Mysql 锁机制笔记

数据库在对资源进行高并发的读写操作时，为了保证数据的一致性，有效性，锁是很重要的机制。Mysql的锁分为三个级别：行级锁，页级锁，表级锁。对于平时常用的存储引擎，MyISAM采用的是表级锁，InnoDB采用的是行级锁加表级锁，而支持页级锁的BDB引擎已经逐渐被InnoDB替代了，这里暂不讨论。

表级锁的开销小，加锁快，不会出现死锁，锁定粒度大，大概率发生锁的冲突，并发度低
行级锁的开销大，加锁满，会出现死锁，锁定粒度小，小概率发生锁的重读，并发度高
上述特点来看，很难说哪种锁更好，只能相对于所处的业务场景来选择更加适合的锁机制。如果仅从锁的角度来看，表级锁更适合以查询为主的应用场景，而行级锁则更适合于大量按索引条件并发更新少量数据的应用场景。

MyISAM表

MyISAM存储引擎只支持表级锁，锁的模式有共享锁和排他锁。共享锁是他人可以读但不能写，排它锁则会阻塞他人的读写操作。MyISAM的读写之间，以及写写之间是串行的。
MyISAM在执行SQL语句时，会自动为SELECT语句加上共享锁，为UDI操作加上排它锁。MYSQL不支持锁升级，如果涉及到更新操作，需要在一开始就加上排它锁。

MyISAM的并发插入

在存储引擎中有一个系统变量concurrent_insert，专门控制其并发插入的行为
concurrent_insert=0时，不允许并发插入
concurrent_insert=1时，如果MyISAM表中没有空洞（即表的中间没有被删除的行），其允许在一个进程读表的同事，另一个进程从表尾插入记录，这也是MySQL的默认设置
concurrent_insert=2时，如果MyISAM表中没有空洞，允许在表尾并发插

入记录

MyISAM的锁调度

在MyISAM存储引擎中，写的重要性要大于读，所以在操作队列中，即使写的操作在读的操作之后，也会让写先拿到排它锁，这也正是MyISAM不适合于大量写入操作的应用场景的原因，这样可能会导致读操作永远在阻塞中，永远在等待写操作的释放锁。当然，除了默认的设置，可以通过设置语句的优先级别来管理这个执行顺序

InnoDB

他与MyISAM的最大区别有两个方面，一个是支持事务，另一个是采用了行级锁
事务的并发处理会带来几个问题
1.不可重复读，A事务在第一次读和第二次读之间，如果B对数据进行的修改，则两次读取的数据会不一致
2.更新丢失，A和B同时操作一个数据，最后执行完毕的会覆盖前一个的执行结果
3.脏读，A事务添加了数据但并未提交，B读到了这条数据后A回滚了，就会导致脏读（很形象）
4.幻读，A事务第二次读取数据之前，B数据提交了满足条件的数据，这种现象就叫幻读

为了解决以上问题，产生了四个隔离级别：未提交读，提交读，可重复读（InnoDB事务默认使用），串行读

锁模式

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。
另外，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。

语句示例：
共享锁（S）：SELECT * FROM table_name WHERE ... LOCK IN SHARE MODE。
排他锁（X)：SELECT * FROM table_name WHERE ... FOR UPDATE。

行锁的实现方式：

InnoDB行锁是通过给索引上的索引项加锁来实现的，这一点MySQL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这种行锁实现特点意味着：只有通过索引条件检索数据，InnoDB才使用行级锁，否则，InnoDB将使用表锁！在实际应用中，要特别注意InnoDB行锁的这一特性，不然的话，可能导致大量的锁冲突，从而影响并发性能。

由于MySQL的行锁是针对索引加的锁，不是针对记录加的锁，所以虽然是访问不同行的记录，但是如果是使用相同的索引键，是会出现锁冲突的

当表有多个索引的时候，不同的事务可以使用不同的索引锁定不同的行，另外，不论是使用主键索引、唯一索引或普通索引，InnoDB都会使用行锁来对数据加锁。如果不同的索引碰巧都落到了同一个行上，那么同样会阻塞。

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，以确认是否真正使用了索引。

间隙锁

当我们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）。

举例来说，假如emp表中只有101条记录，其empid的值分别是1,2,...,100,101，下面的SQL：
Select * from emp where empid > 100 for update;
是一个范围条件的检索，InnoDB不仅会对符合条件的empid值为101的记录加锁，也会对empid大于101（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empid大于100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；另外一方面，是为了满足其恢复和复制的需要

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！

MySQL的恢复机制是通过BINLOG记录来执行IUD操作来同步Slave的，这就要求：在一个事务未提交前，其他并发事务不能插入满足其锁定条件的任何记录，也就是不允许出现幻读，这已经超过了ISO/ANSI SQL92“可重复读”隔离级别的要求，实际上是要求事务要串行化。这也是许多情况下，InnoDB要用到间隙锁的原因，比如在用范围条件更新记录时，无论在Read Commited或是Repeatable Read隔离级别下，InnoDB都要使用间隙锁，但这并不是隔离级别要求的。

INSERT...SELECT...和 CREATE TABLE...SELECT...语句，可能会阻止对源表的并发更新，造成对源表锁的等待。如果查询比较复杂的话，会造成严重的性能问题，我们在应用中应尽量避免使用。实际上，MySQL将这种SQL叫作不确定（non-deterministic）的SQL，不推荐使用。

什么时候使用表锁

对于InnoDB表，在绝大部分情况下都应该使用行级锁，因为事务和行锁往往是我们之所以选择InnoDB表的理由。但在个别特殊事务中，也可以考虑使用表级锁。
第一种情况是：事务需要更新大部分或全部数据，表又比较大，如果使用默认的行锁，不仅这个事务执行效率低，而且可能造成其他事务长时间锁等待和锁冲突，这种情况下可以考虑使用表锁来提高该事务的执行速度。

第二种情况是：事务涉及多个表，比较复杂，很可能引起死锁，造成大量事务回滚。这种情况也可以考虑一次性锁定事务涉及的表，从而避免死锁、减少数据库因事务回滚带来的开销。
如果以上两种事务过多，那我们就可以考虑使用MyISAM引擎了

死锁

发生死锁后，InnoDB一般都能自动检测到，并使一个事务释放锁并回退，另一个事务获得锁，继续完成事务。但在涉及外部锁，或涉及表锁的情况下，InnoDB并不能完全自动检测到死锁，这需要通过设置锁等待超时参数innodb_lock_wait_timeout来解决。需要说明的是，这个参数并不是只用来解决死锁问题，在并发访问比较高的情况下，如果大量事务因无法立即获得所需的锁而挂起，会占用大量计算机资源，造成严重性能问题，甚至拖跨数据库。我们通过设置合适的锁等待超时阈值，可以避免这种情况发生。

在了解InnoDB锁特性后，用户可以通过设计和SQL调整等措施减少锁冲突和死锁，包括：

1. 尽量使用较低的隔离级别；
2. 精心设计索引，并尽量使用索引访问数据，使加锁更精确，从而减少锁冲突的机会；
3. 选择合理的事务大小，小事务发生锁冲突的几率也更小；
4. 给记录集显示加锁时，最好一次性请求足够级别的锁。比如要修改数据的话，最好直接申请排他锁，而不是先申请共享锁，修改时再请求排他锁，这样容易产生死锁；
5. 不同的程序访问一组表时，应尽量约定以相同的顺序访问各表，对一个表而言，尽可能以固定的顺序存取表中的行。这样可以大大减少死锁的机会；
6. 尽量用相等条件访问数据，这样可以避免间隙锁对并发插入的影响；
7. 不要申请超过实际需要的锁级别；除非必须，查询时不要显示加锁；
8. 对于一些特定的事务，可以使用表锁来提高处理速度或减少死锁的可能。