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：

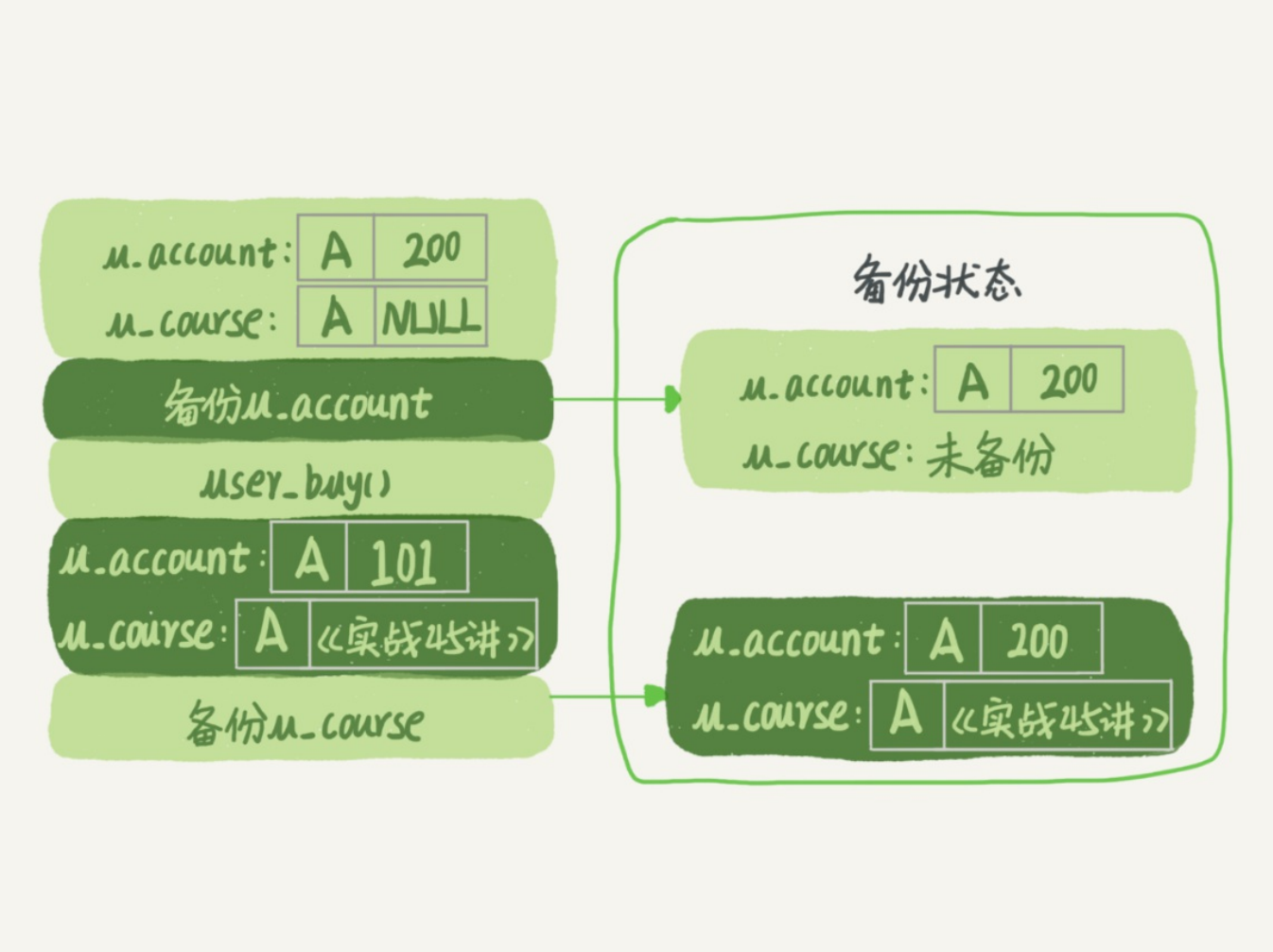


图1 业务和备份状态图

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，其实是有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。

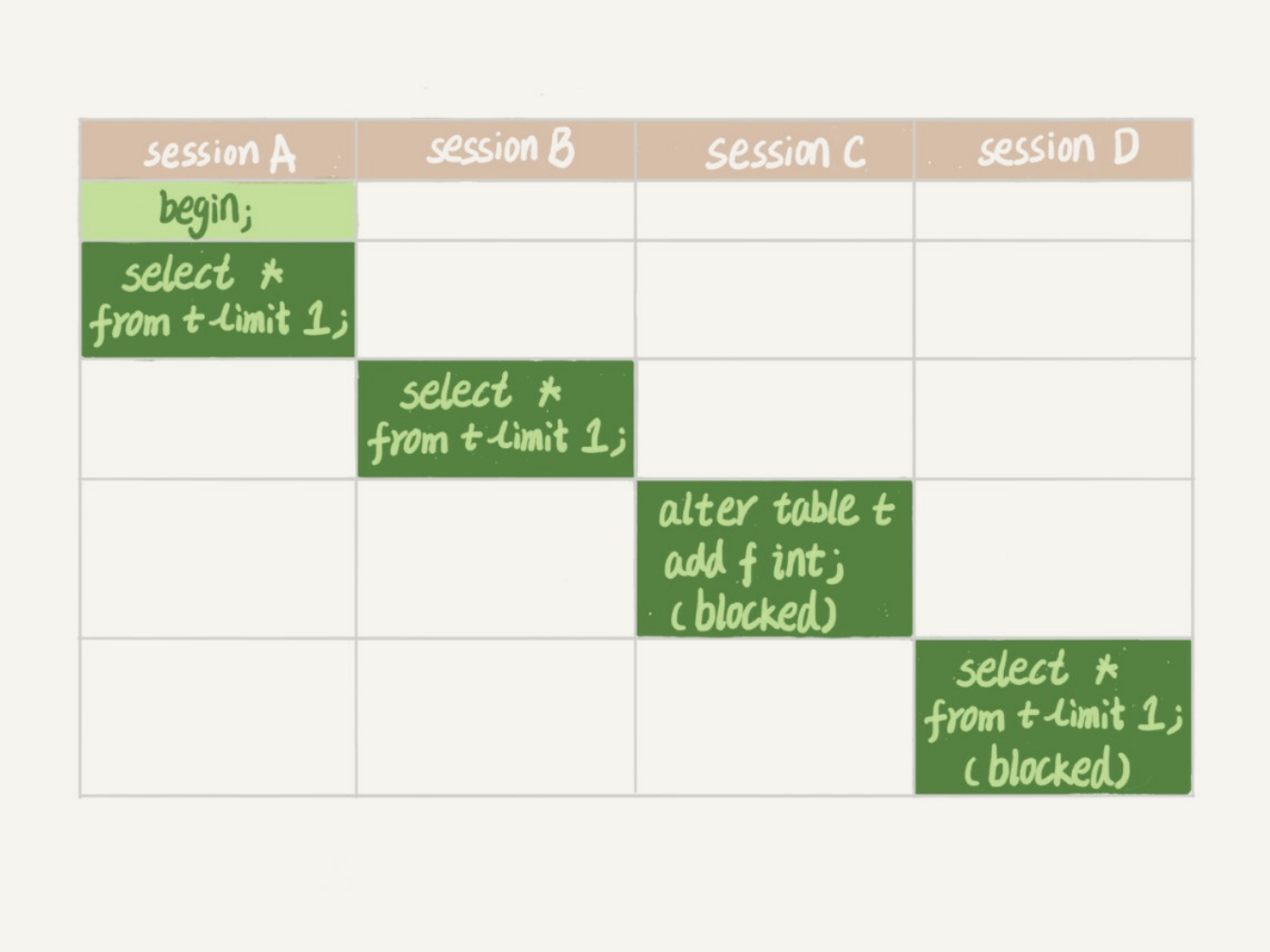
因此，在MySQL 5.5版本中引入了MDL，当对一个表做增删改查操作的时候，加MDL读锁；当要对表做结构变更操作的时候，加MDL写锁。

- 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。



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--
1 2 3 d
1 3 2 d
1 4 3 d
2 1 3 d
2 2 2 d
2 3 4 d
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）
2 1 3
2 2 2
3 1 2
3 1 4
3 2 1
4 2 3
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）
2 2 2
2 3 1
3 1 2
3 2 1
3 4 1
4 3 2

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。

精选留言

echo__陈

78

mysql 5.6不是支持online ddl了吗？也就是对表操作增加字段等功能，实际上不会阻塞读写？

2018-11-26

作者回复

Online DDL的过程是这样的：

1. 拿MDL写锁
2. 降级成MDL读锁
3. 真正做DDL
4. 升级成MDL写锁
5. 释放MDL锁

1、2、4、5如果没有锁冲突，执行时间非常短。第3步占用了DDL绝大部分时间，这期间这个表可以正常读写数据，是因此称为“online ”

我们文中的例子，是在第一步就堵住了

2018-11-26

lionetes

50

FTWRL 前有读写的话 ,FTWRL 都会等待 读写执行完毕后才执行

FTWRL 执行的时候要刷脏页的数据到磁盘,因为要保持数据的一致性 ， 理解的执行FTWRL时候是 所有事务 都提交完毕的时候

mysqldump + -single-transaction 也是保证事务的一致性,但他只针对 有支持事务 引擎,比如 innodb
所以 还是强烈建议大家在创建实例,表时候需要innodb 引擎 为好
全库只读 readonly = true 还有个情况在 slave 上 如果用户有超级权限的话 readonly 是失效的

表级别 锁 ： 一个直接就是表锁 lock table 建议不要使用, 影响太大， 另个就是 MDL 元数据锁

MDL 是并发情况下维护数据的一致性,在表上有事务的时候,不可以对元数据经行写入操作,并且这个是在server层面实现的
当你做 dml 时候增加的 MDL 读锁, update table set id=Y where id=X; 并且由于隔离级别的原因读锁之间不冲突

当你DDL 时候 增加对表的写锁, 同时操作两个alter table 操作 这个要出现等待情况。

但是 如果是 dml 与ddl 之间的交互 就更容易出现不可读写情况,这个情况容易session 爆满,session是占用内存的,也会导致内存升高
MDL 释放的情况就是 事务提交.

主库上的一个小表做了一个 DDL, 同步给slave ,由于这个时候有了先前的 single-transaction,所以slave 就会出现 该表的 锁等待, 并且slave 出现延迟

2018-11-26

作者回复

分析得很好。

尤其readonly 对 super 权限无效这句。

2018-11-26

翟毅 40

MDL作用是防止DDL和DML并发的冲突，个人感觉应该写清楚，一开始理解为select和update之间的并发。

2019-01-24

作者回复

嗯 特意写了是MDL“读锁”。

把你的留言置顶了，希望有疑问的同学能看到这个

2019-01-24

miche 12

1. 上面的那个因为mdl锁把整个库搞挂的例子里，如果用pt工具来操作，会出现同样的情况吗？
2. 那个例子里显示select语句前加了begin，是不是select的时候不加begin，就不会出现同样的情况呢？
3. online ddl 的copy方式和inplace方式，也都是需要 拿MDL写锁、降成读锁、做DDL、升成写锁、释放MDL锁吗？

2018-11-28

作者回复

1. Pt的过程也是有操作表结构的，所以会类似

2. 对，没有begin的话，这样select执行完成以后，MDL就自动释放了哦

3. 是，是否online都是第三步（结合置顶评论看哈）的区别，另外四步还是有的

2018-11-28

壹笙嘌泊 52

总结：

根据加锁范围：MySQL里面的锁可以分为：全局锁、表级锁、行级锁

一、全局锁：

对整个数据库实例加锁。

MySQL提供加全局读锁的方法：Flush tables with read lock(FTWRL)

这个命令可以使整个库处于只读状态。使用该命令之后，数据更新语句、数据定义语句和更新类事务的提交语句等操作都会被阻塞。

使用场景：全库逻辑备份。

风险：

1.如果在主库备份，在备份期间不能更新，业务停摆

2.如果在从库备份，备份期间不能执行主库同步的binlog，导致主从延迟

官方自带的逻辑备份工具mysqldump，当mysqldump使用参数--single-transaction的时候，会启动一个事务，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。

一致性读是好，但是前提是引擎要支持这个隔离级别。

如果要全库只读，为什么不使用set global readonly=true的方式？

1.在有些系统中，readonly的值会被用来做其他逻辑，比如判断主备库。所以修改global变量的方式影响太大。

2.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。

二、表级锁

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

表锁的语法是:lock tables ... read/write

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。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。

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。

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：保证读写的正确性。

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

2018-11-26

作者回复

早啊今天

2018-11-26

马涛 37

索引问题答案解释这个是不是再详细一点，我看还有人和我一样，还是搞不清楚为什么c索引和ca索引一样。

2019-03-02

作者回复

InnoDB会把主键字段放到索引定义字段后面，
当然同时也会去重。

所以，当主键是(a,b)的时候，
定义为c的索引，实际上是（c,a,b);
定义为(c,a)的索引，实际上是(c,a,b)
你看着加是相同的

ps 定义为(c,b）的索引，实际上是（c,b,a)

2019-03-02

栋能 28

没搞懂c的索引树为什么和ca是一样的. c索引树中c有序，(a,b)随意序的呀？这能代表c与ca索引树一致吗？

2018-11-26

Tony Du 26

基于文中的例子MDL（metadata lock)，自己做了一个实验（稍微有一些小改动在session D上），
session A: begin; select * from t limit 1; 最先启动sessionA
session B: begin; select * from t limit 1; 紧接着启动sessionB
session C: alter table t add f int; 然后再是启动sessionC
session D: begin; select * from t limit 1; 最后是启动sessionD
如文中例子，session A和B正常启动，然后session C被block，之后session D也被block。当把 session A 和 session B 都commit掉后，发现session C依然是block的（被 session D阻塞），只有当把 session D 也commit掉后，session C才执行下去。同样的实验，重复了三遍，结果也是一样。

从现象上看，session D会先拿到MDL读锁，当session D commit掉后，然后再是session C获得MDL写锁。请问老师，这里对于MDL锁的获取顺序（也就是说 是session C先获取MDL写锁还是session D先获取MDL读锁）有什么原则？是随机的还是有什么讲究？

另外，在一开始的获取MDL锁的阶段，session A（MDL读锁，正常执行）-> session B (MDL读锁，正常执行) -> session C (MDL写锁，被block) -> session D (MDL读锁，被MDL写锁 block) 。是不是说加MDL锁是看时间先后顺序的，一旦出现过需要获取MDL写锁（即使被block），后续再需要获取MDL读锁，则发现之前已经有获取MDL写锁（即使被block），需要获取读锁的session都会被block。感觉上像进入一个锁的”队列“，根据时间先后顺序。请问老师，这里可以更细节和深入的说明下吗？

作者回复
你这个例子里面，sessionD 被C堵住后是不能输入命令的，之后是什么动作之后，sessionD才能输入commit语句呢

我的回复：
session D被C堵住后，会卡在select * from t limit 1这句。然后当我把A和B都commit掉，session D就会执行select * from t limit 1这句，此时，session C依旧会被堵着。然后把session D commit掉，session C才能执行。实验的时候，我是把sql语句都写在MySQL workbench里的，通过workbench连服务器的，session D的commit语句是写在workbench里执行的。我的问题是，为什么是session D先获取的MDL读锁，而不是session C先获取MDL写锁，对于MDL锁的获取顺序有什么原则？是随机的还是有什么讲究？

2018-11-27

作者回复

你用MySQL 客户端试试，我跑出来是文中的顺序哈。给我一下你的MySQL 版本号和workbench版本号

2018-11-27

阿建 26

没明白为什么ca索引建出来的模型和c建出来的一样？

2018-11-26

Aurora 21

如果mysqldump 备份的是整个schema，某个小表t1只是该schema上其中有一张表
情况1：
master上对小表t1的DDL传输到slave去应用的时刻，mysqldump已经备份完了t1表的数据，此时slave 同步正常，不会有问题。

情况2：
master上对小表t1的DDL传输到slave去应用的时刻，mysqldump正在备份t1表的数据,此时会发生MDL 锁，从库上t1表的所有操作都会Hang 住。

情况3：

master 上对小表t1的DDL传输到slave去应用的时刻，mysqldump 还没对t1表进行备份，该DDL会在slave的t1表应用成功，但是当导出到t1表的时候会报“ERROR 1412 (HY000): Table definition has changed, please retry transaction” 错误，导致导出失败！

2018-11-26

Mr.Strive.Z.H.L 20

关于文中小表DDL的疑惑：
sessionC（DDL操作）被前面的sessionA和B（查询操作，获取MDL 读锁）所阻塞。这里sessionC的DDL操作任务肯定是处于等待的，后续来的sessionD（查询操作）为什么会被sessionC所阻塞？
我理解的是sessionC现在都还没有进行DDL操作，没有获取到MDL写锁，为什么sessionD会被C阻塞？难道mysql Server端对于sessionC，D有一个 队列 来决定谁先执行？

2018-12-07

作者回复

“难道”正确

2018-12-07

倪大人 12

思考题：
由于先用-single-transaction做备份，所以备份线程会启动一个事务获取MDL读锁，文中也说了“MDL 会直到事务提交才释放”，所以要一直等到备份完成主库来的DDL才会在从库执行生效，且备份的数据里并不会有新增的这个列。

再补充下，由于主库来的DDL会等待MDL写锁，所以会导致之后从库上的读写请求都阻塞，相当与文中sessionC和sessionD。

2018-11-26

知非 11

表级锁的例子中：
lock tables t1 read, t2 write
说到“线程A不能读取T2”
查了一下MySQL Reference：
WRITE lock:
The session that holds the lock can read and write the table.
Only the session that holds the lock can access the table. No other session can access it until the lock is released.
Lock requests for the table by other sessions block while the WRITE lock is held.
也就是说表级别write锁，对于本线程是可读可写的，
文章中说的线程A不能读取T2，我这里不太理解

2018-11-26

作者回复

是的，文中写错了。我刚刚修改上去了。抱歉。谢谢提醒

2018-11-26

Jeremy

10

对于思考题，索引ca里面，当a相同时，为什么b一定按照升序排列？

2018-11-26

柳树

8

既然session C blocked，拿不到写锁，那么session D为什么会被blocked呢？

2019-01-05

作者回复

如果说设计初衷，是为了防饿死吧

2019-01-05

Tony Du

8

基于文中的例子MDL（metadata lock)，自己做了一个实验（稍微有一些小改动在session D上），

session A: begin; select * from t limit 1; 最先启动sessionA

session B: begin; select * from t limit 1; 紧接着启动sessionB

session C: alter table t add f int; 然后再是启动sessionC

session D: begin; select * from t limit 1; 最后是启动sessionD

如文中例子，session A和B正常启动，然后session C被block，之后session D也被block。当把 session A 和 session B 都commit掉后，发现session C依然是block的（被 session D阻塞），只有当把 session D 也commit掉后，session C才执行下去。同样的实验，重复了三遍，结果也是一样。

从现象上看，session D会先拿到MDL读锁，当session D commit掉后，然后再是session C获得MDL写锁。请问老师，这里对于MDL锁的获取顺序（也就是说 是session C先获取MDL写锁还是session D先获取MDL读锁）有什么原则？是随机的还是有什么讲究？

另外，在一开始的获取MDL锁的阶段，session A（MDL读锁，正常执行）-> session B (MDL读锁，正常执行) -> session C (MDL写锁，被block) -> session D (MDL读锁，被MDL写锁 block) 。是不是说加MDL锁是看时间先后顺序的，一旦出现过需要获取MDL写锁（即使被block），后续再需要获取MDL读锁，则发现之前已经有获取MDL写锁（即使被block），需要获取读锁的session都会被block。感觉上像进入一个锁的”队列“，根据时间先后顺序。请问老师，这里可以更细节和深入的说明下吗？

作者回复

你这个例子里面，sessionD 被C堵住后是不能输入命令的，之后是什么动作之后，sessionD才能输入commit语句呢

我的回复：

session D被C堵住后，会卡在select * from t limit 1这句。然后当我把A和B都commit掉，session D就会执行select * from t limit 1这句，此时，session C依旧会被堵着。然后把session D commit掉，session C才能执行。实验的时候，我是把sql语句都写在MySQL workbench里的，通过workbench连服务器的，session D的commit语句是写在workbench里执行的。我的问题是，为什么是session D先获取的MDL读锁，而不是session C先获取MDL写锁，对于MDL锁的获取

顺序有什么原则？是随机的还是有什么讲究？

作者回复
你用MySQL 客户端试试，我跑出来是文中的顺序哈。给我一下你的MySQL 版本号和workbench版本号

我的回复
用MySQL客户端试过了（通过命令行），结果和我上面实验的结果是一样的。还是session D先获取的MDL读锁，而不是session C先获取MDL写锁。
MySQL版本：Ver 14.14 Distrib 5.7.17, for Linux (x86_64) using EditLine wrapper
Workbench版本：
MySQL Workbench 6.3
Version 6.3.8 build 1228 CE (64 bits) Community

这里留言回复不太方便，能否留一个邮箱或者微信号，交流起来比较方便。谢谢。

2018-11-28

Larwas 7

上一节的问题，ca 索引的数据和 c 索引一样，是因为c索引查到数据时，回表后返回的数据在主键索引已经排好，所以不需要 ca 索引做排序。做个记录，一开始没理解，刚刚想了一下理解了。

2018-12-12

keepmoving 7

请教，我们在一个mysql 5.7版本的分区大表上增加了一个字段，是在线更新表结构，原本以为会很快，结果足足等了4个多小时。按您的说法系统能正常的做交易。之前上网查原因，一种说法是mysql的表结构加字段，通过创建临时表，copy数据到临时表，再用有新增字段的临时表替换原表的方式来处理。
1、请问以上说法是对的么？
2、如果像您说的表数据更新和查询都会加mdl锁，那就应该不能处理新增交易啊？
3、现在mysql新增字段不支持设置超时时间吧？

谢谢

2018-11-26

forina 6

老师 我想咨询一个问题 ，我有一个大表t 几百万条数据，a是主键(int类型)，另外有一个索引（b,c,d），查询语句 select a from t where b='ZC1093' and c='2018-07-31' and d='AG011' limit 1000,10 执行过程使用了索引只用了0.014s,查询语句 select a from t where b='ZC1093' and c='2018-07-31' and d='AG011' order by a limit 1000,10 执行过程也用了(b,c,d)这个索引 却用了34s 完成，两条查询语句结果也都是一样的 我很疑惑索引 (b,c,d)和(b,c,d,a)不应该是等效的吗 为什么一个快一个慢？

2018-11-27

作者回复

在《”order by 是怎么工作的”》 这篇会提到这个问题哈

2018-11-27

CHeeto

5

老师，我用5.7.20以及5.6.38都做了实验，都是sessionD先要提交了之后sessionC才能够执行，是什么原因呢？

2019-02-12