24 | MySQL是怎么保证主备一致的？

**MySQL主备的基本原理**

如图1 所示基本的主备切换原理

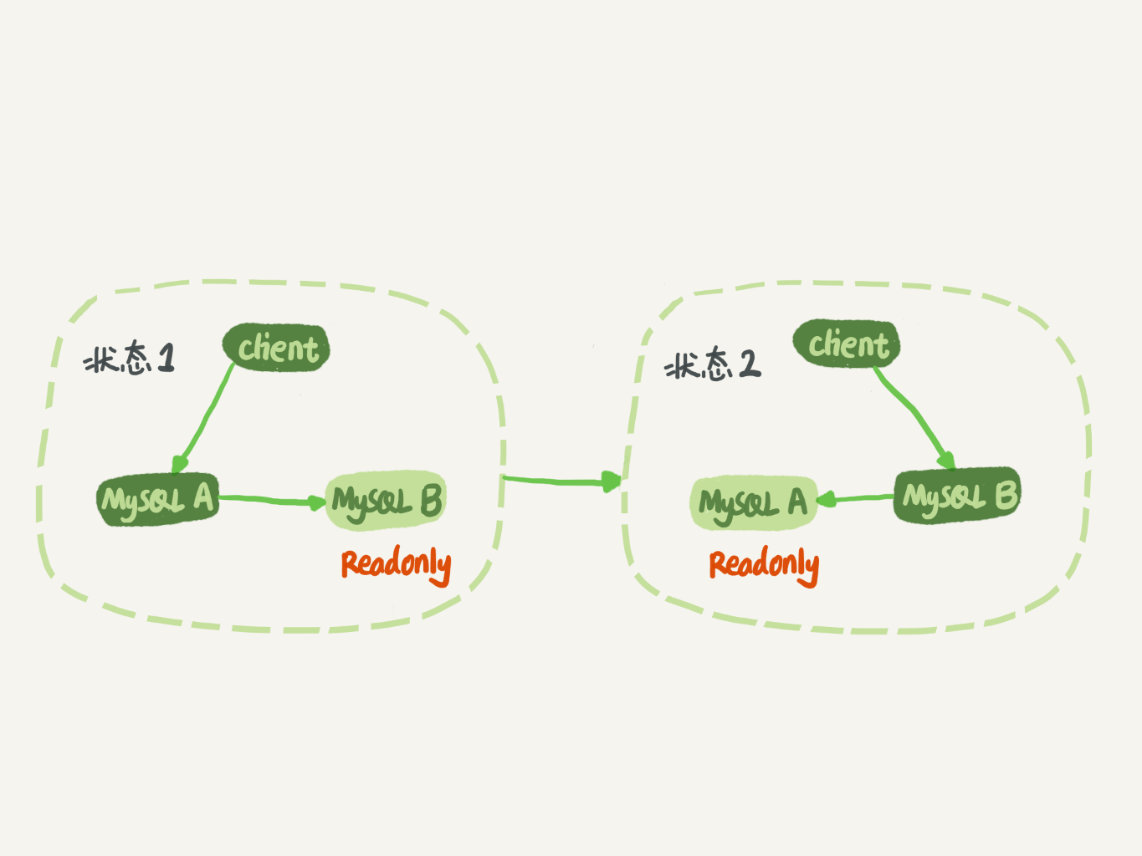


图 1 MySQL 主备切换流程

在状态1中，客户端的读取都是直接访问节点A,而节点B是节点A的备库，只是将A的更新都同步过来，到本地执行，这样可以保证节点B和节点A的数据是相同的。

当切换的时候，就切成状态2，此时客户端读写访问的都是节点B,而节点A是B的备库

在状态1，虽然节点B没有被直接访问，但是建议把节点B（也就是备库）设置成只读（readonly）模式，有一下几点考虑

1.有时候一些运营类的查询语句会被放到备库去查，设置为只读可以防止误操作

2.防止切换逻辑有bug,比如切换过程中出现双写，造成主备不一致

3.可以用readonly状态，来判断节点的角色

疑问？备库设置为只读，还怎么跟主库保持同步更新？

因为readonly设置对超级（super）权限用户是无效的，而用于同步更新的线程，就拥有超级权限。

**节点A到B这条线的内部流程是什么样的？**图2,就是一个update语句在节点A执行，然后同步到节点B的完整流程

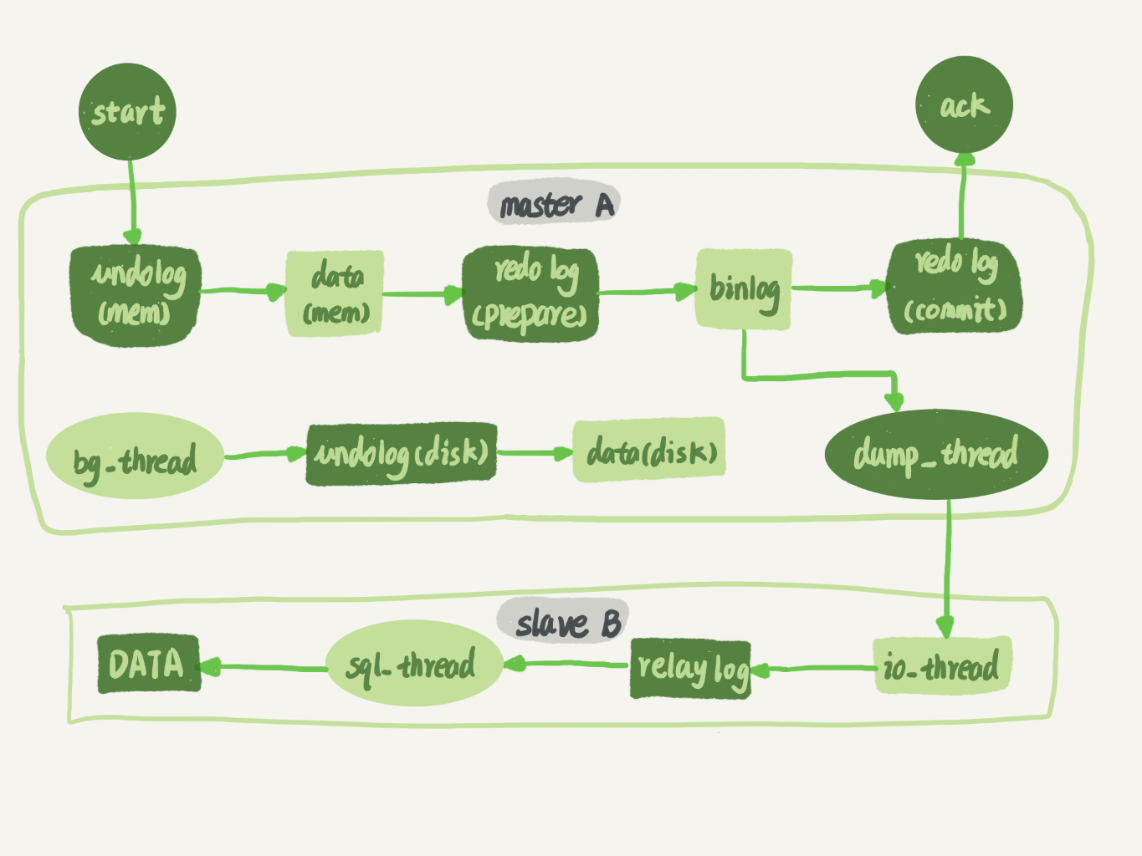


图 2 主备流程图

图2中，包含了binlog和redo log的写入机制相关内容，可以看到：主库接收到客户端的更细请求后，执行内部事务的更新逻辑，同时写binlog.

备库B更主库A维持了一个长连接，主库A内部有一个线程，专门用于服务备库B的这个长连接，一个事务日志同步的过程

1.在备库B上通过change master命令，设置主库A的ip,端口，用户名，密码，以及要从那个位置开始请求binlog,这个位置包含文件和日志偏移量

2.在备库B上执行start slave命令，此时备库会启动两个线程，就是图中的io\_thread和sql\_thread，其中io\_thread负责与主库建立连接。

3.主库A校验完用户名、密码后，开始按照备库B传过来的位子，从本地读取binlog,发给B

4.备库B拿到binlog后，写到本地文件，称为中转日志（relay log）

5.sql\_thread读取中转日志，解析日志里的命令，并执行。

需要说明，后来由于多线程的复制方案的引入，sql\_thread演化成了多个线程，跟我们今天要介绍的原理没有直接关系，不展开介绍

Binlog里面的是什么内容，为什么备库拿过去可以直接执行

**Binlog的三种格式对比**

**一种是statement,一种是row,还要一种是mixed,其实它是前两种格式的混合。**

便于描述binlog三种格式间的区别，创建一个表并初始化几行数据

mysql> CREATE TABLE `t` (

`id` int(11) NOT NULL,

`a` int(11) DEFAULT NULL,

`t\_modified` timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP,

PRIMARY KEY (`id`),

KEY `a` (`a`),

KEY `t\_modified`(`t\_modified`)

) ENGINE=InnoDB;

insert into t values(1,1,'2018-11-13');

insert into t values(2,2,'2018-11-12');

insert into t values(3,3,'2018-11-11');

insert into t values(4,4,'2018-11-10');

insert into t values(5,5,'2018-11-09');

如果在表中删除一行数据的话，看看这个binlog是怎么记录的

下面的语句包含注释，如果用MySQL客户端来做这个实验的话，要记得加-c参数，否则客户端会自动取消注释

mysql> delete from t /\*comment\*/ where a>=4 and t\_modified<='2018-11-10' limit 1;

当binlon=statement时，binlog里面记录的就是SQL语句的原文

mysql> show binlog events in 'master.000001';

命令看binlog中的内容

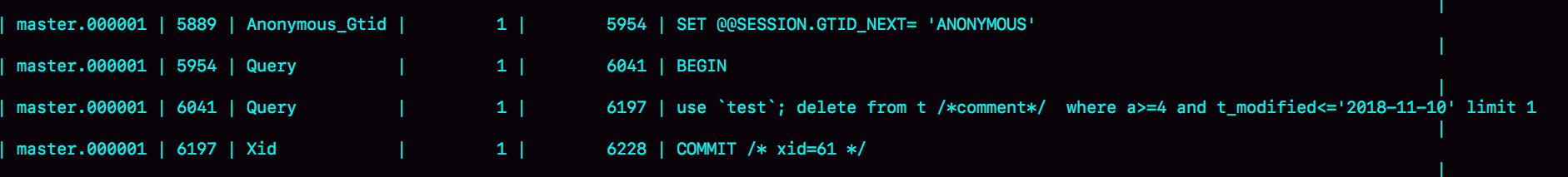


图3 statement格式binlog示例

·第一行set @@SESSION.GTID\_NEXT=’ANONYMOUS’可以选择忽略，后面文章会在主备切换时候提到

·第二行是一个BEGIN,跟第四行的commit对应， 表示中间是一个事务

·第三行就是真实执行的语句了，可以看到，在真实执行的delete命令之前，还有一个“use ‘test’”命令，这个条命令不是我们主动执行的，而是MySQL根据当前要操作的表所在的数据库，自行添加的，这样可以保证日志传到备库执行的时候，不论当前工作线程在哪个库里，都能够正确的更新到test库的表t.

Use test 命令之后的delete语句，就是我们输入的SQL原文了，可以看到，binlog“忠实”地记录了SQL命令，甚至注释也一并记录了

·最后一行是一个COMMIT,里面写这xid=61, 会议第15篇内容

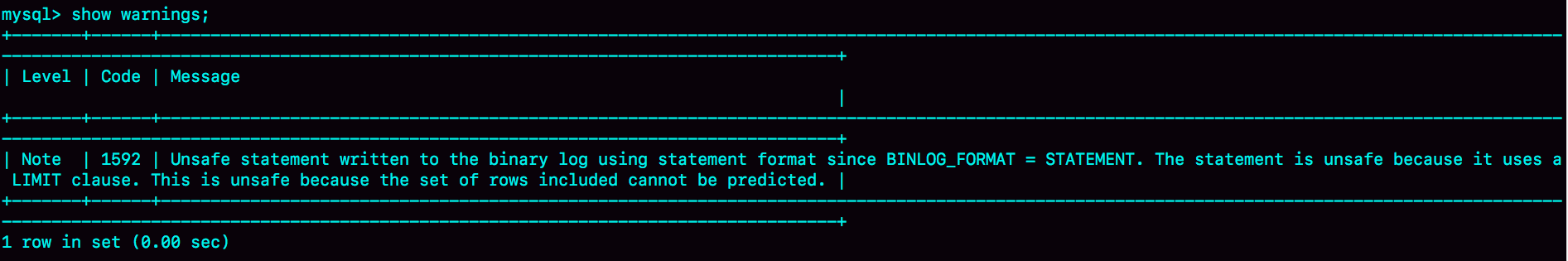
为了说明statement和row格式区别，看delete命令执行效果

图 4 delete执行warnings

运行delete命令产生一个warning，原因是当前binlog设置的是statement格式，并且语句中有limit,所以这个命令可能是unsafe的

为什么这么说？这是因为delete带limit,很可能会出现主备数据不一致的情况，

1. 如果delete语句使用的是索引a,那么会根据索引找到第一个满足条件的行，也就是删除的是a=4这一行。
2. 如果使用的是索引t\_modified,那么删除的就是t\_modified=’2018-11-09’也就是a=5这一行

由于statement格式下，记录到binlog里面的是语句原文，因此可能会出现这样的情况，在主库执行这条SQL语句，用的是索引a;而在备库执行这条SQL语句的时候，却使用了索引t\_modified,因此，MySQL认为这样写是有风险的。

如果把binlog的格式改为binlog\_format=’row’,是否还有这个问题？

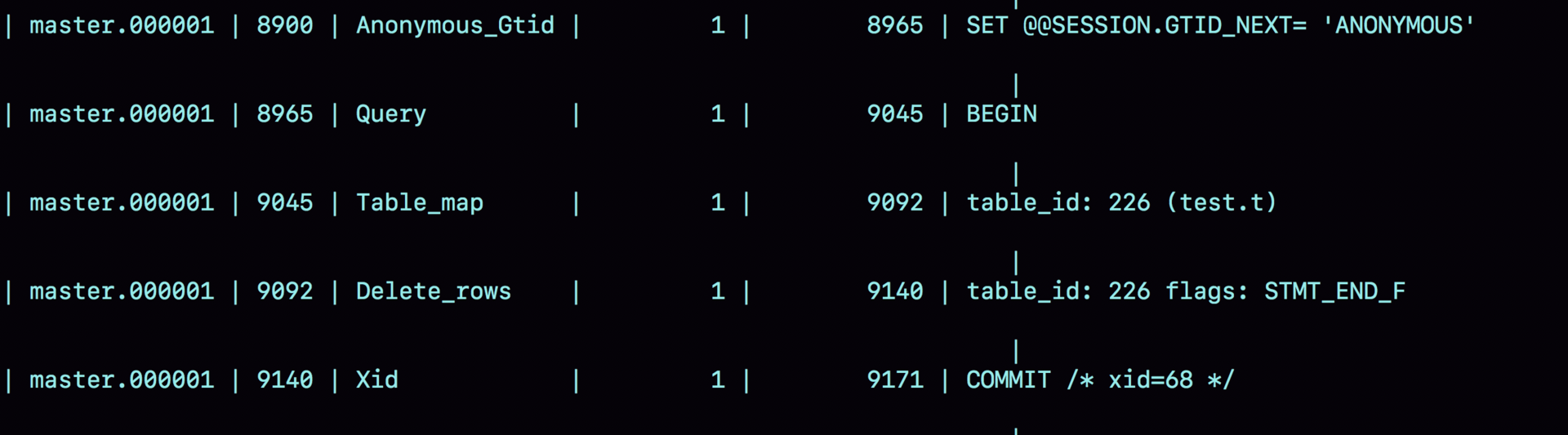


图5 row格式binlog示例

statement格式的binlog相比，前后的BEGIN的COMMIT是一样的，但是row格式的binlog里没有了SQL语句的原文，而是替换成了两个event,Table\_map和Delete\_rows.

1. Table\_map event,用于说明接下来要操作的表是test库的表t;
2. Delete\_rows event,用于定义删除的行为

图5是看不到详细信息，要借助myslqbinlog工具，用下面这个命令解析和查看binlog中的内容，因为图5中的信息显示，这个事务的binlog是从8900这个位置开始的，所以可以用start-position参数来指定从这个位子的日志开始解析

mysqlbinlog -vv data/master.000001 --start-position=8900;

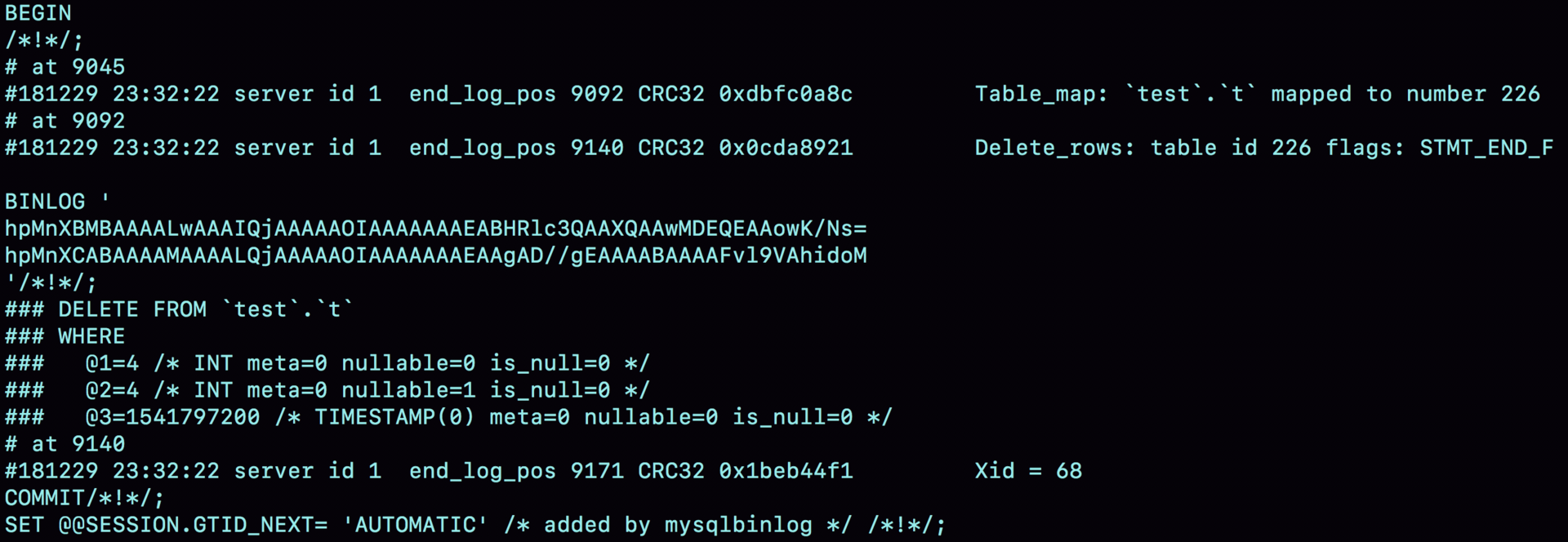


图 6 row格式binlog示例的详细信息

图中可以看到以下信息

1. server id,表示这个事务是在server \_id=1的这个库上执行的
2. 每个event都有CRC32的值，这是因为把参数binlog\_checksum设置为CRC32
3. Table\_map event 跟在图5中看到的相同，显示了接下来要打开的表，map到数字226，现在SQL语句只操作一张表，如果要操作多张表？每个表都有一个对应的Table\_map event、都会map到一个单独的数字，用于区分对不同表的操作
4. 在mysqlbinlog的命令中，使用了-vv参数是为了把内容都解析出来，所以从结果可以看到各个字段的值（比如，@1=4、@2=4这些值）
5. Binlog\_row\_image的默认配置是FULL,因此Delete\_event里面，包含了删掉的行的所有字段的值，如果把binlog\_row\_image设置为MINIMAL,则只会记录必要的信息，在这个例子里，就只会记录id=4这信息
6. 最后Xid event，用于表示事务被正确的提交了

当binlog\_format使用row格式的时候，binlog里面记录了真实删除行的主键id,这样binlog传导备库的时候，就肯定会删除id=4的行，不会有主备删除不同行的问题。

**为什么会有mixed格式的binlog?**

推论是这样的？

·因为有些statement格式的binlog,可能会导致主备不一致，所以要使用row格式

·但row格式的缺点是，很占空间。比如用一个delete语句删掉10万行数据，用statement的话就是一个SQL语句被记录到binlog中，占用几十个字节的空间。但如果用row格式的binlog，就要把这10万条记录都写到binlog中，这样做，不仅占用更大的空间，同时写binlog也要消耗IO资源，影响执行速度

·MySQL取个折中方案，就是mixed格斯的binlog,mixed格式的意思,MySQL自己会判断这条SQL语句是否会引起主备不一致，如果有可能，就用row格式，否则就用statement格式。

Mixed格式可以利用statement格式的有点，同时又避免了数据不一致的风险。

如果线上mysql设置的binlog格式是statement的话，是一个不合理的设置，至少应该binlog格式设置为mixed

比如这个例子，设置为mixed,就会记录为row格式，而如果执行的语句去掉limit 1,就会记录statement格式。

分别从delete,insert和update这三种SQL语句的角度，看看数据恢复的问题

通过图6，即使执行delete语句，row格式的binlog也会把被删掉的行整行信息保存起来，所以执行完一条delete语句以后，发现错误删除数据，可以直接把binlog中记录的delete语句转成insert,把被错删的数据插入回去就可以恢复了。

如果执行错insert语句？Row格式下，insert语句binlog里面会记录所有的字段信息，这些信息可以用来精确定位刚刚被插入的哪一行，此时，直接把insert语句转成delete语句，删除到这被误插入的一行数据就可以

如果执行update语句，binlog里面会记录修改前整行的数据和修改后的整行数据，所以，如果执行了update语句的话，只需要把这个event前后的两行信息对调 一下，再去数据库执行，就能恢复这个个更新操作。

其实，有delete、insert或者update语句导致的数据操作错误，需要恢复到操作之前状态的情况，时有发生，MariaDB的FlashBack工具就是基于上面的原理来回顾数据的

虽然mixed格式的binlog现在已经用的不多了，但是还是借用mixed说明一个问题

mysql> insert into t values(10,10, now());

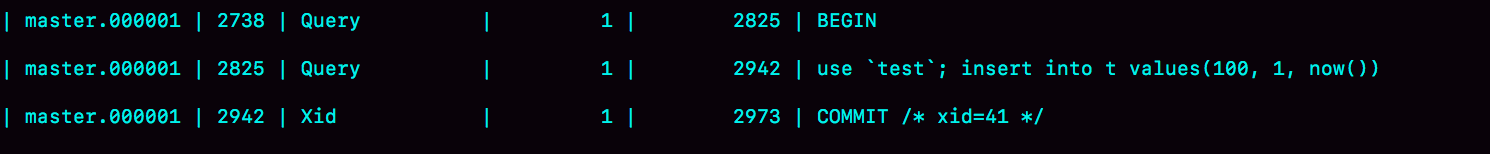


图 7 mixed格式和now()

MySQL居然是statement格式，如果过了1分钟才传给备库的话，那主备数据不就会不一致了吗？

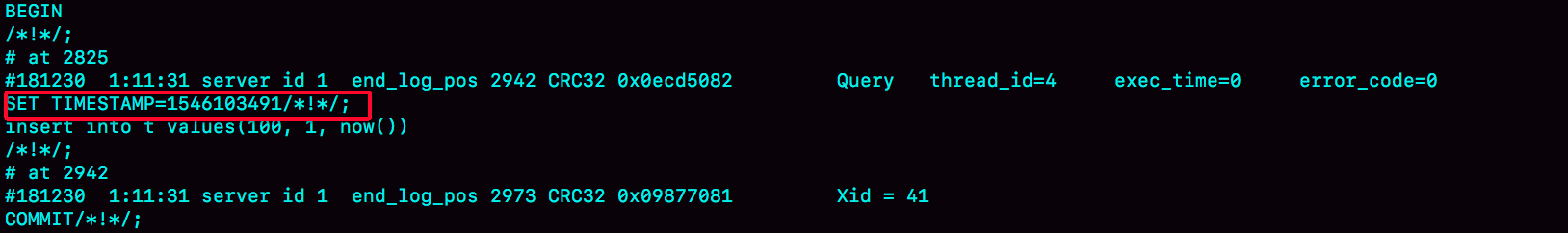


图 8 TIMESTAMP 命令

Binlog在记录event的时候，多记录了一条命令，set TIMESTAMP=1546103491,他利用SET TIMESTAMP命令约定下来的now()函数返回时间

不论这个binlog是1分钟之后被备库执行，还是3天后用来恢复这个库的备份，这个insert语句插入的行，值都是固定的，这条SET TIMESTAMP命令，MySQL就确保了主备数据的一致性。

**所以，用binlog来恢复数据的标准做法: 用mysqlbinlog工具解析出来，然后把解析结果发给MySQL执行。**

mysqlbinlog master.000001 --start-position=2738 --stop-position=2973 | mysql -h127.0.0.1 -P13000 -u$user -p$pwd;

这个命令是将master.000001文件里从第2738字节到2973字节中间这段内容解析出来，放到MySQL去执行

**循环复制问题**

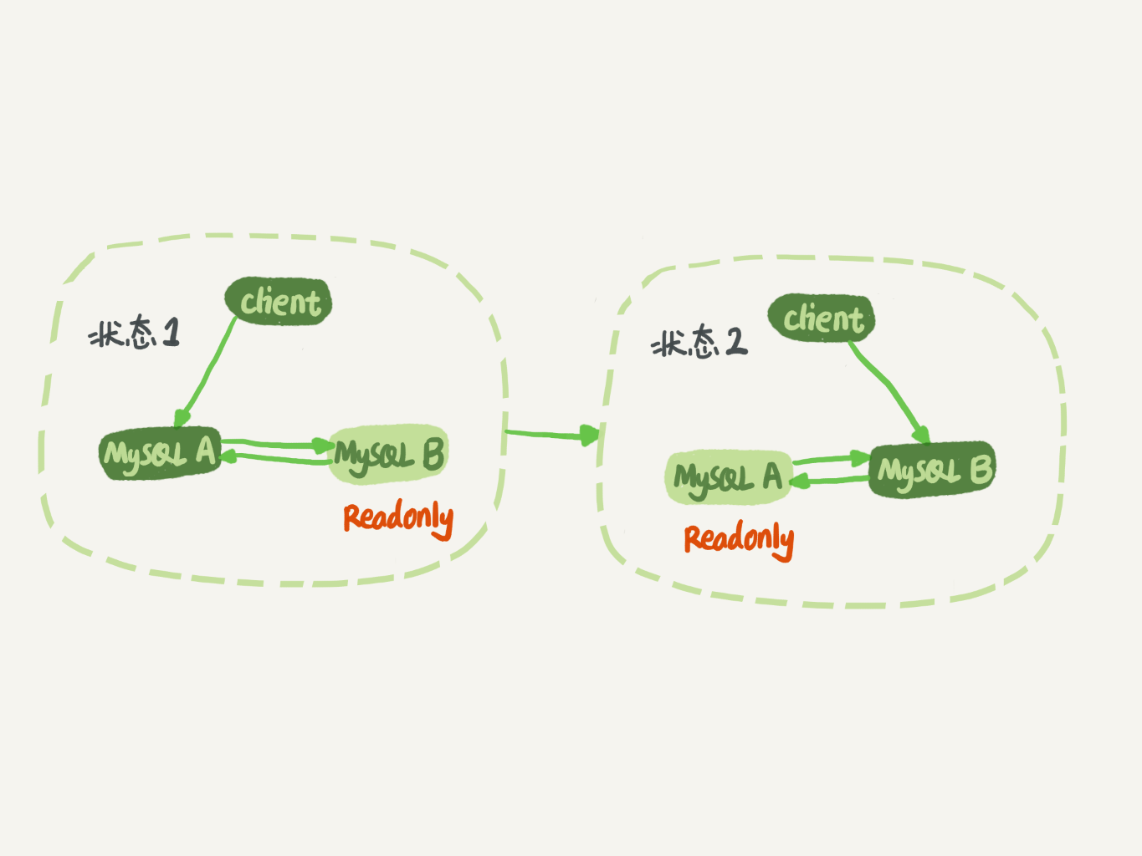
图是M-S结构，但实际生产上比较多的是双M结构，也就是如图9所示的主备切换流程

图9 MySQL主备切换流程--双M结构

对比图9和图1，双M结构和M-S机构，其实区别只是多了一条线，即：节点A和B之间总是互为主备关系，这样在切换的时候就不用在修改主备关系。

双M结构还有一个问题需要解决

如果节点A同时是节点B的备库，相当于有把节点B新生成的binlog拿过来执行了一次，然后节点A和B间，会不断的循环执行这个更新语句，也就是循环复制了，这个怎么解决呢？

1. 规定两个库的server id 必须不同，如果相同，则他们之间不能设定为主备关系
2. 一个备库接到binlog并在重放的过程中，生成与原binlog的server id相同的新的binlog
3. 每个库在收到从自己的主库发过来的日志后，先判断server id，如果跟自己的相同，表示这个日志是自己生成的，直接丢弃这个日志。

按照这个逻辑，如果我们设置了双M结构，日志的执行路程会这样

1. 从节点A更新的事务，binlog里面记的都是A 的server id
2. 传到节点B执行一次以后，节点B生成的binlog的server id也是A的server id
3. 再回传给节点A,A判断到这个server id与自己相同，就不会再处理这个日志，所以，死循环在这里断掉了。