# 6 **全局锁和表锁 ：给表加个字段怎么有这么多阻碍？**

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，

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）和更新类事务的语句。

全局锁的典型使用场景是：做全库逻辑备份.也就是把整个库每个表都加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，在备份过程中真个库处于只读状态。**

但是让库处于只读状态，听上去很危险

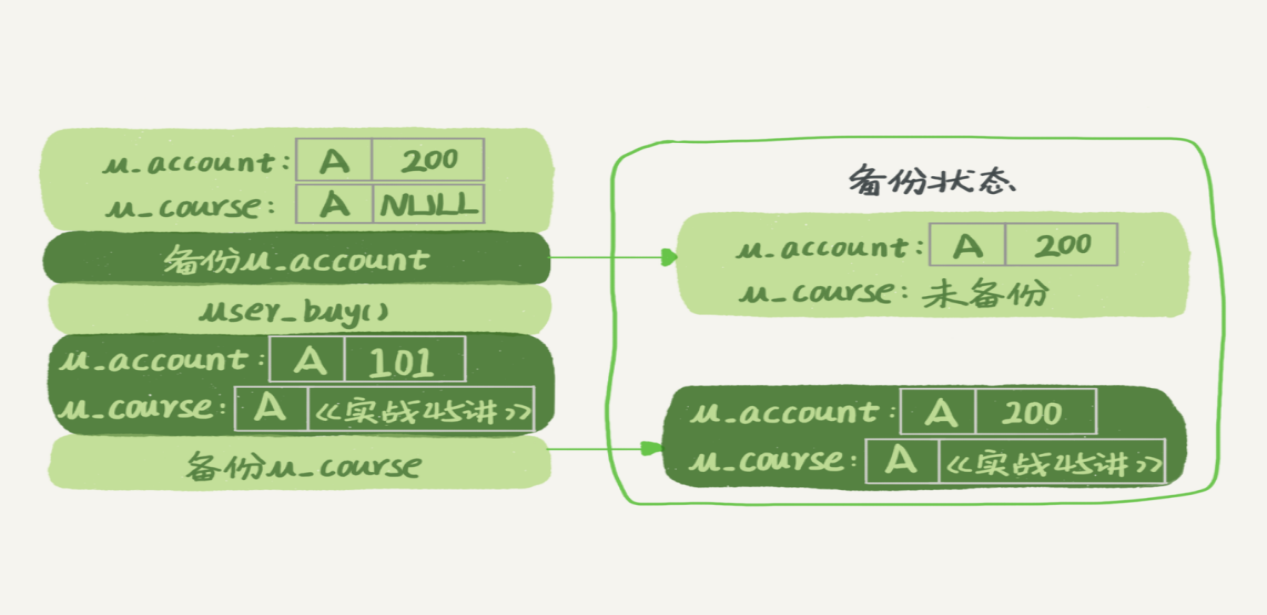
1. 如果在主库上备份，那么备份期间不能执行更新，业务基本上就得停摆
2. 如果在从库上备份，那么备份期间从库不能执行主库同步过来的binlog,会导致主从延迟

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

假设现在维护“极客时间”的购买系统，关注的是用户账户余额和用户课程表。

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

如果时间顺序上先备份账户余额，然后用户购买，然后备份用户课程表，会怎么样？



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”，但是用户课程表里已经多了一门课，如果来面用这个备份来恢复数据，用户A就发现，自己赚了。

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，有可能出现什么样的结果呢？

也就是说，不加锁的话，备份系统得到的库不是一个逻辑时间点，这个是图是逻辑不一致的。

说到视图，我们前面将事务隔离级别，就是一个方法可以拿到一致性视图。

是的，就是在可重复读隔离级别下开启一个事务。

官方自带的逻辑备份工具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支持，这个过程中数据可以是正常更新的。

有了这个功能，为什么还需要FTWRL呢？一致性读是好，但前提是是引擎支持这个隔离级别，。不如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，就破坏了备份的一致性，此时，就需要使用FTWRL命令了。

**所以，single-transaction方法值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代替MyISAM的原因之一。

既然要全库只读，为什么不使用 set global readonly=true的方式呢？确实reanonly方式也可以让全库进入只读 状态，但我还是会建议用FTWRL方式，主要有连个原因。

一是，在有些系统中，readonly的值会被用来处理其他逻辑，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，不建议使用

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（DML）,不论是那种方法。一个库被全局锁上以后，要对里面任何一个表做加字段操作，都会被锁住的。

但是，即使没有被全局锁住，加字段也不是就能一帆风顺的，因为还会有表级锁。

**表级锁**

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

表的语法是 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，也会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时候，表锁是最常用个的处理并发的方式，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。

另一类是表级的锁MDL（meta data lock）.MDL不需要显示使用，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，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更表结构对不上，跟定是不行的。

因此，在MySQL 5.5版本中引入MDL,当对一个表做增删改查操作的时候，加MDL读锁，

当要对表结构变更操作的时候，加MDL写锁。

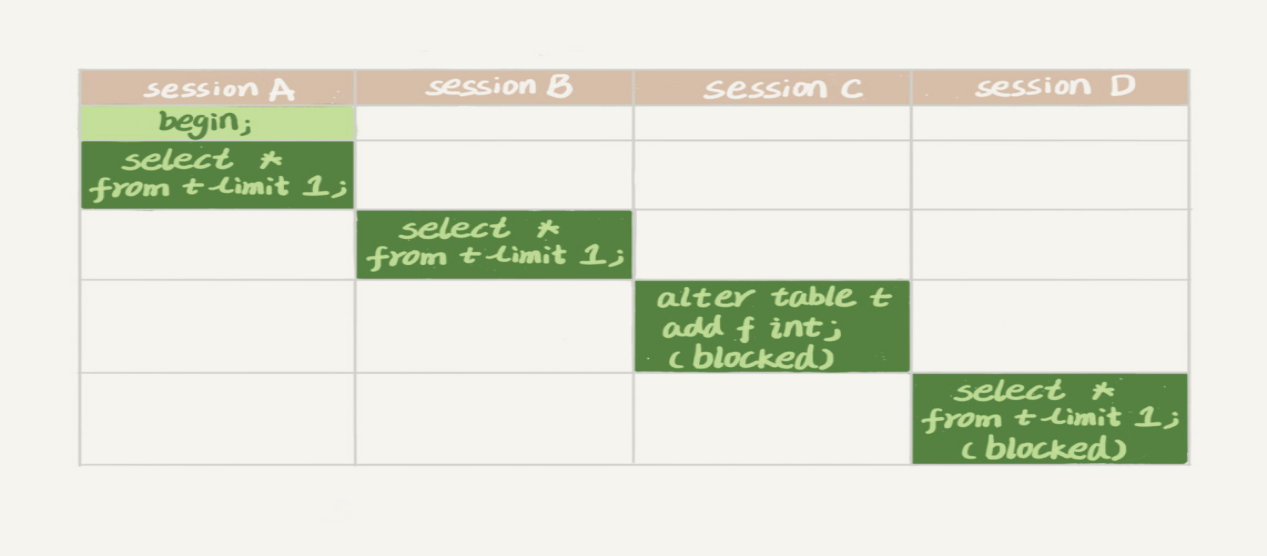
1.读锁之间不互斥，因此可以有多个线程同队对一个表增删改查

2.读写锁之间、写锁之间是互斥的，用来保证变更表结构的操作的安全性，因此，如果连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毕完才能开始执行。

虽然MDL锁是系统默认会加的，但是不能忽略一个机制，比如下面这个例子，经常看到有人调到这个坑里，给一个小表加字段，导致整个库挂了。

给一个表加字段，或者修改字段，或加索引，需要扫描全表的数据，在对大表操作的时候，肯定会小心，以免对线上服务造成影响，即使是小表，操作不慎也会出问题

假设表t是一个小表 -- 实验环境是MySQL5.6

我们可以看到sessionA先启动，这时候会对表t加一个MDL读锁，由于sessionB需要的也是MDL读锁，因此可以正常运行。

之后sessionC会被blocked，是因为SessionA的MDL读锁还没有释放，而sessionC需要读写锁，因此只能被阻塞。

如果只有sessionC自己被阻塞还什么关系，但是之后所有要对表t上新申请MDL读锁的请求也会被sessionC阻塞，前面说了，所有对表的增删改查操作都需要先申请MDL读锁，

就都被锁住了，等于这表现在完全不可读写了。

如果某个表上的查询语句频繁，而客户端有重试机制，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。

现在知道了，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，而是等到整个事务提交后在释放。

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，**要解决长事务，事务不提交，就会一直站着MDL锁，在MySQL的information\_schema库的innodb\_trx表中，可以查到当前执行的事务，如果要做DDL变更的表刚好有长事务进行，要考虑先暂停DDL,或者kill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，而不得不加个字段，该怎样做？

这个时候kill未必管用，因为新的请求马上来，比较理想的的机制是，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在通过重试命令重复这个过程。

ALTER TABLE tbl\_name NOWAIT add column ...

ALTER TABLE tbl\_name WAIT N add column ...

**讨论题**

备份一般都会在备库上执行，你在 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DDL,比如给一个表加上了一列，这时候从备库上看到什么现象呢？

Q1:SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;

Q2:START TRANSACTION WITH CONSISTENT SNAPSHOT；

/\* other tables \*/

Q3:SAVEPOINT sp;

/\* 时刻 1 \*/

Q4:show create table `t1`;

/\* 时刻 2 \*/

Q5:SELECT \* FROM `t1`;

/\* 时刻 3 \*/

Q6:ROLLBACK TO SAVEPOINT sp;

/\* 时刻 4 \*/

/\* other tables \*/

在备份开始的时候，为了确保RR(可重复读)隔离级别，在设置一次RR隔离级别（Q1）

设置一个保存点，这个很重要（Q3）;

Show create是为了拿到表结构（Q4）,然后正式导数据（Q5）,回滚到SAVEPOINT sp,在这里的作用是释放t1的MDL锁（Q6）,当然这部分属于“超纲”

DDL从主库传过来的时间按照效果不同，我打了四个时刻，题目设定为小表，我们假定到达后，如果开始执行，则很快能够执行完成。

1. 如果在Q4语句执行之前到达，现象：没有影响，备份拿到的是DDL后的表结构
2. 如果在“时刻2”到达，则表结构被改过，Q5执行的时候，报 Table definition has changed,

Please retry transaction. 现象mysqldump终止：

1. 如果在“时刻2”和“时刻3”之间到达，mysqldump占着t1的MDL锁，binlog被阻塞，现象：主从延迟，直到Q6执行完成。
2. 从“时刻4”开始，mysqldump释放了MDL读锁， 没有影响，备份拿到是DDL前的表结构