**MySQL锁机制**

1. **MySQL锁概述**

锁是计算机来协调多个进程或线程并发访问某一系统的资源的机制。在数据库中，除传统的计算资源（如CPU、RAM、I/O等）的争用以外，数据也是一种供许多用户共享的资源。如何保证数据并发访问的一致性、有效性是所有数据库必须解决的一个问题，锁冲突也是影响数据库并发访问性能的一个很重要的因素。从这个角度来说，锁对数据库而言显得尤其重要，也更加复杂。

相对其他数据库而言，MySQL的锁机制比较简单，其最显著的特点是不同的存储引擎支持不同的锁机制。比如，MyISAM和MEMORY存储引擎采用的是表级锁（table-level locking）；BDB存储引擎采用的是页面锁（page-level locking），但也支持表级锁；InnoDB存储引擎既支持行级锁（row-level locking），也支持表级锁，但默认情况下是采用行级锁。MySQL的存储引擎和其锁机制如下表所示：



MySQL支持的3种锁的特性可大致归纳如下。

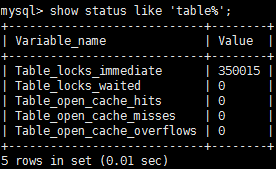
* 表级锁：开销小，加锁快；不会出现死锁；锁定粒度大，发生锁冲突的概率最高,并发度最低。
* 行级锁：开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低,并发度也最高。
* 页面锁：开销和加锁时间界于表锁和行锁之间；会出现死锁；锁定粒度界于表锁和行锁之间，并发度一般。

从上述特点可见，哪种锁更好，很难一概而论，只能根据具体应用的特点来说哪种锁更合适！仅从锁的角度来说：表级锁更适合于以查询为主，只有少量按索引条件更新数据的应用，如Web应用；而行级锁则更适合于有大量按索引条件并发更新少量不同数据，同时又有并发查询的应用，如一些在线事务处理（OLTP）系统。

1. **MyISAM锁机制**

**2.1查看数据库表锁真用情况**

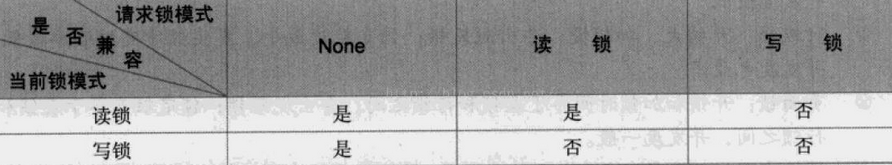
MySQL数据库中，可以通过检查table\_locks\_waited和table\_locks\_immediate两个状态变量来查看数据库表锁争用情况：



如果系统参数中，Table\_locks\_waited参数的值比较高，则说明存在着严重的表锁争用问题。

**2.2 MyISAM的锁模式**

MyISAM支持两种表锁机制：表共享读(Table Read Lock)和表独占写(Table Write Lock),其表锁兼容性如下表所示：

由上表可以分析得到以下几点：

* MyISAM表的读操作：不会阻塞其他用户对同一表的读操作，但会阻塞其他用户对同一表的写操作
* MyISAM表的写操作：会阻塞其他用户对同一表的读操作和写操作
* 共享读锁（S）之间是兼容的，但共享读锁（S）与排他写锁（X）之间，以及排他写锁（X）之间是互斥的，也就是说读和写、写和写都是串行的。

**2.3 MyISAM表锁的几点问题**

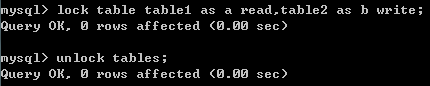
* 在一定条件下，MyISAM 允许查询和插入并发执行，我们可以利用这一点来解决应用中对同一表查询和插入的锁争用问题。系统属性concurrent\_insert专门用以控制其并发插入的行为，其值分别可以为0、1或2
* MyISAM 默认的锁调度机制是写优先，这并不一定适合所有应用，用户可以通过设置LOW\_PRIORITY\_UPDATES 系统参数，或在INSERT、UPDATE、DELETE 语句中指定LOW\_PRIORITY 选项来调节读写锁的争用
* 由于表锁的锁定粒度大，读写之间又是串行的，因此，如果更新操作较多，MyISAM表可能会出现严重的锁等待

**2.4 MyISAM的加锁机制**

MyISAM在执行查询语句前，会自动给涉及的所有表加读锁，在执行更新操作(update、delete、insert)前，会自动给涉及的所有表加写锁，这个过程并不需要用户主动干预，用户也可以使用LOCK TABLE命令显式给MyISAM表加写锁。

使用LOCK TABLE命令显示锁表需要注意的是，必须同时给所有涉及的表加锁，而且由于MySQL不支持锁升级，即执行完LOCK TABLE命令后，只能访问显示加锁的表，不能访问未加锁的表。另外，如果加的是读锁，则只能执行查询，而不能执行更新操作，自动加锁时也是如此。在处理ｓｑｌ时，ＭｙＩＳＡＭ总是一次获得ＳＱＬ语句需要的全部锁，这也是ＭｙＩＳＡＭ不会出现死锁的原因。如下是手动加读锁和写锁的例子：

当使用LOCK TABLES时，不仅需要一次锁定用到的所有表，而且，同一个表在SQL语句中出现多少次，就要通过与SQL语句中相同的别名锁定多少次，否则也会出错！



**2.5 MyISAM的锁调度**

上面提到MyISAM存储引擎的读写锁是互斥的，读写操作时串行的，那么当同一个表的一个读请求和一个写请求同时到达时，MyISAM存储引擎是如何出来的呢？答案是先处理写请求，即写进程先获得锁。不仅如此，即使读请求先到等待队列，写请求后到，写锁也会插入到读锁的前面！这是由于MySQL认为写操作比读操作重要。这也是为什么MyISAM不适合做大量更新和查询操作的应用的原因，因为大量的更新操作可能导致查询操作永远也无法获得读锁，从而永远阻塞。

以上是MySQL的默认处理方式，但是MySQL也为我们提供了几个参数，用于控制这么处理逻辑：

* 通过指定启动参数low-priority-updates，使MyISAM引擎默认给予读请求以优先的权利
* 通过执行命令SET LOW\_PRIORITY\_UPDATES=1，使该连接发出的更新请求优先级降低
* 通过指定INSERT、UPDATE、DELETE语句的LOW\_PRIORITY属性，降低该语句的优先级

另外，MySQL也提供了一种折中的办法来调节读写冲突，即给系统参数max\_write\_lock\_count设置一个合适的值，当一个表的读锁达到这个值后，MySQL就暂时将写请求的优先级降低，给读进程一定获得锁的机会。

另外需要注意的是，一些需要长时间运行的查询操作，也会使写进程“饿死”！因此，应用中应尽量避免出现长时间运行的查询操作，不要总想用一条SELECT语句来解决问题，因为这种看似巧妙的SQL语句，往往比较复杂，执行时间较长，在可能的情况下可以通过使用中间表等措施对SQL语句做一定的“分解”，使每一步查询都能在较短时间完成，从而减少锁冲突。如果复杂查询不可避免，应尽量安排在数据库空闲时段执行。

**2.6 MyISAM并发插入控制**

上面提到过MyISAM表的读和写是串行的，但这是就总体而言的。在一定条件下，MyISAM表也支持查询和插入操作的并发进行。

MyISAM存储引擎有一个系统变量concurrent\_insert，专门用以控制其并发插入的行为，其值分别可以为0、1或2。

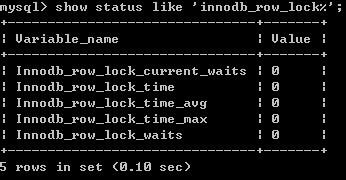
* 当concurrent\_insert设置为0时，不允许并发插入
* 当concurrent\_insert设置为1时，如果MyISAM表中没有空洞（即表的中间没有被删除的行），MyISAM允许在一个进程读表的同时，另一个进程从表尾插入记录。这也是MySQL的默认设置
* 当concurrent\_insert设置为2时，无论MyISAM表中有没有空洞，都允许在表尾并发插入记录

由上可知，我们可以通过设置concurrent\_insert来解决同一表查询和插入的锁争用，并利用OPTIMIZE TABLE语句来整理空间碎片，收回因删除记录而产生的中间空洞。

1. **InnoDB锁问题**

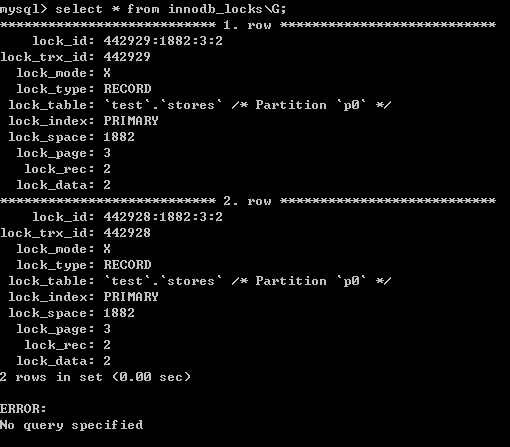
**3.1 获取innodb行锁争用情况**

可以通过检查MySQL提供的InnoDB\_row\_lock状态变量来分析查看系统上的行锁的争夺情况：

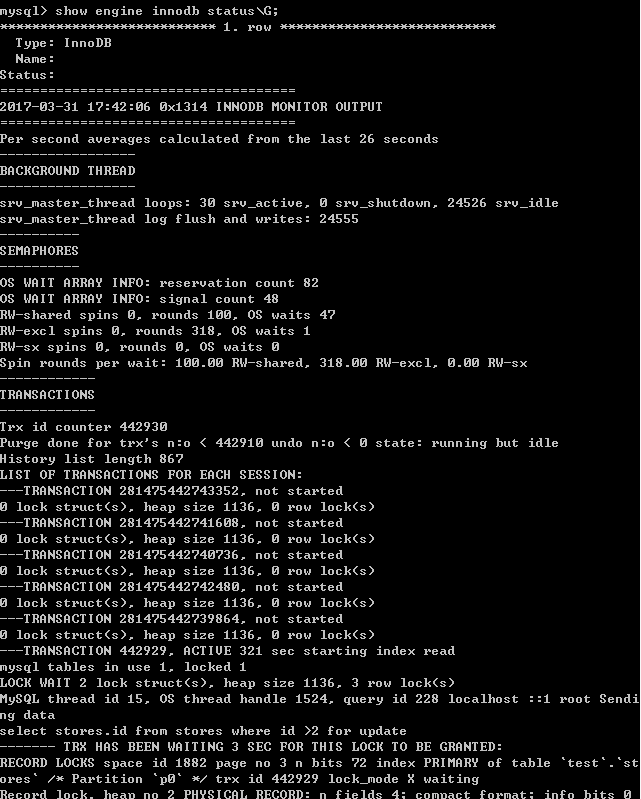


如果发现数据库中锁争用比较严重，如属性InnoDB\_row\_lock\_waits的值和属性InnoDB\_row\_lock\_time\_avg的值比较高，还可以通过设置InnoDB Monitors来进一步观察发生锁冲突的表、数据行等，并分析锁争用的原因。

常见的分析的方法有两种，一种是查看MySQL系统表INFORMATION\_SCHEMA数据库中的INNOSB\_BLOCKS，效果如下：



一种是使用命令，效果如下：



**3.2 InnoDB行锁模式和加锁方法**

InnoDB实现了以下两种类型的行锁，InnoDB行锁是通过给索引项加锁来实现的，即只有通过索引条件检索数据，InnoDB才使用行级锁，否则将使用表锁！：

* 共享锁（S）：允许一个事务去读一行，阻止其他事务获得相同数据集的排他锁。
* 排他锁（X)：允许获得排他锁的事务更新数据，阻止其他事务取得相同数据集的共享读锁和排他写锁。

为了允许行锁和表锁共存，实现多粒度锁机制，InnoDB还有两种内部使用的意向锁（Intention Locks），这两种意向锁都是表锁。

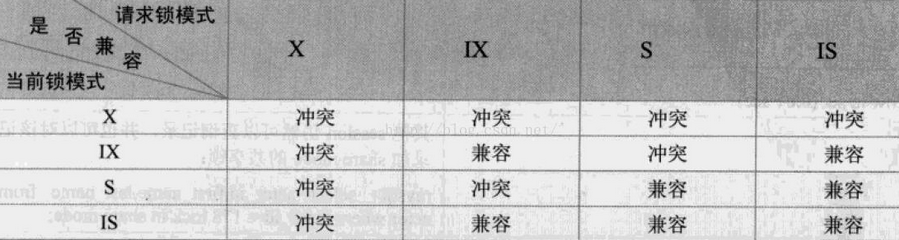
* 意向共享锁（IS）：事务打算给数据行加行共享锁，事务在给一个数据行加共享锁前必须先取得该表的IS锁。
* 意向排他锁（IX）：事务打算给数据行加行排他锁，事务在给一个数据行加排他锁前必须先取得该表的IX锁。

如果一个事务请求的锁模式与当前的锁兼容，InnoDB就将请求的锁授予该事务；反之，如果两者不兼容，该事务就要等待锁释放。

对于UPDATE、DELETE和INSERT语句，InnoDB会自动给涉及数据集加排他锁（X)；对于普通SELECT语句，InnoDB不会加任何锁；

事务可以通过以下语句显示给记录集加共享锁或排他锁

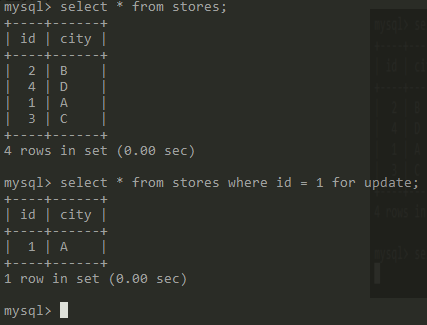
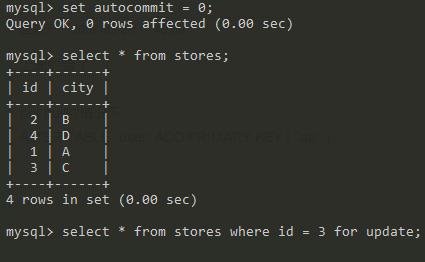
* 共享锁（S）：SELECT \* FROM table\_name WHERE ... LOCK IN SHARE MODE
* 排他锁（X)：SELECT \* FROM table\_name WHERE ... FOR UPDATE



**3.3 InnoDB行锁实现方式**

InnoDB行锁是通过给索引上的索引项加锁来实现的，如果没有索引，InnoDB将通过隐藏的聚簇索引来对记录加锁。这一点与Oracle不同，后者是通过在数据块中对相应数据行加锁来实现的。InnoDB这种行锁实现特点表明只有通过索引条件检索数据，InnoDB才使用行级锁，如果不通过索引条件检索数据，InnoDB将对表中所有的记录加锁，即使用表锁**！**

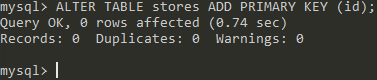
论证如下：

在表没有主键的情况下：

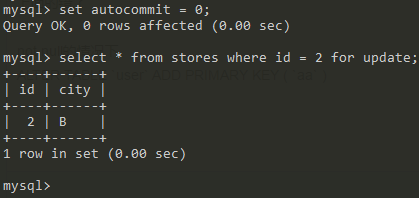
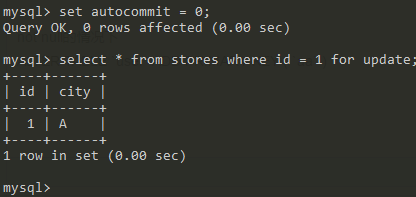
从上图可以看出，在没有锁的情况下，虽然第一个session中看起来只是对id=1的行加锁，但另外一个session请求对id=3的记录加锁时，却出现了锁等待，这就是因为在没有索引的情况下，InnoDB对表加了排它锁导致的。

下面将id这一列设为主键，然后在执行上面的查询，结果如下：

首先添加主键：

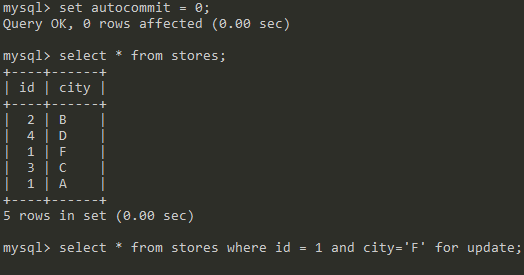
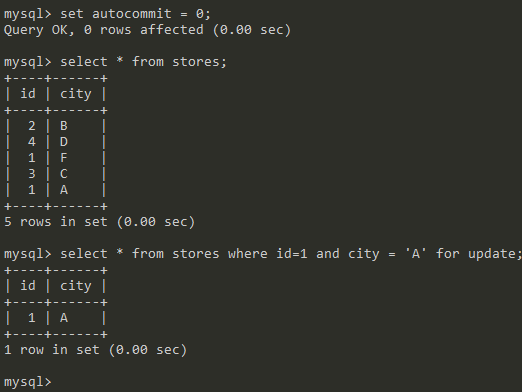


然后执行查询：



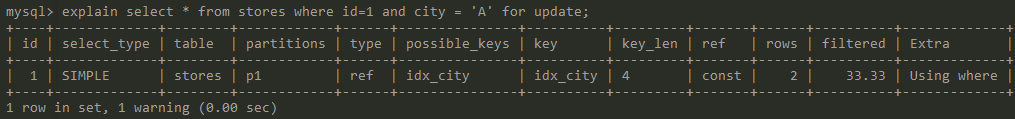
添加索引后，再次查询，发现可以正常执行，这就是因为查询使用索引加锁，分别对索引项id=1和id=2加锁，两者互不影响。

既然InnoDB通过索引加锁，那么我们在查询时需要注意的是，如果查询使用的是同一个索引项，那么也会出现锁等待：



上面例子中，虽然两个session查询的不是同一个记录，但是因为都是用id=1的索引项锁行，所以第二个session在查询时，请求排它锁会出现锁等待。

另外即便在条件中使用了索引字段，但是否使用索引来检索数据是由MySQL通过判断不同执行计划的代价来决定的，如果MySQL认为全表扫描效率更高，比如对一些很小的表，它就不会使用索引，这种情况下InnoDB将使用表锁，而不是行锁。因此，在分析锁冲突时，别忘了检查SQL的执行计划(explain)，以确认是否真正使用了索引。如：



**3.4 InnoDB行锁类型分析**

InnoDB行锁分为三种情形：

* Record lock：对索引项加锁
* Gap lock：对索引之间的间隙加锁
* Next-key lock：前两种的组合，对记录及前面的间隙加锁

当用范围条件而不是相等条件检索数据，并请求共享或排他锁时，InnoDB会给符合条件的已有数据记录的索引项加锁；对于键值在条件范围内但并不存在的记录，叫做“间隙（GAP)”，InnoDB也会对这个“间隙”加锁，这种锁机制就是所谓的间隙锁（Next-Key锁）。

假如stores表中只有5条记录，其id的值分别是 1,2,3,4,5，下面的SQL：

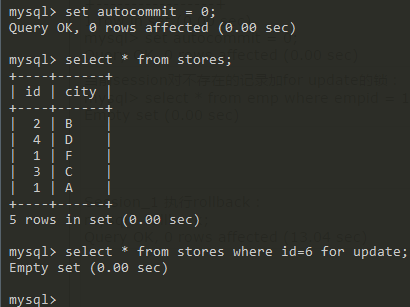
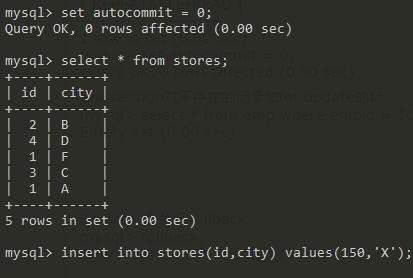
Select \* from stores where id > 4 for update;

这是一个范围条件的查询语句，InnoDB不仅会对符合条件的id值为5的记录加锁，也会对id大于5（这些记录并不存在）的“间隙”加锁。

InnoDB使用间隙锁的目的，一方面是为了防止幻读，以满足相关隔离级别的要求，对于上面的例子，要是不使用间隙锁，如果其他事务插入了id大于5的任何记录，那么本事务如果再次执行上述语句，就会发生幻读。

很显然，在使用范围条件检索并锁定记录时，InnoDB这种加锁机制会阻塞符合条件范围内键值的并发插入，这往往会造成严重的锁等待。因此，在实际应用开发中，尤其是并发插入比较多的应用，应尽量使用相等条件来访问更新数据，避免使用范围条件。

还要特别说明的是，InnoDB除了通过范围条件加锁时使用间隙锁外，如果使用相等条件请求给一个不存在的记录加锁，InnoDB也会使用间隙锁！



1. **MySQL中事务隔离级别和死锁**

**4.1 MySQL事务隔离级别**

相对于串行处理来说，并发事务处理大大的提高了数据库资源的利用率，让数据库可以支持更多的用户，但是并发事务处理同时也带来了另外一些问题，主要就是四类问题：更新丢失、脏读、不可重复读、幻读；这四种问题，除了更新丢失的防止可以交友应用自己解决，其它的问题都应该由数据库来提供一定的事务隔离机制来解决。

另一方面，数据库中的事务隔离和并发性是相矛盾的，事务隔离越严格，并发的副作用就越小，但是付出的代价就越大，并发性就越低。为了解决这种问题，MySQL中定义了四种隔离级别。其内容和作用如下：



**4.2 MySQL中死锁问题**

当两个事务都持有一个锁，并都需要获得对方持有的排它锁才能继续完成事务，这种两个事物循环锁等待就是典型的死锁。

上面总结过，MyISAM存储引擎总是一次获得所需的全部锁，要么全部满足，要么等待，所以MyISAM表锁是deadlock free的，因而MyISAM不会出现死锁。但在InnoDB中，除了单个SQL组成的事务外，其它的锁都是逐步获得的，故在InnoDB中发生死锁是可能的。

在MySQL中，发生死锁后，数据库可以检测到，并使一个事务强制退出，从而释放锁，以让另外一个事务获得锁。但是对于外部锁和涉及表锁的情况下，数据库并不能完全自动检测到死锁的存在，这样就可以通过设置innodb\_lock\_wait\_timeout设置合适的锁等待超时阈值，来解决这个问题。