MYSQL 基于 ROW 格式的复制 replication 与恢复 recovery

摘要：MYSQL BINLOG 有三种格式,分别是 Statement、Row、Mixed。Statement 记录了用户执行的原始 SQL,而 Row 则是记录了行的修改情况,在 MySQL 5.5 版本默认是 STATEMENT 模式（SBR）,在 MySQL 5.6 以上的版本默认是 Mixd 格式,但为了保证复制数据的完整性,建议生产环境都使用 Row 格式,就前面所说的 Row 记录的是行数据的修改情况,而不是原始 SQL。那么线上或者测试环境误删除或者更新几条数据后,又想恢复,那怎么办呢?下面演示基于 BINLOG 格式为 Row 的误操作后数据恢复,那么怎么把 Binlog 解析出来生成反向的原始 SQL 呢?下面基于 mysql5.7 来实现。

二进制日志格式	Statement（SBR）	Row（RBR）	Mixed（MBR）
日志内容	修改数据的 sql 语句	记录修改过的值	两种模式的混合使用 一般的语句修改使用 statment 格式保存 binlog；若一些函数，statement 无法完成主从复制的操作，则采用 row 格式保存 MySQL 会根据执行的每一条具体的 sql 语句来区分对待记录的日志形式，也就是在 Statement 和 Row 之间选择一种
优点	减少了 binlog 日志量，节约 IO，提高性能	非常清楚的记录下每一行数据修改的细节。而且不会出现某些特定情况下的存储过程，或 function，以及 trigger 的调用和触发无法被正确复制的问题	
缺点	在某些情况下会导致 master-slave 中的数据不一致(如 sleep()函数， last_insert_id()，以及 user-defined functions(udf)等会出现问题)	会产生大量的日志，尤其是 alter table 的时候会让日志暴涨	

情景 1：全备份数据丢失或损坏，数据被误删除后如何恢复？

情节 2：数据更新了又想恢复，怎么做？

情景 1：全备份数据丢失或损坏，数据被误删除后如何恢复？

数据库版本	二进制日志默认模式	打开 binlog 并设置为 row 格式
MySQL5.7	ROW（RBR）	[mysqld] log-bin=/var/lib/mysql-binlog/mastera server-id=1
Mariadb5.5	Statement（SBR）	[mysqld] log-bin=/var/lib/mysql-binlog/mastera binlog-format=row

1.1 打开 ***binlog***

```
[root@mastera0 ~]# vim /etc/my.cnf
# 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
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
# Disabling symbolic-links is recommended to prevent assorted security risks
symbolic-links=0
#新增两行#########
log-bin=/var/lib/mysql-binlog/mastera
server-id=1
############
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
```

1.2 查看默认日志格式

```
mysql> show variables like "binlog_format";
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| binlog_format | ROW   |
+---------------+-------+
1 row in set (0.01 sec)
```

1.3 误操作

写一个 sql 脚本，并导入数据库，创建数据库 db1，新建表 t1，并插入三行数据；进入数据库开启一个事务，删除 db1.t1 中的所有数据，并提交。

```
[root@workstation0 ~]# vi mysql.sql
create database db1;
use db1;
create table t1 (id int,name varchar(20));
insert into t1 values (1,'booboo'),(2,'tom'),(3,'mark');
[root@mastera0 ~]# mysql -uroot -p'(UplooOOking)' < mysql.sql

mysql> select * from db1.t1;
+------+--------+
| id   | name   |
+------+--------+
|    1 | booboo |
|    2 | tom    |
|    3 | mark   |
+------+--------+
3 rows in set (0.00 sec)

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> delete from db1.t1;
Query OK, 3 rows affected (0.00 sec)

mysql> select * from db1.t1;
Empty set (0.00 sec)

mysql> rollback;
Query OK, 0 rows affected (0.07 sec)

mysql> select * from db1.t1;
+------+--------+
| id   | name   |
+------+--------+
|    1 | booboo |
|    2 | tom    |
|    3 | mark   |
+------+--------+
3 rows in set (0.00 sec)

mysql> delete from db1.t1;
Query OK, 3 rows affected (0.18 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from db1.t1;
Empty set (0.00 sec)
```

1.4 查看日志

```
[root@mastera0 mysql-binlog]# ls
mastera.000001  mastera.index
```

1.4.1 show binlog events

在mysql数据库中通过【show binlog events in 'mastera.000001';】来查看日志内容；只能看到(db1.t1)表做了改动，但具体改了什么，就不知道了。

```
mysql> show binlog events in 'mastera.000001';
+---------------+------+---------------+-----------+-------------
+-----------------------------------------------------+
| Log_name       | Pos  | Event_type     | Server_id | End_log_pos | Info
|
+---------------+------+---------------+-----------+-------------
+-----------------------------------------------------+
| mastera.000001 |    4 | Format_desc    |         1 |         123 | Server
ver: 5.7.10-log, Binlog ver: 4                |
| mastera.000001 |  123 | Previous_gtids |         1 |         154 |
|
| mastera.000001 |  154 | Anonymous_Gtid |         1 |         219 | SET
@@SESSION.GTID_NEXT= 'ANONYMOUS'                  |
| mastera.000001 |  219 | Query          |         1 |         310 | create
database db1                                    |
| mastera.000001 |  310 | Anonymous_Gtid |         1 |         375 | SET
@@SESSION.GTID_NEXT= 'ANONYMOUS'                  |
| mastera.000001 |  375 | Query          |         1 |         488 | use
`db1`; create table t1 (id int,name varchar(20)) |
| mastera.000001 |  488 | Anonymous_Gtid |         1 |         553 | SET
@@SESSION.GTID_NEXT= 'ANONYMOUS'                  |
| mastera.000001 |  553 | Query          |         1 |         624 | BEGIN
|
| mastera.000001 |  624 | Table_map      |         1 |         671 | table_id:
108 (db1.t1)                                     |
| mastera.000001 |  671 | Write_rows     |         1 |         737 | table_id:
108 flags: STMT_END_F                            |
| mastera.000001 |  737 | Xid            |         1 |         768 | COMMIT /*
xid=7 */                                         |
| mastera.000001 |  768 | Anonymous_Gtid |         1 |         833 | SET
@@SESSION.GTID_NEXT= 'ANONYMOUS'                  |
| mastera.000001 |  833 | Query          |         1 |         901 | BEGIN
|
| mastera.000001 |  901 | Table_map      |         1 |         948 | table_id:
108 (db1.t1)                                     |
| mastera.000001 |  948 | Delete_rows    |         1 |        1014 | table_id:
108 flags: STMT_END_F                            |
| mastera.000001 | 1014 | Xid            |         1 |        1045 | COMMIT /*
xid=16 */                                        |
+---------------+------+---------------+-----------+-------------
+-----------------------------------------------------+
16 rows in set (0.00 sec)
```

1.4.2 mysqlbinlog

通过【mysqlbinlog mastera.000001】这里只能看到(db1.t1)表做了改动，但具体改了什么，同样也不知道。那么怎样才能看到到底改了什么呢？

Mysqlbinlog 参数解释	
--verbose(或-v)	将改动生成带注释的语句；
-v -v	会生成字段的类型、长度、是否为 null 等属性信息。
--base64-output=DECODE-ROWS	去掉 BINLOG 开头的那些信息，如下 BINLOG ' iaxjVw8BAAAAdwAAAHsAAAABAAQANS43LjEwLWxvZwAAAA AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAAACJrGNXEzgNAAgAEgAEBAQEEgAAXw AEGggAAAAICAgCAAAACgoKKioAEjQA AVbNJnk= '/*!*/;
eg. mysqlbinlog mastera.000001 -v -v --base64-output=DECODE-ROWS	

```
[root@mastera0 mysql-binlog]# mysqlbinlog mastera.000001 -v -v --base64-
output=DECODE-ROWS
/*!50530 SET @@SESSION.PSEUDO_SLAVE_MODE=1*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
DELIMITER /*!*/;
# at 4
#160617 15:53:45 server id 1  end_log_pos 123 CRC32 0x7926cd56     Start:
binlog v 4, server v 5.7.10-log created 160617 15:53:45 at startup
# Warning: this binlog is either in use or was not closed properly.
ROLLBACK/*!*/;
# at 123
#160617 15:53:45 server id 1  end_log_pos 154 CRC32 0xb6a9744f     Previous-
GTIDs
# [empty]
# at 154
#160617 16:00:20 server id 1  end_log_pos 219 CRC32 0xeb96e5d8
      Anonymous_GTID   last_committed=0 sequence_number=1
SET @@SESSION.GTID_NEXT= 'ANONYMOUS'/*!*/;
# at 219
#160617 16:00:20 server id 1  end_log_pos 310 CRC32 0x91ad8fe9     Query
      thread_id=2 exec_time=0 error_code=0
SET TIMESTAMP=1466150420/*!*/;
SET @@session.pseudo_thread_id=2/*!*/;
SET @@session.foreign_key_checks=1, @@session.sql_auto_is_null=0,
@@session.unique_checks=1, @@session.autocommit=1/*!*/;
SET @@session.sql_mode=1436549152/*!*/;
SET @@session.auto_increment_increment=1, @@session.auto_increment_offset=1/*!
*/;
/*!\C utf8 *//*!*/;
SET
@@session.character_set_client=33,@@session.collation_connection=33,@@session.
collation_server=8/*!*/;
```

```
SET @@session.lc_time_names=0/*!*/;
SET @@session.collation_database=DEFAULT/*!*/;
create database db1
/*!*/;
# at 310
#160617 16:00:20 server id 1  end_log_pos 375 CRC32 0x60cc2f08
      Anonymous_GTID   last_committed=1 sequence_number=2
SET @@SESSION.GTID_NEXT= 'ANONYMOUS'/*!*/;
# at 375
#160617 16:00:20 server id 1  end_log_pos 488 CRC32 0xeb1f4f7d      Query
      thread_id=2 exec_time=0 error_code=0
use `db1`/*!*/;
SET TIMESTAMP=1466150420/*!*/;
create table t1 (id int,name varchar(20))
/*!*/;
# at 488
#160617 16:00:20 server id 1  end_log_pos 553 CRC32 0x84014ea9
      Anonymous_GTID   last_committed=2 sequence_number=3
SET @@SESSION.GTID_NEXT= 'ANONYMOUS'/*!*/;
# at 553
#160617 16:00:20 server id 1  end_log_pos 624 CRC32 0xbb9b29c3      Query
      thread_id=2 exec_time=0 error_code=0
SET TIMESTAMP=1466150420/*!*/;
BEGIN
/*!*/;
# at 624
#160617 16:00:20 server id 1  end_log_pos 671 CRC32 0x0d432006      Table_map:
`db1`.`t1` mapped to number 108
# at 671
#160617 16:00:20 server id 1  end_log_pos 737 CRC32 0x16745592
      Write_rows: table id 108 flags: STMT_END_F
### INSERT INTO `db1`.`t1`
### SET
###   @1=1 /* INT meta=0 nullable=1 is_null=0 */
###   @2='booboo' /* VARSTRING(20) meta=20 nullable=1 is_null=0 */
### INSERT INTO `db1`.`t1`
### SET
###   @1=2 /* INT meta=0 nullable=1 is_null=0 */
###   @2='tom' /* VARSTRING(20) meta=20 nullable=1 is_null=0 */
### INSERT INTO `db1`.`t1`
### SET
###   @1=3 /* INT meta=0 nullable=1 is_null=0 */
###   @2='mark' /* VARSTRING(20) meta=20 nullable=1 is_null=0 */
# at 737
#160617 16:00:20 server id 1  end_log_pos 768 CRC32 0xbd60b4ba      Xid = 7
COMMIT/*!*/;
# at 768
#160617 16:02:43 server id 1  end_log_pos 833 CRC32 0xe2d2c4ee
      Anonymous_GTID   last_committed=3 sequence_number=4
SET @@SESSION.GTID_NEXT= 'ANONYMOUS'/*!*/;
# at 833
#160617 16:02:43 server id 1  end_log_pos 901 CRC32 0x4eae23f0      Query
      thread_id=3 exec_time=0 error_code=0
SET TIMESTAMP=1466150563/*!*/;
```

```
BEGIN
/*!*/;
# at 901
#160617 16:02:43 server id 1  end_log_pos 948 CRC32 0x888736f3     Table_map:
`db1`.`t1` mapped to number 108
# at 948
#160617 16:02:43 server id 1  end_log_pos 1014 CRC32 0x1c8e251f
      Delete_rows: table id 108 flags: STMT_END_F
### DELETE FROM `db1`.`t1`
### WHERE
###   @1=1 /* INT meta=0 nullable=1 is_null=0 */
###   @2='booboo' /* VARSTRING(20) meta=20 nullable=1 is_null=0 */
### DELETE FROM `db1`.`t1`
### WHERE
###   @1=2 /* INT meta=0 nullable=1 is_null=0 */
###   @2='tom' /* VARSTRING(20) meta=20 nullable=1 is_null=0 */
### DELETE FROM `db1`.`t1`
### WHERE
###   @1=3 /* INT meta=0 nullable=1 is_null=0 */
###   @2='mark' /* VARSTRING(20) meta=20 nullable=1 is_null=0 */
# at 1014
#160617 16:02:43 server id 1  end_log_pos 1045 CRC32 0xfa0233c3    Xid = 16
COMMIT/*!*/;
SET @@SESSION.GTID_NEXT= 'AUTOMATIC' /* added by mysqlbinlog */ /*!*/;
DELIMITER ;
# End of log file
/*!50003 SET COMPLETION_TYPE=@OLD_COMPLETION_TYPE*/;
/*!50530 SET @@SESSION.PSEUDO_SLAVE_MODE=0*/;
```

1.5 恢复数据

假设目前的备份策略是：每天早上 8 点全备份，打开了二进制日志，格式为 row。
现在，有人误操作将 t1 表的内容全部删除了，悲剧的是之前的全备份文件都没了，这个时候怎么恢复数据呢？
思考：如果格式是 *statment* 格式，在以上情况下，数据还能恢复吗？试想一下，*statment* 格式中记录的是 *sql* 语句，就是【*delete from db1.t1;*】。如果要恢复难度跟数据库的二进制日志内容的复杂度有关了。总的来说，恢复起来比较困难的。

1.5.1 截取需要的信息

我们可以看到在 row 格式下，删除的所有数据都被记录了下来。

```
[root@mastera0 mysql-binlot]# mysqlbinlog mastera.000001 -v -v --base64-
output=DECODE-ROWS|sed -n '/# at 948/,/COMMIT/p'
# at 948
#160617 16:02:43 server id 1  end_log_pos 1014 CRC32 0x1c8e251f
      Delete_rows: table id 108 flags: STMT_END_F
### DELETE FROM `db1`.`t1`
### WHERE
###   @1=1 /* INT meta=0 nullable=1 is_null=0 */
###   @2='booboo' /* VARSTRING(20) meta=20 nullable=1 is_null=0 */
### DELETE FROM `db1`.`t1`
### WHERE
###   @1=2 /* INT meta=0 nullable=1 is_null=0 */
###   @2='tom' /* VARSTRING(20) meta=20 nullable=1 is_null=0 */
### DELETE FROM `db1`.`t1`
### WHERE
###   @1=3 /* INT meta=0 nullable=1 is_null=0 */
###   @2='mark' /* VARSTRING(20) meta=20 nullable=1 is_null=0 */
# at 1014
#160617 16:02:43 server id 1  end_log_pos 1045 CRC32 0xfa0233c3     Xid = 16
COMMIT/*!*/;
```

1.5.2 恢复数据

在 row 格式下，第几列表示成“@n”，其中 n 是数字，比如“@1”代表第一列
这个时候要恢复，可以将日志内容需要的部分修改成 recovery.sql

```
[root@mastera0 ~]# vi recovery.sql
insert into `db1`.`t1`
values
(1,'booboo'),
(2,'tom'),
(3,'mark');

mysql> select * from db1.t1;
Empty set (0.03 sec)

[root@mastera0 ~]# mysql -uroot -p'(Uploo00king)' < recovery.sql
mysql: [Warning] Using a password on the command line interface can be
insecure.

mysql> select * from db1.t1;
+------+--------+
| id   | name   |
+------+--------+
|    1 | booboo |
|    2 | tom    |
|    3 | mark   |
+------+--------+
3 rows in set (0.00 sec)
```

这里比较简单就直接手动修改，在真实环境中肯定需要写脚本——见附录 1。

情景 2：数据更新了又想恢复，怎么做？

2.1 数据更新

将 db1.t1 表中的 name 列不小心都改为 dabao，本来是只想改第一行的，结果忘记加 where id=1；为了日志的易读性，这里刷新一个新日志来记录。

```
mysql> flush logs;
Query OK, 0 rows affected (0.60 sec)
mysql> select * from db1.t1;
+------+--------+
| id   | name   |
+------+--------+
|    1 | booboo |
|    2 | tom    |
|    3 | mark   |
+------+--------+
3 rows in set (0.00 sec)

mysql> update db1.t1 set name='dabao';
Query OK, 3 rows affected (0.52 sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql> select * from db1.t1;
+------+-------+
| id   | name  |
+------+-------+
|    1 | dabao |
|    2 | dabao |
|    3 | dabao |
+------+-------+
3 rows in set (0.00 sec)
```

看到第二列 name 已经全部更新为 dabao 了。

2.2 查看日志

row 日志中不仅记录了修改后的数据，还记录了修改前的数据。

```
[root@mastera0 mysql-binlog]# mysqlbinlog mastera.000003 -v -v|sed -n
'/UPDATE/,/COMMIT/p'
### UPDATE `db1`.`t1`
### WHERE
###   @1=1 /* INT meta=0 nullable=1 is_null=0 */
###   @2='booboo' /* VARSTRING(20) meta=20 nullable=1 is_null=0 */
### SET
###   @1=1 /* INT meta=0 nullable=1 is_null=0 */
###   @2='dabao' /* VARSTRING(20) meta=20 nullable=1 is_null=0 */
### UPDATE `db1`.`t1`
### WHERE
###   @1=2 /* INT meta=0 nullable=1 is_null=0 */
###   @2='tom' /* VARSTRING(20) meta=20 nullable=1 is_null=0 */
### SET
###   @1=2 /* INT meta=0 nullable=1 is_null=0 */
###   @2='dabao' /* VARSTRING(20) meta=20 nullable=1 is_null=0 */
### UPDATE `db1`.`t1`
### WHERE
###   @1=3 /* INT meta=0 nullable=1 is_null=0 */
###   @2='mark' /* VARSTRING(20) meta=20 nullable=1 is_null=0 */
### SET
###   @1=3 /* INT meta=0 nullable=1 is_null=0 */
###   @2='dabao' /* VARSTRING(20) meta=20 nullable=1 is_null=0 */
# at 434
#160620 12:48:49 server id 1  end_log_pos 465 CRC32 0x1841a0ec      Xid = 34
COMMIT/*!*/;
```

2.3 恢复数据

根据日志生成 recovery.2.sql，用 vi 编辑器
:%s/###/ /g
:%s/@1/id/g
:%s/@2/name/g
:%s/SET/SEET/g
:%s/WHERE/SET/g
:%s/SEET/WHERE/g
删除不需要的行，添加分号

```
[root@mastera0 ~]# cat recovery.2.sql
  UPDATE `db1`.`t1`
  SET
    name='booboo' /* VARSTRING(20) meta=20 nullable=1 is_null=0 */
  WHERE
    id=1 /* INT meta=0 nullable=1 is_null=0 */;
  UPDATE `db1`.`t1`
  SET
    name='tom' /* VARSTRING(20) meta=20 nullable=1 is_null=0 */
  WHERE
    id=2 /* INT meta=0 nullable=1 is_null=0 */;
  UPDATE `db1`.`t1`
  SET
    name='mark' /* VARSTRING(20) meta=20 nullable=1 is_null=0 */
  WHERE
    id=3 /* INT meta=0 nullable=1 is_null=0 */;

[root@mastera0 ~]# mysql -uroot -p'(UplooOOking)' < recovery.2.sql
mysql: [Warning] Using a password on the command line interface can be
insecure.

mysql> select * from db1.t1;
+------+--------+
| id   | name   |
+------+--------+
|    1 | booboo |
|    2 | tom    |
|    3 | mark   |
+------+--------+
3 rows in set (0.00 sec)
```

这里数据量少，用 vi 编辑器很快就能修改好，如果数据量大，建议自己写一个脚本来实现——附录 1。

附录 1——mysql-binlog-row-recovery