云数据库 GaussDB(for MySQL)

故障排除

文档版本 01

发布日期 2024-09-04

华为云计算技术有限公司

商标声明


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

注意

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

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

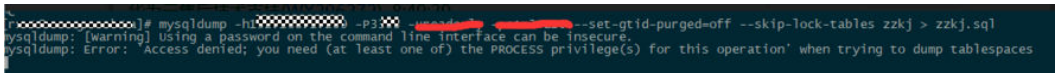
目 录

1 备份恢复

1.1 mysqldump 导数据报错权限不足

场景描述

mysqldump使用指定用户导出数据库数据时，报错：'Access denied; you need (at least one of) the PROCESS privilege(s)



原因分析

mysqldump使用指定用户导出数据时，需要赋予PROCESS权限。

解决方案

使用管理员账户给相应用户授予PROCESS权限。

GRANT SELECT，PROCESS ON *.* TO ‘dump_user’@’%’;

FLUSH PRIVILEGES;

1.2 使用 mysqlbinlog 工具获取 binlog

本文以从弹性云服务器ECS上拉取为例，其他环境下方法类似。

1. 在ECS上安装MySQL客户端，详情请参考**安装MySQL客户端**。

   说明

   GaussDB(for MySQL)兼容社区MySQL 8.0及以上版本，请勿安装8.0以下版本的版本的客户端。

2. 执行命令，下载binlog文件。

   mysqlbinlog -hxxx -uxxx -Pxxx -pxxx binlog.xxxx --read-from-remote-server

   mysqlbinlog的常用参数：

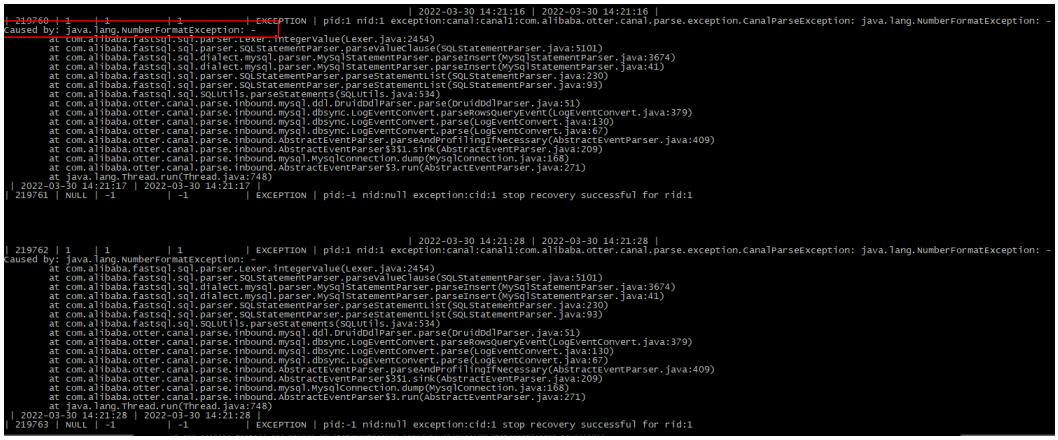
- -h：数据库host。
- -u：用户名。
- -P：端口号。
- -p：密码。
- --start-position：表示从指定的起始位置开始解析。
- --start-datetime：表示从指定的时间开始解析。
- --stop-position：表示解析到指定的位置。
- --stop-datetime：表示解析到指定的时间。
- --skip-gtids：跳过打印gtid_log_event。
- --short-form：表示只显示statements。
- --result-file：将binlog解析生成sql文件。
- --read-from-remote-server：远程下载binlog(用于mysqlbinlog与数据库服务端不再同一台机器的情况)。

1.3 canal 解析 binlog 报错

场景描述

canal解析Binlog出现错误，导致拉取Binlog中断，错误信息如下：

```
com.alibaba.otter.canal.parse.exception.CanalParseException: java.lang.NumberFormatException:- Caused by: java.lang.NumberFormatException:  - at com.alibaba.fastsql.sql.parser.Lexer.integerValue(Lexer.java:2454)
```



原因分析

检查GaussDB(for MySQL)的参数“binlog_rows_query_log_events”的值是否设置为**1**或**ON**。

- 目前canal只能支持ROW格式的Binlog增量订阅。
- 当GaussDB(for MySQL)的参数“binlog_rows_query_log_events”的值设置为**1**或**ON**时，会在Binlog中产生Rows_query类型的event，此类event非ROW格式，一些场景下，会导致canal出现blank topic问题，引发Binlog解析失败。

解决方案

将GaussDB(for MySQL)的参数“binlog_rows_query_log_events”的值修改为**OFF**，重启中断的canal任务。

1.4 使用 mysqldump 导出大表的注意事项

在使用mysqldump导出数据时，倘若添加–q(--quick) 参数时，select出来的结果将不会存放在缓存中，而是直接导出到标准输出中。如果不添加该参数，则会把select的结果放在本地缓存中，然后再输出给客户端。

- 如果只是备份小量数据，足以放在空闲内存buffer中的话，禁用-q参数，则导出速度会快一些。
- 对于大数据集，如果没办法完全储存在内存缓存中时，就会产生swap。对于大数据集的导出，不添加-q参数，不但会消耗主机的内存，也可能会造成数据库主机因无可用内存继而宕机的严重后果。

因此，如果使用mysqldump来备份数据时，建议添加-q参数。

导出示例：

mysqldump -uroot -p-P*8635* **-h***192.168.0.199* **--set-gtid-purged=OFF --single-transaction --flush-logs -q** *test t1*>t1.**sql**

1.5 mysqldump 的 6 大使用场景的导出命令

背景描述

mysqldump是MySQL最常用的逻辑导入导出的工具，下面介绍几种常见使用场景。

mysqldump 选项解析

表 1-1 配置项说明

选项名称	说明
add-drop-table	每个数据表创建之前添加drop数据表语句。
events，E	导出事件。
routines，R	存储过程以及自定义函数。
flush-logs	开始导出之前刷新日志。
no-create-db，n	只导出数据，而不添加CREATE DATABASE语句。
add-drop-database	创建数据库之前添加drop数据库语句。
no-create-info，t	只导出数据，而不添加CREATE TABLE语句。

选项名称	说明
no-data，d	不导出任何数据，只导出数据库表结构。
set-gtid-purged=OFF	不导出gtid相关语句。
hex-blob	使用十六进制格式导出二进制字符串字段。

场景描述

适用场景举例如下。

1. 导出db1、db2两个数据库的所有数据。

   mysqldump -uroot -p -P*8635* **-h***192.168.0.199* **--hex-blob --set-gtid-purged=OFF --single-transaction --order-by-primary --flush-logs -q --databases** *db1 db2* **>db12.sql**

2. 导出db1库的t1和t2表。

   mysqldump -uroot -p -P*8635* **-h***192.168.0.199* **--hex-blob --set-gtid-purged=OFF --single-transaction --order-by-primary --flush-logs -q --databases** *db1* **--tables** *t1 t2* **>***t1_t2***.sql**

3. 条件导出，导出db1表t1中id=1的数据。

   mysqldump -uroot -p -P*8635* **-h***192.168.0.199* **--hex-blob --set-gtid-purged=OFF --single-transaction --order-by-primary --flush-logs -q --databases** *db1* **--tables** *t1* **--where='***id=1***'>***t1_id***.sql**

4. 导出db1下所有表结构，而不导出数据。

   mysqldump -uroot -p -P*8635* **-h***192.168.0.199* **--no-data --set-gtid-purged=OFF --single-transaction --order-by-primary -n --flush-logs -q --databases** db1 **>***db1*_**table.sql**

5. 除db1下的表和数据外，其他对象全部导出。

   mysqldump -uroot -p -h*192.168.0.199* **-P***8635* **--set-gtid-purged=OFF -F -n -t -d -E -R** *db1***> others.sql**

1.6 增加表字段后出现运行卡顿现象

故障描述

当给MySQL实例的表中增加一个字段，出现系统无法访问的现象。

解决方案

因增加表字段而引起数据库出现性能问题，有可能是未对新增字段添加索引，数据量大导致消耗了大量的CPU资源。为此，提出如下建议恢复数据库性能。

- 添加对应索引、主键。
- 优化慢SQL语句。

1.7 怎么解决执行 mysqldump 出现 SET @@SESSION.SQL_LOG_BIN 等 SQL 的问题

场景描述

新购买的华为云数据库，执行mysqldump时，会出现如下如所示代码。

图 1-1 代码显示

backup_procedure.sql

```
-- MySQL dump 10.13  Distrib 5.7.24, for Linux (x86_64)
--
-- Host: 192.168.1.64    Database: rptdb
-- ------------------------------------------------------
-- Server version   5.7.31-2-log

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;
SET @MYSQLDUMP_TEMP_LOG_BIN = @@SESSION.SQL_LOG_BIN;
SET @@SESSION.SQL_LOG_BIN= 0;

--
-- GTID state at the beginning of the backup
--

SET @@GLOBAL.GTID_PURGED='7f47edf7-2e4d-11eb-9a43-fa163eacf6f0:1-36975382';

--
-- Dumping routines for database 'rptdb'
--
/*!50003 DROP FUNCTION IF EXISTS `f_sys_get_partition` */;
/*!50003 SET @saved_cs_client      = @@character_set_client */ ;
/*!50003 SET @saved_cs_results     = @@character_set_results */ ;
/*!50003 SET @saved_col_connection = @@collation_connection */ ;
/*!50003 SET character_set_client  = utf8 */ ;
/*!50003 SET character_set_results = utf8 */ ;
/*!50003 SET collation_connection  = utf8_general_ci */ ;
/*!50003 SET @saved_sql_mode       = @@sql_mode */ ;
```

故障分析

开启了“gtid-mode=ON”参数。

如果一个数据库开启了GTID，使用mysqldump备份或者转储的时候，即使不是MySQL全库(所有库)备份，也会备份整个数据库所有的GTID号。

解决方案

在GaussDB(for MySQL)数据库进行导出备份和恢复的时候，需要注意是否启用数据库用GTID模式。

如果开启，则在mysqldump数据时，应该在mysqldump命令加上参数“–set-gtid-purged=OFF”。

1.8 canal 工具报错权限不足

场景描述

在搭建canal环境，使用指定用户从GaussDB(for MySQL)获取Binlog时，启动canal经常会报如下错误：'show master status' has an error! Access denied: you need (at least one of) the SUPER, REPLICATION CLIENT privilege(s) for this operation

完整报错信息如下：

```
2021-01-10 23:58:32.964 [destination = evoicedc , address = /dbus-mysql:3306 , EventParser] ERROR com.alibaba.ot ter.canal.common.alarm.LogAlarmHandler - destination:evoicedc[com.alibaba.otter.canal.parse.exception.CanalParseEx ception: command : 'show master status' has an error!
Caused by: java.io.IOException: ErrorPacket [errorNumber=1227, fieldCount=-1, message=Access denied; you need (at least one of) the SUPER, REPLICATION CLIENT privilege(s) for this operation, sqlState=42000, sqlStateMarker=#] with command: show master status at com.alibaba.otter.canal.parse.driver.mysql.MysqlQueryExecutor.query(MysqlQueryExecutor.java:61)
```

原因分析

canal拉取Binlog时需要赋予REPLICATION SLAVE, REPLICATION CLIENT权限。

解决方案

使用管理员账户给相应用户授予REPLICATION SLAVE, REPLICATION CLIENT权限。

GRANT SELECT, REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO ‘canal’@’%’;

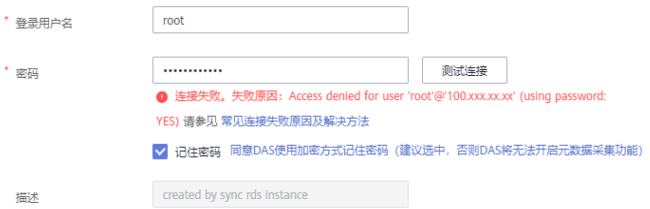
FLUSH PRIVILEGES;

2 连接类

2.1 root 账号的 ssl_type 修改为 ANY 后无法登录

场景描述

在控制台以root账号通过DAS登录实例时，报错Access denied。



原因分析

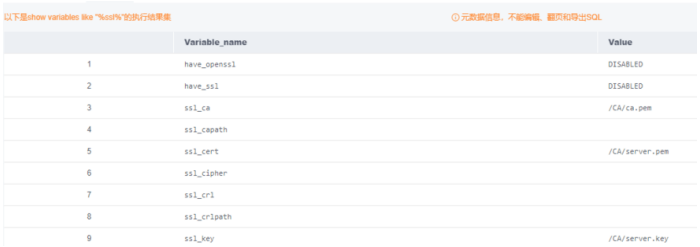
1. 查看mysql.user表中的root账号信息，排查客户端IP范围是否正确、是否使用SSL。

   ```
   SELECT * FROM mysql.user WHERE User='root';
   ```

   如果发现root账号的**ssl_type**被设置为**ANY**，表明root账号需要使用SSL连接。
2. 查看SSL开启情况。

   show variables like '%ssl%';

   发现该实例未开启SSL：

	Variable_name	Value
1	have_openssl	DISABLED
2	have_ssl	DISABLED
3	ssl_ca	/CA/ca.pem
4	ssl_capath	
5	ssl_cert	/CA/server.pem
6	ssl_cipher	
7	ssl_crl	
8	ssl_crlpath	
9	ssl_key	/CA/server.key

   因此，问题原因是自行修改root账号的**ssl_type**为**ANY**后，导致无法登录。

解决方案

将root账号的**ssl_type**修改为空即可，参考命令：

update mysql.user set ssl_type='' where user = 'root';

如果要将其他所有用户账号的**ssl_type**修改为空，参考命令：

update mysql.user set ssl_type='' where user not like 'mysql%';

2.2 SSL 使用与介绍

场景描述

使用SSL无法连接上数据库。

原因分析

优先检查网络是否已经连通，如果不带SSL的连接方式可以连接，则可能是mysql client或对应的数据库驱动的版本不兼容。

解决方案

GaussDB(for MySQL)是兼容社区8.0以上版本的，需要使用8.0及以上版本的mysql client或数据库驱动。

SSL(Secure Socket Layer：安全套接字层)使用数据加密、身份校验和消息完整性校验，为连接提供安全性保证。

SSL提供的功能主要包含：

1. 加密数据传输：利用对称密钥算法对传输的数据进行加密。
2. 身份校验：基于证书使用数字签名的方法对客户端与服务器进行身份验证。
3. 消息完整性校验：消息传输过程中使用MAC算法来检验消息的完整性。

注意

- 当服务端的SSL开启时，客户端的身份验证是可选的，即：即使服务端开启了SSL，客户端也可以不通过SSL的方式连接服务端，此时也可以正常通信，只是数据不会被加密。
- 如果不是通过SSL的方式，那么其在网络中的传输数据会以明文进行，存在安全隐患。
- GaussDB(for MySQL)数据库服务端会默认开启服务端会默认开启SSL（如需关闭可参考：**设置SSL数据加密**），客户业务使用时可以在客户端自行选择是否使用SSL。
- 使用mysql client连接时使用SSL的方式可参考：**通过客户端连接GaussDB(for MySQL)**
- 使用JDBC连接时使用SSL的方式可参考：**通过JDBC连接MySQL数据库**

2.3 对各个 IP 地址的解释说明

购买GaussDB(for MySQL)实例后获得了多个IP地址，以1主1只读为例，在实例基础信息中最多共能找到5个IP，业务可以按自己的需要连接对应的IP。

说明

对于节点读内网地址，如果出现节点故障，故障恢复前IP不可访问。

1. **主节点读内网地址（不推荐使用）**

   IP与节点绑定，可以从内网（同VPC网络内）直接连接IP做读写操作，如果发生故障倒换，节点变为只读节点，则该IP将只能做读操作，不能做写操作。对该IP的操作实际会落到对应的节点上。

2. **只读节点读内网地址（不推荐使用）**

   IP与节点绑定，可以从内网（同VPC网络内）直接连接IP做读操作，如果发生故障倒换，节点变为主节点，则该IP将能做读写操作。对该IP的操作实际会落到对应的节点上。

3. **读写内网地址**

   浮动IP，IP永远与主节点绑定，可以内网（同VPC网络内）直接连接IP做读写操作。如果发生故障倒换，该IP会浮动到新的主节点，依然可以做读写操作。对该IP的操作永远会落到当时的主节点上。

4. **读写公网地址(购买实例后需要单独绑定)**

   购买并绑定公网IP后，可以从公网连接IP做读写操作。与浮动IP相同，也是一直与主节点绑定，且一直可以做读写操作。对该IP的操作永远会落到当时的主节点上。

5. 代理地址**(购买实例后要开通读写分离才会有)**

   购买数据库代理后，可以从数据库代理中查看代理地址，连接该地址可以执行读写操作，数据库代理会自动将写请求发送到当时的主节点，将读请求发送到当时的只读节点，最大化利用实例1主多读的多节点读写能力，降低主节点的压力。读写分离IP当前暂时只支持内网访问，尚未开放绑定公网IP的功能。

说明

故障倒换：

GaussDB(for MySQL)默认最少为2个节点，1主（可读可写）1只读（只可读不可写），主节点仅允许有一个，只读节点可以有多个。

当主节点遇到故障时，高可用系统会迅速发现，并选择一个只读节点将其升级为主节点，并将原主节点修复为只读节点，这个过程叫故障倒换。

2.4 客户端 TLS 版本 MySQL 不一致导致 SSL 连接失败

场景描述

某业务客户端连接到云上GaussDB(for MySQL)失败，但是连接到自建环境或其他环境可以成功，均使用了SSL连接。

原因分析

排查步骤：

1. **查看GaussDB(for MySQL)的错误日志**，观察到如下报错：

```
2021-07-09T10:30:58.476586+08:00 212539 [Warning] SSL errno: 337678594, SSL errmsg: error:14209102:SSL routines:tls_early_post_process_client_hello:unsupported protocol2021-07-09T10:30:58.476647+08:00 212539 [Note] Bad handshake2021-07-09T10:32:43.535738+08:00 212631 [Warning] SSL errno: 337678594, SSL errmsg: error:14209102:SSL routines:tls_early_post_process_client_hello:unsupported protocol2021-07-09T10:32:43.535787+08:00 212631 [Note] Bad handshake2021-07-09T10:50:03.401100+08:00 213499 [Warning] SSL errno: 337678594, SSL errmsg: error:14209102:SSL routines:tls_early_post_process_client_hello:unsupported protocol2021-07-09T10:50:03.401161+08:00 213499 [Note] Bad handshake2021-07-09T10:53:44.458404+08:00 213688 [Warning] SSL errno: 337678594, SSL errmsg: error:14209102:SSL routines:tls_early_post_process_client_hello:unsupported protocol2021-07-09T10:53:44.458475+08:00 213688 [Note] Bad handshake
```

2. 从报错信息unsupported protocol可以看出，很可能和TLS版本相关，使用如下命令，分别查看GaussDB(for MySQL)和自建MySQL的TLS版本。

   show variables like '%tls_version%';

   发现GaussDB(for MySQL)为TLS v1.2版本，自建MySQL为TLS v1.1版本，存在差异。进一步确认客户端TLS版本，与自建MySQL一致，因此出现连接自建MySQL成功，连接云上GaussDB(for MySQL)失败。

解决方案

客户端升级TLS版本到TLS v1.2。

如果使用官方JDBC驱动**mysql-connector/J**，可参考**官方文档**，配置方法：

TLS versions: The allowable versions of TLS protocol can be restricted using the connection properties `tlsVersions` and, for X DevAPI connections and for release 8.0.19 and later, `xdevapi.tls-versions` (when `xdevapi.tls-versions` is not specified, it takes up the value of `tlsVersions`). If no such restrictions have been specified, Connector/J attempts to connect to the server with the TLSv1.2 and TLSv1.3.

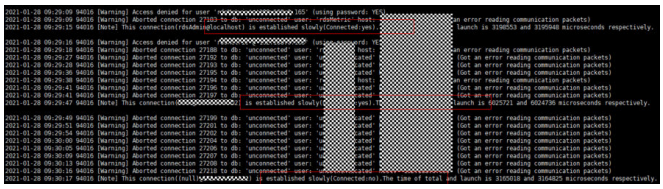
2.5 GaussDB(for MySQL)建立连接慢导致客户端超时

场景描述

业务在高峰期时，客户端经常出现向MySQL建立连接超时，导致系统登录需要十几秒。

原因分析

1. **查看GaussDB(for MySQL)的错误日志**，观察是否有如下信息：connection xxx is established slowly。示例：



有上述日志，说明存在某些连接超过一定时间仍未被MySQL处理，客户端的超时时间大于该时间，就会报错。

2. 进一步查看线程池配置（默认开启），可以在控制台查看。

参数名称	是否可修改	重启有效	值	参数范围
threadpool_oversubscribe	是	否	3	[1-50]
threadpool_size	是	否	1	[1-512]
threadpool_stall_limit	是	否	500	[10-4,294,967,295]

可以看出，threadpool_size为1，threadpool_stall_limit为500ms，threadpool_oversubscribe为3，线程池处理连接等待的时间主要与上述3个参数相关：

- 当线程池所有线程在忙碌，线程池中的调度线程会每隔500ms（threadpool_stall_limit）创建一个新线程，所以此时，每个线程组平均每500ms才能处理一个新的连接。如果队列太长，很可能导致客户端超时；
- 所有线程都在忙碌是指，工作线程达到线程池总线程数，在大量建立连接时，总线程数计算方法：threadpool_size*(threadpool_oversubscribe+1))

解决方案

对于存在大量新建连接，建议调大threadpool_oversubscribe增加线程总数。

减少线程重复创建与销毁部分的开销，提高性能，同时它也限制了MySQL的runing线程数，关键时刻可以保护系统，防止雪崩。

正常情况下，线程池适用于大量短连接的场景，如果客户是长连接，并且连接数量不多（客户端使用了连接池等情况），线程池的作用不大，此时调整threadpool_size和threadpool_oversubscribe两个参数扩大线程总数，或者直接关闭线程池。

2.6 连接数据库报错 Access denied

场景描述

客户端连接数据库异常，返回错误：Error 1045: Access denied for user xxx

处理方法

1. 连接了错误的主机

   问题原因：业务连接了错误的数据库主机，该主机上相应用户或客户端IP没有权限访问。

   解决方案：仔细检查要连接的数据库主机名，确保正确。

2. 用户不存在

   问题原因：客户端连接时，使用的用户不存在。

   解决方案：

   - 使用管理员账户登录数据库，执行如下命令检查目标用户是否存在。
     ```
     SELECT User FROM mysql.user WHERE User='xxx';
     ```
   - 如果用户不存在，创建相应用户。
     ```
     CREATE USER 'xxx'@'xxxxxx' IDENTIFIED BY 'xxxx';
     ```

3. 用户存在，但客户端IP无访问权限

   问题原因：客户端使用的用户存在，但是客户端IP没有该数据库的访问权限。

   解决方案：

   - 使用管理员账户登录数据库，执行如下命令，检查目标用户允许哪些客户端IP连接。
     ```
     SELECT Host, User FROM mysql.user WHERE User='xxx';
     ```
   - 如果上述查询出的Host不包含客户端IP所在网段，则需要赋予相应访问权限。例如，赋予test用户192.168.0网段访问权限。
     ```
     GRANT ALL PRIVILEGES ON *.* TO'root'@'192.168.0.%' IDENTIFIED BY 'password' WITH GRANT OPTION;
     FLUSH PRIVILEGES;
     ```

4. 密码错误

   问题原因：用户对应的密码错误，或忘记密码

   解决方案：

   - 确定目标密码是否错误，由于密码用于身份验证，因此无法从MySQL以明文形式读取用户密码，但可以将密码的哈希字符串与目标密码的“PASSWORD”函数值进行比较，确定目标密码是否正确，示例SQL语句：
     ```
     mysql> SELECT Host, User, authentication_string, PASSWORD('12345') FROM mysql.user WHERE User='test';
     +-----------+------+-------------------------------------------+-------------------------------------------+
     | Host      | User | authentication_string                     | PASSWORD('12345')                         |
     +-----------+------+-------------------------------------------+-------------------------------------------+
     | %         | test | *6A23DC5E7446019DC9C1778554ED87BE6BA61041 | *00A51F3F48415C7D4E8908980D443C29C69B60C9 |
     +-----------+------+-------------------------------------------+-------------------------------------------+
     2 rows in set, 1 warning (0.00 sec)
     ```
     从上面例子可以看出，PASSWORD('12345')的哈希值与authentication_string列不匹配，这表明目标密码“12345”是错误的。
   - 如果需要重置用户密码，参考如下SQL语句：
     ```
     set password for 'test'@'%' = 'new_password';
     ```

5. 密码包含特殊字符被Bash转义

   问题原因：Linux默认的Bash环境下，使用命令行连接数据库，用户密码中包含特殊字符会被环境转义，导致密码失效。

   例如，在Bash环境下，用户test的密码为“test$123”，使用命令mysql -hxxx -u test -ptest$123，连接数据库会报错ERROR 1045 (28000): Access denied。

   解决方案：通过用单引号将密码括起来，防止Bash解释特殊字符。

   ```
   mysql -hxxx -u test -p'test$123'
   ```

6. 用户设置了REQUIRE SSL，但客户端使用非SSL连接

   排查思路：

   - 排查报错用户名是否强制使用SSL连接，执行：show create user 'xxx'，如果出现“REQUIRE SSL”属性，说明该用户必须使用SSL连接。
   - 排查是否使用过如下类似语句给用户授权。
     ```
     GRANT ALL PRIVILEGES ON . TO 'ssluser'@'localhost' IDENTIFIED BY 'password' REQUIRE SSL;
     ```
   - 检查目标用户的ssl_type值，如果不为空，说明该用户需要使用SSL连接。
     ```
     SELECT User, Host, ssl_type FROM mysql.user WHERE User='xxx';
     ```

   解决方案：

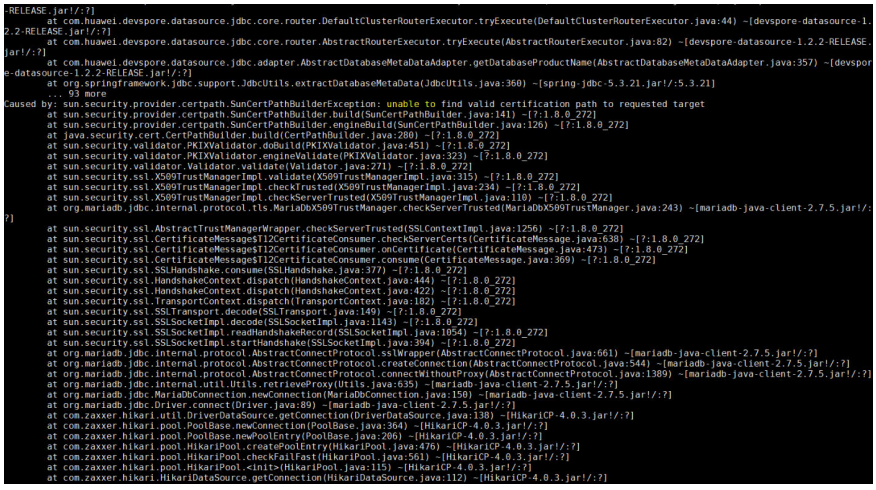
   - 客户端使用SSL方式数据库连接，请参考**SSL连接方式**。
   - 去除用户SSL连接权限，参考命令：**ALTER USER 'username'@'host' REQUIRE NONE;**

2.7 mariadb-connector SSL 方式连接数据库失败

场景描述

使用jdbc无法连接数据库，报如下错误：

```
unable to find certification path to requested target
```



原因分析

从错误截图中可以看出，使用的是mariadb的jar包，而非MySQL的官方驱动包，而mariadb与官方的使用方法略有区别。

解决方案

对于mariadb的连接串应该为：

```
String url = "jdbc:mysql://xxx.xxx.xxx.xxx:xxxx/mysql?useSsl=true&serverSslCert=D:\
\ca.pem&disableSslHostnameVerification=true";
```

注意：GaussDB(for MySQL)实例不支持hostname校验，因此需要设置disableSslHostnameVerification=true，不同mariadb jar包版本设置方式不同，可查看对应版本的**使用说明**。

2.8 使用 root 账号连接数据库失败

场景描述

使用root账号连接数据库失败。

原因分析

1. 查看内核日志error.log，确认是否有拒绝连接的日志。
2. 查看root权限，发现有两个root账号，其中一个root限制host必须是IP必须是192开头。

```
mysql> select * from mysql.user where user='root'\G;
*************************** 1. row ***************************
                Host: %
                User: root
           Select_priv: Y
           Insert_priv: Y
           Update_priv: Y
           Delete_priv: Y
           Create_priv: Y
             Drop_priv: Y
           Reload_priv: Y
         Shutdown_priv: N
          Process_priv: Y
             File_priv: N
            Grant_priv: Y
       References_priv: Y
            Index_priv: Y
            Alter_priv: Y
          Show_db_priv: Y
            Super_priv: N
 Create_tmp_table_priv: Y
      password_lifetime: NULL
         account_locked: N
*************************** 2. row ***************************
                Host: 192.%
                User: root
           Select_priv: Y
           Insert_priv: Y
           Update_priv: Y
           Delete_priv: Y
           Create_priv: Y
             Drop_priv: Y
           Reload_priv: Y
```

解决方案

联系华为云客服协助删除多余的root账号。

2.9 客户端连接实例后会自动断开

故障描述

MySQL客户端连接实例后，会自动断开，报错信息：“ERROR 2013：Lost connection to MySQL server during query”。

解决方案

ERROR 2013是MySQL常见错误，一般为配置错误导致。

- “wait_timeout”：服务器关闭非交互连接之前等待活动的秒数。
- “interactive_timeout”：服务器关闭交互连接之前等待活动的秒数。

步骤1 查看实例状态是否处于正常状态。

经查看实例状态正常，继续排查其他问题。

步骤2 查看错误日志。

步骤3 使用MySQL命令行客户端连接数据库，执行status命令，确认数据库实例是否频繁重启。

```
mysql> status
--------------
mysql  Ver 14.14 Distrib 5.6.34, for Linux (x86_64) using  EditLine wrapper

Connection id:          16288
Current database:
Current user:           root@192.168.0.5
SSL:                    Not in use
Current pager:          stdout
Using outfile:          ''
Using delimiter:        ;
Server version:         5.6.34-log MySQL Community Server (GPL)
Protocol version:       10
Connection:             192.168.0.24 via TCP/IP
Server characterset:    utf8
Db     characterset:    utf8
Client characterset:    utf8
Conn.  characterset:    utf8
TCP port:               8635
Uptime:                 5 hours 5 min 34 sec

Threads: 2  Questions: 62118  Slow queries: 0  Opens: 70  Flush tables: 2  Open tables: 0  Queries per second avg: 3.388
--------------
```

Uptime代表实例的运行时间，从排查结果可知，数据库并没有频繁重启，因而，客户端连接被断开，不是因数据库重启引起的。

步骤4 查看“wait_timeout”和“interactive_timeout”参数设置，MySQL会自动断开超时的空连接。

步骤5 您可根据实际应用需求量，修改“wait_timeout”和“interactive_timeout”参数值，无需重启实例。

步骤6 恢复结果确认，等到10分钟左右，再次执行show databases命令，确认连接是否正常。

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
+--------------------+
3 rows in set (0.00 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
+--------------------+
3 rows in set (0.00 sec)

mysql>
```

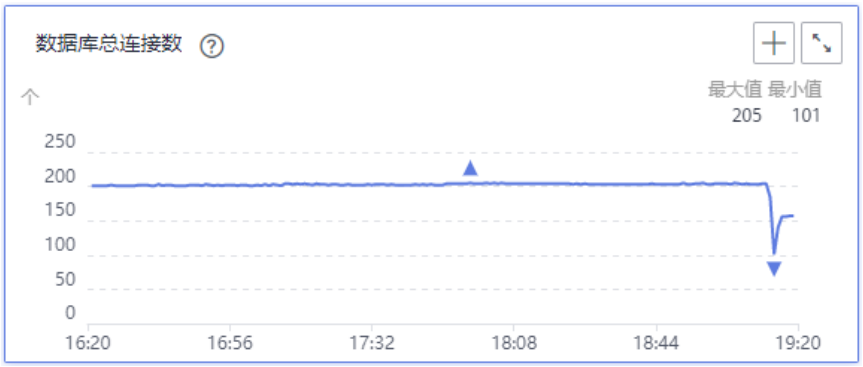
如图所示，说明连接正常。

----结束

2.10 istio-citadel 证书机制导致每隔 45 天出现断连

场景描述

业务侧发现数据库每隔45天同一时间，多台数据库实例的连接数骤降。查看服务端连接数监控指标如下：

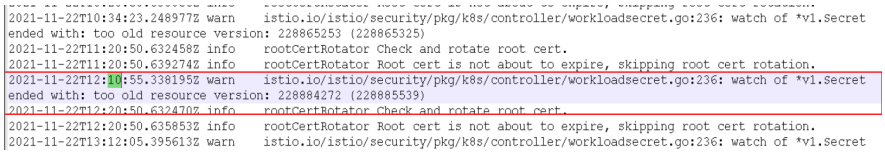


客户端出现大量报错如下：

```
[2022-[redacted]] [http-nio-[redacted]-exec-5~8] [ERROR] [[redacted]] [--] [druid.sql.Stateme
nt.149] - {conn-110005, stmt-883289} execute error. select 1 from dual
java.sql.SQLNonTransientConnectionException: (conn=12518697) unexpected end of stream, read 0 bytes from 4 (socket was closed by server)
        at org.mariadb.jdbc.internal.util.exceptions.ExceptionFactory.createException(ExceptionFactory.java:73)
        at org.mariadb.jdbc.internal.util.exceptions.ExceptionFactory.create(ExceptionFactory.java:153)
        at org.mariadb.jdbc.MariaDbStatement.executeExceptionEpilogue(MariaDbStatement.java:274)
        at org.mariadb.jdbc.MariaDbStatement.executeInternal(MariaDbStatement.java:363)
        at org.mariadb.jdbc.MariaDbStatement.executeQuery(MariaDbStatement.java:612)
```

原因分析

1. 排查业务侧是否有间隔45天的定时任务。
2. 客户端如果使用了istio等证书加密机制，分析证书相关日志，是否有如下类似信息。如果有，说明是证书过期导致。



客户端istio-citadel证书每隔45天过期，导致主动发起数据库断连请求。

解决方案

- 客户端设置合理的istio-citadel证书过期时间，并在过期发生时，主动规避操作。
- 客户端排查是否有其他证书过期问题。

3 SQL 类

3.1 建表时 timestamp 字段默认值无效

场景描述

客户执行一个建表SQL语句失败，详细SQL语句及报错如下：

```
CREATE TABLE cluster_membership
(
...
session_start TIMESTAMP DEFAULT '1970-01-01 00:00:01',
...
);
```

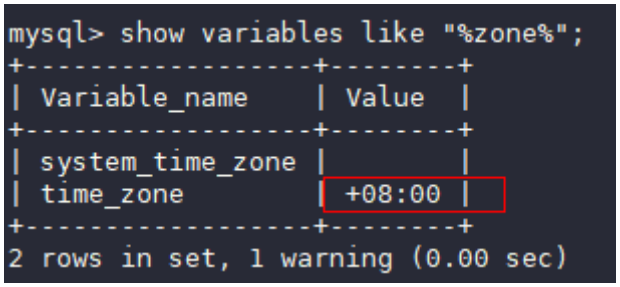
执行失败，失败原因：ERROR 1067: Invalid default value for 'session_start'

原因分析

表字段类型是TIMESTAMP类型，

关于timestamp字段：MySQL会把该字段插入的值从当前时区转换成UTC时间（世界标准时间）存储，查询时，又将其从UTC时间转化为当前时区时间返回

1. timestamp类型字段的时间范围：'1970-01-01 00:00:01' UTC -- '2038-01-19 03:14:07' UTC，详见官方文档：
2. 使用如下命令，查看当前的时区：
   ```
   show variables like "%zone%";
   ```
3. 故障场景中使用的是utc+8时区，如下图，所以timestamp字段默认值需要加8小时才是有效范围，有效支持的范围是从1970-01-01 08:00:01开始；



解决方案

执行命令，修改timestamp字段参数默认值。

```
session_start TIMESTAMP DEFAULT '1970-01-01 08:00:01',
```

3.2 索引长度限制导致修改 varchar 长度失败

场景描述

执行alter table修改表结构失败，报错如下：

```
Specified key was too long; max key length is 3072 bytes
```

原因分析

- 在“innodb_large_prefix”设置为off的情况下，InnoDB表的单字段索引的最大字段长度不能超过767字节，联合索引的每个字段的长度不能超过767字节，且所有字段长度合计不能超过3072字节。
- 当“innodb_large_prefix”设置为on时，单字段索引最大长度可为3072字节，联合索引合计最大长度可为3072字节。
- 索引长度与字符集相关。使用utf8字符集时，一个字符占用三个字节，在“innodb_large_prefix”参数设置为on情况下，索引的所有字段的长度合计最大为1072个字符。

查看表结构如下：

```
CREATE TABLE `xxxxx` (
......
`subscription_type` varchar(64) NOT NULL DEFAULT 'DEVICE_EXCEPTION' COMMENT '订阅类型',
`auth_key` varchar(255) DEFAULT '' COMMENT '签名，接口请求头会根据这个值增加token',
`create_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
`update_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
COMMENT '修改时间',
PRIMARY KEY (`id`) USING BTREE,
```

```
UNIQUE KEY `enterprise_id` (`subscription_type`,`enterprise_id`,`callback_url`) USING BTREE)
) ENGINE=InnoDB AUTO_INCREMENT=1039 DEFAULT CHARSET=utf8 ROW_FORMAT=DYNAMIC
```

该表使用了utf8字符集，一个字符占用三个字节。联合索引“enterprise_id”包含了“callback_url”字段，如果执行DDL操作将“callback_url”修改为varchar(1024)，会超出联合索引最大长度限制，所以报错。

解决方案

MySQL机制约束，建议修改索引或字段长度。

3.3 delete 大表数据后，再查询同一张表时出现慢 SQL

场景描述

一次性删除多条宽列数据（每条记录数据长度在1GB左右），再次对同一张表进行增删改查时均执行缓慢，20分钟左右后恢复正常。

场景案例

1. 假定max_allowed_packet参数大小为1073741824。
2. 创建表。
   ```
   CREATE TABLE IF NOT EXISTS zstest1
   (
   id int  PRIMARY KEY not null,
   c_longtext LONGTEXT
   );
   ```
3. 向表中插入数据。
   ```
   insert into zstest1 values(1, repeat('a', 1073741800));
   insert into zstest1 values(2, repeat('a', 1073741800));
   insert into zstest1 values(3, repeat('a', 1073741800));
   insert into zstest1 values(4, repeat('a', 1073741800));
   insert into zstest1 values(5, repeat('a', 1073741800));
   insert into zstest1 values(6, repeat('a', 1073741800));
   insert into zstest1 values(7, repeat('a', 1073741800));
   insert into zstest1 values(8, repeat('a', 1073741800));
   insert into zstest1 values(9, repeat('a', 1073741800));
   insert into zstest1 values(10, repeat('a', 1073741800));
   ```
4. 删除数据。
   ```
   delete from zstest1;
   ```
5. 执行查询语句。
   ```
   select id from zstest1;   //执行缓慢
   ```

原因分析

执行完delete操作后，后台purge线程会去清理标记为delete mark的记录。由于当前删除的数据量较大，purge遍历释放page的过程中会去获取page所在索引根节点的SX锁，导致select语句无法获取到根节点page的rw-lock，一直在等待。

解决方案

- 该场景为正常现象，等待purge操作完成后即可恢复正常。
- 扩大实例规格，提高purge效率。
- 调整优化业务，避免突然删除大量数据。如果需要删除表中所有数据，建议使用**truncate table**。

3.4 更新 emoji 表情数据报错 Error 1366

场景描述

业务插入或更新带有emoji表情的数据时，报错Error 1366。

```
java.sql.SQLException: Incorrect string value: '\xF0\x9F\x90\xB0\xE5\xA4...' for column 'username' at row 1 ;
uncategorized SQLException for SQL []; SQL state [HY000]; error code [1366];
Incorrect string value: '\xF0\x9F\x90\xB0\xE5\xA4...' for column 'username' at row 1;
```

原因分析

原因是字符集配置有误：

- emoji表情为特殊字符，需要4字节字符集存储。
- 该问题场景下，数据库字符集为utf-8，它最多支持3个字节；utf8mb4才是支持4个字节的字符集；

解决方案

1. 将存储emoji表情的字段的字符集修改为utf8mb4。

   如果涉及的表和字段比较多，建议把对应表、数据库的编码也设置为utf8mb4。参考命令：

   ALTER DATABASE database_name CHARACTER SET= utf8mb4 COLLATE= utf8mb4_unicode_ci;

   ALTER TABLE table_name CONVERTTOCHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

   ALTER TABLE table_name MODIFY 字段名 VARCHAR(128) CHARSET utf8mb4 COLLATE utf8mb4_unicode_ci;
2. 若对应字段的字符集已经是utf8mb4，则为客户端或MySQL服务端字符集转换问题，将客户端和MySQL服务端的字符集都设置为utf8mb4。

3.5 存储过程和相关表字符集不一致导致执行缓慢

场景描述

GaussDB(for MySQL)存储过程执行很慢，处理少量数据耗时1min以上，而单独执行存储过程中的SQL语句却很快。

原因分析

存储过程和相关表、库的字符集不一致，导致查询结果存在大量字符转换，从而执行缓慢。

排查过程：

使用如下命令查看存储过程和相关表的定义，观察存储过程和表的字符集是否一致。

```
SHOW CREATE PROCEDURE xxx;
SHOW CREATE TABLE xxx
```


示例：

```
mysql> SHOW CREATE PROCEDURE testProc \G
*************************** 1. row ***************************
Procedure: showstuscore
sql_mode: STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION
Create Procedure: xxx
character_set_client: utf8mb4
collation_connection: utf8mb4_general_ci
Database Collation: utf8_general_ci
1 row in set (0.01 sec)
```

可以看出，上述存储过程collation为utf8mb4_general_ci，而所在库collation默认为utf8_general_ci，collation值不一致，容易导致性能问题。

解决方案

将存储过程和相关表、库的字符集改成一致后，执行缓慢问题解决。

3.6 报错 ERROR [1412]的解决方法

场景描述

连接GaussDB(for MySQL)执行SQL时，出现如下报错：

```
ERROR[1412]:Table definition has changed, please retry transaction``
```

原因分析

启动一致性快照事务后，其他会话（session）执行DDL语句导致。问题复现步骤：

1. 会话1启动一致性快照事务。

```
mysql> start transaction with consistent snapshot;
Query OK, 0 rows affected (0.00 sec)
```

2. 会话2执行DDL操作，修改表结构。

```
mysql> alter table t_sec_user add test int;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

3. 会话1执行普通的查询语句。

```
mysql> select count(*) from t_sec_user;
ERROR 1412 (HY000): Table definition has changed, please retry transaction
mysql>
```

也可以通过Binlog或者审计日志，分析业务侧是否有同一个表DDL和一致性快照事务一起执行的情况。

解决方案

若经排查，是由上述原因引起的报错，需要业务侧避免同一个表的DDL语句和一致性快照事务同时执行。

3.7 存在外键的表无法删除

场景描述

删除MySQL表时，如果表中有外键（ foreign key ），会出现如下报错，且和用户权限无关：

```
ERROR 1451 (23000): Cannot delete or update parent row: a foreign key constraint fails ·····
```

原因分析

这个表和其他表有外键关系，在MySQL中，设置了外键关联，会造成无法更新或删除数据，避免破坏外键的约束。

可以通过设置变量FOREIGN_KEY_CHECKS值为off，来关闭上述机制，详见**官方文档**。

解决方案

通过设置变量FOREIGN_KEY_CHECKS值为off，来关闭上述机制：

```
set session foreign_key_checks=off;
drop table table_name;
```

3.8 GROUP_CONCAT 结果不符合预期

场景描述

SQL语句中使用GROUP_CONCAT()函数时，出现结果不符合预期的情况。

原因分析

GROUP_CONCAT()函数返回一个字符串结果，该结果由分组中的值连接组合而成。需要注意的是：这个函数的结果长度是有限制的，由group_concat_max_len参数决定。

示例：

```
mysql> show variables like 'group_concat_max_len';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| group_concat_max_len | 1024  |
+----------------------+-------+
1 row in set (0.01 sec)

mysql> select GROUP_CONCAT(c1,c2,c3) from dis;
+------------------------+
| GROUP_CONCAT(c1,c2,c3) |
+------------------------+
| 111,222,322            |
+------------------------+
```

```
mysql> set session group_concat_max_len=8;
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like 'group_concat_max_len';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| group_concat_max_len | 8     |
+----------------------+-------+
1 row in set (0.01 sec)

mysql> select GROUP_CONCAT(c1,c2,c3) from dis;
+-------------------------+
| GROUP_CONCAT(c1,c2,c3) |
+-------------------------+
| 111,222,                |
+-------------------------+
```

解决方案

调整group_concat_max_len参数值，适配GROUP_CONCAT()函数的结果长度。

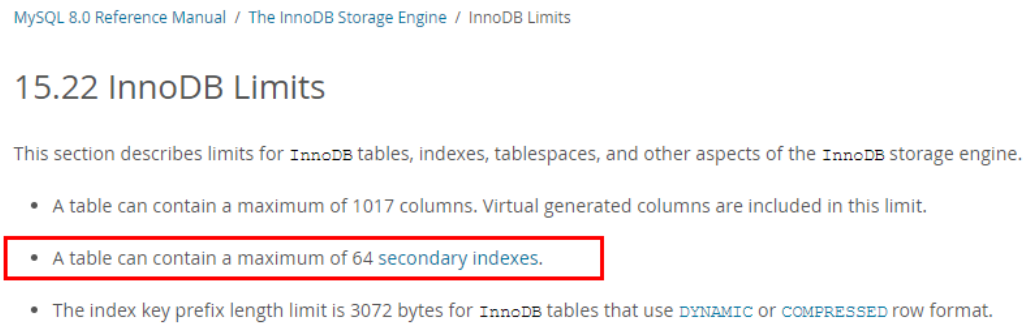
3.9 创建二级索引报错 Too many keys specified

场景描述

创建二级索引失败，报错：Too many keys specified; max 64 keys allowed.

故障分析

MySQL对InnoDB每张表的二级索引的数量上限有限制，限制上限为64个，超过限制会报错“Too many keys specified; max 64 keys allowed”。详见**官方文档**。



解决方案

MySQL机制导致，建议优化业务，避免单表创建过多索引。

说明

InnoDB表的其他限制：

1. 一个表最多可以包含1017列（包含虚拟生成列）。
2. InnoDB对于使用DYNAMIC或COMPRESSED行格式的表，索引键前缀长度限制为3072字节。
3. 多列索引最多允许16列，超过限制会报错。

3.10 distinct 与 group by 优化

场景描述

使用distinct或group by的语句执行比较慢。

原因分析

大部分情况下，distinct是可以转化成等价的group by语句。在MySQL中，distinct关键字的主要作用就是去重过滤。

distinct进行去重的原理是先进行分组操作，然后从每组数据中取一条返回给客户端，分组时有两种场景：

- distinct的字段全部包含于同一索引：该场景下MySQL直接使用索引对数据进行分组，然后从每组数据中取一条数据返回。
- distinct字段未全部包含于索引：该场景下索引不能满足去重分组需要，会用到临时表（首先将满足条件的数据写入临时表中，然后在临时表中对数据进行分组，返回合适的数据）。因为使用临时表会带来额外的开销，所以一般情况下性能会较差。

综上，在使用distinct或group by的时候，尽量在合理的情况下设置可以包含所有依赖字段的索引，优化示例：

- 没有合适索引，导致需要用到临时表。

```
mysql> show create table test;
+-------+-------------------------------------------------------------------------------------------------------
-------+
| Table | Create Table
      |
+-------+-------------------------------------------------------------------------------------------------------
-------+
| test  | CREATE TABLE `test` (
  `id` int NOT NULL,
  `c1` int DEFAULT NULL,
  `c2` int DEFAULT NULL,
  `c3` int DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `c1` (`c1`),
  KEY `c2` (`c2`),
  KEY `c3` (`c3`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+-------+-------------------------------------------------------------------------------------------------------
-------+
1 row in set (0.00 sec)
```

```
mysql> explain select distinct c1,c2,c3 from test;
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-----------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra           |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-----------------+
|  1 | SIMPLE      | test  | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    1 |   100.00 | Using temporary |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-----------------+
1 row in set, 1 warning (0.00 sec)

mysql> explain select c1,c2,c3 from test group by c1,c2,c3;
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-----------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra           |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-----------------+
|  1 | SIMPLE      | test  | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    1 |   100.00 | Using temporary |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-----------------+
1 row in set, 1 warning (0.03 sec)
```

- 有合适的索引，不会使用临时表，直接走索引。

```
mysql> alter table test add key(c1,c2,c3);
Query OK, 0 rows affected (0.10 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> show create table test;
+-------+-------------------------------------
--------------------------------------+
| Table | Create Table
                                       |
+-------+-------------------------------------
--------------------------------------+
| test  | CREATE TABLE `test` (
  `id` int NOT NULL,
  `c1` int DEFAULT NULL,
  `c2` int DEFAULT NULL,
  `c3` int DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `c1` (`c1`),
  KEY `c2` (`c2`),
  KEY `c3` (`c3`),
  KEY `c1_2` (`c1`,`c2`,`c3`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+-------+-------------------------------------
```

```
mysql> explain select c1,c2,c3 from test group by c1,c2,c3;
+----+-------------+-------+------------+-------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table | partitions | type  | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+-------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | test  | NULL       | index | c1_2          | c1_2 | 15      | NULL |    1 |   100.00 | Using index |
+----+-------------+-------+------------+-------+---------------+------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> explain select distinct c1,c2,c3 from test;
+----+-------------+-------+------------+-------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table | partitions | type  | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+-------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | test  | NULL       | index | c1_2          | c1_2 | 15      | NULL |    1 |   100.00 | Using index |
+----+-------------+-------+------------+-------+---------------+------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.01 sec)
```

解决方案

在使用distinct或group by的时候，尽量在合理的情况下，创建可以包含所有依赖字段的索引。

3.11 为什么有时候用浮点数做等值比较查不到数据

原因分析

浮点数的等值比较问题是一种常见的浮点数问题。因为在计算机中，浮点数存储的是近似值而不是精确值，所以等值比较、数学运算等场景很容易出现预期外的情况。

MySQL中涉及浮点数的类型有float和double。如下示例中遇到的问题：

```
mysql> create table f(fnum float, dnum double);
Query OK, 0 rows affected (0.26 sec)

mysql> insert into f values(1.1, 1.2);
Query OK, 1 row affected (0.07 sec)

mysql> insert into f values(2.1, 2.2);
Query OK, 1 row affected (0.00 sec)

mysql> insert into f values(2.1, 3.2);
Query OK, 1 row affected (0.00 sec)

mysql> insert into f values(3.1, 3.2);
Query OK, 1 row affected (0.03 sec)

mysql> select * from f;
+------+------+
| fnum | dnum |
+------+------+
|  1.1 |  1.2 |
|  2.1 |  2.2 |
|  2.1 |  3.2 |
|  3.1 |  3.2 |
+------+------+
4 rows in set (0.00 sec)

mysql> select * from f where fnum = 1.1;
Empty set (0.03 sec)

mysql> select * from f where fnum < 2;
+------+------+
| fnum | dnum |
+------+------+
|  1.1 |  1.2 |
+------+------+
1 row in set (0.00 sec)
```

解决方案

1. 使用精度的方法处理，使用字段与数值的差值的绝对值小于可接受的精度的方法。示例：

```
mysql> select * from f where fnum = 0.01;
Empty set (0.00 sec)

mysql> select * from f where abs(fnum - 1.1) < 0.01;
+------+------+
| fnum | dnum |
+------+------+
|  1.1 |  1.2 |
+------+------+
1 row in set (0.00 sec)
```

2. 使用定点数类型(DECIMAL)取代浮点数类型，示例：

```
mysql> create table d(d1 DECIMAL(5,2), d2 DECIMAL(5,2));
Query OK, 0 rows affected (0.09 sec)

mysql> insert into d values(1.1, 1.2);
Query OK, 1 row affected (0.02 sec)

mysql> insert into d values(2.1, 2.2);
Query OK, 1 row affected (0.01 sec)

mysql> insert into d values(3.1, 3.2);
Query OK, 1 row affected (0.01 sec)

mysql> select * from d;
+------+------+
| d1   | d2   |
+------+------+
| 1.10 | 1.20 |
| 2.10 | 2.20 |
| 3.10 | 3.20 |
+------+------+
3 rows in set (0.00 sec)

mysql> select * from d where d1 = 1.1;
+------+------+
| d1   | d2   |
+------+------+
| 1.10 | 1.20 |
+------+------+
1 row in set (0.00 sec)
```

3.12 开通数据库代理后，还是有大量 select 请求分发到主节点

原因分析：

1. 读权重参数

   设置主节点和只读节点的读权重分配，可以控制读请求的分发配比，仅在存在只读节点时生效。

   例如：一主两只读，设置的读权重为1(主):2(只读1):3(只读2)，那么会按照1:2:3将读请求分发到主和只读实例上；如果将读权重设置为0:2:3，会按照2:3将请求分发的只读实例，不会将读请求分发的主实例。

   更多信息，请参见**设置读写分离权重**。

2. 事务

   事务中的SQL会发往主，若在查询语句前设置set autocommit=0也会被当做事务处理路由到主实例。

3. 连接绑定

   执行了Multi-Statements（如“insert xxx;select xxx”）当前连接的后续请求会全部路由到主节点；创建临时表的SQL会将连接绑定到主，后续此连接的请求都会到主。需断开当前连接并重新连接才能恢复读写分离。

4. 自定义变量

SQL中包含了自定义变量的语句会发到主节点。

5. 带锁的读操作（如SELECT for UPDATE）会被路由到主节点。
6. 通过Hint指定SQL发往主实例或只读实例

   在读写分离权重分配体系之外，在SQL开头添加hint注释进行强制路由：

   - /*FORCE_MASTER*/：强制路由到主节点
   - /*FORCE_SLAVE*/：强制路由到只读节点

   Hint注释仅作为路由建议，非只读SQL、事务中的场景不能强制路由到只读节点。
7. 会话一致性特性

   同一个会话内，对于写入还没有同步到只读节点的数据，读请求也会发送到主节点。

   更多信息，请参见**一致性级别介绍**。

3.13 表空间膨胀问题

场景描述

在使用GaussDB(for MySQL)过程中，经常遇到表空间膨胀问题，例如：表中只有11774行数据，表空间却占用49.9GB，将该表导出到本地只有800M。

原因分析

场景1：DRS全量迁移阶段并行迁移导致

原因：DRS在全量迁移阶段，为了保证迁移性能和传输的稳定性，采用了行级并行的迁移方式。当源端数据紧凑情况下，通过DRS迁移到云上GaussDB(for MySQL)后，可能会出现数据膨胀现象，使得磁盘空间使用远大于源端。

场景2：大量删除操作后在表空间留下碎片所致

原因：当删除数据时，mysql并不会回收被删除数据占据的存储空间，而只做标记删除，尝试供后续复用，等新的数据来填补相应空间，如果短时间内没有数据来填补这些空间，就造成了表空间膨胀，形成大量碎片；

可以通过如下SQL语句，查询某个表详细信息，DATA_FREE字段表示表空间碎片大小：

```
select * from information_schema.tables where table_schema='db_name' and table_name = 'table_name'\G
```

```
mysql> select * from information_schema.tables where table_schema='m[redacted]' and table_name='[redacted]'\G
*************************** 1. row ***************************
  TABLE_CATALOG: def
   TABLE_SCHEMA: mall19wo
     TABLE_NAME: deliveryman_track
     TABLE_TYPE: BASE TABLE
         ENGINE: InnoDB
        VERSION: 10
     ROW_FORMAT: Dynamic
     TABLE_ROWS: 11968
 AVG_ROW_LENGTH: 4479273
    DATA_LENGTH: 53607940096
MAX_DATA_LENGTH: 0
   INDEX_LENGTH: 802816
      DATA_FREE: 54668558336
 AUTO_INCREMENT: 94507
    CREATE_TIME: 2022-06-28 23:39:00
    UPDATE_TIME: 2022-07-07 11:03:22
     CHECK_TIME: NULL
TABLE_COLLATION: utf8mb4_general_ci
       CHECKSUM: NULL
 CREATE_OPTIONS: row_format=DYNAMIC
  TABLE_COMMENT: 骑手轨迹
1 row in set (0.00 sec)
```


解决方案

针对表空间膨胀的问题，可以进行表空间优化整理，从而缩小空间，执行如下SQL命令：

```
optimize table table_name;
```

说明

optimize table命令会有短暂锁表操作，所以进行表空间优化时，建议避开业务高峰期，避免影响正常业务的进行。

3.14 MySQL 创建用户提示服务器错误(ERROR 1396)

场景描述

用户账号在控制台界面上消失，创建不了同名账号，但使用账号名和旧密码还能连接。

创建用户失败的报错信息：

```
ERROR 1396 (HY000): Operation CREATE USER failed for xxx。
```

问题分析

1. 查询确认后，发现消失的账号在mysql.user表中已经被删除，故控制台不再显示；
2. 使用账号名和旧密码还能连接登录，说明使用的是**delete from mysql.user**方式删除用户。使用这种方式删除用户，需要执行**flush privileges**后，才会清理内存中相关数据，该用户才彻底不能登录。
3. 使用**delete from mysql.user**方式删除用户，无法重新创建相应账户（报错ERROR 1396），原因是内存中相关数据仍然存在。

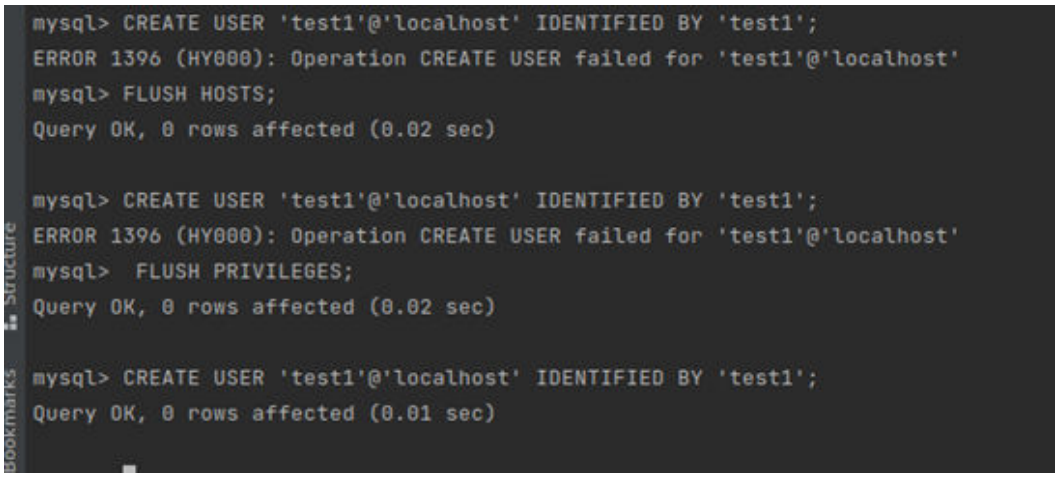
```
mysql> CREATE USER 'test1'@'localhost' IDENTIFIED BY 'test1';
Query OK, 0 rows affected (0.03 sec)

mysql> DELETE FROM mysql.user WHERE Host='localhost'AND User='test1';
Query OK, 1 row affected (0.02 sec)

mysql> CREATE USER 'test1'@'localhost' IDENTIFIED BY 'test1';
ERROR 1396 (HY000): Operation CREATE USER failed for 'test1'@'localhost'
```

正确删除用户的方式为**drop user**语句，注意以下几点：

- **drop user**语句可用于删除一个或多个用户，并撤销其权限。
- 使用**drop user**语句必须拥有MySQL数据库的DELETE权限或全局CREATE USER权限。
- 在**drop user**语句的使用中，若没有明确地给出账户的主机名，则该主机名默认为“%”。

故障场景恢复示例：

创建用户后用**delete**删除用户，再创建同名用户时报错ERROR 1396。通过执行**flush privileges**后，可正常创建同名用户。



解决方案

- 方式一（推荐）：在业务低峰期，使用管理员账户执行**drop user user_name**删除用户，再重新创建该用户，修复该问题。
- 方式二：在业务低峰期，使用管理员账户执行**flush privileges**后，再重新创建该用户，修复该问题。建议开启数据库全量sql洞察功能，便于分析是哪个客户端删除了用户。

3.15 执行 alter table xxx discard/import tablespace 报错

场景描述

在GaussDB(for MySQL)中执行alter table xxx discard/import tablespace会报错：ERROR 3658 (HY000)： Feature IMPORT/DISCARD TABLESPACE is unsupported ().

原因分析

alter table xxx discard/import tablespace是社区MySQL一种基于本地.ibd的表空间文件物理的做数据表内容替换（多用于数据迁移、备份恢复等）的方法。

GaussDB(for MySQL)是存储计算分离架构，实际数据存储于共享存储上，本地没有.ibd文件，所以不支持相应的物理操作。

解决方案

使用其他如导入导出、DRS同步、备份恢复等方式做数据表内容的替换。

3.16 数据库报错 Native error 1461 的解决方案

场景描述

MySQL用户通常在并发读写、大批量插入sql语句或数据迁移等场景出现如下报错信息：

mysql_stmt_prepare failed! error(1461)Can't create more than max_prepared_stmt_count statements (current value: 16382)

故障分析

“max_prepared_stmt_count”的取值范围为0~1048576，默认为“16382”，该参数限制了同一时间在mysqld上所有session中prepared语句的上限，用户业务超过了该参数当前值的范围。

解决方案

请您调大“max_prepared_stmt_count”参数的取值，建议调整为“65535”。

3.17 创建表失败报错 Row size too large 的解决方案

场景描述

MySQL用户创建表失败，出现如下报错信息：

Row size too large. The maximum row size for the used table type, not counting BLOBs, is 65535. This includes storage overhead, check the manual. You have to change some columns to TEXT or BLOBs

故障分析

“varchar”的字段总和超过了65535，导致创建表失败。

解决方案

1. 缩减长度，如下所示。
   ```
   CREATE TABLE t1 (a VARCHAR(10000),b VARCHAR(10000),c VARCHAR(10000),d VARCHAR(10000),e VARCHAR(10000),f VARCHAR(10000) ) ENGINE=MyISAM  CHARACTER SET latin1;
   ```
2. 请参考官方文档修改一个字段为TEXT类型。

3.18 Order by limit 分页出现数据重复问题

问题现象

对一个表执行排序，并对排序结果进行分页，得到的结果不符合预期。

假设有一个名为商品(merchants)的表，只有一个商品id和商品种类category两个字段，表结构如下：

```
mysql> show create table merchants;
+-----------
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------+
| Table     | Create
Table
                                            |
+-----------
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------+
| merchants | CREATE TABLE `merchants` (
  `id` int NOT NULL AUTO_INCREMENT,
  `category` int DEFAULT NULL,
  PRIMARY KEY (`id`)
```

```
) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci |
+-----------
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

执行如下SQL，查看表中的内容。

```
mysql>  select * from merchants;
+----+----------+
| id | category |
+----+----------+
|  1 |        1 |
|  2 |        3 |
|  3 |        2 |
|  4 |        2 |
|  5 |        1 |
|  6 |        2 |
|  7 |        3 |
|  8 |        3 |
|  9 |        2 |
| 10 |        1 |
+----+----------+
10 rows in set (0.00 sec)
```

执行如下SQL，对商品按照类别(category)字段进行排序。

```
mysql> select * from merchants order by category;
+----+----------+
| id | category |
+----+----------+
|  1 |        1 |
|  5 |        1 |
| 10 |        1 |
|  3 |        2 |
|  4 |        2 |
|  6 |        2 |
|  9 |        2 |
|  2 |        3 |
|  7 |        3 |
|  8 |        3 |
+----+----------+
10 rows in set (0.00 sec)
```

执行如下SQL，将排序结果用limit分页，每页两行数据。

```
mysql> select * from merchants order by category limit 0,2;
+----+----------+
| id | category |
+----+----------+
|  1 |        1 |
|  5 |        1 |
+----+----------+
2 rows in set (0.00 sec)

mysql> select * from merchants order by category limit 2,2;
+----+----------+
| id | category |
+----+----------+
|  1 |        1 |
|  9 |        2 |
+----+----------+
2 rows in set (0.00 sec)
```

可以看到，第二页的数据出现了错误。按照没有分页时的排法，第二页应该显示为id为10和id为3的行，但实际结果这里是id为1和9的行。

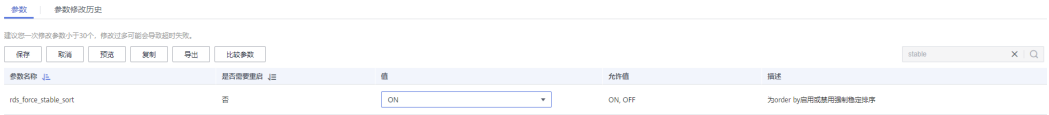
原因分析

优化器在遇到order by limit语句的时候，做了一个优化，内部使用priority queue结构做排序，该排序方式属于不稳定排序算法，筛选出limit n的结果后就直接返回，不能保证有序性。

解决方案

- 方案1：可以在需要排序的字段上加上索引。如案例中，alter table ratings add index idx_category (category);
- 方案2：可以在排序语句的order by后面加入主键列。如案例中，select * from ratings order by category, id limit 2,2;
- 方案3：可以在GaussDB(for MySQL)控制台参数修改页面，开启参数“rds_force_stable_sort”。该参数开启后，将强制使用稳定排序算法，确保排序结果的稳定。

图 3-1 设置参数 rds_force_stable_sort



4 参数类

4.1 客户端修改全局参数失败

场景描述

客户端修改全局参数，报错“ERROR 1227 (42000): Access denied”。

原因分析

GaussDB(for MySQL)不支持在数据库中执行修改全局参数的命令，需要通过管理控制台修改。

解决方案

登录管理控制台修改参数。

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB(for MySQL)”。

步骤4 在“实例管理”页面，选择指定的实例，单击实例名称，进入实例的基本信息页面。

步骤5 在左侧导航栏中选择“参数修改”，在“参数”页签查看并修改对应参数。

----结束

4.2 客户端超时参数设置导致连接超时退出

场景描述

使用数据库时，经常遇到连接退出，导致后续语句执行失败的情况。

原因分析

在使用连接器或API连接数据库时，客户端会有一些默认的参数配置。其中有一些比较重要的参数如socketTimeout、connectTimeout等，会影响客户端连接的超时时间。如果超过这个时间，一直没使用的连接就会断开。

解决方案

- 将socketTimeout、connectTimeout等参数的默认值调整为合适的值。
- 在程序中注意处理断线重连的功能。
- **推荐直接使用连接池**。

4.3 修改全局变量成功但未生效

场景描述

使用Console上的参数修改功能修改long_query_time成功，但未生效。

原因分析

使用Console修改参数时，系统实际使用“set global 变量名=新的变量值;”修改全局参数。

在使用set global命令修改全局变量值时需要注意，该参数在当前连接和已经连接上数据库的其他连接中是不生效的，只对新连接生效，所以此时将所有连接断开重连，即可看到变量修改生效。

示例

举例中使用的是命令的方式做描述。

1. 创建会话1。

```
# 查看参数值。
show variables like 'long_query_time';
+-----------------+-----------+
| Variable_name   | Value     |
+-----------------+-----------+
| long_query_time | 10.000000 |
+-----------------+-----------+
1 row in set (0.08 sec)
# 修改变量值
set global long_query_time=1;
Query OK, 0 rows affected (0.02 sec)
# 重新查看，发现未生效。
show variables like 'long_query_time';
+-----------------+-----------+
| Variable_name   | Value     |
+-----------------+-----------+
| long_query_time | 10.000000 |
+-----------------+-----------+
1 row in set (0.01 sec)
```

2. 创建会话2。

```
show variables like 'long_query_time';
+-----------------+-----------+
| Variable_name   | Value     |
+-----------------+-----------+
| long_query_time | 10.000000 |
```



```
+-----------------+-----------+
1 row in set (0.01 sec)
```

3. 在会话1中执行。

```
#会话1中执行set global后，再次查看，变量未生效。
show variables like 'long_query_time';
+-----------------+-----------+
| Variable_name   | Value     |
+-----------------+-----------+
| long_query_time | 10.000000 |
+-----------------+-----------+
1 row in set (0.01 sec)
# 会话1断开，重新连接，发现修改生效。
show variables like 'long_query_time';
+-----------------+----------+
| Variable_name   | Value    |
+-----------------+----------+
| long_query_time | 1.000000 |
+-----------------+----------+
1 row in set (0.00 sec)
```

4. 会话2断开，重新连接，发现修改生效。

```
show variables like 'long_query_time';
+-----------------+----------+
| Variable_name   | Value    |
+-----------------+----------+
| long_query_time | 1.000000 |
+-----------------+----------+
1 row in set (0.01 sec)
```

4.4 GaussDB(for MySQL) timeout 相关参数简介

MySQL中有多种timeout参数，GaussDB(for MySQL)也将相关参数提供给用户设置，如下表：

表 4-1 参数说明

参数名称	修改是否需要重启	参数含义
connect_timeout	否	控制客户端和MySQL服务端在建连接时，服务端等待三次握手成功的超时时间（秒），网络状态较差时，可以调大该参数。
innodb_flush_log_at_timeout	否	每N秒写入并刷新日志。当innodb_flush_log_at_trx_commit值为2时，此设置有效。
innodb_lock_wait_timeout	否	该变量控制innodb事务获取行锁等待的最长时间（秒），如果超过该时间还未获取到锁资源，则会返回执行失败。
parallel_queue_timeout	否	请求并行执行的查询的等待时间。如果超过该等待时间后，系统中并行执行的线程数仍然大于parallel_max_threads，则不再等待而进入单线程执行。
lock_wait_timeout	否	试图获得元数据锁的超时时间（秒）。

参数名称	修改是否需要重启	参数含义
net_read_timeout	否	中止读数据之前从一个连接等待更多数据的秒数。
net_write_timeout	否	中止写之前等待一个块被写入连接的秒数。
interactive_timeout	否	服务器在关闭交互式连接之前等待活动的秒数。
wait_timeout	否	服务器关闭连接之前等待非交互式连接活动的秒数。

5 性能资源类

5.1 CPU 使用率高问题排查与优化

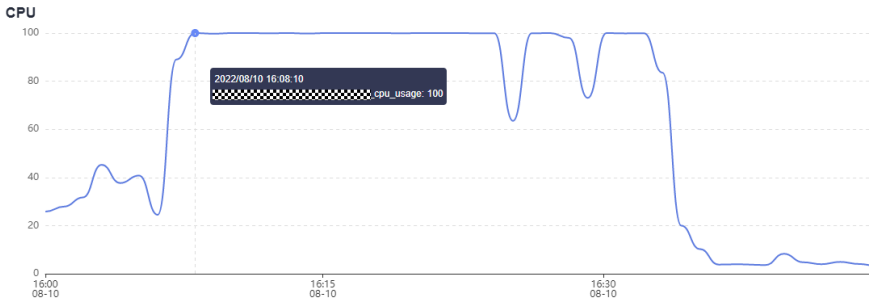
场景描述

业务侧GaussDB(for MySQL)实例的SQL执行速率在16:08分左右开始变慢，应用有超时的报错。

原因分析

1. 查看CPU使用率监控指标，发现在16:08分左右实例的CPU使用率开始飙升到100%，且一直持续在高位线。

   图 5-1 CPU 使用率

2. 查看QPS、慢SQL数以及活跃连接数监控指标，发现在16:08分左右QPS突增，活跃连接数上涨，最终业务侧有较多的慢SQL产生。

图 5-2 QPS

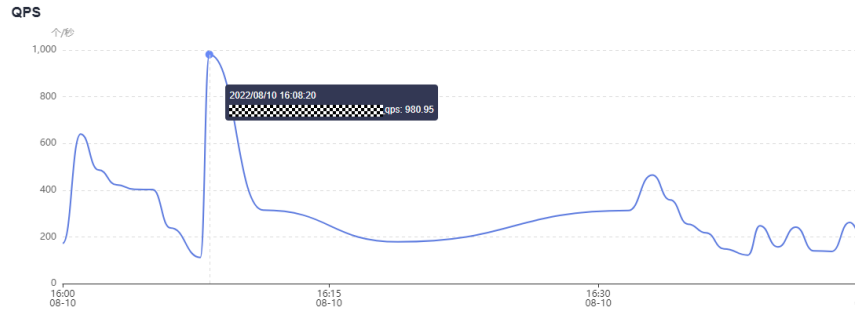


图 5-3 活跃连接数

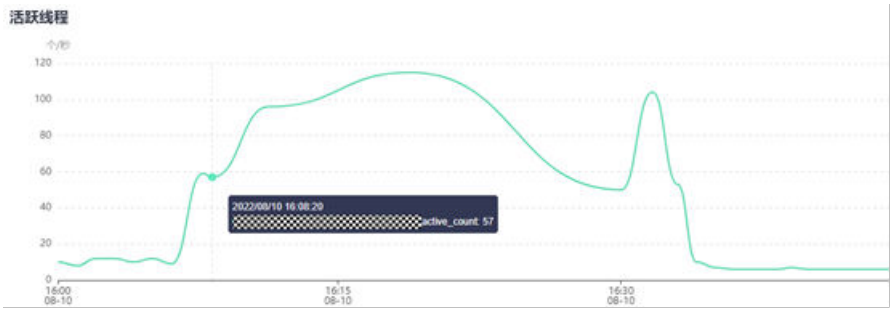
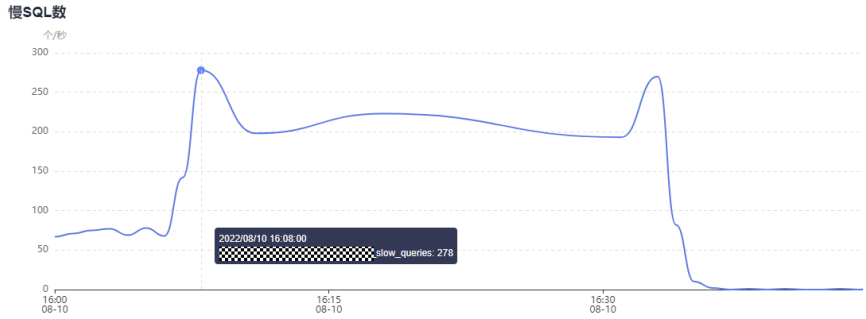
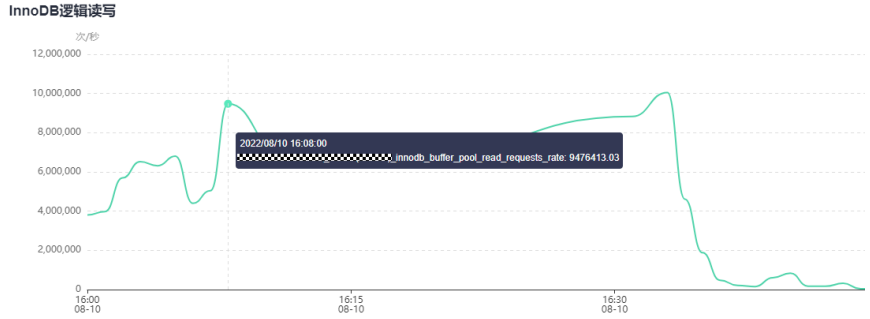


图 5-4 慢 SQL 数

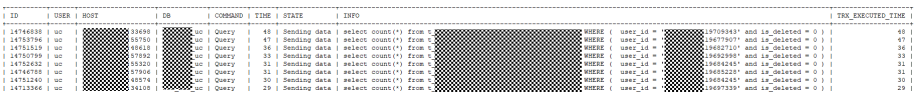


3. 分析业务类型，查看16:08分前左右InnoDB的逻辑读速率有突增，且与慢SQL的速率趋势相似。

图 5-5 InnoDB 逻辑读速率



4. 登录实例，查看实话会话，发现大量会话在执行**SELECT COUNT(*)**。



EXPLAIN确认该SQL的执行计划，发现走全表扫描且单条扫描行数在35万+，其并未走索引。

```
mysql> explain select count(*) from t_[redacted] WHERE (  user_id = 'sx5g316601196779O7' and is_deleted = 0 );
+----+-------------+-------------------------------------+------------+------+-----------------+-----------------+---------+-------+--------+----------+-------------+
| id | select_type | table                               | partitions | type | possible_keys   | key             | key_len | ref   | rows   | filtered | Extra       |
+----+-------------+-------------------------------------+------------+------+-----------------+-----------------+---------+-------+--------+----------+-------------+
|  1 | SIMPLE      | t_[redacted]                        | NULL       | ref  | IDX[redacted]_USERID | IDX_[redacted]_USERID | 2       | const | 356753 |    10.00 | Using where |
+----+-------------+-------------------------------------+------------+------+-----------------+-----------------+---------+-------+--------+----------+-------------+
```

5. 进一步查看该表的表结构，发现该表仅对字段“is_deleted”添加了一个索引“IDX_XX_USERID”，因此上述查询无索引可选。建议业务侧给字段“idx_user_id”新增索引后，实例在16:37分左右CPU下降到正常水平，业务恢复。

解决方案

1. 建议新上业务时，提前对关键SQL通过**EXPLAIN**、SQL诊断等工具进行执行计划分析，根据优化建议添加索引，避免全表扫描。
2. 业务量突增的高并发造成CPU占用率高，可以考虑升级实例规格或使用独享型资源避免出现CPU资源争抢，或者创建只读实例进行读写分离减轻主实例负载。
3. 通过**show processlist**查看当前会话信息来辅助定位：运行状态为Sending data、Copying to tmp table、Copying to tmp table on disk、Sorting result、Using filesort的查询会话可能均包含性能问题。
4. 应急场景可以借助SQL限流以及KILL会话功能来临时kill规避“烂SQL”。

5.2 内存使用超限风险与优化

GaussDB(for MySQL)内存说明

GaussDB(for MySQL)的内存大体可以分为GLOBAL级的共享内存和SESSION级的私有内存两部分：

- 共享内存是实例创建时根据参数即分配的内存空间，并且是所有连接共享的。
- 私有内存用于每个连接到GaussDB(for MySQL)服务器时才分配各自的缓存，且只有断开连接才会释放。

低效的SQL语句或数据库参数设置不当都可能会导致内存利用率升高，遇到突发业务高峰时，可能会导致云数据库内存OOM（Out Of Memory）。

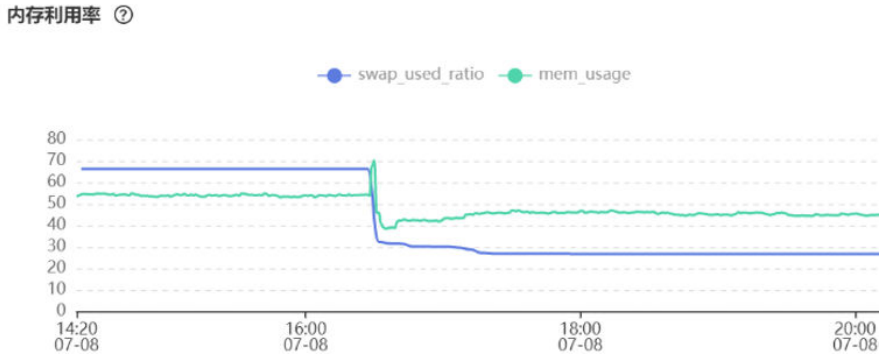
场景描述

GaussDB(for MySQL)实例在16:30分内存使用率突增，触发OOM后实例重启。

原因分析

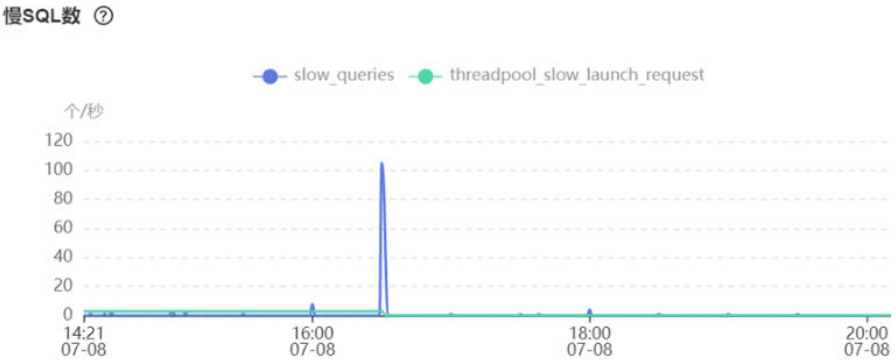
1. 查看内存利用率监控指标，实例的内存使用率在16:30左右率突增，触发OOM后实例重启，内存使用率骤降。

图 5-6 内存利用率



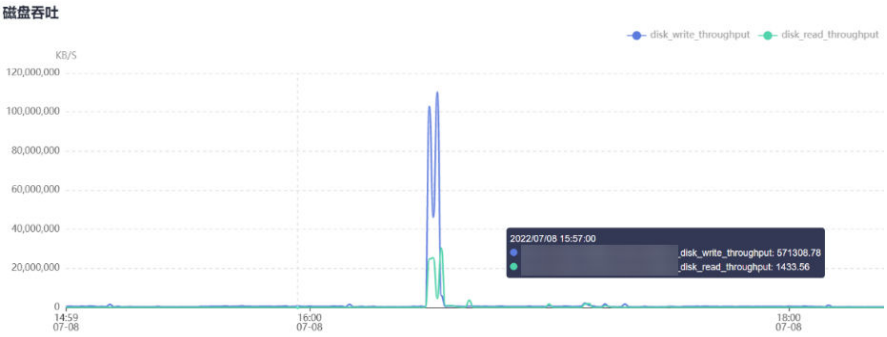
2. 查看该时间段慢SQL数监控指标，确认该时间段慢SQL数量突增。

图 5-7 慢 SQL 数



3. 查看磁盘吞吐相关指标，发现磁盘此时有大量读写操作。

图 5-8 磁盘吞吐



4. 分析对应时间点的慢日志记录，该时间点有大量的多值批量插入语句，该插入方式会导致每个会话申请较多的SESSION级内存，并发高，很容易引起实例OOM。

图 **5-9** 慢日志

```
INSERT INTO [redacted]
(`[redacted]y_id`,`id_[redacted]`,`op[redacted]r`,`[redacted]r_id`,`[redacted]`,`create_time`,`update_time`) VALUES
(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),
(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),
(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),
(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),
(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),
(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),
(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),
(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),
(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),
(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),
(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),
(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),
(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,N,N),(N,N,N,N,N,NN-N-N N:N:NNN-N-N
N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN
N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N
N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN
N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N
N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN
N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N
N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN
N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N
N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN
N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N
N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN
N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N
N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN
N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N
N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN
N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N
N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN
N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N
N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN
N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N
N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:NNN-N-N N:N:N&#39;);
```

解决方案

1. 针对多值插入方式引起的OOM，建议减少单次插入数据量，分多次插入，且及时断开重连会话以释放内存。可执行**show full processlist**查看是否有明显占用内存高的会话。
2. 合理设置SESSION级内存参数大小，可大体根据全局内存+会话级内存*最大会话数来预估可能最大的内存。注意开启“performance_schema”也会带来内存开销。
3. 升级实例规格，将内存利用率维持在合理范围，防止业务突增导致实例OOM。

5.3 表空间膨胀问题

场景描述

在使用GaussDB(for MySQL)过程中，经常遇到表空间膨胀问题，例如：表中只有11774行数据，表空间却占用49.9GB，将该表导出到本地只有800M。

原因分析

场景1：DRS全量迁移阶段并行迁移导致

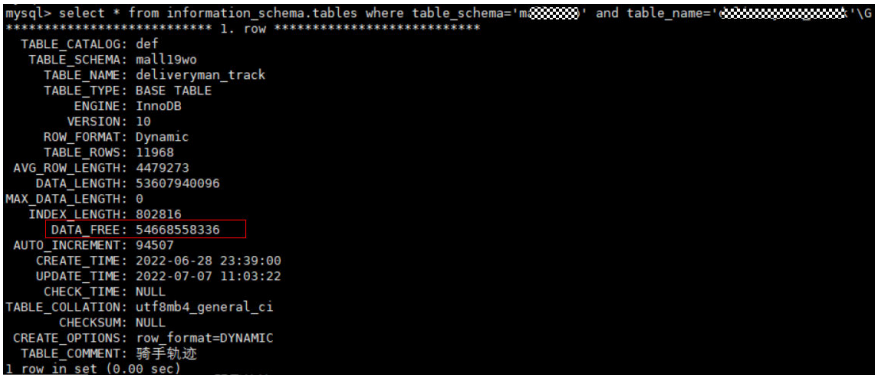
原因：DRS在全量迁移阶段，为了保证迁移性能和传输的稳定性，采用了行级并行的迁移方式。当源端数据紧凑情况下，通过DRS迁移到云上GaussDB(for MySQL)后，可能会出现数据膨胀现象，使得磁盘空间使用远大于源端。

场景2：大量删除操作后在表空间留下碎片所致

原因：当删除数据时，MySQL并不会回收被删除数据占据的存储空间，而只做标记删除，尝试供后续复用，等新的数据来填补相应空间，如果一时半会，没有数据来填补这些空间，就造成了表空间膨胀，形成大量碎片。

可以通过如下SQL语句，查询某个表详细信息，DATA_FREE字段表示表空间碎片大小：

1. 更新统计信息
   ```
   analyze table db_name.table_name;
   ```
2. 查看碎片大小
   ```
   select * from information_schema.tables where table_schema='db_name' and table_name = 'table_name'\G;
   ```



解决方案

针对表空间膨胀的问题，可以进行表空间优化整理，从而缩小空间，执行如下SQL命令：

```
optimize table table_name;
```

optimize table命令会有短暂锁表操作，所以进行表空间优化时，建议避开业务高峰期，避免影响正常业务的进行。

5.4 GaussDB(for MySQL)只读节点磁盘占用远超主节点

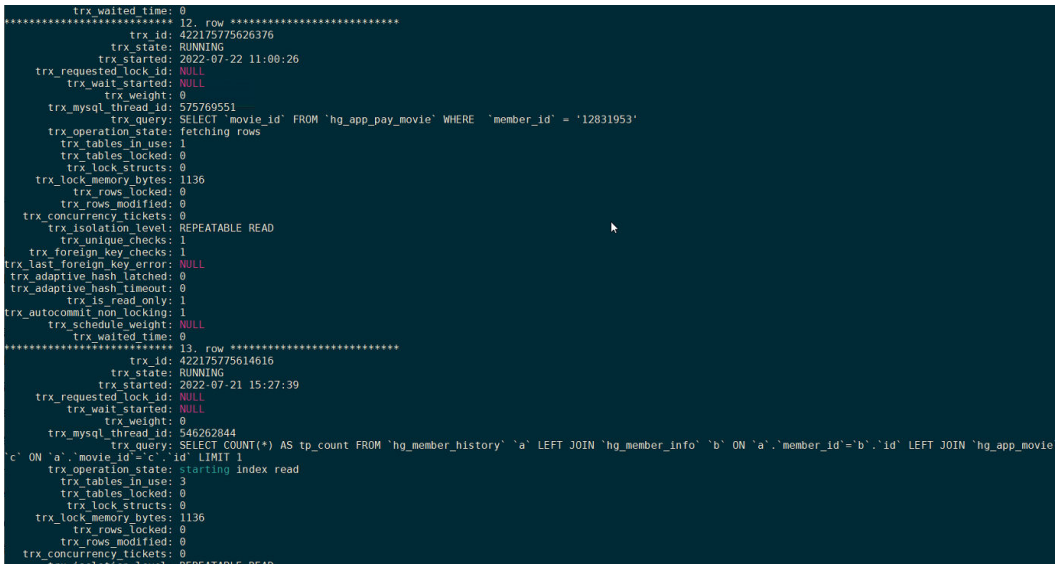
场景描述

GaussDB(for MySQL)只读节点的磁盘占用比主节点高195GB。

原因分析

磁盘空间计算方式为：共享存储中占用空间大小+Binlog占用空间大小+数据盘(存放临时表)占用空间大小

排查只读节点上运行的事务：



发现有一直未提交的长事务，如上图所示（事务一天前开始），该长事务产生的临时表一直未清理，导致磁盘占用高。

解决方案

- 方式一：等待事务提交后，临时表会自动被清理，只读实例的磁盘占用恢复。
- 方式二：kill相应会话，停止长事务。

5.5 冷热数据问题导致 SQL 执行速度慢

场景描述

从自建MySQL或友商MySQL迁移到云上GaussDB(for MySQL)实例，发现同一条SQL语句执行性能远差于原数据库。

原因分析

同一条SQL语句在数据库中执行第一次和第二次可能会性能差异巨大，这是由数据库的buffer_pool机制决定的：

- 第一次执行时，数据在磁盘上，称之为冷数据，读取需要一定的耗时。
- 读取完，数据会被存放于内存的buffer_pool中，称为热数据，读取迅速；对于热数据的访问速度极大的超过冷数据，所以当数据是热数据时，SQL语句的执行速度会远快于冷数据。

该场景中，源端数据库中常用的数据一般是热数据，所以访问时速度极快。当数据迁移到云上GaussDB(for MySQL)时，第一次执行同样的SQL语句，很可能是冷数据，就会访问较慢，但再次访问速度就会得到提升。

解决方案

该场景是正常现象，在同一个数据库中，我们经常会遇到第一次执行一条语句时很慢，但再次执行就很快，也是因为受到了buffer_pool的冷热数据原理的影响。

5.6 复杂查询造成磁盘满

场景描述

主机或只读节点偶尔出现磁盘占用高或磁盘占用满，其他只读节点磁盘空间占用正常。

原因分析

MySQL内部在执行复杂SQL时，会借助临时表进行分组（group by）、排序（order by）、去重（distinct）、Union等操作，当内存空间不够时，便会使用磁盘空间。

排查思路：

1. 因为其他只读节点磁盘占用空间正常，且是偶尔出现，说明该实例磁盘占用高，与承载的业务相关。
2. 获取该实例的慢日志，分析磁盘占用高期间，是否有对应的慢SQL。
3. 如果有慢SQL，执行**explain** *[慢SQL语句]*，分析相应慢SQL语句。
4. 观察explain语句输出的extra列，是否有using temporary、using filesort，如果有，说明该语句用到了临时表或临时文件，数据量大的情况下，会导致磁盘占用高。

解决方案

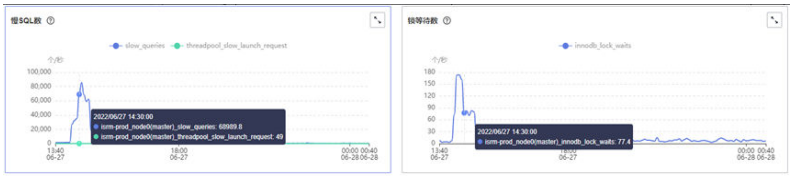
1. 复杂查询语句导致磁盘打满，建议客户从业务侧优化响应查询语句，常见优化措施：
   - 加上合适的索引。
   - 在where条件中过滤更多的数据。
   - 重写SQL，优化执行计划。
   - 如果不得不使用临时表，那么一定要减少并发度。
2. 临时规避措施：考虑业务侧优化复杂查询语句需要一定时间，可以通过临时扩容磁盘空间规避。

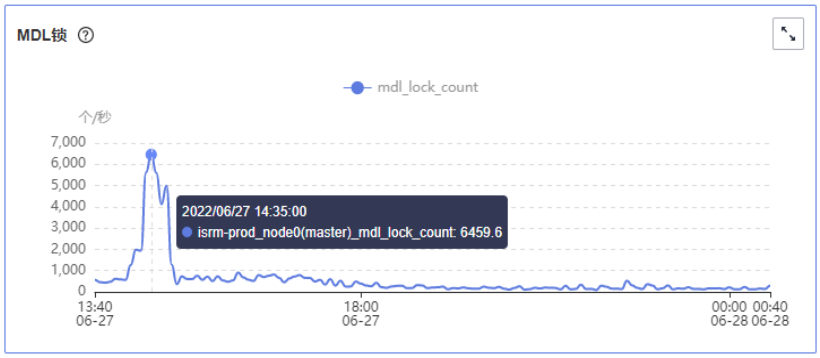
5.7 业务死锁导致响应变慢

场景描述

14点~15点之间数据库出现大量行锁冲突，内核中大量update/insert会话在等待行锁释放，导致CPU使用率达到70%左右，数据库操作变慢。

查看CES指标行锁等待个数、MDL锁数量，下图仅供参考：





发生死锁的表：

```
********* 1. row *********
Table: table_test Create Table: CREATE TABLE table_test(
...
CONSTRAINT act_fk_exe_parent FOREIGN KEY (parent_id_) REFERENCES act_ru_execution (id_) ON DELETE CASCADE,
CONSTRAINT act_fk_exe_procdef FOREIGN KEY (proc_def_id_) REFERENCES act_re_procdef (id_),
CONSTRAINT act_fk_exe_procinst FOREIGN KEY (proc_inst_id_) REFERENCES act_ru_execution (id_) ON DELETE CASCADE ON UPDATE CASCADE, CONSTRAINT act_fk_exe_super FOREIGN KEY (super_exec_) REFERENCES act_ru_execution (id_) ON DELETE CASCADE ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
```

原因分析

1. 部分表发生死锁，导致CPU一定幅度抬升。
2. 死锁的表中有大量的外键，这些表的记录在更新时，不仅需要获取本表的行锁，还需要检查外键关联表的记录，获取相应锁。高并发情况下，比普通表更容易锁冲突或死锁，详解**官方文档**。
3. 当MySQL检查到死锁的表时，会进行事务的回滚。其影响范围不仅是某个表，还会影响外键所在的表，最终导致数据库相关操作变慢。

解决方案

建议排查并优化死锁表相关的业务，业务上合理使用外键，避免更新冲突，避免产生死锁。

5.8 GaussDB(for MySQL)实例 CPU 升高定位思路

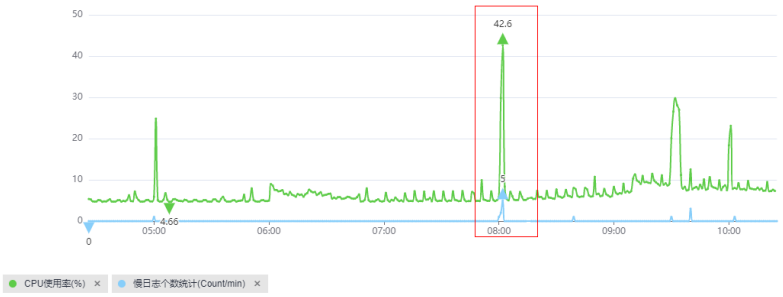
GaussDB(for MySQL)实例CPU升高或100%，引起业务响应慢，新建连接超时等。

场景 1 慢查询导致 CPU 升高

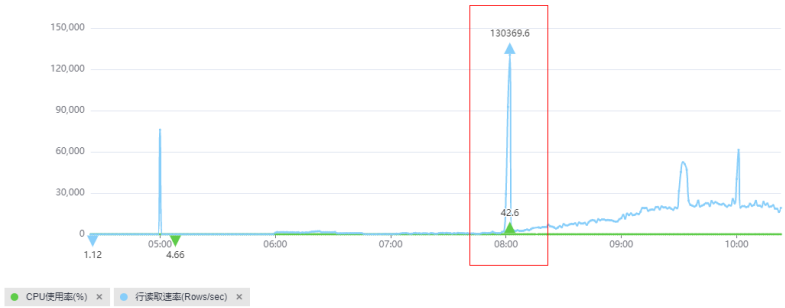
问题原因：大量慢SQL导致实例CPU升高，需要优化相应的慢SQL。

排查思路：

查看CPU使用率和慢日志个数统计监控指标。



- 如果慢日志个数很多，且与CPU曲线吻合，可以确定是慢SQL导致CPU升高。
- 如果慢日志个数不多，但与CPU使用率基本一致，进一步查看行读取速率指标是否与CPU曲线吻合。



如果吻合，说明是少量慢SQL访问大量行数据导致CPU升高：由于这些慢SQL查询执行效率低，为获得预期的结果需要访问大量的数据导致平均IO高，因此在QPS并不高的情况下（例如网站访问量不大），也会导致实例的CPU使用率偏高。

解决方案：

1. 根据CPU使用率过高的时间点，查看对应时间段的慢日志信息。
2. 重点关注扫描行数、返回结果行数超过百万级别的慢查询，以及锁等待时间长的慢查询。
3. 慢查询用户可自行分析，或使用数据管理服务(DAS)的**SQL诊断工具**对慢查询语句进行诊断。
4. 使用数据库代理+只读节点架构，实现读写分离。只读节点专门负责查询，减轻主库压力，提升数据库吞吐能力，详见**读写分离简介**。
5. 通过分析数据库执行中的会话来定位执行效率低的SQL。
   a. 连接数据库。
   b. 执行**show full processlist;**。
   c. 分析执行时间长、运行状态为Sending data、Copying to tmp table、Copying to tmp table on disk、Sorting result、Using filesort的会话，均可能存在性能问题，通过会话来分析其正在执行的SQL。

场景 2 连接和 QPS 升高导致 CPU 上升

问题原因：业务请求增高导致实例CPU升高，需要从业务侧分析请求变化的原因。

排查思路：

查看QPS、当前活跃连接数、数据库总连接数、CPU使用率监控指标是否吻合。

QPS的含义是每秒查询数，QPS和当前活跃连接数同时上升，且QPS和CPU使用率曲线变化吻合，可以确定是业务请求增高导致CPU上升，如下图：

该场景下，SQL语句一般比较简单，执行效率也高，数据库侧优化余地小，需要从业务源头优化。

解决方案：

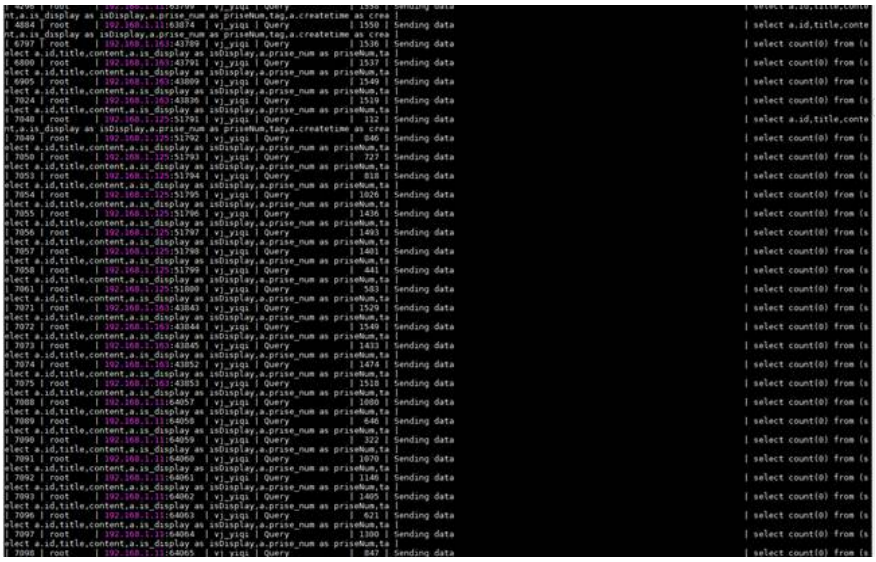
1. 单纯的QPS高导致CPU使用率过高，往往出现在实例规格较小的情况下，建议升级实例CPU规格。
2. 优化慢查询，优化方法参照**场景1 慢查询导致CPU升高**的解决方案。若优化慢查询后效果不明显，建议升级实例CPU规格。
3. 对于数据量大的表，建议通过分库分表减小单次查询访问的数据量。
4. 使用数据库代理+只读节点架构，实现读写分离。只读节点专门负责查询，减轻主库压力，提升数据库吞吐能力，详见**读写分离简介**。

5.9 大并发慢查询导致 CPU 资源耗尽问题

场景描述

数据库实例上存在大量并发的select count(0)慢操作，系统CPU耗尽，随时有宕机的风险。

执行**Show processlist**，显示存在多次并发执行select count(0)进程信息：



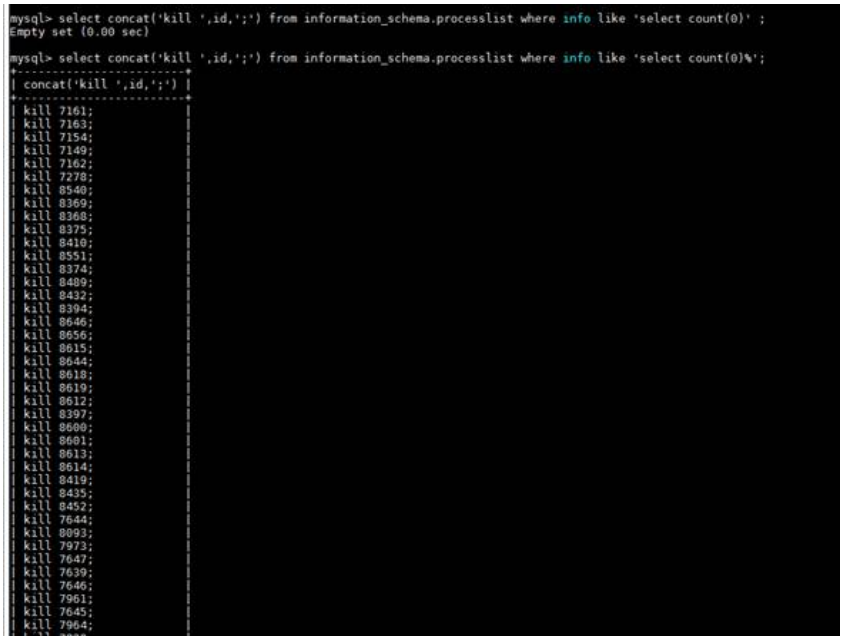
原因分析

应用端大并发触发select count(0)慢操作，导致系统CPU资源耗尽。

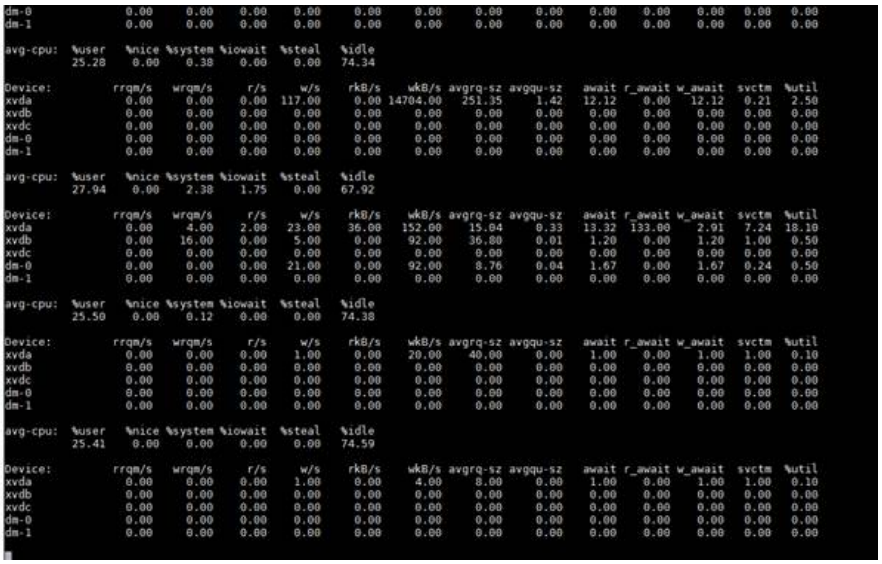
解决方案

步骤1 申请kill权限，间歇性批量执行kill select count(0)慢操作，定位select count(0)触发来源，停止来源，并拆分优化sql。

批量kill动作：



步骤2 CPU idle恢复:



```
dm-0              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
dm-1              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          25.28    0.00    0.38    0.00    0.00   74.34

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
xvda              0.00     0.00    0.00  117.00     0.00 14704.00   251.35     1.42   12.12    0.00   12.12   0.21   2.50
xvdb              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
xvdc              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
dm-0              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
dm-1              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          27.94    0.00    2.38    1.75    0.00   67.92

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
xvda              0.00     4.00    2.00   23.00    36.00   152.00    15.04     0.33   13.32  133.00    2.91   7.24  18.10
xvdb              0.00    16.00    0.00    5.00     0.00    92.00    36.80     0.01    1.20    0.00    1.20   1.00   0.50
xvdc              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
dm-0              0.00     0.00    0.00   21.00     0.00    92.00     8.76     0.04    1.67    0.00    1.67   0.24   0.50
dm-1              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          25.50    0.00    0.12    0.00    0.00   74.38

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
xvda              0.00     0.00    0.00    1.00     0.00    20.00    40.00     0.00    1.00    0.00    1.00   1.00   0.10
xvdb              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
xvdc              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
dm-0              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
dm-1              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          25.41    0.00    0.00    0.00    0.00   74.59

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
xvda              0.00     0.00    0.00    1.00     0.00     4.00     8.00     0.00    1.00    0.00    1.00   1.00   0.10
xvdb              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
xvdc              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
dm-0              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
dm-1              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
```

----结束

6 基本使用类

6.1 查看 GaussDB(for MySQL)的存储容量

GaussDB(for MySQL)是存储计算分离架构，数据存储在共享存储系统中，共享存储容量可以通过管理控制台看到，详情请参考如下步骤操作，数据每30分钟更新一次。

操作步骤

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB(for MySQL)”。

步骤4 在实例列表中，单击目标实例名称，进入实例的“基本信息”页面。

步骤5 在基本信息页面的“存储“存储/备份空间”模块可以看到当前实例占用的共享存储容量。

说明

GaussDB(for MySQL)存储容量的计算与传统MySQL有一定的区别，与传统MySQL使用（数据大小+索引大小+空闲空间）计算的容量数据会有一定的差别。

如果要查询精确的存储使用量，可以使用管理控制台查询或者连接GaussDB(for MySQL)数据库后，执行show spaceusage;命令查看当前数据使用的存储容量，该值为精确值，非估算值。

图 6-1 查看存储容量

- 共享存储
  - 显示的使用状况就是该实例购买的包周期的共享存储容量及目前数据已占用的容量。
  - 如果已使用空间超过购买的共享存储容量，GaussDB(for MySQL)会自动扩容，无需担心磁盘满带来的业务问题。
  - 自动扩容的空间会按照按需使用的收费标准收取，建议超出后使用磁盘容量变更功能扩展包周期的存储容量。
- 备份空间：

  系统会赠送一份与包周期或按需存储容量相同大小的备份空间。

----结束

执行show spaceusage;命令查看存储容量，其值等于表数据、表预分配空间、分区预分配空间、Binlog、Redolog和Undolog之和，详情见下表：

条目	查看方式	说明
表数据	select sum(data_length +index_length+data_free) from information_schema.tables;	传统MySQL的容量计算方式，该语句依赖统计数据的精准度，在统计数据未更新时可能会有偏差。
表预分配空间	select count(*) from information_schema.tables;	每张表会预分配4MB空间，该语句查询出表的数量乘以4MB就是总的表预分配空间。
分区预分配空间	select count(*) from INFORMATION_SCHEMA.PARTITIONS where PARTITION_NAME is not null;	每个分区会预分配4MB空间，该语句查询出分区的数量乘以4MB就是总的分区预分配空间。
Binlog	show binary logs;	将所有binlog的文件大小相加。

Redolog	show lsninfo;	flushed_to_disk_lsn-truncate_lsn
Undolog	无法直接查看	需要时可咨询客服人员。

6.2 修改库名和修改表名

对于库重命名和表重命名，GaussDB(for MySQL)与社区MySQL的用法是相同的。

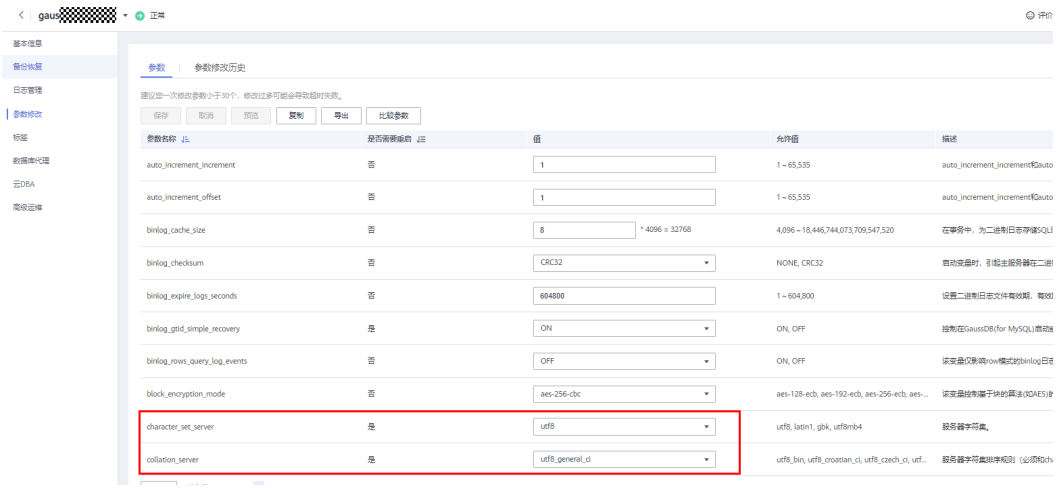
- 支持修改表名：rename table a to b; 注意，该语句是可以跨库执行的，比如：rename table da.ta to db.ta;是将ta表从da库移动到db库。
- 不支持修改库名，如果有修改库名的需求，可以先创建新的库名，然后借助rename table的跨库执行将所有表从原库移动到新库，然后删除原库。语句示例：

```
# 进入原库
use ta;
# 列出原库的所有表名
Show tables;
# 查看原库的创建语句
Show create database ta;
# 使用原库的创建语句创建新库(只改库名，其他参数照抄，这样能尽量保证新库与原库的各类参数相同)
create database tb;
# 将原库所有表移动至新库
rename table da.ta to db.ta;
rename table da.tb to db.tb;
rename table da.tc to db.tc;
...
# 删除原库
Drop database ta;
```

6.3 字符集和字符序的默认选择方式

相关变量设置

参数组中默认character_set_server=utf8、collation_server=utf8_general_ci，可以在界面修改参数值。



默认选择方式

- 在创建数据库时，如果未显式指定库的字符集和字符序，则库的字符集和字符序采用character_set_server和collation_server参数的值；如果显式指定，则使用指定的字符集和字符序。
- 在创建数据表时，如果未显式指定表的字符集和字符序，则表默认字符集和字符序使用所在数据库的字符集和字符序；如果显式指定，则使用指定的字符集和字符序。
- 在创建数据表时，如果未显式指定字段的字符集和字符序，则字段使用所在表的字符集和字符序；如果显式指定，则使用指定的字符集和字符序。

示例1：不显式指定字符集、字符序的情况下创建数据库和数据表。

```
mysql> show variables like 'character_set_server';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| character_set_server | utf8  |
+----------------------+-------+
1 row in set (0.01 sec)

mysql> show variables like 'collation_server';
+------------------+----------------+
| Variable_name    | Value          |
+------------------+----------------+
| collation_server | utf8_general_ci |
+------------------+----------------+
1 row in set (0.01 sec)

mysql> create database test_default;
Query OK, 1 row affected (0.26 sec)

mysql> show create database test_default;
+--------------+-----------------------------------------------------------------------------------------------------------------+
| Database     | Create Database                                                                                                 |
+--------------+-----------------------------------------------------------------------------------------------------------------+
| test_default | CREATE DATABASE `test_default` /*!40100 DEFAULT CHARACTER SET utf8 */ /*!80016 DEFAULT ENCRYPTION='N' */ |
+--------------+-----------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

```
mysql> create table test_default.t_default(name varchar(20));
Query OK, 0 rows affected (0.23 sec)

mysql> show create table test_default.t_default;
+-----------+-----------------------------------------------------------------------------------------------------------------+
| Table     | Create Table                                                                                                    |
+-----------+-----------------------------------------------------------------------------------------------------------------+
| t_default | CREATE TABLE `t_default` (
  `name` varchar(20) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+-----------+-----------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

示例2：显式指定库的字符集、字符序的情况下创建数据库。

```
mysql> create database test_define CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci;
Query OK, 1 row affected (0.00 sec)

mysql> show create database test_define;
+------------+-----------------------------------------------------------------------------------------------------------------------------------------------------+
| Database   | Create Database                                                                                                                                     |
+------------+-----------------------------------------------------------------------------------------------------------------------------------------------------+
| test_define | CREATE DATABASE `test_define` /*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci */ /*!80016 DEFAULT ENCRYPTION='N' */ |
+------------+-----------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)

mysql> create table test_define.t_default(name varchar(20));
Query OK, 0 rows affected (0.08 sec)

mysql> show create table test_define.t_default;
+-----------+-----------------------------------------------------------------------------------------------------------------------------------------------------+
| Table     | Create Table                                                                                                                                        |
+-----------+-----------------------------------------------------------------------------------------------------------------------------------------------------+
| t_default | CREATE TABLE `t_default` (
  `name` varchar(20) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+-----------+-----------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

示例3：显式指定表的字符集、字符序的情况下创建数据表。

```
mysql> create table test_define.t_define(name varchar(20)) CHARACTER SET utf8 COLLATE utf8_bin;
Query OK, 0 rows affected, 2 warnings (0.05 sec)

mysql> show create table test_define.t_define;
+----------+-----------------------------------------------------------------------------------------------------------------------------------------------+
| Table    | Create Table                                                                                                                                  |
+----------+-----------------------------------------------------------------------------------------------------------------------------------------------+
| t_define | CREATE TABLE `t_define` (
  `name` varchar(20) COLLATE utf8_bin DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin |
+----------+-----------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

示例4：显式指定字段的字符集、字符序的情况下创建数据表。

```
mysql> create table test_define.t_v_define(name varchar(20) CHARACTER SET gbk COLLATE gbk_bin, str char(32)) CHARACTER SET utf8 COLLATE utf8_bin;
Query OK, 0 rows affected, 2 warnings (0.06 sec)

mysql> show create table test_define.t_v_define;
+------------+-----------------------------------------------------------------------------------------------------------------------------------------------------------+
| Table      | Create Table                                                                                                                                              |
+------------+-----------------------------------------------------------------------------------------------------------------------------------------------------------+
| t_v_define | CREATE TABLE `t_v_define` (
  `name` varchar(20) CHARACTER SET gbk COLLATE gbk_bin DEFAULT NULL,
  `str` char(32) COLLATE utf8_bin DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin |
+------------+-----------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

6.4 自增字段值跳变的原因

数据表中的自增字段取值不是连续的，自增值跳变。

出现表中的自增字段取值不连续的情况，可能原因有以下几种：

- 初值与步长问题，步长不为1会导致自增字段取值不连续。

```
mysql> show variables like 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 2     |
| auto_increment_offset    | 1     |
+--------------------------+-------+
mysql> select * from auto_test1;
+----+
| id |
+----+
|  2 |
|  4 |
|  6 |
|  8 |
+----+
```

- 直接修改表的AUTO_INCREMENT，会导致自增字段取值跳变。

```
mysql> select * from animals;
+----+-----------+
| id | name      |
+----+-----------+
|  1 | dog       |
|  2 | cat       |
|  3 | penguin   |
+----+-----------+
mysql> show create table animals;
+---------+-------------------------------------------------------+
| Table   | Create Table                                          |
+---------+-------------------------------------------------------+
| animals | CREATE TABLE `animals` (
`id` mediumint NOT NULL AUTO_INCREMENT,
`name` char(30) NOT NULL,
PRIMARY KEY (`id`) )
ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8 |
+---------+-------------------------------------------------------+
mysql> alter table animals AUTO_INCREMENT=100;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0
mysql> show create table animals;
+---------+-------------------------------------------------------+
```

```
| Table   | Create Table                                          |
+---------+-------------------------------------------------------+
| animals | CREATE TABLE `animals` (
`id` mediumint NOT NULL AUTO_INCREMENT,
`name` char(30) NOT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=100 DEFAULT CHARSET=utf8         |
+---------+-------------------------------------------------------+
mysql> INSERT INTO animals (id,name) VALUES(0,'rabbit');
Query OK, 1 row affected (0.00 sec)
mysql> select * from animals;
+-----+-----------+
| id  | name      |
+-----+-----------+
|   1 | dog       |
|   2 | cat       |
|   3 | penguin   |
| 100 | rabbit    |
+-----+-----------+
9 rows in set (0.00 sec)
```

- 插入数据时直接指定自增字段的取值，会导致自增字段取值跳变。

```
mysql> select * from animals;
+----+-----------+
| id | name      |
+----+-----------+
|  1 | dog       |
|  2 | cat       |
|  3 | penguin   |
+----+-----------+
mysql> INSERT INTO animals (id,name) VALUES(100,'rabbit');
Query OK, 1 row affected (0.00 sec)
mysql> select * from animals;
+-----+-----------+
| id  | name      |
+-----+-----------+
|   1 | dog       |
|   2 | cat       |
|   3 | penguin   |
| 100 | rabbit    |
+-----+-----------+
9 rows in set (0.00 sec)
```

- 未提交的事务或回滚的事务，会导致AUTO_INCREMENT增长，但回滚后不会下降。后续如果再次插入数据就会导致数据中的自增字段发生跳变。

```
mysql> show create table auto_test1;
+------------+-----------------------------------------+
| Table      | Create Table                            |
+------------+-----------------------------------------+
| auto_test1 | CREATE TABLE `auto_test1` (
`id` int NOT NULL AUTO_INCREMENT,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------+
1 row in set (0.00 sec)
mysql> select * from auto_test1;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
+----+
mysql> begin;
Query OK, 0 rows affected (0.02 sec)
mysql> insert into auto_test1 values (0),(0),(0);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> select * from auto_test1;
+----+
```



```
| id |
+----+
|  1 |
|  2 |
|  3 |
|  4 |
|  5 |
|  6 |
+----+
6 rows in set (0.00 sec)
mysql> show create table auto_test1;
+------------+----------------------------------------+
| Table      | Create Table                           |
+------------+----------------------------------------+
| auto_test1 |
CREATE TABLE `auto_test1` (
`id` int NOT NULL AUTO_INCREMENT,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8 |
+------------+----------------------------------------+
1 row in set (0.00 sec)
mysql> rollback;
Query OK, 0 rows affected (0.05 sec)
mysql> select * from auto_test1;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
+----+
3 rows in set (0.00 sec)
mysql> show create table auto_test1;
+------------+----------------------------------------+
| Table      | Create Table                           |
+------------+----------------------------------------+
| auto_test1 | CREATE TABLE `auto_test1` (
`id` int NOT NULL AUTO_INCREMENT,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8 |
+------------+----------------------------------------+
mysql> insert into auto_test1 values (0),(0),(0);
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> select * from auto_test1;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
|  7 |
|  8 |
|  9 |
+----+
6 rows in set (0.00 sec)
mysql> show create table auto_test1;
+------------+----------------------------------------+
| Table      | Create Table                           |
+------------+----------------------------------------+
| auto_test1 | CREATE TABLE `auto_test1` (
`id` int NOT NULL AUTO_INCREMENT,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8 |
+------------+----------------------------------------+
```

- 数据插入后，AUTO_INCREMENT变化，然后删除对应的数据行，AUTO_INCREMENT不会下降，后续如果再次插入数据就会导致数据中的自增字段发生跳变。


```
mysql> show create table auto_test1;
+------------+----------------------------------------+
| Table      | Create Table                           |
+------------+----------------------------------------+
| auto_test1 | CREATE TABLE `auto_test1` (
`id` int NOT NULL AUTO_INCREMENT,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8 |
+------------+----------------------------------------+
1 row in set (0.00 sec)
mysql> select * from auto_test1;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
+----+
mysql> insert into auto_test1 values (0),(0),(0);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> select * from auto_test1;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
|  4 |
|  5 |
|  6 |
+----+
6 rows in set (0.00 sec)
mysql> show create table auto_test1;
+------------+----------------------------------------+
| Table      | Create Table                           |
+------------+----------------------------------------+
| auto_test1 | CREATE TABLE `auto_test1` (
`id` int NOT NULL AUTO_INCREMENT,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8 |
+------------+----------------------------------------+
1 row in set (0.00 sec)
mysql> delete from auto_test1 where id>3;
mysql> select * from auto_test1;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
+----+
3 rows in set (0.00 sec)
mysql> show create table auto_test1;
+------------+----------------------------------------+
| Table      | Create Table                           |
+------------+----------------------------------------+
| auto_test1 | CREATE TABLE `auto_test1` (
`id` int NOT NULL AUTO_INCREMENT,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8 |
+------------+----------------------------------------+
mysql> insert into auto_test1 values (0),(0),(0);
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> select * from auto_test1;
+----+
| id |
+----+
```

```
|  1 |
|  2 |
|  3 |
|  7 |
|  8 |
|  9 |
+----+
6 rows in set (0.00 sec)
mysql> show create table auto_test1;
+------------+----------------------------------------+
| Table      | Create Table                           |
+------------+----------------------------------------+
| auto_test1 | CREATE TABLE `auto_test1` (
`id` int NOT NULL AUTO_INCREMENT,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8 |
+------------+----------------------------------------+
```

- 因为一些原因（比如唯一键冲突），使得插入数据最终未成功的，有可能导致AUTO_INCREMENT跳变。

```
mysql> create table auto_test7(`id` int NOT NULL AUTO_INCREMENT, cred_id int UNIQUE, PRIMARY
KEY (`id`));
Query OK, 0 rows affected (0.64 sec)
mysql> insert into auto_test7 values(null, 1);
Query OK, 1 row affected (0.03 sec)
mysql> show create table auto_test7;
+------------+-------------------------------+
| Table      | Create Table                  |
+------------+-------------------------------+
| auto_test7 | CREATE TABLE `auto_test7` (  `id` int NOT NULL AUTO_INCREMENT,  `cred_id` int
DEFAULT NULL,  PRIMARY KEY (`id`),  UNIQUE KEY `cred_id` (`cred_id`)) ENGINE=InnoDB
AUTO_INCREMENT=2 DEFAULT CHARSET=utf8 |
+------------+-------------------------------------------------------------------+
1 row in set (0.00 sec)
mysql> insert into auto_test7 values(null, 1);
ERROR 1062 (23000): Duplicate entry '1' for key 'auto_test7.cred_id'
mysql> show create table auto_test7;
+------------+-------------------------------------------------------------------+
| Table      | Create Table                                                      |
+------------+-------------------------------------------------------------------+
| auto_test7 | CREATE TABLE `auto_test7` (  `id` int NOT NULL AUTO_INCREMENT,  `cred_id` int
DEFAULT NULL,  PRIMARY KEY (`id`),  UNIQUE KEY `cred_id` (`cred_id`)) ENGINE=InnoDB
AUTO_INCREMENT=3 DEFAULT CHARSET=utf8 |
+------------+-------------------------------------------------------------------+
```

- 批量插入数据时（如insert...select、load file等），自增键的申请是分批申请的，每批申请2的n次方个序号，用完继续申请，没用完也不会退回，所以可能会导致AUTO_INCREMENT跳变。

```
mysql> create table auto_test5_tmp(id tinyint not null AUTO_INCREMENT, name varchar(8), PRIMARY
KEY (`id`));
Query OK, 0 rows affected (0.08 sec)
mysql> select * from auto_test5;
+----+------+
| id | name |
+----+------+
|  1 | A    |
|  2 | B    |
|  3 | C    |
|  4 | X    |
|  5 | Y    |
|  6 | Z    |
|  8 | A    |
|  9 | B    |
| 10 | C    |
| 11 | X    |
| 12 | Y    |
| 13 | Z    |
+----+------+
12 rows in set (0.00 sec)
```



```
mysql> insert into auto_test5_tmp select 0,name from auto_test5;
Query OK, 12 rows affected (0.01 sec)
Records: 12  Duplicates: 0  Warnings: 0
mysql> select * from auto_test5_tmp;
+----+------+
| id | name |
+----+------+
|  1 | A    |
|  2 | B    |
|  3 | C    |
|  4 | X    |
|  5 | Y    |
|  6 | Z    |
|  7 | A    |
|  8 | B    |
|  9 | C    |
| 10 | X    |
| 11 | Y    |
| 12 | Z    |
+----+------+
12 rows in set (0.00 sec)
mysql> show create table auto_test5_tmp;
+----------------+-------------------------------------------------------------+
| Table          | Create Table                                                |
+----------------+-------------------------------------------------------------+
| auto_test5_tmp | CREATE TABLE `auto_test5_tmp` (  `id` tinyint NOT NULL AUTO_INCREMENT,
`name` varchar(8) DEFAULT NULL,  PRIMARY KEY (`id`)) ENGINE=InnoDB AUTO_INCREMENT=16
DEFAULT CHARSET=utf8 |
+----------------+-------------------------------------------------------------+
```

6.5 表的自增 AUTO_INCREMENT 初值与步长

数据表中自增字段的AUTO_INCREMENT的初值与步长由auto_increment_increment和auto_increment_offset参数决定。

- auto_increment_offset：AUTO_INCREMENT值的初值。
- auto_increment_increment：AUTO_INCREMENT值每次增长的步长。
- 当 auto_increment_offset > auto_increment_increment 时，实际使用时初值会变为auto_increment_increment。
- 当 auto_increment_offset <= auto_increment_increment，自增值计算方式如下：

  自增值 = auto_increment_offset + N*auto_increment_increment （ N为插入的数据条数 ）

在GaussDB(for MySQL)中这两个参数默认值都为1，参考如下步骤修改。如需修改时需要在控制台-实例详情-参数修改中修改。

步骤1 在“实例管理”页面，选择指定的实例，单击实例名称，进入实例的基本信息页面。

步骤2 在左侧导航栏中选择“参数修改”，在“参数”页签修改相应参数。

----结束

示例：

1. auto_increment_offset=1，auto_increment_increment=1，表示初值为1，步长为1。

```
show variables like 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
```



```
| auto_increment_increment | 1     |
| auto_increment_offset    | 1     |
+--------------------------+-------+
```

2. 修改auto_increment_increment=2，步长变为2。

```
set session auto_increment_offset=2;
Query OK, 0 rows affected (0.02 sec)
show variables like 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 2     |
| auto_increment_offset    | 1     |
+--------------------------+-------+
```

3. auto_increment_offset=10，auto_increment_increment=2，由于auto_increment_offset > auto_increment_increment，因此初值为2，步长为2。

```
set session auto_increment_offset=10;
set session auto_increment_increment=2;
show variables like 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 2     |
| auto_increment_offset    | 10    |
+--------------------------+-------+
create table auto_test2(id int NOT NULL AUTO_INCREMENT, PRIMARY KEY (`id`));
Query OK, 0 rows affected (0.08 sec)
show create table auto_test2;
CREATE TABLE `auto_test2` (   `id` int NOT NULL AUTO_INCREMENT,   PRIMARY KEY (`id`) )
ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.01 sec)
insert into auto_test2 values(0), (0), (0);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
select * from auto_test2;
+----+
| id |
+----+
|  2 |
|  4 |
|  6 |
+----+
3 rows in set (0.01 sec)
```

4. auto_increment_offset=5，auto_increment_increment=10，初值为5，步长为10。

```
set session auto_increment_offset=5;
set session auto_increment_increment=10;
show variables like 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 10    |
| auto_increment_offset    | 5     |
+--------------------------+-------+
create table auto_test3(id int NOT NULL AUTO_INCREMENT, PRIMARY KEY (`id`));
insert into auto_test3 values(0), (0), (0);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
select * from auto_test3;
+----+
| id |
+----+
|  5 |
| 15 |
| 25 |
```

6.6 修改表的自增 AUTO_INCREMENT 值

AUTO_INCREMENT修改时，遵循如下约束限制：

1. 当AUTO_INCREMENT大于表中数据的最大值时，可以在取值范围内任意修改为更大的值。

```
show create table animals;
+---------+-----------------------------------------------------------------+
| Table   | Create Table                                                    |
+---------+-----------------------------------------------------------------+
| animals | CREATE TABLE `animals` (
`id` mediumint NOT NULL AUTO_INCREMENT,   `name` char(30) NOT NULL,
PRIMARY KEY (`id`) ) ENGINE=InnoDB AUTO_INCREMENT=101 DEFAULT CHARSET=utf8   |
+---------+-----------------------------------------------------------------+
1 row in set (0.00 sec)
mysql> select * from animals;
+-----+-----------+
| id  | name      |
+-----+-----------+
| -50 | -middle   |
|   1 | dog       |
|   2 | cat       |
|  50 | middle    |
| 100 | rabbit    |
+-----+-----------+
11 rows in set (0.00 sec)
alter table animals AUTO_INCREMENT=200;
Query OK, 0 rows affected (0.22 sec)
Records: 0  Duplicates: 0  Warnings: 0
 show create table animals;
+---------+-----------------------------------------------------------+
| Table   | Create Table                                              |
+---------+-----------------------------------------------------------+
| animals | CREATE TABLE `animals` (
`id` mediumint NOT NULL AUTO_INCREMENT,   `name` char(30) NOT NULL,
PRIMARY KEY (`id`) ) ENGINE=InnoDB AUTO_INCREMENT=200 DEFAULT CHARSET=utf8 |
+---------+-----------------------------------------------------------+
```

2. 当AUTO_INCREMENT大于表中数据的最大值时，如果修改后的指定值仍大于数据的最大值，则修改为指定值成功。否则，默认会修改为数据最大值+1。

```
mysql> select * from animals;
+-----+-----------+
| id  | name      |
+-----+-----------+
| -50 | -middle   |
|   1 | dog       |
|   2 | cat       |
|  50 | middle    |
| 100 | rabbit    |
+-----+-----------+
mysql> show create table animals;
+---------+-----------------------------------------------------------+
| Table   | Create Table                                              |
+---------+-----------------------------------------------------------+
| animals | CREATE TABLE `animals` (
`id` mediumint NOT NULL AUTO_INCREMENT,   `name` char(30) NOT NULL,
PRIMARY KEY (`id`) ) ENGINE=InnoDB AUTO_INCREMENT=200 DEFAULT CHARSET=utf8 |
+---------+-----------------------------------------------------------+
mysql> alter table animals AUTO_INCREMENT=150;
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0
mysql> show create table animals;
+---------+-----------------------------------------------------------+
| Table   | Create Table                                              |
+---------+-----------------------------------------------------------+
| animals | CREATE TABLE `animals` (
```

```
`id` mediumint NOT NULL AUTO_INCREMENT,   `name` char(30) NOT NULL,
PRIMARY KEY (`id`) ) ENGINE=InnoDB AUTO_INCREMENT=150 DEFAULT CHARSET=utf8 |
+---------+-----------------------------------------------------+
mysql> alter table animals AUTO_INCREMENT=50;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0
mysql> show create table animals;
+---------+-----------------------------------------------------+
| Table   | Create Table                                        |
+---------+-----------------------------------------------------+
| animals | CREATE TABLE `animals` (
`id` mediumint NOT NULL AUTO_INCREMENT,   `name` char(30) NOT NULL,
PRIMARY KEY (`id`) ) ENGINE=InnoDB AUTO_INCREMENT=101 DEFAULT CHARSET=utf8 |
+---------+-----------------------------------------------------+
mysql> delete from animals where id=100;
Query OK, 1 row affected (0.00 sec)
mysql> select * from animals;
+-----+-----------+
| id  | name      |
+-----+-----------+
| -50 | -middle   |
|   1 | dog       |
|   2 | cat       |
|  50 | middle    |
+-----+-----------+
10 rows in set (0.00 sec)
mysql> alter table animals AUTO_INCREMENT=50;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0
mysql> show create table animals;
+---------+-----------------------------------------------------+
| Table   | Create Table                                        |
+---------+-----------------------------------------------------+
| animals | CREATE TABLE `animals` (
`id` mediumint NOT NULL AUTO_INCREMENT,   `name` char(30) NOT NULL,
PRIMARY KEY (`id`) ) ENGINE=InnoDB AUTO_INCREMENT=51 DEFAULT CHARSET=utf8 |
+---------+-----------------------------------------------------+
1 row in set (0.00 sec)
```

3. AUTO_INCREMENT无法修改为负数。

```
alter table animals AUTO_INCREMENT=-1;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to
your MySQL server version for the right syntax to use near '-1' at line 1
```

6.7 自增主键达到上限，无法插入数据

场景描述

插入数据时报错 ERROR 1062 (23000): Duplicate entry 'xxx' for key 'xxx'。

原因分析

自增主键的字段取值达到上限，无法继续增长，导致新插入的数据生成的自增主键值与表中上一条数据相同，因为自增主键的值不可重复，插入失败报错。

解决方案

1. 如果数据变化较多，表中实际数据量远小于自增主键的容量，则可以考虑将该表的数据全量导入新表，删除原表，然后rename将新表名改回原表名。(使用数据导入导出的方法有多种实现方法，此处仅举其中一种例子)

   a. 创建表auto_test5_tmp。


```
create table auto_test5_tmp(id tinyint not null AUTO_INCREMENT, name varchar(8), PRIMARY KEY (`id`));
Query OK, 0 rows affected (0.07 sec)
```

b. 插入数据。

```
insert into auto_test5_tmp select 0,name from auto_test5;
Query OK, 6 rows affected (0.01 sec)
Records: 6  Duplicates: 0  Warnings: 0
```

c. 查询表数据。

```
select * from auto_test5_tmp;
+----+------+
| id | name |
+----+------+
|  1 | A    |
|  2 | B    |
|  3 | C    |
|  4 | X    |
|  5 | Y    |
|  6 | Z    |
+----+------+
```

d. 删除表。

```
drop table auto_test5;
```

e. 重命名。

```
rename table auto_test5_tmp to auto_test5;
Query OK, 0 rows affected (0.12 sec)
```

2. 如果自增主键的取值范围不够，则修改自增主键的字段类型。

```
alter table auto_test6  modify column id int NOT NULL AUTO_INCREMENT;
Query OK, 6 rows affected (0.15 sec)
Records: 6  Duplicates: 0  Warnings: 0
```

6.8 自增字段取值

GaussDB(for MySQL)对自增字段的赋值有以下几种方法：

```
# 表结构
CREATE TABLE animals (
    id MEDIUMINT NOT NULL AUTO_INCREMENT,
    name CHAR(30) NOT NULL,
    PRIMARY KEY (id)
);
```

1. 不对自增字段赋值，数据库会自动将自增值填入字段中，AUTO_INCREMENT自增。

a. 插入数据。

```
 INSERT INTO animals (name) VALUES ('dog'),('cat'),('penguin'),('lax'),('whale'),('ostrich');
```

b. 查询表数据。

```
 select * from animals;
+----+---------+
| id | name    |
+----+---------+
|  1 | dog     |
|  2 | cat     |
|  3 | penguin |
|  4 | lax     |
|  5 | whale   |
|  6 | ostrich |
+----+---------+
```

c. 查询表结构。

```
 show create table animals;
+---------+-----------------------------------------------------+
| Table   | Create Table                                        |
+---------+-----------------------------------------------------+
```

```
| animals | CREATE TABLE `animals` (  `id` mediumint NOT NULL AUTO_INCREMENT,  `name` char(30) NOT NULL,  PRIMARY KEY (`id`)) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8 |
+---------+-----------------------------------------------------+
```

2. 对自增字段赋0或null值，数据库会自动将自增值填入字段中。AUTO_INCREMENT自增。

   a. 插入数据。

   ```
   INSERT INTO animals (id,name) VALUES(0,'groundhog');
   INSERT INTO animals (id,name) VALUES(NULL,'squirrel');
   ```

   b. 查询数据。

   ```
   select * from animals;
   +----+-----------+
   | id | name      |
   +----+-----------+
   |  1 | dog       |
   |  2 | cat       |
   |  3 | penguin   |
   |  4 | lax       |
   |  5 | whale     |
   |  6 | ostrich   |
   |  7 | groundhog |
   |  8 | squirrel  |
   +----+-----------+
   8 rows in set (0.00 sec)
   ```

   c. 查询表结构。

   ```
   show create table animals;
   +---------+-----------------------------------------------------------------+
   | Table   | Create Table                                                    |
   +---------+-----------------------------------------------------------------+
   | animals | CREATE TABLE `animals` (  `id` mediumint NOT NULL AUTO_INCREMENT,  `name` char(30) NOT NULL,  PRIMARY KEY (`id`)) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8 |
   +---------+-----------------------------------------------------------------+
   ```

3. 直接使用大于AUTO_INCREMENT的值A，数据库会将A填入字段并修改AUTO_INCREMENT=A+1。

   a. 插入数据。

   ```
   INSERT INTO animals (id,name) VALUES(100,'rabbit');
   ```

   b. 查询数据。

   ```
   select * from animals;
   +-----+-----------+
   | id  | name      |
   +-----+-----------+
   |   1 | dog       |
   |   2 | cat       |
   |   3 | penguin   |
   |   4 | lax       |
   |   5 | whale     |
   |   6 | ostrich   |
   |   7 | groundhog |
   |   8 | squirrel  |
   | 100 | rabbit    |
   +-----+-----------+
   9 rows in set (0.00 sec)
   ```

   c. 查询表结构。

   ```
   show create table animals;
   +---------+-----------------------------------------------------------------+
   | Table   | Create Table                                                    |
   +---------+-----------------------------------------------------------------+
   | animals | CREATE TABLE `animals` (  `id` mediumint NOT NULL AUTO_INCREMENT,  `name` char(30) NOT NULL,  PRIMARY KEY (`id`)) ENGINE=InnoDB AUTO_INCREMENT=101 DEFAULT CHARSET=utf8 |
   +---------+-----------------------------------------------------------------+
   ```


4. 使用小于AUTO_INCREMENT，但不冲突的值。数据可以插入，但AUTO_INCREMENT不变。

```
mysql> INSERT INTO animals (id,name) VALUES(50,'middle');
Query OK, 1 row affected (0.00 sec)
mysql> select * from animals;
+-----+-----------+
| id  | name      |
+-----+-----------+
|   1 | dog       |
|   2 | cat       |
|   3 | penguin   |
|   4 | lax       |
|   5 | whale     |
|   6 | ostrich   |
|   7 | groundhog |
|   8 | squirrel  |
|  50 | middle    |
| 100 | rabbit    |
+-----+-----------+
10 rows in set (0.00 sec)
mysql> show create table animals;
+---------+-----------------------------------------------------------------------+
| Table   | Create Table                                                          |
+---------+-----------------------------------------------------------------------+
| animals | CREATE TABLE `animals` (  `id` mediumint NOT NULL AUTO_INCREMENT,  `name`
char(30) NOT NULL,  PRIMARY KEY (`id`)) ENGINE=InnoDB AUTO_INCREMENT=101 DEFAULT
CHARSET=utf8 |
+---------+-----------------------------------------------------------------------+
```

5. 使用负值数据可以插入，但AUTO_INCREMENT不变。

   a. 插入数据。

   ```
   INSERT INTO animals (id,name) VALUES(-50,'-middle');
   ```

   b. 查询数据。

   ```
   select * from animals;
   +-----+-----------+
   | id  | name      |
   +-----+-----------+
   | -50 | -middle   |
   |   1 | dog       |
   |   2 | cat       |
   |   3 | penguin   |
   |   4 | lax       |
   |   5 | whale     |
   |   6 | ostrich   |
   |   7 | groundhog |
   |   8 | squirrel  |
   |  50 | middle    |
   | 100 | rabbit    |
   +-----+-----------+
   11 rows in set (0.00 sec)
   ```

   c. 查询表结构。

   ```
   show create table animals;
   +---------+-----------------------------------------------------------------------------------------+
   | Table   | Create Table
   +---------+-----------------------------------------------------------------------------------------+
   | animals | CREATE TABLE `animals` (  `id` mediumint NOT NULL AUTO_INCREMENT,
   `name` char(30) NOT NULL,  PRIMARY KEY (`id`)) ENGINE=InnoDB AUTO_INCREMENT=101
   DEFAULT CHARSET=utf8          |
   +---------+-----------------------------------------------------------------------------------------+
   ```

6.9 自增属性 AUTO_INCREMENT 为何未在表结构中显示

场景描述

创建表时，添加了自增属性AUTO_INCREMENT，执行**show create table**，自增属性未在表结构中显示。

创建表：

```
mysql> CREATE TABLE [redacted] (
    ->   id int(10)  NOT NULL AUTO_INCREMENT COMMENT '自增id',
    ->   account_id bigint(20) unsigned NOT NULL DEFAULT '0' COMMENT '关联account_base主键',
    ->   account_name varchar(128) NOT NULL DEFAULT '' COMMENT '用户名',
    ->   identity varchar(64) NOT NULL DEFAULT '' COMMENT '身份证',
    ->   mobile varchar(32) NOT NULL DEFAULT '' COMMENT '手机号',
    ->   score float(6,2) NOT NULL DEFAULT '0.00' COMMENT '分数',
    ->   utime bigint(20) unsigned NOT NULL DEFAULT '0' COMMENT '创建时间',
    ->   PRIMARY KEY (id),
    ->   UNIQUE KEY uique_index_account_id (account_id)
    -> ) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8mb4 [redacted]
Query OK, 0 rows affected (0.01 sec)
```

执行**show create table** *xxx*，未显示自增属性AUTO_INCREMENT：

```
---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
| aml_stats3 | CREATE TABLE [redacted] (
  "id" int(10) NOT NULL COMMENT '自增id',
  "account_id" bigint(20) unsigned NOT NULL DEFAULT '0' COMMENT '关联account_base主键',
  "account_name" varchar(128) NOT NULL DEFAULT '' COMMENT '用户名',
  "identity" varchar(64) NOT NULL DEFAULT '' COMMENT '身份证',
  "mobile" varchar(32) NOT NULL DEFAULT '' COMMENT '手机号',
  "score" float(6,2) NOT NULL DEFAULT '0.00' COMMENT '分数',
  "utime" bigint(20) unsigned NOT NULL DEFAULT '0' COMMENT '创建时间',
  PRIMARY KEY ("id"),
  UNIQUE KEY "uique_index_account_id" ("account_id")
)                       |
+------------+--------------------------------------------------------------------
---------------------------------------------------------------------------------
```

原因分析

经过排查，是因为参数“sql_mode”设置了NO_FIELD_OPTIONS属性。

参数名称	是否需要重启	值	允许值	描述
sql_mode	否	NO_FIELD_OPTIONS	ALLOW_INVALID_DATES,ANSI_QUOT...	当前SQL服务器模式。

sql_mode相关属性介绍：

- NO_FIELD_OPTIONS：不要在SHOW CREATE TABLE的输出中打印MySQL专用列选项。
- NO_KEY_OPTIONS：不要在SHOW CREATE TABLE的输出中打印MySQL专用索引选项。
- NO_TABLE_OPTIONS：不要在SHOW CREATE TABLE的输出中打印MySQL专用表选项（例如ENGINE）。

解决方案

将sql_mode的NO_FIELD_OPTIONS属性去掉即可。

6.10 空用户的危害

MySQL中是允许用户名为 '' 的用户存在，本章节介绍数据库中存在这种空用户时的危害。

MySQL中使用空用户时，它将可以匹配任何用户名。这一特性也会带来多种安全性、功能性危害。所以，在实际使用过程中应避免使用空用户。

- 安全性危害
  - 当存在空用户时，连接时可以使用任意用户名进行登录。
  - 如果空用户有密码，则使用任意用户名和空用户的密码即可登录数据库，并获得空用户所拥有的所有权限。示例：

```
#没有空用户时，使用非法用户名‘abcd’，连接失败
mysql> select user,host from mysql.user;
+------------------+-----------+
| user             | host      |
+------------------+-----------+
| root             | %         |
| mysql.infoschema | localhost |
| mysql.session    | localhost |
| mysql.sys        | localhost |
+------------------+-----------+
mysql -uabcd -h127.0.0.1 -P3306 -pTest_1234
mysql: [Warning] Using a password on the command line interface can be insecure.
ERROR 1045 (28000): Access denied for user 'abcd'@'localhost' (using password: YES)

# 创建空用户后，使用非法用户名‘abcd’，密码用空用户的密码，连接成功
mysql> create user ''@'localhost' IDENTIFIED BY 'Test_1234';
mysql> select user,host from mysql.user;
+------------------+-----------+
| user             | host      |
+------------------+-----------+
| root             | %         |
|                  | localhost |
| mysql.infoschema | localhost |
| mysql.session    | localhost |
| mysql.sys        | localhost |
+------------------+-----------+
mysql -uabcd -h127.0.0.1 -P3306 -pTest_1234
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 37Server version: 8.0.22-debug Source distribution
Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its affiliates.
Other names may be trademarks of their respective owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
```

  - 如果空用户没有密码，则使用任意用户名即可免密登录数据库，并获得空用户所拥有的所有权限。示例：

```
#存在无密码的空用户时，可以使用任意用户免密登录数据库。
mysql> create user ''@'localhost';
Query OK, 0 rows affected (8.87 sec)
mysql> select user,host from mysql.user;
+------------------+-----------+
| user             | host      |
+------------------+-----------+
| root             | %         |
|                  | localhost |
| mysql.infoschema | localhost |
| mysql.session    | localhost |
| mysql.sys        | localhost |
+------------------+-----------+
mysql -uabcd -h127.0.0.1 -P3306
```

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 39Server version: 8.0.22-debug Source distribution
Copyright (c) 2000, 2020, Oracle and/or its affiliates.
All rights reserved. Oracle is a registered trademark of Oracle Corporation and/or its affiliates.
Other names may be trademarks of their respective owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
#-----------------
mysql -usdhsjkdshk -h127.0.0.1 -P3306
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 40Server version: 8.0.22-debug Source distribution
Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its affiliates.
Other names may be trademarks of their respective owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
```

- 功能性危害

  当存在空用户时，可能因为匹配出错，导致正常的用户名无法登录。

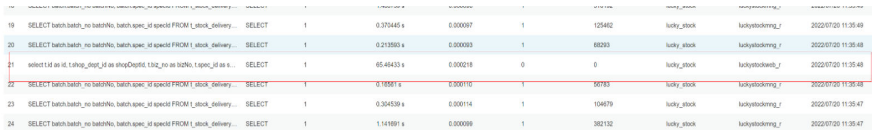
  示例：存在空用户与root用户的host有重叠时，导致root用户无法使用密码登录，或者使用空用户的密码登录后无法进入root的权限。

```
mysql> create user ''@'localhost';
Query OK, 0 rows affected (8.87 sec)
mysql> select user,host from mysql.user;
+------------------+-----------+
| user             | host      |
+------------------+-----------+
| root             | %         |
|                  | localhost |
| mysql.infoschema | localhost |
| mysql.session    | localhost |
| mysql.sys        | localhost |
+------------------+-----------+
# 用root的密码无法登录
mysql -uroot -h127.0.0.1 -P3306 -pTest_root
mysql: [Warning] Using a password on the command line interface can be insecure.
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)
# 用空用户的密码(免密)登录后实际是空用户登录，没有root权限。
mysql -uroot -h127.0.0.1 -P3306
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 45Server version: 8.0.22-debug Source distribution
Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its affiliates.
Other names may be trademarks of their respective owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> select user,host from mysql.user;
ERROR 1142 (42000): SELECT command denied to user ''@'localhost' for table 'user'
```

6.11 慢日志显示 SQL 语句扫描行数为 0

场景描述

查询慢日志中记录SQL执行65秒，但是扫描行数为0。



原因分析

被中断的查询超过慢日志设置阈值也会记录慢日志，但是所记录的扫描行数为0。客户JDBC连接设置了查询超时：

```
jdbc:mysql://10.221.88.78:3306/lucky_stock?
useUnicode=true&characterEncoding=UTF8&autoReconnect=true&failOverReadOn
ly=false&useSSL=false&serverTimezone=Asia/Shanghai&zeroDateTimeBehavior=C
ONVERT_TO_NULL&rewriteBatchedStatements=true&allowMultiQueries=true&conn
ectTimeout=10000&socketTimeout=70000
```

解决方案

优化SQL或者将sockTimeOut设置合理值。

6.12 错误日志页面显示 handle_sync_msg_from_slave my_net_read error:-1

场景描述

在错误日志页面，显示报错信息：handle_sync_msg_from_slave my_net_read error:-1

原因分析

在主从同步时，由于网络抖动，可能会偶现网络数据包错误，产生这个报错信息，系统会自动重试，无需介入处理。

解决方案

无需解决。