ALIBABA CLOUD

阿里云

云数据库RDS
常见问题（旧）

文档版本：20220325

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.连接/网络

1.1. RDS for PostgreSQL/PPAS 如何定位本地IP

问题描述

- 已经将本地设备的公网IP地址添加到RDS白名单，但是仍然无法访问RDS实例，而其他设备能访问该RDS实例。
- 已经将本地设备的公网IP地址添加到RDS白名单，但是仍然无法访问RDS实例，而将RDS白名单设置为公司的网段或者0.0.0.0/0后，该设备可以访问RDS实例。

以上的任意一种情形，都很可能是因为您添加到白名单的本地设备公网IP地址不正确，本文介绍如何查询到本地设备的真实出口IP地址。

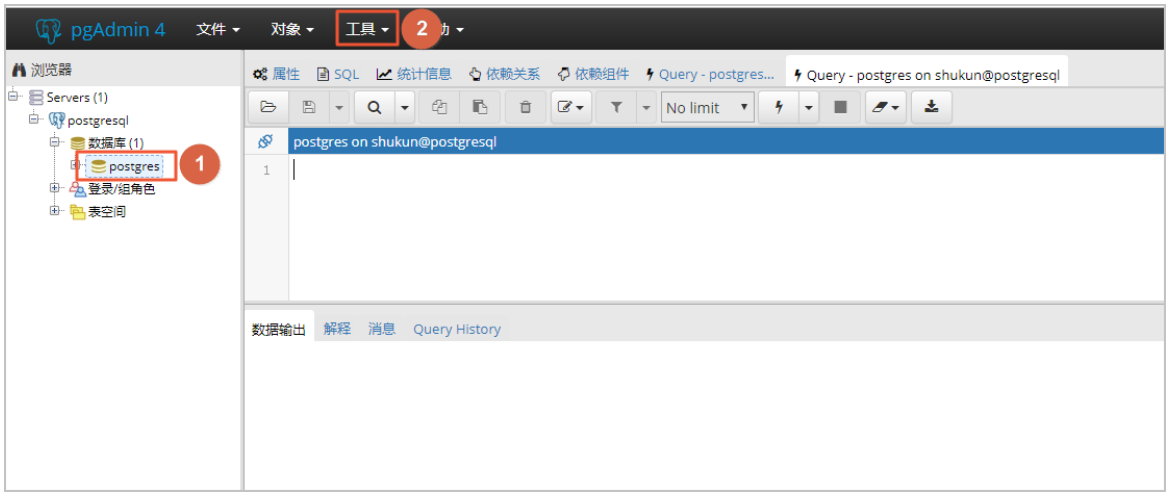
> **说明** 本文只适用于ECS以外的设备访问RDS实例的情况。如果是ECS实例访问RDS实例，可以在ECS实例的详情页面查看准确的公网IP地址和内网IP地址。

注意事项

如果您发现您本地设备的公网IP地址会变化，而且建立的连接是用于生产环境，则建议您改为使用内网连接，或者在白名单中配置合理的公网IP段，确保不会因为IP地址改变而断连。

处理步骤

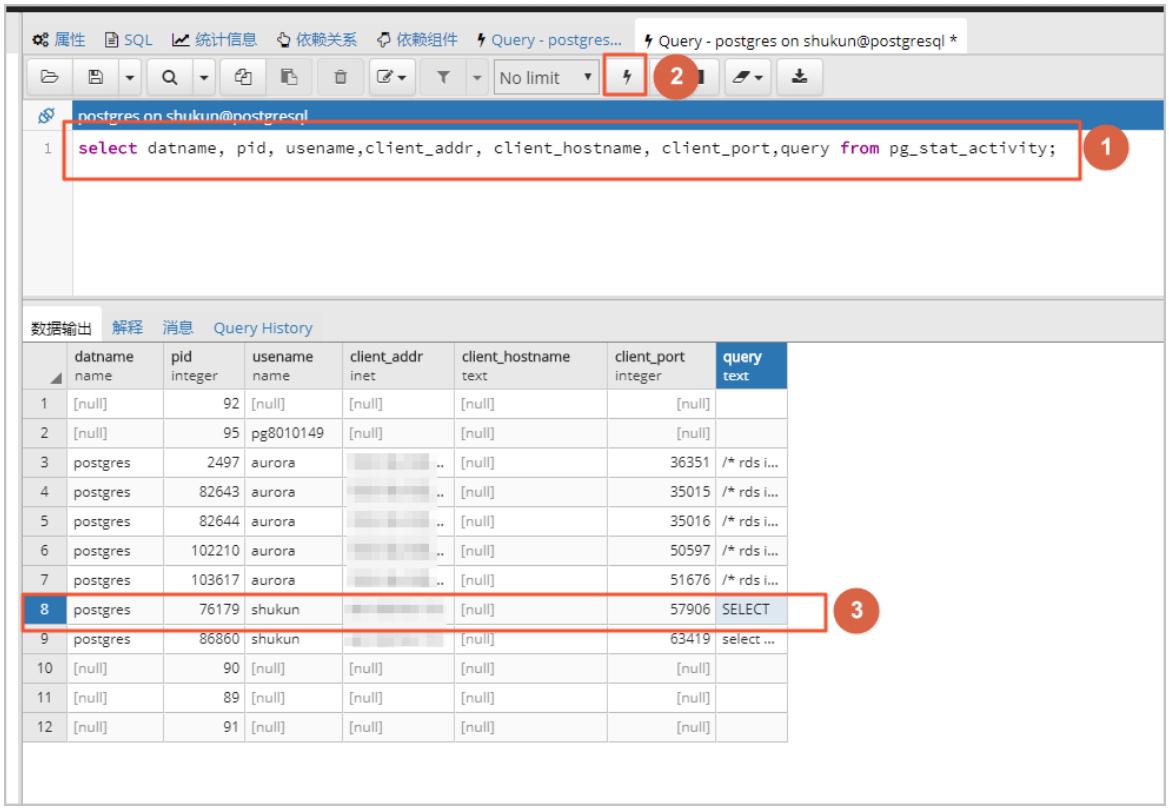
1. 将 IP 地址 0.0.0.0/0 加入 RDS for PostgreSQL/PPAS 的白名单，操作方法请参见设置白名单。
2. 使用 pgAdmin 4客户端连接RDS for PostgreSQL或者RDS for PPAS实例。
3. 在左侧展开**数据库**，选择**postgres**，单击上方**工具 > 查询工具**。



4. 输入如下查询命令并执行。

```
select datname, pid, usename,client_addr, client_hostname, client_port,query  from pg_stat_activity;
```

5. 查看结果里**query**列的值为**select**所对应的**client_addr**列的 IP，即为真实的出口 IP。



6. 将步骤 1 在白名单中添加的 0.0.0.0/0 条目删除，添加上真实的出口IP。

1.2. RDS for MySQL 连接数满情况的处理

连接数满会导致客户端无法连接到RDS for MySQL数据库。

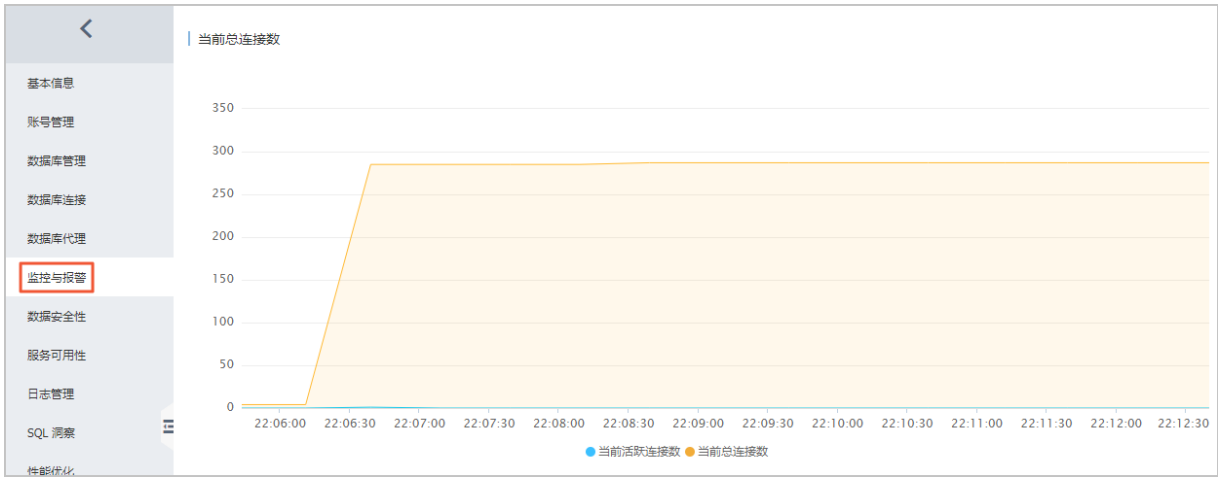
连接数满通常是两种原因导致的：

- 空闲连接过多。
- 活动连接过多。

空闲连接过多

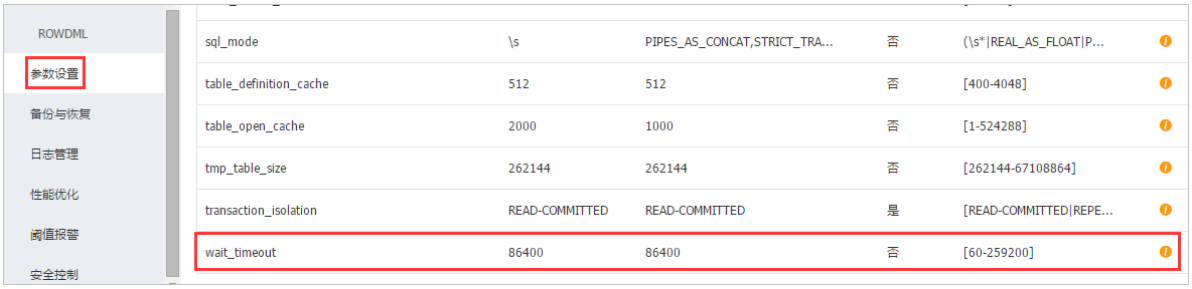
原因

- 应用使用长连接模式：比如Java应用配置的连接池初始连接数设置过高，应用启动后建立了多个到RDS实例的空闲连接。
- 应用使用短连接模式：比如PHP应用，出现大量的空闲连接说明应用没有在查询执行完毕后关闭连接。

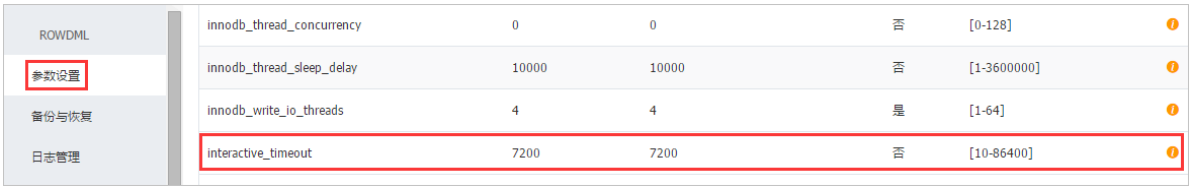


解决方法

- 通过DMS或者 `kill` 命令来终止当前空闲会话，详细步骤请参见RDS for MySQL如何终止会话。
- 修改应用，长连接模式需要启用连接池的复用功能（建议也启用连接检测功能）。
- 修改应用，短连接模式需要在代码中修改查询结束后调用关闭连接的方法。
- 对于非交互模式连接，在控制台的**参数设置**里设置wait_timeout参数为较小值。wait_timeout参数控制非交互模式连接的超时时间（单位秒，默认值为24小时即86400秒），当非交互式连接空闲时间超过wait_timeout指定的时间后，RDS实例会主动关闭连接。



- 对于交互模式连接，在控制台的**参数设置**里设置interactive_timeout参数为较小值。interactive_timeout参数控制交互模式连接的超时时间（单位秒，默认值为2小时即7200秒），当交互式连接空闲时间超过interactive_timeout指定的时间后，RDS实例会主动关闭连接。

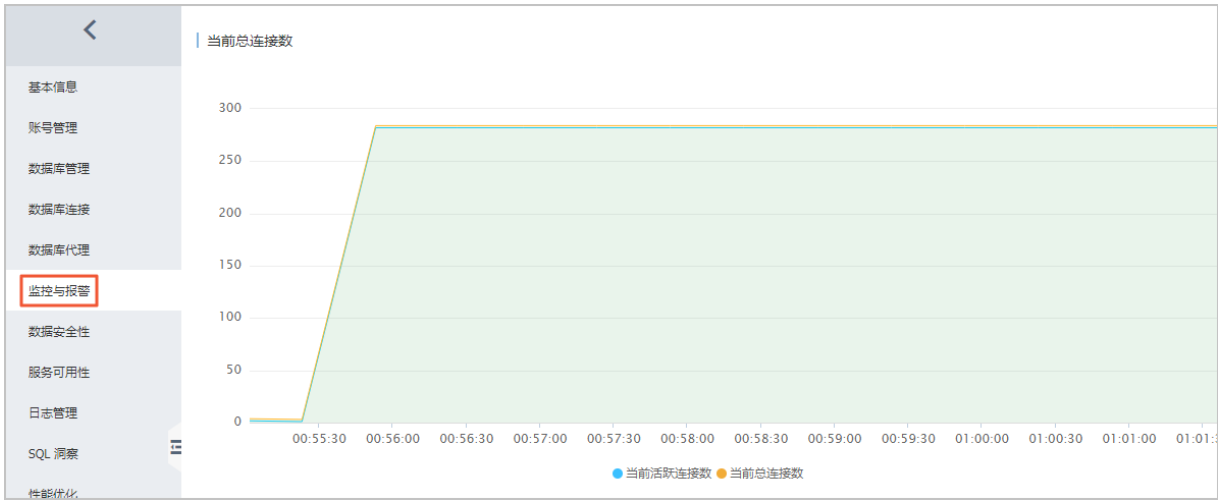


> **说明**
>
> - 在RDS for MySQL实例连接数达到上限的情况下，**通过DMS或者其他方式是无法连接实例的**；因此对于长连接模式，建议连接池的最大连接数要略小于实例规格的连接数限制，比如保留10个连接给DMS或其他管理操作使用。当发生无法连接的情况时，建议先在控制台修改wait_timeout参数为较小值，促使RDS实例主动关闭空闲时间超过阈值的连接。
> - 通常情况下，应用到RDS实例会采用非交互模式，具体采用哪个模式需要查看应用的连接方式配置，比如PHP通过传递MYSQL_CLIENT_INTERACTIVE常量给mysql_connect()函数即可开启连接的交互模式。
> - wait_timeout和interactive_timeout这两个参数的修改，修改前已经存在的会话保持修改前的设置，修改后新创建的会话使用新的参数设置。

活动连接过多

原因

- 锁等待导致活动连接数堆积（包括InnoDB锁等待、表元数据锁等待）。
- CPU使用率过高导致活动连接数堆积。
- IOPS使用率高导致活动连接数堆积。



解决方法

- InnoDB锁等待处理，请参见RDS for MySQL InnoDB 锁等待和锁等待超时的处理。
- 表元数据锁等待，请参见解决MDL锁导致无法操作数据库的问题。
- CPU使用率高导致活动连接数堆积，请参见RDS for MySQL/MariaDB CPU使用率高的原因和解决方法。
- IOPS使用率高导致活动连接数堆积，请参见MySQL IOPS 使用率高的原因和解决方法。

1.3. RDS for PostgreSQL 连接数满情况的处理

连接数满会导致客户端无法连接到RDS for PostgreSQL数据库。

问题现象

无法连接RDS for PostgreSQL数据库，报错如下：

```
FATAL: remaining connection slots are reserved for non-replication superuser connections
```

问题原因

连接数满导致无法连接。

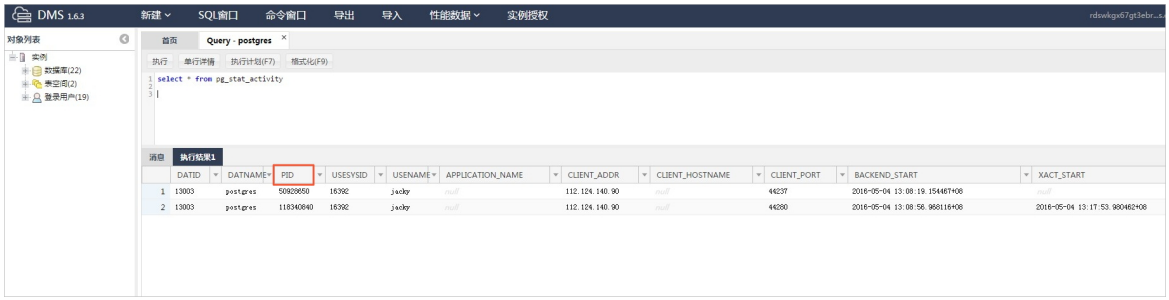
解决方法

1. 通过DMS登录RDS数据库。
2. 选择**SQL操作 > SQL窗口**，通过如下命令检查当前连接数的限制。

```
show max_connections;
```

3. 使用如下命令查看当前连接，记下想要终止的连接的PID。

```
select * from pg_stat_activity;
```



4. 使用如下命令终止连接。

```
SELECT pg_terminate_backend(<PID>) FROM pg_stat_activity;
```

如果问题还未能解决，请通过提交工单联系售后服务。

1.4. JAVA连接RDS for MySQL的测试程序

若您要连接云数据库MySQL版的测试程序，您可以选择以下任意一种方法：

- 通过阿里云SDK

  在使用Java开发RDS管理和连接时，您可以通过阿里云的SDK连接云数据库MySQL版的测试程序。您需要先安装JDK1.7及以上版本，然后通过maven安装阿里云的Java SDK。单击这里，然后下载阿里云关系型数据库所对应的SDK。

- 通过MySQL客户端

  您可以使用MySQL Connector连接云数据库MySQL版的测试程序。单击这里下载，将对应的Jar包引入到build path。

- 通过代码

  您可以通过代码连接云数据库MySQL版的测试程序，示例代码如下：

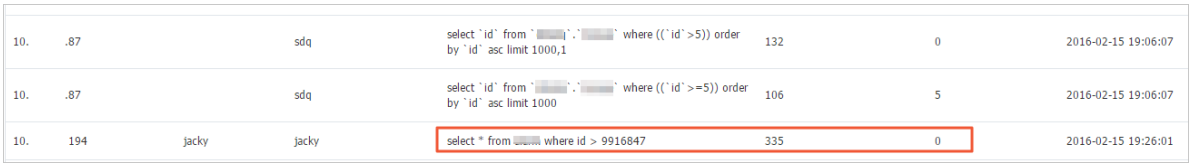
```
    import java.sql.Connection;
    import java.sql.DriverManager;
    import java.sql.ResultSet;
    import java.sql.SQLException;
    import java.sql.Statement;
    public class mysqlconnection {
    public static void main(String[] args) {
    Connection conn = null;
           String sql;
           String url = "jdbc:mysql://rdssoxxxxxxxxx.mysql.rds.aliyuncs.com:3306?zeroDate
TimeBehavior=convertToNull&amp;"
                   + "user=michael&amp;password=password&amp;useUnicode=true&amp;characte
rEncoding=UTF8";
           try {
               Class.forName("com.mysql.jdbc.Driver");
               conn = DriverManager.getConnection(url);
               Statement stmt = conn.createStatement();
              String  sqlusedb="use test_5";
              int result1 = stmt.executeUpdate(sqlusedb);
               sql = "create table teacher(NO char(20),name varchar(20),primary key(NO))"
;
               int result = stmt.executeUpdate(sql);
               if (result != -1) {
                   sql = "insert into teacher(NO,name) values('2016001','wangsan')";
                   result = stmt.executeUpdate(sql);
                   sql = "insert into teacher(NO,name) values('2016002','zhaosi')";
                   result = stmt.executeUpdate(sql);
                   sql = "select * from teacher";
                   ResultSet rs = stmt.executeQuery(sql);
                   System.out.println("学号\t姓名");
                   while (rs.next()) {
                       System.out
                               .println(rs.getString(1) + "\t" + rs.getString(2));
                   }
               }
           } catch (SQLException e) {
               System.out.println("MySQL操作错误");
               e.printStackTrace();
           } catch (Exception e) {
               e.printStackTrace();
           } finally {
               try {
    conn.close();
    } catch (SQLException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
    }
           }
    }
    }
```

1.5. RDS for MySQL SQL审计查询记录返回0的原因

RDS for MySQL的SQL审计功能保存了执行的SQL记录，包括连接的客户端IP、数据库名称、执行语句账号、SQL语句等等。

以下2种情况会导致返回记录数为 0：

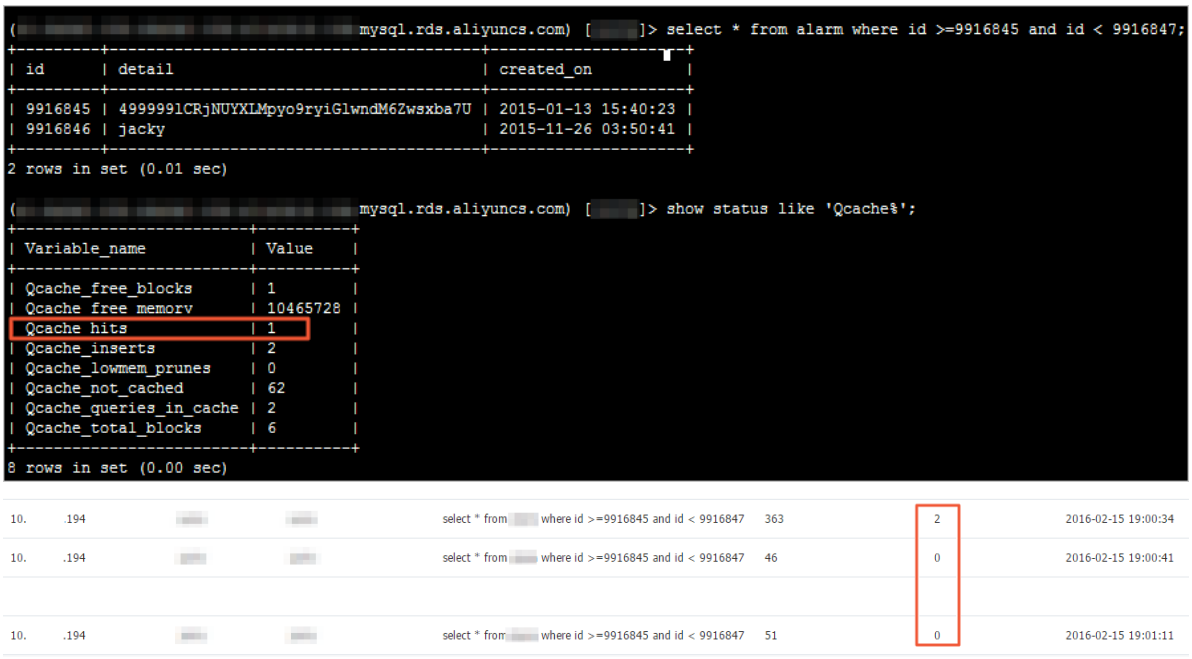
- **未查询到数据**



10.	.87		sdq	select `id` from `[redacted]`.`[redacted]` where ((`id`>5)) order by `id` asc limit 1000,1	132	0	2016-02-15 19:06:07
10.	.87		sdq	select `id` from `[redacted]`.`[redacted]` where ((`id`>=5)) order by `id` asc limit 1000	106	5	2016-02-15 19:06:07
10.	194	jacky	jacky	select * from [redacted] where id > 9916847	335	0	2016-02-15 19:26:01

- **从查询缓存中获取的数据**

  如果实例开启了查询缓存（Query Cache），那么从查询缓存中获取数据的查询，其返回记录数会显示为0，详情请参见RDS MySQL查询缓存（Query Cache）的设置和使用。



```
([redacted] mysql.rds.aliyuncs.com) [[redacted]]> select * from alarm where id >=9916845 and id < 9916847;
+---------+-----------------------------------------+---------------------+
| id      | detail                                  | created_on          |
+---------+-----------------------------------------+---------------------+
| 9916845 | 4999991CRjNUYXLMpyo9ryiGlwndM6Zwsxba7U | 2015-01-13 15:40:23 |
| 9916846 | jacky                                   | 2015-11-26 03:50:41 |
+---------+-----------------------------------------+---------------------+
2 rows in set (0.01 sec)

([redacted] mysql.rds.aliyuncs.com) [[redacted]]> show status like 'Qcache%';
+-------------------------+----------+
| Variable_name           | Value    |
+-------------------------+----------+
| Qcache_free_blocks      | 1        |
| Qcache_free_memory      | 10465728 |
| Qcache_hits             | 1        |
| Qcache_inserts          | 2        |
| Qcache_lowmem_prunes    | 0        |
| Qcache_not_cached       | 62       |
| Qcache_queries_in_cache | 2        |
| Qcache_total_blocks     | 6        |
+-------------------------+----------+
8 rows in set (0.00 sec)
```

10.	.194	[redacted]	[redacted]	select * from [redacted] where id >=9916845 and id < 9916847	363	2	2016-02-15 19:00:34
10.	.194	[redacted]	[redacted]	select * from [redacted] where id >=9916845 and id < 9916847	46	0	2016-02-15 19:00:41
10.	.194	[redacted]	[redacted]	select * from [redacted] where id >=9916845 and id < 9916847	51	0	2016-02-15 19:01:11

1.6. JAVA连接RDS for SQL Server

前提条件

需要jdk1.7及以上版本，参考微软官方要求并下载jar包。

工程代码

使用sqljdbc42.jar导入到项目工程代码片段：

```
String userName = "user";  //默认用户名
String userPwd = "pass321";   //密码
String url="jdbc:sqlserver://rds.sqlserver.rds.aliyuncs.com:3433;DatabaseName=dbtest";
java.sql.Connection  dbConn = DriverManager.getConnection(url, userName, userPwd);
Object sta = dbConn.createStatement();
String sql = "create table student"+ "     (id int , name varchar(10))";
int row = ((Statement) sta).executeUpdate(sql);
System.out.println("successfully");
```

2.备份/恢复/迁移

2.1. RDS for MySQL查看增量数据的方法

RDS for MySQL查看增量数据可以通过SQL洞察、Binlog以及DTS订阅三种方式。

SQL洞察

SQL洞察会统计所有的DML和DDL操作的信息，这些信息是采集系统对网络上的包进行采集得到的。SQL洞察并不会解析实际的参数的值， 并且在SQL查询量较大的时候会丢失少量的记录。因此通过这种方式来统计准确的增量数据是不精确的。

Binlog

Binlog会准确记录数据库所有的增删改操作。该日志可以准确的恢复用户的增量数据。RDS的Binlog会先存储在实例中，系统会定期上传到OSS上面备份，然后清理实例中的Binlog。

操作步骤

1. 登录RDS管理控制台 。
2. 在页面左上角，选择实例所在的地域。

3. 找到目标实例，单击实例ID。
4. 在左侧导航栏单击**备份恢复**。
5. 在**日志备份**找到所需的Binlog，单击右侧**下载**。
6. 单击**复制外网地址**。

7. 在已安装MySQL的Linux系统内使用如下命令下载Binlog：

```
wget -c '<Binlog文件外网下载地址>'
```

8. 解密文件转换为可读形式，命令如下：

```
mysqlbinlog --no-defaults -v --base64-output=decode-rows <Binlog文件名> > binlog.sql
```

9. 用 `more binlog.sql` 查看具体的日志信息，或者导出后检查问题SQL。

DTS订阅

DTS的数据订阅功能可以将RDS的增量数据实时推送给用户，用户可以定制增量数据，可以选择部分表的结构或者数据的增量，详情请参见数据订阅（新版）。

2.2. RDS for MySQL通过mysqlbinlog查看binlog日志

当需要查看binlog日志中具体的SQL语句时，可以通过mysqlbinlog命令查看。

前提条件

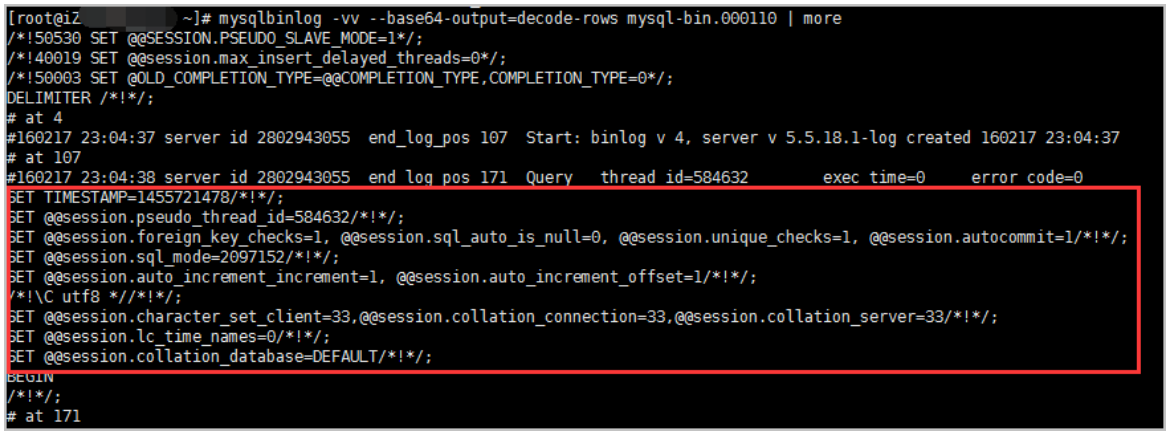
本地Linux主机已安装MySQL。

操作步骤

1. 在安装MySQL的本地Linux主机上下载binlog日志文件。
2. 在命令行中输入如下命令即可查看具体内容。

```
mysqlbinlog -vv --base64-output=decode-rows <binlog文件路径>
```

> 说明
> - -vv：查看具体SQL语句及备注。
> - --base64-output=decode-rows：解码。



常见报错

- 报错1：

```
ERROR: Error in Log_event::read_log_event(): 'Sanity check failed', data_len: 151, event_
type: 35
ERROR: Could not read entry at offset 120: Error in log format or read error.
```

请检查使用的mysqlbinlog版本，例如使用3.3版本会遇到上述错误，3.4版本可以正常查看，这种情况下您可以使用较高版本的mysqlbinlog。

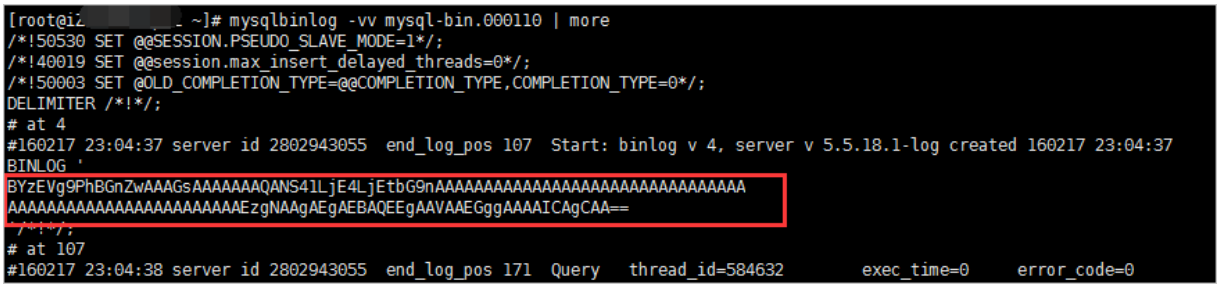
- 报错2：

```
mysqlbinlog: [ERROR] unknown variable 'default-character-set=utf8mb4'
```

my.cnf配置文件中存在 `default-character-set=utf8mb4` 字段，可以通过在命令中加入 `--no-defaults` 参数避免。例如：

常见错误

忘记使用 `--base64-output=decode-rows` 导致输出的是未解码的内容。



2.3. RDS for MySQL恢复单个数据库

2.4. RDS for MySQL备份文件下载工具

功能说明

自动下载RDS for MySQL的备份文件到本地，默认下载前一天的备份文件（时间范围可以自定义修改）。

使用系统

Linux、Windows等支持Python 2.7运行环境的系统。

前提条件

- 需要Python2.7环境，已安装RDS SDK，详情请参见RDS SDK for Python使用参考。
- 工具要足够的执行权限。
- 工具所在的客户端要能访问RDS的公网地址。

工具下载

点击下载

使用方法

运行get_rds_backup.py脚本，后边加上参数**RDS实例ID**、**key**、**key_secret**、备份保存位置，参数之间用空格隔开。

格式

```
python get_rds_backup.py <RDS实例ID> <key> <key_secret> /mnt/
```

示例

```
python get_rds_backup.py rm-m5exxx LTAIPxxxxxxx Cd0xxxxxxx /mnt/
```

2.5. 如何恢复误删除的数据

恢复大量数据方法

- 恢复全量数据
- 恢复SQL Server数据
- 恢复PostgreSQL数据
- 恢复PPAS数据
- 恢复MariaDB数据

恢复少量数据方法

阿里云的数据管理（DMS）提供的数据追踪功能可以逐条恢复数据，且会自动生成回滚语句，便于少量数据的恢复，详细步骤请参见数据追踪中关于数据恢复的内容。

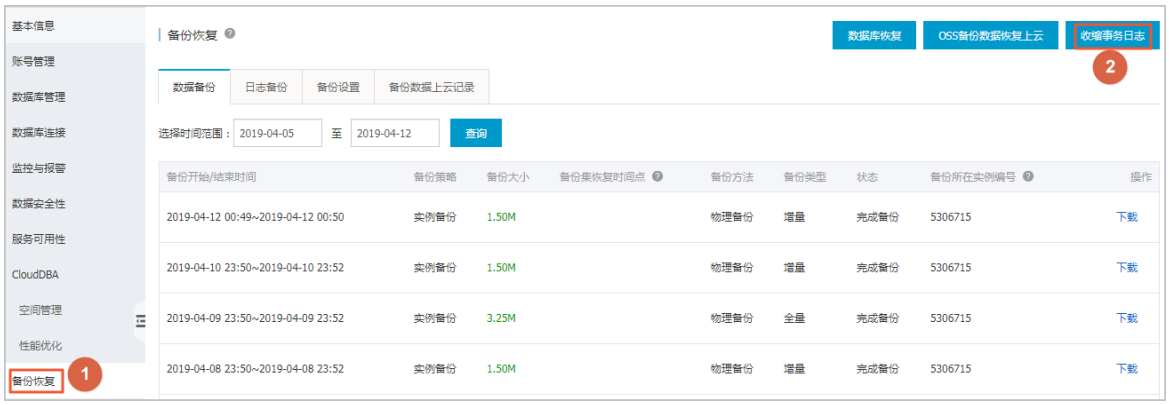
2.6. RDS for SQL Server收缩事务日志

RDS for SQL Server实例可以收缩事务日志，减少日志文件大小。

操作步骤

1. 登录RDS管理控制台。
2. 在页面左上角，选择实例所在地域。

3. 找到目标实例，单击实例ID。
4. 在左侧导航栏单击**备份恢复**。
5. 在右上角单击**收缩事务日志**。



> **说明** 清理后需要等待事务结束，一般需要20分钟左右。另外每次备份时SQL Server也会收缩事务日志。

2.7. 使用RDSBackup工具在Windows下解压MySQL备份文件

使用RDSBackup小工具能够在Windows环境下解压MySQL备份文件。

下载地址： RDSBackup.rar

操作步骤

1. 解压*RDSBackup.rar*。
2. 将下载的MySQL备份文件放入解压后的*RDSBackup*文件夹内。
3. 双击*RDS.bat*，在弹出的对话框中按回车键。

后续操作

解压后的文件需要上传到Linux系统上进行恢复操作，请参见RDS MySQL物理备份文件恢复到自建数据库或RDS MySQL逻辑备份文件恢复到自建数据库。如问题还未解决，请提交工单。

2.8. 如何把数据库迁移到RDS

引擎	
RDS for MySQL	从自建MySQL迁移至RDS for MySQL
RDS for SQL Server	• 从自建SQL Server增量迁移至RDS for SQL Server • 全量备份数据上云（SQL Server 2012、2014、2016、2017和2019） • 全量备份数据上云（SQL Server 2008 R2）
RDS for PostgreSQL	• 从自建数据库迁移到 RDS • 使用pg_dump和pg_restore将自建PostgreSQL数据库迁移到RDS PostgreSQL
RDS for PPAS	• 使用DTS迁移PPAS数据 • 从自建数据库迁移到 RDS
RDS for MariaDB	使用mysqldump将自建MariaDB数据库迁移上云

3.参数/性能

3.1. 解决CPU、内存、空间、IOPS使用率偏高的问题

RDS产品在日常使用过程中会出现CPU、内存、空间以及IOPS使用率高的问题，本文提供了造成问题的原因和解决办法。

RDS for MySQL/MariaDB TX常见问题

- CPU使用率高：MySQL/MariaDB TX CPU 使用率高的原因和解决方法。
- 内存使用率高：MySQL/MariaDB TX 实际内存分配情况介绍。
- 空间使用率高：MySQL/MariaDB TX 实例空间使用率过高的原因和解决方法。
- IOPS使用率高：MySQL/MariaDB TX IOPS 使用率高的原因和解决方法。

SQL Server常见问题

- CPU使用率高：RDS for SQL Server CPU使用率高问题排查。
- 内存使用率高：RDS For SQL Server 查看内存占用情况。
- 空间使用率高：SQL Server实例空间使用率过高的原因和解决方法。

RDS for PostgreSQL/RDS for PPAS常见问题

- CPU使用率高：PostgreSQL/PPAS CPU使用率高的原因及解决办法
- 空间使用率高：磁盘空间占用突然暴增，又很快下降，如何处理。

3.2. RDS for MySQL/MariaDB CPU使用率高的原因和解决方法

RDS for MySQL/MariaDB使用过程中，会遇到CPU使用率过高甚至达到100%的情况。本文将介绍造成该状况的常见原因以及解决方法，并通过CPU使用率为100%的典型场景，来分析引起该状况的原因及其相应的解决方案。

常见原因

系统执行应用提交查询（包括数据修改操作）时需要大量的逻辑读（逻辑IO，执行查询所需访问的表的数据行数），所以系统需要消耗大量的CPU资源以维护从存储系统读取到内存中的数据一致性。

> **说明** 大量行锁冲突、行锁等待或后台任务也有可能会导致实例的CPU使用率过高，但这些情况出现的概率非常低，本文不做讨论。

下文通过一个简化的模型来说明系统资源、语句执行成本以及QPS（Query Per Second每秒执行的查询数）之间的关系：

- 条件：应用模型恒定（应用没有修改）。
- avg_lgc_io：执行每条查询需要的平均逻辑 IO。
- total_lgc_io：实例的 CPU 资源在单位时间内能够处理的逻辑 IO 总量。

- 关系公式：total_lgc_io = avg_lgc_io x QPS -- 单位时间 CPU 资源 = 查询执行的平均成本 x 单位时间执行的查询数量。

解决方法

您可以利用数据管理（DMS）或者CloudDBA解决MySQL/MariaDB实例CPU使用率过高的问题。下文主要介绍使用DMS来解决CPU使用率过高的问题，如何通过CloudDBA来解决CPU使用率过高的问题，可以参考文档利用CloudDBA解决MySQL实例CPU使用率过高的问题。

数据管理工具提供了几种辅助排查并解决实例性能问题的功能，主要有：

- 实例诊断报告。
- SQL 窗口提供的查询优化建议和查看执行计划。
- 实例会话。

其中，实例诊断报告是排查和解决MySQL/MariaDB实例性能问题的最佳工具。无论何种原因导致的性能问题，建议您首先参考下实例诊断报告，尤其是诊断报告中的会话列表和慢SQL汇总。

避免出现CPU使用率达到100%的一般原则

- 设置CPU使用率告警，实例CPU使用率保证一定的冗余度。
- 应用设计和开发过程中，要考虑查询的优化，遵守MySQL优化的一般优化原则，降低查询的逻辑IO，提高应用可扩展性。
- 新功能、新模块上线前，要使用生产环境数据进行压力测试（可以考虑使用阿里云PTS压力测试工具）。
- 新功能、新模块上线前，建议使用生产环境数据进行回归测试。
- 建议经常关注和使用DMS中的诊断报告。

> 注意 关于如何访问DMS中的诊断报告，请参见RDS如何访问诊断报告。

典型示例

以CPU使用率为100%的典型场景为例，下文介绍了两个引起该状况的原因及其解决方案，即应用负载（QPS）高和查询执行成本（查询访问表数据行数avg_lgc_io）高。其中，由于查询执行成本高（查询访问表数据行数多）而导致实例CPU使用率高是MySQL非常常见的问题。

- 应用负载（QPS）高
  - 现象描述
    - 特征：实例的QPS（每秒执行的查询次数）高，查询比较简单、执行效率高、优化余地小。
    - 表现：没有出现慢查询（或者慢查询不是主要原因），且QPS和CPU使用率曲线变化吻合。
    - 常见场景：该状况常见于应用优化过的在线事务交易系统（例如订单系统）、高读取率的热门Web网站应用、第三方压力工具测试（例如 Sysbench）等。

- 解决方案

  对于由应用负载高导致的CPU使用率高的状况，使用SQL查询进行优化的余地不大，建议您从应用架构、实例规格等方面来解决，例如：

  - 升级实例规格，增加CPU资源。
  - 增加只读实例，将对数据一致性不敏感的查询（例如商品种类查询、列车车次查询）转移到只读实例上，分担主实例压力。
  - 使用阿里云DRDS产品，自动进行分库分表，将查询压力分担到多个RDS实例上。
  - 使用阿里云Memcache或者云Redis产品，尽量从缓存中获取常用的查询结果，减轻RDS实例的压力。
  - 对于查询数据比较静态、查询重复度高、查询结果集小于1MB 的应用，考虑开启查询缓存（Query Cache）。

    > **注意** 能否从开启查询缓存（Query Cache）中获益需要经过测试，具体设置请参见RDS for MySQL 查询缓存（Query Cache）的设置和使用。

  - 定期归档历史数据、采用分库分表或者分区的方式减小查询访问的数据量。
  - 尽量优化查询，减少查询的执行成本（逻辑IO，执行需要访问的表数据行数），提高应用可扩展性。

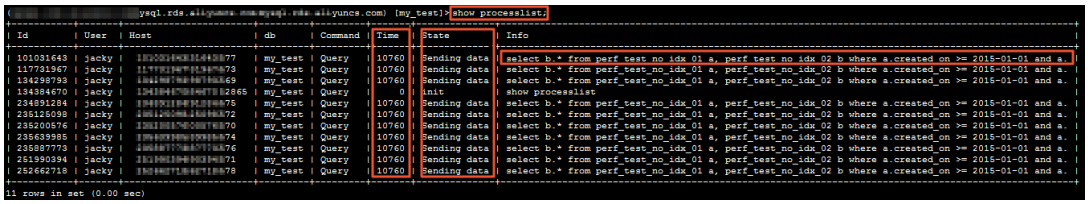
- 查询执行成本（查询访问表数据行数avg_lgc_io）高
  - 现象描述
    - 特征：实例的QPS（每秒执行的查询次数）不高；查询执行效率低、执行时需要扫描大量表中数据、优化余地大。
    - 表现：存在慢查询，QPS和CPU使用率曲线变化不吻合。
    - 原因分析：由于查询执行效率低，为获得预期的结果即需要访问大量的数据（平均逻辑IO高），在QPS并不高的情况下（例如网站访问量不大），就会导致实例的CPU使用率高。
  - 解决方案

    解决该状况的原则是：定位效率低的查询、优化查询的执行效率、降低查询执行的成本。

    操作步骤

    a. 通过如下方式定位效率低的查询：

- 通过 `show processlist;` 或 `show full processlist;` 命令查看当前执行的查询，如下图所示。



对于查询时间长、运行状态（State 列）是Sending data、Copying to tmp table、Copying to tmp table on disk、Sorting result、Using filesort等都可能是有性能问题的查询（SQL）。

> **注意**
>
> - 若在QPS高导致CPU使用率高的场景中，查询执行时间通常比较短，`show processlist;` 命令或实例会话中可能会不容易捕捉到当前执行的查询。您可以执行命令 `explain select b.* from perf_test_no_idx_01 a, perf_test_no_idx_02 b where a.created_on >= 2015-01-01 and a.detail = b.detail` 进行查询：
> - 您可以通过执行类似 `kill 101031643;` 的命令来终止长时间执行的会话，终止会话请参见 RDS for MySQL 如何终止会话。关于长时间执行会话的管理，请参见 RDS for MySQL 管理长时间运行查询。

- 通过DMS查看当前执行的查询，查询步骤如下：
  a. 在DMS控制台上登录数据库。
  b. 选择**性能 > 实例会话**，打开实例会话页面，如下图所示。

  c. 单击SQL列中的查询文本，即可显示完整的查询语句，如下图所示。

b. 得到需要优化的查询后，可以通过DMS控制台上**SQL诊断**来获取查询的优化建议：

> **说明** 诊断报告同样适用于排查历史实例CPU使用率高的问题。

a. 在**DMS 控制台**上登录数据库。

b. 选择**SQL操作 > SQL窗口**，如下图所示。

c. 单击**SQL诊断**，即可得到优化建议，如下图所示。

c. 根据优化建议，添加索引，查询执行成本就会大幅减少，实例CPU使用率100%的问题解决。

3.3. RDS for MySQL如何查看消耗内存高的事件和线程

本文介绍如何查看消耗内存高的事件和线程，为您解决内存相关问题提供参考。

内存是重要的性能参数，内存使用率过高会导致系统响应速度变慢，严重时内存耗尽实例会进行主备切换，导致业务中断。因此我们需要在内存异常升高时及时排查问题，避免影响业务。

前提条件

实例版本如下：

- MySQL 5.7
- MySQL 8.0

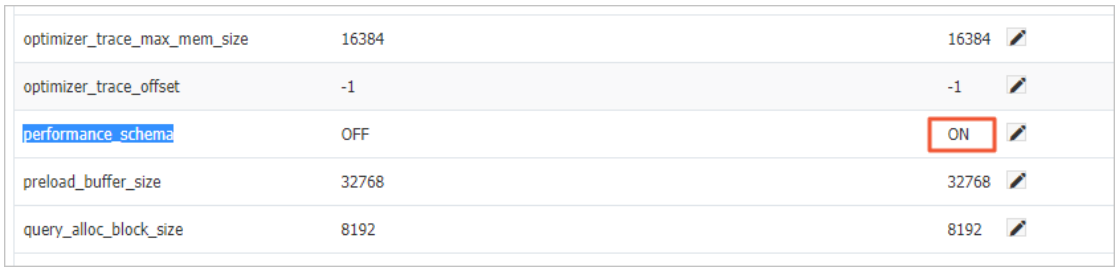
操作步骤

1. 登录RDS管理控制台。
2. 在页面左上角，选择实例所在地域。

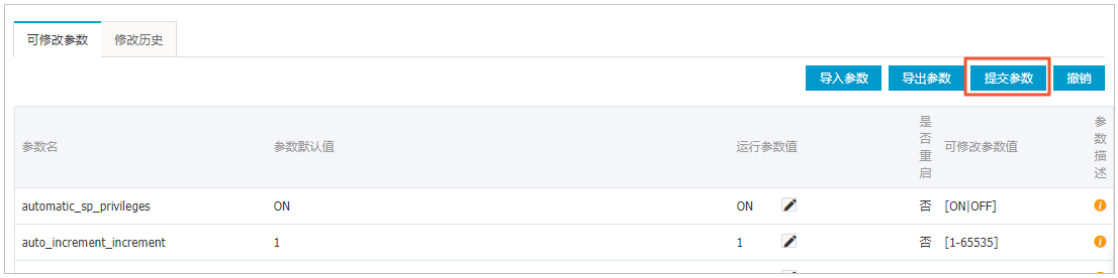
3. 找到目标实例，单击实例ID。
4. 在左侧导航栏中单击**参数设置**。
5. 修改参数performance_schema为ON，如果已经为ON请忽略此步骤。

> **说明** 该操作会重启实例造成连接中断，重启前请做好业务安排，谨慎操作。

i. 单击，修改值为**ON**，单击**确定**。



ii. 单击右上角**提交参数**，等待实例重启完成。



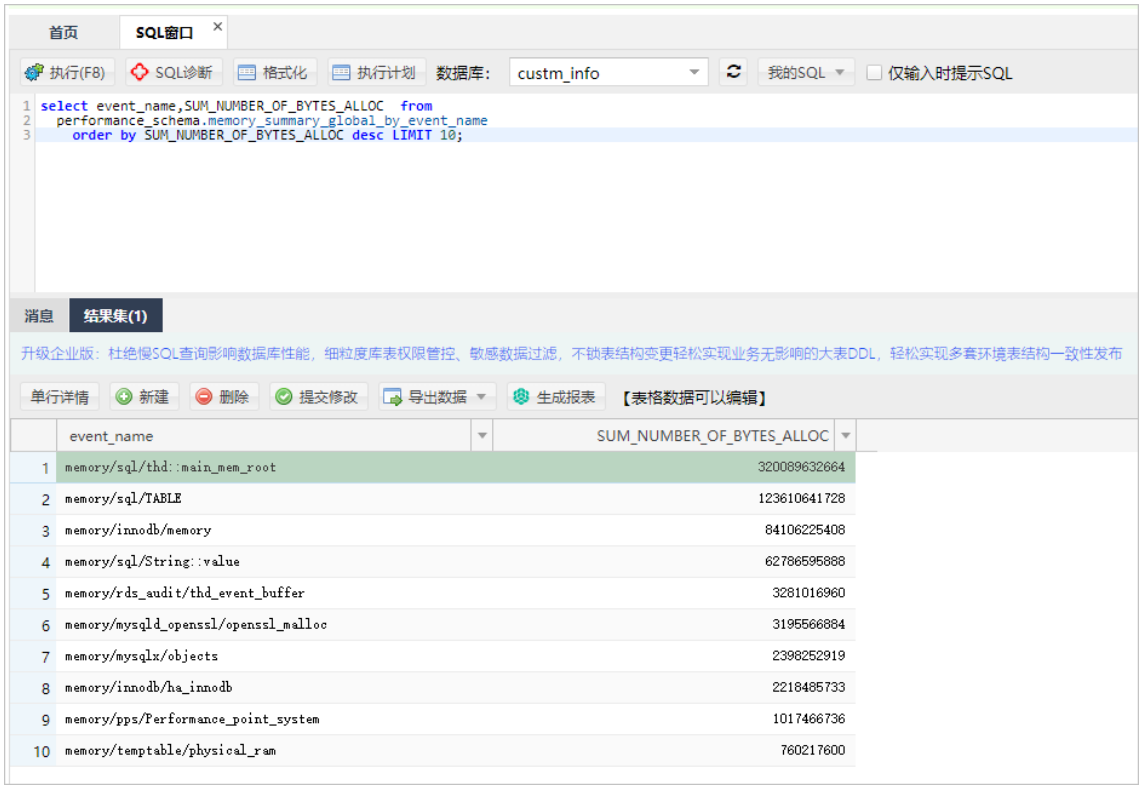
6. 使用DMS或客户端通过客户端、命令行连接RDS MySQL。
7. 执行如下命令打开内存监控：

```
update performance_schema.setup_instruments set enabled = 'yes' where name like 'memory
%';
select * from performance_schema.setup_instruments where name like 'memory%innodb%' lim
it 5;
```

> **说明** 该命令是在线打开内存统计，所以只会统计打开后新增的内存对象，打开前的内存对象不会统计，建议您打开后等待一段时间再执行后续步骤，便于找出内存使用高的线程。

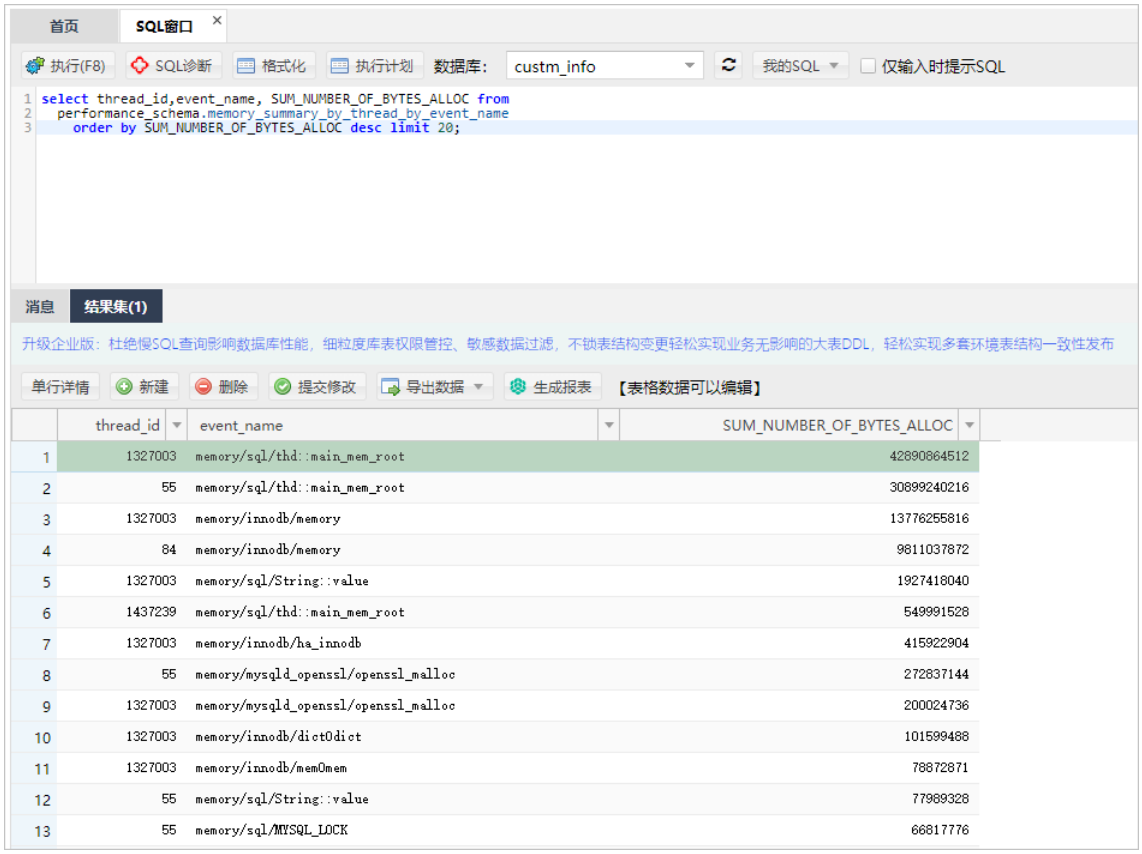
8. 您可以执行命令统计事件和线程的内存消耗量，并进行排序展示。
   - 统计事件消耗内存：

```
select event_name,SUM_NUMBER_OF_BYTES_ALLOC  from
  performance_schema.memory_summary_global_by_event_name
    order by SUM_NUMBER_OF_BYTES_ALLOC desc LIMIT 10;
```



- 统计线程消耗内存：

```
select thread_id,event_name, SUM_NUMBER_OF_BYTES_ALLOC from
  performance_schema.memory_summary_by_thread_by_event_name
    order by SUM_NUMBER_OF_BYTES_ALLOC desc limit 20;
```



	thread_id	event_name	SUM_NUMBER_OF_BYTES_ALLOC
1	1327003	memory/sql/thd::main_mem_root	42890864512
2	55	memory/sql/thd::main_mem_root	30899240216
3	1327003	memory/innodb/memory	13776255816
4	84	memory/innodb/memory	9811037872
5	1327003	memory/sql/String::value	1927418040
6	1437239	memory/sql/thd::main_mem_root	549991528
7	1327003	memory/innodb/ha_innodb	415922904
8	55	memory/mysqld_openssl/openssl_malloc	272837144
9	1327003	memory/mysqld_openssl/openssl_malloc	200024736
10	1327003	memory/innodb/dict0dict	101599488
11	1327003	memory/innodb/mem0mem	78872871
12	55	memory/sql/String::value	77989328
13	55	memory/sql/MYSQL_LOCK	66817776

说明 您也可以执行如下命令查看详细的监控信息：

```
select * from sys.x$memory_by_host_by_current_bytes   ;
select * from sys.x$memory_by_thread_by_current_bytes ;
select * from sys.x$memory_by_user_by_current_bytes   ;
select * from sys.x$memory_global_by_current_bytes    ;
select * from sys.x$memory_global_total               ;
select * from performance_schema.memory_summary_by_account_by_event_name;
select * from performance_schema.memory_summary_by_host_by_event_name;
select * from performance_schema.memory_summary_by_thread_by_event_name;
select * from performance_schema.memory_summary_by_user_by_event_name;
select * from performance_schema.memory_summary_global_by_event_name;
select event_name,current_alloc from sys.memory_global_by_current_bytes where event
_name like '%innodb%';
select event_name,current_alloc from sys.memory_global_by_current_bytes limit 5;
select m.thread_id tid,   USER,   esc.DIGEST_TEXT,   total_allocated  FROM   sys.me
mory_by_thread_by_current_bytes m,   performance_schema.events_statements_current e
sc WHERE m.`thread_id` = esc.THREAD_ID \G
```

下一步

找到问题事件或线程后，您可以排查业务代码和环境，解决内存高的问题。

3.4. RDS for MySQL行锁等待和行锁等待超时的处理

出现场景

当一个连接会话等待另外一个会话持有的互斥行锁时，就会发生行锁等待情况。

通常情况下，持有该互斥行锁的会话会迅速的执行完相关操作并释放掉持有的互斥锁（事务提交或者回滚），然后等待的会话在行锁等待超时时间内获得该互斥行锁，进行下一步操作。

但在某些情况下，比如一个实例未感知到的来自客户端应用的数据库会话中断，持有该互斥行锁的会话长时间不释放该互斥行锁，此时如果有其他会话申请该互斥行锁，则会导致大量的行锁等待与行锁等待超时。

行锁等待超时的报错如下：

```
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

处理方法

本文提供的检查和处理方法，仅当正在发生行锁等待的情况下才成立。因为RDS for MySQL行锁等待默认超时时间为50秒，通常情况下不容易观察到行锁等待的现场，可以通过将**innodb_lock_wait_timeout**参数设置为较大值来复现问题（生产环境不推荐使用过大的**innodb_lock_wait_timeout**参数值）。

1. 通过DMS登录RDS数据库。
2. 单击**性能 > InnoDB锁等待**，检查导致锁等待和锁超时的会话。

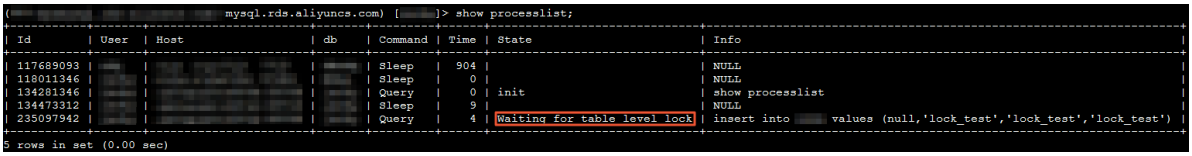
3. 对于标识为**Blocker**的会话（持有锁阻塞其他会话的DML操作，导致行锁等待和行锁等待超时），确认可以接受其对应的事务回滚的情况下，可以将其终止。

3.5. RDS for MySQL表级锁等待

在RDS for MySQL实例日常使用中，会出现表级锁等待的情况，下面列出常见的2个原因。

- **显式lock table**

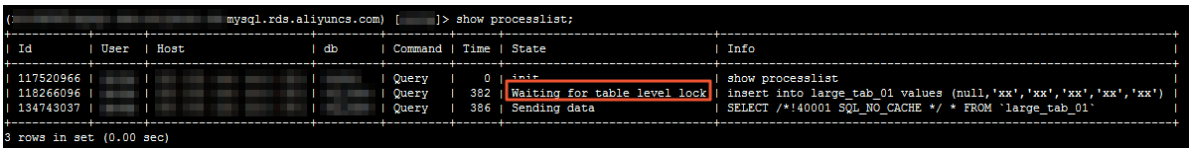
  执行了 `lock tables tab_name read` 导致DML会话等待表级锁。



- **隐式lock table**

  mysqldump使用默认参数进行数据导出时，会默认的开启--lock-tables选项，进而导致导出表上的DML操作等待表级锁。

  > **说明** 对于使用mysqldump导出数据，建议在业务低峰期进行，并且设置--single-transaction选项进行Innodb引擎表导出，避免出现表级锁等待的情况。



3.6. RDS for MySQL管理长时间执行的查询

出现原因

在使用RDS for MySQL的过程中，由于某些原因，例如被SQL注入、SQL执行效率较差、DDL语句引起表元数据锁等待等等，会出现运行时间很长的查询。

- 由于SQL执行效率差而导致的长时间查询。
- 由于被SQL注入而导致的长时间查询。
- 由于DDL语句引起表元数据锁等待。

> **说明** 元数据锁等待的问题请参见解决MDL锁导致无法操作数据库的问题。

长时间执行的查询带来的问题

通常来说，除非是BI/报表类查询，否则长时间执行的查询对于应用缺乏意义，而且会消耗系统资源，比如大量长时间查询可能会引起CPU、IOPS和连接数过高等问题，导致系统不稳定。

如何避免长时间执行的查询

- 应用方面应注意增加防止SQL注入的保护措施。
- 在新功能模块上线前，进行压力测试，避免执行效率很差的SQL大量执行。
- 尽量在业务低峰期进行索引创建删除、表结构修改、表维护和表删除操作。

如何处理长时间执行的查询

请参见解决MDL锁导致无法操作数据库的问题。

3.7. RDS for MySQL如何保证数据库字符编码正确

字符集是数据库设计过程中需要详细考虑的一点，您需要根据业务场景、用户数据等方面来综合考虑。

通过客户端、命令行连接RDS MySQL后，您可以依次执行如下命令查看相应数据库的字符集：

```
use <数据库名>;
```

```
show variables like '%character%';
```

character_set_client	utf8
character_set_connection	utf8
character_set_database	utf8
character_set_filesystem	binary
character_set_results	utf8
character_set_server	utf8
character_set_system	utf8

说明

- 以上参数必须保证除了character_set_filesystem外的参数都统一，才不会出现乱码的情况。
- character_set_client、character_set_connection以及character_set_results是客户端的设置。
- character_set_system、character_set_server以及character_set_database是服务器端的设置。

  服务器端的参数优先级是：

  character_set_database>character_set_server>character_set_system

操作步骤

1. 通过客户端、命令行连接RDS MySQL。
2. 在SQL窗口执行以下命令修改客户端字符集。

   ```
   set names <字符集>;
   ```

3. 在SQL窗口执行以下命令修改character_set_database。

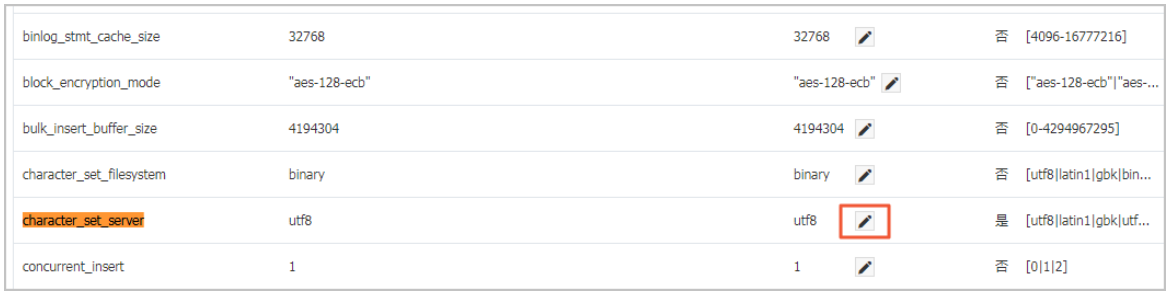
   ```
   ALTER DATABASE <数据库名> CHARACTER SET = <字符集> COLLATE = <字符集排序规则>;
   ```

4. 登录RDS管理控制台。
5. 在页面左上角，选择实例所在地域。

6. 找到目标实例，单击实例ID。
7. 在左侧导航栏中单击**参数设置**。
8. 在**可修改参数**页签下查找到character_set_server，单击右侧

进行修改并单击**确定**。

> **说明** 修改参数character_set_server需要重启实例，建议在业务低峰期进行操作。



9. 在右上角单击**提交参数**，在弹出的对话框中单击**确定**，等待实例重启。

> **说明** 该参数修改后，仅对开启高权限账号的实例后来创建的数据库有效，对当前数据库无效。

character_set_system暂时不提供更改，但是由于其优先级最低因此影响不大。

做完上述的设置之后基本上可以保证不会出现乱码，在代码中设置客户端字符编码时建议通过`set names <字符集>`来修改客户端的设置。

3.8. RDS for MySQL收集表的统计信息

什么是统计信息

RDS for MySQL查询优化器依据表的统计信息计算不同执行计划的代价，因此表上统计信息的准确对查询优化器选取正确的执行计划至关重要。

什么情况下需要收集统计信息

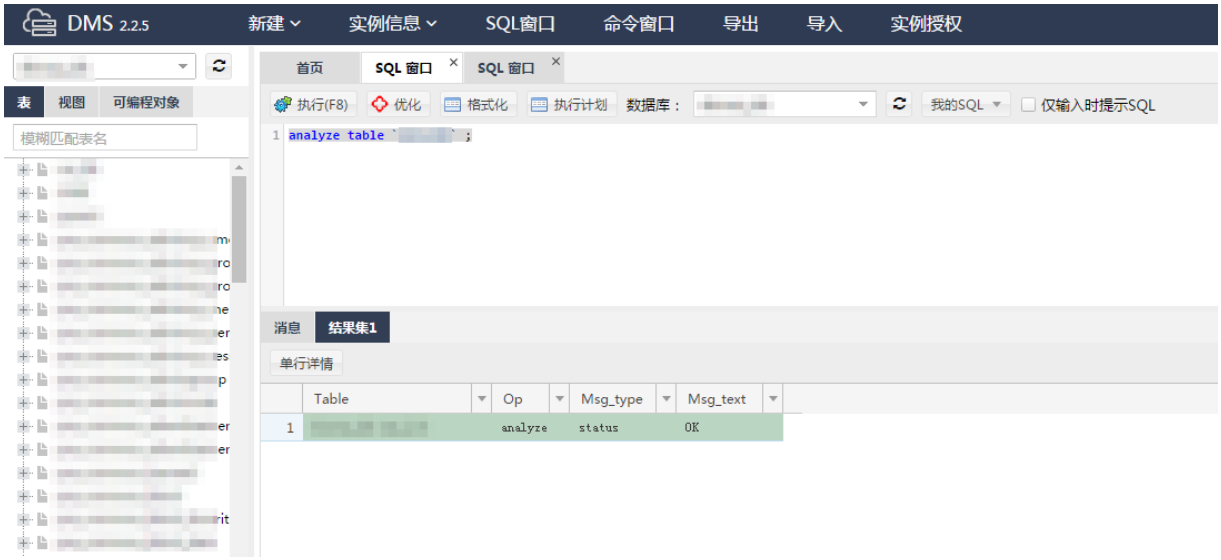
当表上有大量的数据修改时，例如从数据源加载大量数据（ETL）或者大量历史数据归档，建议手动收集下表上的统计信息，以保证查询优化器可以选取最优的执行计划。

如何收集统计信息

您可以通过客户端、命令行连接RDS MySQL后执行以下命令：

```
analyze table <表名>;
```

> **说明** 执行命令期间将对全表加只读锁，建议在业务低峰期执行。



3.9. RDS for MySQL字符集相关说明

字符序命名规则

以字符序对应的字符集名称开头，以 _ci（大小写不敏感）、_cs（大小写敏感）、_bin（按编码值比较，大小写敏感）结尾。

例如：当会话的collation_connction设置为字符序utf8_general_ci时，字符a和字符A是等价的；而当其设置为utf8_bin 时，字符a和字符A是不等价的。

请参考以下示例：

```
(                .mysql.rds.aliyuncs.com) [     ]> show variables like 'coll%';
+----------------------+--------------------+
| Variable_name        | Value              |
+----------------------+--------------------+
| collation_connection | utf8_general_ci    |
| collation_database   | utf8mb4_general_ci |
| collation_server     | utf8mb4_general_ci |
+----------------------+--------------------+
3 rows in set (0.00 sec)

(                .mysql.rds.aliyuncs.com) [     ]> select count(*)  from t where 'a' = 'A';
+----------+
| count(*) |
+----------+
|       16 |
+----------+
1 row in set (0.00 sec)

(                .mysql.rds.aliyuncs.com) [     ]> set collation_connection='utf8_bin';
Query OK, 0 rows affected (0.00 sec)

(                .mysql.rds.aliyuncs.com) [     ]> show variables like 'coll%';
+----------------------+--------------------+
| Variable_name        | Value              |
+----------------------+--------------------+
| collation_connection | utf8_bin           |
| collation_database   | utf8mb4_general_ci |
| collation_server     | utf8mb4_general_ci |
+----------------------+--------------------+
3 rows in set (0.00 sec)

(                .mysql.rds.aliyuncs.com) [     ]> select count(*)  from t where 'a' = 'A';
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (0.00 sec)
```

字符集相关 MySQL 命令

```
show global variables like '%char%';     #查看RDS实例字符集相关参数设置
show global variables like 'coll%';      #查看当前会话字符序相关参数设置
show character set;                      #查看实例支持的字符集
show collation;                          #查看实例支持的字符序
show create table table_name \G          #查看表字符集设置
show create database database_name \G    #查看数据库字符集设置
show create procedure procedure_name \G #查看存储过程字符集设置
show procedure status \G                 #查看存储过程字符集设置
alter database db_name default charset utf8;  #修改数据库的字符集
create database db_name character set utf8;   #创建数据库时指定字符集
alter table tab_name default charset utf8 collate utf8_general_ci;   #修改表字符集和字符序
```

示例如图：

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

控制台修改字符集参数(character_set_server)的方法

注意事项

修改参数character_set_server需要重启实例，建议在业务低峰期进行操作。

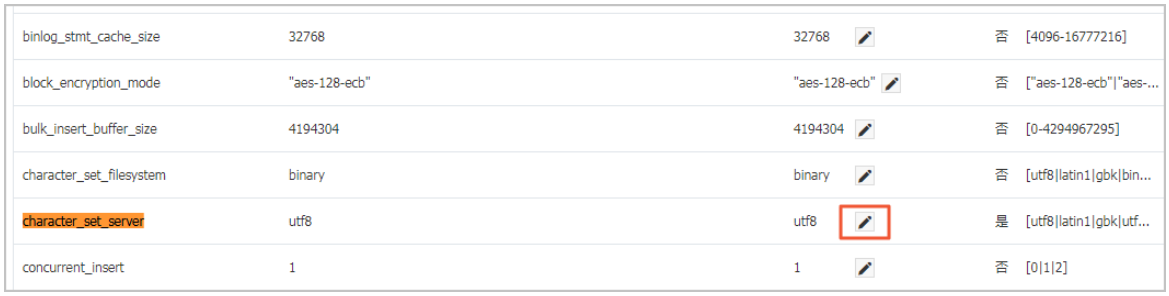
操作步骤

1. 登录RDS管理控制台。
2. 在页面左上角，选择实例所在地域。

3. 找到目标实例，单击实例ID。
4. 在左侧导航栏中单击**参数设置**。
5. 在**可修改参数**页签下查找到character_set_server，单击右侧

   进行修改并单击**确定**。



binlog_stmt_cache_size	32768	32768	否	[4096-16777216]
block_encryption_mode	"aes-128-ecb"	"aes-128-ecb"	否	["aes-128-ecb"|"aes-...
bulk_insert_buffer_size	4194304	4194304	否	[0-4294967295]
character_set_filesystem	binary	binary	否	[utf8|latin1|gbk|bin...
character_set_server	utf8	utf8	是	[utf8|latin1|gbk|utf...
concurrent_insert	1	1	否	[0|1|2]

6. 在右上角单击**提交参数**，在弹出的对话框中单击**确定**，等待实例重启。

   > **说明** 该参数修改后，仅对开启高权限账号的实例后来创建的数据库有效，对当前数据库无效。

7.

使用SQL语句修改数据库字符集的方法

语法如下：

```
修改库：ALTER DATABASE <库名> CHARACTER SET  <字符集名称> COLLATE  <排序规则名称>;
修改表:ALTER TABLE <表名> CONVERT TO CHARACTER SET <字符集名称>  COLLATE  <排序规则名称>;
修改一列:ALTER TABLE <表名> MODIFY <列名>  <字段类型> CHARACTER SET  <字符集名称>  COLLATE <排序规则名称>;
```

示例：三条sql 分别将库dbsdq、表tt2 、表tt2中的c2列修改为utf8mb4 字符集，命令如下：

```
alter database dbsdq character set utf8mb4 collate utf8mb4_unicode_ci;
use dbsdq;
alter table tt2 convert to character set utf8mb4 collate utf8mb4_unicode_ci;
alter table tt2 modify c2  varchar(10) character set utf8mb4 collate utf8mb4_unicode_ci;
```

> 说明
>
> - 修改列时，当前列中的所有行都会立即转化为新的字符集。
> - alter table 会对表加元数据锁(metadata lock), 详情请参见RDS MySQL 表上 Metadata lock 的产生和处理。

如何正确设置数据库字符集编码

请参见RDS for MySQL如何保证数据库字符编码正确。

3.10. RDS for MySQL如何修改为utf8mb4字符集

操作步骤

1. 通过客户端、命令行连接RDS MySQL。
2. 在SQL窗口使用如下命令进行修改。

```
修改库：
ALTER DATABASE <数据库名> CHARACTER SET = utf8mb4 COLLATE = utf8mb4_unicode_ci;
修改表：
ALTER TABLE <表名> CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
修改一列：
ALTER TABLE <表名> CHANGE <列名> <字段类型> CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode
_ci;
```

3.11. RDS for MySQL查看和设置时区

查看时区

1. 通过客户端、命令行连接RDS MySQL。
2. 在SQL窗口使用如下命令查看时区设置。

```
show global variables like '%time_zone%';
```

> 说明
> - system_time_zone：系统默认时区。
> - time_zone：当前使用的时区。

修改时区

注意事项

修改时区需要重启实例，建议在业务低峰期进行操作。

操作步骤

1. 登录RDS管理控制台。
2. 在页面左上角，选择实例所在地域。

3. 找到目标实例，单击实例ID。
4. 在左侧导航栏中单击**参数设置**。
5. 在**可修改参数**页签下查找到default_time_zone，单击右侧

   进行修改并单击**确定**。

> 说明 default_time_zone支持部分半时区情况，例如+5:30，请参见右侧的**可修改参数值**。

6. 在右上角单击**提交参数**，在弹出的对话框中单击**确定**，等待实例重启。

3.12. RDS for MySQL无法查询performance_schema的原因

本文介绍在RDS for MySQL实例上执行 `select * from performance_schema.threads` 返回结果为空的原因及解决办法。

performance_schema介绍

MySQL中的performance_schema主要用于收集数据库服务器性能参数，它提供以下功能：

- 提供进程等待的详细信息，包括锁、互斥变量、文件信息。
- 保存历史的事件汇总信息，为优化MySQL服务器性能提供详细的数据。
- 新增和删除监控事件点，并可以随意改变MySQL服务器的监控周期。

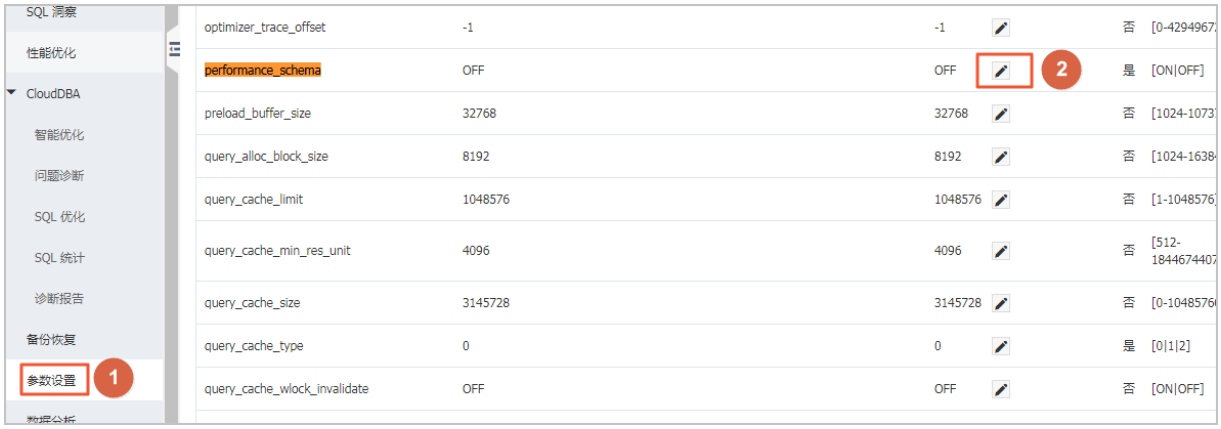
故障原因

在RDS for MySQL实例上执行 `select * from performance_schema.threads` 命令进行查询，未返回任何结果是因为开启performance_schema后会影响实例的性能，所以RDS中该功能默认是关闭的。

解决方法

针对MySQL 5.6/5.7，可以在控制台修改performance_schema参数值，详细步骤请参见设置实例参数。
MySQL 5.5暂不支持修改此参数。

> **说明** 修改performance_schema参数需要重启实例，重启前请做好业务安排，谨慎操作。



3.13. RDS for MySQL全文检索相关问题的处理

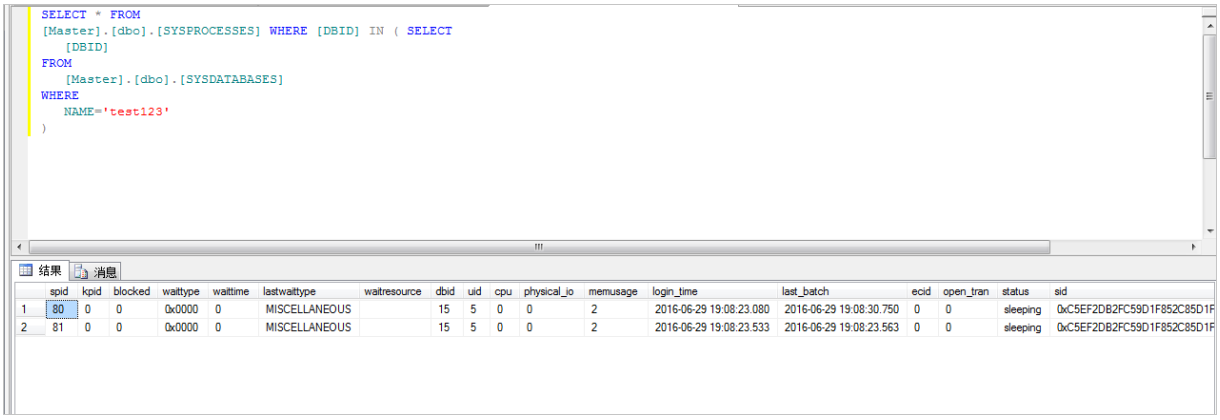
RDS for MySQL对全文检索的支持

3.14. RDS for SQL Server查看当前连接以及其执行的SQL

通过SYSPROCESSES系统视图查看连接

Master.dbo.SYSPROCESSES视图主要包括正在运行的进程的相关信息。用户可以与Master.dbo.SYSDATABASES系统视图联合查找某个数据库的所有连接。具体SQL如下：

```
SELECT * FROM
[Master].[dbo].[SYSPROCESSES] WHERE [DBID] IN ( SELECT
   [DBID]
FROM
   [Master].[dbo].[SYSDATABASES]
WHERE
   NAME='<数据库名>'
)
```



```
SELECT * FROM
[Master].[dbo].[SYSPROCESSES] WHERE [DBID] IN ( SELECT
   [DBID]
FROM
   [Master].[dbo].[SYSDATABASES]
WHERE
   NAME='test123'
)
```

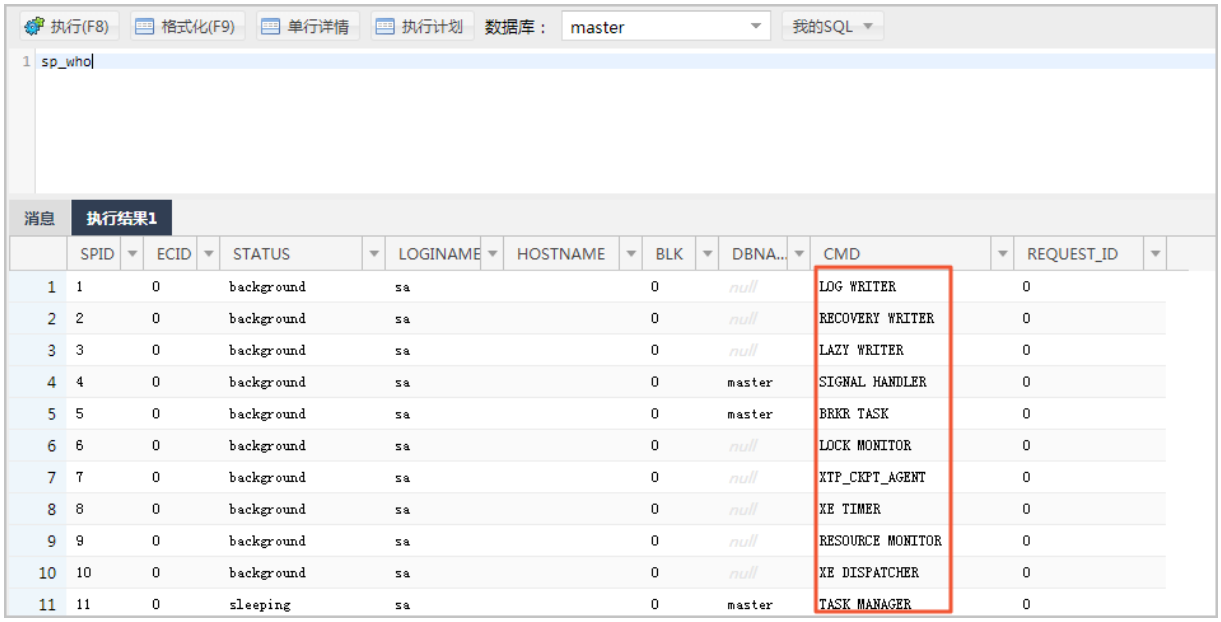
结果 消息

	spid	kpid	blocked	waittype	waittime	lastwaittype	waitresource	dbid	uid	cpu	physical_io	memusage	login_time	last_batch	ecid	open_tran	status	sid
1	80	0	0	0x0000	0	MISCELLANEOUS		15	5	0	0	2	2016-06-29 19:08:23.080	2016-06-29 19:08:30.750	0	0	sleeping	0xC5EF2DB2FC59D1F852C85D1F
2	81	0	0	0x0000	0	MISCELLANEOUS		15	5	0	0	2	2016-06-29 19:08:23.533	2016-06-29 19:08:23.563	0	0	sleeping	0xC5EF2DB2FC59D1F852C85D1F

通过sp_who查看连接会话和SQL

命令格式：

```
sp_who    --查看所有的连接会话
sp_who '<用户名>'    --查看某个用户的连接会话
```



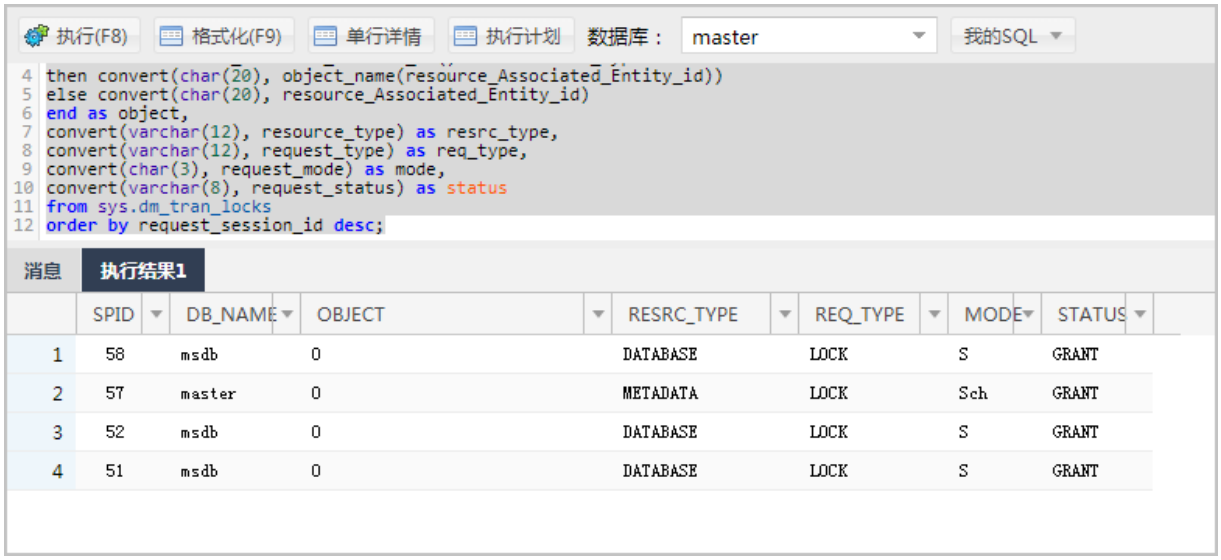
	SPID	ECID	STATUS	LOGINAME	HOSTNAME	BLK	DBNA...	CMD	REQUEST_ID
1	1	0	background	sa		0	null	LOG WRITER	0
2	2	0	background	sa		0	null	RECOVERY WRITER	0
3	3	0	background	sa		0	null	LAZY WRITER	0
4	4	0	background	sa		0	master	SIGNAL HANDLER	0
5	5	0	background	sa		0	master	BRKR TASK	0
6	6	0	background	sa		0	null	LOCK MONITOR	0
7	7	0	background	sa		0	null	XTP_CKPT_AGENT	0
8	8	0	background	sa		0	null	XE TIMER	0
9	9	0	background	sa		0	null	RESOURCE MONITOR	0
10	10	0	background	sa		0	null	XE DISPATCHER	0
11	11	0	sleeping	sa		0	master	TASK MANAGER	0

3.15. RDS for SQL Server查看锁情况

查看锁

通过sys.dm_tran_locks系统视图查看锁的情况，查询哪些数据库有锁，SQL如下：

```
select str(request_session_id, 4, 0) as spid,
convert(varchar(20), db_name(resource_database_id)) as DB_Name,
case when resource_database_id = db_id() and resource_type = 'OBJECT'
then convert(char(20), object_name(resource_Associated_Entity_id))
else convert(char(20), resource_Associated_Entity_id)
end as object,
convert(varchar(12), resource_type) as resrc_type,
convert(varchar(12), request_type) as req_type,
convert(char(3), request_mode) as mode,
convert(varchar(8), request_status) as status
from sys.dm_tran_locks
order by request_session_id desc;
```



说明 MODE列是锁的模式，介绍如下：

- 共享（S）用于不更改或不更新数据的操作（只读操作），如SELECT语句。
- 更新（U）用于可更新的资源中。防止当多个会话在读取、锁定以及随后可能进行的资源更新时发生常见形式的死锁。
- 排它（X）用于数据修改操作，例如INSERT、UPDATE或DELETE，确保不会同时同一资源进行多重更新。
- 意向锁用于建立锁的层次结构。意向锁的类型为：意向共享（IS）、意向排它（IX）以及与意向排它共享（SIX）。
- 架构锁在执行依赖于表架构的操作时使用。架构锁的类型为：架构修改（Sch-M）和架构稳定性（Sch-S）。
- 大容量更新（BU）向表中大容量复制数据并指定了TABLOCK提示时使用。

也可以查询哪些表有锁，SQL如下：

```
select request_session_id sessionid,
resource_type type,
convert(varchar(20), db_name(resource_database_id)) as db_name,
OBJECT_NAME(resource_associated_entity_id, resource_database_id) objectname,
request_mode rmode,
request_status rstatus
from sys.dm_tran_locks
where resource_type in ('OBJECT')
```

```
select request_session_id sessionid,
resource_type type,
convert(varchar(20), db_name(resource_database_id)) as db_name,
OBJECT_NAME(resource_associated_entity_id, resource_database_id) objectname,
request_mode rmode,
request_status rstatus
from sys.dm_tran_locks
where resource_type in ('OBJECT')
```

结果 消息

	sessionid	type	db_name	objectname	rmode	rstatus
1	84	OBJECT	test123	student	IX	GRANT
2	80	OBJECT	test123	student	IX	GRANT

说明

- sessionid：锁表的进程。
- objectname：被锁的表名。

3.16. RDS for SQL Server死锁处理方法

问题现象

当应用程序频繁读写某个表或者资源时，容易出现死锁现象。出现死锁时，SQL Server会选择终止其中一个事务，并且向发起该事务的客户端发送如下错误信息：

```
Error Message: Msg 1205, Level 13, State 47, Line 1Transaction (Process ID 53) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction.
```

处理方法

1. 使用客户端连接实例，请参见连接SQL Server实例。
2. 监控相关视图：
   - 循环监控sys.sysprocesses，SQL如下：

```
while 1=1
begin
select * from sys.sysprocesses where blocked<>0
waitfor delay '00:00:01'  --循环间隔时间可以自定义
end
```

	spid	kpid	blocked	waittype	waittime	lastwait...	waitresource	dbid	uid	cpu	physical_io	memusage	login_time	last_batch
1	56	4600	53	0x0003	12208	LCK_M_S	KEY: 5:72057594038845440 (98ec012aa510)	5	12	0	0	2	2016-12-01 14:54:31.157	2016-12-01 14

	spid	kpid	blocked	waittype	waittime	lastwait...	waitresource	dbid	uid	cpu	physical_io	memusage	login_time	last_batch
1	56	4600	53	0x0003	13209	LCK_M_S	KEY: 5:72057594038845440 (98ec012aa510)	5	12	0	0	2	2016-12-01 14:54:31.157	2016-12-01 14

	spid	kpid	blocked	waittype	waittime	lastwait...	waitresource	dbid	uid	cpu	physical_io	memusage	login_time	last_batch
1	53	-22908	56	0x0005	951	LCK_M_X	KEY: 5:72057594038910976 (c3a9ea6d9700)	5	12	0	0	2	2016-12-01 14:53:51.737	2016-12-01
2	56	4600	53	0x0003	14210	LCK_M_S	KEY: 5:72057594038845440 (98ec012aa510)	5	12	0	0	2	2016-12-01 14:54:31.157	2016-12-01

	spid	kpid	blocked	waittype	waittime	lastwait...	waitresource	dbid	uid	cpu	physical_io	memusage	login_time	last_batch
1	53	-22908	56	0x0005	1952	LCK_M_X	KEY: 5:72057594038910976 (c3a9ea6d9700)	5	12	0	0	2	2016-12-01 14:53:51.737	2016-12-01
2	56	4600	53	0x0003	15211	LCK_M_S	KEY: 5:72057594038845440 (98ec012aa510)	5	12	0	0	2	2016-12-01 14:54:31.157	2016-12-01

	spid	kpid	blocked	waittype	waittime	lastwait...	waitresource	dbid	uid	cpu	physical_io	memusage	login_time	last_batch
1	53	-22908	56	0x0005	2953	LCK_M_X	KEY: 5:72057594038910976 (c3a9ea6d9700)	5	12	0	0	2	2016-12-01 14:53:51.737	2016-12-01
2	56	4600	53	0x0003	16212	LCK_M_S	KEY: 5:72057594038845440 (98ec012aa510)	5	12	0	0	2	2016-12-01 14:54:31.157	2016-12-01

> **说明** 监控结果中blocked列的值为阻塞该会话的阻塞源会话ID，waitresource为被阻塞的会话等待的资源。从上述结果可以看到，spid 53和spid 56相互阻塞，形成了死锁。

- 循环监控sys.dm_tran_locks，sys.dm_os_waiting_tasks等视图，SQL如下：

```
while 1=1
    Begin
    SELECT
    db.name DBName,
    tl.request_session_id,  --被阻塞session
    wt.blocking_session_id,  --阻塞头session
    OBJECT_NAME(p.OBJECT_ID) BlockedObjectName,
    tl.resource_type,
    h1.TEXT AS RequestingText,
    h2.TEXT AS BlockingText,
    tl.request_mode
    FROM sys.dm_tran_locks AS tl
    INNER JOIN sys.databases db ON db.database_id = tl.resource_database_id
    INNER JOIN sys.dm_os_waiting_tasks AS wt ON tl.lock_owner_address = wt.resource_a
ddress
    INNER JOIN sys.partitions AS p ON p.hobt_id = tl.resource_associated_entity_id
    INNER JOIN sys.dm_exec_connections ec1 ON ec1.session_id = tl.request_session_id
    INNER JOIN sys.dm_exec_connections ec2 ON ec2.session_id = wt.blocking_session_id
    CROSS APPLY sys.dm_exec_sql_text(ec1.most_recent_sql_handle) AS h1
    CROSS APPLY sys.dm_exec_sql_text(ec2.most_recent_sql_handle) AS h2
    waitfor delay '00:00:01' --循环间隔时间可以自定义
    End
```

DBName	request_session_id	blocking_session_id	BlockedObjectName	resource_type	RequestingText	BlockingText	request_mod
[illegible]	62	56	Tbl1	KEY	(@1 int,@2 int)INSERT INTO [dbo].[Tbl2]([id],[c...	BEGIN TRAN INSERT into dbo.Tbl1 (id, col) VALUE...	S

DBName	request_sessio...	blocking_sessio...	BlockedObject...	resource_...	RequestingText	BlockingText	request_mode
[illegible]	56	62	Tbl2	KEY	(@1 int,@2 int)INSERT INTO [dbo].[Tbl2]([id],[c...	(@1 int,@2 int)INSERT INTO [dbo].[Tbl2]([id],[c...	X
[illegible]	62	56	Tbl1	KEY	(@1 int,@2 int)INSERT INTO [dbo].[Tbl2]([id],[c...	(@1 int,@2 int)INSERT INTO [dbo].[Tbl2]([id],[c...	S

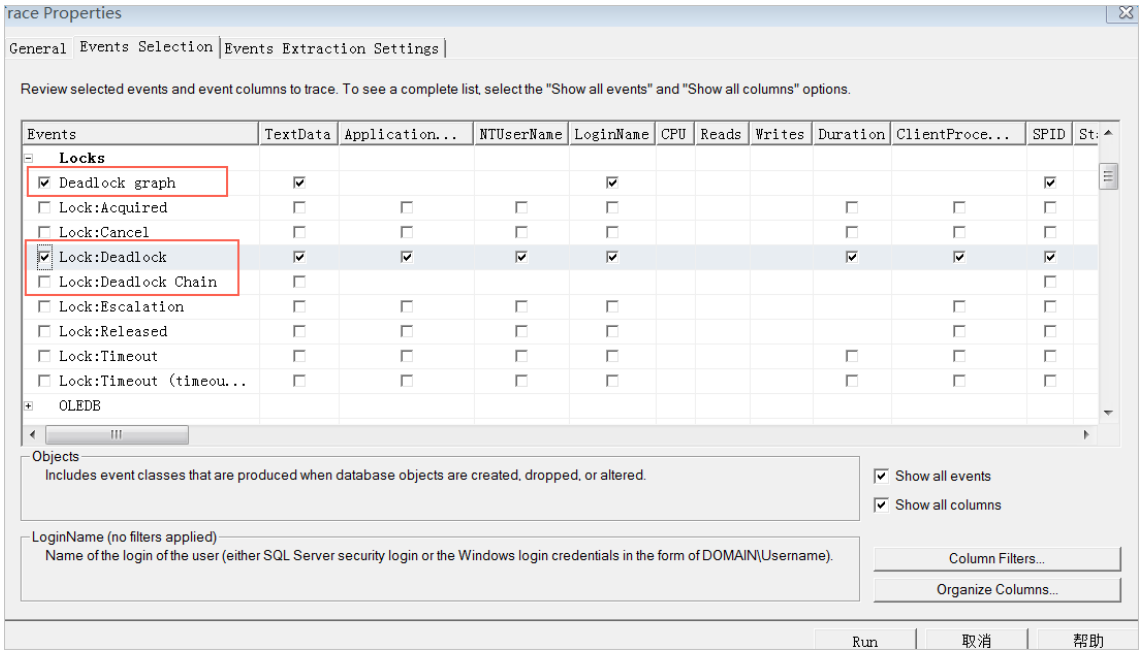
DBName	request_sessio...	blocking_sessio...	BlockedObject...	resource_...	RequestingText	BlockingText	request_mode
[illegible]	56	62	Tbl2	KEY	(@1 int,@2 int)INSERT INTO [dbo].[Tbl2]([id],[c...	(@1 int,@2 int)INSERT INTO [dbo].[Tbl2]([id],[c...	X
[illegible]	62	56	Tbl1	KEY	(@1 int,@2 int)INSERT INTO [dbo].[Tbl2]([id],[c...	(@1 int,@2 int)INSERT INTO [dbo].[Tbl2]([id],[c...	S

DBName	request_sessio...	blocking_sessio...	BlockedObject...	resource_...	RequestingText	BlockingText	request_mode
[illegible]	56	62	Tbl2	KEY	(@1 int,@2 int)INSERT INTO [dbo].[Tbl2]([id],[c...	(@1 int,@2 int)INSERT INTO [dbo].[Tbl2]([id],[c...	X
[illegible]	62	56	Tbl1	KEY	(@1 int,@2 int)INSERT INTO [dbo].[Tbl2]([id],[c...	(@1 int,@2 int)INSERT INTO [dbo].[Tbl2]([id],[c...	S

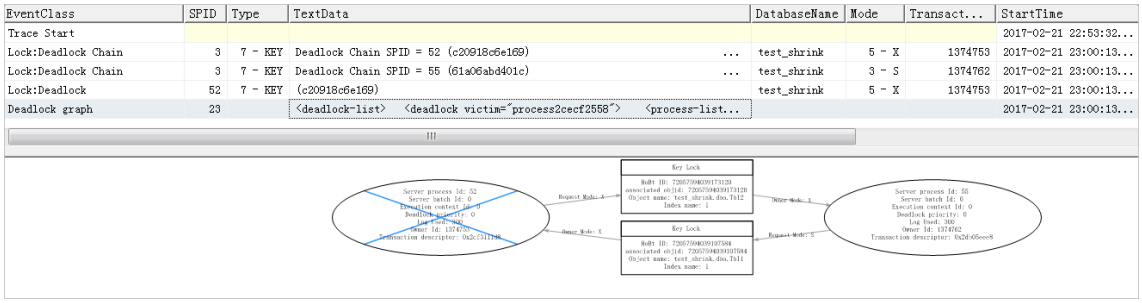
DBName	request_sessio...	blocking_sessio...	BlockedObject...	resource_...	RequestingText	BlockingText	request_mode
[illegible]	56	62	Tbl2	KEY	(@1 int,@2 int)INSERT INTO [dbo].[Tbl2]([id],[c...	(@1 int,@2 int)INSERT INTO [dbo].[Tbl2]([id],[c...	X
[illegible]	62	56	Tbl1	KEY	(@1 int,@2 int)INSERT INTO [dbo].[Tbl2]([id],[c...	(@1 int,@2 int)INSERT INTO [dbo].[Tbl2]([id],[c...	S

> 说明
>
> - DBName：request_session_id操作的数据库。
> - request_session_id：当前请求的会话 ID，即被阻塞的会话。
> - blocking_session_id：阻塞源会话ID。
> - BlockedObjectName：被阻塞的会话操作的对象。
> - resource_type：等待的资源类型。
> - RequestingText：当前会话执行的语句，即被阻塞的语句。
> - BlockingText：阻塞源会话执行的语句。
> - request_mode：当前会话请求的锁模式。

- 如果您使用的是RDS for SQL Server 2012，您还可以使用SQL Server Profiler来监控和抓取死锁图谱。



抓取的死锁图谱如下。



3. 调优：
   - 关闭阻塞源会话，可以帮助快速解除阻塞。
   - 查看是否有长时间未提交的事务，及时提交事务。
   - 使用with（nolock）查询。

> **说明** 如果有S锁参与死锁，并且应用允许脏读，可以使用with nolock，让select语句避免申请锁，从而避免死锁，例如 `select * from table with(nolock)` 。

- 检查应用程序逻辑，按顺序访问某个资源。

3.17. RDS for SQL Server CPU使用率高问题排查

RDS for SQL Server使用过程中，会遇到CPU使用率过高甚至达到100%的情况。本文将介绍造成该状况的常见原因以及解决方法。

常见原因

RDS for SQL Server CPU使用率高的因素有很多，其中最常见的是应用的负载高、查询语句的成本高，或者是实例的并行度设置不合理。

实例的并行度设置不合理

问题排查

多线程并行处理任务时，由于每个线程处理的数据量不一致，会出现CXPACKET等待情况，CXPACKET等待发生比较多的话，造成CPU使用率高。可以通过SQL Server Management Studio的活动监视器或者下面语句（多次执行取差值），监控是否存在大量CXPACKET等待。

> **说明** CXPACKET指线程正在等待彼此完成并行处理。当SQL Server发现一条指令复杂时，会决定用多个线程并行来执行，由于某些并行线程已完成工作，在等待其它并行线程来同步，这种等待就叫CXPACKET。

```
WITH [Waits] AS
    (SELECT
        [wait_type],
        [wait_time_ms] / 1000.0 AS [WaitS],
        ([wait_time_ms] - [signal_wait_time_ms]) / 1000.0 AS [ResourceS],
        [signal_wait_time_ms] / 1000.0 AS [SignalS],
        [waiting_tasks_count] AS [WaitCount],
       100.0 * [wait_time_ms] / SUM ([wait_time_ms]) OVER() AS [Percentage],
        ROW_NUMBER() OVER(ORDER BY [wait_time_ms] DESC) AS [RowNum]
    FROM sys.dm_os_wait_stats
    WHERE [wait_type] NOT IN (
        N'BROKER_EVENTHANDLER', N'BROKER_RECEIVE_WAITFOR',
        N'BROKER_TASK_STOP', N'BROKER_TO_FLUSH',
        N'BROKER_TRANSMITTER', N'CHECKPOINT_QUEUE',
        N'CHKPT', N'CLR_AUTO_EVENT',
        N'CLR_MANUAL_EVENT', N'CLR_SEMAPHORE',
        -- Maybe uncomment these four if you have mirroring issues
        N'DBMIRROR_DBM_EVENT', N'DBMIRROR_EVENTS_QUEUE',
        N'DBMIRROR_WORKER_QUEUE', N'DBMIRRORING_CMD',
        N'DIRTY_PAGE_POLL', N'DISPATCHER_QUEUE_SEMAPHORE',
        N'EXECSYNC', N'FSAGENT',
        N'FT_IFTS_SCHEDULER_IDLE_WAIT', N'FT_IFTSHC_MUTEX',
        -- Maybe uncomment these six if you have AG issues
```

```
        N'HADR_CLUSAPI_CALL', N'HADR_FILESTREAM_IOMGR_IOCOMPLETION',
        N'HADR_LOGCAPTURE_WAIT', N'HADR_NOTIFICATION_DEQUEUE',
        N'HADR_TIMER_TASK', N'HADR_WORK_QUEUE',
        N'KSOURCE_WAKEUP', N'LAZYWRITER_SLEEP',
        N'LOGMGR_QUEUE', N'MEMORY_ALLOCATION_EXT',
        N'ONDEMAND_TASK_QUEUE',
        N'PREEMPTIVE_XE_GETTARGETSTATE',
        N'PWAIT_ALL_COMPONENTS_INITIALIZED',
        N'PWAIT_DIRECTLOGCONSUMER_GETNEXT',
        N'QDS_PERSIST_TASK_MAIN_LOOP_SLEEP', N'QDS_ASYNC_QUEUE',
        N'QDS_CLEANUP_STALE_QUERIES_TASK_MAIN_LOOP_SLEEP',
        N'QDS_SHUTDOWN_QUEUE', N'REDO_THREAD_PENDING_WORK',
        N'REQUEST_FOR_DEADLOCK_SEARCH', N'RESOURCE_QUEUE',
        N'SERVER_IDLE_CHECK', N'SLEEP_BPOOL_FLUSH',
        N'SLEEP_DBSTARTUP', N'SLEEP_DCOMSTARTUP',
        N'SLEEP_MASTERDBREADY', N'SLEEP_MASTERMDREADY',
        N'SLEEP_MASTERUPGRADED', N'SLEEP_MSDBSTARTUP',
        N'SLEEP_SYSTEMTASK', N'SLEEP_TASK',
        N'SLEEP_TEMPDBSTARTUP', N'SNI_HTTP_ACCEPT',
        N'SP_SERVER_DIAGNOSTICS_SLEEP', N'SQLTRACE_BUFFER_FLUSH',
        N'SQLTRACE_INCREMENTAL_FLUSH_SLEEP',
        N'SQLTRACE_WAIT_ENTRIES', N'WAIT_FOR_RESULTS',
        N'WAITFOR', N'WAITFOR_TASKSHUTDOWN',
        N'WAIT_XTP_RECOVERY',
        N'WAIT_XTP_HOST_WAIT', N'WAIT_XTP_OFFLINE_CKPT_NEW_LOG',
        N'WAIT_XTP_CKPT_CLOSE', N'XE_DISPATCHER_JOIN',
        N'XE_DISPATCHER_WAIT', N'XE_TIMER_EVENT')
    AND [waiting_tasks_count] > 0
    )
SELECT
    MAX ([W1].[wait_type]) AS [WaitType],
    CAST (MAX ([W1].[WaitS]) AS DECIMAL (16,2)) AS [Wait_S],
    CAST (MAX ([W1].[ResourceS]) AS DECIMAL (16,2)) AS [Resource_S],
    CAST (MAX ([W1].[SignalS]) AS DECIMAL (16,2)) AS [Signal_S],
    MAX ([W1].[WaitCount]) AS [WaitCount],
    CAST (MAX ([W1].[Percentage]) AS DECIMAL (5,2)) AS [Percentage],
    CAST ((MAX ([W1].[WaitS]) / MAX ([W1].[WaitCount])) AS DECIMAL (16,4)) AS [AvgWait_S],
    CAST ((MAX ([W1].[ResourceS]) / MAX ([W1].[WaitCount])) AS DECIMAL (16,4)) AS [AvgRes_S
],
    CAST ((MAX ([W1].[SignalS]) / MAX ([W1].[WaitCount])) AS DECIMAL (16,4)) AS [AvgSig_S]
FROM [Waits] AS [W1]
INNER JOIN [Waits] AS [W2]
    ON [W2].[RowNum] <= [W1].[RowNum]
GROUP BY [W1].[RowNum]
HAVING SUM ([W2].[Percentage]) - MAX( [W1].[Percentage] ) < 95; -- percentage threshold
GO
```

解决方案

- **从语句级别进行设置**

  i. 通过查询语句寻找消耗CPU的语句，SQL如下：

```
SELECT TOP 50
[Avg. MultiCore/CPU time(sec)] = qs.total_worker_time / 1000000 / qs.execution_count,
[Total MultiCore/CPU time(sec)] = qs.total_worker_time / 1000000,
[Avg. Elapsed Time(sec)] = qs.total_elapsed_time / 1000000 / qs.execution_count,
[Total Elapsed Time(sec)] = qs.total_elapsed_time / 1000000,
qs.execution_count,
[Avg. I/O] = (total_logical_reads + total_logical_writes) / qs.execution_count,
[Total I/O] = total_logical_reads + total_logical_writes,
Query = SUBSTRING(qt.[text], (qs.statement_start_offset / 2) + 1,
(
(
CASE qs.statement_end_offset
WHEN -1 THEN DATALENGTH(qt.[text])
ELSE qs.statement_end_offset
END - qs.statement_start_offset
) / 2
) + 1
),
Batch = qt.[text],
[DB] = DB_NAME(qt.[dbid]),
qs.last_execution_time,
qp.query_plan
FROM sys.dm_exec_query_stats AS qs
CROSS APPLY sys.dm_exec_sql_text(qs.[sql_handle]) AS qt
CROSS APPLY sys.dm_exec_query_plan(qs.plan_handle) AS qp
where qs.execution_count > 5        --more than 5 occurences
ORDER BY [Total MultiCore/CPU time(sec)] DESC
```

ii. 对于RDS for SQL Server 2008 R2实例，可以在控制台查看慢日志统计，查找消耗CPU的语句。

iii. 找到语句之后，查看其执行计划，对于并行度较高的语句，可以在语句级别使用hint查询，限制语句并行度。示例如下：

```
SELECT column1,column2
    FROM table1 o INNER JOIN table2 d ON (o.d_id = d.d_id)
    OPTION (maxdop 1)
```

- **从实例级别进行设置**

  i. 查看当前实例的MAXDOP值，SQL如下：

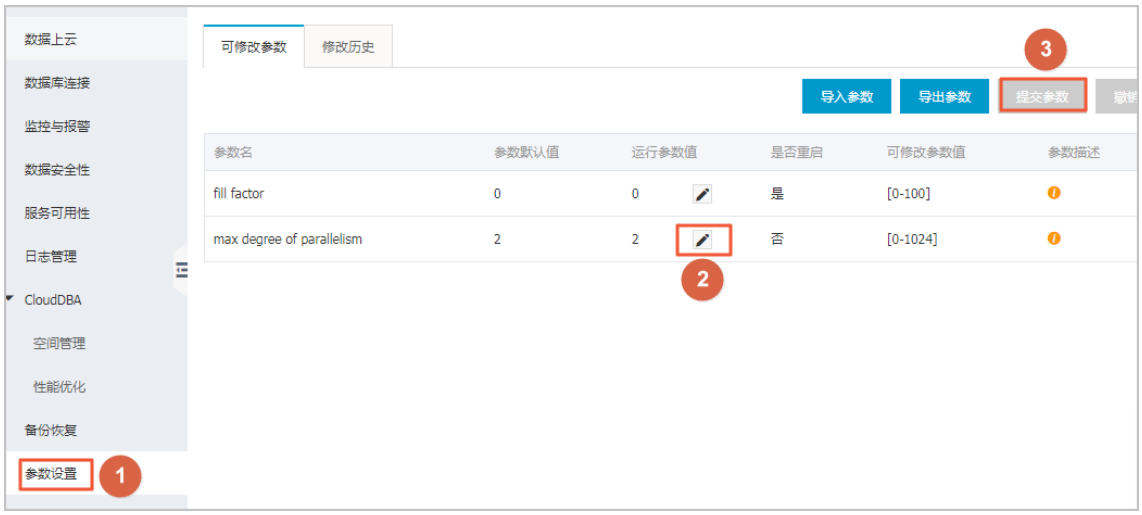
```
select * from sys.configurations where name like '%max%'
```

	configuratio...	name	value	minimum	maximum	value_in...	description
1	503	max worker threads	0	128	65535	0	Maximum worker threads
2	1536	max text repl size (B)	65536	-1	2147483647	65536	Maximum size of a text field in replication.
3	1539	max degree of parallelism	0	0	32767	0	maximum degree of parallelism
4	1544	max server memory (MB)	1024	128	2147483647	1024	Maximum size of server memory (MB)
5	1563	max full-text crawl range	4	0	256	4	Maximum crawl ranges allowed in full-text inde...
6	1565	ft notify bandwidth (max)	100	0	32767	100	Max number of full-text notifications buffers
7	1567	ft crawl bandwidth (max)	100	0	32767	100	Max number of full-text crawl buffers

ii. 在实例级别设置该参数，对所有查询均生效，SQL如下：

```
sp_rds_configure 'max degree of parallelism', 1;
GO
```

对于RDS for SQL Server 2008 R2实例，可以在RDS管理控制台的**参数设置**中进行手动设置，需提交参数生效。



应用负载高

现象

没有出现慢查询（或者慢查询不是问题主要原因），QPS和CPU使用率曲线变化吻合。常见于应用优化过的在线事务交易系统（比如订单系统）、高读取率的热门Web网站应用等。

特征

实例的QPS高，查询比较简单、执行效率高、优化余地小。

解决方案

建议从应用架构、实例规格等方面来解决：

- 升级实例规格，增加CPU资源。
- 尽量优化查询，减少查询的执行成本（逻辑IO，执行需要访问的表数据行数），提高应用可扩展性。

查询语句的读写过高

现象

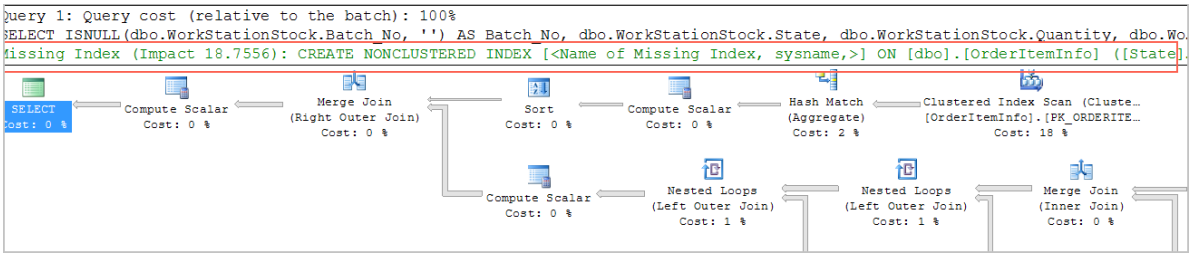
存在慢查询，QPS和CPU使用率曲线变化不吻合，检查消耗CPU的语句，存在I/O较大的语句。

特征

实例的QPS不高；查询执行效率低、执行需要扫描大量表中数据、优化余地大。

解决方案

- 对于大表查询，检查是否有合适的索引。检查实际执行计划，针对全表扫描操作进行优化，执行计划中也会给出缺失索引的建议。



- 通过CloudDBA检查性能问题。

避免出现CPU使用100%的一般原则

- 设置CPU使用率告警，实例CPU使用率保证一定的冗余度。
- 应用设计和开发过程中，要考虑查询的优化，遵守SQL优化的一般优化原则，降低查询的逻辑 I/O，提高应用可扩展性。
- 新功能、新模块上线前，要使用生产环境数据进行压力测试。
- 经常使用CloudDBA查看实例各项性能，及时发现问题。

3.18. RDS for SQL Server查看常用参数值

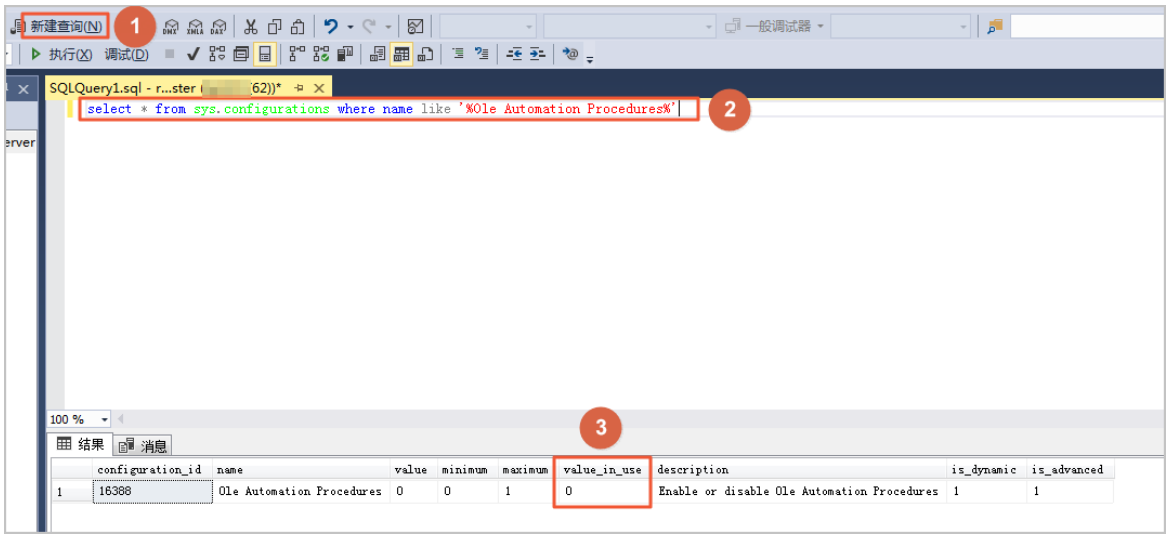
SQL Server中有两个系统参数表，分别是服务器参数表sys.configurations和数据库参数表sys.databases。

本文以查看参数Ole Automation Procedures状态为例进行步骤说明。

操作步骤

1. 使用客户端连接实例，请参见连接SQL Server实例。
2. 单击**新建查询**，在SQL窗口执行如下命令：

```
select * from sys.configurations where name like '%Ole Automation Procedures%'
```



> 说明 value_in_use=0，表示当前为关闭状态.。

3.19. RDS for SQL Server常用视图

本文将介绍RDS for SQL Server日常使用和维护时，常用的系统视图及相关查询语句。

前提条件

使用客户端连接实例，请参见连接SQL Server实例。

查询语句

- **查看系统参数配置**

```
use master
select * from sys.configurations
```

	configuratio...	name	value	minimum	maximum	value_in...	description	is_dynamic	is_advanced
1	101	recovery interval (min)	0	0	32767	0	Maximum recovery interval in minutes	1	1
2	102	allow updates	0	0	1	0	Allow updates to system tables	1	0
3	103	user connections	32760	0	32767	32760	Number of user connections allowed	0	1
4	106	locks	0	5000	2147483647	0	Number of locks for all users	0	1
5	107	open objects	0	0	2147483647	0	Number of open database objects	0	1
6	109	fill factor (%)	0	0	100	0	Default fill factor percentage	0	1
7	114	disallow results from triggers	0	0	1	0	Disallow returning results from triggers	1	1
8	115	nested triggers	1	0	1	1	Allow triggers to be invoked within triggers	1	0
9	116	server trigger recursion	1	0	1	1	Allow recursion for server level triggers	1	0
10	117	remote access	1	0	1	1	Allow remote access	0	0
11	124	default language	0	0	9999	0	default language	1	0
12	400	cross db ownership chaining	0	0	1	0	Allow cross db ownership chaining	1	0
13	503	max worker threads	0	128	32767	0	Maximum worker threads	0	1
14	505	network packet size (B)	4096	512	32767	4096	Network packet size	1	1
15	518	show advanced options	0	0	1	0	show advanced options	1	0
16	542	remote proc trans	0	0	1	0	Create DTC transaction for remote procedures	1	0
17	544	c2 audit mode	0	0	1	0	c2 audit mode	0	1
18	1126	default full-text language	1033	0	2147483647	1033	default full-text language	1	1
19	1127	two digit year cutoff	2049	1753	9999	2049	two digit year cutoff	1	1
20	1505	index create memory (KB)	0	704	2147483647	0	Memory for index create sorts (kBytes)	1	1
21	1517	priority boost	0	0	1	0	Priority boost	0	1
22	1519	remote login timeout (s)	20	0	2147483647	20	remote login timeout	1	0
23	1520	remote query timeout (s)	600	0	2147483647	600	remote query timeout	1	0

> **说明** 参数详细解释请参见sys.configurations。

- **查看数据库的文件相关信息**

```
use <数据库名>
select * from sys.sysfiless
```

	fileid	groupid	size	maxsize	growth	status	perf	name	filename
1	1	1	512	-1	10	1048578	0	master	d:\ms3003\MSSQL10_50.MS3003\MSSQL\DATA\master.mdf
2	2	0	160	-1	10	1048642	0	mastlog	d:\ms3003\MSSQL10_50.MS3003\MSSQL\DATA\mastlog.ldf

查看数据库文件大小

```
select name, convert(float,size) * (8192.0/1024.0)/1024 AS Size_MB,*  from <数据库名>.dbo.
sysfiles
```

	name	Size_MB	fileid	size	maxsize	growth	filename	groupid	status	perf	name
1	data1	512	1	65536	1310720	65536	d:\MS3003\Data\muyuan\data1.mdf	1	2	0	data1
2	log	512	2	65536	1310720	65536	d:\MS3003\Log\muyuan\log.ldf	0	66	0	log

查看数据库文件的I/O统计信息

```
select * from sys.dm_io_virtual_file_stats(DB_ID('<数据库名>'),<file_id>)
```

	base_id	file_id	size_on_disk_bytes	sample_ms	num_of_reads	num_of_bytes_read	io_stall_read_ms	num_of_writes	num_of_bytes_written	io_stall_write_ms
1		1	536870912	-1597559356	358	73924608	978	649	11051008	19

- **查看实例中的所有未提交的事务及其执行的语句**

```
SELECT DB_NAME(dbid) AS DBNAME,
(SELECT text FROM sys.dm_exec_sql_text(sql_handle)) AS SQLSTATEMENT
FROM master..sysprocesses WHERE open_tran > 0
```

	DBNAME	SQLSTATEMENT
1		begin tran select * from sys.sysfilegroups
2		begin tran select * from sys.sysfiles

- **查看数据和索引的碎片**

`DBCC SHOWCONTIG` 显示了指定表或者视图的数据以及索引的碎片情况，详细解释请参考DBCC SHOWCONTIG。

```
DBCC SHOWCONTIG
```

```
DBCC SHOWCONTIG scanning 'test' table...
Table: 'test' (21575115); index ID: 0, database ID: 6
TABLE level scan performed.
- Pages Scanned.....: 1
- Extents Scanned.....: 1
- Extent Switches.....: 0
- Avg. Pages per Extent.....: 1.0
- Scan Density [Best Count:Actual Count].....: 100.00% [1:1]
- Extent Scan Fragmentation .....: 0.00%
- Avg. Bytes Free per Page.....: 8030.0
- Avg. Page Density (full).....: 0.79%
DBCC SHOWCONTIG scanning 'tUser' table...
Table: 'tUser' (2137058649); index ID: 0, database ID: 6
TABLE level scan performed.
- Pages Scanned.....: 1
- Extents Scanned.....: 1
- Extent Switches.....: 0
- Avg. Pages per Extent.....: 1.0
- Scan Density [Best Count:Actual Count].....: 100.00% [1:1]
- Extent Scan Fragmentation .....: 0.00%
- Avg. Bytes Free per Page.....: 7970.0
- Avg. Page Density (full).....: 1.53%
DBCC execution completed. If DBCC printed error messages, contact your system administrator.
```

查看数据库中的索引碎片

```
select * from sys.dm_db_index_physical_stats(DB_ID(N'<数据库名>'),NULL,NULL,NULL,DEFAULT)
```

	database_id	object_id	index_id	index_type_desc	alloc_unit_type_...	index_depth	index_level	avg_fragmentation_in_percent	avg_fragment_size_in_pages
1	5	181575685	1	CLUSTERED INDEX	IN_ROW_DATA	1	0	0	1
2	5	373576369	0	HEAP	IN_ROW_DATA	0	0	0	0
3	5	389576426	0	HEAP	IN_ROW_DATA	1	0	25	7.33333333333333
4	5	501576825	0	HEAP	IN_ROW_DATA	0	0	0	0
5	5	517576882	0	HEAP	IN_ROW_DATA	0	0	0	0
6	5	597577167	0	HEAP	IN_ROW_DATA	0	0	0	0

- **查看近期执行语句**

```
SELECT
    p.spid, p.status, p.hostname, p.loginame, p.cpu, r.start_time, r.command,
    p.program_name, text
FROM
    sys.dm_exec_requests AS r,
    master.dbo.sysprocesses AS p
    CROSS APPLY sys.dm_exec_sql_text(p.sql_handle)
WHERE
    p.status NOT IN ('sleeping', 'background')
AND r.session_id = p.spid
```

spid	status	hostname	loginame	cpu	start_time	command	text	program_name
60	runnable	[redacted]	[redacted]	47	2017-03-07 10:14:17.710	SELECT	SELECT p.spid, p.status, p.hostname, p.log...	Microsoft SQL Server Management Stu

3.20. RDS for SQL Server字符集修改

RDS for SQL Server控制台支持的字符集

- Chinese_PRC_CI_AS
- Chinese_PRC_CS_AS
- SQL_Latin1_General_CP1_CI_AS
- SQL_Latin1_General_CP1_CS_AS
- Chinese_PRC_BIN

> **说明** 后缀说明：
>
> - _CI：不分区大小写。
> - _CS：区分大小写。
> - _BIN：按二进制排序，区分大小写。

修改表的字符集

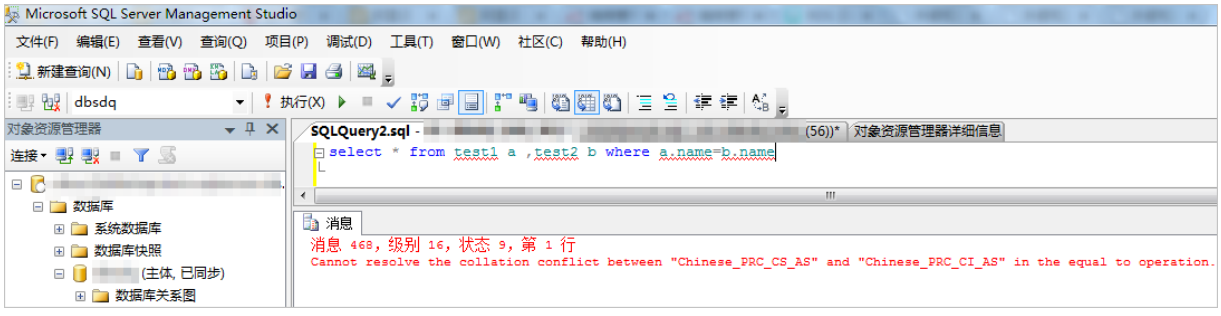
RDS for SQL Server不支持修改数据库级别字符集，只支持修改表中列的字符集。SQL如下：

```
alter table <表名> alter column <列名> <列数据类型> collate <字符集>
```

示例

```
alter table test2 alter column name varchar(10) collate Chinese_PRC_CI_AS
```

字符集导致的查询错误



原因

由于列的字符集不同，导致查询失败。

解决方案

可以修改表的字符集保持一致，或者查询时指定字符集，SQL如下：

```
select * from test1 a ,test2 b where a.name=b.name collate Chinese_PRC_CI_AS
```

3.21. PostgreSQL/PPAS CPU使用率高的原因及解决办法

RDS for PostgreSQL/PPAS使用过程中，可能会遇到CPU使用率过高甚至达到100%的情况。本文将介绍造成该状况的常见原因以及解决方法，并通过CPU使用率为100%的典型场景，来分析引起该状况的原因及其相应的解决方案。

CPU利用率到达100%，首先检查是不是业务高峰活跃连接陡增，而数据库预留的资源不足。我们需要查看问题发生时，活跃的连接数是否比平时多很多。对于RDS for PostgreSQL/PPAS，数据库上的连接数变化，可以从控制台的监控信息中看到。而当前活跃的连接数，可以直接连接数据库，使用下列查询语句得到。

```
select count( * ) from pg_stat_activity where state not like '%idle';
```

追踪慢SQL

如果活跃连接数的变化处于正常范围，则可能是当时有性能很差的SQL被大量执行。由于RDS有慢SQL日志，我们可以通过这个日志，定位到当时比较耗时的SQL来进一步做分析。但通常问题发生时，整个系统都处于停滞状态，所有SQL都慢下来，当时记录的慢SQL可能非常多，并不容易找到目标。这里我们介绍几种追查慢SQL的方法。

1. 第一种方法是使用pg_stat_statements插件定位慢SQL，仅适用于PostgreSQL，步骤如下：
   i. 如果没有pg_stat_statements插件，需要先手动创建。我们要利用插件和数据库系统里面的计数信息（如SQL执行时间累积等），而这些信息是不断累积的，包含了历史信息。为了更方便的排查当前的CPU过高问题，我们要先使用如下命令重置计数器。

   ```
   create extension pg_stat_statements;
   select pg_stat_reset();
   select pg_stat_statements_reset();
   ```

   ii. 等待一段时间（例如1分钟），使计数器积累足够的信息。
   iii. 使用如下命令查询最耗时的SQL（一般就是导致问题的直接原因）。

   ```
   select * from pg_stat_statements order by total_time desc limit 5;
   ```

   iv. 使用如下命令查询读取Buffer次数最多的SQL，这些SQL可能由于所查询的数据没有索引，而导致了过多的Buffer读，也同时大量消耗了CPU。

   ```
   select * from pg_stat_statements order by shared_blks_hit+shared_blks_read desc lim
   it 5;
   ```

2. 第二种方法是直接通过pg_stat_activity视图，利用下面的查询命令，查看当前长时间执行，一直不结束的SQL。这些SQL也可能造成CPU过高。

```
 select datname, usename, client_addr, application_name, state, backend_start, xact_sta
rt, xact_stay, query_start, query_stay, replace(query, chr(10), ' ') as query from (sel
ect pgsa.datname as datname, pgsa.usename as usename, pgsa.client_addr client_addr, pgs
a.application_name as application_name, pgsa.state as state, pgsa.backend_start as back
end_start, pgsa.xact_start as xact_start, extract(epoch from (now() - pgsa.xact_start))
as xact_stay, pgsa.query_start as query_start, extract(epoch from (now() - pgsa.query_s
tart)) as query_stay , pgsa.query as query from pg_stat_activity as pgsa where pgsa.sta
te != 'idle' and pgsa.state != 'idle in transaction' and pgsa.state != 'idle in transac
tion (aborted)') idleconnections order by query_stay desc limit 5;
```

3. 第3种方法是从数据表上表扫描（Table Scan）的信息开始查起，查找缺失索引的表。数据表如果缺失索引，大部分热数据又都在内存时（例如内存8G，热数据6G），此时数据库只能使用表扫描，并需要处理已在内存中的大量无关记录，导致耗费大量CPU。特别是对于表记录数超过100的表，一次表扫描占用大量CPU（基本把一个CPU占满）和多个连接并发（例如上百连接）。
   i. 使用如下命令，查出使用表扫描最多的表。

   ```
   select * from pg_stat_user_tables where n_live_tup > 100000 and seq_scan > 0 order
   by seq_tup_read desc limit 10;
   ```

   ii. 使用如下命令查询当前正在运行的访问到上述表的慢查询。

   ```
   select * from pg_stat_activity where query ilike '%<table name>%' and query_start -
   now() > interval '10 seconds';
   ```

   > **说明** 也可以通过pg_stat_statements插件定位涉及到这些表的查询。
   >
   > ```
   > select * from pg_stat_statements where query ilike '%<table>%'order by shared_b
   > lks_hit+shared_blks_read desc limit 3;
   > ```

处理慢SQL

对于上面的方法查出来的慢SQL，首先需要做的是Kill掉他们，使业务先恢复。

```
select pg_cancel_backend(pid) from pg_stat_activity where  query like '%<query text>%' and
pid != pg_backend_pid();
select pg_terminate_backend(pid) from pg_stat_activity where  query like '%<query text>%' a
nd pid != pg_backend_pid();
```

如果这些SQL确实是业务上必需的，则需要对他们做如下优化：

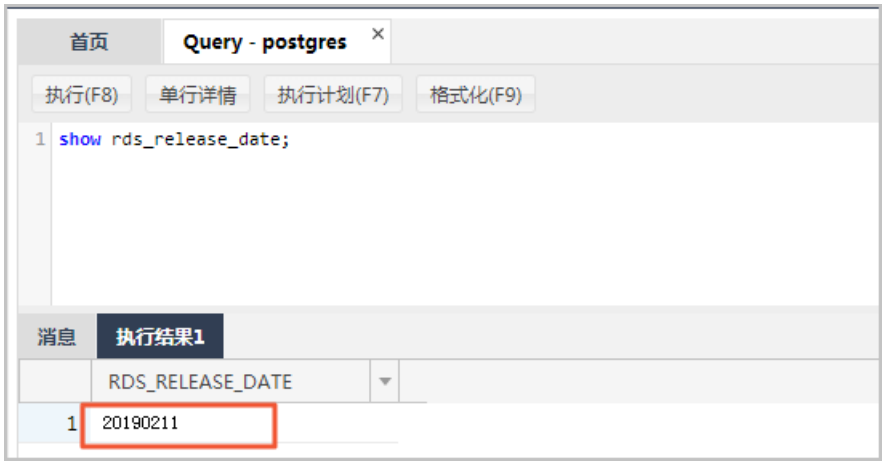
1. 对查询涉及的表，执行 `ANALYZE <table>` 或 `VACUUM ANZLYZE <table>`，更新表的统计信息，使查询计划更准确。为避免对业务影响，最好在业务低峰执行。
2. 执行 `explain <query text>` 或 `explain (buffers true, analyze true, verbose true) <query text>` 命令，查看SQL的执行计划（前者不会实际执行SQL，后者会实际执行而且能得到详细的执行信息），对其中的Table Scan涉及的表，建立索引。
3. 重新编写SQL，去除掉不必要的子查询、改写UNION ALL、使用JOIN CLAUSE固定连接顺序等，都是进一步深度优化SQL的手段，这里不再深入说明。

3.22. RDS for PostgreSQL查看数据库内核小版本

操作步骤

1. 连接PostgreSQL实例。
2. 在SQL窗口执行如下命令。

```
show rds_release_date;
```



3.23. RDS访问实例诊断报告

RDS实例在DMS和CloudDBA中均提供了诊断报告的功能。

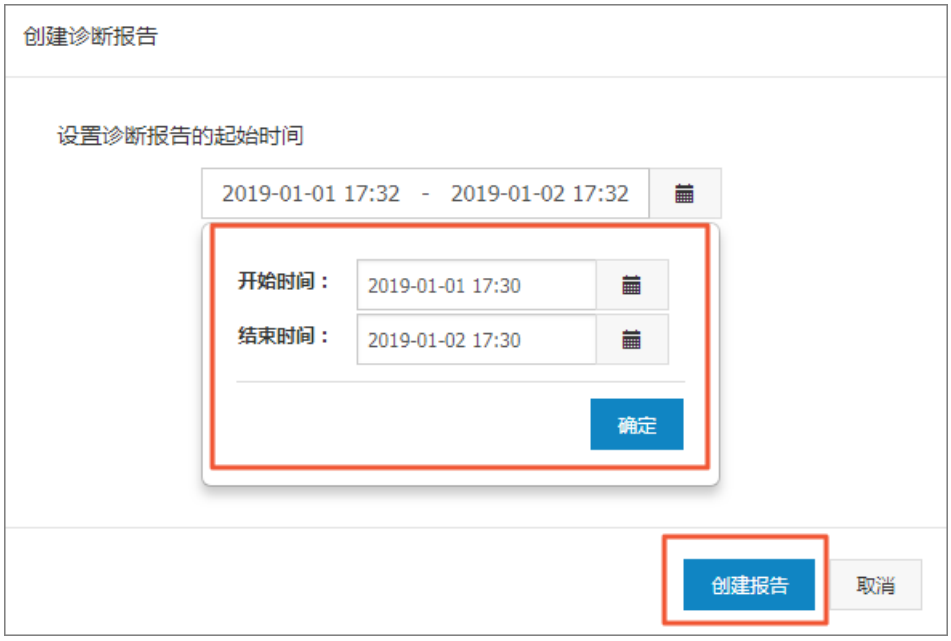
通过CloudDBA访问实例诊断报告

> **说明** 目前仅如下版本实例支持此功能：
> - MySQL 5.5高可用版
> - MySQL 5.6版
> - MySQL 5.7 高可用版
> - PostgreSQL 10.0版
> - PPAS 10.0版

1. 登录RDS管理控制台。
2. 选择目标实例所在地域。
3. 单击目标实例ID，进入**基本信息**页面。
4. 在左侧导航栏中，选择**CloudDBA > 诊断报告**，进入**诊断报告**页面。
5. 点击**创建诊断报告**，如下图所示。

6. 选择诊断数据的起始时间，点击**确定**保存后单击**创建报告**，如下图所示。

> **注意** PostgreSQL 10.0版本和PPAS 10.0版本无法选择起始时间，默认生成当前时间的诊断报告。



7. 诊断完成后，可在列表中查看诊断得分并进行查看报告或删除报告的操作，如下图所示。

> **说明** 诊断报告列表可以保存最近30天内的诊断记录，超时数据将会被自动删除。

具体操作步骤如下：

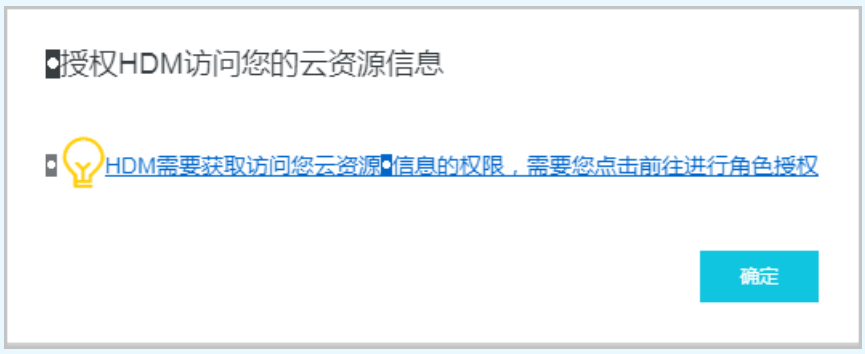
- 查看诊断报告：单击**查看报告**。
- 删除诊断报告：
  a. 单击**删除**。
  b. 在弹出的确认框中，单击**确认**。

通过DMS访问实例诊断报告

> **说明** 仅支持RDS for MySQL。

1. 登录RDS管理控制台。
2. 选择目标实例所在地域。
3. 单击目标实例ID，进入**基本信息**页面。
4. 单击页面右上角的**登录数据库**，进入数据管理控制台的快捷登录页面。
5. 选择**性能 > 诊断报告**，或者点击界面右侧的**查看诊断报告**按钮，跳转到混合云数据库管理平台（Hybrid Cloud Database Management，HDM）。

> **说明** 对于第一次进入的用户需要对HDM进行授权。



6. 查看已有的诊断报告，或者单击**发起诊断**生成新的诊断报告。

> **注意** 如果问题正在发生，建议先点击**发起诊断**。

如问题还未解决，请联系售后技术支持。

4.网络/IP

4.1. RDS实例如何变更虚拟交换机VSwitch

背景

实例网络类型为专有网络（VPC），需要变更虚拟交换机。

操作步骤

> **说明** MySQL部分实例支持直接变更VPC和交换机，详情请参见切换专有网络VPC和虚拟交换机。

- 对于支持从VPC切换到经典网络，以及支持从经典网络切换到VPC的实例：
  i. 将网络模式从VPC切换为经典网络。
  ii. 将网络模式从经典网络切换至目的VPC，同时选择目的虚拟交换机。

  > **说明** 切换步骤请参见切换网络类型。

- 对于不支持网络类型切换的实例：

  购买新的实例（购买时选择目的VPC和目的虚拟交换机），然后将数据迁移到新的实例，具体的迁移步骤请参见如下文档：

  - MySQL实例间数据迁移
  - SQL Server实例间数据迁移
  - PostgreSQL实例间数据迁移
  - PPAS实例间数据迁移
  - MariaDB实例间数据迁移

4.2. RDS for MySQL如何终止会话

RDS for MySQL有两种方式来终止会话。

通过DMS终止

1. 通过DMS登录RDS数据库。
2. 在上方选择**性能 > 实例会话**。

3. 在弹出的会话框中选择新版并进行授权（仅首次登录需授权）。
4. 通过Shift、Ctrl键选择多个会话，然后通过**kill选中会话**按钮来终止相关会话。

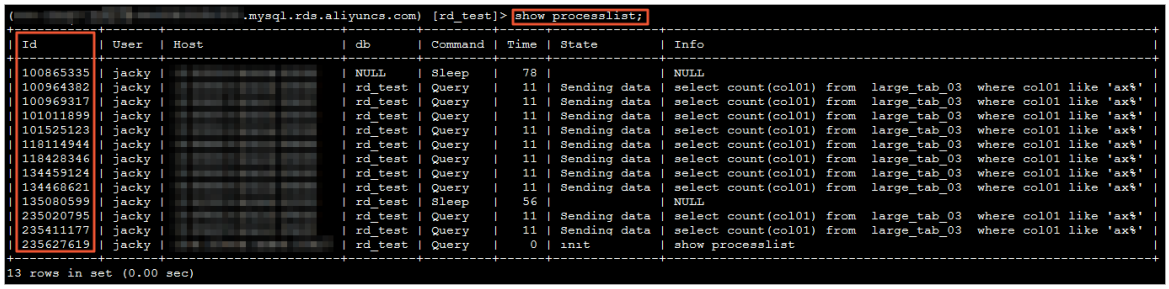
通过kill命令终止

1. 通过MySQL命令行工具连接实例，请参见通过客户端、命令行连接RDS MySQL。

> **说明** RDS实例在连接数已满的情况下，是无法通过DMS或者MySQL命令行工具连接实例的。如果无法通过DMS或MySQL命令行工具连接，建议先在控制台的**参数设置**中将 wait_timeout参数（单位秒）设置为比较小的值（比如 60），让RDS实例主动关闭空闲时间超过60秒的连接，以便稍后可以通过DMS或者MySQL命令行工具连接访问实例。

2. 通过如下命令查看当前会话情况，记下想要kill的会话的Id。

```
show processlist;
```



3. 通过如下命令kill会话。

```
kill <Id>;
```

4.3. 阿里云在RDS遭受攻击时提供什么安全服务

当您的数据库的公网访问遇到以下情况时，阿里云的安全体系认为数据库受到攻击，会自动启动流量清洗或黑洞处理。

流量清洗

以下任一条件将会触发流量清洗（针对公网入流量）。

- PPS（Package Per Second ）达到3万；
- BPS（Bits per Second）达到180Mb；
- 每秒新建并发连接达到1万；
- 激活连接数达到1万；
- 非激活链接数达到10万。

> **说明** 此时RDS仍然可以被正常访问。

黑洞处理

以下任一条件将会触发黑洞（针对公网入流量）。

- BPS（Bits per Second）达到2Gb；
- 流量清洗无效。

> **说明** 触发黑洞后RDS在4小时内无法访问。

5.数据库/账号/表

5.1. RDS for MySQL使用utf8mb4字符集存储emoji表情

基本原则

如果要实现存储emoji表情到RDS for MySQL实例，需要客户端、到RDS实例的连接、RDS实例内部三个方面统一使用或者支持utf8mb4字符集。

- 客户端：客户端需要保证输出的字符串的字符集为utf8mb4。
- 应用到RDS实例的连接：支持utf8mb4字符集。以常见的JDBC连接为例，需要使用MySQL Connector/J 5.1.13（含）以上的版本，JDBC的连接串中，建议不配置characterEncoding选项。
- RDS实例配置：控制台设置参数character_set_server为utf8mb4、数据库的字符集为utf8mb4，以及表的字符集为utf8mb4。


```
(                mysql.rds.aliyuncs.com) [      ]> create table emoji_01 (id int auto_increment primary key, content varchar(255)) default charset utf8mb4;
Query OK, 0 rows affected (0.01 sec)

(                mysql.rds.aliyuncs.com) [      ]> show create table emoji_01 \G
*************************** 1. row ***************************
       Table: emoji_01
Create Table: CREATE TABLE `emoji_01` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `content` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
1 row in set (0.00 sec)
```

通过set names命令设置会话字符集

对于JDBC连接串设置了characterEncoding为utf8，或者做了上述配置仍旧无法正常插入emoji数据的情况，建议在代码中指定连接的字符集为utf8mb4，示例代码如下：

```
String query = "set names utf8mb4";
stat.execute(query);
```

5.2. RDS for MySQL 5.6 版本GTID特性对临时表限制的处理

由于RDS for MySQL 5.6版本引入了GTID特性，因此要求应用不能够在事务中创建和删除临时表。

错误信息

```
When @@GLOBAL.ENFORCE_GTID_CONSISTENCY = 1, the statements CREATE TEMPORARY TABLE and DROP TEMPORARY TABLE ca n be executed in a non-transactional context only, and require that AUTO COMMIT = 1.
```

解决方法

- 将 `create temporary table` 语句更改为 `create table` ，使用普通表替代临时表，规避这个问题。
- 修改代码，将临时表的创建和删除操作放在事务外，并且保证会话的参数 `autocommit=1` 。

如果问题还未能解决，请联系售后技术支持。

5.3. RDS for MySQL引擎表索引方式更改为Hash无效原因

问题描述

MySQL包含的索引方式主要包括Btree、Hash、FullText和Rtree，经常使用的主要是Btree和Hash两种。通过DMS登录RDS实例后，执行DDL语句可以在InnoDB引擎表上创建Hash方式的索引，命令如下：

```
drop table if exists auth_order;
create table auth_order (
    id smallint not null comment '主键',
 member_id varchar(30) not null comment '会员id',
 name varchar(100) not null comment '名称',
 primary key (id),
 key auth_mem_name (member_id) using hash
) engine=innodb default charset=utf8 comment='会员信息';
```

```
drop table if exists auth_order;

create table auth_order (
    id smallint not null comment '主键',
    member_id varchar(30) not null comment '会员id',
    name varchar(100) not null comment '名称',
    primary key (id),
    key auth_mem_name (member_id) using hash
) engine=innodb default charset=utf8 comment='会员信息';
```

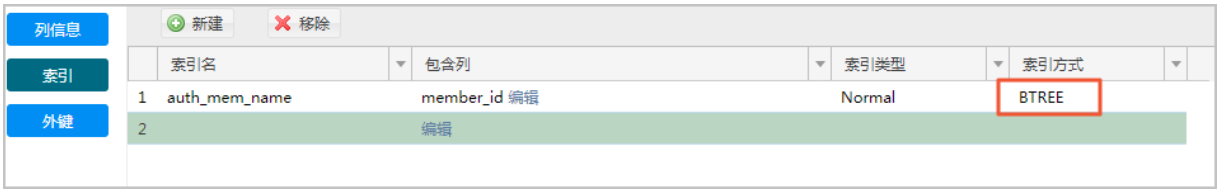
消息

```
【拆分SQL完成】：将执行SQL语句数量：（2 rows），拆分SQL耗时：（0ms.）

【执行SQL：(1)】
drop table if exists auth_order
执行成功，耗时：[8ms.]

【执行SQL：(2)】
create table auth_order (
    id smallint not null comment '主键',
    member_id varchar(30) not null comment '会员id',
    name varchar(100) not null comment '名称',
    primary key (id),
    key auth_mem_name (member_id) using hash
) engine=innodb default charset=utf8 comment='会员信息'
执行成功，耗时：[11ms.]
```

可以看到该表中的索引是Hash方式，但是在DMS中查看表结构的时候发现该索引的方式是Btree。



原因

由于MySQL的InnoDB引擎不支持Hash索引，而MySQL服务层是有Hash索引选项的，因此建表语句可以使用子句 Using Hash，而实际创建的索引类型仍然是Btree方式的索引。

5.4. RDS for MySQL auto_increment自增字段相关参数

RDS for MySQL经常会使用到auto_increment自增长字段，下面是相关参数的说明。

参数名称	默认值	取值范围	作用
auto_increment_increment	1	1~65535	控制增量的幅度。
auto_increment_offset	1	1~65535	增量开始的位置（开始的偏移量）。

> 说明
>
> - 两个参数均可以在全局和会话级别设置。
> - 如果auto_increment_offset的值大于auto_increment_increment，则auto_increment_offset被忽略。

示例：

```
Create table au (id int auto_increment primary key);
```

插入奇数

查看当前会话auto_increment相关参数设置：

```
show variables like 'auto_i%';
```

插入4行数据：

```
insert into au values (null),(null),(null),(null);
```

检查表内数据：

```
select * from au;
```

```
[redacted].mysql.rds.aliyuncs.com) [ [redacted] ]> show variables like 'auto_i%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 2     |
| auto_increment_offset    | 1     |
+--------------------------+-------+
2 rows in set (0.00 sec)

([redacted].mysql.rds.aliyuncs.com) [ [redacted] ]> insert into au values (null),(null),(null),(null);
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

([redacted].mysql.rds.aliyuncs.com) [ [redacted] ]> select * from au;
+-----+
| col |
+-----+
|   1 |
|   3 |
|   5 |
|   7 |
+-----+
4 rows in set (0.00 sec)
```

插入偶数

```
(                                .mysql.rds.aliyuncs.com) [      ]> show variables like 'auto_i%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 2     |
| auto_increment_offset    | 2     |
+--------------------------+-------+
2 rows in set (0.00 sec)

(                                .mysql.rds.aliyuncs.com) [      ]> insert into au values (null),(null),(null),(null);
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0

(                                .mysql.rds.aliyuncs.com) [      ]> select * from au;
+------+
| col  |
+------+
|    2 |
|    4 |
|    6 |
|    8 |
+------+
4 rows in set (0.01 sec)
```

插入以300001开始的奇数

设置表auto increment初始值为300001：

```
alter table au auto_increment=300001;
```

```
(                                .mysql.rds.aliyuncs.com) [      ]> alter table au auto_increment=300001;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

(                                .mysql.rds.aliyuncs.com) [      ]> show variables like 'auto_i%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 2     |
| auto_increment_offset    | 1     |
+--------------------------+-------+
2 rows in set (0.00 sec)

(                                .mysql.rds.aliyuncs.com) [      ]> insert into au values (null),(null),(null),(null);
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

(                                .mysql.rds.aliyuncs.com) [      ]> select * from au;
+--------+
| col    |
+--------+
| 300001 |
| 300003 |
| 300005 |
| 300007 |
+--------+
4 rows in set (0.00 sec)
```

插入以300002开始的偶数

设置表auto increment字段初始值为300002：

```
alter table au auto_increment=300002;
```

```
(                                .mysql.rds.aliyuncs.com) [      ]> alter table au auto increment=300002;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

(                                .mysql.rds.aliyuncs.com) [      ]> show variables like 'auto_i%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 2     |
| auto_increment_offset    | 2     |
+--------------------------+-------+
2 rows in set (0.00 sec)

(                                .mysql.rds.aliyuncs.com) [      ]> insert into au values (null),(null),(null),(null);
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

(                                .mysql.rds.aliyuncs.com) [      ]> select * from au;
+--------+
| col    |
+--------+
| 300002 |
| 300004 |
| 300006 |
| 300008 |
+--------+
4 rows in set (0.00 sec)
```

插入初始值为3，增量为5的记录

```
(                                .mysql.rds.aliyuncs.com) [      ]> show variables like 'auto_i%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 5     |
| auto_increment_offset    | 3     |
+--------------------------+-------+
2 rows in set (0.00 sec)

(                                .mysql.rds.aliyuncs.com) [      ]> insert into au values (null),(null),(null),(null);
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

(                                .mysql.rds.aliyuncs.com) [      ]> select * from au;
+-----+
| col |
+-----+
|   3 |
|   8 |
|  13 |
|  18 |
+-----+
4 rows in set (0.00 sec)
```

5.5. RDS for MySQL数据库自增列不连续的问题

问题现象

使用RDS for MySQL数据库时， 发现自增列不连续。

问题原因

由于数据库的列存在约束条件，插入数据失败，导致的了自增列不连续。

关于自增列的原理详情请参考MySQL的官方文档：点此查看

5.6. RDS for MySQL函数group_concat相关问题

group_concat返回结果的长度

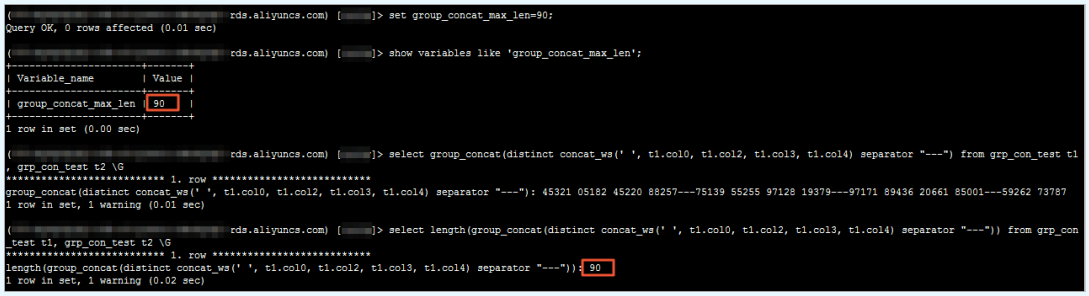
函数group_concat返回结果的长度受参数**group_concat_max_len**控制，默认值为1024，即默认返回1024字节长度结果。

参数名称	默认值	取值范围	说明
group_concat_max_len	1024	4-18446744073709547520	group_concat函数返回结果的最大长度，单位：Byte。

说明 您可以设置参数group_concat_max_len在全局生效或会话级别生效：

- 全局生效：在控制台的**参数设置**页面修改。
- 会话级别生效：

```
set group_concat_max_len=90; -- 设置当前会话 group_concat_max_len 为 90 字节
show variables like 'group_concat_max_len'; -- 查看当前会话的 group_concat_max_len 值
select group_concat(distinct concat_ws(' ', t1.col0, t1.col2, t1.col3, t1.col4) separator "---")
from grp_con_test t1, grp_con_test t2 \G  -- 查询结果
select length(group_concat(distinct concat_ws(' ', t1.col0, t1.col2, t1.col3, t1.col4) separator "---"))
from grp_con_test t1, grp_con_test t2 \G  -- 查询结果的长度
```

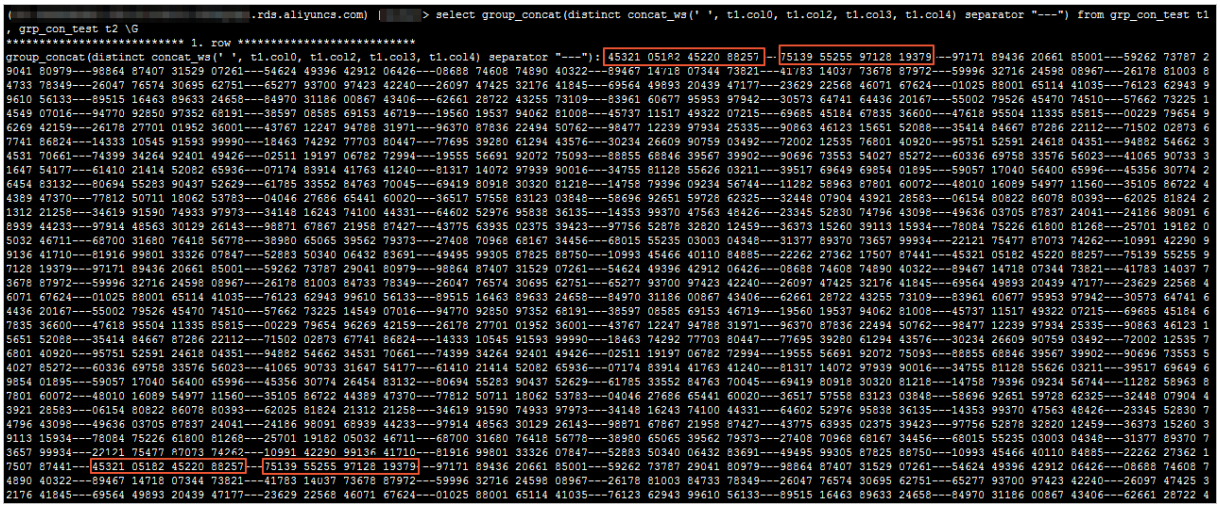


group_concat(distinct) 去除重复数据失效的处理

失效原因

当设置group_concat_max_len为较大值时，使用 `group_concat(distinct)` 去除结果中的重复数据，会出现失效的情况，例如：

```
select group_concat(distinct concat_ws(' ', t1.col0, t1.col2, t1.col3, t1.col4) separator "---")
from grp_con_test t1, grp_con_test t2 \G -- 查询结果
```



可以看到，结果中出现了多个重复值。出现这个问题的原因当group_concat返回结果集比较大，会出现内存临时表无法承载全部结果集数据，进而会使用磁盘临时表；而group_concat在使用磁盘临时表时会触发BUG导致无法去除重复数据。

解决方法

调整tmp_table_size参数设置，增大内存临时表的最大尺寸，命令如下：

```
set tmp_table_size=1*1024*1024 -- 设置当前会话 tmp_table_size 为 1 MB
show variables like 'tmp_table_size' -- 查看当前会话 tmp_table_size 的设置
select group_concat(distinct concat_ws(' ', t1.col0, t1.col2, t1.col3, t1.col4) separator "
---")
from grp_con_test t1, grp_con_test t2 \G
```

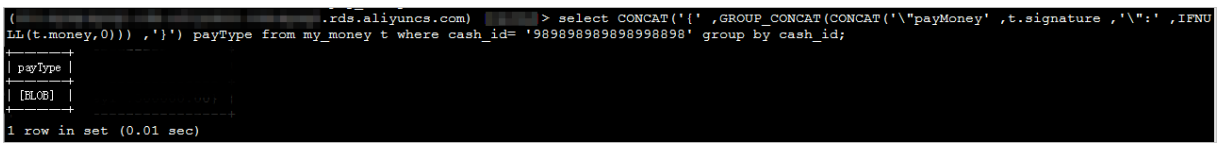
> **说明** 也可以在控制台的**参数设置**页面修改参数tmp_table_size。

group_concat和concat结合使用返回异常

异常原因

group_concat和concat结合使用某些情况下会出现返回BLOB字段类型的情况，例如：

```
select concat('{' ,group_concat(concat('\"payMoney' ,t.signature ,'\":' ,ifnull(t.money,0))
) ,'}') payType
from my_money t
where cash_id='9898989898998998'
group by cash_id;
```



这是由于函数concat按字节返回结果，如果concat的输入有多种类型，其结果是不可预期的。

解决方法

通过cast函数进行约束，为concat返回结果为字符串类型，将上面例子修改为：

```
select concat('{' ,cast(group_concat(concat('\"payMoney' ,t.signature ,'\":' ,ifnull(t.mone
y,0))) as char) ,'}') payType
from my_money y t
where cash_id='989898989898998898'
group by cash_id;
```

```
(                                ds.aliyuncs.com) [      ]> select CONCAT('{' ,cast(GROUP_CONCAT(CONCAT('\"payMoney' ,t.signature ,'\":'
,IFNULL(t.money,0))) as char) ,'}') payType from my_money t where cash_id= '989898989898998898' group by cash_id;
+-------------------------+
| payType                 |
+-------------------------+
| {"payMoney1":500000.00} |
+-------------------------+
1 row in set (0.00 sec)
```

5.7. RDS for MySQL查看表的主键字段的方法

查看系统表

```
SELECT
  t.TABLE_NAME,
  t.CONSTRAINT_TYPE,
  c.COLUMN_NAME,
  c.ORDINAL_POSITION
FROM
  INFORMATION_SCHEMA.TABLE_CONSTRAINTS AS t,
  INFORMATION_SCHEMA.KEY_COLUMN_USAGE AS c
WHERE
  t.TABLE_NAME = c.TABLE_NAME
  AND t.CONSTRAINT_TYPE = 'PRIMARY KEY'
  AND t.TABLE_NAME='<表名>'
  AND t.TABLE_SCHEMA='<数据库名>';
```

查看建表语句

```
show create table <表名>;
```

查看表结构

```
desc <表名>;
```

5.8. RDS for MySQL为无主键表添加主键

本文介绍RDS for MySQL的表在没有主键时如何添加主键。

查看无主键表

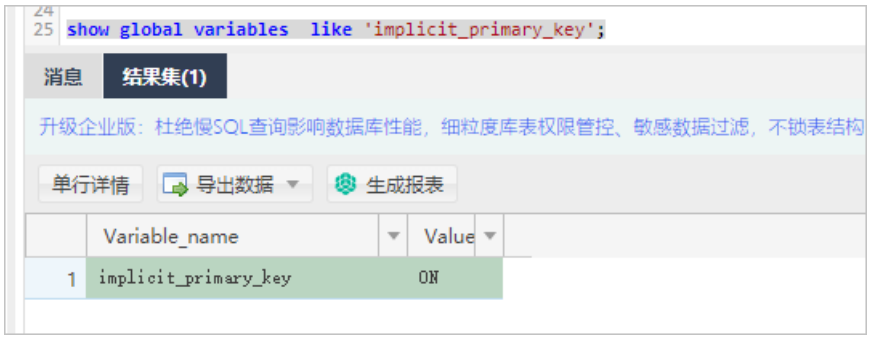
执行如下命令查看是否有无主键表：

```
select table_schema,table_name from information_schema.tables
where (table_schema,table_name) not in(
    select distinct table_schema,table_name from information_schema.columns where COLUMN_KE
Y='PRI'
)
and table_schema not in (
    'sys','mysql','information_schema','performance_schema'
);
```

解决方法

- 添加隐式主键
  i. 使用如下命令查看参数implicit_primary_key的值是否为ON。

  ```
  show global variables  like 'implicit_primary_key';
  ```

  > **说明** 如果查询没有该参数或值为OFF，请您提交工单进行修改。

  ii. 执行如下命令修改无主键表：

  ```
  alter table <表名> engine=innodb;
  ```

  > **说明**
  > - 执行过程中会禁止数据写入表，但是仍然可以读取。
  > - 当implicit_primary_key为ON时，修改表的引擎会自动添加隐式主键。

- 直接添加主键

  执行如下命令为无主键表添加主键：

  ```
  ALTER TABLE <表名> ADD PRIMARY KEY (<需要设为主键的列名>);
  ```

5.9. RDS for MySQL排序分页查询数据顺序错乱的处理

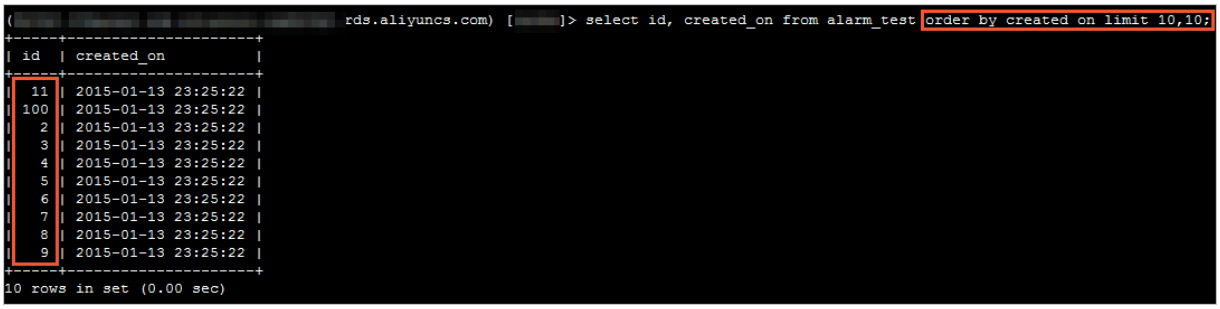
MySQL排序分页查询数据顺序错乱的原因

MySQL排序分页查询某些时候会出现数据顺序错乱的情况，例如：

```
CREATE TABLE alarm_test (
  id bigint(20) NOT NULL DEFAULT '0',
  detail varchar(255) CHARACTER SET utf8 NOT NULL,
  created_on timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

按照created_on字段值排序，取前10行，如下图。

按照created_on字段值排序，取从11行开始的10行，如下图。



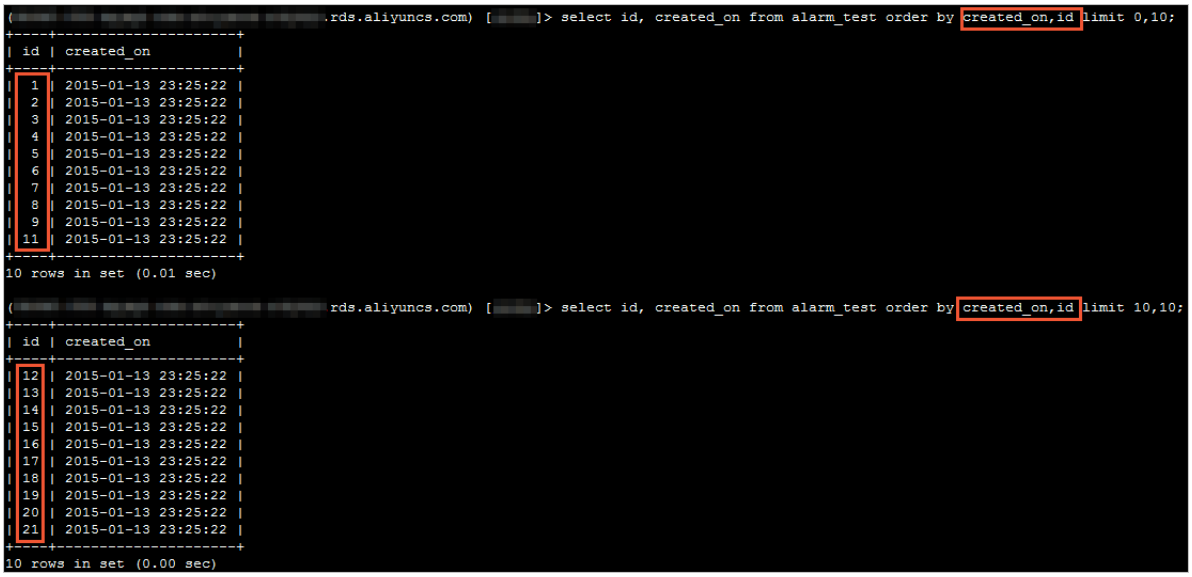
可以看出，2次排序分页操作得到的数据是有重合而且无序的（实际上排序分页结果会根据情况的不同而变化，结果不可预料）。出现这个问题的原因在于created_on字段的值在前21行记录中有20行数据相同。

如何使排序字段结果有序

使排序字段结果有序有如下两个方法：

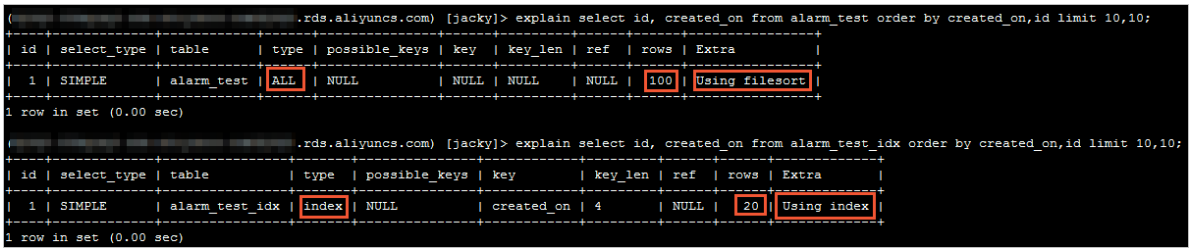
- 修改排序规则，加入主键字段，使排序字段不存在重复记录，例如：

  ```
  select id, created_on from alarm_test order by created_on, id limit 0,10;
  ```



- 在出现重复值的排序字段上添加索引，例如：

```
CREATE TABLE alarm_test_idx (
  id bigint(20) NOT NULL DEFAULT '0',
  detail varchar(255) CHARACTER SET utf8 NOT NULL,
  created_on timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (id),
  KEY created_on (created_on)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

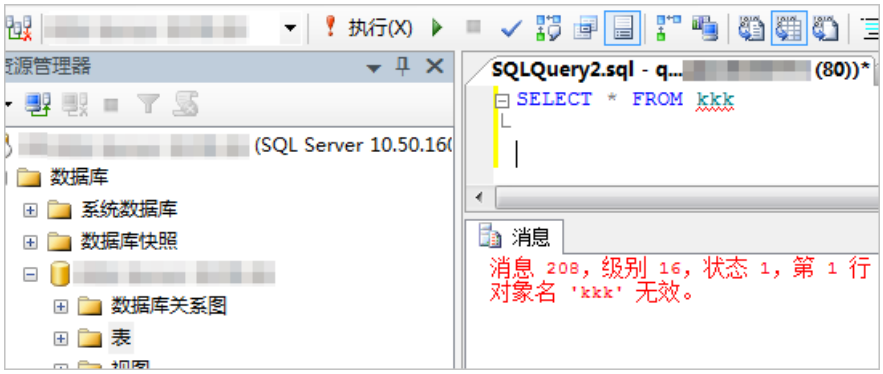


> **说明** 推荐使用添加索引的方法，在提供可预期的结果同时，提高查询的执行效率。

5.10. RDS for SQL Server如何修改schema为dbo

问题现象

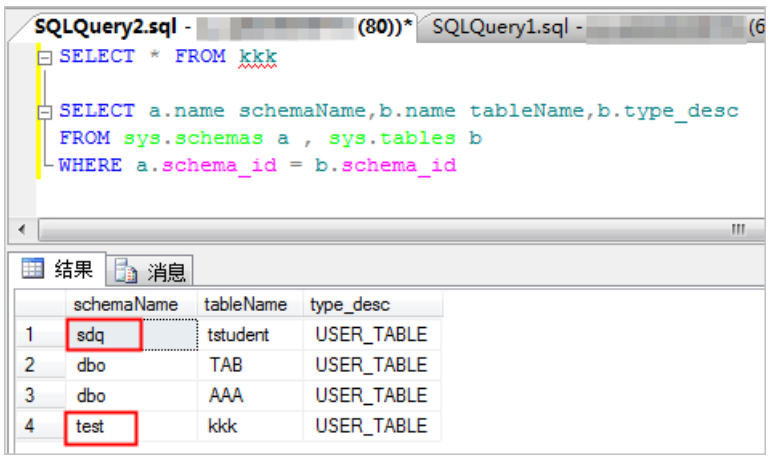
经常会遇到schema 为非 dbo的情况，从而导致查询时提示对象名无效的的错误，如图所示。



原因

查看该表的相关信息，命令如下：

```
SELECT a.name schemaName,b.name tableName,b.type_desc FROM sys.schemas a , sys.tables b
WHERE a.schema_id = b.schema_id
```



可以看到schema不是dbo，导致问题出现。

解决方案

- 修改单个表的schema

  命令如下：

  ```
  ALTER SCHEMA dbo TRANSFER test.kkk
  ```

- 批量修改表的schema

  命令如下：

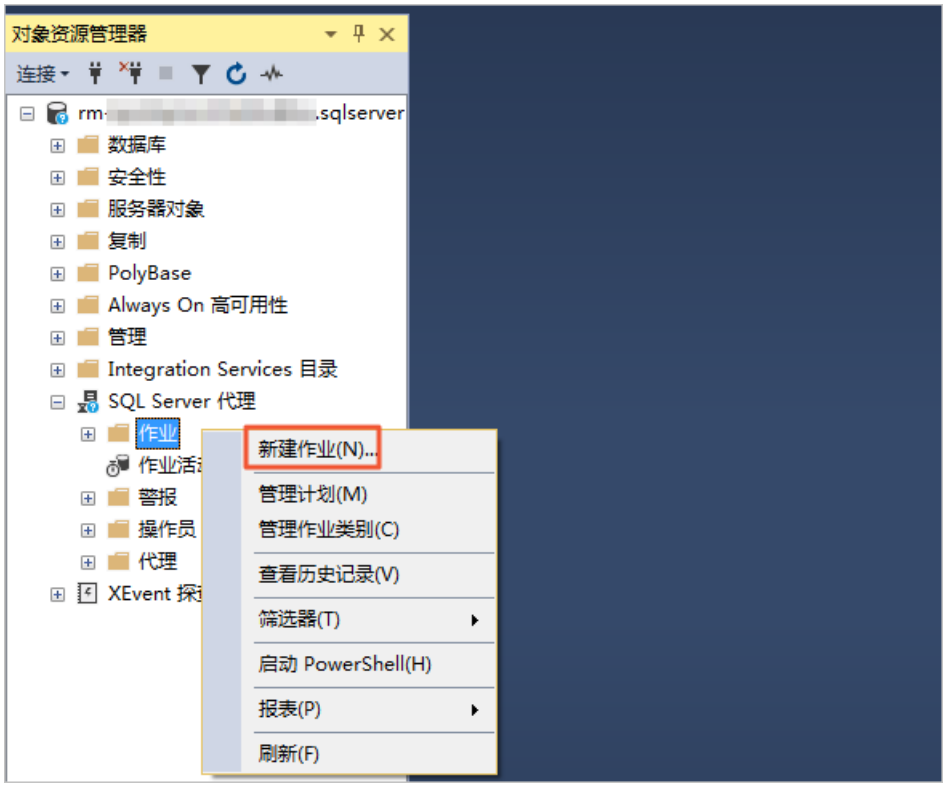
  ```
  exec sp_msforeachtable 'alter schema dbo transfer ?'
  ```

5.11. RDS for SQL Server添加作业计划

使用RDS for SQL Server时，如果您需要添加作业计划，可以通过客户端连接实例，进行计划的添加执行。

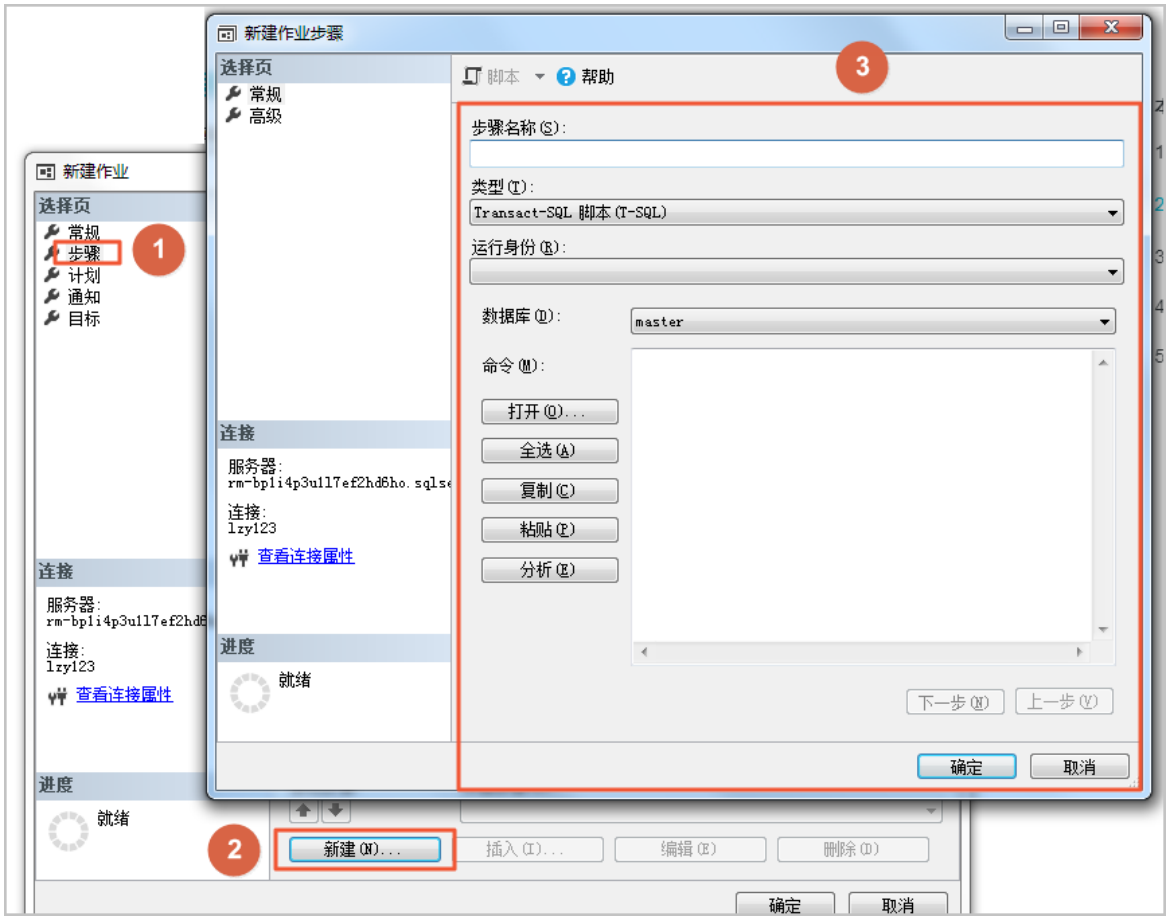
1. 使用客户端连接SQL Server实例。

2. 在**SQL Server 代理 > 作业**上单击鼠标右键，选择**新建作业**。

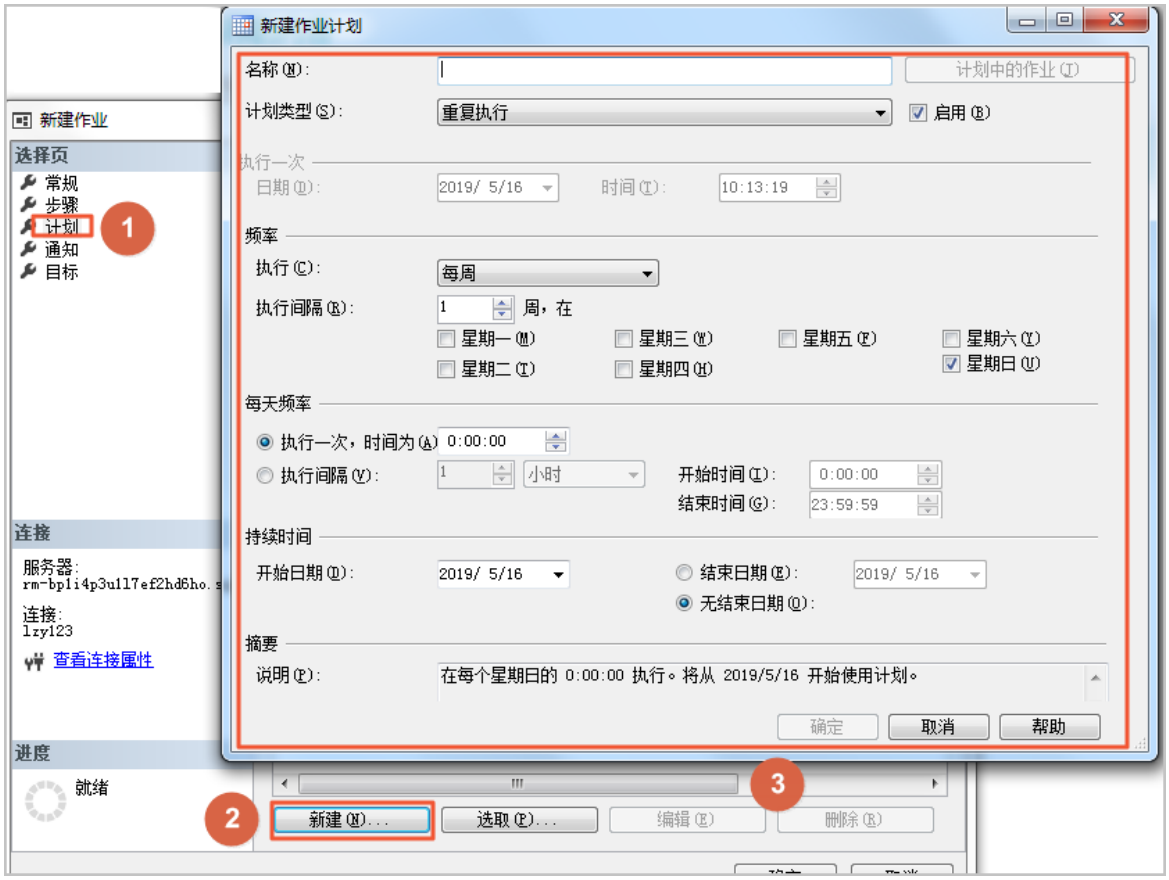


3. 在**常规**页面填写作业名称。

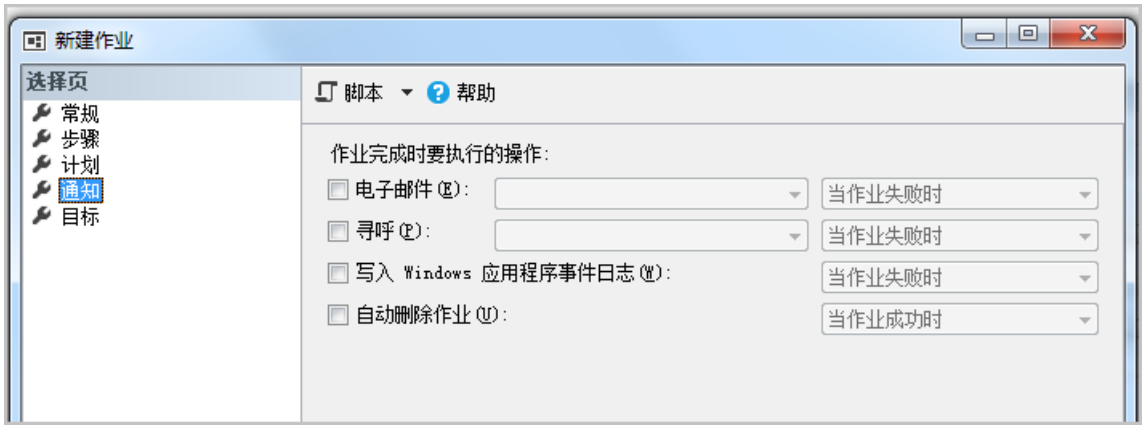
4. 在**步骤**页面填写作业步骤。



5. 在**计划**页面填写作业计划。



6. 最后两项**通知**和**目标**，如果没有特殊要求，按照默认设置即可。

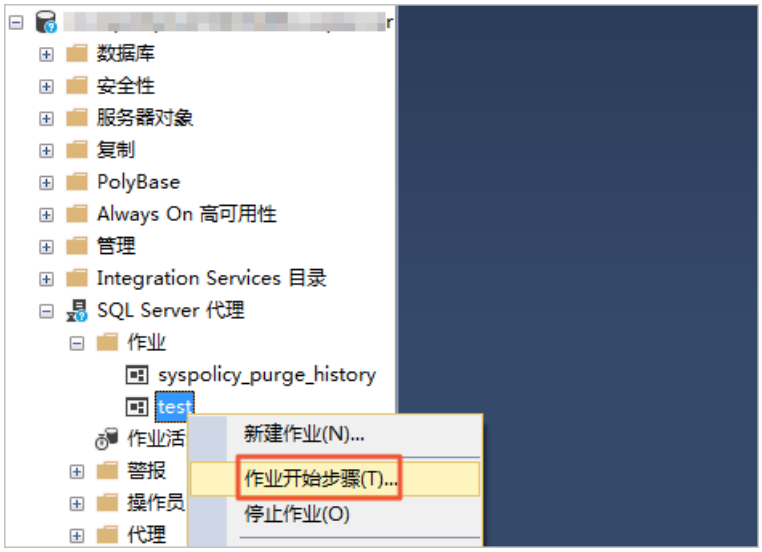


7. 单击**确定**。

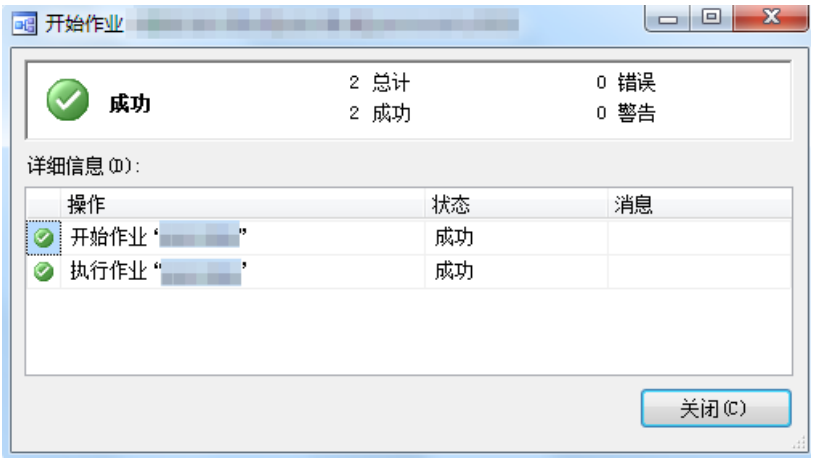
5.12. RDS for SQL Server手动执行作业及查看执行记录

手动执行作业

1. 使用客户端连接SQL Server实例。
2. 展开**SQL Server 代理 > 作业**，在已创建的作业上单击鼠标右键。
3. 单击**作业开始步骤**。

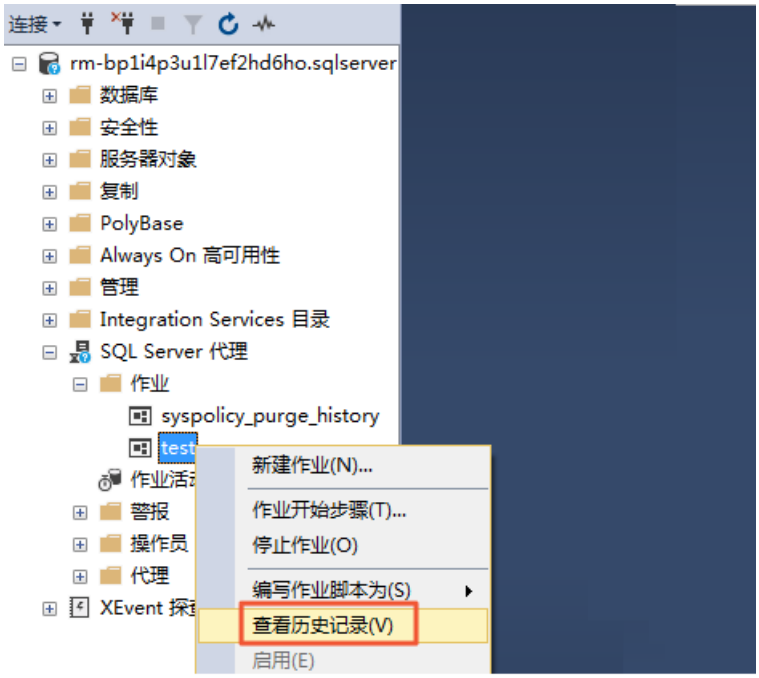


执行作业后会显示成功或失败。

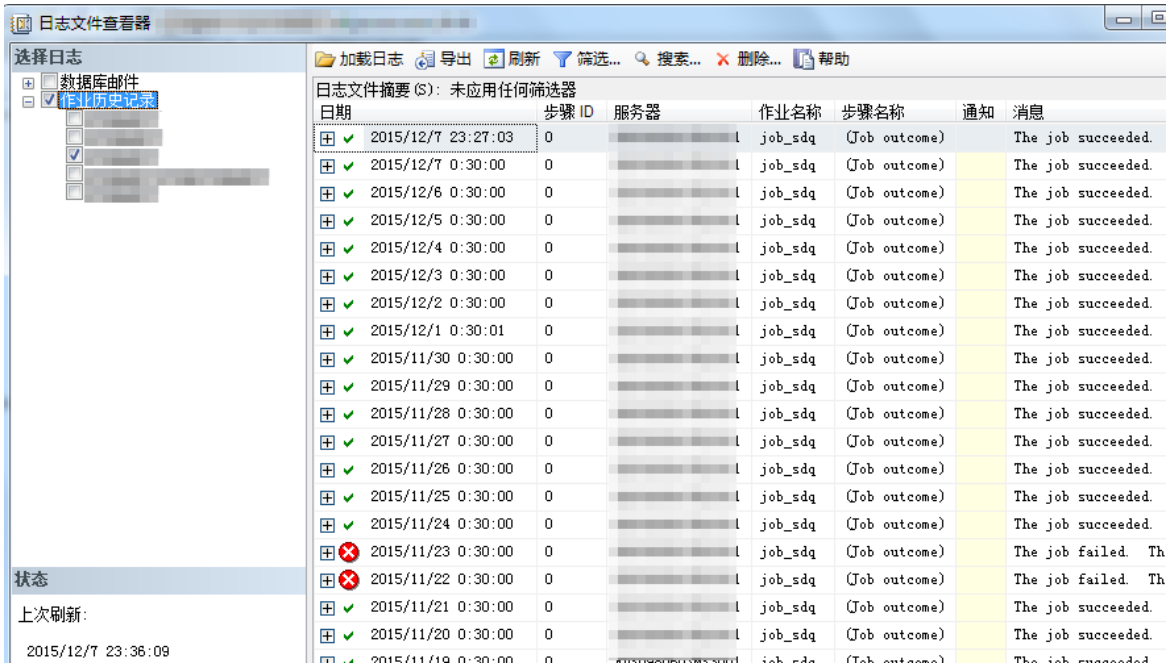


查看执行记录

1. 使用客户端连接SQL Server实例。
2. 展开**SQL Server 代理 > 作业**，在已创建的作业上单击鼠标右键。
3. 单击**查看历史记录**。



4. 在**日志文件查看器**里选择相应的作业，即可查看到执行记录。



5.13. RDS for SQL Server批量导入数据

RDS for SQL Server支持通过Bulk Insert批量导入数据，但存在一定限制。

> **说明** Bulk Insert会触发RDS for SQL Server 2008R2版本实例的一个Bug，因此需要在使用时将开启CheckConstraints选项。

通过BCP命令方式

1. 生成XML格式文件，示例如下：

```
bcp jacky.dbo.my_bcp_test format nul /c /t"," /x /f "d:\tmp\my_bcp_test.xml"  /U jacky
/P xxxx /S "xxx.sqlserver
.rds.aliyuncs.com,3333"
```

2. 导入数据，示例如下：

```
bcp jacky.dbo.my_bcp_test in "d:\tmp\my_test_data_file.txt" /f "d:\tmp\my_bcp_test.xml"
/q /k /h "CHECK_CONSTRAINTS"  /U jacky /P xxx /S "xxx.sqlserver.rds.aliyuncs.com,3333"
```

通过JDBC SQLBulkCopy方式

示例如下：

```
SQLServerBulkCopyOptions copyOptions = new SQLServerBulkCopyOptions();
                        copyOptions.setCheckConstraints(true);
```

> **说明** 详情请参见通过JDBC驱动程序使用大容量复制。

通过ADO.NET SQLBulkCopy方式

将SqlBulkCopy指定为SqlBulkCopyOptions.CheckConstraints即可，示例如下：

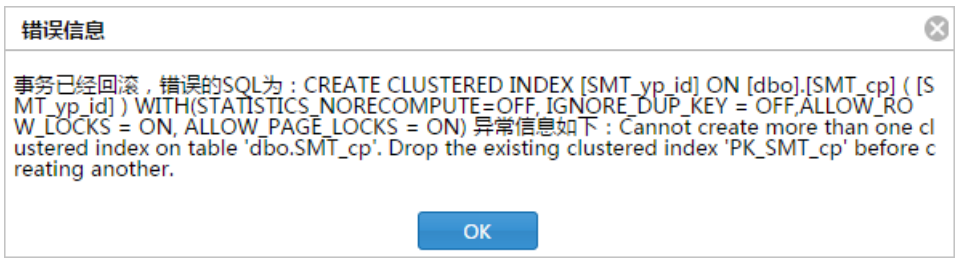
```
static void Main()
        {
            string srcConnString = "Data Source=(local);Integrated Security=true;Initial Ca
talog=testdb";
            string desConnString = "Data Source=****.sqlserver.rds.aliyuncs.com,3433;User I
D=**;Password=**;Initial Catalog=testdb";
            SqlConnection srcConnection = new SqlConnection();
            SqlConnection desConnection = new SqlConnection();
            SqlCommand sqlcmd = new SqlCommand();
            SqlDataAdapter da = new SqlDataAdapter();
            DataTable dt = new DataTable();
            srcConnection.ConnectionString = srcConnString;
            desConnection.ConnectionString = desConnString;
            sqlcmd.Connection = srcConnection;
            sqlcmd.CommandText = @"SELECT top 1000000 [PersonType],[NameStyle],[Title],[Fir
stName],[MiddleName],[LastName],[Suffix],[EmailPromotion]
                              ,[AdditionalContactInfo],[Demographics],NULL as rowguid,[Modif
iedDate] FROM [testdb].[dbo].[Person]";
            sqlcmd.CommandType = CommandType.Text;
            sqlcmd.Connection.Open();
            da.SelectCommand = sqlcmd;
            da.Fill(dt);
            using (SqlBulkCopy blkcpy = new SqlBulkCopy(desConnString, SqlBulkCopyOptions.C
heckConstraints))
            //using (SqlBulkCopy blkcpy = new SqlBulkCopy(desConnString, SqlBulkCopyOptions
.Default))
            {
                blkcpy.BatchSize = 2000;
                blkcpy.BulkCopyTimeout = 5000;
                blkcpy.SqlRowsCopied += new SqlRowsCopiedEventHandler(OnSqlRowsCopied);
                blkcpy.NotifyAfter = 2000;
```

```
                foreach (DataColumn dc in dt.Columns)
                {
                    blkcpy.ColumnMappings.Add(dc.ColumnName, dc.ColumnName);
                }
                try
                {
                    blkcpy.DestinationTableName = "Person";
                    blkcpy.WriteToServer(dt);
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex.Message);
                }
                finally
                {
                    sqlcmd.Clone();
                    srcConnection.Close();
                    desConnection.Close();
                }
            }
        }
        private static void OnSqlRowsCopied(
            object sender, SqlRowsCopiedEventArgs e)
        {
            Console.WriteLine("Copied {0} so far...", e.RowsCopied);
        }
```

5.14. RDS for SQL Server创建聚簇索引注意事项

一个表只能创建一个聚簇索引

如果已创建聚簇索引，再次创建会失败，报错如下。



使用sp_helpindex查看索引

命令如下：

```
ues  <数据库名>
go
sp_helpindex '<表名>'
```

使用drop index删除聚簇索引

命令如下：

```
DROP INDEX <索引名> ON <数据库名>.<表名>
```

重新计算统计信息

统计信息指的是创建索引时的STATISTICS_NORECOMPUTE选项。一般来说都需要重新计算，详情请参见CREATE INDEX。

> **说明** STATISTICS_NORECOMPUTE的默认值是OFF，即需要重新计算（因为该选项本身就是否定的意思）。

5.15. RDS for PostgreSQL 9.4中如何支持jsonb_set等新的 jsonb函数

解决方案

1. 使用客户端连接数据库后执行如下命令：

   ```
   create extension jsonbx;
   ```

   > **说明** 如果遇到以下错误，请通过控制台重启数据库实例以升级到最新的9.4版本。
   >
   > ```
   > postgres=> create extension jsonbx;
   > ERROR:  invalid extension name: "jsonbxx"
   > DETAIL:  Extension is not supported.
   > ```

2. 执行如下命令：

   ```
   jsonb_pretty (in 9.5)
   jsonb_concat (in 9.5)
   jsonb_delete(jsonb, text) (in 9.5)
   jsonb_delete_idx(jsonb, int) (in 9.5)
   jsonb_delete_path(jsonb, text[]) (in 9.5)
   jsonb_set(jsonb, text[], jsonb) (in 9.5)
   concatenation operator (||) (in 9.5)
   delete key operator (jsonb - text) (in 9.5)
   delete key by index operator (jsonb - int) (in 9.5)
   delete key by path operator (jsonb - text[]) (in 9.5)
   ```

5.16. RDS for PostgreSQL使用中文分词

启用中文分词

可以使用下面的命令，启用中文分词：

```
CREATE EXTENSION zhparser;
CREATE TEXT SEARCH CONFIGURATION testzhcfg (PARSER = zhparser);
ALTER TEXT SEARCH CONFIGURATION testzhcfg ADD MAPPING FOR n,v,a,i,e,l WITH simple;
--可选的参数设定
alter role all set zhparser.multi_short=on;
--简单测试
SELECT * FROM ts_parse('zhparser', 'hello world! 2010年保障房建设在全国范围内获全面启动，从中央到地方纷纷加大了保障房的建设和投入力度。2011年，保障房进入了更大规模的建设阶段。住房城乡建设部党组书记、部长姜伟新去年底在全国住房城乡建设工作会议上表示，要继续推进保障性安居工程建设。');
SELECT to_tsvector('testzhcfg','"今年保障房新开工数量虽然有所下调，但实际的年度在建规模以及竣工规模会超以往年份，相对应的对资金的需求也会创历史纪录。"陈国强说。在他看来，与2011年相比，2012年的保障房建设在资金配套上的压力将更为严峻。');
SELECT to_tsquery('testzhcfg', '保障房资金压力');
```

利用分词进行全文索引的方法如下：

```
--为T1表的name字段创建全文索引
create index idx_t1 on t1 using gin (to_tsvector('zhcfg',upper(name) ));
--使用全文索引
 select * from t1 where to_tsvector('zhcfg',upper(t1.name)) @@ to_tsquery('zhcfg','(防火)')
;
```

自定义中文分词词典

自定义中文分词词典，示例如下：

```
-- 确实的分词结果
SELECT to_tsquery('testzhcfg', '保障房资金压力');
-- 往自定义分词词典里面插入新的分词
insert into pg_ts_custom_word values ('保障房资');
-- 使新的分词生效
select zhprs_sync_dict_xdb();
-- 退出此连接
\c
-- 重新查询，可以得到新的分词结果
SELECT to_tsquery('testzhcfg', '保障房资金压力');
```

> **说明** 内核小版本20160801和之后的版本才支持自定义中文分词词典，您可以通过命令 `show rds_release_date ;` 来查询内核版本。

使用自定义分词的注意事项如下：

- 最多支持一百万条自定义分词，超出部分不做处理，用户必须保证分词数量在这个范围之内。自定义分词与缺省的分词词典将共同产生作用。
- 每个词的最大长度为128字节，超出部分将会截取。
- 增删改分词之后必须执行 `select zhprs_sync_dict_xdb();` 并且重新建立连接才会生效。

5.17. RDS for PostgreSQL跨库查询

RDS for PostgreSQL可以通过dblink插件来实现跨库查询。

创建dblink插件

登录源库，创建dblink插件，命令如下：

```
create extension dblink;
```

创建到目的库的连接

创建到目的库的连接，命令如下：

```
select dblink_connect('<连接名称>','host=<目的实例域名> port=<目的实例端口> dbname=<目的库名称>
user=<目的库账号> password=<目的库密码>');
```

示例

```
select dblink_connect('test1_test2','host=rdsxxx.pg.rds.aliyuncs.com port=3433 dbname=test2
user=pgtest2 password=123456');
```

> **说明** 如果使用dblink访问相同实例的不同数据库，则其中的host=localhost，可以通过 `show port;` 来获得本地的端口。

跨库查询

跨库查询示例如下：

```
select * from dblink('test1_test2','select * from area_j02') as area_j02(id int, name varch
ar(20));
```

5.18. RDS for PPAS如何管理插件

本文介绍PPAS如何管理插件。

RDS for PPAS没有对外开放超级用户，为了解决插件问题，RDS在模板库template1上创建了一个插件管理函数rds_manage_extension。

您可以在创建数据库时使用模板库，示例如下：

```
create database mydb template template1;
```

这样您的数据库中就存在插件管理函数。

用RDS高权限用户登录自己的数据库，使用这个插件可以创建和删除外部插件，示例如下：

```
select rds_manage_extension('create','dblink');  --创建插件
select rds_manage_extension('drop','dblink');  --删除插件
```

目前支持管理的插件如下：

支持的插件	支持的插件
pg_stat_statements	pg_prewarm
oss_fdw	pg_trgm
btree_gin	postgres_fdw
btree_gist	sslinfo
chkpass	tablefunc
citext	tsearch2
cube	unaccent
dblink	postgis
dict_int	postgis_topology
earthdistance	fuzzystrmatch
hstore	postgis_tiger_geocoder
intagg	plperl
intarray	pltcl
isn	plv8
ltree	uuid-ossp
pgcrypto	plpgsql
pgrowlocks	

6.功能/付费方式

7.空间/内存

7.1. 释放MySQL表空间

MySQL可以通过 `optimize table` 命令释放表空间，重组表数据和索引的物理页，减少所占空间和优化读写性能。

如果使用 `delete` 命令删除数据库，表空间不会直接回收，您可以用 `optimize table` 释放表空间。

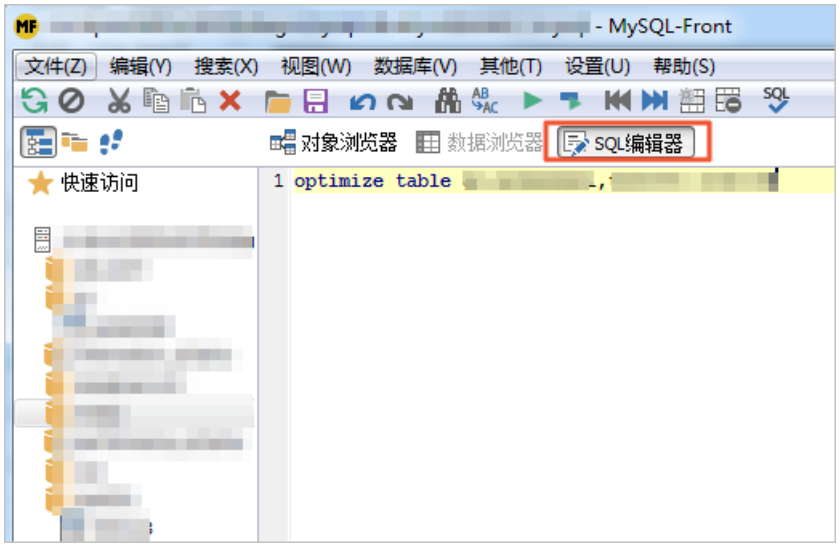
注意事项

- 操作将会锁表，建议在业务低峰期操作。
- 仅Innodb和MyISAM引擎支持 `optimize table` 命令。

通过命令行操作

1. 使用通用客户端连接MySQL数据库，本文以MySQL-Front为例进行演示。
2. 在右侧选择**SQL编辑器**页签。
3. 在命令行中执行如下命令释放表空间：

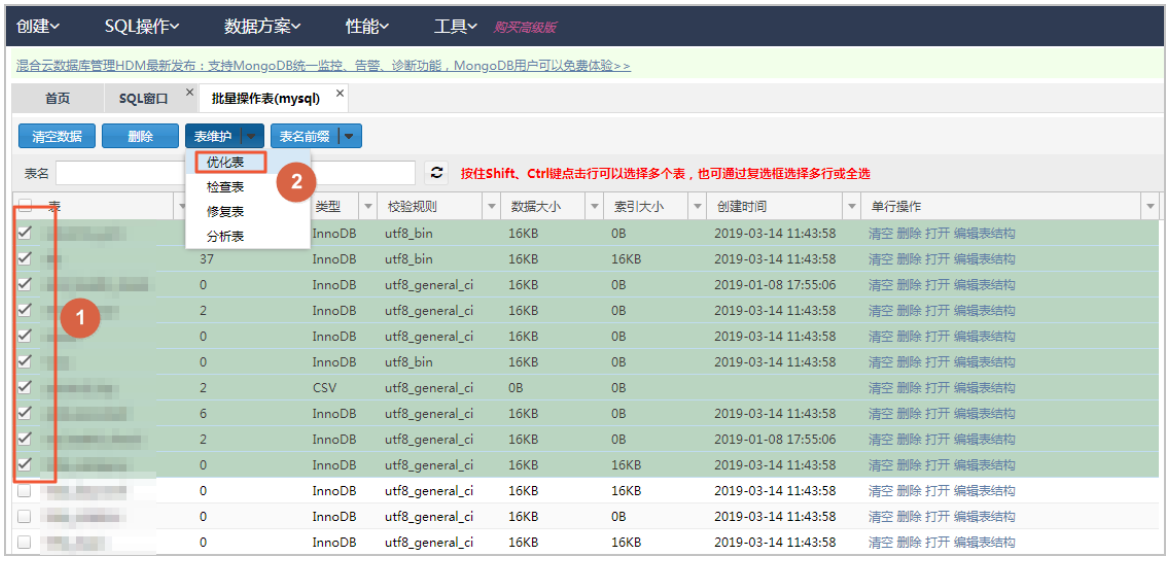
```
optimize table <数据库名1>.<表名1>,<数据库名2>.<表名2>
```



通过DMS操作

1. 通过DMS登录MySQL数据库。
2. 在左侧选择数据库，在任意表上单击鼠标右键，单击**批量操作表 > 更多批量操作**。

3. 在弹出的**批量操作表**中勾选需要释放空间的表，单击**表维护 > 优化表**。



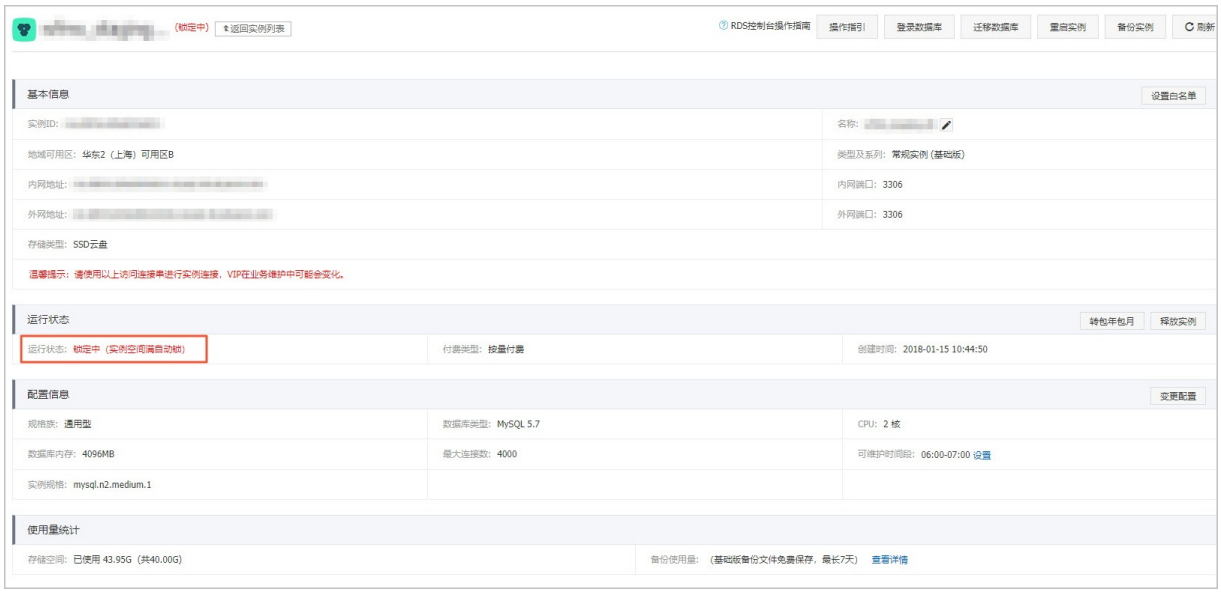
4. 在弹出的对话框中单击**Yes**。

7.2. MySQL数据文件导致实例空间满的解决办法

MySQL实例可能会由于数据文件长时间未整理导致实例空间满，为避免数据丢失，RDS会对实例进行自动锁定，磁盘锁定之后，将无法进行写入操作。

背景信息

当实例由于实例空间满自动锁定时，控制台可以在**基本信息 > 运行状态**看到如下信息。



前提条件

- 对于MySQL 5.6/5.7/8.0版本的实例，请升级内核小版本（需要版本号高于20190815，支持实例锁优化），升级后即可执行删除数据的操作。
- 对于MySQL 5.5版本的实例，请提交工单联系客服临时解锁实例，再进行后续操作。

实施步骤

注意事项

- 删除表前请确保有数据备份，以免造成损失。
- 推荐使用 `drop` 或 `truncate` 命令释放空间。
- `optimize` 操作将会锁表，建议在业务低峰期操作。
- 清理数据文件有延迟，请耐心等待实例已使用空间的下降。

操作步骤

1. 通过 DMS登录数据库。
2. 选择**SQL操作 > SQL窗口**，执行如下命令查看数据库的文件大小，分析其中可以删除的历史数据文件或无用数据文件。

```
SELECT file_name, concat(TOTAL_EXTENTS,'M') as 'FIle_size' FROM INFORMATION_SCHEMA.FILES  order by TOTAL_EXTENTS DESC
```

3. 使用 `drop` 、 `truncate` 或者 `delete` 命令均可以清理数据。

   > **说明** 请确保已参照前提条件解锁实例。

   - **drop**：使用 `drop table <数据库名>.<表名>` 删除不需要的表。
   - **truncate**：使用 `truncate table <数据库名>.<表名>` 删除不需要的表。
   - **delete**：

     > **说明** 未临时解锁的实例不支持delete操作，仅支持drop和truncate。

     a. 使用 `delete from <数据库名>.<表名>` 删除表中数据。
     b. 使用 `optimize table <数据库名>.<表名>` 回收空间。

后续维护

对于经常有 `delete` 操作的表，容易产生数据空洞，可以在业务低峰期使用 `optimize table <数据库名>.<表名>` 回收空间。

更多信息

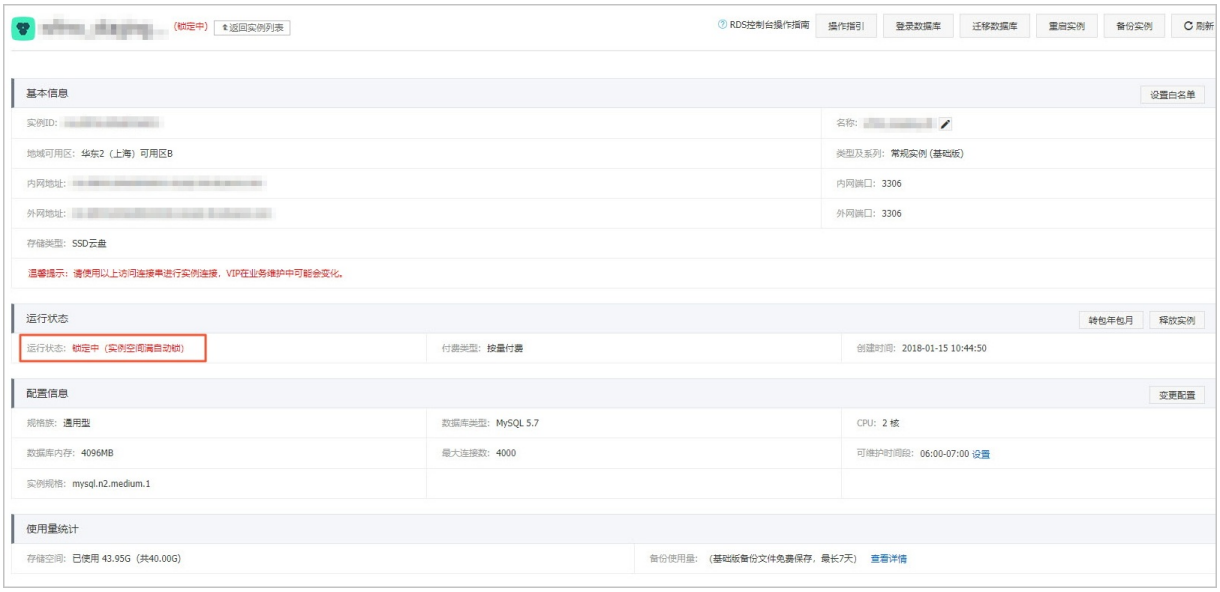
- MySQL Binlog文件导致实例空间满的解决办法
- MySQL临时文件导致实例空间满的解决办法
- MySQL系统文件导致实例空间满的解决办法

7.3. MySQL Binlog文件导致实例空间满的解决办法

MySQL实例可能会由于大事务快速生成Binlog文件，导致实例空间满，为避免数据丢失，RDS会对实例进行自动锁定，磁盘锁定之后，将无法进行写入操作。

背景信息

当实例由于实例空间满自动锁定时，控制台可以在**基本信息 > 运行状态**看到如下信息。



前提条件

- 对于MySQL 5.6版本的实例，升级实例存储空间后即可解锁实例，关于如何升级实例配置，请参见变更配置，若实例存储空间已到最大值，请提交工单联系客服临时解锁实例，再进行后续操作。
- 对于MySQL 5.5/5.7版本的实例，请提交工单联系客服临时解锁实例，再进行后续操作。

实施步骤

注意事项

- Binlog 文件记录实例的事务信息，是MySQL实例高可用性、可恢复性的基础，建议不要关闭。可以在**备份恢复**页面通过**一键上传Binlog**到阿里云OSS来释放磁盘空间或者修改本地Binlog设置。
- 清理Binlog文件有延迟，请耐心等待实例已使用空间的下降。
- 一键上传Binlog会在后台异步提交清理任务，清理任务会将已写入的Binlog上传到OSS （非用户购买的OSS）上，然后再从实例空间中删除Binlog文件，当前正在被写入的Binlog文件由于未完成写入，是不可以被清理的。因此，清理过程会有一定延迟，建议您单击**一键上传 Binlog**后耐心等待一定时间，请勿多次单击该按钮，可以到基本信息页中查看已用空间是否减小。
- 由于DML等操作（例如涉及大字段的 DML 操作）导致快速生成Binlog，可能会出现上传Binlog文件到备份空间并且从实例空间中删除的处理速度跟不上实例生成Binlog文件的速度，在这种情况下，建议考虑升级磁盘空间，并且排查Binlog快速生成的原因。

一键上传Binlog

1. 登录RDS管理控制台。
2. 在页面左上角，选择实例所在地域。
3. 找到目标实例，单击实例ID。
4. 在左侧导航栏中单击**备份恢复**。
5. 在右上角单击**一键上传Binlog**，在弹出的对话框中单击**确定**。

修改本地Binlog设置

1. 登录RDS管理控制台。
2. 在页面左上角，选择实例所在地域。
3. 找到目标实例，单击实例ID。
4. 在左侧导航栏中单击**备份恢复**。
5. 选择**本地日志设置**页签，单击**编辑**，将**空间使用率不超过**设置为*30%*，**可用空间保护**设置为*开启*。

> **说明**
>
> - **保留时长**为每隔多久上传一次Binlog日志到OSS，并清理本地Binlog日志。若设置为0，表示本地不保存Binlog日志，直接上传到OSS。
> - 本地Binlog空间使用率 = 本地Binlog大小 / 实例总可用（购买）空间大小。此为循环使用空间，超出后，则从最早的Binlog开始清理，直到空间使用率低于该比例。
> - 当实例总空间使用率超过80%或剩余空间不足5GB时，会强制清理Binlog。从最早的开始清理，直到总空间使用率降到80%以下且剩余空间大于5GB。

更多信息

- MySQL数据文件导致实例空间满的解决办法
- MySQL临时文件导致实例空间满的解决办法

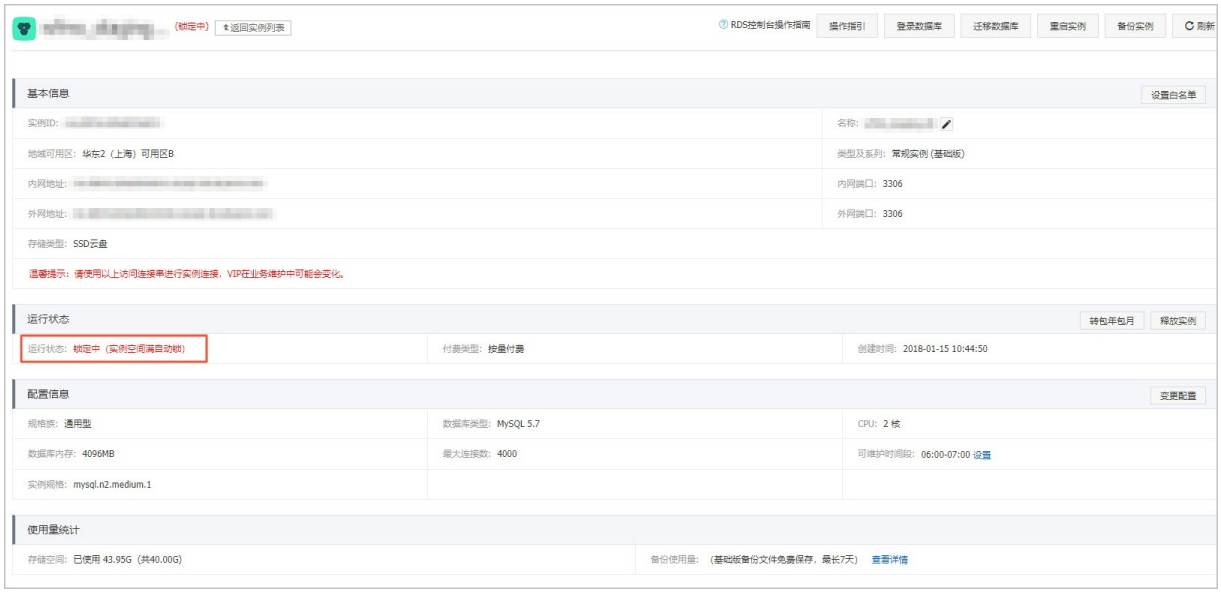
- MySQL系统文件导致实例空间满的解决办法

7.4. MySQL临时文件导致实例空间满的解决办法

MySQL实例可能会由于查询语句的排序、分组、关联表产生的临时表文件，或者大事务未提交前产生的binlog cache文件，导致实例空间满，为避免数据丢失，RDS会对实例进行自动锁定，磁盘锁定之后，将无法进行写入操作。

背景信息

当实例由于实例空间满自动锁定时，控制台可以在**基本信息 > 运行状态**看到如下信息：



前提条件

- 对于MySQL 5.6版本的实例，升级实例存储空间后即可解锁实例，关于如何升级实例配置，请参见变更配置，若实例存储空间已到最大值，请提交工单联系客服临时解锁实例，再进行后续操作。
- 对于MySQL 5.5/5.7版本的实例，请提交工单联系客服临时解锁实例，再进行后续操作。

实施步骤

注意事项

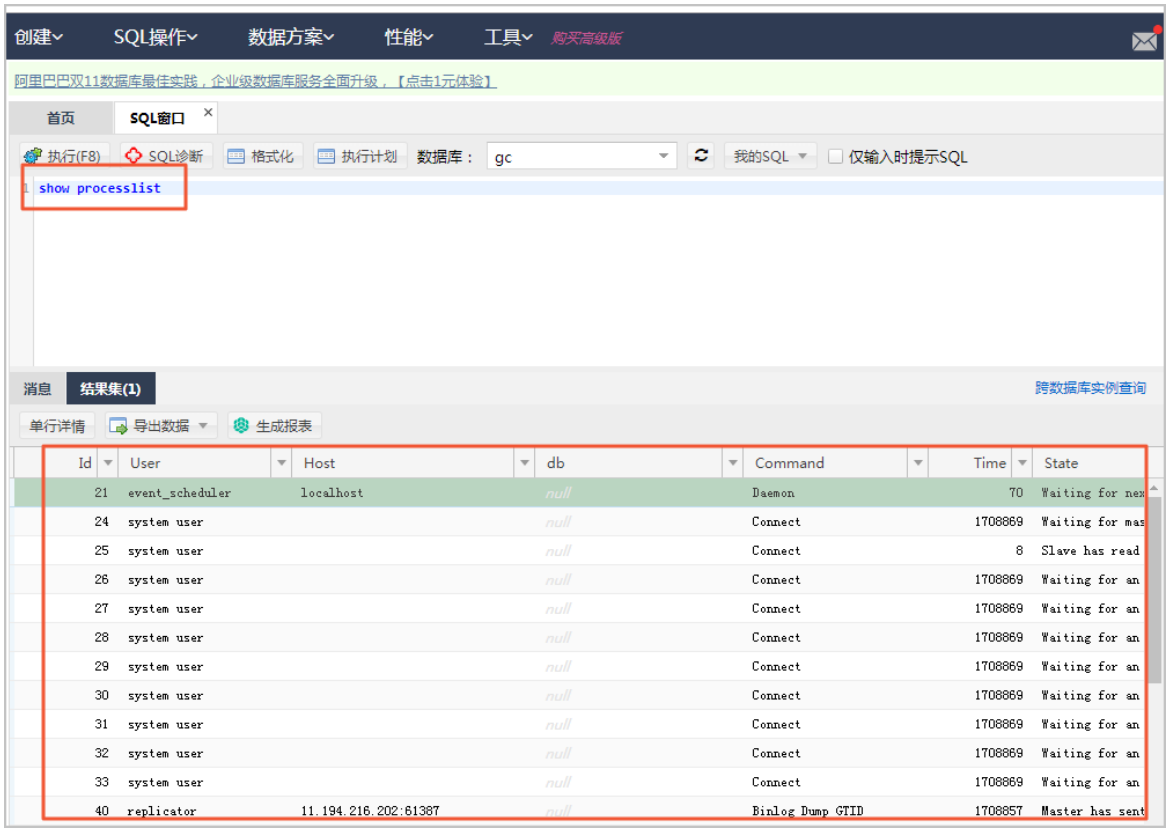
清理临时文件有延迟，请耐心等待实例已使用空间的下降。

操作步骤

1. 通过 DMS登录数据库。
2. 选择**SQL操作 > SQL窗口**，执行如下命令查看数据库的会话。

   ```
   show processlist
   ```

3. 单击显示结果中的State进行状态排序，在状态栏查看是否有大量 Copy to tmp table 、Sending data 等信息，再根据Info列的语句确定是哪个SQL语句在建立临时表，记下该语句的Id。



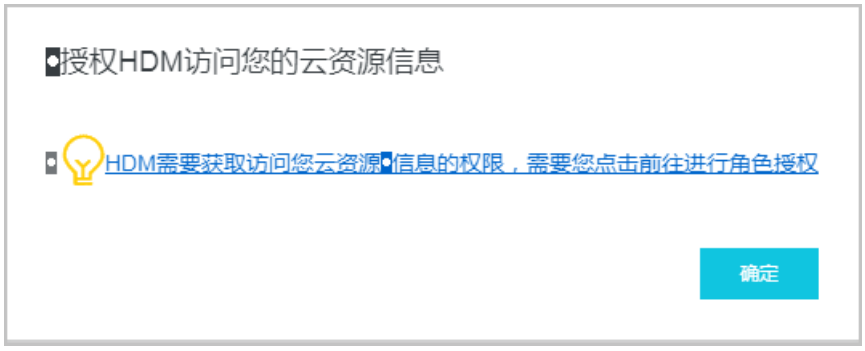
4. 可以通过命令行或者混合云数据库管理平台HDM（Hybrid Cloud Database Management）来终止会话。
   - 命令行：
     a. 在**SQL窗口**执行如下命令终止会话。

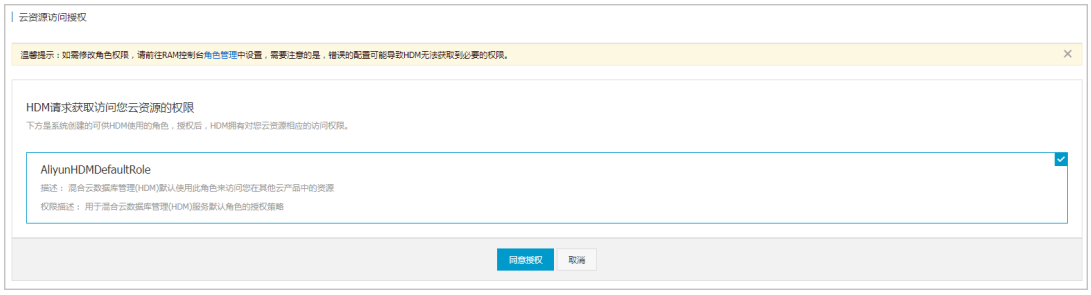
     ```
     kill <会话Id>
     ```

   - 混合云数据库管理平台：
     a. 选择**性能 > 空间**，跳转到混合云数据库管理平台（Hybrid Cloud Database Management，HDM）。

     > **说明** 对于第一次进入的用户需要对HDM进行授权，若已授权，请跳到第4步。

b. 在**授权HDM访问您的云资源信息**页面单击**确定**，并在弹出的**云资源访问授权**页面单击**同意授权**。



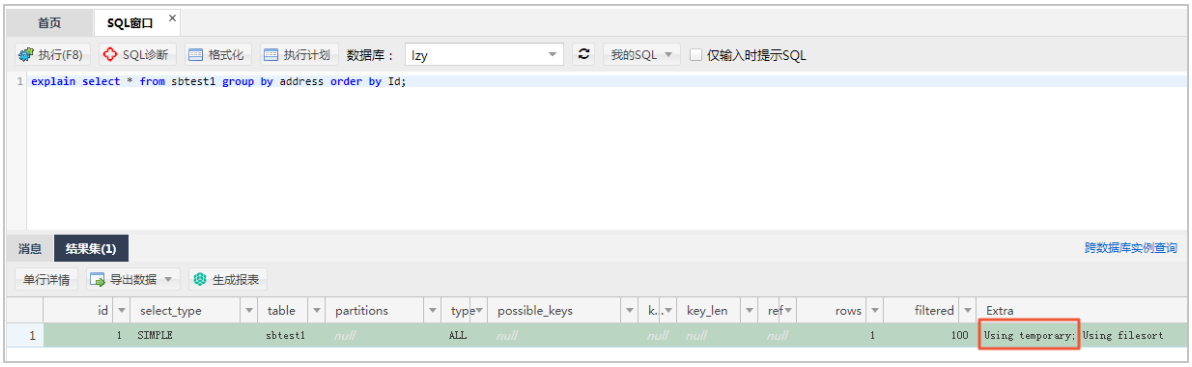


c. 在**实例会话**中找到之前记下的Id，单击该行任意位置选中该行，在右上角单击**kill选中会话**。

后续维护

- 针对查询产生的临时文件，应该优化SQL语句，避免频繁使用 order by、group by 操作，可以适当调大tmp_table_size和max_heap_table_size，但是为了减少磁盘使用而调高 tmp_table_size 和 max_heap_table_size 并不明智，因为内存资源远比磁盘资源宝贵；可以通过 `explain` +SQL语句查看是否使用内部临时表，在 Extra 字段中有 Using temporary 字样的代表会使用内部临时表。示例如下：

```
explain select * from alarm group by created_on order by default;
```



- 针对binlog cache，应该少执行大事务，尤其应该减少在多个连接同时执行大事务，如果大事务比较多，可以适当调大binlog_cache_size，但是同样不应该为了节省磁盘调整这个参数，使用短连接执行大事务可以有效降低临时空间开销。

更多信息

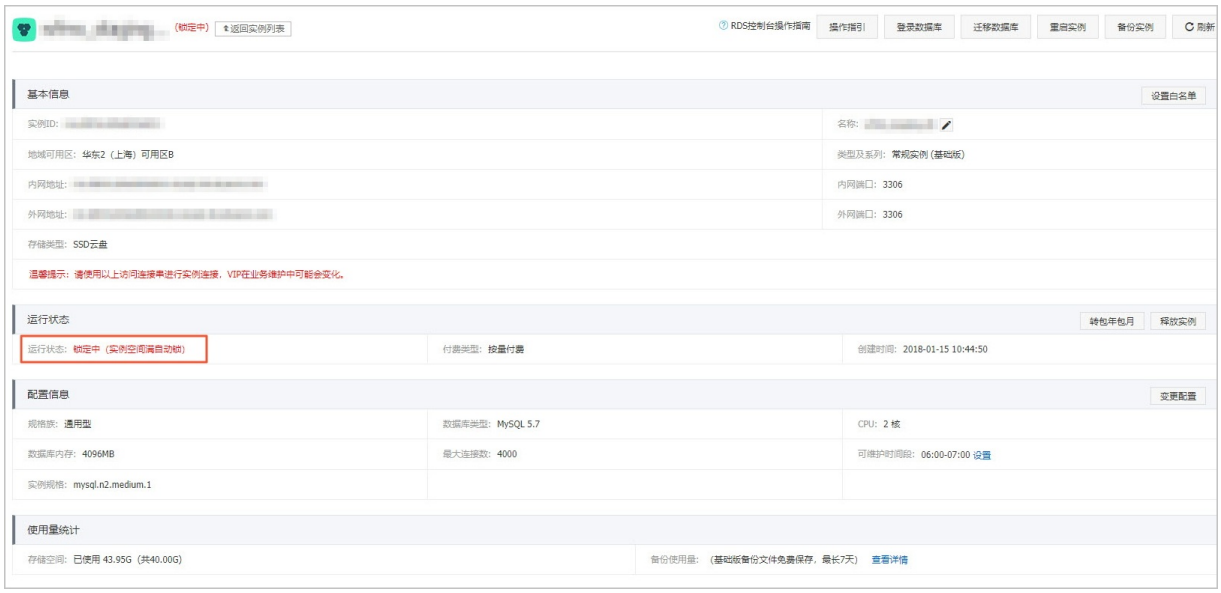
- MySQL数据文件导致实例空间满的解决办法
- MySQL Binlog文件导致实例空间满的解决办法
- MySQL系统文件导致实例空间满的解决办法

7.5. MySQL系统文件导致实例空间满的解决办法

MySQL实例可能会由于长时间不结束的查询导致 ibdata1 文件过大且无法收缩，导致实例空间满，为避免数据丢失，RDS会对实例进行自动锁定，磁盘锁定之后，将无法进行写入操作。

背景信息

当实例由于实例空间满自动锁定时，控制台可以在**基本信息 > 运行状态**看到如下信息。



前提条件

- 对于MySQL 5.6版本的实例，升级实例存储空间后即可解锁实例，关于如何升级实例配置，请参见变更配置，若实例存储空间已到最大值，请提交工单联系客服临时解锁实例，再进行后续操作。
- 对于MySQL 5.5/5.7版本的实例，请提交工单联系客服临时解锁实例，再进行后续操作。

实施步骤

注意事项

- 清理临时文件有延迟，请耐心等待实例已使用空间的下降。
- 由于MySQL 5.7开始采用独立的临时表空间ibtmp1，可以通过重启实例的方式释放空间。对于MySQL5.5/5.6实例，在不升级磁盘空间的前提下，比较好的解决方法是在同地域同可用区购买相同配置的RDS实例，通过 DTS 工具将数据迁移到新实例中。

操作步骤

1. 在同地域同可用区购买相同配置的RDS实例，具体请参见创建RDS MySQL实例。
2. 登录RDS管理控制台。
3. 在右上角单击**迁移数据库**。

4. 具体迁移配置请参见RDS实例间的数据迁移。

后续维护

- 避免出现执行效率很差的SQL大量执行的情况。
- 尽量在业务低峰期进行索引创建删除、表结构修改、表维护和表删除操作。
- 建议您监控和清理执行时间过长的会话或事务。

更多信息

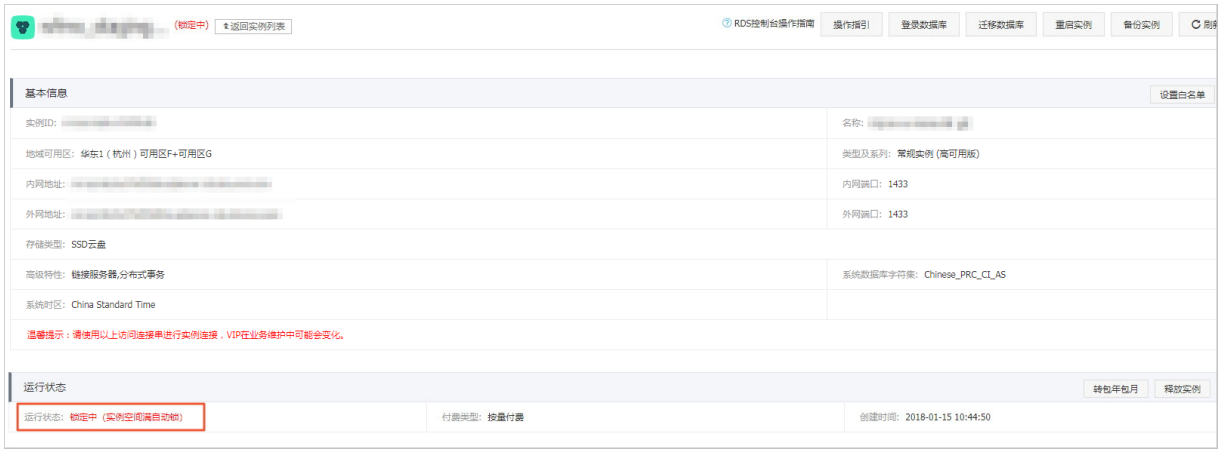
- MySQL数据文件导致实例空间满的解决办法
- MySQL Binlog文件导致实例空间满的解决办法
- MySQL临时文件导致实例空间满的解决办法

7.6. 解决SQL Server实例空间满自动锁的问题

SQL Server实例可能会由于SQL语句、外部攻击等原因导致实例空间满，为避免数据丢失，RDS会对实例进行自动锁定，磁盘锁定之后，将无法进行写入操作。

背景信息

当实例由于实例空间满自动锁定时，控制台可以在**基本信息 > 运行状态**看到如下信息：



本文将介绍造成实例空间满的常见原因及其相应的解决方法。

常见原因

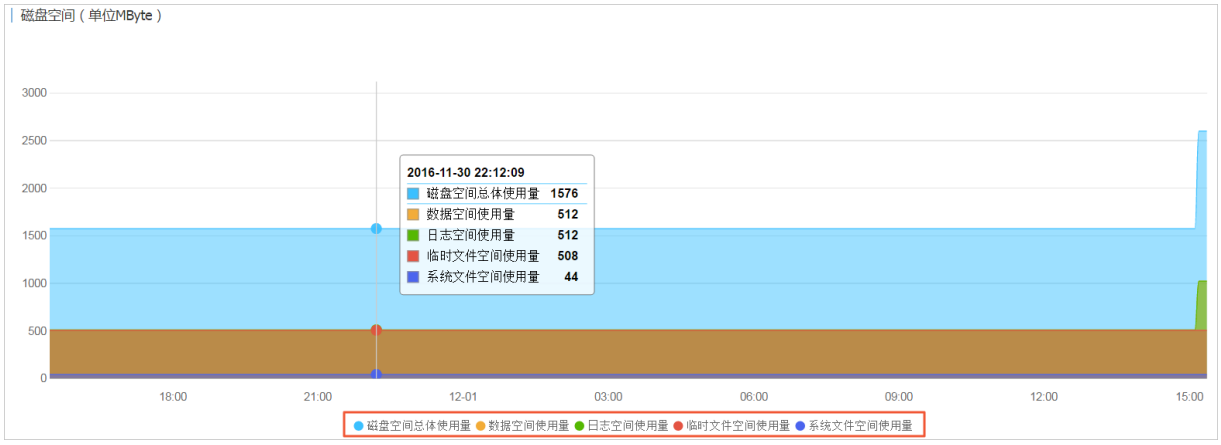
造成SQL Server实例空间满的主要有如下三种原因：

- 日志文件占用量高。
- 数据文件占用量高。
- 临时文件占用量高。

查看空间使用状况

方法一

通过RDS管理控制台的监控页面查看空间使用情况，详情请参见查看资源和引擎监控



参数	说明
磁盘空间总体使用量	所有用户数据库的数据文件和日志文件的大小。
数据空间使用量	所有用户数据库的数据文件（mdf和ndf文件）的大小。
日志空间使用量	所有用户数据库的日志文件（ldf文件）。
临时文件空间使用量	tempdb的所有mdf、ndf和ldf文件的大小。

参数	说明
系统文件空间使用量	master、msdb和model数据库的数据文件和日志文件，以及SQL Server实例目录下面的一些系统文件（错误日志和dll文件等）的大小。

方法二

通过SQL语句查看所有数据库的数据文件（mdf和ndf文件）和日志文件（ldf文件）的大小，详情请参见RDS for SQL Server如何查看实例、数据库及表占用的空间大小。

解决方法

- **升级实例的存储空间**

  升级实例存储空间后即可解锁实例，关于如何升级实例存储空间，请参见变更配置，若实例存储空间已到最大值，请提交工单联系客服临时解锁实例，再进行后续操作。

- **日志文件占用量高的解决方法**

  原因

  如果应用程序中有大量的大事务操作，就会导致事务日志持续增长，并且有可能会导致超过实例磁盘空间上限而使实例被锁定。

  解决方法一

  i. 客户端连接实例后执行如下命令。

  ```
  select name,log_reuse_wait,log_reuse_wait_desc from sys.databases
  ```

  ii. 若log_reuse_wait_desc的值是LOG_BACKUP，请收缩事务日志。

  > **说明** 若日志文件非常大，日志备份的时间会比较长，并且在收缩日志文件时，如果遇到未提交的事务，会导致单次收缩效果不明显。在单次收缩效果不明显的情况下，建议您再次收缩事务日志。

  解决方法二

  事务日志增长过快的根本原因是事务较多或者有大事务。例如，一个事务中操作了500万行数据，在有这种大事务的情况下，建议您将事务拆分，每个事务操作10万行数据，分50次执行。

- **数据文件占用量高的解决方法**

  如果数据库文件占用空间比较多，可以先检查数据文件的使用率。对于文件大但使用率低的数据库，可以进行相应处理。详细步骤如下：

  i. 执行如下命令，查看数据库的空闲空间。

  ```
  USE <数据库名>
  GO
  SELECT SUM(unallocated_extent_page_count) AS [free pages],
  (SUM(unallocated_extent_page_count)*1.0/128) AS [free space in MB]
  FROM sys.dm_db_file_space_usage;
  ```

  ii. 找到空间使用率较高的数据库，然后执行如下命令，收缩该数据库。

  ```
  DBCC SHRINKDATABASE(<数据库名>)
  ```

> **说明** 您也可以执行如下命令来收缩单个文件。
>
> ```
> DBCC SHRINKFILE(file_id,size)
> ```
>
> size为收缩以后的大小，而不是要收缩多少，单位：MB。

- **临时文件用量高的解决方法**

  您可以从实例监控中初步判定临时文件是否占用太多空间。另外，如果临时文件的空间不够，Error Log中也会有相应的记录。关于如何排查临时文件空间不足的情况，请参见Troubleshooting Insufficient Disk Space in tempdb。

  建议您执行如下操作：

  - 重启实例来快速释放临时文件的空间。
  - 及时释放临时表、行版本、表变量等。

7.7. RDS for SQL Server如何查看实例、数据库及表占用的空间大小

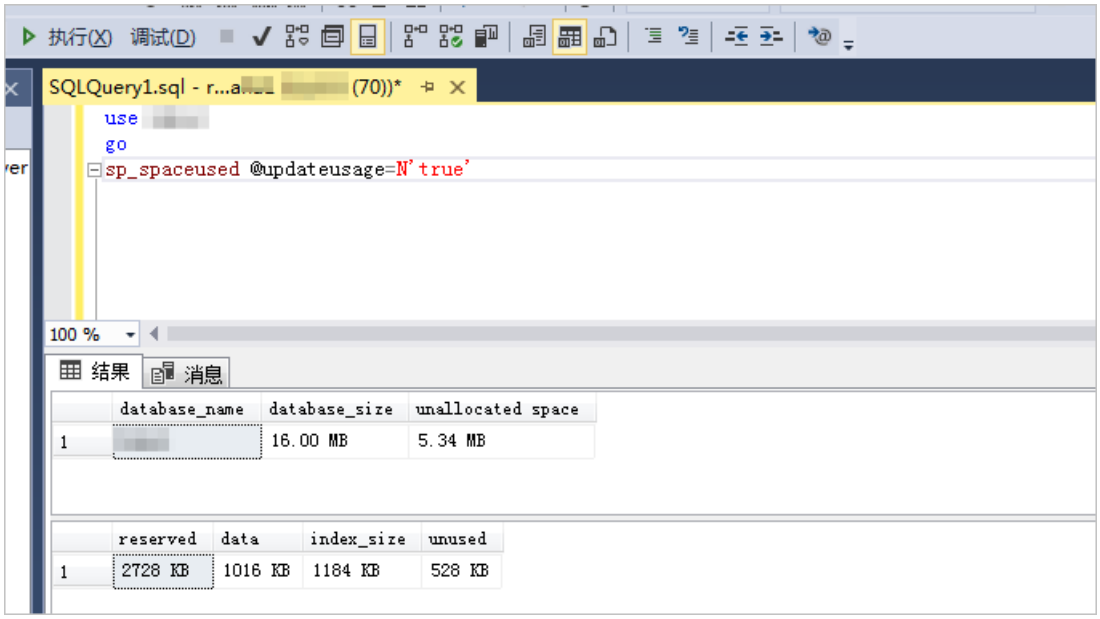
查看实例空间的大小

在控制台**基本信息**页面查看实例空间情况。

查看数据库的大小

1. 使用客户端连接实例，请参见连接SQL Server实例。
2. 在SQL窗口中执行以下命令。

```
use <数据库名>
go
sp_spaceused @updateusage=N'true'
```



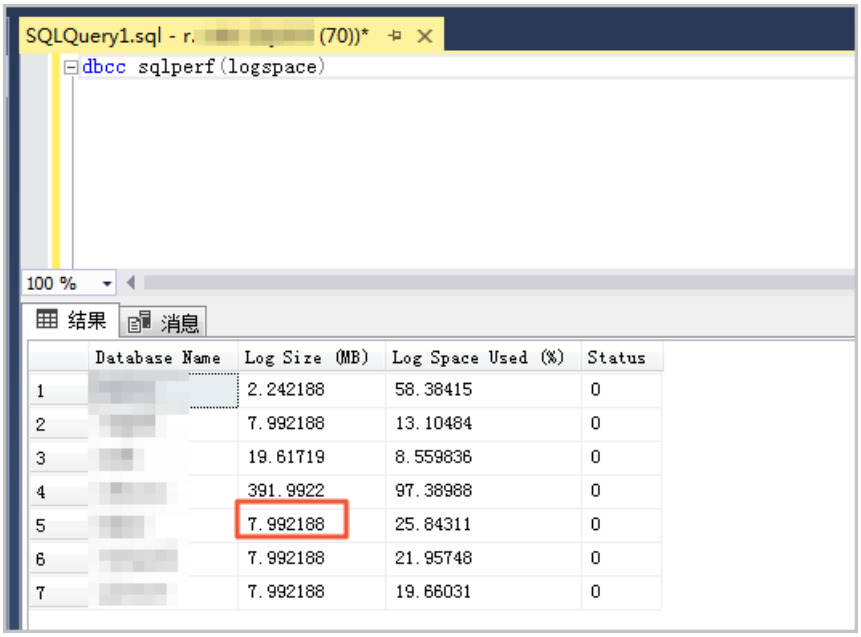
参数	说明
database_size	即数据库的大小，它包含日志的大小，始终大于reserved+unallocated_space。
unallocated space	数据库的未分配空间。
reserved	已分配的空间总量。
data	数据使用的空间总量。
index_size	索引使用的空间。
unused	未用的空间量。

说明 查看所有的数据库，则需要使用脚本来实现，脚本如下：

```
USE master
go
DECLARE @insSize TABLE(dbName sysname,checkTime VARCHAR(19),dbSize VARCHAR(50),logS
ize VARCHAR(50))
INSERT INTO @insSize ( dbName, checkTime, dbSize, logSize )
EXEC sp_msforeachdb 'select ''?'' dbName,CONVERT(VARCHAR(19),GETDATE(),120) checkTi
me,LTRIM(STR(SUM(CASE WHEN RIGHT(FILENAME,3)<>''ldf'' THEN convert (dec (15,2),size
) * 8 / 1024 ELSE 0 END),15,2)+'' MB'') dbSize,
                LTRIM(STR(SUM(CASE WHEN RIGHT(FILENAME,3)=''ldf''  THEN convert (d
ec (15,2),size) * 8 / 1024 ELSE 0 END),15,2)+'' MB'') logSize from ?.dbo.sysfiles'
SELECT * FROM @insSize ORDER BY CONVERT(DECIMAL,LTRIM(RTRIM(SUBSTRING(dbSize,1,LEN(
dbSize)-2)))) DESC
```

该结果未包含日志文件的大小，查看日志文件的大小的SQL语句是：

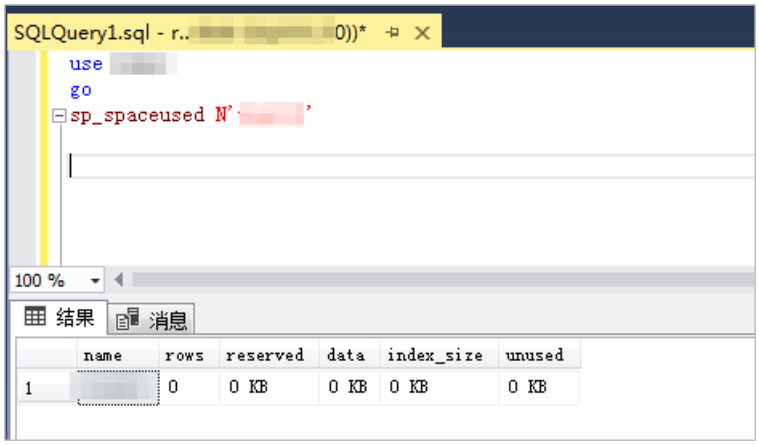
```
dbcc sqlperf(logspace)
```



查看数据库中表的大小

1. 使用客户端连接实例，请参见连接SQL Server实例。
2. 在SQL窗口中执行以下命令。

```
use <数据库名>
go
sp_spaceused N'<表名>'
```



> **说明** 查看该库所有的表，则需要使用脚本来实现，脚本如下：

```
use <数据库名>
go
declare @tabSize table(name nvarchar(100),rows char(20),reserved varchar(18) ,data
varchar(18) ,index_size varchar(18) ,nnused varchar(18) )
insert into @tabSize exec sp_msforeachtable 'sp_spaceused ''?'''
select * from @tabSize order by convert(int,replace(data,'KB','')) desc,2 desc
```

7.8. SQL Server数据库空间查看工具

功能说明： 查询SQL Server数据库空间的工具。

使用场景： 查询阿里云上SQL Server数据库空间。

使用系统： Windows操作系统。

注意事项：

1. 录入的用户必须具有所有数据库的读写权限，否则只能查看有权限的数据库空间大小；
2. 总空间=库空间+日志空间；
3. 库空间=已分配空间+未分配空间；
4. 已分配空间=数据库空间+日志空间+未使用空间；
5. 未使用空间比率=（已分配未使用空间+未分配空间）/总空间；
6. 为避免占用资源影响业务，建议在业务空闲期查看。

使用方法： 输入**数据库地址**、**端口号**、**用户名**、**密码**，点击**查看所有数据库空间**按钮。

工具下载： 点击下载

7.9. RDS for SQL Server查看内存占用情况

操作步骤

1. 使用客户端连接实例，请参见连接SQL Server实例。
2. 在SQL窗口中执行以下命令。

```
select count(*)*8/1024 as 'cache size(MB)',
case database_id
when 32767 then 'ResourceDb'
else DB_NAME(database_id)
end as 'datebase'
from sys.dm_os_buffer_descriptors
group by DB_NAME(database_id),database_id
order by 'cache size(MB)' desc
```

7.10. RDS for SQL Server 回收表空间

RDS for SQL Server在删除变长列或者减小变长列的长度后，表的大小不会自动减小。

变长列包括的字段：varchar、nvarchar、varchar(max)、nvarchar(max)、varbinary、text、ntext、image、sql_variant、varbinary(max)、xml。

解决方法

- SQL Server 提供了一个 `DBCC CLEANTABLE` 命令可以回收表或索引视图中，已删除可变长度列的空间。语法如下：

```
DBCC CLEANTABLE
(
    { database_name | database_id | 0 }
    , { table_name | table_id | view_name | view_id }
    [ , batch_size ]
)
[ WITH NO_INFOMSGS ]
```

说明

- database_name | database_id | 0：要清除的表所在的数据库。如果指定0，则使用当前数据库。
- table_name | table_id | view_name | view_id：要清除的表或索引视图。
- batch_size：每个事务处理的行数。如果未指定，或指定为0，则该语句将在一个事务中处理整个表。
- WITH NO_INFOMSGS：取消显示所有信息性消息。

示例

```
DBCC CLEANTABLE (testDB,'testTable', 0)
WITH NO_INFOMSGS;
GO
```

- 重建索引或者reorganized索引。

空间是不会自动回收的，每个记录都占了一个位置。即使删除了数据，位置也会空在那里，下次插入记录的时候，会优先选这些空的槽位。您可以考虑定时重建聚集索引。另外，即使收缩了表的空间，数据库的数据文件大小不会变小。要收缩一个SQLServer的数据文件，必须用DBCC SHRINKDATABASE命令收缩指定数据库的指定数据或日志文件大小，或者用DBCC SHRINKFILE收缩当前数据库的指定数据或日志文件大小。MySQL表的空间是独立的一个文件，所以收缩MySQL的大表，可以收缩整体数据库的大小，但是SQL Server所有的表都是在数据库的文件里，只有收缩文件才可以缩小空间。

7.11. PostgreSQL/PPAS磁盘空间占用剧增后下降情况解析

情况一

原因

大量更新导致日志剧增，系统来不及归档和删除，占用了磁盘空间。

解决方案

提高实例的磁盘空间容量或降低更新频率。

情况二

原因

查询操作含有大量的排序、连接等操作，处理过程中产生临时表并溢出到磁盘，短时间内造成大量空间占用。

解决方案

- 对于PostgreSQL用户，通过RDS初始帐号执行如下命令：

```
alter role all set temp_file_limit = <临时表空间上限>;
```

- 对于PPAS用户，通过RDS初始帐号执行如下命令：

```
select rds_set_conf_for_all_roles('temp_file_limit', '<临时表空间上限>');
```

说明 以上命令用于指定每个查询可以使用的临时表空间上限（单位为KB），执行成功后，单个查询生成的临时表空间达到上限就会报错。这样就能及时发现有问题的SQL语句，并避免磁盘空间被占满。

示例

```
alter role all set temp_file_limit = 512000;
```

```
select rds_set_conf_for_all_roles('temp_file_limit', '512000');
```

7.12. 使用OSS扩展PostgreSQL/PPAS的表空间

使用OSS扩展PostgreSQL/PPAS的表空间，打破2TB空间限制。

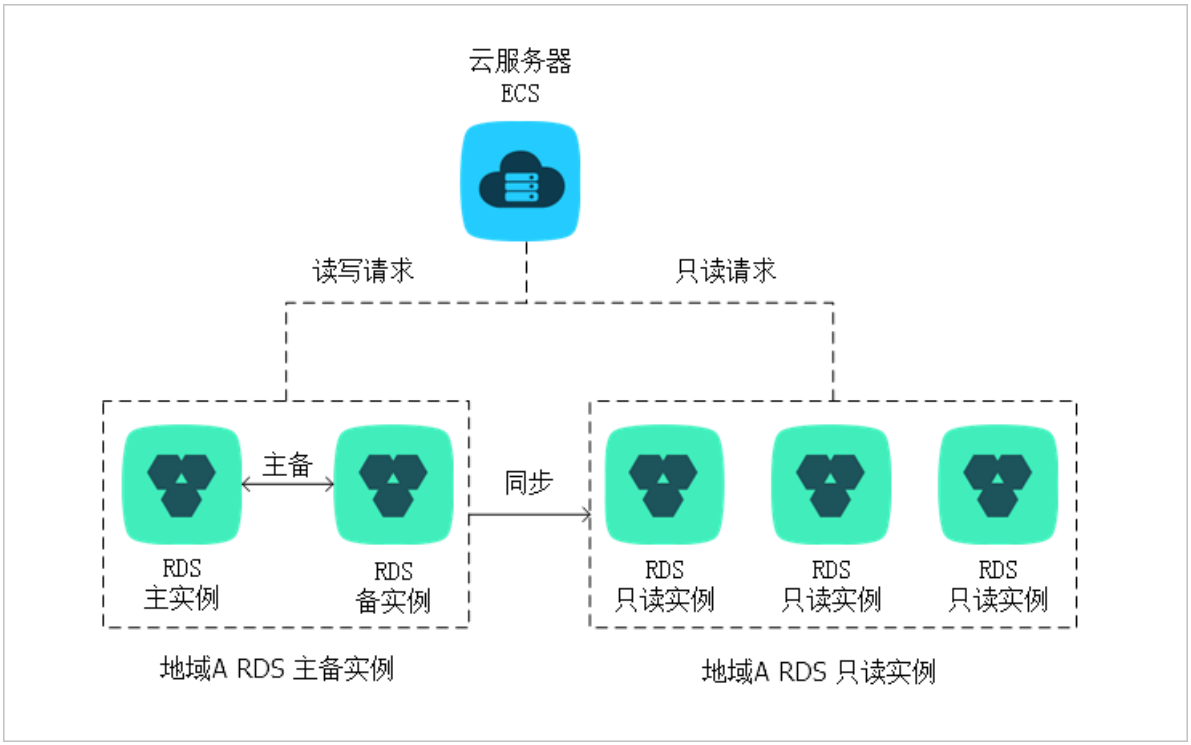
请参见读写外部数据文本文件（oss_fdw）。

8.读写分离/只读实例

8.1. RDS for MySQL只读实例同步延迟原因与处理

只读实例延迟介绍

RDS for MySQL只读实例通常用于分担主实例的查询压力，或者用于运行OLAP类型的分析应用，避免复杂统计查询对主实例的性能影响。



由于RDS只读实例采用MySQL原生的基于Binlog的复制技术（异步复制或半异步复制），必然会有延迟。延迟会导致只读实例与主实例的数据出现不一致，进而导致业务出现问题。另外延迟也有可能引起Binlog堆积，导致只读实例空间被迅速消耗（如果主实例当前正产生大量的Binlog），这种情况下有可能会使只读实例被锁定。

只读实例产生延迟的原因及解决方案

- **只读实例规格过小**

  分析

  这类延迟场景经常出现在只读实例规格和主实例规格相差较大，而且只读实例上负载较重，比如只读实例上IOPS打满。

  只读实例的数据为了和主实例保持同步，采用了MySQL原生的binlog复制技术，由一个IO线程和一个SQL线程来完成。IO线程负责将主实例的binlog拉取到只读实例，SQL线程负责将这些binlog日志应用到只读实例。这两个线程会消耗只读实例的IO资源，所以当只读实例的IOPS配置不够的时候，会导致只读实例的数据出现延迟。

  解决方案

建议您升级只读实例规格，避免由于只读实例规格较小导致延迟。RDS推荐只读实例的配置大于或者等于主实例的配置。

- **主实例的TPS（Transaction Per Second）过高**

  分析

  由于只读实例与主实例同步采用的是单线程同步，而主实例的压力是并发多线程写入，这样在主实例TPS过高的情况下容易出现只读实例的数据延迟，可以通过观察只读实例的TPS与主实例的TPS性能数据来判断。

  解决方案

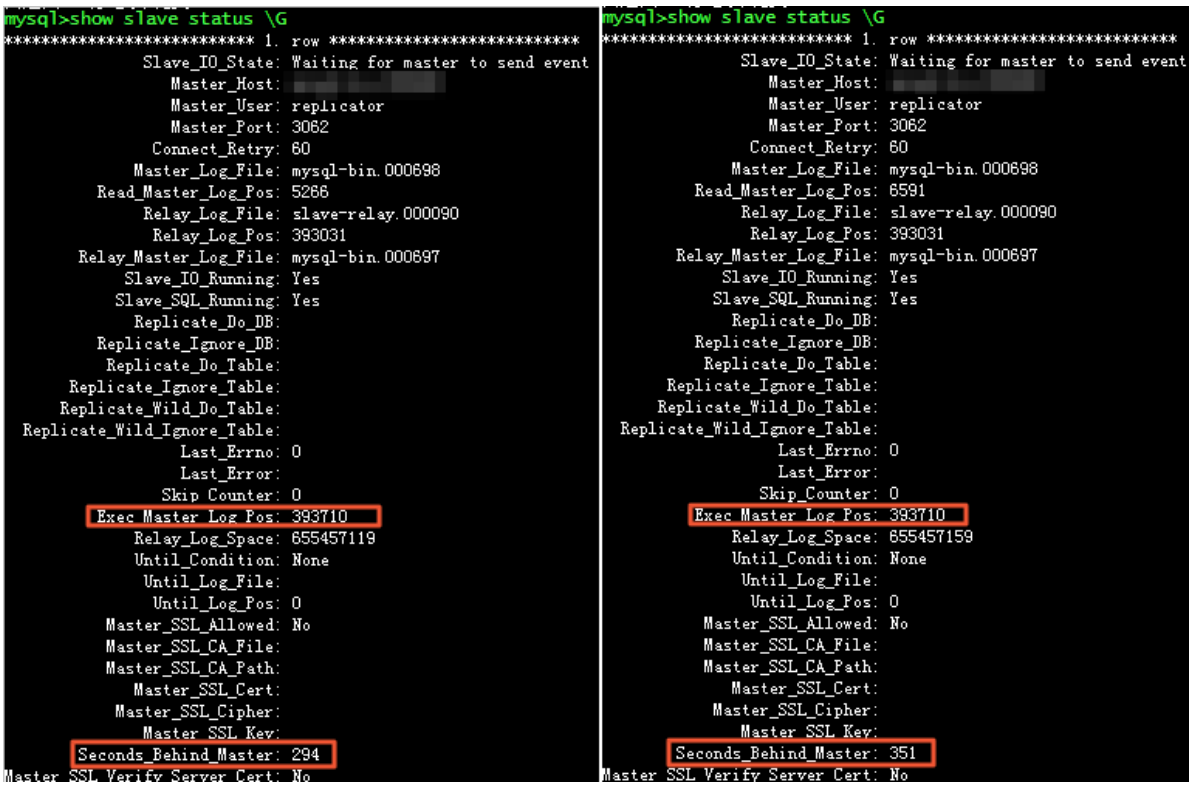
  排查主实例的TPS是否正常，如果正常则需要对业务进行优化或者拆分，保证主实例的TPS不会导致只读实例出现延迟。

- **主实例的大事务**

  分析

  主实例执行一个涉及数据量非常大的update、delete、insert...select、replace...select等事务操作，生成大量的binlog数据传送到只读实例。只读实例需要花费与主实例相同的时间来完成该事务，进而导致了只读实例的同步延迟。例如在主实例上执行一个持续80秒的删除操作，只读实例进行相同操作时也需要花费很长时间，这时就出现了延迟。

  在只读实例出现大事务导致延迟时，通过 `show slave status \G` 命令，可以看到Seconds Behind_Master不断变化，而Exec_Master_Log_Pos却保持不变，这样可以判断只读实例的SQL线程在执行一个大的事务或者DDL操作。



解决方案

建议将大事务拆分为小事务（例如在delete语句中增加where条件子句，限制每次删除的数据量，将一次删除操作拆分为多次数据量较小的删除操作进行），这样只读实例可以迅速的完成事务的执行，不会造成数据的延迟。

- **主实例的DDL语句执行时间长**

  分析

  - 只读实例和主实例数据同步是串行进行的，如果DDL操作在主实例执行时间很长，那么同样在只读实例也会消耗同样的时间导致延迟。常见操作例如create index、repair table、alter table add column等。
  - 只读实例上执行的查询或未完成的事务阻塞了来自主实例的DDL执行。在只读实例上执行 `show proceslist` 命令查看SQL线程的状态为**waiting for table metadata lock**。

  解决方案

  - 对于DDL直接引起的只读实例延迟，建议在业务低峰期执行这些DDL。
  - 对于来自主实例的DDL在只读实例上被阻塞的情况，需要kill掉只读实例上引起阻塞的会话来恢复只读实例和主实例的数据同步，详情请参见解决MDL锁导致无法操作数据库的问题。

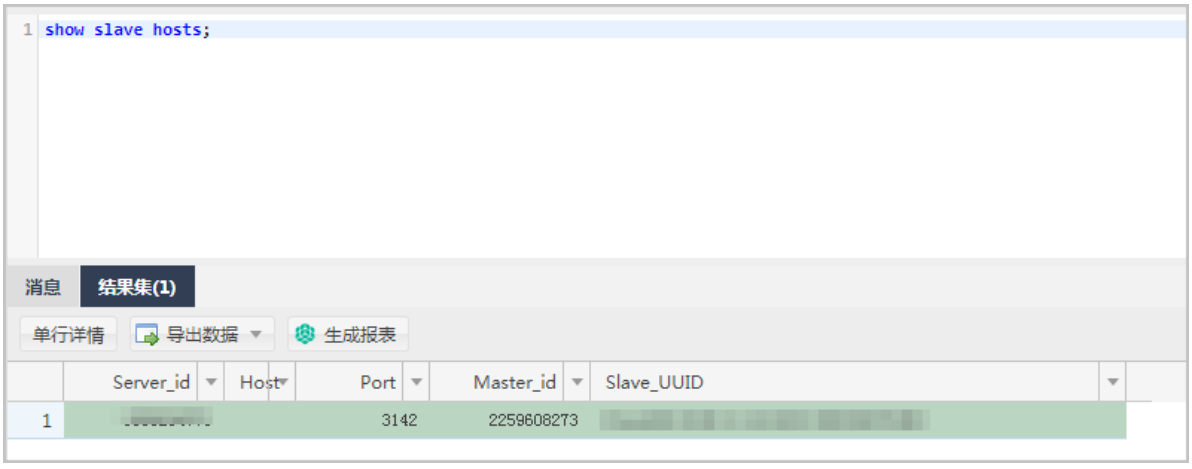
总结

当只读实例出现延迟的时候，排查方向如下：

- 控制台查看监控，检查只读实例的IOPS，确认只读实例是否存在资源瓶颈。
- 控制台查看监控，检查只读实例的MySQL_COMDML，确认主实例是否TPS过高。
- 检查只读实例的binlog增长量，确定是否存在大事务。
- 只读实例执行 `show slave status \G` ，确定是否有元数据锁。
- 控制台查看SQL慢日志，是否有alter、repair、create等DDL操作。
- 检查只读实例是否存在无主键表的删除或者更新操作，可以通过在只读实例执行 `show engine innodb status\G` 查看，或者执行 `show open tables` 后查看输出结果的**in_use**列里值为1的表。

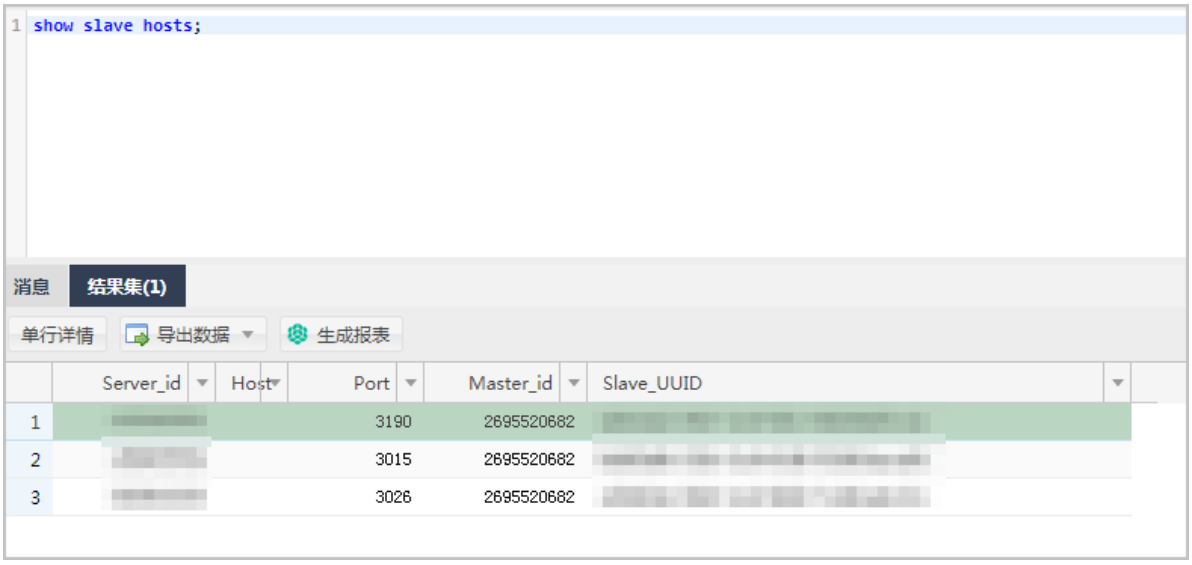
8.2. RDS for MySQL实例show slave hosts结果说明

- 当RDS实例单独使用时（即未使用只读实例），执行 `show slave hosts;` 结果如下。



显示的slave端是RDS实例的备实例，以确保RDS实例的高可用。

- 当RDS实例创建过只读实例，通过 `show slave hosts;` 显示三个slave端，如下图。



它们分别是：主实例的备实例、只读实例、只读实例的备实例，其中两个备实例用于保证RDS主实例与RDS只读实例的高可用。

9.错误代码

9.1. RDS for MySQL权限问题（错误代码：1227，1725）

错误信息

```
[Err] 1227 - Access denied; you need (at least one of) the SUPER privilege(s) for this oper
ation --常见于RDS MySQL 5.6
ERROR 1725 (HY000) at line 1936: OPERATION need to be executed set by ADMIN --常见于RDS MySQ
L 5.5
```

出现场景

- 在创建 存储过程、函数、触发器、事件、视图的时候出现这个错误。
- 从本地数据库导出SQL，在RDS上应用该SQL的时候出现该错误。
- 从RDS for MySQL 5.6实例下载逻辑备份，导入到RDS或本地数据库中。

错误原因

- 导RDS for MySQL实例时，SQL语句中含有需要Supper权限才可以执行的语句，而RDS for MySQL不提供Super权限，因此需要去除这类语句。
- 本地MySQL实例没有启用GTID。

解决办法

1. 去除DEFINER子句：
   i. 检查SQL文件，去除下面类似的子句。

   ```
   DEFINER=`root`@`%`
   ```

   ii. 在Linux平台下，可以尝试使用下面的语句去除。

   ```
   sed -e 's/DEFINER[ ]*=[ ]*[^*]*\*/\*/ ' your.sql > your_revised.sql
   ```

2. 去除GTID_PURGED子句
   i. 检查SQL文件，去除下面类似的子句。

   ```
   SET @@GLOBAL.GTID_PURGED='d0502171-3e23-11e4-9d65-d89d672af420:1-373,
   d5deee4e-3e23-11e4-9d65-d89d672a9530:1-616234';
   ```

   ii. 在Linux平台下，可以尝试使用下面的语句去除。

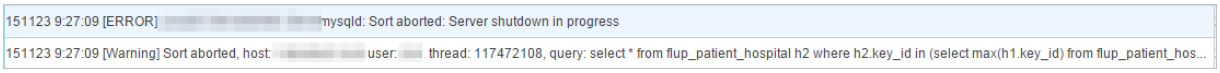
   ```
   awk '{ if (index($0,"GTID_PURGED")) { getline; while (length($0) > 0) { getline; }
   } else { print $0 } }' your.sql | grep -iv 'set @@' > your_revised.sql
   ```

   > **说明** 也可以导出的时候在mysqldump命令后添加参数--set-gtid-purged=off来取消输出GTID_PURGED子句。

9.2. mysqld:Sort aborted:Server shutdown in progress错误处理

错误信息

在控制台查看**日志管理 > 错误日志**，出现类似如下错误。



错误原因及解释

- 在查询执行排序的过程中实例进行了重启，导致查询中断。

  由于重启导致的查询中断是正常现象。

- 在查询执行排序的过程中，通过kill命令或者DMS实例会话功能终止了查询会话。

  这是由于MySQL的bug导致输出的错误信息，可以忽略。详情请参见MySQL BUG。

9.3. 安装ThinkSNS网站调用RDS数据库提示OPERATION need to be executed set by ADMIN

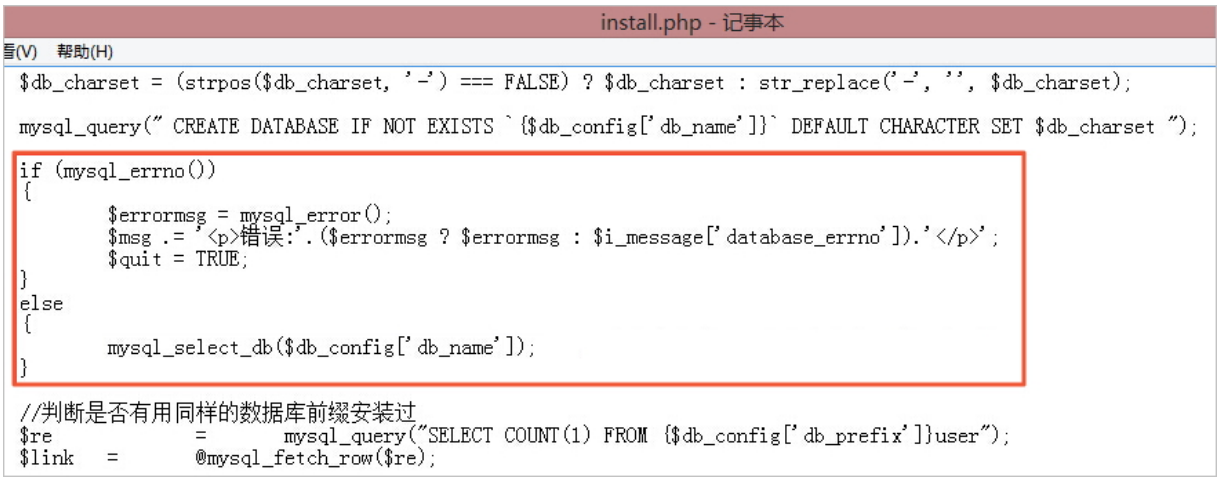
错误信息

在已经创建了数据库，以及授权操作账号的情况下，安装网站ThinkSNS调用阿里云RDS时，返回如下错误：

```
OPERATION need to be executed set by ADMIN
```

解决方案

找到安装包文件并进行备份，路径为*ThinkSNS_V4.1.170_Beta_Full\ThinkSNS_V4_Beta_Full\install\install.php*，在542行左右，找到以下内容。



将上图中红框内的代码修改为：

```
mysql_select_db($db_config['db_name']);
```

9.4. The maximum column size is 767 bytes 错误处理

错误信息

RDS for MySQL在大字段上创建索引时，有时会遇到以下错误：

```
ERROR 1709 (HY000): Index column size too large. The maximum column size is 767 bytes.
```

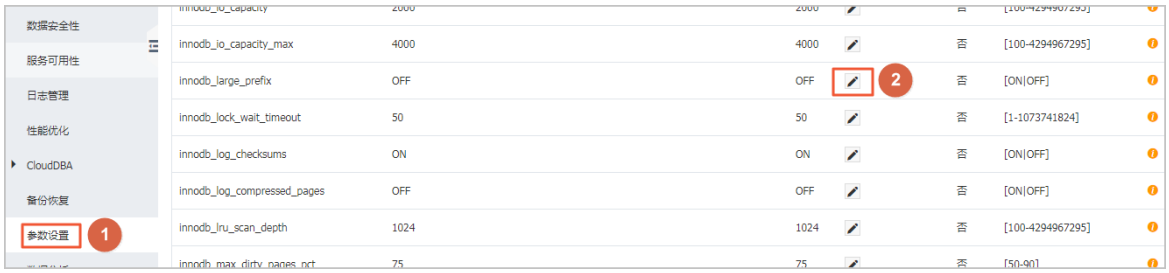
错误原因

由于MySQL的Innodb引擎表索引字段长度的限制为767字节，因此对于多字节字符集的大字段或者多字段组合，创建索引时会出现此错误。

以utf8mb4字符集字符串类型字段为例，utf8mb4是4字节字符集，则默认支持的索引字段最大长度是191字符（767字节/4字节每字符≈191字符），因此在varchar(255)或char(255)类型字段上创建索引会失败。详情请参见MySQL官网文档。

解决方案

1. 在控制台的**参数设置**里修改参数**innodb_large_prefix**为ON或者1，然后单击**提交参数**。



> **说明** 将innodb_large_prefix修改为ON或者1后，对于Dynamic和Compressed格式的InnoDB引擎表，其最大的索引字段长度支持到3072字节。

2. 创建表的时候指定表的row_format格式为Dynamic或者Compressed，示例如下：

```
create table idx_length_test_02
(
  id int auto_increment primary key,
  name varchar(255)
)
ROW_FORMAT=DYNAMIC default charset utf8mb4;
```

> 说明 对已经创建的表，修改表的row_format格式命令如下：

```
alter table <表名> row_format=dynamic;
alter table <表名> row_format=compressed;
```

9.5. Cannot add foreign key constraint错误处理

错误信息

RDS for MySQL是支持外键约束的，但在创建外键约束时提示如下错误：

```
Cannot add foreign key constraint
```

错误原因

要关联的字段在要关联的表中不是主键。

解决方案

以tstudent表和tscore表为例说明如何解决此问题。

1. 先查看表结构，判断要关联的字段在要关联的表中是不是主键。

```
mysql>show create table tstudent;
+-----------------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Table           | Create Table                                                                                                                                                    |
+-----------------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------+
| tstudent        | CREATE TABLE `tstudent` (
  `sno` varchar(4) NOT NULL DEFAULT '',
  `sname` varchar(3) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+-----------------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------+
共返回 1 行记录,花费 267.17 ms.
mysql>show create table tscore;
+-----------------+---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Table           | Create Table                                                                                                                                                                                                                                            |
+-----------------+---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| tscore          | CREATE TABLE `tscore` (
  `sno` char(4) DEFAULT NULL,
  `cno` char(3) DEFAULT NULL,
  `score` int(10) DEFAULT '0'
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+-----------------+---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
共返回 1 行记录,花费 128.94 ms.
```

说明 可以看到表tstudent没有主键。

2. 为表tstudent添加主键，命令如下：

```
alter table tstudent add primary key(sno);
```

3. 再次创建外键约束即可成功。

```
mysql>alter table tscore add constraint fk_tscore_sno foreign key(sno) references tstudent(sno);
执行成功,花费 161.55 ms.
```

9.6. SELECT command denied to user 'username'@'ip' for table 'user'错误处理

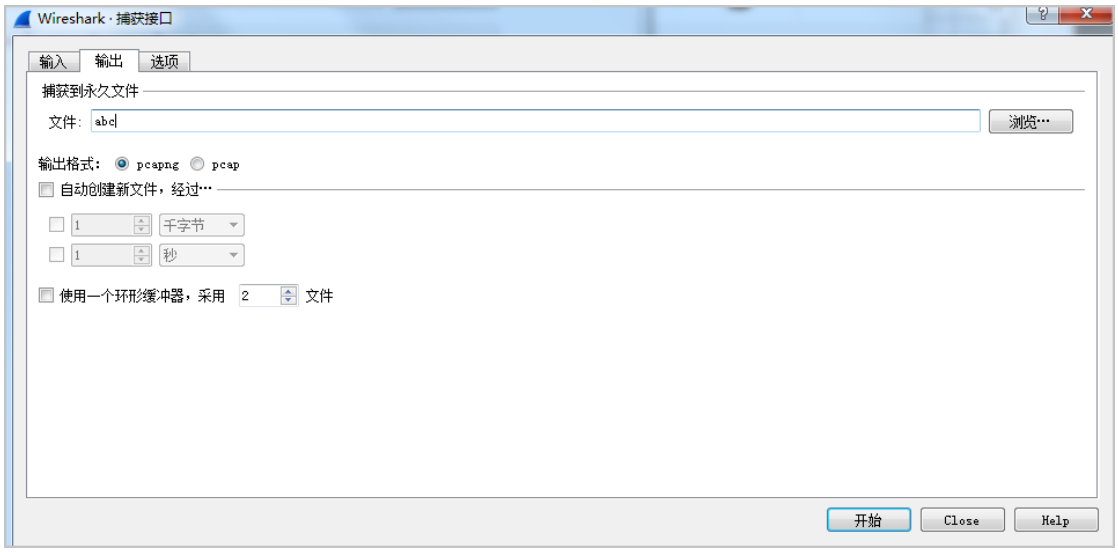
错误信息

使用RDS for MySQL，程序执行查询SQL时报错如下：

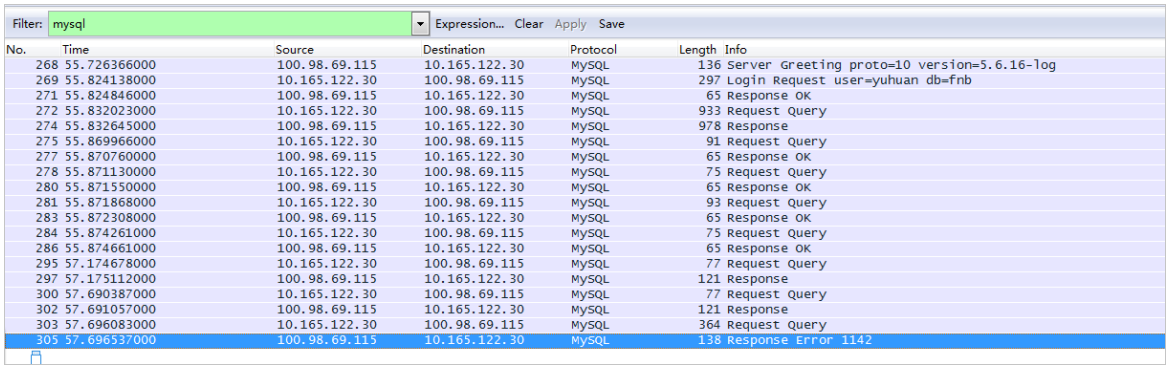
```
SELECT command denied to user 'username'@'ip' for table 'user'
```

排查步骤

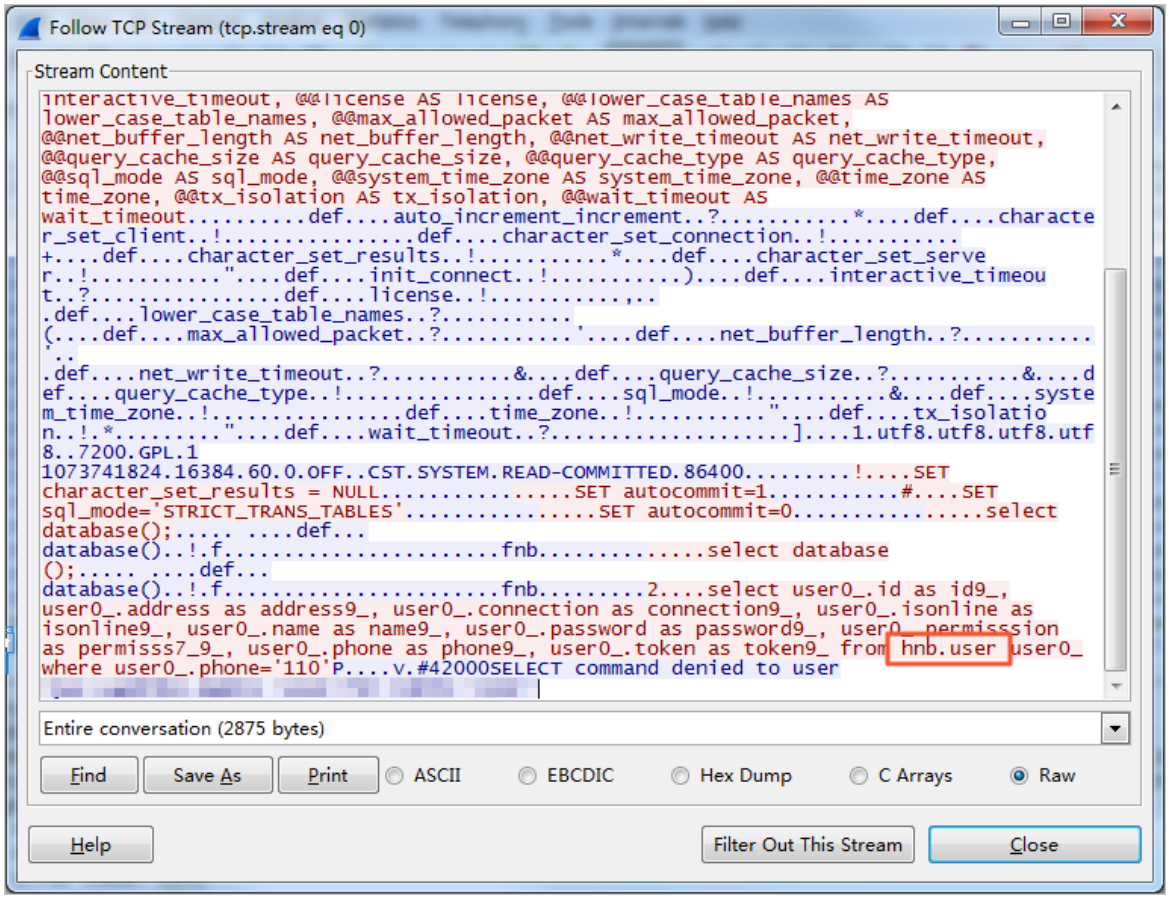
1. 测试RDS实例是否异常。可以使用mysql客户端连接到RDS，查询对应的表，如果可以正常查询，说明RDS没有问题。
2. 用Wireshark软件抓包本机发出的实际请求：
   i. 在Wireshark界面中，选择**捕获 > 选项**，在**输入**页签选择连接RDS的内网网卡，在**输出**页签填写输出的文件名，然后单击**开始**。



   ii. 复现问题，问题复现后，停止抓包。
3. 打开生成的抓包文件，在显示过滤器里输入mysql，过滤出mysql协议，找到报错的包。



4. 在报错的条目上单击右键，选择**追踪流 > TCP流**。



5. 检查发送的SQL是否正确。

> **说明** 上面的案例中，报错原因在于库名是fnb，而程序拼接出来的是hnb.user，数据库名拼接错误导致报错， 修正数据库名后问题解决。

9.7. the table '/home/mysql/xxxx/xxxx/#tab_name' is full错误处理

错误信息

在使用RDS for MySQL的过程中，有时会遇到如下的错误信息：

```
the table '/home/mysql/xxxx/xxxx/#tab_name' is full
```

示例

```
the table '/home/mysql/data3077/tmp/#sql_19472_5' is full
```

错误原因

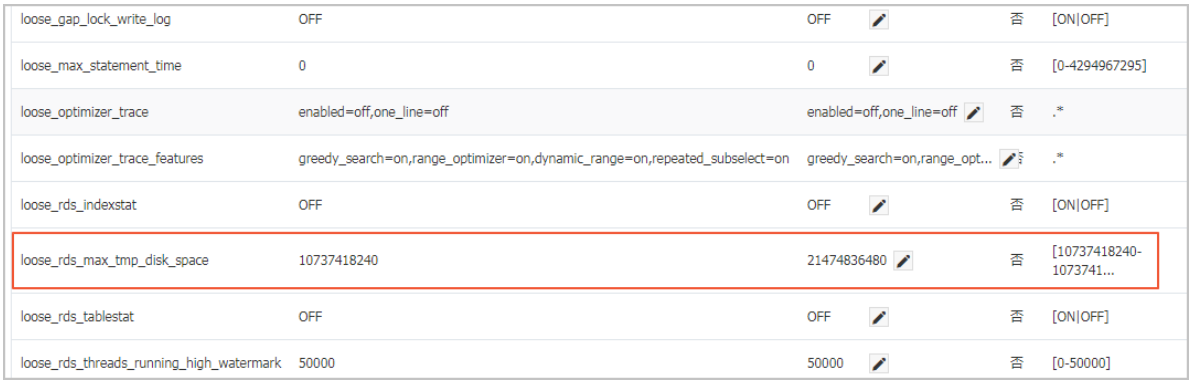
在SQL查询进行group by、order by、distinct、union、多表更新、group_concat、count(distinct)、子查询或表连接的情况下，MySQL有可能会使用内部临时表。MySQL首先在内存中创建Memory引擎临时表，当临时表的尺寸过大时，会自动转换为磁盘上的MyISAM引擎临时表（当查询涉及到Blob或Text类型字段，MySQL会直接使用磁盘临时表）。

这个错误信息，说明磁盘上的临时表#sql_19472_5的物理尺寸受到限制，已经无法再继续扩展了。导致这个错误信息的原因是查询语句使用的内部磁盘临时表（MyISAM 引擎表）总大小已经达到了实例参数loose_rds_max_tmp_disk_space指定的限制（默认10GB）。

解决方案

- 在控制台的**参数设置**中根据RDS实例当前空闲空间和应用空间使用情况，调高参数loose_rds_max_tmp_disk_space（仅MySQL 5.5/5.6支持该参数），建议设置为略小于当前空闲空间（保留一部分空间以便Binlog和数据文件使用），以避免磁盘临时表总占用空间过高，超过实例规格而导致实例锁定，影响业务。

  > 说明 参数loose_rds_max_tmp_disk_space单位是字节（Byte）。默认10737418240字节，即10 GB；上限107374182400字节，即100 GB。



loose_gap_lock_write_log	OFF	OFF	否	[ON|OFF]
loose_max_statement_time	0	0	否	[0-4294967295]
loose_optimizer_trace	enabled=off,one_line=off	enabled=off,one_line=off	否	.*
loose_optimizer_trace_features	greedy_search=on,range_optimizer=on,dynamic_range=on,repeated_subselect=on	greedy_search=on,range_opt...		.*
loose_rds_indexstat	OFF	OFF	否	[ON|OFF]
loose_rds_max_tmp_disk_space	10737418240	21474836480	否	[10737418240-1073741...
loose_rds_tablestat	OFF	OFF	否	[ON|OFF]
loose_rds_threads_running_high_watermark	50000	50000	否	[0-50000]

- 减少同时使用磁盘临时表的会话数量。因为参数loose_rds_max_tmp_disk_space指定的是磁盘临时表文件的总大小，因此减少并发使用磁盘临时表的会话数量可以避免超过该参数指定的限制。
- 在控制台的**参数设置**中调高参数tmp_table_size来提高内存临时表的上限。

  > 说明 tmp_table_size单位是字节（Byte），默认2097152字节，即2 MB；上限67108864字节，即64 MB。

thread_cache_size	100	100	否	[0-16384]
thread_stack	262144	262144	是	[131072-1073741824]
tls_version	TLSv1,TLSv1.1,TLSv1.2	TLSv1,TLSv1.1,TLSv1.2	是	[TLSv1,TLSv1.1,TLSv1...
tmp_table_size	2097152	2097152	否	[262144-67108864]
transaction_alloc_block_size	8192	8192	否	[1024-131072]

- 在查询中，尽量避免使用Blog和Text类型字段。
- 优化查询逻辑，避免过大的中间数据集操作。

如何判断查询是否使用内部临时表

使用explain查看执行计划，在Extra字段中有Using temporary表示会使用内部临时表。

示例SQL

```
select * from alarm group by created_on order by detail;
```

```
mysql.rds.aliyuncs.com) [ ]> explain select * from alarm group by created_on order by detail;
+----+-------------+-------+-------+----------------+----------------+---------+------+---------+----------------------------------+
| id | select_type | table | type  | possible_keys  | key            | key_len | ref  | rows    | Extra                            |
+----+-------------+-------+-------+----------------+----------------+---------+------+---------+----------------------------------+
|  1 | SIMPLE      | alarm | index | idx_alarm_cr_on | idx_alarm_cr_on | 4       | NULL | 9474312 | Using temporary; Using filesort |
+----+-------------+-------+-------+----------------+----------------+---------+------+---------+----------------------------------+
1 row in set (0.00 sec)
```

9.8. Out of resources when opening file './xxx.MYD' (Errcode: 24)错误处理

错误信息

在使用RDS for MySQL时遇到如下错误：

```
Out of resources when opening file './xxx.MYD' (Errcode: 24)
```

错误原因

出现这个错误是因为RDS中打开的文件数超过了open_files_limit限制。

解决方案

在RDS管理控制台的**参数设置**中修改参数**open_files_limit**，修改后需要重启实例才生效。

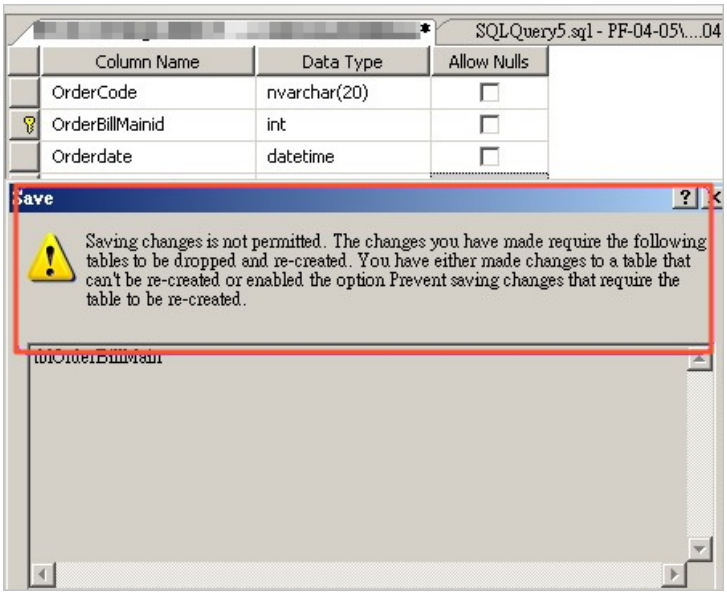
net_retry_count	10	10	否	[1-4294967295]	
net_write_timeout	60	60	否	[1-31536000]	
open_files_limit	65535	65535	是	[4000-65535]	
query_alloc_block_size	8192	8192	否	[1024-16384]	
query_cache_limit	1048576	1048576	否	[1-1048576]	
query_cache_size	3145728	3145728	否	[0-104857600]	

> **说明** 参数open_files_limit默认值是65535，在RDS中取值范围是4000~65535。

9.9. The Changes you have made require the following table错误处理

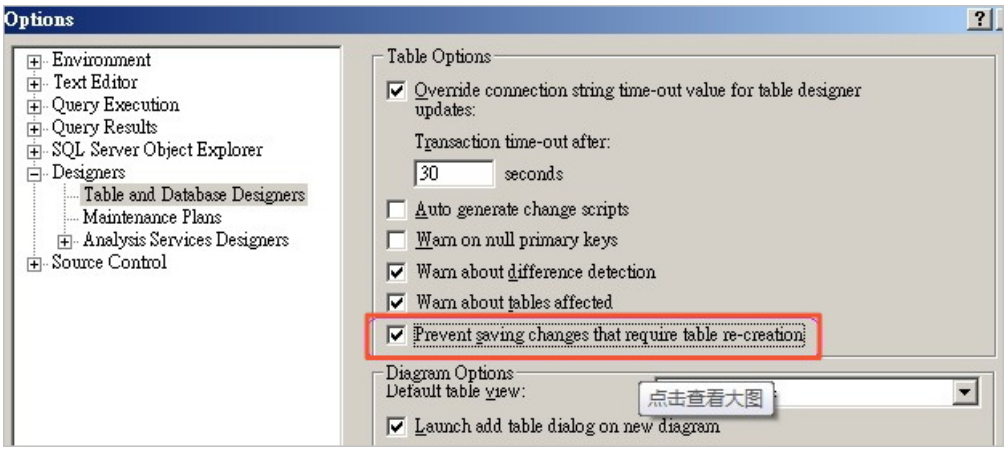
错误信息

RDS for SQL Server实例中修改表结构，保存时报错。



原因及解决方案

RDS for SQL Server开启了**Prevent saving changes that require the table re-creation**选项，在**Tools > Designers > Table and Database Designers > Table options**里关闭该选项即可。



9.10. There are 2 other sessions using the database错误处理

错误信息

RDS for PostgreSQL删除数据库时报错如下：

```
ERROR: database "mctest" is being accessed by other users  详细：There are 2 other sessions using the database.
```

原因

当前有其他连接在使用该库。

解决方案

断开mctest上所有的连接，命令如下：

```
select pg_terminate_backend(pid) from  (select pid from pg_stat_activity where datname = '<
数据库名>'  ) a;
```

然后再删除数据库即可。

9.11. relation "xxx" already exists错误处理

错误信息

RDS for PostgreSQL实例修改表名时报错。

```
alter table blacklist rename to BLACKLIST
ERROR: relation "blacklist" already exists
```

原因及解决方案

这是由于默认PostgreSQL大小写不敏感，需要对大写表名使用双引号，命令如下：

```
alter table blacklist rename to "BLACKLIST"
```

> **说明** 后续查询时也需要使用大写表名。

9.12. Caused By: ERROR: syntax error at end of input错误处理

错误信息

使用JDBC访问RDS for PPAS时出现如下报错：

```
CREATE OR REPLACE PACKAGE package_name
IS
PROCEDURE pro_name1 ();
PROCEDURE pro_name2 ();
PROCEDURE pro_name3 ();
END package_name;
[ERROR] Caused By: ERROR: syntax error at end of input
[ERROR] Position: xxx
```

解决方案

1. 确认以上语法通过命令行交互式客户端工具psql是否可以执行成功，如果执行不成功，请检查语法问题。
2. 如果不是语法问题，请确认当前正在使用的JDBC驱动，如果使用的是PostgreSQL的JDBC也会触发此问题，这是由于使用的是Oracle语法，应该使用edb-jdbc驱动。

下载PPAS专用的JDBC驱动请通过此链接：PPAS开发驱动程序。

> ⓘ **说明** jdk1.5及以下请用edb-jdbc14.jar，jdk1.6及以上请使用edb-jdbc16.jar。

9.13. Unknown MySQL server host错误处理

错误信息

ECS实例使用Linux系统连接RDS for MySQL实例时报错如下：

```
Unknown MySQL server host
```

错误原因

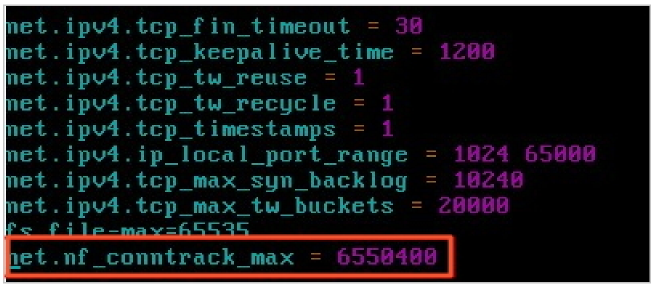
查看ECS系统日志，发现是由于iptables开启导致域名解析的数据包被丢弃。查看系统日志有如下报错。

```
[root@[redacted] ~]# tail -f /var/log/messages |grep conntrack
May 11 18:29:59 [redacted] kernel: nf_conntrack: table full, dropping packet.
May 11 18:29:59 [redacted] kernel: nf_conntrack: table full, dropping packet.
May 11 18:29:59 [redacted] kernel: nf_conntrack: table full, dropping packet.
May 11 18:29:59 [redacted] kernel: nf_conntrack: table full, dropping packet.
May 11 18:29:59 [redacted] kernel: nf_conntrack: table full, dropping packet.
May 11 18:29:59 [redacted] kernel: nf_conntrack: table full, dropping packet.
May 11 18:29:59 [redacted] kernel: nf_conntrack: table full, dropping packet.
May 11 18:29:59 [redacted] kernel: nf_conntrack: table full, dropping packet.
May 11 18:29:59 [redacted] kernel: nf_conntrack: table full, dropping packet.
May 11 18:29:59 [redacted] kernel: nf_conntrack: table full, dropping packet.
```

解决方案

根据内存实际情况来调整sysctl.conf文件内的参数net.nf_conntrack_max。示例如下：

```
vim /etc/sysctl.conf     --编辑文件
net.nf_conntrack_max = 6550400     --调整参数值
sysctl -p     --更新配置
```



```
net.ipv4.tcp_fin_timeout = 30
net.ipv4.tcp_keepalive_time = 1200
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_timestamps = 1
net.ipv4.ip_local_port_range = 1024 65000
net.ipv4.tcp_max_syn_backlog = 10240
net.ipv4.tcp_max_tw_buckets = 20000
fs.file-max=65535
net.nf_conntrack_max = 6550400
```

> ⓘ **说明** 更新内核参数如果报错 `error: "nf_conntrack_max" is an unknown key` ，需加载ip_conntrack模块，建议加入到/etc/rc.local启动项中。以上操作基于CentOS 6.5版本，其他低版本的参数是net.ipv4.ip_conntrack_max。

9.14. 临时实例报错The operation is not permitted due to no backup处理

错误信息

在控制台创建临时实例报错如下：

```
The operation is not permitted due to no backup
```

错误原因

实例进行过覆盖性恢复后，RDS API会禁止在覆盖性恢复的时间到上一个备份时间之间的时间段内创建临时实例，即用户的覆盖性恢复的时间点到上一个备份集之间是做了限制的，不能再进行还原（即不能再创建临时实例）。

详细说明

- **场景一**

  当前时间a，最近的一个备份集时间点b，b的上一个备份集时间点为c。

  用户在当前时间a，回滚数据到了时间点b；回滚后，在c到b时间点内是不能创建临时实例的；c时间点之前的创建是不受影响的。

- **场景二**

  当前时间a，最近的一个备份集时间点b，b的上一个备份集时间点为c。

  用户在b-a之间的某个时间点a1创建临时实例，并进行回滚到a1，b--a1之间是不能创建临时实例的，b时间点之前的创建是不受影响的。

- **场景三**

  当前时间a，最近的一个备份集时间点b，b的上一个备份集时间点为c。

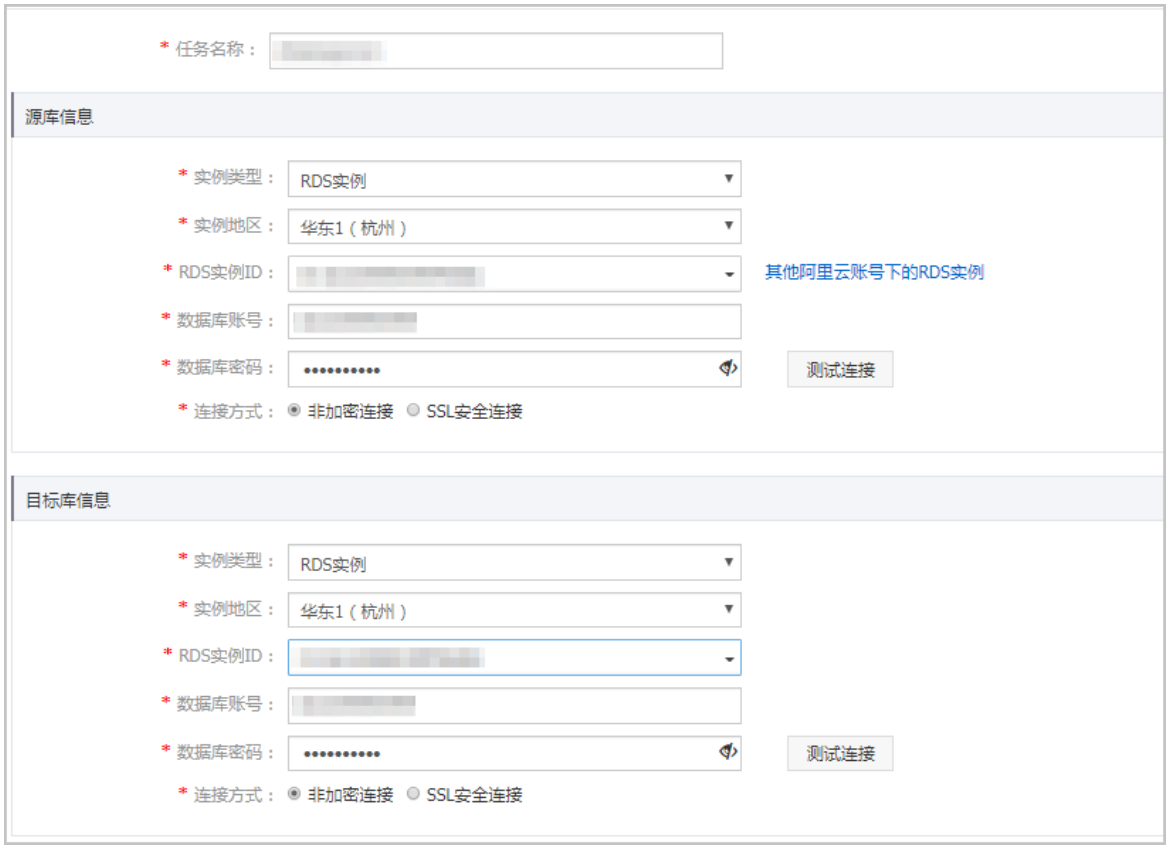
  用户在c--b之间的某个时间点b1创建临时实例，并进行回滚到b1，c--b1之间是不能创建临时实例的，c时间点之前的创建是不受影响的。

10.DTS相关问题

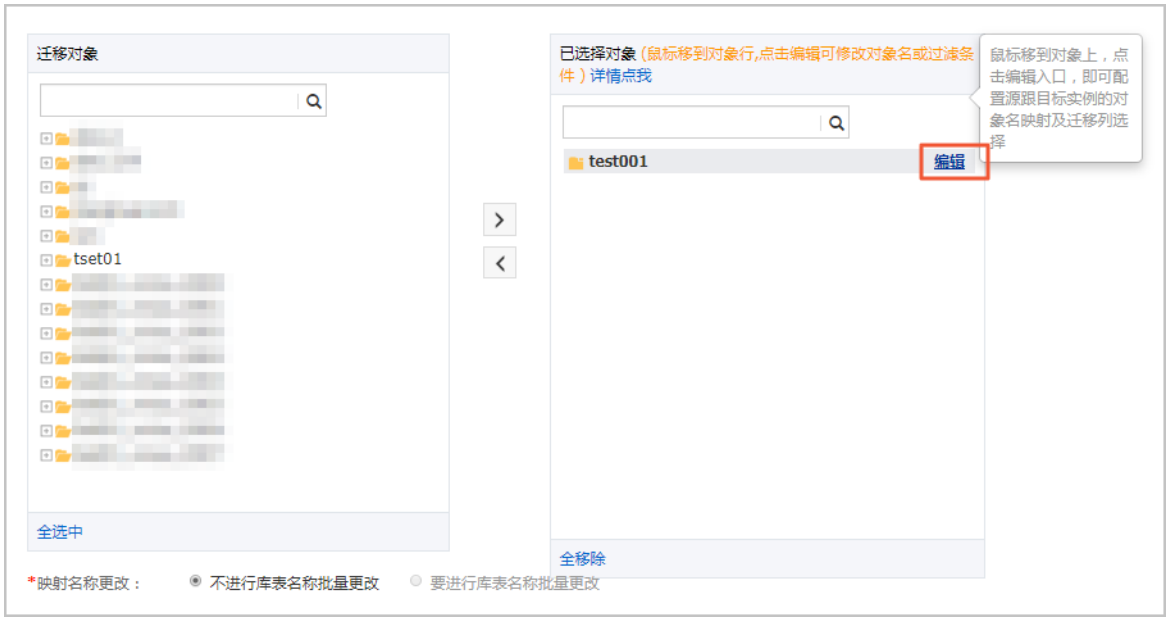
10.1. 通过DTS实现同实例下的数据库复制和改名

操作步骤

1. 进入DTS管理控制台。
2. 创建迁移任务，源库和目标库都选择同一个RDS实例，填写账号密码后单击**授权白名单并进入下一步**。



3. 选择要复制和改名的数据库，移动到**已选择对象**框后单击**编辑**进行改名。



4. 完成后续迁移步骤。

10.2. DTS订阅报错：Connection timed out

错误信息

DTS数据订阅出现超时错误如下：

```
2015-11-11 16:16:36,724 INFO [com.aliyun.drc.clusterclient.partition.PartitionPool] - clien
t partition is empty,wait partition balance
2015-11-11 16:16:46,748 INFO [com.aliyun.drc.clusterclient.partition.PartitionPool] - clien
t partition is empty,wait partition balance
2015-11-11 16:16:56,802 INFO [com.aliyun.drc.clusterclient.partition.PartitionPool] - start
new partition: {"tables":["xxxx.*"],"topic":"aliyun_bj_ecs_rdsxxxxxxxx-1-0","guid":"dts_rds
xxxxxxxx_nSj","partition":{"name":"0f59329ace868558b1xxxxxxxx","gmt":1447229810},"offset":"
::::1447229563:","group":"111111111111111"}
2015-11-11 16:08:34,933 ERROR [com.aliyun.drc.clusterclient.partition.PartitionPool] - keep
alive error
java.net.ConnectException: Connection timed out
```

解决方案

检查运行SDK的服务器是否设置了 `context.setUsePublicIp(false);` ，修改为 `context.setUsePublicIp(true);` 。

10.3. DTS订阅报错：java.io.IOException: Parse message attribute failed

错误信息

DTS订阅报错如下：

```
java.io.IOException: Parse message attribute failed because Store aliyun_hz_ecs_rdsxxxxxxxx
x-1-0.0000000003 find aliyun_hz_ecs_rdsxxxxxxxxx-1-0 without server id and 1444709748 faile
d error
at com.aliyun.drc.client.message.Builder.build(Builder.java:48)
at com.aliyun.drc.client.impl.HttpHandler.recvDRCPResponse(HttpHandler.java:245)
at com.aliyun.drc.client.impl.ServerProxy.getResponse(ServerProxy.java:138)
at com.aliyun.drc.client.impl.DRCClientImpl.run(DRCClientImpl.java:363)
at java.lang.Thread.run(Unknown Source)
```

错误原因

DTS订阅通道只能查询最近24小时之内的数据，时间点不在该范围内就会报这个错误。

解决方案

您可以修改DTS的消费位点，请参见修改消费位点。

10.4. DTS迁移过程中显示已完成的值超过总数的原因

本文介绍DTS迁移过程中显示已完成的值超过总数的原因。

问题现象

查看迁移详情发现已完成的行数超过总数。

对象名称	源库	目标库	总数	已完成	耗时(秒)	状态
[redacted]	[redacted]	[redacted]	159307	0	10分钟54秒	迁移中
[redacted]	[redacted]	[redacted]	159307	0	10分钟54秒	迁移中
[redacted]	[redacted]	[redacted]	159307	7940897	21小时43分钟6秒	迁移中

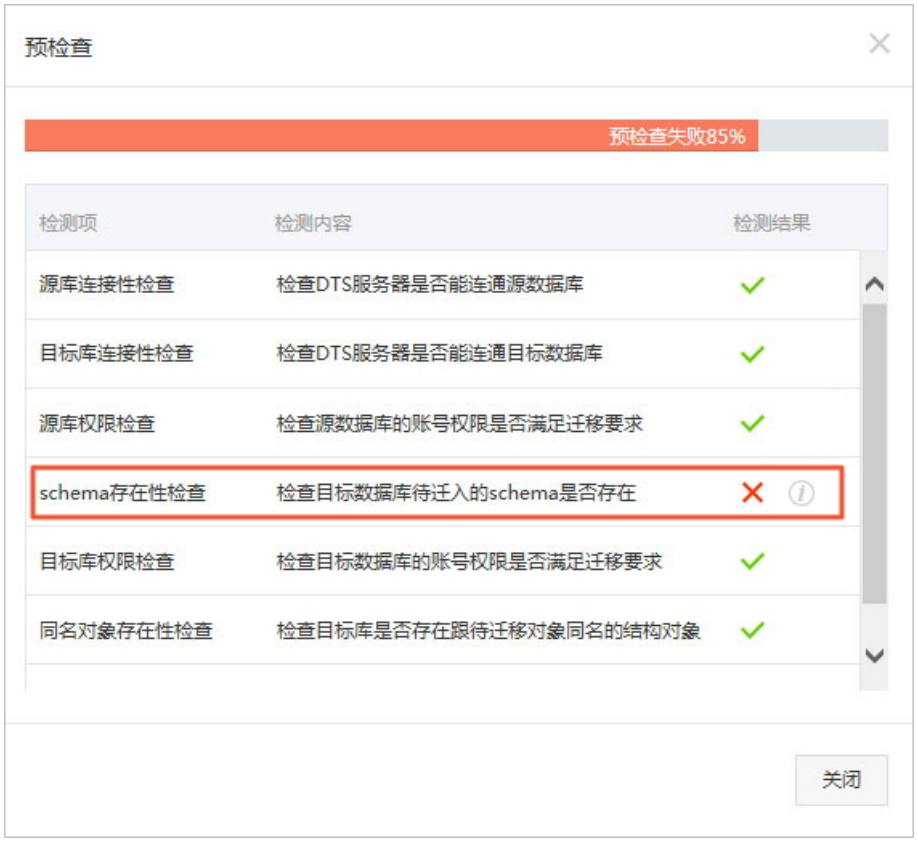
原因

这个总数是用MySQL预估的值，在迁移完成后会将总数调整为准确值。

10.5. DTS预检查时出现schema存在性检查错误处理

错误信息

使用DTS进行RDS的数据迁移时，在预检查时出现schema 存在性检查错误，如下图。



解决方案

如错误提示所述，该错误是由于目标库不存在或不匹配导致的。您需要在目标RDS实例下先创建与源库同名（注意大小写）的目标库。注意事项如下：

- 库名由小写字母、数字、下划线、中划线组成，字母开头，字母或者数字结尾。
- 如果源库与目标库名称不一致，在DTS迁移过程中可以修改迁移的库名。
- 由于目标库还未创建或创建异常，该错误通常还会伴随出现目标库权限检查的错误提示，目标库配置完成后，该错误也会解决。错误信息如下。

10.6. DTS订阅日志提示client partition is empty原因

日志内容

DTS订阅日志如下：

```
[ Thread-1:11039 ] - [ INFO ] client partition is empty,wait partition balance
```

原因

一个订阅ID只能用于一个Client端拉取数据，如果这个订阅ID启动了多个Client的话，只有一个可以拉取到数据，其他Client就会报这个日志。只有当拉取数据的Client挂了，另一个Client才可以拉取数据，这是DTS的容灾功能。

10.7. DTS订阅无法选择RDS只读实例

本文介绍DTS订阅时为什么无法选择RDS只读实例。

原因

DTS订阅需要读取实例数据库Binlog日志，目前只读实例禁止Binlog读取，所以DTS订阅无法选择RDS只读实例。

11.DMS相关问题

11.1. DMS中创建存储过程报错的处理

错误信息

DMS中使用SQL语句创建存储过程时报如下错误。

错误原因

dms中默认是以一个分号（;）作为一条语句结束的标志，但存储过程需要执行一段SQL，这些SQL是不可分割的。

解决方案

使用DELIMITER临时设置新的结束符。以双斜杠（//）为例，修改SQL代码如下：

```
DELIMITER //
CREATE  PROCEDURE  p_test()
BEGIN
  select CURRENT_DATE  as curDate ;
END//
DELIMITER ;
```

> **说明** `DELIMITER ;` 表示还原为以分号（;）作为结束标识符的默认设置。