

# 06 | 全局锁和表锁：给表加个字段怎么有这么多阻碍？

2018-11-26 林晓斌



今天我要跟你聊聊**MySQL**的锁。数据库锁设计的初衷是处理并发问题。作为多用户共享的资源，当出现并发访问的时候，数据库需要合理地控制资源的访问规则。而锁就是用来实现这些访问规则的重要数据结构。

根据加锁的范围，**MySQL**里面的锁大致可以分成全局锁、表级锁和行锁三类。今天这篇文章，我会和你分享全局锁和表级锁。而关于行锁的内容，我会留着在下一篇文章中再和你详细介绍。

这里需要说明的是，锁的设计比较复杂，这两篇文章不会涉及锁的具体实现细节，主要介绍的是碰到锁时的现象和其背后的原理。

## 全局锁

顾名思义，全局锁就是对整个数据库实例加锁。**MySQL**提供了一个加全局读锁的方法，命令是**Flush tables with read lock (FTWRL)**。当你需要让整个库处于只读状态的时候，可以使用这个命令，之后其他线程的以下语句会被阻塞：数据更新语句（数据的增删改）、数据定义语句（包括建表、修改表结构等）和更新类事务的提交语句。

全局锁的典型使用场景是，做全库逻辑备份。也就是把整库每个表都**select**出来存成文本。

以前有一种做法，是通过**FTWRL**确保不会有其他线程对数据库做更新，然后对整个库做备份。注意，在备份过程中整个库完全处于只读状态。

但是让整库都只读，听上去就很危险：

- 如果你在主库上备份，那么在备份期间都不能执行更新，业务基本上就得停摆；
- 如果你在从库上备份，那么备份期间从库不能执行主库同步过来的binlog，会导致主从延迟。

看来加全局锁不太好。但是细想一下，备份为什么要加锁呢？我们来看一下不加锁会有什么问题。

假设你现在要维护“极客时间”的购买系统，关注的是用户账户余额表和用户课程表。

现在发起一个逻辑备份。假设备份期间，有一个用户，他购买了一门课程，业务逻辑里就要扣掉他的余额，然后往已购课程里面加上一门课。

如果时间顺序上是先备份账户余额表(`u_account`)，然后用户购买，然后备份用户课程表(`u_course`)，会怎么样呢？你可以看一下这个图：

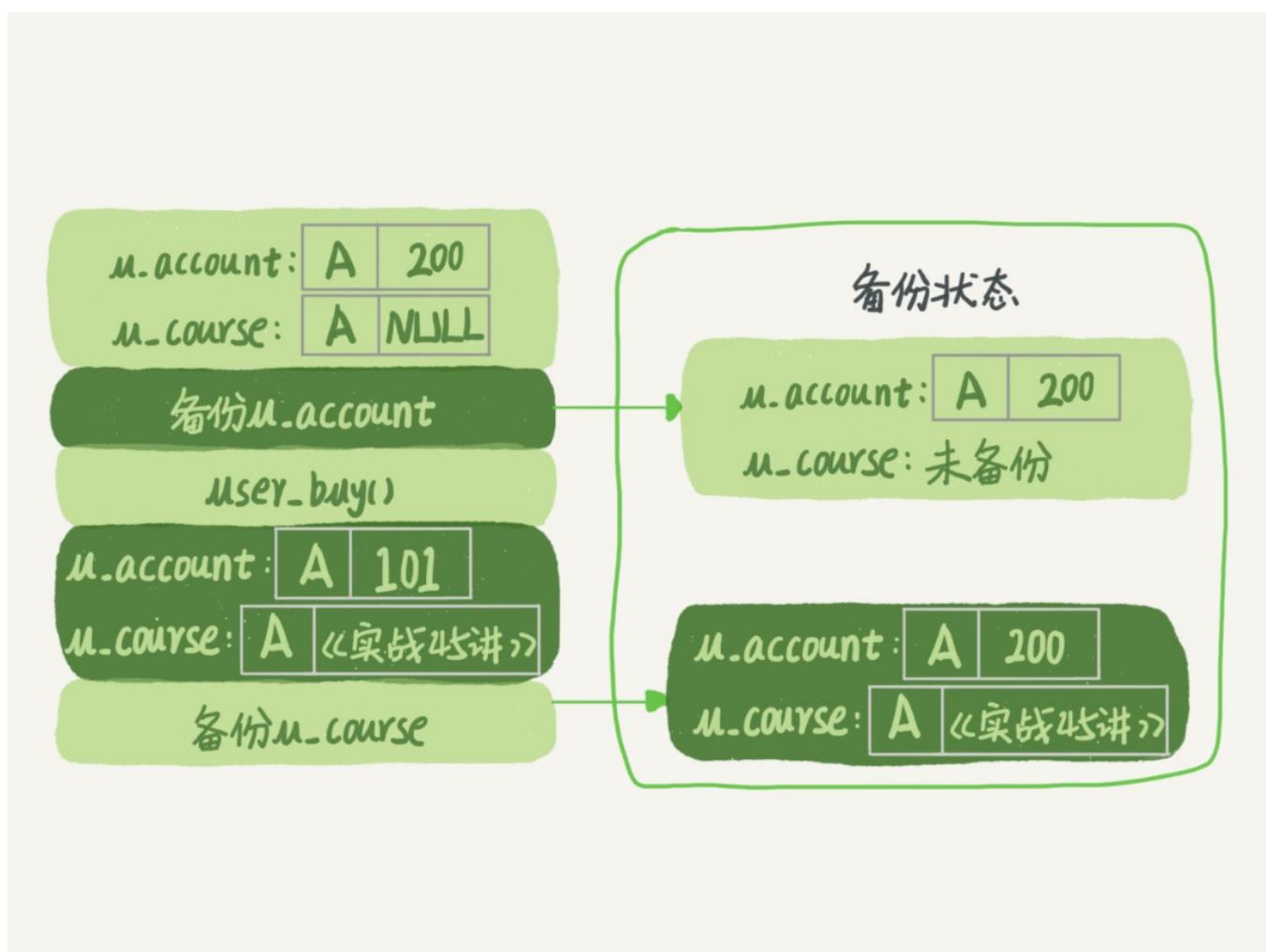


图1 业务和备份状态图

可以看到，这个备份结果里，用户A的数据状态是“账户余额没扣，但是用户课程表里面已经多了一门课”。如果后面用这个备份来恢复数据的话，用户A就发现，自己赚了。

作为用户可别觉得这样可真好啊，你可以试想一下：如果备份表的顺序反过来，先备份用户课程

表再备份账户余额表，又可能会出现什么结果？

也就是说，不加锁的话，备份系统备份的得到的库不是一个逻辑时间点，这个视图是逻辑不一致的。

说到视图你肯定想起来了，我们在前面讲事务隔离的时候，其实是一个方法能够拿到一致性视图的，对吧？

是的，就是在可重复读隔离级别下开启一个事务。

备注：如果你对事务隔离级别的概念不是很清晰的话，可以再回顾一下第3篇文章 [《事务隔离：为什么你改了我还看不见？》](#) 中的相关内容。

官方自带的逻辑备份工具是`mysqldump`。当`mysqldump`使用参数`-single-transaction`的时候，导数据之前就会启动一个事务，来确保拿到一致性视图。而由于MVCC的支持，这个过程中数据是可以正常更新的。

你一定在疑惑，有了这个功能，为什么还需要`FTWRL`呢？**一致性读是好，但前提是引擎要支持这个隔离级别。**比如，对于`MyISAM`这种不支持事务的引擎，如果备份过程中有更新，总是只能取到最新的数据，那么就破坏了备份的一致性。这时，我们就需要使用`FTWRL`命令了。

所以，`single-transaction`方法只适用于所有的表使用事务引擎的库。如果有的表使用了不支持事务的引擎，那么备份就只能通过`FTWRL`方法。这往往是DBA要求业务开发人员使用`InnoDB`替代`MyISAM`的原因之一。

你也许会问，既然要全库只读，为什么不使用`set global readonly=true`的方式呢？确实`readonly`方式也可以让全库进入只读状态，但我还是会建议你用`FTWRL`方式，主要有两个原因：

- 一是，在有些系统中，`readonly`的值会被用来做其他逻辑，比如用来判断一个库是主库还是备库。因此，修改`global`变量的方式影响面更大，我不建议你使用。
- 二是，在异常处理机制上有差异。如果执行`FTWRL`命令之后由于客户端发生异常断开，那么`MySQL`会自动释放这个全局锁，整个库回到可以正常更新的状态。而将整个库设置为`readonly`之后，如果客户端发生异常，则数据库就会一直保持`readonly`状态，这样会导致整个库长时间处于不可写状态，风险较高。

业务的更新不只是增删改数据（`DML`），还有可能是加字段等修改表结构的操作（`DDL`）。不论是哪种方法，一个库被全局锁上以后，你要对里面任何一个表做加字段操作，都是会被锁住的。

但是，即使没有被全局锁住，加字段也不是就能一帆风顺的，因为你还会碰到接下来我们要介绍的表级锁。

## 表级锁

MySQL里面表级别的锁有两种：一种是表锁，一种是元数据锁（meta data lock，MDL）。

表锁的语法是 **lock tables ...read/write**。与FTWRL类似，可以用**unlock tables**主动释放锁，也可以在客户端断开的时候自动释放。需要注意，**lock tables**语法除了会限制别的线程的读写外，也限定了本线程接下来的操作对象。

举个例子，如果在某个线程A中执行**lock tables t1 read, t2 write**；这个语句，则其他线程写t1、读写t2的语句都会被阻塞。同时，线程A在执行**unlock tables**之前，也只能执行读t1、读写t2的操作。连写t1都不允许，自然也不能访问其他表。

在还没有出现更细粒度的锁的时候，表锁是最常用的处理并发的方式。而对于InnoDB这种支持行锁的引擎，一般不使用**lock tables**命令来控制并发，毕竟锁住整个表的影响面还是太大。

另一类表级的锁是**MDL**（**metadata lock**）。MDL不需要显式使用，在访问一个表的时候会被自动加上。MDL的作用是，保证读写的正确性。你可以想象一下，如果一个查询正在遍历一个表中的数据，而执行期间另一个线程对这个表结构做变更，删了一列，那么查询线程拿到的结果跟表结构对不上，肯定是不行的。

因此，在MySQL 5.5版本中引入了**MDL**，当对一个表做增删改查操作的时候，加**MDL**读锁；当要对表做结构变更操作的时候，加**MDL**写锁。

- 读锁之间不互斥，因此你可以有多个线程同时对一张表增删改查。
- 读写锁之间、写锁之间是互斥的，用来保证变更表结构操作的安全性。因此，如果有两个线程要同时给一个表加字段，其中一个要等另一个执行完才能开始执行。

虽然**MDL**锁是系统默认会加的，但却是你不能忽略的一个机制。比如下面这个例子，我经常看到有人掉到这个坑里：给一个小表加个字段，导致整个库挂了。

你肯定知道，给一个表加字段，或者修改字段，或者加索引，需要扫描全表的数据。在对大表操作的时候，你肯定会特别小心，以免对线上服务造成影响。而实际上，即使是小表，操作不慎也会出问题。我们来看一下下面的操作序列，假设表t是一个小表。

备注：这里的实验环境是MySQL 5.6。

session A	session B	session C	session D
begin; select * from t limit 1;			
	select * from t limit 1;		
		alter table t add f int; (blocked)	
			select * from t limit 1; (blocked)

我们可以看到**session A**先启动，这时候会对表t加一个**MDL**读锁。由于**session B**需要的也是**MDL**读锁，因此可以正常执行。

之后**session C**会被**blocked**，是因为**session A**的**MDL**读锁还没有释放，而**session C**需要**MDL**写锁，因此只能被阻塞。

如果只有**session C**自己被阻塞还没什么关系，但是之后所有要在表t上新申请**MDL**读锁的请求也会被**session C**阻塞。前面我们说了，所有对表的增删改查操作都需要先申请**MDL**读锁，就都被锁住，等于这个表现在完全不可读写了。

如果某个表上的查询语句频繁，而且客户端有重试机制，也就是说超时后会再起一个新**session**再请求的话，这个库的线程很快就会爆满。

你现在应该知道了，事务中的**MDL**锁，在语句执行开始时申请，但是语句结束后并不会马上释放，而会等到整个事务提交后再释放。

基于上面的分析，我们来讨论一个问题，如何安全地给小表加字段？

首先我们要解决长事务，事务不提交，就会一直占着**MDL**锁。在MySQL的**information\_schema**库的 **innodb\_trx** 表中，你可以查到当前执行中的事务。如果你要做**DDL**变更的表刚好有长事务在执行，要考虑先暂停**DDL**，或者**kill**掉这个长事务。

但考虑一下这个场景。如果你要变更的表是一个热点表，虽然数据量不大，但是上面的请求很频繁，而你不得不加个字段，你该怎么做呢？

这时候`kill`可能未必管用，因为新的请求马上就来了。比较理想的机制是，在`alter table`语句里面设定等待时间，如果在这个指定的等待时间里面能够拿到`MDL`写锁最好，拿不到也不要阻塞后面的业务语句，先放弃。之后开发人员或者`DBA`再通过重试命令重复这个过程。

`MariaDB`已经合并了`AliSQL`的这个功能，所以这两个开源分支目前都支持`DDL NOWAIT/WAIT n`这个语法。

```
ALTER TABLE tbl_name NOWAIT add column ...
ALTER TABLE tbl_name WAIT N add column ...
```

## 小结

今天，我跟你介绍了`MySQL`的全局锁和表级锁。

全局锁主要用在逻辑备份过程中。对于全部是`InnoDB`引擎的库，我建议你选择使用`-single-transaction`参数，对应用会更友好。

表锁一般是在数据库引擎不支持行锁的时候才会被用到的。如果你发现你的应用程序里有`lock tables`这样的语句，你需要追查一下，比较可能的情况是：

- 要么是你的系统现在还在用`MyISAM`这类不支持事务的引擎，那要安排升级换引擎；
- 要么是你的引擎升级了，但是代码还没升级。我见过这样的情况，最后业务开发就是把`lock tables`和`unlock tables`改成`begin`和`commit`，问题就解决了。

`MDL`会直到事务提交才释放，在做表结构变更的时候，你一定要小心不要导致锁住线上查询和更新。

最后，我给你留一个问题吧。备份一般都会在备库上执行，你在用`-single-transaction`方法做逻辑备份的过程中，如果主库上的一个小表做了一个`DDL`，比如给一个表上加了一列。这时候，从备库上会看到什么现象呢？

你可以把你的思考和观点写在留言区里，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。

说明：这篇文章没有介绍到物理备份，物理备份会有一篇单独的文章。

## 上期问题时间

上期的问题是关于对联合主键索引和`InnoDB`索引组织表的理解。

我直接贴@老杨同志 的回复略作修改如下（我修改的部分用橙色标出）：

表记录

-a--|-b--|-c--|-d--

123d

132d

143d

213d

222d

234d

主键 a, b 的聚簇索引组织顺序相当于 `order by a,b`，也就是先按 a 排序，再按 b 排序，c 无序。

索引 ca 的组织是先按 c 排序，再按 a 排序，同时记录主键

-c--|-a--|-主键部分b--（注意，这里不是ab，而是只有b）

213

222

312

314

321

423

这个跟索引c的数据是一模一样的。

索引 cb 的组织是先按 c 排序，在按 b 排序，同时记录主键

-c--|-b--|-主键部分a--（同上）

222

231

312

321

341

432

所以，结论是ca可以去掉，cb需要保留。

评论区留言点赞：

@浪里白条 帮大家总结了复习要点；

@约书亚 的问题里提到了MRR优化；

@HwangZHen 留言言简意赅。

# 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

