集合。
- 用户可以看作是具有登录权限的角色。
- 角色可以看作是没有登录权限的用户。

DWS提供的权限包括“管理面”各组件的操作维护权限，在实际应用时需根据业务场景为各用户分别配置不同权限。为了提升权限管理的易用性，“管理面”引入角色的功能，通过选取指定的权限并统一授予角色，以权限集合的形式实现了权限集中查看和管理。

集中权限管理中权限、角色和用户的关系如下图所示。



DWS提供多种权限，根据业务场景实际需要选择指定的权限授予不同角色，可能是一个或者多个权限对应一个角色。

通过GRANT把角色授予用户后，用户即具有了角色的所有权限。推荐使用角色进行高效权限分配。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。

- 角色A：授予操作权限A和B，用户A和用户B通过分配角色A取得对应的权限。
- 角色B：授予操作权限C，用户C通过分配角色B取得对应的权限。
- 角色C：授予操作权限D和E，用户C通过分配角色C取得对应的权限。

2.4 如何查看 DWS 数据库用户的创建时间?

方式一：

在创建DWS数据库用户时，如果指定了用户的生效时间（VALID BEGIN）与用户创建时间一致，且之后未修改过该用户生效时间的情况下，就可以使用视图**PG_USER**查看用户列表，并通过valbegin字段查看该用户的生效时间，即该用户的创建时间。

示例：

创建用户jerry指定生效时间为当前创建时间。

```
CREATE USER jerry PASSWORD 'password' VALID BEGIN '2022-05-19 10:31:56';
```

通过查询视图PG_USER查看用户列表。valbegin字段显示了jerry的生效时间，即jerry的创建时间。

```
SELECT * FROM PG_USER;
 usename | usesysid | usecreatedb | usesuper | usecatupd | userepl |  passwd  |        valbegin        | valuntil |
respool    | parent | spacelimit | useconfig | nodegroup | tempspacelimit |
 spillspacelimit
---------+----------+-------------+----------+-----------+---------+----------+------------------------+----------
+-------------+--------+------------+-----------+-----------+----------------+
-----------------
 Ruby    |       10 | t          | t        | t         | t       | ******** |                        |          | default_pool |      0
|           |          |          |             |

 dbadmin |    16393 | f          | f        | f         | f       | ******** |                        |          | default_pool |      0
|           |          |          |             |

 jack    |   451897 | f          | f        | f         | f       | ******** |                        |          | default_pool |      0
|           |          |          |             |

 emma    |   451910 | f          | f        | f         | f       | ******** |                        |          | default_pool |      0
|           |          |          |             |

 jerry   |   457386 | f          | f        | f         | f       | ******** | 2022-05-19 10:31:56+08 |          | default_pool
|      0 |          |          |             |
(5 rows)
```

方式二：

通过系统表**PG_AUTH_HISTORY**的passwordtime字段查看用户最初的密码创建时间，即该用户的创建时间。需要有系统管理员权限才可以访问此系统表。

```
SELECT roloid, min(passwordtime) as create_time FROM pg_auth_history group by roloid order by roloid;
```

示例：

通过查询视图PG_USER获取用户jerry的OID为457386， 查询passwordtime字段获取到用户jerry的创建时间为2022-05-19 10:31:56。

```
SELECT roloid, min(passwordtime) as create_time FROM pg_auth_history group by roloid order by roloid;
 roloid |          create_time
--------+-------------------------------
     10 | 2022-02-25 09:53:38.711785+08
  16393 | 2022-02-25 09:55:17.992932+08
 451897 | 2022-05-18 09:42:26.897855+08
 451910 | 2022-05-18 09:46:33.152354+08
 457386 | 2022-05-19 10:31:56.037706+08
(5 rows)
```

2.5 如何选择 DWS 区域和可用分区

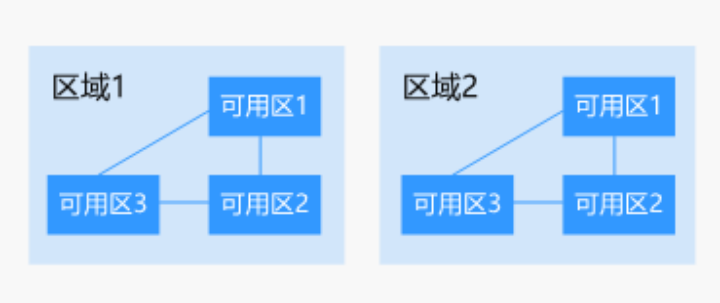
什么是区域、可用分区?

区域和**可用分区**用来描述数据中心的位置，您可以在特定的区域、可用分区创建资源。

- 区域（Region）：从地理位置和网络时延维度划分，同一个Region内共享弹性计算、块存储、对象存储、VPC网络、弹性公网IP、镜像等公共服务。Region分为通用Region和专属Region，通用Region指面向公共租户提供通用云服务的Region；专属Region指只承载同一类业务或只面向特定租户提供业务服务的专用Region。
- 可用分区（AZ，Availability Zone）：一个AZ是一个或多个物理数据中心的集合，有独立的风火水电，AZ内逻辑上再将计算、网络、存储等资源划分成多个集群。一个Region中的多个AZ间通过高速光纤相连，以满足用户跨AZ构建高可用性系统的需求。

图2-1阐明了区域和可用分区之间的关系。

图 2-1 区域和可用分区



目前，华为云已在全球多个地域开放云服务，您可以根据需求选择适合自己的区域和可用分区。更多信息请参见华为云全球站点。

如何选择区域?

选择区域时，您需要考虑以下几个因素：

- 地理位置

  一般情况下，建议就近选择靠近您或者您的目标用户的区域，这样可以减少网络时延，提高访问速度。不过，在基础设施、BGP网络品质、资源的操作与配置等方面，中国大陆各个区域间区别不大，如果您或者您的目标用户在中国大陆，可以不用考虑不同区域造成的网络时延问题。

中国香港、曼谷等其他地区和国家提供国际带宽，主要面向非中国大陆地区的用户。如果您或者您的目标用户在中国大陆，使用这些区域会有较长的访问时延，不建议使用。

- 在除中国大陆以外的亚太地区有业务的用户，可以选择“亚太-曼谷”或“亚太-新加坡”区域。
- 在非洲地区有业务的用户，可以选择“南非-约翰内斯堡”区域。
- 在欧洲地区有业务的用户，可以选择“欧洲-巴黎”区域。

- 资源的价格

  不同区域的资源价格可能有差异，请参见**华为云服务价格详情**。

如何选择可用分区?

是否将资源放在同一可用分区内，主要取决于您对容灾能力和网络时延的要求。

- 如果您的应用需要较高的容灾能力，建议您将资源部署在同一区域的不同可用分区内。
- 如果您的应用要求实例之间的网络延时较低，则建议您将资源创建在同一可用分区内。

区域和终端节点

当您通过API使用资源时，您必须指定其区域终端节点。有关华为云的区域和终端节点的更多信息，请参阅**地区和终端节点**。

2.6 数据在 DWS 中是否安全?

安全。在大数据时代，数据是用户的核心资产。公有云将继续秉承多年来向社会做出的“上不碰应用，下不碰数据”的承诺，保证用户核心资产的安全。这是对用户和社会的承诺，也是公有云及其伙伴商业成功的保障和基石。

数据仓库服务工程师对整个数据仓库系统进行了电信系统级别的安全增强，大量地采用了多年来在电信行业里积累的各种经验和知识，特别是针对数据安全，用户隐私方面的技术和专利。因此，公有云数据仓库服务是一款符合电信级质量要求的产品，满足各级政府、金融机构、电信运营商对数据安全和用户隐私的要求，并在以上各行业被广泛使用。公有云数据仓库服务还获得了如下安全认证：

- 网络安全实验室ICSL的认证：该认证是遵从英国当局颁布的网络安全标准设立的。
- 隐私和安全管理当局PSA的官方认证：该认证满足欧盟对数据安全和隐私的要求。

业务数据安全

数据仓库服务构建在公有云的基础软件设施之上，包括云主机弹性云服务器和对象存储服务OBS。弹性云服务器和OBS服务2017年双双通过了中国数据中心联盟的可信云认证 。

DWS用户的业务数据是直接存放在集群的云主机当中，集群的云主机对DWS用户本身不可见，只向用户提供数据仓库访问服务，用户以及公有云的运维管理员均无法登录DWS集群云主机进行操作。

DWS集群云主机操作系统进行了严格的安全加固，包括内核安全加固、系统最新补丁、权限控制、端口管理、协议与端口防攻击等。

DWS提供完整的密码策略、身份认证、会话管理、用户权限管理和数据库审计等安全措施。

快照数据安全

DWS的备份数据是以快照的形式存储在OBS上。OBS已通过中国数据中心联盟的可信云安全认证。OBS上的数据支持访问权限控制，密匙访问，数据加密。DWS的快照数据仅用于数据的备份和恢复，无法被外界任何用户访问操作，包括DWS用户本身。DWS系统管理员可以通过DWS Console的快照管理和公有云账单看到快照数据在OBS的空间使用情况。

网络访问安全

DWS的如下网络安全部署设计使租户之间实现100%的二三层网络隔离，满足政务，金融用户的高等级安全隔离需要。

- DWS部署在租户专属的云主机环境中，不和任何其他租户共享，从物理上隔绝了数据因为计算资源共享而被泄露的可能性。
- DWS集群的虚拟机通过虚拟私有云隔离，避免被其他租户发现和入侵。
- 网络划分为业务平面和管理平面，两个平面采用物理隔离的方式进行部署，保证业务、管理各自网络的安全性。
- 安全组规则保护，租户可以通过自定义安全组的功能， 配置安全域的访问规则，提供灵活的网络安全性配置。
- 外部应用软件访问数据仓库服务支持SSL网络安全协议。
- 支持数据从OBS导入的加密传输。

2.7 可以修改 DWS 集群的安全组吗?

DWS集群创建成功后，支持修改安全组，也可以在当前的安全组中添加、删除或修改安全组规则。

修改为其他安全组：

1. 登录**DWS控制台**。
2. 在左侧导航树，选择“集群 > 集群列表”。
3. 在集群列表中找到所需要的集群，然后单击集群名称。
4. 在“集群详情”页面中，找到“安全组”参数，单击此处安全组名称右边的修改按钮，选择需要变更的安全组名称。
5. 单击确认，即修改安全组完成。

修改已有安全组规则：

1. 登录**DWS控制台**。
2. 在左侧导航树，选择“集群 > 集群列表”。
3. 在集群列表中找到所需要的集群，然后单击集群名称。
4. 在“集群详情”页面中，找到“安全组”参数，单击安全组名称进入安全组详情页面，您可以对安全组进行设置。

说明

- 安全组的配置操作需联系组织安全保密管理员执行。
- 集群更换安全组过程中有短暂的业务中断，请谨慎操作。为了更好的网络性能，选择安全组时不要多于5个。

2.8 数据库、数据仓库、数据湖、湖仓一体分别是什么?

如今随着互联网以及物联网等技术的不断发展，越来越多的数据被生产出来，数据管理工具也得到了飞速的发展，大数据相关概念如雨后春笋一般应运而生，如数据库、数据仓库、数据湖、湖仓一体等。这些概念分别指的是什么，又有着怎样的联系，同时，对应的产品与方案又是什么呢？本文将一一进行对比介绍。

什么是数据库?

数据库是“按照数据结构来组织、存储和管理数据的仓库”。

广义上的数据库，在20世纪60年代已经在计算机中应用了。但这个阶段的数据库结构主要是层次或网状的，且数据和程序之间具备非常强的依赖性，应用较为有限。

现在通常所说的数据库指的是关系型数据库。关系数据库是指采用了关系模型来组织数据的数据库，其以行和列的形式存储数据，具有结构化程度高、独立性强、冗余度低等优点。1970年关系型数据库的诞生，真正彻底把软件中的数据和程序分开来，成为主流计算机系统不可或缺的组成部分。关系型数据库已经成为目前数据库产品中最重要的一员，几乎所有的数据库厂商新出的数据库产品都支持关系型数据库，即使一些非关系数据库产品也几乎都有支持关系数据库的接口。

关系型数据库的主要用于联机事务处理OLTP（On-Line Transaction Processing）主要进行基本的、日常的事务处理，例如银行交易等场景。

什么是数据仓库?

随着数据库的大规模应用，以及信息行业的数据爆炸式的增长。为了研究数据之间的关系，挖掘数据隐藏的价值，人们越来越多需要使用联机分析处理OLAP（On-Line Analytical Processing）进行数据分析，探究一些深层次的关系和信息。但是不同的数据库之间很难做到数据共享，数据之间的集成与分析也存在非常大的挑战。

为解决企业的数据集成与分析问题，数据仓库之父比尔·恩门于1990年提出数据仓库（Data Warehouse）。数据仓库主要功能是将OLTP经年累月所累积的大量数据，通过数据仓库特有的数据储存架构进行OLAP，最终帮助决策者能快速有效地从大量数据中，分析出有价值的信息，提供决策支持。自从数据仓库出现之后，信息产业就开始从以关系型数据库为基础的运营式系统慢慢向决策支持系统发展。

数据仓库相比数据库，主要有以下两个特点：

- 数据仓库是面向主题集成的。数据仓库是为了支撑各种业务而建立的，数据来自于分散的操作型数据。因此需要将所需数据从多个异构的数据源中抽取出来，进行加工与集成，按照主题进行重组，最终进入数据仓库。
- 数据仓库主要用于支撑企业决策分析，所涉及的数据操作主要是数据查询。因此数据仓库通过表结构优化、存储方式优化等方式提高查询速度、降低开销。

表 2-2 数据仓库与数据库的对比

维度	数据仓库	数据库
应用场景	OLAP	OLTP
数据来源	多数据源	单数据源
数据标准化	非标准化Schema	高度标准化的静态Schema
数据读取优势	针对读操作进行优化	针对写操作进行优化

什么是数据湖?

在企业内部，数据是一类重要资产已经成为了共识。随着企业的持续发展，数据不断堆积，企业希望把生产经营中的所有相关数据都完整保存下来，进行有效管理与集中治理，挖掘和探索数据价值。

数据湖就是在这种背景下产生的。数据湖是一个集中存储各类结构化和非结构化数据的大型数据仓库，它可以存储来自多个数据源、多种数据类型的原始数据，数据无需经过结构化处理，就可以进行存取、处理、分析和传输。数据湖能帮助企业快速完成异构数据源的联邦分析、挖掘和探索数据价值。

数据湖的本质，是由“数据存储架构+数据处理工具”组成的解决方案。

- 数据存储架构：要有足够的扩展性和可靠性，可以存储海量的任意类型的数据，包括结构化、半结构化和非结构化数据。
- 数据处理工具，则分为两大类：
  - 第一类工具，聚焦如何把数据“搬到”湖里。包括定义数据源、制定数据同步策略、移动数据、编制数据目录等。
  - 第二类工具，关注如何对湖中的数据进行分析、挖掘、利用。数据湖需要具备完善的数据管理能力、多样化的数据分析能力、全面的数据生命周期管理能力、安全的数据获取和数据发布能力。如果没有这些数据治理工具，元数据缺失，湖里的数据质量就没法保障，最终会由数据湖变质为数据沼泽。

随着大数据和AI的发展，数据湖中数据的价值逐渐水涨船高，价值被重新定义。数据湖能给企业带来多种能力，例如实现数据的集中式管理，帮助企业构建更多优化后的运营模型，也能为企业提供其他能力，如预测分析、推荐模型等，这些模型能刺激企业能力的后续增长。

对于数据仓库与数据湖的不同之处，可以类比为仓库和湖泊的区别：仓库存储着来自特定来源的货物；而湖泊的水来自河流、溪流和其他来源，并且是原始数据。

表 2-3 数据湖与数据仓库的对比

维度	数据湖	数据仓库
应用场景	可以探索性分析所有类型的数据，包括机器学习、数据发现、特征分析、预测等。	通过历史的结构化数据进行数据分析。
使用成本	起步成本低，后期成本较高。	起步成本高，后期成本较低。

维度	数据湖	数据仓库
数据质量	包含大量原始数据，使用前需要清洗和标准化处理。	质量高，可作为事实依据。
适用对象	数据科学家、数据开发人员为主。	业务分析师为主。

什么是湖仓一体?

虽然数据仓库和数据湖的应用场景和架构不同，但它们并不是对立关系。数据仓库存储结构化的数据，适用于快速的BI和决策支撑，而数据湖可以存储任何格式的数据，往往通过挖掘能够发挥出数据的更大作为，因此在一些场景上二者的并存可以给企业带来更多收益。

湖仓一体，又被称为Lake House，其出发点是通过数据仓库和数据湖的打通和融合，让数据流动起来，减少重复建设。Lake House架构最重要的一点，是实现数据仓库和数据湖的数据/元数据无缝打通和自由流动。湖里的“显性价值”数据可以流到仓里，甚至可以直接被数仓使用；而仓里的“隐性价值”数据，也可以流到湖里，低成本长久保存，供未来的数据挖掘使用。

数据智能方案

数据治理中心DataArts Studio为大型政企客户量身定制跨越孤立系统、感知业务的数据资源智能管理解决方案，实现全域数据入湖，帮助政企客户从多角度、多层次、多粒度挖掘数据价值，实现数据驱动的数字化转型。

数据治理中心DataArts Studio的核心主要是智能数据湖FusionInsight，包含数据库、数据仓库、数据湖等各计算引擎平台，提供了数据使能的全套能力，支持数据的采集、汇聚、计算、资产管理、数据开放服务的全生命周期管理。

拥有强大的湖、仓、库引擎技术，比如数据湖敏捷构建、GaussDB数据库快速迁移，数仓的实时分析等，对应服务如下：

- 数据库：
  - 关系型数据库包括：**云数据库RDS**、**云数据库 TaurusDB**、**云数据库GaussDB**、**云数据库PostgreSQL**等。
  - 非关系型数据库包括：**文档数据库服务DDS**、**云数据库GeminiDB**等。
- 数据仓库：**数据仓库服务 DWS**。
- 数据湖\湖仓一体：**MapReduce服务MRS**，**数据湖探索DLI**等。
- 数据治理中心：**数据治理中心DataArts Studio**。

2.9 DWS 的脏页是如何产生的?

产生原因

DWS采用多版本控制技术（Multi-Version Concurrency Control，简称MVCC）的并发控制机制保证多个事务访问数据库时的一致性和并发性，其优点是读写互不阻塞，缺点则是会造成磁盘膨胀的问题，而MVCC机制是产生脏页的主要原因。

具体表现为如下场景：

- 当对表执行delete操作时，删除的数据只是在逻辑上被标记为已删除，并未真正从磁盘页面中移除。
- 当对表执行update操作时，DWS将待更新的原数据进行逻辑上的删除标记，同时插入新数据。

对于表中的delete、update操作，被标记为已删除的数据在数据库内部统称为废弃元组，废弃元组在整张表中的占比即为脏页率。因此当表的脏页率高时，则认为表内部被标记为已删除的数据占比高。

处理方案

针对表的脏页率过高的问题，DWS提供了查询脏页率的系统视图，具体使用请参见**PGXC_STAT_TABLE_DIRTY**。

为了解决脏页率高导致磁盘空间膨胀的问题，DWS提供了VACUUM的功能，可以有效清理delete、update操作后标记的已删除数据，具体请参见**VACUUM**。

VACUUM不会释放已经分配好的空间，如果要彻底回收已删除的空间，则需要使用VACUUM FULL。

说明

- VACUUM FULL可以清理并释放已删除数据的空间，进而提高数据库的性能和效率。但是执行VACUUM FULL命令更加耗费时间和资源，并且可能会导致一些表被锁定，因此VACUUM FULL命令需要在数据库负载较低的情况下使用。
- 为降低磁盘膨胀对数据库性能的影响，建议对查询脏页率超过80%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。

2.10 如何使用 VPC 共享来处理 DWS 资源?

背景信息

共享VPC功能支持多个账号在一个集中管理、共享的VPC内创建云资源，比如DWS、ELB、ECS等。VPC的所有者可以将VPC内的子网共享给一个或者多个账号使用。通过共享VPC功能，可以简化网络配置，帮助您统一配置和运维多个账号下的资源，有助于提升资源的管控效率，降低运维成本。更多信息请参见**共享VPC**。

约束与限制

- 所有者和使用者的子网在同一个VPC内，子网默认网络互通。但是由于使用者和所有者位于共享子网内的资源关联不同的安全组内，因此资源之间网络隔离，如果需要资源之间互通，需要添加安全组规则放通不同安全组之间的网络，具体方法请参见**添加安全组规则**。

  比如，放通账号A和账号B内两个DWS的安全组，则需要分别在两个安全组内添加入方向规则，源地址选择对方安全组。
- 单个使用者最多可同时接收100个共享子网，当共享子网数量超过100个时，使用者将无法接收到超出数量的共享子网。
- 单个子网最多可同时共享给100个使用者，当使用者数量超过100个时，超出数量的使用者将无法接收到共享子网。

共享 VPC 内所有者和使用者的权限

所有者将VPC子网共享给使用者后，所有者和使用者对共享子网、以及子网内关联云资源的操作权限如表2-4所示。

表 2-4 共享 VPC 内所有者和使用者的权限

角色	所有者将子网共享给使用者时	所有者停止子网共享后	使用者退出子网共享后
所有者	• 所有者不可以修改、删除使用者创建的资源，比如DWS集群、ECS、ELB等。 • 在子网的“IP地址管理”页面中，所有者可以查看使用者创建资源的IP地址和资源ID等信息。	• 所有者可以正常使用、删除、管理VPC下的所有资源。 • 如果使用者在已停止共享的子网中仍拥有资源，则所有者无法删除共享子网或共享子网所在的VPC。	• 所有者可以正常使用、删除、管理VPC下的所有资源。 • 如果使用者退出子网共享后，在共享的子网中仍拥有资源，则所有者无法删除共享子网或共享子网所在的VPC。
使用者	• 使用者可以在共享VPC子网内新建资源，比如ECS、ELB、RDS实例等。 • 在子网的“IP地址管理”页面中，使用者可以查看自己创建资源的IP地址和资源ID等信息，无法查看所有者和其他使用者创建的资源信息。	使用者可以继续使用自己创建的资源，无法在该共享子网内新创建资源。	使用者可以继续使用自己创建的资源，无法在该共享子网内新创建资源。

如何在共享 VPC 中使用 DWS 资源

1. 您可以使用**RAM管理控制台**或者**VPC管理控制台**，创建子网共享，**表2-5**中详细为您介绍**方法A**和**方法B**。
2. 共享后创建DWS集群时“配置网络 > 虚拟私有云”模块即可选择共享的VPC资源。

表 2-5 共享子网创建流程说明

方法	说明	操作指导
方法A	1. 通过RAM管理控制台，所有者创建共享，将子网共享给使用者。配置如下： a. 选择共享子网。 b. 为共享子网选择权限，即指定使用者对该共享子网具备的权限。 c. 指定共享子网的使用者，可以指定多个。 2. 共享创建完成后，通过RAM管理控制台，使用者可以选择接受或者拒绝共享申请。 ● 使用者接受共享申请，子网共享成功。如果后续使用者不再需要使用该共享子网，可以退出该共享。 ● 使用者拒绝共享申请，子网共享失败。	1. **创建共享** 2. **接受共享邀请** **退出共享**
方法B	1. 通过RAM管理控制台，所有者创建共享，将子网共享给使用者。配置如下： a. 选择共享子网。 b. 为共享子网选择权限，即指定使用者对该共享子网具备的权限。 c. 指定共享子网的使用者，可以指定多个。 2. 通过VPC管理控制台，所有者创建子网共享，将子网添加至**1**中已创建的共享中即可。 3. 共享创建完成后，通过RAM管理控制台，使用者可以选择接受或者拒绝共享申请。 ● 使用者接受共享申请，子网共享成功。如果后续使用者不再需要使用该共享子网，可以退出该共享。 ● 使用者拒绝共享申请，子网共享失败。	1. **创建共享** 2. **将VPC子网共享给其他账号** 3. **接受共享邀请** **退出共享**

3 数据库连接

3.1 如何与 DWS 进行通信?

业务应用与DWS进行通信的基本原则是确保业务应用所在网络与DWS的网络能连通，以下是当前主流连接场景，请根据实际情况进行选择。

表 3-1 如何与 DWS 进行通信

场景		通信方式	支持的连接方式
云上	**业务应用与DWS在同一个区域内同一个VPC下**	同一个VPC的两个内网IP直接通信。	• gsql • Data Studio • JDBC/ODBC 更多连接方式参见**连接集群的方式**。
	业务应用与DWS在同一个区域内不同VPC下	将两个VPC建立**对等连接**后，两个内网IP直接通信。	
	业务应用与DWS在不同区域下	两个区域建立**云连接**（CC）后，再通过内网IP进行通信。	
云下和云上	**业务应用在云下数据中心，需要与DWS进行通信**	• 使用DWS的**公网IP/公网域名**进行通信。 • 使用**云专线**（DC）进行通信。	

业务应用与 DWS 在同一个区域内同一个 VPC 下

为保证业务低时延，建议将业务应用和DWS都部署在同一个区域内。例如业务应用部署在ECS，建议将DWS集群部署在跟ECS在同一个虚拟私有云（以下简称VPC）下，应用通过内网IP直接跟DWS进行通信。该场景下，在创建DWS集群时，选择DWS集群跟ECS保持在同一个区域和VPC内。

例如ECS部署在“北京四”，则DWS选择在“北京四”下，同时选择DWS跟ECS保持在同一个VPC1下，ECS的内网IP为192.168.120.1，DWS的内网IP为192.168.120.2，即可确保通过内网IP进行通信。

检查通信的基本要点是ECS出方向规则和DWS入方向规则，检查步骤如下：

步骤1 检查ECS出方向规则：

应确保ECS的安全组的出方向规则放通，如下。如果没有放通，请参见ECS的**配置安全组规则**。

策略	协议端口	类型	目的地址
允许	全部	IPv4	0.0.0.0/0

步骤2 检查DWS入方向规则：

DWS创建时如果没有专门设置安全组，系统默认的安全组入规则已放通所有IPv4地址、端口为8000的TCP类型访问。为确保安全，也支持只开放某个对应的IP，详情参见**使用公网IP连接集群时如何设置白名单?**

允许	TCP : 8000	IPv4	0.0.0.0/0

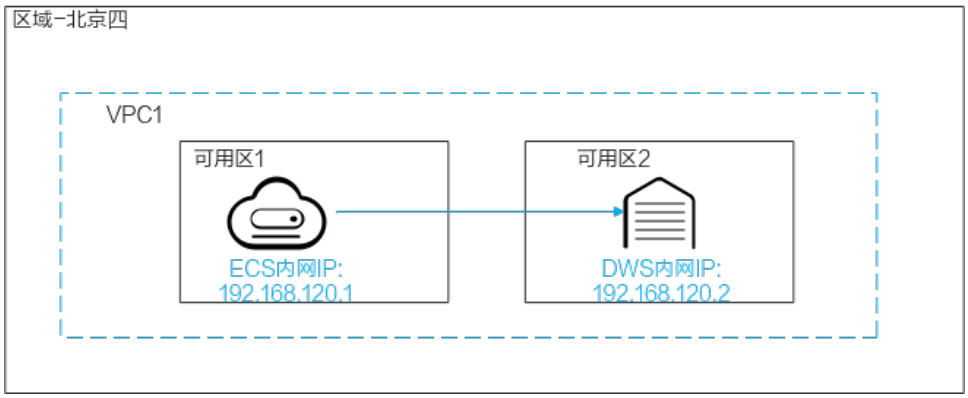
步骤3 登录ECS，能ping通DWS的内网IP，表示网络连通，如果ping不通，请检查以上配置，如果ECS有防火墙，请一并检查防火墙配置。

----结束

使用gsql连接示例：

gsql -d gaussdb -h *192.168.120.2* **-p 8000 -U dbadmin -W** *password* **-r**

图 3-1 内网 IP 访问



业务应用与 DWS 在同一个区域内不同 VPC 下

为保证业务低时延，建议将业务应用和DWS都部署在同一个区域内。例如业务应用部署在ECS，建议将DWS集群部署在跟ECS在同一个虚拟私有云（以下简称VPC）下，如果DWS集群选择了不同的VPC，则ECS与DWS无法直接连通。

例如ECS和DWS都部署在“北京四”下，但是ECS在VPC1下，DWS在VPC2下，此时需要将VPC1和VPC2建立**对等连接**后，ECS即可通过DWS的内网IP进行访问。

检查通信的基本要点是ECS出方向规则、DWS入方向规则、两个VPC已建立对等连接，检查步骤如下：

步骤1 检查ECS出方向规则：

应确保ECS的安全组的出方向规则放通，如下。如果没有放通，请参见ECS的**配置安全组规则**。

策略	协议端口	类型	目的地址
允许	全部	IPv4	0.0.0.0/0

步骤2 检查DWS入方向规则：

DWS创建时如果没有专门设置安全组，系统默认的安全组入规则已放通所有IPv4地址、端口为8000的TCP类型访问。为确保安全，也支持只开放某个对应的IP，详情参见**使用公网IP连接集群时如何设置白名单?**

允许	TCP : 8000	IPv4	0.0.0.0/0

步骤3 将ECS所在的VPC1和DWS所在的VPC2建立**对等连接**。

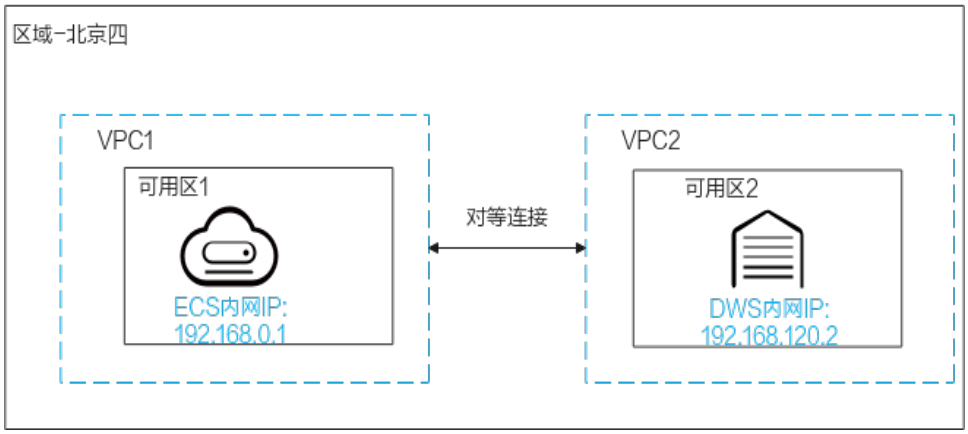
步骤4 登录ECS，能ping通DWS的内网IP，表示网络连通，如果ping不通，请检查以上配置，如果ECS有防火墙，请一并检查防火墙配置。

----结束

使用gsql连接示例：

gsql -d gaussdb -h *192.168.120.2* **-p 8000 -U dbadmin -W** *password* **-r**

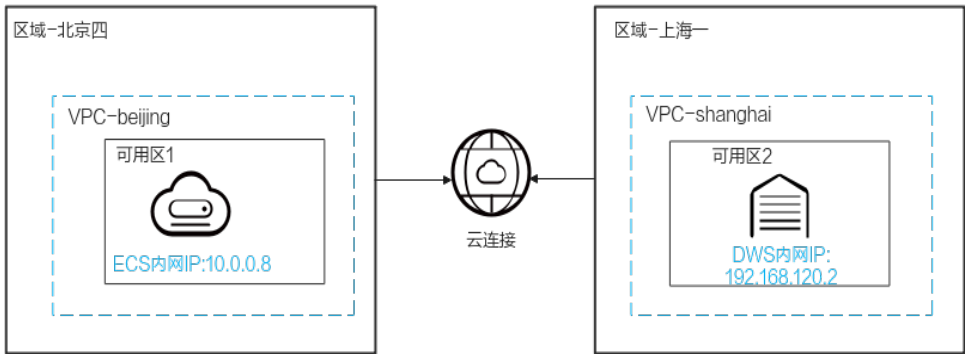
图 3-2 对等连接访问



业务应用与 DWS 在不同区域下

如果业务应用与DWS在不同区域下，例如ECS在“北京四”、DWS在“上海一”，此时需要将两个区域建立**云连接**后才能通信。

图 3-3 云连接访问

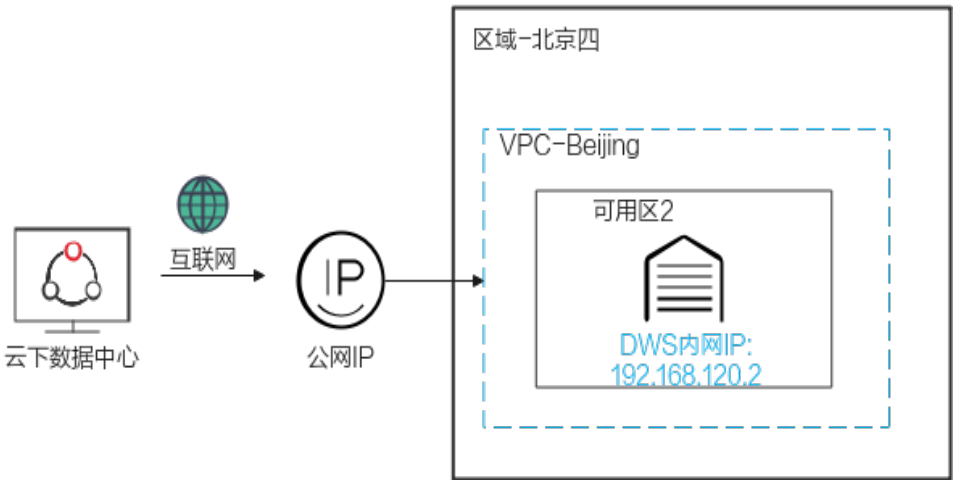


业务应用在云下数据中心，需要与 DWS 进行通信

如果业务应用不在云上，在本地数据中心，此时需要与云上DWS进行通信，分为两种场景：

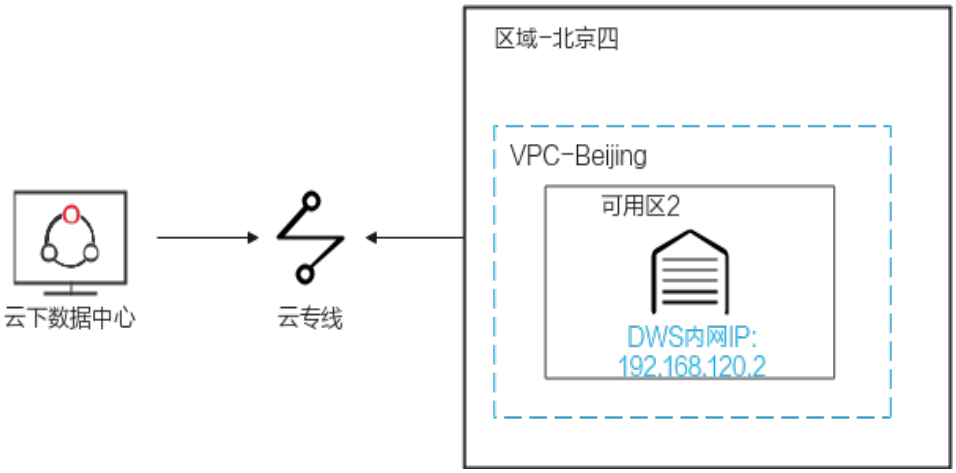
- **场景一**：云下业务应用通过DWS的公网IP进行通信。
  使用gsql连接示例：
  gsql -d gaussdb -h *公网IP* **-p 8000 -U dbadmin -W** *password* **-r**

  图 3-4 公网 IP 访问



- **场景二**：云下业务无法访问外网，则需要通过**云专线**进行通信。

图 3-5 云专线访问



3.2 DWS 是否支持第三方客户端以及 JDBC 和 ODBC 驱动程序?

推荐使用DWS客户端和驱动程序。与开源的PostgreSQL客户端和驱动程序相比，有两个主要的优点：

- 安全强化：PostgreSQL驱动程序只支持MD5认证，但DWS驱动程序支持SHA256和MD5。
- 数据类型增强：DWS驱动程序支持新的数据类型smalldatetime和tinyint。

DWS支持开源PostgreSQL客户端和JDBC和ODBC驱动程序。

兼容的客户端和驱动程序版本如下：

- PostgreSQL的psql 9.2.4或更高版本
- PostgreSQL JDBC驱动程序9.3-1103或更高版本
- PSQL ODBC 09.01.0200或更高版本

使用JDBC/ODBC连接DWS，可参见《数据仓库服务开发指南》的**教程：使用JDBC或ODBC开发**。

须知

- 建议使用官方推荐的方式连接数据库。参见**连接集群方式**。
- 其它客户端不能保证100%兼容性，需要客户自行验证。
- 使用其它客户端如果出现不兼容而报错，且不能替换客户端的情况，可尝试替换客户端中的libpq驱动。替换方法：参见**工具下载**下载并解压gsql客户端压缩包，获取gsql目录下的libpg.so，替换到客户端指定目录中。

3.3 无法连接 DWS 集群时怎么处理?

检查原因

基本原因可能有以下几种：

- 集群状态是否正常。
- 连接命令是否正确，用户名、密码、IP地址或端口无误。
- 安装客户端的操作系统类型、版本是否正确。
- 安装客户端的操作是否正确。

如果是在公有云环境无法连接，还需要检查以下可能导致异常的原因：

- 弹性云服务器是否与集群在相同可用分区、虚拟私有云、子网和安全组。
- 安全组的出入规则是否正确。

如果是在互联网环境无法连接，还需要检查以下可能导致异常的原因：

- 用户网络是否与互联网可以正常连通。
- 用户网络防火墙策略是否限制了访问。
- 用户网络是否需要通过代理才能访问互联网。

联系服务人员

如果无法确定原因并解决问题，请提交工单反馈问题。您可以登录管理控制台，在右上方单击“工单>新建工单”填写并提交工单。

3.4 为什么在互联网环境连接 DWS 后，解绑了 EIP 不会立即返回失败消息?

这是因为解绑了EIP后，会导致网络断开。但是此过程中，TCP协议层因keepalive等的设置，无法及时识别物理连接已经故障，导致gsql，ODBC和JDBC等客户端无法及时识别网络故障。

客户端等待数据库返回的时间与keepalive参数的设置相关，具体可以表示为：keepalive_time + keepalive_probes * keepalive_intvl。

因为keepalive参数涉及到网络的通信的稳定性，所以可根据具体的业务压力与网络状况进行调整。

如果是Linux环境，使用sysctl命令修改如下参数：

- net.ipv4.tcp_keepalive_time
- net.ipv4.tcp_keeaplive_probes
- net.ipv4.tcp_keepalive_intvl

以修改net.ipv4.tcp_keepalive_time参数值为例，执行如下命令将参数值修改为120秒：

sysctl net.ipv4.tcp_keepalive_time=120

如果是Windows环境，修改注册表“HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\Tcpip\Parameters”中的如下配置信息：

- KeepAliveTime
- KeepAliveInterval
- TcpMaxDataRetransmissions（相当于tcp_keepalive_probes）

说明

如果以上参数不在注册表“HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\Tcpip\Parameters”中，可以在注册表编辑器对应路径下右键单击“新建 > DWORD值”进行添加。

3.5 使用公网 IP 连接 DWS 集群时如何设置白名单?

用户可以登录VPC管理控制台手动创建一个安全组，然后回到DWS创建集群页面，单击“安全组”下拉列表旁边的 按钮，刷新后在“安全组”下拉列表中选择新建的安全组。

为了使DWS客户端可以连接集群，用户需要在新建的安全组中添加一条入规则，开放DWS集群的数据库端口的访问权限。

- 协议：TCP。
- 端口范围：8000。指定为创建DWS集群时设置的数据库端口，这个端口是DWS用于接收客户端连接的端口。
- 源地址：选中“IP地址”，然后指定为客户端主机的IP地址，例如“192.168.0.10/32”。

图 3-6 添加入方向规则

添加完成后，即设置白名单成功。

3.6 使用 API 调用和直连数据库两种方式有哪些差异?

API调用跟直连数据库两种连接方式的优劣可用从安全性、性能维度进行对比：

安全性

表 3-2 安全性对比

维度	API 访问	直连数据库
通信加密	天然支持 HTTPS + TLS 加密。	需要手动配置SSL。
身份鉴权	支持更灵活的方式（AK/SK或Token认证）。	依赖数据库账号密码，权限粒度较粗（如 Schema/Table 级别）。
访问控制	可在 API 网关层限制 IP、频率、黑名单等。	依赖数据库网络白名单（如 VPC、安全组）。
防注入攻击	参数化接口天然防SQL注入。	需要代码层防御SQL注入。
审计与日志	集中记录API调用日志（请求来源、参数等）。	依赖数据库日志（SQL 语句、来源 IP 等）。

API 访问在安全性上更优，尤其是对外部系统暴露时，适合作为公共接口；直连数据库需依赖额外安全配置。

性能

表 3-3 性能对比

维度	API 访问	直连数据库
网络开销	需额外 HTTP 协议头，序列化/反序列化成本较高。	使用高效二进制协议（如libpq）。
延迟	通常更高（需经过 API 网关、应用服务器等）。	更低延迟（直接与数据库通信）。
吞吐量	受限于API服务器性能。	直接利用数据库高性能引擎。
长连接复用	通常短连接，需频繁建连。	支持连接池，减少建连开销。
复杂查询	可能无法直接传递复杂SQL。	支持原生SQL，优化器可发挥最大性能。

直连数据库在性能上更优，尤其是对复杂查询或高吞吐场景；API 访问适用于轻量级、高频次简单请求。

适用场景

通过以上对比，两种方式适配不同场景：

- 直连数据库：适合大数据量操作，安全性不高 ，需要提供数据库名称账号 、密码等。
- 通过API访问 适合低频访问，封装固定操作，安全性高，需要一个服务层，同时可以实现字段控制。

4 数据迁移

4.1 DWS 的 OBS 外表与 GDS 外表支持的数据格式有什么区别?

OBS与GDS外表支持格式文件区别如下：

OBS导入支持ORC、TEXT、JSON、CSV、CARBONDATA、PARQUET文件格式，导出支持ORC、CSV、TEXT、PARQUET文件格式，缺省值为TEXT。

GDS导入导出支持的文件格式：TEXT、CSV和FIXED，缺省值为TEXT。

4.2 数据如何存储到 DWS?

DWS支持多数据源高效入库，典型的入库方式如下所示。详细指导请参见**导入数据**。

- 从OBS导入数据。
  数据上传到OBS对象存储服务中，再从OBS中导入，支持CSV，TEXT格式数据。
- 通过INSERT语句直接插入数据。
  用户可以通过DWS提供的客户端工具（gsql）或者JDBC/ODBC驱动从上层应用向DWS写入数据。DWS支持完整的数据库事务级别的增删改(CRUD)操作。这是最简单的一种方式，这种方式适合数据写入量不太大， 并发度不太高的场景。
- 从MRS导入数据，将MRS作为ETL。
- 通过COPY FROM STDIN方式导入数据。
  通过COPY FROM STDIN命令写数据到一个表。
- 使用GDS从远端服务器导入数据到DWS。
  当用户需要将普通文件系统（例如，弹性云服务器）中的数据文件导入到DWS时，可以使用DWS提供的GDS导入数据的功能。
- 使用CDM迁移数据到DWS。

4.3 DWS 可以存储多少业务数据?

数据仓库集群每个节点默认能够支持1.49TB、2.98TB、4.47TB、160GB、1.68TB、13.41TB六种规格的存储容量，一个集群支持的节点数范围为3～256，集群总的存储容量随集群规模等比例扩充。

为增强可靠性，每个节点都有一个副本，副本会占用一半的存储空间，选择容量时副本容量会自动翻倍存储。

数据仓库系统会备份数据，生成索引、临时缓存文件、运行日志等内容，并占用存储容量。每个节点实际存储的数据，大致为总存储容量的一半。

4.4 如何使用 DWS 的\copy 导入导出?

由于云上DWS是全托管服务，用户无法登录后台，无法使用copy进行导入导出文件，所以云上将copy语法禁掉。云上推荐将数据文件放到obs上，使用obs外表进行入库，如果需要使用copy导入导出数据，可以参考如下方法：

1. 将数据文件放到客户端的机器上。
2. 使用gsql连接集群。
3. 执行如下命令导入数据，输入数据文件在客户端的目录信息和文件名，with中指定导入选项，跟正常copy一样，但是需要在copy前添加"\"标识，入库成功后不会有消息提示。
   ```
   \copy tb_name from '/directory_name/file_name' with(...);
   ```
4. 执行如下命令，使用默认参数直接导出数据到本地文件。
   ```
   \copy table_name to '/directory_name/file_name';
   ```
5. 使用copy_option参数导出为CSV文件。
   ```
   \copy table_name to '/directory_name/file_name' CSV;
   ```
6. 使用with指定option参数，导出为CSV文件，分隔符为'|'。
   ```
   \copy table_name to '/directory_name/file_name' with(format 'csv',delimiter '|') ;
   ```

4.5 如何实现 DWS 不同编码库之间数据容错导入

要实现从数据库A（UTF8编码）至数据库B（GBK编码）的数据导入，常规方法导入数据时会出现字符集编码不匹配的错误，导致数据无法导入。

针对小批量数据导入的场景，可以通过\COPY命令来完成，具体方法如下：

步骤1 创建数据库A和B，其中数据库A的编码格式为UTF8，数据库B的编码格式为GBK。

```
postgres=> CREATE DATABASE A ENCODING 'UTF8' template = template0;
postgres=> CREATE DATABASE B ENCODING 'GBK' template = template0;
```

步骤2 查看数据库列表，可以看到已经创建的数据库A和B。

```
postgres=> \l
                        List of databases
   Name    |  Owner  | Encoding  | Collate | Ctype | Access privileges
-----------+---------+-----------+---------+-------+-------------------
 a         | dbadmin | UTF8      | C       | C     |
 b         | dbadmin | GBK       | C       | C     |
 gaussdb   | Ruby    | SQL_ASCII | C       | C     |
 postgres  | Ruby    | SQL_ASCII | C       | C     |
 template0 | Ruby    | SQL_ASCII | C       | C     | =c/Ruby          +
           |         |           |         |       | Ruby=CTc/Ruby
```

```
 template1 | Ruby    | SQL_ASCII | C       | C     | =c/Ruby          +
           |         |           |         |       | Ruby=CTc/Ruby
 xiaodi    | dbadmin | UTF8      | C       | C     |
(7 rows)
```

步骤3 切换到数据库A，输入用户密码。创建表test01，并向表中插入数据。

```
postgres=> \c a
Password for user dbadmin:
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_128_GCM_SHA256, bits: 128)
You are now connected to database "a" as user "dbadmin".

a=> CREATE TABLE test01
 (
    c_customer_sk             integer,
    c_customer_id             char(5),
    c_first_name              char(6),
    c_last_name               char(8)
 )
 with (orientation = column,compression=middle)
 distribute by hash (c_last_name);
CREATE TABLE
a=> INSERT INTO test01(c_customer_sk, c_customer_id, c_first_name) VALUES (3769, 'hello', 'Grace');
INSERT 0 1
a=> INSERT INTO test01 VALUES (456, 'good');
INSERT 0 1
```

步骤4 使用\COPY命令从utf8库中将数据以Unicode编码格式导出成文件test01.dat。

```
\copy test01 to '/opt/test01.dat' with (ENCODING 'Unicode');
```

步骤5 切换到数据库B，创建同名表test01。

```
a=> \c b
Password for user dbadmin:
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_128_GCM_SHA256, bits: 128)
You are now connected to database "b" as user "dbadmin".

b=> CREATE TABLE test01
 (
    c_customer_sk             integer,
    c_customer_id             char(5),
    c_first_name              char(6),
    c_last_name               char(8)
 )
 with (orientation = column,compression=middle)
 distribute by hash (c_last_name);
```

步骤6 使用\COPY命令将文件test01.dat导入到数据库B。

```
\copy test01 from '/opt/test01.dat' with (ENCODING 'Unicode' ,COMPATIBLE_ILLEGAL_CHARS 'true');
```

说明

- 容错性参数COMPATIBLE_ILLEGAL_CHARS指定导入时对非法字符进行容错处理，非法字符转换后入库。不报错，不中断导入。
- 此参数不支持BINARY格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。
- 此参数仅对COPY FROM导入有效。

步骤7 在数据库B里查看表test01中的数据；

```
b=> select * from test01;
 c_customer_sk | c_customer_id | c_first_name | c_last_name
---------------+---------------+--------------+-------------
          3769 | hello         | Grace        |
           456 | good          |              |
(2 rows)
```

步骤8 通过以上操作就可完成将数据从数据库A（UTF8）至数据库B（GBK）的导入。

----结束

4.6 DWS 导入性能都和哪些因素有关联?

dws的导入性能受多方面因素影响，主要有以下几点：

1. 集群规格：磁盘io、网络吞吐、内存、cpu规格等。
2. 业务规划：表字段的类型、是否压缩、行存还是列存。
3. 数据存储：集群本地、OBS等。
4. 数据导入的方式选择等。

5 数据库使用

5.1 如何调整 DWS 分布列?

在数据仓库类型的数据库中，大表的分布列选择对于数据库和语句查询性能都有至关重要的影响。 如果表的分布列选择不当，在数据导入后有可能出现数据分布倾斜，进而导致某些磁盘的使用明显高于其他磁盘，极端情况下会导致集群只读。对于Hash分表策略，存在数据倾斜情况下，查询时出现部分DN的I/O短板，从而影响整体查询性能。合理的选择分布列，并对已经创建的表，进行分布列的调整，对表查询的性能至关重要。

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。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。

针对分布不均匀的表，尽可能通过调整分布列，以减少数据倾斜，避免带来潜在的数据库性能问题。

选择合适的分布列

Hash分布表的分布列选取至关重要，需要满足以下基本原则：

- 列值应比较离散，以便数据能够均匀分布到各个DN。例如，考虑选择表的主键为分布列，如在人员信息表中选择身份证号码为分布列。
- 在满足第一条原则的情况下尽量不要选取存在常量filter的列。
- 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。
- 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。

如何调整

通过**select version();**语句查询当前数据库版本号，版本号不同，调整的方式不同：

```
test_lhy=> select version();
                                                                                                version
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
 PostgreSQL 9.2.4 (GaussDB 8.1.1 build 7ab61a49) compiled at 2021-06-26 12:05:53 commit 2518 last mr 3356 release
(1 row)
```


- **8.0.x及以前版本，通过重建表时指定分布列来调整**：

步骤1 通过Data Studio或者Linux下使用gsql访问数据库。

步骤2 创建新表。

说明

以下步骤语句中，table1为原表名，table1_new为新表名，column1和column2为分布列名称。

```
CREATE TABLE IF NOT EXISTS table1_new
( LIKE table1 INCLUDING ALL EXCLUDING DISTRIBUTION)
DISTRIBUTE BY
HASH (column1, column2);
```

步骤3 迁移数据到新表。

```
START TRANSACTION;
LOCK TABLE table1 IN ACCESS EXCLUSIVE MODE;
INSERT INTO table1_new SELECT * FROM table1;
COMMIT;
```

步骤4 查看表数据是否迁移成功，删除原表。

```
SELECT COUNT(*) FROM table1_new;
DROP TABLE table1;
```

步骤5 替换原表。

```
ALTER TABLE table1_new RENAME TO table1;
```

----结束

- **8.1.0及以后版本后**，通过ALTER TABLE语法进行调整，以下为示例。

步骤1 查询当前表定义，返回结果显示该表分布列为c_last_name。

```
SELECT pg_get_tabledef('customer_t1');
```

```
gaussdb=> select pg_get_tabledef ('customer_t1');
                                pg_get_tabledef
--------------------------------------------------------------------------------
 SET search_path = public;                                                     +
 CREATE  TABLE customer_t1 (                                                   +
         c_customer_sk integer,                                                +
         c_customer_id character(5),                                           +
         c_first_name character(6),                                            +
         c_last_name character(8)                                              +
 )                                                                             +
 WITH (orientation=column, compression=middle, colversion=2.0, enable_delta=false)+
 DISTRIBUTE BY HASH(c_last_name)                                               +
 TO GROUP group_version1;
(1 row)
```

步骤2 更新分布列中的数据时报错。

```
UPDATE customer_t1 SET c_last_name = 'Jimy' WHERE c_customer_sk = 6885;
```

```
gaussdb=> update customer_t1 set c_last_name = 'Jimy' where c_customer_sk = 6885;
ERROR:  Distributed key column can't be updated in current version
```

步骤3 将该表的分布列修改为不会更新的列，例如c_customer_sk。

```
ALTER TABLE customer_t1 DISTRIBUTE BY hash (c_customer_sk);
```

```
gaussdb=> alter table customer_t1 DISTRIBUTE BY hash (c_customer_sk);
ALTER TABLE
```

步骤4 重新执行更新旧的分布列的数据。更新成功。

```
UPDATE customer_t1 SET c_last_name = 'Jimy' WHERE c_customer_sk = 6885;
```

```
gaussdb=> update customer_t1 set c_last_name = 'Jimy' where c_customer_sk = 6885;
UPDATE 1
```

----结束

5.2 如何查看和设置 DWS 数据库的字符集编码格式

查看数据库字符集编码

使用server_encoding参数查看当前数据库的字符集编码。例如，查看到数据库music的字符集编码为UTF8。

```
music=> SHOW server_encoding;
 server_encoding
-----------------
 UTF8
(1 row)
```

设置数据库的字符集编码

说明

DWS不支持修改已创建数据库的字符编码格式。

如果需要指定数据库的字符集编码格式，可按照下面的CREATE DATABASE语法格式，使用template0新建一个数据库。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UTF8编码。

CREATE DATABASE 语法格式

```
CREATE DATABASE database_name
   [ [ WITH ] { [ OWNER [=] user_name ] |
            [ TEMPLATE [=] template ] |
            [ ENCODING [=] encoding ] |
            [ LC_COLLATE [=] lc_collate ] |
            [ LC_CTYPE [=] lc_ctype ] |
            [ DBCOMPATIBILITY [=] compatibility_type ] |
            [ CONNECTION LIMIT [=] connlimit ]}[...] ];
```

- **TEMPLATE [=] template**

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

  取值范围：已有数据库的名称。不指定时，系统默认拷贝template1。另外，不支持指定为gaussdb数据库。

  须知

  目前不支持模板库中含有SEQUENCE对象。如果模板库中有SEQUENCE，则会创建数据库失败。

- **ENCODING [=] encoding**

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

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

常用取值：GBK、UTF8、Latin1。

须知

指定新的数据库字符集编码必须与所选择的本地环境中（LC_COLLATE和LC_CTYPE）的设置兼容。

当指定的字符编码集为GBK时，部分中文生僻字无法直接作为对象名。这是因为GBK第二个字节的编码范围在0x40-0x7E之间时，字节编码与ASCII字符@A-Z[\]^_`a-z{|}重叠。其中@[\]^_'{|}是数据库中的操作符，直接作为对象名时，会语法报错。例如“俦”字，GBK16进制编码为0x8240，第二个字节为0x40，与ASCII“@”符号编码相同，因此无法直接作为对象名使用。如果确实要使用，可以在创建和访问对象时，通过增加双引号来规避这个问题。

示例

创建一个UTF8编码的数据库music（本地环境的编码格式必须也为UTF8）。

```
CREATE DATABASE music ENCODING 'UTF8' template = template0;
```

5.3 如何处理 DWS 建表时 date 类型字段自动转换为 timestamp 类型的问题?

创建数据库时，可通过**DBCOMPATIBILITY**参数指定兼容的数据库的类型，**DBCOMPATIBILITY**取值范围：ORA、TD、MySQL。分别表示兼容Oracle、Teradata和MySQL数据库。如果创建数据库时不指定该参数，则默认为ORA，在ORA兼容模式下，date类型会自动转换为timestamp(0)，只有在MySQL兼容模式下才支持date类型。

为解决以上问题，需要将兼容模式修改为MySQL。DWS不支持修改现有数据库的兼容模式，只能在创建数据库时指定兼容模式。DWS从8.1.1集群版本开始支持MySQL兼容模式类型，可参考如下示例进行操作：

```
gaussdb=> CREATE DATABASE mydatabase DBCOMPATIBILITY='mysql';
CREATE DATABASE
gaussdb=> \c mydatabase
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "mydatabase" as user "dbadmin".
mydatabase=> create table t1(c1 int, c2 date);
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
```

若无法采用重建数据库更改兼容模式的方法，可通过修改字段类型来规避date类型会自动转换为timestamp类型。例如，将日期以string类型的形式插入表中，可参考如下示例操作：

```
gaussdb=> CREATE TABLE mytable (a date,b int);
CREATE TABLE
gaussdb=> INSERT INTO mytable VALUES(date '12-08-2023',01);
INSERT 0 1
```

```
gaussdb=> SELECT * FROM mytable;
          a          | b
---------------------+---
 2023-12-08 00:00:00 | 1
(1 row)
gaussdb=> ALTER TABLE mytable MODIFY a VARCHAR(20);
ALTER TABLE
gaussdb=> INSERT INTO mytable VALUES('2023-12-10',02);
INSERT 0 1
gaussdb=> SELECT * FROM mytable;
          a          | b
---------------------+---
 2023-12-08 00:00:00 | 1
 2023-12-10          | 2
(2 rows)
```

5.4 DWS 是否需要定时对常用的表做 VACUUM FULL 和 ANALYZE 操作?

需要。

对于频繁增、删、改的表，需要定期执行VACUUM FULL和ANALYZE，该操作可回收已更新或已删除的数据所占据的磁盘空间，防止因数据膨胀和统计信息不准造成性能下降。

- 一般情况下，对表执行完大量**增、改**操作后，建议进行ANALYZE。
- 对表执行过**删除**操作后，建议进行VACUUM，一般不建议日常使用VACUUM FULL选项，但是可以在特殊情况下使用。例如，用户删除了一个表的大部分行之后，希望从物理上缩小该表以减少磁盘空间占用。VACUUM和VACUUM FULL具体的差异可以参考**VACUUM和VACUUM FULL**。

语法格式

指定某张表进行分析。

```
ANALYZE table_name;
```

对数据库中的所有表（非外部表）进行分析。

```
ANALYZE;
```

指定某张表进行VACUUM。

```
VACUUM table_name;
```

指定某张表进行VACUUM FULL。

```
VACUUM FULL table_name;
```

更多语法参见**VACUUM**和**ANALYZE | ANALYSE**。

说明

- 如果执行VACUUM FULL命令后所占用物理空间无变化（未减少），请确认是否有其他活跃事务（删除数据事务开始之前开始的事务，并在VACUUM FULL执行前未结束）存在，如果有，需等其他活跃事务退出后进行重试。
- 8.1.3及以上版本中Vacuum/Vacuum Full可在管理控制台操作，详情可参见**智能运维**。

VACUUM 和 VACUUM FULL

在DWS中，VACUUM的本质就是一个“吸尘器”，用于吸收“尘埃”。而尘埃其实就是旧数据，如果这些数据没有及时清理，那么将会导致数据库空间膨胀，性能下降，更严重的情况会导致异常退出。

VACUUM的作用：

- 空间膨胀问题：清除废旧元组以及相应的索引。包括提交的事务delete的元组（以及索引）、update的旧版本（以及索引），回滚的事务insert的元组（以及索引）、update的新版本（以及索引）、copy导入的元组（以及索引）。
- FREEZE：防止因事务ID回卷问题（Transaction ID wraparound）而导致的异常退出，将小于OldestXmin的事务号转化为freeze xid，更新表的relfrozenxid，更新库的relfrozenxid、truncate clog。
- 更新统计信息：VACUUM ANALYZE时，会更新统计信息，使得优化器能够选择更好的方案执行SQL语句。

VACUUM命令存在两种形式，VACUUM和VACUUM FULL，目前VACUUM对行存表有作用，对列存表无显著的作用，列存表只能依靠VACUUM FULL释放空间。具体区别见下表：

表 5-1 VACUUM 和 VACUUM FULL

差异项	VACUUM	VACUUM FULL
空间清理	如果删除的记录位于表的末端，其所占用的空间将会被物理释放并归还操作系统。而如果不是末端数据，会将表中或索引中dead tuple（死亡元组）所占用的空间置为可用状态，从而复用这些空间。	不论被清理的数据处于何处，这些数据所占用的空间都将被物理释放并归还于操作系统。当再有数据插入后，分配新的磁盘页面使用。
锁类型	共享锁，可以与其他操作并行。	排他锁，执行期间基于该表的操作全部挂起。
物理空间	不会释放。	会释放。
事务ID	不回收。	回收。
执行开销	开销较小，可以定期执行。	开销很大，建议确认数据库所占磁盘页面空间接近临界值再执行操作，且最好选择数据量操作较少的时段完成。
执行效果	执行后基于该表的操作效率有一定提升。	执行完后，基于该表的操作效率大大提升。

5.5 如何导出 DWS 某张表结构?

使用 SQL 编辑器进行数据导出

建议使用SQL编辑器进行表数据导出，登录数据源，编辑框面板上方选择对应的数据库和模式，输入查询表数据的SQL语句，单击“**导出**”按钮。支持以下几种导出方式：

- **本地导出**：将查询SQL的所有结果导出到xlsx或csv文件，可直接在本地打开查看，最多支持导出20000条数据。
- **全量导出**：将查询SQL的所有结果导出到指定的OBS桶下的路径中，默认为csv文件。

具体操作请参见**导出数据**。

使用 gs_dump 和 gs_dumpall 工具进行数据导出

除了使用SQL编辑器，您也可以通过gs_dump和gs_dumpall工具进行数据导出，支持以下几种常见场景：

- 导出单个数据库：
  - 数据库级导出。
  - 模式级导出。
  - 表级导出。
- 导出所有数据库：
  - 数据库级导出。
  - 各库全局对象导出。

具体操作请参见**gs_dump**和**gs_dumpall**。

5.6 DWS 是否有高效的删除表数据的方法?

有。删除大批量的日志数据时，使用delete语法需要花费更大的时间，此时可以通过TRUNCATE语法进行大批量删除操作，它的删除速度比delete快得多。

详情请参见**TRUNCATE**。

功能描述

清理表数据，TRUNCATE在清理表数据时，可以快速地从表中删除所有行。它和在目标表上进行无条件的DELETE有同样的效果，由于TRUNCATE不做表扫描，在大表上操作效果更明显，效率会提升很多。

功能特点

- TRUNCATE TABLE在功能上与不带WHERE子句DELETE语句相同：二者均删除表中的全部行。
- TRUNCATE TABLE比DELETE速度快且使用系统和事务日志资源少：

  - DELETE语句每次删除一行，并在事务日志中为所删除每行记录一项。
  - TRUNCATE TABLE通过释放存储表数据所用数据页来删除数据，并且只在事务日志中记录页的释放。
- TRUNCATE，DELETE，DROP三者的差异如下：
  - TRUNCATE TABLE，删除内容，释放空间，但不删除定义。
  - DELETE TABLE，删除内容，不删除定义，不释放空间。
  - DROP TABLE，删除内容和定义，释放空间。

示例

- 创建表。

```
CREATE TABLE tpcds.reason_t1 AS TABLE tpcds.reason;
```

清空表tpcds.reason_t1。

```
TRUNCATE TABLE tpcds.reason_t1;
```

删除表。

```
DROP TABLE tpcds.reason_t1;
```

- 创建分区表。

```
CREATE TABLE tpcds.reason_p
(
  r_reason_sk integer,
  r_reason_id character(16),
  r_reason_desc character(100)
)PARTITION BY RANGE (r_reason_sk)
(
  partition p_05_before values less than (05),
  partition p_15 values less than (15),
  partition p_25 values less than (25),
  partition p_35 values less than (35),
  partition p_45_after values less than (MAXVALUE)
);
```

插入数据。

```
INSERT INTO tpcds.reason_p SELECT * FROM tpcds.reason;
```

清空分区p_05_before。

```
ALTER TABLE tpcds.reason_p TRUNCATE PARTITION p_05_before;
```

清空13所在的分区p_15。

```
ALTER TABLE tpcds.reason_p TRUNCATE PARTITION for (13);
```

清空分区表。

```
TRUNCATE TABLE tpcds.reason_p;
```

删除表。

```
DROP TABLE tpcds.reason_p;
```

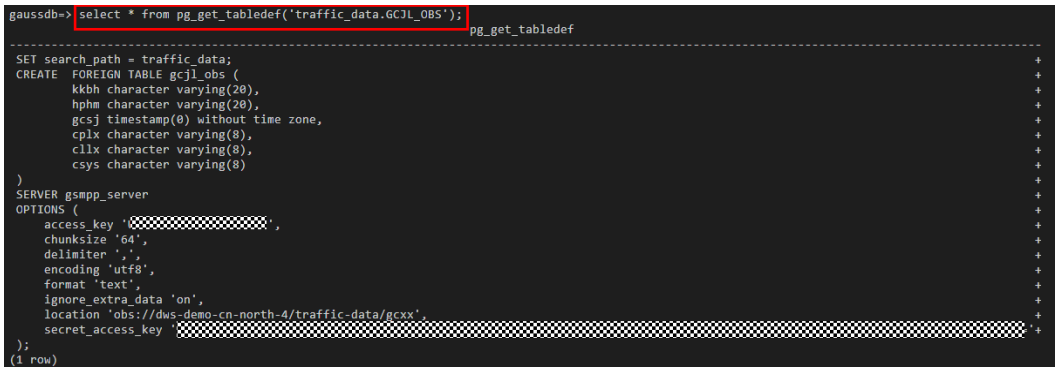
5.7 如何查看 DWS 外部表信息?

如果需要查询OBS、GDS等外表信息（如OBS路径），可以执行以下语句查询。

```
SELECT * FROM pg_get_tabledef('外表名称')
```

例如，表名为traffic_data.GCJL_OBS，查询如下：

```
SELECT * FROM pg_get_tabledef('traffic_data.GCJL_OBS');
```



5.8 如果 DWS 建表时没有指定分布列，数据会怎么存储?

说明

8.1.2及以上集群版本，可通过GUC参数default_distribution_mode来查询和设置表的默认分布方式。

如果建表时没有指定分布列，数据会以下几种场景来存储：

- 场景一

  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。

```
CREATE TABLE warehouse1
(
    W_WAREHOUSE_SK          INTEGER          PRIMARY KEY,
    W_WAREHOUSE_ID          CHAR(16)           NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)
);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "warehouse1_pkey" for table
"warehouse1"
CREATE TABLE

SELECT getdistributekey('warehouse1');
 getdistributekey
------------------
 w_warehouse_sk
(1 row)
```

- 场景二

  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。

```
CREATE TABLE warehouse2
(
    W_WAREHOUSE_SK          INTEGER                  ,
    W_WAREHOUSE_ID          CHAR(16)           NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)
);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'w_warehouse_sk' as the distribution
column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

SELECT getdistributekey('warehouse2');
 getdistributekey
------------------
 w_warehouse_sk
(1 row)
```

- 场景三

  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。

```
CREATE TABLE warehouse3
(
    W_WAREHOUSE_ID          CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)
);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'w_warehouse_id' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

SELECT getdistributekey('warehouse3');
 getdistributekey
------------------
 w_warehouse_id
(1 row)
```

5.9 如何将 DWS 联结查询的 null 结果替换成 0?

在执行outer join（left join、right join、full join）联结查询时，outer join在匹配失败的情况下结果集会补空，产生大量NULL值， 可以在联结查询时将这部分null值替换为0。

可使用coalesce函数，它的作用是返回参数列表中第一个非NULL的参数值。例如：

```
SELECT coalesce(NULL,'hello');
 coalesce
----------
 hello
(1 row)
```

有表course1和表course2，使用left join对两表进行联结查询：

```
SELECT * FROM course1;
  stu_id  |  stu_name  |     cour_name
----------+------------+--------------------
 20110103 | ALLEN      | Math
 20110102 | JACK       | Programming Design
 20110101 | MAX        | Science
(3 rows)

SELECT * FROM course2;
 cour_id |     cour_name      | teacher_name
---------+--------------------+--------------
    1002 | Programming Design | Mark
    1001 | Science            | Anne
(2 rows)

SELECT course1.stu_name,course2.cour_id,course2.cour_name,course2.teacher_name FROM course1 LEFT JOIN course2 ON course1.cour_name = course2.cour_name ORDER BY 1;
  stu_name  | cour_id |     cour_name      | teacher_name
------------+---------+--------------------+--------------
 ALLEN      |         |                    |
 JACK       |    1002 | Programming Design | Mark
 MAX        |    1001 | Science            | Anne
(3 rows)
```

使用coalesce函数将查询结果中的空值替换为0或其他非0值：

```
SELECT course1.stu_name,
 coalesce(course2.cour_id,0) AS cour_id,
 coalesce(course2.cour_name,'NA') AS cour_name,
 coalesce(course2.teacher_name,'NA') AS teacher_name
```

```
 FROM course1
 LEFT JOIN course2 ON course1.cour_name = course2.cour_name
 ORDER BY 1;
  stu_name  | cour_id |     cour_name      | teacher_name
------------+---------+--------------------+--------------
 ALLEN      |       0 | NA                 | NA
 JACK       |    1002 | Programming Design | Mark
 MAX        |    1001 | Science            | Anne
(3 rows)
```

5.10 如何查看 DWS 表是行存还是列存?

表的**存储方式**由建表语句中的ORIENTATION参数控制，row表示行存，column表示列存。

不指定ORIENTATION参数，默认为row行存。

查看已创建的表是行存还是列存，可通过表定义函数PG_GET_TABLEDEF查询。

如下orientation=column表示为列存表。

目前暂不支持通过ALTER TABLE语句修改ORIENTATION参数，即行存表和列存表无法直接进行转换。

```
SELECT * FROM PG_GET_TABLEDEF('customer_t1');
                                pg_get_tabledef
--------------------------------------------------------------------------------
 SET search_path = tpchobs;                                                    +
 CREATE  TABLE customer_t1 (                                                     +
         c_customer_sk integer,                                                +
         c_customer_id character(5),                                           +
         c_first_name character(6),                                            +
         c_last_name character(8)                                              +
 )                                                                           +
 WITH (orientation=column, compression=middle, colversion=2.0, enable_delta=false)+
 DISTRIBUTE BY HASH(c_last_name)                                                 +
 TO GROUP group_version1;
(1 row)
```

5.11 DWS 列存表的常用信息查询

使用列存表时，一些常用信息查询SQL示例：

先创建列存分区表my_table，并向表中插入数据。

```
CREATE TABLE my_table
(
    product_id INT,
    product_name VARCHAR2(40),
    product_quantity INT
 )
WITH (ORIENTATION = COLUMN)
PARTITION BY range(product_quantity)
(
partition my_table_p1 values less than(600),
partition my_table_p2 values less than(800),
partition my_table_p3 values less than(950),
partition my_table_p4 values less than(1000));

INSERT INTO my_table VALUES(1011, 'tents', 720);
INSERT INTO my_table VALUES(1012, 'hammock', 890);
INSERT INTO my_table VALUES(1013, 'compass', 210);
INSERT INTO my_table VALUES(1014, 'telescope', 490);
INSERT INTO my_table VALUES(1015, 'flashlight', 990);
INSERT INTO my_table VALUES(1016, 'ropes', 890);
```

查看已创建的列存分区表：

```
SELECT * FROM my_table;
 product_id | product_name | product_quantity
------------+--------------+------------------
       1013 | compass      |              210
       1014 | telescope    |              490
       1011 | tents        |              720
       1015 | flashlight   |              990
       1012 | hammock      |              890
       1016 | ropes        |              890
(6 rows)
```

查询分区边界

```
SELECT relname, partstrategy, boundaries FROM pg_partition where parentid=(select parentid from
pg_partition where relname='my_table');
   relname   | partstrategy | boundaries
-------------+--------------+------------
 my_table    | r            |
 my_table_p1 | r            | {600}
 my_table_p2 | r            | {800}
 my_table_p3 | r            | {950}
 my_table_p4 | r            | {1000}
(5 rows)
```

查询列存表列数

```
SELECT count(*) FROM ALL_TAB_COLUMNS where table_name='my_table';
 count
-------
     3
(1 row)
```

查询数据在各 DN 分布

```
SELECT table_skewness('my_table');
           table_skewness
-------------------------------------
 ("dn_6007_6008         ",3,50.000%)
 ("dn_6009_6010         ",2,33.333%)
 ("dn_6003_6004         ",1,16.667%)
 ("dn_6001_6002         ",0,0.000%)
 ("dn_6005_6006         ",0,0.000%)
 ("dn_6011_6012         ",0,0.000%)
(6 rows)
```

查询某一有数据分布 DN 上分区 P1 所对应的 cudesc 和 delta 表名称

```
EXECUTE DIRECT ON (dn_6003_6004) 'select a.relname from pg_class a, pg_partition b where
(a.oid=b.reldeltarelid or a.oid=b.relcudescrelid) and b.relname=''my_table_p1''';
       relname
----------------------
 pg_delta_part_60317
 pg_cudesc_part_60317
(2 rows)
```

5.12 DWS 查询时索引失效场景解析

对表建立索引可提高数据库查询性能，但有时会出现建立了索引，但查询计划中却发现索引没有被使用的情况。针对这种情况，本文将列举几种常见的场景和优化方法。

场景一：返回结果集很大

以行存表的Seq Scan和Index Scan为例：

- Seq Scan：按照表的记录的排列顺序从头到尾依次检索扫描，每次扫描要取到所有的记录。这也是最简单最基础的扫表方式，扫描的代价比较大。
- Index Scan：对于指定的查询，先扫描一遍索引，从索引中找到符合要求的记录的位置（指针），再定位到表中具体的Page去获取，即先走索引，再读表数据。

因此，根据两种扫描方式的特点可知，多数情况下，Index Scan要比Seq Scan快。但是如果获取的结果集占所有数据的比重很大时（超过70%），这时Index Scan因为要先扫描索引再读表数据反而不如直接全表扫描的速度快。

场景二：未及时 ANALYZE

ANALYZE更新表的统计信息，如果表未执行ANALYZE或最近一次执行完ANALYZE后表进行过数据量较大的增删操作，会导致统计信息不准，该场景下也可能导致查询表时没有使用索引。

优化方法：对表执行ANALYZE更新统计信息。

场景三：过滤条件使用了函数或隐式类型转化

如果在过滤条件中使用了计算、函数、隐式类型转化，都可能导致无法选择索引。

例如创建表，并在a，b，c三列上都分别创建了索引。

```
CREATE TABLE test(a int, b text, c date);
```

- 在索引字段进行计算操作。

  从下面的执行结果可以看出：where a = 101，where a = 102 - 1都使用了a列上的索引，但是where a + 1 = 102没有使用索引。

```
explain verbose select * from test where a  = 101;
                                  QUERY PLAN
------------------------------------------------------------------------------------------------------------
 id |                  operation                  | E-rows | E-distinct | E-memory | E-width | E-costs
----+---------------------------------------------+--------+------------+----------+---------+---------
  1 | ->  Streaming (type: GATHER)                |      1 |            |          |      44 | 16.27
  2 |    ->  Index Scan using index_a on public.test |      1 |            | 1MB      |      44 | 8.27

Predicate Information (identified by plan id)
-----------------------------------------------
  2 --Index Scan using index_a on public.test
        Index Cond: (test.a = 101)

Targetlist Information (identified by plan id)
------------------------------------------------
  1 --Streaming (type: GATHER)
        Output: a, b, c
        Node/s: dn_6005_6006
  2 --Index Scan using index_a on public.test
        Output: a, b, c
        Distribute Key: a

  ====== Query Summary =====
-------------------------------
System available mem: 3358720KB
Query Max mem: 3358720KB
Query estimated mem: 1024KB
(24 rows)
 explain verbose select * from test where a  = 102 - 1;
                                  QUERY PLAN
```

```
-----------------------------------------------------------------------------------------------------
 id |                  operation                   | E-rows | E-distinct | E-memory | E-width | E-costs
----+----------------------------------------------+--------+------------+----------+---------+---------
  1 | ->  Streaming (type: GATHER)                 |      1 |            |          |      44 | 16.27
  2 |    ->  Index Scan using index_a on public.test |      1 |            | 1MB      |      44 | 8.27

Predicate Information (identified by plan id)
-----------------------------------------------
  2 --Index Scan using index_a on public.test
        Index Cond: (test.a = 101)

Targetlist Information (identified by plan id)
------------------------------------------------
  1 --Streaming (type: GATHER)
        Output: a, b, c
        Node/s: dn_6005_6006
  2 --Index Scan using index_a on public.test
        Output: a, b, c
        Distribute Key: a

  ====== Query Summary =====
-------------------------------
System available mem: 3358720KB
Query Max mem: 3358720KB
Query estimated mem: 1024KB
(24 rows)
explain verbose select * from test where a + 1 = 102;
                                QUERY PLAN
-----------------------------------------------------------------------------------------
 id |          operation           | E-rows | E-distinct | E-memory | E-width | E-costs
----+------------------------------+--------+------------+----------+---------+---------
  1 | ->  Streaming (type: GATHER) |      1 |            |          |      44 | 22.21
  2 |    ->  Seq Scan on public.test |      1 |            | 1MB      |      44 | 14.21

Predicate Information (identified by plan id)
-----------------------------------------------
  2 --Seq Scan on public.test
        Filter: ((test.a + 1) = 102)

Targetlist Information (identified by plan id)
------------------------------------------------
  1 --Streaming (type: GATHER)
        Output: a, b, c
        Node/s: All datanodes
  2 --Seq Scan on public.test
        Output: a, b, c
        Distribute Key: a

  ====== Query Summary =====
-------------------------------
System available mem: 3358720KB
Query Max mem: 3358720KB
Query estimated mem: 1024KB
(24 rows)
```

优化方式：尽量使用常量代替表达式，或者常量计算尽量写在等号的右侧。

- 在索引字段上使用函数。

  从下面的执行结果可以看出：在索引列上使用函数也会导致无法选择索引。

```
explain verbose select * from test where to_char(c, 'yyyyMMdd') =
to_char(CURRENT_DATE,'yyyyMMdd');
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------
 id |          operation           | E-rows | E-distinct | E-memory | E-width | E-costs
----+------------------------------+--------+------------+----------+---------+---------
  1 | ->  Streaming (type: GATHER) |      1 |            |          |      44 | 22.28
  2 |    ->  Seq Scan on public.test |      1 |            | 1MB      |      44 | 14.28
```

```
                                Predicate Information (identified by plan id)

-------------------------------------------------------------------------------------------------------------------------------------------
--------------------
  2 --Seq Scan on public.test
        Filter: (to_char(test.c, 'yyyyMMdd'::text) = to_char(('2022-11-30'::pg_catalog.date)::timestamp
with time zone, 'yyyyMMdd'::text))

 Targetlist Information (identified by plan id)
 ----------------------------------------------
  1 --Streaming (type: GATHER)
        Output: a, b, c
        Node/s: All datanodes
  2 --Seq Scan on public.test
        Output: a, b, c
        Distribute Key: a

  ====== Query Summary =====
 -------------------------------
 System available mem: 3358720KB
 Query Max mem: 3358720KB
 Query estimated mem: 1024KB
(24 rows)
explain verbose select * from test where c = current_date;
                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------
 id |                  operation                   | E-rows | E-distinct | E-memory | E-width | E-costs
----+----------------------------------------------+--------+------------+----------+---------+---------
  1 | ->  Streaming (type: GATHER)                 |      1 |            |          |      44 | 16.27
  2 |    ->  Index Scan using index_c on public.test |      1 |            | 1MB      |      44 | 8.27

       Predicate Information (identified by plan id)
 -----------------------------------------------------------
  2 --Index Scan using index_c on public.test
        Index Cond: (test.c = '2022-11-30'::pg_catalog.date)

 Targetlist Information (identified by plan id)
 ----------------------------------------------
  1 --Streaming (type: GATHER)
        Output: a, b, c
        Node/s: All datanodes
  2 --Index Scan using index_c on public.test
        Output: a, b, c
        Distribute Key: a

  ====== Query Summary =====
 -------------------------------
 System available mem: 3358720KB
 Query Max mem: 3358720KB
 Query estimated mem: 1024KB
(24 rows)
```

优化方法：尽量减少索引列上没有必要的函数调用。

- 数据类型隐式转化。

  此类场景较常见，例如字段b的类型是text，过滤条件是where b = 2，在生成计划时，text类型会隐式转化为bigint类型，实际的过滤条件变成where b::bigint = 2，导致b列上的索引失效。

```
explain verbose select * from test where b = 2;
                                  QUERY PLAN
-------------------------------------------------------------------------------------------
 id |          operation           | E-rows | E-distinct | E-memory | E-width | E-costs
----+------------------------------+--------+------------+----------+---------+---------
  1 | ->  Streaming (type: GATHER) |      1 |            |          |      44 | 22.21
  2 |    ->  Seq Scan on public.test |      1 |            | 1MB      |      44 | 14.21

 Predicate Information (identified by plan id)
 ---------------------------------------------
  2 --Seq Scan on public.test
```



```
        Filter: ((test.b)::bigint = 2)

 Targetlist Information (identified by plan id)
 ------------------------------------------------
   1 --Streaming (type: GATHER)
         Output: a, b, c
         Node/s: All datanodes
   2 --Seq Scan on public.test
         Output: a, b, c
         Distribute Key: a

   ====== Query Summary =====
 --------------------------------
 System available mem: 3358720KB
 Query Max mem: 3358720KB
 Query estimated mem: 1024KB
(24 rows)
explain verbose select * from test where b = '2';
                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------
 id |                   operation                    | E-rows | E-distinct | E-memory | E-width | E-costs
----+------------------------------------------------+--------+------------+----------+---------+---------
  1 | ->  Streaming (type: GATHER)                   |      1 |            |          |      44 | 16.27
  2 |    ->  Index Scan using index_b on public.test |      1 |            | 1MB      |      44 | 8.27

 Predicate Information (identified by plan id)
 -----------------------------------------------
   2 --Index Scan using index_b on public.test
         Index Cond: (test.b = '2'::text)

 Targetlist Information (identified by plan id)
 -----------------------------------------------
   1 --Streaming (type: GATHER)
         Output: a, b, c
         Node/s: All datanodes
   2 --Index Scan using index_b on public.test
         Output: a, b, c
         Distribute Key: a

   ====== Query Summary =====
 --------------------------------
 System available mem: 3358720KB
 Query Max mem: 3358720KB
 Query estimated mem: 1024KB
(24 rows)
```

优化方法：索引条件上的常量尽可能使用和索引列相同类型的常量，避免发生隐式类型转化。

场景四：用 nestloop + indexscan 代替 hashjoin

此类语句的特征是两个表关联的时候，其中一个表上where条件过滤之后的结果集行数很小，同时，最终满足条件的结果集行数也很小。此时，使用nestloop+indexscan的效果往往要优于hashjoin。较优的执行计划如下：

可以看到第5层的Index Cond: (t1.b = t2.b)已经把join条件下推到了基表扫描上。

```
explain verbose select t1.a,t1.b from t1,t2 where t1.b=t2.b and t2.a=4;
 id |                   operation                    | E-rows | E-distinct | E-memory | E-width | E-costs
----+------------------------------------------------+--------+------------+----------+---------+---------
  1 | ->  Streaming (type: GATHER)                   |     26 |            |          |       8 | 17.97
  2 |    ->  Nested Loop (3,5)                       |     26 |            | 1MB      |       8 | 11.97
  3 |       ->  Streaming(type: BROADCAST)           |      2 |            | 2MB      |       4 | 2.78
  4 |          ->  Seq Scan on public.t2             |      1 |            | 1MB      |       4 | 2.62
  5 |       ->  Index Scan using t1_b_idx on public.t1 |     26 |            | 1MB      |       8 | 9.05
(5 rows)

 Predicate Information (identified by plan id)
```

```
------------------------------------------------
  4 --Seq Scan on public.t2
        Filter: (t2.a = 4)
  5 --Index Scan using t1_b_idx on public.t1
        Index Cond: (t1.b = t2.b)
(4 rows)

 Targetlist Information (identified by plan id)
------------------------------------------------
  1 --Streaming (type: GATHER)
        Output: t1.a, t1.b
        Node/s: All datanodes
  2 --Nested Loop (3,5)
        Output: t1.a, t1.b
  3 --Streaming(type: BROADCAST)
        Output: t2.b
        Spawn on: datanode2
        Consumer Nodes: All datanodes
  4 --Seq Scan on public.t2
        Output: t2.b
        Distribute Key: t2.a
  5 --Index Scan using t1_b_idx on public.t1
        Output: t1.a, t1.b
        Distribute Key: t1.a
(15 rows)

   ====== Query Summary =====
---------------------------------
 System available mem: 9262694KB
 Query Max mem: 9471590KB
 Query estimated mem: 5144KB
(3 rows)
```

如果优化器没有选择这种执行计划，可以通过以下方式优化：

```
set enable_index_nestloop = on;
set enable_hashjoin = off;
set enable_seqscan = off;
```

场景五：使用 hint 指定索引时指定的索引方式不对

DWS的plan hint当前支持指定的Scan方式有三种：tablescan、indexscan和indexonlyscan。

- tablescan：全表扫描，比如行存表的Seq Scan，列存表的CStore Scan。
- indexscan：先扫索引，再根据索引取表记录。
- indexonlyscan：覆盖索引扫描，所需的返回结果能被所扫描的索引全部覆盖。与index scan相比，index only scan所包含的字段集合，涵盖了查询语句中的字段，这样，提取出相应的index就不必再根据索引取表记录了。

因此，对于需要indexonlyscan的场景，如果hint指定了indexscan，该hint是无法生效的：

```
explain verbose select/*+ indexscan(test)*/ b from test where b = '1';
WARNING:  unused hint: IndexScan(test)
                                      QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 id |                    operation                    | E-rows | E-distinct | E-memory | E-width | E-costs
----+-------------------------------------------------+--------+------------+----------+---------+---------
  1 | ->  Streaming (type: GATHER)                    |      1 |            |          |      32 | 16.27
  2 |    ->  Index Only Scan using index_b on public.test |      1 |            | 1MB      |      32 | 8.27

  Predicate Information (identified by plan id)
------------------------------------------------
  2 --Index Only Scan using index_b on public.test
        Index Cond: (test.b = '1'::text)
```



```
  Targetlist Information (identified by plan id)
--------------------------------------------------
  1 --Streaming (type: GATHER)
        Output: b
        Node/s: All datanodes
  2 --Index Only Scan using index_b on public.test
        Output: b
        Distribute Key: a

  ====== Query Summary =====
-------------------------------
System available mem: 3358720KB
Query Max mem: 3358720KB
Query estimated mem: 1024KB
(24 rows)
explain verbose select/*+ indexonlyscan(test)*/ b from test where b = '1';
                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------
 id |                    operation                     | E-rows | E-distinct | E-memory | E-width | E-costs
----+--------------------------------------------------+--------+------------+----------+---------+---------
  1 | ->  Streaming (type: GATHER)                     |      1 |            |          |      32 | 16.27
  2 |    ->  Index Only Scan using index_b on public.test |      1 |            | 1MB      |      32 | 8.27

  Predicate Information (identified by plan id)
--------------------------------------------------
  2 --Index Only Scan using index_b on public.test
        Index Cond: (test.b = '1'::text)

  Targetlist Information (identified by plan id)
--------------------------------------------------
  1 --Streaming (type: GATHER)
        Output: b
        Node/s: All datanodes
  2 --Index Only Scan using index_b on public.test
        Output: b
        Distribute Key: a

  ====== Query Summary =====
-------------------------------
System available mem: 3358720KB
Query Max mem: 3358720KB
Query estimated mem: 1024KB
(24 rows)
```

优化方法：使用hint时正确指定indexscan和indexonlyscan。

场景六：全文检索 GIN 索引

为了加速文本搜索，进行全文检索时可以创建GIN索引：

```
CREATE INDEX idxb ON test using gin(to_tsvector('english',b));
```

创建GIN索引时，必须使用to_tsvector的两参数版本，并且只有当查询时也使用了两参数版本，且参数值与索引中相同时，才会使用该索引：

说明

to_tsvector()函数有两个版本，只输入一个参数的版本和输入两个参数的版本。输入一个参数时，系统默认采用default_text_search_config所指定的分词器。创建索引时必须使用to_tsvector的两参数版本，否则索引内容可能不一致。

```
explain verbose select  * from test where to_tsvector(b) @@ to_tsquery('cat') order by 1;
                                QUERY PLAN
-------------------------------------------------------------------------------------------
 id |            operation             | E-rows | E-distinct | E-memory | E-width | E-costs
----+----------------------------------+--------+------------+----------+---------+---------
  1 | ->  Streaming (type: GATHER)     |      2 |            |          |      44 | 22.23
```

```
 2 |    ->  Sort                     |      2 |           | 16MB     |      44 | 14.23
 3 |       ->  Seq Scan on public.test |      1 |           | 1MB      |      44 | 14.21

      Predicate Information (identified by plan id)
-------------------------------------------------------------
 3 --Seq Scan on public.test
       Filter: (to_tsvector(test.b) @@ '''cat'''::tsquery)

Targetlist Information (identified by plan id)
-----------------------------------------------
 1 --Streaming (type: GATHER)
       Output: a, b, c
       Merge Sort Key: test.a
       Node/s: All datanodes
 2 --Sort
       Output: a, b, c
       Sort Key: test.a
 3 --Seq Scan on public.test
       Output: a, b, c
       Distribute Key: a

 ====== Query Summary =====
-------------------------------
System available mem: 3358720KB
Query Max mem: 3358720KB
Query estimated mem: 1024KB
(29 rows)
explain verbose select  * from test where to_tsvector('english',b) @@ to_tsquery('cat') order by 1;
                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------
 id |               operation                | E-rows | E-distinct | E-memory | E-width | E-costs
----+----------------------------------------+--------+------------+----------+---------+---------
  1 | ->  Streaming (type: GATHER)           |      2 |            |          |      44 | 20.03
  2 |    ->  Sort                            |      2 |           | 16MB     |      44 | 12.03
  3 |       ->  Bitmap Heap Scan on public.test |      1 |           | 1MB      |      44 | 12.02
  4 |          ->  Bitmap Index Scan         |      1 |           | 1MB      |       0 | 8.00

                  Predicate Information (identified by plan id)
-------------------------------------------------------------------------------------
 3 --Bitmap Heap Scan on public.test
       Recheck Cond: (to_tsvector('english'::regconfig, test.b) @@ '''cat'''::tsquery)
 4 --Bitmap Index Scan
       Index Cond: (to_tsvector('english'::regconfig, test.b) @@ '''cat'''::tsquery)

Targetlist Information (identified by plan id)
-----------------------------------------------
 1 --Streaming (type: GATHER)
       Output: a, b, c
       Merge Sort Key: test.a
       Node/s: All datanodes
 2 --Sort
       Output: a, b, c
       Sort Key: test.a
 3 --Bitmap Heap Scan on public.test
       Output: a, b, c
       Distribute Key: a

 ====== Query Summary =====
-------------------------------
System available mem: 3358720KB
Query Max mem: 3358720KB
Query estimated mem: 2048KB
(32 rows)
```

优化方式：查询时使用to_tsvector的两参数版本，且保证参数值与索引中相同。

5.13 如何使用 DWS 自定义函数改写 CRC32()函数

DWS目前未内置CRC32函数，但如果需要实现MySQL中的CRC32()函数功能，用户可使用DWS的自定义函数语句对其进行改写。

- 函数：CRC32(expr)
- 描述：用于计算循环冗余值。入参expr为字符串。如果参数为NULL，则返回NULL；否则，在计算冗余后返回32位无符号值。

DWS的自定义函数语句改写CRC32函数示例：

```
CREATE OR REPLACE FUNCTION crc32(text_string text) RETURNS bigint AS $$
DECLARE
    val bigint;
    i int;
    j int;
    byte_length int;
    binary_string bytea;
BEGIN
    IF text_string is null THEN
        RETURN null;
    ELSIF text_string = '' THEN
        RETURN 0;
    END IF;

    i = 0;
    val = 4294967295;
    byte_length = bit_length(text_string) / 8;
    binary_string = decode(replace(text_string, E'\\', E'\\\\'), 'escape');
    LOOP
        val = (val # get_byte(binary_string, i))::bigint;
        i = i + 1;
        j = 0;
        LOOP
            val = ((val >> 1) # (3988292384 * (val & 1)))::bigint;
            j = j + 1;
            IF j >= 8 THEN
                EXIT;
            END IF;
        END LOOP;
        IF i >= byte_length THEN
            EXIT;
        END IF;
    END LOOP;
    RETURN (val # 4294967295);
END
$$ IMMUTABLE LANGUAGE plpgsql;
```

验证改写后的结果：

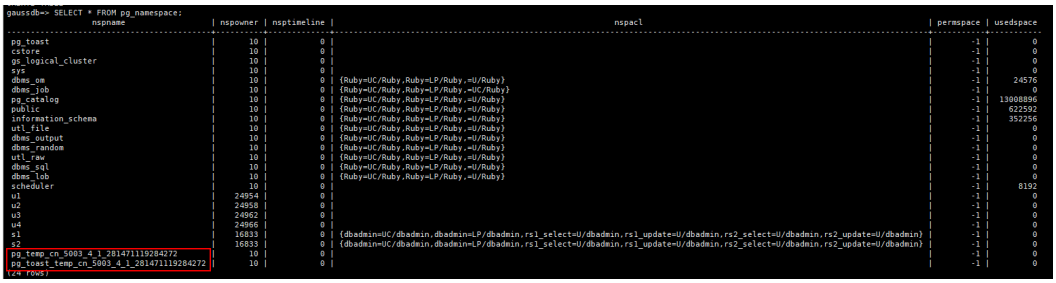
```
select crc32(null),crc32(''),crc32('1');
 crc32 | crc32 |   crc32
-------+-------+------------
       |     0 | 2212294583
(1 row)
```

有关自定义函数的更多用法，可参考**CREATE FUNCTION**章节。

5.14 DWS 以 pg_toast_temp*或 pg_temp*开头的 Schema 是什么?

查询Schema列表的时候，发现查询结果存在pg_temp*或pg_toast_temp*的Schema，如下图所示。

```
SELECT * FROM pg_namespace;
```



这些Schema是在创建临时表时，该临时表通过每个会话独立的以pg_temp开头的Schema来保证只对当前会话可见，因此，不建议用户在日常操作中手动删除以pg_temp，pg_toast_temp开头的Schema。

临时表只在当前会话可见，本会话结束后会自动删除，这些相应的Schema也会被删除。

5.15 DWS 查询时结果不一致的常见场景和解决方法

DWS中使用SQL语句查询时会出现同一条查询语句返回结果不一致的问题，此类问题大部分都是由于语法使用不当或用法不合理导致，通过合理的业务使用可以避免此类问题。以下列举了几种常见的查询结果不一致的场景和对应的解决办法供参考：

窗口函数中使用排序后取部分结果

场景：

窗口函数row_number()中使用排序后查询表t3的c列，两次查询结果不同。

```
SELECT * FROM t3 order by 1,2,3;
 a | b | c
---+---+---
 1 | 2 | 1
 1 | 2 | 2
 1 | 2 | 3
(3 rows)

SELECT c,rn FROM (select c,row_number() over(order by a,b) as rn from t3) where rn = 1;
 c | rn
---+----
1  |  1
(1 row)
SELECT c,rn FROM (select c,row_number() over(order by a,b) as rn from t3) where rn = 1;
 c | rn
---+----
 3 |  1
(1 row)
```

原因分析：

如上所示，执行同一条语句：*select c,rn from (select c,row_number() over(order by a,b) as rn from t3) where rn = 1;* 两次查询结果不同，因为在窗口函数的排序列a、b上存在重复值1、2且重复值在c列上的值不同，就会导致每次按照a，b列排序结果取第一条时，所取的数据是随机的，造成结果集不一致。

解决方法：

该场景需要将取值列c列也加到排序中，使排序结果获取的第一条数据固定。

```
SELECT c,rn FROM (select c,row_number() over(order by a,b,c) as rn from t3) where rn = 1;
 c | rn
---+----
 1 |  1
(1 row)
```

子视图/子查询中使用排序

场景：

创建表test和视图v后，子查询中使用排序查询表test，出现查询结果不一致。

```
CREATE TABLE test(a serial ,b int);
INSERT INTO test(b) VALUES(1);
INSERT INTO test(b) SELECT b FROM test;
...
INSERT INTO test(b) SELECT b FROM test;
CREATE VIEW v as SELECT * FROM test ORDER BY a;
```

问题SQL：

```
SELECT * FROM v limit 1;
 a | b
---+---
 3 | 1
(1 row)

SELECT * FROM (select * from test order by a) limit 10;
 a  | b
----+---
 14 | 1
(1 row)

SELECT * FROM test order by a limit 10;
 a | b
---+---
 1 | 1
(1 row)
```

原因分析：

在子视图和子查询中，ORDER BY子句是无效的。

解决方法：

不建议在子视图和子查询中使用order by，若要保证结果有序，需在最外层查询中使用order by。

子查询 limit

场景：子查询中使用limit，两次查询结果不一致。

```
SELECT * FROM (select a from test limit 1 ) order by 1;
 a
---
 5
(1 row)
```

```
SELECT * FROM (select a from test limit 1 ) order by 1;
 a
---
 1
(1 row)
```

原因分析：

子查询中的limit会导致获取随机结果，从而最终查询结果为随机提取。

解决方法：

要保证最终查询结果的稳定，需避免在子查询中使用limit。

使用 string_agg

场景：使用string_agg查询表employee，出现查询结果不一致。

```
SELECT * FROM employee;
 empno | ename  |   job   | mgr |      hiredate       |  sal  | comm | deptno
-------+--------+---------+-----+---------------------+-------+------+--------
  7654 | MARTIN | SALEMAN | 7698 | 2022-11-08 00:00:00 | 12000 | 1400 |     30
  7566 | JONES  | MANAGER | 7839 | 2022-11-08 00:00:00 | 32000 |    0 |     20
  7499 | ALLEN  | SALEMAN | 7698 | 2022-11-08 00:00:00 | 16000 |  300 |     30
(3 rows)

SELECT count(*) FROM (select deptno, string_agg(ename, ',') from employee group by deptno) t1, (select
deptno, string_agg(ename, ',') from employee group by deptno) t2 where t1.string_agg = t2.string_agg;
 count
-------
     2
(1 row)

SELECT count(*) FROM (select deptno, string_agg(ename, ',') from employee group by deptno) t1, (select
deptno, string_agg(ename, ',') from employee group by deptno) t2 where t1.string_agg = t2.string_agg;
 count
-------
     1
(1 row)
```

原因分析：

String_agg函数的作用是将组内的数据合并成一行，但是如果某用户的用法是string_agg(ename, ',') ，结果集就是不稳定的，因为没有指定组合的顺序。例如，上述语句中，对于*select deptno, string_agg(ename, ',') from employee group by deptno;*

输出结果既可以是：

```
30 | ALLEN,MARTIN
```

也可能是：

```
30 |MARTIN,ALLEN
```

两个结果都是合理的，因此上述关联场景下，有可能出现t1这个subquery中的结果和t2这个subquery中的结果对于deptno=30时的输出结果不一致。

解决方法：

String_agg中增加order by排序，保证按顺序拼接。

```
SELECT count(*) FROM (select deptno, string_agg(ename, ',' order by ename desc) from employee group by
deptno) t1 ,(select deptno, string_agg(ename, ',' order by ename desc) from employee group by deptno) t2
where t1.string_agg = t2.string_agg;
```


数据库兼容模式

场景：在数据库中查询空串结果不一致。

database1（TD兼容模式）：

```
td=# select '' is null;
 isnull
--------
 f
(1 row)
```

database2（ORA兼容模式）：

```
ora=# select '' is null;
 isnull
--------
 t
(1 row)
```

原因分析：

查询空串结果不同是由于不同数据库兼容模式下空串与null语法有差异导致。

目前，DWS支持三种数据库兼容模式：Oracle、TD和MySql，不同兼容模式下语法和行为存在差异，兼容性差异说明可参考**Oracle、Teradata和MySQL语法兼容性差异**。

不同兼容模式下的database表现出不同的兼容性行为属于正常现象。可以通过查看*select datname, datcompatibility from pg_database;*确认数据库兼容性设置是否相同。

解决方法：

这种场景下只能将两个database的兼容性模式设置为一致的才能解决。Database的DBCOMPATIBILITY属性不支持ALTER，只能通过新建数据库的方法，在创建数据库时指定相同的DBCOMPATIBILITY属性解决。

数据库兼容性行为配置项 behavior_compat_options 设置不同

场景：add_months函数计算结果不一致。

database1:

```
SELECT add_months('2018-02-28',3) from dual;
add_months
---------------------
2018-05-28 00:00:00
(1 row)
```

database2:

```
SELECT add_months('2018-02-28',3) from dual;
add_months
---------------------
2018-05-31 00:00:00
(1 row)
```

原因分析：

数据库兼容性配置项behavior_compat_options不同会导致部分行为不同，该参数选项可参考**behavior_compat_options**中的相关选项描述。

此场景中behavior_compat_options配置项中的end_month_calculate参数控制add_months函数计算逻辑配置项。设置end_month_calculate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result的月末日期保持一致。

解决方法：

需要将数据库中参数behavior_compat_options的兼容性配置项设置为一致。该参数类型为USERSET类型，可session级别设置或集群级修改。

自定义函数属性设置不合理

场景：自定义函数get_count()并调用该函数出现结果不一致场景。

```
CREATE FUNCTION get_count() returns int
SHIPPABLE
as $$
declare
    result int;
begin
    result = (select count(*) from test);  --test表是hash表
    return result;
end;
$$
language plpgsql;
```

调用该函数。

```
SELECT get_count();
 get_count
-----------
      2106
(1 row)

SELECT get_count() FROM t_src;
 get_count
-----------
      1032
(1 row)
```

原因分析：

由于该函数指定了SHIPPABLE的函数属性，因此生成计划时该函数会下推到DN上执行，该函数下推到DN后，由于函数定义中的test表是hash表，因此每个DN上只有该表的一部分数据，所以select count(*) from test; 返回的结果不是test表全量数据的结果，而是每个DN上部分数据的结果，因此导致加上from表后函数返回预期发生变化。

解决方法：

以下两种方法任选其一即可（推荐第一种方法）：

1. 将函数改为不下推：ALTER FUNCTION get_count() not shippable;
2. 将函数中用到的表改为复制表，这样每个DN上都是一份该表的全量数据，即使下推到DN执行，也能保证结果集符合预期。

使用 UNLOGGED 表

场景：

使用unlogged表后，在集群重启后，关联查询结果集异常，查看unlogged表缺少部分数据。

原因分析：

如果设置max_query_retry_times为0，且在建表时指定UNLOGGED关键字，则创建的表为非日志表。在非日志表中写入的数据不会被写入到预写日志中，这样就会比普通

表快很多。但是非日志表在冲突、执行操作系统重启、强制重启、切断电源操作或异常关机后会被自动截断，会造成数据丢失的风险。非日志表中的内容也不会被复制到备服务器中。在非日志表中创建的索引也不会被自动记录。因此当集群发生异常重启（进程重启、节点故障、集群重启）时，会导致部分内存中的数据未及时落盘，造成部分数据丢失，从而导致结果集异常。

解决方法：

unlogged表在集群异常情况下的安全性无法保证，一般不能作为业务表使用，更多的场景是作为临时表使用。当出现集群故障后，为了保证数据正常，需要重建unlogged表或将数据备份后重新导入数据库。

5.16 DWS 哪些系统表不能做 VACUUM FULL

从功能实现上，DWS的系统表都可以做VACUUM FULL，但是会上八级锁，涉及这些系统表的业务会被阻塞。

根据数据库版本不同，建议如下：

8.1.3 及以上版本

- 8.1.3及以上版本的集群，AUTO VACUUM默认是打开的（由GUC参数**autovacuum**控制），用户通过设置对应的GUC参数后，系统会自动触发所有系统表和用户的**行存表**进行VACUUM FULL，用户不需要手动执行vacuum。
  - **autovacuum_max_workers** = 0，系统表和普通表都不会触发。
  - **autovacuum** = off，普通表不会触发，但系统表会触发。
- 以上仅针对**行存表**的AUTO VACCUM触发，如果需要针对**列存表**做自动触发VACUUM，还需要用户在管理控制台上配置智能调度任务。具体参见**运维计划**。

8.1.1 及历史版本

1. 以下系统表在做VACUUM FULL时会影响所有业务，请选择空闲时间窗或停止业务时操作。
   - pg_statistic（统计信息，建议不要清理，会影响业务查询性能）
   - pg_attribute
   - pgxc_class
   - pg_type
   - pg_depend
   - pg_class
   - pg_index
   - pg_proc
   - pg_partition
   - pg_object
   - pg_shdepend
2. 以下系统表主要影响资源的监控和表大小的查询接口，不影响其他业务。
   - gs_wlm_user_resource_history
   - gs_wlm_session_info

- gs_wlm_instance_history
- gs_respool_resource_history
- pg_relfilenode_size

3. 其余系统表不占用空间，通常不做清理。
4. 建议日常运维活动中，每周监控以下系统表的大小，如果一定要回收这些空间，优先处理关键系统表。

   语句如下：

   ```
   SELECT c.oid,c.relname, c.relkind, pg_relation_size(c.oid) AS size  FROM pg_class c WHERE c.relkind IN ('r') AND c.oid <16385 ORDER BY size DESC;
   ```

5.17 DWS 语句处于 idle in transaction 状态常见场景

在使用PGXC_STAT_ACTIVITY视图查询用户SQL相关信息时，查询结果中的state字段有时会显示“idle in transaction”。idle in transaction具体含义为：后端在事务中，但事务中没有语句在执行。该状态表示该条语句已经执行完成，因此query_id为0，但是本事务还未提交或回滚。此状态下的语句已经执行完成，不占用CPU和IO等资源，会占用连接数，并发数等连接资源。

若业务中出现语句处于idle in transaction状态，可参考如下常见场景及对应的解决方法来处理：

场景一：事务开启后没有提交，语句处于 idle in transaction

手动BEGIN/START TRANSACTION开启事务，执行某语句后，不执行COMMIT/ROLLBACK，此时执行如下命令查看视图PGXC_STAT_ACTIVITY：

```
SELECT state, query, query_id FROM pgxc_stat_activity;
```

查看结果显示：该语句状态为idle in transaction。

```
       state        |                            query                             |      query_id
--------------------+--------------------------------------------------------------+--------------------
 active             |                                                              |                 0
 idle               |                                                              |                 0
 idle               |                                                              |                 0
 active             | WLM fetch collect info from data nodes                       |  73464968921613282
 active             | WLM calculate space info process                             |                 0
 active             | WLM monitor update and verify local info                     |  73464968921613276
 active             | WLM arbiter sync info by CCN and CNs                         |                 0
 idle in transaction | select count(1) from t group by a order by 1 desc limit 1;  |                 0
 idle               |                                                              |                 0
 active             | select state,query,query_id from pgxc_stat_activity;         |  73464968921613283
 active             |                                                              |                 0
 idle               |                                                              |                 0
 idle               |                                                              |                 0
 active             | WLM fetch collect info from data nodes                       | 145522562959541153
 active             | WLM calculate space info process                             |                 0
 active             | WLM monitor update and verify local info                     | 145522562959541123
 active             | WLM arbiter sync info by CCN and CNs                         |                 0
 active             | SELECT * FROM pg_stat_activity                               |  73464968921613283
 idle               |                                                              |                 0
(19 rows)
```

解决方法：这种场景下需要手动对开启的事务执行COMMIT/ROLLBACK即可。

场景二：存储过程中有 DDL 语句，该存储过程结束前，其他节点上 DDL 语句执行完后的状态是 idle in transaction

先创建存储过程：

```
CREATE OR REPLACE FUNCTION public.test_sleep()
RETURNS void
LANGUAGE plpgsql
AS $$

BEGIN
   truncate t1;
   truncate t2;
   EXECUTE IMMEDIATE 'select pg_sleep(6)';
   RETURN;
END$$;
```

再执行如下命令查看PGXC_STAT_ACTIVITY视图：

```
SELECT coorname,pid,query_id,state,query,usename FROM pgxc_stat_activity WHERE usename='jack';
```

查看结果显示：truncate t2处于idle in transaction状态，coorname为coordinator2。说明cn2上该语句已经执行完成，该存储过程在执行下一条语句。

```
   coorname   |       pid       |     query_id      |        state        |        query         | usename
--------------+-----------------+-------------------+---------------------+----------------------+---------
 coordinator1 | 139767124588288 | 73464968921614213 | active              | select test_sleep(); | jack
 coordinator2 | 140055318353664 |                 0 | idle in transaction | truncate t2          | jack
(2 rows)
```

解决方法：此类场景是由于存储过程执行慢导致，等存储过程执行完成即可，也可考虑优化存储过程中执行时间较长的语句。

场景三：大量 SAVEPOINT/RELEASE 语句处于 idle in transaction（8.1.0 之前集群版本）

执行如下命令查看PGXC_STAT_ACTIVITY视图：

```
SELECT coorname,pid,query_id,state,query,usename FROM pgxc_stat_activity WHERE usename='jack';
```

结果显示SAVEPOINT/RELEASE语句处于idle in transaction。

```
   coorname   |       pid       |     query_id      |        state        |        query         | usename
--------------+-----------------+-------------------+---------------------+----------------------+---------
 coordinator1 | 140127877723904 | 77687093572141691 | active              | select test_sleep1(); | jack
 coordinator2 | 139773127153408 |                 0 | idle in transaction | release s1           | jack
 coordinator3 | 140193352906496 |                 0 | idle in transaction | release s1           | jack
(3 rows)
```

解决方法：

SAVEPOINT和RELEASE语句是带EXCEPTION的存储过程执行时系统自动生成的（8.1.0之后的集群版本不再向CN下发SAVEPOINT），DWS带EXCEPTION的存储过程在实现上基于子事务实现，简单对应关系如下：

```
begin
   (Savepoint s1)
   DDL/DML
exception
   (Rollback to s1)
   (Release s1)
    ...
end
```

存储过程启动时如果有EXCEPTION，则会启动一个子事务，如果执行过程中出现EXCEPTION，则会回滚当前事务并进行异常的处理；如果没有出现EXCEPTION则会直接提交前面的子事务。

当此类存储过程较多且有嵌套时容易出现，与场景二类似，等整个存储过程执行完即可。如果RELEASE较多，说明存储过程触发了多个EXCEPTION，可分析存储过程逻辑是否合理。

5.18 DWS 如何实现行转列及列转行?

本节介绍DWS中如何使用SQL语句实现行转列、列转行。

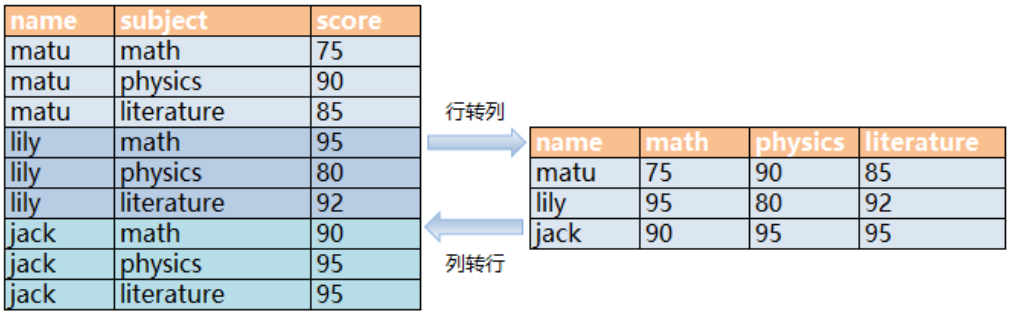
场景介绍

以学生成绩为例：

老师会按照学科录入成绩，每科老师都会单独录入每个学生对应学科的成绩，而每位学生只关注自己各科的成绩。如果把老师录入数据作为原始表，那么学生查看自己的成绩就要用到行转列；如果让学生自己填写各科的成绩并汇总，然后老师去查自己学科所有学生的成绩，那就是列转行。

行转列与列转行的示意图如下：

图 5-1 示意图



- 行转列

  将多行数据转换成一行显示，或将一列数据转换成多列显示。
- 列转行

  将一行数据转换成多行显示，或将多列数据转换成一列显示。

示例表

- 创建行存表students_info并插入数据。

```
CREATE TABLE students_info(name varchar(20),subject varchar(100),score bigint) distribute by hash(name);
INSERT INTO students_info VALUES('lily','math',95);
INSERT INTO students_info VALUES('lily','physics',80);
INSERT INTO students_info VALUES('lily','literature',92);
INSERT INTO students_info VALUES('matu','math',75);
INSERT INTO students_info VALUES('matu','physics',90);
INSERT INTO students_info VALUES('matu','literature',85);
INSERT INTO students_info VALUES('jack','math',90);
INSERT INTO students_info VALUES('jack','physics',95);
INSERT INTO students_info VALUES('jack','literature',95);
```

  查看表students_info信息。

```
SELECT * FROM students_info;
 name |  subject   | score
------+------------+-------
 matu | math       |    75
 matu | physics    |    90
 matu | literature |    85
```

```
 lily | math       |    95
 lily | physics    |    80
 lily | literature |    92
 jack | math       |    90
 jack | physics    |    95
 jack | literature |    95
```

- 创建列存表students_info1并插入数据。

```
CREATE TABLE students_info1(name varchar(20), math bigint, physics bigint, literature bigint) with
(orientation = column) distribute by hash(name);
INSERT INTO students_info1 VALUES('lily',95,80,92);
INSERT INTO students_info1 VALUES('matu',75,90,85);
INSERT INTO students_info1 VALUES('jack',90,95,95);
```

查看表students_info1信息。

```
SELECT * FROM students_info1;
 name | math | physics | literature
------+------+---------+------------
 matu |   75 |      90 |         85
 lily |   95 |      80 |         92
 jack |   90 |      95 |         95
(3 rows)
```

静态行转列

静态行转列需要手动指定每一列的列名，如果存在则取其对应值，否则将赋其默认值0。

```
SELECT name,
sum(case when subject='math' then score else 0 end) as math,
sum(case when subject='physics' then score else 0 end) as physics,
sum(case when subject='literature' then score else 0 end) as literature FROM students_info GROUP BY
name;
 name | math | physics | literature
------+------+---------+------------
 matu |   75 |      90 |         85
 lily |   95 |      80 |         92
 jack |   90 |      95 |         95
(3 rows)
```

动态行转列

8.1.2及以上集群版本可使用GROUP_CONCAT生成列存语句。

```
SELECT group_concat(concat('sum(IF(subject = ''', subject, ''', score, 0)) AS "', name, '"'))FROM students_info;
      group_concat

------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------
------------------------
 sum(IF(subject = 'literature', score, 0)) AS "jack",sum(IF(subject = 'literature', score, 0)) AS
"lily",sum(IF(subject = 'literature', score, 0)) AS "matu",sum(IF(subject = 'math', score, 0)) AS "jack",sum(IF
(subject = 'math', score, 0)) AS "lily",sum(IF(subject = 'math', score, 0)) AS "matu",sum(IF(subject =
'physics', score, 0)) AS "jack",sum(IF(subject = 'physics', score, 0)) AS "lily",sum(IF(subject = 'physics
', score, 0)) AS "matu"
(1 row)
```

8.1.1及更低版本中可用LISTAGG生成列存语句。

```
SELECT listagg(concat('sum(case when subject = ''', subject, ''' then score else 0 end) AS "', subject, '"'),',')
within GROUP(ORDER BY 1)FROM (select distinct subject from students_info);
                                                                  listagg

------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

```
--
 sum(case when subject = 'literature' then score else 0 end) AS "literature",sum(case when subject =
'physics' then score else 0 end) AS "physics",sum(case when subject = 'math' then score else 0 end) AS
"math
"
(1 row)
```

再通过视图动态重建：

```
CREATE OR REPLACE FUNCTION build_view()
RETURNS VOID
LANGUAGE plpgsql
AS $$ DECLARE
sql text;
rec record;
BEGIN
sql := 'select LISTAGG(
    CONCAT( ''sum(case when subject = ''''', subject, ''''' then score else 0 end) AS "", subject, ''"'' )
    ,'','' ) within group(order by 1) from (select distinct subject from students_info);';
    EXECUTE sql INTO rec;
    sql := 'drop view if exists get_score';
    EXECUTE sql;
    sql := 'create view get_score as select name, ' || rec.LISTAGG || ' from students_info group by name';
    EXECUTE sql;
END$$;
```

执行重建：

```
CALL build_view();
```

查询视图：

```
SELECT * FROM get_score;
 name | literature | physics | math
------+------------+---------+------
 matu |         85 |      90 |   75
 lily |         92 |      80 |   95
 jack |         95 |      95 |   90
(3 rows)
```

列转行

使用union all，将各科目（math、physics和literature）整合为一列，示例如下：

```
SELECT * FROM
(
SELECT name, 'math' AS subject, math AS score FROM students_info1
union all
SELECT name, 'physics' AS subject, physics AS score FROM students_info1
union all
SELECT name, 'literature' AS subject, literature AS score FROM students_info1
)
order by name;
 name |  subject   | score
------+------------+-------
 jack | math       |    90
 jack | physics    |    95
 jack | literature |    95
 lily | math       |    95
 lily | physics    |    80
 lily | literature |    92
 matu | math       |    75
 matu | physics    |    90
 matu | literature |    85
(9 rows)
```

5.19 DWS 唯一约束和唯一索引有什么区别?

- 唯一约束和唯一索引概念上不同

  唯一约束确保一列或者一组列中包含的数据对于表中所有的行都是唯一的。 如果没有声明DISTRIBUTE BY REPLICATION，则唯一约束的列集合中必须包含分布列。

  唯一索引用于限制索引字段值的唯一性，或者是多个字段组合值的唯一性。CREATE UNIQUE INDEX创建唯一索引。

- 唯一约束和唯一索引功能上不同

  约束主要是为了保证数据的完整性，索引主要是为了辅助查询。

- 唯一约束和唯一索引使用方法上不同

  a. 唯一约束和唯一索引，都可以实现列数据的唯一，列值可以有NULL。

  b. 创建唯一约束，会自动创建一个同名的唯一索引，该索引不能单独删除，删除约束会自动删除索引。唯一约束是通过唯一索引来实现数据的唯一。DWS行存表支持唯一约束，而列存表不支持。

  c. 创建一个唯一索引，这个索引独立的、可以单独删除。目前，DWS只有B-Tree可以创建唯一索引。

  d. 如果一个列上想有约束和索引，且两者可以单独的删除。可以先建唯一索引，再建同名的唯一约束。

  e. 如果表的一个字段，要作为另外一个表的外键，这个字段必须有唯一约束（或是主键），如果只是有唯一索引，就会报错。

示例：创建两个列的复合索引，并不要求是唯一索引。

```
CREATE TABLE t (n1 number,n2 number,n3 number,PRIMARY KEY (n3));
CREATE INDEX t_idx ON t(n1,n2);
```

DWS支持多个唯一索引。

```
CREATE UNIQUE INDEX u_index ON t(n3);
CREATE UNIQUE INDEX u_index1 ON t(n3);
```

可以使用上述示例创建的索引t_idx来创建唯一约束t_uk，而且它只在列n1上唯一，也就是说唯一约束比索引更加严格。

```
ALTER TABLE t ADD CONSTRAINT t_uk UNIQUE USING INDEX u_index;
```

5.20 DWS 函数和存储过程有什么区别?

函数和存储过程是数据库管理系统中常见的两种对象，它们在实现特定功能时具有相同点，也有不同点。了解它们的特点和适用场景，对于合理设计数据库结构和提高数据库性能具有重要意义。

表 5-2 函数和存储过程的区别

函数	存储过程
两者都可以用于实现特定的功能。无论是函数还是存储过程，都可以封装一系列的SQL语句，以完成某些特定的操作。	

函数	存储过程
两者都可以接收输入参数，并且根据参数的不同来进行相应的操作。	
函数的标识符为FUNCTION。	存储过程的标识符为PROCEDURE。
函数必须返回一个具体的值，并且规定返回值的数值类型。	存储过程可以没有返回值，也可以有返回值，甚至可以有多个返回值，可以通过输出参数返回结果，也可以直接在存储过程中使用SELECT语句返回结果集。
函数适用于需要返回单个值的情况，比如计算某个数值、字符串处理、返回表等。	存储过程适用于需要执行DML操作的情况，比如批量插入、更新、删除数据等。

- **创建并调用函数**

  创建表emp并插入数据，查询表数据如下：

```
SELECT * FROM emp;
 empno | ename |   job    | mgr |      hiredate       |   sal   |  comm  | deptno
-------+-------+----------+-----+---------------------+---------+--------+--------
  7369 | SMITH | CLERK    | 7902 | 1980-12-17 00:00:00 |  800.00 |        |     20
  7499 | ALLEN | SALESMAN | 7698 | 1981-02-20 00:00:00 | 1600.00 | 300.00 |     30
  7566 | JONES | MANAGER  | 7839 | 1981-04-02 00:00:00 | 2975.00 |        |     20
  7521 | WARD  | SALESMAN | 7698 | 1981-02-22 00:00:00 | 1250.00 | 500.00 |     30
(4 rows)
```

  创建函数emp_comp，用于接受两个数字作为输入并返回计算值：

```
CREATE OR REPLACE FUNCTION emp_comp (
    p_sal           NUMBER,
    p_comm          NUMBER
) RETURN NUMBER
IS
BEGIN
    RETURN (p_sal + NVL(p_comm, 0)) * 24;
END;
/
```

  使用SELECT命令调用函数：

```
SELECT ename "Name", sal "Salary", comm "Commission", emp_comp(sal, comm) "Total
Compensation" FROM emp;
 Name  | Salary  | Commission | Total Compensation
-------+---------+------------+--------------------
 SMITH |  800.00 |            |           19200.00
 ALLEN | 1600.00 |     300.00 |           45600.00
 JONES | 2975.00 |            |           71400.00
 WARD  | 1250.00 |     500.00 |           42000.00
(4 rows)
```

- **创建并调用存储过程**

  创建表MATCHES并插入数据，查询表数据如下：

```
SELECT * FROM MATCHES;
 matchno | teamno | playerno | won | lost
---------+--------+----------+-----+------
       1 |      1 |        6 |   3 |    1
       7 |      1 |       57 |   3 |    0
       8 |      1 |        8 |   0 |    3
       9 |      2 |       27 |   3 |    2
      11 |      2 |      112 |   2 |    3
(5 rows)
```

  创建存储过程delete_matches，用于删除给定球员参加的所有比赛：

```
CREATE PROCEDURE delete_matches(IN p_playerno INTEGER)
AS
BEGIN
  DELETE FROM MATCHES WHERE playerno = p_playerno;
END;
/
```

调用存储过程delete_matches：

```
CALL delete_matches(57);
```

再次查询表MATCHES，由返回结果可知，playerno为57的数据已被删除：

```
SELECT * FROM MATCHES;
 matchno | teamno | playerno | won | lost
---------+--------+----------+-----+------
      11 |      2 |      112 |   2 |    3
       8 |      1 |        8 |   0 |    3
       1 |      1 |        6 |   3 |    1
       9 |      2 |       27 |   3 |    2
(4 rows)
```

5.21 DWS 字符截取函数 substrb()、substr()及substring()的用法及差异

DWS支持字符截取功能的函数：substrb()、substr()和substring()，这些函数都可以操作字符串截取，但在字符截取时的用法和差异是什么呢，本节进行详细介绍。

函数形式

substrb()、substr()和substring()均为字符串截取函数，都可带两个或三个参数，用于提取字符串中指定截取的开始位置和截取的长度。函数定义如下：

```
substrb(string, from [, count])
substr(string, from [, count])
substring(string, from [, count])
```

参数描述：从参数string中抽取子字符串，from表示抽取的起始位置，count表示抽取的字符串长度。

返回值类型：text

截取单位差异

- substrb()，按字节截取。
- substr()，按字符截取。
- substring()，按字符截取。

以utf8编码为例，1个汉字占3个字节，当使用substrb()截取长度3的子串时，只能截取到一个字符，而substr()/substring()可以截取到三个字符。

示例：

```
SELECT substrb('data数据库',3,5),substr('data数据库',3,5),substring('data数据库',3,5);
 substrb |  substr  | substring
---------+----------+-----------
 ta数    | ta数据库 | ta数据库
```

截取规则差异

DWS目前支持三种兼容模式：ORA、TD和MySQL，在不同兼容模式下，函数差异具体如下：

- substrb()与ORA、TD和MySQL兼容模式行为一致。
  ```
  substrb(string, from [, count])
  ```
  从参数string中抽取子字符串，from表示抽取的起始位置，count表示抽取的字符串长度。

  可以表示为substrb(string, s[, n])：from的起始位置用s表示，count抽取的字符长度用n来表示。

  表 5-3 substrb()与 ORA、TD 和 MySQL 兼容模式行为

参数场景	ORA/TD/MySQL兼容	示例
s > 0	从第s个字符位置开始，截取n个字节	select substrb('data数据库',5,3); substrb --------- 数 (1 row)
s = 0	截取前n个字节	select substrb('data数据库',0,3); substrb --------- dat (1 row)
s < 0	从倒数第|s|个字节位置开始，截取n个字节	select substrb('data数据库',-3,3); substrb --------- 库 (1 row)
n > 0	截取n个字节	select substrb('data数据库',1,3); substrb --------- dat (1 row)
n <= 0	空串(ORA模式输出null)	select substrb('data数据库',5,0),substrb('data数据库',5,-1); substrb | substrb ---------+--------- | (1 row)

- substr()与ORA、TD和MySQL兼容模式行为差异。
  ```
  substr(string, from [, count])
  ```
  从参数string中抽取子字符串，from表示抽取的起始位置，count表示抽取的字符串长度。

  可以表示为substr(string, s[, n])：from的起始位置用s表示，count抽取的字符长度用n来表示。

  s=0时存在兼容行为差异。

  表 5-4 substr()与 ORA、TD 和 MySQL 兼容模式行为

参数场景	ORA兼容	TD兼容	MySQL兼容
s > 0	从第s个字符位置开始，截取n个字节	从第s个字符位置开始，截取n个字符	从第s个字符位置开始，截取n个字符

参数场景	ORA兼容	TD兼容	MySQL兼容
s = 0	截取前n个字节 ora_db=> select substr('data数据库',0,3); substr -------- dat (1 row)	截取前n个字符 td_db=> select substr('data数据库',0,3); substr -------- dat (1 row)	空串 mysql_db=> select substr('data数据库',0,3); substr -------- (1 row)
s < 0	从倒数第|s|个字节位置开始，截取n个字节	从倒数第|s|个字符位置开始，截取n个字符	从倒数第|s|个字符位置开始，截取n个字符
n > 0	截取n个字节	截取n个字符	截取n个字符
n <= 0	null	空串	空串

- substring()与ORA、TD和MySQL兼容模式行为差异。

  ```
  substring(string, from [, count])
  ```

  从参数string中抽取子字符串，from表示抽取的起始位置，count表示抽取的字符串长度。

  substring(string, s[, n])：from的起始位置用s表示，count抽取的字符长度用n来表示。

  s<=0和n<0时存在兼容行为差异。

  表 5-5 substring()与 ORA、TD 和 MySQL 兼容模式行为

参数场景	ORA兼容	TD兼容	MySQL兼容
s > 0	从第s个字符位置开始，截取n个字节	从第s个字符位置开始，截取n个字符	从第s个字符位置开始，截取n个字符
s = 0	截取前n - 1个字节 ora_db=> select substring('data数据库',0,3); substring ----------- da (1 row)	截取前n - 1个字符 td_db=> select substring('data数据库',0,3); substring ----------- da (1 row)	空串 mysql_db=> select substring('data数据库',0,3); substring ----------- (1 row)
s < 0	截取前s + n - 1个字节 ora_db=> select substring('data数据库',-1,3); substring ----------- d (1 row)	截取前s + n - 1个字节 td_db=> select substring('data数据库',-1,3); substring ----------- d (1 row)	从倒数第|s|个字符位置开始，截取n个字符 mysql_db=> select substring('data数据库',-1,3); substring ----------- 库 (1 row)
n > 0	截取n个字节	截取n个字符	截取n个字符
n = 0	null	空串	空串

参数场景	ORA兼容	TD兼容	MySQL兼容
n < 0	报错 ora_db=> select substring('data数据库',3,-1); ERROR: negative substring length not allowed CONTEXT: referenced column: substring	报错 td_db=> select substring('data数据库',3,-1); ERROR: negative substring length not allowed CONTEXT: referenced column: substring	空串 mysql_db=> select substring('data数据库',3,-1); substring ----------- (1 row)

5.22 如何删除 DWS 重复的表数据?

清理数据库脏数据时，可能会有多条重复数据只保留一条的场景，此场景可以使用聚合函数或窗口函数来实现。

构建表数据

步骤1 创建表t_customer，向表中插入包含重复记录的数据：

```
CREATE TABLE t_customer (
  id int NOT NULL,
  cust_name varchar(32) NOT NULL COMMENT '名字',
  gender varchar(10) NOT NULL COMMENT '性别',
  email varchar(32) NOT NULL COMMENT 'email',
  PRIMARY KEY (id)
) ;

INSERT INTO t_customer VALUES ('1', 'Tom', 'Male', 'high_salary@sample.com');
INSERT INTO t_customer VALUES ('2', 'Jennifer', 'Female', 'good_job@sample.com');
INSERT INTO t_customer VALUES ('3', 'Tom', 'Male', 'high_salary@sample.com');
INSERT INTO t_customer VALUES ('4', 'John', 'Male', 'good_job@sample.com');
INSERT INTO t_customer VALUES ('5', 'Jennifer', 'Female', 'good_job@sample.com');
INSERT INTO t_customer VALUES ('6', 'Tom', 'Male', 'high_salary@sample.com');
```

步骤2 查询表t_customer：

```
SELECT * FROM t_customer ORDER BY id;
```

```
 id | cust_name | gender |         email
----+-----------+--------+-----------------------
  1 | Tom       | Male   | high_salary@sample.com
  2 | Jennifer  | Female | good_job@sample.com
  3 | Tom       | Male   | high_salary@sample.com
  4 | John      | Male   | good_job@sample.com
  5 | Jennifer  | Female | good_job@sample.com
  6 | Tom       | Male   | high_salary@sample.com
```

----结束

当客户的名字、性别、邮件都相同时，则判定它们为重复的记录。对于表t_customer，id等于1、3、6的为重复数据，id为2、5的也是重复数据，删除多余的数据的同时需要保留其中的一条。

方法一：使用聚合函数min(expr)

使用聚合函数通过子查询取出id最小的不重复行，然后通过NOT IN删除重复数据。

步骤1 查询id最小的不重复行：

```
SELECT
   min(id) id,
   cust_name,
   gender,
   COUNT( cust_name ) count
FROM t_customer
GROUP BY cust_name,gender
ORDER BY id;
```

```
 id | cust_name | gender | count
----+-----------+--------+-------
  1 | Tom       | Male   |     3
  2 | Jennifer  | Female |     2
  4 | John      | Male   |     1
```

通过查询结果可知，重复的数据行id为3、5、6的数据被过滤掉了。

步骤2 使用NOT IN过滤重复数据行并删除：

```
DELETE from t_customer where id not in (
   SELECT
   min(id) id
   FROM t_customer
   GROUP BY cust_name,gender
);
```

步骤3 查询删除重复数据后的表t_customer：

```
SELECT * FROM t_customer ORDER BY id;
```

```
 id | cust_name | gender |         email
----+-----------+--------+------------------------
  1 | Tom       | Male   | high_salary@sample.com
  2 | Jennifer  | Female | good_job@sample.com
  4 | John      | Male   | good_job@sample.com
```

由返回结果可知，重复数据已被删除。

----结束

方法二：使用窗口函数row_number()

通过PARTITION BY对列进行分区排序并生成序号列，然后将序号大于1的行删除。

步骤1 分区查询，对列进行分区排序并生成序号列：

```
SELECT
   id,
   cust_name,
   gender,
   ROW_NUMBER() OVER (PARTITION BY cust_name,gender ORDER BY id) num
FROM t_customer;
```

```
 id | cust_name | gender | num
----+-----------+--------+-----
  4 | John      | Male   |   1
  1 | Tom       | Male   |   1
  3 | Tom       | Male   |   2
  6 | Tom       | Male   |   3
  2 | Jennifer  | Female |   1
  5 | Jennifer  | Female |   2
```

由返回结果可知，num>1的数据即为重复数据。

步骤2 删除num>1的数据：

```
DELETE FROM t_customer WHERE id in (
  SELECT id FROM(
   SELECT * FROM (
      SELECT ROW_NUMBER() OVER w AS row_num,id
   FROM t_customer
   WINDOW w AS (PARTITION BY cust_name,gender ORDER BY id) )
  WHERE row_num >1 )
);
```

步骤3 查询删除重复数据后的表t_customer：

```
SELECT * FROM t_customer ORDER BY id;
```

```
 id | cust_name | gender |          email
----+-----------+--------+-------------------------
  1 | Tom       | Male   | high_salary@sample.com
  2 | Jennifer  | Female | good_job@sample.com
  4 | John      | Male   | good_job@sample.com
```

----结束

6 集群管理

6.1 如何清理与回收 DWS 存储空间?

DWS数据仓库中保存的数据在删除后，可能没有释放占用的磁盘空间形成脏数据，导致磁盘浪费、创建及恢复快照性能下降等问题，如何清理?

清理与回收存储空间对系统的影响如下：

- 删除无用的脏数据，释放存储空间。
- 数据库将进行大量读写操作，可能影响正常使用，建议选择空闲时间执行。
- 数据库的存储空间越大，即数据可能越多，清理的时间越长。

定期进行脏数据清理，可以清理与回收存储空间。不同集群版本下操作步骤有所不同，具体如下：

8.1.3 及以上版本：通过管理控制台“智能运维”功能进行启动清理

步骤1 登录**DWS控制台**。

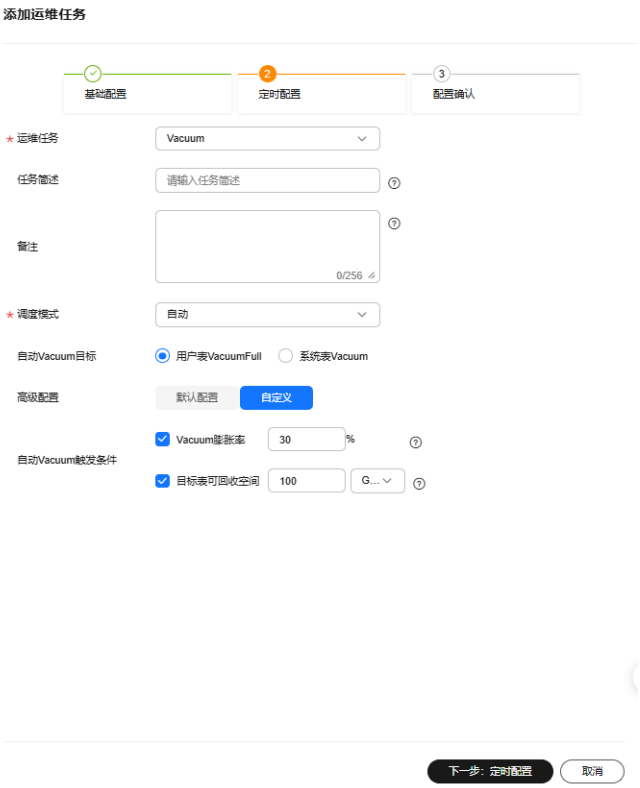
步骤2 在集群列表中单击指定集群名称。

步骤3 进入“集群详情”页面，切换至“智能运维”页签。

步骤4 在运维详情部分切换至运维计划模块。单击“添加运维任务”按钮。

步骤5 弹出添加运维任务边栏，

- 运维任务选择“Vacuum”。
- 调度模式选择“自动”，DWS将自动扫描Vacuum目标。
- Vacuum目标选择系统表或用户表：
  - 如果用户业务UPDATE、DELETE较多，选择用户表。
  - 如果创建表、删除表较多，选择系统表。



步骤6 单击“下一步：定时配置”，配置Vacuum类型，推荐选择“周期型任务”，DWS将自动在自定义时间窗内执行Vacuum。

说明

对于自动Vacuum运维任务，系统对于用户表的处理方法实际采用的是VACUUM FULL操作。VACUUM FULL执行过程中，本身持有8级锁，会阻塞其他业务，导致锁冲突产生，业务本身会陷入锁等待，20分钟后超时报错。因此，在用户配置时间窗内，应尽量避开执行所有处理表的相关业务。

步骤7 确认无误后，单击“下一步：配置确认”，完成配置。

----结束

8.1.2 及以前版本：手动执行 VACUUM FULL 进行清理

须知

1. VACUUM FULL操作**会锁表，**VACUUM FULL期间，该表的所有访问会阻塞，并等待VACUUM FULL结束，请合理安排调度时间，避免锁表影响业务。
2. VACUUM FULL是对当前表的有效数据抽出来重新整理，同时清理脏数据，该操作会临时占用**额外的整理空间**（这部分空间待整理完成后释放），因此空间会先增后降，请提前计算好VACUUM FULL所需要的空间再行处理（**额外的整理空间大小=表大小* (1 – 脏页率)**）。

步骤1 连接数据库，执行以下SQL语句查询脏页率超过30%的较大表，并且按照表大小从大到小排序。

```
SELECT schemaname AS schema, relname AS table_name, n_live_tup AS analyze_count,
pg_size_pretty(pg_table_size(relid)) as table_size, dirty_page_rate
FROM PGXC_GET_STAT_ALL_TABLES
WHERE schemaName NOT IN ('pg_toast', 'pg_catalog', 'information_schema', 'cstore', 'pmk')
AND dirty_page_rate > 30
ORDER BY table_size DESC, dirty_page_rate DESC;
```

步骤2 判断是否有回显结果。

- 是，对于表大小超过10G的表，则执行**步骤3**。
- 否，操作结束。

步骤3 将脏页Top5的表，进行VACUUM FULL清理（清理时，如果最高磁盘空间>70%，请串行清理）。

```
VACUUM FULL ANALYZE schema.table_name;
```

----结束

6.2 为什么 DWS 扩容后已使用存储容量比扩容前减少了很多?

原因分析

扩容前，如果您没有执行vacuum清理和回收存储空间，DWS数据仓库中之前被删除的数据，可能没有释放占用的磁盘空间形成脏数据，导致磁盘浪费。

而在扩容时，系统会做一次重分布，集群扩容时新节点添加完成后，原节点存储的业务数据明显多于新节点，此时系统自动在所有节点重新分布保存数据。在开始做重分

布时，系统会自动执行一次vacuum，从而释放了存储空间，因此，扩容后已使用存储容量减少了很多。

处理方法

建议您定期做vacuum full清理与回收存储空间，防止数据膨胀。

如果执行vacuum后，已使用存储容量仍然占用过高，请分析现有集群规格是否满足业务需求，若不满足，建议您对集群进行扩容。

6.3 DWS 的磁盘空间/容量是如何统计的?

1. DWS的磁盘总容量统计：以存算一体3个数据节点为例，假设每个节点320G，总容量为960G。当存入一个1G的数据，DWS因为副本机制会将这1G的数据在两个节点中都各存一份，共占2G的空间，如果再加上元数据、索引等，实际1G的数据，存入DWS后占用的空间不止2G。所以总容量为960G的3节点集群，总量能存480G的数据。因为存储硬盘本身不贵，客户数据才珍贵。在存算分离场景下，磁盘作为缓存盘，容量统计为真实磁盘容量；3个存算分离的数据节点，每个节点320G，总容量为960G。

   客户在DWS控制台上创建集群的时候，页面已经是按照一个节点的真正容量空间来统计的。比如存算一体dwsx2.xlarge，在创建页面是160G，但实际这个节点的磁盘是有320G的，已经将这个320G显示为160G了，便于客户按实际落盘数据进行创建。如存算分离的dwsx3.4U16G.4DPU规格，在创建页面是320G，申请的磁盘容量也是320，页面显示的容量就是实际申请的真实容量。

2. 单个节点磁盘的使用情况的确认：

   同样的，以总容量为960G，3个数据节点为例，那么每个节点的磁盘容量就是320G。

   登录**DWS控制台**，选择“监控>节点监控>概览”显示每个节点的磁盘以及其他资源占用情况。

   说明

   - 节点管理看到的磁盘空间是DWS集群内所有的磁盘即系统盘、数据盘加到一起的容量，而在存算一体集群概览里看到的磁盘空间只是集群内能做表数据存储的可用空间，另外DWS集群中表是有备份的副本数的，表的备份数据也是需要占用磁盘存储的。存算分离集群概览中的磁盘空间，为缓存大小，是实际申请的磁盘空间。
   - 如果已确定由于磁盘空间不足导致集群状态只读，告警磁盘不可用的异常场景时，可参考**扩容集群**章节进行节点扩容处理。

6.4 DWS 添加云监控服务的告警规则时会话数阈值如何设置?

连接数据库后，执行以下SQL语句可以查看当前全局最大并发会话数。

```
show max_active_statements;
```

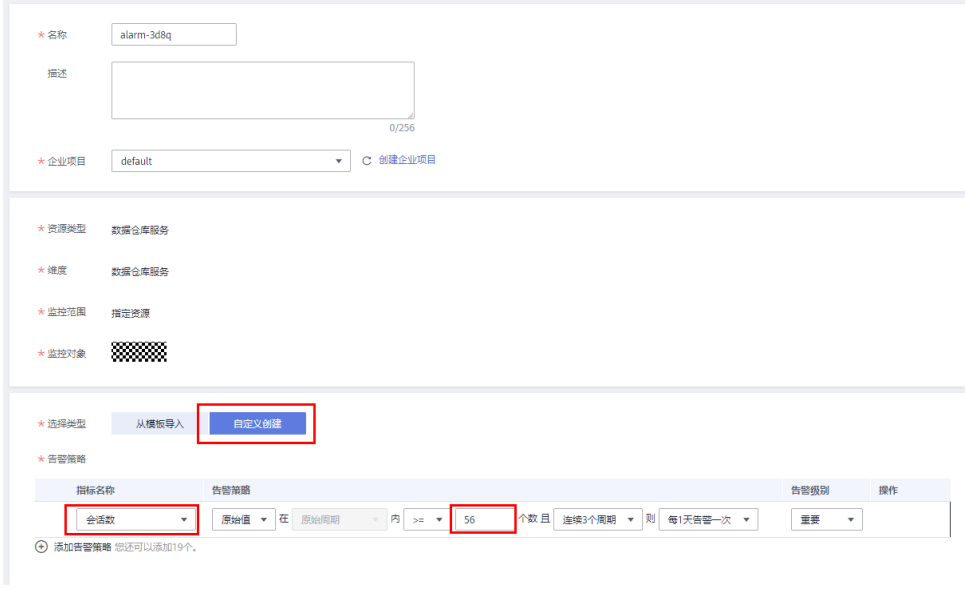
进入到云监控界面，根据查出的全局最大并发会话数，取70%-80%为阈值即可。例如，查询到max_active_statements为80，则阈值设置为80*70%=56。

设置方法：

1. 在**DWS控制台**，选择“集群 > 集群列表”。

2. 单击集群所在行右侧的“查看监控指标”，进入云监控服务界面。
3. 单击左上角 ，单击集群名称所在行右侧“创建告警规则”。

4. “选择类型”选择“自定义创建”，指标名称选择“会话数”，告警策略填写“56”，告警级别为“重要”，单击“立即创建”。



6.5 如何判断 DWS 集群是 x86 还是 ARM 架构?

操作步骤

步骤1 登录**DWS控制台**。

步骤2 单击“集群 > 集群列表”。默认显示用户所有的集群列表。

步骤3 在集群列表中，单击指定集群名称进入“集群详情”页面，在“基本信息”模块查看指定集群的节点规格。

步骤4 根据节点规格在表格中查找对应的集群架构。规格说明如下所示：

表 6-1 规格说明

节点规格	vCPU核数	内存大小（GB）	架构	规格类型
dws2.olap.4xlarge.m6	16	128	X86	ECS/存算分离EVS
dws2.olap.8xlarge.m6	32	256	X86	ECS/存算分离EVS
dws2.olap.16xlarge.m6	64	512	X86	ECS/存算分离EVS
dws2.olap.4xlarge.kc1	16	64	ARM	ECS/存算分离EVS
dws2.olap.4xlarge.km1	16	128	ARM	ECS/存算分离EVS
dws2.olap.6xlarge.km1	24	192	ARM	ECS/存算分离EVS
dws2.olap.8xlarge.km1	32	256	ARM	ECS/存算分离EVS
dws2.olap.12xlarge.km1	48	384	ARM	ECS/存算分离EVS
dws2.m6.4xlarge.8	16	128	X86	ECS/EVS
dws2.m6.8xlarge.8	32	256	X86	ECS/EVS
dws2.m6.16xlarge.8	64	512	X86	ECS/EVS
dws2.km1.4xlarge.8	16	128	ARM	ECS/EVS
dws2.km1.6xlarge.8	24	192	ARM	ECS/EVS
dws2.km1.8xlarge.8	32	256	ARM	ECS/EVS
dws2.km1.12xlarge.8	48	384	ARM	ECS/EVS
dws2.km1.xlarge	4	32	ARM	ECS/EVS
dws2.kc1.4xlarge	16	64	ARM	ECS/EVS

节点规格	vCPU核数	内存大小（GB）	架构	规格类型
dws2.olap.4xlarge.i3	16	128	X86	ECS/本地直通
dws2.olap.8xlarge.i3	32	256	X86	ECS/本地直通
dws2.olap.16xlarge.i3	64	512	X86	ECS/本地直通
dws2.olap.4xlarge.ki1	16	64	ARM	ECS/本地直通
dws2.olap.8xlarge.ki1	32	128	ARM	ECS/本地直通
dws2.olap.16xlarge.ki1	64	228	ARM	ECS/本地直通
dws2.physical.ki1ne.4xlarge	128	512	ARM	BMS
dws2.physical.ki1ne.4xlarge.a	128	512	ARM	BMS
dws2.physical.c6sd.6xlarge	104	768	X86	BMS
dws2.physical.c6sd.6xlarge.7	104	768	X86	BMS
dws2.physical.c6sd.6xlarge.7.cbg	104	768	X86	BMS
dws2.physical.c6sd.6xlarge.a.7	104	768	X86	BMS
dws2.physical.io6.3xlarge.4a	104	384	X86	BMS
dws2.physical.c6d.6xlarge.3a.cbg	88	768	X86	BMS
dws2.physical.c6sd.6xlarge.a.7.cbg	104	768	X86	BMS
dws2.physical.c6sd.6xlarge.1.cbg	104	768	X86	BMS

----结束

6.6 DWS 扩容检查不通过怎么办?

问题描述

扩容或者添加空闲节点时单击“确认”按钮后弹窗警告，无法进入下一步操作。

原因分析

在提交扩容前会对必须检查项进行检查，包括资源配额、IAM权限等，如果不通过会禁止提交扩容操作，从而避免扩容失败。

解决办法

- 配额检查不通过，根据检查项检查对应资源配额是否充足，如果可使用的节点配额不足，用户可以单击“申请扩大配额”，以提工单的形式申请更多节点配额。
- IAM权限：只有IAM子账号时会出现此问题，可使用主账号扩容或主账号授予子账号VPC、EVC/BMS等相关操作权限即可。
- 若检查无误，弹窗依旧存在，可联系技术工程师协助。

6.7 DWS 增加 CN 和扩容集群分别在什么场景下使用?

CN 并发介绍

CN全称为：协调节点（Coordinator Node），是和用户关系最密切也是DWS内部非常重要的一个组件。它负责提供外部应用接口、优化全局执行计划、向Datanode分发执行计划，以及汇总、处理执行结果。CN是外部应用的接口，CN的并发能力直接决定了业务的并发度。

单CN的并发能力受如下几个参数控制：

- **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。默认值与集群规格有关，具体参见**管理数据库连接**章节。
- **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。默认值60，最多允许60个作业同时运行，其余作业将会排队。

选择增加 CN 还是扩容集群

- 连接数不足：初次创建集群时，集群默认的CN节点数是3，能基本满足客户的连接需求。当集群属于高并发请求，各CN节点的连接数很大，或CN节点的CPU明显高于DN节点的CPU时，建议增加CN节点数量，具体参见**管理CN节点**章节。
- 存储容量和性能不足：随着您的业务规模扩张，对数据存储容量和性能有更高的要求时，或者集群整体CPU不足时，建议通过扩容集群进行集群节点的扩容。详情请参见**集群扩容**章节。

随着业务规模扩大，集群扩容到一定节点规模后，也必然需要增加相应的CN节点，满足DWS的分布式要求。简言之，增加CN，不一定需要扩容集群，但是扩容集群后，会伴随着增加CN节点的需求。

6.8 DWS 经典变更规格与弹性变更规格、扩容、缩容分别在什么场景下使用?

经典变更规格相对来说比较重量级，对业务的影响也比较大，相当于实现将老集群迁移到新集群的功能，并同时实现规格的升降、节点数量的增减。建议用户优先使用扩容、缩容、弹性变更规格等轻量级操作。功能特性具体适用场景如下表所示：

表 6-2 特性功能区别

功能特性	适用场景	备注
扩容	随着您的业务规模扩张，对数据存储容量和性能有更高的要求时，或者集群整体CPU不足时，建议通过扩容集群进行集群节点的扩容	-
缩容	集群容量大量闲置的业务低谷期，可通过缩容操作来减少节点数量以实现减少成本的目的。	-
弹性变更规格	仅对现有集群做规格的调整，该功能不会涉及节点数量的改变，只是对节点的CPU、内存等做配置升级或降级，用以实现根据业务量调整集群性能，以实现业务诉求。	弹性变更规格目前仅支持ECS+EVS形态的存算一体集群。
经典变更规格	若有以下需求可考虑使用调经典变更规格功能，例如： ● BMS(裸金属服务器)集群或不支持弹性变更规格功能的集群，该场景只能进行经典变更规格操作实现规格变更。 ● 用户想要改变集群拓扑结构，因为扩容/缩容操作都是按环增加/减少节点。 ● 集群版本低需变更为新集群，同时不想进行业务数据迁移。	经典变更规格目前仅支持存算一体集群。

6.9 DWS 在 CPU 核数、内存相同的情况下，小规格多节点与大规格三节点集群如何选择?

- 小规格多节点：

  如果数据量不大、集群节点数量需要伸缩，但不能忍受太高的成本，可以选择小规格多节点集权。

  例如，规格为8核32G的小规格集群（dwsx2.h.2xlarge.4.c6），可以提供较强的计算能力，由于集群节点数量较多，可以处理集群的高并发请求。这种情况下，只需要保证节点间网速通畅，避免集群性能受限。

- 大规格三节点：

  如果需要处理大量数据、需要高性能计算，但可以承受较高的成本，可以选择大规格三节点集群。

  例如，规格为32核256G的大规格集群（dws2.m6.8xlarge.8），拥有更快的CPU处理能力和更大的内存容量，可以更快速地处理数据。但是三节点的集群节点数量有限，高并发场景下性能较差。

6.10 DWS SSD 云盘和 SSD 本地盘的区别?

SSD云盘支持后期进行扩容，推荐您使用SSD云盘。两者的区别如下：

- **SSD云盘**：
  - 使用SSD类型的EVS作为数据存储介质，存储容量更加灵活，且可以随着数据的增长，进行磁盘扩容操作。
  - 由于SSD云盘不和ECS规格进行强绑定，因此可以根据实际需求进行规格调整。
- **SSD本地盘**：
  - 使用ECS规格自带的本地磁盘作为数据存储介质，容量固定，性能更高，但是不能对磁盘进行扩容。
  - 在容量不足的情况下，只能够新增节点。SSD本地盘规格不允许进行规格调整。

6.11 DWS 热数据存储和冷数据存储的区别?

热数据存储和冷数据存储最大的区别在于存储介质不同：

- 热数据存储是直接将频繁被查询或更新，对访问的响应时间要求很高的热数据存储在**DN数据盘**中。
- 冷数据存储将不更新，偶尔被查询，对访问的响应时间要求不高的冷数据存储在**OBS**中。

因为其存储介质的不同，决定了两者的成本、性能、以及适用场景，如表6-3所示：

表 6-3 冷热数据存储区别

存储名称	读取、写入速度	成本	容量	适用场景
热数据存储	快	高	固定，有限制	适用于那些数据量有限，需要频繁读取、更新的场景。
冷数据存储	慢	低	大、无限制	适用于一些归档类业务，利用其低成本，大容量的特点，在完整保存数据的同时，还能节省成本，不占用本地空间。

6.12 DWS 缩容按钮置灰如何处理?

问题描述

用户进行缩容操作时，页面“缩容”按钮置灰，无法进入下一步缩容操作。

原因分析

在进行缩容操作前系统会提前检查集群是否满足缩容条件，缩容条件不满足，“缩容”按钮置灰。

解决办法

需排查集群配置信息，检查缩容是否满足以下条件：

- 集群按照环的方式配置，比如4个或5个主机组成一个环，这些主机上的DN主节点、备节点和从节点都部署在这些节点里，这些节点组成一个集群环 ，缩容的最小单元是一个集群环，集群至少有2个环才能支持缩容，缩容按照集群环从后往前缩容节点。
- 缩容节点不能包含GTM组件，CM Server组件，CN组件。
- 集群状态为Normal，无其他任务信息。
- 集群租户账号不能处于只读，冻结，受限状态。
- 集群非逻辑集群模式。
- 包周期集群不能处于已过期进入宽限期。
- 集群不能有空闲节点。

7 账户与权限

7.1 DWS 如何实现业务隔离

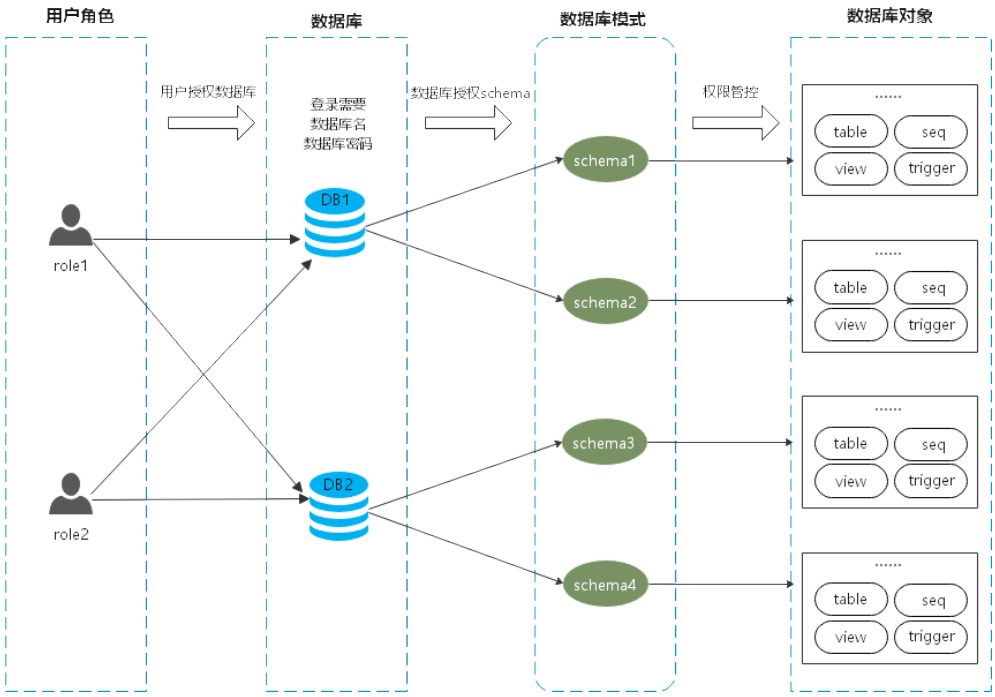
业务隔离

DWS中可以使用Database和Schema实现业务的隔离，区别在于：

- Database之间无法直接互访，通过连接隔离实现彻底的权限隔离。各个Database之间共享资源极少，可实现连接隔离、权限隔离等。
- 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，从而赋予业务更多的灵活性。

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。

图 7-1 权限控制



DATABASE

数据库Database是数据库对象的物理集合，不同Database之间资源完全隔离（除部分共享对象之外）。即Database是对业务的物理隔离，不同Database的之间的对象不能相互访问。比如在Database A中无法访问Database B中的对象。因此登录集群的时候必须显示指定要连接的Database。

SCHEMA

数据库里面通过Schema把数据库对象进行逻辑划分，在Database中，通过Schema实现对数据库对象的逻辑隔离。

通过权限管理实现在同一个session下对不同Schema下对象的访问和操作权限。Schema下则是各种应用程序会接触到的对象，比如表，索引，数据类型，函数，操作符等。

同一个Schema下，不能存在同名的数据库对象；但是不同Schema下的对象名可以重复。

```
gaussdb=> CREATE SCHEMA myschema;
CREATE SCHEMA
gaussdb=> CREATE SCHEMA myschema_1;
CREATE SCHEMA

gaussdb=> CREATE TABLE myschema.t1(a int, b int) DISTRIBUTE BY HASH(b);
CREATE TABLE
gaussdb=> CREATE TABLE myschema.t1(a int, b int) DISTRIBUTE BY HASH(b);
ERROR:  relation "t1" already exists
gaussdb=> CREATE TABLE myschema_1.t1(a int, b int) DISTRIBUTE BY HASH(b);
CREATE TABLE
```

Schema实现了对业务的逻辑划分，反过来这些业务对象也对Schema形成一种依赖关系，因此当Schema下存在对象时，删除Schema的时候会报错，并提示具体的依赖信息。

```
gaussdb=> DROP SCHEMA myschema_1;
ERROR: cannot drop schema myschema_1 because other objects depend on it
Detail: table myschema_1.t1 depends on schema myschema_1
Hint: Use DROP ... CASCADE to drop the dependent objects too.
```

当删除Schema的时候加上CASCADE选项，把Schema以及依赖此Schema的选项连带删除。

```
gaussdb=> DROP SCHEMA myschema_1 CASCADE;
NOTICE:  drop cascades to table myschema_1.t1
gaussdb=> DROP SCHEMA
```

USER/ROLE

用户或角色是数据库服务器(集群)全局范围内的权限控制系统，是集群业务的所有者和执行者，用于各种集群范围内所有的对象权限管理。因此角色不特定于某个单独的数据库，但角色登录集群的时候必须要显式指定登录的用户名，以保证当前连接执行的操作者的透明性。同时数据库也会通过权限管理限定用户的访问和操作权限。

用户是权限的最终体现者，所有的权限管理最终都体现在用户对数据库对象的操作权限是否被允许。

权限管理

DWS中的权限管理分为三种场景：

- 系统权限

  系统权限又称为用户属性，包括SYSADMIN、CREATEDB、CREATEROLE、AUDITADMIN和LOGIN。

  系统权限一般通过CREATE/ALTER ROLE语法来指定。其中，SYSADMIN权限可以通过GRANT/REVOKE ALL PRIVILEGE授予或撤销。但系统权限无法通过ROLE和USER的权限被继承，也无法授予PUBLIC。

- 用户权限

  将一个角色或用户的权限授予一个或多个其他角色或用户。在这种情况下，每个角色或用户都可视为拥有一个或多个数据库权限的集合。

  当声明了WITH ADMIN OPTION，被授权的用户可以将该权限再次授予其他角色或用户，以及撤销所有由该角色或用户继承到的权限。当授权的角色或用户发生变更或被撤销时，所有继承该角色或用户权限的用户拥有的权限都会随之发生变更。

  数据库系统管理员可以给任何角色或用户授予/撤销任何权限。拥有CREATEROLE权限的角色可以赋予或者撤销任何非系统管理员角色的权限。

- 数据对象权限

  将数据库对象（表和视图、指定字段、数据库、函数、模式等）的相关权限授予特定角色或用户。GRANT命令将数据库对象的特定权限授予一个或多个角色。这些权限会追加到已有的权限上。

SCHEMA 隔离应用示例

示例一：

Schema的owner默认拥有该Schema下对象的所有权限，包括删除权限；Database的owner默认拥有该Database下对象的所有权限，包括删除权限。因此建议对Database和Schema的创建要做比较严格的控制，一般建议使用管理员创建Database和Schema，然后把相关的权限控制赋予业务用户。

步骤1 dbadmin在数据库testdb下把创建Schema的权限赋予普通用户user_1。

```
testdb=> GRANT CREATE ON DATABASE testdb to user_1;
GRANT
```

步骤2 切换到普通用户user_1。

```
testdb=> SET SESSION AUTHORIZATION user_1 PASSWORD '********';
SET
```

用户user_1在数据库testdb下创建名为myschema_2的Schema。

```
testdb=> CREATE SCHEMA myschema_2;
CREATE SCHEMA
```

步骤3 切换到管理员dbadmin。

```
testdb=> RESET SESSION AUTHORIZATION;
RESET
```

管理员dbadmin在模式myschema_2下创建表t1。

```
testdb=> CREATE TABLE myschema_2.t1(a int, b int) DISTRIBUTE BY HASH(b);
CREATE TABLE
```

步骤4 切换到普通用户user_1。

```
testdb=> SET SESSION AUTHORIZATION user_1 PASSWORD '********';
SET
```

普通用户user_1删除管理员dbadmin在模式myschema_2下创建的表t1。

```
testdb=> drop table myschema_2.t1;
DROP TABLE
```

----结束

示例二：

因为Schema的逻辑隔离的功能，访问数据库对象实际上要通过Schema和具体对象的两层校验。

步骤1 把表myschema.t1的权限赋予用户user_1。

```
gaussdb=> GRANT SELECT ON TABLE myschema.t1 TO user_1;
GRANT
```

步骤2 切换到用户user_1。

```
SET SESSION AUTHORIZATION user_1 PASSWORD '********';
SET
```

查询表myschema.t1。

```
gaussdb=> SELECT * FROM myschema.t1;
ERROR:  permission denied for schema myschema
LINE 1: SELECT * FROM myschema.t1;
```

步骤3 切换到管理员dbadmin。

```
gaussdb=> RESET SESSION AUTHORIZATION;
RESET
```

把myschema.t1的权限赋予用户user_1。

```
gaussdb=> GRANT USAGE ON SCHEMA myschema TO user_1;
GRANT
```

步骤4 切换到普通用户user_1。

```
gaussdb=> SET SESSION AUTHORIZATION user_1 PASSWORD '********';
SET
```

查询表myschema.t1。


```
gaussdb=> SELECT * FROM myschema.t1;
 a | b
---+---
(0 rows)
```

----结束

7.2 DWS 数据库账户密码到期了，如何修改?

数据库账户密码到期修改方式如下：

- 数据库管理员dbadmin的密码，可登录**DWS控制台**，选择集群所在行右边的“更多 > 重置密码”进行修改。

  图 7-2 dbadmin 重置密码

- 数据库账户（普通用户和管理员dbadmin）的密码有效期，也可以在连接数据库后使用ALTER USER进行修改：

  ```
  ALTER USER username PASSWORD EXPIRATION 90;
  ```

7.3 如何给 DWS 指定用户赋予某张表的权限?

给指定用户赋予某张表的权限主要通过以下语法实现，本章主要介绍常见的几种场景，包括只读（SELECT）、插入（INSERT）、改写（UPDATE）和拥有所有权限。

语法格式

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER | ANALYZE |
ANALYSE } [, ...]
    | ALL [ PRIVILEGES ] }
  ON { [ TABLE ] table_name [, ...]
    | ALL TABLES IN SCHEMA schema_name [, ...] }
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

场景介绍

假设当前有用户u1~u5，在系统中有对应的同名Schema u1~u5，各用户的权限管控如下：

- u2作为只读用户，需要表u1.t1的SELECT权限。
- u3作为插入用户，需要表u1.t1的INSERT权限。
- u4作为改写用户，需要表u1.t1的UPDATE权限。
- u5作为拥有所有权限的用户，需要表u1.t1的所有权限。

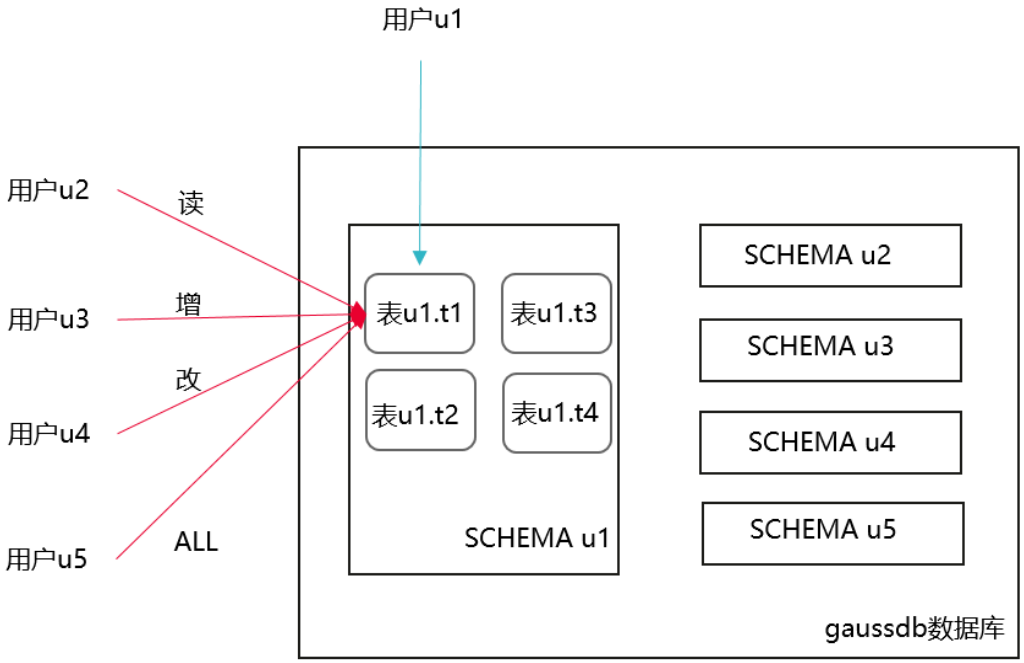


表 7-1 表 u1.t1 的表权限分类

用户名	用户类型	Grant授权语句	查询	插入	修改	删除
u1	所有者	-	√	√	√	√
u2	只读用户	GRANT SELECT ON u1.t1 TO u2;	√	x	x	x
u3	插入用户	GRANT INSERT ON u1.t1 TO u3;	x	√	x	x
u4	改写用户	GRANT SELECT,UPDATE ON u1.t1 TO u4; **须知** 授予UPDATE权限必须同时授予SELECT权限，否则会出现信息泄露。	√	x	√	x

用户名	用户类型	Grant授权语句	查询	插入	修改	删除
u5	拥有所有权限的用户	GRANT ALL PRIVILEGES ON u1.t1 TO u5;	√	√	√	√

操作步骤

以下将演示不同权限的授权方法和验证过程。

步骤1 打开窗口1（即dbadmin连接会话窗口，后续不再提示），使用系统管理员dbadmin连接DWS数据库，创建用户u1~u5（系统默认会创建u1~u5的同名SCHEMA）。

```
CREATE USER u1 PASSWORD '{password}';
CREATE USER u2 PASSWORD '{password}';
CREATE USER u3 PASSWORD '{password}';
CREATE USER u4 PASSWORD '{password}';
CREATE USER u5 PASSWORD '{password}';
```

步骤2 在SCHEMA u1下创建表u1.t1。

```
CREATE TABLE u1.t1 (c1 int, c2 int);
```

步骤3 为表中插入两条数据。

```
INSERT INTO u1.t1 VALUES (1,2);
INSERT INTO u1.t1 VALUES (1,2);
```

步骤4 DWS中引入了SCHEMA层概念，如果有SCHEMA，需要先给用户赋予SCHEMA的使用权限。

```
GRANT USAGE ON SCHEMA u1 TO u2,u3,u4,u5;
```

步骤5 给只读用户u2赋予表u1.t1的查询权限。

```
GRANT SELECT ON u1.t1 TO u2;
```

步骤6 打开窗口2（即用户u2连接会话窗口，后续不再提示），使用用户u2连接DWS数据库，验证u2可以查询u1.t1表，但是不能写入和修改，此时u2为只读用户。

```
SELECT * FROM u1.t1;
INSERT INTO u1.t1 VALUES (1,20);
UPDATE u1.t1 SET c2 = 3 WHERE c1 =1;
```

```
gaussdb=> SELECT * FROM u1.t1;
 c1 | c2
----+----
  1 |  2
  1 |  2
(2 rows)

gaussdb=> INSERT INTO u1.t1 VALUES (1,20);
ERROR:  permission denied for relation t1
gaussdb=> UPDATE u1.t1 SET c2 = 3 WHERE c1 =1;
ERROR:  permission denied for relation t1
```

步骤7 切换回窗口1，分别给u3、u4、u5赋予对应的权限。

```
GRANT INSERT ON u1.t1 TO u3;   --插入用户u3，可以插入数据
GRANT SELECT,UPDATE ON u1.t1 TO u4;  --改写用户u4，可以修改表
GRANT ALL PRIVILEGES ON u1.t1 TO u5;  --拥有所有权限的用户u5，可以对表进行查询、插入、改写和删除
```

步骤8 打开窗口3，使用用户u3连接DWS数据库，验证u3可以插入u1.t1，但是不能查询和修改，此时u3为插入用户。

```
SELECT * FROM u1.t1;
INSERT INTO u1.t1 VALUES (1,20);
UPDATE u1.t1 SET c2 = 3 WHERE c1 =1;
```

```
gaussdb=> SELECT * FROM u1.t1;
ERROR:  permission denied for relation t1
gaussdb=> INSERT INTO u1.t1 VALUES (1,20);
INSERT 0 1
gaussdb=> UPDATE u1.t1 SET c2 = 3 WHERE c1 =1;
ERROR:  permission denied for relation t1
```

步骤9 打开窗口4，使用用户u4连接DWS数据库，验证u4可以修改u1.t1，同时还可以查询，但是不能插入，此时u4为改写用户。

```
SELECT * FROM u1.t1;
INSERT INTO u1.t1 VALUES (1,20);
UPDATE u1.t1 SET c2 = 3 WHERE c1 =1;
```

```
gaussdb=> SELECT * FROM u1.t1;
 c1 | c2
----+----
  1 |  2
  1 |  2
  1 | 20
(3 rows)

gaussdb=> INSERT INTO u1.t1 VALUES (1,20);
ERROR:  permission denied for relation t1
gaussdb=> UPDATE u1.t1 SET c2 = 3 WHERE c1 =1;
UPDATE 3
```

步骤10 打开窗口5，使用用户u5连接DWS数据库，验证u5可以查询、插入、修改和删除u1.t1，此时u5为拥有所有权限的用户。

```
SELECT * FROM u1.t1;
INSERT INTO u1.t1 VALUES (1,20);
UPDATE u1.t1 SET c2 = 3 WHERE c1 =1;
DELETE FROM u1.t1;
```

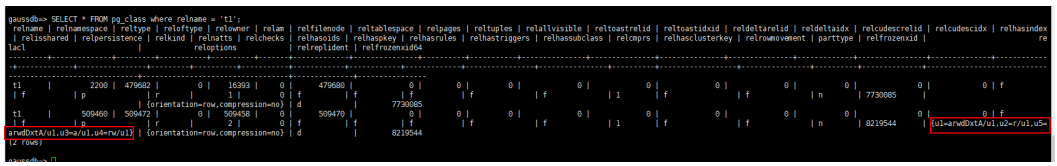
```
gaussdb=> SELECT * FROM u1.t1;
 c1 | c2
----+----
  1 |  3
  1 |  3
  1 |  3
(3 rows)

gaussdb=> INSERT INTO u1.t1 VALUES (1,20);
INSERT 0 1
gaussdb=> UPDATE u1.t1 SET c2 = 3 WHERE c1 =1;
UPDATE 4
gaussdb=> DELETE FROM u1.t1;
DELETE 4
```

步骤11 最后切换回窗口1，通过函数has_table_privilege分别查询每个用户的权限。

```
SELECT * FROM pg_class WHERE relname = 't1';
```

返回结果，查看relacl字段，该字段回显结果如下。"*rolename*=xxxx/yyyy" --表示rolename对该表有xxxx权限，且权限来自yyyy；

例如下图，与以上验证结果完全一致。



- u1=arwdDxtA/u1，表示u1为owner，拥有所有权限。
- u2=r/u1，表示u2拥有读权限。
- u3=a/u1，表示u3拥有插入权限。
- u4=rw/u1，表示u4拥有读和修改权限。
- u5=arwdDxtA/u1，表示u5拥有所有权限。

----结束

7.4 如何给 DWS 指定用户赋予某个 SCHEMA 的权限?

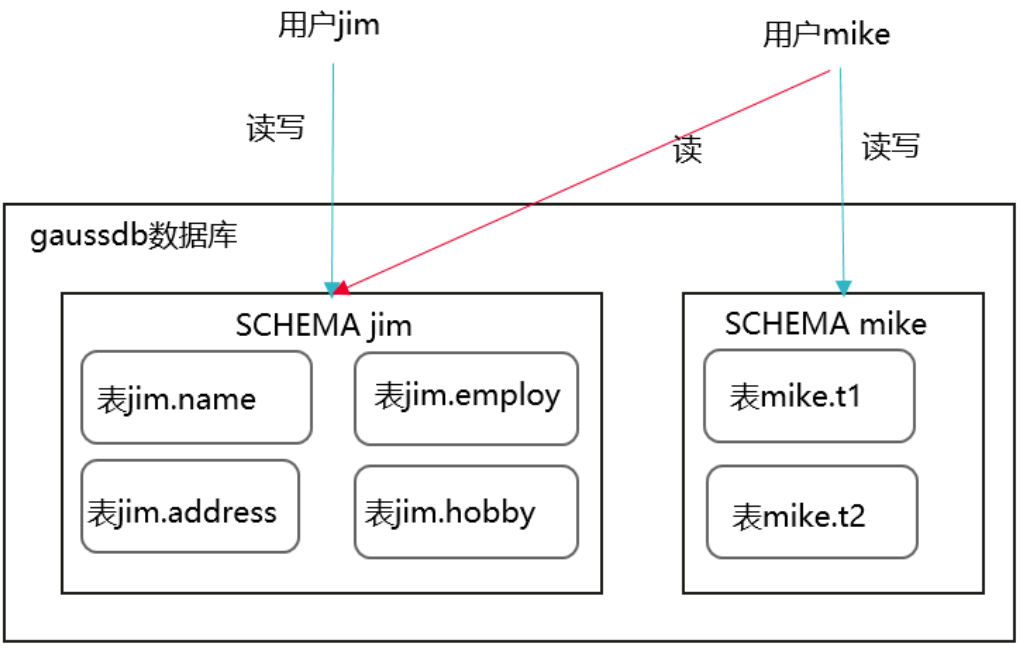
给某个用户授权某个SCHEMA的权限，包括三个场景（本章节针对SCHEMA层级权限，仅演示授权查询权限，如果需要其他权限，可以参考**如何给指定用户赋予某张表的权限?** ）：

- SCHEMA下当前某张表权限。
- SCHEMA下当前所有表的权限。
- SCHEMA下未来创建的表的权限。

> **⚠ 注意**
>
> 不支持将外表的VACUUM，DROP，ALTER的权限赋予用户。

如**图7-3**所示，假设有两个用户jim和mike，对应的同名SCHEMA是jim和mike，用户mike需要访问SCHEMA jim的表（包括当前的某张表、所有表、未来创建的表）。

图 7-3 用户 mike 访问 SCHEMA jim 下的表



步骤1 打开窗口1（即dbadmin连接会话窗口，后续不再提示），使用系统管理员dbadmin连接DWS数据库，创建用户jim和mike（系统默认会创建jim和mike的同名SCHEMA）。

```
CREATE USER jim PASSWORD '{password}';
CREATE USER mike PASSWORD '{password}';
```

步骤2 在SCHEMA jim下创建表jim.name和jim.address。

```
CREATE TABLE jim.name (c1 int, c2 int);
CREATE TABLE jim.address (c1 int, c2 int);
```

步骤3 给用户mike赋予SCHEMA jim的访问权限。

```
GRANT USAGE ON SCHEMA jim TO mike;
```

步骤4 （**某张表权限**）给用户mike赋予SCHEMA jim下某张表jim.name的查询权限。

```
GRANT SELECT ON jim.name TO mike;
```

步骤5 打开窗口2（即用户mike连接会话窗口，后续不再提示），使用用户mike连接DWS数据库，验证mike可以查询jim.name表，但是不能查询jim.address表。

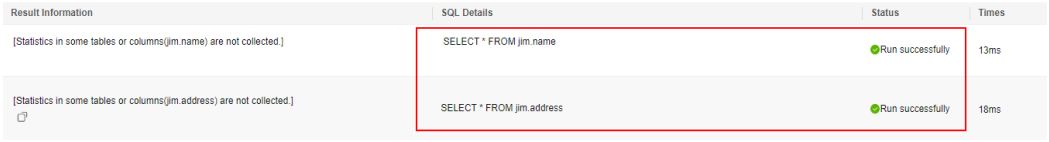
```
SELECT * FROM jim.name;
SELECT * FROM jim.address;
```

步骤6 （**所有表权限**）切换回窗口1，使用dbadmin给用户mike赋予SCHEMA jim下所有表的查询权限。

```
GRANT SELECT ON ALL TABLES IN SCHEMA jim TO mike;
```

步骤7 切换回窗口2，再次验证mike可以查询所有表。

```
SELECT * FROM jim.name;
SELECT * FROM jim.address;
```



步骤8 切换回窗口1，创建一张新的表jim.employ。

```
CREATE TABLE jim.employ (c1 int, c2 int);
```

步骤9 切换回窗口2，验证发现用户mike没有jim.employ的查询权限，说明mike虽然有SCHEMA jim下所有表的访问权限，但是对于赋权后新创建的表还是没有访问权限（即mike对SCHEMA jim未来的表权限是没有的）。

```
SELECT * FROM jim.employ;
```

步骤10 （**未来表权限**）切换回窗口1，给用户mike赋予SCHEMA jim未来创建的表的访问权限，并创建一张新的表jim.hobby。

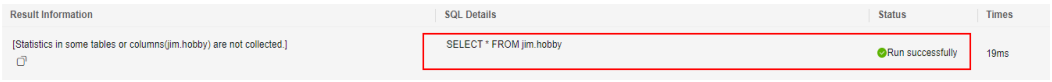
说明

ALTER DEFAULT PRIVILEGES 用来授予将来要创建的对象的权限。

```
ALTER DEFAULT PRIVILEGES FOR ROLE jim IN SCHEMA jim GRANT SELECT ON TABLES TO mike;
CREATE TABLE jim.hobby (c1 int, c2 int);
```

步骤11 切换回窗口2，验证发现用户mike可以访问jim.hobby。但是对于之前的jim.employ还是没有访问权限（以上ALTER DEFAULT PRIVILEGES语句的授权范围只是未来创建的表，对于已经创建的表，还是要使用GRANT单张表重新授权），此时需要参考**步骤4**重新授权jim.employ即可。

```
SELECT * FROM jim.hobby;
```



----结束

7.5 如何创建 DWS 数据库只读用户?

场景介绍

在业务开发场景中，数据库管理员通过SCHEMA来划分不同的业务，例如在金融行业中，负债业务属于SCHEMA s1，资产业务属于SCHEMA s2。

当前需要在数据库中创建一个只读用户user1，允许这个用户访问负债业务SCHEMA s1下所有的表（包括未来创建的新表），供日常读取，但是不允许进行数据插入、修改或删除。

实现原理

DWS有基于角色的用户管理，需要先创建一个的只读角色role1，再将对应的角色授权到实际的用户user1即可。具体参见**基于角色的权限管理**。

操作步骤

步骤1 使用系统管理员dbadmin连接DWS数据库。

步骤2 执行以下SQL语句创建角色role1。

```
CREATE ROLE role1 PASSWORD disable;
```

步骤3 执行以下SQL语句，为角色role1进行授权。

```
GRANT usage ON SCHEMA s1 TO role1;    --赋予SCHEMA s1的访问权限;
GRANT select ON ALL TABLES IN SCHEMA s1 TO role1;    --赋予SCHEMA s1下所有表的查询权限;
ALTER DEFAULT PRIVILEGES FOR USER tom IN SCHEMA s1 GRANT select ON TABLES TO role1;    --赋予SCHEMA s1未来创建的表的权限，其中tom为SCHEMA s1的owner
```

步骤4 执行以下SQL语句，将角色role1授权到实际用户user1。

```
GRANT role1 TO user1;
```

步骤5 如果访问的SCHEMA s1中包含有外表，还需要对只读用户user1进行使用外表的授权，执行以下命令。

```
ALTER USER user1 USEFT;
```

否则以只读用户查询外表时会报以下错误：“ERROR: permission denied to select from foreign table in security mode”。

步骤6 使用只读用户user1进行SCHEMA s1下所有表数据的日常读取。

----结束

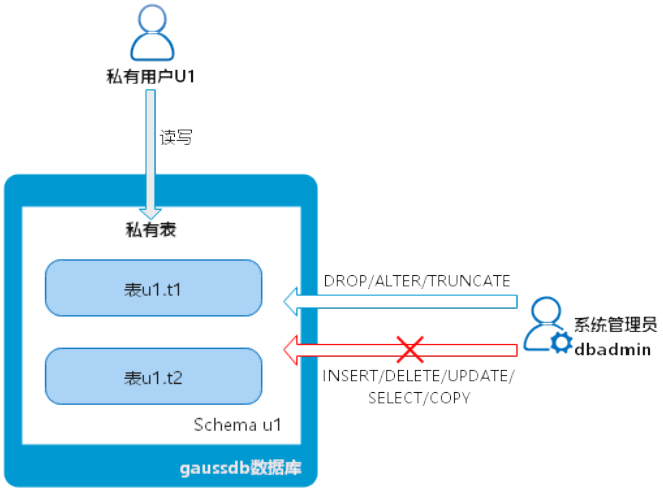
7.6 如何创建 DWS 数据库私有用户和私有表?

场景介绍

在业务场景中，普通用户创建的表，系统管理员dbadmin默认是有访问权限，并未完全私有。而在三权分立开启下的业务场景中，管理员dbadmin对普通用户的表没有访问权限，同时也没有控制权限（DROP、ALTER、TRUNCATE）。

若当前业务中需要创建一个私有用户和私有表（即私有用户创建的表），私有表只有私有用户本身可以访问，系统管理员dbadmin和其他普通用户均无权访问（进行INSERT、DELETE、UPDATE、SELECT、COPY操作），但同时需要满足：未经私有用户授权的情况下，可允许系统管理员dbadmin执行DROP/ALTER/TRUNCATE操作。那么可以通过创建INDEPENDENT属性的用户（私有用户）实现。

图 7-4 私有用户



实现原理

通过创建INDEPENDENT属性的用户来实现。

INDEPENDENT | NOINDEPENDENT：定义私有、独立的角色。具有INDEPENDENT属性的角色，管理员对其进行的控制、访问的权限被分离，具体规则如下：

- 未经INDEPENDENT角色授权，管理员无权对其表对象进行增、删、查、改、拷贝、授权操作。
- 未经INDEPENDENT角色授权，管理员无权修改INDEPENDENT角色的继承关系。
- 管理员无权修改INDEPENDENT角色的表对象的属主。
- 管理员无权修改INDEPENDENT角色的数据库口令，INDEPENDENT角色需管理好自身口令，口令丢失无法重置。
- 管理员属性用户不允许定义修改为INDEPENDENT属性。

操作步骤

步骤1 使用系统管理员dbadmin连接DWS数据库。

步骤2 执行以下SQL语句创建私有用户u1。

```
CREATE USER u1 WITH INDEPENDENT IDENTIFIED BY 'password';
```

步骤3 切换到u1用户，创建测试表test，并插入数据。

```
CREATE TABLE test (id INT, name VARCHAR(20));
INSERT INTO test VALUES (1, 'joe');
INSERT INTO test VALUES (2, 'jim');
```

步骤4 切换到dbadmin用户，并执行以下SQL语句验证dbadmin用户是否可以访问私有用户u1创建的私有表test。

```
SELECT * FROM u1.test;
```

查询结果提示dbadmin无权访问，即私有用户和私有表创建成功。

```
gaussdb=> SELECT * FROM u1.test;
ERROR:  SELECT permission denied to user "dbadmin" for relation "u1.test"
```

步骤5 dbadmin用户执行DROP语句，可成功删除test表。

```
DROP TABLE u1.test;
```

```
gaussdb=> drop table u1.test;
DROP TABLE
```

----结束

7.7 DWS 如何 REVOKE 某用户的 connect on database 权限?

业务场景

某业务中需要撤销指定用户u1连接某数据库的权限，在执行**REVOKE CONNECT ON DATABASE *gaussdb* FROM u1;**命令成功后，使用u1还能继续连接数据库，撤销权限没有生效。

原因分析

若直接使用REVOKE CONNECT ON DATABASE gaussdb from u1命令撤销u1用户的权限不会生效，因为数据库的CONNECT权限授予了PUBLIC，需指定关键字PUBLIC实现。

- 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。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。
- DWS会将某些类型的对象上的权限授予PUBLIC。默认情况下，对表、表字段、序列、外部数据源、外部服务器、模式或表空间对象的权限不会授予PUBLIC。以下这些对象的权限会授予PUBLIC：
  - 数据库的CONNECT权限。
  - CREATE TEMP TABLE权限。
  - 函数的EXECUTE特权。
  - 语言和数据类型（包括域）的USAGE特权。
- 对象拥有者可以撤销默认授予PUBLIC的权限并专门授予权限给其他用户。

操作示例

撤销用户u1访问数据库gaussdb的权限：

步骤1 连接DWS集群的数据库gaussdb。

```
gsql -d gaussdb -p 8000 -h 192.168.x.xx -U dbadmin -W password -r
gaussdb=>
```

步骤2 创建用户u1。

```
gaussdb=> CREATE USER u1 IDENTIFIED BY 'xxxxxxxx';
```

步骤3 确认用户u1可正常访问数据库gaussdb。

```
gsql -d gaussdb -p 8000 -h 192.168.x.xx -U u1 -W password -r
gaussdb=>
```

步骤4 使用管理员用户dbadmin连接数据库gaussdb，执行REVOKE命令撤销public的connect on database权限。

```
gsql -d gaussdb -h 192.168.x.xx -U dbadmin -p 8000 -r
gaussdb=> REVOKE CONNECT ON DATABASE gaussdb FROM public;
REVOKE
```

步骤5 验证结果，使用u1连接数据库，显示如下信息表示用户u1的connect on database权限已成功撤销。

```
gsql -d gaussdb -p 8000 -h 192.168.x.xx -U u1 -W password -r
gsql: FATAL:  permission denied for database "gaussdb"
DETAIL:  User does not have CONNECT privilege.
```

----结束

7.8 如何查看 DWS 某个用户有哪些表的权限?

场景一：查看用户有哪些表的权限，可使用information_schema.table_privileges系统表查看。例如

```
SELECT * FROM information_schema.table_privileges WHERE GRANTEE='user_name';
```

```
gaussdb=> SELECT * FROM information_schema.table_privileges WHERE GRANTEE='u2';
 grantor | grantee | table_catalog | table_schema | table_name | privilege_type | is_grantable | with_hierarchy
---------+---------+---------------+--------------+------------+----------------+--------------+----------------
 u2      | u2      | gaussdb       | u2           | t2         | INSERT         | YES          | NO
 u2      | u2      | gaussdb       | u2           | t2         | SELECT         | YES          | YES
 u2      | u2      | gaussdb       | u2           | t2         | UPDATE         | YES          | NO
 u2      | u2      | gaussdb       | u2           | t2         | DELETE         | YES          | NO
 u2      | u2      | gaussdb       | u2           | t2         | TRUNCATE       | YES          | NO
 u2      | u2      | gaussdb       | u2           | t2         | REFERENCES     | YES          | NO
 u2      | u2      | gaussdb       | u2           | t2         | TRIGGER        | YES          | NO
 u2      | u2      | gaussdb       | u2           | t2         | ANALYZE        | YES          | NO
 u2      | u2      | gaussdb       | u2           | t2         | VACUUM         | YES          | NO
 u2      | u2      | gaussdb       | u2           | t2         | ALTER          | YES          | NO
 u2      | u2      | gaussdb       | u2           | t2         | DROP           | YES          | NO
 u1      | u2      | gaussdb       | u1           | t1         | SELECT         | NO           | YES
(12 rows)
```

表 7-2 table_privileges 字段

字段	数据类型	描述
grantor	sql_identifier	赋权用户。
grantee	sql_identifier	被赋权用户。
table_catalog	sql_identifier	包含该表的数据库名。
table_schema	sql_identifier	包含该表的模式名。
table_name	sql_identifier	表名。
privilege_type	character_data	被赋予的权限类型：SELECT，INSERT，UPDATE，DELETE，TRUNCATE，REFERENCES，ANALYZE，VACUUM，ALTER，DROP或TRIGGER。
is_grantable	yes_or_no	权限是否可赋予其他用户，YES表示可授予，NO表示不可授予。
with_hierarchy	yes_or_no	是否允许在表继承层级上的特定操作。当特定操作为SELECT时显示YES，否则为NO。

如上图所示，表示用户u2拥有Schema u2下的**t2的所有权限**和Schema u1下的**t1的SELECT权限**。

需注意，在查询有哪些表权限时，information_schema.table_privileges只能查到当前用户被直接授予的权限，而函数has_table_privilege()除了能查询被直接授予的权限外还能查到间接的权限（即GRANT role to user获取的）。例如：

```
CREATE TABLE t1 (c1 int);
CREATE USER u1 password '********';
CREATE USER u2 password '********';
GRANT dbadmin to u2;     // 间接通过角色成员关系赋予权限
GRANT SELECT on t1 to u1;    // 直接授予权限

SET ROLE u1 password '********';
SELECT * FROM public.t1;    直接授权可以访问表
 c1
----
(0 rows)

SET ROLE u2 password '********';
SELECT * FROM public.t1;    //间接授权可以访问表
 c1
----
(0 rows)
```

```
RESET role;    //切换回到dbadmin
SELECT * FROM information_schema.table_privileges WHERE table_name = 't1';    //
information_schema.table_privileges仅能看到直接授权
 grantor |  grantee   | table_catalog | table_schema | table_name | privilege_type | is_grantable |
with_hierarchy
---------+------------+---------------+--------------+------------+----------------+--------------+----------------
 dbadmin | u1         | gaussdb       | public       | t1         | SELECT         | NO           | YES
(1 rows)

SELECT has_table_privilege('u2', 'public.t1', 'select');    // has_table_privilege还可以看到间接授权
 has_table_privilege
---------------------
 t
(1 row)
```

场景二：查看用户是否有某张表的权限，可以通过以下方法。

步骤1 执行以下语句查询pg_class系统表。

```
SELECT * FROM pg_class WHERE relname = 'tablename';
```

查看relacl字段，该字段回显结果如下，权限参数参见**表7-3**。

- "*rolename*=xxxx/yyyy" --表示rolename对该表有xxxx权限，且权限来自yyyy；
- "=xxxx/yyyy" -- 表示public对该表有xxxx权限，且权限来自yyyy。

例如下图：

joe=arwdDxtA，表示joe用户有所有权限（ALL PRIVILEGES）。

leo=arw/joe，表示leo用户拥有读、写、改权限，该权限来自joe授权。

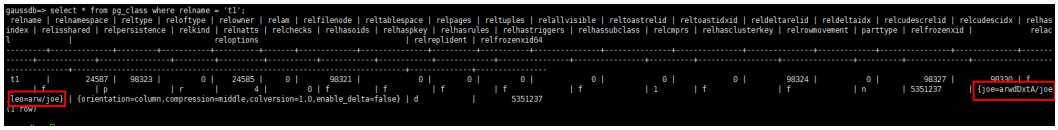


表 7-3 权限的参数说明

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

参数	参数说明
T	TEMPORARY
A	ANALYZE|ANALYSE
arwdDxtA	ALL PRIVILEGES（用于表）
*	给前面权限的授权选项

步骤2 如果要查某用户对某张表是否有某种权限，也可以通过访问权限查询函数has_table_privilege进行查询。

```
SELECT * FROM has_table_privilege('用户名','表名','select');
```

例如，查询joe对表t1是否有查询权限。

```
SELECT * FROM has_table_privilege('joe','t1','select');
```

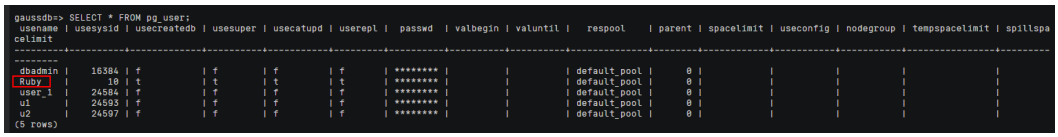
```
gaussdb=> select * from has_table_privilege('joe','t1','select');
 has_table_privilege
---------------------
 t
(1 row)
```

----结束

7.9 DWS 数据库中的 Ruby 是什么用户?

在执行SELECT * FROM pg_user语句查看当前系统的用户时，看到Ruby用户且拥有很多权限。

Ruby用户为官方运维使用账户，DWS数据库创建后，默认生成Ruby账户，不涉及安全风险，请放心使用。



8 数据库性能

8.1 为什么 DWS 使用一段时间后执行 SQL 很慢?

数据库在使用一段时间后，随着业务的增加使得表数据增加，或者对表数据经常进行增、删、改之后，引发数据膨胀和统计信息不准造成性能下降。

建议对于频繁增、删、改的表，定期执行vacuum full和analyze操作。操作步骤如下：

步骤1 默认每30000条数据收集100条做统计信息，当数据量大的时候，发现SQL时快时慢，一般是执行计划发生了变化，统计信息的需要调整采样率。set default_statistics_target可以提高采样率，对优化器生成最优计划有所帮助。

```
gaussdb=> set default_statistics_target=-2;
SET
```

步骤2 重新执行analyze。详细信息请参见**ANALYZE | ANALYSE**。

```
gaussdb=> ANALYZE customer_t1;
ANALYZE
```

----结束

说明

若用户想要知道是否是磁盘碎片的问题影响了数据库的性能，可以使用以下函数进行查询：

```
SELECT * FROM pgxc_get_stat_dirty_tables(30,100000);
```

8.2 为什么 DWS 的性能在极端场景下并未比单机数据库好

DWS中由于MPP架构的限制导致少部分PG系统方法、函数无法下推到DN节点来执行，仅能在CN端出现性能瓶颈。

原理解释：

- 一个操作能够并行执行是有条件的，需要逻辑上能够并行，比如做汇总（SUM），可以各个节点（DN）并行汇总后，最后的汇总一定是不能并行，要在某一个节点（CN）上执行，由于大部分的汇总工作已经在DN节点完成，CN端的工作是比较轻量的。

- 某些场景必须要集中执行，比如事务号，必须要保证全局唯一，该任务在系统里是通过GTM来实现的，因此，GTM也是全局唯一的组件（主备）。所有需要全局唯一的任务都是通过DWS中的GTM来完成，只是在设计上尽量避免阻塞在GTM上，因此GTM并没有太多瓶颈，而且有些场景下还可以GTM-Free和GTM-Lite。
- 从传统单机数据库的应用开发模式到并行数据库，为确保获得更好的性能，可能需要对业务进行少量改动，尤其是传统Oracle的存储过程互相嵌套的开发模式，如果要保证高性能，需要进行业务修改及对应的适配。

解决方案：

- 如遇到此类问题，请参考**《数据仓库服务数据库开发指南》**中的“优化查询性能”章节。
- 或者，请联系技术人员进行业务适配的修改调优。

8.3 DWS 业务读写阻塞，如何查看某个时间段的 SQL 执行记录?

当您的数据库集群出现读写阻塞时，可通过TopSQL功能查看某个时间段所执行的SQL语句，支持查看当前CN或者所有CN的sql语句。

TopSQL功能包括查看实时SQL语句和历史SQL语句：

- 实时SQL语句查询请参见：**实时TopSQL语句**。
- 历史SQL语句查询请参见：**历史TopSQL语句**。

8.4 DWS 中“算子下盘”是什么含义?

用户业务查询执行过程中，当集群内存不足时，数据库可能会选择将临时结果暂存到磁盘。当临时结果的磁盘存储量超过一定值后，用户会收到告警“查询语句触发下盘量超阈值”，那这个告警中的“下盘量”或者经常数据库中经常出现的“算子下盘”如何理解呢?

算子下盘的概念

任何计算都需要耗费内存空间，差别在于多少而已，对于如果耗费内存过多，会导致其他作业运行内存空间不足，导致作业不稳定，因此需要对查询语句的作业内存使用进行限制，保证作业运行的稳定性。

假如作业想要使用500M内存，但实际上因为内存限制最终只分配到300M内存，则需要把临时不用的数据写到磁盘上，内存中只保留正在使用的数据，这就是中间数据落盘的由来。当发生中间数据落盘时，称之为**算子下盘**。算子落盘空间太大除了会对查询性能有较大影响，还有可能导致数据库只读甚至磁盘满，因此DWS提供了用户算子空间限制，可以限制用户算子落盘的大小，在超限时查询报错退出。

哪些算子会发生下盘

当前DWS可下盘算子有六类（向量化及非向量化共10种）：Hash(VecHashJoin)，Agg(VecAgg)，Sort(VecSort)，Material(VecMaterial)， SetOp(VecSetOp)，WindowAgg(VecWindowAgg)。

哪些参数可以控制下盘

- work_mem：可以判断执行作业可下盘算子是否触发已使用内存量下盘点，当内存使用超过该参数后将触发算子下盘。该参数仅在非内存自适应场景（enable_dynamic_workload=off）时生效。work_mem参数设置通常是一个权衡，即要保证并发的吞吐量，又要保证单查询作业的性能，故需要根据实际执行情况（结合Explain Performance输出）进行调优。
- temp_file_limit：可以限制落盘算子的落盘文件大小，一般建议根据实际情况设置，防止下盘文件将磁盘空间占满，超过该值将报错退出。

如何判断语句是否发生了下盘

- 通过下盘文件确认：下盘文件位于实例目录的base/pgsql_tmp路径下，下盘文件以*pgsql_tmp$queryid_$pid*命名，可以根据queryid确认是哪条SQL发生了下盘。
- 根据等待视图（pgxc_thread_wait_status）确认：等待视图中，当出现write file时，表示发生了中间结果下盘。
- 根据执行计划确认（explain performance）：performance中出现spill、written disk、temp file num等关键字时，说明对应的算子出现了下盘。
- 根据topsql确认（前提：topsql功能已开启）：**实时TopSQL语句**或**历史TopSQL语句**中，spill_info字段中会包含下盘信息，如果该字段不为空，说明有DN实例出现了下盘。

如何避免下盘

发生算子下盘时，算子运算数据将写入磁盘，由于磁盘操作相对内存访问缓慢导致性能下降，查询响应时间出现极大劣化，因此应尽可能避免查询执行过程中的算子下盘，建议使用以下方法：

- 减小中间结果集：发生下盘时往往是由于中间结果集过大，因此可以增加过滤条件减少中间结果集大小。
- 避免数据倾斜：数据倾斜严重时会导致单DN上数据量过大，引起单DN下盘。
- 及时analyze：当统计信息不准时，行数估算可能偏小，导致计划选择非最优，从而出现下盘。
- 单点调优：对业务SQL进行单点调优。
- 非内存自适应场景下，当中间结果集无法减少时，应根据实际情况适当调大work_mem参数。
- 内存自适应场景下，内存使用配置要使得数据库可用内存尽量增大，减少下盘概率。

8.5 DWS 的 CPU 资源隔离管控介绍

CPU 资源管控概述

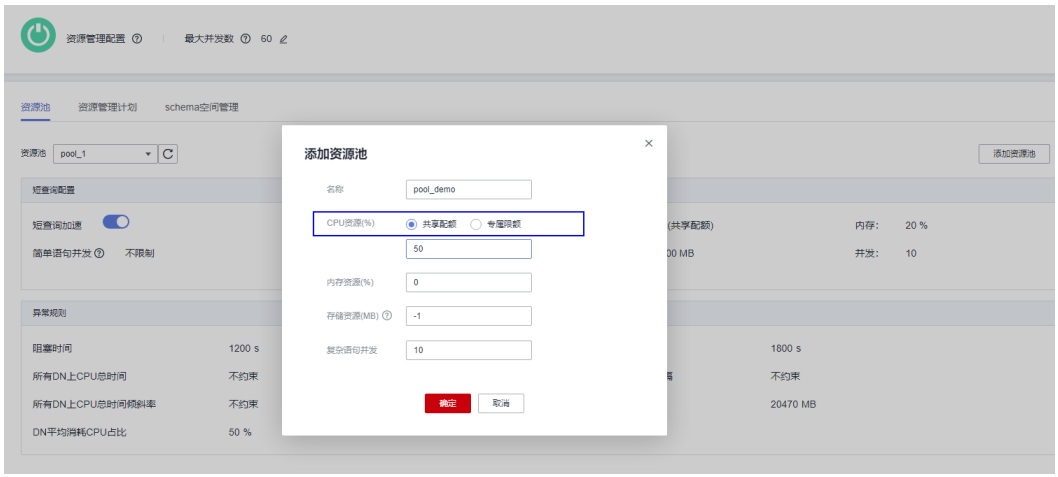
在不同的业务场景中，对数据库的系统资源（CPU资源、内存资源、IO资源和存储资源）进行合理的分配，保证执行查询时有充足的系统资源，确保查询性能，可以维持业务稳定性。

DWS的资源管理功能支持用户根据自身业务将资源按需划分成不同的资源池，不同资源池之间资源互相隔离。再通过关联数据库用户将其关联至不同的资源池，用户SQL查询时将根据“用户-资源池”的关联关系将查询转至资源池中执行。通过指定资源池

上可并行运行的查询数、单查询内存上限以及资源池可使用的内存和CPU资源，从而实现对不同业务之间的资源限制和隔离，满足数据库混合负载需求。

DWS主要利用cgroup（control group，控制组）进行CPU资源管控，涉及CPU、cpuacct、cpuset子系统。CPU共享配额管控基于CPU子系统的cpu.shares实现，该配置方法的好处是：OS CPU没有占满的情况下，不触发CPU管控；CPU专属限额管控基于cpuset实现；cpuacct子系统主要用于CPU资源使用的监控。

在**DWS控制台**使用资源管理配置功能创建资源池时，根据业务需要对CPU资源管理的“共享配额”和“专属配额”进行配置。



共享配额

共享配额：关联在当前资源池的用户在执行作业时可以使用的CPU时间比例。

共享配额有两层含义：

- 共享：CPU是所有控制组共享的，其他控制组能够使用空闲的CPU资源。
- 配额：业务繁忙、CPU满负载情况下，控制组之间按照配额比例进行CPU抢占。

共享配额基于cpu.shares实现，只有在CPU满负载情况下生效，因此在CPU空闲情况下并不能保证控制组能够抢占到配额比例的CPU资源。CPU空闲并不能理解为没有CPU资源争抢，控制组内任务可以任意使用CPU。虽然CPU平均使用率可能不高，但是某个特定时刻还是可能存在CPU资源争抢的。

例如：10个CPU上运行10个作业，每个CPU上运行1个作业，这种情况下各作业在任意时刻请求CPU都可以瞬间得到响应，作业之间没有任何CPU资源的争抢；但假如10个CPU上运行20个作业，因为作业不会一直占用CPU，在某些时间可能等待IO、网络等，因此CPU使用率可能并不高，此时CPU资源看似空闲，但是在某个时刻可能出现2~N作业同时请求一个CPU的情况出现，即会导致CPU资源争抢，影响作业性能。

专属限额

专属限额：限定资源池中数据库用户在执行作业时可使用的最大CPU核数占总核数的百分比。

专属限额有两层含义：

- 专属：CPU是某个控制组专属的，其他已设置限额的控制组不能使用空闲的CPU资源。

- 限额：只能使用限额配置的CPU资源，其他控制组空闲的CPU资源，也不能抢占。

专属限额基于cpuset.cpu实现，通过合理的限额设置可以实现控制组之间CPU资源的绝对隔离，各控制组间任务互不影响。但因为CPU的绝对隔离，因此在控制组空闲时就会导致CPU资源的极大浪费，因此限额设置不能太大。从作业性能来看并不是限额越大越好。

例如：10个作业运行在10个CPU上，CPU平均使用率5%左右；10个作业运行在5个CPU上，CPU平均使用率10%左右。通过上面共享配额的分析可知：虽然10个作业运行在5个CPU上CPU使用率很低，看似空闲，但是相对10个作业运行在10个CPU上还是存在某种程度的CPU资源争抢，因此10个作业运行在10个CPU上性能要好于运行在5个CPU上。但也不是越多越好，10个作业运行在20个CPU上，在任意一个时刻，总会至少10个CPU是空闲的，因此理论上10个作业运行在20个CPU上并不会比运行在10个CPU上性能更好。对于并发为N的控制组，分配cpus小于N的情况下，CPU越多作业性能越好；但是当分配CPUS大于N的情况下，性能就不会有任何提升了。

CPU 资源管理应用场景

CPU共享配额和专属限额的管控方式各有优劣，共享配额能够实现CPU资源的充分利用，但是各控制组之间资源隔离不彻底，可能影响查询性能；专属限额的管控方式可以实现CPU资源的绝对隔离，但是在CPU资源空闲时会造成CPU资源的浪费。相对专属限额来说，共享配额拥有更高的CPU使用率和更高的整体作业吞吐量；相对共享配额来说，专属限额CPU隔离彻底，更满足性能敏感用户的使用诉求。

数据库系统中运行多种类型作业出现CPU争抢时，可根据不同场景，选择不同的CPU资源管控方式：

- 场景一：实现CPU资源的充分利用，不关注单一类型作业的性能，主要关注CPU整体吞吐量。

  应用建议：不建议进行用户之间的CPU隔离管控，无论哪一种CPU管控都会对CPU整体使用率产生影响。
- 场景二：允许一定程度的CPU资源争抢和性能损耗，在CPU空闲情况下实现CPU资源充分利用，在CPU满负载情况下需要各业务类型按比例使用CPU。

  应用建议：可以采用基于cpu.shares的共享配额管控方式，在实现满负载CPU隔离管控前提下，尽量提高CPU整体使用率。
- 场景三：部分作业对性能敏感，允许CPU资源的浪费。

  应用建议：可以采用基于cpuset.cpu的专属限额管控方式，实现不同类型作业之间的CPU绝对隔离。

8.6 为什么 DWS 普通用户比 dbadmin 用户执行的慢?

DWS在使用过程中会出现普通用户比dbadmin用户执行慢的场景主要有以下三种：

场景一：普通用户受资源管理的管控

普通用户在排队：waiting in queue/waiting in global queue/waiting in ccn queue.

1. 普通用户主要在waiting in queue/waiting in global queue时。

   当前的活跃语句数超过max_active_statements限制导致的普通用户排队，由于管理员用户不受管控所以无需排队。可通过在**DWS控制台**修改max_active_statements参数值处理：

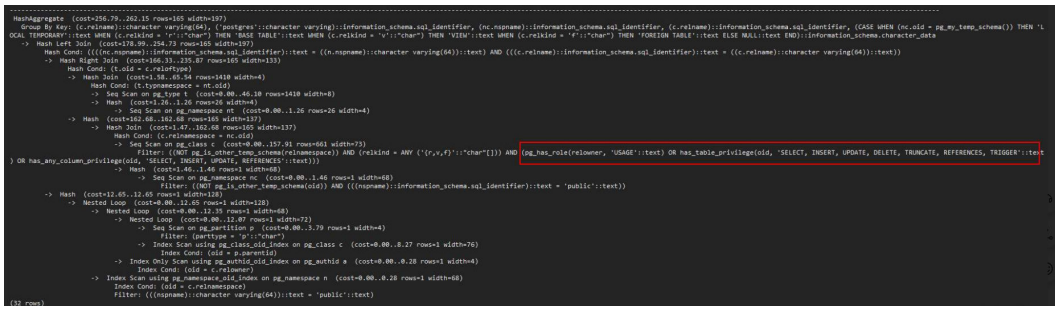
a. 登录**DWS控制台**。
b. 在左侧导航树，单击“集群 > 集群列表”。
c. 在集群列表中找到所需要的集群，单击集群名称，进入“集群详情”页面。
d. 单击“参数修改”页签，搜索“max_active_statements”参数，修改其参数值，单击“保存”，确认无误后再单击“保存”。

场景二：执行计划中的 or 条件对普通用户执行语句逐一判断耗时

执行计划中的or条件里有权限相关的判断，此场景多发生在使用系统视图时。例如以下SQL：

```
SELECT distinct(dtp.table_name),
 ta.table_catalog,
 ta.table_schema,
 ta.table_name,
 ta.table_type
from information_schema.tables ta left outer join DBA_TAB_PARTITIONS dtp
on (dtp.schema = ta.table_schema and dtp.table_name = ta.table_name)
where ta.table_schema = 'public';
```

一部分执行计划如下：



可以看到系统视图中的权限判断中多用or条件判断：

```
pg_has_role(c.relowner, 'USAGE'::text) OR has_table_privilege(c.oid, 'SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER'::text) OR has_any_column_privilege(c.oid, 'SELECT, INSERT, UPDATE, REFERENCES'::text)
```

由于dbadmin用户pg_has_role总能返回true，因此or之后的条件无需继续判断；

而普通用户的or条件需要逐一判断，如果数据库中表个数比较多，最终会导致普通用户比dbadmin需要更长的执行时间。

这种场景如果输出结果集很少，可以考虑尝试设置set enable_hashjoin = off; set enable_seqscan = off; 走index + nestloop的计划。

场景三：普通用户和管理员用户所分配资源池有差异

通过执行如下查询命令，查看用户所对应的资源池是否相同，如果不同，可在界面查看两个资源池上所分配的租户资源是否有差别。

```
SELECt * FROM pg_user;
```

8.7 DWS 中单表查询性能与哪些因素有关?

DWS采用Shared-nothing架构，数据是被分布式存储，因此分布键设计、单表存储数据量、分区数量都会影响单表的整体查询性能。

1. 分布键设计

   DWS默认会选择主键的第一列作为分布键。如果同时设置主键和分布键，则主键必须包含分布键。分布键决定了数据在各个分区之间的分布情况，如果分布键很好地分布在各个分区中，则可以使查询性能变得更好。

   分布列选择不当，在数据导入后有可能出现数据分布倾斜，进而导致某些磁盘的使用明显高于其他磁盘，极端情况下会导致集群只读。合理的选择分布键，对表查询的性能至关重要。此外，合适的分布键还可以使数据的索引更快地创建和维护。

2. 单表存储数据量

   单表存储的数据量越大，查询性能就越差。当表中的数据量很大时，则需要考虑将数据进行分区存储。普通表若要转成分区表，需要新建分区表，然后把普通表中的数据导入到新建的分区表中。因此在初始设计表时，请根据业务提前规划是否使用分区表。

   对表进行分区，一般需要遵循以下原则：

   - 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。
   - 分区名称应当体现分区的数据特征。比如关键字+区间特征。
   - 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。

3. 分区数量

   利用分区，可以将表和索引划分为一些更小、更易管理的单元。大幅减少搜索空间，从而提升访问性能。

   使用分区数量会影响查询的性能。如果分区数量太小，则可能会使查询性能下降。

   DWS支持范围分区（Range Partitioning）和列表分区（List Partitioning）功能，即根据表的一列或者多列，将要插入表的记录分为若干个范围（这些范围在不同的分区里没有重叠），然后为每个范围创建一个分区，用来存储相应的数据。其中，列表分区（List Partitioning）仅8.1.3及以上集群版本支持。

因此，在设计数据仓库时，需要考虑这些因素并进行实验来确定最佳设计方案。

8.8 DWS 表膨胀原因有哪些？该如何处理？

表膨胀的原因

- 未开启autovacuum

  DWS提供自动执行VACUUM和ANALYZE命令的系统自动清理进程（autovacuum），用于回收被标识为删除状态的记录空间，并更新表的统计数据。

  用户未开启autovacuum的同时又没有合理的自定义vacuum调度，导致表的脏数据没有及时回收，新的数据又不断插入或更新，膨胀是必然的。

- 资源回收不及时

  开启了autovacuum，但是各种原因导致回收不及时，并且新的数据又不断产生，从而导致膨胀。回收不及时有以下原因：

  - IO性能差

    当数据库非常繁忙时，如果IO性能较差，会导致回收脏数据变慢，从而导致表膨胀。

    这种情况一般出现在占用数据库内存较大的表上，并且这些表正在执行整表vacuum，因此产生大量IO，导致表自身或其他表膨胀。

- autovacuum触发较迟

  触发autovacuum操作的阈值设置过高，大量表上被删除、插入或更新的记录数没有及时处理，导致表膨胀。

- autovacuum线程繁忙

  所有自动清理线程繁忙，某些表产生的脏数据超过阈值，但是在此期间没有autovacuum线程可以处理脏数据回收的事情，可能发生表膨胀。

  如果数据库的表很多，而且都比较大，那么当需要vacuum的表超过了配置autovacuum_max_workers的数量，这些表就要等待空闲的autovacuum线程。这个阶段就容易出现表的膨胀。

- 数据库中存在长SQL或带XID的长事务

  当DWS数据库中存在未结束的SQL语句或者未结束的持有事务ID的事务，在此SQL执行时间范围内或在此事务过程中产生的脏数据无法回收，导致数据库膨胀。

- 开启了autovacuum_vacuum_cost_delay

  在开启了autovacuum_vacuum_cost_delay后，会使用基于成本的脏数据回收策略，可以有利于降低VACUUM带来的IO影响，但是对于IO性能高的系统，开启autovacuum_vacuum_cost_delay反而会使得垃圾回收的时间变长。

- autovacuum_naptime设置间隔时间过长
- 批量删除或更新大表

  例如对于一个10GB的表，一条SQL或一个事务中删除或更新9GB的数据，这9GB的数据必须在事务结束后才能进行脏数据回收，无形中增加了膨胀的可能。

减少或避免表膨胀

- 开启autovacuum。
- 提高系统的IO能力。
- 调整触发阈值，让触发阈值和记录数匹配。
- 增加autovacuum_max_workers和autovacuum_work_mem，同时增加系统内存。
- IO性能较好的系统，关闭autovacuum_vacuum_cost_delay。
- 设计应用程序时，避免使用大批量的更新、删除操作，可以切分为多个事务进行。
- 应用程序设计时，尽量避免下列操作：
  - 打开游标后不关闭。
  - 在不必要的场景使用repeatable read或serializable事务隔离级别。
  - 对大的数据库执行gs_dump进行逻辑备份（隐式repeatable read隔离级别的全库备份）。
  - 长时间不关闭申请了事务号的事务（增、删、改等DDL操作的SQL）。

相关空间回收参数说明

- autovacuum = on

  控制数据库自动清理进程（autovacuum）的启动。自动清理进程运行的前提是将track_counts设置为on。

- log_autovacuum_min_duration = 0

  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。

- autovacuum_max_workers = 10

  设置能同时运行的自动清理线程的最大数量。

- autovacuum_naptime = 1

  设置两次自动清理操作的时间间隔。

- autovacuum_vacuum_cost_delay = 0

  设置在自动VACUUM操作里使用的开销延迟数值。

更多关于空间回收参数说明，参见**自动清理**。

8.9 如何优化包含多个 CASE WHEN 条件的 SQL 查询?

在业务查询中，CASE WHEN语句常用来进行条件判断，但如果在SQL查询中存在大量冗余的CASE WHEN，例如：

```
SELECT
  SUM(CASE WHEN a > 1 THEN 1 ELSE 0 END) AS a1,
  SUM(CASE WHEN a > 2 THEN 1 ELSE 0 END) AS a2,
  ...
FROM test
WHERE dt = '20241225';
```

该语句冗长，执行时每个分支的CASE WHEN均需执行，导致查询时间成倍增加，影响查询性能。

DWS提供以下优化策略来解决此类问题。

使用临时结果集或者子查询

将复杂的CASE WHEN计算部分提取出来，放到一个临时的结果集中或者子查询中。这样可以减少在主查询中的重复计算逻辑。

例如，先创建一个子查询来计算中间结果：

```
SELECT
  sub.a1,
  sub.a2
FROM (
  SELECT
    sum(case when a > 1 then 1 else 0 end) AS a1,
    sum(case when a > 2 then 1 else 0 end) AS a2
  FROM test
  WHERE dt = '20241225'
) sub;


SELECT
    SUM(case_when_a1) as a1,
    SUM(case_when_a2) as a2,
    ...
FROM (
    SELECT
        CASE WHEN a > 1 THEN 1 ELSE 0 END AS case_when_a1,
        CASE WHEN a > 2 THEN 1 ELSE 0 END AS case_when_a2,
        ...
    FROM test
```

```
    WHERE dt = '20241225'
) AS subquery;
```

使用自定义函数

将CASE WHEN的逻辑封装成一个函数。这样在查询中只需要调用该函数，而不是多次编写相同的CASE WHEN逻辑。

例如，创建一个简单的函数count_a_gt_value。

```
CREATE OR REPLACE FUNCTION count_a_gt_value(val INT)
RETURNS INT AS $$
DECLARE
    result INT;
BEGIN
    SELECT sum(CASE WHEN a > val THEN 1 ELSE 0 END)
      INTO result
      FROM test
      WHERE dt = '20241225';
    RETURN result;
END;
$$ LANGUAGE plpgsql;
```

再使用自定义函数count_a_gt_value进行查询。

```
SELECT
  count_a_gt_value(1) AS a1,
  count_a_gt_value(2) AS a2
FROM test;
```

9 备份恢复

9.1 为什么 DWS 自动快照创建很慢，很长时间都没有创建好?

自动快照备份很慢，可能是由于所需备份的数据量较大。自动快照是增量备份，备份频率是可以设置的，如果一周只备份一次，当增量数据量较大时，就会出现备份慢的情况。建议您适当地增加备份频率。

其中快照备份和恢复速率如下所示（此速率为实验室测试环境下数据，介质为SSD本地盘，仅供用户参考。在实际使用中，由于磁盘、网络、带宽等因素可能会产生一定的差异）：

- 备份速率：200 MB/s/DN
- 恢复速率：125 MB/s/DN

9.2 DWS 快照是否与 EVS 快照功能相同?

数据仓库服务的快照功能与云硬盘快照不同。

数据仓库服务的快照针对DWS集群的所有配置数据和业务数据，用于还原创建快照时的集群数据；云硬盘快照是针对于云服务器的数据盘或者系统盘的某个时段的业务数据。

DWS 快照

数据仓库服务快照是对DWS集群在某一时间点的一次全量数据和增量数据的备份，记录了当前数据库的数据以及集群的相关信息，其中包括节点数量、节点规格和数据库管理员用户名称等。快照创建方式包括手动创建快照和自动创建快照。

从快照恢复到集群时，DWS会根据快照记录的集群信息来创建新集群，然后从快照记录的数据中还原数据库信息。

有关快照的更多内容，请参见**管理快照**。

EVS 快照

云硬盘快照指的是云硬盘数据在某个时刻的完整拷贝或镜像，是一种重要的数据容灾手段，当数据丢失时，可通过快照将数据完整的恢复到快照时间点。

您可以创建快照，从而快速保存指定时刻云硬盘的数据。同时，您还可以通过快照创建新的云硬盘，这样云硬盘在初始状态就具有快照中的数据。

创建快照可以快速保存指定时刻云硬盘的数据，实现数据容灾：

- 当数据丢失时，可通过快照将数据完整的恢复到快照时间点。
- 通过快照创建新的云硬盘，新的云硬盘具有快照中的数据。

有关快照的更多内容，请参见**云硬盘快照**。

