数据仓库服务（DWS）

故障排除

天翼云科技有限公司

目 录

1 数据库连接

1.1 执行 gsql 连接数据库命令提示 gsql：command not found

问题现象

执行 gsql -d postgres -p 26000 -r 出现如下错误：

```
gsql: command not found…
```

原因分析

- 没有在 gsql 的 bin 目录下执行。
- 未执行环境变量。

处理方法

步骤 1 在客户端目录下执行环境变量，例如客户端在/opt 目录下。

cd /opt

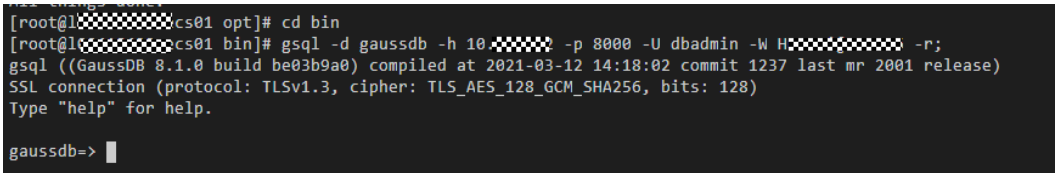
source gsql_env.sh

```
[root@l[redacted]1 opt]# ll
total 16300
drwxr-xr-x 2 root root     4096 Mar 12  2021 bin
-rw-r--r-- 1 root root 16668016 May  6 14:41 dws_client_8.1.x_redhat_x64.zip
drwxr-xr-x 5 root root     4096 Mar 12  2021 gds
-rwxr-xr-x 1 root root     1465 Mar 12  2021 gsql_env.sh
drwxr-xr-x 3 root root     4096 Mar 12  2021 lib
drwxr-xr-x 3 root root     4096 Mar 12  2021 sample
[root@l[redacted]1 opt]# source gsql_env.sh
Configuring LD_LIBRARY_PATH and PATH for gsql ............... done
All things done.
[root@l[redacted] opt]#
```

步骤 2 进入 gsql 的 bin 目录下，执行 gsql 命令进行数据库连接。

cd bin

gsql -d gaussdb -h *数据库IP* **-p 8000 -U dbadmin -W** *数据库用户密码* **-r;**



----结束

1.2 通过 gsql 客户端无法连接数据库

问题现象

用户通过客户端工具 gsql 无法连接到数据库。

原因分析

- 系统连接数量超过了最大连接数量，会显示如下错误信息。

```
gsql -d human_resource -h 10.168.0.74 -U user1 -p 8000 -W password -r
gsql: FATAL:  sorry, too many clients already
```

- 用户不具备访问该数据库的权限，会显示如下错误信息。

```
gsql -d human_resource -h 10.168.0.74 -U user1 -p 8000 -W password -r
gsql: FATAL:  permission denied for database "human_resource"
DETAIL:  User does not have CONNECT privilege.
```

- 网络连接故障。

解决办法

- 系统连接超过最大连接数量。

  用户可在 GaussDB(DWS) 控制台设置最大连接数 max_connections。

  max_connections 设置方法如下：

  a. 登录 GaussDB(DWS) 管理控制台。

  b. 在左侧导航栏中，单击“集群管理”。

  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“基本信息”页面。

  d. 单击“参数修改”页签，修改参数“max_connections”的值，然后单击“保存”。

  e. 在“修改预览”窗口，确认修改无误后，单击“保存”。

  关于查看用户会话连接数的方法如表 1-1。

表1-1 查看会话连接数

描述	命令
查看指定用户的会话连	执行如下命令查看连接到指定用户 USER1 的会话连接数

描述	命令

接数上限。	上限。其中-1 表示没有对用户 user1 设置连接数的限制。
	`SELECT ROLNAME,ROLCONNLIMIT FROM PG_ROLES WHERE ROLNAME='user1';`
	`rolname | rolconnlimit`
	`---------+--------------`
	`user1 | -1`
	`(1 row)`

查看指定用户已使用的会话连接数。	执行如下命令查看指定用户 USER1 已使用的会话连接数。其中，1 表示 USER1 已使用的会话连接数。
	`SELECT COUNT(*) FROM V$SESSION WHERE USERNAME='user1';`
	`count`
	`-------`
	`1`
	`(1 row)`

查看指定数据库的会话连接数上限。	执行如下命令查看连接到指定数据库 guassdb 的会话连接数上限。其中-1 表示没有对数据库 guassdb 设置连接数的限制。
	`SELECT DATNAME,DATCONNLIMIT FROM PG_DATABASE WHERE DATNAME='guassdb';`
	`datname | datconnlimit`
	`----------+--------------`
	`guassdb | -1`
	`(1 row)`

查看指定数据库已使用的会话连接数。	执行如下命令查看指定数据库 guassdb 上已使用的会话连接数。其中，1 表示数据库 guassdb 上已使用的会话连接数。
	`SELECT COUNT(*) FROM PG_STAT_ACTIVITY WHERE DATNAME='guassdb';`
	`count`
	`-------`
	`1`
	`(1 row)`

查看所有用户已使用会话连接数。	执行如下命令查看所有用户已使用的会话连接数。
	`SELECT COUNT(*) FROM PG_STAT_ACTIVITY;`
	`count`
	`-------`
	`10`
	`(1 row)`

- 用户不具备访问该数据库的权限。
  a. 使用管理员用户 dbadmin 连接数据库。

     ```
     gsql -d human_resource -h 10.168.0.74 -U dbadmin -p 8000 -W password -r
     ```

b. 赋予该用户访问数据库的权限。

```
GRANT CONNECT ON DATABASE human_resource TO user1;
```

说明

实际上，常见的许多错误操作也可能产生用户无法连接上数据库的现象。如用户连接的数据库不存在，用户名或密码输入错误等。这些错误操作在客户端工具也有相应的提示信息。

```
gsql -d human_resource -p 8000
gsql: FATAL:  database "human_resource" does not exist

gsql -d human_resource -U user1 -W gauss@789 -p 8000
gsql: FATAL:  Invalid username/password,login denied.
```

- 网络连接故障。

  请检查客户端与数据库服务器间的网络连接。如果发现从客户端无法 PING 到数据库服务器端，则说明网络连接出现故障。请联系技术支持工程师提供技术支持。

```
ping -c 4 10.10.10.1
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
From 10.10.10.1: icmp_seq=2 Destination Host Unreachable
From 10.10.10.1 icmp_seq=2 Destination Host Unreachable
From 10.10.10.1 icmp_seq=3 Destination Host Unreachable
From 10.10.10.1 icmp_seq=4 Destination Host Unreachable
--- 10.10.10.1 ping statistics ---
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 2999ms
```

1.3 连接 GaussDB(DWS) 数据库时，提示客户端连接数太多

问题现象

连接 GaussDB(DWS) 数据库时报错，提示客户端连接数太多。

- 使用 gsql 等 SQL 客户端工具连接数据库时，出现如下所示的报错信息：

```
FATAL:  Already too many clients, active/non-active/reserved: 5/508/3.
```

- 使用客户端并发连接数据库时，出现如下所示的报错信息：

```
[2019/12/25 08:30:35] [ERROR] ERROR: pooler: failed to create connections in
parallel mode for thread 140530192938752, Error Message: FATAL:  dn_6001_6002:
Too many clients already, active/non-active: 468/63.
FATAL:  dn_6001_6002: Too many clients already, active/non-active: 468/63.
```

原因分析

- **当前数据库连接已经超过了最大连接数**

  错误信息中，non-active 的个数表示空闲连接数，例如，non-active 为 508，说明当前有大量的空闲连接。

- **创建用户时设置了该用户的最大连接数**

  查询数据库连接数，如果显示连接数未达设定上限，可能是由于创建用户时设置了该用户的最大连接数。

处理方法

出现该问题时，建议通过如下方法进行处理：

1. 临时将所有 non-active 的连接释放掉。

   ```
   SELECT PG_TERMINATE_BACKEND(pid) from pg_stat_activity WHERE state='idle';
   ```

2. 检查应用程序是否未主动释放连接，导致连接残留。建议优化代码，合理释放连接。
3. 在 GaussDB(DWS) 控制台设置会话闲置超时时长 session_timeout，在闲置会话超过所设定的时间后服务端将主动关闭连接。

   session_timeout 默认值为 600 秒，设置为 0 表示关闭超时限制，一般不建议设置为 0。

   session_timeout 设置方法如下：

   a. 登录 GaussDB(DWS) 管理控制台。
   b. 在左侧导航栏中，单击“集群管理”。
   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“基本信息”页面。
   d. 单击“参数修改”页签，修改参数“session_timeout”，然后单击“保存”。
   e. 在“修改预览”窗口，确认修改无误后，单击“保存”。
4. 如果当前的最大连接数无法满足业务需求，您也可以通过管理控制台调整最大连接数 max_connections。

   max_connections 具体设置方法参考 3，

   在管理控制台上，集群“基本信息”页面，单击“参数修改”页签，修改参数“max_connections”为合适的值，然后单击“保存”。

1.4 无法 ping 通/连接集群访问地址

问题现象

在客户端主机上，无法 ping 通 GaussDB(DWS) 集群访问地址。

原因分析

- **网络不通**

  如果客户端主机通过 GaussDB(DWS) 集群的内网地址进行连接，需要排查客户端主机跟 GaussDB(DWS) 集群是否在相同的 VPC 和子网内，如果不在相同的 VPC 和子网内，则网络不通。
- **安全组规则禁止 ping**

GaussDB(DWS) 集群所属的安全组入规则需要放开 ICMP 协议端口才能允许 ping，如果未开放 ICMP 协议端口，就无法 ping 通。创建 GaussDB(DWS) 集群时自动创建的安全组默认**只放开了 TCP 协议和 8000 端口**。

如果安全组入规则已开放 ICMP 协议端口，需要检查相应入规则的源地址是否涵盖了客户端主机的 IP 地址，如果没有，也无法 ping 通。

处理方法

- **网络不通**

  如果客户端主机通过 GaussDB(DWS) 集群的内网地址进行连接，应重新申请一台弹性云主机作为客户端主机，且该弹性云主机必须和 GaussDB(DWS) 集群处于相同的 VPC 和子网内。

- **安全组规则禁止 ping**

  检查 GaussDB(DWS) 集群所属的安全组规则，查看是否对客户端主机的 IP 地址开放了 ICMP 协议端口。具体操作如下：

  a. 登录 GaussDB(DWS) 管理控制台。

  b. 在“集群管理”页面，找到所需要的集群，点击集群名称进入“基本信息”页面。

  c. 在“基本信息”页面，找到“安全组”参数，单击安全组名称，进入相应的安全组详情页面。

  d. 进入“入方向规则”页签，检查是否存在开放 ICMP 协议端口的入规则，如果不存在，请单击“添加规则”按钮，添加入方向规则开放 ICMP 协议端口。

  - 协议端口：选择“ICMP”和“全部”。
  - 源地址：选择“IP 地址”，然后根据客户端主机的 IP 地址输入相应的 IP 地址与掩码。0.0.0.0/0 表示任意地址。

  图1-1 入方向规则

  e. 单击“确定”完成入规则的添加。

2 JDBC 连接报错

2.1 JDBC 问题定位

JDBC 问题是比较宽泛的，现网大部分问题主要体现在 JDBC 层面，造成 JDBC 问题的原因主要是三个方面：

1. 应用程序和应用程序框架问题。
2. JDBC 业务功能问题。
3. 数据库配置问题。

问题表现可以分为三个大的方面：

1. 执行报错，JDBC 抛出异常。
2. 执行效率低，耗时异常。
3. 特性不支持，JDBC 未实现的 JDK 接口。

JDBC 问题具体分类可参见表 2-1。

表2-1 JDBC 问题分类

问题分类	问题原因
2.2 建立数据库连接失败	JDBC 客户端配置问题：包括 URL 格式不对，或用户名密码错误。
	网络不通。
	Jar 包冲突。
	数据库配置问题，数据库未配置远程访问权限。
2.3 执行业务抛出异常	传入 SQL 有误，内核不支持。
	内核 BUG，业务处理异常，返回异常报文。
	网络故障。
	数据库链接超时，socket 已关闭。

问题分类	问题原因
2.4 性能问题	SQL 内核执行慢。
	结果集过大，导致应用程序段响应慢。
	用户传入 SQL 过长，JDBC 解析慢。
2.5 功能支持问题	JDK 未提供标准接口。
	JDBC 未实现接口。

2.2 建立数据库连接失败

Check that the hostname and port are correct and that the postmaster is accepting TCP/IP connections.

问题分析：可能原因为客户端与服务端网络不通、或端口错误或待连接 CN 异常。

处理方法：

- 客户端 ping 服务端 IP，看网络是否畅通，网络不通则解决网络问题。
- 检查 URL 中连接 CN 的端口是否正确，端口不正确修改为正确的端口（默认为 8000）。

FATAL: Invalid username/password,login denied.

问题分析：用户名或密码配置错误。

处理方法：检查用户名密码是否为数据库用户名密码，将其修改为正确的数据库用户名密码。

No suitable driver found for XXXX

问题分析：URL 格式书写错误。

处理方法：将 URL 格式修改为正确的格式。

- gsjdbc4.jar 对应 jdbc:postgresql://host:port/database
  - 在使用 pom 依赖时对应 8.1.x 版本
- gsjdbc200.jar 对应 jdbc:gaussdb://host:port/database
  - 在使用 pom 依赖时对应 8.1.x-200 版本

conflict

问题分析：JDBC jar 包和应用程序冲突。例如 JDBC 和应用程序拥有相同路径相同名称的类导致：

- gsjdbc4.jar 和开源 postgresql.jar 冲突，两者具有完全相同的类名。
- gsjdbc4.jar 由于 iam 特性引入了一些其他工具，例如 fastjson，和应用程序中的 fastjson 冲突。

处理方法：

- 针对 JDBC 引入的 jar 和应用程序中引入 jar 的冲突，可以通过 maven 的 shade 修改了 jar 里类的路径，解决了这类冲突。
- 排查使用的 jdbc 驱动是 gsjdbc4.jar 还是 gsjdbc200.jar，若是 gsjdbc4.jar 应该替换为 gsjdbc200.jar，尝试建立连接。

说明

对于 pom 依赖，将对于 8.1.x 版本替换为 8.1.x-200 版本。

org.postgresql.util.PSQLException: FATAL: terminating connection due to administrator command Session unused timeout

问题分析：会话超时导致连接断开。

处理方法：排查 CN 和客户端 JDBC 上的超时配置，按业务实际情况调长超时时间或关闭超时设置。

1. 查看报错的 CN 日志，如果有 session unused timeout 这样的日志，说明是会话超时导致的。

   解决办法：

   a. 登录控制台单击指定集群名称。

   b. 在左侧导航栏选择“参数修改”，搜索参数“session_timeout”查看超时时间参数。

   c. 将 session_timeout 的 CN、DN 参数值设置为 0，详情可参见《数据仓库服务用户指南》中“修改数据库参数”章节。

Connection refused: connect.

问题分析：第三方工具的默认驱动不兼容。

处理方法：用户可更换 JDBC 驱动包，查看是否正常连接。

Connections could not be acquired from the underlying database!

问题分析：按照新建连接排查项进行排查：

- 驱动配置是否有误。
- 数据库连接地址是否有误。
- 密码或帐号是否有误。

- 数据库未启动或无权访问。
- 项目未引入对应的驱动 jar 包。

处理方法：

- 排查驱动配置，将其修改为正确的驱动配置。
  - gsjdbc4.jar　driver=org.postgresql.Driver
- 排查数据库连接地址，将其修改为正确的数据库连接地址。
  - gsjdbc4.jar 对应 jdbc:postgresql://host:port/database
- 排查用户名密码是否为数据库用户名密码，将其修改为正确的数据库用户名密码。
- 排查数据库是否启动或有权限访问。
- 检查使用的 JDBC 驱动是 gsjdbc4.jar 还是 gsjdbc200.jar，请使用正确 JDBC 驱动 jar 包。
  - gsjdbc4.jar：与 PostgreSQL 保持兼容，其中类名、类结构与 PostgreSQL 驱动完全一致，曾经运行于 PostgreSQL 的应用程序可以直接移植到当前系统中使用。

JDBC DEV 环境没问题，测试环境连接出错报空指针或 URI 报错 uri is not hierarchical

问题分析：某些虚拟环境不支持获取扩展参数，需关闭。

处理方法：在连接时可设置连接参数“connectionExtraInfo=false”，详情可参见《数据仓库服务用户指南》中“使用 JDBC 连接数据库”章节。

```
jdbc:postgresql://host:port/database?connectionExtraInfo=false
```

使用开源 JDBC SSL 方式连接 DWS 报错

问题分析：使用开源 JDBC 会进行 SSL 全校验，校验 url 是否完全匹配。

处理方法：在开源连接时可设置连接参数“sslmode=require”。

```
jdbc:postgresql://host:port/database?sslmode=require
```

在 CopyManager 场景使用连接池获取连接，Connection 无法转换为 BaseConnection

问题分析：BaseConnection 为非公开类，需要对连接池对象解封装然后获取原始 PGConnection。

处理方法： 对当前对象解包装，返回原始对象以允许访问未公开方法。

```
//解封装
PGConnection unwrap = connection.unwrap(PGConnection.class);
//转换 BaseConnection
BaseConnection baseConnection = (BaseConnection)unwrap;
CopyManager copyManager = new CopyManager(baseConnection);
```

2.3 执行业务抛出异常

Broken pipe, connection reset by peer

问题分析：网络故障，数据库连接超时。

处理方法：检查网络状态，修复网络故障，影响数据库连接超时的因素，例如数据库参数 session_timeout。

步骤 1 登录控制台单击指定集群名称。

步骤 2 在左侧导航栏选择“参数修改”，搜索参数“session_timeout”查看超时时间参数。

步骤 3 将 session_timeout 的 CN、DN 参数值设置为 0，详情可参见《数据仓库服务用户指南》中“修改数据库参数”章节。

----结束

The column index is out of range

问题分析：应用程序获取的结果集和预期不一致，列数不一致。

处理方法：检查数据库表定义和查询 SQL，对返回结果集做一个正确预期，若结果集只有 3 列，取值时传入的 index 最大为 3。

2.4 性能问题

在 processResult 阶段耗时

设置 loglevel=3，打开 JDBC 日志，主要耗时在 processResult 阶段，可分为两种情况：

1. JDBC 端等待数据库返回的报文时间过长。

   问题分析：用户可查看 FE=> Syncr 日志和<=BE ParseComplete 日志之间的时间间隔，若时间间隔较久，则判断为内核执行慢。

   处理方法：分析 SQL 执行慢的原因，详情可参见《数据仓库服务故障排除》中“SQL 执行很慢，性能低，有时长时间运行未结束”章节。

2. 结果集过大，一次性全部加载，消耗大量时间。

   问题分析：查看日志，若<=BE DataRow 日志出现次数过多，或直接执行 select count(*)查询结果数目过大，则判断为结果集过大。

处理方法：设置 fetchSize 参数为一个较小的值，使数据按批次返回，客户端得到快速响应。

```
statement.setFetchSize(10);
```

在 modifyJdbcCall 和 createParameterizedQuery 阶段耗时

问题分析：若主要耗时在 modifyJdbcCall 阶段（校验传入的 sql 是否符合规范）和 createParameterizedQuery 阶段（将传入的 sql 解析为 preparedQuery,以获取由 simplequery 组成的 subqueries）阶段，则需要看一下传入的 sql 时间是否过长。

处理方法：JDBC 本身没办法优化这部分耗时，可在应用端查看是否可优化传入的 sql 语句。

2.5 功能支持问题

not yet implemented

问题分析：JDBC 未实现接口。

处理方法：需要技术人员研究接口是否可实现，或是否有其他接口已提供相同功能，调整业务使用已提供接口。

JDK 标准接口中未提供相关功能

问题分析：JDK 未提供标准接口

处理方法：理论上若 JDK 未提供接口，则 JDBC 不支持。实际使用中可以使用 JDBC 类中的 public 方法获取部分过程数据，绝大部分情况下明确不支持。

3 数据导入导出

3.1 使用 copy from 导入 GaussDB(DWS)时报 ERROR: invalid byte sequence for encoding "UTF8": 0x00 错误

问题现象

使用 copy from 导入 GaussDB(DWS)时，报 ERROR: invalid byte sequence for encoding "UTF8": 0x00 错误

原因分析

数据文件是从 oracle 导入的，文件编码为 utf-8。这个报错还会提示行数，由于文件特别大，vim 打不开文件，于是用 sed 命令把报错行数提出来，再用 vim 打开，发现并没有什么异常。用 split 命令按行数切割后，部分文件也可以导入。

参考 GaussDB(DWS)官方文档发现，直接原因是 varchar 型的字段或变量不接受含有'\0'（也即数值 0x00、UTF 编码'\u0000'）的字符串 ，给出的解决方法：事先去掉字符串中的'\0'。

处理方法

用 sed 命令替换掉 0x00

```
sed -i 's/\x00//g;' file
```

说明

参数说明 -i 表示在原文件直接替换，s/表示替换，/g 表示全局替换。

3.2 GDS 导入/导出类问题

GDS 导入/导出容易遇到字符集的问题，特别是不同类型的数据库或者不同编码类型的数据库进行迁移的过程中，往往会导致数据入不了库，严重阻塞数据迁移等现场作业。

区域支持

区域支持指的是应用遵守文化偏好的问题，包括字母表、排序、数字格式等。区域是在使用 initdb 创建一个数据库时自动被初始化的。默认情况下，initdb 将会按照它的执行环境的区域设置初始化数据库，即系统已经设置好的区域。如果想要使用其他的区域，可以使用手工指定（initdb －locale=xx）。

如果想要将几种区域的规则混合起来，可以使用以下区域子类来控制本地化规则的某些方面。这些类名转换成 initdb 的选项名来覆盖某个特定分类的区域选择。

表3-1 区域支持

字段	描述
LC_COLLATE	字符串排序顺序
LC_CTYPE	字符分类（什么是一个字符？它的大写形式是否等效？）
LC_MESSAGES	消息使用的语言 Language of messages
LC_MONETARY	货币数量使用的格式
LC_NUMERIC	数字的格式
LC_TIME	日期和时间的格式

如果想要系统表现得没有区域支持，可以使用区域 C 或者等效的 POSIX。使用非 C 或非 POSIX 区域的缺点是性能影响。它降低了字符处理的速度并且阻止了在 LIKE 中对普通索引的使用。因此，只能在真正需要的时候才使用它。

一些区域分类的值必需在数据库被创建时的就被固定。不同的数据库可以使用不同的设置，但是一旦一个数据库被创建，就不能在数据库上修改这些区域分类的值。LC_COLLATE 和 LC_CTYPE 就属于上述情形。它们影响索引的排序顺序，因此它们必需保持固定，否则在文本列上的索引将会崩溃。这些分类的默认值在 initdb 运行时被确定，并且这些值在新数据库被创建时使用，除非在 CREATE DATABASE 命令中特别指定。其它区域分类可以在任何时候被更改，更改的方式是设置与区域分类同名的服务器配置参数。被 initdb 选中的值实际上只是被写入到配置文件 postgresql.conf 中作为服务器启动时的默认值。如果你将这些赋值从 postgresql.conf 中除去，那么服务器将会从其执行环境中继承该设置。

区域设置特别影响下面的 SQL 特性：

- 在文本数据上使用 ORDER BY 或标准比较操作符的查询中的排序顺序
- 函数 upper、lower 和 initcap
- 模式匹配操作符（LIKE、SIMILAR TO 和 POSIX 风格的正则表达式）；区域影响大小写不敏感匹配和通过字符类正则表达式的字符分类
- to_char 函数家族

因此，在上述场景遇到查询结果集不一致的情况，就可以猜测可能是字符集问题。比如上交所 GBK 编码入库生僻字，like 与等值查询结果集不一致的问题。

排序规则支持

排序规则特性允许指定每一列甚至每一个操作的数据的排序顺序和字符分类行为。这放松了数据库的 LC_COLLATE 和 LC_CTYPE 设置自创建以后就不能更改这一限制。

一个表达式的排序规则可以是"默认"排序规则，它表示数据库的区域设置。一个表达式的排序规则也可能是不确定的。在这种情况下，排序操作和其他需要知道排序规则的操作会失败。

当数据库系统必须要执行一次排序或者字符分类时，它使用输入表达式的排序规则。这会在使用例如 ORDER BY 子句以及函数或操作符调用（如<）时发生。应用于 ORDER BY 子句的排序规则就是排序键的排序规则。应用于函数或操作符调用的排序规则从它们的参数得来，具体如下文所述。除比较操作符之外，在大小写字母之间转换的函数会考虑排序规则，例如 lower、upper 和 initcap。模式匹配操作符和 to_char 及相关函数也会考虑排序规则。

对于一个函数或操作符调用，其排序规则通过检查在执行指定操作时参数的排序规则来获得。如果该函数或操作符调用的结果是一种可排序的数据类型，万一有外围表达式要求函数或操作符表达式的排序规则，在解析时结果的排序规则也会被用作函数或操作符表达式的排序规则。

一个表达式的排序规则派生可以是显式或隐式。该区别会影响多个不同的排序规则出现在同一个表达式中时如何组合它们。当使用一个 COLLATE 子句时，将发生显式排序规则派生。所有其他排序规则派生都是隐式的。当多个排序规则需要被组合时（例如在一个函数调用中），将使用下面的规则：

1. 如果任何一个输入表达式具有一个显式排序规则派生，则在输入表达式之间的所有显式派生的排序规则必须相同，否则将产生一个错误。如果任何一个显式派生的排序规则存在，它就是排序规则组合的结果。
2. 否则，所有输入表达式必须具有相同的隐式排序规则派生或默认排序规则。如果任何一个非默认排序规则存在，它就是排序规则组合的结果。否则，结果是默认排序规则
3. 如果在输入表达式之间存在冲突的非默认隐式排序规则，则组合被认为是具有不确定排序规则。这并非一种错误情况，除非被调用的特定函数要求提供排序规则的知识。如果它确实这样做，运行时将发生一个错误。

字符集

PG 里面的字符集支持各种字符集存储文本，包括单字节字符集，比如 ISO 8859 系列，以及多字节字符集，比如 EUC（扩展 Unix 编码 Extended Unix Code）、UTF-8 和 Mule 内部编码。MPPDB 中目前主要使用的字符集包括 GBK、UTF-8 和 LATIN1。所有被支持的字符集都可以被客户端透明地使用，但少数只能在服务器上使用（即作为一种服务器端编码，GBK 编码在 PG 中只是客户端编码，不是服务端编码，MPPDB 将 GBK 引入到服务端编码，这是很多问题的根源）。默认的字符集是在使用 initdb 初始化 PG 数据库时选择的。在创建一个数据库实例时可以重载字符集，因此可能会有多个数据库实例并且每一个使用不同的字符集。一个重要的限制是每个数据库的字符集必须和数据库 LC_CTYPE（字符分类）和 LC_COLLATE （字符串排序顺序）设置兼容。对于 C 或 POSIX，任何字符集都是允许的，但是对于其他区域只有一种字符集可以正确工作。不过，在 Windows 上 UTF-8 编码可以和任何区域配合使用。

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

数据库系统支持某种编码，主要涉及三个方面：数据库服务器支持，数据访问接口支持以及客户端工具支持。

- 数据库服务器字符编码

  数据库服务器支持某种编码，是指数据库服务器能够从客户端接收、存储以及向客户端提供该种编码的字符（包括标识符、字符型字段值），并能将该种编码的字符转换到其它编码（如 UTF-8 编码转到 GBK 编码）。

  指定数据库服务器编码：创建数据库时指定：CREATE DATABASE … ENCODING … //可以取 ASCII、UTF-8、EUC_CN、……；

  查看数据库编码：show server_encoding。

- 数据库访问接口编码

  数据库访问接口支持某种编码，是指数据库访问接口要做到能对该种编码的字符进行正确读写，不应出现数据丢失、数据失真等情况。以 JDBC 接口为例：

  JDBC 接口一般根据 JVM 的 file.encoding 设置 client_encoding：set client_encoding to file_encoding；

  将 String 转换成 client_encoding 编码的字节流，传给服务器端：原型 String.getBytes(client_encoding) ；

  收到服务器的字节流后，使用 client_encoding 构造 String 对象作为 getString 的返回值给应用程序：原型 String(byte[], …, client_encoding)。

- 客户端编码

  客户端工具支持某种编码，是指客户端工具能够显示从数据库读取该种编码的字符，也能通过本工具将该种编码的字符提交到服务器端。

  指定会话的客户端编码：SET CLIENT_ENCODING TO 'value'；

  查看数据库编码：Show client_encoding。

GDS 导入/导出遇到的字符集问题和解决办法

问题一：0x00 字符无法入库：ERROR: invalid byte sequence for encoding "UTF8": 0x00

原因：PG 本身不允许文本数据中出现 0x00 字符，基线问题，其他数据库不存在该问题。

解决方法：

1. 替换 0x00 字符。
2. Copy、GDS 都有“compatible_illegal_chars”这个选项，把这个开关打开（COPY 命令、GDS 外表可 Alter），会把单字节/多字节的非法字符替换成“（空格）”/“？”。这样可以顺利导入数据，但会更改原数据。

3. 建立 encoding 为 SQL_ASCII 的库，然后 client_encoding 也设置为 SQL_ASCII（COPY 命令中可设置，GDS 外表也可设置），这种情况下可以避免字符集的特殊处理和转换，所有库内相关的排序、比较以及处理全部按照单字节处理。

问题二： GBK 字符无法导入 UTF-8 库

原因：缺少 GBK 到 UTF-8 的转换函数。

解决方法：目前在 r8c10 已经补充了缺少的转换函数，包含 106 个字符，会尽快同步到 r7c10 的发货版本。具体字符，参考如下。

a2e3：€
a6 d9-df：‘ ° 、 ： ； ！ ？
a6 ec ed：〓 ￢
a6 f3：¦
a8 bc bf
ḿ ń
a9 89-8f 90-95
fe 50-5f
fe 60-6f
fe 70-7E
fe 80-8f
fe 90-9f
fe a0

问题三：GBK 转义符\（0x5C）问题。

原因：gbk 本身作为 server 端编码存在很多问题，违背了 PG 的设计原则，即 ascii 码不能作为多字节字符的一部分（多字节字符每个字节的首位必须为 1），不遵循这个原则可能会出现误判的问题。例子如下所示，gbk 多字节字符可能会使用'\'作为第二个字符。

```
/* Else, it's the traditional escaped style */
for (bc = 0, tp = inputText; *tp != '\0'; bc++)
{
if (tp[0] != '\\')
tp++;
```

上述问题属于 PG 基线的代码实现问题。当然可以通过枚举所有可能出现的非法情形来解决，但是实现难度比较大，更为关键的是会增加判断逻辑降低处理效率，这也是社区坚决不允许引入 gbk、SJIS（日文）等字符集的原因所在。

解决方法：尽量不使用 GBK 作为 server 端字符编码，可使用 utf-8 替换。另外，尽量不使用 SQL_ASCII 作为 server 端编码，无论导入的数据是什么字符集，SQL_ASCII 都会按单字节入库，内部逻辑也都会按单字节处理，同样会遇到上述问题。GBK 中包含 0x5C 的两字节字符参考如下。

815c乗 825c俓 835c僜 845c刓 855c匼 865c哱 875c嘰 885c坄 895c塡 8a5c奬 8b5c媆 8c5c孿 8d5c峔 8e5c嶿 8f5c廫 905c怽 915c慭 925c抃 935c揬 945c擻 955c昞 965c朶 975c梊 985c榎 995c橕 9a5c歕 9b5c氿 9c5c淺 9d5c湡 9e5c漒 9f5c焅 a05c燶 a85c╘ a95c─ aa5c猏 ab5c玕 ac5c瑥 ad5c璡 ae5c瞙 af5c疷 b05c癨 b15c皖 b25c瞈 b35c砛 b45c磡 b55c礬 b65c禱 b75c穃 b85c窡 b95c笿 ba5c箌 bb5c篭 bc5c簗 bd5c糪 be5c紽 bf5c縗 c05c繺 c15c羂 c25c耚 c35c肻 c45c膢 c55c臶 c65c艿 c75c茪 c85c莬 c95c萛 ca5c蔦 cb5c蓄 cc5c蘚 cd5c虭 ce5c蜦 cf5c螥 d05c蟎 d15c裓 d25c襖 d35c覶 d45c診 d55c誠 d65c謀 d75c譢 d85c豛 d95c賊 da5c赲 db5c踈 dc5c蹨 dd5c軝 de5c轡 df5c運 e05c郲 e15c醆 e25c鈧 e35c錵 e45c鋅 e55c錦 e65c鎈 e75c鏢 e85c鐪 e95c閈 ea5c闬 eb5c雋 ec5c霛 ed5c韁 ee5c頻 ef5c颸 f05c餦 f15c駭 f25c驚 f35c骪 f45c鬦 f55c鮸 f65c鯸 f75c鰛 f85c鳿 f95c鵟 fa5c鶿 fb5c鸤 fc5c黒 fd5c齄 fe5c㘚

3.3 创建 GDS 外表失败，提示不支持 ROUNDROBIN

问题现象

创建 GDS 外表失败，提示不支持 ROUNDROBIN，报错信息如下所示：

```
ERROR:  For foreign table ROUNDROBIN distribution type is built-in support.
```

原因分析

GDS 外表系统内部默认以 ROUNDROBIN 分布方式创建，不支持在创建外表时显式添加 ROUNDROBIN 分布信息。

处理方法

在创建 GDS 外表时，去除指定的分布信息，即去掉语句中显示指定的“DISTRIBUTE BY ROUNDROBIN”即可。

3.4 通过 CDM 将 MySQL 数据导入 GaussDB(DWS)时出现字段超长，数据同步失败

问题现象

MySQL 5.*x* 版本字段长度 varchar(n) ，用 CDM 同步数据到 GaussDB(DWS)，同样设置长度为 varchar(n) ，但是会出现字段超长，数据同步失败。

原因分析

- MySQL5.0.3 之前 varchar(n)这里的 n 表示字节数。
- MySQL5.0.3 之后 varchar(n)这里的 n 表示字符数，比如 varchar(200)，不管是英文还是中文都可以存放 200 个。
- GaussDB(DWS)的 varchar(n)这里的 n 表示字节数。

根据字符集，字符类型若为 gbk，每个字符占用 2 个字节；字符类型若为 utf8，每个字符最多占用 3 个字节。根据转换规则，同样的字段长度，会导致 GaussDB(DWS）出现字段超长的问题。

处理方法

假设 MySQL 字段为 varchar(n) ，则将 GaussDB(DWS)对应的字段长度设置为 varchar(n*3)即可。

3.5 执行创建 OBS 外表的 SQL 语句时，提示 OBS 访问被拒绝

问题现象

执行创建 OBS 外表的 SQL 语句时，返回 OBS 错误信息，提示访问被拒绝“Access Denied”。

原因分析

- 创建 OBS 外表语句中的访问密钥 AK 和 SK 错误，会出现如下所示的错误信息：

  ```
  ERROR:  Fail to connect OBS in node:cn_5001 with error code: AccessDenied
  ```

- 账户 OBS 权限不足，对 OBS 桶没有读、写权限，会出现如下所示的错误信息：

  ```
  dn_6001_6002: Datanode 'dn_6001_6002' fail to read OBS object bucket:'obs-bucket-name' key:'xxx/xxx/xxx.csv' with OBS error code:AccessDenied message: Access Denied
  ```

  默认情况下，您不具备访问其他账号的 OBS 数据的权限，此外，IAM 用户（相当于子用户）也不具备访问其所属账号的 OBS 数据的权限。

处理方法

- **创建 OBS 外表语句中的访问密钥 AK 和 SK 错误**

  请获取正确的访问密钥 AK 和 SK，写入创建 OBS 外表的 SQL 语句中。获取访问密钥的步骤如下：

  a. 登录 GaussDB(DWS) 管理控制台。

  b. 单击右上角的用户名并选择菜单“我的凭证”。

  c. 进入“我的凭证”后，在左侧导航树单击“访问密钥”。

  在访问密钥页面，可以查看已有的访问密钥 ID（即 AK）。

  d. 如果要同时获取 AK 和 SK，单击“新增访问密钥”创建并下载访问密钥。

- **账户 OBS 权限不足，对 OBS 桶没有读、写权限**

  您必须给指定的用户授予所需的 OBS 访问权限：

  - 通过 OBS 外表导入数据到 GaussDB(DWS) 时，执行导入操作的用户必须具备数据源文件所在的 OBS 桶和对象的**读取**权限。
  - 通过 OBS 外表导出数据时，执行导出操作的用户必须具备数据导出路径所在的 OBS 桶和对象的**读取和写入**权限。

  有关配置 OBS 权限的具体操作，请参见《对象存储服务用户指南》中的“控制台指南 > 权限控制”章节。

3.6 GDS 导入失败后，磁盘占用空间增大

问题背景与现象

使用 GDS 导入数据失败，触发作业重跑，重新开始数据导入，完成导入作业，查看磁盘空间，发现磁盘占用空间比导入数据量大很多。

原因分析

在导入数据失败后，占用的磁盘空间没有释放。

解决办法

步骤 1 检测 GDS 导入作业的日志，查看是否有执行失败的现象。

步骤 2 对表或者分区执行清理操作。

```
vacuum [full] table_name;
```

----结束

3.7 GDS 导入数据时，脚本执行报错：out of memory

问题现象

在使用 GDS 导入数据时，脚本执行报错：out of memory.

原因分析

1. 使用 copy 命令或者导入数据时，源数据单行数据的大小超过 1GB。
2. 由于源文件中的格式符不成对出现，比如引号，文件格式异常导致系统识别的单行数据过大超过 1GB。

处理方法

1. 确保源文件中的引号是成对的。
2. 检查创建外表时命令中参数的取值、格式设置是否合理。
3. 检查源文件单行数据是否超过 1GB，参考报错行号进行检查，可根据实际情况手动调整或删除该行数据。

3.8 使用 GDS 传输数据的过程中，报错"connection failure error."

问题现象

在使用 GDS 传输数据的过程中，报错"connection failure error."

原因分析

1. GDS 进程崩溃。执行命令检查 GDS 进程是否崩溃：

   ```
   ps ux|grep gds
   ```

   若返回结果如下，则说明 GDS 进程启动成功：

2. GDS 启动参数-H 配置不正确。

   -H *address_string*：允许哪些主机连接和使用 GDS 服务。参数需为 CIDR 格式。此参数配置的目的是允许 GaussDB(DWS)集群可以访问 GDS 服务进行数据导入，请保证所配置的网段包含 GaussDB(DWS)集群各主机。

处理方法

1. 重新启动 GDS。具体步骤参见《数据仓库服务开发指南》中的“数据迁移>导入数据>使用 GDS 从远端服务器导入数据>安装配置和启动 GDS”章节。
2. 修改 GDS 启动命令中的 - H 参数，可以尝试修改成 - H 参数为 0/0，验证是否是该原因。若修改参数为 0/0 后命令可以执行，说明原本的参数设置不合理，所配置的网段未包含 GaussDB(DWS)集群各主机，需要修改。

3.9 使用 DLF 服务创建 GaussDB(DWS) 外表时不支持中文，如何处理

问题现象

使用 DLF 服务创建 GaussDB(DWS) 的 OBS 外表，并且在创建外表语句中指定 OBS 文件编码格式是 UTF-8，但是导入数据时报错，如何处理？

原因分析

存储在 OBS 中的源文件含有非 UTF-8 的数据。

处理方法

排查报错的源文件，检查是否含有非 UTF-8 的数据，例如中文字符。如果源文件中含有非 UTF-8 的数据，请先将源文件转换成 UTF-8 的格式，并重新上传到 OBS，然后再执行导入数据的操作。

4 帐户、密码、权限

4.1 帐号被锁住了，如何解锁？

问题现象

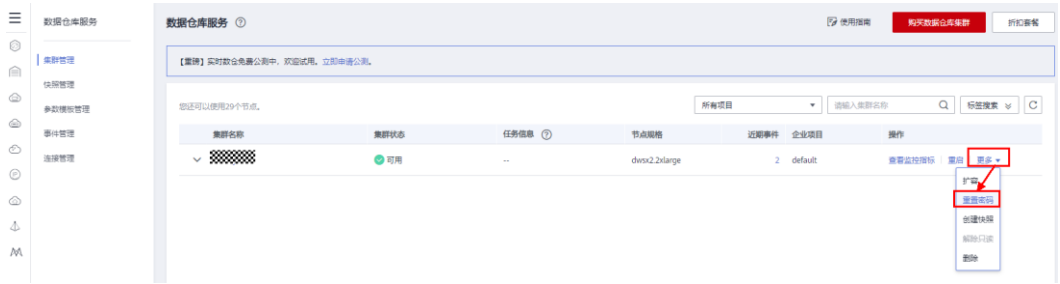
帐号被锁住了，如何解锁？连接集群时报错：The account has been locked。

原因分析

在连接集群中的数据库时，如果连续输错密码的次数过多，错误次数达到上限时（默认为 10 次），会导致帐号被锁。

管理员用户（默认为 dbadmin）解锁方法

您可以登录 GaussDB(DWS) 管理控制台重置管理员密码，重置密码后帐号即可自动解锁。在 GaussDB(DWS) 管理控制台，进入“集群管理”页面，找到所需要的集群，然后单击“更多 > 重置密码”。



数据库普通用户解锁方法

使用管理员用户（默认为 dbadmin）连接数据库，然后执行以下命令进行解锁，其中 user_name 请替换为需要解锁的用户名：

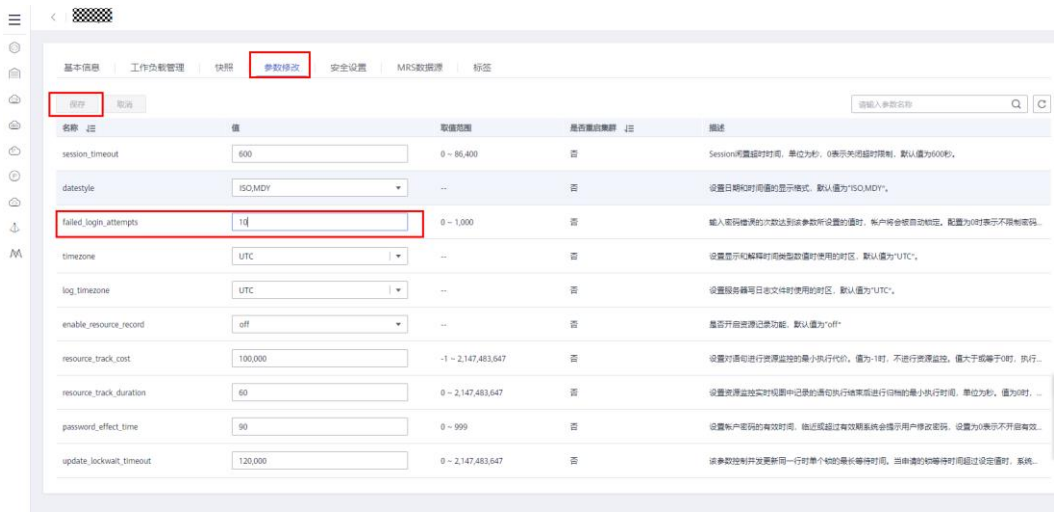
```
gsql -d gaussdb -p 8000 -U dbadmin -W 密码 -h 集群 IP
ALTER USER user_name ACCOUNT UNLOCK;
```

设置尝试登录失败次数

输错密码的次数上限可以在集群的“参数修改”页面通过参数 failed_login_attempts 进行设置，当 failed_login_attempts 配置为 0 时表示不限制密码输入错误的次数，一般不推荐设置为 0。

您可以通过如下步骤修改参数：

1. 登录 GaussDB(DWS) 管理控制台。
2. 在左侧导航树，单击“集群管理”。
3. 在集群列表中找到所需要的集群，然后单击集群名称。
4. 进入集群的“参数修改”页面，找到“failed_login_attempts”参数，修改其参数值，然后单击“保存”，确认无误后再单击“保存”。



4.2 重置密码后再次登录仍提示用户被锁

问题现象

某客户连接集群时，提示用户被锁，重置用户密码后，再次登录，仍提示用户被锁。

```
FATAL: The account has been locked.
```

原因分析

DWS 默认情况下，在连续输入错误密码 10 次后会锁定该用户（这个允许输入的错误次数由参数 failed_login_attempts 控制，可以参见设置尝试登录失败次数登录 DWS 管理控制台进行修改）。

当客户重置密码后，重试还是被锁，说明在重置后，有其他人或者应用又用错误的密码再次连了 10 次，再次导致了用户被锁。

处理方法

步骤 1 使用系统管理员 dbadmin 用户连接数据库，执行以下 SQL 语句查看系统时间。

```
SELECT NOW();
```

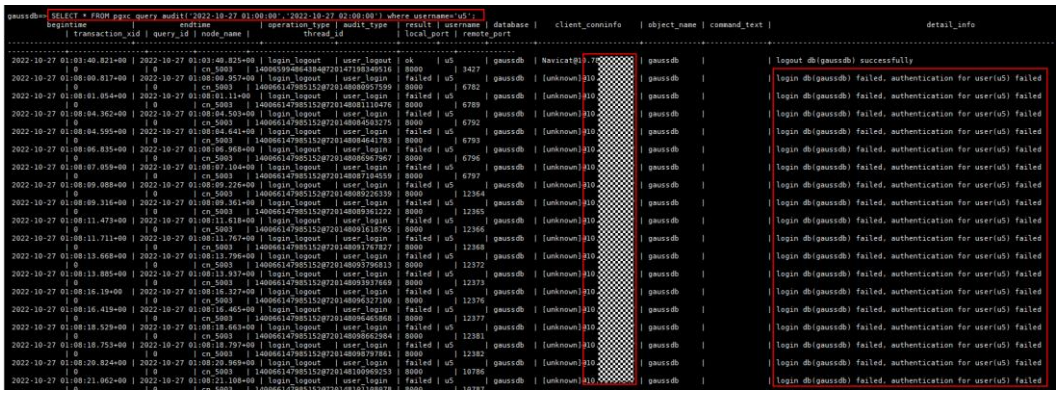
```
gaussdb=> SELECT NOW();
              now
-------------------------------
 2022-10-27 01:14:09.24637+00
(1 row)
```

从回显结果可以看出，DWS 默认显示的系统时间是 UTC 时间，即北京时间-8H。

步骤 2 使用系统管理员 dbadmin 用户连接数据库，执行以下 SQL 语句查询客户端连接情况。其中，

- username 请替换为实际被锁的用户名。
- 时间段，请根据实际时间修改。例如要查北京时间当天上午 9 点到 10 点这个时间段的连接情况，需由北京时间转为 UTC 时间，即-8H，则 UTC 时间为当天上午 1 点到 2 点的时间段。

```
SELECT * FROM pgxc_query_audit('2022-10-27 01:00:00','2022-10-27 02:00:00') where username='username';
```



从以上回显看出，IP 地址为 x.x.x.x 的客户端一直使用错误的密码进行连接。

步骤 3 可选择以下处理办法：

- 如果确认这个 IP 地址是某个作业，可以先停止该作业连接，再通过系统管理员 dbadmin 连接数据库后，使用以下 SQL 语句解锁该用户，再调整该作业设置，使用正确的密码进行连接。

  ```
  ALTER USER username ACCOUNT UNLOCK;
  ```

- 如果不确认这个 IP 地址是哪个作业，可以先参见设置尝试登录失败次数将参数 failed_login_attempts 修改为 0，然后使用以下 SQL 语句再次重置一个新密码，这样正在连接的其他作业即使再次输错 10 次密码也不会被锁定。

  ```
  ALTER USER username IDENTIFIED BY 'password';
  ```

须知

这个只是临时解决方案，为保证数据库安全，failed_login_attempts 不建议设置为 0。建议后续定位到这个作业后，还是尽快将作业的连接设置调整为正确密码后，将 failed_login_attempts 设置为 10。

----结束

4.3 将 Schema 中的表的查询权限赋给其他用户，赋权后仍无法查询 Schema 中的表

问题现象

有权限的用户使用“grant select on all tables in schema *schema_name* to *B*”命令给其他用户赋予 schema 下表的权限后，B 用户仍然无法访问该 schema 下的表。

原因分析

将模式中的表或者视图对象授权给其他用户时，需要将表或视图所属的模式的 USAGE 权限同时授予该用户，若没有该权限，则只能看到这些对象的名字，并不能实际进行对象访问。

如果要将该 schema 下未来创建的表的权限也赋予 B 用户，则需要用到 ALTER DEFAULT PRIVILEGES 更改默认权限。

处理方法

请使用具有 schema 权限的用户登录数据库，执行以下命令将 schema 中的表权限赋给指定的用户：

```
GRANT USAGE ON SCHEMA schema_name TO B;
GRANT SELECT ON ALL TABLES IN SCHEMA schema_name TO B;
```

执行以下命令，将 schema 中未来新建的表的权限也赋予指定的用户：

```
ALTER DEFAULT PRIVILEGES IN SCHEMA schema_name GRANT SELECT ON TABLES TO B;
```

上述 SQL 语句中的“GRANT SELECT”表示赋予的是表的查询权限，如果您要给其他用户赋予表的其他权限，请参考 GRANT 语法说明。

4.4 某张表执行过 grant select on table t1 to public，如何针对某用户回收权限

问题现象

假设当前有两个普通用户 use1 和 use2，当前数据库下有两张表 t1 和 t2， 执行：

```
grant select on table t1 to public;
```

用户 use1 和 use2 对该表有访问权限，并且新建用户 use3 后，新用户 use3 对该表也有访问权限，且执行 revoke select on table t1 from use3 无效。

```
test=# revoke select on table t1 from use3;
REVOKE
```

```
test=# \c - use3
Password for user use3:
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "test" as user "use3".
test=> select * from t1;
 a
---
(0 rows)

test=> select relname, relacl from pg_class where relname = 't1';
 relname |                relacl
---------+---------------------------------------------
 t1      | {liukunpeng=arwdDxt/liukunpeng,=r/liukunpeng}
(1 row)
```

原因分析

这是因为之前执行过 grant select on table t1 to public 这条 sql，该 sql 中关键字 public 表示该权限要赋予给所有角色，包括以后创建的角色，所以新用户 use3 对该表也有访问权限。public 可以看做是一个隐含定义好的组，它总是包含所有角色。

因此，执行完 revoke select on table t1 from use3 之后，虽然 use3 用户没有了该表的访问权限（通过该表的 relacl 字段也可以看到），但是他仍然有 public 的权限，所以仍能访问该表。

处理方法

需要 revoke 回 public 的权限，然后对 use3 用户的权限单独管控。但是由于 revoke 回 public 的权限后可能导致原来能访问该表的用户（use1 和 use2）无法访问该表，影响现网业务，因此需要先对这些用户执行 grant 赋予相应权限，然后 revoke 回 public 的权限。

```
test=# --查看所有用户
test=# select * from pg_user where usesysid >= 16384;
 usename | usesysid | usecreatedb | usesuper | usecatupd | userepl |  passwd  |
valbegin | valuntil |   respool    | parent | spacelimit | useconfig | nodegroup |
tempspacelimit | spillspacelimit
---------+----------+-------------+----------+-----------+---------+----------+----
------+----------+--------------+--------+------------+-----------+-----------+----
------------+-----------------
 jack    |    16408 | f           | f        | f         | f       | ******** |
| default_pool |     0 |           |          |           |              |
 tom     |    16412 | f           | f        | f         | f       | ******** |
| default_pool |     0 |           |          |           |              |
 use1    |    16437 | f           | f        | f         | f       | ******** |
| default_pool |     0 |           |          |           |              |
 use2    |    16441 | f           | f        | f         | f       | ******** |
| default_pool |     0 |           |          |           |              |
 use3    |    16448 | f           | f        | f         | f       | ******** |
| default pool |     0 |           |          |           |              |
(5 rows)
test=# --对原用户执行 grant
test=# grant select on table t1 to jack,tom,use1,use2;
GRANT
```

```
test=# --回收 public 的权限
test=# revoke select on table t1 from public;
test=# \c - use3
Password for user use3:
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "test" as user "use3".
test=> select * from t1;
ERROR:  permission denied for relation t1
```

4.5 普通用户执行创建或删除 GDS/OBS 外表语句时报错，提示没有权限或权限不足

问题现象

创建 GDS 或 OBS 外表语句，超级用户可以执行成功，普通用户执行时报错：ERROR: permission denied to create foreign table in security mode。

原因分析

该错误信息表明普通用户没有创建外表的权限。

处理方法

可使用 ALTER USER 或者 ALTER ROLE 语法指定 USEFT 参数赋予角色或用户使用外表的权限。

参数 USEFT | NOUSEFT 决定一个新角色或用户是否能操作外表，包括：新建外表、删除外表、修改外表、读写外表。

- 指定 USEFT 表示角色或用户可操作外表。
- 缺省为 NOUSEFT。表示新角色或用户没有操作外表的权限。

请使用数据库管理员用户给普通用户或角色赋予使用外表的权限，示例如下：

```
ALTER USER user_name USEFT;
```

修改用户或角色权限等信息的详细内容请参见《数据仓库服务 SQL 语法参考》中的 ALTER USER 或者 ALTER ROLE。

4.6 赋予用户 schema 的 all 权限后建表仍然报错：ERROR: current user does not have privilege to role tom

问题现象

有两个用户 tom 和 jerry，jerry 想要在 tom 的同名 schema 下创建表，于是 tom 把该 schema 的 all 权限赋给 jerry，但是创建表时仍然报错：

```
postgres=# grant all on schema tom to jerry;
GRANT
postgres=# \c - jerry
Password for user jerry:
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "postgres" as user "jerry".
postgres=>
postgres=> create table tom.t(a int);
ERROR:  current user does not have privilege to role tom
```

原因分析

根据报错内容，jerry 需要角色 tom 的权限。

处理方法

把角色 tom 的权限赋予 jerry 后，建表执行成功。

```
postgres=# grant tom to jerry;
GRANT ROLE
postgres=# \c - jerry
Password for user jerry:
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "postgres" as user "jerry".
postgres=>
postgres=> create table tom.t(a int);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution
column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution
column.
CREATE TABLE
```

4.7 执行语句过程中报错：无权限操作

问题现象

执行语句后提示

```
ERROR:  permission denied for xxx
```

原因分析

该用户无对应的权限。

处理方法

步骤 1 使用 GRANT 语法对表/schema 进行赋权，示例： 假设当前有两个用户 tom 和 jerry，如果想要用户 jerry 能够对当前 tom 创建的所有表以及将来创建的表都有查询权限，如何处理：

- 将用户 tom 下的同名 schema 权限赋给 jerry

```
grant usage on schema tom to jerry;
```

- 将用户 tom 已经创建的表的 select 权限赋给 jerry

```
grant select on all tables in schema tom to jerry;
```

- 将用户 tom 未来在同名 schema 下创建的表的 select 权限赋给 jerry

```
alter default privileges for user tom in schema tom grant select on tables to jerry;
```

----结束

4.8 create extension 命令执行失败，提示没有权限

问题现象

create extension 命令执行失败，提示没有权限，报如下所示的错误：

```
ERROR: permission denied to create extension "uuid-ossp" Hint: Must be superuser to create this extension.
```

原因分析

GaussDB(DWS) 不支持 postgres 社区的扩展功能。

5 数据库使用

5.1 插入或更新数据时报错，提示分布键不能被更新

问题现象

往数据库插入或更新数据时报错，提示分布键不能被更新，错误信息如下所示：

```
ERROR: Distributed key column can't be updated in current version
```

原因分析

GaussDB(DWS) 分布键不允许被更新。

处理方法

方法一：分布键目前暂不支持更新，直接跳过该报错。

方法二：将分布列修改为一个不会更新的列（8.1.0 版本后，支持调整分布列，以下为示例）。

步骤 1 查询当前表定义，回显发现该表分布列为 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 尝试执行更新分布列中的数据提示错误信息。

```
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 执行 SQL 语句时，提示 Connection reset by peer

问题现象

执行 SQL 语句时，提示"Connection reset by peer"。

```
ERROR:  Failed to read response from Datanodes Detail: Connection reset by peer
```

原因分析

在网络压力大的情况下会因为 socket 通信问题，出现断连现象。

解决办法

- 通过流控机制防止网络压力过大，需要设置以下 GUC 参数控制网络流量峰值。

  ```
  comm_quota_size = 400
  comm_usable_memory = 100
  ```

- 数据库在识别此类错误后，会自动进行重试，重试次数使用 GUC 变量 max_query_retry_times 来控制。

说明

目前仅支持单条 SQL 语句的重试，暂不支持事务块中出错 SQL 重试。

5.3 VARCHAR(n)存储中文字符，提示 value too long for type character varying?

问题现象

VARCHAR(18)的字段，存储 8 个中文字符长度不够，报如下所示的错误：

```
org.postgresql.util.PSQLException: ERROR: value too long for type character
varying(18)
```

原因分析

以 UTF-8 编码为例，一个中文占 3~4 个字节，即 8 个中文占 24~32 字节，超出 VARCHAR(18)的最大 18 字节限制。

当表中某一字段包含有中文字符时，可使用 char_length 或 length 函数来查询字段字符长度。

```
 SELECT length('数据库 database');
 length
--------
     17
(1 row)
```

处理方法

varchar(n)为变长类型，n 代表可存储的最大字节数。中文字符通常占用 3~4 个字节。

请根据实际的中文字符长度，增加该字段的字段长度。示例中某字段要存储 8 个中文字符，则需要设置 n 至少为 32，即 VARCHAR(32)。

5.4 SQL 语句中字段名大小写敏感问题

问题现象

某表 table01 中存在以大小写字母组合的名称为“ColumnA”的字段，使用 SELECT 语句查询该字段时，提示字段不存在，报错：column "columna" does not exist.

```
select ColumnA from table01 limit 100;
ERROR:  column "columna" does not exist
LINE 1: select columna from TABLE_01;
               ^
CONTEXT:  referenced column: columna
```

原因分析

在 GaussDB(DWS) 中，SQL 语句中的表字段等名称带双引号时大小写敏感；不带双引号时大小写不敏感，按全小写处理。

处理方法

- 在大小写不敏感的场景下，将字段名称的双引号去掉。
- 在大小写敏感的场景下，执行 SQL 语句时需要给字段名称加上双引号。

  示例：

  table01 中，使用 SELECT 语句查询 ColumnA 时，ColumnA 字段带上双引号，查询成功：

  ```
  select "ColumnA" from table01 limit 100;
  ```

5.5 删除表时报错：cannot drop table test because other objects depend on it

问题现象

删除表时出现如下报错：cannot drop table test because other objects depend on it

```
tddb=# create table t1  (a int, b serial) distribute by hash(a);
NOTICE:  CREATE TABLE will create implicit sequence "t1_b_seq" for serial column "t1.b"
CREATE TABLE
tddb=# create table t2  (a int, b int default nextval('t1_b_seq')) distribute by hash(a);
CREATE TABLE
tddb=# drop table t1;
ERROR:  cannot drop table t1 because other objects depend on it
DETAIL:  default for table t2 column b depends on sequence t1_b_seq
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
```

原因分析

创建 t1 表后，隐式创建了 sequence，然后创建 t2 表的时候，引用了该 sequence，然后删除 t1 表的时候，由于会级联删除 sequence，但是该 sequence 被其他对象依赖，因此导致该报错。

处理方法

如不需保留 t2，可通过 DROP CASCADE 的方式级联删除，如需保留该表和该 sequence，可以通过以下命令进行删除：

```
tddb=# drop table t1;
ERROR:  cannot drop table t1 because other objects depend on it
DETAIL:  default for table t2 column b depends on sequence t1_b_seq
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
tddb=#
tddb=# alter sequence t1_b_seq owned by none;
ALTER SEQUENCE
tddb=#
tddb=# drop table t1;
DROP TABLE
tddb=#
```

5.6 多个表同时进行 merge into update 时，执行失败

问题现象

多个表同时进行 merge into update 时，执行失败。

原因分析

查看日志，发现有如下错误日志：

```
dn_6007_6008 YY003 79375943437085786 [BACKEND] DETAIL:  blocked by hold lock thread
0, statement <pending twophase transaction>, hold lockmode (null).
```

这是由于分布式锁导致的，两个 DN 节点都锁住了自己的数据块，然后又在等待对方的数据块，所以导致锁超时。

这种行为是两阶段锁的特性，分布式情况下都会面临这样的问题。

处理方法

建议对单表执行 merge，将并发操作改为串行。

5.7 session_timeout 设置导致 JDBC 业务报错

问题现象

通过 JDBC 连接集群执行 COPY 导入时报错：

```
org.postgresql.util.PSQLException: Database connection failed when starting copy at
org.postgresql.core.v3.QueryExecutorImpl.startCopy(QueryExecutorImpl.java:804) at
org.postgresql.copy.CopyManager.copyIn(CopyManager.java:52) at
org.postgresql.copy.CopyManager.copyIn(CopyManager.java:161) at
org.postgresql.copy.CopyManager.copyIn(CopyManager.java:146) at
copy.main(copy.java:95) Caused by: java.io.EOFExceptionat
org.postgresql.core.PGStream.ReceiveChar(PGStream.java:284) at
org.postgresql.core.v3.QueryExecutorImpl.processCopyResults(QueryExecutorImpl.java:
1008) at
org.postgresql.core.v3.QueryExecutorImpl.startCopy(QueryExecutorImpl.java:802) ...
4 more
```

原因分析

数据库默认设置 session_timeout=10min，即连接空闲超过 10 分钟，自动断开连接，导致连接失效。

处理方法

可登录 GaussDB(DWS) 管理控制台，设置 session_timeout 为 0 或预期时长，使会话保持长时间连接。

1. 登录 GaussDB(DWS) 管理控制台。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。
2. 单击“参数修改”页签，修改 session_timeout 参数值，然后单击“保存”。

说明

COPY 导入执行完成后，建议继续设置 session_timeout=10min，因为如果有客户端长时间连接数据库，但对数据库不进行任何操作，该连接将一直占用一个线程，如果这样的客户端连接很多，就会出现大量的线程都被空闲的连接占用，从而导致数据库连接满或者资源浪费。

5.8 DROP TABLE 失败

问题现象

DROP TABLE 失败的两种现象：

- 在使用"SELECT * FROM DBA_TABLES;"语句（或者 gsql 客户端也可以使用\dt+ 命令）查看数据库中无表 table_name；CREATE TABLE table_name 时报 table_name 已经存在的错误，使用 DROP TABLE table_name 失败，报不存在该表的错误，导致无法再次创建 table_name 表。
- 在使用"SELECT * FROM DBA_TABLES;"语句（或者 gsql 客户端也可以使用\dt+ 命令）查看数据库中有表 table_name；使用 DROP TABLE table_name 失败，报不存在该表的错误，导致无法再次创建 table_name 表。

原因分析

导致该错误的原因有的节点上有该张表，有的节点上无该张表。

解决办法

当遇到上面两种现象时，使用 DROP TABLE table_name 失败，请再次使用 DROP TABLE if exists table_name 命令，使得 DROP 表可以成功。

5.9 使用 string_agg 函数后执行结果不稳定

问题现象

某客户反馈，SQL 语句的执行结果不正确，怀疑数据库出现了结果不一致的问题。

原因分析

客户的 SQL 语句中使用了 string_agg 函数，语句逻辑如下：

```
postgres=# select * from employee;
 empno | ename  |   job    | mgr  |      hiredate       |  sal  | comm | deptno
-------+--------+----------+------+---------------------+-------+------+--------
  7499 | ALLEN  | SALEMAN | 7698 | 2014-11-12 00:00:00 | 16000 |  300 |     30
  7566 | JONES  | MANAGER | 7839 | 2015-12-12 00:00:00 | 32000 |    0 |     20
  7654 | MARTIN | SALEMAN | 7698 | 2016-09-12 00:00:00 | 12000 | 1400 |     30
(3 rows)
```

执行如下 SQL 语句：

```
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;
```

在循环多次执行这个语句的时候，发现结果不稳定，输出结果有时候是 t1，有时候是 t2，因此怀疑是数据库有问题，结果解不正确。

String_agg 函数的作用是将组内的数据合并成一行，但是如果用户用法是 string_agg(ename, ',')这种情况下，结果集就是不稳定的，因为没有指定组合的顺序。

例如，上述 SQL 语句中的输出结果可以是以下任意一种，且都是合理的。

30 | ALLEN,MARTIN

30 |MARTIN,ALLEN

因此有可能出现 t1 这个 subquery 中的结果和 t2 这个 subquery 中的结果对于 deptno=30 的时候的输出结果是不一样的。

处理方法

String_agg 中增加 order by，语句修改为如下格式保证 ename 字段是按照相同的顺序来拼接的，从而满足查询结果是稳定的。

```
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;
```

5.10 查询表大小时报错：could not open relation with OID xxxx

问题现象

在执行 pg_table_size 查询表大小时，出现报错：could not open relation with OID xxxx

原因分析

通过执行 pg_table_size 这个查询接口，对于不存在的表会返回 NULL 或者报错。

处理方法

1. 通过 Function 的 exception 方式屏蔽该报错，将大小统一到一个值，对于不存在的表，可以用大小为-1 来表示，函数如下

```
CREATE OR REPLACE FUNCTION public.pg_t_size(tab_oid OID,OUT retrun_code text)
RETURNS text
LANGUAGE plpgsql
AS $$ DECLARE
v_sql text;
ts text;
BEGIN
V_SQL:='select pg_size_pretty(pg_table_size('||tab_oid||'))';
EXECUTE IMMEDIATE V_SQL into ts;
IF ts IS NULL
THEN RETRUN_CODE:=-1;
ELSE
```

```
return ts;
END IF;
EXCEPTION
WHEN OTHERS THEN
RETRUN_CODE:=-1;
END$$;
```

2. 执行如下命令查询结果：

```
call public.pg_t_size('1','');
 retrun_code
-------------
 -1
(1 row)

select oid from pg_class limit 2;
 oid
------
 2662
 2659
(2 rows)

call public.pg_t_size('2662','');
 retrun_code
-------------
 120 KB
(1 row)
```

5.11 drop table if exists 语法误区

问题现象

使用 drop table if exists 执行 drop 表的语句存在语法误区，理解不当将会删除错误。

原因分析

drop table if exists 语法可以简单这样理解：

1. 判断当前 cn 是否存在该 table；
2. 如果存在，就给其他 cn 和 dn 下发 drop 命令；如果不存在，则跳过。

而不是：

1. 将 drop table if exists 下发给所有 cn 和 dn；
2. 各个 cn 和 dn 判断自己有没有该 table，如果有的话执行 drop。

处理方法

如果出现某些表定义在部分 cn/dn 存在，部分 cn/dn 不存在时，是不可以直接用 drop table if exists 修复的。

这种情况的通用修复方式为：

1. create table if not exists 将所有 cn 和 dn 的表定义补齐。
2. 如果表没有用，就在 cn 上 drop table 删掉所有 cn 和 dn 的表定义，如果表还有用，继续使用即可。

5.12 不同用户查询同表显示数据不同

问题现象

2 个用户登录相同数据库 human_resource，分别执行的查询语句如下：select count(*) from areas，查询同一张表 areas 时，查询结果却不一致。

原因分析

请先判断同名的表是否确实是同一张表。在关系型数据库中，确定一张表通常需要 3 个因素：database，schema，table。从问题现象描述看，database，table 已经确定，分别是 human_resource、areas。接着，需要检查 schema。使用 dbadmin，user01 分别登录发现，search_path 依次是 public 和"$user"。dbadmin 作为集群管理员，默认不会创建 dbadmin 同名的 schema，即不指定 schema 的情况下所有表都会建在 public 下。而对于普通用户如 user01，则会在创建用户时，默认创建同名的 schema，即不指定 schema 时表都会创建在 user01 的 schema 下。最终确定该案例发生时，确实因为 2 个用户之间交错对表进行操作，导致了同名不同表的情况。

解决办法

在操作表时加上 schema 引用，格式：schema.table。

5.13 修改索引只调用索引名提示索引不存在

问题现象

创建分区表索引 HR_staffS_p1_index1，不指定索引分区的名字。

```
CREATE INDEX HR_staffS_p1_index1 ON HR.staffS_p1 (staff_ID) LOCAL;
```

创建分区索引 HR_staffS_p1_index2，并指定索引分区的名字。

```
CREATE INDEX HR_staffS_p1_index2 ON HR.staffS_p1 (staff_ID) LOCAL
(
   PARTITION staff_ID1_index,
   PARTITION staff_ID2_index TABLESPACE example3,
   PARTITION staff_ID3_index TABLESPACE example4
) TABLESPACE example;
```

修改索引分区 staff_ID1_index 的表空间为 example1：

调用“ALTER INDEX HR_staffS_p1_index2 MOVE PARTITION staff_ID2_index TABLESPACE example1;”提示索引不存在。

原因分析

重新创建索引 CREATE INDEX HR_staffS_p1_index2 MOVE PARTITION staff_ID2_index TABLESPACE example1，提示索引已存在，然后执行以下 SQL 命令或者 gsql 客户端元命令\d+ HR.staffS_p1 查询索引时发现索引已存在。

```
SELECT * FROM DBA_INDEXES WHERE index_name = HR.staffS_p1 ;
```

推测是当前模式是 public 模式，而不是 hr 模式，导致检索不到该索引。

使用“ALTER INDEX hr.HR_staffS_p1_index2 MOVE PARTITION staff_ID2_index TABLESPACE example1;”验证推测，发现调用成功。

接着调用 ALTER SESSION SET CURRENT_SCHEMA TO hr;再次调用“ALTER INDEX HR_staffS_p1_index2 MOVE PARTITION staff_ID2_index TABLESPACE example1;”发现设置成功。

解决办法

在操作表、索引、视图时加上 schema 引用，格式：schema.table。

5.14 在执行 SQL 语句时报错，这个 schema 已存在

问题现象

执行 CREATE SCHEMA 语句时，报错这个 schema 已存在

```
ERROR: schema "schema" already exists
```

原因分析

在 SQL 语句中，字段名称是区分大小写的，默认为小写字段名。

解决办法

在大小写有严格要求的场景下，需要给字段名称加上双引号，如下图重新执行 SQL 语句，则创建成功。

```
1  CREATE SCHEMA "SCHEMA";
2
```

```
[INFO] 操作影响的记录行数：0
[INFO] 执行时间： 8 sec
[INFO] 执行成功...
```

5.15 删除数据库失败，提示有 session 正在连接

问题现象

删除数据库失败，提示有 session 正在连接。

原因分析

可能当前仍有 session 正在连接数据库，或者可能有 session 在不停地连接该数据库，故删除数据库失败。需要查看数据库中的 session，检查是否仍有 session 在连接，如果有，排查连接数据库的机器，停止连接后再删除数据库。

处理方法

步骤 1 使用 SQL 客户端工具连接数据库。

步骤 2 执行如下命令查看当前会话。

```
select * from pg_stat_activity;
```

查询结果中的关键字段，说明如下：

- datname：用户会话所连接的数据库名称。
- usename：连接数据库的用户名。
- client_addr：连接数据库的客户端主机的 IP 地址。

在查询结果中，找出待删除的数据库名称及对应的客户端主机 IP 地址。

步骤 3 请根据客户端主机的 IP 地址排查连接数据库的机器及应用，并停止相关的连接。

```
CLEAN CONNECTION TO ALL FOR DATABASE xxx;
```

步骤 4 重新执行删除数据库的命令。

```
DROP DATABASE [ IF EXISTS ] database_name;
```

----结束

5.16 在 Java 中，读取 character 类型的表字段时返回类型为什么是 byte？

问题现象

新建一个数据库表，某个表字段使用 character 类型，在 Java 中读取 character 类型的字段时返回类型为什么是 byte？

例如，创建如下所示的表：

```
CREATE TABLE IF NOT EXISTS table01(
  msg_id character(36),
```

```
  msg character varying(50)
);
```

在 Java 中，读取 character 类型的字段代码如下：

```
ColumnMetaInfo(msg_id,1,Byte,true,false,1,true);
```

原因分析

CHARACTER(n)是定长字符串类型，当实际字符串长度不够时数据库会用空格补全，Java 用 byte 类型接收。CHARACTER VARYING(n)是变长字符串类型，Java 使用 String 类型接收。

5.17 执行表分区操作时，提示 ERROR:start value of partition "XX" NOT EQUAL up-boundary of last partition.

问题现象

进行 ALTER TABLE PARTITION 时，收到如下报错：

```
ERROR:start value of partition "XX" NOT EQUAL up-boundary of last partition.
```

原因分析

在同一条 ALTER TABLE PARTITION 语句中，既存在 DROP PARTITION 又存在 ADD PARTITION 时，无论它们的相对顺序是什么，GaussDB(DWS)总会先执行 DROP PARTITION 再执行 ADD PARTITION。执行完 DROP PARTITION 删除末尾分区后，再执行 ADD PARTITION 操作会出现分区间隙，导致报错。

解决办法

为防止出现分区间隙，需要将 ADD PARTITION 的 START 值前移。

示例：对于分区表 partitiontest

```
CREATE TABLE partitiontest
(
 c_int integer,
 c_time TIMESTAMP WITHOUT TIME ZONE
)
PARTITION BY range (c_int)
(
 partition p1 start(100)end(108),
 partition p2 start(108)end(120)
);
```

使用如下两种语句会发生报错：

```
ALTER TABLE partitiontest ADD PARTITION p3 start(120)end(130), DROP PARTITION p2;
ERROR:  start value of partition "p3" NOT EQUAL up-boundary of last partition.
ALTER TABLE partitiontest DROP PARTITION p2,ADD PARTITION p3 start(120)end(130) ;
ERROR:  start value of partition "p3" NOT EQUAL up-boundary of last partition.
```

可以修改语句为：

```
ALTER TABLE partitiontest ADD PARTITION p3 start(108)end(130), DROP PARTITION p2;
ALTER TABLE partitiontest DROP PARTITION p2,ADD PARTITION p3 start(108)end(130) ;
```

5.18 重建索引失败

问题现象

当 Desc 表的索引出现损坏时，无法进行一系列操作，报错信息可能为：

```
index \"%s\" contains corrupted page at
block %u" ,RelationGetRelationName(rel),BufferGetBlockNumber(buf),
please reindex it.
```

原因分析

在实际操作中，索引会由于软件问题或者硬件问题引起崩溃。例如，当索引分裂完而磁盘空间不足、出现页面损坏等问题时，会导致索引损坏。

解决办法

若此表是，则说明 desc 表的索引表损坏，通过 desc 表的索引表表名，找到对应主表的 oid 和表，执行 REINDEX INTERNAL TABLE name 语句重建 cudesc 表的索引。

5.19 视图查询时执行失败

问题现象

在连接集群数据库之后，使用某个视图执行查询，可能会操作失败，提示错误信息如下：

```
[GAUSS-01850] : object with oid 16420 is not a partition object
```

原因分析

此视图是建立分区表的某个分区上的，查询此视图时需要访问到对应的分区，从而必须首先查询对应的分区是否存在。当对应的分区已经被删除后，无法访问到此分区，从而导致视图访问也失败，报出如上类似的信息。

解决办法

步骤 1 确定是针对视图对象进行的 SQL 操作，并获得视图的名字。

直接检查 SQL 语句的 FROM 对象，确定是否为视图。若是，则直接获得该视图的名字。

步骤 2 使用已获得的视图的名字和 schema，删除该视图。

步骤 3 重新执行 SQL 语句。对于查询操作，由于对应的分区已经被删除，视图的存在没有意义。

----结束

5.20 全局 SQL 查询

通过 pgxc_stat_activity 函数和视图实现全局 SQL 查询。

1. 执行如下命令连接数据库。

```
gsql -d postgres -p 8000
```

2. 执行如下命令创建 pgxc_stat_activity 函数。

```
DROP FUNCTION PUBLIC.pgxc_stat_activity() cascade;
CREATE OR REPLACE FUNCTION PUBLIC.pgxc_stat_activity
(
OUT coorname text,
OUT datname text,
OUT usename text,
OUT pid bigint,
OUT application_name text,
OUT client_addr inet,
OUT backend_start timestamptz,
OUT xact_start timestamptz,
OUT query_start timestamptz,
OUT state_change timestamptz,
OUT waiting boolean,
OUT enqueue text,
OUT state text,
OUT query_id bigint,
OUT query text
)
RETURNS setof RECORD
AS $$
DECLARE
row_data pg_stat_activity%rowtype;
coor_name record;
fet_active text;
fetch_coor text;
BEGIN
--Get all the node names
fetch_coor := 'SELECT node_name FROM pg_catalog.pgxc_node WHERE
node_type=''C''';
FOR coor_name IN EXECUTE(fetch_coor) LOOP
coorname := coor_name.node_name;
fet_active := 'EXECUTE DIRECT ON (' || coorname || ') ''SELECT * FROM
pg_catalog.pg_stat_activity WHERE pid != pg_catalog.pg_backend_pid() and
application_name not in (SELECT node_name FROM pg_catalog.pgxc_node WHERE
node_type=''''C''''); ''';
FOR row_data IN EXECUTE(fet_active) LOOP
datname := row_data.datname;
pid := row_data.pid;
usename := row_data.usename;
```

```
application_name := row_data.application_name;
client_addr := row_data.client_addr;
backend_start := row_data.backend_start;
xact_start := row_data.xact_start;
query_start := row_data.query_start;
state_change := row_data.state_change;
waiting := row_data.waiting;
enqueue := row_data.enqueue;
state := row_data.state;
query_id := row_data.query_id;
query := row_data.query;
RETURN NEXT;
END LOOP;
END LOOP;
return;
END; $$
LANGUAGE 'plpgsql';
```

3. 执行如下命令创建 pgxc_stat_activity 视图。

```
CREATE VIEW PUBLIC.pgxc_stat_activity AS SELECT * FROM
PUBLIC.pgxc_stat_activity();
```

4. 执行如下 sql 语句查询全局会话信息。

```
SELECT * FROM PUBLIC.pgxc_stat_activity order by coorname;
```

5.21 如何判断表是否做过 update 或 delete 操作

问题现象

DWS 中有两种情况需要关注表是否做过 update 及 delete 操作：

1. 对表频繁做 update 或者 delete 操作会产生大量的磁盘页面碎片，从而逐渐降低查询的效率，需要将磁盘页面碎片恢复并交换操作系统，即 vacuum full 操作，这时候需要找出那些表做过 update；
2. 判断一张表是否是维度表，是否可以从 Hash 表变更为复制表，可以查看这张表是否做过 update 或 delete，如果做过 update 或 delete 操作，则不可以修改为复制表。

处理方法

可以通过以下命令找出那些表做过 update 及 delete 操作：

```
analyze tablename;
SELECT
   n.nspname , c.relname,
   pg_stat_get_tuples_deleted(x.pcrelid) as deleted,
pg_stat_get_tuples_updated(x.pcrelid) as updated
FROM pg_class c
INNER JOIN pg_namespace n ON n.oid = c.relnamespace
INNER JOIN pgxc_class x ON x.pcrelid = c.oid
WHERE c.relkind = 'r' and c.relname='tablename' ;
```

5.22 执行业务报错：Can't fit xid into page

问题现象

场景一：执行 vacuum full 时报错：Can't fit xid into page, now xid is 34181619720, base is 29832807366, min is 3, max is 3.

场景二：其他非 vacuum full 操作，例如某局点给用户赋权 function 时，报错信息如下：

Can't fit xid into page. relation "pg_proc", now xid is 34181619720, base is 29832807366, min is 3, max is 3.

原因分析

系统中存在老事务导致的该报错。

处理方法

场景一处理步骤：

步骤 1 排查是否存在老事务。

```
select * from pgxc_gtm_snapshot_status();
    xmin    |    xmax     |     csn     | oldestxmin   |   xcnt | running_xids
------------+-------------+-------------+--------------+--------+--------------
 34730350588 | 34730350588 |  34730350553 | 34730350553  |    0
(1 row)
```

- 若查询结果中 oldestxmin 与报错中 xid“34181619720”十分接近，且大于 base+min 和 base+max，说明系统中老事务不会影响 freeze 作业，可直接执行步骤 4。
- 若查询结果中 oldestxmin 小于 base+min，且小很多，说明系统中存在老事务，且导致 vacuum freeze 执行未产生作用，需继续执行步骤 2。

步骤 2 使用如下命令查询集群中老事务信息。

```
select * from pgxc_running_xacts where xmin::text::bigint < $base+$min and
xmin::text::bigint > 0;
```

步骤 3 通过 pgxc_stat_activity 视图查询步骤 2 中的业务，确认后执行如下命令终止对应的线程。

```
select pg_terminate_backend(pid) from pgxc_running_xacts where xmin::text::bigint
<$base+$min  and xmin::text::bigint > 0;
```

说明

pgxc_running_xacts 只能查询 CN 上的活跃事务。如果报错的是 DN，需要到对应 DN 上使用 pg_running_xacts 视图进行查询。

步骤 4 对报错表执行 vacuum full freeze。

```
vacuum full freeze table_name;
```

步骤 5 登录 GaussDB(DWS) 管理控制台，查看 vacuum_freeze_min_age 参数，如果设置值为 50 亿，则按照以下方式重新设置为 20 亿：

在集群列表中找到所需集群，单击集群名称，进入“集群详情”页面。单击“参数修改”页签，修改 vacuum_freeze_min_age 参数值，然后单击“保存”。

----结束

场景二处理步骤：

非 vacuum full 操作中的报错信息，可以确认系统中存在老事务，按照场景一中的步骤 2 和步骤 3 清理掉后即可，无需再继续执行 vacuum freeze。

5.23 执行业务报错：unable to get a stable set of rows in the source table

问题现象

merge into 的作用是将源表内容根据匹配条件对目标表做更新或插入，当目标表匹配到多行满足条件时，GaussDB(DWS)有以下两种行为：

1. 业务报错：unable to get a stable set of rows in the source table
2. 随机匹配一行数据，可能会导致实际与预期不符

原因分析

进行 merge into 操作对目标表做更新或插入，目标表匹配到多行满足条件时出现该报错。

处理方法

这两种行为由参数 behavior_compat_options 控制，当参数 behavior_compat_options 缺省的情况下，匹配到多行会报错，如果 behavior_compat_options 被设置为 merge_update_multi，这种情况下不会报错，而是会随机匹配一行数据。

因此，当出现 merge into 的结果与预期不符时，需查看该参数是否被设置，同时排查是否匹配了多行数据，并修改业务逻辑。

5.24 DWS 元数据不一致-分区索引异常

问题现象

某局点查看表定义报错：ERROR: The local index xxx on the partition xxx not exist

原因分析

该问题主表索引无损坏，报错原因由于单个分区索引记录在系统表 pg_partition 中与其他系统表不一致，未找到对应分区索引信息报错。

问题复现

1. 创建表带 p1，p2 两个分区。

```
create table a_0317(a int) partition by range(a) (partition p1 values less than (4), partition p2 values less than (8));
```

2. 创建主表与分区索引。

```
create index a_0317_index on a_0317(a) local (partition p1_index, partition p2_inde);
```

3. 查看分区表分区索引信息如下：

   a. 查看主表索引信息；

```
select oid,* from pg class where relname ='a 0317 index';
  oid   |   relname    | relnamespace | reltype | reloftype | relowner |
relam | relfilenode | reltablespace | relpages | reltuples | relallvisible
| reltoastrelid | reltoastidxid | reldeltarelid | reld
eltaidx | relcudescrelid | relcudescidx | relhasindex | relisshared |
relpersistence | relkind | relnatts | relchecks | relhasoids | relhaspkey
| relhasrules | relhastriggers | relhassubclass | relcmprs
 | relhasclusterkey | relrowmovement | parttype | relfrozenxid | relacl |
reloptions | relreplident | relfrozenxid64
--------+--------------+--------------+---------+-----------+----------+--
------+-------------+---------------+----------+-----------+---------------
+---------------+---------------+---------------+-----
--------+----------------+--------------+-------------+-------------+-----
-----------+---------+----------+-----------+------------+------------+---
----------+----------------+----------------+---------
-+------------------+----------------+----------+--------------+---------+-
-----------+--------------+----------------
 241487 | a_0317_index |        2200 |      0 |        0 |    16393 |   403 |
241487 |           0 |       0 |        0 |           0 |           0 |
0 |           0 |
     0 |            0 |          0 | f         | f         | p            |
i      |       1 |        0 | f         | f         | f          | f
| f            | 0
 | f              | f            | p       | 0           |       |          |
n          |            0
(1 row)
```

   b. 根据主表索引信息查看分区索引信息；

```
select * from pg partition where parentid= 241487;
 relname  | parttype | parentid | rangenum | intervalnum | partstrategy |
```

```
relfilenode | reltablespace | relpages | reltuples | relallvisible |
reltoastrelid | reltoastidxid | indextblid | indisusable |
reldeltarelid | reldeltaidx | relcudescrelid | relcudescidx | relfrozenxid
| intspnum | partkey | intervaltablespace | interval | boundaries |
transit | reloptions | relfrozenxid64
----------+----------+----------+----------+-------------+--------------+-
------------+---------------+----------+----------+---------------+------
--------+---------------+------------+-------------+-
--------------+-------------+----------------+--------------+-------------
-+----------+---------+--------------------+----------+------------+------
---+------------+----------------
 p1_index | x       |   241487 |       0 |          0 | n           |
241488 |           0 |       0 |        0 |           0 |           0 |
0 |    241485 | t          |
          0 |          0 |            0 |          0 | 0           |         |
|                 |         |           |        |           |            0
 p2_inde  | x       |   241487 |       0 |          0 | n           |
241489 |           0 |       0 |        0 |           0 |           0 |
0 |    241486 | t          |
          0 |          0 |            0 |          0 | 0           |         |
|                 |         |           |        |           |            0
(2 rows)
```

4. 连接 CN 开启读写事务，从 pg_partition 系统表删除 p1 分区的索引信息。

```
start transaction read write;
delete from pg_partition where relname = 'p1_index';
```

5. 查看表定义报错与现场报错相同，问题复现：

```
\d+ a_0317
ERROR:  The local index 700633 on the partition 700647 not exist.CONTEXT:
referenced column: pg_get_indexdef
```

处理方法

1. 删除该表索引信息。

```
DROP INDEX a_0317_index;
```

2. 对该表索引进行重建。

```
create index a_0317_index on a_0317(a) local (partition p1_index, partition
p2_inde);
```

3. 查看表定义无报错。

```
\d+ a_0317
                    Table "public.a_0317"
 Column |  Type   | Modifiers | Storage | Stats target | Description
--------+---------+-----------+---------+--------------+-------------
 a      | integer |           | plain   |              |
Indexes:
    "a_0317_index" btree (a) LOCAL(PARTITION p1_index, PARTITION p2_inde)
TABLESPACE pg_default
Range partition by(a)
Number of partition: 2 (View pg_partition to check each partition range.)
Has OIDs: no
Distribute By: HASH(a)
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no
```

5.25 对系统表 pg_catalog.gs_wlm_session_info 执行 truncate 命令报错

问题现象

清理磁盘空间，因系统表 pg_catalog.gs_wlm_session_info 较大（有 20G），在不需要查询历史 sql 语句的前提下，对此系统表执行 TRUNCATE 命令，执行时报错 permission denied for relation gs_wlm_session_info。

```
10  truncate TABLE pg_catalog.gs_wlm_session_info
```

```
truncate TABLE pg_catalog.gs_wlm_session_info
> ERROR:  permission denied for relation gs_wlm_session_info
```

原因分析

对系统表执行 truncate 命令是版本 8.x 及以上集群才支持的，低版本集群不支持 truncate 系统表。

集群版本可登录 GaussDB(DWS) 管理控制台，在“集群管理”页面进入对应集群的“集群详情”页面进行查看。

处理方法

- 8.0 以下版本集群清理系统表需要先执行 DELETE FROM，再执行 VACUUM FULL。

  此处仅以 gs_wlm_session_info 系统表为例：

  ```
  DELETE FROM pg_catalog.gs_wlm_session_info;
  VACUUM FULL pg_catalog.gs_wlm_session_info;
  ```

- 8.0 及以上（8.0.x/8.1.1/8.1.3）版本集群可执行以下命令清理系统表：

  ```
  TRUNCATE TABLE dbms_om.gs_wlm_session_info;
  ```

  说明

  此系统表的 schema 是 dbms_om。

5.26 分区表插入数据报错：inserted partition key does not map to any table partition

问题现象

给范围分区表插入数据报错：inserted partition key does not map to any table partition。

```
CREATE TABLE startend_pt (c1 INT, c2 INT)
DISTRIBUTE BY HASH (c1)
PARTITION BY RANGE (c2) (
    PARTITION p1 START(1) END(1000) EVERY(200) ,
    PARTITION p2 END(2000),
    PARTITION p3 START(2000) END(2500) ,
    PARTITION p4 START(2500),
    PARTITION p5 START(3000) END(5000) EVERY(1000)
);
SELECT partition name,high value FROM dba tab partitions WHERE
table name='startend pt';
 partition_name | high_value
----------------+------------
 p1_0           | 1
 p1_1           | 201
 p1_2           | 401
 p1_3           | 601
 p1_4           | 801
 p1_5           | 1000
 p2             | 2000
 p3             | 2500
 p4             | 3000
 p5_1           | 4000
 p5_2           | 5000
(11 rows)

INSERT INTO startend_pt VALUES (1,5001);
ERROR:  dn_6003_6004: inserted partition key does not map to any table partition
```

原因分析

范围分区是根据表的一列或者多列，将要插入表的数据分为若干个范围，这些范围在不同的分区里没有重叠。划分好分区后，根据分区键值将数据映射到已创建的某个分区上，如果可以映射到已创建的某一分区上，则把记录插入到对应的分区上，否则给出报错和提示信息。

示例中的分区表 tpcds.startend_pt 是以 c2 列为 partition_key，将插入表的数据分 5 个没有重叠的分区。而插入的数据中，c2 列对应的数据 5001 已超过了分区表中划分的分区范围（即 5001>5000），因此报错。

处理方法

应根据数据实际情况规划分区，以保证插入的数据都在规划好的分区中。

若已规划的分区无法满足实际应用条件，可以增加分区，增加分区 c2 介于 5000 和 MAXVALUE 之间：

```
ALTER TABLE startend_pt ADD PARTITION P6 VALUES LESS THAN (MAXVALUE);
SELECT partition_name,high_value FROM dba_tab_partitions WHERE
table_name='startend_pt';
 partition_name | high_value
----------------+------------
 p1_0           | 1
 p1_1           | 201
 p1_2           | 401
 p1_3           | 601
 p1 4           | 801
 p1 5           | 1000
 p2             | 2000
 p3             | 2500
 p4             | 3000
 p5 1           | 4000
 p5 2           | 5000
 p6             | MAXVALUE
(12 rows)

INSERT INTO startend_pt VALUES (1,5001);
SELECT * FROM startend_pt;
 c1 |  c2
----+------
  1 | 5001
(1 row)
```

5.27 查询表报错：missing chunk number %d for toast value %u in pg_toast_XXXX

问题现象

查询表报错：missing chunk number %d for toast value %u in pg_toast_XXXX

原因分析

表关联的 toast 表的 data 发生损坏。

toast 是 The OverSized Attribute Storage Technique(超尺寸字段存储技术)的缩写，是超长字段在 GaussDB(DWS)的一种存储方式。当某表中有超长字段的时候，那么该表会有与之相关联的 toast 表。根据 toast 表的命名规则，假设存在表名为 test 的 OID 为 2619，那么如果存在与之相关联的 toast 表，则 toast 表名为 pg_toast_2619。此报错中 pg_toast_2619 非固定表名，可根据实际报错对 pg_toast_2619 进行替换。

处理方法

步骤 1 通过 toast 的 OID（示例中为 2619，来源于报错信息的 pg_toast_2619）查询出哪张表发生了损坏：

```
SELECT 2619::regclass;
   regclass
--------------
 pg_statistic
(1 row)
```

步骤 2 对已定位的损坏表（步骤 1 中查询得到的表 pg_statistic），执行 REINDEX 和 VACUUM ANALYZE 操作。显示 REINDEX/VACUUM，表示修复完成。若修复过程中出现报错，请继续执行步骤 3。

```
REINDEX table pg_toast.pg_toast_2619;
REINDEX table pg_statistic;
VACUUM ANALYZE pg_statistic;
```

步骤 3 执行以下的命令定位该表中损坏的数据行。

```
DO $$
declare
 v_rec record;
BEGIN
for v_rec in SELECT * FROM pg_statistic loop
       raise notice 'Parameter is: %', v_rec.ctid;
raise notice 'Parameter is: %', v_rec;
end loop;
END;
$$
LANGUAGE plpgsql;
NOTICE:  00000: Parameter is: (46,9)
ERROR:  XX000: missing chunk number 0 for toast value 30982 in pg_toast_2619
CONTEXT:  PL/pgSQL function inline_code_block line 7 at RAISE
```

步骤 4 将步骤 3 中定位的损坏数据行记录删除:

```
DELETE FROM pg_statistic WHERE ctid ='(46,9)';
```

步骤 5 重复执行步骤 3、步骤 4，直到全部有问题的数据记录被删除。

步骤 6 损坏的数据行被删除后，执行步骤 2 中的 REINDEX 和 VACUUM ANALYZE 操作对该表重新进行修复。

----结束

5.28 向表中插入数据报错：duplicate key value violates unique constraint "%s"

问题现象

向表中插入数据报错：duplicate key value violates unique constraint "%s".

```
CREATE TABLE films (
code       char(5) PRIMARY KEY,
title      varchar(40) NOT NULL,
did        integer NOT NULL,
date_prod   date,
```

```
kind      varchar(10),
len       interval hour to minute
);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "films_pkey" for
table "films"
CREATE TABLE

INSERT INTO films VALUES ('UA502', 'Bananas', 105, DEFAULT, 'Comedy', '82 minutes');
INSERT INTO films VALUES ('UA502', 'Bananas', 105, '1971-07-13', 'Comedy', '82
minutes');
ERROR:  dn_6003_6004: duplicate key value violates unique constraint "films_pkey"
DETAIL:  Key (code)=(UA502) already exists.
```

原因分析

创建表 films 时带有主键约束，使用 PRIMARY KEY 声明主键 code，即表中的 code 字段只能包含唯一的非 NULL 值。同时主键会为表 films 创建索引 films_pkey。

插入表中的数据 code 字段值 UA502 与表中已存在的中 code 字段值重复，因此插入数据报错。

处理方法

- 方法一：检查数据冲突，修改插入数据。例如，修改示例重复字段 UA502 为 UA509。

  ```
  INSERT INTO films VALUES ('UA509', 'Bananas', 105, '1971-07-13', 'Comedy', '82
  minutes');
  INSERT 0 1
  ```

- 方法二：删除表 films 主键约束。

  ```
  ALTER TABLE films DROP CONSTRAINT films_pkey;
  ALTER TABLE
  INSERT INTO films VALUES ('UA502', 'Bananas', 105, '1971-07-13', 'Comedy', '82
  minutes');
  INSERT 0 1
  ```

5.29 使用 GaussDB(DWS) 的 ODBC 驱动，SQL 查询结果中字符类型的字段内容会被截断

问题现象

使用 GaussDB(DWS) 的 ODBC 驱动，SQL 查询结果中字符类型的字段内容会被截断，需通过 SQL 语法 CAST BYTEA 转成二进制才能完整取出字段信息。但是，同样的程序连接 ORACLE、SQL SERVER 却没有问题。

原因分析

ODBC 客户端设置了 max varchar=255，导致超过 255 的字段被截断。

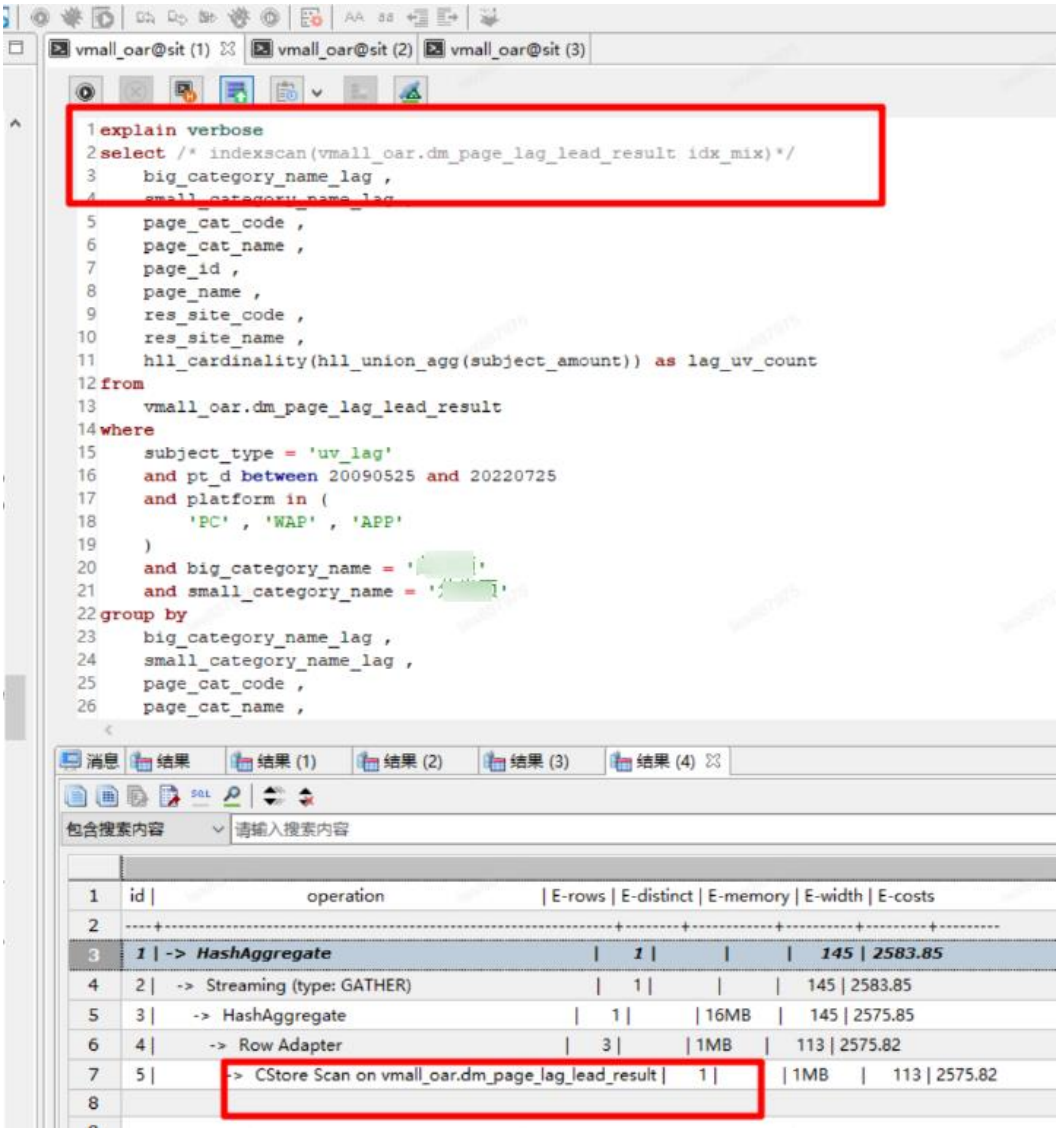
处理方法

在 ODBC 客户端，增大 max varchar 的值即可。

5.30 执行 Plan Hint 的 Scan 方式不生效

问题现象

DWS 中指定了 Plan Hint 的 scan 方式，但是并未生效。



原因分析

Plan Hint 语法使用错误。Plan Hint 的 Scan 语法应在 SELECT 语句中增加“**/*+ indexscan(table_name　index_name) */**”，明显语句中缺少了“**+**”号 。

处理方法

步骤 1 Plan Hint 的 Scan 语法应在 SELECT 语句中增加**/*+ indexscan(table_name index_name) */** 格式。

```
explain verbose
select /*+ indexscan (vmall_oar.dm_page_lag_lead_result idx_mix)*/
......
```

----结束

6 数据库性能、资源

6.1 锁等待检测

操作场景

在日常作业开发中，数据库事务管理中的锁一般指的是**表级锁**，GaussDB(DWS)中支持的锁模式有 8 种，按排他级别分别为 1~8。每种锁模式都有与之相冲突的锁模式，由锁冲突表定义相关的信息，锁冲突表如表 6-1 所示。

举例：用户 u1 对某张表 test 执行 INSERT 事务时，此时持有 RowExclusiveLock 锁；此时用户 u2 也对 test 表进行 VACUUM FULL 事务，则该事务与 INSERT 事务产生锁冲突，处于锁等待状态。

常用的锁等待检测主要通过查询视图 **pgxc_lock_conflicts**、**pgxc_stat_activity**、**pgxc_thread_wait_status**、**pg_locks** 进行。其中 **pgxc_lock_conflicts** 视图在 8.1.x 版本后支持，根据集群版本号不同，检测方式不同。

表6-1 锁冲突

编号	名称	用途	冲突关系
1	AccessShareLock	SELECT	8
2	RowShareLock	SELECT FOR UPDATE/FOR SHARE	7 | 8
3	RowExclusiveLock	INSERT/UPDATE/DELETE	5 | 6 | 7 | 8
4	ShareUpdateExclusiveLock	VACUUM	4 | 5 | 6 | 7 | 8
5	ShareLock	CREATE INDEX	3 | 4 | 6 | 7 | 8
6	ShareRowExclusiveLock	ROW SELECT...FOR UPDATE	3 | 4 | 5 | 6 | 7 | 8
7	ExclusiveLock	BLOCK ROW	2 | 3 | 4 | 5 | 6 | 7 | 8

编号	名称	用途	冲突关系
		SHARE/SELECT...FOR UPDATE	
8	AccessExclusiveLock	DROP CLASS/VACUUM FULL	1 | 2 | 3 | 4 | 5 | 6 | 7 | 8

操作步骤

构造锁等待场景：

步骤 1 打开一个新的连接会话，使用普通用户 u1 连接 GuassDB(DWS)数据库，在自己的同名 SCHEMA u1 下创建测试表 u1.test。

```
CREATE TABLE test (id int, name varchar(50));
```

步骤 2 开启事务 1，进行 INSERT 操作。

```
START TRANSACTION;
INSERT INTO test VALUES (1, 'lily');
```

步骤 3 打开一个新的连接会话，使用系统管理员 dbadmin 连接 GaussDB(DWS)数据库，执行 VACUUM FULL 操作，发现语句阻塞。

```
VACUUM FULL u1.test;
```

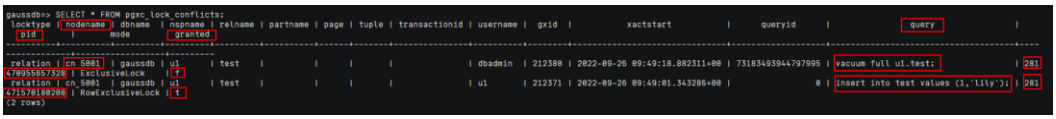
----结束

锁等待检测（8.1.x 及以上版本）

步骤 1 打开一个新的连接会话，使用系统管理员 dbadmin 连接 GaussDB(DWS)数据库，通过 pgxc_lock_conflicts 视图查看锁冲突情况。

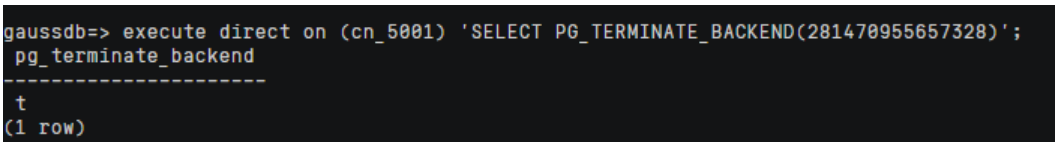
如下图，回显中查看 granted 为“f”，表示 VACUUM FULL 语句正在等待其他锁。granted 为“t”，表示 INSERT 语句是持有锁。nodename，表示锁产生在的位置，即 CN 或 DN 位置，例如 cn_5001。

```
SELECT * FROM pgxc_lock_conflicts;
```



步骤 2 据语句内容确认是中止持锁语句。如果终止，则执行以下语句。*pid* 从步骤 1 获取，cn_5001 为上面查询到的 nodename。

```
execute direct on (cn_5001) 'SELECT PG_TERMINATE_BACKEND(pid)';
```

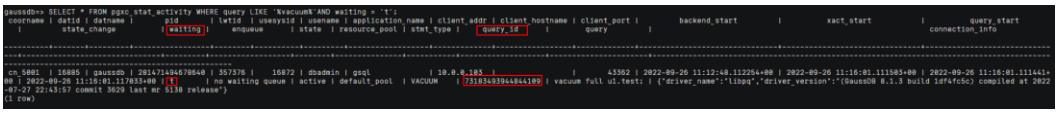


----结束

锁等待检测（8.0.x 及以前版本）

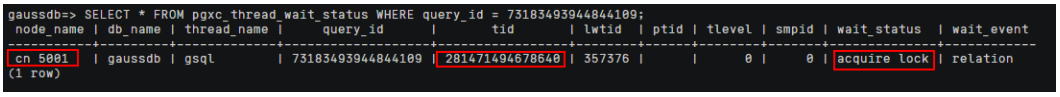
步骤 1 在数据库中执行以下语句，获取 VACUUM FULL 操作对应的 query_id。

```
SELECT * FROM pgxc_stat_activity WHERE query LIKE '%vacuum%'AND waiting = 't';
```



步骤 2 根据获取的 query_id，执行以下语句查看是否存在锁等待，并获取对应的 tid。其中，*{query_id}*从步骤 1 获取。

```
SELECT * FROM pgxc_thread_wait_status WHERE query_id = {query_id};
```

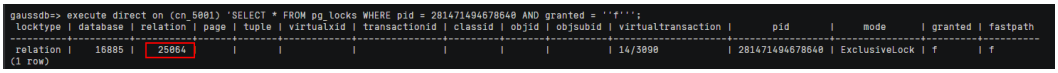


回显中“wait_status”存在“acquire lock”表示存在锁等待。同时查看“node_name”显示在对应的 CN 或 DN 上存在锁等待，记录相应的 CN 或 DN 名称，例如 cn_5001 或 dn_600x_600y。

步骤 3 执行以下语句，到等锁的对应 CN 或 DN 上从 pg_locks 中查看 VACUUM FULL 操作在等待哪个锁。以下以 cn_5001 为例，如果在 DN 上等锁，则改为相应的 DN 名称。pid 为步骤 2 获取的 tid。

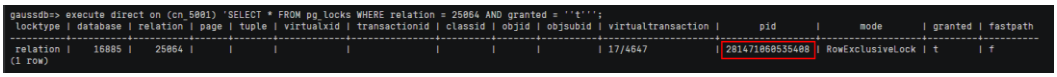
回显中记录 relation 的值。

```
execute direct on (cn_5001) 'SELECT * FROM pg_locks WHERE pid = {tid} AND granted =
''f''';
```



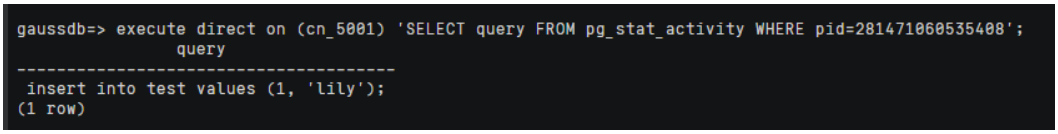
步骤 4 根据获取的 relation，从 pg_locks 中查看当前持有锁的 pid。*{relation}*从步骤 3 获取。

```
execute direct on (cn_5001) 'SELECT * FROM pg_locks WHERE relation = {relation} AND
granted = ''t''';
```



步骤 5 根据 pid，执行以下语句，查到对应的 SQL 语句。*{pid}*从步骤 4 获取。

```
execute direct on (cn_5001) 'SELECT query FROM pg_stat_activity WHERE pid={pid}';
```



步骤 6 据语句内容确认是中止持锁语句还是另找时间做 VACUUM FULL。如果终止，则执行以下语句。*pid* 从步骤 4 获取。

中止结束后，再尝试重新执行 VACUUM FULL。

```
execute direct on (cn_5001) 'SELECT PG_TERMINATE_BACKEND(pid)';
```

```
gaussdb=> execute direct on (cn_5001) 'SELECT PG_TERMINATE_BACKEND(281471060535408)';
 pg_terminate_backend
----------------------
 t
(1 row)
```

----结束

6.2 执行 SQL 时出现表死锁，提示 LOCK_WAIT_TIMEOUT 锁等待超时

问题现象

执行 SQL 时出现 LOCK_WAIT_TIMEOUT 锁等待超时的错误。

原因分析

锁等待超时一般是因为有其他的 SQL 语句已经持有了锁，当前 SQL 语句需要等待持有锁的 SQL 语句执行完毕释放锁之后才能执行。当申请的锁等待时间超过 GUC 参数 lockwait_timeout 的设定值时，系统会报 LOCK_WAIT_TIMEOUT 的错误。

处理方法

1. 执行以下 SQL 查询查看是否有阻塞的 SQL 语句，如果有，将阻塞的 SQL 会话强制结束。

   ```
   SELECT w.query as waiting_query,
   w.pid as w_pid,
   w.usename as w_user,
   l.query as locking_query,
   l.pid as l_pid,
   l.usename as l_user,
   t.schemaname || '.' || t.relname as tablename
   from pg_stat_activity w join pg_locks l1 on w.pid = l1.pid
   and not l1.granted join pg_locks l2 on l1.relation = l2.relation
   and l2.granted join pg_stat_activity l on l2.pid = l.pid join
   pg_stat_user_tables t on l1.relation
   = t.relid
   where w.waiting;
   ```

   查询到阻塞的表及模式信息后，请根据会话 ID 结束会话：

   ```
   SELECT PG_TERMINATE_BACKEND(PID);
   ```

2. 这种情况一般是因为业务调度不太合理，建议合理安排各个业务的调度时间。
3. 还可以通过设置 GUC 参数 lockwait_timeout，控制单个锁的最长等待时间，即单个锁的等待超时时间。

   lockwait_timeout 单位为毫秒（ms），默认值为 20 分钟。

   lockwait_timeout 参数属于 SUSET 类型参数，请参考《数据仓库服务开发指南》中的“配置 GUC 参数 > 设置 GUC 参数”中对应的设置方法进行设置。

6.3 SQL 执行很慢，性能低，有时长时间运行未结束

问题现象

SQL 执行很慢，性能低，有时长时间运行未结束。

原因分析

SQL 运行慢，主要可以从以下几方面进行分析：

1. 使用 EXPLAIN 命令查看 SQL 执行计划，根据执行计划判断是否需要进行 SQL 调优。
2. 分析查询是否被阻塞，导致语句运行时间过长，可以强制结束有问题的会话。
3. 审视和修改表定义。选择合适的分布列，避免数据倾斜。
4. 分析 SQL 语句是否使用了不下推的函数，建议更换为支持下推的语法或函数。
5. 对表定期做 vacuum full 和 analyze，可回收已更新或已删除的数据所占据的磁盘空间。
6. 检查表有无索引支撑，建议例行重建索引。

   数据库经过多次删除操作后，索引页面上的索引键将被删除，造成索引膨胀。例行重建索引，可有效的提高查询效率。
7. 对业务进行优化，分析能否将大表进行分表设计。

处理方法

GaussDB(DWS) 提供了分析查询和改进查询的方法，并且为用户提供了一些常见案例以及错误处理办法。您可以参考《数据仓库服务开发指南》中的“优化查询性能”章节对 SQL 进行性能调优。常见问题也可以优先参考以下两种方法进行分析：

方法一：对表定期做统计优化查询

- 经常变化的表，如果经常 insert 语句到表中，需要做 analyze 表，具体语句为：

  ```
  Analyze tablename;
  ```
- 经常变化的表，如果经常删除 delete 数据，需要做 vacuum full 表，具体语句为：

  ```
  Vacuum full tablename;
  ```

  说明

  执行 vacuum full 语句时注意不能有其他任务在跑。
- 查询表大小，如果表非常大，而实际只有很少数据，那么应该执行 vacuum full 对表进行磁盘碎片整理。

  ```
  select * from pg_size_pretty(pg_table_size('tablename'));
  ```

方法二：通过 pgxc_stat_activity 查询正在运行的 sql 相关信息

步骤 1 查询后台活跃 sql

```
SELECT pid,datname, usename, state,waiting, query FROM pgxc_stat_activity WHERE
state <> 'idle';
```

步骤 2 查询后台业务有锁的 sql

```
SELECT pid,datname, usename, state,waiting, query FROM pgxc_stat_activity WHERE
state <> 'idle' and waiting=true;
```

步骤 3 判断是否被锁

1. 如果没有锁，查找相关业务 sql，按照**方法一**中的判断方法进行处理。
2. 如果有锁，则查找出 pid 字段，使用如下函数，结束任务，释放锁。

```
SELECT pg_terminate_backend(pid);
```

----结束

6.4 数据倾斜导致 SQL 执行慢，大表 SQL 执行无结果

问题现象

某场景下 SQL 执行慢，涉及大表的 SQL 执行不出来结果。

原因分析

GaussDB(DWS)支持 Hash 表、复制表和 ROUNDROBIN（8.1.2 集群及以上版本支持 ROUNDROBIN）分布方式。如果创建了 Hash 分布的表，未指定分布键，则选择表的第一列作为分布键，这种情况就可能存在倾斜。倾斜造成以下负面影响：

- SQL 的性能会非常差，因为数据只分布在部分 DN，那么 SQL 运行的时候就只有部分 DN 参与计算，没有发挥分布式的优势。
- 会导致资源倾斜，尤其是磁盘。可能部分磁盘的空间已经接近极限，但是其他磁盘利用率很低。
- 可能出现部分节点 CPU 过高等等问题。

分析过程

步骤 1 登录 GaussDB(DWS) 管理控制台。在“集群管理”页面，找到需要查看监控的集群。在指定集群所在行的“操作”列，单击“监控面板”。选择“监控>节点监控>磁盘”，查看磁盘使用率。

说明

各个数据磁盘的利用率，会有不均衡的现象。正常情况下，利用率最高和利用率最低的磁盘空间相差不大，如果磁盘利用率相差超过了 5%就要注意是不是有资源倾斜的情况。

步骤 2 连接数据库，通过等待视图查看作业的运行情况，发现作业总是等待部分 DN，或者个别 DN。

```
select wait_status, count(*) as cnt from pgxc_thread_wait_status where wait_status
not like '%cmd%' and wait_status not like '%none%' and wait_status not like
'%quit%' group by 1 order by 2 desc;
```

步骤 3 执行慢语句的 explain performance 显示，发现各个 DN 的基表 scan 的时间和行数不均衡。

```
explain performance select avg(ss_wholesale_cost) from store_sales;
```

```
rmance select avg(ss_wholesale_cost) from store_sales;

      operation                                 |      A-time       |  A-rows  | E-rows  | E-distinct |   Peak Memory   | E-memory | A-wi
------------------------------------------------+-------------------+----------+---------+------------+-----------------+----------+-----
                                                | 2452.321          |        1 |       1 |            | 11KB            |          |
te                                              | 2452.238          |        1 |       1 |            | 184KB           |          |
aming (type: GATHER)                            | 2452.010          |        4 |       4 |            | 108KB           |          |
ggregate                                        | [12.219,2425.225] |        4 |       4 |            | [183KB, 184KB]  | 1MB      |
r Partition Iterator                            | [12.139,1189.187] | 22831616 | 2880404 |            | [17KB, 17KB]    | 1MB      |
rtitioned CStore Scan on public.store_sales     | [5.929,1173.555]  | 22831616 | 2880404 |            | [482KB, 499KB]  | 1MB      |


identified by plan id)
--------------------------
rator

can on public.store_sales
s:  1..7
```

- 基表 scan 的时间：最快的 DN 耗时 5ms，最慢的 DN 耗时 1173ms。
- 数据分布情况：某些 DN 有 22831616 行，其他 DN 都是 0 行，数据有严重倾斜。

```
5 --Vector Partition Iterator
      datanode1 (actual time=0.620..1189.187 rows=22831616 loops=1)
      datanode2 (actual time=14.346..14.346 rows=0 loops=1)
      datanode3 (actual time=14.424..14.424 rows=0 loops=1)
      datanode4 (actual time=12.139..12.139 rows=0 loops=1)
      datanode1 (CPU: ex c/r=1, ex row=22831616, ex cyc=40540820, inc cyc=3088825848)
      datanode2 (CPU: ex c/r=0, ex row=0, ex cyc=20852952, inc cyc=37297912)
      datanode3 (CPU: ex c/r=0, ex row=0, ex cyc=20005612, inc cyc=37501436)
      datanode4 (CPU: ex c/r=0, ex row=0, ex cyc=16147884, inc cyc=31560524)
6 --Partitioned CStore Scan on public.store_sales
      datanode1 (actual time=2.611..1173.555 rows=22831616 loops=7)
      datanode2 (actual time=6.327..6.327 rows=0 loops=7)
      datanode3 (actual time=6.732..6.732 rows=0 loops=7)
      datanode4 (actual time=5.929..5.929 rows=0 loops=7)
      datanode1 (Buffers: shared hit=1359)
      datanode2 (Buffers: shared hit=55)
      datanode3 (Buffers: shared hit=55)
      datanode4 (Buffers: shared hit=55)
      datanode1 (CPU: ex c/r=133, ex row=22831616, ex cyc=3048285028, inc cyc=3048285028)
      datanode2 (CPU: ex c/r=0, ex row=0, ex cyc=16444960, inc cyc=16444960)
      datanode3 (CPU: ex c/r=0, ex row=0, ex cyc=17495824, inc cyc=17495824)
      datanode4 (CPU: ex c/r=0, ex row=0, ex cyc=15412640, inc cyc=15412640)
```

步骤 4 通过倾斜检查接口可以发现数据倾斜。

```
select table_skewness('store_sales');
```

```
tpcds1xcpm=# select table_skewness('store_sales');
                table_skewness
-----------------------------------------------
 ("datanode1            ",22831616,100.000%)
 ("datanode2            ",0,0.000%)
 ("datanode3            ",0,0.000%)
 ("datanode4            ",0,0.000%)
(4 rows)
```

```
select table_distribution('public','store_sales');
```

```
tpcds1xcpm=# select table_distribution('public','store_sales');
           table_distribution
-----------------------------------------
 (public,store_sales,datanode1,1011752960)
 (public,store_sales,datanode4,33792000)
 (public,store_sales,datanode2,32129024)
 (public,store_sales,datanode3,33349632)
(4 rows)
```

步骤 5 通过资源监控发现，个别节点的 CPU/IO 明显比其他节点高。

----结束

处理方法

如何找到倾斜的表：

1. 数据库中表个数少于 1W 的场景，直接使用倾斜视图查询当前库内所有表的数据倾斜情况。

```
SELECT * FROM pgxc_get_table_skewness ORDER BY totalsize DESC;
```

2. 数据库中表个数非常多（至少大于 1W）的场景，因 PGXC_GET_TABLE_SKEWNESS 视图涉及全库查并计算非常全面的倾斜字段，所以可能会花费比较长的时间（小时级），建议参考 PGXC_GET_TABLE_SKEWNESS 视图定义，执行以下操作：
   - 8.1.2 及之前集群版本中使用 table_distribution()函数自定义输出，减少输出列进行计算优化，例如：

```
SELECT schemaname,tablename,max(dnsize) AS maxsize, min(dnsize) AS minsize
FROM pg_catalog.pg_class c
INNER JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
INNER JOIN pg_catalog.table_distribution() s ON s.schemaname = n.nspname
AND s.tablename = c.relname
INNER JOIN pg_catalog.pgxc_class x ON c.oid = x.pcrelid AND
x.pclocatortype = 'H'
GROUP BY schemaname,tablename;
```

   - 8.1.3 及以上集群版本中支持使用 gs_table_distribution()函数，全库查询所有表的数据倾斜情况。全库表查询时，gs_table_distribution()函数优于 table_distribution()函数；在大集群大数据量场景下，如果进行全库表表查询，建议优先使用 gs_table_distribution()函数。

```
SELECT schemaname,tablename,max(dnsize) AS maxsize, min(dnsize) AS minsize
FROM pg_catalog.pg_class c
INNER JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
INNER JOIN pg_catalog.gs_table_distribution() s ON s.schemaname =
n.nspname AND s.tablename = c.relname
INNER JOIN pg_catalog.pgxc_class x ON c.oid = x.pcrelid AND
x.pclocatortype = 'H'
GROUP BY schemaname,tablename;
```

说明

使用如下语句可快速查询到大表：

```
select schemaname||'.'||tablename as table, sum(dnsize) as size from
gs_table_distribution() group by 1 order by 2 desc limit 10;
```

使用如下语句可快速查询表的倾斜率：

```
WITH skew AS
(
      SELECT
            schemaname,
            tablename,
            pg_catalog.sum(dnsize) AS totalsize,
            pg_catalog.avg(dnsize) AS avgsize,
            pg_catalog.max(dnsize) AS maxsize,
            pg_catalog.min(dnsize) AS minsize,
            (pg_catalog.max(dnsize) - pg_catalog.min(dnsize)) AS skewsize,
            pg_catalog.stddev(dnsize) AS skewstddev
      FROM pg_catalog.pg_class c
```

```
        INNER JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
        INNER JOIN pg_catalog.gs_table_distribution() s ON s.schemaname =
n.nspname AND s.tablename = c.relname
        INNER JOIN pg_catalog.pgxc_class x ON c.oid = x.pcrelid AND
x.pclocatortype IN('H', 'N')
        GROUP BY schemaname,tablename
)
SELECT
        schemaname,
        tablename,
        totalsize,
        avgsize::numeric(1000),
        (maxsize/totalsize)::numeric(4,3)  AS maxratio,
        (minsize/totalsize)::numeric(4,3)  AS minratio,
        skewsize,
        (skewsize/totalsize)::numeric(4,3)  AS skewratio,
        skewstddev::numeric(1000)
FROM skew
WHERE totalsize > 0;
```

表的分布键的选择方法：

1. 若此列的 distinct 值比较大，并且没有明显的数据倾斜，也可以把多列定义成分布列。

   如何看 distinct 的大小？

   ```
   select count(distinct column1) from table;
   ```

   如何看数据是不是有倾斜？

   ```
   select count(*) cnt, column1 from table group by column1 order by cnt limit 100;
   ```

2. 选用经常做 JOIN 字段/group by 的列，可以减少 STREAM 运算。
3. 不推荐的实践：
   a. 分布列用默认值（第一列）。
   b. 分布列用 sequence 自增生成。
   c. 分布列用随机数生成（除非任意列，或者任意两列的组合做分布键都是倾斜的，一般不选用这种方法）。

6.5 VACUUM FULL 一张表后，表文件大小无变化

问题现象

使用 VACUUM FULL 命令对一张表进行清理，清理完成后表大小和清理前一样大。

原因分析

假定该表的名称为 table_name，对于该现象可能有以下两种原因：

1. table_name 表本身没有 delete 过数据，使用 VACUUM FULL table_name 后无需清理 delete 的数据。因此表大小清理前后一样大。

2. 在执行 VACUUM FULL table_name 时有并发的事务存在，可能会导致 VACUUM FULL 跳过清理那些最近删除的数据，导致清理不完全。

解决办法

对于可能原因的第二种情况，给出如下两种处理方法：

- 如果在 VACUUM FULL 时有并发的事务存在，此时需要等待所有事务结束，再次执行 VACUUM FULL 命令对该表进行清理。
- 如果使用上面的方法清理后，表文件大小仍然无变化，确认集群中没有正在运行的任务，且所有数据都已经保存后，可使用如下操作方式：

步骤 1 执行以下命令查询当前的事务 XID。

```
select txid_current();
```

步骤 2 再执行以下命令查看活跃事务列表：

```
select txid_current_snapshot();
```

步骤 3 如果发现活跃事务列表中有 XID 比当前的事务 XID 小时，重启集群后，再次使用 VACUUM FULL 命令对该表进行清理。

----结束

6.6 Delete 表数据后执行了 VACUUM，但是空间并没有释放

问题现象

Delete 表数据后执行了 VACUUM，但是存储空间并没有释放。

原因分析

- 执行 VACUUM 时，对某些表可能没有权限，或者数据库本身并没有太多的数据膨胀。
- 执行 VACUUM，默认清理当前用户在数据库中拥有权限的每一个表，没有权限的表则直接跳过回收操作。
- 参数 vacuum_defer_cleanup_age 不是 0，该参数在老版本默认为 8000，表示最近 8000 个事务产生的脏数据不进行回收。
- 为了保证事务可见性，产生脏数据的事务号，如果大于当前活跃的老事务号，则这部分脏数据也不会清理。

处理方法

- 建议对单个表执行 vacuum full 命令，命令格式为“vacuum full 表名”。
- 如果您对表没有权限，请联系数据库管理员或表的所有者进行处理。
- 对于 vacuum_defer_cleanup_age 不是 0 的场景，可以将此参数改为 0，取消 vacuum 的事务延迟。

- 对于存在老事务的场景，重启集群再重新做 vacuum full 可以保证空间一定回收，否则只能等老事务结束再做 vacuum full。

6.7 表数据膨胀导致 SQL 查询慢，用户前台页面数据加载不出

问题现象

GaussDB 数据库性能时快时慢问题，原先执行几秒钟的 sql，当前执行 20 几秒未出结果，导致前台页面数据加载超时，无法对用户提供图表显示。

原因分析

- 大量表频繁增删改，未及时清理，导致脏数据过多，表数据膨胀，查询慢。
- 内存参数设置不合理

分析过程

步骤 1 和客户确认是部分业务慢，可以提供部分慢 sql，打印执行计划，耗时主要在 index scan 上，怀疑是 IO 争抢导致，通过监控 IO，发现并没有 IO 资源使用瓶颈。

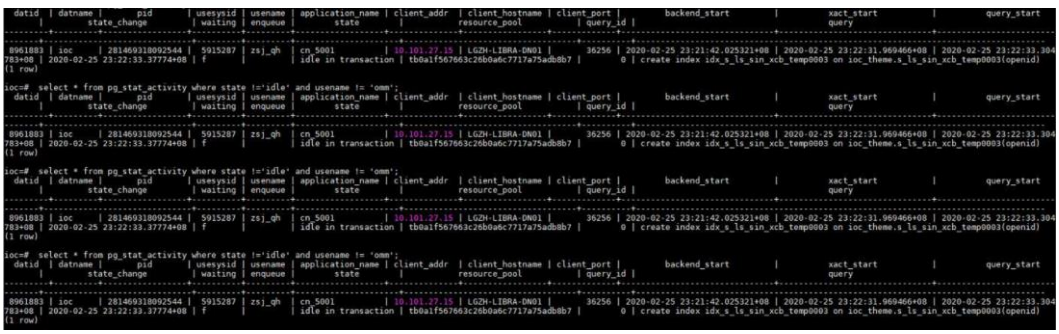
```
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------
   3 --Hash Join (4,18)
       Hash Cond: ((t1.area_code)::text = (t5.area_code)::text)
   5 --Hash Join (6,16)
       Hash Cond: ((t1.measure_unit_code)::text = (t4.measure_unit_code)::text)
   7 --Hash Join (8,14)
       Hash Cond: ((t1.value_type_code)::text = (t3.value_type_code)::text)
   9 --Hash Join (10,12)
       Hash Cond: ((t1.index_code)::text = (t2.index_code)::text)
  11 --Index Scan using idx_m_ss_index_event_index on ioc_dm.m_ss_index_event t1
       Filter: ((t1.data_status = 1::numeric) AND (length((t1.occur_period)::text) = 2) AND ((t1.index_code)::bigint = 100100010011::bigint) AND ((t
1.area_code)::bigint = 440307000000::bigint) AND ((t1.time_type_code)::bigint = 8) AND (to_char(t1.update_time, 'yyyymmdd'::text) = to_char(('2020-02-
25'::date)::timestamp with time zone, 'yyyymmdd'::text)))
       Rows Removed by Filter: 51913
(11 rows)
```

id	operation	A-time	A-rows	E-rows	Peak Memory	A-width	E-width	E-costs
1	-> Streaming (type: GATHER)	21306.396	16	15	303KB		339	15154.56
2	-> Sort	[42.902,21259.337]	16	15	[31KB, 33KB]	[0,58]	339	15153.23
3	-> Hash Join (4,18)	[42.872,21259.214]	16	15	[8KB, 8KB]		339	15153.22
4	-> Streaming(type: REDISTRIBUTE)	[21213.855,21216.616]	16	15	[142KB, 144KB]		270	15124.36
5	-> Hash Join (6,16)	[21218.306,21249.577]	16	15	[7KB, 7KB]		270	15122.84
6	-> Streaming(type: BROADCAST)	[21217.594,21248.730]	240	225	[144KB, 144KB]		190	15097.21
7	-> Hash Join (8,14)	[0.313,21225.478]	16	15	[7KB, 7KB]		190	15092.59
8	-> Streaming(type: BROADCAST)	[21198.318,21224.950]	96	225	[144KB, 144KB]		110	15066.96
9	-> Hash Join (10,12)	[21208.801,21218.437]	16	15	[7KB, 7KB]		110	15064.65
10	-> Streaming(type: BROADCAST)	[21161.272,21215.516]	240	15	[144KB, 144KB]		41	15039.07
11	-> Index Scan using idx_m_ss_index_event_index on ioc_dm.m_ss_index_event t1	[762.317,21177.799]	16	1	[40KB, 40KB]		41	15036.76
12	-> Hash	[0.469,55.742]	2310	152	[291KB, 291KB]	[46,48]	116	25.25
13	-> Seq Scan on ioc_ods.o_gwd_ioc_index t2	[0.335,55.571]	2310	150	[16KB, 16KB]		116	25.25
14	-> Hash	[0.003,15.530]	8	152	[259KB, 291KB]	[0,32]	116	25.25
15	-> Seq Scan on ioc_ods.o_gwd_ioc_value_type t3	[0.001,15.516]	8	150	[15KB, 15KB]		116	25.25
16	-> Hash	[0.049,0.105]	160	152	[291KB, 291KB]	[25,29]	116	25.25
17	-> Seq Scan on ioc_ods.o_gwd_ioc_measure_unit t4	[0.037,0.085]	160					

```
11 --Index Scan using idx_m_ss_index_event_index on ioc_dm.m_ss_index_event t1
        dn_6001_6002 (actual time=0.209..2142.458 rows=3 loops=1)
        dn_6003_6004 (actual time=762.317..762.317 rows=0 loops=1)
        dn_6005_6006 (actual time=9.738..20835.282 rows=2 loops=1)
        dn_6007_6008 (actual time=7345.547..7345.547 rows=0 loops=1)
        dn_6009_6010 (actual time=0.257..7235.551 rows=4 loops=1)
        dn_6011_6012 (actual time=20024.688..20024.688 rows=0 loops=1)
        dn_6013_6014 (actual time=17878.685..17878.685 rows=0 loops=1)
        dn_6015_6016 (actual time=17078.916..17078.916 rows=0 loops=1)
        dn_6017_6018 (actual time=17827.799..17827.799 rows=0 loops=1)
        dn_6019_6020 (actual time=0.253..15975.299 rows=2 loops=1)
        dn_6021_6022 (actual time=21177.799..21177.799 rows=0 loops=1)
        dn_6023_6024 (actual time=0.278..15016.516 rows=1 loops=1)
        dn_6025_6026 (actual time=0.264..16628.971 rows=2 loops=1)
        dn_6027_6028 (actual time=0.270..16635.989 rows=2 loops=1)
        dn_6029_6030 (actual time=12725.526..12725.526 rows=0 loops=1)
        dn_6001_6002 (Buffers: shared hit=485 read=22013 written=5)
        dn_6003_6004 (Buffers: shared hit=512 read=22041 written=4)
        dn_6005_6006 (Buffers: shared hit=481 read=22080 written=43)
        dn_6007_6008 (Buffers: shared hit=539 read=22105 written=12)
        dn_6009_6010 (Buffers: shared hit=463 read=22074 written=13)
        dn_6011_6012 (Buffers: shared hit=481 read=22128 written=65)
        dn_6013_6014 (Buffers: shared hit=534 read=22067 written=73)
        dn_6015_6016 (Buffers: shared hit=560 read=22153 written=50)
        dn_6017_6018 (Buffers: shared hit=535 read=21961 written=44)
        dn_6019_6020 (Buffers: shared hit=507 read=22133 written=58)
        dn_6021_6022 (Buffers: shared hit=466 read=22190 written=41)
        dn_6023_6024 (Buffers: shared hit=476 read=22087 written=44)
        dn_6025_6026 (Buffers: shared hit=502 read=21973 written=44)
        dn_6027_6028 (Buffers: shared hit=442 read=22111 written=44)
        dn_6029_6030 (Buffers: shared hit=476 read=22009 written=39)
        dn_6001_6002 (CPU: ex c/r=35707713, ex cyc=107123139, inc cyc=107123139)
        dn_6003_6004 (CPU: ex c/r=0, ex cyc=38115922, inc cyc=38115922)
        dn_6005_6006 (CPU: ex c/r=520883939, ex cyc=1041767878, inc cyc=1041767878)
        dn_6007_6008 (CPU: ex c/r=0, ex cyc=367278665, inc cyc=367278665)
        dn_6009_6010 (CPU: ex c/r=90444697, ex cyc=361778791, inc cyc=361778791)
        dn_6011_6012 (CPU: ex c/r=0, ex cyc=1001235015, inc cyc=1001235015)
        dn_6013_6014 (CPU: ex c/r=0, ex cyc=893934788, inc cyc=893934788)
        dn_6015_6016 (CPU: ex c/r=0, ex cyc=853946318, inc cyc=853946318)
        dn_6017_6018 (CPU: ex c/r=0, ex cyc=891390498, inc cyc=891390498)
        dn_6019_6020 (CPU: ex c/r=399382685, ex cyc=798765371, inc cyc=798765371)
        dn_6021_6022 (CPU: ex c/r=0, ex cyc=1058894369, inc cyc=1058894369)
        dn_6023_6024 (CPU: ex c/r=750828892, ex cyc=750828892, inc cyc=750828892)
        dn_6025_6026 (CPU: ex c/r=415725991, ex cyc=831451982, inc
```

步骤 2 查询当前活跃 sql，发现有大量的 create index 语句，需要和客户确认该业务是否合理。

```
select * from pg_stat_activity where state !='idle' and usename !='Ruby';
```



步骤 3 根据执行计划，发现在部分 DN 上耗时较高，查询表的倾斜情况，并未发现有倾斜的情况。

```
select table_skewness('table name');
```

```
ioc=# select table_skewness('                                    ')
ioc-# ;
          table_skewness
-----------------------------------------
 ("dn_6017_6018          ",3536,6.809%)
 ("dn_6019_6020          ",3514,6.767%)
 ("dn_6007_6008          ",3513,6.765%)
 ("dn_6013_6014          ",3512,6.763%)
 ("dn_6003_6004          ",3495,6.730%)
 ("dn_6023_6024          ",3491,6.723%)
 ("dn_6015_6016          ",3490,6.721%)
 ("dn_6005_6006          ",3470,6.682%)
 ("dn_6009_6010          ",3466,6.674%)
 ("dn_6029_6030          ",3452,6.647%)
 ("dn_6021_6022          ",3446,6.636%)
 ("dn_6001_6002          ",3419,6.584%)
 ("dn_6027_6028          ",3413,6.572%)
 ("dn_6011_6012          ",3363,6.476%)
```

步骤 4 联系运维人员登录集群实例，检查内存相关参数，设置不合理，需要优化。

- 单节点总内存大小为 256G
- max_process_memory 为 12G，设置过小
- shared_buffers 为 32M，设置过小
- work_mem：CN：64M 、DN：64M
- max_active_statements: -1（不限制并发数）

步骤 5 按以下值设置：

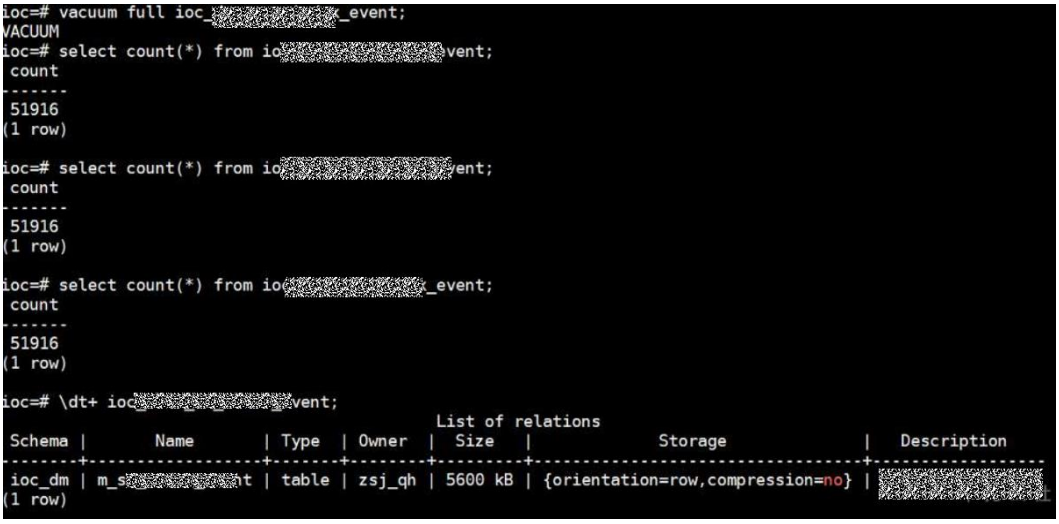
gs_guc set -Z coordinator -Z datanode -N all -I all -c "max_process_memory=25GB"

gs_guc set -Z coordinator -Z datanode -N all -I all -c "shared_buffers=8GB"

gs_guc set -Z coordinator -Z datanode -N all -I all -c "work_mem=128MB"

步骤 6 进一步分析扫描慢的原因，发现表数据膨胀严重，对其中一张 8G 大小的表，总数据量 5 万条，做完 vacuum full 后大小减小为 5.6M。

```
ioc=# \dt+  io                    x_event;
                                  List of relations
 Schema |       Name       | Type  | Owner  |  Size   |            Storage             |  Description
--------+------------------+-------+--------+---------+--------------------------------+---------------
 ioc_dm | m_s              nt | table | zsj_qh | 8416 MB | {orientation=row,compression=no} |
(1 row)
```



----结束

处理方法

步骤 1 对业务涉及到的常用的大表，执行 vacuum full 操作，清理脏数据。

步骤 2 设置 GUC 内存参数。

----结束

6.8 CPU 使用率持续很高，如何定位占用 CPU 高的业务 sql?

问题现象

CPU 使用率持续很高。

原因分析

在执行 SQL 的过程中，某个线程占用 CPU 过高。

处理方法

1. 找一个 gaussdb 进程，确认进程号和对应 dn 的端口号（例如进程号：32089）。
2. 执行如下命令，可以查到进程内各个线程占用 CPU 的情况。

```
ps H -eo pid,tid,pcpu | sort -n -k 3 | grep 32089,
32089    48369    81.9
32089    41335    82.3
32089    10596    85.6
```

3. 连续执行多次，查询没有变化的那条高线程，连接到此 dn。

```
select query_id from pg_thread_wait_status where lwtid = 10596;
```

查到 query_id 为 95174458

4. 执行如下命令，则可查询到对应的 SQL 语句。

```
select * from pg_stat_activity where query_id = 95174458;
```

6.9 执行 VACUUM FULL 命令时提示 Lock wait timeout 错误

问题现象

执行 vacuum full 命令时报错：

```
[0]ERROR: dn_6009_6010: Lock wait timeout: thread 140158632457984 on node
dn_6009_6010 waiting for AccessExclusiveLock on relation 2299036 of database 14522
after 1202001.968 ms
Detail: blocked by hold lock thread 140150147380992, statement <<backend
information not available>>, hold lockmode AccessShareLock.
Line Number: 1
```

原因分析

日志中的“Lock wait timeout”说明锁等待超时。锁等待超时一般是因为有其他的 SQL 语句已经持有了锁，当前 SQL 语句需要等待持有锁的 SQL 语句执行完毕释放锁之后才能执行。当申请的锁等待时间超过 GUC 参数 lockwait_timeout 的设定值时，系统会报 LOCK_WAIT_TIMEOUT 的错误。

执行 vacuum full 命令时出现报错的原因为一般为执行命令超时，如果对整个数据库执行 vacuum full 执行时间较长可能会超时。

处理方法

建议对单个表执行 vacuum full 命令，命令格式为“vacuum full 表名”，同时增加执行“vacuum full”命令的频率。尤其是对于频繁增、删、改的表，建议定期做 vacuum full 操作。

6.10 VACUUM FULL 执行慢

问题现象

对表进行 VACUUM FULL 进行空间回收时执行很慢。

原因分析

造成 VACUUM FULL 执行慢主要有以下原因：

- 存在锁等待
- IO/网络问题
- 系统表过大

- 表使用了局部聚簇（PCK），psort_work_mem 设置过小。

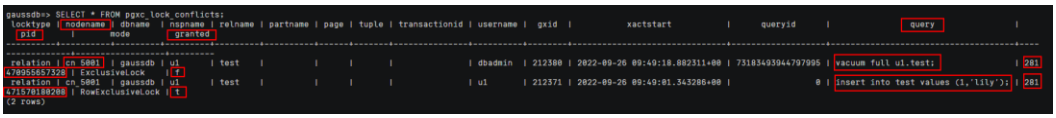
处理方法

- **8.1.x 及以上版本参见如下方法：**

步骤 1 常见的原因是存在锁等待，先排查此原因。在数据中执行以下语句，通过 pgxc_lock_conflicts 视图查看锁冲突情况。

- 如下图，回显中查看 granted 为“f”，表示 VACUUM FULL 语句正在等待其他锁。granted 为“t”，表示 INSERT 语句是持有锁。nodename，表示锁产生在的位置，即 CN 或 DN 位置，例如 cn_5001，继续执行步骤 2。
- 如果回显为 0 rows，则表示不存在锁冲突。继续执行步骤 7。

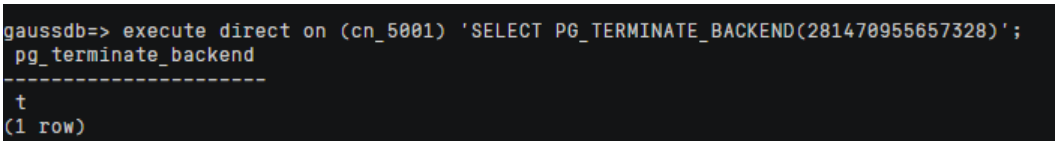
```
SELECT * FROM pgxc_lock_conflicts;
```



步骤 2 据语句内容确认是中止持锁语句还是另找时间做 VACUUM FULL。如果终止，则执行以下语句。*pid* 从步骤 1 获取，cn_5001 为上面查询到的 nodename。

中止结束后，再尝试重新执行 VACUUM FULL。

```
execute direct on (cn_5001) 'SELECT PG_TERMINATE_BACKEND(pid)';
```

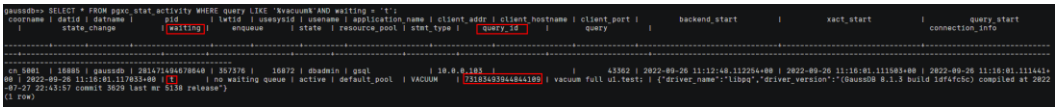


----结束

- **8.0.x 及以前版本参见如下方法：**

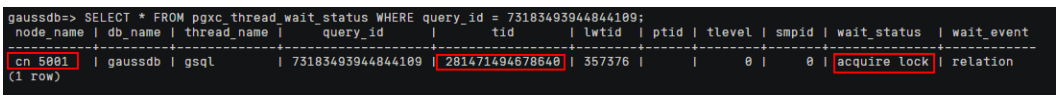
步骤 1 常见的原因是存在锁等待，先排查此原因。在数据库中执行以下语句，获取 VACUUM FULL 操作对应的 query_id。

```
SELECT * FROM pgxc_stat_activity WHERE query LIKE '%vacuum%'AND waiting = 't';
```



步骤 2 根据获取的 query_id，执行以下语句查看是否存在锁等待。其中，*{query_id}*从步骤 1 获取。

```
SELECT * FROM pgxc_thread_wait_status WHERE query_id = {query_id};
```



- 回显中“wait_status”存在“acquire lock”表示存在锁等待。同时查看“node_name”显示在对应的 CN 或 DN 上存在锁等待，记录相应的 CN 或 DN 名称，例如 cn_5001 或 dn_600x_600y，继续执行步骤 3。

- 回显中“wait_status”不存在“acquire lock”，排除锁等待情况，执行步骤 7。

步骤 3 执行以下语句，到等锁的对应 CN 或 DN 上从 pg_locks 中查看 VACUUM FULL 操作在等待哪个锁。以下以 cn_5001 为例，如果在 DN 上等锁，则改为相应的 DN 名称。pid 为步骤 2 获取的 tid。

回显中记录 relation 的值。

```
execute direct on (cn_5001) 'SELECT * FROM pg_locks WHERE pid = {tid} AND granted = ''f''';
```

```
gaussdb=> execute direct on (cn_5001) 'SELECT * FROM pg_locks WHERE pid = 281471494678640 AND granted = ''f''';
 locktype | database | relation | page | tuple | virtualxid | transactionid | classid | objid | objsubid | virtualtransaction |       pid       |     mode      | granted | fastpath
----------+----------+----------+------+-------+------------+---------------+---------+-------+----------+--------------------+-----------------+---------------+---------+----------
 relation |    16885 |    25064 |      |       |            |               |         |       |          | 14/3090            | 281471494678640 | ExclusiveLock | f       | f
(1 row)
```

步骤 4 根据获取的 relation，从 pg_locks 中查看当前持有锁的 pid。*{relation}*从步骤 3 获取。

```
execute direct on (cn_5001) 'SELECT * FROM pg_locks WHERE relation = {relation} AND granted = ''t''';
```

```
gaussdb=> execute direct on (cn_5001) 'SELECT * FROM pg_locks WHERE relation = 25064 AND granted = ''t''';
 locktype | database | relation | page | tuple | virtualxid | transactionid | classid | objid | objsubid | virtualtransaction |       pid       |       mode       | granted | fastpath
----------+----------+----------+------+-------+------------+---------------+---------+-------+----------+--------------------+-----------------+------------------+---------+----------
 relation |    16885 |    25064 |      |       |            |               |         |       |          | 17/4647            | 281471060535408 | RowExclusiveLock | t       | f
(1 row)
```

步骤 5 根据 pid，执行以下语句，查到对应的 SQL 语句。*{pid}*从步骤 4 获取。

```
execute direct on (cn_5001) 'SELECT query FROM pg_stat_activity WHERE pid={pid}';
```

```
gaussdb=> execute direct on (cn_5001) 'SELECT query FROM pg_stat_activity WHERE pid=281471060535408';
                query
--------------------------------------
 insert into test values (1, 'lily');
(1 row)
```

步骤 6 据语句内容确认是中止持锁语句还是另找时间做 VACUUM FULL。如果终止，则执行以下语句。*pid* 从步骤 4 获取。

中止结束后，再尝试重新执行 VACUUM FULL。

```
execute direct on (cn_5001) 'SELECT PG_TERMINATE_BACKEND(pid)';
```

```
gaussdb=> execute direct on (cn_5001) 'SELECT PG_TERMINATE_BACKEND(281471060535408)';
 pg_terminate_backend
----------------------
 t
(1 row)
```

步骤 7 如果排查了锁等待，则执行一个简单的建表语句，如果很慢说明可能存在 IO/网络问题，可以进一步排查 IO 和网络情况。

步骤 8 如果非 IO/网络问题，可尝试对空表进行 VACUUM FULL 查看执行速度。

由于 VACUUM FULL 任意表都会扫描 pg_class，pg_partition，pg_proc 这三张系统表，如果 VACUUM FULL 空表也比较慢，则为系统表过大导致。

步骤 9 排除以上情况外，当存在 PCK 时，表做 VACUUM FULL 会进行全排序，此时如果表较大或 psort_work_mem 设置较小，就会导致 PCK 排序时产生下盘，进行外排，效率急剧下降。可以通过调大 psort_work_mem 进行规避。

1. 执行以下语句查看表定义。回显中存在“PARTIAL CLUSTER KEY”信息，表示存在 PCK。

   ```
   SELECT * FROM pg_get_tabledef('table name');
   ```

```
postgres=> select * from pg_get_tabledef('customer_t1');
                        pg_get_tabledef
-----------------------------------------------------------------------
 SET search_path = dbadmin;                                           +
 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)                        +
 DISTRIBUTE BY HASH(c_last_name)                                      +
 TO GROUP group_version1;                                             +
 ALTER TABLE customer_t1 ADD PARTIAL CLUSTER KEY (c_customer_sk);
(1 row)
```

2. 登录 DWS 管理控制台，左侧选择“集群管理”。
3. 单击对应的集群名称，进入集群详情。
4. 左侧选择“参数修改”，在搜索栏中输入 psort_work_mem 进行搜索，将 CN 参数值和 DN 参数值同时调大，单击“保存”。

5. 等待集群重启生效后，重新进行 VACUUM FULL 操作。
6. 如果以上方法仍无法解决，请联系技术支持。

----结束

6.11 集群报错内存溢出

问题现象

查看日志提示：

```
[ERROR] Mpp task queryDataAnalyseById or updateDataAnalyseHistoryEndTimesAndResult
fail, dataAnalyseId:17615 org.postgresql.util.PSQLException: ERROR: memory is
temporarily unavailable
sql: vacuum full dws_customer_360.t_user_resource;
```

原因分析

存在部分 SQL 语句使用内存资源过多，造成内存资源耗尽，其余语句执行作业时无法分配到内存就提示内存不足。

处理方法

查询当前集群的内存使用情况，找到内存使用过高的语句并及时终止，释放资源之后集群内存就会恢复。

- **8.1.1 及之前集群版本连接数据库后执行以下步骤：**

步骤 1 执行以下语句查询当前集群的内存使用情况，观察是否有实例的 dynamic_used_memory 已经大于或者接近于该实例的 max_dynamic_memory，出现上述报错，一般为 dynamic_used_memory 达到上限。

```
select * from pgxc_total_memory_detail;
```

步骤 2 开启 topsql 的情况下，使用实时 topsql 查询正在执行的高内存 query 语句，根据结果中的 max_peak_memory 以及 memory_skew_percent 值，较大的值就是消耗内存较多的语句。

```
select
nodename,pid,dbname,username,application_name,min_peak_memory,max_peak_memory,avera
ge_peak_memory,memory_skew_percent,substr(query,0,50) as query from
pgxc_wlm_session_statistics;
```

步骤 3 根据步骤 2 查询的会话信息，通过执行 pg_terminate_backend 函数结束相应会话，即可恢复内存。恢复后可根据实际业务情况重新优化该 SQL 语句。

```
SELECT pg_terminate_backend(pid);
```

----结束

- **8.1.2 及以上集群版本支持登录 GaussDB(DWS) 管理控制台，在实时查询监控页面执行以下步骤：**

须知

- 实时查询仅 8.1.2 及以上集群版本支持。
- 启动实时查询功能需要在“监控设置>监控采集”页面打开“实时查询监控”指标项。

步骤 1 登录 GaussDB(DWS) 管理控制台，在“集群管理”页面，找到需要查看监控的集群，在集群所在行的“操作”列，单击“监控面板”，系统将显示数据库监控页面。

步骤 2 在左侧导航栏选择“监控>实时查询”，进入实时查询监控页面。

步骤 3 根据选择的指定时间段浏览集群中正在运行的所有查询信息。

步骤 4 单击指定实时查询监控的会话查询 ID，进入该会话 ID 的实时查询的详情页面，在详情页面中会展示当前监控的详细内容。例如用户名称，数据库名称，执行时间，查询语句，查询状态，排队状态，dn 最小内存峰值，dn 最大内存峰值，dn 每秒最大 io 峰值，dn 每秒最小 io 峰值和内存使用平均值等信息。

根据“dn 最大内存峰值”和“内存使用平均值”查询结果，值较大的就是消耗内存较多的语句。

步骤 5 若已确认所查询的占用内存高的语句需要被终止，勾选指定的查询 ID 后，单击“终止查询”按钮，终止查询。

若设置了细粒度权限控制功能，只有配置了操作权限的用户才能使用终止查询按钮。只读权限用户登入后终止查询按钮为灰色。

----结束

6.12 带自定义函数的语句不下推

问题现象

SQL 语句不下推。

原因分析

目前最新版本可以支持绝大多数常用函数的下推，不下推函数的场景主要出现在自定义函数属性定义错误的场景。

不下推语句的执行方式没有利用分布式的优势，其在执行过程中，相当于把大量的数据和计算过程汇集到一个节点上去做，因此性能往往非常差。

分析过程

步骤 1 通过 explain verbose 打印语句执行计划。

```
tpcds1xcpm=#  explain verbose
 select func_add_sql(sr_customer_sk,sr_store_sk )
 ,sum(SR_FEE) as ctr_total_return
 from store_returns
 ,date_dim
 where sr_returned_date_sk = d_date_sk
 and d_year =2000
 group by 1;
                                                              QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=1820.37..2419.67 rows=47944 width=46)
   Output: ((store_returns.sr_customer_sk + store_returns.sr_store_sk)), sum(store_returns.sr_fee)
   Group By Key: (store_returns.sr_customer_sk + store_returns.sr_store_sk)
   ->  Hash Join  (cost=4.56..1580.65 rows=47944 width=14)
         Output: (store_returns.sr_customer_sk + store_returns.sr_store_sk), store_returns.sr_fee
         Hash Cond: (store_returns.sr_returned_date_sk = date_dim.d_date_sk)
         ->  Data Node Scan on store_returns "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00 rows=287514 distinct=1993 width=18)
               Output: store_returns.sr_customer_sk, store_returns.sr_store_sk, store_returns.sr_fee, store_returns.sr_returned_date_sk
               Node/s: All datanodes
               Remote query: SELECT sr_customer_sk, sr_store_sk, sr_fee, sr_returned_date_sk FROM ONLY public.store_returns WHERE true
         ->  Hash  (cost=0.00..0.00 rows=365 distinct=365 width=4)
               Output: date_dim.d_date_sk
               ->  Data Node Scan on date_dim "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00 rows=365 width=4)
                     Output: date_dim.d_date_sk
                     Node/s: All datanodes
                     Remote query: SELECT d_date_sk FROM ONLY public.date_dim WHERE d_year = 2000
(16 rows)
```

上述执行计划中有__REMOTE 关键字，这就表明当前的语句是不下推执行的。

步骤 2 不下推语句在 pg_log 中会打印不下推的原因，上述语句在 CN 的日志中会找到类似以下的日志。

```
2020-11-07 17:20:28.894 CST zyl tpcds1xcpm [local] 140573226825472 0 [BACKEND] LOG:  SQL can't be shipped, reason: Function func_add_sql() can not be shipped
2020-11-07 17:20:28.894 CST zyl tpcds1xcpm [local] 140573226825472 0 [BACKEND] STATEMENT:  explain verbose
        select func_add_sql(sr_customer_sk,sr_store_sk )
        ,sum(SR_FEE) as ctr_total_return
        from store_returns
        ,date_dim
        where sr_returned_date_sk = d_date_sk
        and d_year =2000
        group by 1;
```

----结束

处理方法

步骤 1 审视用户自定义函数的 provolatile 属性是否定义正确。如果定义不正确，要修改对应的属性，使它能够下推执行。

具体判断方法可以参考如下说明：

- 函数相关的所有属性都在系统表中可以查到，与函数能否下推相关的两个属性是 provolatile 和 proshippable。
- 其中 provolatile 的本质含义是描述函数是 IMMUTABLE/STABLE/VOLATILE 的。举例如下：
  - 如果一个函数对于同样的输入，一定有相同的输出，那么这类函数就是 IMMUTABLE 的，例如绝大部分的字符串处理函数，这类函数始终可以下推。
  - 如果一个函数的返回结果在一个 SQL 语句的调用过程中，结果是相同的，那么他就是 STABLE 的。例如时间相关的处理函数，它的最终显示结果可能与具体的 GUC 参数相关（例如控制时间显示格式的参数），这类函数都是 STABLE 的，此类函数仅当其属性是 SHIPPABLE 的时候，才能下推。
  - 如果一个函数的返回结果可能随着每一次的调用而返回不同的结果。例如 nextval，random 这种函数，每次调用结果都是不可预期的，这类函数就是 VOLATILE 的，此类函数仅当其属性是 SHIPPABLE 的时候，才能下推。

----结束

6.13 列存表更新失败或多次更新后出现表膨胀

问题现象

- 对列存表更新或 update 会失败。
- 多次对列存表 update，发现表大小膨胀了十多倍。

原因分析

- 列存表不支持并发更新。
- 列存表的更新操作，旧记录空间不会回收。

处理方法

- **方法一**

步骤 1 登录 GaussDB(DWS) 管理控制台。

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“指定目标”，智能运维将在指定时间窗内，自动下发表级 Vacuum 任务。

  用户可配置需要 Vacuum 的列存表，其中一行对应一张表，每张表以数据库名、模式名、表名表示，以空格进行分割。

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”，GaussDB(DWS)将自动在自定义时间窗内执行 Vacuum。

步骤 7 确认无误后，单击“下一步：配置确认”，完成配置。

----结束

- **方法二**

对列存表更新操作后，需要进行 VACUUM FULL 清理，更多用法请参见《数据仓库服务开发指南》的“VACUUM”章节。

```
VACUUM FULL table_name;
```

6.14 列存表多次插入后出现表膨胀

问题现象

列存表多次执行 INSERT 后，发现表膨胀。

原因分析

列存表数据按列存储，一列的每 60000 行存储为一个 CU，同一列的 CU 连续存储在一个文件中，当该文件大于 1GB 时，切换到新文件中。CU 文件数据不能更改只能追加写。对频繁进行删除和更新的列存表 VACUUM 后，由于列存表的 CU 无法更改，即使标识为可用的空间也是无法进行复用的 (复用需要更改 CU) 。因此不建议在 GaussDB(DWS)中对列存表频繁进行删除和更新 。

处理方法

建议开启列存表的 delta 表功能。

```
ALTER TABLE table_name SET (ENABLE_DELTA = ON);
```

说明

- 开启列存表的 delta 表功能，在导入单条或者小规模数据进入表中时，能够防止小 CU 的产生，所以开启 delta 表能够带来显著的性能提升，例如在 3CN、6DN 的集群上操作，每次导入 100 条数据，导入时间能减少 25%，存储空间减少 97%，所以在需要多次插入小批量数据前应该先开启 delta 表，等到确定接下来没有小批量数据导入了再关闭。
- delta 表就是列存表附带的行存表，那么将数据插入 delta 表后将失去列存表的高压缩比等优势，正常情况下使用列存表的场景都是大批量数据导入，所以默认关闭 delta 表，如果开启 delta 表做大批量数据导入，反而会额外消耗更多时间和空间，同样在 3CN、6DN 的集群上操作，每次导入 10000 条数据时，开启 delta 表会比不开启时慢 4 倍，额外消耗 10 倍以上的空间。所以开启 delta 表需谨慎，根据实际业务需要来选择开启和关闭。

6.15 执行计划中有 SubPlan 关键字导致 SQL 性能差

问题现象

用户的 SQL 性能差，执行计划中有 SubPlan 的关键字。

原因分析

执行计划中有 SubPlan 的语句往往性能比较差，这是因为引用 SubPlan 结果的算子可能需要反复的调用获取这个 SubPlan 的值，即 SubPlan 以下的结果要重复执行很多次。

分析过程

步骤 1 执行计划中有 SubPlan，这类语句的性能往往比较差。

----结束

处理方法

步骤 1 这类问题通常通过改写 SQL 来规避。往往这种场景的 SQL 语句的改写是比较困难，而且很容易出现改写后的结果不一致问题。

由于我们在比较高的版本上已经支持了很多场景下的 SubPlan 的自动转化为 join 的操作，因此一种比较方便的思路是打印他在高版本下的执行计划(explain verbose)，然后根据 explain verbose 演绎出来改写后的 SQL 语句。

以上述为例，他在高版本的执行计划如下：

```
tpcds1xcpm=#  explain verbose
tpcds1xcpm-#  select
tpcds1xcpm-#      1,
tpcds1xcpm-#      (select count(*) from customer_address a4 where a4.ca_address_sk = a.ca_address_sk) as GZCS
tpcds1xcpm-# from customer_address a;
 id |                          operation                          | E-rows | E-distinct | E-memory | E-width | E-costs
----+-------------------------------------------------------------+--------+------------+----------+---------+---------
  1 | ->  Row Adapter                                             |  50000 |            |          |       8 | 4810.75
  2 |    ->  Vector Streaming (type: GATHER)                      |  50000 |            |          |       8 | 4810.75
  3 |       ->  Vector Hash Right Join (4, 6)                     |  50000 | 200, 12500 | 16MB     |       8 | 2857.62
  4 |          ->  Vector Sonic Hash Aggregate                    |  50000 |            | 16MB     |      12 | 1296.00
  5 |             ->  CStore Scan on public.customer_address a4 |  50000 |            | 1MB      |       4 | 1108.50
  6 |          ->  CStore Scan on public.customer_address a       |  50000 |            | 1MB      |       4 | 1108.50
(6 rows)

      Predicate Information (identified by plan id)
-----------------------------------------------------------
   3 --Vector Hash Right Join (4, 6)
         Hash Cond: (a4.ca_address_sk = a.ca_address_sk)
(2 rows)

 Targetlist Information (identified by plan id)
------------------------------------------------
   1 --Row Adapter
         Output: 1, (COALESCE((count(*)), 0))
   2 --Vector Streaming (type: GATHER)
         Output: 1, (COALESCE((count(*)), 0))
         Node/s: All datanodes
   3 --Vector Hash Right Join (4, 6)
         Output: 1, COALESCE((count(*)), 0)
   4 --Vector Sonic Hash Aggregate
         Output: count(*), a4.ca_address_sk
         Group By Key: a4.ca_address_sk
   5 --CStore Scan on public.customer_address a4
         Output: a4.ca_address_sk
         Distribute Key: a4.ca_address_sk
   6 --CStore Scan on public.customer_address a
         Output: a.ca_address_sk
         Distribute Key: a.ca_address_sk
(16 rows)
```

步骤 2 根据上述信息，SQL 语句可以改写为：

```
explain verbose
select 1, COALESCE(a4.c1, 0)
from
(select
count(*) c1, a4.ca_address_sk
from public.customer_address a4
group by a4.ca_address_sk ) a4
right join
public.customer_address a
on a4.ca_address_sk = a.ca_address_sk;
```

步骤 3 为了确认改写后的语句与原来的语句是等价的，可以再次打印改写后的执行计划，对比：

```
 id |                        operation                        | E-rows | E-distinct | E-memory | E-width | E-costs
----+---------------------------------------------------------+--------+------------+----------+---------+---------
  1 | ->  Row Adapter                                         |  50000 |            |          |       8 | 4810.75
  2 |    ->  Vector Streaming (type: GATHER)                  |  50000 |            |          |       8 | 4810.75
  3 |       ->  Vector Hash Right Join (4, 6)                 |  50000 | 200, 12500 | 16MB     |       8 | 2857.62
  4 |          ->  Vector Sonic Hash Aggregate                |  50000 |            | 16MB     |      12 | 1296.00
  5 |             ->  CStore Scan on public.customer_address a4 |  50000 |            | 1MB      |       4 | 1108.50
  6 |          ->  CStore Scan on public.customer_address a   |  50000 |            | 1MB      |       4 | 1108.50
(6 rows)

      Predicate Information (identified by plan id)
-------------------------------------------------------------
   3 --Vector Hash Right Join (4, 6)
         Hash Cond: (a4.ca_address_sk = a.ca_address_sk)
(2 rows)

    Targetlist Information (identified by plan id)
-----------------------------------------------------
   1 --Row Adapter
         Output: 1, (COALESCE((count(*)), 0::bigint))
   2 --Vector Streaming (type: GATHER)
         Output: 1, (COALESCE((count(*)), 0::bigint))
         Node/s: All datanodes
   3 --Vector Hash Right Join (4, 6)
         Output: 1, COALESCE((count(*)), 0::bigint)
   4 --Vector Sonic Hash Aggregate
         Output: count(*), a4.ca_address_sk
         Group By Key: a4.ca_address_sk
   5 --CStore Scan on public.customer_address a4
         Output: a4.ca_address_sk
         Distribute Key: a4.ca_address_sk
   6 --CStore Scan on public.customer_address a
         Output: a.ca_address_sk
         Distribute Key: a.ca_address_sk
```

----结束

6.16 往 GaussDB(DWS) 写数据慢，客户端数据会有积压

问题现象

客户端往 GaussDB(DWS) 写入数据较慢，客户端数据会有积压。

原因分析

如果通过单条 INSERT INTO 语句的方式单并发写数据入库，客户端很可能会出现瓶颈。INSERT 是最简单的一种数据写入方式，适合数据写入量不大，并发度不高的场景。

处理方法

如果遇到写数据慢的问题，建议通过以下两种方式进行处理：

- 建议选择其他更加高效的数据导入方式，例如使用 COPY 方式导入数据。
  有关导入方式的详细信息，请参见《数据仓库服务开发指南》中的“导入数据 > 导入方式说明”。
- 增大客户端并发数。

6.17 列存表有 PCK 时执行 vacuum full 特别慢

问题现象

某一张业务表执行 vacuum full 时特别慢，其他表都正常。

原因分析

查看该表的定义，发现该表有 PCK（局部聚簇），查看 psort_work_mem 参数，仅为 512M；

处理方法

列存表进行 vacuum full 时，如果有 PCK，就会将 PARTIAL_CLUSTER_ROWS 中多条记录全都加载到内存中再进行排序，如果 psort_work_mem 不足以将 PARTIAL_CLUSTER_ROWS 这么多条记录全都加载到内存，就会产生下盘（数据库选择将临时结果暂存到磁盘），进行外部排序；一旦进行外部排序，时间消耗就会增加很多。

根据表中数据的 tuple length，合理调整 PARTIAL_CLUSTER_ROWS 和 psort_work_mem 的大小。

6.18 分析查询效率异常降低的问题

通常在几十毫秒内完成的查询，有时会突然需要几秒的时间完成；而通常需要几秒完成的查询，有时需要半小时才能完成。如何分析这种查询效率异常降低的问题呢？

解决办法

通过下列的操作步骤，可以分析出查询效率异常降低的原因。

步骤 1 使用 analyze 命令分析数据库。

analyze 命令更新所有表中数据大小以及属性等相关统计信息，该命令较为轻量级，可以经常执行。如果此命令执行后性能恢复或者有所提升，则表明 autovacuum 未能很好的完成它的工作，有待进一步分析。

步骤 2 检查查询语句是否返回了多余的数据信息。

例如，如果查询语句先查询一个表中所有的记录，而只用到结果中的前 10 条记录。对于包含 50 条记录的表，查询起来是很快的；但是，当表中包含的记录达到 50000，查询效率将会有所下降。

若业务应用中存在只需要部分数据信息，但是查询语句却是返回所有信息的情况，建议修改查询语句，增加 LIMIT 子句来限制返回的记录数。这样至少使数据库优化器有了一定的优化空间，一定程度上会提升查询效率。

步骤 3 检查查询语句单独运行时是否仍然较慢。

尝试在数据库没有其他查询或查询较少的时候运行查询语句，并观察运行效率。如果效率较高，则说明可能是由于之前运行数据库系统的主机负载过大导致查询低效。此外，还可能是由于执行计划比较低效，但是由于主机硬件较快使得查询效率较高。

步骤 4 检查重复相同查询语句的执行效率。

查询效率低的一个重要原因是查询所需信息没有缓存在内存中，这可能是由于内存资源紧张，缓存信息被其他查询处理覆盖。

重复执行相同的查询语句，如果后续执行的查询语句效率提升，则可能是由于上述原因导致。

----结束

6.19 未收集统计信息导致查询性能差

问题现象

SQL 查询性能差，对语句执行 explain verbose 时有 Warning 信息。

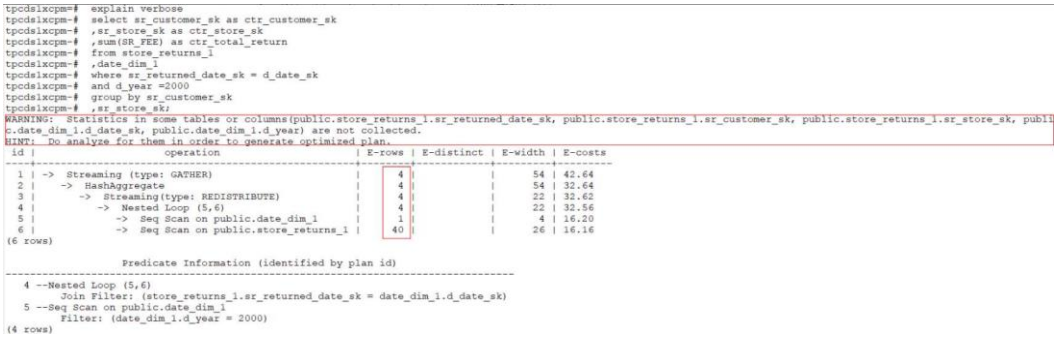
原因分析

查询中涉及到的表或列没有收集统计信息。统计信息是优化器生成执行计划的基础，没有收集统计信息，优化器生成的执行计划会非常差，如果统计信息未收集，会导致多种多样表现形式的性能问题。例如，等值关联走 NestLoop，大表 broadcast，集群 CPU 持续增高等等问题。

分析过程

步骤 1 通过 explain verbose/explain performance 打印语句的执行计划。

执行计划中会有语句未收集统计信息的告警，并且通常 E-rows 估算非常小。



步骤 2 上述例子中，在打印的执行计划中有 Warning 提示信息，提示有哪些列在这个执行计划中用到了，但是这些列没有统计信息。

在 CN 的 pg_log 日志中也有会有类似的 Warning 信息。同时，E-rows 会比实际值小很多。

----结束

处理方法

周期性地运行 analyze，或者在对表的大部分内容做了更改之后马上执行 analyze。

6.20 执行计划中有 NestLoop 导致 SQL 语句执行慢(not in 和 not exists)

问题现象

客户的 SQL 语句执行慢，执行计划中有 NestLoop。

原因分析

- NestLoop 是导致语句性能慢的主要原因。
- Hashjoin 只能做等值关联，NestLoop 的条件中有 or 条件，所以无法用 Hashjoin 求解。
- 导致出现这个现象的原因是由 not in 的语义决定的(具体可以参考外网关于 not in 和 not exists 的介绍)。

分析过程

步骤 1 通过 explain verbose 打印语句执行计划，观察 SQL 语句中有 not in 语法。

```
explain verbose
select sr_customer_sk as ctr_customer_sk
,sr_store_sk as ctr_store_sk
,sum(SR_FEE) as ctr_total_return
from store_returns
where sr_returned_date_sk not in ( select d_date_sk from date_dim where d_year = 2000)
group by sr_customer_sk
,sr_store_sk;
```

步骤 2 执行计划中有 NestLoop。

```
 id |                                operation                                | E-rows | E-distinct | E-width |  E-costs
----+-------------------------------------------------------------------------+--------+------------+---------+-----------
  1 | ->  Row Adapter                                                         | 218254 |            |      46 | 310182.72
  2 |    ->  Vector Streaming (type: GATHER)                                  | 218254 |            |      46 | 310182.72
  3 |       ->  Vector Hash Aggregate                                         | 218254 |            |      46 | 309792.10
  4 |          ->  Vector Streaming(type: REDISTRIBUTE)                       | 218254 |            |      14 | 308837.23
  5 |             ->  Vector Nest Loop Anti Join (6, 8)                       | 218254 |            |      14 | 307372.95
  6 |                ->  Vector Partition Iterator                            | 287514 |            |      18 | 2249.88
  7 |                   ->  Partitioned CStore Scan on public.store_returns   | 287514 |            |      18 | 2249.88
  8 |                ->  Vector Materialize                                   |   1460 |            |       4 | 966.86
  9 |                   ->  Vector Streaming(type: BROADCAST)                 |   1460 |            |       4 | 965.03
 10 |                      ->  Vector Partition Iterator                      |    365 |            |       4 | 928.92
 11 |                         ->  Partitioned CStore Scan on public.date_dim  |    365 |            |       4 | 928.92
(11 rows)

                                Predicate Information (identified by plan id)
-------------------------------------------------------------------------------------------------------------------------------
  5 --Vector Nest Loop Anti Join (6, 8)
       Join Filter: ((store_returns.sr_returned_date_sk = date_dim.d_date_sk) OR (store_returns.sr_returned_date_sk IS NULL))
  6 --Vector Partition Iterator
       Iterations: 7
  7 --Partitioned CStore Scan on public.store_returns
       Selected Partitions:  1..7
 10 --Vector Partition Iterator
       Iterations: 1
 11 --Partitioned CStore Scan on public.date_dim
       Filter: (date_dim.d_year = 2000)
       Selected Partitions:  3
(11 rows)
```

----结束

处理方法

步骤 1 大多数场景下，客户需要的结果集其实是可以通过 not exists 获得的，因此上述语句可以通过修改将 not in 修改为 not exists。

```
tpcds1xcpm=#  explain verbose
tpcds1xcpm-#  select sr_customer_sk as ctr_customer_sk
tpcds1xcpm-#  ,sr_store_sk as ctr_store_sk
tpcds1xcpm-#  ,sum(SR_FEE) as ctr_total_return
tpcds1xcpm-#  from store_returns
tpcds1xcpm-#  where  not exists ( select 1 from date_dim where d_date_sk =sr_returned_date_sk and  d_year = 2000)
tpcds1xcpm-#  group by sr_customer_sk
tpcds1xcpm-#  ,sr_store_sk;
 id |                                operation                                | E-rows | E-distinct | E-width | E-costs
----+-------------------------------------------------------------------------+--------+------------+---------+---------
  1 | ->  Row Adapter                                                         | 239700 |            |      46 | 7068.30
  2 |    ->  Vector Streaming (type: GATHER)                                  | 239700 |            |      46 | 7068.30
  3 |       ->  Vector Hash Aggregate                                         | 239700 |            |      46 | 6677.68
  4 |          ->  Vector Streaming(type: REDISTRIBUTE)                       | 239701 |            |      14 | 5628.99
  5 |             ->  Vector Hash Anti Join (6, 8)                            | 239701 |            |      14 | 4020.85
  6 |                ->  Vector Partition Iterator                            | 287514 |       1990 |      18 | 2249.88
  7 |                   ->  Partitioned CStore Scan on public.store_returns   | 287514 |            |      18 | 2249.88
  8 |                ->  Vector Streaming(type: BROADCAST)                    |   1460 |        365 |       4 | 965.03
  9 |                   ->  Vector Partition Iterator                         |    365 |            |       4 | 928.92
 10 |                      ->  Partitioned CStore Scan on public.date_dim     |    365 |            |       4 | 928.92
(10 rows)

                Predicate Information (identified by plan id)
-------------------------------------------------------------------------------
  5 --Vector Hash Anti Join (6, 8)
       Hash Cond: (store_returns.sr_returned_date_sk = date_dim.d_date_sk)
  6 --Vector Partition Iterator
       Iterations: 7
  7 --Partitioned CStore Scan on public.store_returns
       Selected Partitions:  1..7
  9 --Vector Partition Iterator
       Iterations: 1
 10 --Partitioned CStore Scan on public.date_dim
       Filter: (date_dim.d_year = 2000)
       Selected Partitions:  3
(11 rows)
```


----结束

6.21 未分区剪枝导致 SQL 查询慢

问题现象

三条 sql 查询慢，查询的分区表总共 185 亿条数据，查询条件中没有涉及分区键。

```
select passtime from table where passtime<'2020-02-19 15:28:14' and passtime>'2020-02-18 15:28:37' order by passtime desc limit 10;
select max(passtime) from table where passtime<'2020-02-19 15:28:14' and passtime>'2020-02-18 15:28:37';
```

列存表，分区键为 createtime，哈希分布键为 motorvehicleid。

原因分析

慢 sql 过滤条件中未涉及分区字段，导致执行计划未分区剪枝，走了全表扫描，性能严重裂化。

分析过程

步骤 1 和客户确认部分业务慢，慢的业务中都涉及到了同一张表 tb_motor_vehicle。

步骤 2 收集几个典型的慢 sql，分别打印执行计划。从执行计划中可以看出来，两条 sql 的耗时都集中在 Partitioned CStore Scan on public.tb_motor_vehicle 列存表的分区扫描上。

```
id |                                operation                                |        A-time         | A-rows | E-rows | E-distinct |  Peak Memory   | E-memory
---+-------------------------------------------------------------------------+-----------------------+--------+--------+------------+----------------+----------
 1 | ->  Row Adapter                                                         | 658657.403            |      1 |      1 |            | 10KB           |
 2 |   ->  Vector Aggregate                                                  | 658657.395            |      1 |      1 |            | 176KB          |
 3 |     ->  Vector Streaming (type: GATHER)                                 | 658657.203            |     24 |     24 |            | 157KB          |
 4 |       ->  Vector Aggregate                                              | [293523.439,657533.737] |     24 |     24 |            | [176KB, 176KB] | 1MB
 5 |         ->  Vector Partition Iterator                                   | [292832.499,656876.182] | 58042877 |    864 |            | [17KB, 17KB]   | 1MB
 6 |           ->  Partitioned CStore Scan on public.tb_motor_vehicle        | [290866.340,652888.555] | 58042877 |    864 |            | [1MB, 1MB]     | 1MB
(6 rows)

                                                  Predicate Information (identified by plan id)
--------------------------------------------------------------------------------------------------------------------------------------------------------------
  5 --Vector Partition Iterator
      Iterations: 398
  6 --Partitioned CStore Scan on public.tb_motor_vehicle
      Filter: ((tb_motor_vehicle.passtime < '2020-02-19 15:28:14'::timestamp without time zone) AND (tb_motor_vehicle.passtime > '2020-02-18 15:28:37'::timestam
      Rows Removed by Filter: 1638714321
      Selected Partitions:  1..398
(6 rows)
```

```
iyd |                                operation                                |        A-time         | A-rows | E-rows | E-distinct |  Peak Memory   | E-memory
---+-------------------------------------------------------------------------+-----------------------+--------+--------+------------+----------------+----------
 1 | ->  Row Adapter                                                         | 712811.232            |     10 |     10 |            | 10KB           |
 2 |   ->  Vector Limit                                                      | 712811.206            |     10 |     10 |            | 1KB            |
 3 |     ->  Vector Streaming (type: GATHER)                                 | 712811.200            |    240 |    240 |            | 672KB          |
 4 |       ->  Vector Limit                                                  | [365578.263,712268.299] |    240 |    240 |            | [1KB, 1KB]     | 1MB
 5 |         ->  Vector Sort                                                 | [365578.257,712268.294] |    240 |    864 |            | [254KB, 254KB] | 16MB
 6 |           ->  Vector Partition Iterator                                 | [365216.092,711913.426] | 58042877 |    864 |            | [17KB, 17KB]   | 1MB
 7 |             ->  Partitioned CStore Scan on public.tb_motor_vehicle      | [360943.136,701785.728] | 58042877 |    864 |            | [1MB, 1MB]     | 1MB
(7 rows)

                                                  Predicate Information (identified by plan id)
--------------------------------------------------------------------------------------------------------------------------------------------------------------
  6 --Vector Partition Iterator
      Iterations: 398
  7 --Partitioned CStore Scan on public.tb_motor_vehicle
      Filter: ((tb_motor_vehicle.passtime < '2020-02-19 15:28:14'::timestamp without time zone) AND (tb_motor_vehicle.passtime > '2020-02-18 15:28:37'::timestamp w
      Rows Removed by Filter: 1638678418
      Selected Partitions:  1..398
(6 rows)
```

步骤 3 和客户确认，该表的分区键为 createtime，而涉及到的 sql 中无任何 createtime 的筛选和过滤条件，基本可以确认是由于慢 sql 的计划没有走分区剪枝，导致了全表扫描，对于 185 亿条数据量的表，全表扫描性能会很差。

步骤 4 通过在筛选条件中增加分区键过滤条件，优化后的 sql 和执行计划如下，性能从十几分钟，优化到了 12 秒左右，性能有明显提升。

```
SELECT passtime FROM tb_motor_vehicle WHERE createtime > '2020-02-19 00:00:00' AND
createtime < '2020-02-20 00:00:00' AND passtime > '2020-02-19 00:00:00' AND
passtime < '2020-02-20 00:00:00' ORDER BY passtime DESC LIMIT 10000;
```

```
explain performance SELECT passtime FROM tb_motor_vehicle WHERE createtime > '2020-02-19 00:00:00' AND createtime < '2020-02-20 00:00:00' AND passtime > '2020-02-19 00
ORDER BY passtime DESC LIMIT 10000;
+----+-------------------------------------------------------------------------+-----------------------+--------+--------+------------+----------------+----------+
| id | operation                                                               | A-time                | A-rows | E-rows | E-distinct | Peak Memory    | E-memory |
+----+-------------------------------------------------------------------------+-----------------------+--------+--------+------------+----------------+----------+
|  1 | ->  Row Adapter                                                         | 12285.727             |  10000 |  10000 | NULL       | 10KB           |          |
|  2 |   ->  Vector Limit                                                      | 12284.854             |  10000 |  10000 | NULL       | 1KB            |          |
|  3 |     ->  Vector Streaming (type: GATHER)                                 | 12284.840             |  10000 | 240000 | NULL       | 1464KB         |          |
|  4 |       ->  Vector Limit                                                  | [10300.383,12227.410] | 240000 | 240000 | NULL       | [1KB, 1KB]     | 1MB      |
|  5 |         ->  Vector Sort                                                 | [10300.368,12227.399] | 240000 | 11542800 | NULL     | [962KB, 962KB] | 16MB     |
|  6 |           ->  Vector Partition Iterator                                 | [8461.684,10419.017]  | 57415974 | 11542791 | NULL     | [25KB, 25KB]   | 1MB      |
|  7 |             ->  Partitioned CStore Scan on public.tb_motor_vehicle      | [8389.677,10356.890]  | 57415974 | 11542791 | NULL     | [1MB, 1MB]     | 1MB      |
+----+-------------------------------------------------------------------------+-----------------------+--------+--------+------------+----------------+----------+
7 rows in set
```

----结束

处理方法

在慢 sql 的过滤条件中增加分区筛选条件，避免走全表扫描。

6.22 行数估算过小，优化器选择走 NestLoop 导致性能下降

问题现象

查询语句执行慢，卡住无法返回结果。SQL 语句的特点是 2~3 张表 left join，然后通过 select 查询结果，执行计划如下：

```
id |                                                                operation                                                                | E-rows  | E-distinct | E-memory | E-width |  E-costs
---+-----------------------------------------------------------------------------------------------------------------------------------------+---------+------------+----------+---------+-----------
 1 | ->  Row Adapter                                                                                                                         |       2 |            |          |     771 | 116895.08
 2 |   ->  Vector Streaming (type: GATHER)                                                                                                   |       2 |            |          |     771 | 116895.08
 3 |     -> Vector Nest Loop Left Join (4, 9)                                                                                                |       2 |            | 1MB      |     771 | 91866.92
 4 |       ->  Vector Streaming(type: LOCAL GATHER dop: 1/8)                                                                                 |       1 |            | 16MB     |     118 | 89602.67
 5 |         ->  Vector Hash Aggregate                                                                                                       |       1 |            | 16MB     |      94 | 89602.63
 6 |           ->  Vector Streaming(type: LOCAL REDISTRIBUTE dop: 8/8)                                                                       |       1 |            | 17MB     |      22 | 89602.61
 7 |             ->  Vector Partition Iterator                                                                                               |       1 |            | 1MB      |      22 | 89602.58
 8 |               ->  Partitioned CStore Scan on scm_sdrplus.t58_pppoe_h a                                                                  |       1 |            | 1MB      |      22 | 89602.57
 9 |         ->  Vector Hash Left Join (10, 22)                                                                                              |       2 | 200, 77    | 16MB     |     664 | 2264.24
10 |           -> Vector Nest Loop Left Join (11, 21)                                                                                        |       2 |            | 1MB      |     654 | 2186.18
11 |             ->  Vector Hash Right Join (12, 13)                                                                                         |       2 | 7220, 2256 | 16MB     |     643 | 1987.21
12 |               ->  CStore Scan on scm_sdrplus.ne_location 1                                                                              |   43320 |            | 1MB      |     115 | 1735.22
13 |               ->  Vector Partition Iterator                                                                                             |       2 |            | 1MB      |     555 | 231.10
14 |                 ->  Partitioned CStore Index Scan using user_database_account_idx on scm_sdrplus.user_database a                       |       2 |            | 16MB     |     555 | 231.10
15 |       ->  Row Adapter  [14, InitPlan 1 (returns $1)]                                                                                   |       6 |            | 1MB      |      43 | 25026.85
16 |         ->  Vector Aggregate                                                                                                            |       6 |            | 1MB      |      43 | 25026.85
17 |           ->  Vector Streaming(type: BROADCAST)                                                                                         |      36 |            | 2MB      |      43 | 25026.85
18 |             ->  Vector Aggregate                                                                                                        |       6 |            | 1MB      |      43 | 25026.72
19 |               ->  Vector Partition Iterator                                                                                             | 3330361 |            | 1MB      |      11 | 23639.06
20 |                 ->  Partitioned CStore Scan on scm_sdrplus.user_database a                                                              | 3330361 |            | 1MB      |      11 | 23639.06
21 |           ->  CStore Index Scan using i_mac_oui on scm_sdrplus.mac_oui c                                                                |       6 |            | 1MB      |      21 | 198.96
22 |         ->  CStore Scan on scm_sdrplus.thai_province m                                                                                  |     462 |            | 1MB      |      10 | 77.08
```

原因分析

优化器在选择执行计划时，对结果集评估较小，导致执行计划走了 NestLoop，性能下降。

分析过程

步骤 1 排查当前的 IO，内存，CPU 使用情况，没有发现资源占用高的情况。

步骤 2 查看慢 sql 的线程等待状态。

根据线程等待状态，并没有出现都在等待某个 DN 的情况，初步排除中间结果集偏斜到了同一个 DN 的情况。

```
select * from pg_thread_wait_status where query_id='149181737656737395';
```

```
  node_name   | db_name  | thread_name |      query_id       |       tid       | lwtid | ptid  | tlevel | smpid |          wait_status           | wait_event
--------------+----------+-------------+---------------------+-----------------+-------+-------+--------+-------+--------------------------------+------------
dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140346588657408 | 34622 | 21227 |     17 |     0 | synchronize quit               |
dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140345923331840 | 34624 | 21227 |      4 |     0 | flush data: dn_6001_6002(0)    |
dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140346705573632 | 34626 | 21227 |      4 |     1 | flush data: dn_6001_6002(0)    |
dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140345734571776 | 34630 | 21227 |      4 |     2 | flush data: dn_6001_6002(0)    |
dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140345708902144 | 34631 | 21227 |      4 |     3 | flush data: dn_6001_6002(0)    |
dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140345664861952 | 34634 | 21227 |      4 |     5 | flush data: dn_6001_6002(0)    |
dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140345690023680 | 34633 | 21227 |      4 |     4 | flush data: dn_6001_6002(0)    |
dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140345645983488 | 34636 | 21227 |      4 |     6 | flush data: dn_6001_6002(0)    |
dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140345620297472 | 34637 | 21227 |      4 |     7 | flush data: dn_6001_6002(0)    |
dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140345593034496 | 34640 | 34624 |      6 |     0 | synchronize quit               |
dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140345140049664 | 34641 | 34624 |      6 |     1 | synchronize quit               |
dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140344882620160 | 34643 | 34624 |      6 |     2 | synchronize quit               |
dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140342933812992 | 34645 | 34624 |      6 |     3 | synchronize quit               |
dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140342917031680 | 34646 | 34624 |      6 |     4 | synchronize quit               |
dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140342900250368 | 34648 | 34624 |      6 |     5 | synchronize quit               |
dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140342883469056 | 34649 | 34624 |      6 |     6 | synchronize quit               |
dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140342711473920 | 34652 | 34624 |      6 |     7 | synchronize quit               |
dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140342430512896 | 21227 |       |      0 |     0 | none                           |
cn_5002      | db_sdrplus | gsql        | 149181737656737395 | 140085516814080 | 21777 |       |      0 |     0 | wait node: dn_6011_6012, total 6 |
dn_6003_6004 | db_sdrplus | cn_5002     | 149181737656737395 | 139965483169536 | 34621 |       |      0 |     0 | none                           |
dn_6003_6004 | db_sdrplus | cn_5002     | 149181737656737395 | 139965415552768 | 34623 | 34621 |     17 |     0 | synchronize quit               |
dn_6003_6004 | db_sdrplus | cn_5002     | 149181737656737395 | 139978527995648 | 34625 | 34621 |      4 |     0 | flush data: dn_6003_6004(0)    |
dn_6003_6004 | db_sdrplus | cn_5002     | 149181737656737395 | 139965444912896 | 34627 | 34621 |      4 |     1 | flush data: dn_6003_6004(0)    |
dn_6003_6004 | db_sdrplus | cn_5002     | 149181737656737395 | 139965396674304 | 34628 | 34621 |      4 |     2 | flush data: dn_6003_6004(0)    |
dn_6003_6004 | db_sdrplus | cn_5002     | 149181737656737395 | 139965370988288 | 34629 | 34621 |      4 |     3 | flush data: dn_6003_6004(0)    |
dn_6003_6004 | db_sdrplus | cn_5002     | 149181737656737395 | 139964910663424 | 34632 | 34621 |      4 |     4 | flush data: dn_6003_6004(0)    |
dn_6003_6004 | db_sdrplus | cn_5002     | 149181737656737395 | 139964893882112 | 34635 | 34621 |      4 |     5 | flush data: dn_6003_6004(0)    |
dn_6003_6004 | db_sdrplus | cn_5002     | 149181737656737395 | 139964644316928 | 34638 | 34621 |      4 |     6 | flush data: dn_6003_6004(0)    |
dn_6003_6004 | db_sdrplus | cn_5002     | 149181737656737395 | 139964627535616 | 34639 | 34621 |      4 |     7 | flush data: dn_6003_6004(0)    |
```

步骤 3 联系运维人员登录到相应的实例节点上，打印等待状态为 none 的线程堆栈信息如下。

通过反复打印堆栈信息，发现堆栈在变化，并没有 hang 死，所以初步判断该问题为性能慢的问题。另堆栈中有 VecNestLoopRuntime，结合执行计划，初步判断是由于统计信息不准，优化器评估结果集较少，执行计划走了 NestLoop 导致性能下降。

gstack *14104*

```
Thread 1 (Thread 0x7f4fa20ff700 (LWP 14104)):
#0  0x000055b3faed6b37 in heap_hot_search_buffer(ItemPointerData*, RelationData*, int, SnapshotData*, HeapTupleData*, HeapTupleHeaderData*, bool*, bool) ()
#1  0x000055b3faef0890 in index_fetch_heap(IndexScanDescData*) ()
#2  0x000055b3faef0b21 in index_getnext(IndexScanDescData*, ScanDirection) ()
#3  0x000055b3faeefeb3 in systable_getnext_ordered(SysScanDescData*, ScanDirection) ()
#4  0x000055b3fadef4b1 in CStore::LoadCUDesc(int, LoadCUDescCtl*, bool, SnapshotData*) ()
#5  0x000055b3fadf4728 in CStore::LoadCUDescIfNeed() ()
#6  0x000055b3fadf4fbb in CStore::CStoreScan(CStoreScanState*, VectorBatch*) ()
#7  0x000055b3fb22e569 in ExecCStoreScan(CStoreScanState*) ()
#8  0x000055b3fb230468 in VectorBatch* ExecCstoreIndexScanT<(IndexType)2>(CStoreIndexScanState*) ()
#9  0x000055b3fb23080a in ExecCstoreIndexScan(CStoreIndexScanState*) ()
#10 0x000055b3fb2707b4 in VectorEngine(PlanState*) ()
#11 0x000055b3fb241d11 in VectorBatch* VecNestLoopRuntime::JoinQualT<(JoinType)1, true, false, false>() ()
#12 0x000055b3fb245bc1 in VectorBatch* VecNestLoopRuntime::JoinT<true>() ()
#13 0x000055b3fb2707b4 in VectorEngine(PlanState*) ()
#14 0x000055b3fb2190c8 in HashJoinTbl::probeHashTable(hashSource*) ()
#15 0x000055b3fb2182e5 in ExecVecHashJoin(VecHashJoinState*) ()
#16 0x000055b3fb2707b4 in VectorEngine(PlanState*) ()
#17 0x000055b3fb241d11 in VectorBatch* VecNestLoopRuntime::JoinQualT<(JoinType)1, true, false, false>() ()
#18 0x000055b3fb245bc1 in VectorBatch* VecNestLoopRuntime::JoinT<true>() ()
#19 0x000055b3fb2707b4 in VectorEngine(PlanState*) ()
#20 0x000055b3fb1562a5 in standard_ExecutorRun(QueryDesc*, ScanDirection, long) ()
#21 0x000055b3fb156a9d in ExecutorRun(QueryDesc*, ScanDirection, long) ()
#22 0x000055b3fb6fc0e8 in PortalRunSelect(PortalData*, bool, long, _DestReceiver*) ()
#23 0x000055b3fb6fc860 in PortalRun(PortalData*, long, bool, _DestReceiver*, _DestReceiver*, char*) ()
#24 0x000055b3fb6ededa in exec_simple_plan(PlannedStmt*) ()
#25 0x000055b3fb6f45c0 in PostgresMain(int, char**, char const*, char const*) ()
```

```
#0  0x000055b3fadef618 in CStore::LoadCUDesc(int, LoadCUDescCtl*, bool, SnapshotData*) ()
#1  0x000055b3fadf4728 in CStore::LoadCUDescIfNeed() ()
#2  0x000055b3fadf4fbb in CStore::CStoreScan(CStoreScanState*, VectorBatch*) ()
#3  0x000055b3fb22e569 in ExecCStoreScan(CStoreScanState*) ()
#4  0x000055b3fb230468 in VectorBatch* ExecCstoreIndexScanT<(IndexType)2>(CStoreIndexScanState*) ()
#5  0x000055b3fb23080a in ExecCstoreIndexScan(CStoreIndexScanState*) ()
#6  0x000055b3fb2707b4 in VectorEngine(PlanState*) ()
#7  0x000055b3fb241d11 in VectorBatch* VecNestLoopRuntime::JoinQualT<(JoinType)1, true, false, false>() ()
#8  0x000055b3fb245bc1 in VectorBatch* VecNestLoopRuntime::JoinT<true>() ()
#9  0x000055b3fb2707b4 in VectorEngine(PlanState*) ()
#10 0x000055b3fb2190c8 in HashJoinTbl::probeHashTable(hashSource*) ()
#11 0x000055b3fb2182e5 in ExecVecHashJoin(VecHashJoinState*) ()
#12 0x000055b3fb2707b4 in VectorEngine(PlanState*) ()
#13 0x000055b3fb241d11 in VectorBatch* VecNestLoopRuntime::JoinQualT<(JoinType)1, true, false, false>() ()
#14 0x000055b3fb245bc1 in VectorBatch* VecNestLoopRuntime::JoinT<true>() ()
#15 0x000055b3fb2707b4 in VectorEngine(PlanState*) ()
#16 0x000055b3fb1562a5 in standard_ExecutorRun(QueryDesc*, ScanDirection, long) ()
#17 0x000055b3fb156a9d in ExecutorRun(QueryDesc*, ScanDirection, long) ()
#18 0x000055b3fb6fc0e8 in PortalRunSelect(PortalData*, bool, long, _DestReceiver*) ()
#19 0x000055b3fb6fc860 in PortalRun(PortalData*, long, bool, _DestReceiver*, _DestReceiver*, char*) ()
#20 0x000055b3fb6ededa in exec_simple_plan(PlannedStmt*) ()
#21 0x000055b3fb6f45c0 in PostgresMain(int, char**, char const*, char const*) ()
#22 0x000055b3fb59593e in SubPostmasterMain(int, char**) ()
```

步骤 4 对表执行 analyze 后性能并没有太大改善。

步骤 5 对 sql 增加 hint 关闭索引，让优化器强行走 hashjoin，发现 hint 功能没有生效，原因是 hint 无法改变子查询中的计划。

步骤 6 通过 set enable_indexscan = off，执行计划被改变，走了 Hash Left Join，慢 sql 在 3 秒左右跑出结果，满足客户需求。

id	operation	A-time	A-rows	E-rows	Peak Memory	E-memory	A-width	E-width	E-costs
1	-> Row Adapter	3464.656	357053	6	260KB			768	154118.20
2	-> Vector Streaming (type: GATHER)	3376.601	357053	6	1503KB			768	154118.20
3	-> Vector Streaming(type: LOCAL GATHER dop: 1/8)	[3271.144,3381.566]	357053	6	[789KB,789KB]	16MB		768	129090.04
4	-> Vector Hash Right Join (5, 20)	[3207.552,3371.885]	357053	2	[2MB,2MB]	16MB		768	129089.89
5	-> Vector Streaming(type: LOCAL REDISTRIBUTE dop: 8/1)	[544.447,606.631]	1110265	1110155	[809KB,825KB]	2MB		664	39371.57
6	-> Vector Hash Left Join (7, 19)	[951.633,1195.812]	1110265	1110155	[4MB,4MB]	16MB		664	24171.42
7	-> Vector Hash Left Join (8, 16)	[250.763,270.695]	1110265	1110120	[3MB,3MB]	16MB		653	15628.64
8	-> Vector Partition Iterator	[60.531,69.685]	1110265	1110120	[17KB,17KB]	1MB		555	9256.68
9	-> Partitioned CStore Scan on user_database t2	[58.111,67.092]	1110265	1110120	[3MB,3MB]	1MB		555	9256.68
10	-> Row Adapter [9, InitPlan 1 (returns $1)]	[0.178,0.212]	6	6	[10KB,10KB]	1MB		43	25026.85
11	-> Vector Aggregate	[0.168,0.202]	6	6	[176KB,176KB]	1MB		43	25026.85
12	-> Vector Streaming(type: BROADCAST)	[0.129,0.165]	36	36	[95KB,95KB]	2MB		43	25026.85
13	-> Vector Aggregate	[35.907,70.240]	6	6	[177KB,177KB]	1MB		43	25026.72
14	-> Vector Partition Iterator	[16.292,31.784]	3330361	3330361	[17KB,17KB]	1MB		11	23639.06
15	-> Partitioned CStore Scan on user_database	[13.908,27.591]	3330361	3330361	[580KB,580KB]	1MB		11	23639.06
16	-> Vector Hash Left Join (17, 18)	[115.823,125.157]	43320	43320	[418KB,418KB]	16MB	[304,304]	125	1912.42
17	-> CStore Scan on ne_location 1	[103.422,111.513]	43320	43320	[1MB,1MB]	1MB		115	1735.22
18	-> CStore Scan on thai_province m	[0.385,0.731]	462	462	[219KB,219KB]	1MB	[26,26]	10	77.08
19	-> CStore Scan on mac_oui c	[515.285,755.601]	188418	188418	[478KB,478KB]	1MB	[53,53]	21	1151.40
20	-> Vector Hash Aggregate	[2619.810,2781.678]	357053	1	[2MB,2MB]	16MB	[133,134]	91	89602.63
21	-> Vector Streaming(type: LOCAL REDISTRIBUTE dop: 8/8)	[2338.077,2505.698]	39697152	1	[155KB,155KB]	17MB		19	89602.61
22	-> Vector Partition Iterator	[1633.245,2331.055]	39697152	1	[25KB,25KB]	1MB		19	89602.58
23	-> Partitioned CStore Scan on t58_pppoe_h t1	[1629.094,2325.774]	39697152	1	[1MB,1MB]	1MB			

(23 rows)

----结束

处理方法

通过 **set enable_indexscan = off** 关闭索引功能，让优化器生成的执行计划不走 NestLoop，而走 Hashjoin。

6.23 语句中存在“in 常量”导致 SQL 执行无结果

问题现象

简单的大表过滤的 SQL 语句中有一个“in 常量”的过滤条件，常量的个数非常多(约有 2000 多个)，基表数据量比较大，SQL 语句执行无结果。

原因分析

执行计划中，in 条件还是作为普通的过滤条件存在。这种场景下，最优的执行计划应该是将“in 常量”转化为 join 操作性能更好。

分析过程

步骤 1 打印语句的执行计划：

```
SET
 id |                        operation                        |  E-rows  | E-distinct | E-memory | E-width |  E-c
----+---------------------------------------------------------+----------+------------+----------+---------+------
  1 | ->  Row Adapter                                         | 14586792 |            |          |      44 | 26121
  2 |   ->  Vector Streaming (type: GATHER)                   | 14586792 |            |          |      44 | 26121
  3 |     ->  Vector Streaming(type: LOCAL GATHER dop: 1/24)  | 14586792 |            | 48MB     |      44 | 26073
  4 |       ->  Vector Partition Iterator                     | 14586798 |            | 1MB      |      44 | 26073
  5 |         ->  Partitioned CStore Scan on public.tb_motor_vehicle | 14586798 |            | 1MB      |      44 | 26073
(5 rows)
```

步骤 2 执行计划中，in 条件还是作为普通的过滤条件存在。这种场景下，最优的执行计划应该是将“in 常量”转化为 join 操作性能更好。

----结束

处理方法

步骤 1 可以控制把“in 常量”转 join 的行为，默认是 cost_base 的。如果优化器估算不准，可能会出现需要转化的场景没有做转化，导致性能较差。

步骤 2 这种情况下可以通过设置 qrw_inlist2join_optmode 为 rule_base 来规避解决。

```
set qrw_inlist2join_optmode to rule_base;
```

----结束

6.24 单表点查询性能差

问题现象

单表查询的场景下，客户预期 1s 以内返回结果，实际执行耗时超过 10s。

原因分析

这种场景属于行列存表选择错误导致的问题，这种场景应该使用行存表+btree 索引。

分析过程

步骤 1 通过抓取问题 SQL 的执行信息，发现大部分的耗时都在“CStore Scan”。

```
 id |                           operation                           |        A-time        | A-rows | E-rows | Peak Memory | A-width | E-width | E-costs
----+---------------------------------------------------------------+----------------------+--------+--------+-------------+---------+---------+------------
  1 | ->  Row Adapter                                               | 15168.721            |      1 |      1 | 37KB        |         |      88 | 9662341.28
  2 |    ->  Vector Limit                                           | 15168.715            |      1 |      1 | 2KB         |         |      88 | 9662341.28
  3 |       ->  Vector Aggregate                                    | 15168.711            |      1 |      1 | 629KB       |         |      88 | 9662341.28
  4 |          ->  Vector Streaming (type: GATHER)                  | 15168.140            |     48 |      1 | 319KB       |         |      88 | 9662341.28
  5 |             ->  Vector Aggregate                              | [1769.545,14168.103] |     48 |      1 | [535KB,535KB] |         |      88 | 9662339.75
  6 |                ->  Vector Partition Iterator                  | [1769.516,14150.029] |      0 |      1 | [17KB,17KB] |         |       8 | 9662339.72
  7 |                   ->  Partitioned CStore Scan on sym_saacntxn | [1769.419,14129.592] |      0 |      1 | [2MB,3MB]   |         |       8 | 9662339.72
(7 rows)
```

步骤 2 分析出问题的场景：基表是一张十亿级别的表，每晚有批量增量数据入库，同时会有少量的数据清洗的工作。白天会有高并发的查询操作，查询不涉及表关联，并且返回结果都不大。

----结束

处理方法

步骤 1 调整表定义，表修改为行存表，同时建立 btree 索引，索引建立的原则：

1. 基于充分分析客户 SQL 的背景下去建立索引。
2. 索引要建立的刚刚好，不要有冗余。
3. 建立组合索引时候，要把过滤性比较好的列往前放。
4. 尽可能多的过滤条件都用到索引。

----结束

6.25 动态负载管理下的 CCN 排队

问题现象

业务整体缓慢，只有少量语句在执行，其余业务语句都在排队中（wait in ccn queue）。

原因分析

动态负载管理下，语句会根据估算内存计数排序，例如，最大动态可用内存为 10G(单实例)，语句估算使用内存大小为 5G，这样的语句运行 2 个，其余语句就会等待前两个语句运行完毕才能执行，此时的状态即为 wait in ccn queue。

处理方法

- 场景一：语句估算内存过大，造成排队
  - 查询 pg_session_wlmstat 视图，查看 status 为 running 的语句是否个数很少，而且 statement_mem 字段数值是否较大（单位为 MB，一般认为大于

max_dynamic_memory 1/3 即为大内存语句）。如果都符合就可以判断是此类语句占据内存导致整体运行缓慢。

```
select usename,substr(query,0,20),threadid,status,statement_mem from
pg_session_wlmstat where usename not in (''omm'',''Ruby'') order by
statement_mem,status desc;
```

```
 usename |        substr        |     threadid     | status  | statement_mem
---------+----------------------+------------------+---------+---------------
 dzx     | explain /*Q18*/ perf | 140635882325760 | running |          1288
 dzx     | explain /*Q18*/ perf | 140635599181568 | running |          1288
 dzx     | explain /*Q18*/ perf | 140635978802944 | pending |          1288
 dzx     | explain /*Q18*/ perf | 140635683088128 | pending |          1288
 dzx     | explain /*Q18*/ perf | 140635632744192 | pending |          1288
 dzx     | explain /*Q18*/ perf | 140635615962880 | pending |          1288
 dzx     | explain /*Q18*/ perf | 140635649525504 | pending |          1288
 dzx     | explain /*Q18*/ perf | 140635808921344 | pending |          1288
 dzx     | explain /*Q18*/ perf | 140635582400256 | pending |          1288
 dzx     | explain /*Q18*/ perf | 140635666306816 | pending |          1288
(10 rows)
```

如上图所示，只有最后一个语句是 running 状态，其余语句都是 pending 状态。根据 statement_mem 可以看到该语句占据 2576MB 内存。此时根据语句的 threadid，使用以下语句进行查杀，查杀后即可释放资源，其余语句正常运行。

```
pg_terminate_backend(threadid);
```

- 场景二：所有语句状态都是 pending 状态，没有运行的语句。此时应是管控机制出现异常，直接查杀所有线程，即可恢复正常。

```
select pg_terminate_backend(pid) from pg_stat_activity where usename not in
('omm','Ruby');
```

7 数据库参数修改

7.1 数据库时间与系统时间不一致，如何更改数据库默认时区

问题现象

数据库时间与操作系统不一致，查询 GaussDB(DWS) 数据库默认时间 SYSDATE，结果数据库时间比北京时间慢 8 个小时，导致无法准确定位到更新数据。

原因分析

GaussDB(DWS) 数据库显示和解释时间类型数值时使用的时区默认为“UTC”。如果操作系统时间所设置的时区不是 UTC 时区，就会出现 GaussDB(DWS) 数据库时间和系统时间不一致的情况。通常情况下集群时区不需要进行修改，设置客户端时区可以对 SQL 执行产生影响。

前提条件

建议用户在修改“timezone”参数时在业务低峰期时操作。

处理方法

方法一：更改某个 GaussDB(DWS) 集群的数据库默认时区。

步骤 1 登录 GaussDB(DWS) 管理控制台。

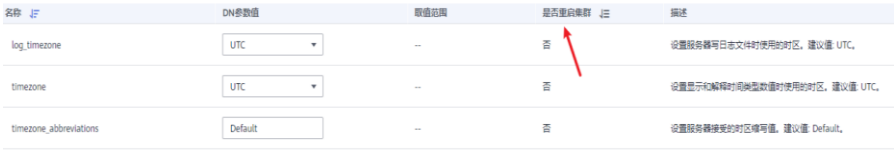
步骤 2 在左侧导航栏中，单击“集群管理”。

步骤 3 在集群列表中找到所需要的集群，单击集群名称，进入集群“基本信息”页面。

步骤 4 单击“参数修改”页签，修改参数“timezone”，修改为您所在的时区，然后单击“保存”。

步骤 5 在“修改预览”窗口，确认修改无误后，单击“保存”。

步骤 6 用户可根据界面中参数“timezone”所在行的“是否重启”列，判断修改参数后无需进行重启操作。



说明

修改“timezone”参数后无需重启集群操作 ，则修改后立即生效。

----结束

方法二：通过后台命令查询和更改数据库时区。

步骤 1 查询客户端时区和当前时间。其中客户端时区为 UTC 时区，now()函数返回当前时间。

```
show time zone;
 TimeZone
----------
 UTC
(1 row)

select now();
              now
-------------------------------
 2022-05-16 06:05:58.711454+00
(1 row)
```

步骤 2 创建数据表，其中 timestamp，timestamptz 是常用的时间类型。timestamp 不保存时区，timestamptz 保存时区。

```
CREATE TABLE timezone_test (id int, t1 timestamp, t2 timestamptz) DISTRIBUTE BY
HASH (id);

\d timezone_test
         Table "public.timezone_test"
 Column |           Type           | Modifiers
--------+--------------------------+-----------
 id     | integer                  |
 t1     | timestamp without time zone |
 t2     | timestamp with time zone    |
```

步骤 3 向 timezone_test 表插入当前时间并查询当前表。

```
insert into timezone_test values (1, now(), now() );
show time zone;
 TimeZone
----------
 UTC
(1 row)
select * from timezone test;
 id |           t1           |            t2
----+----------------------------+-------------------------------
  1 | 2022-05-16 06:10:04.564599 | 2022-05-16 06:10:04.564599+00
(1 row)
```

t1 (timestamp 类型)在保存数据时丢弃了时区信息，t2(timestamptz 类型)保存了时区信息。

步骤 4 把客户端时区设置为东 8 区（UTC-8），再次查询 timezone_test 表。

```
set time zone 'UTC-8';
show time zone;
 TimeZone
----------
 UTC-8
(1 row)
select now();
              now
-------------------------------
 2022-05-16 14:13:43.175416+08
(1 row)
```

步骤 5 继续插入当前时间到 timezone_test 表，并查询。 此时 t1 新插入的值是用的东 8 区时间，t2 根据客户端时区对查询结果进行转换。

```
insert into timezone_test values (2, now(), now() );
select * from timezone_test;
 id |           t1           |            t2
----+----------------------------+-------------------------------
  1 | 2022-05-16 06:10:04.564599 | 2022-05-16 14:10:04.564599+08
  2 | 2022-05-16 14:15:03.715265 | 2022-05-16 14:15:03.715265+08
(2 rows)
```

说明

- timestamp 类型只受数据在插入时的时区影响，查询结果不受客户端时区影响。
- timestamptz 类型在数据插入时记录了时区信息，查询时会根据客户端时区做转换，以客户端时区显示数据。

----结束

7.2 业务报错：Cannot get stream index, maybe comm_max_stream is not enough

问题现象

用户执行业务报错："ERROR: Failed to connect dn_6001_6002, detail:1021 Cannot get stream index, maybe comm_max_stream is not enough.",

原因分析

用户数据库的 comm_max_datanode 参数为默认值 1024，在正常批量业务运行时查到 DN 之间 stream 数量大约为 600~700，当批量任务运行时如果有临时查询，就会超过上限，导致上述报错。

分析过程

1. GUC 参数 comm_max_stream 表示任意两个 DN 之间 stream 的最大数量。

   在 CN 上查询当前任意两个 DN 之间 stream 情况：

   ```
   select node_name,remote_name,count(*) from pgxc_comm_send_stream group by 1,2
   order by 3 desc limit 100;
   ```

   在 DN 上查询当前 DN 与其他 DN 之间 stream 情况：

   ```
   select node_name,remote_name,count(*) from pg_comm_send_stream group by 1,2
   order by 3 desc limit 100;
   ```

2. comm_max_stream 参数值必须大于并发数*每并发平均 stream 算子数*（smp 的平方）。

   该参数默认值为：通过公式 min(query_dop_limit * query_dop_limit * 2 * 20, max_process_memory(字节) * 0.025 /(最大 CN 数+当前 DN 数) / 260)计算，小于 1024 按照 1024 取值，其中，query_dop_limit = 单个机器 CPU 核数 / 单个机器 DN 数。

   - 不建议该参数值设置过大，因为 comm_max_stream 会占用内存（占用内存 =256byte*comm_max_stream*comm_max_datanode），若并发数据流数过大，查询较为复杂及 smp 过大都会导致内存不足。
   - 如果 comm_max_datanode 参数值较小，进程内存充足，可以适当将 comm_max_stream 值调大。

处理方法

根据评估，内存充足，将 comm_max_stream 参数值调大为 2048。（参数值 2048 仅适用示例场景，请根据实际业务的内存查询结果进行参数值调整。）

步骤 1 登录 GaussDB(DWS) 管理控制台。

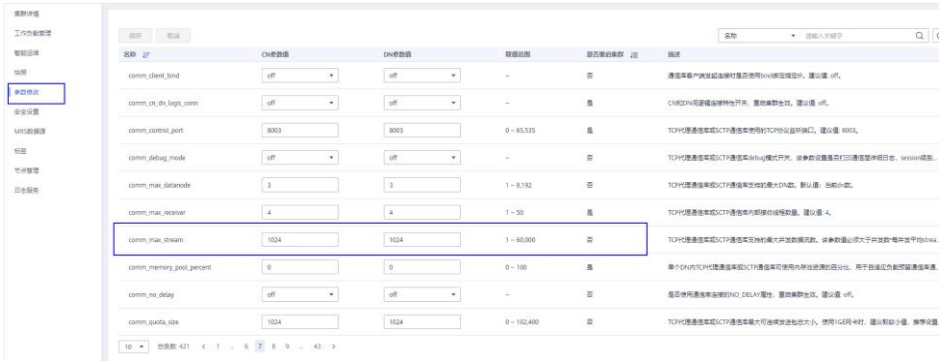
步骤 2 在左侧导航栏中，单击“集群管理”。

步骤 3 在集群列表中找到所需要的集群，单击集群名称，进入集群“基本信息”页面。

步骤 4 单击“参数修改”页签，修改参数“comm_max_stream”，然后单击“保存”。

步骤 5 在“修改预览”窗口，确认修改无误后，单击“保存”。

步骤 6 参数“comm_max_stream”所在行“是否重启”列显示为“否”，表示该参数修改后无需进行重启操作，即修改后立即生效。



----结束

8 集群使用

8.1 Oracle/TD 兼容模式下查询结果不一致

问题现象

客户有两套集群环境，在数据量一致的情况下，对比发现两套环境同样的 SQL 的执行结果不一样。

步骤 1 客户使用的语法可以简化为以下逻辑：

```
create table test (a text, b int);
insert into test values('', 1);
insert into test values(null, 1);
select count(*) from test a, test b where a.a = b.a;
```

步骤 2 在两套环境中执行结果分别如下：

```
结果 1：
select count(*) from test a, test b where a.a = b.a;
 count
-------
     0
(1 row)

结果 2：
tpcds1xcpm=# select count(*) from test a, test b where a.a = b.a;
 count
-------
     1
(1 row)
```

----结束

原因分析

目前支持两种数据库兼容模式，TD 和 ORA。

在 TD 兼容模式下，空和 NULL 是不相等的，在 ORA 兼容模式下，空和 NULL 是相等的。因此上述场景可能是因为两个环境的 database 的兼容性模式设置不一样导致的。

可以通过以下方法确认：

```
select datname, datcompatibility from pg_database;
```

处理方法

这种场景下只能将两个环境的 database 的兼容性模式设置为一致的才能解决。Database 的 datcompatibility 属性不支持 alter，只能通过新建数据库的方法来指定。

8.2 磁盘监控告警阈值太低，告警频繁

问题现象

DWS 集群磁盘使用率达到 80%就预警，告警频繁。

原因分析

集群配置的告警监控阈值不合理。

处理方法

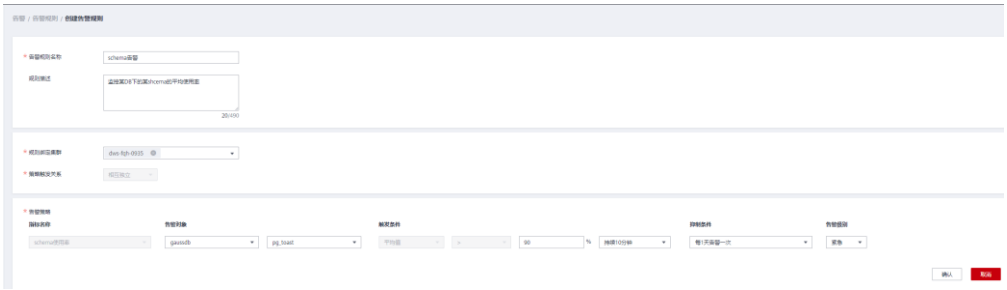
可在 GaussDB(DWS) 管理控制台设置告警的触发条件，指定达到磁盘使用率、告警持续时间及告警频次。

须知

集群磁盘使用率达到 90%就会触发集群只读，需要预留时间来处理问题，避免使用率达到只读阈值。

1. 登录 GaussDB(DWS) 管理控制台。
2. 在左侧导航栏，单击“告警管理”，切换至“告警”页签。
3. 单击左上角的“查看告警规则”按钮，进入告警规则页面。
4. 在指定告警规则名称所在行操作列，单击“修改”按钮进入修改告警规则页面。将触发条件修改为平均值大于 90%，抑制条件修改为“每 1 天告警一次”。（此处仅做举例，实际情况以业务诉求为准。）
   - 触发条件：定义对监控指标做阈值判断的计算规则。目前主要使用一段时间内的平均值来降低告警震荡的几率。
   - 抑制条件：在指定的时间段内，抑制同类型告警的反复触发和消除。

图8-1 设置告警规则



9 修订记录

修改记录

文档版本	发布日期	修改说明
03	2022-11-25	第三次正式发布，适配 DWS 8.1.3.110。
02	2022-10-13	第二次正式发布。
01	2022-6-20	第一次正式发布，适配 DWS 8.1.1.100。