MySQL学习笔记（Day034：事物_2）

MySQL学习

一. 事物演示

1.1. Flat Transactions（平事物）

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

mysql> create table t_trx_1 (a int, b int , c int, primary key(a));
Query OK, 0 rows affected (0.13 sec)

mysql> insert into t_trx_1 values (10, 11, 12), (20,21,22);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> select * from t_trx_1;
+----+------+------+
| a  | b    | c    |
+----+------+------+
| 10 |   11 |   12 |
| 20 |   21 |   22 |
+----+------+------+
2 rows in set (0.00 sec)

mysql> commit; -- or rollback
Query OK, 0 rows affected (0.03 sec)
```

> 在 存储过程 中使用事物，需要使用的是 `start transaction` ，因为 `begin` 是存储过程中的表示代码块开始的 关键字

```
mysql> show variables like "%autocommit%";
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| autocommit    | OFF   |  -- 默认情况下 为 ON，  my.cnf中定义为off（模拟oracle中的用法）
+---------------+-------+     这样就需要显示的去commit;
1 row in set (0.00 sec)

mysql> select @@autocommit;
+--------------+
| @@autocommit |
+--------------+
|            0 |
+--------------+
1 row in set (0.00 sec)
```

1.2. Flat Transactions with Savepoints（带保存点的平事物）

```
-- 在MySQL5.6.27上测试通过，MySQL5.7.9上测试出现问题
--
-- MySQL 5.7.9
--
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into t_trx_1 values(30,31,32);
Query OK, 1 row affected (0.00 sec)

mysql> savepoint s1;
ERROR 1290 (HY000): The MySQL server is running with the ——transaction-write-set-extraction!=OFF option so it cannot execute this statement

--
-- MySQL 5.7.9
--

mysql> begin;
Query OK, 0 rows affected (0.04 sec)

mysql> insert into t_trx_1 values (30, 31, 32);
Query OK, 1 row affected (0.00 sec)

mysql> savepoint s1;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into t_trx_1 values (40, 41, 42);
Query OK, 1 row affected (0.00 sec)

mysql> savepoint s2;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from t_trx_1;
+----+------+------+
| a  | b    | c    |
+----+------+------+
| 10 |   11 |   12 |
| 20 |   21 |   22 |
| 30 |   31 |   32 |  -- s1的数据
| 40 |   41 |   42 |  -- s2的数据
+----+------+------+
4 rows in set (0.00 sec)

mysql> rollback to savepoint s1;  -- 回滚到s1的保存点，且该事物没有rollback或者commit
Query OK, 0 rows affected (0.00 sec)

mysql> select * from t_trx_1;
+----+------+------+
| a  | b    | c    |
+----+------+------+
| 10 |   11 |   12 |
| 20 |   21 |   22 |
| 30 |   31 |   32 | -- 只有s1的数据
+----+------+------+
3 rows in set (0.00 sec)

-- 如果执行rollback，则整个事物rollback
-- 如果此时执行commit，则s1之前的SQL会被提交，因为s1--s2之间的SQL已经被rollback掉了
```

> 带保存点的平事物 在写 存储过程 的时候，做逻辑判断时，可能会用到
> 当系统发生崩溃时，所有保存点将丢失，恢复时，需要从开始处进行恢复。

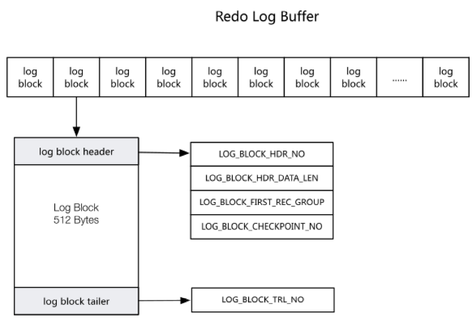
总结
以 `begin/start transaction` 开始，以 `rollback/commit` 结尾。大部分都是成功 `commit` 的，如果系统中出现大量的 `rollback` 说明系统就有问题了

二. REDO

2.1. REDO 的组成

1. redo log buffer
2. redo log file
   REDO即常说的重做日志，用来实现持久性

   - **redo log buffer**
   - `--innodb_log_buffer_size`
   - 通常 `8M` 已经足够使用
   - 由 `log block` 组成，每个log block `512字节`（不需要double write）




```
mysql> show variables like "%innodb_log_buffer%";
+------------------------+----------+
| Variable_name          | Value    |
+------------------------+----------+
| innodb_log_buffer_size | 16777216 |  -- 调整成了16M
+------------------------+----------+
1 row in set (0.00 sec)
```

   - **redo log file**
   - `--innodb_log_file_size` — 单个 innodb redo文件的大小（推荐8G，官方建议等于buffer_pool_size）
     - 之前不建议调大是因为有bug，如果调的太大，恢复速度会很慢 O(N^2)
     - 5.5版本的redo文件的 总大小（num * size）是有限制的（小于4G）

        - 5.6以后限制为512G
        - 调大以后，唯一的问题就是恢复的内容变多了
        - 5.6以后，正常关闭MySQL，然后增加/缩小该值后，MySQL自动扩展/减小该文件大小
    - --innodb_log_files_in_group
    - --innodb_log_group_home_dir
        - redo文件 和 数据文件 分开
        - 选择更快的磁盘
    - 同redo log buffer一样，由log block组成

```
[root@MyServer 5.7]> ll -h | grep ib_log  # 两个 redo logfile 文件
-rw-r-----. 1 mysql mysql 128M Feb 21 16:30 ib_logfile0
-rw-r-----. 1 mysql mysql 128M Feb 21 16:30 ib_logfile1


mysql> show variables like "%innodb_log_file%";
+---------------------------+-----------+
| Variable_name             | Value     |
+---------------------------+-----------+
| innodb_log_file_size      | 134217728 |
| innodb_log_files_in_group | 2         |  -- 默认为 2 个，注意，  size * in_group < 512G
+---------------------------+-----------+
2 rows in set (0.00 sec)

mysql> show variables like "%innodb_log_group_home_dir%";
+---------------------------+-------+
| Variable_name             | Value |
+---------------------------+-------+
| innodb_log_group_home_dir | ./    |  -- redo文件存放的位置。默认和数据文件放一起。线上建议分开放
+---------------------------+-------+
1 row in set (0.00 sec)
```

2.2. REDO Log Buffer的刷新条件

1. master thread 每秒 刷新redo的buffer到logfile
    - 5.6版本后，增加 innodb_flush_log_at_timeout 参数，可以设置刷新间隔，默认为 1
2. redo log buffer 使用量 大于1/2 时进行刷新
3. 事物提交时进行刷新
    - --innodb_flush_log_at_trx_commit = {0|1|2}

```
                    REDO Log Buffer
 +-------+-------+-------+----------+-------+-------+
 |  log  |  log  |  log  |          |  log  |  log  |
 |       |       |       |  ......  |       |       |  Size = 8M
 | Block | Block | Block |          | Block | Block |
 +-------+-------+-------+----------+-------+-------+


[1] Start Write                             [2] END
        +                                       +
        |                                       |
        v ------------+             +-------------v
        |             |             |             |
+-------> ib_logfile0 +-------------> ib_logfile1 +----------+
|       |             |             |             |          |
|       +-------------+             +-------------+          |
|                         (loop)                             |
+------------------------------------------------------------+
                     [3] then recover

# 当ib_logfile0 写满时，去写ib_logfile1
# 当ib_logfile1 也写满时，会回过头去覆盖ib_logfile0，而不是像Oracle一样去做日志归档
```

> **优点：**
> 这样做的好处是 不需要归档 ， 减少了IO操作 ；
>
> **缺点：** 如果 redo_log_file太小 ，则可能需要 等待 。因为当要 覆盖 log_file中的 log_block 时，如果该 log_block 中的脏页还没有进行刷新的话，则需要等待这个脏页进行刷新。
>
> **所以需要把 `redo log file` 设置的尽可能的大**

- **如何判断 `innodb_log_file_size` 设置小了？**

```
mysql> show engine innodb status\G
-- ----------省略部分输出----------
---
LOG
---
Log sequence number 4006324424  -- 当前内存中的LSN
Log flushed up to   4006324424
Pages flushed up to 4006324424  -- 最后刷新到磁盘的页上的，最新的LSN
Last checkpoint at  4006324415

-- 两者之差表示 redo log 还有多少没有刷新的磁盘
-- 如果该差值 接近 重做日志的总大小 的75%时，表明你的innodb_log_file_size设置小了(75%左右就强制刷新了)
```

- **innodb_flush_log_at_trx_commit**
    - **0** - 事物提交的时候并不把日志（redo log buffer）写入到磁盘（*1s 或者 大于1/2 时刷日志*）
    - **1** - 事物每次提交的时候要确保日志（redo log buffer）写入磁盘，即使宕机，也可以通过redo恢复，达到持久性的要求
    - **2** - 事物提交的时候，仅将日志（redo log buffer）写入到操作系统缓存

> 1 可以保证数据不丢失， 0 可能会丢失1秒的数据， 2 如果是mysql停止，不会丢数据，因为在缓存里面，但是当系统宕机了，在缓存里面的数据就丢失了。
> 不建议 **设置为2，** 建议设置为1

- **innodb_flush_log_at_timeout**
    - 该值设置的越大，相对性能就好一点（io操作变少），但是万一发生宕机，丢失的数据也就越多。

2.3. 组提交

1. 一次 fsync 刷新多个事物
    - 性能提高10~100倍
    - 通过sysbench工具，测试update_non_index.lua脚本（5.5有bug，性能较差）
2. InnoDB存储引擎原生支持

- **fsync**
    - O_DIRECT仅对写数据时有用 ， redo日志 是 不会 通过 O_DIRECT 方式写入到磁盘的，而是写到 文件系统的缓存 中
    - O_DIRECT 仅仅写数据到磁盘，但是数据的 元数据 没有同步，比如time、owner、size等等
        - 从数据的角度看，fsync可以将元数据同步到磁盘
    - fsync 可以将 redo日志 （redo log buffer）从文件系统的 缓存同步到磁盘

假设HDD磁盘IOPS为100（即每秒只有100次fsync），且一个事物中只有一条 insert into 的SQL时，那1秒钟内就只能插入100条数据，即TPS就只有100

> **IOPS--决定-->fsync--决定-->TPS**

假设使用 组提交 ，一次 fsync 可以刷新5个事物（假设），那在IOPS为100的情况下，也可以提交500个事物。这样性能就得到了提升（一次IO提交多个事物）

2.4 REDO日志内容

REDO日志是 物理逻辑 日志，根据页进行记录，记录的内容是逻辑的（页的变化）

```
+--------------+----------+---------+--------------+
| redo_log_type | space no | page no | redo log body |
+--------------+----------+---------+--------------+
#  redo log 类型    表空间号     页号        redo log 内容

MLOG_REC_INSERT
+------+--------+------+---------+------------+-------+---------+-----------+----------+
| type | space  | page | cur_rec | len &      | info  | origin  | mis_match | rec body |
|      |   no   |  no  | _offset | extra_info | _bits | _offset | _index    |          |
+------+--------+------+---------+------------+-------+---------+-----------+----------+

MLOG_REC_DELETE
+------+----------+---------+--------+
| type | space no | page no | offset |
+------+----------+---------+--------+
```

> **rec body 中记录的是 页的变化 ，并非SQL语句**

三. binlog

在 my.cnf 中的一个配置

```
log_bin=bin.log  #以bin开头的，在datadir中以 bin000001.log,  bin00002.log.....显示
```

> 为什么在MySQL中要搞两份日志
>
> 1. binlog 在MySQL层产生
> 2. redo 在INNODB层产生
>
> 如果MySQL只有一个INNODB，其实只有redo日志是可以的（类似Oracle），但是MySQL还有其他存储引擎，如果不使用binlog，意味着每个存储引擎都要实现一次binlog所提供的功能，诸如复制恢复等等。
>
> 一次事物提交，既要写**binlog**，又要写**redo log**

3.1. 查看当前binlog状态

```
mysql> show master status;
+------------+----------+--------------+------------------+-------------------------------------------+
| File       | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set                         |
+------------+----------+--------------+------------------+-------------------------------------------+
| bin.000056 |      194 |              |                  | c1f87a6a-98f2-11e5-b873-5254a03976fb:1-1831 |
+------------+----------+--------------+------------------+-------------------------------------------+
-- File：当前binlog写入的文件
-- Position：写入的偏移量，如果有新的数据插入，这个值就会改变（字节数）
1 row in set (0.00 sec)
```

```
mysql> show variables like "%max_binlog_size%";  -- binlog 文件的大小
+-----------------+------------+
| Variable_name   | Value      |
+-----------------+------------+
| max_binlog_size | 1073741824 |  -- 默认1G
+-----------------+------------+
1 row in set (0.00 sec)
```

> binlog 会不断的增大，默认超过1G以后（最后一个事物要执行完，可能大于1G）就会分割binlog（ 一个事物不能跨越两个binlog ）
> binlog 的 Position 表示当前该binlog的写入的 字节数

3.2. binlog的类型

1. **statement**
   - 易于理解
   - 记录SQL语句（那如果有一些不确定的SQL语句，类似UUID()的函数、limit without order by，主从会出现不一致）
2. **row** – 设置RC隔离级别，必须选这个（5.7.7默认）
   - 记录SQL语句操作的哪些行（行的变化）
   - 每张表一定要有主键（性能较高）
   - 数据一致性高，可flashback
3. **mixed**
   - 混合statement和row格式（不推荐）

```
mysql> show variables like "binlog_format";
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| binlog_format | ROW   |
+---------------+-------+
1 row in set (0.00 sec)
```

3.3 binlog events

查看当前binlog中的内容

```
mysql> show binlog events in 'bin.000056';
+------------+-----+----------------+-----------+-------------+-------------------------------------------+
| Log_name   | Pos | Event_type     | Server_id | End_log_pos | Info                                      |
+------------+-----+----------------+-----------+-------------+-------------------------------------------+
| bin.000056 |   4 | Format_desc    |      5709 |         123 | Server ver: 5.7.9-log, Binlog ver: 4      |
| bin.000056 | 123 | Previous_gtids |      5709 |         194 | c1f87a6a-98f2-11e5-b873-5254a03976fb:1-1831 |
+------------+-----+----------------+-----------+-------------+-------------------------------------------+
2 rows in set (0.01 sec)
-- 第一行是mysql版本和日志版本
-- 第二行是GTID号
-- 上面的两行记录在每个binlog的开头都有（5.6以后）


mysql> flush binary logs;  -- or flush logs 刷新日志，并且会产生一个新的日志
Query OK, 0 rows affected (0.06 sec)

mysql> show master status;
+------------+----------+--------------+------------------+-------------------------------------------+
| File       | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set                         |
+------------+----------+--------------+------------------+-------------------------------------------+
| bin.000057 |      194 |              |                  | c1f87a6a-98f2-11e5-b873-5254a03976fb:1-1831 |
+------------+----------+--------------+------------------+-------------------------------------------+
1 row in set (0.00 sec)


mysql> show binlog events in 'bin.000056';  -- 查看56，发现最后一行的记录是rotate，日志分割了
+------------+-----+----------------+-----------+-------------+-------------------------------------------+
| Log_name   | Pos | Event_type     | Server_id | End_log_pos | Info                                      |
+------------+-----+----------------+-----------+-------------+-------------------------------------------+
| bin.000056 |   4 | Format_desc    |      5709 |         123 | Server ver: 5.7.9-log, Binlog ver: 4      |
| bin.000056 | 123 | Previous_gtids |      5709 |         194 | c1f87a6a-98f2-11e5-b873-5254a03976fb:1-1831 |
| bin.000056 | 194 | Rotate         |      5709 |         235 | bin.000057;pos=4                          |
+------------+-----+----------------+-----------+-------------+-------------------------------------------+
3 rows in set (0.00 sec)

mysql> show binlog events in 'bin.000057';  -- 新产生的日志
+------------+-----+----------------+-----------+-------------+-------------------------------------------+
| Log_name   | Pos | Event_type     | Server_id | End_log_pos | Info                                      |
+------------+-----+----------------+-----------+-------------+-------------------------------------------+
| bin.000057 |   4 | Format_desc    |      5709 |         123 | Server ver: 5.7.9-log, Binlog ver: 4      |
| bin.000057 | 123 | Previous_gtids |      5709 |         194 | c1f87a6a-98f2-11e5-b873-5254a03976fb:1-1831 |
+------------+-----+----------------+-----------+-------------+-------------------------------------------+
2 rows in set (0.00 sec)
```

3.4. binlog 演示

```
mysql> show master status;
+------------+----------+--------------+------------------+-------------------------------------------+
| File       | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set                         |
+------------+----------+--------------+------------------+-------------------------------------------+
| bin.000057 |      194 |              |                  | c1f87a6a-98f2-11e5-b873-5254a03976fb:1-1831 |
+------------+----------+--------------+------------------+-------------------------------------------+
1 row in set (0.00 sec)

mysql> show binlog events in "bin.000057";
+------------+-----+----------------+-----------+-------------+-------------------------------------------+
| Log_name   | Pos | Event_type     | Server_id | End_log_pos | Info                                      |
+------------+-----+----------------+-----------+-------------+-------------------------------------------+
| bin.000057 |   4 | Format_desc    |      5709 |         123 | Server ver: 5.7.9-log, Binlog ver: 4      |
| bin.000057 | 123 | Previous_gtids |      5709 |         194 | c1f87a6a-98f2-11e5-b873-5254a03976fb:1-1831 |
+------------+-----+----------------+-----------+-------------+-------------------------------------------+
2 rows in set (0.00 sec)

mysql> set binlog_format="STATEMENT";  -- 临时修改为 STATEMENT 格式
Query OK, 0 rows affected (0.00 sec)

mysql> create  table test_bin_1 (a int);
Query OK, 0 rows affected (0.14 sec)

mysql> show binlog events in "bin.000057";
+------------+-----+----------------+-----------+-------------+-------------------------------------------------------------------------+
| Log_name   | Pos | Event_type     | Server_id | End_log_pos | Info                                                                    |
+------------+-----+----------------+-----------+-------------+-------------------------------------------------------------------------+
| bin.000057 |   4 | Format_desc    |      5709 |         123 | Server ver: 5.7.9-log, Binlog ver: 4                                    |
| bin.000057 | 123 | Previous_gtids |      5709 |         194 | c1f87a6a-98f2-11e5-b873-5254a03976fb:1-1831                             |
| bin.000057 | 194 | Gtid           |      5709 |         259 | SET @@SESSION.GTID_NEXT= 'c1f87a6a-98f2-11e5-b873-5254a03976fb:1832'    |
| bin.000057 | 259 | Query          |      5709 |         375 | use `burn_test`; create  table test_bin_1 (a int)                       |
+------------+-----+----------------+-----------+-------------+-------------------------------------------------------------------------+
4 rows in set (0.00 sec)
-- 出现了 Query的类型，并且能显示SQL语句

mysql> insert into test_bin_1 values(1),(2);  -- 插入数据时，MySQL5.7 直接提示必须是row格式
ERROR 1665 (HY000): Cannot execute statement: impossible to write to binary log since BINLOG_FORMAT = STATEMENT and at least one table uses a storage engine limited to row-based logging. InnoDB is limited to row-logging when transaction isolation level is READ COMMITTED or READ UNCOMMITTED.

mysql> set binlog_format="ROW";  -- 改成ROW格式
Query OK, 0 rows affected (0.00 sec)

mysql> insert into test_bin_1 values(1),(2);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> insert into test_bin_1 values(1),(2);  -- 重复插入两次
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> commit;  -- 因为autocommit=0，所以我要显示的提交
Query OK, 0 rows affected (0.03 sec)

mysql> show binlog events in "bin.000057";
+------------+-----+----------------+-----------+-------------+-------------------------------------------------------------------------+
| Log_name   | Pos | Event_type     | Server_id | End_log_pos | Info                                                                    |
+------------+-----+----------------+-----------+-------------+-------------------------------------------------------------------------+
| bin.000057 |   4 | Format_desc    |      5709 |         123 | Server ver: 5.7.9-log, Binlog ver: 4                                    |
| bin.000057 | 123 | Previous_gtids |      5709 |         194 | c1f87a6a-98f2-11e5-b873-5254a03976fb:1-1831                             |
| bin.000057 | 194 | Gtid           |      5709 |         259 | SET @@SESSION.GTID_NEXT= 'c1f87a6a-98f2-11e5-b873-5254a03976fb:1832'    |
| bin.000057 | 259 | Query          |      5709 |         375 | use `burn_test`; create  table test_bin_1 (a int)                       |
| bin.000057 | 375 | Gtid           |      5709 |         440 | SET @@SESSION.GTID_NEXT= 'c1f87a6a-98f2-11e5-b873-5254a03976fb:1833'    |
| bin.000057 | 440 | Query          |      5709 |         517 | BEGIN                                                                   |
| bin.000057 | 517 | Table_map      |      5709 |         575 | table_id: 159 (burn_test.test_bin_1)                                    |
| bin.000057 | 575 | Write_rows     |      5709 |         620 | table_id: 159 flags: STMT_END_F                                         |
| bin.000057 | 620 | Xid            |      5709 |         651 | COMMIT /* xid=80 */                                                     | -- 第一次插入1，2
| bin.000057 | 651 | Gtid           |      5709 |         716 | SET @@SESSION.GTID_NEXT= 'c1f87a6a-98f2-11e5-b873-5254a03976fb:1834'    |
| bin.000057 | 716 | Query          |      5709 |         793 | BEGIN                                                                   |
| bin.000057 | 793 | Table_map      |      5709 |         851 | table_id: 159 (burn_test.test_bin_1)                                    |
| bin.000057 | 851 | Write_rows     |      5709 |         896 | table_id: 159 flags: STMT_END_F                                         |
| bin.000057 | 896 | Xid            |      5709 |         927 | COMMIT /* xid=86 */                                                     | -- 第二次插入1，2
+------------+-----+----------------+-----------+-------------+-------------------------------------------------------------------------+
14 rows in set (0.00 sec)
9 rows in set (0.00 sec)
-- Table_map : 把表名映射到ID，并且记录列的类型，和表的元数据信息
-- Write_rows: 插入的类型，但是看不到插入的数据

mysql> update test_bin_1 set a=3 where a=1;  -- update 操作，对应的就是 Update_rows 的类型
Query OK, 2 rows affected (0.01 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> commit;  -- 同样要显示commit
Query OK, 0 rows affected (0.02 sec)

mysql> show binlog events in "bin.000057";
+------------+------+----------------+-----------+-------------+-------------------------------------------------------------------------+
| Log_name   | Pos  | Event_type     | Server_id | End_log_pos | Info                                                                    |
+------------+------+----------------+-----------+-------------+-------------------------------------------------------------------------+
-- ---------------------------------省略其他输出-------------------------------------
| bin.000057 |  927 | Gtid           |      5709 |         992 | SET @@SESSION.GTID_NEXT= 'c1f87a6a-98f2-11e5-b873-5254a03976fb:1835'    |
| bin.000057 |  992 | Query          |      5709 |        1069 | BEGIN                                                                   |
| bin.000057 | 1069 | Table_map      |      5709 |        1127 | table_id: 159 (burn_test.test_bin_1)                                    |
| bin.000057 | 1127 | Update_rows    |      5709 |        1183 | table_id: 159 flags: STMT_END_F                                         |
| bin.000057 | 1183 | Xid            |      5709 |        1214 | COMMIT /* xid=93 */                                                     |
+------------+------+----------------+-----------+-------------+-------------------------------------------------------------------------+
19 rows in set (0.00 sec)

mysql> delete from test_bin_1 where a=3;  -- delte操作，对应的是 Delete_rows 的操作
Query OK, 2 rows affected (0.00 sec)

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

mysql> show binlog events in "bin.000057";
+------------+------+----------------+-----------+-------------+-------------------------------------------------------------------------+
| Log_name   | Pos  | Event_type     | Server_id | End_log_pos | Info                                                                    |
+------------+------+----------------+-----------+-------------+-------------------------------------------------------------------------+
-- ---------------------------------省略其他输出-------------------------------------
| bin.000057 | 1214 | Gtid           |      5709 |        1279 | SET @@SESSION.GTID_NEXT= 'c1f87a6a-98f2-11e5-b873-5254a03976fb:1836'    |
| bin.000057 | 1279 | Query          |      5709 |        1356 | BEGIN                                                                   |
| bin.000057 | 1356 | Table_map      |      5709 |        1414 | table_id: 159 (burn_test.test_bin_1)                                    |
| bin.000057 | 1414 | Delete_rows    |      5709 |        1459 | table_id: 159 flags: STMT_END_F                                         |
| bin.000057 | 1459 | Xid            |      5709 |        1490 | COMMIT /* xid=97 */                                                     |
+------------+------+----------------+-----------+-------------+-------------------------------------------------------------------------+
24 rows in set (0.00 sec)
```

3.5. mysqlbinlog

```
[root@MyServer 5.7]> mysqlbinlog  bin.000057
/*!50530 SET @@SESSION.PSEUDO_SLAVE_MODE=1*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
DELIMITER /*!*/;
# at 4
#160227 16:29:35 server id 5709  end_log_pos 123 CRC32 0xe902425e       Start: binlog v 4, server v 5.7.9-log created 160227 16:29:35
# Warning: this binlog is either in use or was not closed properly.
BINLOG '
b17RVg9NFgAAdwAAAHsAAAABAAQANS43LjktbG9nAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAEzgNAAgAEgAEBAQEEgAAXwAEGggAAAAICAgCAAAACgoKKioAEjQA
AV5CAuk=
'/*!*/;
# at 123
#160227 16:29:35 server id 5709  end_log_pos 194 CRC32 0xe374b961       Previous-GTIDs
# c1f87a6a-98f2-11e5-b873-5254a03976fb:1-1831

## ---------------------------------省略其他输出-------------------------------------


[root@MyServer 5.7]> mysqlbinlog  bin.000057 -vv  ##增加 -vv 参数，可以看到注释信息
/*!50530 SET @@SESSION.PSEUDO_SLAVE_MODE=1*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
DELIMITER /*!*/;
# at 4
#160227 16:29:35 server id 5709  end_log_pos 123 CRC32 0xe902425e       Start: binlog v 4, server v 5.7.9-log created 160227 16:29:35
# Warning: this binlog is either in use or was not closed properly.
## ---------------------------------省略其他输出-------------------------------------
BINLOG '
W2XRVhNNFgAAOgAAAIYFAAAAAJ8AAAAAAAEACWJ1cm5fdGVzdAAKdGVzdF9iaW5fMQABAwAB8/uH
Iw==
W2XRViBNFgAALQAAALMFAAAAAJ8AAAAAAAEAAgAB//4DAAAA/gMAAADj7o8u
'/*!*/;
### DELETE FROM `burn_test`.`test_bin_1`    # 增加了-vv参数可以看到注释信息，但是这些信息不是SQL
### WHERE
###   @1=3 /* INT meta=0 nullable=1 is_null=0 */  # 看到列的信息
### DELETE FROM `burn_test`.`test_bin_1`
### WHERE
###   @1=3 /* INT meta=0 nullable=1 is_null=0 */
# at 1459
#160227 16:59:14 server id 5709  end_log_pos 1490 CRC32 0x4534dc52      Xid = 97
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*/;
```

> **注意：这些注释信息不是SQL语句，他只是记录了页的变化**

四. REDO 和 binlog的总结

1. redolog是 InnoDB 层的；binlog是 MySQL 层的。
2. redolog是 物理逻辑 日志；binlog是 逻辑 日志
3. 写入的时间点不一样
   - redo log 可以有多个写入点，比如master thread刷新，buffer大于1/2，还是事物提交等等
   - binlog只会在 事物提交 的时候写入

binlog

T1	T4	T3	T2	T8	T6	T7	T5

redo log

T1	T2	T1	*T2	T3	T1	*T3	*T1

(星号代表提交)