22.Mysql-日志管理（工具日志）√

1. general log 普通日志

1.1 作用

> 一般不会开启开功能，因为log的量会非常庞大。但个别情况下可能会临时的开一会儿general log以供调试使用 操作审计采集信息。

1.2 查询及配置

Bash | Copy

```
show variables like 'general_log'; -- 查看日志是否开启
show variables like 'general_log_file'; -- 看看日志文件保存位置
show variables like 'log_output'; -- 看看日志输出类型 table或file
set global general_log=on; -- 开启日志功能
set global general_log_file='/data/logs/general.log'; -- 设置日志文件保存位置
set global log_output='table'; -- 设置输出类型为 table
set global log_output='file'; -- 设置输出类型为 file
set global log_output='table,file'; -- 设置多位置保存
```

```
mysql> show variables like '%general_log%';
+------------------+---------------------------+
| Variable_name    | Value                     |
+------------------+---------------------------+
| general_log      | OFF                       |
| general_log_file | /data/3306/data/db01.log |
+------------------+---------------------------+
2 rows in set (0.00 sec)
```

1.3 使用

> 检测mysqldump备份过程。

2.error log 错误日志

2.1 作用

> 记录数据库启动以来，状态，报错，警告等（默认是开启）

2.2 查询及配置

```
log_error ./db01.err 当前目录指的是数据目录下
log_error_suppression_list
log_error_verbosity 显示信息的版本（一般设置为最高3） set global log_error_verbosity=3;
```

```
mysql> show variables like '%log_error%';
+---------------------------+----------------------------------------+
| Variable_name             | Value                                  |
+---------------------------+----------------------------------------+
| binlog_error_action       | ABORT_SERVER                           |
| log_error                 | ./db01.err                             |
| log_error_services        | log_filter_internal; log_sink_internal |
| log_error_suppression_list |                                        |
| log_error_verbosity       | 2                                      |
+---------------------------+----------------------------------------+
5 rows in set (0.00 sec)
```

2.3 使用

> DBA日常工作中的巡检，重点关注日志文件中【error】的上下文

3.binary log 二进制日志（SQL层日志）

3.1 作用

> 以二进制格式，记录所有数据库修改类的操作日志 （对于DML语句只记录已提交的事务的变化）。
> 1.数据备份 2.主从复制环境 都依赖二进制日志
> 8.0之后，默认自动开启。

3.2查看与配置

```
8.0版本自动开启
log_bin on （核心配置）
log_bin_name /data/3306/data/binlog --文件名的前缀
log_bin_index /data/3306/data/binlog --索引文件

基础参数查看:
开关: [(none)]>select @@log_bin;
日志路径及名字 ： [(none)]>select @@log_bin_basename;
服务ID号: [(none)]>select @@server_id;
二进制日志格式: [(none)]>select @@binlog_format; 二进制日志只记录修改类的操作日志。

双一标准:（可以解决主从延时的问题）
select  @@innodb_flush_log_at_trx_commit=1/2;
select @@sync_binlog=1;
```

```
mysql> show variables like '%log_bin%';
+---------------------------------+-------------------------------+
| Variable_name                   | Value                         |
+---------------------------------+-------------------------------+
| log_bin                         | ON                            |
| log_bin_basename                | /data/3306/data/binlog        |
| log_bin_index                   | /data/3306/data/binlog.index  |
| log_bin_trust_function_creators | OFF                           |
| log_bin_use_v1_row_events       | OFF                           |
| sql_log_bin                     | ON                            |
+---------------------------------+-------------------------------+
6 rows in set (0.00 sec)
```

3.3 配置

1.创建存放日志的目录路径

mkdir –p /data/3306/binlog

chown –R mysql. /data

2.编辑mysql配置文件

vim /etc/my.cnf

【mysqld】

server_id=51 (5.7版本之后必须设定)

log_bin=/data/3306/binlog/mysql-bin

3.重启数据库生效

/etc/init.d/mysqld restart

4.查看会发现在我设定的目录有了日志文件

```
[root@db01 binlog]# pwd
/data/3306/binlog
[root@db01 binlog]# ll
总用量 12
-rw-r----- 1 mysql mysql 179 3月  16 13:31 mysql-bin.000001
-rw-r----- 1 mysql mysql 156 3月  16 13:31 mysql-bin.000002
-rw-r----- 1 mysql mysql  70 3月  16 13:31 mysql-bin.index
[root@db01 binlog]#
```

3.4 binlog的存储内容与记录格式

> binlog是以事件（events）作为最小单元，多个事件组合成一次事务性流程操作，进行记录修改类的操作。
> 针对: DDL DCL,事件直接以Statement(语句模式)
> 针对: DML语句 (update delete insert),记录格式受binlog_format参数控制 常用ROW格式
> 格式种类:
> statement（SBR）：语句模式.做啥记录啥.
> row （RBR）：行模式.记录行变化.，日志量比较大
> mixed(MBR)：混合模式. 大几率是statement（不常用）
> 区别:
> statement：日志量少.可读性强,随机函数类操作时,记录有可能错误数据.
> ROW ：日志量大.可读性弱.数据记录准确.
> 生产建议 ROW 对DML语句有效

3.5 binlog的使用

Bash | Copy

```
mysql> show binary logs; 查看所有binlog日志
mysql> flush logs; 刷新生成新的日志开始记录 （重启一次数据库再登陆也会重新生成新的日志文件开始记录）
```

```
mysql> show binary logs;
+------------------+-----------+-----------+
| Log_name         | File_size | Encrypted |
+------------------+-----------+-----------+
| mysql-bin.000001 |       179 | No        |
| mysql-bin.000002 |       156 | No        |
+------------------+-----------+-----------+
2 rows in set (0.01 sec)
```

```
show master status; 查看正在使用的binlog日志文件
```

```
mysql> show master status;
+------------------+----------+--------------+------------------+-------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+------------------+----------+--------------+------------------+-------------------+
| mysql-bin.000002 |      156 |              |                  |                   |
+------------------+----------+--------------+------------------+-------------------+
1 row in set (0.00 sec)
```

3.6 查看binlog日志内容

方法一：

```
show binlog events in '日志名称';
例子: show binlog events in 'mysql-bin.000002';
```

每个binlog日志文件的前156字节都是文件的头格式

```
mysql> show binlog events in 'mysql-bin.000002';
+------------------+-----+----------------+-----------+-------------+---------------------------------+
| Log_name         | Pos | Event_type     | Server_id | End_log_pos | Info                            |
+------------------+-----+----------------+-----------+-------------+---------------------------------+
| mysql-bin.000002 |   4 | Format_desc    |        51 |         125 | Server ver: 8.0.20, Binlog ver: 4 |
| mysql-bin.000002 | 125 | Previous_gtids |        51 |         156 |                                 |
+------------------+-----+----------------+-----------+-------------+---------------------------------+
2 rows in set (0.00 sec)
```

方法二： mysqlbinlog(解析二进制)

mysql安装时自带的工具 mysqlbinlog(解析二进制)

> 格式 mysqlbinlog + mysql-bin.000002 会把解析出来的打印在面板上 -d (查看指定的数据库)
> at 与 at 之间为一个事件

```
[root@db01 binlog]# mysqlbinlog mysql-bin.000002
/*!50530 SET @@SESSION.PSEUDO_SLAVE_MODE=1*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
DELIMITER /*!*/;
# at 4
#210316 13:31:03 server id 51  end_log_pos 125 CRC32 0xf0bbd854         Start: binlog v 4, server v 8.0.20 created 2
10316 13:31:03 at startup
# Warning: this binlog is either in use or was not closed properly.
ROLLBACK/*!*/;
BINLOG '
l0JQYA8zAAAAeQAAAH0AAAABAAQAOC4wLjIwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAACXQlBgEwANAAgAAAAABAAEAAAAYQAEGggAAAAICAgCAAAACgoKKioAEjQA
CigBVNi78A==
'/*!*/;
# at 125
#210316 13:31:03 server id 51  end_log_pos 156 CRC32 0x7fb088aa         Previous-GTIDs
# [empty]
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*/;
[root@db01 binlog]#
```

方法三：

翻译row格式的命令（翻译出来的是伪代码不能拿来直接用）

Bash | Copy

```
mysqlbinlog --base64-output=decode-rows -vv '翻译的row格式内容' mysql-bin.00000x
```

方法四：根据时间进行查看

Bash | Copy

```
mysqlbinlog --start-datetime='开始时间点' --stop-datetime='结束时间点' /data/binlog/mysql-bin.00000x(指定那个日志文件
```

3.7 binlog日志截取与恢复（基于position号）

1.截取

a.分析日志 确认起点和终点

b.截取日志通过mysqlbinlog工具

```
#截取日志语法格式
mysqlbinlog --start-position=xx --stop-position=yy /data/3306/binlog/mysql-bin.00000x >/tmp/bin.sql
     命令          起点               终点               日志文件                          导出到文件中
```

2.恢复

c. 恢复日志

```
mysql> set sql_log_bin=0 临时关闭对当前终端的二进制日志的记录
mysql> source /tmp/bin.sql 导入截取的信息
set sql_log_bin=1        恢复后开启对二进制日志的记录
```

3.模拟使用

①创建库

```
建库
create database oldguo;
建表
create table oldguo.t1 (id int);
```

②模拟删除操作

```
drop database oldguo;
```

③截取日志分析

```
mysql> show binlog events in 'mysql-bin.000002';
+------------------+-----+----------------+-----------+-------------+-----------------------------------
--------+
| Log_name         | Pos | Event_type     | Server_id | End_log_pos | Info                                |
+------------------+-----+----------------+-----------+-------------+-----------------------------------
--------+
| mysql-bin.000002 |   4 | Format_desc    |        51 |         125 | Server ver: 8.0.20, Binlog ver: 4   |
| mysql-bin.000002 | 125 | Previous_gtids |        51 |         156 |                                     |
| mysql-bin.000002 | 156 | Anonymous_Gtid |        51 |         233 | SET @@SESSION.GTID_NEXT= 'ANONYMOUS'  |
| mysql-bin.000002 | 233 | Query          |        51 |         347 | create database oldguo /* xid=6 */    |
| mysql-bin.000002 | 347 | Anonymous_Gtid |        51 |         424 | SET @@SESSION.GTID_NEXT= 'ANONYMOUS'  |
| mysql-bin.000002 | 424 | Query          |        51 |         541 | create table oldguo.t1 (id int) /* xid=8 */ |
```

```
| mysql-bin.000002 | 541 | Anonymous_Gtid |        51 |         618 | SET @@SESSION.GTID_NEXT= 'ANONYMOUS'         |
| mysql-bin.000002 | 618 | Query          |        51 |         728 | drop database oldguo /* xid=18 */            |
+------------------+-----+----------------+-----------+-------------+----------------------------------------------
--------+
8 rows in set (0.00 sec)
```

Bash | Copy

```
mysqlbinlog --start-position=223 --stop-position=541 /data/3306/binlog/mysql-bin.000002 >/tmp/bin.sql
```

④恢复

Bash | Copy

```
mysql> set sql_log_bin=0  恢复前关闭binlog的记录，防止记录恢复操作
mysql> source /tmp/bin.sql
set sql_log_bin=1         恢复后开启binlog的记录
```

⑤ 再次查看数据已经恢复

3.8 binlog恢复的痛点

> 1.数据库运行多年?　　解决：（全备）备份恢复+binlog恢复
> 2.需要的日志跨了多个文件 （position不支持跨文件）？　　解决：5.5之前mysqlbinlog --start-datetime=name(时间点) 或者5.6版本之后GTID
> 3.binlog记录了所有库 所有表，那么如何锁定呢　　解决：binlog2sql工具可以针对表的级别，截取日志

3.9 binlog常见列子

Bash | Copy

```
1.binlog属于全局日志，日志中有其他库的操作，怎么排除掉?
mysqlbinlog -d oldboy mysql-bin.000008 > /tmp/bin.sql

2.binlog中100w个事件，怎么快速找到drop database的位置点?
mysql> pager grep "DROP"
[root@db01 ~]# mysql -e "show binlog events in 'mysql-bin.000014'" |less
[root@db01 ~]# mysql -e "show binlog events in 'mysql-bin.000014'" |grep
```

4.binlog的特性-GTID （解决跨文件痛点）

4.0 简介与特点

> GTID（ Global Transaction Identifier）全局事务标识，由主库上生成的与事务绑定的唯一标识，这个标识不仅在主库上是唯一的，在MySQL集群内也是唯一的。GTID是 MySQL 5.6 版本引入的一个有关于主从复制的重大改进，相对于之前版本基于Binlog文件+Position的主从复制，基于GTID的主从复制，数据一致性更高，主从数据复制更健壮，主从切换、故障切换不易出错，很少需要人为介入处理。

> mysql5.7版本加强 默认开启
> mysql8.0 版本默认开启

特点

> 0.幂等性: GTID会根据GTID号来检查我们操作的幂等性（是否做过重复操作）会影响到数据恢复的过程
> 1.事务提交产生GTID，GTID与事务及事务提交所在的节点绑定，GTID与事务一起写入Binlog，但是从库应用Binlog并不会生成新的GTID。
> 2.集群中的任何一个节点，根据其GTID值就可以知道哪些事务已经执行，哪些事务没有执行，如果发现某个GTID已执行，重复执行该GTID，将会被忽略，即同一个GTID只能被应用一次。
> 3.当一个连接执行一个特定GTID的事务，但是还没有提交，此时有另外一个连接也要执行相同GTID的事务，那么第二个连接的执行将会被阻塞，直到第一个事务提交或者回滚。如果第一个事务成功提交，第二个事务将会被忽略。如果第一个事务回滚，第二个事务正常执行。

5.6版本生成 5.7火起来的 5.7版本之后默认开启

4.1 参数

Bash | Copy

```
Variable_name                   | Value     |
+---------------------------------+-----------+
| binlog_gtid_simple_recovery      | ON        |
| enforce_gtid_consistency         | OFF       | 强制gtid的一致性 *
| gtid_executed                    |           |
| gtid_executed_compression_period | 1000      |
| gtid_mode                        | OFF       | gtid 开关*
| gtid_next                        | AUTOMATIC |
| gtid_owned                       |           |
| gtid_purged                      |           |
| session_track_gtids              | OFF       |
```

4.2 配置GTID（默认是匿名模式，手动开启gtid后，gtid号才开始产生）

①编辑配置文件

Bash | Copy

```
vim /etc/my.cnf
 [mysqld]
enforce_gtid_consistency=on
gtid_mode=on
log-slave-updates=1 主从环境需要设置（针对从库   ）
```

②重启数据库生效

Bash | Copy

```
/etc/init.d/mysqld restart
```

③查看gtid（针对DDL语句一个操作会递增gtid号，针对DML语句一个已提交的事务会递增gtid号）

```
mysql> show master status;
+------------------+----------+--------------+------------------+-------------------------------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set                         |
+------------------+----------+--------------+------------------+-------------------------------------------+
| mysql-bin.000003 |     1062 |              |                  | 1bc7779a-48f0-11eb-bdae-000c29ef43a9:1-4 |
+------------------+----------+--------------+------------------+-------------------------------------------+
1 row in set (0.00 sec)
```

4.3 基于GTID的数据恢复

①模拟删除库操作

```
drop database cry;
```

②分析日志中的gtid，确定正在使用的日志文件是mysql-bin.000003

```
mysql> show master status;
+------------------+----------+--------------+------------------+-------------------------------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set                         |
+------------------+----------+--------------+------------------+-------------------------------------------+
| mysql-bin.000003 |     1240 |              |                  | 1bc7779a-48f0-11eb-bdae-000c29ef43a9:1-5 |
+------------------+----------+--------------+------------------+-------------------------------------------+
```

③查看mysql-bin.000003日志的事件信息

```
mysql> show binlog events in 'mysql-bin.000003';
+------------------+------+----------------+-----------+-------------+-----------------------------------------
| Log_name         | Pos  | Event_type     | Server_id | End_log_pos | Info
+------------------+------+----------------+-----------+-------------+-----------------------------------------
| mysql-bin.000003 |    4 | Format_desc    |        51 |         125 | Server ver: 8.0.20, Binlog ver: 4
| mysql-bin.000003 |  125 | Previous_gtids |        51 |         156 |
| mysql-bin.000003 |  156 | Gtid           |        51 |         233 | SET @@SESSION.GTID_NEXT= '1bc7779a-48f0
| mysql-bin.000003 |  233 | Query          |        51 |         338 | create database cry /* xid=6 */
| mysql-bin.000003 |  338 | Gtid           |        51 |         415 | SET @@SESSION.GTID_NEXT= '1bc7779a-48f0
| mysql-bin.000003 |  415 | Query          |        51 |         526 | create table cry.t1 (id int) /* xid=13
| mysql-bin.000003 |  526 | Gtid           |        51 |         605 | SET @@SESSION.GTID_NEXT= '1bc7779a-48f0
| mysql-bin.000003 |  605 | Query          |        51 |         676 | BEGIN
| mysql-bin.000003 |  676 | Table_map      |        51 |         723 | table_id: 81 (cry.t1)
| mysql-bin.000003 |  723 | Write_rows     |        51 |         763 | table_id: 81 flags: STMT_END_F
| mysql-bin.000003 |  763 | Xid            |        51 |         794 | COMMIT /* xid=16 */
| mysql-bin.000003 |  794 | Gtid           |        51 |         873 | SET @@SESSION.GTID_NEXT= '1bc7779a-48f0
| mysql-bin.000003 |  873 | Query          |        51 |         944 | BEGIN
| mysql-bin.000003 |  944 | Table_map      |        51 |         991 | table_id: 81 (cry.t1)
| mysql-bin.000003 |  991 | Write_rows     |        51 |        1031 | table_id: 81 flags: STMT_END_F
| mysql-bin.000003 | 1031 | Xid            |        51 |        1062 | COMMIT /* xid=17 */
| mysql-bin.000003 | 1062 | Gtid           |        51 |        1139 | SET @@SESSION.GTID_NEXT= '1bc7779a-48f0
| mysql-bin.000003 | 1139 | Query          |        51 |        1240 | drop database cry /* xid=19 */
+------------------+------+----------------+-----------+-------------+-----------------------------------------
```

④截取日志

GTID的号向上看

```
   GTID截取语法格式
]# mysqlbinlog    --skip-gtids       --include-gtids='1bc7779a-48f0-11eb-bdae-000c29ef43a9:1-4' /data/3306/bin
      命令        解决gtid的幂等性        gtid选项              gtid号码从1到4                                  指定截取那个
```

⑤恢复数据

```
mysql> set sql_log_bin=0  恢复前关闭binlog的记录，防止记录恢复操作
mysql> source /tmp/gtid.sql  恢复数（据导入截取的日志记录的sql操作到数据库中）
set sql_log_bin=1         恢复开启binlog的记录
```

4.4 gtid截取日志跳过语法格式

```
跳过1-5中间的4
mysqlbinlog    --skip-gtids --include-gtids='1bc7779a-48f0-11eb-bdae-000c29ef43a9:1-5' --exculde-gtids='1bc
```

5.binlog2sql(可以针对解析单表数据)

5.0 作用

仅限对DML语句，且row格式

> 1.将binlog日志中的row格式转化为sql伪代码，辅助我们查看分析日志
> 2.截取误操作进行反操作 delent---->insert,实现闪回功能

5.1使用

①安装（python开发）

#软件包

binlog2sql-master.zip（15 kB）

#依赖包

requirements.txt（0 kB）

```
上传并进行解压软件包
unzip binlog2sql-master.zip
因为是python开发的软件，所以需要安装python环境
yum -y install python3
解析py文件
pip3 install -r requirements.txt -i https://mirrors.aliyun.com/pypi/simple/
```

②创建模拟环境

```
建表
create table oldguo.t2 (id int not null primary key auto_increment, name varchar(32) not null );
插入数据
insert into oldguo.t2 values(1,'a'),(2,'b'),(3,'c');
```

③确定正在使用那个目录

```
mysql> show master status;
+------------------+----------+--------------+------------------+-------------------------------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set                         |
+------------------+----------+--------------+------------------+-------------------------------------------+
| mysql-bin.000003 |     1795 |              |                  | 1bc7779a-48f0-11eb-bdae-000c29ef43a9:1-7 |
+------------------+----------+--------------+------------------+-------------------------------------------+
```

④binlog2sql 使用步骤

第一步 切入到工具目录中

```
[root@db01 binlog2sql-master]# pwd
/opt/binlog2sql-master
[root@db01 binlog2sql-master]# ls
binlog2sql.py  binlog2sql_util.py  binlog2sql_util.pyc  __init__.py  README.md
[root@db01 binlog2sql-master]#
```

第二步 通过python命令调用binlog2sql

```
]# python3 binlog2sql.py -h 10.0.0.51 -p3306     -uremote -p123     -d oldguo -t t2    --start-file='mysql-bin.
   py命令    py脚本        指定地址      指定端口    指定用户 指定密码   指定数据库   指定表   指定日志文件
INSERT INTO `oldguo`.`t2`(`id`, `name`) VALUES (1, 'a'); #start 1579 end 1764 time 2021-03-16 20:23:47 gtid
INSERT INTO `oldguo`.`t2`(`id`, `name`) VALUES (2, 'b'); #start 1579 end 1764 time 2021-03-16 20:23:47 gtid
INSERT INTO `oldguo`.`t2`(`id`, `name`) VALUES (3, 'c'); #start 1579 end 1764 time 2021-03-16 20:23:47 gtid
```

5.2 binlog2sql数据恢复（闪回）

①模拟误删除操作

```
mysql> delete from oldguo.t2 where id=3;
```

②binlog2sql解析日志,做的delete操作也解析出来了

```
[root@db01 binlog2sql-master]# python3 binlog2sql.py  -h 10.0.0.51 -P3306 -uremote -p123 -d oldguo -t t2 --sta
INSERT INTO `oldguo`.`t2`(`id`, `name`) VALUES (1, 'a'); #start 1579 end 1764 time 2021-03-16 20:23:47 gtid
INSERT INTO `oldguo`.`t2`(`id`, `name`) VALUES (2, 'b'); #start 1579 end 1764 time 2021-03-16 20:23:47 gtid
INSERT INTO `oldguo`.`t2`(`id`, `name`) VALUES (3, 'c'); #start 1579 end 1764 time 2021-03-16 20:23:47 gtid
DELETE FROM `oldguo`.`t2` WHERE `id`=3 AND `name`='c' LIMIT 1; #start 1874 end 2045 time 2021-03-16 21:00:58 g
```

③ binlog2sql 选项 --sql-type=DML (delete update insert) 过滤指定sql语句

```
[root@db01 binlog2sql-master]# python3 binlog2sql.py  -h 10.0.0.51 -P3306 -uremote -p123 -d oldguo -t t2 --sta
DELETE FROM `oldguo`.`t2` WHERE `id`=3 AND `name`='c' LIMIT 1; #start 1874 end 2045 time 2021-03-16 21:00:58 g
```

④恢复数据，实现闪退 使用 -B 进行反操作

```
[root@db01 binlog2sql-master]# python3 binlog2sql.py -B -h 10.0.0.51 -P3306 -uremote -p123 -d oldguo -t t2 --s
INSERT INTO `oldguo`.`t2`(`id`, `name`) VALUES (3, 'c'); #start 1874 end 2045 time 2021-03-16 21:00:58 gtid
```

⑤将反操作后的sql语句，复制粘贴到mysql中运行就实现了数据恢复

如果数据过多可以导入到文件中(>/tmp/binlog2sql.sql)，再在mysql中加载 （source /tmp/binlog2sql.sql）

```
mysql> INSERT INTO `oldguo`.`t2`(`id`, `name`) VALUES (3, 'c');
Query OK, 1 row affected (0.01 sec)

mysql> select * from oldguo.t2;
+----+------+
| id | name |
+----+------+
|  1 | a    |
|  2 | b    |
|  3 | c    |
+----+------+
3 rows in set (0.00 sec)
```

5.3 应用场景:3000万数据，误删10行数据，怎么恢复?

binlog2sql工具进行转移sql语句+position进行定位

```
[root@db01 binlog2sql]# python3 binlog2sql.py  -h 10.0.0.51 -P3306 -uroot -p123 -d test -t t1  --start-file='my
[root@db01 binlog2sql]# python3 binlog2sql.py  -h 10.0.0.51 -P3306 -uroot -p123 -d test -t t1  --start-file='my
```

6.清理日志与日志滚动

6.1 自动清理日志

```
mysql5.7版本 以天为单位
show variables like '%expire%';
expire_logs_days 0
自动清理时间,是要按照全备周期+1
set global expire_logs_days=8;
永久生效:
my.cnf expire_logs_days=15;
企业建议,至少保留两个全备周期+1的binlog
mysql8.0版本 以秒为单位
binlog_expire_logs_seconds=2592000
```

6.2 手动清理日志

```
  手动清理binlog日志, purge master logs
  删除在“指定日期前”或“指定日志前”的所有二进制binlog日志文件
  PURGE {MASTER | BINARY} LOGS TO 'log_name'
  PURGE {MASTER | BINARY} LOGS BEFORE 'date'

实例:
PURGE MASTER LOGS TO 'MySQL-bin.010';
PURGE MASTER LOGS BEFORE '2003-04-02 22:46:26';
    ##BEFORE变量的date格式为'YYYY-MM-DD hh:mm:ss'。MASTER/BINARY是同义词。


stat指令查看文件信息
[root@db01 tmp]# stat g1.sql
  文件: "g1.sql"
  大小: 49781950   块: 97232      IO 块: 4096   普通文件
设备: 803h/2051d Inode: 33580209    硬链接: 1
权限: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
最近访问: 2021-06-08 14:15:02.753178692 +0800
最近更改: 2021-06-08 14:17:32.020170305 +0800
最近改动: 2021-06-08 14:17:32.020170305 +0800
创建时间: -


注意:不要手工 rm binlog文件

1. my.cnf binlog关闭掉,启动数据库
2. 把数据库关闭,开启binlog,启动数据库
删除所有binlog,并从000001开始重新记录日志
```

6.3 日志滚动

```
更新产生新的日志进行记录，更新前的日志变成历史日志
手动滚动
mysql> flush logs;
自动滚动
1.重启数据库
2.日志文件达到1G大小
关于日志文件大小的参数是 max_binlog_size,默认大小1G
| max_binlog_size | 1073741824
3.备份时,加入参数也可以自动滚动
```

7.slow log（对记录的慢语句进行优化）

7.1 作用

> 记录慢SQL语句的日志,定位低效SQL语句的工具日志

7.2参数配置

```
#slow log的开关默认没有开启slow_query_log=1;    查看 select @@slow_query_log;
#文件位置及名字              slow_query_log_file=/data/3306/data/db01-slow.log;
#设定慢查询的时间限定:      long_query_time=0.01;
#没走索引的语句也记录:      log_queries_not_using_indexes=1;
```

7.3 模拟慢语句使用

a.配置慢语句（在线设置）

```
mysql>set global slow_query_log=1;
mysql>set global long_query_time=0.01;
mysql>set global log_queries_not_using_indexes=1;
```

b.模拟生成慢语句

```
mysql>select * from t100w limit 10;
mysql>select * from t100w limit 100;
mysql>select * from t100w limit 1000;
mysql>select * from t100w where k1='aa' limit 10;
mysql>select * from t100w where k1='aa' limit 100;
mysql>select * from t100w where k1='aa' limit 1000;
```

c.查看slow log日志文件

文件默认在数据目录下 db01-slow.log（文本文件可以使用vim），以时间为单位来记录

```
/usr/local/mysql/bin/mysqld, Version: 8.0.20 (MySQL Community Server - GPL). started with:
Tcp port: 3306  Unix socket: /tmp/mysql.sock
Time                 Id Command    Argument
# Time: 2021-03-16T13:46:10.508988Z
# User@Host: root[root] @ localhost []  Id:    33
# Query_time: 0.000183  Lock_time: 0.000088 Rows_sent: 10  Rows_examined: 10
use test;
SET timestamp=1615902370;
select * from t100w limit 10;
# Time: 2021-03-16T13:46:12.367487Z
# User@Host: root[root] @ localhost []  Id:    33
# Query_time: 0.000503  Lock_time: 0.000150 Rows_sent: 100  Rows_examined: 100
SET timestamp=1615902372;
select * from t100w limit 100;
# Time: 2021-03-16T13:46:13.996912Z
# User@Host: root[root] @ localhost []  Id:    33
# Query_time: 0.002082  Lock_time: 0.000182 Rows_sent: 1000  Rows_examined: 1000
SET timestamp=1615902373;
select * from t100w limit 1000;
# Time: 2021-03-16T13:46:22.279709Z
# User@Host: root[root] @ localhost []  Id:    33
# Query_time: 0.004882  Lock_time: 0.000208 Rows_sent: 10  Rows_examined: 9274
SET timestamp=1615902382;
select * from t100w where k1='aa' limit 10;
# Time: 2021-03-16T13:46:24.013150Z
# User@Host: root[root] @ localhost []  Id:    33
# Query_time: 0.039734  Lock_time: 0.000166 Rows_sent: 100  Rows_examined: 76446
SET timestamp=1615902383;
select * from t100w where k1='aa' limit 100;
```

d.分析日志文件，进行 合并，排序的操作

mysql提供了mysqldumpslow工具，辅助我们进行对慢语句的优化

Bash | Copy

```
mysqldumpslow工具语法格式

mysqldumpslow      -s       c        -t   3      文件名
                  solt排序  按照次数  top排名
```

```
[root@db01 data]# mysqldumpslow -s c -t 3 db01-slow.log

Reading mysql slow query log from db01-slow.log
Count: 9  Time=0.17s (1s)  Lock=0.00s (0s)  Rows=370.0 (3330), root[root]@localhost
  select * from t100w where k1='S' limit N

Count: 6  Time=0.00s (0s)  Lock=0.00s (0s)  Rows=370.0 (2220), root[root]@localhost
  select * from t100w limit N

Died at /usr/local/mysql/bin/mysqldumpslow line 162, <> chunk 15.
[root@db01 data]#
```

e.针对排名靠前的慢语句进行优化

Bash | Copy

```
1  select * from t100w where k1='S' limit N
```

第一步 查看慢语句的执行计划（desc）关注 索引 执行计划 锁等问题
第二步 对这条语句进行优化，增加索引 alter table t100w add index idx(k1);
分析慢语句也可以使用pt工具（pt-query-digest）

7.4 PT工具的应用（percona-toolkit）

7.4.1 PT安装与命令参数

```
# 安装工具：
[root@master ~]# yum install -y percona-toolkit-3.1.0-2.el7.x86_64.rpm
#命令语法：
pt-query-digest [OPTIONS] [FILES] [DSN]
--create-review-table 当使用--review参数把分析结果输出到表中时，如果没有表就自动创
建。
--create-history-table 当使用--history参数把分析结果输出到表中时，如果没有表就自动
创建。
--filter 对输入的慢查询按指定的字符串进行匹配过滤后再进行分析
--limit 限制输出结果百分比或数量，默认值是20,即将最慢的20条语句输出，如果是50%则按总
响应时间占比从大到小排序，输出到总和达到50%位置截止。
--host mysql服务器地址
--user mysql用户名
--password mysql用户密码
--history 将分析结果保存到表中，分析结果比较详细，下次再使用--history时，如果存在相同的
语句，且查询所在的时间区间和历史表中的不同，则会记录到数据表中，可以通过查询同一CHECKSUM
来比较某类型查询的历史变化。
```

--review 将分析结果保存到表中，这个分析只是对查询条件进行参数化，一个类型的查询一条记录，比较简单。当下次使用--review时，如果存在相同的语句分析，就不会记录到数据表中。
--output 分析结果输出类型，值可以是report(标准分析报告)、slowlog(Mysql slow log)、json、json-anon，一般使用report，以便于阅读。
--since 从什么时间开始分析，值为字符串，可以是指定的某个”yyyy-mm-dd [hh:mm:ss]”格式的时间点，也可以是简单的一个时间值：s(秒)、h(小时)、m(分钟)、d(天)，如12h就表示从12小时前开始统计。
--until 截止时间，配合—since可以分析一段时间内的慢查询。

7.4.2 PT结果说明

Overall：总共有多少条查询
Time range：查询执行的时间范围
unique：唯一查询数量，即对查询条件进行参数化以后，总共有多少个不同的查询
total：总计 min：最小 max：最大 avg：平均
95%：把所有值从小到大排列，位置位于95%的那个数，这个数一般最具有参考价值
median：中位数，把所有值从小到大排列，位置位于中间那个数

7.4.3

Response: 总的响应时间。
time: 该查询在本次分析中总的时间占比。
calls: 执行次数，即本次分析总共有多少条这种类型的查询语句。
R/Call: 平均每次执行的响应时间。
Item : 查询对象
每部分详细统计结果
1号查询的详细统计结果，最上面的表格列出了执行次数、最大、最小、平均、95%等各项目的统计。
Databases: 库名
Users: 各个用户执行的次数（占比）
Query_time distribution : 查询时间分布, 长短体现区间占比，本例中查询集中在10ms。
Tables: 查询中涉及到的表
Explain: 示例
每部分详细统计结果
1号查询的详细统计结果，最上面的表格列出了执行次数、最大、最小、平均、95%等各项目的统计。
Databases: 库名
Users: 各个用户执行的次数（占比）
Query_time distribution : 查询时间分布, 长短体现区间占比，本例中查询集中在1s+。
Tables: 查询中涉及到的表
Explain: 执行计划

7.4.4 命令行应用实例：

用法示例
1.直接分析慢查询文件:

pt-query-digest slow.log > slow_report.log
2.分析最近12小时内的查询：
pt-query-digest --since=12h slow.log > slow_report2.log
3.分析指定时间范围内的查询：
pt-query-digest slow.log --since '2019-01-07 09:30:00' --until '2019-01-07 10:00:00'> > slow_report3.log
4.分析指含有select语句的慢查询
pt-query-digest --filter '$event->{fingerprint} =~ m/^select/i' slow.log> slow_report4.log
5.针对某个用户的慢查询
pt-query-digest --filter '($event->{user} || "") =~ m/^root/i' slow.log> slow_report5.log
6.查询所有所有的全表扫描或full join的慢查询
pt-query-digest --filter '(($event->{Full_scan} || "") eq "yes") || (($event->{Full_join} || "") eq "yes")' slow.log> slow_report6.log
7.把结果保存到query_review表
pt-query-digest --user=root -password=abc123 --review h=localhost,D=test,t=query_review --create-review-table slow.log
8.把结果保存到query_history表
pt-query-digest --user=root -password=abc123 --review h=localhost,D=test,t=query_history --create-review-table slow.log_0001
pt-query-digest --user=root -password=abc123 --review h=localhost,D=test,t=query_history --create-review-table slow.log_0002
9.通过tcpdump抓取mysql的tcp协议数据，然后再分析
tcpdump -s 65535 -x -nn -q -tttt -i any -c 1000 port 3306 > mysql.tcp.txt
pt-query-digest --type tcpdump mysql.tcp.txt> slow_report9.log
10.分析binlog
mysqlbinlog mysql-bin.000093 > mysql-bin000093.sql
pt-query-digest --type=binlog mysql-bin000093.sql > slow_report10.log
11.分析general log
pt-query-digest --type=genlog localhost.log > slow_report11.log
项目：
gui
anemometer
lepus

%E6%97%A5%E5%BF%97%E7%AE%A1%E7%90%86%EF%BC%88%E5%B7%A5%E5%85%B7%E6%97%A5%E5%BF%97%EF%BC%89%E2%88%9A%20%7C%201.%20general%20log%20%E6%99%AE%E9%