RDS for MySQL

Troubleshooting

Issue	01
Date	2025-08-20

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei Cloud and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Contents

1 Backup and Restoration Issues

1.1 No SUPER Permissions When Restoring an RDS for MySQL Full Backup to a Local MySQL Database

Scenario

If you want to set up a local standby MySQL database for your RDS for MySQL instance, you can restore your instance data to the local database from full backups. When you run the **change master** command to establish a primary/standby relationship between the local database and your RDS for MySQL instance, the following error may occur:

Error 1227

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

Possible Causes

The **root** user of your RDS for MySQL instance does not have the SUPER permission.

Solution

Grant the SUPER permission to the **root** user. Do as follows:

1. To enable password-free authentication for the local MySQL database, add **skip-grant-tables=on** under **[mysqld]** in the **my.cnf** configuration file. Example:


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

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

2. Restart the mysqld process.

   systemctl restart mysqld

3. Log in to the local database as user **rdsAdmin** in password-free mode.

   mysql -urdsAdmin

4. Grant permissions to the **root** user.

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

   flush privileges;

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

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

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

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

5. To disable password-free authentication, delete **skip-grant-tables=on** under **[mysqld]** from the **my.cnf** configuration file.
6. Restart the mysqld process.
7. Log in to the local database as **root** and check the permissions.

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

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

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

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

Run the **change master** command as the **root** user again.

1.2 Backup Failures Due to DDL Operations

Scenario

Backups failed to be created for a DB instance two days in a row because there were DDL operations performed during the time window defined in the backup policy.

Possible Causes

RDS for MySQL uses XtraBackup to create full backups. To ensure data consistency, there are metadata lock conflicts between full backups and DDL operations. As a result, backups are blocked and fail due to timeouts.

Run the **show processlist** command to check whether any DDL operation was performed in the backup time window.

Solution

1. Stop the DDL operations and perform a manual backup.
2. Do not perform DDL operations in the backup time window.

1.3 Restoring an On-Premises or Huawei Cloud Backup File to an RDS DB Instance

To restore a Huawei Cloud RDS backup file to an RDS DB instance:

Restore the backup file to a DB instance by referring to **Restoring a DB Instance from Backups**.

To restore an on-premises MySQL backup file to an RDS DB instance:

Use the DRS real-time migration to migrate on-premises MySQL databases to RDS by referring to **To the Cloud**.

1.4 RDS for MySQL Backup Job Failure

Scenario

When a user runs mysqldump to back up RDS for MySQL data to an ECS in a different subnet from RDS, the backup job runs for 300 seconds and then fails.

Troubleshooting

Replace the ECS where the backup job is executed with an ECS that is in the same subnet as RDS. The backup job is successfully executed.

- Network: There are no differences on latency and bandwidth between the two ECSs.
- Database: The **net_write_timeout** parameter is set to **300** on the RDS for MySQL database. The connection between the ECS and RDS for MySQL is interrupted after 300s regardless of whether data writes have been completed.

```
| net_write_timeout             | 300      |
```

Procedure

Step 1 Identify the backup data flow, protocol, and port.

mysqldump uses TCP to connect to port 8635 of RDS. After the connection is established, the backup job starts.

Step 2 Compare the hardware configuration and OS version of the two ECSs.

1. Both of them use the same hardware configuration: two cores and 6 GB of memory.
2. Both of them use the same OS version: CentOS 7.4.

Step 3 Check whether the NIC rates are the same.

Step 4 Check whether the kernel parameter settings are the same. The result shows that the network parameters on the ECS where the backup job failed are not optimized.


```
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
vm.swappiness = 0
net.ipv4.neigh.default.gc_stale_time=120
net.ipv4.conf.all.rp_filter=0
net.ipv4.conf.default.rp_filter=0
net.ipv4.conf.default.arp_announce = 2
net.ipv4.conf.lo.arp_announce=2
net.ipv4.conf.all.arp_announce=2
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_synack_retries = 2
vm.swappiness=5
net.ipv4.tcp_fack=1
net.ipv4.tcp_tw_reuse=1
net.ipv4.tcp_tw_recycle=1
net.ipv4.tcp_fin_timeout=30
net.ipv4.tcp_keepalive_time=600
net.ipv4.tcp_keepalive_probes=5
net.ipv4.tcp_keepalive_intvl=15
net.ipv4.tcp_max_syn_backlog=4096
                                = 5000
net.core.rmem_default=434176
net.core.wmem_default=434176
net.core.rmem_max=1048576
net.core.wmem_max=1048576
kernel.shmmax=5153960755
kernel.sem= 512 524288 32 1024
kernel.msgmni=128
net.core.somaxconn=20000
fs.file-max=800000
                = 524288 786432 1048576
                = 4096 16384 65536
                = 4096 87380 65536
```

Step 5 Set the kernel parameters of the ECS where the backup job failed to the same as those of the ECS where the backup job succeeded. Start a backup job again. The backup job is successful.

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

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

----End

Solution

There is a large volume of data writes during the backup process across networks. The data write capability and TCP buffer on the backup end do not match the sending capability of the RDS. When the timeout period reaches the preset threshold (300s), the backup job failed. You can increase the TCP buffer by modifying the ECS kernel parameters to resolve this issue.

1.5 Manual Backups Taking More Time Than Automated Full Backups

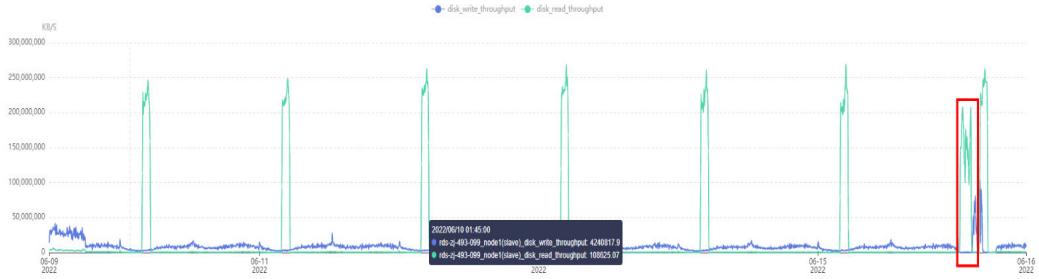
Scenario

When 20 GB of data needs to be backed up, a full backup automatically performed by RDS takes less time than a full backup performed manually.

Possible Causes

The disk throughput during the manual backup was lower than that during the automated backup.

Check the disk reads and writes on the Cloud Eye console. The following figure is for reference only.



Backups are stored in OBS buckets. The automated backup was performed during off-peak hours of OBS, and therefore took a short period of time. The manual backup was performed during peak hours of OBS, and therefore took much more time.

Solution

Perform full backups during OBS off-peak hours.

1.6 Incorrect Login Password During Data Restoration from a Local Backup File

Scenario

After data is restored from a local backup, a message is displayed when you log in to the local database, indicating that the login credentials were incorrect.

Possible Causes

When data is restored from a local backup, the password of the local database may have been overwritten by the password of the cloud database. As a result, you cannot log in to the database using the password of the local database.

Solution

Use the root password of the cloud database for login or reset the password of the local database.

1.7 Automated Incremental Backup Failed Due to Full Storage

Scenario

Automated incremental backup failed.

Possible Causes

As the amount of service data grows over time, it can eventually use up all the space available for your DB instance. When the DB instance status is **Storage full**, data cannot be written to databases, and the instance is no longer possible to perform an incremental backup.

Solution

Scale up storage space for your RDS instance. On the **Instance Management** page, locate the target DB instance and choose **More** > **Scale Storage Space** in the **Operation** column. Perform a full backup after the storage space is successfully scaled up and the incremental backup will be successful.

1.8 RDS Point-in-Time Restoration Task Failure

Scenario

A point-in-time restoration of an RDS DB instance fails.

Possible Causes

The backup timestamp is incorrect. As a result, an error is reported when the DB instance is restored to a specific point in time.

Solution

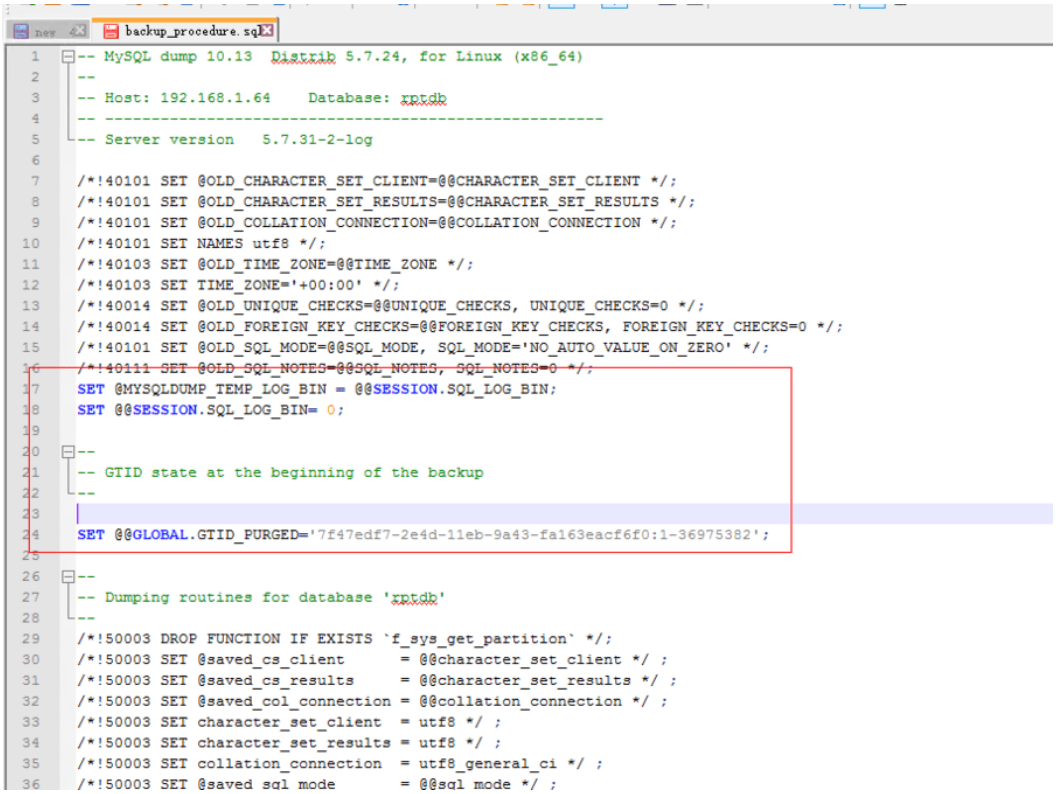
Restore your instance data from a full backup file. For details, see **Restoration Solutions**.

1.9 SQL Statements Such as SET @@SESSION.SQL_LOG_BIN Displayed After You Run mysqldump

Scenario

When you run mysqldump on a database, the following code is displayed:

Figure 1-1 Code



Fault Analysis

The parameter **gtid-mode** is set to **ON**.

If GTID is enabled for a database, you can use mysqldump to back up or dump all Global Transaction Identifiers (GTIDs) in the database or even to back up the whole RDS for MySQL database.

Solution

When the primary and standby RDS for MySQL databases are exported for backup and restoration, check whether GTID is enabled.

If GTID is enabled, add **–set-gtid-purged=OFF** to the **mysqldump** command during data dump.

1.10 Insufficient Permissions During Data Export Using mysqldump

Case 1

When you export database data with mysqldump using a specified user account, the error message "Access denied; you need (at least one of) the PROCESS privilege(s)" is displayed.

- Cause analysis: The user used to export data does not have the PROCESS privilege.
- Solution: Use the administrator account to grant the PROCESS privilege to the user.

  GRANT SELECT, PROCESS ON *.* TO '*<username>***'@'%';**

  FLUSH PRIVILEGES;

Case 2

The error message "Access denied; you need (at least one of) the RELOAD privilege(s) for this operation (1227)" is displayed when mysqldump is used to export data.

- Cause analysis: The user used to export data does not have the RELOAD privilege.
- Solution: Use Data Admin Service (DAS) to grant the RELOAD privilege to the user.

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

  FLUSH PRIVILEGES;

Case 3

The error message "Access denied; you need (at least one of) the LOCK TABLES privilege(s) for this operation (1227)mysqldump" is displayed when mysqldump is used to export data.

- Cause analysis: The user used to export data does not have the LOCK TABLES privilege.
- Solution: Grant the LOCK TABLES privilege to the user.

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

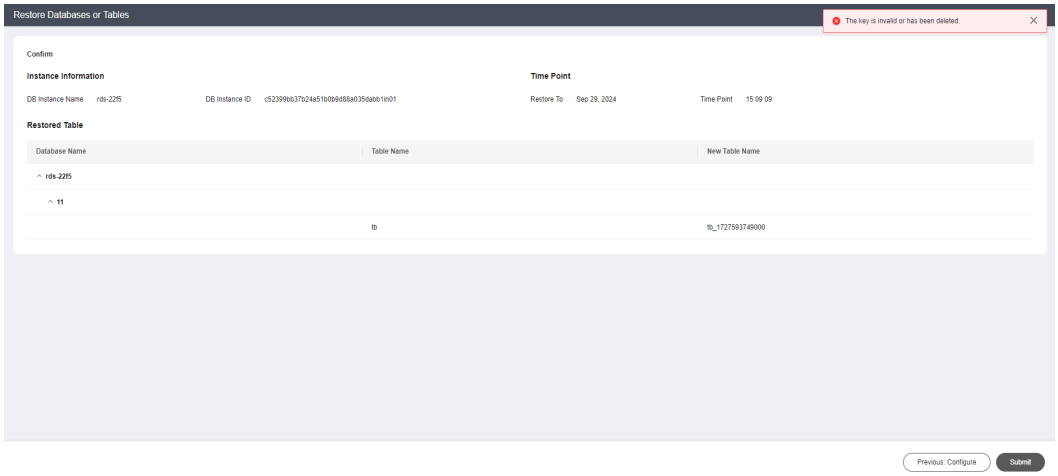
  FLUSH PRIVILEGES;

1.11 Key Considered Invalid or Deleted During Table-Level PITR

Scenario

During table-level point-in-time recovery (PITR) of an RDS for MySQL instance, a message is displayed indicating that the key is invalid or has been deleted.

Figure 1-2 Restoring tables to a specified point in time

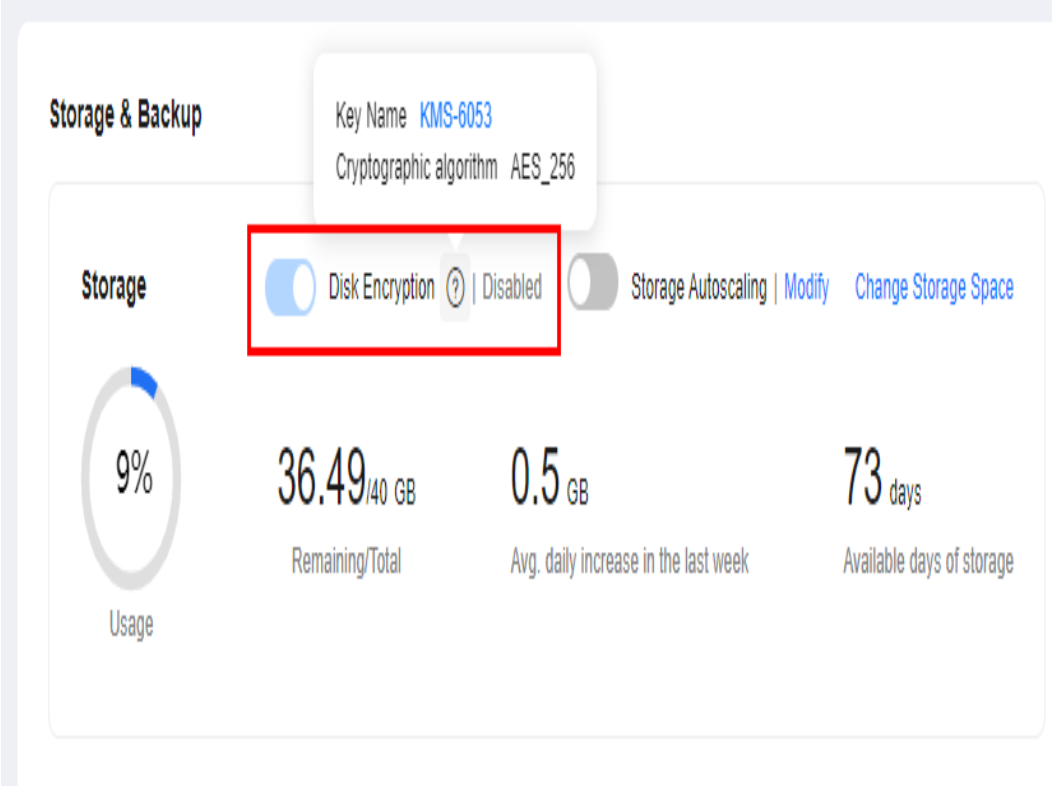


Troubleshooting

On the **Overview** page of the DB instance, check whether the key associated with the instance is disabled.

If the key is disabled or deleted, you cannot scale up the storage space, change specifications, or restore databases or tables for the instance.

Figure 1-3 Checking a key



Solution

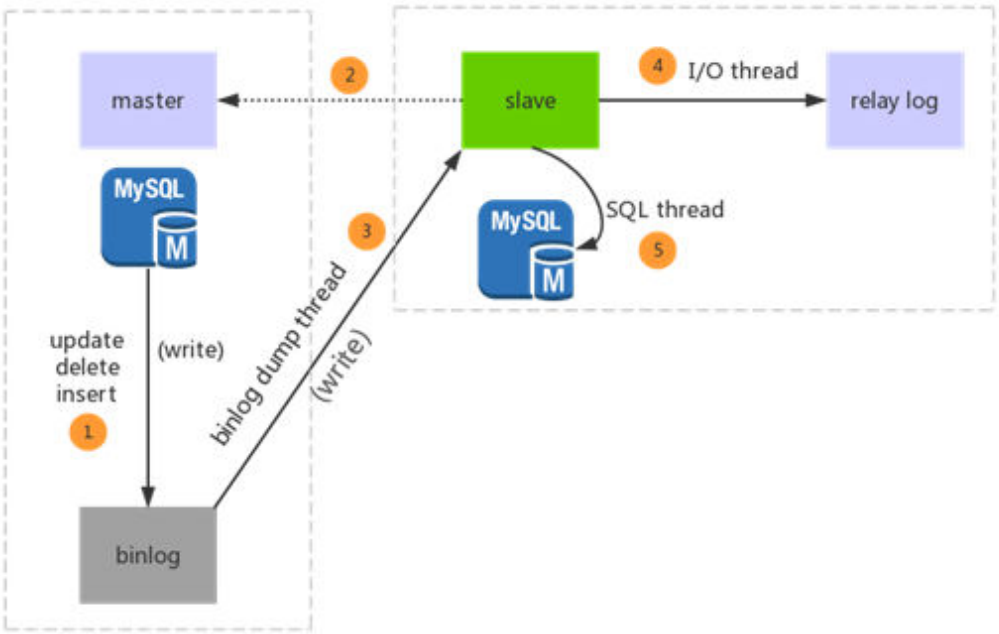
Enable the key.

2 Primary/Standby Replication Issues

2.1 How Primary/Standby Replication Works

RDS for MySQL standby instances, read replicas, self-built standby databases, and DR instances established through DRS all use MySQL primary/standby (also called "master/slave") replication. This section describes how MySQL Primary/Standby Replication works.

Primary/Standby Replication Process



- When data is updated on the master node, the update is written to the binlog of the master node as an event. The event types include INSERT, DELETE, UPDATE, and QUERY.
- When slave nodes are connected to the master node, a binlog dump thread is created for each slave node.

- When the binlog of the master node changes, the binlog dump threads notify all the slave nodes of the change and push the binlog updates to the slave nodes.
- After receiving the binlog updates, the I/O thread of each slave node writes the updates to the local relay log.
- The SQL thread of each slave node reads the relay log and replays the operation (such as DML and DDL) based on the event in the relay log.

Seconds_Behind_Master Calculation Method

Seconds_Behind_Master indicates the primary/standby replication delay, which can be obtained by running **show slave status**. The following pseudocode shows how the Seconds_Behind_Master value is calculated.

```
if (SQL thread is running)
//If the SQL thread is started
{
    if (SQL thread processed all the available relay log)
    //If the binlog pulled by the I/O thread from the master node is the same as that read by the SQL
thread from the relay log
    {
        if (IO thread is running)
            //If the I/O thread is started, Seconds_Behind_Master is set to 0.
            print 0;
        else
            //If the I/O thread is not started, Seconds_Behind_Master is set to NULL.
            print NULL;
    }
    else
        //If the SQL thread did not process all events written by the I/O thread, the value of
Seconds_Behind_Master needs to be calculated.
        Calculate the value of Seconds_Behind_Master using the formula;
}
else
    //If the SQL thread is not started, Seconds_Behind_Master is set to NULL.
    print NULL;
```

In the pseudocode, the following formula is used to calculate the value of **Seconds_Behind_Master**:

Seconds_Behind_Master = time(0) - last_master_timestamp - clock_diff_with_master

Explanations of the variables:

1. **time(0)**: The system time of the current slave server.
2. **clock_diff_with_master**: The difference between the system time of the slave server and that of the master server. Generally, the value is 0. If the system time of the slave server is different from that of the master server, the calculated value of **Seconds_Behind_Master** is inaccurate.
3. **last_master_timestamp**: The execution time of an event on the master node that is updated by the SQL thread. This variable is calculated and updated when the slave node replays the event in the relay log. The update time in the multi-threaded slave (MTS) replication is different from that in single-threaded replication. MTS is enabled by default.
   - MTS: The SQL thread of the slave node updates the value of **last_master_timestamp** after each transaction is executed. The update is performed by transaction. Therefore, large transactions and DDL operations may cause a long primary/standby replication delay. For

details, see **Automatic Recovery of Extended Primary/Standby Replication Delay**.

- Single-threaded replication: After the SQL thread of the slave node reads a transaction from the relay log, **last_master_timestamp** is updated before the transaction is executed. The update is performed by transaction.

The formula for calculating **Seconds_Behind_Master** can be understood as follows:

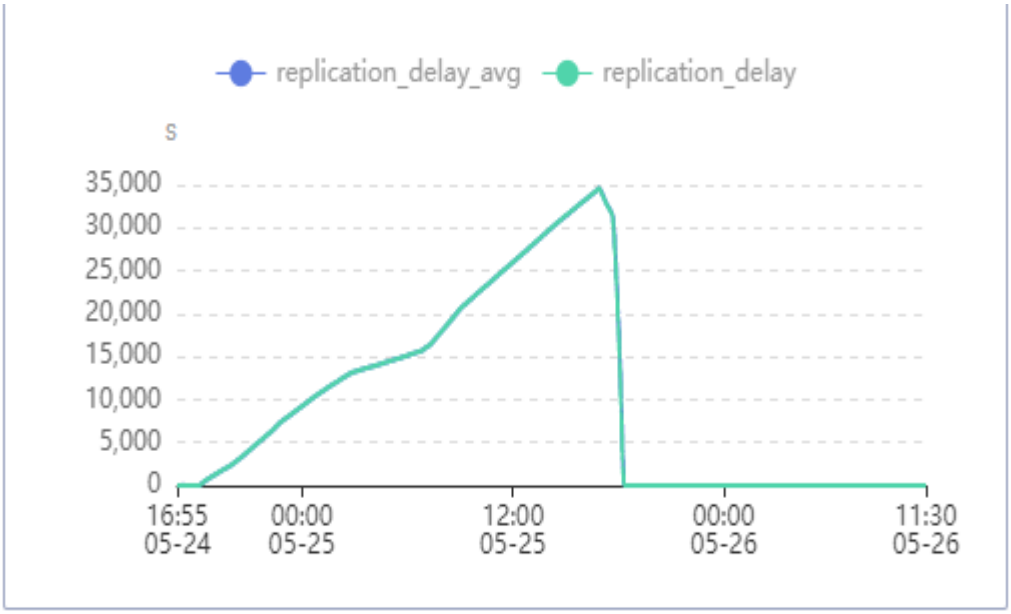
Seconds_Behind_Master = System time of the current slave server – Execution time of the transaction on the master node that is updated by the SQL thread – Difference between the system time of the slave server and that of the master server

2.2 Automatic Recovery of Extended Primary/Standby Replication Delay

Scenario

The primary/standby replication delay of a DB instance was long, kept increasing for a period of time, and then automatically recovered.

The following figure is an example showing how the real-time replication delay metric changes on the Cloud Eye console.

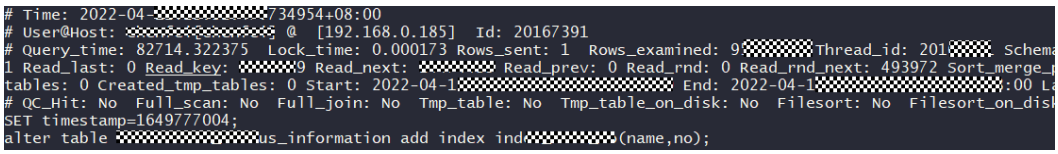


Possible Causes

According to **Primary/Standby Replication Delay Scenarios and Solutions** and **How Primary/Standby Replication Works**, this problem is caused by large transactions or DDL operations.

You can analyze full logs or slow query logs to check whether there are large transactions or DDL operations.

As shown in the following figure, if a DDL operation for adding an index was recorded in the slow query logs, the table contained hundreds of millions of data records, and the execution took about one day, the replication delay kept increasing when the DDL operation was replayed on the read replica or standby node. After the DDL operation was replayed, the replication delay dropped back to the normal range.



Solution

- Wait until the DDL operation is complete.
- Add indexes during off-peak hours.

2.3 Primary/Standby Replication Delay Scenarios and Solutions

RDS for MySQL standby instances, read replicas, self-built standby databases, and DR instances established through DRS all use MySQL Primary-Secondary Replication. Primary/standby replication can be implemented in asynchronous or semi-synchronous mode. In either mode, primary/standby replication delay is inevitable for some statements.

Symptom: The replication delay of the standby instance or read replica is too long, or even an alarm is generated for too long replication delay.

Scenario 1: Large Transaction Executed on the Primary Instance

A large transaction is a transaction containing a large quantity of data update operations, for example, a transaction including tens of thousands of DML (INSERT, UPDATE, and DELETE) operations, or an SQL statement that updates tens of thousands of rows of data in batches. The execution of a large transaction usually takes a long time (minutes). After a large transaction is executed on the primary instance, a large number of binlogs are generated. It takes much more time for the standby instance or read replica to obtain these binlogs than for a common transaction. In addition, replaying the transaction also takes much time, at least the same as the time for the transaction being executed on the primary instance. As a result, replication delay occurs on the standby node or read replica.

Troubleshooting:

- For a large transaction that contains a large number of DML statements, run the following command to find the statement that is executed for a long time:

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

- For a large transaction where an SQL statement executes a large amount of data updates, run **show full processlist** to check whether any DELETE or UPDATE statements are taking a long time to execute.

- Analyze full logs or slow query logs to check whether there are any large transactions.

Solution:

- To ensure data consistency between the primary and standby instances, wait until the large transaction is complete.
- Split a large transaction into small transactions and execute them in batches. For example, you can use the WHERE condition or LIMIT statement to limit the amount of data to be updated each time.

Scenario 2: Table Without a Primary Key Updated

RDS for MySQL binlogs use row-based logging. When data in a row is updated, an event record in row format is generated in the binlog. For example, if an UPDATE statement updates 100 rows of data in a table, 100 rows of update records will be generated in the binlog. During playback on the standby instance or read replica, 100 single-row updates are executed.

When replaying binlog events of the primary instance, the standby instance or read replica searches for rows to be modified based on the primary key or unique secondary index of the table. If no primary key is created for the table, a large number of full table scans are generated. As a result, the application speed of the binlog is reduced and replication delay occurs.

Troubleshooting:

Use **show create table** *xxx* to find out which table has the problematic UPDATE or DELETE statement and check whether the table has a primary key.

Solution:

Add a primary key to the table without a primary key or add a unique secondary index as required.

Scenario 3: DDL Operation Performed

A DDL operation typically takes a long time to complete, especially if the table contains a large amount of data. Generally, the time for replaying a DDL operation on the standby instance or read replica is almost the same as that for executing the DDL operation on the primary instance. Therefore, after a DDL operation is performed on a large table in the primary instance, the replication time will increase when the standby instance or read replica replays the operation.

Solution:

Wait for the DDL execution to complete, or perform DDL operations during off-peak hours.

Scenario 4: Read Replica Waiting for a Metadata Lock

When a long transaction is being executed on a table in a read replica, the DDL operation of the table synchronized from the primary instance will be blocked due to a metadata lock, and subsequent binlog playback of the table is also blocked. As a result, the replication delay increases.

Troubleshooting:

1. Log in to the read replica and run the following command to check whether there are any transactions that are taking a long time to complete:

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

2. View the metadata lock view of the read replica to check whether a metadata lock conflict occurs.

   select * from information_schema.metadata_lock_info;

   Find the blocked session based on the thread ID in the metadata lock view. For more information, see **MDL Views**.

Solution:

Kill the long transaction that blocks DDL operations on the read replica, or commit the long transaction on the application.

Scenario 5: Read Replica Specifications Smaller Than the Primary Instance

If the specifications of the read replica or DR instance established through DRS are smaller than those of the primary instance and the write load of the primary instance increases somewhat, the read replica or DR instance cannot replay binlogs in a timely manner due to insufficient resources. As a result, the replication delay increases.

Solution:

Scale up the specifications of the read replica or DR instance to match the specifications of the primary instance.

Scenario 6: Sudden Increase in Read Requests

In addition to synchronizing data from the primary instance, the read replica also processes read requests. If read requests soar up, the replay threads of the read replica may be affected, leading to an increase in the replication delay.

2.4 Abnormal Replication Between Primary and Standby RDS DB Instances

Scenario

The replication relationship between primary and standby RDS DB instances is abnormal. The default security group rule may have been deleted.

Solution

Step 1 **Log in to the management console**.

Step 2 Click  in the upper left corner and select a region.

Step 3 Click  in the upper left corner of the page and choose **Databases** > **Relational Database Service**.

Step 4 On the **Instances** page, click the target instance name.

Step 5 On the **Overview** page, click the security group name.

Step 6 On the **Inbound Rules** tab, click **Add Rule**. In the displayed dialog box, select **All** for **Protocol & Port** and **Security group** for **Source**. In the drop-down list, select the same security group as the displayed one on the **Basic Information** page.

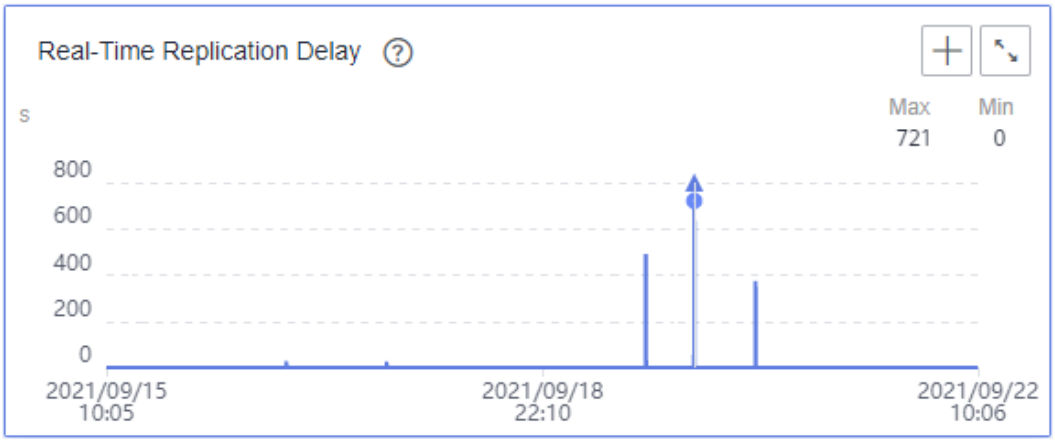
Step 7 Wait until the rule is added. Check that the replication relationship between primary and standby DB instances is restored.

----End

2.5 Primary/Standby Replication Delay Increases Sharply and Then Decreases

Scenario

The following figure shows a Cloud Eye console display indicating that the **Real-Time Replication Delay** metric of a read replica increased sharply and then fell back down again.

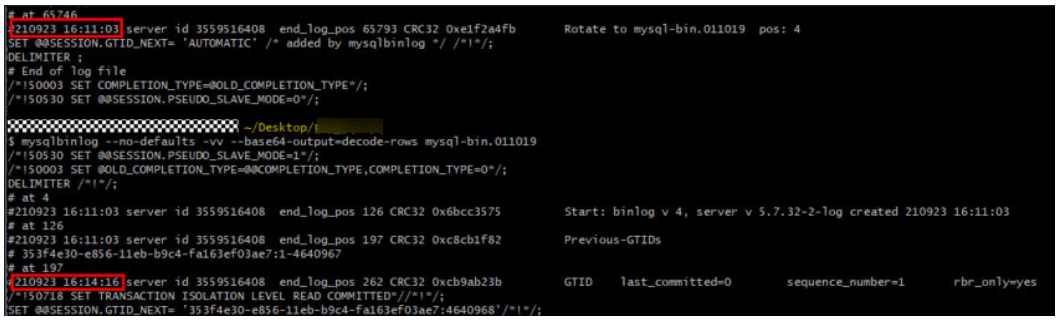


Possible Causes

- This problem is related to how the value of **Seconds_Behind_Master** is calculated. For details about how to calculate this value, see **How Primary/Standby Replication Works**.
- The replication delay peak occurred because the I/O thread of the read replica received a new binlog file, but the SQL thread did not start to replay the new binlog file. As a result, the value of **last_master_timestamp** for calculating **Seconds_Behind_Master** was the time when the previous binlog transaction was executed on the primary node, which was different from the system time (**time (0)**) of the read replica. When the SQL thread started to parse the new binlog file, the replication delay immediately decreased.
- This problem occurs occasionally and does not affect your workloads.

The following symptom was found in the binlog files generated during the period when the replication delay increased sharply and then fell back:

The difference between the start time of the first transaction in the new binlog file and the end time of the last transaction in the previous binlog file was the same as the difference between the sudden increase and fallback times.



Solution

No action is required. This happens sometimes but is completely normal.

2.6 Insufficient Permissions Reported for Canal

Scenario

When you start Canal while obtaining binlogs from RDS for MySQL using a specified user account, the following error message is often displayed: 'show master status' has an error! Access denied: you need (at least one of) the SUPER, REPLICATION CLIENT privilege(s) for this operation.

The complete message will look something like this:

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

Possible Causes

The user account does not have the REPLICATION SLAVE or REPLICATION CLIENT permissions.

Solution

Grant the REPLICATION SLAVE and REPLICATION CLIENT permissions to the user account as the administrator.

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

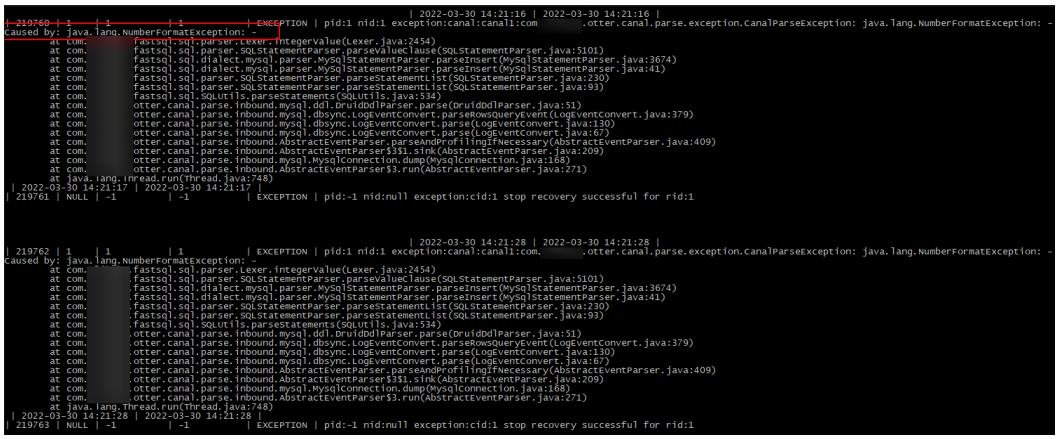
FLUSH PRIVILEGES;

2.7 Canal Fails to Parse Binlogs

Scenario

An error occurred when Canal parsed binlogs, interrupting binlog collection. The error message is as follows:

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



Troubleshooting

Check whether the value of **binlog_rows_query_log_events** of your RDS for MySQL instance is set to **1** or **ON**.

- Canal supports only subscriptions to binlogs in row format.
- When the value of **binlog_rows_query_log_events** is set to **1** or **ON**, Rows_query events are generated in binlogs. These events are not in row format. In certain scenarios, blank topics may occur in Canal, resulting in a binlog parsing failure.

Solution

Change the value of **binlog_rows_query_log_events** to **OFF** and restart the interrupted Canal task.

2.8 RDS for MySQL Binlog Issues

Scenario 1

The binlog retention period was set to 7 days for an RDS for MySQL instance. A binlog was generated every 5 minutes. The actual data volume was not the same as expected.

Cause analysis:

After a DB instance is created, a full backup is generated automatically and a binlog is generated every 5 minutes.

If there are no data changes, no binlog is generated.

Scenario 2

The service volume remained almost unchanged, but the storage usage of generated binlogs increased greatly.

Cause analysis:

RDS for MySQL binlogs are row-based, so the entire rows of data before and after the modification are recorded.

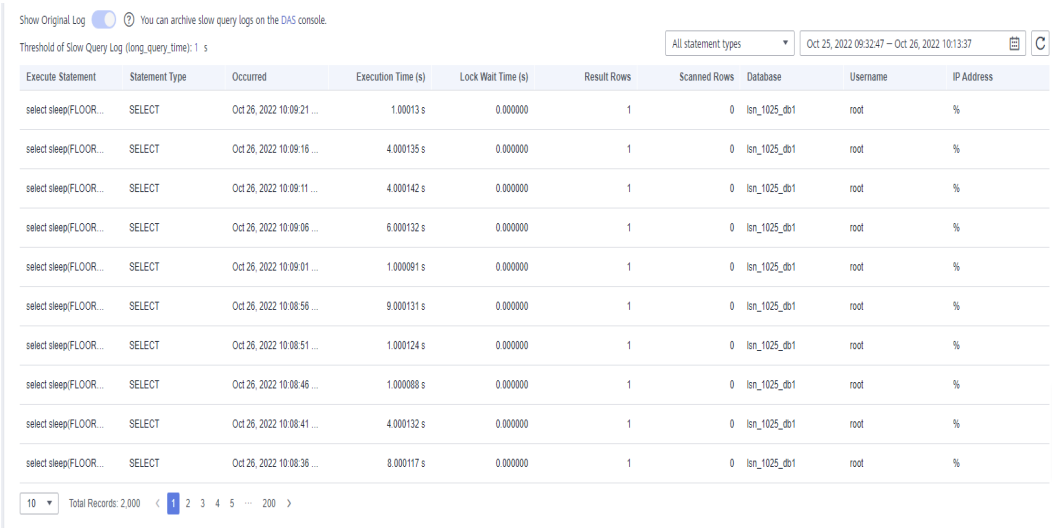
Suppose that there is a table containing a column with much data and you perform an **update** operation on certain data except those in that column. But binlogs completely record the data of the whole rows before and after the **update** operation. As a result, the storage usage of binlogs increases.

3 Parameter-related Issues

3.1 long_query_time Changes Fail to Be Applied

Scenario

After the value of **long_query_time** was changed from 0.1s to 0.2s on the console, slow query logs still included slow SQL statements whose execution duration was less than 0.2s.


Show Original Log You can archive slow query logs on the DAS console.

Threshold of Slow Query Log (long_query_time): 1 s

All statement types | Oct 25, 2022 09:32:47 – Oct 26, 2022 10:13:37

Execute Statement	Statement Type	Occurred	Execution Time (s)	Lock Wait Time (s)	Result Rows	Scanned Rows	Database	Username	IP Address
select sleep(FLOOR...	SELECT	Oct 26, 2022 10:09:21 ...	1.00013 s	0.000000	1	0	lsn_1025_db1	root	%
select sleep(FLOOR...	SELECT	Oct 26, 2022 10:09:16 ...	4.000135 s	0.000000	1	0	lsn_1025_db1	root	%
select sleep(FLOOR...	SELECT	Oct 26, 2022 10:09:11 ...	4.000142 s	0.000000	1	0	lsn_1025_db1	root	%
select sleep(FLOOR...	SELECT	Oct 26, 2022 10:09:06 ...	6.000132 s	0.000000	1	0	lsn_1025_db1	root	%
select sleep(FLOOR...	SELECT	Oct 26, 2022 10:09:01 ...	1.000091 s	0.000000	1	0	lsn_1025_db1	root	%
select sleep(FLOOR...	SELECT	Oct 26, 2022 10:08:56 ...	9.000131 s	0.000000	1	0	lsn_1025_db1	root	%
select sleep(FLOOR...	SELECT	Oct 26, 2022 10:08:51 ...	1.000124 s	0.000000	1	0	lsn_1025_db1	root	%
select sleep(FLOOR...	SELECT	Oct 26, 2022 10:08:46 ...	1.000088 s	0.000000	1	0	lsn_1025_db1	root	%
select sleep(FLOOR...	SELECT	Oct 26, 2022 10:08:41 ...	4.000132 s	0.000000	1	0	lsn_1025_db1	root	%
select sleep(FLOOR...	SELECT	Oct 26, 2022 10:08:36 ...	8.000117 s	0.000000	1	0	lsn_1025_db1	root	%

10 | Total Records: 2,000 | 1 2 3 4 5 ... 200

Possible Causes

Changes to **long_query_time** on the console take effect globally, but only for new connections. As existing connections continue to use the original value (0.1s in this case), slow SQL queries whose execution duration was less than 0.2s are still reported for existing connections.

This problem is caused by the MySQL engine. Changing any of the parameters that take effect for all databases may also have this problem.

Solution

If you need certain session connections to use the new value, close the session connections and re-establish them.

3.2 Incorrect GROUP_CONCAT Results

Scenario

When the GROUP_CONCAT() function is used in an SQL statement, the result does not meet expectations.

Possible Causes

The GROUP_CONCAT() function returned a string result consisting of concatenated values in the group. However, the **group_concat_max_len** parameter limited the result length of this function.

Example:

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

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

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

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

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


Solution

Change the value of **group_concat_max_len** to adapt to the result length of the GROUP_CONCAT() function.

3.3 [ERROR] 1071 Reported When an Index Creation Fails for RDS for MySQL

Scenario

The index failed to be created because it was too long. The following error was reported:

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

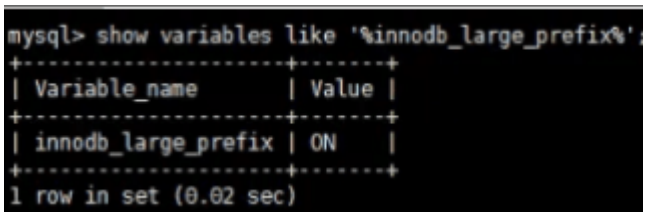
This problem may occur in MySQL-8.0.20.5.

Fault Analysis

The InnoDB table engine has a length limit on index prefixes.

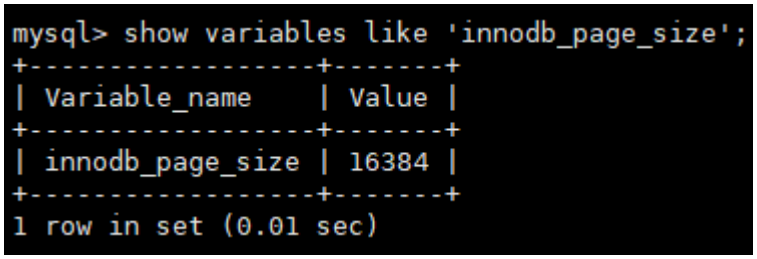
By default, an index prefix can contain a maximum of 767 bytes, but if **innodb_large_prefix** is set to **ON**, the index prefix length is increased to 3,072 bytes.

```
SHOW VARIABLES LIKE '%innodb_large_prefix%';
```



The length of an index prefix also depends on the InnoDB page size. If the parameter **innodb_page_size** is set to its default value 16 KB, the maximum index prefix length is 3,072 bytes, but if this parameter is set to 8 KB, the maximum index prefix length is 1,536 bytes. If this parameter is set to 4 KB, the maximum length of the index prefix is 768 bytes.

```
SHOW VARIABLES LIKE '%innodb_page_size%';
```



Check the structure of the problematic table and query all supported character sets and their byte usage:

```
SHOW CHARACTER SET;
```



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

When a problematic table uses the utf8mb4 character set, each character uses 4 bytes. This means that if the index prefix has 3072 bytes, it can only contain 768 (3072/4 = 768) characters. You just need to set the index prefix length in the CREATE TABLE statement to **768** or modify the index field to keep it less than 3072 bytes.

Solutions

Change the length of the index field.


```
create table IF NOT EXISTS `[redacted]`(
    [redacted]d` INTEGER NOT NULL AUTO_INCREMENT,
    [redacted]ct_id` VARCHAR(255) NOT NULL,
    [redacted]_id` VARCHAR(255) NOT NULL,
    [redacted]_INTEGER NOT NULL,
    [redacted]rce_id` VARCHAR(64) NOT NULL,
    [redacted]rce_type` VARCHAR(64) NOT NULL,
    [redacted]ey` VARCHAR(255) NOT NULL,
    [redacted]alue` VARCHAR(255) NULL,
    [redacted]e_time` TIMESTAMP NULL,
    [redacted]modified_time` TIMESTAMP NULL,
    PRIMARY KEY (`tag_id`),
    UNIQUE KEY `tag_key` (`resource_type`, `resource_id`, `tag_key`, `tag_value`)
)ENGINE = INNODB DEFAULT CHARSET = utf8mb4;
```

3.4 Tables Failed to Be Found After Case-Sensitivity Setting Changes for RDS for MySQL

Scenario

The RDS for MySQL parameter **lower_case_table_names** was set to case sensitive, and then a table containing uppercase letters was created. The parameter setting was later changed to case insensitive, and now the table containing uppercase letters, such as tbl_newsTalking, cannot be found.

Case: When a backup is restored to a new instance, restoration will fail if the parameter controlling case-sensitivity for the new instance is different from that of the original instance.

For more sensitive parameters, see **Suggestions on Tuning RDS for MySQL Parameters**.

NOTE

- For RDS for MySQL 5.6 and 5.7, you can specify case sensitivity for table names when creating an instance on the console or using APIs, or set the **lower_case_table_names** parameter after an instance is created.
- For RDS for MySQL 8.0, you can only specify case sensitivity for table names when creating an instance on the console or using APIs.
- For community version 8.0, **lower_case_table_names** can only be configured during server initialization. Changing the **lower_case_table_names** setting after the server is initialized is prohibited. For details, see the **community documentation**.

Solution

Step 1 **Log in to the management console**.

Step 2 Click in the upper left corner and select a region.

Step 3 Click in the upper left corner of the page and choose **Databases** > **Relational Database Service**.

Step 4 On the **Instances** page, click the target instance name to go to the **Overview** page.

Step 5 In the navigation pane, choose **Parameters**.

Step 6 Change the value of **lower_case_table_names** to **0**, indicating that table names are case sensitive.

Step 7 Click **Save**. In the displayed dialog box, click **Yes**.

Step 8 Return to the DB instance list, locate the DB instance, and choose **More** > **Reboot** in the **Operation** column.

Step 9 In the displayed dialog box, click **OK** to reboot the DB instance for the modification to take effect.

Step 10 Log in to the database and change uppercase letters in table names to lowercase letters.

Step 11 Change the value of **lower_case_table_names** to **1**, indicating that table names are case insensitive.

Step 12 Reboot the instance again.

----End

NOTE

- Database names and variable names must be case sensitive.
- Column names and aliases are case insensitive by default.
- You can set **Table Name** to **Case sensitive** or **Case insensitive** on the RDS console during instance creation. For details, see **Buying an RDS for MySQL DB Instance**.
- You can set parameter **lower_case_table_names** to **0** or **1** when calling an API to create a DB instance. For details, see **Creating a DB Instance**.

  Value range:

  - **0**: Table names are case sensitive.
  - **1**: Table names are stored in lowercase and are case insensitive.

3.5 Timeout Parameters

The following table lists the RDS for MySQL timeout parameters.

Table 3-1 Parameter description

Parameter	Reboot Required	Description
connect_timeout	No	Number of seconds that the mysqld server waits for a connection packet before responding with **Bad handshake**. If the network quality is poor, you can increase the value of this parameter.

Parameter	Reboot Required	Description
idle_readonly_transaction_timeout	No	Number of seconds that the server waits for idle read-only transactions before killing the connection. (Supported in versions later than 5.7.23)
idle_transaction_timeout	No	Number of seconds that the server waits for idle transactions before killing the connection. If this parameter is set to the default value **0**, connections are never killed. (Supported in versions later than 5.7.23)
idle_write_transaction_timeout	No	Number of seconds that the server waits for idle read-write transactions before killing the connection. If this parameter is set to the default value **0**, connections are never killed. (Supported in versions later than 5.7.23)
innodb_lock_wait_timeout	No	Number of seconds an InnoDB transaction waits for a row lock before giving up.
innodb_rollback_on_timeout	Yes	InnoDB rolls back only the last statement on a transaction timeout by default. If **innodb_rollback_on_timeout** is specified, a transaction timeout causes InnoDB to abort and roll back the entire transaction.
lock_wait_timeout	No	Timeout in seconds for attempts to acquire metadata locks
net_read_timeout	No	Number of seconds to wait for more data from a connection before aborting the read.
net_write_timeout	No	Number of seconds to wait for a network packet to be written to a TCP connection before aborting the write.
interactive_timeout	No	Number of seconds the server waits for activity on an interactive connection before closing it.
wait_timeout	No	Number of seconds the server waits for activity on a noninteractive connection before closing it.

3.6 Global Parameters Fail to Change

Scenario

A global parameter failed to be changed for an RDS for MySQL instance.

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

Solution

Change global parameters on the RDS console because RDS for MySQL does not support global parameter changes using commands. For details, see **Can I Use SQL Commands to Modify Global Parameters of My RDS Instance?**

4 Performance Issues

4.1 High CPU Usage

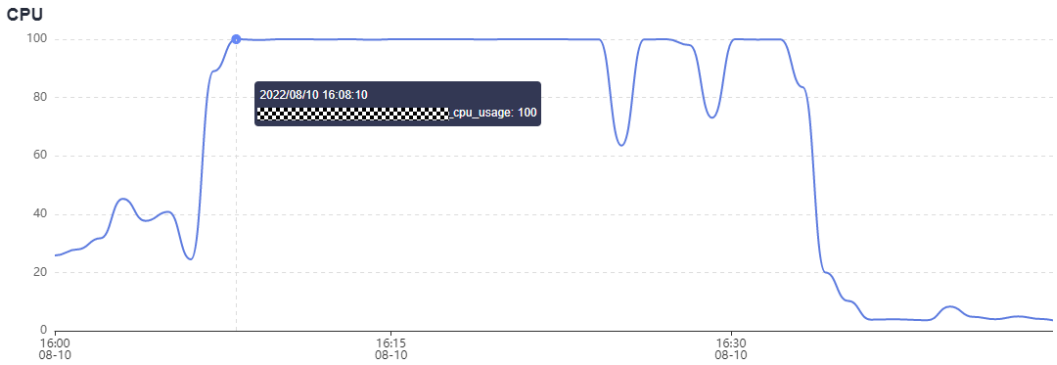
Scenario

The SQL statement execution of an RDS for MySQL instance slows down, and a timeout error is reported.

Troubleshooting

1. Check the CPU usage. In this example, the CPU usage of the instance soared to 100% at about 16:08 and remained at the high line.

   Figure 4-1 CPU usage

2. Check the QPS, slow SQL queries, and active connections. The QPS and active connections increased sharply at about 16:08 and a large number of slow SQL queries were generated.

Figure 4-2 QPS

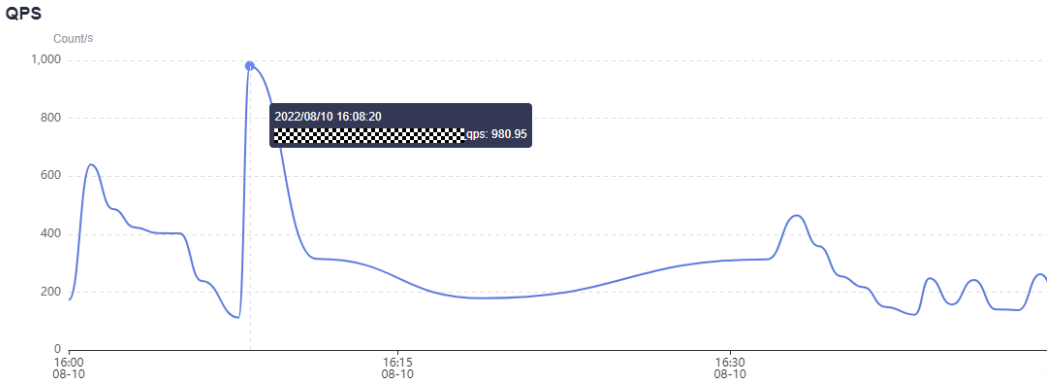


Figure 4-3 Active connections

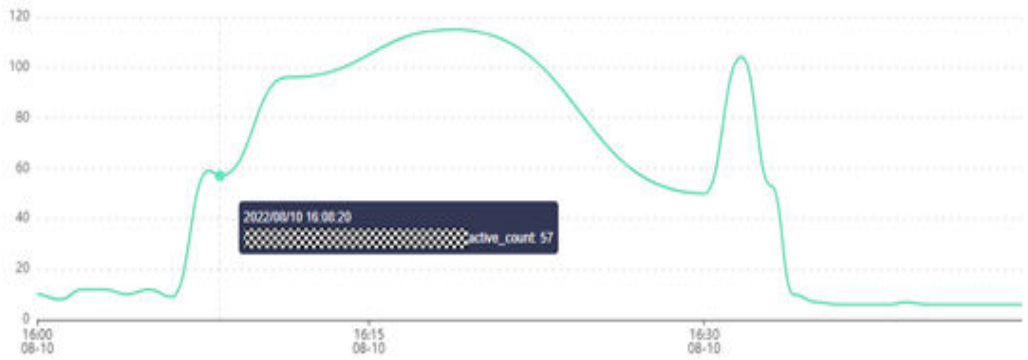
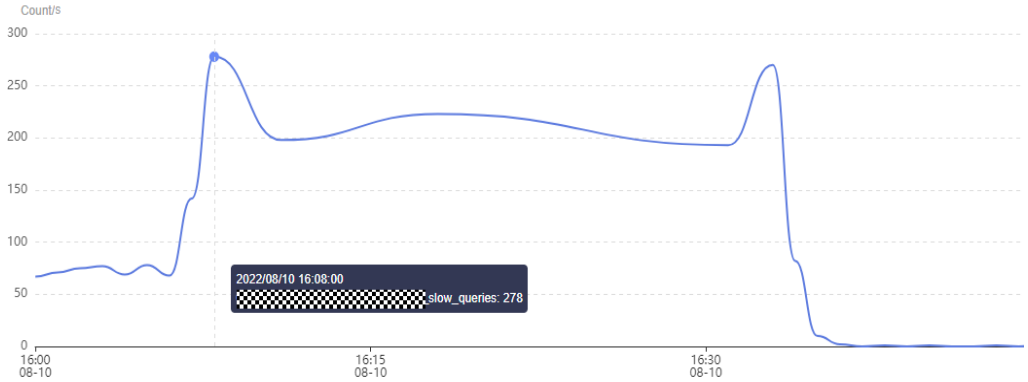
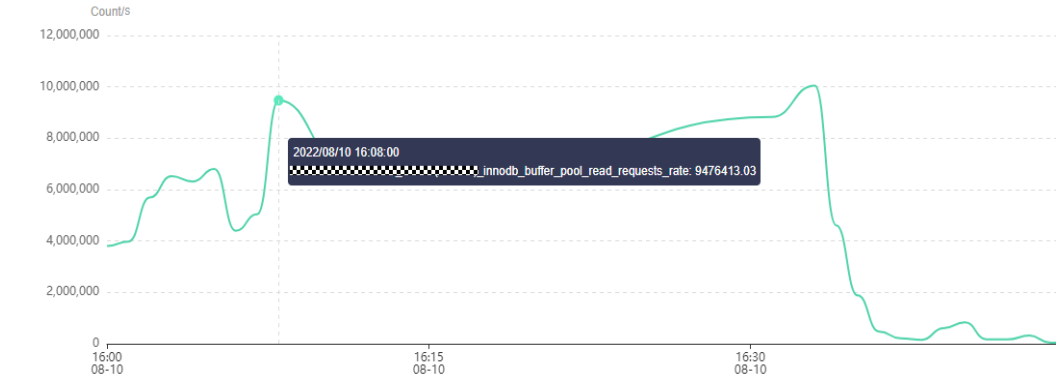


Figure 4-4 Slow SQL queries

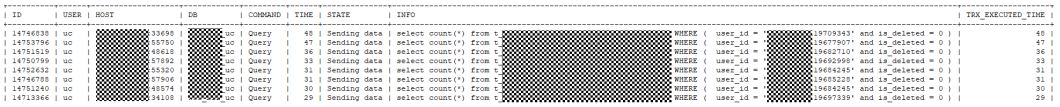


3. Check the InnoDB logical read rate. The logical read rate of InnoDB also increased sharply around 16:08, and the pattern was similar to that for the slow SQL queries.

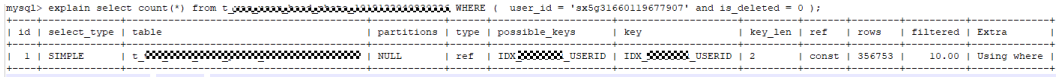
Figure 4-5 InnoDB logical read rate



4. Log in to the instance and check real-time sessions. There were a large number of sessions executing **SELECT COUNT(*)**.



Run **EXPLAIN** to check the execution plan of the SQL statement. It was found that the SQL statement was not indexed and the entire table was scanned with rows to be scanned reaching over 350,000.



5. Check the table structure. Only the IDX_XX_USERID index was added for the **is_deleted** field. Therefore, no index was available for the preceding query. After an index was added for the **idx_user_id** field, the CPU usage of the instance decreased to a normal level at about 16:37 and services were recovered.

Solution

1. Before deploying new workloads, use **EXPLAIN** and SQL diagnosis tools to analyze the execution plans of key SQL statements and add indexes based on the optimization suggestions to avoid full table scanning.
2. If the high CPU usage is caused by a large number of concurrent requests, upgrade the DB instance specifications or use exclusive resources to avoid CPU contention, or create read replicas to reduce the read pressure of the primary instance.
3. Use **show processlist** to view the current session information to locate the fault. Any session whose status is **Sending data**, **Copying to tmp table**, **Copying to tmp table on disk**, **Sorting result**, or **Using filesort** may have performance problems.
4. In emergency scenarios, enable SQL throttling or kill sessions to temporarily limit the number of slow SQL queries.

4.2 Out of Memory (OOM) Errors

RDS for MySQL Memory Description

The memory of an RDS for MySQL instance can be roughly divided into two parts: globally shared memory and session-level private memory.

- Shared memory is allocated upon the creation of an instance based on parameter settings and is shared by all connections.
- Private memory is allocated by the system upon connection to the RDS for MySQL instance and is released only when the connection is released.

Inefficient SQL statements or improper database parameter settings may increase memory usage and even cause an OOM error during peak hours.

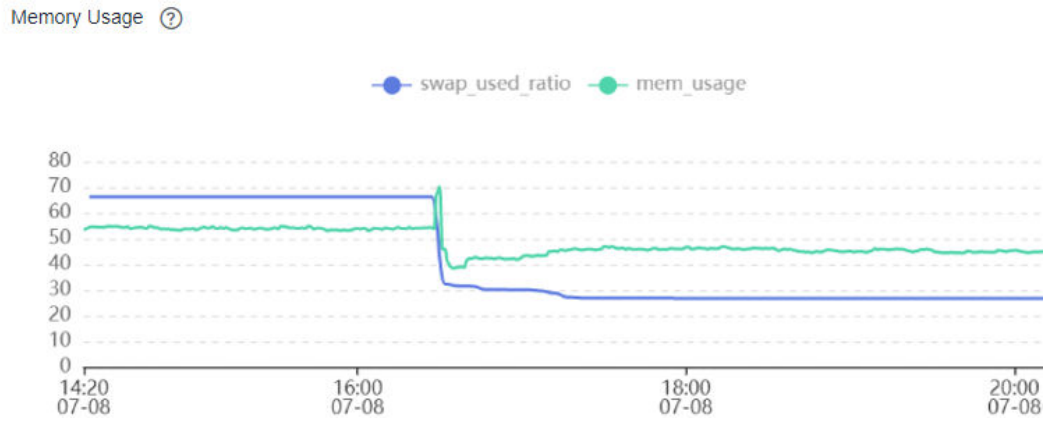
Scenario

The memory usage of an RDS for MySQL instance increased sharply. An OOM error occurred and then the instance restarted.

Troubleshooting

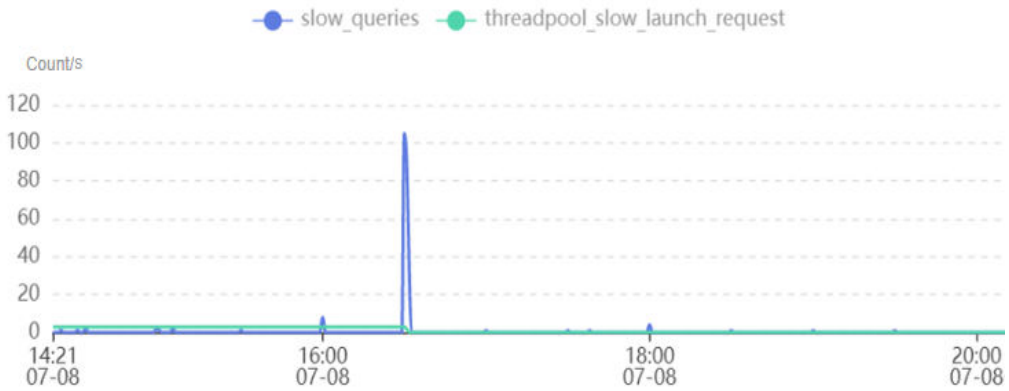
1. Check the memory usage. In this example, it shot up around 16:30.

   Figure 4-6 Memory usage

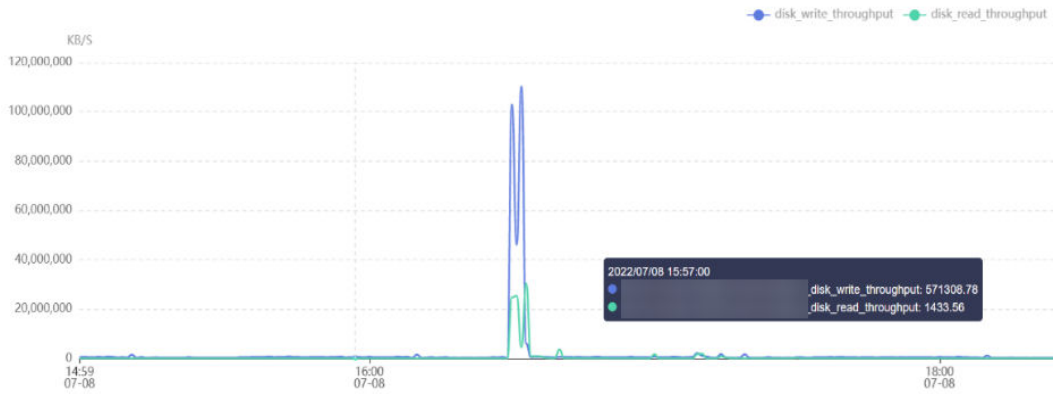
2. Check for slow SQL queries. The number of slow SQL queries increased sharply in that period.

Figure 4-7 Slow SQL queries



3. Check the disk throughput. There were a large number of read and write operations being performed on the disk in that period.

Figure 4-8 Disk throughput



4. Analyze slow query logs generated in that period. There were a large number of multi-value INSERT statements, which cause every session to request a large amount of session-level memory at the same time. Therefore, an OOM error occurred.

Figure 4-9 Slow query logs

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

Solution

1. For the OOM error caused by multi-value INSERT statements, reduce the amount of data inserted at a time and disconnect sessions to release memory. You can run the **show full processlist** command to check whether there are sessions with high memory usage.
2. Set the session-level memory parameter to an appropriate value. You can estimate the maximum memory based on the following formula: Global memory + Session-level memory x Maximum number of sessions. Note that setting **performance_schema** to **ON** also causes memory overhead.
3. Upgrade the instance specifications to maintain the memory usage within a proper range, preventing a sudden increase in traffic from causing an OOM crash.

4.3 Insufficient Disk Bandwidth

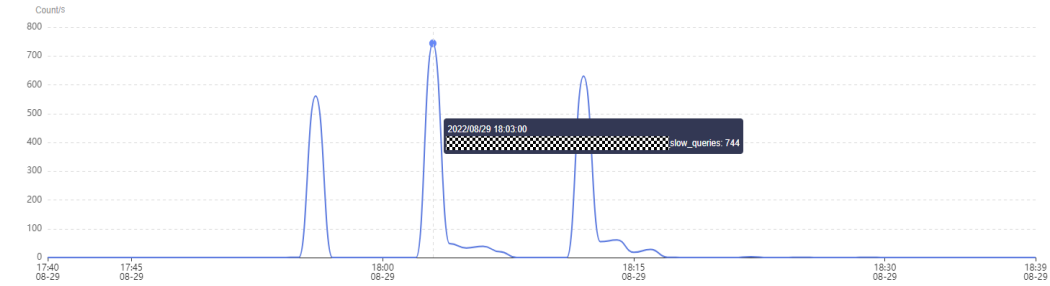
Scenario

The SQL statement execution of an RDS for MySQL instance slows down (for more than 5 seconds), and an error is reported due to a timeout.

Troubleshooting

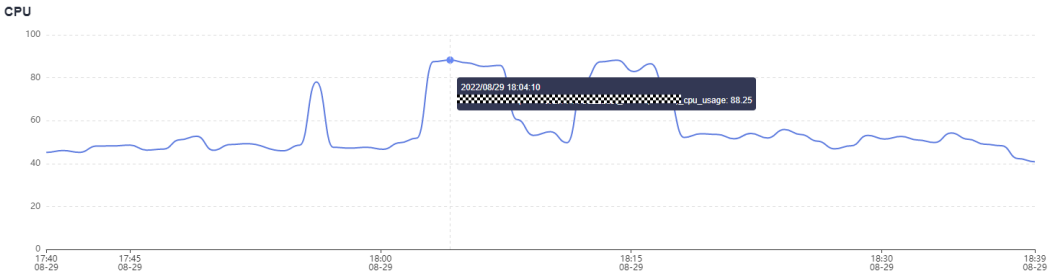
1. Check for slow SQL queries. In this example, slow SQL queries of the instance started to increase at 18:03 and reached up to 700 per second.

   Figure 4-10 Slow SQL queries

2. Check the CPU usage. The CPU usage was 88% at the time and was not a performance bottleneck.

   Figure 4-11 CPU usage

3. Check the QPS. The QPS increased by more than three times from 18:03 to 18:05, indicating that the service was being provided during peak hours.

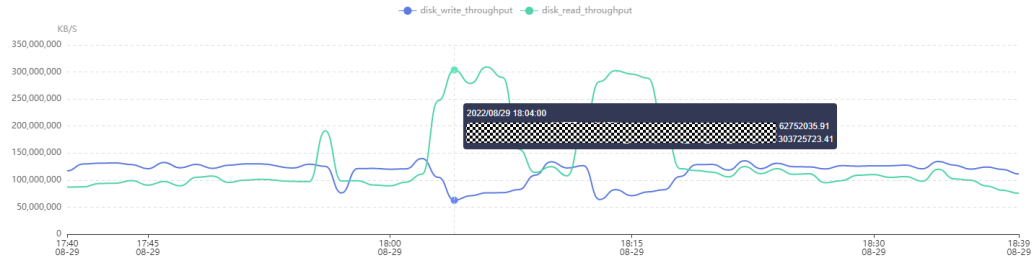
   Figure 4-12 QPS

4. Check the disk read and write throughput. The disk throughput reached 350 MB/s, making it a performance bottleneck.

For details about storage performance, see **DB Instance Storage Types**.

Figure 4-13 Disk throughput



Solution

1. Adjust the value of **innodb_io_capacity** or **innodb_io_capacity_max** to prevent high I/O throughput because underlying data reads and writes will generate physical I/Os if the requested data page cannot be hit in the buffer pool.
2. Purchase a DB instance with high-performance extreme SSD as the storage type or upgrade the instance memory specifications to cache more data to the buffer pool.

4.4 Slow SQL Statements Due to Improper Composite Index Settings

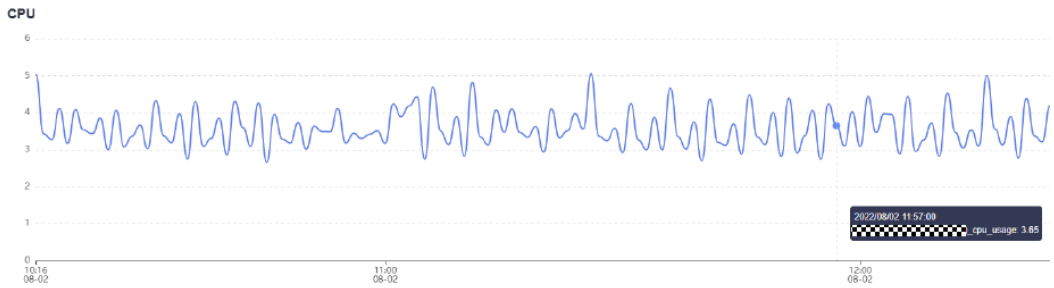
Scenario

On an RDS for MySQL instance, an SQL query that ran at 11:00 and was expected to take 8 seconds took more than 30 seconds.

Troubleshooting

1. Check the CPU usage. In this example, during that time period, the CPU usage of the instance did not increase sharply and remained low, so we know that the slow query was not caused by high CPU usage.

Figure 4-14 CPU usage



2. Analyze slow query logs generated in that time period. In this example, shown below, there were several SQL statements that involved millions of rows being scanned. These were the slow statements. But no large amount of data was inserted into the table during that time, so we know that the slow execution

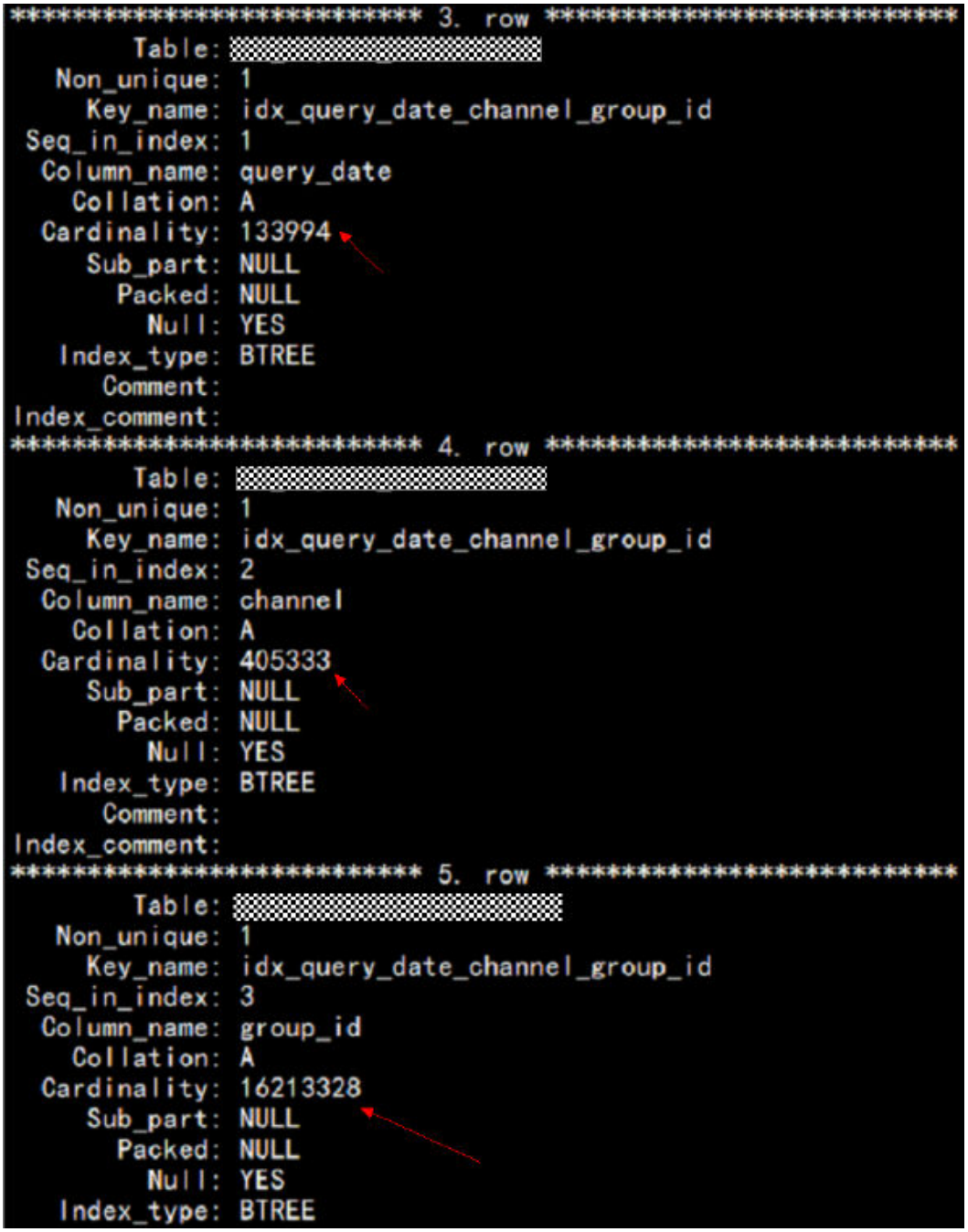
was caused by missing or incorrect index settings. By running **EXPLAIN**, you can find that the execution plan of the SQL statement was full table scanning.

Figure 4-15 Slow query logs

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

3. Run **SHOW INDEX FROM** to check the index cardinalities of the three fields.

Figure 4-16 Viewing index cardinality



The **query_date** field with the smallest cardinality was in the first place of the composite index, and the **group_id** field with the largest cardinality was in the last place of the composite index. In addition, the SQL statement contained the range query of the **query_date** field. As a result, only the **query_date** field was indexed. Therefore, the SQL statement could only use the index of the **query_date** column. Additionally, the optimizer may have selected full table scanning during cost estimation because the cardinality was too small.

A new composite index was created with the **group_id** field in the first place and the **query_date** field in the last place. The query time met the expectation.

Solution

1. Check whether the slow query is caused by a performance bottleneck, such as insufficient CPU resources.
2. Check whether the table structure is properly designed and whether index settings are correct.
3. Execute the **ANALYZE TABLE** statement periodically to prevent incorrect execution plans because performing a large number of INSERT or DELETE operations for table data may result in outdated statistics.

4.5 DB Instance Becoming Read-Only Due to Insufficient Storage

Scenario

The following error is displayed for an RDS for MySQL instance:

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

Troubleshooting

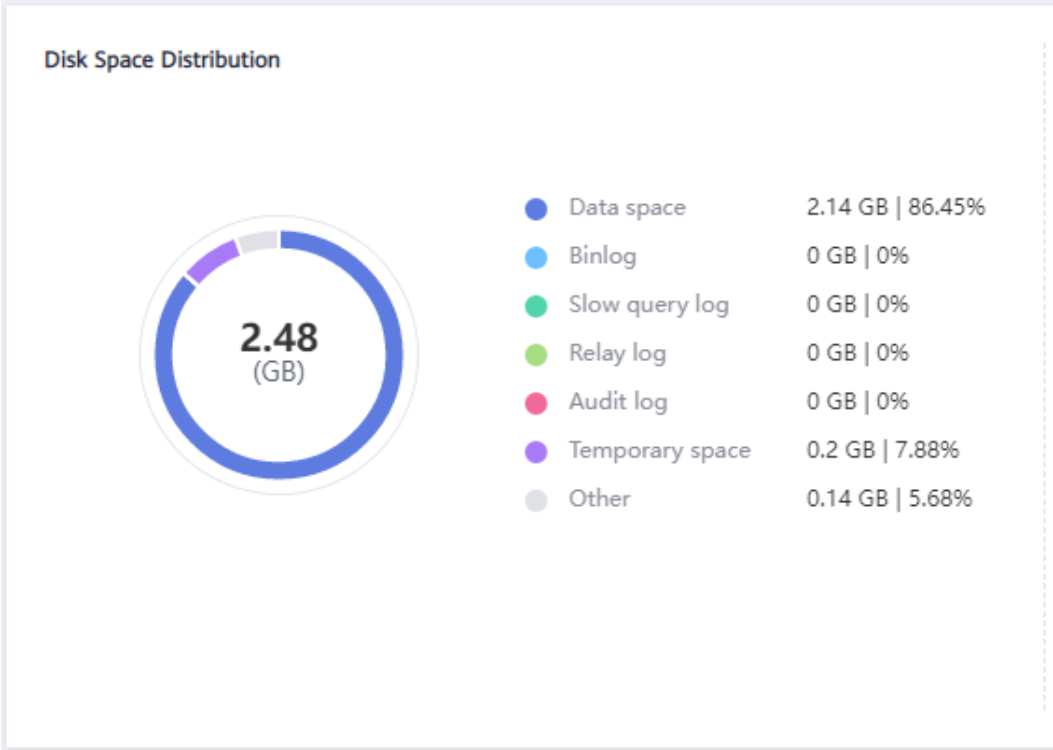
1. Go to the instance details page and check whether the storage is full.

2. Log in to the database and check the **read_only** variable.

   show variables like 'read_only';

3. From the previous steps, you can find that the storage space is full and the instance status has changed to read-only. As a result, the SQL statement fails to be executed.
4. Check disk space distribution on the **Storage Analysis** page. For details, see **Storage Analysis**.

Figure 4-17 Disk space distribution



Solution

1. For insufficient storage caused by increased workload data, **scale up storage space**.

   If the original storage has reached the maximum, **upgrade the specifications** first.

   For instances using cloud disks, you can configure **storage autoscaling** so that RDS can autoscale your storage when the storage usage reaches the specified threshold.

2. If too much data is stored, delete unnecessary historical data.

   a. If the instance becomes read-only, **unlock it from this state first**. If it is not read-only, delete the data directly.

   b. Check the top 50 databases and tables with large physical files and identify the historical table data that can be deleted. For details, see **Storage Analysis**.

   c. To clear up space, you can optimize tables with a high fragmentation rate during off-peak hours.

      To delete data of an entire table, run **DROP** or **TRUNCATE**. To delete part of table data, run **DELETE** and **OPTIMIZE TABLE**.

3. If binlog files occupy too much space, **clear local binlogs**.

4. If temporary files generated by sorting queries occupy too much storage space, optimize your SQL statements.

   You can query **slow query logs** and **top SQL statements**, and analyze and optimize the problematic SQL statements.

5. Subscribe to daily health reports to obtain SQL and performance analysis results, including slow SQL analysis, all SQL analysis, performance & storage analysis, and performance metric trend charts. You can receive a diagnosis report if there are any risks detected.

   For details, see **Daily Reports**.

4.6 High Storage Usage Due to Uncleared Old Binlogs

Scenario

The storage usage of a read replica or primary instance was high. By running **SHOW BINARY LOGS** or **SHOW MASTER LOGS**, it was found that a large number of old binlogs were not cleared.

Possible Causes

In most cases, if the **binlog retention period** is configured, the binlogs that have been backed up to OBS but exceed the retention period will be automatically deleted. If binlogs are not deleted for a long period of time, check whether it is caused by replication exceptions.

Troubleshooting:

1. **Check error logs of the instance** for any records about binlog purging failures.
   ```
   2022-01-18T05:39:03.139207+08:00 29 [Warning] file ./mysql-bin.106259 was not purged because it was being readby thread number 27490757
   ```
2. Check whether a local standby database exists or whether tools such as Canal are used to listen to binlogs of the instance. If the primary database does not receive the information that the binlogs have been obtained by the standby database or tool, the binlogs will not be deleted. As a result, binlogs are stacked.
3. Based on the records obtained in **1**, analyze the logs of the local standby database or Canal and check network connectivity to identify the cause.

Solution

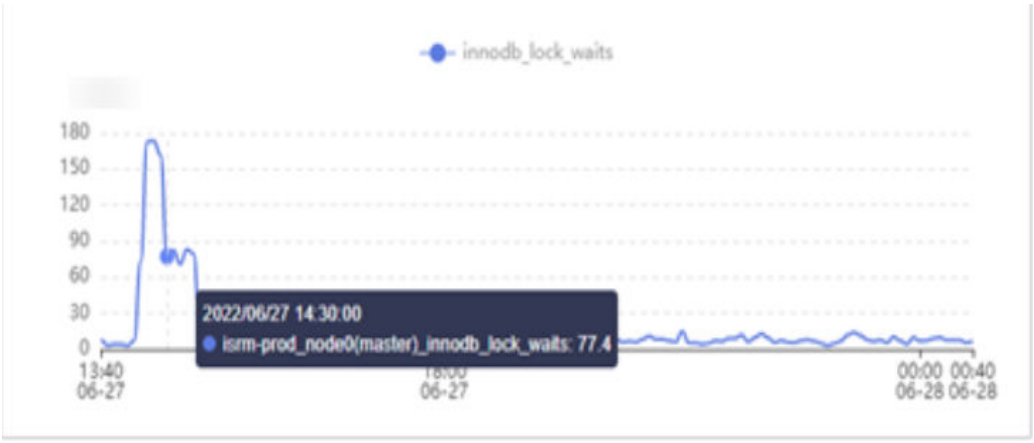
- Temporarily stop the binlog listening task of the instance so that the instance can automatically clear binlogs.
- If there is a local standby database, re-establish the replication relationship.
- If a tool such as Canal is used, re-create a binlog pull task.

4.7 Slow Response Due to Deadlocks

Scenario

A large number of row lock conflicts occurred in a database between 14:00 and 15:00. The database response became slow because a large number of update and insert sessions in the kernel were waiting for row lock release and the CPU usage reached about 70%.

The following figure shows the row lock waits and metadata locks on the Cloud Eye console.



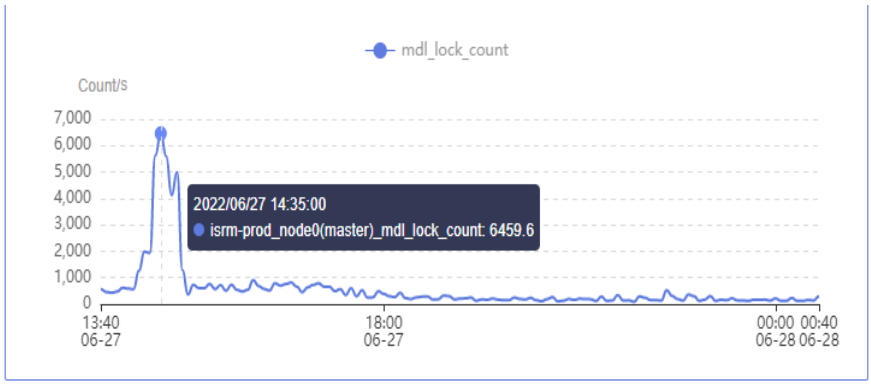


Table where a deadlock occurred:

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

Possible Causes

1. Deadlocks occurred in some tables. As a result, the CPU usage increased.
2. If a table contains a large number of foreign keys, updating records in the table requires not only the row lock of the table but also the corresponding locks of the tables associated with its foreign keys. In high concurrency scenarios, lock conflicts or deadlocks are more likely to occur than common tables. For details, see **FOREIGN KEY Constraints**.
3. If it detects a deadlocked table, RDS for MySQL rolls back the transaction. The tables associated with the foreign keys of the deadlocked table are also impacted. As a result, the database responsiveness slows down.

Solution

Check and optimize deadlocked tables and use the right foreign keys to avoid update conflicts and deadlocks.

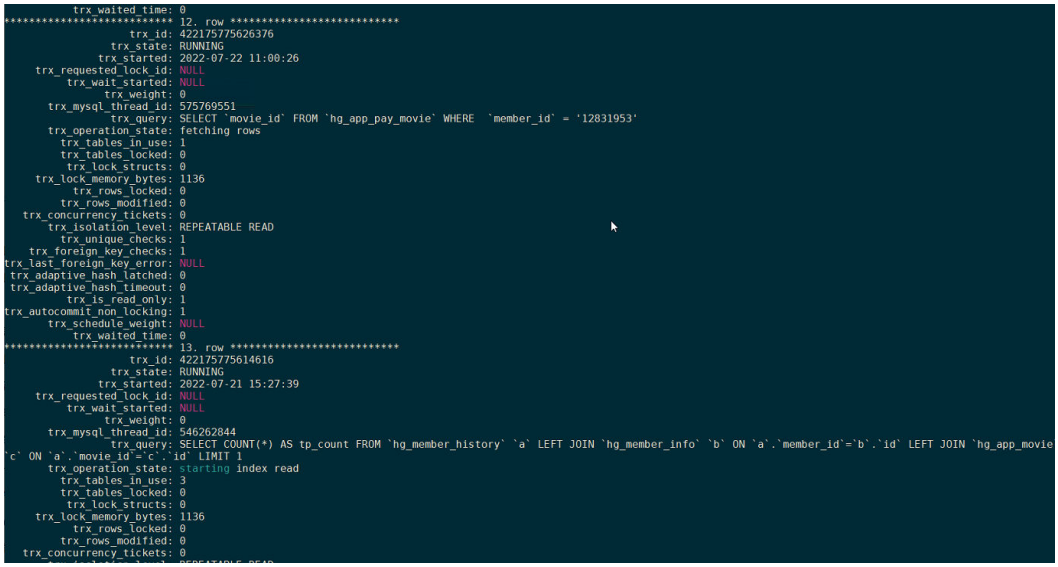
4.8 Read Replica Uses Far More Storage Than the Primary Instance

Scenario

The storage usage of an RDS for MySQL read replica was, for example, 195 GB higher than that of the primary DB instance.

Troubleshooting

Check the transactions running on the read replica.



As shown in the preceding figure, there was a long transaction started a day ago that has still not committed. The undo logs accumulated for almost a whole day and were not cleared, which takes up a lot of storage.

Solution

- Method 1: Wait until the transaction is committed. After that, the undo logs will be cleared, releasing storage space.
- Method 2: Kill the corresponding session to stop the long transaction.

4.9 CPU Usage Increase

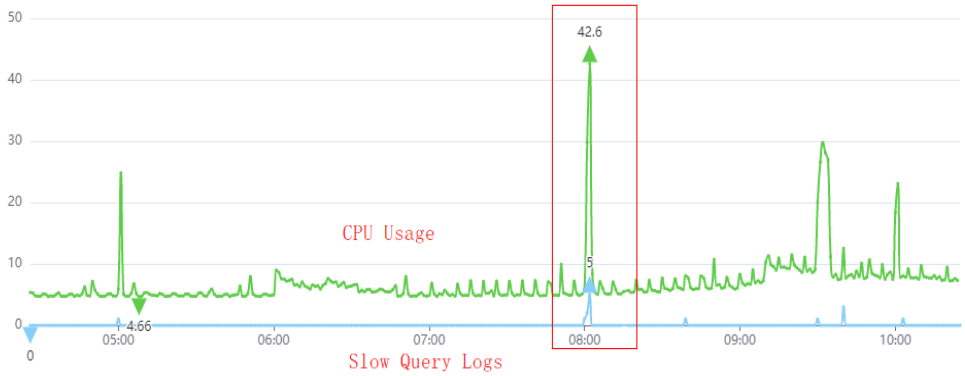
If the CPU usage of an RDS for MySQL instance increases or reaches 100%, the database response may become slow and new connections may time out.

Scenario 1: CPU Usage Increase Caused by Slow Queries

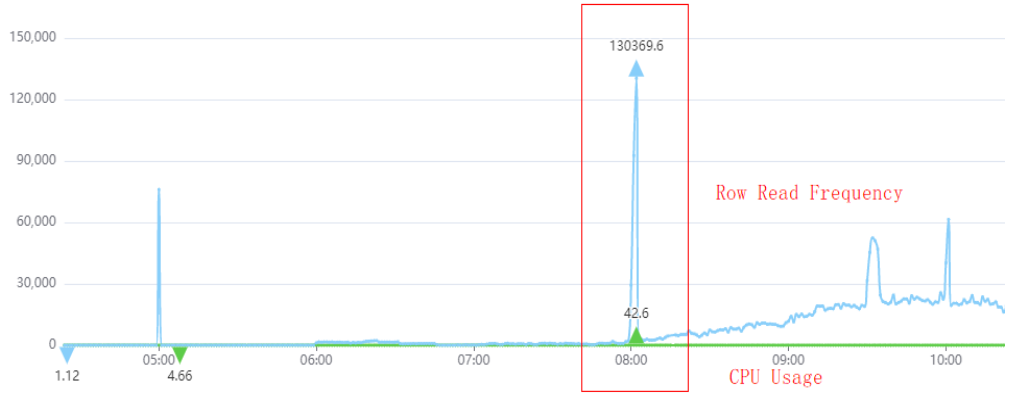
Cause: Too many slow SQL queries increase CPU usage. The slow SQL queries need to be optimized.

Troubleshooting

View the CPU usage and slow query logs.



- If a large number of slow query logs are generated and the change is consistent with the CPU usage curve, you know it is the slow SQL queries that are increasing the CPU usage.
- If there are not very many slow query logs but the change is basically consistent with the CPU usage curve, check whether the row read rate change is consistent with the CPU curve.



If yes, the CPU usage increase is caused by access to a large amount of row data. Although there are a small number of slow SQL queries, the queries need to access a large amount of row data, causing high average I/O. Therefore, even if the QPS is not high (for example, the website access traffic is not heavy), the CPU usage of the instance is also high.

Solution:

1. View slow query logs generated within the corresponding time period.
2. Note any slow queries with more than a million rows scanned or more than a million rows returned, and slow queries with long lock waiting time.

3. Analyze slow queries or use **SQL Diagnosis**.
4. Create read replicas and enable Database Proxy to split read and write requests. Read replicas can offload the read pressure from the primary instance, thus improving the database throughput. For details, see **Introducing Read/Write Splitting**.
5. Analyze live sessions on the database to locate slow SQL statements.
   a. Connect to the database.
   b. Run the **show full processlist;** command.
   c. Analyze sessions that take a long time to execute and are in the **Sending data**, **Copying to tmp table**, **Copying to tmp table on disk**, **Sorting result**, or **Using filesort** state.

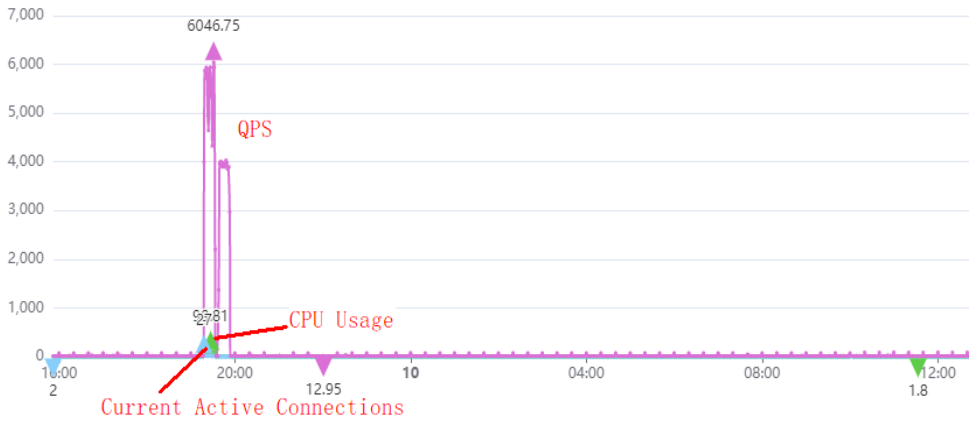
Scenario 2: CPU Usage Increase Caused by Increased Connections and QPS

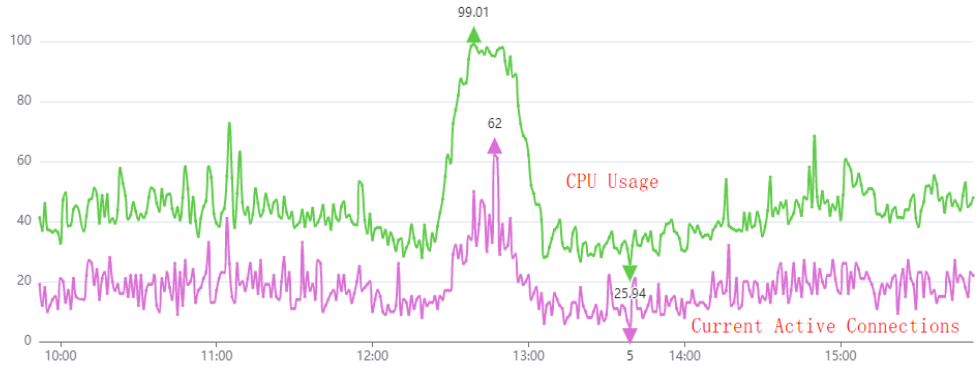
Cause: Too many requests

Troubleshooting:

Check whether the changes of the QPS, active connections, total connections, and CPU usage are consistent.

QPS refers to the number of queries per second. If the QPS and active connections increase at the same time, and the QPS curve matches the CPU usage curve, the CPU usage increase is caused by increased requests, as shown in the following figure.





In this scenario, SQL statements are usually simple and the execution efficiency is high. There is little room for optimization on SQL statements. You need to optimize the database.

Solution:

1. Upgrade the vCPU specifications of your instance because this problem usually occurs in instances with vCPU specifications such as 1 vCPU, 2 vCPUs, and 4 vCPUs.
2. Optimize slow queries by referring to **Scenario 1: CPU Usage Increase Caused by Slow Queries**. If this method is not so helpful, upgrade the vCPU specifications of your instance.
3. Use database and table sharding for tables with a large amount of data to reduce the amount of data accessed in a single query.
4. Create read replicas and enable Database Proxy to split read and write requests. Read replicas can offload the read pressure from the primary instance, thus improving the database throughput. For details, see **Introducing Read/Write Splitting**.

4.10 Slow SQL Execution Due to Hot and Cold Data Problems

Scenario

When you migrate data from a self-built MySQL database or a peer vendor's MySQL database to an RDS for MySQL instance, the execution speed of the same SQL statement is much lower than that of the source database.

Possible Causes

The execution speed of a given SQL statement differs greatly when it is executed the first and second times. This is determined by the MySQL buffer pool mechanism.

- The first time the statement is executed, data is stored on the disk. This data is cold data. Reading cold data takes a certain period of time.
- The data you have queried is then cached in the buffer pool of the memory. It is called hot data and can be quickly accessed in the memory. When you

execute the same statement for the second time, data is read from the buffer pool, which is much faster than reading data from a disk.

In this example, the data you queried in the source database is frequently accessed data. It is hot data. It can be read very quickly. After the data is migrated to the RDS for MySQL instance, when you execute the same SQL statement on the new database for the first time, the data you expect to query is probably cold data. This time, the access speed is slow. If you run the statement again, the data will become hot data again and the speed will improve significantly.

Solution

This issue is not an exception. Within a given database, it usually takes much more time to execute a statement for the first time, but when the statement is executed again later, it gets much faster. The access speed improves because the subsequent reads are reads of hot data from the buffer pool, which is much faster than reading cold data from a disk.

4.11 High Table Fragmentation Rate

Scenario

High table fragmentation rate is a common problem in RDS for MySQL instances. Table fragments mean that table data and indexes are scattered in different physical blocks. These physical blocks may be discontinuous or have some free space, so the storage of table data and indexes on disks is not optimal.

This problem is caused by operations (such as deletion, update, and insertion) on table data. If data rows in tables are frequently modified and moved, data segments in the tables become discontinuous.

Impact and Risk

- Tablespace bloat

  High table fragmentation rate causes a large amount of unused space in the instance. It is a waste of space.

- Poor query optimization

  If the table fragmentation rate is too high, the optimizer cannot correctly and effectively use indexes, affecting execution plan selection and degrading the query performance.

- Slow SQL execution

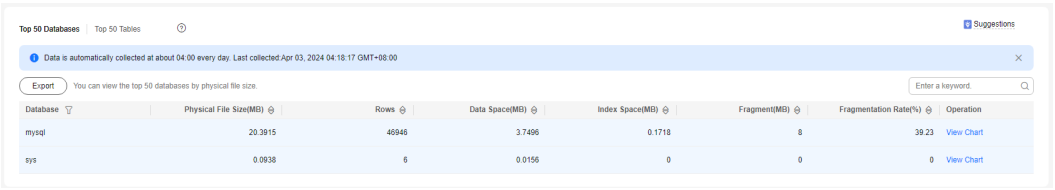
  If the table fragmentation rate is too high, extra time is required for I/O scanning and defragmentation when SQL statements are executed. As a result, the query and update operations are slow and the response time is prolonged.

Troubleshooting

Method 1: Use DBA Assistant to view the storage usage of your DB instance in real time to prevent storage space from being insufficient.

1. **Log in to the management console**.
2. Click in the upper left corner and select a region.
3. Click in the upper left corner of the page and choose **Databases** > **Relational Database Service**.
4. On the **Instances** page, click the DB instance name.
5. In the navigation pane, choose **DBA Assistant** > **Real-Time Diagnosis**.
6. Click the **Storage Analysis** tab. On the displayed page, you can view the fragment space and fragmentation rate in the top 50 databases and tables area.

Figure 4-18 Top 50 databases and tables



Method 2: Run commands to view the fragmentation rate.

1. Run the following command to analyze a table and update the table statistics:
   ```
   ANALYZE TABLE table_name;
   ```
2. Run the following commands to view details about a table:
   ```
   SELECT
      table_name,
      data_length,
      data_free
   FROM
      information_schema.tables
   WHERE
      table_schema = 'database_name'
      AND
      table_name = 'table_name';
   ```
   - *table_name*: name of the table
   - *data_length*: size of data stored in the table (in byte)
   - *data_free*: size of free space of the table (in byte)

   Generally, you can preliminarily determine the fragmentation rate based on the ratio of **data_free** to **data_length**.

Possible Causes

Cause 1: Parallel Migration During DRS Full Migration

During a full migration, DRS uses row-level parallel migration to ensure migration performance and transmission stability. If the source database data is compact, a high fragmentation rate may cause data bloat after data is migrated to RDS for MySQL. As a result, storage usage of the destination database is much higher than that of the source database.

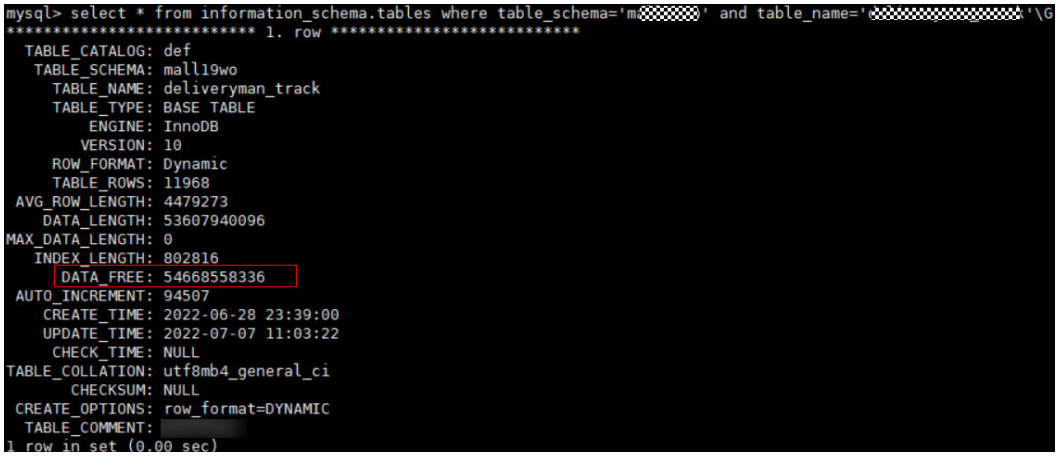
Cause 2: Table Fragmentation After a Large Number of Deletions

When data is deleted, RDS for MySQL does not reclaim the storage occupied by the deleted data. Instead, it only marks the deletion and fills the space with new data if there is any. If there is no data to fill up the space, tablespace bloat is the result, along with table fragmentation.

You can run the SQL statement shown below to query details about a table. The **DATA_FREE** field in the command output indicates the size of tablespace fragments.

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

Figure 4-19 Command output



Solution

Optimize the table fragmentation rate in the following scenarios:

- The instance has been running for a long period of time.

  Data operations, such as insertion, update, and deletion, may generate table fragments.
- There are a large number of data changes.

  A large number of data changes may cause fragments.
- Database performance deteriorates.

  If you identify obvious performance deterioration when querying a given amount of data, you may need to check the fragmentation rate.
- The storage space is insufficient.

  If the storage space usage is too high, you can check the fragment space and defragment the table to release the storage space.

To solve the problem of high table fragmentation rate, you are advised to periodically analyze fragments of frequently accessed tables, clear the fragments, and optimize tablespaces to improve performance.

To optimize a table, run the following command:

```
OPTIMIZE TABLE table_name;
```


NOTICE

The **optimize table** statement locks the table for a short period of time. The overall execution time depends on the table size. Generally, the execution takes a long time and occupies many resources (the storage space that is 1.5 times the size of the table to be optimized must be reserved). To avoid impact on your workloads, you are advised to optimize a table during off-peak hours.

4.12 Full Storage Caused by Complex Queries

Scenario

The storage usage of a primary instance or read replica is occasionally high or reaches 100%, while the storage usage of the standby instance or other read replicas is within a normal range.

Possible Causes

When you run complex queries on the data of an RDS for MySQL database, RDS creates temporary tables to store the data and operations such as GROUP BY, ORDER BY, DISTINCT, and UNION are executed on the data in the temporary tables. When memory is insufficient, storage space is consumed.

Troubleshooting:

1. Check the storage usage of the standby instance and other read replicas. If the storage usage of such instances is normal, the high storage usage of the primary instance or read replica is related to SQL queries running on it.
2. Check the instance slow query logs to find whether there were any slow queries when the storage usage was high.
3. If there is a slow query, run the **explain** *[slow SQL statement]* command to analyze the SQL statement.
4. Check whether the **extra** column in the command output contains **using temporary** or **using filesort**. If yes, a temporary table or file is used during the statement execution. If a large amount of data is queried, the storage usage is high.

Solution

1. Optimize the query statement by adopting the following measures:
   - Add a proper index.
   - Use the WHERE condition.
   - Rewrite the SQL statement to optimize the execution plan.
   - If temporary tables are necessary, reduce the number of concurrent requests.
2. Workaround: Temporarily scale up storage space. Optimizing complex query statements cannot reduce the storage usage right away.

4.13 Why Is My SQL Query So Slow?

1. You can view the slow SQL logs for slow SQL queries and view their performance characteristics (if any) to locate the cause.

   To learn how to view RDS for MySQL logs, see **Viewing and Downloading Slow Query Logs**.
2. View the CPU usage metric of your DB instance.

   For details, see **Viewing Performance Metrics of a DB Instance**.
3. Create read replicas to offload read pressure on primary DB instances.
4. When multiple associated tables are queried, indexes must be created for the associated fields.
5. Do not use the SELECT statement to scan all tables. You can specify fields or add the WHERE condition.

4.14 Instance Class Change or Minor Version Upgrade Failure Caused by Long Transactions

Scenario

An instance class change or a minor version upgrade fails due to long transactions.

Possible Causes

- To minimize the impact on workloads, an instance class change or a minor version upgrade is performed in rolling mode, so a primary/standby switchover is required.
- To ensure data consistency, the primary instance needs to be set to read-only before the switchover. This is a requirement for a successful class change or version upgrade.
- If there are long transactions executing on the primary instance, the primary instance may not be able to be set to read only. This operation may time out or fail. As a result, the instance class change or minor version upgrade fails.

Solution

1. Run the **show processlist** command to view the transactions that are being executed. Then run the following command to identify which one is taking too long:

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

   Example output:


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

In the preceding command output, **idle_time** is the calculated time the transaction stays idle. **trx_mysql_thread_id** indicates the thread ID of the transaction, which is the thread ID queried by running **show processlist**.

The value of **trx_query** is **NULL**, but it does not mean that the transaction is not executed. If the transaction contains multiple SQL statements and all of them have been executed, no statement is displayed in the **trx_query** field. If the transaction is being executed, InnoDB cannot determine whether there are any subsequent SQL statements and when the transaction is committed. In this case, **trx_query** cannot determine which SQL statement is being executed, so **NULL** is displayed.

2. Kill the long transaction, and then change the instance class or upgrade the minor version.
3. Do not execute any long transactions when you are changing the instance class or upgrading the minor version.

4.15 Native Error 1461 Reported by an RDS for MySQL DB Instance

Scenario

The following error is displayed when there are a lot of concurrent read and write requests, large amounts of SQL statements, or in data migration scenarios:

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

Fault Analysis

The **max_prepared_stmt_count** value ranges from 0 to 1048576. The default value is **16382**. This parameter limits the total number of prepared statements in

all sessions on mysqld. The current value exceeds the value range of this parameter.

Solution

Set **max_prepared_stmt_count** to a larger value. The recommended value is **65535**.

4.16 System Inaccessible After Field Addition to an RDS for MySQL Database Table

Description

After a field was added to an RDS for MySQL database table, the system becomes inaccessible.

Solution

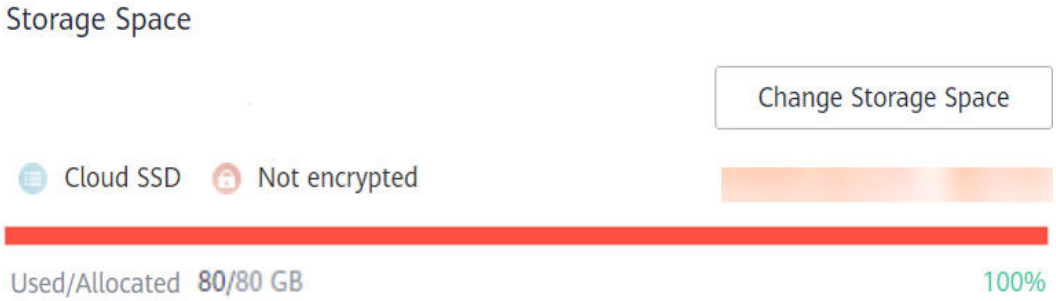
The database performance is affected due to the addition of table fields. A possible reason is that indexes are not added to the new table fields. As a result, a large amount of data consumes a large number of CPU resources. You are advised to:

- Add indexes and primary keys.
- Optimize slow SQL statements.

4.17 Storage Filled Up by Undo Logs Due to Long Transactions

Scenario

An instance triggered an alarm reporting full storage. After a period of time, the alarm was automatically cleared.



Possible Causes

- Because Multi-Version Concurrency Control (MVCC) is used, RDS generates undo logs when you update data in database tables and stores the undo logs in your storage. Only after the transactions of all your sessions are committed or rolled back, the undo logs can be removed from the storage.

- If there is a long-running transaction in one of the sessions, the undo logs generated for the table update sessions cannot be cleared as long as the long transaction is not committed. As a result, the storage usage keeps increasing.

Troubleshooting:

- Run the following statement to check whether any transaction has been running for a long period of time and is still not committed:

  select t.*,to_seconds(now())-to_seconds(t.trx_started) idle_time from INFORMATION_SCHEMA.INNODB_TRX t \G;
- View audit logs or slow query logs to check whether a large amount of data is inserted at a time by a large transaction.

Solution

- Kill the long-running transaction.
- Do not execute long-running transactions or insert a large amount of data when the storage is limited.
- Scale up storage space in advance.

4.18 Locating Long Transactions

Scenario

If a long transaction alarm is generated, you can locate the long transaction.

Troubleshooting

Run the following statement to check the execution duration of the current transaction and determine whether it is a long transaction:

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

In the command output, **trx_query** indicates the SQL statement that is being executed by the transaction. If its value is **NULL**, the transaction is in the waiting state and does not execute any SQL statement.

For details, see **The INFORMATION_SCHEMA INNODB_TRX Table**.

4.19 Sharp Increase in the Commit Time of Some SQL Statements

Scenario

The commit time of some SQL statements of an RDS for MySQL instance occasionally surges from several milliseconds to hundreds of milliseconds.

Possible Causes

When the thread pool is enabled, SQL requests are put in the task queue and wait to be processed by the worker thread. There is no performance optimization for

low-concurrency persistent connections. As a result, a short delay may occur due to the thread pool scheduling mechanism.

If there are a large number of concurrent requests or short connections, performance may deteriorate due to too much thread creation and destruction and context switching.

Solution

Set the **threadpool_enabled** parameter to **OFF**, restart the application or database during off-peak hours, and observe the delay. This setting takes effect immediately only for new connections.

4.20 Oversized ibdata1

Possible Causes

ibdata1 is the system tablespace of InnoDB. It contains:

- Data related to multi-version concurrency control (MVCC): undo logs
- Metadata of InnoDB tables, such as data dictionaries
- Change buffer and double write buffer

The primary reason for the increase in ibdata1 is excessive undo logs. The reasons for excessive undo logs are as follows:

- Long-running transactions not committed block undo logs from being purged.
- A large number of concurrent writes lead to excessive undo logs which cannot be purged fast enough.

To see the number of undo logs that are not purged, you can check the value of **History list length** in the command output of **show engine innodb status**.

Solution

- If the ibdata1 file of the primary instance is too large but that of the standby instance is not, perform a primary/standby switchover.
- If the ibdata1 file of the standby instance is too large but that of the primary instance is not, **submit a service ticket** to rebuild the standby instance.
- If the ibdata1 files of both the primary instance and standby instance are large, use Data Replication Service (DRS) to migrate data.

5 SQL Issues

5.1 Double Quotation Marks Cannot Be Identified During SQL Statement Execution

Scenario

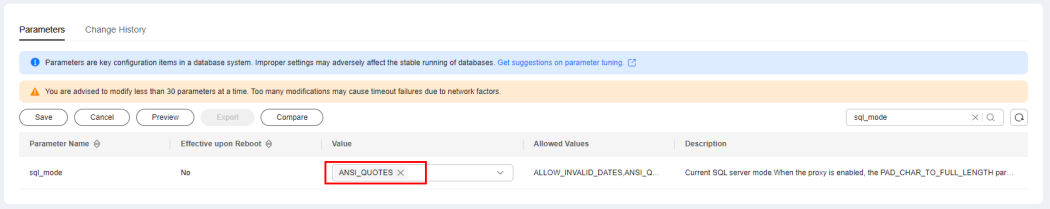
Double quotation marks cannot be identified when SQL statements are executed on an RDS for MySQL instance.

Troubleshooting

Check whether the **sql_mode** parameter is set to **ANSI_QUOTES**.

If it is set to **ANSI_QUOTES**, double quotation marks are considered as identifier quote characters. For details, see **Server SQL Modes**.

Figure 5-1 Checking the parameter value



Solution

Delete **ANSI_QUOTES** from the values of **sql_mode**.

5.2 Error 1366 Reported When Data Containing Emojis Is Updated

Scenario

Error 1366 was reported when data containing emojis was inserted or updated.

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

Possible Causes

The character set of the RDS for MySQL instance is incorrectly configured.

- An emoji is a special character and needs to be stored in a 4-byte character set.
- In this scenario, the MySQL character set is utf-8, which supports a maximum of three bytes. You need to change the character set to utf8mb4 to support four bytes.

Solution

1. Change the character set for the field that stores emojis to utf8mb4.

   If a large number of tables and fields are involved, you are advised to set the encoding format of the tables and databases to utf8mb4. Sample commands:

   ALTER DATABASE database_name CHARACTER SET= utf8mb4 COLLATE= utf8mb4_unicode_ci;

   ALTER TABLE table_name CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

   ALTER TABLE table_name MODIFY *{field name}* **VARCHAR(128) CHARSET utf8mb4 COLLATE utf8mb4_unicode_ci;**

2. If the character set for the field is already utf8mb4, set the character sets of the client and RDS for MySQL server to utf8mb4.

5.3 Failed to Change the varchar Length Due to the Index Length Limit

Scenario

The **alter table** command failed to modify a table structure. The following error information was displayed:

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



```
1 ALTER TABLE `uac_callback` MODIFY COLUMN `callback_url` varchar(1024);
```

Executed SQL Statements | Messages

```
---------------Execute---------------

[SQL statement split]: SQL statements to be executed: (1)

[Execute SQL statement: (1)]
ALTER TABLE `uac_callback` MODIFY COLUMN `callback_url` varchar(1024)
Failed. Cause: (conn=7264001) Specified key was too long; max key length is 3072 bytes
```

Possible Causes

- If **innodb_large_prefix** is set to **OFF**, the maximum length allowed for a single-column index in an InnoDB table is 767 bytes, while that for a composite index is 3,072 bytes, with each column in the composite index no more than 767 bytes.
- If **innodb_large_prefix** is set to **ON**, the allowed maximum length for a single-column index is 3,072 bytes, and that for a composite index is also 3,072 bytes.
- The index length is related to the character set. When the utf8 character set is used, a character occupies three bytes. If **innodb_large_prefix** is set to **ON**, the allowed maximum length for all columns in an index is 1,072 characters.

The table structure is as follows:

```
CREATE TABLE `xxxxx` (
.....
`subscription_type` varchar(64) NOT NULL DEFAULT 'DEVICE_EXCEPTION' COMMENT 'Subscription type',
`auth_key` varchar(255) DEFAULT'' COMMENT 'Signature. A token is added to the API request header based on the value of this parameter',
`create_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT 'Creation time',
`update_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT 'Update time',
PRIMARY KEY (`id`) USING BTREE,
UNIQUE KEY `enterprise_id` (`subscription_type`,`enterprise_id`,`callback_url`) USING BTREE)
) ENGINE=InnoDB AUTO_INCREMENT=1039 DEFAULT CHARSET=utf8 ROW_FORMAT=DYNAMIC
```

This table uses the utf8 character set. Each character occupies three bytes. The composite index **enterprise_id** contains the **callback_url** column. If a DDL operation is performed and **callback_url** to is changed to **varchar(1024)**, the maximum length of the composite index is exceeded. As a result, an error is reported.

Solution

Modify the index or column length.

5.4 Invalid TIMESTAMP Default Value during Table Creation

Scenario

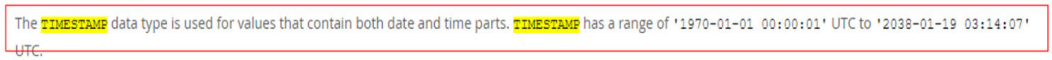
The CREATE TABLE statement failed to be executed, and the error message "ERROR 1067: Invalid default value for 'session_start'" was displayed.

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

Possible Causes

RDS for MySQL converts the value inserted to the TIMESTAMP column from the current time zone to the UTC time for storage. During query, it returns the value by converting the UTC time to the current time zone.

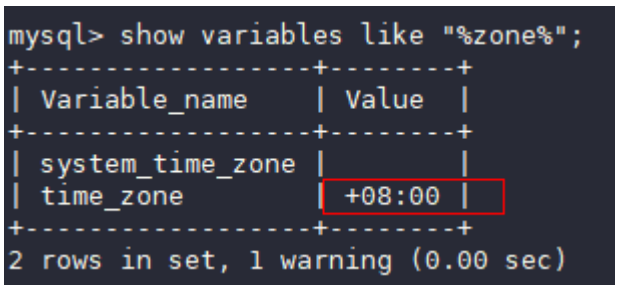
The time range for the TIMESTAMP column is from '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC. For details, see **The DATE, DATETIME, and TIMESTAMP Types**.



Run the following command to check the time zone:

show variables like "%zone%";

UTC+8 is the time zone, so the valid range for the default value starts with 1970-01-01 08:00:01.



Solution

Change the default value of the TIMESTAMP column.

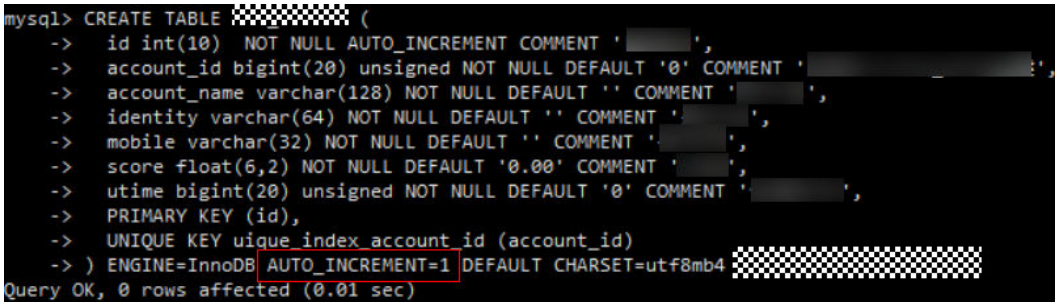
```
session_start TIMESTAMP DEFAULT '1970-01-01 08:00:01',
```


5.5 AUTO_INCREMENT Not Displayed in the Table Structure

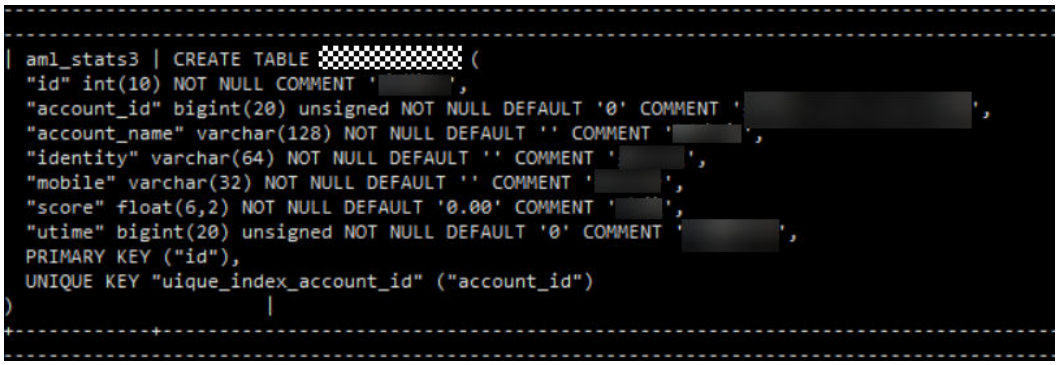
Scenario

When a table was created, **AUTO_INCREMENT** was set to **1**. After **show create table** was executed, **AUTO_INCREMENT** was not displayed in the table structure.

A table was created:

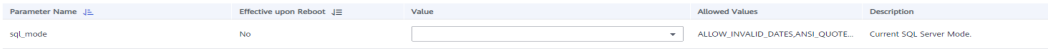


After **show create table** *table_name* was executed, **AUTO_INCREMENT** was not displayed in the table structure:



Possible Causes

sql_mode was set to **NO_FIELD_OPTIONS**.



Valid values for **sql_mode** are as follows:

- **NO_FIELD_OPTIONS**: Do not print MySQL-specific column options in the output of SHOW CREATE TABLE.
- **NO_KEY_OPTIONS**: Do not print MySQL-specific index options in the output of SHOW CREATE TABLE.
- **NO_TABLE_OPTIONS**: Do not print MySQL-specific table options (such as ENGINE) in the output of SHOW CREATE TABLE.

Solution

Change the value of **sql_mode**.

5.6 Slow Stored Procedure Execution Due to Inconsistent Collations

Scenario

It took more than a minute to process just a small amount of data using a stored procedure in an RDS for MySQL instance. Executing the SQL statement in the stored procedure was much faster.

Possible Causes

The collation of the stored procedure is inconsistent with that of the related table and database. As a result, a large number of characters need to be converted in the query result, and the execution is slow.

Troubleshooting:

Run the following commands to view the definitions of the stored procedure and related table and check whether the collations are the same:

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

Example:

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

The collation of the stored procedure is **utf8mb4_general_ci**, but the collation of the database is **utf8_general_ci** by default. The collations are inconsistent, which may cause performance issues.

Solution

Change the collation of the stored procedure to be the same as that of the related table and database.

5.7 ERROR [1412] Reported for a DB Instance

Scenario

When an SQL statement was executed on an RDS for MySQL instance, the following error message was displayed:

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


Possible Causes

After a transaction with consistent snapshot was started, another session was executing DDL statements. Procedure for reproducing the problem:

1. Session 1 starts a transaction with consistent snapshot.

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

2. Session 2 executes a DDL statement to modify the table structure.

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

3. Session 1 executes a common query statement.

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

You can also analyze binlogs or audit logs to check whether a DDL statement and transaction with consistent snapshot are executed concurrently on the same table.

Solution

Do not execute a DDL statement and transaction with consistent snapshot concurrently on the same table.

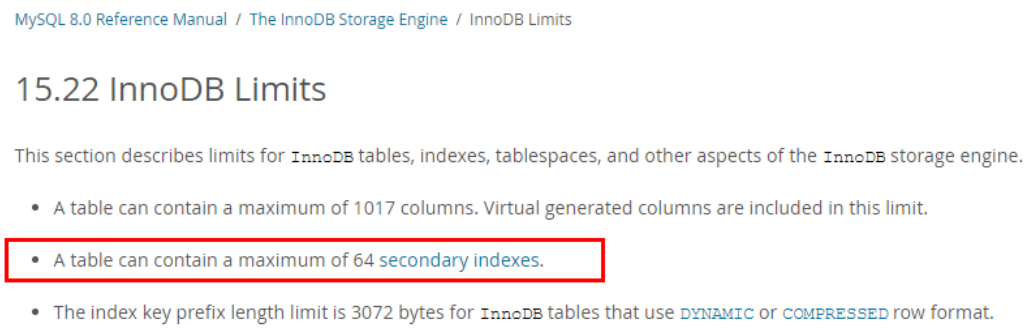
5.8 Error Message "Too many keys specified" Displayed When a Secondary Index Is Created

Scenario

A secondary index failed to be created, and the error message "Too many keys specified; max 64 keys allowed" was displayed.

Fault Analysis

RDS for MySQL limits the maximum number of secondary indexes in each InnoDB table to 64. If the number of secondary indexes exceeds 64, the error message "Too many keys specified; max 64 keys allowed" will be displayed. For details, see **InnoDB Limits**.


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

15.22 InnoDB Limits

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

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

Solution

Do not create too many indexes for a single table.

NOTE

Other restrictions on InnoDB tables include the following:

1. A table can contain a maximum of 1,017 columns (including virtual generated columns).
2. The index key prefix limit is 3,072 bytes for InnoDB tables that use the DYNAMIC or COMPRESSED row format.
3. A maximum of 16 columns is permitted for multicolumn indexes. Exceeding the limit returns an error.

5.9 Failed to Delete a Table with a Foreign Key

Scenario

When an RDS for MySQL table with a foreign key is deleted, the following error message will be displayed, which is irrelevant to user permissions:

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

Possible Causes

There is a foreign key relationship between this table and another table. A link is established between the data in the two tables. To prevent foreign key constraints from being violated, data in the tables cannot be updated or deleted.

You can set **FOREIGN_KEY_CHECKS** to **off** to remove the foreign key relationship. For details, see **FOREIGN KEY Constraints**.

Solution

Set **FOREIGN_KEY_CHECKS** to **off**.

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

5.10 DISTINCT and GROUP BY Optimization

Scenario

The execution of the DISTINCT or GROUP BY statement is slow.

Possible Causes

In most cases, DISTINCT can be converted into an equivalent GROUP BY statement. In RDS for MySQL, DISTINCT is mainly used to remove duplicate records from database tables and fetch only the unique records.

The DISTINCT statement groups data first, and then fetches a piece of data from each group and returns the data to the client. There are two scenarios for grouping data:

- All DISTINCT fields are included in the same index. In this scenario, RDS for MySQL directly uses the index to group data, obtains a piece of data from each group, and returns the data.
- Not all DISTINCT fields are included in the index. In this scenario, qualified data is written to a temporary table and grouped in the temporary table. Using temporary tables causes extra overhead, deteriorating the performance.

In conclusion, when using DISTINCT or GROUP BY, set an index that contains all dependent fields. The following is an optimization example:

- No suitable index is available, so temporary tables are used.

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

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

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

- A suitable index is found, and temporary tables are not required.

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



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

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

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

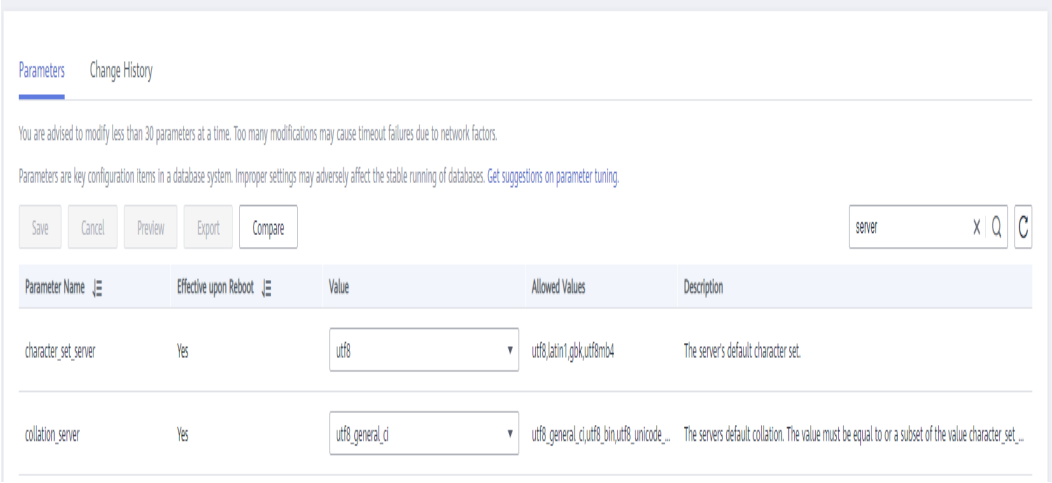
Solution

When using DISTINCT or GROUP BY, create an index that contains all dependent fields.

5.11 Character Set and Collation Settings

Related Variables

By default, **character_set_server** is set to **utf8** and **collation_server** to **utf8_general_ci** for your DB instance. You can change the values on the RDS console.



Configuring Character Sets and Collations for Databases, Tables, and Fields

- If the character set and collation are not explicitly specified for a database during database creation, the values of **character_set_server** and **collation_server** are used for the database. If the character set and collation are explicitly specified, the specified character set and collation are used for the database.
- If the character set and collation are not explicitly specified for a table during table creation, the character set and collation of the database hosting the table are used for the table. If the character set and collation are explicitly specified, the specified character set and collation are used for the table.
- If the character set and collation are not explicitly specified for a field during table creation, the character set and collation of the table hosting the field are used for the field. If the character set and collation are explicitly specified, the specified character set and collation are used for the field.

Example 1: Create a database and table without explicitly specifying the character set and collation.

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

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

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

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



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

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

Example 2: Create a database with the character set and collation explicitly specified.

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

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

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

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

Example 3: Create a table with the character set and collation explicitly specified for the table.

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

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

Example 4: Create a table with the character set and collation explicitly specified for a field.

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

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

5.12 An Error Message Is Displayed When a User Is Created for a DB Instance

Scenario

A user account disappeared from the console, but the account and its password could still be used to connect to the instance.

When a new account with the same name as the missing account was created, the following error was displayed:

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

Possible Causes

1. The account was deleted from the **mysql.user** table and therefore was not displayed on the console.
2. Because the account and its password could still be used to log in to the instance, the account was deleted using **delete from mysql.user**. If you use **delete from mysql.user** to delete an account, you also need to run the **flush privileges** command to delete related data from the memory. Then, the account can no longer log in to the instance.
3. The reason why a new account with the same name as the missing account could not be created is that there was still related data about the disappeared account in the memory.

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

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

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

The correct way to delete an account is using the **drop user** statement. When running **drop user**, note that:

- **drop user** can be used to delete one or more users and revoke their permissions.
- **drop user** requires the DELETE permission or the global CREATE USER permission on the RDS for MySQL instance.
- If the host name of the account is not specified in the **drop user** statement, the host name **%** is used by default.

Troubleshooting example:

After an account is created, the **delete** statement is used to delete the account. When a new account with the same name as the deleted account is created, error 1396 is reported. After the **flush privileges** command is executed, an account with the same name can be created.

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

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

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


Solution

- Method 1 (recommended): During off-peak hours, run the **drop user user_name** command as the administrator to delete the disappeared account and then create an account with the same name.
- Method 2: During off-peak hours, run the **flush privileges** command as the administrator and then create an account with the same name. You are advised to enable SQL Audit to locate which client deletes the user.

5.13 Slow SQL Queries After a Large Amount of Data Is Deleted from a Large Table

Scenario

After multiple wide columns of data (records about 1 GB long) are deleted at the same time, performing an INSERT, DELETE, UPDATE, or SELECT operation on the same table again takes a long time. After about 20 minutes, the problem is resolved.

Problem Reproduction

1. Suppose, for example, that the value of **max_allowed_packet** is **1073741824**.
2. Create a table.
```
CREATE TABLE IF NOT EXISTS zstest1
(
id int  PRIMARY KEY not null,
c_longtext LONGTEXT
);
```
3. Insert data to the table.
```
insert into zstest1 values(1, repeat('a', 1073741800));
insert into zstest1 values(2, repeat('a', 1073741800));
insert into zstest1 values(3, repeat('a', 1073741800));
insert into zstest1 values(4, repeat('a', 1073741800));
insert into zstest1 values(5, repeat('a', 1073741800));
insert into zstest1 values(6, repeat('a', 1073741800));
insert into zstest1 values(7, repeat('a', 1073741800));
insert into zstest1 values(8, repeat('a', 1073741800));
insert into zstest1 values(9, repeat('a', 1073741800));
insert into zstest1 values(10, repeat('a', 1073741800));
```
4. Delete data from the table.
```
delete from zstest1;
```
5. Execute a query.
```
select id from zstest1;    //The execution is slow.
```

Possible Causes

After the DELETE operation is performed, the background purge thread clears all records marked with a delete mark. Due to the large amount of data to be deleted, the purge thread obtains the SX lock of the index root node where the page is located when traversing and releasing the page. As a result, the SELECT statement cannot obtain the RW lock of the root page and keeps waiting.

Solution

- This phenomenon is normal. After the purge operation is complete, the issue will resolve itself.
- Scale up the instance specifications to improve the purge efficiency.
- Do not delete a large amount of data at the same time. To delete all data from a table, use the **truncate table** statement.

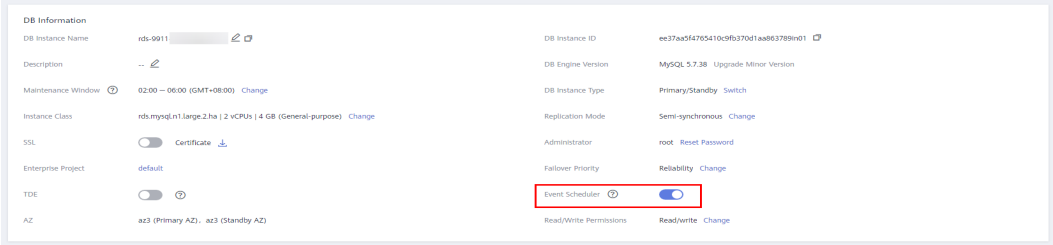
5.14 Event Scheduler Not Taking Effect Immediately After Being Enabled

Scenario

The event scheduler did not take effect immediately after being enabled.

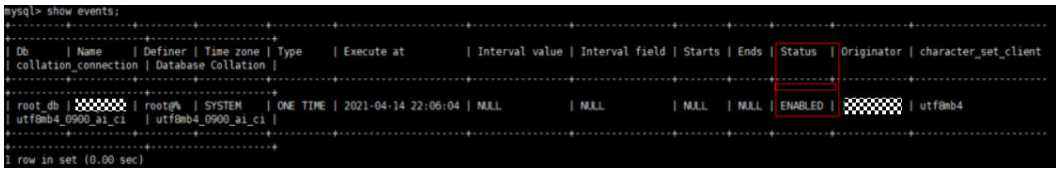
Troubleshooting

1. On the **Overview** page of the DB instance, check whether **Event Scheduler** is enabled.



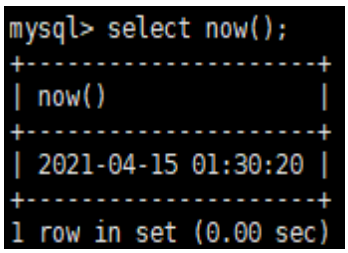
2. Check whether the scheduled event is in the **ENABLED** state.

   show events;



3. Check the time zone configured for the instance.

   show variables like "%time_zone%";




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

In the preceding figure, the instance uses UTC time, so the event scheduler, which was configured based on Beijing time, does not take effect immediately.

Solution

Configure the event scheduler based on the time zone of your instance.

5.15 Equivalent Comparison Failures with Floating-Point Numbers

Possible Causes

Equivalent comparison of floating-point numbers is a common problem. In computing, floating-point numbers are stored as approximate values instead of exact values. Therefore, unexpected problems may occur during equivalent comparison and mathematical operations.

In RDS for MySQL, FLOAT and DOUBLE are floating-point types. The following figure shows an example for equivalent comparison using floating-point numbers:


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

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

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

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

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

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

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

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

Solution

1. Decide on an acceptable tolerance for differences between the field and the value and then do the comparison against the tolerance value. Example:

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

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

2. Use the fixed-point number type (DECIMAL) to replace the floating-point number type. Example:


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

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

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

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

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

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

5.16 A Large Number of SELECT Requests Routed to The Primary Instance After Database Proxy Is Enabled

Possible Causes

1. Delay threshold parameter

   This parameter specifies the maximum delay for data to be synchronized from the primary instance to read replicas. It is only applied when there are read replicas. To prevent long-time data inconsistency between the primary instance and read replicas, when the delay of a read replica exceeds the preset threshold, read requests are not forwarded to the read replica regardless of the read weight distributed to it.

   For more information, see **Configuring Delay Threshold and Distributing Read Weight**.

2. Read weight parameter

   This parameter specifies read weights distributed to the primary instance and read replicas. It takes effect only when there are read replicas.

   For example, if a primary instance has two read replicas and the read weights are set to 1, 2, and 3 for the primary instance and two read replicas, respectively, read requests are distributed to the primary instance and read replicas based on the ratio of 1:2:3. If the read weights are set to 0, 2, and 3,

respectively, read requests are distributed to only the read replicas based on the ratio of 2:3.

For more information, see **Configuring Delay Threshold and Distributing Read Weight**.

3. Transactions

   SQL statements in a transaction are sent to the primary instance. If **set autocommit=0** is configured before a query statement is executed, the query statement is routed to the primary instance as a transaction.

4. Connection binding

   If multi-statements (for example, **insert xxx;select xxx**) are executed, all subsequent requests will be routed to the primary instance because the SQL statement for creating temporary tables binds the connection to the primary instance. To restore read/write splitting, disconnect your application from the RDS instance and connect to the instance again.

5. Custom variables

   SQL statements containing custom variables will be routed to the primary instance.

6. Read operations with locks (for example, **SELECT for UPDATE**) will be routed to the primary instance.

7. Using hints to specify whether an SQL statement is routed to the primary instance or read replica

   In addition to weight distribution rules, you can add one of the following hints before an SQL statement for forcible routing:

   /*FORCE_MASTER*/: The SQL statement is forcibly routed to the primary instance.

   /*FORCE_SLAVE*/: The SQL statement is forcibly routed to a read replica.

   Hints are only used as routing suggestions. In non-read-only SQL and transaction scenarios, SQL statements cannot be routed to read replicas.

5.17 RENAME USER Execution Failure

Scenario

The execution of the RENAME USER statement failed.

The failure may occur in MySQL-5.6.41.5.

Fault Analysis

The username does not exist but the memory indicates that it actually does.

Solution

Perform the following statements:

```
drop user 'xxx'@'%';
flush privileges;
```


5.18 ERROR[1451] Reported When a Table with Foreign Keys Cannot Be Deleted

Scenario

The **root** user does not have the permissions required to delete or modify tables in the database. Error message is as follows:

ERROR[1451] -Cannot deleteorupdatea parent row:

aforeignkeyconstraintfails (...)

Fault Analysis

The FRM file also exists in sys_tables. This table has foreign key relationships with other tables and cannot be deleted directly.

A foreign key association has been configured in the RDS for MySQL DB instance. As a result, data cannot be updated or deleted. You can use the **foreign_key_checks** parameter to avoid this problem.

Solution

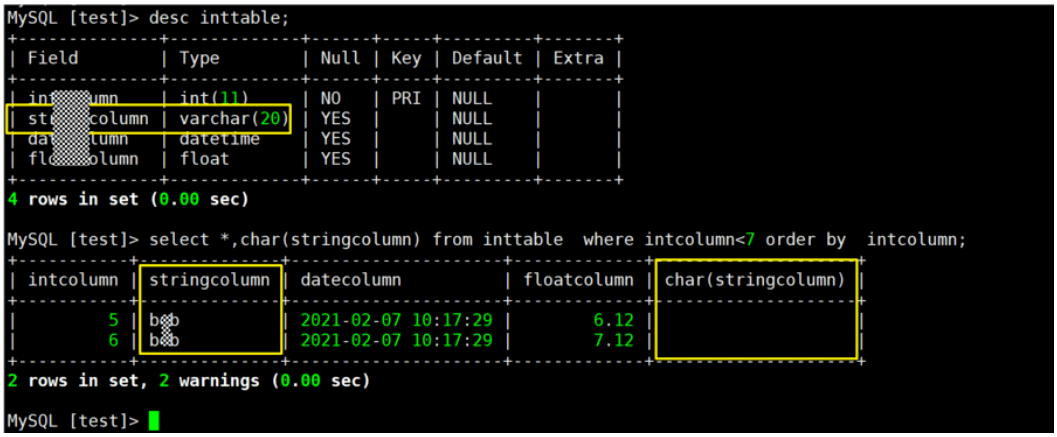
```
set session foreign_key_checks=off;
drop table table_name;
```

To delete the table, set **foreign_key_checks** to **off**.

5.19 Solution to the Failure of Converting the Field Type

Scenario

The **varchar** field is read using the char data type, and cannot be converted through the char() function.



Fault Analysis

The char() function cannot be used to convert data types.

Solution

The CAST() function and CONVERT() function of RDS for MySQL can be used to obtain values of one type and generate values of another type. The syntax is as follows:

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

That is, CAST(xxx AS type) and CONVERT(xxx, type).

NOTE

The data types that can be converted are limited. The supported data types are as follows:

- BINARY: the same as adding a binary prefix to a data string
- CHAR(): characters. You can specify the length of the characters.
- DATE: calendar date
- Time: time of day
- DATETIME: date and time
- DECIMAL: floating point number
- SIGNED: integer
- UNSIGNED: unsigned integer

5.20 "Row size too large" Reported When an RDS for MySQL Table Failed to Be Created

Scenario

An RDS for MySQL table failed to be created and the following information is displayed:

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

Fault Analysis

The total length of the **varchar** fields exceeds 65535, resulting in a table creation failure.

Solution

1. Reduce the length.
   ```
   CREATE TABLE t1 (a VARCHAR(10000),b VARCHAR(10000),c VARCHAR(10000),d VARCHAR(10000),e VARCHAR(10000),f VARCHAR(10000) ) ENGINE=MyISAM  CHARACTER SET latin1;
   ```
2. Change a column to **TEXT** by referring to the **official documentation**.

5.21 ERROR [1412] Reported by an RDS for MySQL DB Instance

Scenario

The following error is displayed:

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

This problem may occur in MySQL-5.7.31.2.

Fault Analysis

Cause 1: A transaction is started using START TRANSACTION WITH CONSISTENT SNAPSHOT.

Scenario 1

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

Scenario 2

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

Scenario 3

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

Cause 2: DDL operations are performed for binlog files.

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

Solution

If the error is caused by any of the preceding causes, the fault needs to be rectified on the service side.

5.22 Instance Reboot Failure or ERROR 1146: Table 'xxx' doesn't exist Reported During Table Operations

Scenario

- After an RDS for MySQL DB instance is rebooted, the following error message is displayed when you perform any table operation:
  ```
  ERROR 1146: Table `xxx` doesn't exist
  ```
  In addition, the following information is displayed in the error log:
  ```
  [Warning] InnoDB: Load table `xxx` failed, the table has missing foreign key indexes. Turn off 'foreign_key_checks' and try again.
  [Warning] InnoDB: Cannot open table 'xxx' from the internal data dictionary of InnoDB through the .frm file for the table exists.
  ```
- A DB instance fails to be rebooted due to incorrect foreign key usage and the following error information is displayed in the error log:
  ```
  [Warning] InnoDB: Load table `xxx` failed, the table has missing foreign key indexes. Turn off 'foreign_key_checks' and try again.
  [Warning] InnoDB: Cannot open table xxx/xxx from the internal data dictionary of InnoDB through the .frm file for the table exists.
  ```

Fault Analysis

The possible cause is that the foreign key added does not meet the corresponding conditions and constraints. For details about the foreign key usage rules, see **FOREIGN KEY Constraints**.

RDS for MySQL uses variable **foreign_key_checks** (default value: **ON**) to control foreign key constraint check. When **foreign_key_checks** is set to **OFF**, the foreign key constraint check does not take effect. In this case, no error will be reported even if you add an incorrect foreign key. After the DB instance is rebooted, **foreign_key_checks** is enabled by default. InnoDB checks foreign key constraints when opening a table, thus generating an error.

The common reasons are as follows:

- The character set of the foreign key-related column in the parent or child table has been changed.

  RDS for MySQL 5.6, 5.7, and 8.0 allow you to modify the character set of the foreign key-related column in the parent or child table when **foreign_key_checks** is set to **OFF**. After the DB instance is rebooted:
  - RDS for MySQL 5.6 and 5.7: A warning will be displayed in the error log and the parent table cannot be used.
  - RDS for MySQL 8.0: No warning will be displayed in the error log and the parent table can be used.
- The index of the foreign key-related column in the parent or child table has been deleted.
  - RDS for MySQL 5.7 and 8.0: The index of the foreign key-related column in the parent or child table cannot be deleted when **foreign_key_checks** is set to **OFF**.
  - RDS for MySQL 5.6: The index of the foreign key-related column in the parent or child table can be deleted when **foreign_key_checks** is set to

OFF. After the index is deleted, a warning will be displayed when the DB instance is rebooted. In addition, the table whose index was deleted cannot be used.

Solution

- If the problem is caused by character set modification, set **foreign_key_checks** to **OFF** and change the character sets of the foreign key-related columns in the parent and child tables to be the same.
- If the problem is caused by index deletion, set **foreign_key_checks** to **OFF** and recreate an index.

5.23 Error Reported During Pagination Query

Scenario

The following error is reported during pagination query of an RDS for MySQL instance: ERROR 1038 (HY001): Out of sort memory, consider increasing server sort buffer size.

Solution

Set the **sort_buffer_size** parameter to a value greater than the default value **256 KB**.

sort_buffer_size is a MySQL server system variable that may affect your query performance. It is defined on a per-session basis and affects MySQL memory consumption.

5.24 Error Reported During User Creation

Scenario

The **saas_cash_user** user fails to be created after the following SQL statement is executed:

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

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

Possible Causes

If a user is deleted using the **delete** command, an error is reported when the user is created again.

Generally, you can use the **create user** or **grant** statement to create a user. A user created using the **create** syntax does not have any permission, and you need to use the **grant** syntax to assign permissions. A user created using the **grant** syntax has the assigned permissions.

When you use the **drop user** statement to delete a user, the database table and permission table are deleted together. However, **delete from mysql.user** only

deletes records in the user table. If you run **show grants for username**, you can find that the permissions of the user still exist. In this case, creating a same user can fail due to a verification failure.

Solution

To delete a user, run the **drop** command.

5.25 Syntax Error Reported When GRANT Is Used to Grant All Privileges

Scenario

The **grant** statement can be used to grant all privileges on a database whose name is in letters. If this statement is used to grant all privileges on a database whose name is in digits, the following error is reported: You have an error in your SOL syntax.

Database name in letters:

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

Database name in digits:

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

Possible Causes

This problem is caused by a syntax error. In MySQL, backquotes (`) are introduced to distinguish MySQL keywords from common characters. You can enter a backquote (`) to ensure correct syntax.

Solution

If the database name is in digits, add a backquote (`) before and after the database name.

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

5.26 Error Reported During Table Creation for an RDS for MySQL 5.6 DB Instance

Scenario

An error is reported when a table creation statement is executed on an RDS for MySQL 5.6 DB instance.

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


Possible Causes

When you create tables for an RDS for MySQL 5.6 instance, use indexes based on the following rules:

- For a single-field index, the field length cannot exceed 767 bytes.
- For a composite index, the length of each field cannot exceed 767 bytes, and the total length of all fields cannot exceed 3,072 bytes.
- Use the 4-byte character set **utf8mb4** for table creation.

If the maximum length of an index is 767 bytes, the length allowed for a varchar field is 191.75 bytes (767/4=191.75). If the length of the varchar field in the table creation statement exceeds this value, an error is reported.

Solution

1. Ensure that the value of **innodb_large_prefix** is **ON**.
2. Create a table by referring to the rules in **Possible Causes**.

5.27 Inconsistent Data Obtained on the Primary and Standby Nodes When a Query Is Performed Using an Auto-Increment Primary Key Value

Scenario

When an auto-increment primary key value is used to query data on the primary and standby nodes, data inconsistency occurs in the query results.

Possible Causes

For a table without a primary key, the order of data in the table is determined by the ROWID of the storage engine. The ROWIDs may be different on the primary and standby nodes, so the orders of data on the primary and standby nodes may be different. When an auto-increment primary key is added to the table, the values of the auto-increment primary key are initialized based on the data order in the table. As a result, the auto-increment primary key values are different for the same data, that is, the data queried on the primary and standby nodes by using the same auto-increment primary key value is different. For details, see **Replication and AUTO_INCREMENT**.

Solution

To add an auto-increment column to a table containing data, create a new table with the same table structure, add the auto-increment column to the new table, and then import data from the original table to the new table. (When importing data, ensure that no write operation is being performed on the original table, to prevent data inconsistency between the two tables.)

The detailed procedure is as follows:

Step 1 On the primary node, create a new table named **t2** that is the same as the table without a primary key (original table **t1**), and add an auto-increment primary key to the new table.

Example:

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

Step 2 Insert all data of the original table **t1** to the new table **t2**.

Example:

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

NOTE

To ensure that the orders of data in the corresponding tables on the primary and standby nodes are the same, the ORDER BY clause must contain all columns of **t1**.

Step 3 Delete **t1** and rename **t2** as **t1**.

Example:

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

----End

5.28 "Data too long for column" Displayed When Data Is Inserted into an RDS for MySQL Instance

Scenario

The following message is displayed when data is inserted into an RDS for MySQL instance through JDBC:

```
Data truncation: Data too long for column 'field_name'
```

Troubleshooting

Check whether the **sql_mode** parameter is set to **STRICT_TRANS_TABLES** at the session level.

STRICT_TRANS_TABLES indicates the strict mode. If **sql_mode** is set to **STRICT_TRANS_TABLES**, errors about long fields can be reported.

Solution

- Increase the allowed field length during off-peak hours.

  ALTER TABLE *table_name* **MODIFY COLUMN** *field_name* **VARCHAR(128);**
- Insert data through Data Admin Service (DAS). If there is overlong data, DAS truncates the data.

6 Connection Issues

6.1 "Access denied" Displayed During Database Connection

Scenario

A client failed to connect to a database, and the error message "Error 1045: Access denied for user *username*" was displayed.

Handling Methods

1. An invalid host is connected.

   Cause: An invalid database host is connected, and the user or client IP address does not have the access permission.

   Solution: Ensure that the host name of the database to be connected is correctly specified.

2. The user does not exist.

   Cause: The user account used for connecting to the database does not exist.

   Solution:

   - Log in to the database as an administrator and run the following command to check whether the user exists:
     ```
     SELECT User FROM mysql.user WHERE User='username';
     ```
   - If the user does not exist, create the user.
     ```
     CREATE USER 'xxx'@'xxxxxx' IDENTIFIED BY 'xxxx';
     ```

3. The client IP address does not have the access permission.

   Cause: The user used by the client exists, but the client IP address is not allowed to access the database.

   Solution:

   - Log in to the database as an administrator and run the following command to check which client IP addresses are allowed to connect to the database for the user:
     ```
     SELECT Host, User FROM mysql.user WHERE User='username';
     ```


- If the client IP address is not within the allowed network segment, assign the access permission to the client IP address. For example, run the following command to grant the **test** user the permission to access the **192.168.0** network segment:

```
GRANT ALL PRIVILEGES ON *.* TO'root'@'192.168.0.%' IDENTIFIED BY 'password' WITH GRANT OPTION;
FLUSH PRIVILEGES;
```

4. The password is incorrect.

   Cause: The password of the user is incorrect.

   Solution:

   - Check whether the password is correct. Because the password is used for identity authentication, the user password cannot be read from RDS for MySQL in plain text. However, you can compare the hash string with the **PASSWORD** function value of the password to check whether the password is correct. The following is an example of SQL statements:

```
mysql> SELECT Host, User, authentication_string, PASSWORD('12345') FROM mysql.user WHERE User='test';
+-----------+------+-------------------------------------------+-------------------------------------------+
| Host      | User | authentication_string                     | PASSWORD('12345')                         |
+-----------+------+-------------------------------------------+-------------------------------------------+
| %         | test | *6A23DC5E7446019DC9C1778554ED87BE6BA61041 | *00A51F3F48415C7D4E8908980D443C29C69B60C9 |
+-----------+------+-------------------------------------------+-------------------------------------------+
2 rows in set, 1 warning (0.00 sec)
```

     The preceding example shows that the hash value of **PASSWORD('12345')** does not match the **authentication_string** field, indicating that the password **12345** is incorrect.

   - To reset the user password, run the following SQL statement:

```
set password for 'test'@'%' = 'new_password';
```

5. The password contains special characters, which are treated as escape characters by Bash.

   Cause: In the default Linux Bash shell, when connecting to a database, special characters in the password will be interpreted as escape characters. As a result, the password becomes invalid.

   For example, in a Bash shell, if the password **test$123** is used, when you run the **mysql -hxxx -u test -ptest$123** command to connect to a database, the error "ERROR 1045 (28000): Access denied" will be displayed.

   Solution: Enclose the password in single quotation marks to prevent Bash from interpreting special characters.

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

6. **REQUIRE SSL** is configured for the user, but the client uses a non-SSL connection.

   Troubleshooting:

   - Run the **show create user 'xxx'** command to check whether the user must use the SSL connection. If the **REQUIRE SSL** attribute is displayed, the user must use the SSL connection.
   - Check whether statements similar to the following have been used to grant permissions to the user:

```
GRANT ALL PRIVILEGES ON . TO 'ssluser'@'localhost' IDENTIFIED BY 'zdh1234' REQUIRE SSL;
```

   - Check the **ssl_type** value of the user. If the value is not empty, the user must use SSL.

```
SELECT User, Host, ssl_type FROM mysql.user WHERE User='xxx';
```


Solution:

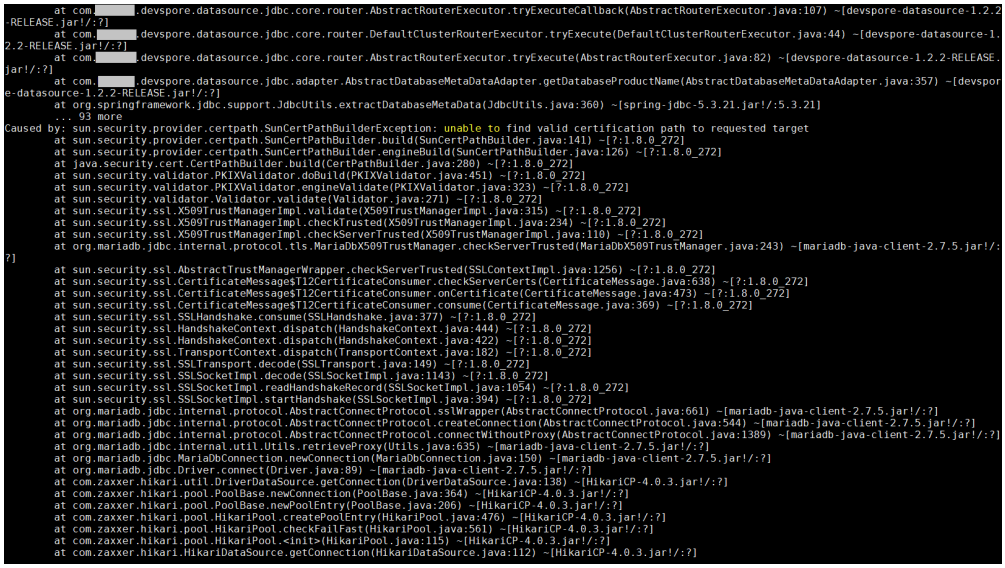
- Connect the client to the database in SSL mode. For details, see **Using MySQL CLI to Connect to an Instance Through a Private Network**.
- Run the **ALTER USER 'test'@'xxxxx' REQUIRE NONE;** command to remove the SSL permission from the user.

6.2 Failed to Connect to a Database Using mariadb-connector in SSL Mode

Scenario

A database could not be connected using JDBC, and the following error message was displayed:

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



Possible Causes

In the figure above, a MariaDB JAR package is used to connect to the database, which is slightly different from the official driver package of MySQL.

Solution

The connection string for mariadb-java-client-2.7.5 is as follows:

- If the CA certificate is not provided and the certificate is not verified:
  ```
  String url = "jdbc:mysql://ip:port/mysql?useSsl=true&trustServerCertificate=true";
  ```
- If the CA certificate is provided and the certificate is verified:
  ```
  String url = "jdbc:mysql://ip:port/mysql?useSsl=true&serverSslCert=D:\
  \ca.pem&disableSslHostnameVerification=true";
  ```

Note: RDS for MySQL DB instances do not support hostname verification, so you need to set **disableSslHostnameVerification** to **true**. The way you set this parameter depends on the MariaDB JAR package version. For details, see the **notes on usage** of the corresponding version.

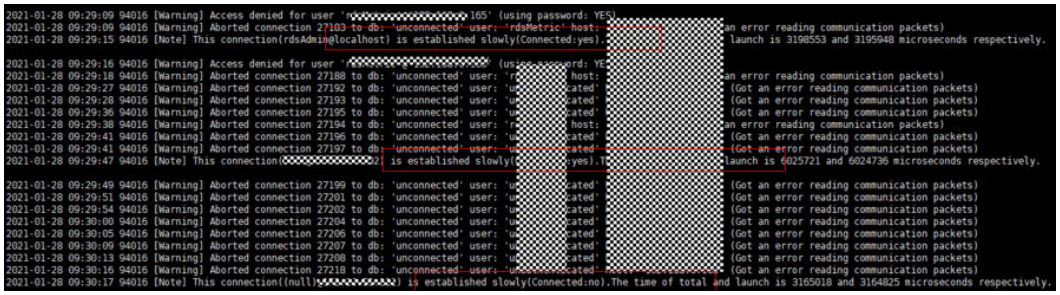
6.3 Error Message "connection established slowly"

Scenario

During peak hours, the connection between a client and an RDS for MySQL instance often times out. As a result, it takes more than 10 seconds to log in to the instance.

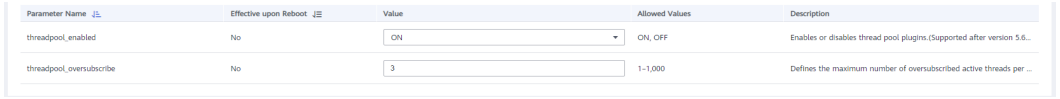
Possible Causes

1. **View error logs of the instance** to check whether the information "connection *xxx* is established slowly" is displayed. Example:



   If yes, some connections have timed out and have not been processed by the RDS for MySQL instance yet. When the connection between a client and the instance exceeds the specified timeout duration, an error is reported.

2. Check the thread pool configuration (enabled by default) on the console.



   In the preceding figure, **threadpool_oversubscribe** is set to **3**. The wait time for the thread pool to process connections is related to this parameter.

Solution

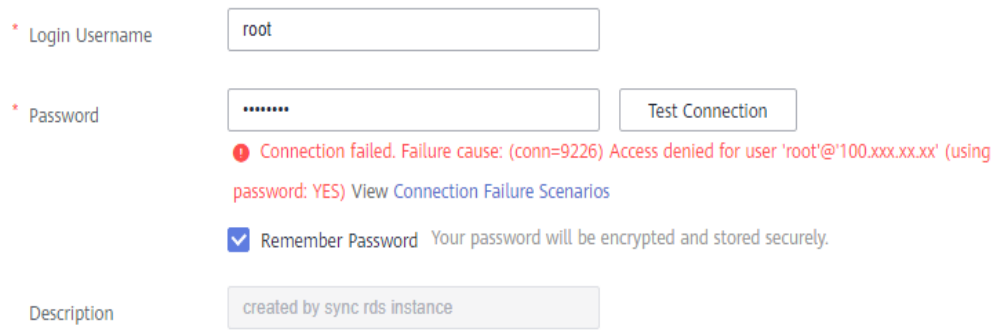
If there are a large number of new connections, increase the value of **threadpool_oversubscribe** to increase the total number of threads. This reduces the overhead of repeated thread creation and destruction, and limits the number of running threads to protect the system against avalanche.

In normal cases, the thread pool is used when there are a large number of short connections. If persistent connections are used and there are a few connections (for example, the client uses a connection pool), the thread pool is not so helpful. In this case, adjust the value of **threadpool_oversubscribe** to increase the total number of threads, or close the thread pool.

6.4 Login Failed After ssl_type of root Is Changed to ANY

Scenario

When user **root** was used to log in to a DB instance through DAS on the console, the error message "Access denied" was displayed.



Troubleshooting

1. View the **root** account information in the **mysql.user** table to check whether the client IP address is within the allowed range and whether SSL is enabled.

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

   If **ssl_type** of the **root** account is set to **ANY**, the **root** account needs to use SSL.

2. Check whether SSL is enabled.

   show variables like '%ssl%';

   SSL was not enabled for the instance.

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

   The reason is that **ssl_type** of the **root** account was changed to **ANY**. As a result, the login failed.

Solution

Run the following command to change the value of **ssl_type** to be empty for the **root** account:

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

To change the **ssl_type** of all other user accounts to be empty, run the following command:

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

6.5 Error Reported During Login to a DB Instance Through DAS

Scenario

The following error is reported when user **root** is used to log in to an RDS for MySQL instance on the DAS console: Client does not support authentication protocol requested by server. plugin type was = 'sha256_password'.

Possible Causes

DAS does not support login of database accounts using sha256_password for password encryption.

Solution

Run the following statement to change the password encryption mode to mysql_native_password:

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

6.6 "Your password does not satisfy the current policy requirements" Displayed When Permissions Are Granted or Revoked on DAS

Scenario

The error message "Your password does not satisfy the current policy requirements" is displayed when the **root** user is used to log in to an instance and run **GRANT** or **REVOKE** on Data Admin Service (DAS).

Solution

- MySQL 5.7

  If permissions are granted to a user that does not exist, the message "Your password does not satisfy the current policy requirements" will be displayed.

  In MySQL 5.7, the **GRANT** statement can automatically create the user when a user does not exist. You are advised to add **IDENTIFIED BY** to the end of the statement so that the user can be created successfully.

  Check whether a user exists.

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

GRANT statement:

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

- MySQL 8.0

  If permissions are granted to a user that does not exist, the message "You are not allowed to create a user with GRANT" will be displayed.

  In MySQL 8.0, you need to create a user using **create** before granting permissions to it.

6.7 SSL Connection Failed Due to Inconsistent TLS Versions

Scenario

A client failed to connect to an RDS for MySQL instance using SSL, but could connect to a self-built MySQL database using SSL.

Possible Causes

Troubleshooting:

1. **View error logs of the instance**. The following error was displayed:

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

2. Based on the **unsupported protocol** in the error, we know that the problem may be related to the TLS version. Run the following command to check the TLS versions of the RDS for MySQL instance and self-built MySQL database:

   show variables like '%tls_version%';

   It was found that the RDS for MySQL instance used TLS v1.2 and the self-built MySQL database used TLS v1.1. The TLS version of the client was the same as that of the self-built MySQL database. The self-built MySQL database was able to connect, but the RDS for MySQL instance could not.

Solution

Upgrade the TLS version of the client to TLS v1.2.

If the official JDBC driver **mysql-connector/J** is used, see **Connecting Securely Using SSL** for the configuration method.

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

6.8 Failed to Connect to a Database as root

Scenario

A database could not be connected to using the **root** account.

Troubleshooting

1. Check the kernel error.log to for any records of a connection denial.
2. Use another account to log in to the database and check the **root** permissions. There are two **root** accounts. One of them is allowed to access only hosts whose IP addresses start with 192.

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

```
     password_lifetime: NULL
        account_locked: N
*************************** 2. row ***************************
                Host: 192.%
                User: root
           Select_priv: Y
           Insert_priv: Y
           Update_priv: Y
           Delete_priv: Y
           Create_priv: Y
             Drop_priv: Y
           Reload_priv: Y
```

Solution

Contact technical support to delete the extra **root** account.

6.9 RDS for MySQL Client Automatically Disconnected from a DB Instance

Description

The RDS for MySQL client was automatically disconnected from the DB instance. The following error is displayed: ERROR 2013: Lost connection to MySQL server during query.

Solution

ERROR 2013 is usually caused by inappropriate settings.

- **wait_timeout**: indicates the number of seconds the server waits for activity on a non-interactive connection before closing it.
- **interactive_timeout**: indicates the number of seconds the server waits for activity on an interactive connection before closing it.

Step 1 Check whether the DB instance is available.

If the DB instance is available, check for other possible causes.

Step 2 View error logs.

Step 3 Use the RDS for MySQL command-line client to connect to the database. Run **status** to check whether the DB instance has been rebooted a lot.

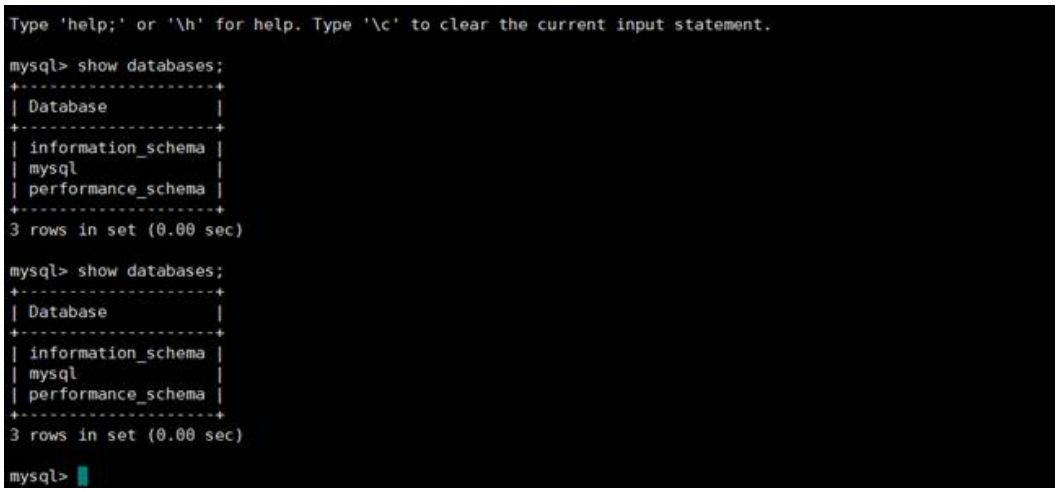
```
mysql> status
--------------
mysql  Ver 14.14 Distrib 5.6.34, for Linux (x86_64) using  EditLine wrapper

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

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

Uptime indicates the running time of the DB instance. The command output shows that the database has not been restarted frequently. Therefore, the client disconnection is not caused by a database restart.

Step 4 Check parameters. If the values of **wait_timeout** and **interactive_timeout** are too small, the RDS for MySQL client automatically stops connections as they time out.

Step 5 You can change the values of **wait_timeout** and **interactive_timeout** based on service requirements, and there is no need to reboot the DB instance.

Step 6 After about 10 minutes, run the **show databases** command to check whether the connection is normal.



If information similar to the preceding figure is returned, the connection is normal.

----End

6.10 RDS for MySQL DB Instance Inaccessible

Scenario

When the MySQL client attempted to connect to a database, the following error information is displayed:

- Fault 1
  ERROR 1045 (28000): Access denied for user 'root'@'192.168.0.30' (using password:YES)
- Fault 2
  ERROR 1226 (42000): User 'test' has exceeded the 'max_user_connections' resource (current value:10)
- Fault 3
  ERROR 1129 (HY000): Host '192.168.0.111' is blocked because of many connection errors; unblock with 'mysqladmin flush-hosts'

Fault 1

Step 1 Check whether the password of user **root** is correct.

ERROR 1045 (28000) is usually caused by an incorrect password so you need to check the password first.

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



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

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

An incorrect password will cause a login failure as shown in the following figure.

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

Step 2 Check whether the ECS or device has the permission to connect to the DB instance.

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

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

If you want to log in to the database from another ECS or device, log in to the database as user **root** and grant the permission to the user.

For example, if the IP address of the ECS or device is 192.168.0.76, run the following command:

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

flush privileges;


```
[root@ecs-lwt-0921 ~]# mysql -uroot -ptest1i@123  -P 8635 -h192.168.0.73
Warning: Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6961
Server version: 5.6.35-log MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

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

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

mysql>
```

Step 3 Check whether the RDS for MySQL client can ping the virtual IP address of the DB instance and use telnet to check the connection to the port.

If the ping and telnet operations are successful, ERROR 1045 (28000) is not caused by the virtual IP address or port of the DB instance.

Step 4 Check the security group and rules of the DB instance.

Step 5 Query the user table and check user information.

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

The command output indicates that there are two **root** users.

When the client is in the network segment 192.168.***.***, RDS for MySQL authenticates the user **root@'192.168.%'**. However, if the password is **root@'%'**, there will be a login failure. ERROR 1045 (28000) is caused by an invalid password.

NOTE

User **root@'%'** is used for setting a password when you create a DB instance on the RDS console.

----End

Fault 2

Step 1 Check whether the **max_user_connections** option is set when you create an RDS for MySQL user, which limits the number of connections.

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



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

The command output indicates that the connection failed because the **max_user_connections** option was used.

Step 2 Increase the maximum number of connections for the user.

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

Step 3 Query the modification result and check whether the database can be connected to.

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

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

----End

Fault 3

Step 1 Check whether the number of failed connection attempts (not caused by incorrect passwords) of the RDS for MySQL client exceeds the value of **max_connection_errors**.

Step 2 Log in to the MySQL database as user **root** and run **flush hosts**.

Alternatively, run the following command:

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

Step 3 Use the MySQL client to connect to the database again.

----End

6.11 Login Failed After the authentication_string Field Is Changed to Display the Password for RDS for MySQL

Scenario

After you use Navicat to modify the field **authentication_string** of the root account in the user table to display the password, the user cannot log in to the client.

This problem may occur in MySQL-8.0.20.6.

Possible Causes

The way the password was changed was incorrect. The hash key of the **authentication_string** field in the user table should not be changed directly. The password of the **root** user should be reset on the console.

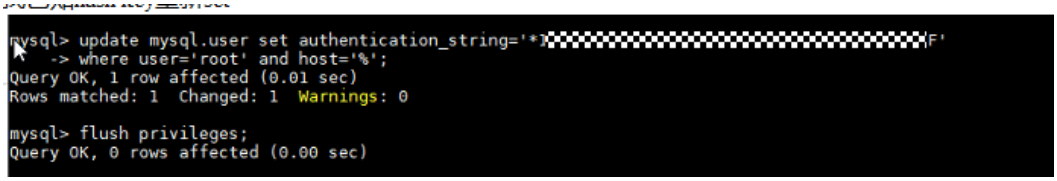
Solution

MySQL 8.0 does not support the password function. Therefore, you should perform the following steps to rectify the fault:

Step 1 Locate the **authentication_string** field of account **rdsAdmin** and run the following command:

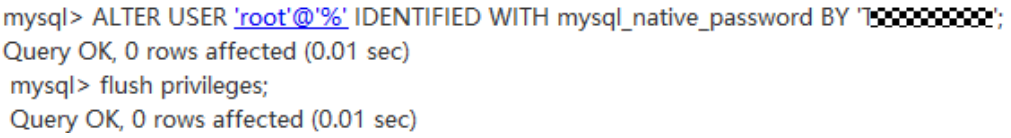
```
update mysql.user set authentication_string='XXX'
```

XXX indicates the new password.



Step 2 Reset the password of account **root**.

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



After the password is changed, the user can log in to the system as user **root**.

----End

6.12 MySQL-server Connection Failure After a Version Upgrade of RDS for MySQL

Scenario

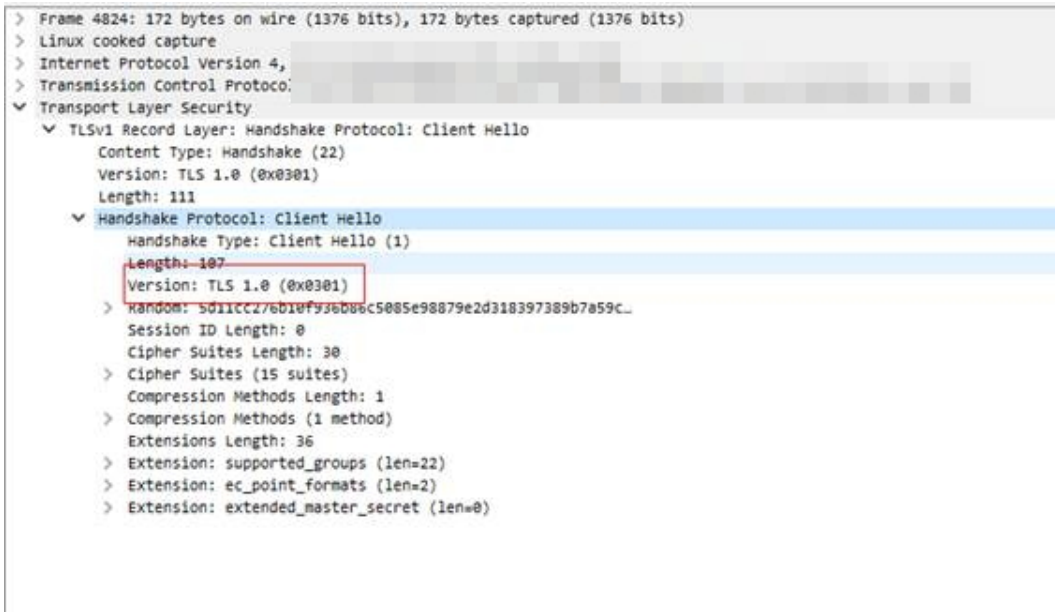
The following error is displayed when a database is connected to using commands:

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

MySQL-server connection failed after RDS for MySQL 5.7.23 is upgraded to 5.7.25. **Figure 6-1** shows the captured packet.

The TLS version sent from the client to the server during the TLS handshake is 1.0. A total number of 15 supported cipher suites are provided.

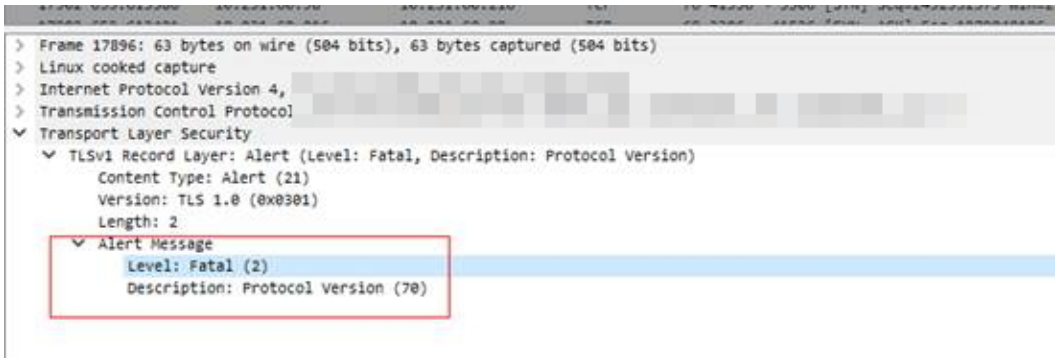
Figure 6-1 Packets captured when connection failed



Fault Analysis

As shown in the MySQL-server response in **Figure 6-2**, the server rejects the client connection because OpenSSL has been upgraded to 1.1.1a on MySQL 5.7.25, resulting in the rejection of the insecure TLS version and password suite.

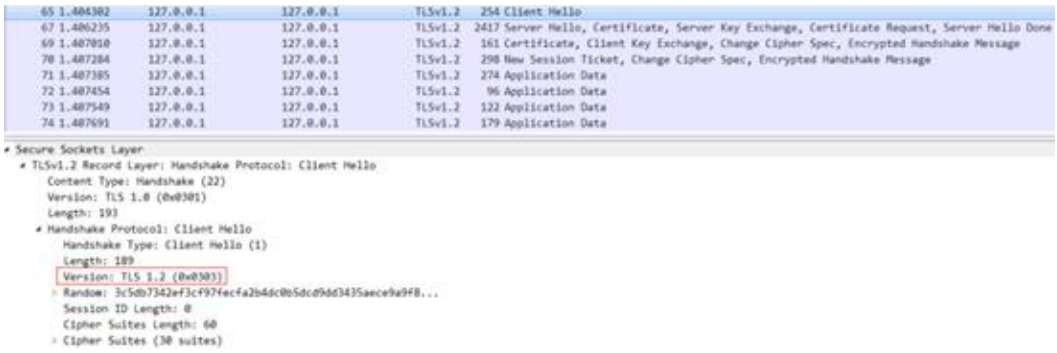
Figure 6-2 MySQL-server response



Solution

Upgrade your JDK client to **JDK 8 or a later version**. By default, TLS 1.2 is supported and 30 cipher suites are provided. **Figure 6-3** shows a normal captured packet.

Figure 6-3 Packets captured when connection is normal



6.13 Connection Exit Due to Improper Timeout Parameter Settings

Scenario

There are frequent database connection exits. As a result, subsequent statements fail to be executed.

Possible Causes

When a connector or API is used to connect to a database, the client has some default parameter settings. The settings of some important parameters, such as **socketTimeout** and **connectTimeout**, determine the client connection timeout duration. If the wait time of a connection exceeds the value of one of these parameters, the connection will be interrupted.

Solution

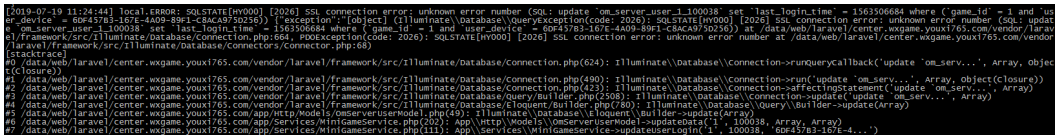
- Change the default values of parameters such as **socketTimeout** and **connectTimeout** to appropriate values.
- Pay attention to the reconnection function in the program.
- **Using connection pools is recommended.**

6.14 Database Connection Through Code (php/java/python) Failed After SSL Is Enabled

Scenario

After SSL is enabled, an error message is displayed when a database is connected to using commands.

Figure 6-4 Connection failure



Troubleshooting

Check whether the connection command uses SSL.

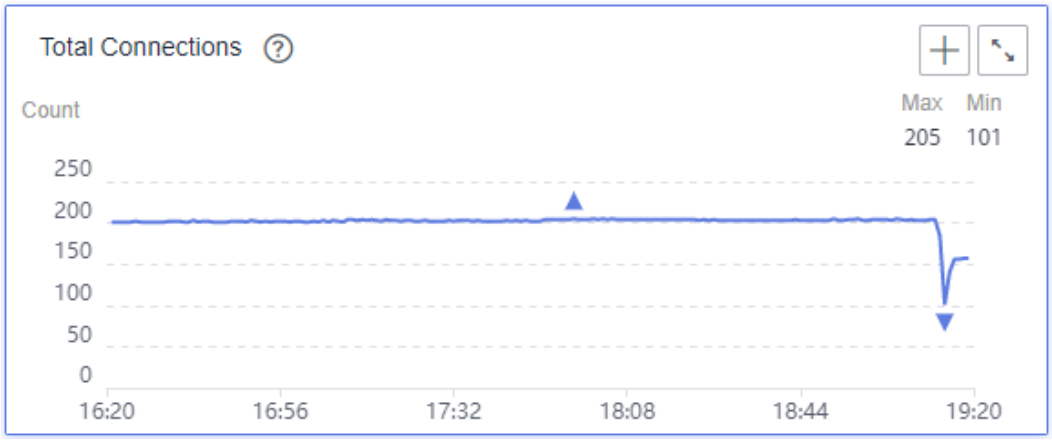
Solution

- Enable SSL and use an SSL connection to connect to a database. For details, see **Using MySQL CLI to Connect to an Instance Through a Private Network**.
- Disable SSL and use a non-SSL connection to connect to a database. For details, see **Buying a DB Instance and Connecting to It Using a MySQL Client**.

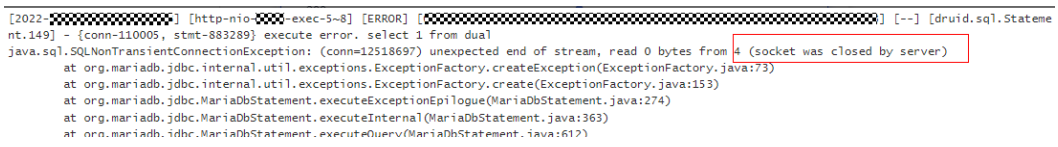
6.15 There Is a Disconnection Every 45 Days Due to the istio-citadel Certificate System

Scenario

The number of connections of multiple DB instances decreased sharply at the same time every 45 days. The following figure shows the number of total connections on the Cloud Eye console.

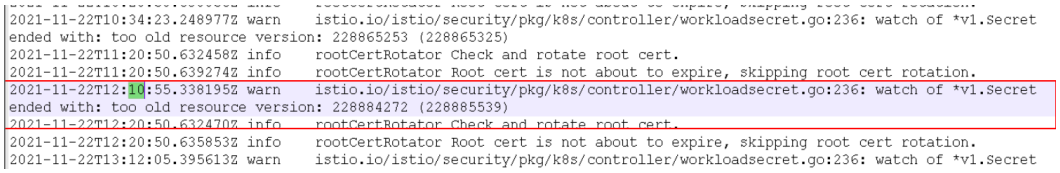


A large number of errors were reported on the client, as shown in the following figure.



Troubleshooting

1. Check whether a scheduled task with an interval of 45 days exists on the service side.
2. If the client uses a certificate encryption system, such as istio, analyze certificate-related logs and check whether information similar to the following is displayed: If yes, the problem is caused by expired certificates.



The life of an istio-citadel certificate is 45 days on the client. When the certificate has expired, the client initiates a database disconnection request.

Solution

- Set a proper expiration time for the istio-citadel certificate on the client and take preventive measures when the certificate expires.
- Check whether any other certificates have expired on the client.

6.16 Error 1251 Reported During Login to a DB Instance on the Navicat Client After the Database Version Is Upgraded

Scenario

The following error is reported when a regular user logs in to a DB instance through the Navicat client after the database version is upgraded: 1251 - Client does not support authentication protocol requested by server;consider upgrading MySQL client.

Possible Causes

The authentication plugin caching_sha2_password is used for accounts. The database proxy does not support this plugin of RDS for MySQL 8.0, so an error is reported during the login.

Solution

- Update the Navicat driver.
- Change the encryption rule for user login to mysql_native_password for RDS for MySQL 8.0 DB instances.

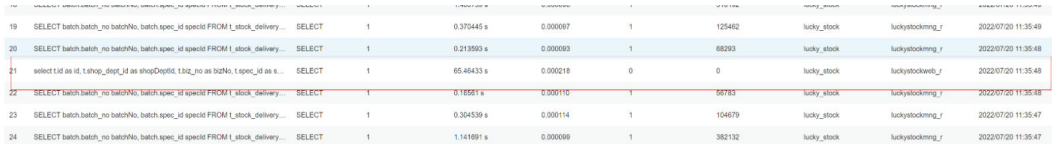
  Run the **select plugin from mysql.user where user="***user_name***";** statement on the DAS console to change the identity authentication plugin.

7 Other Issues

7.1 No Scanned Rows Recorded in Slow Query Logs

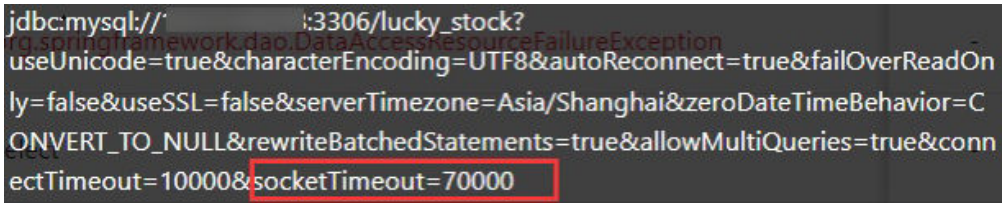
Scenario

In slow query logs, an SQL statement was executed for 65 seconds, but the number of scanned rows was 0.



Possible Causes

If an SQL statement is interrupted but its execution time exceeds the slow log threshold, the statement will be recorded in slow query logs and the number of scanned rows is 0. Timeout thresholds have been configured for the JDBC connection from the client.



Solution

Optimize the SQL statement or set **socketTimeout** to a more appropriate value.

7.2 Rows Recorded in the SQL Diagnosis Result Far Less Than the Scanned Rows Recorded in Slow Query Logs

Scenario

When SQL diagnosis is performed on the DAS console for an RDS for MySQL DB instance, the number of rows recorded in the execution plan in the SQL statement diagnosis result is far less than the number of scanned rows in slow SQL logs.

Possible Causes

When the query optimizer determines to query a table through full table scanning, the **rows** column in the execution plan indicates the estimated number of rows to be read. The number of rows in the execution plan is not the number of scanned rows, because the table may be scanned repeatedly during the query.

7.3 Millisecond-Level SQL Statements Recorded in Slow Query Logs

Scenario

The value of **long_query_time** is **1** (unit: second) for RDS for MySQL slow query logs. However, there are SQL statements whose average execution time is less than 1 second in slow query logs.

Troubleshooting

Check whether **log_queries_not_using_indexes** is set to **ON**. If yes, the SQL statements not using indexes will also be recorded in slow query logs.

7.4 Viewing Storage of RDS DB Instances

Scenario

The storage of an RDS instance refers to the data disk storage you have purchased, not including the ECS system disks.

You can use Cloud Eye to monitor the size, usage, and utilization of your storage space and set alarm rules.

NOTE

Storage of primary/standby DB instances refers to the storage of the primary DB instance.

Solution

Step 1 **Log in to the management console**.

Step 2 Click in the upper left corner and select a region.

Step 3 Click in the upper left corner of the page and choose **Databases** > **Relational Database Service**.

Step 4 On the **Instances** page, click the DB instance name.

Step 5 On the **Overview** page, view the storage space usage in the **Storage & Backup** area.

----End

7.5 "The table is full" Displayed in Error Logs

Scenario

The error message "ERROR [MY-013132] [Server] The table '*table_name*' is full!" is displayed in the error logs of an RDS for MySQL instance.

Possible Causes

This error may be reported when the TempTable engine is used. For details, see the **official documentation**.

Solution

On the RDS console, change the value of **internal_tmp_mem_storage_engine** to **MEMORY**.

7.6 Audit Log Upload Policy Description

Scenario

On the RDS console, audit logs are normally uploaded to OBS and a log file is generated every half an hour or for every 100 MB. In this case, however, the audit logs were uploaded twice within two minutes and the size of the audit logs was less than 100 MB.

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

Possible Causes

Audit log rotation policy: Logs are uploaded to OBS every half an hour or when the accumulated size reaches 100 MB.

1. The rotation interval for audit logs is half an hour. The time under **Updated** on the console is when the last SQL statement is written to the audit log file. Based on the update time, you can easily analyze downloaded logs.
2. In certain special cases, such as when the standby instance is being rebuilt or a primary/standby switchover is being performed, audit logs are forcibly rotated. As a result, a new audit log file less than 100 MB is generated within half an hour. This scenario is normal.

7.7 Auto-increment Field Values

RDS for MySQL uses the following methods to assign values to an auto-increment field:

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

1. If no value is specified for the auto-increment field, RDS for MySQL automatically enters the value of **AUTO_INCREMENT** to the field.

```
mysql> INSERT INTO animals (name) VALUES ('fish'),('cat'),('penguin'),('lax'),('whale'),('ostrich');
Query OK, 6 rows affected (0.01 sec)
Records: 6  Duplicates: 0  Warnings: 0
mysql> select * from animals;
+----+---------+
| id | name    |
+----+---------+
|  1 | fish    |
|  2 | cat     |
|  3 | penguin |
|  4 | lax     |
|  5 | whale   |
|  6 | ostrich |
+----+---------+
6 rows in set (0.00 sec)
mysql> show create table animals;
+---------+-----------------------------------------------------+
| Table   | Create Table                                        |
+---------+-----------------------------------------------------+
| animals | CREATE TABLE `animals` (  `id` mediumint NOT NULL AUTO_INCREMENT,  `name`
char(30) NOT NULL,  PRIMARY KEY (`id`)) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT
CHARSET=utf8 |
+---------+-----------------------------------------------------+
```

2. If **0** or **NULL** is specified for the auto-increment field, RDS for MySQL automatically enters the value of **AUTO_INCREMENT** to the field.

```
mysql> INSERT INTO animals (id,name) VALUES(0,'groundhog');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO animals (id,name) VALUES(NULL,'squirrel');
Query OK, 1 row affected (0.01 sec)
mysql> select * from animals;
+----+-----------+
| id | name      |
+----+-----------+
|  1 | fish      |
|  2 | cat       |
|  3 | penguin   |
```



```
|  4 | lax      |
|  5 | whale    |
|  6 | ostrich  |
|  7 | groundhog |
|  8 | squirrel |
+----+----------+8
rows in set (0.00 sec)
mysql> show create table animals;
+---------+-----------------------------------------------------------------+
| Table   | Create Table                                                    |
+---------+-----------------------------------------------------------------+
| animals | CREATE TABLE `animals` (  `id` mediumint NOT NULL AUTO_INCREMENT,  `name`
char(30) NOT NULL,  PRIMARY KEY (`id`)) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT
CHARSET=utf8 |
+---------+-----------------------------------------------------------------+
```

3. If the value *X* that is greater than the value of **AUTO_INCREMENT** is specified for the auto-increment field, RDS for MySQL inserts *X* to the field and changes **AUTO_INCREMENT** to *X* + 1.

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

4. If a value less than the value of **AUTO_INCREMENT** is specified for the auto-increment field, RDS for MySQL enters the value to the field and **AUTO_INCREMENT** remains unchanged.

```
mysql> INSERT INTO animals (id,name) VALUES(50,'middle');
Query OK, 1 row affected (0.00 sec)
mysql> select * from animals;
+-----+----------+
| id  | name     |
+-----+----------+
|   1 | fish     |
|   2 | cat      |
|   3 | penguin  |
|   4 | lax      |
|   5 | whale    |
|   6 | ostrich  |
|   7 | groundhog |
|   8 | squirrel |
|  50 | middle   |
| 100 | rabbit   |
+-----+----------+
10 rows in set (0.00 sec)
mysql> show create table animals;
+---------+-----------------------------------------------------------------+
| Table   | Create Table                                                    |
```



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

5. If a negative value is specified for the auto-increment field, RDS for MySQL enters the value to the field and **AUTO_INCREMENT** remains unchanged.

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

7.8 Starting Value and Increment of AUTO_INCREMENT

The starting value and increment of **AUTO_INCREMENT** are determined by the **auto_increment_offset** and **auto_increment_increment** parameters.

- **auto_increment_offset** determines the starting point for the AUTO_INCREMENT column value.
- **auto_increment_increment** controls the interval between successive column values.
- When the value of **auto_increment_offset** is greater than that of **auto_increment_increment**, the value of **auto_increment_offset** is ignored.
- When the value of **auto_increment_offset** is less than or equal to that of **auto_increment_increment**, the value of **AUTO_INCREMENT** is calculated as **auto_increment_offset** + *N* x **auto_increment_increment** (*N* indicates the number of inserted data records).

In RDS for MySQL, the values of **auto_increment_increment** and **auto_increment_offset** are both **1** by default. You can change them on the RDS console. For details, see **Modifying Parameters of an RDS for MySQL Instance**.

Example:

- If both **auto_increment_offset** and **auto_increment_increment** are set to **1**, the starting value is 1 and the increment is 1.

```
mysql> show variables like 'auto_inc%';
+--------------------------+-------+
```



```
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 1     |
| auto_increment_offset    | 1     |
+--------------------------+-------+
mysql> create table auto_test1(id int NOT NULL AUTO_INCREMENT, PRIMARY KEY (`id`));
Query OK, 0 rows affected (0.09 sec)
mysql> show create table auto_test1;
+------------+-----------------------------------------------------------------------------------+
| Table      | Create Table                                                                      |
+------------+-----------------------------------------------------------------------------------+
| auto_test1 | CREATE TABLE `auto_test1` (
`id` int NOT NULL AUTO_INCREMENT,   PRIMARY KEY (`id`) ) ENGINE=InnoDB DEFAULT
CHARSET=utf8 |
+------------+-----------------------------------------------------------------------------------+
mysql> insert into auto_test1 values(0), (0), (0);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> select * from auto_test1;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
+----+
3 rows in set (0.01 sec)
mysql> show create table auto_test1;
+------------+-----------------------------------------------------------------------------------------------+
| Table      | Create Table                                                                                  |
+------------+-----------------------------------------------------------------------------------------------+
| auto_test1 | CREATE TABLE `auto_test1` (
`id` int NOT NULL AUTO_INCREMENT,   PRIMARY KEY (`id`) ) ENGINE=InnoDB AUTO_INCREMENT=4
DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

- If **auto_increment_increment** is set to **2**, the increment is 2.

```
mysql> set session auto_increment_offset=2;
Query OK, 0 rows affected (0.02 sec)
mysql> show variables like 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 2     |
| auto_increment_offset    | 1     |
+--------------------------+-------+
mysql> insert into auto_test1 values(0), (0), (0);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> select * from auto_test1;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
|  4 |
|  6 |
|  8 |
+----+
6 rows in set (0.00 sec)
mysql> show create table auto_test1;
+------------+-----------------------------------------------+
| Table      | Create Table                                  |
+------------+-----------------------------------------------+
| auto_test1 | CREATE TABLE `auto_test1` (
`id` int NOT NULL AUTO_INCREMENT,   PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8      |
```



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

- If **auto_increment_offset** is set to **10** and **auto_increment_increment** is set to **2**, the starting value is 2 (because the value of **auto_increment_offset** is greater than that of **auto_increment_increment**) and the increment is 2.

```
mysql> set session auto_increment_offset=10;
mysql> set session auto_increment_increment=2;
mysql> show variables like 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 2     |
| auto_increment_offset    | 10    |
+--------------------------+-------+
mysql> create table auto_test2(id int NOT NULL AUTO_INCREMENT, PRIMARY KEY (`id`)); Query OK,
0 rows affected (0.08 sec)
mysql> show create table auto_test2;
+------------
+-----------------------------------------------------------------------------------------------------------------------
------+
| Table      | Create Table                                                                                                 |
+------------
+-----------------------------------------------------------------------------------------------------------------------
------+
| auto_test2 | CREATE TABLE `auto_test2` (   `id` int NOT NULL AUTO_INCREMENT,   PRIMARY KEY
(`id`) ) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+------------
+-----------------------------------------------------------------------------------------------------------------------
------+
1 row in set (0.01 sec)
mysql> insert into auto_test2 values(0), (0), (0);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> select * from auto_test2;
+----+
| id |
+----+
|  2 |
|  4 |
|  6 |
+----+
3 rows in set (0.01 sec)
mysql> show create table auto_test2;
+------------+-----------------------------------------------------------------------------------------------+
| Table      | Create Table                                                                                  |
+------------+-----------------------------------------------------------------------------------------------+
| auto_test2 | CREATE TABLE `auto_test2` (
`id` int NOT NULL AUTO_INCREMENT,   PRIMARY KEY (`id`) ) ENGINE=InnoDB AUTO_INCREMENT=8
DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------------------------------------------------------------+
```

- If **auto_increment_offset** is set to **5** and **auto_increment_increment** is set to **10**, the starting value is 5 and the increment is 10.

```
mysql> set session auto_increment_offset=5; mysql> set session auto_increment_increment=10;
mysql>  show variables like 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 10    |
| auto_increment_offset    | 5     |
+--------------------------+-------+
mysql> create table auto_test3(id int NOT NULL AUTO_INCREMENT, PRIMARY KEY (`id`));
mysql> show create table auto_test3;
+------------
+-----------------------------------------------------------------------------------------------------------------------
------+
| Table      | Create Table                                                                                                 |
+------------
```



```
+-------------------------------------------------------------------------------------------------------------------------
------+
| auto_test3 | CREATE TABLE `auto_test3` (   `id` int NOT NULL AUTO_INCREMENT,   PRIMARY KEY
(`id`) ) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+------------
+-------------------------------------------------------------------------------------------------------------------------
------+
mysql> insert into auto_test3 values(0), (0), (0);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> select * from auto_test3;
+----+
| id |
+----+
|  5 |
| 15 |
| 25 |
+----+
mysql> show create table auto_test3;
+------------+-----------------------------------------+
| Table      | Create Table                            |
+------------+-----------------------------------------+
| auto_test3 | CREATE TABLE `auto_test3` (
`id` int NOT NULL AUTO_INCREMENT,   PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=35 DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------+
```

7.9 AUTO_INCREMENT Value Exceeding the Maximum Value of This Field plus 1

If the value of **AUTO_INCREMENT** is not equal to the maximum value of this field plus 1 in a data table, the possible causes are as follows:

- If the increment is not 1, the value of **AUTO_INCREMENT** is equal to the maximum value of this field plus the increment. For details, see **Starting Value and Increment of AUTO_INCREMENT**.

```
mysql> show variables like 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 2     |
| auto_increment_offset    | 1     |
+--------------------------+-------+
mysql> select * from auto_test1;
+----+
| id |
+----+
|  2 |
|  4 |
|  6 |
|  8 |
+----+
mysql> show create table auto_test1;
+------------+-----------------------------------------+
| Table      | Create Table                            |
+------------+-----------------------------------------+
| auto_test1 | CREATE TABLE `auto_test1` (
`id` int NOT NULL AUTO_INCREMENT,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------+
```

- The value of **AUTO_INCREMENT** was changed.

```
mysql> select * from animals;
+----+-----------+
```



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

- A transaction was not committed or was rolled back, so the value of **AUTO_INCREMENT** increased but did not go back down after the transaction was rolled back.

```
mysql> show create table auto_test1;
+------------+----------------------------------------+
| Table      | Create Table                           |
+------------+----------------------------------------+
| auto_test1 | CREATE TABLE `auto_test1` (
`id` int NOT NULL AUTO_INCREMENT,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8 |
+------------+----------------------------------------+
1 row in set (0.00 sec)
mysql> select * from auto_test1;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
+----+
mysql> begin;
Query OK, 0 rows affected (0.02 sec)
mysql> insert into auto_test1 values (0),(0),(0);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> select * from auto_test1;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
|  4 |
|  5 |
|  6 |
+----+
6 rows in set (0.00 sec)
mysql> show create table auto_test1;
```



```
+------------+-----------------------------------------+
| Table      | Create Table                            |
+------------+-----------------------------------------+
| auto_test1 | CREATE TABLE `auto_test1` (
`id` int NOT NULL AUTO_INCREMENT,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------+
1 row in set (0.00 sec)
mysql> rollback;
Query OK, 0 rows affected (0.05 sec)
mysql> select * from auto_test1;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
+----+
3 rows in set (0.00 sec)
mysql> show create table auto_test1;
+------------+-----------------------------------------+
| Table      | Create Table                            |
+------------+-----------------------------------------+
| auto_test1 | CREATE TABLE `auto_test1` (
`id` int NOT NULL AUTO_INCREMENT,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------+
```

- After data is inserted, the value of **AUTO_INCREMENT** changed, but when the corresponding data row was deleted, the value of **AUTO_INCREMENT** did not decrease.

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



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

7.10 Auto-Increment Field Value Jump

If the values of the auto-increment field are discontinuous, possible causes including the following:

- If the increment is not 1, the values of the auto-increment field are discontinuous.

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

- The value of **AUTO_INCREMENT** was changed.

```
mysql> select * from animals;
+----+-----------+
| id | name      |
+----+-----------+
|  1 | fish      |
|  2 | cat       |
|  3 | penguin   |
+----+-----------+
mysql> show create table animals;
+---------+-------------------------------------------------------+
| Table   | Create Table                                          |
+---------+-------------------------------------------------------+
| animals | CREATE TABLE `animals` (
`id` mediumint NOT NULL AUTO_INCREMENT,
`name` char(30) NOT NULL,
PRIMARY KEY (`id`) )
```



```
ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8 |
+---------+-------------------------------------------------------+
mysql> alter table animals AUTO_INCREMENT=100;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0
mysql> show create table animals;
+---------+-------------------------------------------------------+
| Table   | Create Table                                          |
+---------+-------------------------------------------------------+
| animals | CREATE TABLE `animals` (
`id` mediumint NOT NULL AUTO_INCREMENT,
`name` char(30) NOT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=100 DEFAULT CHARSET=utf8         |
+---------+-------------------------------------------------------+
mysql> INSERT INTO animals (id,name) VALUES(0,'rabbit');
Query OK, 1 row affected (0.00 sec)
mysql> select * from animals;
+-----+-----------+
| id  | name      |
+-----+-----------+
|   1 | fish      |
|   2 | cat       |
|   3 | penguin   |
| 100 | rabbit    |
+-----+-----------+
9 rows in set (0.00 sec)
```

- The value of the auto-increment field was specified when data was inserted.

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

- A transaction was not committed or was rolled back, so the value of **AUTO_INCREMENT** increased, but then it did not go back down after the rollback. When data is inserted again, the value of the auto-increment field jumps.

```
mysql> show create table auto_test1;
+------------+----------------------------------------+
| Table      | Create Table                           |
+------------+----------------------------------------+
| auto_test1 | CREATE TABLE `auto_test1` (
`id` int NOT NULL AUTO_INCREMENT,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8 |
+------------+----------------------------------------+
1 row in set (0.00 sec)
mysql> select * from auto_test1;
+----+
| id |
+----+
|  1 |
```



```
|  2 |
|  3 |
+----+
mysql> begin;
Query OK, 0 rows affected (0.02 sec)
mysql> insert into auto_test1 values (0),(0),(0);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> select * from auto_test1;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
|  4 |
|  5 |
|  6 |
+----+
6 rows in set (0.00 sec)
mysql> show create table auto_test1;
+------------+-----------------------------------------+
| Table      | Create Table                            |
+------------+-----------------------------------------+
| auto_test1 |
CREATE TABLE `auto_test1` (
`id` int NOT NULL AUTO_INCREMENT,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------+
1 row in set (0.00 sec)
mysql> rollback;
Query OK, 0 rows affected (0.05 sec)
mysql> select * from auto_test1;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
+----+
3 rows in set (0.00 sec)
mysql> show create table auto_test1;
+------------+-----------------------------------------+
| Table      | Create Table                            |
+------------+-----------------------------------------+
| auto_test1 | CREATE TABLE `auto_test1` (
`id` int NOT NULL AUTO_INCREMENT,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------+
mysql> insert into auto_test1 values (0),(0),(0);
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> select * from auto_test1;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
|  7 |
|  8 |
|  9 |
+----+
6 rows in set (0.00 sec)
mysql> show create table auto_test1;
+------------+-----------------------------------------+
| Table      | Create Table                            |
```

```
+------------+-----------------------------------------+
| auto_test1 | CREATE TABLE `auto_test1` (
`id` int NOT NULL AUTO_INCREMENT,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------+
```

- After data is inserted, the value of **AUTO_INCREMENT** changes. But when the corresponding data row is deleted, the value of **AUTO_INCREMENT** does not decrease. When data is inserted again, the value of the auto-increment field jumps.

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



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

- If data insertion fails due to some reasons (for example, unique key conflict), the value of **AUTO_INCREMENT** may jump.

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

- When data is inserted in batches (such as **insert...select** and **load file**), the auto-increment key is requested in batches. Two to the power of *n* sequence numbers are requested in each batch. If the sequence numbers are not used up, the sequence numbers will not be returned. As a result, the value of **AUTO_INCREMENT** may jump.

```
mysql> create table auto_test5_tmp(id tinyint not null AUTO_INCREMENT, name varchar(8), PRIMARY
KEY (`id`));
Query OK, 0 rows affected (0.08 sec)
mysql> select * from auto_test5;
+----+------+
| id | name |
+----+------+
|  1 | A    |
```



```
|  2 | B    |
|  3 | C    |
|  4 | X    |
|  5 | Y    |
|  6 | Z    |
|  8 | A    |
|  9 | B    |
| 10 | C    |
| 11 | X    |
| 12 | Y    |
| 13 | Z    |
+----+------+
12 rows in set (0.00 sec)
mysql> insert into auto_test5_tmp select 0,name from auto_test5;
Query OK, 12 rows affected (0.01 sec)
Records: 12  Duplicates: 0  Warnings: 0
mysql> select * from auto_test5_tmp;
+----+------+
| id | name |
+----+------+
|  1 | A    |
|  2 | B    |
|  3 | C    |
|  4 | X    |
|  5 | Y    |
|  6 | Z    |
|  7 | A    |
|  8 | B    |
|  9 | C    |
| 10 | X    |
| 11 | Y    |
| 12 | Z    |
+----+------+
12 rows in set (0.00 sec)
mysql> show create table auto_test5_tmp;
+-----------------+-------------------------------------------------------------+
| Table           | Create Table                                                |
+-----------------+-------------------------------------------------------------+
| auto_test5_tmp | CREATE TABLE `auto_test5_tmp` (  `id` tinyint NOT NULL AUTO_INCREMENT,
`name` varchar(8) DEFAULT NULL,  PRIMARY KEY (`id`)) ENGINE=InnoDB AUTO_INCREMENT=16
DEFAULT CHARSET=utf8 |
+-----------------+-------------------------------------------------------------+
```

7.11 Changing the AUTO_INCREMENT Value of a Table

The methods are as follows:

- If the value of **AUTO_INCREMENT** is greater than the maximum value of the auto-increment column in the table, **AUTO_INCREMENT** can be changed to a larger value within the value range.

```
mysql> show create table animals;
+---------+-----------------------------------------------------------------------+
| Table   | Create Table                                                          |
+---------+-----------------------------------------------------------------------+
| animals | CREATE TABLE `animals` (
`id` mediumint NOT NULL AUTO_INCREMENT,   `name` char(30) NOT NULL,
PRIMARY KEY (`id`) ) ENGINE=InnoDB AUTO_INCREMENT=101 DEFAULT CHARSET=utf8   |
+---------+-----------------------------------------------------------------------+
1 row in set (0.00 sec)
mysql> select * from animals;
+-----+-----------+
| id  | name      |
+-----+-----------+
| -50 | -middle   |
|   1 | fish      |
|   2 | cat       |
|  50 | middle    |
```



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

- If the new value of **AUTO_INCREMENT** is still greater than the maximum value of the auto-increment column in the table, the change was successful. Otherwise, the value is changed to the maximum value of the auto-increment column plus 1 by default.

```
mysql> select * from animals;
+-----+-----------+
| id  | name      |
+-----+-----------+
| -50 | -middle   |
|   1 | fish      |
|   2 | cat       |
|  50 | middle    |
| 100 | rabbit    |
+-----+-----------+
mysql> show create table animals;
+---------+-------------------------------------------------------+
| Table   | Create Table                                          |
+---------+-------------------------------------------------------+
| animals | CREATE TABLE `animals` (
`id` mediumint NOT NULL AUTO_INCREMENT,   `name` char(30) NOT NULL,
PRIMARY KEY (`id`) ) ENGINE=InnoDB AUTO_INCREMENT=200 DEFAULT CHARSET=utf8 |
+---------+-------------------------------------------------------+
mysql> alter table animals AUTO_INCREMENT=150;
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0
mysql> show create table animals;
+---------+-------------------------------------------------------+
| Table   | Create Table                                          |
+---------+-------------------------------------------------------+
| animals | CREATE TABLE `animals` (
`id` mediumint NOT NULL AUTO_INCREMENT,   `name` char(30) NOT NULL,
PRIMARY KEY (`id`) ) ENGINE=InnoDB AUTO_INCREMENT=150 DEFAULT CHARSET=utf8 |
+---------+-------------------------------------------------------+
mysql> alter table animals AUTO_INCREMENT=50;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0
mysql> show create table animals;
+---------+-------------------------------------------------------+
| Table   | Create Table                                          |
+---------+-------------------------------------------------------+
| animals | CREATE TABLE `animals` (
`id` mediumint NOT NULL AUTO_INCREMENT,   `name` char(30) NOT NULL,
PRIMARY KEY (`id`) ) ENGINE=InnoDB AUTO_INCREMENT=101 DEFAULT CHARSET=utf8 |
+---------+-------------------------------------------------------+
mysql> delete from animals where id=100;
Query OK, 1 row affected (0.00 sec)
mysql> select * from animals;
+-----+-----------+
| id  | name      |
+-----+-----------+
| -50 | -middle   |
|   1 | fish      |
|   2 | cat       |
|  50 | middle    |
```



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

- The value of **AUTO_INCREMENT** cannot be changed to a negative number.

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

7.12 Failed to Insert Data Because Values for the Auto-increment Primary Key Field Reach the Upper Limit

Scenario

The error message "ERROR 1062 (23000): Duplicate entry 'xxx' for key 'xxx'" was displayed when data was inserted into a table.

Possible Causes

The values for the auto-increment primary key field reach the upper limit and cannot be increased. As a result, the auto-increment primary key value generated for the newly inserted data is the same as that of the previous data record in the table. Since the auto-increment primary key values cannot be duplicate, an error is reported.

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



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

Solution

- If there are many data changes and the actual data volume in the table is far less than the capacity of the auto-increment primary key, import all data in the table to a new table, delete the original table, and change the name of the new table to the original table name. (There are multiple methods for importing and exporting data. The following is only an example.)

```
mysql>  create table auto_test5_tmp(id tinyint not null AUTO_INCREMENT, name varchar(8),
PRIMARY KEY (`id`));
Query OK, 0 rows affected (0.07 sec)
mysql> insert into auto_test5_tmp select 0,name from auto_test5;
Query OK, 6 rows affected (0.01 sec)
Records: 6  Duplicates: 0  Warnings: 0
mysql> select * from auto_test5_tmp;
+----+------+
| id | name |
+----+------+
|  1 | A    |
|  2 | B    |
|  3 | C    |
|  4 | X    |
|  5 | Y    |
|  6 | Z    |
+----+------+
mysql> drop table auto_test5;
mysql> rename table auto_test5_tmp to auto_test5;
Query OK, 0 rows affected (0.12 sec)
mysql> select * from auto_test5;
+----+------+
| id | name |
+----+------+
|  1 | A    |
|  2 | B    |
|  3 | C    |
|  4 | X    |
|  5 | Y    |
|  6 | Z    |
+----+------+
6 rows in set (0.01 sec)
mysql> show create table auto_test5;
+------------+-----------------------------------------------------------------+
| Table      | Create Table                                                    |
+------------+-----------------------------------------------------------------+
| auto_test5 | CREATE TABLE `auto_test5` (
`id` tinyint NOT NULL AUTO_INCREMENT,   `name` varchar(8) DEFAULT NULL,
```



```
PRIMARY KEY (`id`) ) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------------------------------+
```

- If the values for the auto-increment primary key are not enough, change the field type of the auto-increment primary key to store more data.

```
mysql> select * from auto_test6;
+-----+------+
| id  | name |
+-----+------+
|   1 | A    |
|   2 | B    |
|   3 | C    |
| 125 | X    |
| 126 | Y    |
| 127 | Z    |
+-----+------+
6 rows in set (0.00 sec)
mysql> show create table auto_test6;
+------------+-----------------------------------------------------------------+
| Table      | Create Table                                                    |
+------------+-----------------------------------------------------------------+
| auto_test6 | CREATE TABLE `auto_test6` (
`id` tinyint NOT NULL AUTO_INCREMENT,   `name` varchar(8) DEFAULT NULL,
PRIMARY KEY (`id`) ) ENGINE=InnoDB AUTO_INCREMENT=127 DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------------------------------+
mysql> alter table auto_test6  modify column id int NOT NULL AUTO_INCREMENT;
Query OK, 6 rows affected (0.15 sec)
Records: 6  Duplicates: 0  Warnings: 0
mysql> show create table auto_test6;
+------------+-----------------------------------------------------------------+
| Table      | Create Table                                                    |
+------------+-----------------------------------------------------------------+
| auto_test6 | CREATE TABLE `auto_test6` (
`id` int NOT NULL AUTO_INCREMENT,   `name` varchar(8) DEFAULT NULL,
PRIMARY KEY (`id`) ) ENGINE=InnoDB AUTO_INCREMENT=128 DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------------------------------+
1 row in set (0.00 sec)
mysql> insert into auto_test6(name) values('D');
Query OK, 1 row affected (0.01 sec)
mysql> select * from auto_test6;
+-----+------+
| id  | name |
+-----+------+
|   1 | A    |
|   2 | B    |
|   3 | C    |
| 125 | X    |
| 126 | Y    |
| 127 | Z    |
| 128 | D    |
+-----+------+
7 rows in set (0.00 sec)
mysql> show create table auto_test6;
+------------+-----------------------------------------------------------------+
| Table      | Create Table                                                    |
+------------+-----------------------------------------------------------------+
| auto_test6 | CREATE TABLE `auto_test6` (
`id` int NOT NULL AUTO_INCREMENT,   `name` varchar(8) DEFAULT NULL,
PRIMARY KEY (`id`) ) ENGINE=InnoDB AUTO_INCREMENT=129 DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------------------------------+
1 row in set (0.01 sec)
```

7.13 The Impact of Creating an Empty Username

The username '' is allowed in RDS for MySQL instances, but using such an empty username negatively impacts instances.

When you perform operations on an RDS for MySQL instance using an empty username, any username can be matched in RDS. This impacts both security and functionality. You are advised not to use empty usernames.

- Security impact
  - Your instance can be connected to using any username if an empty username exists.
  - Your database can be logged in to using any username and the password of the empty username and the login user will obtain all permissions of the empty username. Example:

```
#If there is no empty username created and the invalid username abcd is used to connect to the instance, the connection fails.
mysql> select user,host from mysql.user;
+------------------+-----------+
| user             | host      |
+------------------+-----------+
| root             | %         |
| mysql.infoschema | localhost |
| mysql.session    | localhost |
| mysql.sys        | localhost |
+------------------+-----------+
mysql -uabcd -h127.0.0.1 -P3306 -pTest_1234
mysql: [Warning] Using a password on the command line interface can be insecure.
ERROR 1045 (28000): Access denied for user 'abcd'@'localhost' (using password: YES)

#If an empty username has been created and the invalid username abcd and the password of the empty username are used to connect to the instance, the connection is successful.
mysql> create user ''@'localhost' IDENTIFIED BY 'Test_1234';
mysql> select user,host from mysql.user;
+------------------+-----------+
| user             | host      |
+------------------+-----------+
| root             | %         |
|                  | localhost |
| mysql.infoschema | localhost |
| mysql.session    | localhost |
| mysql.sys        | localhost |
+------------------+-----------+
mysql -uabcd -h127.0.0.1 -P3306 -pTest_1234
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 37Server version: 8.0.22-debug Source distribution
Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its affiliates.
Other names may be trademarks of their respective owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
```

  - If the empty user does not have a password, you can use any username to log in to the instance without a password and obtain all permissions of the empty user. Example:

```
#If there is an empty username that does not have a password, the database can be logged in to using any username without a password.
mysql> create user ''@'localhost';
Query OK, 0 rows affected (8.87 sec)
mysql> select user,host from mysql.user;
+------------------+-----------+
| user             | host      |
+------------------+-----------+
| root             | %         |
|                  | localhost |
| mysql.infoschema | localhost |
| mysql.session    | localhost |
| mysql.sys        | localhost |
+------------------+-----------+
mysql -uabcd -h127.0.0.1 -P3306
```



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

- Functional impact

  If an empty username exists, the database cannot be logged in to using a normal username due to a name matching error.

  Example: If the host of an empty user overlaps that of the **root** user, the **root** user cannot log in to the database using its password or it can log in to the database using the password of the empty username but cannot obtain **root** permissions.

```
mysql> create user ''@'localhost';
Query OK, 0 rows affected (8.87 sec)
mysql> select user,host from mysql.user;
+------------------+-----------+
| user             | host      |
+------------------+-----------+
| root             | %         |
|                  | localhost |
| mysql.infoschema | localhost |
| mysql.session    | localhost |
| mysql.sys        | localhost |
+------------------+-----------+
#The database cannot be logged in to using the password of the root user.
mysql -uroot -h127.0.0.1 -P3306 -pTest_root
mysql: [Warning] Using a password on the command line interface can be insecure.
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)
#The user who logs in to the database using the password of the empty user (password-free) is actually an empty user so the user does not have the root permissions.
mysql -uroot -h127.0.0.1 -P3306
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 45Server version: 8.0.22-debug Source distribution
Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its affiliates.
Other names may be trademarks of their respective owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> select user,host from mysql.user;
ERROR 1142 (42000): SELECT command denied to user ''@'localhost' for table 'user'
mysql>
```

7.14 Connection to a Primary/Standby DB Instance Suspended Using pt-osc

Scenario

The pt-online-schema-change (pt-osc) tool can be used to perform online DDL operations on a local single instance, but cannot be used on an RDS for MySQL primary/standby instance and the connection is suspended, as shown in the following figure.

Possible Causes

How pt-osc works:

1. Create an empty table with the same structure as the original table but with the suffix **_new** added to the table name.
2. Modify the structure of the empty table created in 1.
3. Add three triggers to the original table: **delete**, **update**, and **insert**. The triggers are used to execute the statements to be executed in the original table in the new table during data copy.
4. Copy the data in the original table to the new table in the form of data chunks.
5. Rename the original table, change the name of the new table to that of the original table, and delete the original table.
6. Delete the triggers.

A large amount of data needs to be copied. There will be a replication delay between the RDS for MySQL primary and standby instances. Workloads running on the standby instance may be affected. Considering the replication delay, the pt-osc provides the following options:

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

If the replication delay of the standby instance exceeds the value of **max-lag**, the tool stops copying data for **check-interval** seconds. If you specify **check-slave-lag**, the tool only monitors that particular server for replication delay. It does not monitor other servers. **recursion-method** is used to control exactly which servers the tool monitors. Its values include **processlist** (default value, monitoring the primary/standby replication delay), **hosts**, **dsn**, and **none** (ignoring the primary/standby replication delay). For more information, see **pt-online-schema-change**.

In this case:

- When pt-osc is used to connect to the RDS for MySQL primary/standby instance, the connection is suspended because there is a primary/standby replication delay and the tool stops copying data. You can add **--recursion-method=none** to solve the problem.
- If the primary/standby replication delay is ignored, data copy becomes fast. To minimize the impact on workloads, you can set the **--max-load** configuration item.

Solution

Add the **--recursion-method=none** configuration item to the pt-osc command to ignore the replication delay.

Common uses of pt-osc:

- Adding a field

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

- Deleting a field

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

- Modifying a field

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

- Renaming a field

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

- Adding an index

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

- Deleting an index

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

If the primary/standby replication delay is important for your workloads, adjust the following parameters as required: **max-lag**, **check-interval**, **recursion-method**, and **check-slave-lag**. For more information, see **pt-online-schema-change**.

7.15 Error Reported During Payment for a DB Instance

Scenario

When a user purchases a yearly/monthly RDS DB instance, the page does not respond or the following error is reported after the user clicks **Pay Now**: Policy doesn't allow bss:order:update to be performed.

Possible Causes

The user does not have operation permissions on orders.

Solution

Contact the master account administrator to add the **bss:order:update** permission.

7.16 Failed to Change a Database Name

Scenario

Database names cannot be changed for RDS for MySQL DB instances on the RDS or DAS console.

Solution

Use Data Replication Service (DRS) to migrate data from the source database to the destination database with a different name. For details, see **Migration Solution Overview**.

7.17 Error Reported When a DB Instance Is Purchased

Scenario

When an IAM user purchases an RDS DB instance, an error message is displayed, indicating that the user is not granted the IAM agency permission.

Possible Causes

If the user selects **Enable autoscaling** when purchasing a DB instance, an error may be reported due to the lack of required permissions.

Solution

Configure required actions for the user. For details, see "Common operations and supported actions" > "Configuring autoscaling" in **Permissions**.

