

讲堂 > MySQL实战45讲 > 文章详情

## 15 | 答疑文章（一）：日志和索引相关问题

2018-12-17 林晓斌



### 15 | 答疑文章（一）：日志和索引相关问题

朗读人：林晓斌 20'16" | 18.57M

在今天这篇答疑文章更新前，MySQL 实战这个专栏已经更新了 14 篇。在这些文章中，大家在评论区留下了很多高质量的留言。现在，每篇文章的评论区都有热心的同学帮忙总结文章知识点，也有不少同学提出了很多高质量的问题，更有一些同学帮忙解答其他同学提出的问题。

在浏览这些留言并回复的过程中，我倍受鼓舞，也尽我所知地帮助你解决问题、和你讨论。可以说，你们的留言活跃了整个专栏的氛围、提升了整个专栏的质量，谢谢你们。

评论区的大多数的留言我都直接回复了，对于需要展开说明的问题，我都拿出小本子记了下来。这些被记下来的问题，就是我们今天这篇答疑文章的素材了。

到目前为止，我已经收集了 47 个问题，很难通过今天这一篇文章全部展开。所以，我就先从中找了几个联系非常紧密的问题，串了起来，希望可以帮你解决关于日志和索引的一些疑惑。而其他问题，我们就留着后面慢慢展开吧。

### 日志相关问题

我在第 2 篇文章 [《日志系统：一条 SQL 更新语句是如何执行的？》](#) 中，和你讲到 binlog（归档日志）和 redo log（重做日志）配合崩溃恢复的时候，用的是反证法，说明了如果没有两阶段提交，会导致 MySQL 出现主备数据不一致等问题。

在这篇文章下面，很多同学在问，在两阶段提交的不同瞬间，MySQL 如果发生异常重启，是怎么保证数据完整性的？

现在，我们就从这个问题开始吧。

我再放一次两阶段提交的图，方便你学习下面的内容。

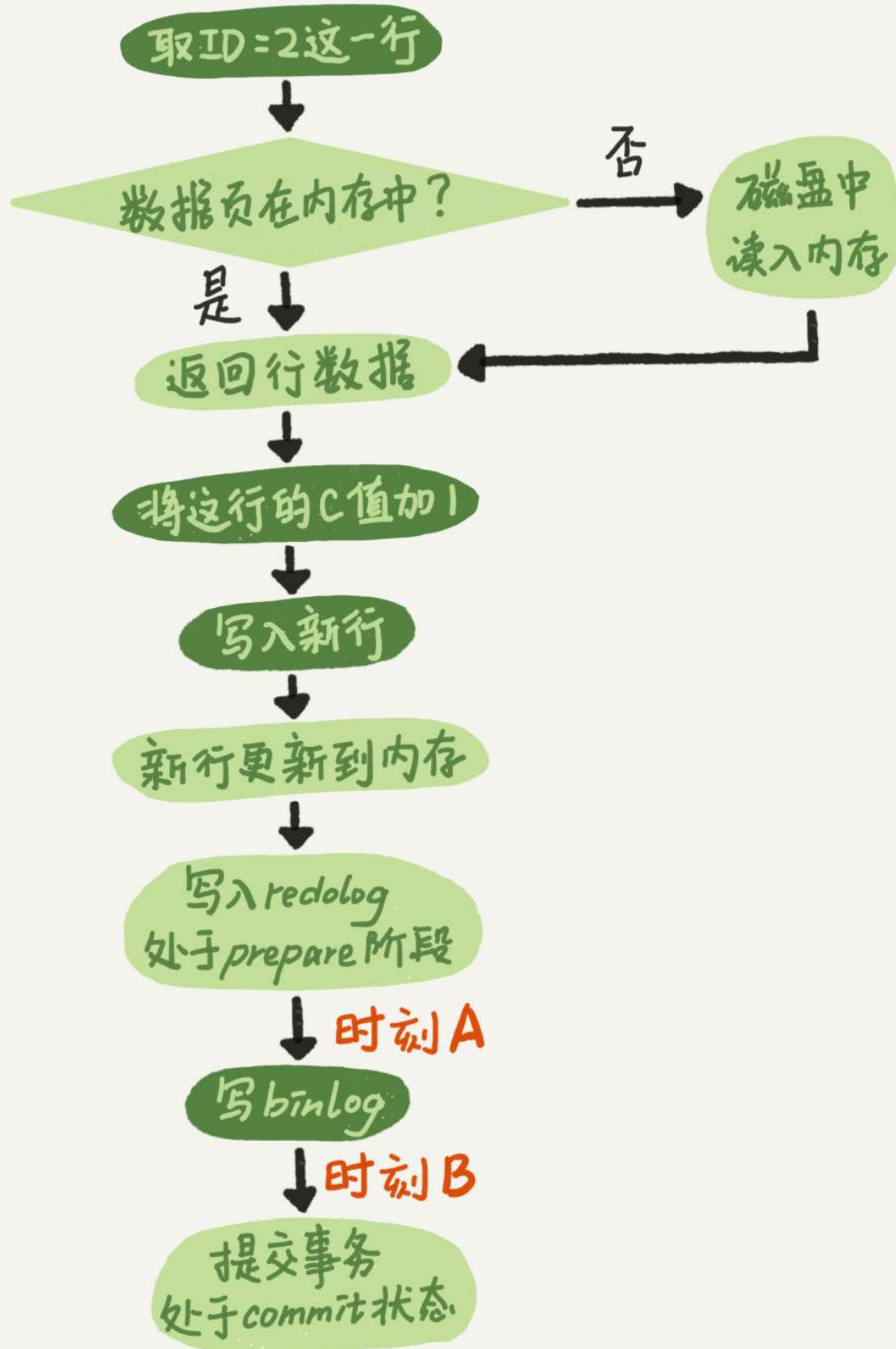


图 1 两阶段提交示意图

这里，我要先和你解释一个误会式的问题。有同学在评论区间问到，这个图不是一个 update 语句的执行流程吗，怎么还会调用 commit 语句？

他产生这个疑问的原因，是把**两个“commit”的概念**混淆了：

- 他说的“commit 语句”，是指 MySQL 语法中，用于提交一个事务的命令。一般跟 begin/start transaction 配对使用。
- 而我们图中用到的这个“commit 步骤”，指的是事务提交过程中的一个小步骤，也是最后一步。当这个步骤执行完成后，这个事务就提交完成了。
- “commit 语句”执行的时候，会包含“commit 步骤”。

而我们这个例子里面，没有显式地开启事务，因此这个 update 语句自己就是一个事务，在执行完成后提交事务时，就会用到这个“commit 步骤”。

接下来，我们就一起分析一下**在两阶段提交的不同时刻，MySQL 异常重启会出现什么现象。**

如果在图中时刻 A 的地方，也就是写入 redo log 处于 prepare 阶段之后、写 binlog 之前，发生了崩溃（crash），由于此时 binlog 还没写，redo log 也还没提交，所以崩溃恢复的时候，这个事务会回滚。这时候，binlog 还没写，所以也不会传到备库。到这里，大家都可以理解。

大家出现问题的地方，主要集中在时刻 B，也就是 binlog 写完，redo log 还没 commit 前发生 crash，那崩溃恢复的时候 MySQL 会怎么处理？

我们先来看一下崩溃恢复时的判断规则。

1. 如果 redo log 里面的事务是完整的，也就是已经有了 commit 标识，则直接提交；
2. 如果 redo log 里面的事务只有完整的 prepare，则判断对应的事务 binlog 是否存在并完整：
  - a. 如果是，则提交事务；
  - b. 否则，回滚事务。

这里，时刻 B 发生 crash 对应的就是 2(a) 的情况，崩溃恢复过程中事务会被提交。

现在，我们继续延展一下这个问题。

### 追问 1：MySQL 怎么知道 binlog 是完整的？

回答：一个事务的 binlog 是有完整格式的：

- statement 格式的 binlog，最后会有 COMMIT；
- row 格式的 binlog，最后会有一个 XID event。

另外，在 MySQL 5.6.2 版本以后，还引入了 binlog-checksum 参数，用来验证 binlog 内容的正确性。对于 binlog 日志由于磁盘原因，可能会在日志中间出错的情况，MySQL 可以通过校验 checksum 的结果来发现。所以，MySQL 还是有办法验证事务 binlog 的完整性的。

## 追问 2: redo log 和 binlog 是怎么关联起来的?

回答: 它们有一个共同的数据字段, 叫 XID。崩溃恢复的时候, 会按顺序扫描 redo log:

- 如果碰到既有 prepare、又有 commit 的 redo log, 就直接提交;
- 如果碰到只有 prepare、而没有 commit 的 redo log, 就拿着 XID 去 binlog 找对应的事务。

## 追问 3: 处于 prepare 阶段的 redo log 加上完整 binlog, 重启就能恢复, MySQL 为什么要这么设计?

回答: 其实, 这个问题还是跟我们在反证法中说到的数据与备份的一致性有关。在时刻 B, 也就是 binlog 写完以后 MySQL 发生崩溃, 这时候 binlog 已经写入了, 之后就会被从库 (或者用这个 binlog 恢复出来的库) 使用。

所以, 在主库上也要提交这个事务。采用这个策略, 主库和备库的数据就保证了一致性。

## 追问 4: 如果这样的话, 为什么还要两阶段提交呢? 干脆先 redo log 写完, 再写 binlog。崩溃恢复的时候, 必须得两个日志都完整才可以。是不是一样的逻辑?

回答: 其实, 两阶段提交是经典的分布式系统问题, 并不是 MySQL 独有的。

如果必须要举一个场景, 来说明这么做的必要性的话, 那就是事务的持久性问题。

对于 InnoDB 引擎来说, 如果 redo log 提交完成了, 事务就不能回滚 (如果这还允许回滚, 就可能覆盖掉别的事务的更新)。而如果 redo log 直接提交, 然后 binlog 写入的时候失败, InnoDB 又回滚不了, 数据和 binlog 日志又不一致了。

两阶段提交就是为了给所有人一个机会, 当每个人都说 “我 ok” 的时候, 再一起提交。

## 追问 5: 不引入两个日志, 也就没有两阶段提交的必要了。只用 binlog 来支持崩溃恢复, 又能支持归档, 不就可以了?

回答: 这位同学的意思是, 只保留 binlog, 然后可以把提交流程改成这样: ... -> “数据更新到内存” -> “写 binlog” -> “提交事务”, 是不是也可以提供崩溃恢复的能力?

答案是不可以。

如果说**历史原因**的话, 那就是 InnoDB 并不是 MySQL 的原生存储引擎。MySQL 的原生引擎是 MyISAM, 设计之初就没有支持崩溃恢复。

InnoDB 在作为 MySQL 的插件加入 MySQL 引擎家族之前, 就已经是一个提供了崩溃恢复和事务支持的引擎了。

InnoDB 接入了 MySQL 后，发现既然 binlog 没有崩溃恢复的能力，那就用 InnoDB 原有的 redo log 好了。

而如果说**实现上的原因**的话，就有很多了。就按照问题中说的，只用 binlog 来实现崩溃恢复的流程，我画了一张示意图，这里就没有 redo log 了。

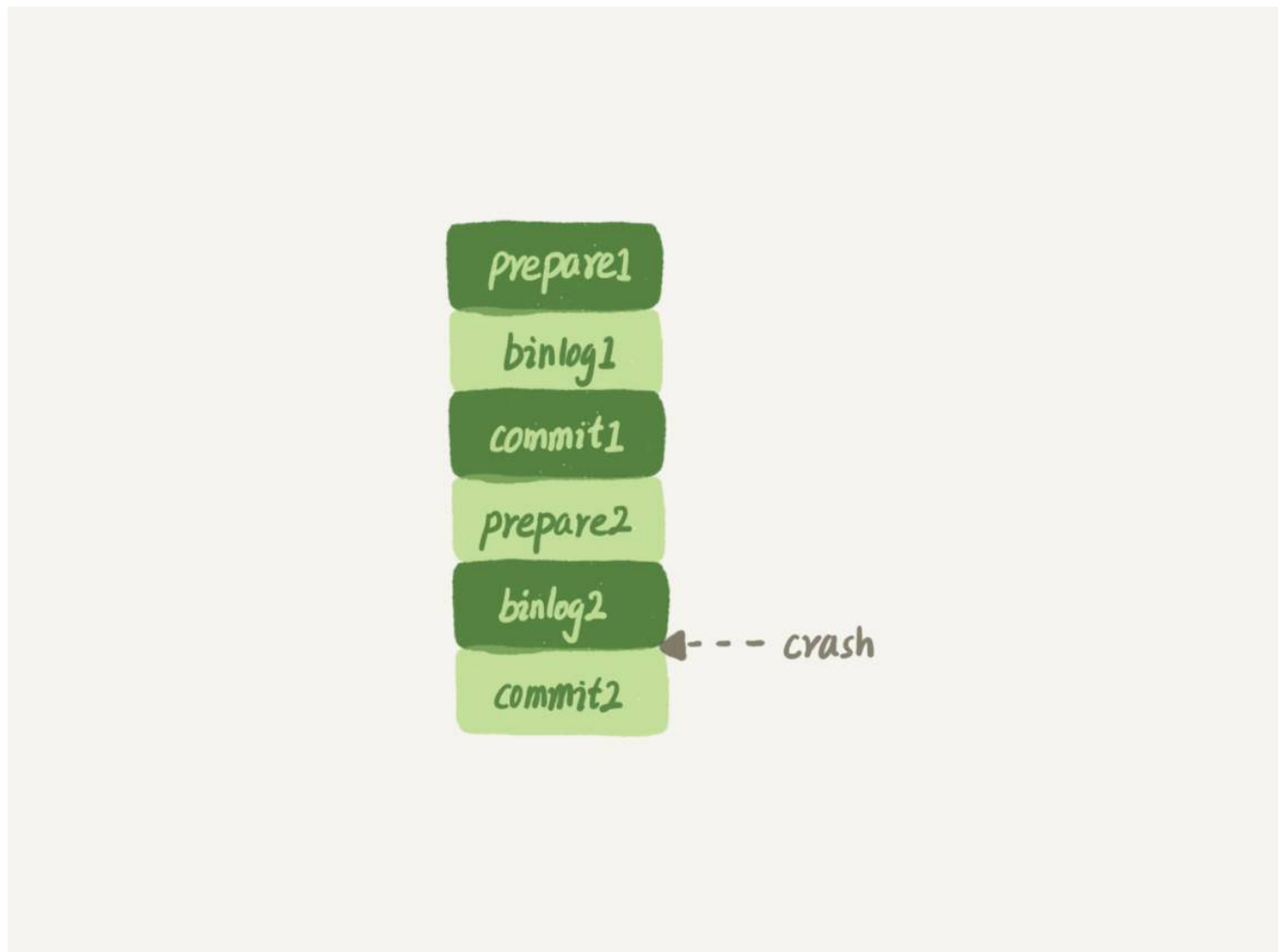


图 2 只用 binlog 支持崩溃恢复

这样的流程下，binlog 还是不能支持崩溃恢复的。我说一个不支持的点吧：binlog 没有能力恢复“数据页”。

如果在图中标的位置，也就是 binlog2 写完了，但是整个事务还没有 commit 的时候，MySQL 发生了 crash。

重启后，引擎内部事务 2 会回滚，然后应用 binlog2 可以补回来；但是对于事务 1 来说，系统已经认为提交完成了，不会再应用一次 binlog1。

但是，InnoDB 引擎使用的是 WAL 技术，执行事务的时候，写完内存和日志，事务就算完成了。如果之后崩溃，要依赖于日志来恢复数据页。

也就是说在图中这个位置发生崩溃的话，事务 1 也是可能丢失了的，而且是数据页级的丢失。此时，binlog 里面并没有记录数据页的更新细节，是补不回来的。

你如果说，那我优化一下 binlog 的内容，让它来记录数据页的更改可以吗？但，这其实就是又做了一个 redo log 出来。

所以，至少现在的 binlog 能力，还不能支持崩溃恢复。

### 追问 6：那能不能反过来，只用 redo log，不要 binlog？

回答：如果只从崩溃恢复的角度来讲是可以的。你可以把 binlog 关掉，这样就没有两阶段提交了，但系统依然是 crash-safe 的。

但是，如果你了解一下业界各个公司的使用场景的话，就会发现在正式的生产库上，binlog 都是开着的。因为 binlog 有着 redo log 无法替代的功能。

一个是归档。redo log 是循环写，写到末尾是要回到开头继续写的。这样历史日志没法保留，redo log 也就起不到归档的作用。

一个就是 MySQL 系统依赖于 binlog。binlog 作为 MySQL 一开始就有的功能，被用在了很多地方。其中，MySQL 系统高可用的基础，就是 binlog 复制。

还有很多公司有异构系统（比如一些数据分析系统），这些系统就靠消费 MySQL 的 binlog 来更新自己的数据。关掉 binlog 的话，这些下游系统就没法输入了。

总之，由于现在包括 MySQL 高可用在内的很多系统机制都依赖于 binlog，所以“鸠占鹊巢”redo log 还做不到。你看，发展生态是多么重要。

### 追问 7：redo log 一般设置多大？

回答：redo log 太小的话，会导致很快就被写满，然后不得不强行刷 redo log，这样 WAL 机制的能力就发挥不出来了。

所以，如果是现在常见的几个 TB 的磁盘的话，就不要太小气了，直接将 redo log 设置为 4 个文件、每个文件 1GB 吧。

### 追问 8：正常运行中的实例，数据写入后的最终落盘，是从 redo log 更新过来的还是从 buffer pool 更新过来的呢？

回答：这个问题其实问得非常好。这里涉及到了，“redo log 里面到底是什么”的问题。

实际上，redo log 并没有记录数据页的完整数据，所以它并没有能力自己去更新磁盘数据页，也就不存在“数据最终落盘，是由 redo log 更新过去”的情况。




1. 如果是正常运行的实例的话，数据页被修改以后，跟磁盘的数据页不一致，称为脏页。最终数据落盘，就是把内存中的数据页写盘。这个过程，甚至与 redo log 毫无关系。
2. 在崩溃恢复场景中，InnoDB 如果判断到一个数据页可能在崩溃恢复的时候丢失了更新，就会将它读到内存，然后让 redo log 更新内存内容。更新完成后，内存页变成脏页，就回到了第一种情况的状态。

## 追问 9：redo log buffer 是什么？是先修改内存，还是先写 redo log 文件？

回答：这两个问题可以一起回答。

在一个事务的更新过程中，日志是要写多次的。比如下面这个事务：

```
1 begin;
2 insert into t1 ...
3 insert into t2 ...
4 commit;
```

 复制代码

这个事务要往两个表中插入记录，插入数据的过程中，生成的日志都得先保存起来，但又不能在还没 commit 的时候就直接写到 redo log 文件里。

所以，redo log buffer 就是一块内存，用来先存 redo 日志的。也就是说，在执行第一个 insert 的时候，数据的内存被修改了，redo log buffer 也写入了日志。

但是，真正把日志写到 redo log 文件（文件名是 ib\_logfile+ 数字），是在执行 commit 语句的时候做的。

单独执行一个更新语句的时候，InnoDB 会自己启动一个事务，在语句执行完成的时候提交。过程跟上面是一样的，只不过是“压缩”到了一个语句里面完成。

以上这些问题，就是把大家提过的关于 redo log 和 binlog 的问题串起来，做的一次集中回答。如果你还有问题，可以在评论区继续留言补充。

## 业务设计问题

接下来，我再和你分享 @ithunter 同学在第 8 篇文章 [《事务到底是隔离的还是不隔离的？》](#) 的评论区提到的跟索引相关的一个问题。我觉得这个问题挺有趣、也挺实用的，其他同学也可能会碰上这样的场景，在这里解答和分享一下。

问题是这样的（我文字上稍微做了点修改，方便大家理解）：

业务上有这样的需求，A、B 两个用户，如果互相关注，则成为好友。设计上是有两张表，一个是 like 表，一个是 friend 表，like 表有 user\_id、liker\_id 两个字



段，我设置为复合唯一索引即 `uk_user_id_liker_id`。语句执行逻辑是这样的：

以 A 关注 B 为例：

第一步，先查询对方有没有关注自己（B 有没有关注 A）

```
select * from like where user_id = B and liker_id = A;
```

如果有，则成为好友

```
insert into friend;
```


没有，则只是单向关注关系

```
insert into like;
```

但是如果 A、B 同时关注对方，会出现不会成为好友的情况。因为上面第 1 步，双方都没关注对方。第 1 步即使使用了排他锁也不行，因为记录不存在，行锁无法生效。请问这种情况，在 MySQL 锁层面有没有办法处理？

首先，我要先赞一下这样的提问方式。虽然极客时间现在的评论区还不能追加评论，但如果大家能够一次留言就把问题讲清楚的话，其实影响也不大。所以，我希望你在留言提问的时候，也能借鉴这种方式。

接下来，我把 @ithunter 同学说的表模拟出来，方便我们讨论。

 复制代码

```
1 CREATE TABLE `like` (  
2   `id` int(11) NOT NULL AUTO_INCREMENT,  
3   `user_id` int(11) NOT NULL,  
4   `liker_id` int(11) NOT NULL,  
5   PRIMARY KEY (`id`),  
6   UNIQUE KEY `uk_user_id_liker_id` (`user_id`,`liker_id`)  
7 ) ENGINE=InnoDB;  
8  
9 CREATE TABLE `friend` (  
10  `id` int(11) NOT NULL AUTO_INCREMENT,  
11  `friend_1_id` int(11) NOT NULL,  
12  `friend_2_id` int(11) NOT NULL,  
13  UNIQUE KEY `uk_friend` (`friend_1_id`,`friend_2_id`)  
14  PRIMARY KEY (`id`)  
15 ) ENGINE=InnoDB;
```

虽然这个题干中，并没有说到 friend 表的索引结构。但我猜测 friend\_1\_id 和 friend\_2\_id 也有索引，为便于描述，我给加上唯一索引。

顺便说明一下，“like”是关键字，我一般不建议使用关键字作为库名、表名、字段名或索引名。

我把他的疑问翻译一下，在并发场景下，同时有两个人，设置为关注对方，就可能导致无法成功加为朋友关系。

现在，我用你已经熟悉的时刻顺序表的形式，把这两个事务的执行语句列出来：

session 1 (操作逻辑：A喜欢B)	session 2 (操作逻辑：B喜欢A)
begin; select * from `like` where user_id = B and liker_id = A; (返回空)	
	begin; select * from `like` where user_id = A and liker_id = B; (返回空)
	insert into `like` (user_id, liker_id) values(B, A);
insert into `like` (user_id, liker_id) values(A, B);	
commit;	
	commit;

图 3 并发“喜欢”逻辑操作顺序

由于一开始 A 和 B 之间没有关注关系，所以两个事务里面的 select 语句查出来的结果都是空。

因此，session 1 的逻辑就是“既然 B 没有关注 A，那就只插入一个单向关注关系”。session 2 也同样是这个逻辑。

这个结果对业务来说就是 bug 了。因为在业务设定里面，这两个逻辑都执行完成以后，是应该在 friend 表里面插入一行记录的。

如提问里面说的，“第 1 步即使使用了排他锁也不行，因为记录不存在，行锁无法生效”。不过，我想到了另外一个方法，来解决这个问题。

首先，要给“like”表增加一个字段，比如叫作 relation\_ship，并设为整型，取值 1、2、3。

- 值是 1 的时候，表示 user\_id 关注 liker\_id;
- 值是 2 的时候，表示 liker\_id 关注 user\_id;
- 值是 3 的时候，表示互相关注。

然后，当 A 关注 B 的时候，逻辑改成如下所示的样子：

应用代码里面，比较 A 和 B 的大小，如果  $A < B$ ，就执行下面的逻辑

```
1 mysql> begin; /* 启动事务 */
2 insert into `like`(user_id, liker_id, relation_ship) values(A, B, 1) on duplicate key update rel
3 select relation_ship from `like` where user_id=A and liker_id=B;
4 /* 代码中判断返回的 relation_ship,
5  如果是 1, 事务结束, 执行 commit
6  如果是 3, 则执行下面这两个语句:
7  */
8 insert ignore into friend(friend_1_id, friend_2_id) values(A,B);
9 commit;
```

[复制代码](#)

如果  $A > B$ , 则执行下面的逻辑

```
1 mysql> begin; /* 启动事务 */
2 insert into `like`(user_id, liker_id, relation_ship) values(B, A, 2) on duplicate key update rel
3 select relation_ship from `like` where user_id=B and liker_id=A;
4 /* 代码中判断返回的 relation_ship,
5  如果是 1, 事务结束, 执行 commit
6  如果是 3, 则执行下面这两个语句:
7  */
8 insert ignore into friend(friend_1_id, friend_2_id) values(B,A);
9 commit;
```

[复制代码](#)

这个设计里, 让 “like” 表里的数据保证  $user\_id < liker\_id$ , 这样不论是 A 关注 B, 还是 B 关注 A, 在操作 “like” 表的时候, 如果反向的关系已经存在, 就会出现行锁冲突。

然后, insert ... on duplicate 语句, 确保了在事务内部, 执行了这个 SQL 语句后, 就强行占住了这个行锁, 之后的 select 判断 relation\_ship 这个逻辑时就确保了是在行锁保护下的读操作。

操作符 “|” 是逻辑或, 连同最后一句 insert 语句里的 ignore, 是为了保证重复调用时的幂等性。

这样, 即使在双方 “同时” 执行关注操作, 最终数据库里的结果, 也是 like 表里面有一条关于 A 和 B 的记录, 而且 relation\_ship 的值是 3, 并且 friend 表里面也有了 A 和 B 的这条记录。

不知道你会不会吐槽: 之前明明还说尽量不要使用唯一索引, 结果这个例子一上来我就创建了两个。这里我要再和你说明一下, 之前文章我们讨论的, 是在 “业务开发保证不会插入重复记录” 的情况下, 着重要解决性能问题的时候, 才建议尽量使用普通索引。

而像这个例子里, 按照这个设计, 业务根本就是保证 “我一定会插入重复数据, 数据库一定要要有唯一性约束”, 这时就没啥好说的了, 唯一索引建起来吧。

## 小结

这是专栏的第一篇答疑文章。

我针对前 14 篇文章，大家在评论区中的留言，从中摘取了关于日志和索引的相关问题，串成了今天这篇文章。这里我也要再和你说一声，有些我答应在答疑文章中进行扩展的话题，今天这篇文章没来得及扩展，后续我会再找机会为你解答的。所以，篇幅所限，评论区见吧。

最后，虽然这篇是答疑文章，但课后问题还是要有的。

我们创建了一个简单的表 t，并插入一行，然后对这一行做修改。

```
1 mysql> CREATE TABLE `t` (  
2   `id` int(11) NOT NULL primary key auto_increment,  
3   `a` int(11) DEFAULT NULL  
4 ) ENGINE=InnoDB;  
5 insert into t values(1,2);
```

[复制代码](#)

这时候，表 t 里有唯一的一行数据 (1,2)。假设，我现在要执行：

```
1 mysql> update t set a=2 where id=1;
```

[复制代码](#)

你会看到这样的结果：

```
mysql> update t set a=2 where id = 1;  
Query OK, 0 rows affected (0.00 sec)  
Rows matched: 1   Changed: 0   Warnings: 0
```

结果显示，匹配 (rows matched) 了一行，修改 (Changed) 了 0 行。

仅从现象上看，MySQL 内部在处理这个命令的时候，可以有以下三种选择：

1. 更新都是先读后写的，MySQL 读出数据，发现 a 的值本来就是 2，不更新，直接返回，执行结束；
2. MySQL 调用了 InnoDB 引擎提供的“修改为 (1,2)”这个接口，但是引擎发现值与原来相同，不更新，直接返回；
3. InnoDB 认真执行了“把这个值修改成 (1,2)”这个操作，该加锁的加锁，该更新的更新。

你觉得实际情况会是以上哪种呢？你可否用构造实验的方式，来证明你的结论？进一步的，可以思考一下，MySQL 为什么要选择这种策略呢？

你可以把你的验证方法和思考写在留言区里，我会在下一篇文章的末尾和你讨论这个问题。感谢你的收听，也欢迎你把这篇文章分享给更多的朋友一起阅读。

## 上期问题时间

上期的问题是，用一个计数表记录一个业务表的总行数，在往业务表插入数据的时候，需要给计数值加 1。

逻辑实现上是启动一个事务，执行两个语句：

1. insert into 数据表;
2. update 计数表, 计数值加 1。

从系统并发能力的角度考虑，怎么安排这两个语句的顺序。

这里，我直接复制 @阿建 的回答过来供你参考：

并发系统性能的角度考虑，应该先插入操作记录，再更新计数表。


知识点在 [《行锁功过：怎么减少行锁对性能的影响？》](#)

因为更新计数表涉及到行锁的竞争，先插入再更新能最大程度地减少事务之间的锁等待，提升并发度。

评论区有同学说，应该把 update 计数表放后面，因为这个计数表可能保存了多个业务表的计数值。如果把 update 计数表放到事务的第一个语句，多个业务表同时插入数据的话，等待时间会更长。

这个答案的结论是对的，但是理解不太正确。即使我们用一个计数表记录多个业务表的行数，也肯定会给表名字段加唯一索引。类似于下面这样的表结构：

```
1 CREATE TABLE `rows_stat` (  
2   `table_name` varchar(64) NOT NULL,  
3   `row_count` int(10) unsigned NOT NULL,  
4   PRIMARY KEY (`table_name`)  
5 ) ENGINE=InnoDB;
```

 复制代码

在更新计数表的时候，一定会传入 where table\_name=\$table\_name，使用主键索引，更新加行锁只会锁在一行上。

而在不同业务表插入数据，是更新不同的行，不会有行锁。

评论区留言点赞板：

@北天魔狼、@斜面镜子 Bil 和 @Bin 等同学，都给出了正确答案；

@果然如此 同学提了一个好问题，虽然引入事务，避免看到“业务上还没提交的更新”，但是 Redis 的计数被提前看到了。核心原因还是两个系统，不支持一致性视图；

@ 帆帆帆帆帆帆帆 同学的问题提醒了大家，count(id) 也是可以走普通索引得到的。



# MySQL 实战 45 讲

从原理到实战，丁奇带你搞懂 MySQL

林晓斌

网名丁奇  
前阿里资深技术专家



新版升级：点击「👤 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

©版权归极客邦科技所有，未经许可不得转载

上一篇 14 | count(\*)这么慢，我该怎么办？

写留言

## 精选留言



某、人

👍 2

老师提几个问题：

- 1.事务在执行过程中，binlog是否像redo log一样记录到binlog\_cache里？
- 2.为什么把redo log buffer设置成全局参数，binlog cache设置为事务级别参数？
- 3.为什么一般是binlog落盘比redo log更耗时？
- 4.如果sync为1，dump线程是等到binlog 成功flush，再从binlog cache中把binlog event 发送给从库？如果非1,是在最后xid写入就从binlog cache中把binlog event发送给从库？

2018-12-17



天王

👍 0



我认为是第一种。第二种如果出现了脏页，内存中的数据是和磁盘中数据不一致，最后还是会被磁盘中的数据页覆盖。第三种，内存中的数据页flush到磁盘，是有机制的，更新内存里，再统一把内存刷到磁盘上，性能更好，第三种性能不好。所以选择第一种

2018-12-17



amazon1011

0

"这个事务要往两个表中插入记录，插入数据的过程中，生成的日志都得先保存起来，但又不能在还没 commit 的时候就直接写到 redo log 文件里。"老师，如果是一个长事务占用了redo log buffer 的1/2的时候且事务还没有完成，后台线程就直接刷到redo log file 吧。

2018-12-17



臧天霸

0

老师，我把前面的问题挪过来，烦请解惑：

1.alter table t engine = innodb;后表及索引的统计信息也会跟着自动重建吗？联想到在oracle中迁移完数据，没有及时做统计信息收集所引发的事故了.....

2.如果表及索引的统计信息没有跟着自动重建，那此时会使用表及索引旧的统计信息？或者是达到统计信息重新收集阈值后再收集？

2018-12-17



峰

0

追问5中，老师说到binlog没有能力恢复数据页，但这和它能不能恢复数据库是两回事呀，本质上都是更新在存储层次上的几种展现吗？一个是记录顶层数据页的变化，一个是行变化。

业务设计问题，这个问题其实抽象点就是写倾斜问题，类似的问题基本都符合先select读个什么东东出来，作为条件，然后再决定是否写入。老师的解决方法核心是实体化冲突，把原先不能加锁的无记录，人为引入可加锁的对象。当然还有种解决方式就是索引区间锁的方式，当然这是串行化级别下的东东了。这个问题让我好好回顾了一遍隔离，还是很有收获的，也希望老师能够把47个问题种剩下的也发出来，不需要完整的解答，点点就好。

今日问题：在两个事务中，一个执行更新后（没提交），发现另一个事务就没办法对相同行进行更新了。所以处理方式3。至于为什么这么做，如果说更新不改变就不加锁，（1）调用方就可能需要确定这个更新有没有加上锁，才进行相关的操作。（2）对代码引入不必要的复杂度。（3）想的头大了，想不出来了。。。😓

2018-12-17