云数据库 RDS for MySQL

故障排除

文档版本 01

发布日期 2025-09-01

华为云计算技术有限公司

版权所有 © 华为云计算技术有限公司 2025。 保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

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
1.1 RDS for MySQL 全备恢复到本地提示无 super 权限 1
1.2 备份期间 DDL 操作导致备份失败 3
1.3 如何将华为云上或本地的数据库备份文件恢复到 RDS 实例 3
1.4 RDS for MySQL 备份任务失败分析思路 4
1.5 手动下发全量备份比自动下发全量备份时间长 5
1.6 下载备份文件在本地恢复，登录密码错误 6
1.7 磁盘空间满导致自动增量备份失败 6
1.8 RDS 实例恢复到指定时间点失败 7
1.9 怎么解决执行 mysqldump 出现 SET @@SESSION.SQL_LOG_BIN 等 SQL 的问题 7
1.10 mysqldump 导出数据报错权限不足 8
1.11 MySQL 表级时间点恢复时提示密钥失效或已删除 9
2 主备复制 11
2.1 MySQL 主备复制原理简介 11
2.2 主备复制延迟持续增长后自动恢复 13
2.3 MySQL 主备复制延迟场景及解决方案 13
2.4 RDS 主备复制关系异常 15
2.5 主备复制时延瞬间飚高回落 16
2.6 canal 工具报错权限不足 17
2.7 canal 解析 Binlog 报错 18
2.8 RDS for MySQL Binlog 生成的机制 18
3 参数类 20
3.1 控制台上修改 long_query_time 参数后未生效 20
3.2 GROUP_CONCAT 结果不符合预期 21
3.3 RDS for MySQL 创建索引失败报错[ERROR] 1071 的解决方案 22
3.4 RDS for MySQL 大小写参数敏感类问题 24
3.5 RDS MySQL timeout 相关参数简介 25
3.6 命令行修改 MySQL global 参数失败 26
4 性能资源类 28
4.1 CPU 使用率高问题排查与优化 28
4.2 内存使用超限风险与优化 30
4.3 磁盘性能带宽超上限 33

 版权所有 © 华为云计算技术有限公司

4.4 联合索引设置不当导致慢 SQL 34
4.5 数据库磁盘满导致被设置 read_only 37
4.6 Binlog 未清理导致磁盘占用高 39
4.7 业务死锁导致响应变慢 39
4.8 MySQL 只读实例磁盘占用远超主实例 40
4.9 RDS for MySQL CPU 升高定位思路 41
4.10 冷热数据问题导致 sql 执行速度慢 43
4.11 CPU/内存配置与 TPS 和 QPS 性能相关的问题 44
4.12 表碎片率过高可能导致的问题 47
4.13 复杂查询造成磁盘满 49
4.14 怎么解决查询运行缓慢的问题 50
4.15 长事务导致规格变更或小版本升级失败 50
4.16 RDS for MySQL 数据库报错 Native error 1461 的解决方案 51
4.17 RDS for MySQL 增加表字段后出现运行卡顿现象 52
4.18 长事务导致 UNDO 增多引起磁盘空间满 52
4.19 RDS for MySQL 如何定位一直存在的长事务告警 53
4.20 RDS for MySQL 部分 SQL 的 commit 时间偶现从几毫秒陡增到几百毫秒 53
4.21 本地 SSD 盘规格降配选不到资源 53
4.22 ibdata1 为什么会变大 54

5 SQL 类 55

5.1 RDS for MySQL 执行 SQL 报错无法识别双引号 55
5.2 更新 emoji 表情数据报错 Error 1366 55
5.3 索引长度限制导致修改 varchar 长度失败 56
5.4 建表时 timestamp 字段默认值无效 57
5.5 自增属性 AUTO_INCREMENT 为什么未在表结构中显示 58
5.6 存储过程和相关表字符集不一致导致执行缓慢 59
5.7 RDS MySQL 报错 ERROR [1412]的解决方法 60
5.8 创建二级索引报错 Too many keys specified 60
5.9 存在外键的表删除问题 61
5.10 distinct 与 group by 优化 61
5.11 字符集和字符序的默认选择方式 63
5.12 MySQL 创建用户提示服务器错误 65
5.13 delete 大表数据后，再次查询同一张表时出现慢 SQL 66
5.14 设置事件定时器后未生效 67
5.15 为什么有时候用浮点数做等值比较查不到数据 68
5.16 开通数据库代理后有大量 select 请求分发到主节点 70
5.17 执行 RENAME USER 失败的解决方法 71
5.18 有外键的表无法删除报错 ERROR[1451]的解决方案 72
5.19 表字段类型转换失败的解决方法 72
5.20 RDS for MySQL 创建表失败报错 Row size too large 的解决方案 73
5.21 RDS for MySQL 数据库报错 ERROR [1412]的解决方案 74
5.22 外键使用不规范导致实例重启失败或执行表操作报错 ERROR 1146: Table 'xxx' doesn't exist 74

 版权所有 © 华为云计算技术有限公司

5.23 RDS for MySQL 在分页查询时报错：Out of sort memory, consider increasing server sort buffer size 75
5.24 RDS for MySQL 创建用户报错：Operation CREATE USER failed 76
5.25 RDS for MySQL 使用 grant 授权 all privileges 报语法错误 76
5.26 RDS for MySQL 5.6 版本实例创建表报错 77
5.27 无主键表添加自增主键后导致主备节点查询数据不一致 78
5.28 RDS for MySQL 插入数据提示 Data too long for column 78
6 连接类 80
6.1 连接数据库报错 Access denied 80
6.2 mariadb-connector SSL 方式连接数据库失败 81
6.3 RDS for MySQL 建立连接慢导致客户端超时报 connection established slowly 82
6.4 root 账号的 ssl_type 修改为 ANY 后无法登录 83
6.5 通过 DAS 登录实例报错 Client does not support authentication protocol requested by server 84
6.6 通过 DAS 执行 SQL 报错 ALTER USER failed 85
6.7 主实例上执行 show slave status 的结果为什么不为空 85
6.8 通过 DAS 授权或取消授权时报错 Your password does not satisfy the current policy requirements 85
6.9 客户端 TLS 版本与 RDS for MySQL 不一致导致 SSL 连接失败 86
6.10 使用 root 账号连接数据库失败 86
6.11 RDS for MySQL 客户端连接实例后会自动断开 87
6.12 RDS for MySQL 实例无法访问 89
6.13 RDS for MySQL 数据库修改 authentication_string 字段为显示密码后无法登录 92
6.14 RDS for MySQL 升级版本后，导致现有配置无法正常连接到 MySQL-server 92
6.15 客户端超时参数设置不当导致连接超时退出 94
6.16 RDS for MySQL 在启用了 SSL 验证连接功能后，导致代码（php/java/python）等连接数据库失败 94
6.17 istio-citadel 证书机制导致每隔 45 天出现断连 95
6.18 数据库版本升级后 Navicat 客户端登录实例报错 1251 96
7 其他使用问题 97
7.1 慢日志显示 SQL 语句扫描行数为 0 97
7.2 SQL 诊断结果中记录的行数远小于慢日志中的扫描行数 97
7.3 RDS for MySQL 慢日志里面有毫秒级别的 SQL 98
7.4 查看 RDS 存储空间使用量 98
7.5 错误日志报错 The table is full 99
7.6 审计日志上传策略说明 99
7.7 自增字段取值 100
7.8 表的自增 AUTO_INCREMENT 初值与步长 102
7.9 表的自增 AUTO_INCREMENT 超过数据中该字段的最大值加 1 105
7.10 自增字段值跳变的原因 108
7.11 修改表的自增 AUTO_INCREMENT 值 113
7.12 自增主键达到上限，无法插入数据 114
7.13 空用户的危害 117
7.14 pt-osc 工具连接 RDS for MySQL 主备实例卡住 119
7.15 购买 RDS 实例支付报错：Policy doesn't allow bss:order:update to be performed 120

 版权所有 © 华为云计算技术有限公司

7.16 RDS for MySQL 是否可以修改数据库名称..... 121
7.17 购买 RDS 实例报错：无 IAM 的 agency 相关权限..... 121

 版权所有 © 华为云计算技术有限公司

1 备份恢复

1.1 RDS for MySQL 全备恢复到本地提示无 super 权限

场景描述

使用RDS for MySQL时，如果想搭建本地MySQL从库，会使用云上RDS for MySQL全量备份恢复到本地环境。在和云上RDS for MySQL实例建立主备关系时（执行**change master**命令），通常会出现如下错误：

报错ERROR 1227：

```
ERROR 1227 (42000): Access denied; you need (at least one of) the SUPER privilege(s) for this operation
```

原因分析

RDS for MySQL的root用户没有super权限，恢复到本地需要手动添加。

解决方案

手动给root用户赋予super权限，详细步骤如下：

1. 对本地恢复的MySQL，设置免密登录：在配置文件“my.cnf”的[mysqld]组下，添加如下配置项：skip-grant-tables=on。示例：

 版权所有 © 华为云计算技术有限公司

```
# For advice on how to change settings please see
# http://dev.mysql.com/doc/refman/5.7/en/server-configuration-defaults.html

[mysqld]
#
# Remove leading # and set to the amount of RAM for the most important data
# cache in MySQL. Start at 70% of total RAM for dedicated server, else 10%.
# innodb_buffer_pool_size = 128M
#
# Remove leading # to turn on a very important data integrity option: logging
# changes to the binary log between backups.
# log_bin
#
# Remove leading # to set options mainly useful for reporting servers.
# The server defaults are faster for transactions and fast SELECTs.
# Adjust sizes as needed, experiment to find the optimal values.
# join_buffer_size = 128M
# sort_buffer_size = 2M
# read_rnd_buffer_size = 2M
skip-grant-tables=on
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
```

2. 重启mysqld进程。

   systemctl restart mysqld

3. 使用rdsAdmin账户免密登录数据库。

   mysql -urdsAdmin

4. 给root用户授权。

   grant all on *.* to root @'%';

   flush privileges;

```
mysql> show grants for root@'%';
ERROR 1290 (HY000): The MySQL server is running with the --skip-grant-tables option so it cannot execute this statement
mysql> grant all on *.* to root@'%';
ERROR 1290 (HY000): The MySQL server is running with the --skip-grant-tables option so it cannot execute this statement
mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)

mysql> show grants for root@'%';
+------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
-----------------------------------+
| Grants for root@%

                                   |
+------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
-----------------------------------+
| GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, PROCESS, REFERENCES, INDEX, ALTER, SHOW DATABASES, CREATE TEMPORARY TABLES, LOCK
 TABLES, EXECUTE, REPLICATION SLAVE, REPLICATION CLIENT, CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER ON
 *.* TO 'root'@'%' WITH GRANT OPTION |
+------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
-----------------------------------+
1 row in set (0.00 sec)

mysql> grant all on *.* to root@'%';
Query OK, 0 rows affected (0.00 sec)
```

5. 去掉免密登录设置：在配置文件“my.cnf”的[mysqld]组下，删除如下配置项：skip-grant-tables=on。
6. 重启mysqld进程。
7. 使用root账户登录数据库，并检查权限。

```
[root@ecs-xjytest mysql]# mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.27 MySQL Community Server (GPL)

Copyright (c) 2000, 2019, 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> show grants for root@'%';
+-----------------------------------------------------------+
| Grants for root@%                                         |
+-----------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' WITH GRANT OPTION |
+-----------------------------------------------------------+
1 row in set (0.00 sec)

mysql> set global sort_buffer_size=2097152;
Query OK, 0 rows affected (0.00 sec)
```

此时，再使用root用户执行**change master**操作，不会出现super权限错误。

1.2 备份期间 DDL 操作导致备份失败

场景描述

实例连续两天备份失败，备份时间窗内有DDL操作。

问题原因

MySQL全量备份基于xtrabackup，为保证数据一致性，全量备份操作与DDL操作存在元数据锁冲突，会导致备份一直阻塞等待，超时失败。

执行“show processlist”命令，排查业务侧在备份时间窗内是否有DDL操作。

解决方案

1. 停止相应的DDL操作后，重试手动备份。
2. 建议此后的DDL业务变更操作应尽量避开备份窗口。

1.3 如何将华为云上或本地的数据库备份文件恢复到 RDS 实例

华为云上的RDS备份文件恢复到实例

通过RDS备份文件的恢复功能，将备份文件数据恢复到实例上。具体请参见**恢复备份**。

本地的MySQL备份文件恢复到实例

 版权所有 © 华为云计算技术有限公司

通过数据复制服务的实时迁移功能，将云下MySQL数据库迁移到RDS，具体请参见入云迁移。

1.4 RDS for MySQL 备份任务失败分析思路

场景描述

客户使用RDS for MySQL数据库服务，使用自有备份脚本，通过mysqldump命令进行异机备份。备份数据到一台与RDS在不同子网的ECS主机，但备份任务运行300秒后就会中断，无法完成备份任务。

原因分析

为了排除网络问题，将执行备份任务的ECS主机更换为一台与RDS服务器同一子网的ECS主机上，备份任务执行成功。

- 网络排查反馈：跨子网只是在虚拟网络设备中转换，类似于网关，时延和带宽没有大的差异。
- 数据库排查反馈: 每次中断都是300秒就中断了，是因为RDS服务器端的参数导致，如果客户端300秒内没有完成数据写入，就会根据参数“net_write_timeout”的值断开数据库连接。

处理过程

步骤1 了解mysqldump的备份的数据流向，采用的协议和端口。

mysqldump使用TCP协议连接RDS服务器的8635端口，建立连接后，通过网络将数据备份到本地磁盘。

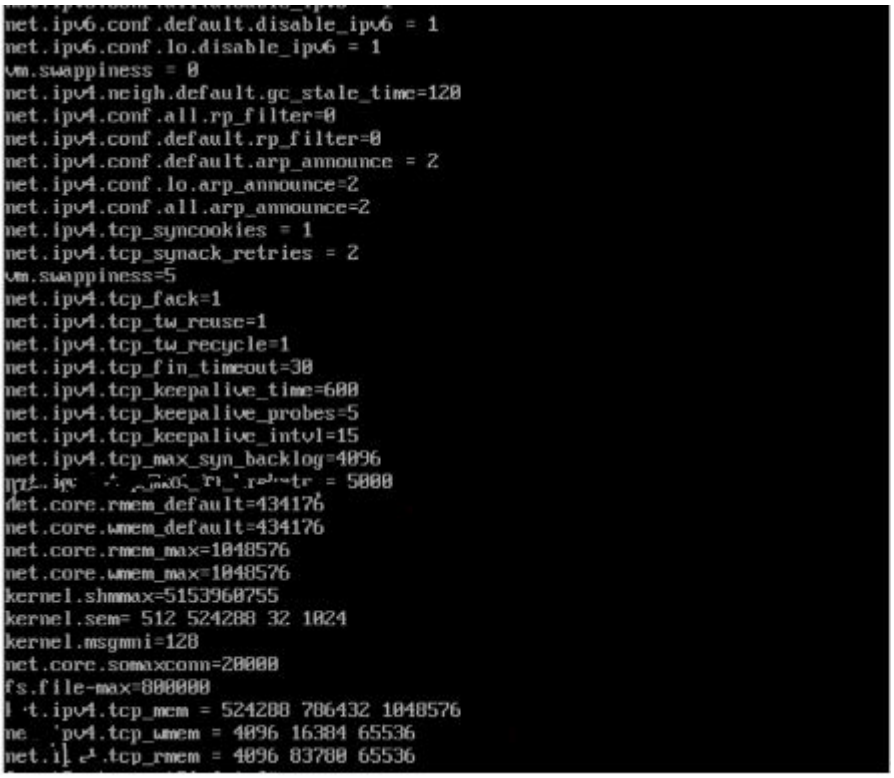
步骤2 分析两台ECS主机的差异点：

1. 查看两台主机硬件配置是否一致：同为2CORE 6GB。
2. 查看两台主机OS版本是否一致：同为centos7.4。

步骤3 查看网卡速率是否一致。

步骤4 查看内核参数配置是否一致，发现备份失败主机未做优化，备份成功的主机有做网络参数优化。

 版权所有 © 华为云计算技术有限公司



步骤5 初步判断问题与TCP缓存参数设置相关，将成功ECS主机的内核参数覆盖到问题ECS主机上并使其生效，运行备份任务，备份任务未再中断，成功完成了备份。

```
[root@szt-test01 data]# tail -fn 20 database_dump.20180413.sql
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2018-04-13 13:41:40
```

----结束

解决方案

mysqldump在备份过程中会产生大数据读写，本场景是跨主机通过网络异机备份，备份端数据写入能力和TCP缓存无法匹配RDS端的发送能力，超时时间达到数据库写超时设定的300秒，最终导致备份中断。可以通过修改内核参数增加TCP的缓存，提高备份端网络处理能力来解决问题。

1.5 手动下发全量备份比自动下发全量备份时间长

场景描述

周期性触发的自动全备比客户手动触发的全备耗时短，且备份数据量接近，均为20GB。

版权所有 © 华为云计算技术有限公司

原因分析

查看磁盘吞吐，发现手动备份的磁盘吞吐低于自动备份的磁盘吞吐。

查看CES指标硬盘读吞吐量、硬盘写吞吐量，下图仅供参考：

备份任务会依赖OBS服务，自动备份处于OBS的业务低峰期，所以耗时短，而手动备份恰好处于OBS业务的高峰期，所以耗时长。

解决方案

建议客户避开OBS高峰期进行全量备份，可以缩短备份时间。

1.6 下载备份文件在本地恢复，登录密码错误

场景描述

下载备份文件在本地进行恢复，本地数据库登录密码提示错误，无法登录。

原因分析

下载备份文件在本地恢复，备份恢复之后，本地原自建数据库密码被云上数据库密码覆盖，导致使用本地原自建数据库密码无法登录。

解决方案

使用云上的root密码或在本地自建数据库重置密码。

1.7 磁盘空间满导致自动增量备份失败

场景描述

数据库未进行自动增量备份。

原因分析

随着业务数据的增加，实例负载太高，原来申请的数据库磁盘容量可能会不够用，尤其当实例显示“磁盘空间满”状态，且数据库不可进行写入操作，导致增量备份失败。

版权所有 © 华为云计算技术有限公司

解决方案

您需要为RDS实例进行扩容。在“实例管理”页面，选择目标实例，单击“操作”列的“更多 > 磁盘扩容”，进入“磁盘扩容”页面。磁盘扩容完成后进行一次全量备份，即可成功进行增量备份。

1.8 RDS 实例恢复到指定时间点失败

场景描述

使用恢复到指定时间点备份创建实例失败。

原因分析

备份时间戳出现偏差，导致恢复到指定时间点报错。

解决方案

通过全量备份文件恢复实例数据。具体请参见**恢复方案概览**。

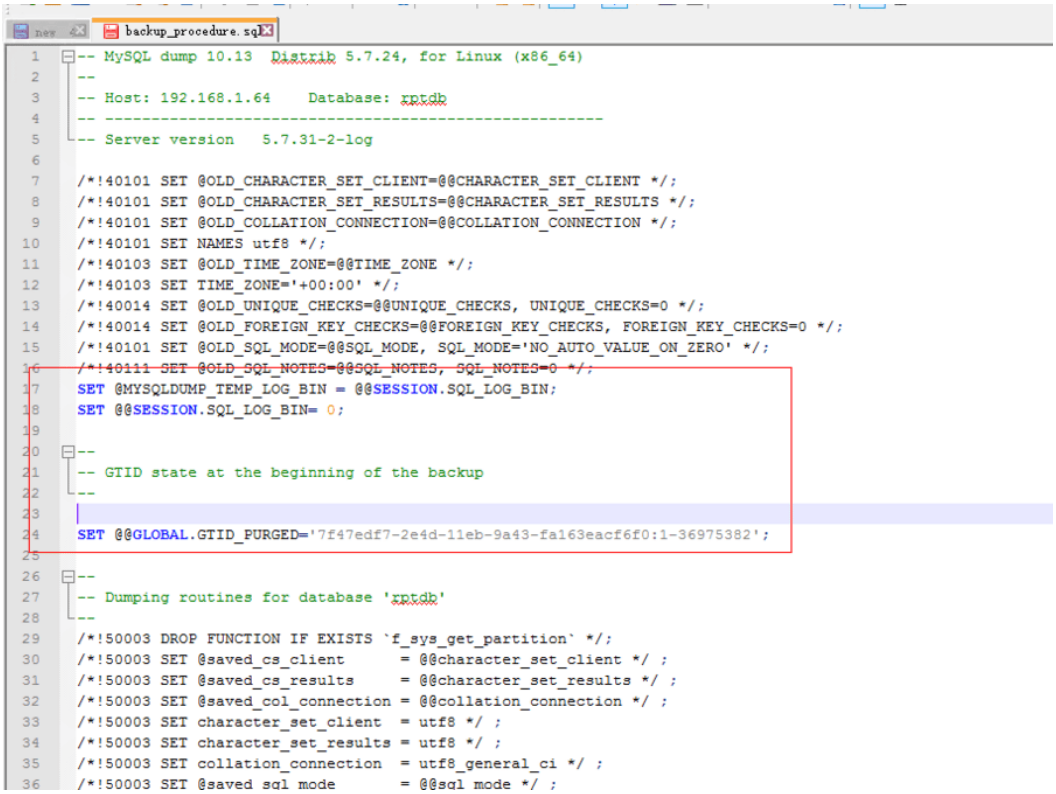
1.9 怎么解决执行 mysqldump 出现 SET @@SESSION.SQL_LOG_BIN 等 SQL 的问题

场景描述

执行mysqldump时，会出现如下如所示代码。

版权所有 © 华为云计算技术有限公司

图 1-1 代码显示



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RDS for 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

RDS for MySQL数据库在主从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”。

1.10 mysqldump 导出数据报错权限不足

案例一

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 '*<username>*'@'%';
  FLUSH PRIVILEGES;

 版权所有 © 华为云计算技术有限公司

案例二

mysqldump导出数据时报错：Access denied; you need (at least one of) the RELOAD privilege(s) for this operation (1227)

- 原因分析：使用具有RELOAD权限的用户执行mysqldump。
- 解决方案：使用DAS授予当前用户RELOAD全局权限。

  GRANT RELOAD ON *.* TO '*<username>*';

  FLUSH PRIVILEGES;

案例三

mysqldump导出数据时报错： Access denied; you need (at least one of) the LOCK TABLES privilege(s) for this operation (1227)mysqldump

- 原因分析：使用具有LOCK TABLES权限的用户执行mysqldump。
- 解决方案：授予当前用户LOCK TABLES权限。

  GRANT LOCK TABLES ON DATABASE.* TO '*<username>*';

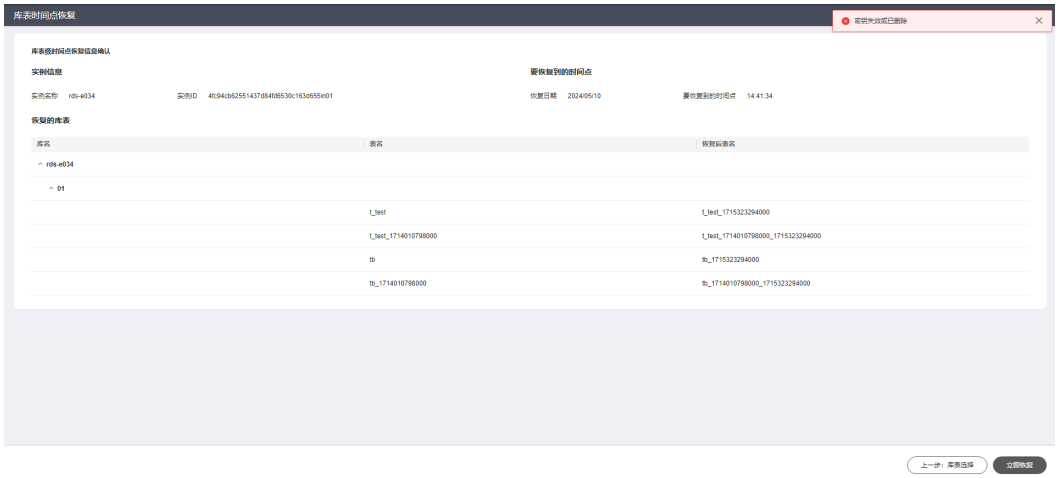
  FLUSH PRIVILEGES;

1.11 MySQL 表级时间点恢复时提示密钥失效或已删除

场景描述

MySQL表级时间点恢复时提示密钥失效或已删除。

图 1-2 表级时间点恢复



原因分析

在MySQL实例概览页面，查看实例绑定的密钥已经停用。

如果密钥被禁用或删除，实例无法进行磁盘扩容、规格变更、库表级恢复等操作。

版权所有 © 华为云计算技术有限公司

图 1-3 查看密钥

解决方案

将实例绑定的密钥设置为启用。

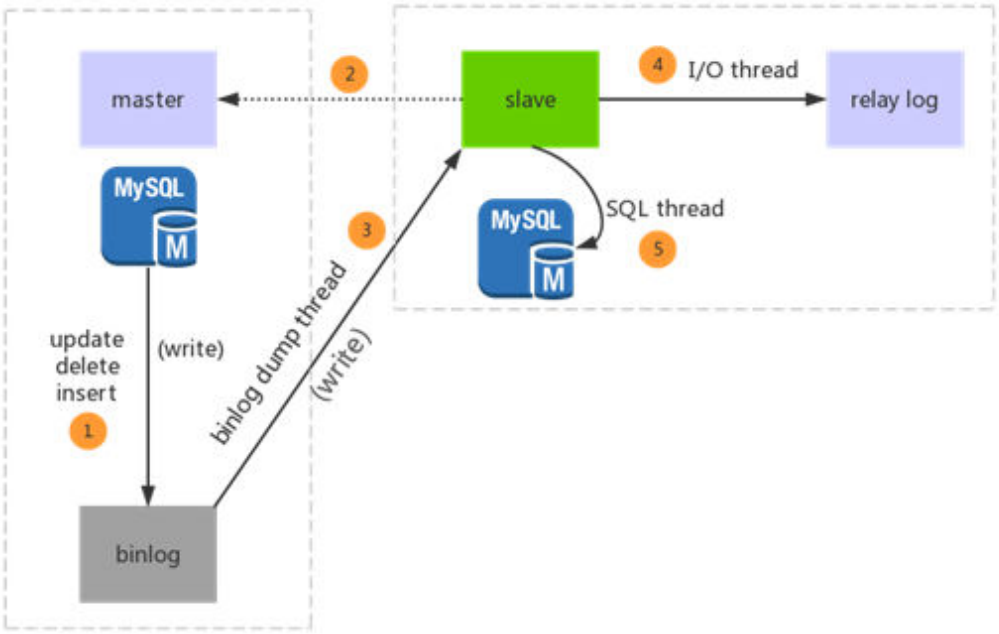
版权所有 © 华为云计算技术有限公司

2 主备复制

2.1 MySQL 主备复制原理简介

RDS for MySQL的默认备库、只读实例、自建从库、DRS链路灾备实例均采用MySQL的Binlog复制技术，也称为MySQL主备复制或主从复制技术。本章节介绍MySQL的主从复制原理。

主备复制流程



- 主节点（Master）中有数据更新时，会按照Binlog格式，将更新的操作以event形式写入到主节点的Binlog中。event有多种类型：INSERT、DELETE、UPDATE、QUERY等。
- 从节点（Slave）连接主节点时，有多少个从节点就会创建多少个Binlog dump线程。
- 当主节点的Binlog发生变化时，Binlog dump线程会通知所有从节点，并将相应Binlog内容推给从节点。

 版权所有 © 华为云计算技术有限公司

- 从节点的I/O thread收到Binlog内容后，会将内容写到本地relay log（中继日志）。
- 从节点的SQL thread会读取I/O thread写入的relay log，并且根据relay log中的event，回放对应的操作（DML、DDL等）。

Seconds_Behind_Master 计算方式

Seconds_Behind_Master即主备复制时延，通过**show slave status**查询获取。Seconds_Behind_Master计算的伪代码实现如下：

```
if (SQL thread is running)
//如果SQL线程启动
{
    if (SQL thread processed all the available relay log)
    //IO thread拉取主库Binlog的位置和sql thread应用的relay log相对于主库Binlog的位置相等
    {
        if (IO thread is running)
            //如果IO线程启动，设置延迟为0
            print 0;
        else
            //如果IO线程未启动，设置延迟为null
            print NULL;
    }
    else
        //如果SQL线程没有应用完所有的IO线程写入的event，那么需要计算Seconds_Behind_Master
        按公式计算Seconds_Behind_Master的值;
}
else
    //如果SQL线程也没有启动，则设置为空值
    print NULL;
```

上述伪代码中，Seconds_Behind_Master的计算公式为：

Seconds_Behind_Master = time(0) - last_master_timestamp - clock_diff_with_master

相关变量含义如下：

1. time(0)：当前从节点服务器的系统时间。
2. clock_diff_with_master：从节点的系统时间和主节点服务器系统时间的差值，一般为0。如果主从节点系统时间不一致，那么计算出的从节点复制时延会不准确。
3. last_master_timestamp：从节点在回放relay log中event过程中计算和更新，该变量在并行复制（MTS）和非并行复制方式下，更新的时机是不同的，默认全部开启并行复制：
   - 并行复制：可以简单理解为，从节点的SQL线程在每个事务执行完成后，更新last_master_timestamp值，其更新是以事务为单位。所以大事务、DDL容易导致主备延迟大，具体请参见**主备复制延迟持续增长后自动恢复**。
   - 非并行复制：从节点的SQL线程读取了relay log中的事务后，事务未执行前便会更新last_master_timestamp，其更新是以事务为单位。

综上所述，Seconds_Behind_Master的计算公式可以理解为：

Seconds_Behind_Master = 当前从节点服务器的系统时间 - 从节点SQL线程处理中事务在主节点的执行时间 - 从节点的系统时间和主节点服务器系统时间的差值

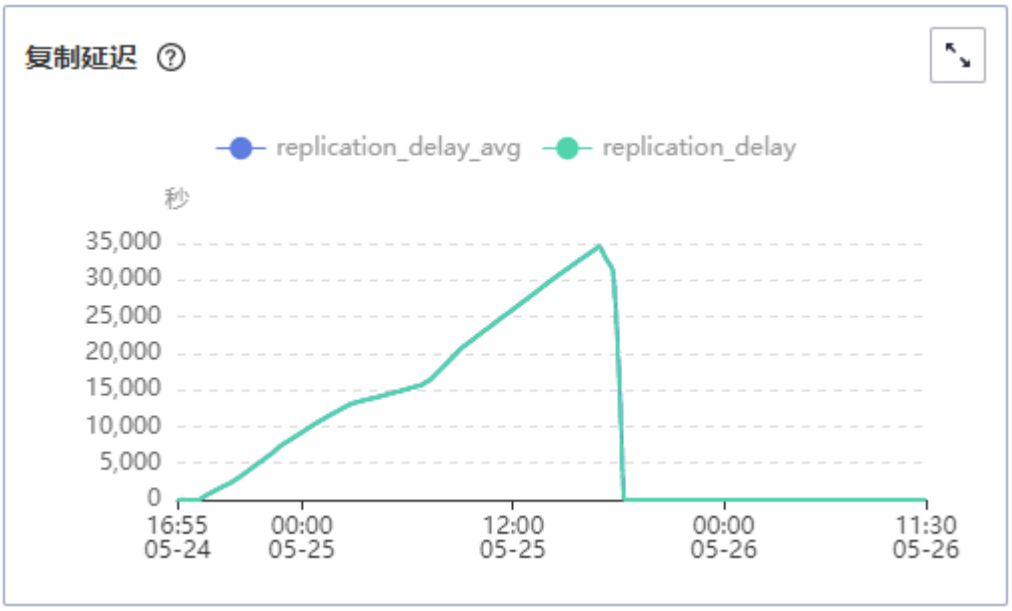
版权所有 © 华为云计算技术有限公司

2.2 主备复制延迟持续增长后自动恢复

场景描述

实例只读复制延迟很大，且在一段时间内持续增长，然后自动恢复。

查看CES指标实时复制时延，下图仅供参考：



原因分析

根据**MySQL主备复制延迟场景及解决方案**和**MySQL主备复制原理简介**的相关分析，可以推测此场景为大事务或DDL操作导致。

可以通过分析全量日志或慢日志，观察是否有大事务或DDL操作：

慢日志中有如下加索引的DDL操作，表的数据量上亿，耗时近一天，从而导致只读或备机在回放该DDL时复制延迟持续增长，回放完DDL后延迟恢复正常。

```
# Time: 2022-04-[redacted]734954+08:00
# User@Host: [redacted] @  [192.168.0.185]  Id: 20167391
# Query_time: 82714.322375  Lock_time: 0.000173 Rows_sent: 1  Rows_examined: 9[redacted] Thread_id: 201[redacted] Schema
1 Read_last: 0 Read_key: [redacted]9 Read_next: [redacted] Read_prev: 0 Read_rnd: 0 Read_rnd_next: 493972 Sort_merge_
tables: 0 Created_tmp_tables: 0 Start: 2022-04-1[redacted] End: 2022-04-1[redacted]:00 La
# QC_Hit: No  Full_scan: No  Full_join: No  Tmp_table: No  Tmp_table_on_disk: No  Filesort: No  Filesort_on_disk
SET timestamp=1649777004;
alter table [redacted]us_information add index ind[redacted](name,no);
```

解决方案

- 该场景属于正常现象，等待DDL执行完成后，延迟会自动恢复。
- 建议在业务低峰期进行加索引的操作。

2.3 MySQL 主备复制延迟场景及解决方案

RDS for MySQL的默认备库、只读实例、自建从库、DRS链路灾备实例均基于MySQL的Binlog复制技术，也称为MySQL主备复制或主从复制技术。主备复制实现又分为异

版权所有 © 华为云计算技术有限公司

步复制或半同步复制，无论哪种方式，由于业务执行的语句的原因，不可避免地存在主备复制延迟。

现象表现为：RDS for MySQL备机或只读存在复制时延过高，甚至产生复制时延大的告警。

场景 1：主库执行了大事务

大事务一般指一个事务中包含大量的数据更新操作，例如一个事务包含几万次DML（insert，update，delete）操作、一条SQL语句批量更新了上万行数据等，大事务往往本身的执行时间很长（分钟级）。当主实例执行了大事务后，会产生大量的Binlog日志，备机或只读节点拉取这些Binlog耗时比一般事务长，且至少需要花费与主实例相同的时间来回放这些事务的更新，从而导致备机或只读节点出现复制延迟。

排查方法：

- 对于包含大量DML语句的大事务，使用如下命令，找到长时间执行的事务。

  select t.*,to_seconds(now())-to_seconds(t.trx_started) idle_time from INFORMATION_SCHEMA.INNODB_TRX t \G;
- 对于一条SQL语句执行大量数据的大事务，执行**show full processlist**，查找是否存在长时间执行的delete或update语句。
- 分析全量日志或慢日志，检查是否有大事务。

解决方法：

- 为了保证主从数据的一致性，需要等待大事务执行完成，主备复制延迟才能恢复。
- 业务侧避免此类大事务，可以将大事务拆分为小事务，分批执行。例如，通过where条件或limit语句限制每次要更新的数据量。

场景 2：对无主键表更新

RDS for MySQL的Binlog采用row格式，对每一行的数据更新，都会形成row格式Binlog event记录。例如：一个update语句更新100行数据，那么row格式的Binlog中会形成100行update记录，备机或只读回放时会执行100次单行update。

只读节点和备机在回放主库的Binlog event时，会根据表的主键或者唯一二级索引来检索需要更改的行。如果对应表未创建主键，则会产生大量的全表扫描，从而降低了Binlog日志的应用速度，产生复制延迟。

排查方法：

通过**show create table** *xxx*，分析执行慢的update和delete语句对应的表，分析是否有主键。

解决方法：

给无主键表增加主键，或者酌情增加唯一二级索引。

场景 3：DDL 操作

DDL操作往往执行时间很长，尤其是表数据量很大时。通常情况下，只读节点或备机回放一个DDL操作的时间和主库花费的时间基本一致。因此，当主机执行了大表的DDL操作后，备机和只读节点在回放该DDL期间，复制时间必然是一致增大的。

解决方法：

版权所有 © 华为云计算技术有限公司

该场景为正常现象，等DDL执行完成后，主备复制延迟才能恢复。建议在业务低峰期执行DDL操作。

场景 4：只读实例等待 MDL 锁

只读实例上往往有业务流量，如果存在只读长事务正在执行，会阻塞主实例同步过来的相应表的DDL操作，卡在了表MDL锁获取，进而阻塞所有同表的后续Binlog回放，导致复制延迟越来越大。

排查方法：

1. 登录只读节点，使用如下命令，观察是否有长时间执行的事务。

   select t.*,to_seconds(now())-to_seconds(t.trx_started) idle_time from INFORMATION_SCHEMA.INNODB_TRX t \G;

2. 查看只读节点的MDL锁视图，观察是否有MDL锁冲突。

   select * from information_schema.metadata_lock_info;

   根据MDL锁视图中的线程ID，找到阻塞的session。更多信息，请参见**MDL锁视图**。

解决方法：

kill只读节点上阻塞DDL操作的长事务，或者在业务侧提交该长事务。

场景 5：只读实例规格小于主实例

只读实例、DRS灾备实例的规格小于主实例时，一旦主实例写负载升高到一定程度，只读实例或DRS灾备实例会因为自身资源不足，无法及时回放Binlog，导致复制时延增加。

解决方法：

只读实例或DRS灾备实例扩大规格，与主实例规格匹配。

场景 6：读业务压力突然增大

只读库除了要同步主库的数据之外还要承担数据读的业务，当读业务压力突增时，可能会影响只读的回放线程，从而导致只读复制时延增加。

2.4 RDS 主备复制关系异常

操作场景

有时候客户会遇到华为云关系型数据库主备复制关系异常的情况，可能原因是误删除默认安全组策略，下面主要针对这个场景进行分析，供您参考。

解决方案

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。

版权所有 © 华为云计算技术有限公司

步骤3 单击页面左上角的☰，选择“数据库 > 云数据库 RDS”，进入RDS信息页面。

步骤4 在“实例管理”页面，选择指定的实例，单击实例名称。

步骤5 在“概览”页面，单击目标安全组名称，进入实例安全组页面。

步骤6 在“入方向规则”页签，单击“添加规则”，选择Any协议，源地址为自身安全组，即安全组的源端要有安全组自己 。

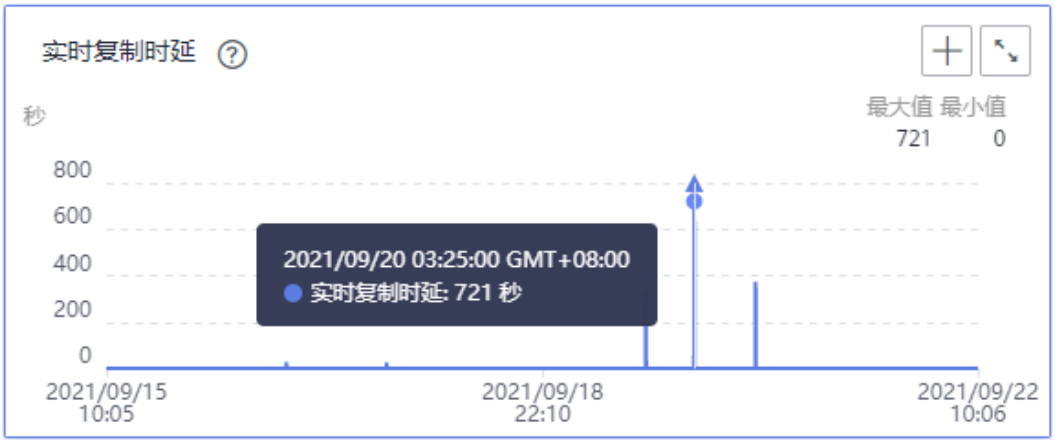
步骤7 添加完策略之后，主备复制的关系便会恢复正常。

----结束

2.5 主备复制时延瞬间飚高回落

场景描述

只读实例的CES监控上，出现只读复制时延会在某个瞬间升高并回落，如下图：



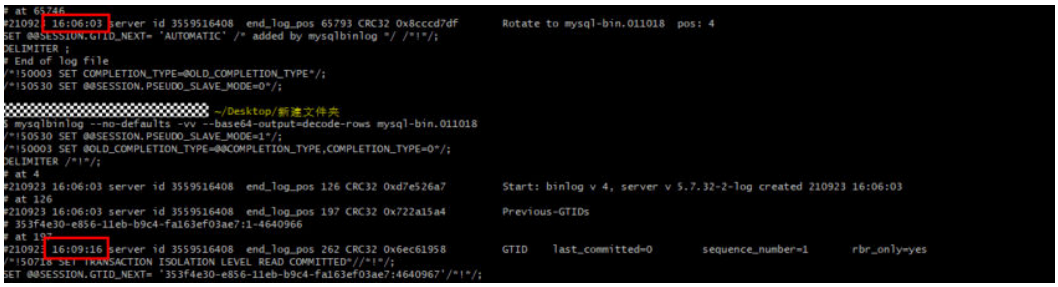
原因分析

- 此类问题与复制时延（Seconds_Behind_Master）的计算方式相关，关于复制时延的计算方式，详见**MySQL主备复制原理简介**。
- 出现复制时延尖峰是因为：只读节点IO线程刚好接收到了一个新的Binlog文件，而其SQL线程还没开始回放新的Binlog。导致计算复制时延的last_master_timestamp值还停留在上一个Binlog事务在主机的执行时间，与当前只读节点系统时间time(0)存在时间差，从而出现复制时延尖峰。当SQL线程开始解析新的Binlog时，复制时延立刻回落。
- 此类问题为偶现现象，不影响实际业务。

下载复制时延飚高回落时间段的Binlog，会发现如下现象：

新的Binlog的第一个事务执行时间与上一个Binlog最后一个事务的结束时间刚好与突增回落的时间差匹配。

版权所有 © 华为云计算技术有限公司



解决方案

无需解决。该场景为RDS for MySQL主备复制机制中的正常行为，为偶现现象。

2.6 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RDS 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

报错信息如下：

```
2021-01-10 23:58:32.964 [destination = evoicedc , address = /dbus-mysql:3306 , EventParser] ERROR xxx.common.alarm.LogAlarmHandler - destination:evoicedc[xxx.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
xxx.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;

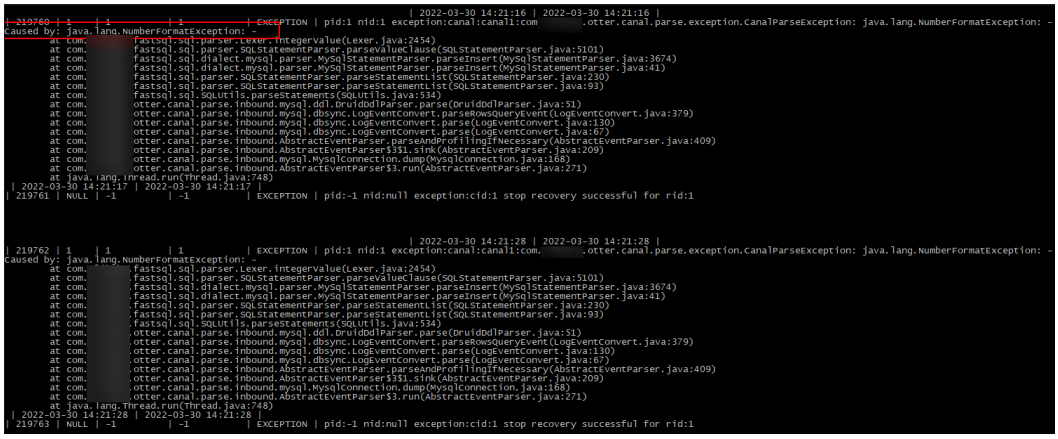
 版权所有 © 华为云计算技术有限公司

2.7 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：

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



原因分析

检查RDS 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RDS 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RDS for MySQL的参数“binlog_rows_query_log_events”的值修改为**OFF**，重启中断的canal任务。

2.8 RDS for MySQL Binlog 生成的机制

场景一

RDS for MySQL实例设置了7天的Binlog保留，按照5分钟生成一个Binlog与实际的数据量不符。

原因分析：

RDS for MySQL实例自创建完成时起，生成全量自动备份文件之后，每5分钟会生成Binlog。

如果没有数据，不会生成日志。

版权所有 © 华为云计算技术有限公司

场景二

业务量没有明显增加，但是生成的Binlog增量备份文件占用大幅增长。

原因分析：

RDS for MySQL Binlog是row模式，在row模式下，Binlog会记录修改前整行的数据和修改后的整行数据。

示例：表内有一列的数据比较大，实际**update**操作时不会更新该数据。但是Binlog会完整地记录**update**操作前后所有列的数据，导致Binlog备份文件占用增大。

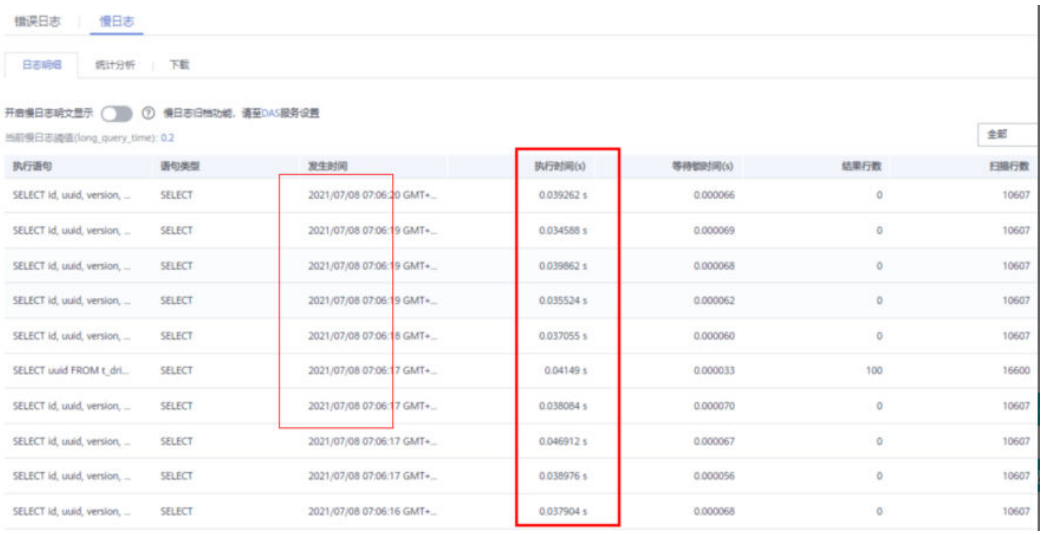
版权所有 © 华为云计算技术有限公司

3 参数类

3.1 控制台上修改 long_query_time 参数后未生效

场景描述

在控制台将“long_query_time”参数值从0.1s修改为0.2s后，慢日志中还存在小于0.2s的慢SQL执行记录。



原因分析

控制台上修改“long_query_time”参数是全局级别生效，修改完后，后续新建连接会使用最新设置的参数，但是旧连接的“long_query_time”属性值不会被改变，仍然保持旧的值（该案例中是0.1s），所以小于0.2s的慢SQL是在旧连接上产生的。

出现该现象的原因是MySQL机制导致，所以不仅“long_query_time”参数会出现此类问题，其他控制台上可以修改的全局参数，也会发生类似现象：只有新建连接生效，旧连接不生效。

版权所有 © 华为云计算技术有限公司

解决方案

如果想让某些会话连接采用最新的“long_query_time”值，关闭相应会话连接，重新建立连接即可生效。

3.2 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。

版权所有 © 华为云计算技术有限公司

3.3 RDS for MySQL 创建索引失败报错[ERROR] 1071 的解决方案

场景描述

创建索引长度超限制导致创建失败。创建表索引时出现如下错误：

[ERROR] 1071 - Specified key was too long; max key length is 3072 bytes

问题可能出现的版本：MySQL-8.0.20.5

故障分析

InnoDB表引擎有限制。

默认情况下，索引前缀长度限制为 767 字节，当开启了“innodb_large_prefix”选项时，索引前缀长度扩展到 3072 字节。

```
SHOW VARIABLES LIKE ‘%innodb_large_prefix%’;
```

```
mysql> show variables like '%innodb_large_prefix%';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| innodb_large_prefix | ON    |
+----------------------+-------+
1 row in set (0.02 sec)
```

索引前缀长度还和 InnoDB 的 page size 有关。“innodb_page_size”选项默认是 16KB 的时候，最长索引前缀长度是 3072 字节，如果是 8KB 的时候，最长索引前缀长度是 1536 字节；4KB 的时候，最长索引前缀长度是 768 字节。

```
SHOW VARIABLES LIKE ‘%innodb_page_size%’;
```

```
mysql> show variables like 'innodb_page_size';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| innodb_page_size | 16384 |
+------------------+-------+
1 row in set (0.01 sec)
```

查看问题表结构，并查询所有支持的字符集及其字节占用情况：

```
SHOW CHARACTER SET;
```


版权所有 © 华为云计算技术有限公司

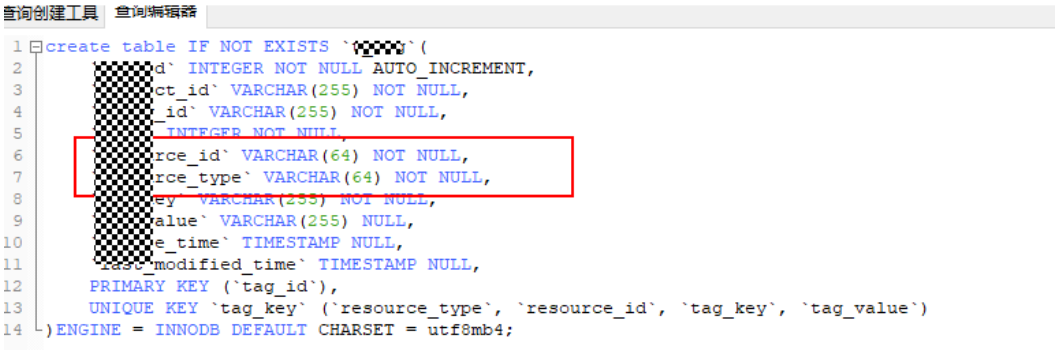
```
mysql> SHOW CHARACTER SET;
+----------+---------------------------------+---------------------+--------+
| Charset  | Description                     | Default collation   | Maxlen |
+----------+---------------------------------+---------------------+--------+
| armscii8 | ARMSCII-8 Armenian              | armscii8_general_ci |      1 |
| ascii    | US ASCII                        | ascii_general_ci    |      1 |
| big5     | Big5 Traditional Chinese        | big5_chinese_ci     |      2 |
| binary   | Binary pseudo charset           | binary              |      1 |
| cp1250   | Windows Central European        | cp1250_general_ci   |      1 |
| cp1251   | Windows Cyrillic                | cp1251_general_ci   |      1 |
| cp1256   | Windows Arabic                  | cp1256_general_ci   |      1 |
| cp1257   | Windows Baltic                  | cp1257_general_ci   |      1 |
| cp850    | DOS West European               | cp850_general_ci    |      1 |
| cp852    | DOS Central European            | cp852_general_ci    |      1 |
| cp866    | DOS Russian                     | cp866_general_ci    |      1 |
| cp932    | SJIS for Windows Japanese       | cp932_japanese_ci   |      2 |
| dec8     | DEC West European               | dec8_swedish_ci     |      1 |
| eucjpms  | UJIS for Windows Japanese       | eucjpms_japanese_ci |      3 |
| euckr    | EUC-KR Korean                   | euckr_korean_ci     |      2 |
| gb18030  | China National Standard GB18030 | gb18030_chinese_ci  |      4 |
| gb2312   | GB2312 Simplified Chinese       | gb2312_chinese_ci   |      2 |
| gbk      | GBK Simplified Chinese          | gbk_chinese_ci      |      2 |
| geostd8  | GEOSTD8 Georgian                | geostd8_general_ci  |      1 |
| greek    | ISO 8859-7 Greek                | greek_general_ci    |      1 |
| hebrew   | ISO 8859-8 Hebrew               | hebrew_general_ci   |      1 |
| hp8      | HP West European                | hp8_english_ci      |      1 |
| keybcs2  | DOS Kamenicky Czech-Slovak      | keybcs2_general_ci  |      1 |
| koi8r    | KOI8-R Relcom Russian           | koi8r_general_ci    |      1 |
| koi8u    | KOI8-U Ukrainian                | koi8u_general_ci    |      1 |
| latin1   | cp1252 West European            | latin1_swedish_ci   |      1 |
| latin2   | ISO 8859-2 Central European     | latin2_general_ci   |      1 |
| latin5   | ISO 8859-9 Turkish              | latin5_turkish_ci   |      1 |
| latin7   | ISO 8859-13 Baltic              | latin7_general_ci   |      1 |
| macce    | Mac Central European            | macce_general_ci    |      1 |
| macroman | Mac West European               | macroman_general_ci |      1 |
| sjis     | Shift-JIS Japanese              | sjis_japanese_ci    |      2 |
| swe7     | 7bit Swedish                    | swe7_swedish_ci     |      1 |
| tis620   | TIS620 Thai                     | tis620_thai_ci      |      1 |
| ucs2     | UCS-2 Unicode                   | ucs2_general_ci     |      2 |
| ujis     | EUC-JP Japanese                 | ujis_japanese_ci    |      3 |
| utf16    | UTF-16 Unicode                  | utf16_general_ci    |      4 |
| utf16le  | UTF-16LE Unicode                | utf16le_general_ci  |      4 |
| utf32    | UTF-32 Unicode                  | utf32_general_ci    |      4 |
| utf8     | UTF-8 Unicode                   | utf8_general_ci     |      3 |
| utf8mb4  | UTF-8 Unicode                   | utf8mb4_0900_ai_ci  |      4 |
+----------+---------------------------------+---------------------+--------+
```

问题所在的表的字符集是 utf8mb4 时，一个字符将占用 4 个字节。这意味着索引前缀最大长度为 3072 字节时，只能容纳 3072 / 4 = 768 个字符。因此只要将上面建表语句索引字段的前缀长度设为768或者修改索引字段，让其小于3072字节。

解决方案

修改索引字段长度，即可成功创建索引。

版权所有 © 华为云计算技术有限公司



3.4 RDS for MySQL 大小写参数敏感类问题

场景描述

用户将RDS for MySQL的“lower_case_table_names”设置成“大小写敏感”的状态时，创建了带有大写字母的表，如“tbl_newsTalking”，但后期改变了大小写敏感的设置状态后，无法找到该表。

案例：在执行备份恢复到新实例的时候，如果新实例的“大小写敏感”参数值与备份时原实例的参数值不一致，会导致恢复失败。

更多敏感参数，请参见《云数据库RDS用户指南》中“**RDS for MySQL参数调优建议**”的内容。

说明

- 对于RDS for MySQL 5.6、5.7版本，支持在管理控制台或API创建数据库实例时指定表名大小写敏感，以及实例创建完成后设置表名大小写敏感（lower_case_table_names）。
- 对于RDS for MySQL 8.0版本，仅支持在管理控制台或API创建数据库实例时指定表名大小写敏感，创建完成的MySQL 8.0实例不支持设置表名大小写敏感（lower_case_table_names）。
- 对于社区8.0版本，只能在初始化服务器时配置表名大小写敏感，禁止在服务器初始化后更改表名大小写敏感（lower_case_table_names），详见**社区文档**。

解决方案

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。

步骤3 单击页面左上角的，选择“数据库 > 云数据库 RDS”，进入RDS信息页面。

步骤4 在“实例管理”页面，单击主实例名称，进入实例的“概览”页面。

步骤5 在左侧导航栏中选择“参数修改”。

步骤6 修改“lower_case_table_names”值为“0”，即区分大小写。

步骤7 单击“保存”，在弹出框中单击“是”，保存修改。

步骤8 返回实例列表，选择“更多 > 重启实例”。

版权所有 © 华为云计算技术有限公司

步骤9 在弹框中，单击“确定”重启实例，使参数修改生效。

步骤10 登录数据库，将带大写字母的表名，改为小写字母。

步骤11 修改“lower_case_table_names”值为“1”，即不区分大小写。

步骤12 再次重启实例。

----结束

说明

- 数据库名、变量名严格区分大小写。
- 列名与列的别名在所有的情况下均是忽略大小写；字段值默认忽略大小写。
- 通过管理控制台的购买实例页面设置是否区分表名大小写。详情请参见**购买实例**。

图 3-1 数据库配置

- 通过API创建数据库实例设置“lower_case_table_names”指定大小写是否敏感。详情请参考**创建数据库实例**。

  取值范围：

  - 0：表名称大小写敏感。
  - 1：表名将被存储成小写且表名称大小写不敏感。

3.5 RDS MySQL timeout 相关参数简介

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

表 3-1 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。

版权所有 © 华为云计算技术有限公司

参数名称	修改是否需要重启	参数含义
idle_readonly_transaction_timeout	否	空闲的只读事务被kill前的等待时间，以秒为单位。(5.7.23版本之后支持)
idle_transaction_timeout	否	空闲事务被kill前的等待时间，以秒为单位。默认值设为0，代表永不kill。(5.7.23版本之后支持)
idle_write_transaction_timeout	否	空闲的读写事务被kill前的等待时间，以秒为单位。默认值设为0，代表永不kill。(5.7.23版本之后支持)
innodb_lock_wait_timeout	否	放弃事务前，InnoDB事务等待行锁的时间。
innodb_rollback_on_timeout	是	innodb_rollback_on_timeout确定后，事务超时后InnoDB回滚完整的事务。
lock_wait_timeout	否	试图获得元数据锁的超时时间（秒）。
net_read_timeout	否	中止读数据之前从一个连接等待客户端网络包的秒数。
net_write_timeout	否	中止写数据之前等待一个网络包被写入TCP连接的秒数。
interactive_timeout	否	MySQL服务端在关闭交互式连接之前等待活动的秒数。
wait_timeout	否	MySQL服务端在关闭非交互式连接之前等待活动的秒数。

3.6 命令行修改 MySQL global 参数失败

场景描述

修改MySQL global参数失败：

```
MySQL [(none)]> show variables like 'binlog_expire_logs_seconds';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| binlog_expire_logs_seconds | 3600  |
+----------------------------+-------+
1 row in set (0.00 sec)
MySQL [(none)]> set global binlog_expire_logs_seconds=600;
ERROR 1227 (42000): Access denied; you need (at least one of) the SUPER or SYSTEM_VARIABLES_ADMIN privilege(s) for this operation
MySQL [(none)]>
```


版权所有 © 华为云计算技术有限公司

解决方案

RDS for MySQL不支持在数据库中执行修改全局参数的命令，您可以到控制台修改参数。详见**是否支持使用SQL命令修改全局参数**。

 版权所有 © 华为云计算技术有限公司

4 性能资源类

4.1 CPU 使用率高问题排查与优化

场景描述

业务侧RDS 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。

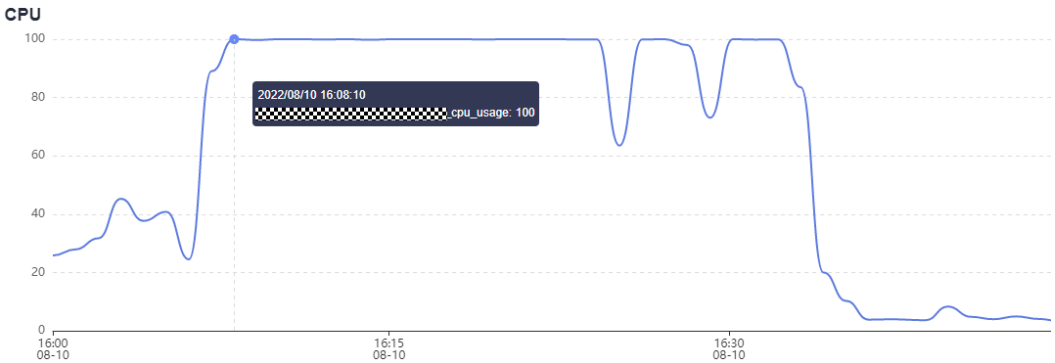
指标说明

CPU使用率：CPU执行的工作时间的比例。

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 4-1 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。

版权所有 © 华为云计算技术有限公司

图 4-2 QPS

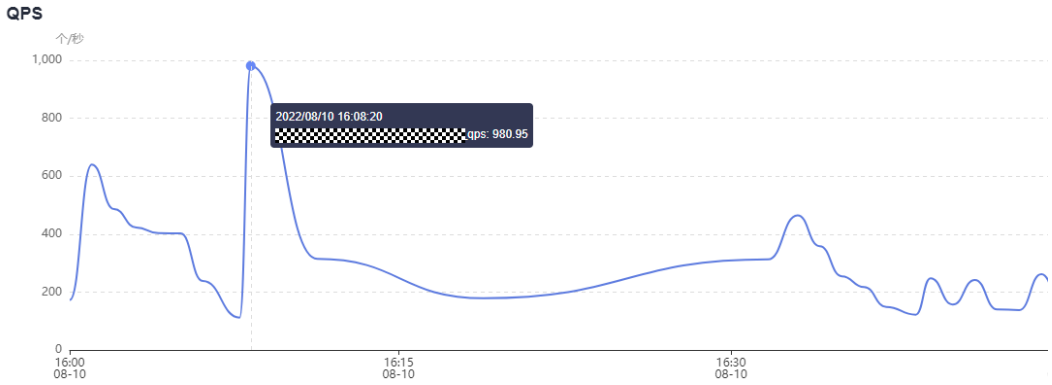


图 4-3 活跃连接数

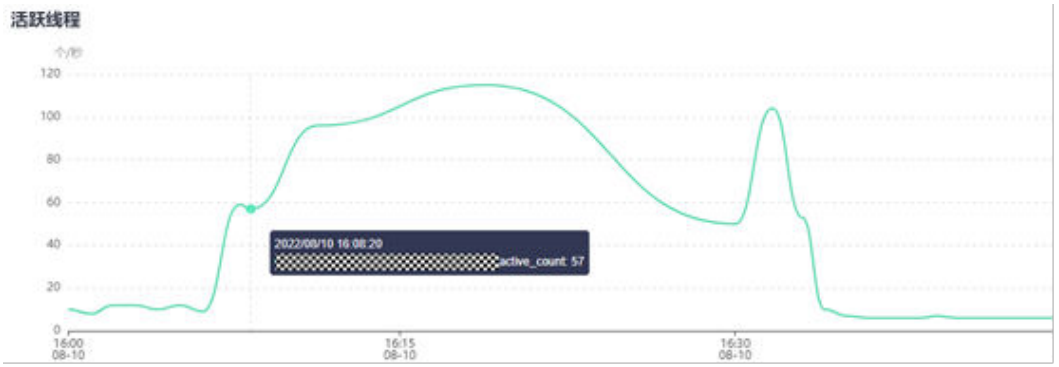
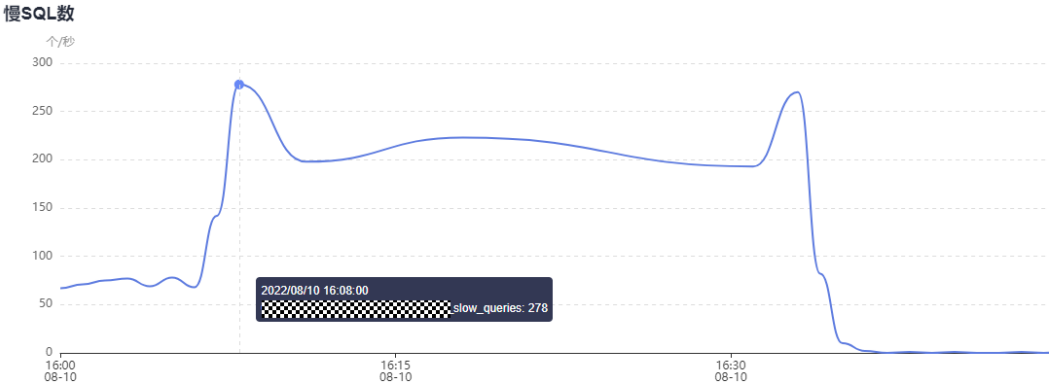


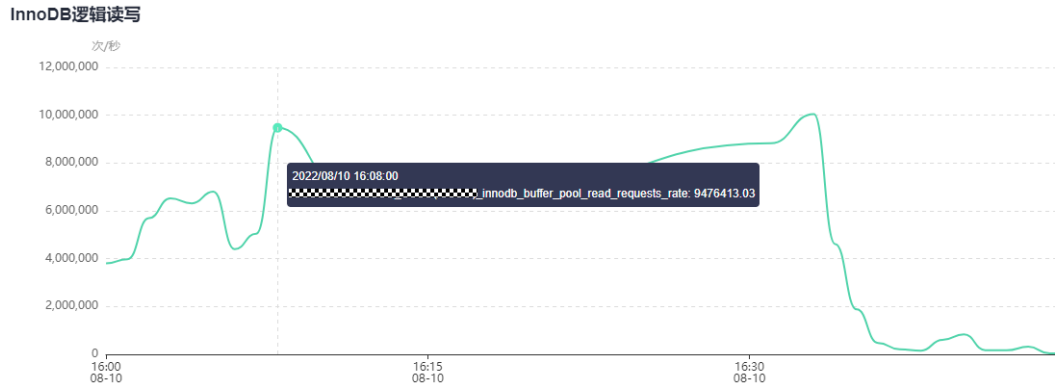
图 4-4 慢 SQL 数



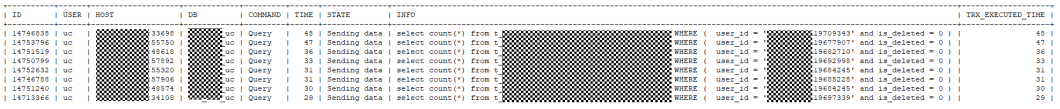
3. 分析业务类型，查看16:08分前左右InnoDB的逻辑读速率有突增，且与慢SQL的速率趋势相似。

版权所有 © 华为云计算技术有限公司

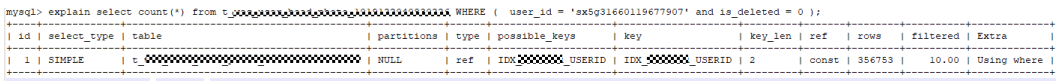
图 4-5 InnoDB 逻辑读速率



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。



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”。

4.2 内存使用超限风险与优化

RDS for MySQL 内存说明

RDS 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值分配的内存空间，并且是所有连接共享的。
- 私有内存用于每个连接到MySQL服务器时才分配各自的缓存，且只有断开连接才会释放。

版权所有 © 华为云计算技术有限公司

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）。

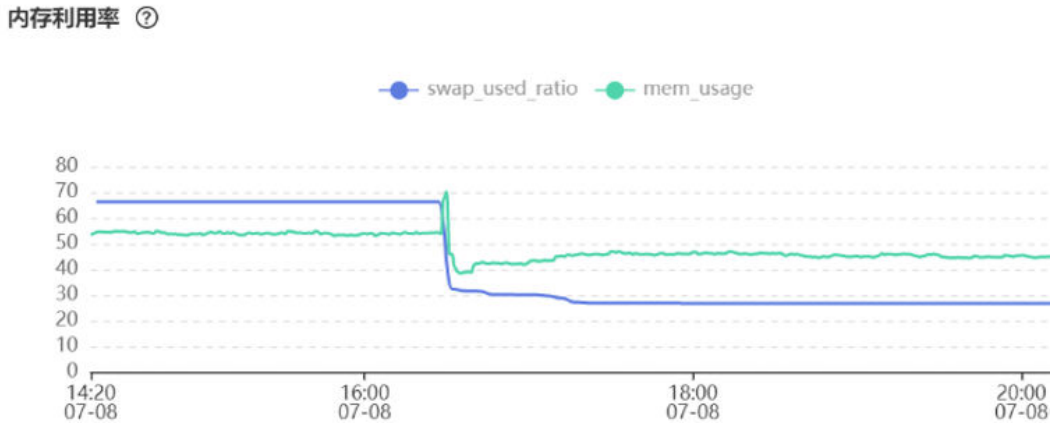
场景描述

RDS 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

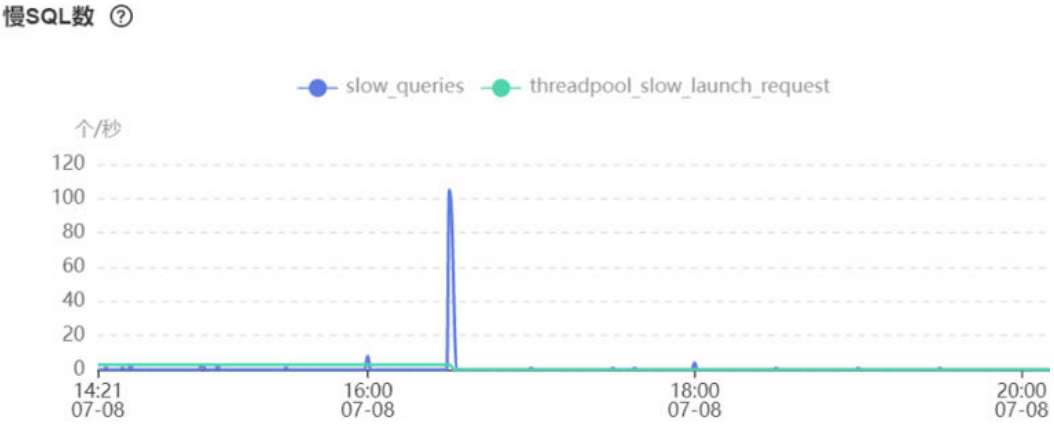
1. 查看内存利用率监控指标，实例的内存使用率在16:30左右率突增，触发OOM后实例重启，内存使用率骤降。

   图 4-6 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。

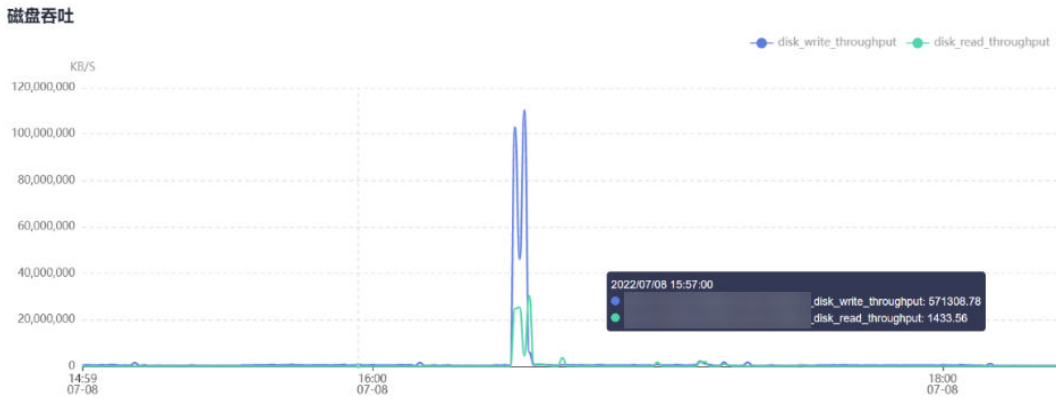
   图 4-7 慢 SQL 数

3. 查看磁盘吞吐相关指标，发现磁盘此时有大量读写操作。

版权所有 © 华为云计算技术有限公司

图 4-8 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 4-9 慢日志

INSERT INTO
(`category_id`,`id_xxxx`,`operator`,`xxxxxx_id`,`xxxxxx`,`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: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

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。

4.3 磁盘性能带宽超上限

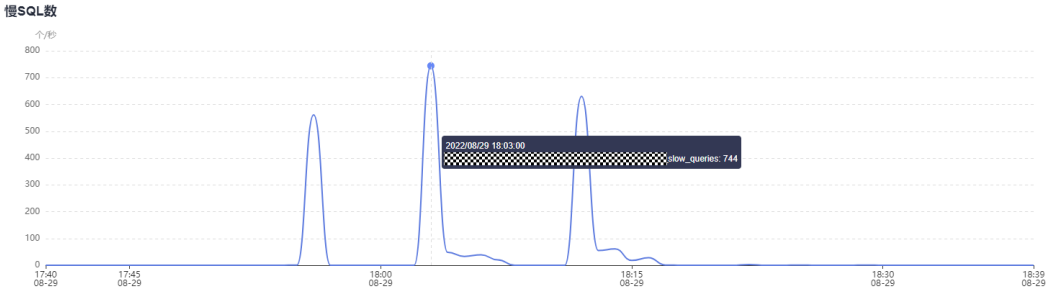
场景描述

业务侧在18:04分左右，RDS for MySQL实例业务SQL执行变慢（超过5秒），业务侧有超时返回报错。

原因分析

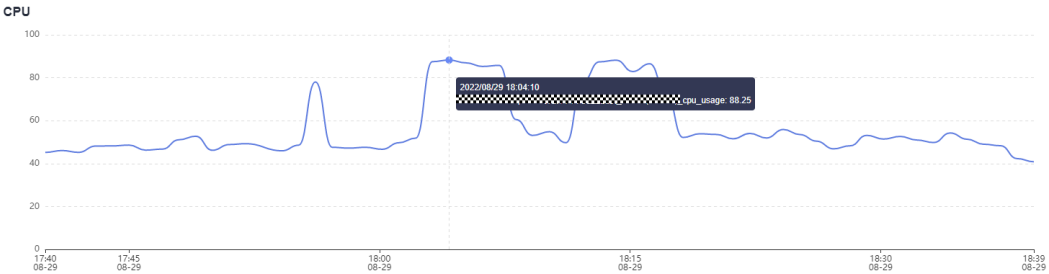
1. 查看慢SQL数监控指标，发现实例的慢SQL速率在18:03分开始上涨，且最高值达到700个/秒。

图 4-10 慢 SQL 数



2. 查看实例的CPU使用率监控指标，发现此时的CPU使用率在88%，并未达到性能瓶颈。

图 4-11 CPU 使用率



3. 查看实例的QPS监控指标，在18:03开始上涨到18:05有超过3倍增长，说明此时是业务的高峰期。

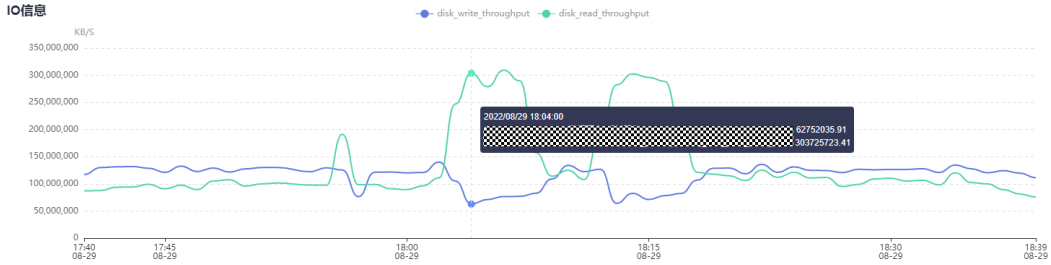
版权所有 © 华为云计算技术有限公司

图 4-12 QPS

4. 排查磁盘读写吞吐量监控指标，发现磁盘的吞吐量达到350MB/s，达到性能瓶颈。

   关于存储性能说明，请参见**数据库实例存储类型**。

图 4-13 磁盘吞吐量



解决方案

1. MySQL在读写业务时，查询更新请求的数据页如果不在Buffer Pool中，则需要读写底层存储的数据会产生物理I/O。可优先通过调整“innodb_io_capacity”或“innodb_io_capacity_max”参数来影响刷新脏页和写入缓冲池的速率，防止过高的I/O吞吐。
2. 购买高性能的极速型SSD云盘，或者升级实例内存规格将更多数据缓存到Buffer Pool解决高I/O吞吐问题。

4.4 联合索引设置不当导致慢 SQL

场景描述

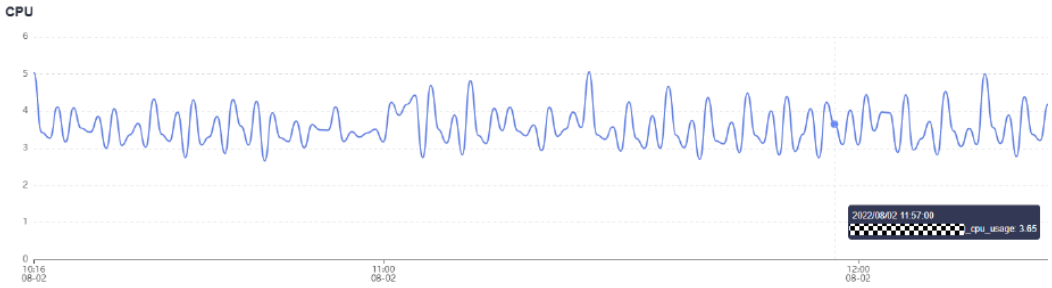
业务侧RDS for MySQL实例上以往执行耗时8秒的查询，在11:00后耗时超过30秒。

原因分析

1. 查看查询变慢对应的时间段中，实例CPU监控指标并无飙升情况且使用率一直都较低，因此排除了CPU冲高导致查询变慢的可能。

版权所有 © 华为云计算技术有限公司

图 4-14 CPU 使用率



2. 分析对应时间段该实例的慢日志，该SQL执行快时其扫描行数为百万级，当SQL执行慢时其扫描行数为千万级，与业务确认该表短期内并无大量数据插入，因此推断执行慢是因为未走索引或选错索引。且通过**EXPLAIN**查看该SQL的执行计划确实是全表扫描。

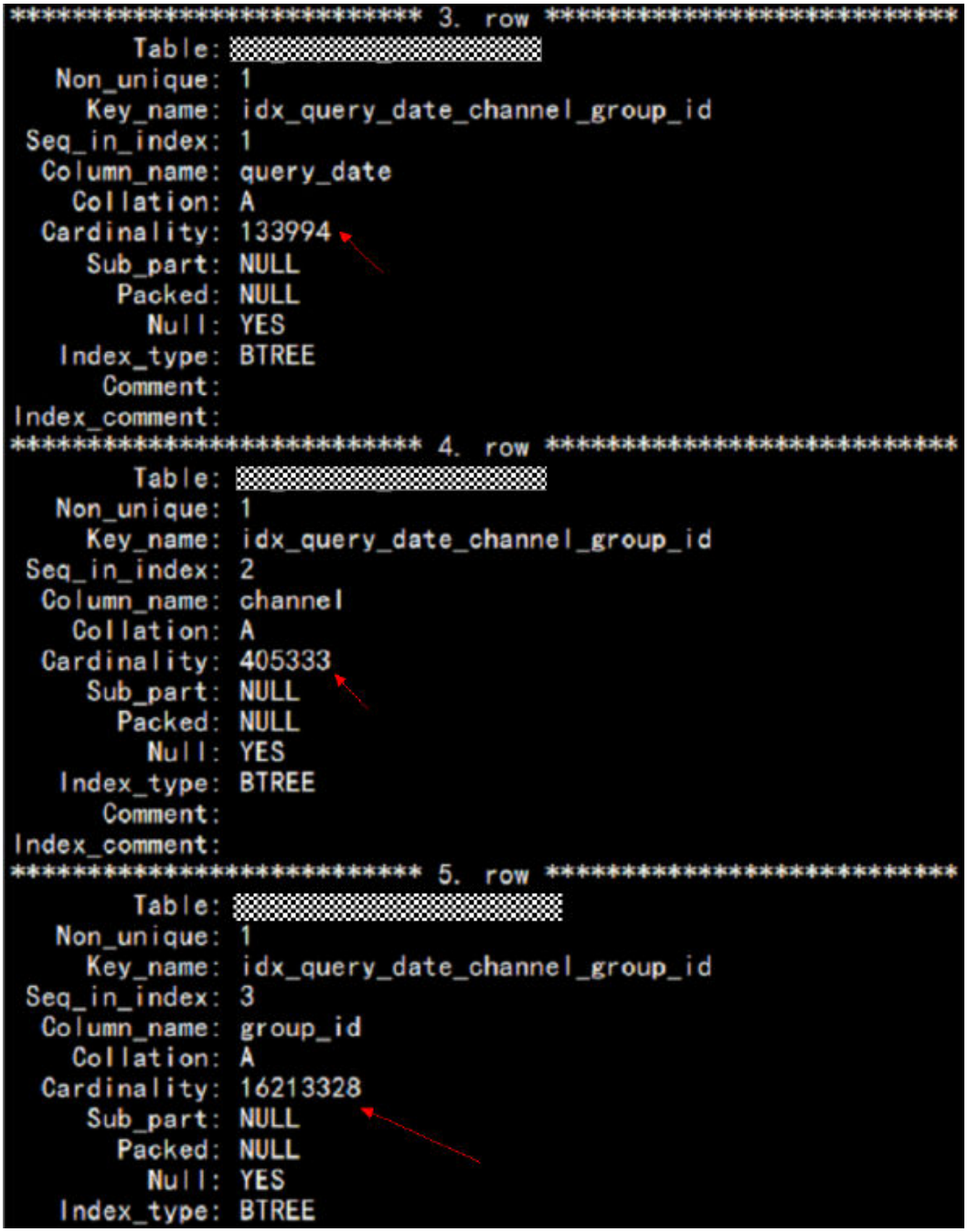
图 4-15 慢日志

select query_date, sum(queue) queue, sum(server_user_num) serverUserNum, su...	SELECT	1	6.027126 s	0.000105	125	2119000
select query_date, sum(queue) queue, sum(server_user_num) serverUserNum, su...	SELECT	1	5.479857 s	0.000104	123	2085096
select query_date, sum(queue) queue, sum(server_user_num) serverUserNum, su...	SELECT	1	5.288658 s	0.000106	123	2085096
select query_date, sum(queue) queue, sum(server_user_num) serverUserNum, su...	SELECT	1	33.601792 s	0.000064	140	16961077
select query_date, sum(queue) queue, sum(server_user_num) serverUserNum, su...	SELECT	1	34.342761 s	0.000171	140	16961077
select query_date, sum(queue) queue, sum(server_user_num) serverUserNum, su...	SELECT	1	44.536072 s	0.000167	140	16961077
select query_date, sum(queue) queue, sum(server_user_num) serverUserNum, su...	SELECT	1	46.501796 s	0.000095	140	16961077
select query_date, sum(queue) queue, sum(server_user_num) serverUserNum, su...	SELECT	1	33.050387 s	0.000099	139	16944097
select query_date, sum(queue) queue, sum(server_user_num) serverUserNum, su...	SELECT	1	38.523306 s	0.000101	139	16944097
select query_date, sum(queue) queue, sum(server_user_num) serverUserNum, su...	SELECT	1	40.108127 s	0.000090	139	16944097

3. 在实例上对该表执行**SHOW INDEX FROM**检查三个字段的索引区分度（或基数）。

版权所有 © 华为云计算技术有限公司

图 4-16 查看索引区分度



可知基数最小的字段“query_date”在联合索引的第一位，基数最大的字段“group_id”在联合索引最后一位，而且原SQL包含对“query_date”字段的范围查询，导致当索引走到“query_date”就会停止匹配，后面两个字段已经无序，无法走索引。所以该SQL本质上只能利用到对“query_date”这一列的索引，而且还有可能因为基数太小，导致优化器成本估计时选择了全表扫描。

业务重新创建了联合索引将“group_id”字段放在第一位，“query_date”字段放在最后一位后，查询耗时符合预期。

 版权所有 © 华为云计算技术有限公司

解决方案

1. 查询变慢首先确认是否由于CPU等资源达到性能瓶颈导致执行慢。
2. 库表结构设计不合理，索引缺失或索引设置不恰当会导致慢SQL。
3. 表数据大批量插入删除等操作可能会导致统计信息未能及时更新，建议定期执行**ANALYZE TABLE**防止执行计划走错。

4.5 数据库磁盘满导致被设置 read_only

场景描述

业务侧出现如下报错：

```
The MySQL server is running with the --read-only option so it cannot execute this statement
```

原因分析

1. 进入实例详情页面，查看磁盘空间是否已满。

2. 登录数据库，查看read_only变量。
   show variables like 'read_only';
3. 分析原因为实例磁盘空间满，数据库状态变更为只读，导致SQL语句执行失败。
4. 通过智能DBA助手查看磁盘空间占用分布，具体操作请参见**容量预估**。

版权所有 © 华为云计算技术有限公司

图 4-17 磁盘空间分布

解决方案

1. 随着业务数据的增加，原来申请的数据库磁盘容量可能会不足，建议用户**扩容磁盘空间**，确保磁盘空间足够。

   如果原有规格的磁盘已是最大，请先**升级规格**。

   云盘实例可以设置**存储空间自动扩容**，在实例存储空间达到阈值时，会触发自动扩容。
2. 针对数据空间过大，可以删除无用的历史表数据。
   a. 如果实例变为只读状态，您需要先**解除只读**状态；如果实例非只读状态，则可以直接执行删除操作。
   b. 查看物理文件大小Top50库表，识别可以删除的历史表数据，具体操作请参见**容量预估**。
   c. 可在业务低峰期对碎片率高的表执行optimize优化，以便释放空间：

      清理整张表使用**DROP**或**TRUNCATE**操作；删除部分数据，使用**DELETE**操作，如果是执行**DELETE**操作，需要使用**OPTIMIZE TABLE**来释放空间。
3. 如果是RDS for MySQL Binlog日志文件占用过多，可以**清理本地Binlog日志**，来释放磁盘空间。
4. 针对大量排序查询导致的临时文件过大，建议优化SQL查询。

   查询数据库**慢SQL**和**Top SQL**，分析数据量大，行数多，响应时间长的SQL语句，并进行优化。
5. 您还可以订阅实例健康日报来获取SQL及性能分析结果，包括慢SQL分析、全量SQL分析、性能 & 磁盘分析、性能指标趋势图，当发生风险点时及时收到诊断报告。

   具体操作请参见**诊断日报**。

版权所有 © 华为云计算技术有限公司

4.6 Binlog 未清理导致磁盘占用高

场景描述

只读实例或主实例磁盘占用高，通过执行**SHOW BINARY LOGS**或**SHOW MASTER LOGS**，和其他实例对比发现，大量老的Binlog文件未被清理，导致磁盘占用很高。

原因分析

正常情况下，设置了**Binlog过期时间**，当Binlog备份至OBS，且超过过期时间后，会自动清理，如果长时间未清理，需考虑是否有其他复制异常因素导致。

排查思路：

1. **查看MySQL的错误日志**，查找是否有类似无法purge binlog的日志记录。
   2022-01-18T05:39:03.139207+08:00 29 [Warning] file ./mysql-bin.106259 was not purged because it was being readby thread number 27490757
2. 分析是否有本地搭建复制关系、使用canal等工具监听该实例的Binlog，当主库未收到对应Binlog已被从库或工具获取的信息，会导致对应Binlog不被删除，导致Binlog积压。
3. 结合1中的异常binlog purge记录，分析本地从库或canal工具相应日志，排查网络状况等原因确认Binlog未被清理的原因。

解决方案

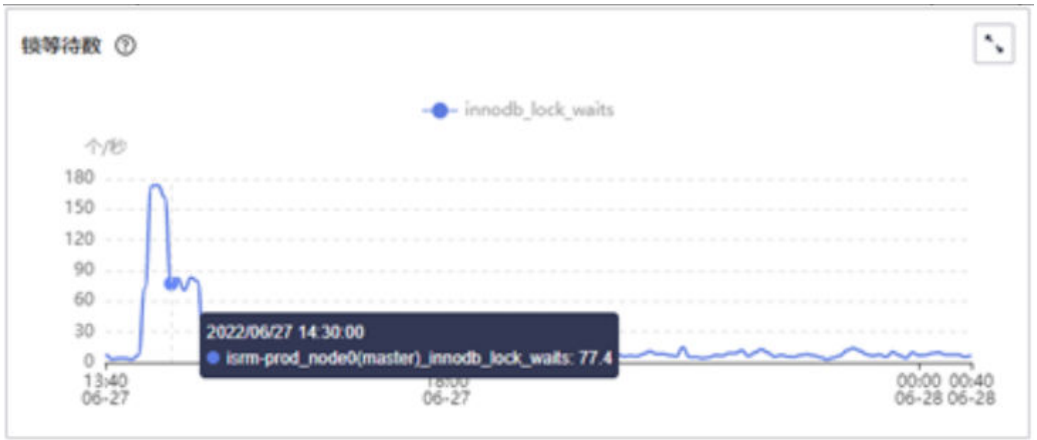
- 暂时停止该实例的其他Binlog监听任务，让该实例自动清理Binlog。
- 如果有本地从库，重新搭建复制关系。
- 如果使用canal等工具，重新建立Binlog拉取任务。

4.7 业务死锁导致响应变慢

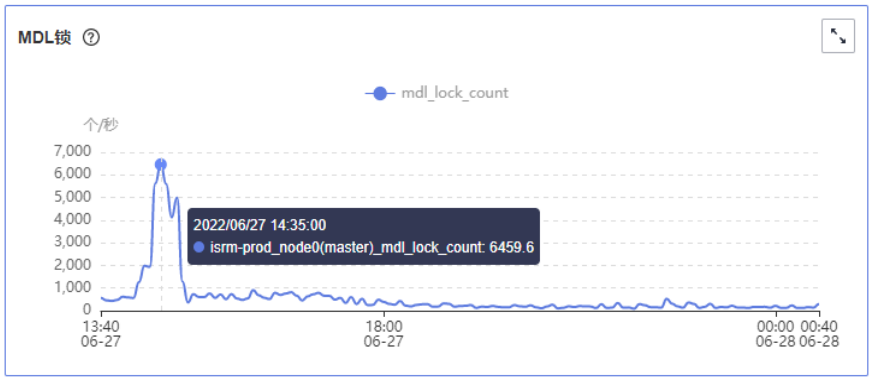
场景描述

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：



版权所有 © 华为云计算技术有限公司



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。

4.8 MySQL 只读实例磁盘占用远超主实例

场景描述

MySQL只读实例的磁盘占用比主实例高195GB。

原因分析

排查只读实例上运行的事务：

版权所有 © 华为云计算技术有限公司

```
         trx_waited_time: 0
*************************** 12. row ***************************
                  trx_id: 422175775626376
               trx_state: RUNNING
             trx_started: 2022-07-22 11:00:26
   trx_requested_lock_id: NULL
        trx_wait_started: NULL
              trx_weight: 0
       trx_mysql_thread_id: 575769551
                 trx_query: SELECT `movie_id` FROM `hg_app_pay_movie` WHERE  `member_id` = '12831953'
       trx_operation_state: fetching rows
         trx_tables_in_use: 1
         trx_tables_locked: 0
          trx_lock_structs: 0
   trx_lock_memory_bytes: 1136
         trx_rows_locked: 0
        trx_rows_modified: 0
  trx_concurrency_tickets: 0
      trx_isolation_level: REPEATABLE READ
        trx_unique_checks: 1
   trx_foreign_key_checks: 1
trx_last_foreign_key_error: NULL
 trx_adaptive_hash_latched: 0
 trx_adaptive_hash_timeout: 0
          trx_is_read_only: 1
trx_autocommit_non_locking: 1
       trx_schedule_weight: NULL
        trx_waited_time: 0
*************************** 13. row ***************************
                  trx_id: 422175775614616
               trx_state: RUNNING
             trx_started: 2022-07-21 15:27:39
   trx_requested_lock_id: NULL
        trx_wait_started: NULL
              trx_weight: 0
       trx_mysql_thread_id: 546262844
                 trx_query: SELECT COUNT(*) AS tp_count FROM `hg_member_history` `a` LEFT JOIN `hg_member_info` `b` ON `a`.`member_id`=`b`.`id` LEFT JOIN `hg_app_movie
`c` ON `a`.`movie_id`=`c`.`id` LIMIT 1
       trx_operation_state: starting index read
         trx_tables_in_use: 3
         trx_tables_locked: 0
          trx_lock_structs: 0
   trx_lock_memory_bytes: 1136
         trx_rows_locked: 0
        trx_rows_modified: 0
  trx_concurrency_tickets: 0
```

发现有一直未提交的长事务，如上图所示（事务一天前开始），该长事务导致undo log一直未清理，累计了近一天的undo log，导致磁盘占用高。

解决方案

- 方式一：等待事务提交后，undo log会被清理，只读实例的磁盘占用恢复。
- 方式二：kill相应会话，停止长事务。

4.9 RDS for MySQL CPU 升高定位思路

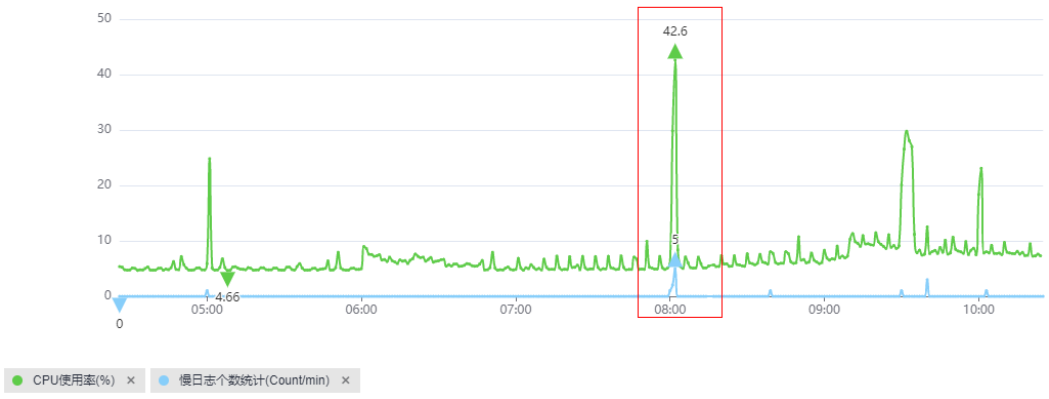
RDS for MySQL实例CPU升高或100%，引起业务响应慢，新建连接超时等。

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：

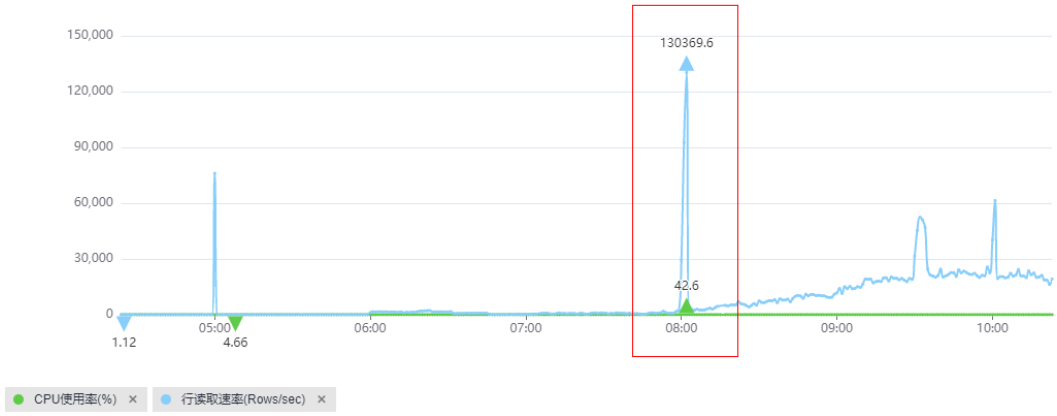
查看CPU使用率和慢日志个数统计监控指标。



- 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。

版权所有 © 华为云计算技术有限公司

- 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+只读实例架构，实现读写分离。只读实例专门负责查询，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

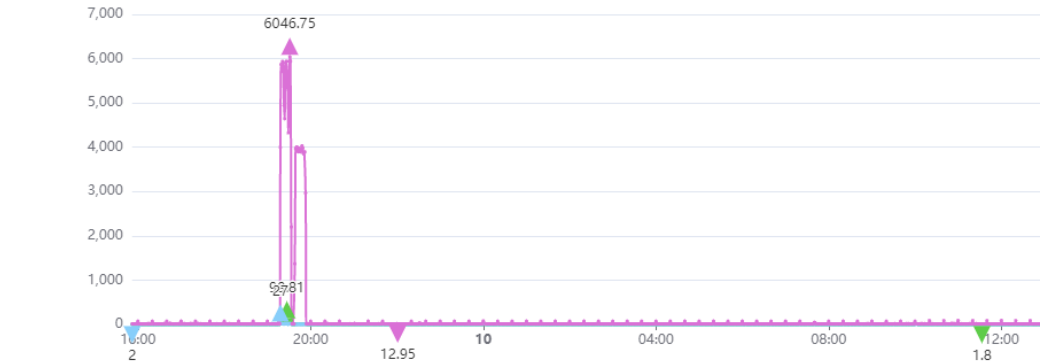
问题原因：业务请求增高导致实例CPU升高，需要从业务侧分析请求变化的原因。

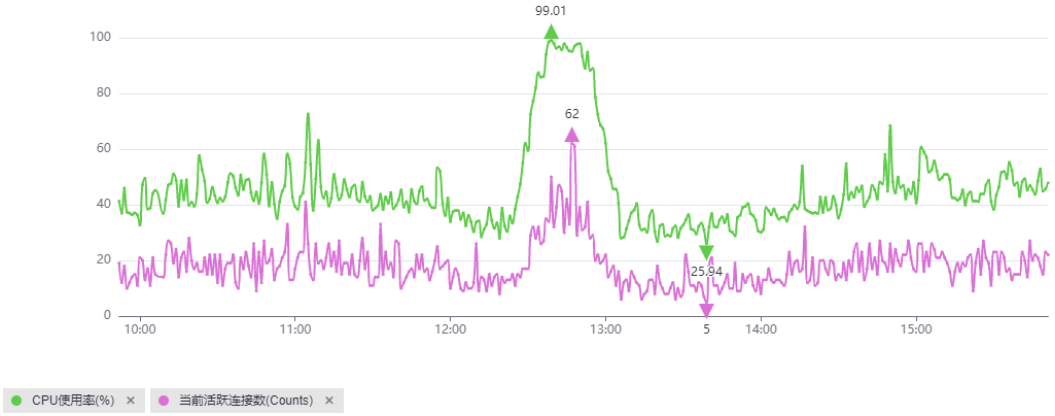
排查思路：

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：

版权所有 © 华为云计算技术有限公司





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。例如：1U、2U、4U，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少单次查询访问的数据量。
4. 使用数据库代理+只读实例架构，实现读写分离。只读实例专门负责查询，减轻主库压力，提升数据库吞吐能力，详见**读写分离简介**。

4.10 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RDS for MySQL实例，发现同一条sql语句执行性能远差于原数据库。

版权所有 © 华为云计算技术有限公司

原因分析

同一条sql语句在数据库中执行第1次和第2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1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RDS for MySQL时，第1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

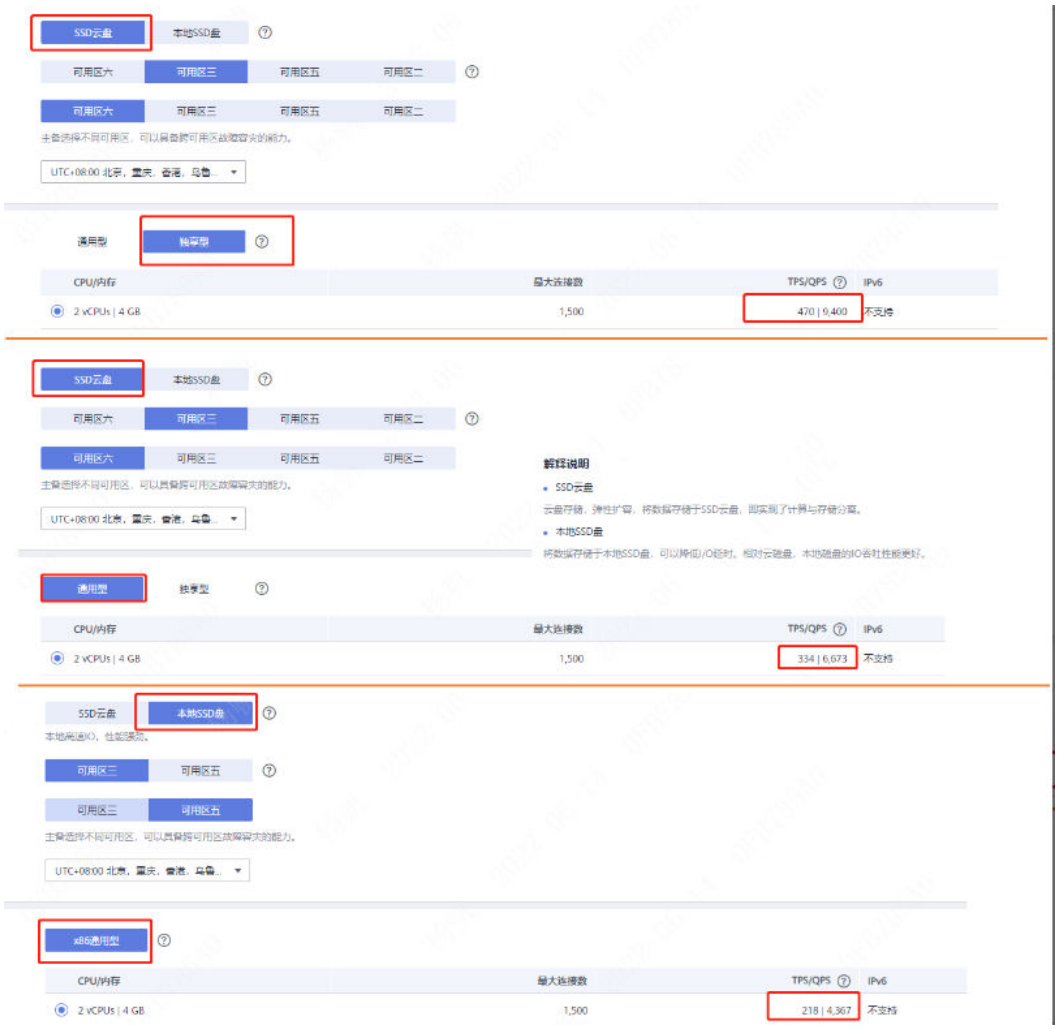
该场景是正常现象，在同一个数据库中，我们经常会遇到第1次执行一条语句时很慢，但再次执行就很快，也是因为受到了buffer_pool的冷热数据原理的影响。

4.11 CPU/内存配置与 TPS 和 QPS 性能相关的问题

场景 1

问：如下图中，为什么本地SSD盘支持更高的硬盘吞吐量，但TPS和QPS反而比SSD云盘低?

版权所有 © 华为云计算技术有限公司



同样2U8GB的配置，本地SSD盘的性能比SSD云盘更好，但2U4GB配置下本地SSD盘性能不如SSD云盘。

 版权所有 © 华为云计算技术有限公司

答：

云盘性能比本地盘好的原因：本地盘有IOPS限制，2U4GB规格配置下本地盘的带宽上限不如云盘，导致当前压测模型下本地盘性能不如云盘。

结合业务实际情况选择合适的磁盘类型：

1. 对于小规格，由于本地盘IOPS限制，导致本地盘的带宽优势无法发挥，因此可能云盘性能会更好。
2. 规格增大后，由于云盘有磁盘带宽上限的限制，当磁盘带宽成为瓶颈时，本地盘优势才会体现。
3. 对于时延比较敏感的业务，本地盘会更占优势。

场景 2

问：本地盘IOPS限制是磁盘本身限制吗？

答：本地盘IOPS限制是为了性能稳定。磁盘能力是有限的，如果所有实例都不限制的话，都会争抢IO能力。

场景 3

问：规格增大后，达到什么标准本地盘性能高于云盘？

答：针对磁盘瓶颈的模型下，一般达到8U及以上规格，本地盘的IO能力可以超过云盘的能力上限，此时本地盘性能可以优于云盘。

场景 4

问：是否对于时延比较敏感的业务，小规格的本地盘会有优势？

答：本地盘是单次IO访问时延比云盘小，和规格没有关系。

版权所有 © 华为云计算技术有限公司

4.12 表碎片率过高可能导致的问题

场景描述

在使用RDS for MySQL过程中，经常遇到表碎片率过高的问题。表碎片是指在MySQL数据库中，表的数据和索引分散在不同的物理块中，这些物理块可能不连续，或者有一些空闲的空间，从而导致表的数据和索引在磁盘上的存储不是最优的。

这种现象主要是由于MySQL表的数据操作（比如删除、更新、插入等操作）引起的，会使得表中的数据行不断地被修改和移动，从而导致表中的数据片段变得不连续。

影响及风险

- 表空间膨胀

  表碎片率过高会导致数据库中存在大量未使用的空间，这些空间无法重复利用，从而浪费磁盘空间。
- 查询优化不佳

  表碎片率过高会导致优化器无法正确及有效地利用索引，从而影响执行计划的选择，导致查询性能下降。
- SQL执行效率降低

  表碎片率过高会导致数据库执行SQL时需要花费额外的时间进行I/O扫描及整理碎片，导致查询和更新操作变慢，响应时间变长。

排查方法

方法一：通过智能DBA功能实时查看实例磁盘空间使用情况，避免出现磁盘空间不足的问题。

1. 登录管理控制台。
2. 单击管理控制台左上角的，选择区域。
3. 单击页面左上角的，选择“数据库 > 云数据库 RDS”，进入RDS信息页面。
4. 在“实例管理”页面，选择目标实例，单击实例名称，进入实例的“概览”页签。
5. 在左侧导航栏选择“智能DBA助手 > 实时诊断”。
6. 选择“容量预估”页签，页面中的Top50库表空间统计中可以查看碎片空间及碎片率数据。

图 4-18 Top50 库表

方法二：通过命令方式查看碎片率数据。

版权所有 © 华为云计算技术有限公司

1. 使用以下命令对表进行分析，更新表的统计信息。
   ```
   ANALYZE TABLE table_name;
   ```
2. 使用以下命令可以查看表的详细信息。
   ```
   SELECT
      table_name,
      data_length,
      data_free
   FROM
      information_schema.tables
   WHERE
      table_schema = 'database_name'
      AND
      table_name = 'table_name';
   ```
   - table_name：表名称。
   - data_length：表存储的数据大小（单位：字节）。
   - data_free：表剩余的空闲空间大小（单位：字节）。

   一般来说，可以通过data_free占data_length的比例来初步判断碎片率，估计碎片情况。

原因分析

原因1：DRS全量迁移阶段并行迁移导致

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RDS for MySQL后，更高的碎片率可能会导致数据膨胀，使得磁盘空间使用远大于源端。

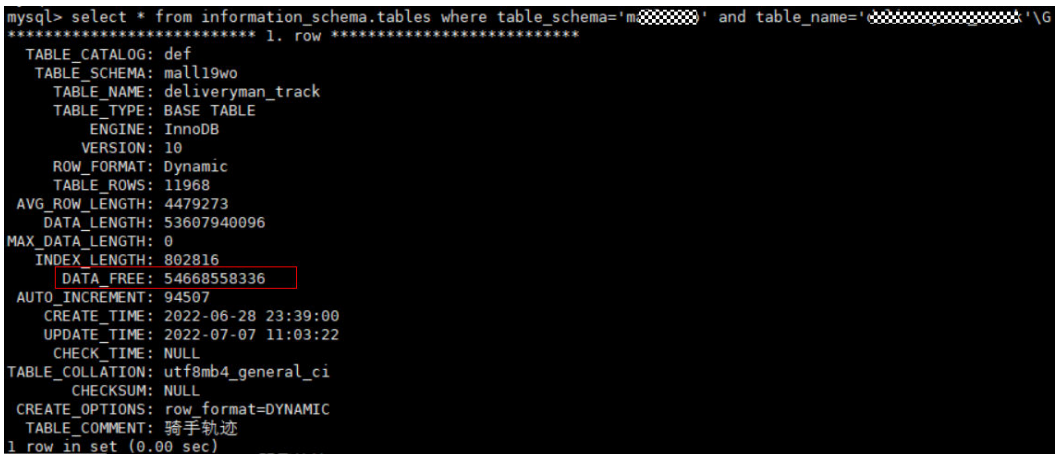
原因2：大量删除操作后在表空间留下碎片

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，如果没有数据来及时填补这些空间，就造成了表空间膨胀，形成大量碎片。

可以通过如下SQL语句，查询某个表的详细信息，回显中的“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;
```

图 4-19 回显信息



 版权所有 © 华为云计算技术有限公司

解决方案

以下场景可能需要优化表碎片率：

- 数据库长时间运行。
  长时间运行产生的插入、更新和删除等数据操作可能会产生表碎片。
- 大量数据变更
  数据库大量的数据变更，可能会导致碎片产生。
- 数据库性能下降
  当同样数据量级的查询，用户识别到了明显的性能下降，可能需要排查碎片率的影响。
- 磁盘空间不足
  磁盘空间使用率过高的情况下，可以排查碎片空间的大小，通过整理碎片达到释放磁盘空间的效果。

针对表碎片率过高的问题，建议定期对频繁访问的表做碎片分析并清理碎片，进行表空间优化整理，从而缩小空间，优化性能。

执行如下命令对表进行分析：

```
OPTIMIZE TABLE table_name;
```

须知

optimize table命令会有短暂锁表操作，整体执行时间与表大小有关。一般执行时间较长，且较为占用资源（必须预留被optimize表1.5倍大小的磁盘空间），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。

4.13 复杂查询造成磁盘满

场景描述

主机或只读实例偶尔出现磁盘占用高或磁盘占用满，其他只读实例或者备机磁盘空间占用正常。

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. 因为其他只读实例和备机磁盘占用空间正常，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。

版权所有 © 华为云计算技术有限公司

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。

4.14 怎么解决查询运行缓慢的问题

1. 通过查看慢SQL日志来确定是否存在运行缓慢的SQL查询以及各个查询的性能特征（如果有），从而定位查询运行缓慢的原因。

   查询RDS for MySQL日志，请参见**查看或下载慢日志**。
2. 查看实例的CPU使用率指标，协助定位问题。

   请参见**查看性能指标**。
3. 创建只读实例专门负责查询。减轻主实例负载，分担数据库压力。
4. 多表关联查询时，关联字段要加上索引。
5. 尽量避免用select*语句进行全表扫描，可以指定字段或者添加where条件。

4.15 长事务导致规格变更或小版本升级失败

场景描述

长事务导致实例规格变更失败或小版本升级失败。

原因分析

- 规格变更过程或者小版本升级由于采用滚动执行的方式，最大程度减小对客户业务的影响，因此需要做主备切换。
- 主备切换时，为了保证数据一致性，需要先把主机设置readonly后，让主备的执行事务完全一致，才进行切换。
- 当主机上有长事务时，会导致主机设置readonly超时或失败，从而导致规格变更或小版本升级失败。

解决方案

1. 执行**show processlist**，查看正在执行的事务，使用如下命令，找到长事务。

   select t.*,to_seconds(now())-to_seconds(t.trx_started) idle_time from INFORMATION_SCHEMA.INNODB_TRX t \G;

   示例输出：

 版权所有 © 华为云计算技术有限公司

```
mysql> select t.*,to_seconds(now())-to_seconds(t.trx_started) idle_time from INFORMATION_SCHEMA.INNODB_TRX t \G
*************************** 1. row ***************************
                    trx_id: 6168
                 trx_state: RUNNING
               trx_started: 2021-09-16 11:08:27
     trx_requested_lock_id: NULL
          trx_wait_started: NULL
                trx_weight: 3
       trx_mysql_thread_id: 231
                 trx_query: NULL
       trx_operation_state: NULL
         trx_tables_in_use: 0
         trx_tables_locked: 1
          trx_lock_structs: 3
     trx_lock_memory_bytes: 1136
           trx_rows_locked: 2
         trx_rows_modified: 0
   trx_concurrency_tickets: 0
       trx_isolation_level: REPEATABLE READ
         trx_unique_checks: 1
    trx_foreign_key_checks: 1
trx_last_foreign_key_error: NULL
 trx_adaptive_hash_latched: 0
 trx_adaptive_hash_timeout: 0
          trx_is_read_only: 0
trx_autocommit_non_locking: 0
                idle_time: 220
```

上述结果中idle_time是计算产生的，也是事务的持续时间。trx_mysql_thread_id是该事务的thread_id，和**show processlist**中的线程ID对应。

事务的trx_query是NULL，但不表示未执行事务，一个事务可能包含多个SQL，如果SQL执行完毕就不再显示。如果事务正在执行，InnoDB也无法得知该事务后续还有没有SQL以及提交时间。此时，trx_query不能提供有意义的信息，因此为NULL。

2. kill长事务，再进行规格变更或小版本升级。
3. 建议客户在进行规格变更或小版本升级时，避免长事务执行。

4.16 RDS for MySQL 数据库报错 Native error 1461 的解决方案

场景描述

RDS for 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”。

版权所有 © 华为云计算技术有限公司

4.17 RDS for MySQL 增加表字段后出现运行卡顿现象

故障描述

当给RDS for 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。

4.18 长事务导致 UNDO 增多引起磁盘空间满

场景描述

实例触发磁盘满告警，一段时间后磁盘满告警自动恢复。

原因分析

- 由于MVCC机制，MySQL更新表中数据时会生成undo日志，会占用磁盘空间；所有会话的相关事务提交或回滚后，undo日志会被清理，导致磁盘空间下降。
- 当存在长事务时，长事务只要不提交，其他会话对相关表更新生成的undo就无法清理，导致磁盘空间一直上涨。

排查思路：

- 通过如下语句，检查是否有长时间不提交事务。

  select t.*,to_seconds(now())-to_seconds(t.trx_started) idle_time from INFORMATION_SCHEMA.INNODB_TRX t \G;
- 通过审计日志或慢日志，检查是否存在大事务一次性插入大量数据。

解决方案

- kill相应的长事务。
- 建议业务侧避免在磁盘空间紧张时，执行长事务不提交，或执行大量插入。

版权所有 © 华为云计算技术有限公司

- 提前进行磁盘扩容。

4.19 RDS for MySQL 如何定位一直存在的长事务告警

场景描述

长事务告警一直存在，如何定位长事务。

原因分析

执行以下语句，查看当前事务的运行时间，根据运行时间定位长事务。

Select t.*,to_seconds(now())-to_seconds(t.trx_started) idle_time from INFORMATION_SCHEMA.INNODB_TRX t;

执行语句后返回的参数“trx_query”是当前事务执行的SQL语句，如果参数值为**NULL**，则表示当前事务在等待状态下不执行SQL。

具体操作请参考MySQL官方文档。

4.20 RDS for MySQL 部分 SQL 的 commit 时间偶现从几毫秒陡增到几百毫秒

场景描述

RDS for MySQL部分SQL的commit时间偶现从几毫秒陡增到几百毫秒。

原因分析

开启线程池时，SQL请求需通过任务队列进入worker线程处理，在低并发长连接时并无性能优化作用，可能导致由于线程池调度机制偶现短暂延迟。

在高并发或大量短连接的情况下，可能会因为大量创建和销毁线程以及上下文切换导致性能劣化。

解决方案

线程池参数“threadpool_enabled”设置为关闭，然后在业务低峰期重启应用或者数据库，并观察延迟情况。已建立的连接不会生效，针对新建立的连接会立即生效。

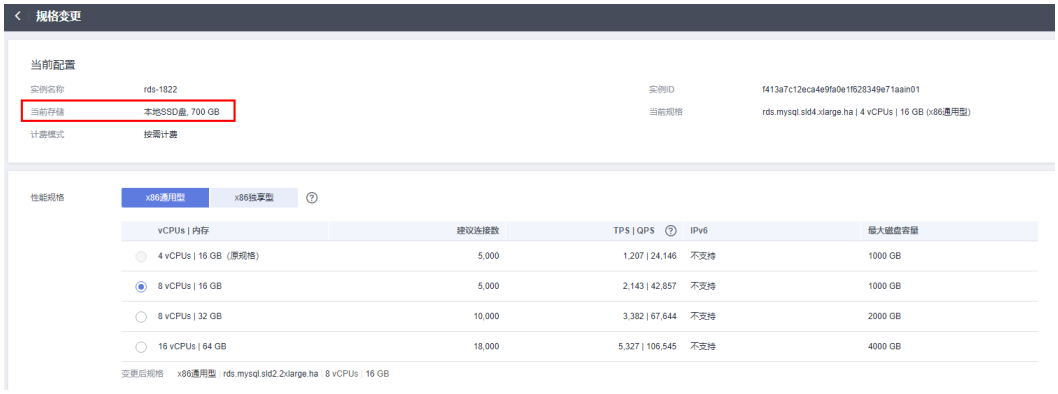
4.21 本地 SSD 盘规格降配选不到资源

场景描述

RDS for MySQL本地SSD盘实例当前规格为4 vCPUs | 16 GB，当前存储容量700GB，需要降配至4 vCPUs | 8 GB，界面上选不到4 vCPUs | 8 GB资源。

版权所有 © 华为云计算技术有限公司

图 4-20 规格变更



原因分析

本地SSD盘规格变更时，变更后的目标规格的存储上限不能小于当前实例的存储容量。

参考**RDS for MySQL实例规格**，本地SSD盘4 vCPUs | 8 GB的最大存储容量是500GB，小于实例当前存储容量700GB，所以选不到4 vCPUs | 8 GB资源。

4.22 ibdata1 为什么会变大

原因分析

ibdata1是InnoDB的系统表空间，主要包括：

- 多版本并行事务控制（MVCC）相关的数据：undolog
- Innodb表的元数据，如数据字典
- change buffer/double write buffer等

其中，undolog是ibdata1增大的最主要原因，而undolog过大的主要原因如下：

- 长事务久未提交，导致undolog purge被阻塞。
- 写入并发太大生成大量的undolog，purge速度跟不上。

通过**show engine innodb status**中的“History list length”可以查看未被purge的undolog数量。

解决方案

- 如果主库的ibdata1过大，备库的不大，则进行主备倒换。
- 如果备库的ibdata1过大，主库的不大，请**提交工单**联系客服重建备库。
- 如果一个实例所有库的ibdata1都大，则通过DRS迁移数据。

版权所有 © 华为云计算技术有限公司

5 SQL 类

5.1 RDS for MySQL 执行 SQL 报错无法识别双引号

场景描述

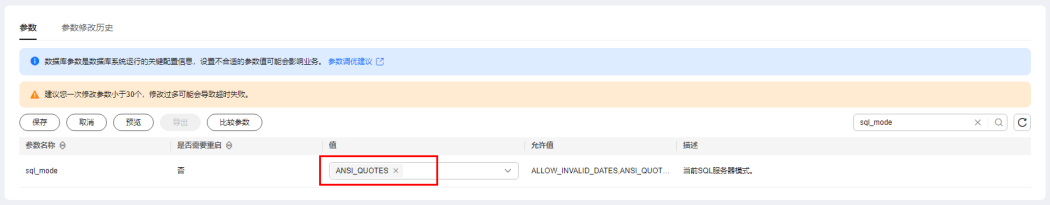
RDS for MySQL执行SQL语句识别不到双引号。

原因分析

查看实例的“sql_mode”参数值包含“ANSI_QUOTES”。

启用“ANSI_QUOTES”后，双引号会被视为标识符引号字符。更多内容，详见**sql_mode官方文档**。

图 5-1 查看参数值



解决方案

“sql_mode”参数值去掉“ANSI_QUOTES”。

5.2 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;
```


版权所有 © 华为云计算技术有限公司

原因分析

RDS for MySQL的字符集配置有误：

- emoji表情为特殊字符，需要4字节字符集存储。
- 该报错场景下MySQL字符集为utf-8，最多支持3个字节，需要修改为支持4个字节的字符集utf8mb4。

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 CONVERT TO CHARACTER 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。

5.3 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
```

版权所有 © 华为云计算技术有限公司

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。

5.4 建表时 timestamp 字段默认值无效

场景描述

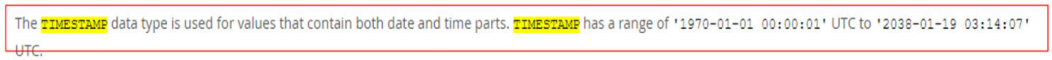
执行建表SQL语句失败，报错：ERROR 1067: Invalid default value for 'session_start'

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

原因分析

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。

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**。



使用如下命令查看时区：

版权所有 © 华为云计算技术有限公司

show variables like "%zone%";

由于使用的是UTC +8时区，所以timestamp字段默认值需要加8小时才是有效范围，即有效支持的范围是从1970-01-01 08:00:01开始。

```
mysql> show variables like "%zone%";
+------------------+--------+
| Variable_name    | Value  |
+------------------+--------+
| system_time_zone |        |
| time_zone        | +08:00 |
+------------------+--------+
2 rows in set, 1 warning (0.00 sec)
```

解决方案

修改timestamp字段的默认值：

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

5.5 自增属性 AUTO_INCREMENT 为什么未在表结构中显示

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)
```

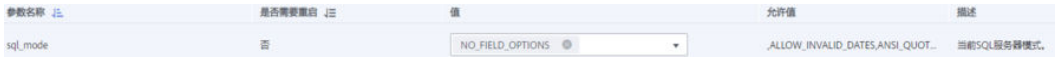
执行**show create table** *xxx*，未显示自增属性AUTO_INCREMENT：

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

| 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。



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。

5.6 存储过程和相关表字符集不一致导致执行缓慢

场景描述

RDS 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。

版权所有 © 华为云计算技术有限公司

5.7 RDS MySQL 报错 ERROR [1412]的解决方法

场景描述

连接RDS 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。

5.8 创建二级索引报错 Too many keys specified

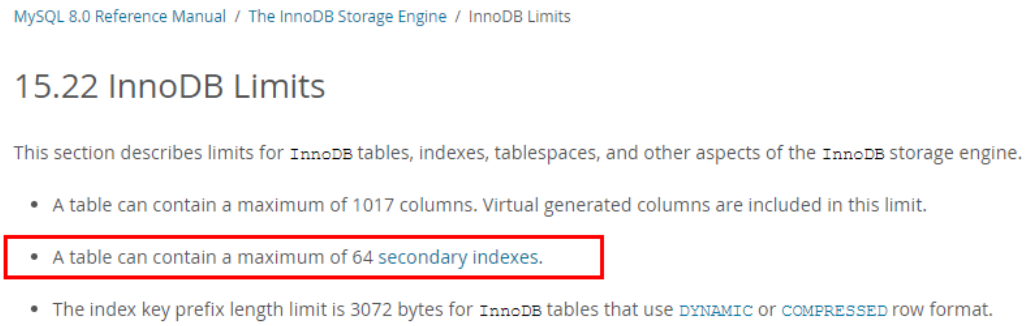
场景描述

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。

版权所有 © 华为云计算技术有限公司


MySQL 8.0 Reference Manual / The InnoDB Storage Engine / InnoDB Limits

15.22 InnoDB Limits

This section describes limits for InnoDB tables, indexes, tablespaces, and other aspects of the InnoDB storage engine.

- A table can contain a maximum of 1017 columns. Virtual generated columns are included in this limit.
- A table can contain a maximum of 64 secondary indexes.
- The index key prefix length limit is 3072 bytes for InnoDB tables that use DYNAMIC or COMPRESSED row format.

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。

5.9 存在外键的表删除问题

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;
```

5.10 distinct 与 group by 优化

场景描述

使用distinct或group by的语句执行比较慢。

版权所有 © 华为云计算技术有限公司

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
```


版权所有 © 华为云计算技术有限公司

```
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

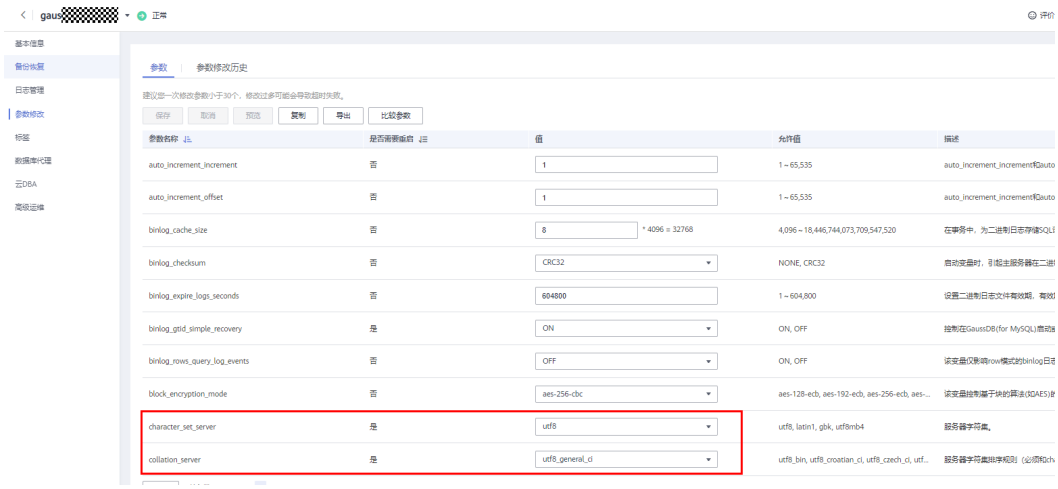
在使用distinct或group by的时候，尽量在合理的情况下，创建可以包含所有依赖字段的索引。

5.11 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。

版权所有 © 华为云计算技术有限公司



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。

版权所有 © 华为云计算技术有限公司

```
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)
```

5.12 MySQL 创建用户提示服务器错误

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
```

原因分析

1. 查询确认，发现消失的账号在mysql.user表中已经被删除，因此在控制台不再显示。
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。

版权所有 © 华为云计算技术有限公司

```
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。

```
mysql> CREATE USER 'test1'@'localhost' IDENTIFIED BY 'test1';
ERROR 1396 (HY000): Operation CREATE USER failed for 'test1'@'localhost'
mysql> FLUSH HOSTS;
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE USER 'test1'@'localhost' IDENTIFIED BY 'test1';
ERROR 1396 (HY000): Operation CREATE USER failed for 'test1'@'localhost'
mysql>  FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE USER 'test1'@'localhost' IDENTIFIED BY 'test1';
Query OK, 0 rows affected (0.01 sec)
```

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SQL审计日志，便于分析是哪个客户端删除了用户。

5.13 delete 大表数据后，再次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。

版权所有 © 华为云计算技术有限公司

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**。

5.14 设置事件定时器后未生效

场景描述

设置事件定时器后，未立即生效。

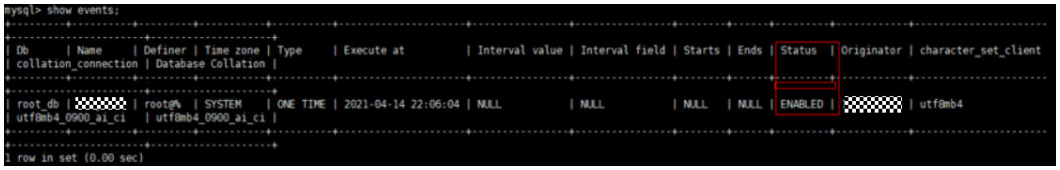
原因分析

1. 在实例“概览”页面，检查事件定时器开关是否打开。

版权所有 © 华为云计算技术有限公司

2. 检查定时事件的状态是否是**ENABLE**。

   show events;



3. 检查数据库设置的时间是北京时间还是UTC时间。

   show variables like "%time_zone%";

```
mysql> select now();
+---------------------+
| now()               |
+---------------------+
| 2021-04-15 01:30:20 |
+---------------------+
1 row in set (0.00 sec)
```

```
mysql> show variables like "%time_zone%";
+------------------+--------+
| Variable_name    | Value  |
+------------------+--------+
| system_time_zone | UTC    |
| time_zone        | SYSTEM |
+------------------+--------+
2 rows in set (0.00 sec)
```

   以上图为例，实例的时区为UTC，因此以北京时间设置事件定时器不会立即生效，等待时间到达对应UTC时间才会生效。

解决方案

事件定时器按照对应时区时间设置，即可立即生效。

5.15 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：

版权所有 © 华为云计算技术有限公司

```
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：

版权所有 © 华为云计算技术有限公司

```
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)
```

5.16 开通数据库代理后有大量 select 请求分发到主节点

原因分析：

1. 延时阈值参数

   只读实例同步主实例数据时允许的最长延迟时间。延时阈值仅在存在只读实例时生效。为避免只读实例读取的数据长时间和主实例不一致，当一个只读实例的延迟时间超过设置的延迟阈值，则不论该只读实例的读权重是多少，读请求都不会转发至该只读实例。

   更多信息，请参见**设置延时阈值和读写分离权重**。

2. 读权重参数

   设置主实例和只读实例的读权重分配，可以控制读请求的分发配比，仅在存在只读实例时生效。

   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**设置延时阈值和读写分离权重**。

3. 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。

版权所有 © 华为云计算技术有限公司

4. 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。

5. 自定义变量

   SQL中包含了自定义变量的语句会发到主节点。

6. 带锁的读操作（如SELECT for UPDATE）会被路由到主节点。

7. 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。

5.17 执行 RENAME USER 失败的解决方法

场景描述

用户执行RENAME USER语句，提示执行失败。

可能出现问题的版本：MySQL-5.6.41.5

故障分析

查询发现该user用户不存在，但实际在内存中存在。

版权所有 © 华为云计算技术有限公司

解决方案

执行以下操作解决：

```
drop user ‘xxx’@’%’;
flush privileges;
```

5.18 有外键的表无法删除报错 ERROR[1451]的解决方案

场景描述

数据库中的表，root用户也没有权限删除或修改。报错信息如下：

ERROR[1451] -Cannot deleteorupdatea parent row:

aforeignkeyconstraintfails (…)

故障分析

该表的frm文件在sys_tables里也存在，这个表跟其他表有外键关系，因此不能直接删除。

因为RDS for MySQL中设置了foreign key关联，造成无法更新或删除数据，可以通过设置“FOREIGN_KEY_CHECKS”变量来避免这种情况。

解决方案

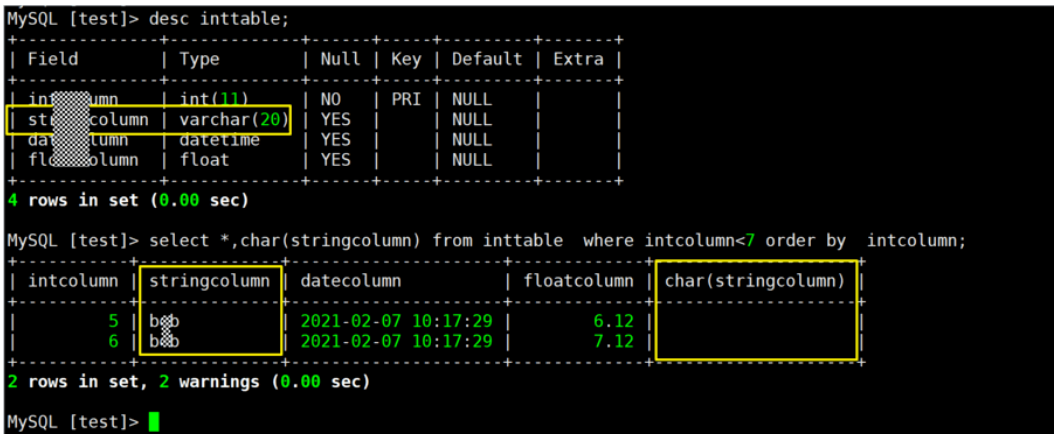
```
set session foreign_key_checks=off;
drop table table_name;
```

关闭foreign_key_checks，即可删除表。

5.19 表字段类型转换失败的解决方法

场景描述

varchar字段使用char类型读取，不能用如下所示的方式转换：



 版权所有 © 华为云计算技术有限公司

故障分析

char()函数不能用于转换数据类型。

解决方案

RDS for MySQL的CAST()和CONVERT()函数可用来获取一个类型的值，并产生另一个类型的值。两者具体的语法如下：

```
CAST(value as type);
CONVERT(value, type);
```

就是CAST(xxx AS 类型), CONVERT(xxx,类型)。

说明

可以转换的类型是有限制的。这个类型可以是以下值其中的一个：

- 二进制，同带binary前缀的效果 : BINARY；
- 字符型，可带参数 : CHAR()；
- 日期 : DATE；
- 时间: TIME；
- 日期时间型 : DATETIME；
- 浮点数 : DECIMAL；
- 整数 : SIGNED；
- 无符号整数 : UNSIGNED。

5.20 RDS for MySQL 创建表失败报错 Row size too large 的解决方案

场景描述

RDS for 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。

 版权所有 © 华为云计算技术有限公司

5.21 RDS for MySQL 数据库报错 ERROR [1412]的解决方案

场景描述

用户使用时，出现如下报错信息：

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

问题可能出现的版本：MySQL-5.7.31.2

故障分析

原因一：启动一致性快照事务引起。

场景1：

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

场景2：

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

场景3：

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

原因二：对binlog进行了DDL类操作。

```
# at 13793936
#210325 18:44:06 server id 1605802263  end_log_pos 13794065 CRC32 0x4df2db06    Query   thread_id=19314688      exec_time=1     error_code=0
use `zzkj`/*!*/;
SET TIMESTAMP=1616669046/*!*/;
SET @@session.sql_mode=1346371584/*!*/;
/*!\C utf8mb4 *//*!*/;
SET @@session.character_set_client=45,@@session.collation_connection=45,@@session.collation_server=33/*!*/;
ALTER TABLE bas_cabinet_box  ADD lock_status tinyint(4)
/*!*/;
```

解决方案

若经排查，是由上述原因引起的报错，请客户从业务侧解决问题。

5.22 外键使用不规范导致实例重启失败或执行表操作报错 ERROR 1146: Table 'xxx' doesn't exist

场景描述

- 重启RDS for MySQL实例后，在进行表相关操作时，提示如下错误：
  ERROR 1146: Table `xxx` doesn't exist

版权所有 © 华为云计算技术有限公司

同时，在error log中可见如下记录：

```
[Warning] InnoDB: Load table `xxx` failed, the table has missing foreign key indexes. Turn off 'foreign_key_checks' and try again.
[Warning] InnoDB: Cannot open table 'xxx' from the internal data dictionary of InnoDB through the .frm file for the table exists.
```

- 由于外键使用不规范，导致实例重启失败，error log中可见如下错误：

```
[Warning] InnoDB: Load table `xxx` failed, the table has missing foreign key indexes. Turn off 'foreign_key_checks' and try again.
[Warning] InnoDB: Cannot open table xxx/xxx from the internal data dictionary of InnoDB through the .frm file for the table exists.
```

故障分析

发生该问题的原因可能是用户添加的外键不满足相应条件和限制，具体外键使用规则参考**FOREIGN KEY Constraints**。

RDS for MySQL通过变量foreign_key_checks（默认值为ON）来控制外键限制检查，当foreign_key_checks设置为OFF时，外键限制检查不生效，此时用户添加或修改的外键可以不满足外键限制而不会报错，在实例重启时，foreign_key_checks默认开启，InnoDB打开表时会进行外键限制检查，此时会报错。

常见的情况有以下两种：

- 更改了父表和子表外键相关列的字符集

  MySQL 5.6、5.7、8.0允许在foreign_key_checks设置为OFF的情况下，修改父表和子表外键相关列的字符集。实例重启以后：

  - MySQL 5.6、5.7会在error log中提示warning，同时无法使用父表。
  - MySQL 8.0不会在error log中提示warning，可以使用父表。
- 删除了父表和子表外键相关列的索引
  - 对于MySQL 5.7、8.0：不允许在foreign_key_checks设置为OFF的情况下，删除父表和子表外键相关列的索引。
  - 对于MySQL 5.6：允许在foreign_key_checks设置为OFF的情况下，删除父表和子表外键相关列的索引。删除相关列索引后，重启实例会提示warning，同时被删除索引的表无法使用。

解决方案

- 对于修改字符集导致的问题，将foreign_key_checks设置为OFF，将父表和子表外键相关列的字符集修改一致。
- 对于删除索引导致的问题，将foreign_key_checks设置为OFF，重建索引。

5.23 RDS for MySQL 在分页查询时报错：Out of sort memory, consider increasing server sort buffer size

场景描述

RDS for MySQL在分页查询时报错：ERROR 1038 (HY001): Out of sort memory, consider increasing server sort buffer size

解决方案

将参数“sort_buffer_size”设置为高于其默认值256KB。

版权所有 © 华为云计算技术有限公司

“sort_buffer_size”是一个MySQL服务器系统变量，可能会影响您的查询性能，它是在每个会话级别上定义的，会影响MySQL内存消耗。

5.24 RDS for MySQL 创建用户报错：Operation CREATE USER failed

场景描述

执行以下SQL新建用户**saas_cash_user**失败：

```
/*COMMON SETTINGS*/ CREATE USER 'saas_cash_user'@'10.11.3.%' IDENTIFIED BY '******'
```

失败原因：(conn=288831) Operation CREATE USER failed for 'saas_cash_user'@'10.11.3.%'

原因分析

如果用户旧账号是通过**delete**删除，再次创建用户会报错。

创建用户时，一般使用**create user**或者**grant**语句来创建，**create**语法创建的用户没有任何权限，需要再使用**grant**语法来分配权限，而**grant**语法创建的用户直接拥有所分配的权限。

使用**drop user**方法删除用户的时候，会连通db表和权限表一起清除。而使用**delete from mysql.user**只会删除user表里的记录，如果用**show grants for username**来查看，会发现这个用户的相关权限依然有残留，这时候再新建一样的用户，就会触发校验导致创建失败。

解决方案

建议删除用户使用**drop**命令。

5.25 RDS for MySQL 使用 grant 授权 all privileges 报语法错误

场景描述

通过**grant**授权，库名是英文可以正常授权，如果库名是数字就会报错：You have an error in your SOL syntax

库名是英文：

```
grant all PRIVILEGES  on aaaaa.*  to  'TA01'@'%';
```

库名是数字：

```
grant all PRIVILEGES  on 11111.*  to  'TA01'@'%';
```

原因分析

这个问题是语法上的错误，在MySQL中，为了区分MySQL的关键字与普通字符，引入一个反引号。英文键盘输入环境下，输入反引号（ ` ），SQL语法正常执行。

版权所有 © 华为云计算技术有限公司

解决方案

库名是数字时，数字前后加上反引号（ ` ）：

```
grant all PRIVILEGES  on `11111`.*  to  'TA01'@'%';
```

图 5-2 执行成功

5.26 RDS for MySQL 5.6 版本实例创建表报错

场景描述

RDS for MySQL 5.6实例执行建表语句报错：

```
Index column size too large. The maximum column size is 767 bytes.
```

原因分析

RDS for MySQL 5.6实例建表时索引使用建议如下：

- 如果是单字段索引，则字段长度不应超过767字节。
- 如果是联合索引，则每个字段长度都不应超过767字节，且所有字段长度合计不应超过3072字节。
- 建表时使用utf8mb4字符集，这是一个4字节字符集。

当索引最大限制是767字节时，那么一个varchar字段长度：767/4=191.75。如果建表语句中varchar字段超出了这个长度，就会出现767字节长度报错。

解决方案

1. 查看“innodb_large_prefix”参数，确保值为“ON”。
2. 参考**原因分析**中的使用建议建表。

版权所有 © 华为云计算技术有限公司

5.27 无主键表添加自增主键后导致主备节点查询数据不一致

场景描述

在主备节点上使用添加的自增主键值进行查询，查询结果中出现数据不一致问题。

原因分析

对于无主键表，表中数据的顺序由存储引擎的RowID决定，无主键表中的数据在主备节点上的RowID可能不同，因此表中的数据在主备节点上的顺序不同。当对主备节点中数据顺序不同的表添加自增主键时，自增主键的值是按照表中数据的顺序初始化的，从而导致同样的数据对应的自增主键值不同，即用相同的自增主键值分别在主备节点上查询的数据不同。详情请见**MySQL官方文档**。

解决方案

为已有数据的表添加自增列时，请先创建相同表结构的新表，再在新表上添加自增列，将原表数据导入（导入数据时，请尽量保持原表无写入操作，否则会造成原表与新表数据不一致）。

按照如下步骤解决主备节点查询数据不一致问题。

步骤1 在主节点上创建一个与无主键表（称之为原无主键表t1）相同的新表t2，并为新表中添加自增主键。

示例如下：

```
CREATE TABLE t2 LIKE t1;
ALTER TABLE t2 ADD id INT AUTO_INCREMENT PRIMARY KEY;
```

步骤2 将原无主键表的数据全部插入到新表t2中。

示例如下：

```
INSERT INTO t2(col1, col2) SELECT col1, col2 FROM t1 ORDER BY col1, col2;
```

说明

为了确保主备节点对应表中数据的顺序相同，ORDER BY子句必须包含原无主键表的所有列。

步骤3 删除原无主键表t1，并将新表重命名为原无主键表名。

示例如下：

```
DROP TABLE t1;
RENAME TABLE t2 TO t1;
```

----结束

5.28 RDS for MySQL 插入数据提示 Data too long for column

场景描述

RDS for MySQL通过JDBC插入数据提示：

版权所有 © 华为云计算技术有限公司

```
Data truncation: Data too long for column '字段名
```

原因分析

检查业务上是否在session级别设置“sql_mode”参数为“STRICT_TRANS_TABLES”。

“STRICT_TRANS_TABLES”为严格模式，设置为严格模式时会报字段超长。

解决方案

- 加大字段长度，建议在业务低峰期执行。
  ALTER TABLE *表名* **MODIFY COLUMN** *字段名* **VARCHAR(128);**
- 通过DAS可以正常插入数据，如果有超长数据，DAS会做对应的截断。

版权所有 © 华为云计算技术有限公司

6 连接类

6.1 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

 版权所有 © 华为云计算技术有限公司

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 'zdh1234' 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 'test'@'xxxxx' REQUIRE NONE;**

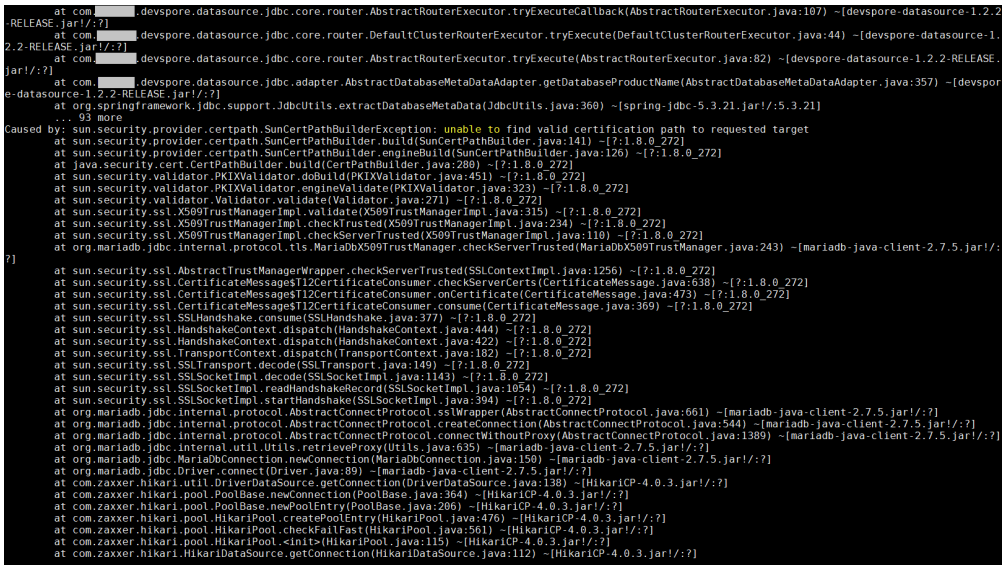
6.2 mariadb-connector SSL 方式连接数据库失败

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
```


版权所有 © 华为云计算技术有限公司



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-java-client-2.7.5的连接串应该为：

- 不提供CA证书，不对服务端证书校验：
  ```
  String url = "jdbc:mysql://ip:port/mysql?useSsl=true&trustServerCertificate=true";
  ```
- 提供CA证书，对服务端证书进行校验：
  ```
  String url = "jdbc:mysql://ip:port/mysql?useSsl=true&serverSslCert=D:\
  \ca.pem&disableSslHostnameVerification=true";
  ```

注意：RDS 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**。

6.3 RDS for MySQL 建立连接慢导致客户端超时报 connection established slowly

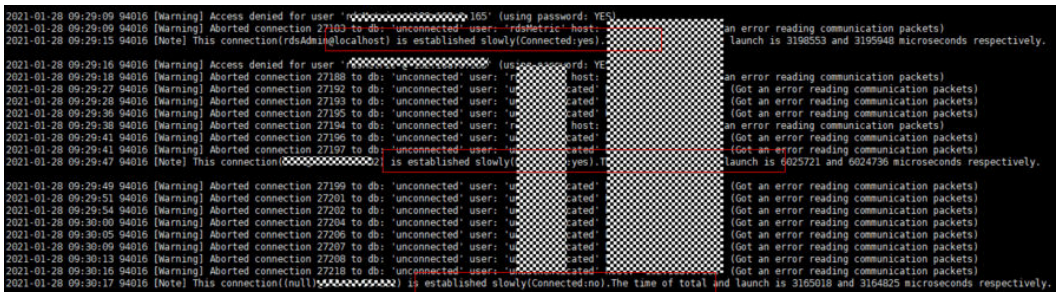
场景描述

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

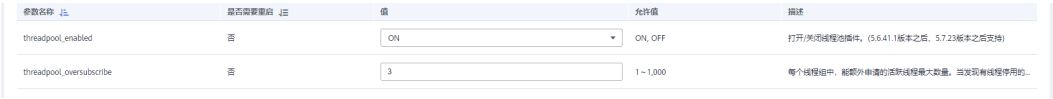
1. **查看RDS MySQL的错误日志**，观察是否有如下信息：connection xxx is established slowly。示例：

版权所有 © 华为云计算技术有限公司



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。



可以看出，threadpool_oversubscribe为3，线程池处理连接等待与该参数相关。

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。

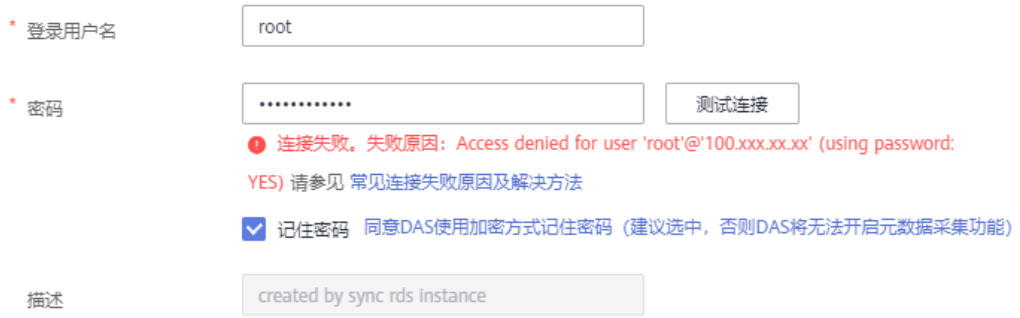
减少线程重复创建与销毁部分的开销，提高性能，同时它也限制了MySQL的runing线程数，关键时刻可以保护系统，防止雪崩。

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_oversubscribe扩大线程总数，或者直接关闭线程池。

6.4 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';
```

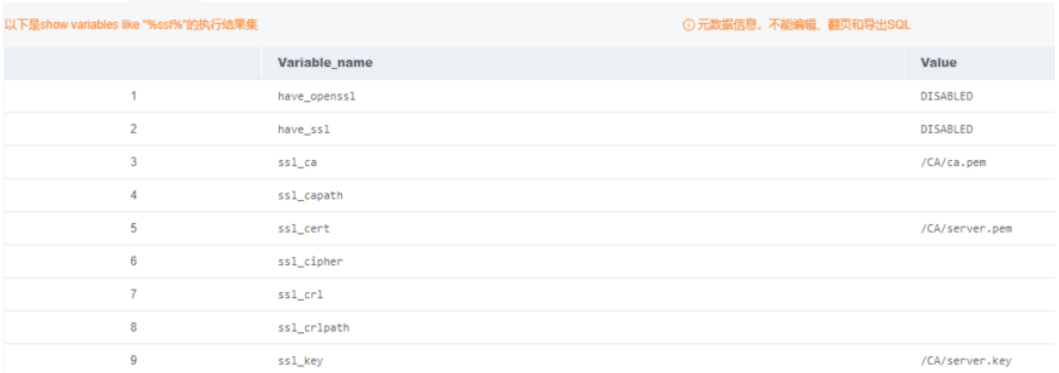

版权所有 © 华为云计算技术有限公司

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：



以下是show variables like "%ssl%"的执行结果集　　元数据信息，不能编辑、翻页和导出SQL

	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 'rds%';

6.5 通过 DAS 登录实例报错 Client does not support authentication protocol requested by server

场景描述

在控制台以root账号通过DAS登录RDS for MySQL实例报错：Client does not support authentication protocol requested by server. plugin type was = 'sha256_password'

原因分析

DAS暂不支持密码的加密方式为sha256_password的数据库用户登录。

解决方案

执行如下语句将密码的加密方式改为mysql_native_password。

```
alter user 'user_name'@'%' identified with mysql_native_password by 'password';
```


版权所有 © 华为云计算技术有限公司

6.6 通过 DAS 执行 SQL 报错 ALTER USER failed

场景描述

RDS for MySQL实例在DAS上执行ALTER USER语句报错：ALTER USER failed for 'user'@'***'

解决方案

“账号名@ip”表示数据库的一个用户，以账号名是user为例：

- 如果主机IP是“%”，则SQL中需要填写为 'user'@'%'；
- 如果主机IP是具体的IP，则SQL中需要填写为 'user'@'IP地址'

如何查看主机IP，请参见**修改主机IP**。

6.7 主实例上执行 show slave status 的结果为什么不为空

场景描述

对于RDS for MySQL主备实例，在主实例上执行**show slave status**的结果不为空。

解决方案

该场景是正常情况，产品这样设计是为了主备倒换时可以减少配置复制关系的时间，加快主备倒换的时间。

6.8 通过 DAS 授权或取消授权时报错 Your password does not satisfy the current policy requirements

场景描述

在DAS上使用root登录数据库进行**GRANT**授权或者**REVOKE**取消授权时，报错：Your password does not satisfy the current policy requirements

解决方案

- MySQL 5.7

  在没有创建用户的前提下授权，会提示：Your password does not satisfy the current policy requirements

  5.7版本的**GRANT**语句会在用户不存在的时候自动创建用户，建议在授权命令后面加上**IDENTIFIED BY**，此时用户就被创建成功。

  查询用户是否存在：

  select * from mysql.user where user='*<username>***';**

  授权语句：

  grant select on mysql.* to '*<username>***'@'%' IDENTIFIED BY '***<password>***';**

版权所有 © 华为云计算技术有限公司

- MySQL 8.0

  在没有创建用户的前提下授权，会提示：You are not allowed to create a user with GRANT

  8.0版本需要先**create**创建用户再执行授权。

6.9 客户端 TLS 版本与 RDS for 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RDS 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RDS 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RDS for MySQL和自建MySQL的TLS版本。

   show variables like '%tls_version%';

   发现RDS 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RDS 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.

6.10 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。

版权所有 © 华为云计算技术有限公司

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. 使用其他账号登录数据库，查看root权限，发现有两个root账号，其中一个root限制Host的IP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

联系技术支持协助删除多余的root账号。

6.11 RDS for MySQL 客户端连接实例后会自动断开

故障描述

RDS for 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RDS for MySQL常见错误，一般为配置错误导致。

版权所有 © 华为云计算技术有限公司

- “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RDS for 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，RDS for 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

 版权所有 © 华为云计算技术有限公司

6.12 RDS for MySQL 实例无法访问

故障描述

客户端无法连接数据库，连接数据库时返回如下报错信息：

- 故障一
  ERROR 1045 (28000): Access denied for user ‘root'@‘192.168.0.30' (using password:YES)
- 故障二
  ERROR 1226 (42000):User‘test' has exceeded the‘max_user_connections' resource (current value:10)
- 故障三
  ERROR 1129 (HY000): Host ‘192.168.0.111' is blocked because of many connection errors; unblock with 'mysqladmin flush-hosts'

故障一

步骤1 排查密码root账号的密码是否正确。

一般情况下，ERROR 1045报错为密码错误引起的，因此需要首要排除是否密码错误问题。

```
select password(‘Test1i@123');
select host,user,Password from mysql.user where user=‘test1';
```

```
mysql> select host,user,Password from mysql.user where user='testl';
+--------------+-------+-------------------------------------------+
| host         | user  | Password                                  |
+--------------+-------+-------------------------------------------+
| 192.168.0.74 | test1 | *094C0901889F4D899028A83C061EF44272590E22 |
+--------------+-------+-------------------------------------------+
1 row in set (0.00 sec)

mysql> select password('testli@123');
+-------------------------------------------+
| password('testli@123')                    |
+-------------------------------------------+
| *094C0901889F4D899028A83C061EF44272590E22 |
+-------------------------------------------+
1 row in set (0.01 sec)
```

使用错误的密码登录就会失败。

```
[root@ecs-lwt-0921 ~]# mysql -utest1 -ptestli@321  -P 8635 -h192.168.0.73
Warning: Using a password on the command line interface can be insecure.
ERROR 1045 (28000): Access denied for user 'test1'@'192.168.0.74' (using password: YES)
```

步骤2 确认该主机是否有连接数据库实例的权限。

```
select user, host from mysql.user where user=‘username';
```

 版权所有 © 华为云计算技术有限公司

```
mysql> select user,host from mysql.user where user='test1';
+-------+--------------+
| user  | host         |
+-------+--------------+
| test1 | 192.168.0.74 |
+-------+--------------+
1 row in set (0.00 sec)
```

如果该数据库用户需要从其他主机登录，则需要使用root用户连接数据库，并给该用户授权。

以加入主机IP为192.168.0.76举例：

GRANT all privileges ON test.* TO *'test1'@'192.168.0.76'* **identified by** *'Test1i@123'*;

flush privileges;

```
[root@ecs-lwt-0921 ~]# mysql -uroot -ptest1i@123  -P 8635 -h192.168.0.73
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 6961
Server version: 5.6.35-log MySQL Community Server (GPL)

Copyright (c) 2000, 2016, 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> GRANT all privileges ON test.* TO 'test1'@'192.168.0.76' identified by 'test1i@123';
Query OK, 0 rows affected (0.01 sec)

mysql>  flush privileges;
Query OK, 0 rows affected (0.03 sec)

mysql>
```

步骤3 确认RDS for MySQL客户端和实例VIP的连通性。

尝试进行ping连接性能，若可以ping通，排除telnet数据库端口的问题。

步骤4 查看实例安全组，排查是否因安全策略问题引起的报错。

步骤5 查询user表信息，确认用户信息。

```
mysql> select user,host ,max_connections,max_user_connections,password_expired from mysql.user order by user;
+-----------+-----------+-----------------+---------------------+------------------+
| user      | host      | max_connections | max_user_connections | password_expired |
+-----------+-----------+-----------------+---------------------+------------------+
| mysql.sys | localhost |               0 |                   0 | N                |
| rdsAdmin  | localhost |          100000 |              100000 | N                |
| rdsBackup | localhost |          100000 |              100000 | N                |
| rdsMetric | localhost |          100000 |              100000 | N                |
| rdsRepl   | 172.16.%  |          100000 |              100000 | N                |
| root      | %         |               0 |                   0 | N                |
| root      | 192.168.% |               0 |                   0 | N                |
+-----------+-----------+-----------------+---------------------+------------------+
7 rows in set (0.00 sec)
```

在排查中发现存在两个root用户。

如果用户的客户端处于192.168的网段，RDS for MySQL数据库的是对root@'192.168.%'这个用户进行认证的。而用户登录时使用的为root@'%'这个账号所对应的密码，因而导致连接失败，无法正常访问。此次问题是因密码错误引起的访问失败。

 版权所有 © 华为云计算技术有限公司

说明

在此案例中，root@'%'为console创建实例时设置密码的账号。

----结束

故障二

步骤1 排查是否在创建RDS for MySQL用户时，添加了max_user_connections选项，导致限制了连接数。

```
select user,host ,max_user_connections from mysql.user where user='test';
```

```
mysql> select user,host,max_user_connections from mysql.user where user='test';
+------+---------------+----------------------+
| user | host          | max_user_connections |
+------+---------------+----------------------+
| test | 192.168.0.100 |                   10 |
+------+---------------+----------------------+
1 row in set (0.00 sec)
```

经排查发现由于设置了max_user_connections选项，导致连接失败。

步骤2 增加该用户最大连接数。

```
alter user test@'192.168.0.100' with max_user_connections 15。
```

步骤3 查询变更结果，检查是否可正常访问数据库。

```
mysql> alter user  test@'192.168.0.100' with max_user_connections 15;
Query OK, 0 rows affected (0.01 sec)

mysql> select user,host,max_user_connections from mysql.user where user='test';
+------+---------------+----------------------+
| user | host          | max_user_connections |
+------+---------------+----------------------+
| test | 192.168.0.100 |                   15 |
+------+---------------+----------------------+
1 row in set (0.00 sec)
```

----结束

故障三

步骤1 排查是否由于RDS for MySQL客户端连接数据库的失败次数(不包括密码错误)，超过了max_connection_errors的值。

步骤2 在数据库端解除超出限制问题。使用root用户登录mysql，执行flush hosts。

或者执行如下命令。

mysqladmin flush-hosts –u*<user>* **–p***<password>* **-h***<ip>* **-P***<port >*。

步骤3 再次连接，检查是否可正常访问数据库。

----结束

 版权所有 © 华为云计算技术有限公司

6.13 RDS for MySQL 数据库修改 authentication_string 字段为显示密码后无法登录

场景描述

客户通过navicat修改RDS for MySQL的user表root账号的“authentication_string”字段，修改为为显示密码后无法登录客户端。

问题可能出现的版本：MySQL-8.0.20.6

原因分析

修改密码方式错误，不应直接改user表的authentication_string字段的hash key，而是要通过console重置root密码方式修改。

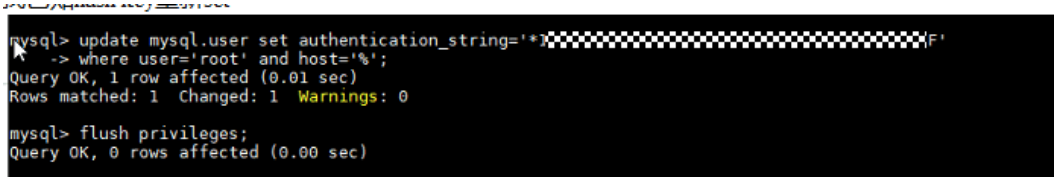
解决方案

由于8.0版本不支持password函数，因此需要通过以下步骤恢复：

步骤1 找出rdsAdmin账号的authentication_string字段，使用以下命令更新：

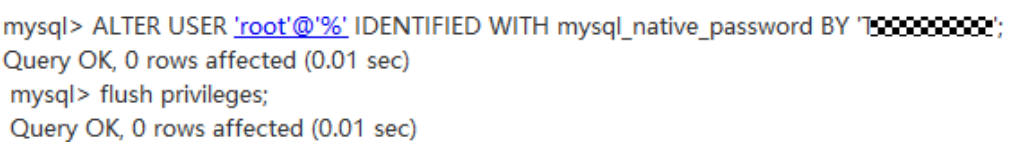
```
update mysql.user set authentication_string='XXX'
```

其中*XXX*为新修改的密码。



步骤2 重置root账号密码。

```
ALTER USER 'root'@'%' IDENTIFIED WITH mysql_native_password BY 'XXX';
flush privileges;
```



修改后用户就可以正常登录root账号。

----结束

6.14 RDS for MySQL 升级版本后，导致现有配置无法正常连接到 MySQL-server

场景描述

用户新建实例，用代码连接该数据库时出现报错：

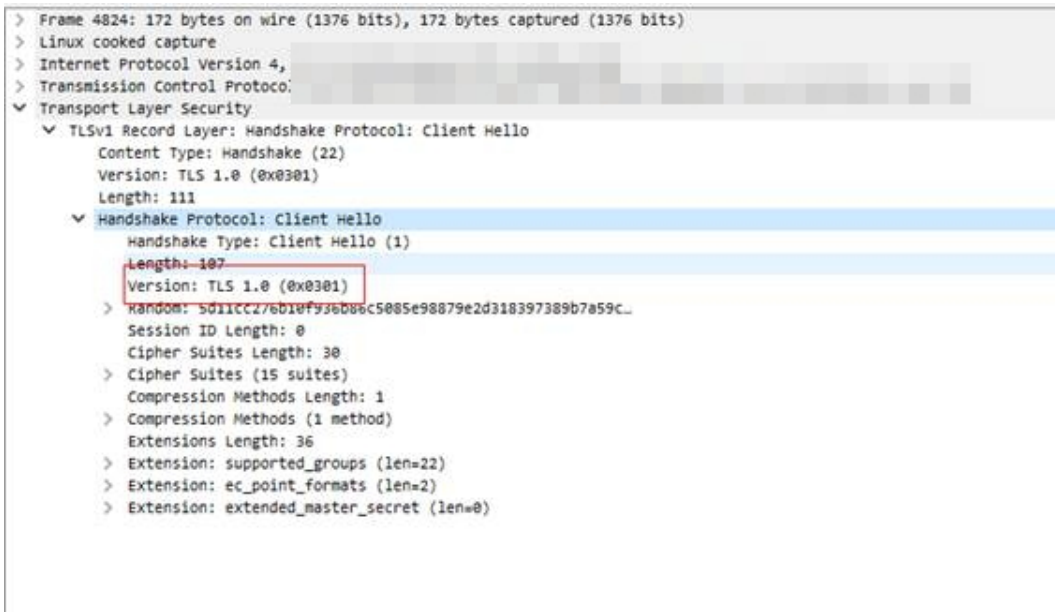
Caused by: javax.net.ssl.SSLException: Received fatal alert: protocol_version

版权所有 © 华为云计算技术有限公司

RDS for MySQL原有版本为5.7.23，升级到5.7.25版本后，导致现有配置无法正常连接到MySQL-server，抓包结果如下图6-1：

可以看出，客户端进行TLS握手时向服务端发送的TLS版本号是1.0，并提供了15个支持的密码套件。

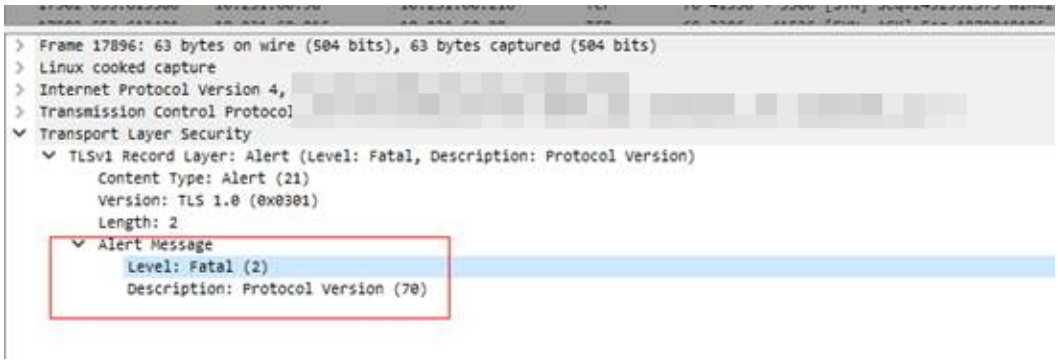
图 6-1 连接失败抓包结果



故障分析

从MySQL-server的回复中如图6-2可以看到，服务器拒绝了客户端的链接，原因是MySQL 5.7.25升级了openssl版本（1.1.1a），导致拒绝了不安全的TLS版本和密码套件。

图 6-2 MySQL-server 的回复

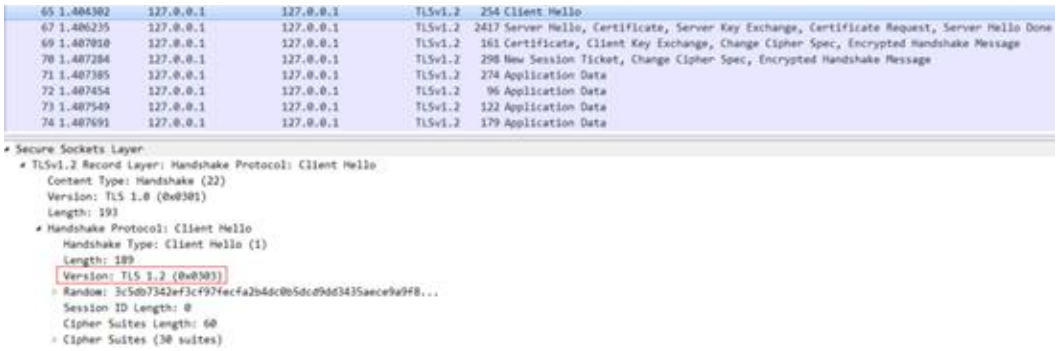


解决方案

升级您的JDK客户端到**JDK 8或以上版本**，则默认支持的TLS为1.2版本，如图6-3，可以正常连接的客户端支持TLS1.2，并支持30个密码套件。

 版权所有 © 华为云计算技术有限公司

图 6-3 正常连接抓包结果



6.15 客户端超时参数设置不当导致连接超时退出

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

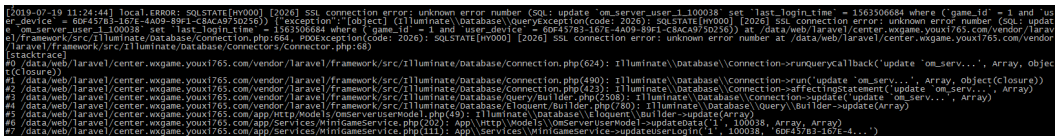
- 将socketTimeout、connectTimeout等参数的默认值调整为合适的值。
- 在程序中注意处理断线重连的功能。
- **推荐直接使用连接池**。

6.16 RDS for MySQL 在启用了 SSL 验证连接功能后，导致代码（php/java/python）等连接数据库失败

场景描述

用户开启了SSL验证连接功能，使用代码执行SQL连接数据库出现如下报错：

图 6-4 连接失败



故障分析

该场景下用户开启了SSL连接功能，需要使用SSL连接方式。请检查连接命令是否使用了SSL方式。

版权所有 © 华为云计算技术有限公司

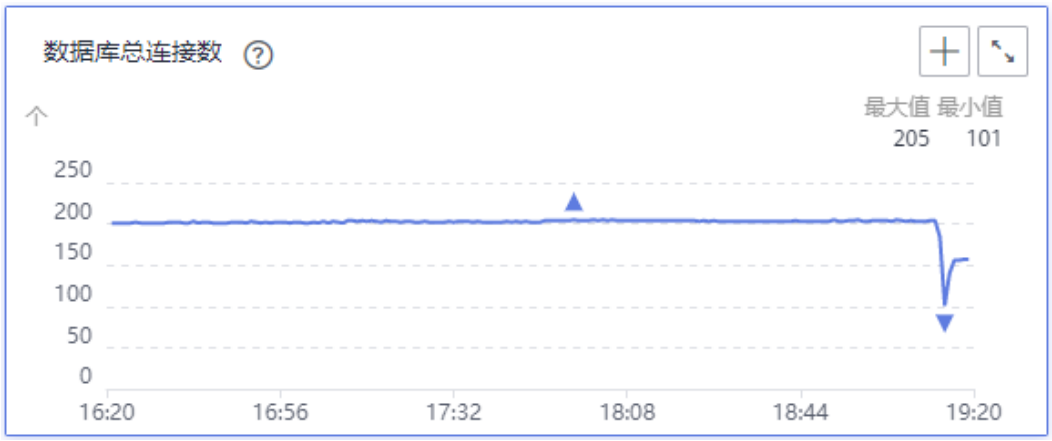
解决方案

- 开启SSL验证连接功能，使用SSL连接方式连接数据库，请参考**SSL连接方式**。
- 关闭SSL验证连接功能，使用非SSL连接方式连接数据库，请参考**非SSL连接方式**。

6.17 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。

```
2021-11-22T10:34:23.248977Z warn    istio.io/istio/security/pkg/k8s/controller/workloadsecret.go:236: watch of *v1.Secret
ended with: too old resource version: 228865253 (228865325)
2021-11-22T11:20:50.632458Z info    rootCertRotator Check and rotate root cert.
2021-11-22T11:20:50.639274Z info    rootCertRotator Root cert is not about to expire, skipping root cert rotation.
2021-11-22T12:10:55.338195Z warn    istio.io/istio/security/pkg/k8s/controller/workloadsecret.go:236: watch of *v1.Secret
ended with: too old resource version: 228884272 (228885539)
2021-11-22T12:20:50.632470Z info    rootCertRotator Check and rotate root cert.
2021-11-22T12:20:50.635853Z info    rootCertRotator Root cert is not about to expire, skipping root cert rotation.
2021-11-22T13:12:05.395613Z warn    istio.io/istio/security/pkg/k8s/controller/workloadsecret.go:236: watch of *v1.Secret
```

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。

版权所有 © 华为云计算技术有限公司

6.18 数据库版本升级后 Navicat 客户端登录实例报错 1251

场景描述

数据库版本升级后普通用户通过Navicat客户端登录实例报错：1251 - Client does not support authentication protocol requested by server;consider upgrading MySQL client

原因分析

检查用户名的身份认证插件发现是“caching_sha2_password”，而数据库代理不支持RDS for MySQL 8.0的“caching_sha2_password”身份认证插件，导致登录报错。

解决方案

- 更新Navicat驱动来解决问题。
- 将RDS for MySQL 8.0实例该用户登录的加密规则修改为“mysql_native_password”。

  通过DAS执行**select plugin from mysql.user where user="***用户名***";**修改身份认证插件解决问题。

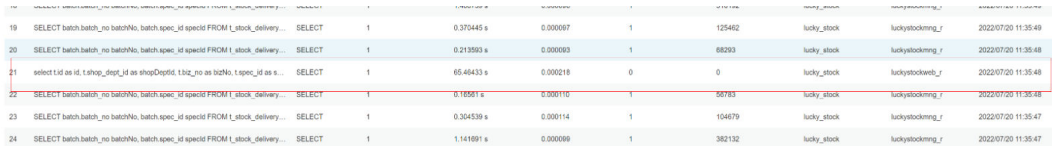
 版权所有 © 华为云计算技术有限公司

7 其他使用问题

7.1 慢日志显示 SQL 语句扫描行数为 0

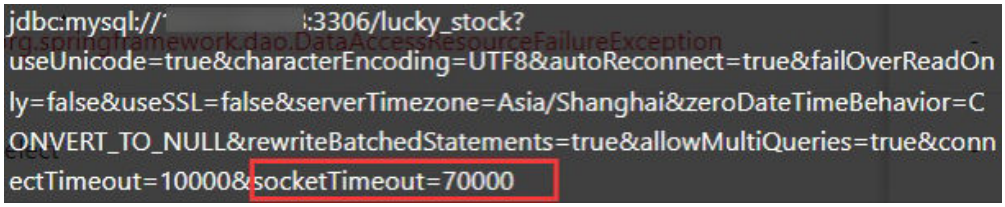
场景描述

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：



解决方案

优化SQL或者将socketTimeout设置合理值。

7.2 SQL 诊断结果中记录的行数远小于慢日志中的扫描行数

场景描述

RDS for MySQL实例在DAS界面执行SQL诊断时，SQL语句诊断结果中执行计划记录的行数远小于慢SQL日志中的扫描行数。

 版权所有 © 华为云计算技术有限公司

原因分析

在查询优化器决定使用全表扫描的方式对某个表执行查询的时候，执行计划的rows列就代表预估需要读取的记录行数。这个执行计划的行数不是扫描的行数，执行的时候可能会反复扫描表。

7.3 RDS for MySQL 慢日志里面有毫秒级别的 SQL

场景描述

RDS for MySQL慢日志阈值“long_query_time”为1秒，但是在慢日志明细中，出现平均执行时间低于1秒阈值的慢SQL。

原因分析

查看参数“log_queries_not_using_indexes”值为“ON”，即相关的SQL语句没有使用索引。如果运行的SQL语句没有使用索引，则数据库同样会将这条SQL语句记录到慢查询日志文件。

7.4 查看 RDS 存储空间使用量

场景描述

云数据库RDS实例的存储空间为客户购买的数据盘存储，不包括客户后台弹性云服务器的系统盘。

云监控服务（Cloud Eye）目前可以对客户存储空间的大小、使用量、利用率等做出监控及设置告警策略。

说明

主备实例的存储空间大小是指主实例存储空间。

解决方案

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。

步骤3 单击页面左上角的，选择“数据库 > 云数据库 RDS”，进入RDS信息页面。

步骤4 在“实例管理”页面，选择指定的实例，单击实例名称。

步骤5 进入“概览”页面。在“存储与备份”模块查看存储空间使用情况。

版权所有 © 华为云计算技术有限公司

图 7-1 存储空间

----结束

7.5 错误日志报错 The table is full

场景描述

RDS for MySQL错误日志报错：ERROR [MY-013132] [Server] The table '*表名*' is full!

原因分析

使用TempTable引擎可能会报该错误，详见**官方文档**。

解决方案

在RDS界面修改参数“internal_tmp_mem_storage_engine”的值为“MEMORY”。

7.6 审计日志上传策略说明

场景描述

RDS for MySQL控制台审计日志是半小时或100MB上传一次并生成文件，但是存在用户审计日志中有出现不满100MB，两分钟上传并生成一个文件的情况：

文件名	文件大小	更新时间
09d56d1e6580f3a51f25c0002a83504f_2da288ca20f84336a7bf4a1a8350de6fin01/39779385_202...	1.00 KB	2022/06/15 15:51:27
09d56d1e6580f3a51f25c0002a83504f_2da288ca20f84336a7bf4a1a8350de6fin01/39779385_202...	10.00 KB	2022/06/15 15:43:10
09d56d1e6580f3a51f25c0002a83504f_2da288ca20f84336a7bf4a1a8350de6fin01/39779385_202...	8.00 KB	2022/06/15 11:26:46
09d56d1e6580f3a51f25c0002a83504f_2da288ca20f84336a7bf4a1a8350de6fin01/39779385_202...	9.00 KB	2022/06/15 10:11:53
09d56d1e6580f3a51f25c0002a83504f_2da288ca20f84336a7bf4a1a8350de6fin01/39779386_202...	0.00 KB	2022/06/14 16:26:37
09d56d1e6580f3a51f25c0002a83504f_2da288ca20f84336a7bf4a1a8350de6fin01/39779386_202...	0.00 KB	2022/06/14 16:24:57
09d56d1e6580f3a51f25c0002a83504f_2da288ca20f84336a7bf4a1a8350de6fin01/39779386_202...	0.00 KB	2022/06/14 15:57:24
09d56d1e6580f3a51f25c0002a83504f_2da288ca20f84336a7bf4a1a8350de6fin01/39779386_202...	0.00 KB	2022/06/14 13:07:11
09d56d1e6580f3a51f25c0002a83504f_2da288ca20f84336a7bf4a1a8350de6fin01/39779386_202...	6.00 KB	2022/06/14 08:40:21
09d56d1e6580f3a51f25c0002a83504f_2da288ca20f84336a7bf4a1a8350de6fin01/39779387_202...	0.00 KB	2022/06/14 02:17:07

原因分析

审计日志轮转策略：半小时或累积到100MB上传存入日志。

版权所有 © 华为云计算技术有限公司

1. 半小时是审计日志文件轮转周期，控制台上“更新时间”是审计日志文件中最后的SQL写入时间，便于用户下载后分析。
2. 在某些特殊情况下，如备机重建、主备切换等操作过程中，会引起审计日志强制轮转进而导致半小时以内且不到100MB审计日志文件产生，该场景属于正常现象。

7.7 自增字段取值

RDS for MySQL对自增字段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为自增。

```
mysql> INSERT INTO animals (name) VALUES ('fish'),('cat'),('penguin'),('lax'),('whale'),('ostrich');
Query OK, 6 rows affected (0.01 sec)
Records: 6  Duplicates: 0  Warnings: 0
mysql> select * from animals;
+----+---------+
| id | name    |
+----+---------+
|  1 | fish    |
|  2 | cat     |
|  3 | penguin |
|  4 | lax     |
|  5 | whale   |
|  6 | ostrich |
+----+---------+
6 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=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为自增。

```
mysql> INSERT INTO animals (id,name) VALUES(0,'groundhog');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO animals (id,name) VALUES(NULL,'squirrel');
Query OK, 1 row affected (0.01 sec)
mysql> select * from animals;
+----+-----------+
| id | name      |
+----+-----------+
|  1 | fish      |
|  2 | cat       |
|  3 | penguin   |
|  4 | lax       |
|  5 | whale     |
|  6 | ostrich   |
|  7 | groundhog |
|  8 | squirrel  |
+----+-----------+8
rows in set (0.00 sec)
mysql> show create table animals;
+---------+-----------------------------------------------------------------+
```


版权所有 © 华为云计算技术有限公司

```
| 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*X*，数据库会将*X*填入字段并修改AUTO_INCREMENT=*X*+1。

```
mysql> INSERT INTO animals (id,name) VALUES(100,'rabbit');
Query OK, 1 row affected (0.00 sec)
mysql> select * from animals;
+-----+-----------+
| id  | name      |
+-----+-----------+
|   1 | fish      |
|   2 | cat       |
|   3 | penguin   |
|   4 | lax       |
|   5 | whale     |
|   6 | ostrich   |
|   7 | groundhog |
|   8 | squirrel  |
| 100 | rabbit    |
+-----+-----------+
9 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 |
+---------+------------------------------------------------------------------------------+
```

4. 直接使用小于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 | fish      |
|   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. 直接使用负值。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      |
```

 版权所有 © 华为云计算技术有限公司

```
+-----+-----------+
| -50 | -middle   |
|   1 | fish      |
|   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)
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         |
+---------+-----------------------------------------------------------------------------------------------------+
```

7.8 表的自增 AUTO_INCREMENT 初值与步长

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为auto_increment_increment。
- 当 auto_increment_offset <= auto_increment_increment 时，自增值计算方式：值 = auto_increment_offset + N*auto_increment_increment（ N为插入的数据条数 ）

在RDS for MySQL中“auto_increment_increment”和“auto_increment_offset”参数默认都为1，如需修改请在控制台修改，具体操作请参见**修改RDS for MySQL实例参数**。

举例：

- auto_increment_offset=1，auto_increment_increment=1，那么初值为1，步长为1。

```
mysql> show variables like 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 1     |
| auto_increment_offset    | 1     |
+--------------------------+-------+
mysql> create table auto_test1(id int NOT NULL AUTO_INCREMENT, PRIMARY KEY (`id`));
Query OK, 0 rows affected (0.09 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 DEFAULT
CHARSET=utf8 |
+------------+-----------------------------------------------------------------------------+
mysql> insert into auto_test1 values(0), (0), (0);
```

 版权所有 © 华为云计算技术有限公司

```
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> select * from auto_test1;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
+----+
3 rows in set (0.01 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=4
DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

- 修改auto_increment_increment=2，步长变为2。

```
mysql> set session auto_increment_offset=2;
Query OK, 0 rows affected (0.02 sec)
mysql> show variables like 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 2     |
| auto_increment_offset    | 1     |
+--------------------------+-------+
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 |
|  6 |
|  8 |
+----+
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      |
+------------+-----------------------------------------------+
1 row in set (0.01 sec)
```

- auto_increment_offset=10，auto_increment_increment=2，初值为2（因为auto_increment_offset > auto_increment_increment），步长为2。

```
mysql> set session auto_increment_offset=10;
mysql> set session auto_increment_increment=2;
mysql> show variables like 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 2     |
| auto_increment_offset    | 10    |
+--------------------------+-------+
mysql> create table auto_test2(id int NOT NULL AUTO_INCREMENT, PRIMARY KEY (`id`)); Query OK,
0 rows affected (0.08 sec)
mysql> show create table auto_test2;
```

 版权所有 © 华为云计算技术有限公司

```
+------------
+-----------------------------------------------------------------------------------------------------------------------
------+
| Table      | 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)
mysql> insert into auto_test2 values(0), (0), (0);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> select * from auto_test2;
+----+
| id |
+----+
|  2 |
|  4 |
|  6 |
+----+
3 rows in set (0.01 sec)
mysql> show create table auto_test2;
+------------+-----------------------------------------------------------------------------------------------------+
| Table      | Create Table                                                                                        |
+------------+-----------------------------------------------------------------------------------------------------+
| auto_test2 | CREATE TABLE `auto_test2` (
`id` int NOT NULL AUTO_INCREMENT,   PRIMARY KEY (`id`) ) ENGINE=InnoDB AUTO_INCREMENT=8
DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------------------------------------------------------------------+
```

- auto_increment_offset=5，auto_increment_increment=10，初值为5，步长为10。

```
mysql> set session auto_increment_offset=5; mysql> set session auto_increment_increment=10;
mysql>  show variables like 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 10    |
| auto_increment_offset    | 5     |
+--------------------------+-------+
mysql> create table auto_test3(id int NOT NULL AUTO_INCREMENT, PRIMARY KEY (`id`));
mysql> show create table auto_test3;
+------------
+-----------------------------------------------------------------------------------------------------------------------
------+
| Table      | Create Table                                                                                                 |
+------------
+-----------------------------------------------------------------------------------------------------------------------
------+
| auto_test3 | CREATE TABLE `auto_test3` (   `id` int NOT NULL AUTO_INCREMENT,   PRIMARY KEY
(`id`) ) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+------------
+-----------------------------------------------------------------------------------------------------------------------
------+
mysql> insert into auto_test3 values(0), (0), (0);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> select * from auto_test3;
+----+
| id |
+----+
|  5 |
| 15 |
| 25 |
+----+
mysql> show create table auto_test3;
```

 版权所有 © 华为云计算技术有限公司

```
+------------+-----------------------------------------+
| Table      | Create Table                            |
+------------+-----------------------------------------+
| auto_test3 | CREATE TABLE `auto_test3` (
`id` int NOT NULL AUTO_INCREMENT,   PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=35 DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------+
```

7.9 表的自增 AUTO_INCREMENT 超过数据中该字段的最大值加 1

在数据表中会发现AUTO_INCREMENT的值不等于表中字段最大值+1，可能原因有以下几种：

- 如果步长不为1，则AUTO_INCREMENT=最大值+步长。关于步长不为1的参数说明，请参见**14.1.62 表的自增AUTO_INCREMENT初值与步长**。

```
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 |
+----+
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 |
+------------+-----------------------------------------+
```

- 直接修改表的AUTO_INCREMENT，会导致AUTO_INCREMENT变化。

```
mysql> select * from animals;
+----+-----------+
| id | name      |
+----+-----------+
|  1 | fish      |
|  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;
+---------+-------------------------------------------------------+
```


版权所有 © 华为云计算技术有限公司

```
| 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         |
+---------+-----------------------------------------------------+
```

- 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。

```
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                           |
```

 版权所有 © 华为云计算技术有限公司

```
+------------+----------------------------------------+
| auto_test1 | CREATE TABLE `auto_test1` (
`id` int NOT NULL AUTO_INCREMENT,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=7 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。

```
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`)
```


版权所有 © 华为云计算技术有限公司

```
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8 |
+------------+----------------------------------------+
```

7.10 自增字段值跳变的原因

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 | fish      |
|  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 | fish      |
|   2 | cat       |
|   3 | penguin   |
| 100 | rabbit    |
```


版权所有 © 华为云计算技术有限公司

```
+-----+-----------+
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 | fish      |
|  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 | fish      |
|   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`)
```

 版权所有 © 华为云计算技术有限公司

```
) 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 |
+----+
```


版权所有 © 华为云计算技术有限公司

```
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 |
+------------+----------------------------------------+
```


版权所有 © 华为云计算技术有限公司

- 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    |
```

 版权所有 © 华为云计算技术有限公司

```
| 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 |
+----------------+-------------------------------------------------------------+
```

7.11 修改表的自增 AUTO_INCREMENT 值

修改方法如下：

- 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。

```
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   |
+---------+-----------------------------------------------------------------+
1 row in set (0.00 sec)
mysql> select * from animals;
+-----+-----------+
| id  | name      |
+-----+-----------+
| -50 | -middle   |
|   1 | fish      |
|   2 | cat       |
|  50 | middle    |
| 100 | rabbit    |
+-----+-----------+
11 rows in set (0.00 sec)
mysql> alter table animals AUTO_INCREMENT=200;
Query OK, 0 rows affected (0.22 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=200 DEFAULT CHARSET=utf8 |
+---------+-----------------------------------------------------------+
```

- 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 | fish      |
|   2 | cat       |
|  50 | middle    |
| 100 | rabbit    |
+-----+-----------+
mysql> show create table animals;
+---------+-----------------------------------------------------------+
| Table   | Create Table                                              |
+---------+-----------------------------------------------------------+
| animals | CREATE TABLE `animals` (
```

 版权所有 © 华为云计算技术有限公司

```
`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 | fish      |
|   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)
```

- AUTO_INCREMENT无法修改为负数。

```
mysql> 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
```

7.12 自增主键达到上限，无法插入数据

场景现象

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。

版权所有 © 华为云计算技术有限公司

```
mysql> create table auto_test5(id tinyint not null AUTO_INCREMENT, name varchar(8), PRIMARY KEY
(`id`));
Query OK, 0 rows affected (0.06 sec) mysql> insert into auto_test5(name) values('A'),('B'),('C');
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> select * from auto_test5;
+----+------+
| id | name |
+----+------+
|  1 | A    |
|  2 | B    |
|  3 | C    |
+----+------+
3 rows in set (0.00 sec)
mysql> alter table auto_test5 AUTO_INCREMENT=125;
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0
mysql> show create table auto_test5;
+------------+-----------------------------------------------------------------+
| Table      | Create Table                                                    |
+------------+-----------------------------------------------------------------+
| auto_test5 | CREATE TABLE `auto_test5` (
`id` tinyint NOT NULL AUTO_INCREMENT,   `name` varchar(8) DEFAULT NULL,
PRIMARY KEY (`id`) ) ENGINE=InnoDB AUTO_INCREMENT=125 DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------------------------------+
mysql> insert into auto_test5(name) values('X'),('Y'),('Z');
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> select * from auto_test5;
+-----+------+
| id  | name |
+-----+------+
|   1 | A    |
|   2 | B    |
|   3 | C    |
| 125 | X    |
| 126 | Y    |
| 127 | Z    |
+-----+------+
6 rows in set (0.00 sec)
mysql> show create table auto_test5;
+------------+-----------------------------------------------------------------+
| Table      | Create Table                                                    |
+------------+-----------------------------------------------------------------+
| auto_test5 | CREATE TABLE `auto_test5` (   `id` tinyint NOT NULL AUTO_INCREMENT,   `name` varchar(8)
DEFAULT NULL,   PRIMARY KEY (`id`) ) ENGINE=InnoDB AUTO_INCREMENT=127 DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------------------------------+
mysql> insert into auto_test5(name) values('D');
ERROR 1062 (23000): Duplicate entry '127' for key 'auto_test5.PRIMARY'
```

解决方案

- 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。（ 数据导入导出的方法有多种，此处仅为示例 ）

  ```
  mysql>  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)
  mysql> 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
  mysql> select * from auto_test5_tmp;
  +----+------+
  | id | name |
  +----+------+
  |  1 | A    |
  |  2 | B    |
  |  3 | C    |
  ```

 版权所有 © 华为云计算技术有限公司

```
|  4 | X    |
|  5 | Y    |
|  6 | Z    |
+----+------+
mysql> drop table auto_test5;
mysql> rename table auto_test5_tmp to auto_test5;
Query OK, 0 rows affected (0.12 sec)
mysql> select * from auto_test5;
+----+------+
| id | name |
+----+------+
|  1 | A    |
|  2 | B    |
|  3 | C    |
|  4 | X    |
|  5 | Y    |
|  6 | Z    |
+----+------+
6 rows in set (0.01 sec)
mysql> show create table auto_test5;
+------------+-----------------------------------------------------------------+
| Table      | Create Table                                                    |
+------------+-----------------------------------------------------------------+
| auto_test5 | CREATE TABLE `auto_test5` (
`id` tinyint NOT NULL AUTO_INCREMENT,   `name` varchar(8) DEFAULT NULL,
PRIMARY KEY (`id`) ) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------------------------------+
```

- 如果确实是自增主键的取值范围不够，则修改自增主键的字段类型，使其能存更多、更大的数据。

```
mysql> select * from auto_test6;
+-----+------+
| id  | name |
+-----+------+
|   1 | A    |
|   2 | B    |
|   3 | C    |
| 125 | X    |
| 126 | Y    |
| 127 | Z    |
+-----+------+
6 rows in set (0.00 sec)
mysql> show create table auto_test6;
+------------+-----------------------------------------------------------------+
| Table      | Create Table                                                    |
+------------+-----------------------------------------------------------------+
| auto_test6 | CREATE TABLE `auto_test6` (
`id` tinyint NOT NULL AUTO_INCREMENT,   `name` varchar(8) DEFAULT NULL,
PRIMARY KEY (`id`) ) ENGINE=InnoDB AUTO_INCREMENT=127 DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------------------------------+
mysql> 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
mysql> show create table auto_test6;
+------------+-----------------------------------------------------------------+
| Table      | Create Table                                                    |
+------------+-----------------------------------------------------------------+
| auto_test6 | CREATE TABLE `auto_test6` (
`id` int NOT NULL AUTO_INCREMENT,   `name` varchar(8) DEFAULT NULL,
PRIMARY KEY (`id`) ) ENGINE=InnoDB AUTO_INCREMENT=128 DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------------------------------+
1 row in set (0.00 sec)
mysql> insert into auto_test6(name) values('D');
Query OK, 1 row affected (0.01 sec)
mysql> select * from auto_test6;
+-----+------+
| id  | name |
+-----+------+
|   1 | A    |
|   2 | B    |
```


版权所有 © 华为云计算技术有限公司

```
|   3 | C    |
| 125 | X    |
| 126 | Y    |
| 127 | Z    |
| 128 | D    |
+-----+------+
7 rows in set (0.00 sec)
mysql> show create table auto_test6;
+------------+-----------------------------------------------------------------+
| Table      | Create Table                                                    |
+------------+-----------------------------------------------------------------+
| auto_test6 | CREATE TABLE `auto_test6` (
`id` int NOT NULL AUTO_INCREMENT,  `name` varchar(8) DEFAULT NULL,
PRIMARY KEY (`id`) ) ENGINE=InnoDB AUTO_INCREMENT=129 DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------------------------------+
1 row in set (0.01 sec)
```

7.13 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>
```

 版权所有 © 华为云计算技术有限公司

- 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'
mysql>
```

 版权所有 © 华为云计算技术有限公司

7.14 pt-osc 工具连接 RDS for MySQL 主备实例卡住

场景描述

使用pt-osc工具（pt-online-schema-change）进行online DDL操作时，会遇到连接本地单机实例执行命令成功，但连接云上RDS for MySQL主备实例会卡住的情况，如下图所示，没有输出：

原因分析

pt-osc的工作原理：

1. 创建一个与原表结构相同的空表，表名是 _new 后缀。
2. 修改1创建的空表的表结构。
3. 在原表上加三个触发器：delete/update/insert，用于复制数据过程中，将原表中要执行的语句在新表中执行。
4. 将原表数据以数据块（chunk）的形式复制到新表。
5. **rename**原表为旧表，并把新表**rename**为原表名，然后删除旧表。
6. 删除触发器。

因为涉及大量数据复制，对于有从库的MySQL，必然会带来主备复制延迟，可能影响从库业务。考虑从库延迟情况，pt-osc工具提供以下几个控制选项：

- --max-lag
- --check-interval
- --recursion-method
- --check-slave-lag

因此，从库延迟超过max-lag，则停止复制数据，等待check-interval秒后再开始复制数据；check-slave-lag指定slave的机器，只会对比这台slave的延迟情况。recursion-method是主库寻找从库的方法：processlist（默认值，关注从库延迟）、hosts、dsn、none（忽略从库延迟）。更多信息，请参见**pt-osc官方文档**。

本案例场景中：

- pt-osc工具连接云上RDS for MySQL主备实例卡住，是因为存在主从复制延迟导致工具停止复制数据。配置项“--recursion-method=none”表示忽略主从延迟，添加该配置项可以解决问题。
- 忽略主从延迟，会导致复制数据比较快，当需要尽可能的对服务产生小的影响，可以设置“--max-load”配置项。

解决方案

pt-osc工具加上“--recursion-method=none”配置项，忽略复制延迟，即可解决问题。

pt-osc常见命令参考：

 版权所有 © 华为云计算技术有限公司

- 增加字段

  pt-online-schema-change --user=root --password=xxx --host=xxx --alter “ADD COLUMN content text” D=aaa,t=tmp_test --no-check-replication-filters --alter-foreign-keys-method=auto --recursion-method=none --print --execute

- 删除字段

  pt-online-schema-change --user=root --password=xxx --host=xxx --alter "DROP COLUMN content " D=aaa,t=tmp_test --no-check-replication-filters --alter-foreign-keys-method=auto --recursion-method=none --quiet --execute

- 修改字段

  pt-online-schema-change --user=root --password=xxx --host=xxx --alter “MODIFY COLUMN age TINYINT NOT NULL DEFAULT 0” D=aaa,t=tmp_test --no-check-replication-filters --alter-foreign-keys-method=auto --recursion-method=none --quiet --execute

- 字段改名

  pt-online-schema-change --user=root --password=xxx --host=xxx --alter “CHANGE COLUMN age address varchar(30)” D=aaa,t=tmp_test --no-check-alter --no-check-replication-filters --alter-foreign-keys-method=auto --recursion-method=none --quiet --execute

- 增加索引

  pt-online-schema-change --user=root --password=xxx --host=xxx --alter “ADD INDEX idx_address(address)” D=aaa,t=tmp_test --no-check-alter --no-check-replication-filters --alter-foreign-keys-method=auto --recursion-method=none --print --execute

- 删除索引

  pt-online-schema-change --user=root --password=xxx --host=xxx --alter “DROP INDEX idx_address” D=aaa,t=tmp_test --no-check-alter --no-check-replication-filters --alter-foreign-keys-method=auto --recursion-method=none --print --execute

如果业务需要关注复制延迟，可以根据业务需要调整如下参数：max-lag、check-interval、recursion-method、check-slave-lag。更多信息，请参见**pt-osc官方文档**。

7.15 购买 RDS 实例支付报错：Policy doesn't allow bss:order:update to be performed

场景描述

购买包年/包月RDS实例时，单击“去支付”后页面无反应或者报错：Policy doesn't allow bss:order:update to be performed

原因分析

当前账号没有操作订单的权限。

解决方案

联系主账号管理员添加“bss:order:update”权限。

版权所有 © 华为云计算技术有限公司

7.16 RDS for MySQL 是否可以修改数据库名称

场景描述

RDS界面和DAS界面，都不支持修改RDS for MySQL数据库名称。

解决方案

使用DRS将RDS for MySQL数据从源库A迁移到目标库B，两个数据库名称不同。具体迁移方法，请参见**迁移方案总览**。

7.17 购买 RDS 实例报错：无 IAM 的 agency 相关权限

场景描述

使用IAM子账号购买RDS实例时报错：无IAM的agency相关权限，请授权后重试。

原因分析

如果购买实例时勾选了“存储空间自动扩容”，那么可能是当前IAM子账号没有添加存储空间自动扩容的权限。

解决方案

给IAM子账号添加自动扩容授权项，详见**权限管理**中“常用操作与对应授权项 > 存储空间自动扩容”的内容。

版权所有 © 华为云计算技术有限公司