**MySQL 表锁和行锁机制**

行锁变表锁，是福还是坑？如果你不清楚MySQL加锁的原理，你会被它整的很惨！不知坑在何方？没事，我来给你们标记几个坑。遇到了可别乱踩。通过本章内容，带你学习MySQL的行锁，表锁，两种锁的优缺点，行锁变表锁的原因，以及开发中需要注意的事项。还在等啥？经验等你来拿！

MySQL的存储引擎是从MyISAM到InnoDB，锁从表锁到行锁。后者的出现从某种程度上是弥补前者的不足。比如：MyISAM不支持事务，InnoDB支持事务。表锁虽然开销小，锁表快，但高并发下性能低。行锁虽然开销大，锁表慢，但高并发下相比之下性能更高。事务和行锁都是在确保数据准确的基础上提高并发的处理能力。本章重点介绍InnoDB的行锁。

**案例分析**

目前，MySQL常用的存储引擎是InnoDB，相对于MyISAM而言。InnoDB更适合高并发场景，同时也支持事务处理。我们通过下面这个案例(坑)，来了解行锁和表锁。

业务：因为订单重复导入，需要用脚本将订单状态为"待客服确认"且平台是"xxx"的数据批量修改为"已关闭"。  
说明：避免直接修改订单表造成数据异常。这里用innodb\_lock 表演示InnoDB的行锁。表中有三个字段：id，k(key值)，v(value值)。表在github上：<https://github.com/ITDragonBlog/daydayup/tree/master/MySQL/>  
步骤：  
第一步：连接数据库，这里为了方便区分命名为Transaction-A，设置autocommit为零，表示需手动提交事务。  
第二步：Transaction-A，执行update修改id为1的命令。  
第三步：新增一个连接，命名为Transaction-B，能正常修改id为2的数据。再执行修改id为1的数据命令时，却发现该命令一直处理阻塞等待中。  
第四步：Transaction-A，执行commit命令。Transaction-B，修改id为1的命令自动执行，等待37.51秒。

总结：**多个事务操作同一行数据时，后来的事务处于阻塞等待状态。这样可以避免了脏读等数据一致性的问题。后来的事务可以操作其他行数据，解决了表锁高并发性能低的问题**。

# Transaction-A

mysql> set autocommit = 0;

mysql> update innodb\_lock set v='1001' where id=1;

mysql> commit;

# Transaction-B

mysql> update innodb\_lock set v='2001' where id=2;

Query OK, 1 row affected (0.37 sec)

mysql> update innodb\_lock set v='1002' where id=1;

Query OK, 1 row affected (37.51 sec)

有了上面的模拟操作，结果和理论又惊奇的一致，似乎可以放心大胆的实战。。。。。。但现实真的很残酷。

现实：当执行批量修改数据脚本的时候，行锁升级为表锁。其他对订单的操作都处于等待中，，，  
原因：InnoDB只有在通过索引条件检索数据时使用行级锁，否则使用表锁！而模拟操作正是通过id去作为检索条件，而id又是MySQL自动创建的唯一索引，所以才忽略了行锁变表锁的情况。  
步骤：  
第一步：还原问题，Transaction-A，通过k=1更新v。Transaction-B，通过k=2更新v，命令处于阻塞等待状态。  
第二步：处理问题，给需要作为查询条件的字段添加索引。用完后可以删掉。

总结：**InnoDB的行锁是针对索引加的锁，不是针对记录加的锁。并且该索引不能失效，否则都会从行锁升级为表锁**。索引失效的原因在上一章节中已经介绍：<http://www.cnblogs.com/itdragon/p/8146439.html>

Transaction-A

mysql> update innodb\_lock set v='1002' where k=1;

mysql> commit;

mysql> create index idx\_k on innodb\_lock(k);

Transaction-B

mysql> update innodb\_lock set v='2002' where k=2;

Query OK, 1 row affected (19.82 sec)

从上面的案例看出，行锁变表锁似乎是一个坑，可MySQL没有这么无聊给你挖坑。这是因为MySQL有自己的执行计划。

当你需要更新一张较大表的大部分甚至全表的数据时。而你又傻乎乎地用索引作为检索条件。一不小心开启了行锁(没毛病啊！保证数据的一致性！)。可MySQL却认为大量对一张表使用行锁，会导致事务执行效率低，从而可能造成其他事务长时间锁等待和更多的锁冲突问题，性能严重下降。所以MySQL会将行锁升级为表锁，即实际上并没有使用索引。  
我们仔细想想也能理解，既然整张表的大部分数据都要更新数据，一行一行地加锁效率则更低。其实我们可以通过explain命令查看MySQL的执行计划，你会发现key为null。表明MySQL实际上并没有使用索引，行锁升级为表锁也和上面的结论一致。

本章重点介绍InnoDB的行锁及其相关的事务知识。如果想了解MySQL的执行计划，请看[上一章节](http://www.cnblogs.com/itdragon/p/8146439.html)。

**行锁**

行锁的劣势：开销大；加锁慢；会出现死锁  
行锁的优势：锁的粒度小，发生锁冲突的概率低；处理并发的能力强  
加锁的方式：自动加锁。对于UPDATE、DELETE和INSERT语句，InnoDB会自动给涉及数据集加排他锁；对于普通SELECT语句，InnoDB不会加任何锁；当然我们也可以显示的加锁：  
共享锁：select \* from tableName where ... + lock in share more  
排他锁：select \* from tableName where ... + for update  
InnoDB和MyISAM的最大不同点有两个：一，InnoDB支持事务(transaction)；二，默认采用行级锁。加锁可以保证事务的一致性，可谓是有人(锁)的地方，就有江湖(事务)；我们先简单了解一下事务知识。

**MySQL 事务属性**

事务是由一组SQL语句组成的逻辑处理单元，事务具有ACID属性。  
**原子性**（Atomicity）：事务是一个原子操作单元。在当时原子是不可分割的最小元素，其对数据的修改，要么全部成功，要么全部都不成功。  
**一致性**（Consistent）：事务开始到结束的时间段内，数据都必须保持一致状态。  
**隔离性**（Isolation）：数据库系统提供一定的隔离机制，保证事务在不受外部并发操作影响的"独立"环境执行。  
**持久性**（Durable）：事务完成后，它对于数据的修改是永久性的，即使出现系统故障也能够保持。

**事务常见问题**

**更新丢失**（Lost Update）  
原因：当多个事务选择同一行操作，并且都是基于最初选定的值，由于每个事务都不知道其他事务的存在，就会发生更新覆盖的问题。类比github提交冲突。

**脏读**（Dirty Reads）  
原因：事务A读取了事务B已经修改但尚未提交的数据。若事务B回滚数据，事务A的数据存在不一致性的问题。

**不可重复读**（Non-Repeatable Reads）  
原因：事务A第一次读取最初数据，第二次读取事务B已经提交的修改或删除数据。导致两次读取数据不一致。不符合事务的隔离性。

**幻读**（Phantom Reads）  
原因：事务A根据相同条件第二次查询到事务B提交的新增数据，两次数据结果集不一致。不符合事务的隔离性。

幻读和脏读有点类似  
脏读是事务B里面修改了数据，  
幻读是事务B里面新增了数据。

**事务的隔离级别**

数据库的事务隔离越严格，并发副作用越小，但付出的代价也就越大。这是因为事务隔离实质上是将事务在一定程度上"串行"进行，这显然与"并发"是矛盾的。根据自己的业务逻辑，权衡能接受的最大副作用。从而平衡了"隔离" 和 "并发"的问题。MySQL默认隔离级别是可重复读。  
脏读，不可重复读，幻读，其实都是数据库读一致性问题，必须由数据库提供一定的事务隔离机制来解决。

+------------------------------+---------------------+--------------+--------------+--------------+

| 隔离级别 | 读数据一致性 | 脏读 | 不可重复 读 | 幻读 |

+------------------------------+---------------------+--------------+--------------+--------------+

| 未提交读(Read uncommitted) | 最低级别 | 是 | 是 | 是 |

+------------------------------+---------------------+--------------+--------------+--------------+

| 已提交读(Read committed) | 语句级 | 否 | 是 | 是 |

+------------------------------+---------------------+--------------+--------------+--------------+

| 可重复读(Repeatable read) | 事务级 | 否 | 否 | 是 |

+------------------------------+---------------------+--------------+--------------+--------------+

| 可序列化(Serializable) | 最高级别，事务级 | 否 | 否 | 否 |

+------------------------------+---------------------+--------------+--------------+--------------+

查看当前数据库的事务隔离级别：show variables like 'tx\_isolation';

mysql> show variables like 'tx\_isolation';

+---------------+-----------------+

| Variable\_name | Value |

+---------------+-----------------+

| tx\_isolation | REPEATABLE-READ |

+---------------+-----------------+

**间隙锁**

当我们用范围条件检索数据，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)。

Transaction-A

mysql> update innodb\_lock set k=66 where id >=6;

Query OK, 1 row affected (0.63 sec)

mysql> commit;

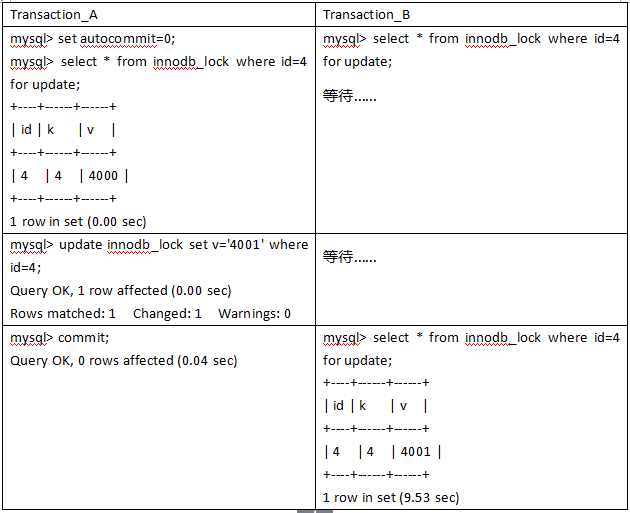
Transaction-B

mysql> insert into innodb\_lock (id,k,v) values(7,'7','7000');

Query OK, 1 row affected (18.99 sec)

危害(坑)：**若执行的条件是范围过大，则InnoDB会将整个范围内所有的索引键值全部锁定，很容易对性能造成影响**。

**排他锁**

排他锁，也称写锁，独占锁，当前写操作没有完成前，它会阻断其他写锁和读锁。  


# Transaction\_A

mysql> set autocommit=0;

mysql> select \* from innodb\_lock where id=4 for update;

+----+------+------+

| id | k | v |

+----+------+------+

| 4 | 4 | 4000 |

+----+------+------+

1 row in set (0.00 sec)

mysql> update innodb\_lock set v='4001' where id=4;

Query OK, 1 row affected (0.00 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> commit;

Query OK, 0 rows affected (0.04 sec)

# Transaction\_B

mysql> select \* from innodb\_lock where id=4 for update;

+----+------+------+

| id | k | v |

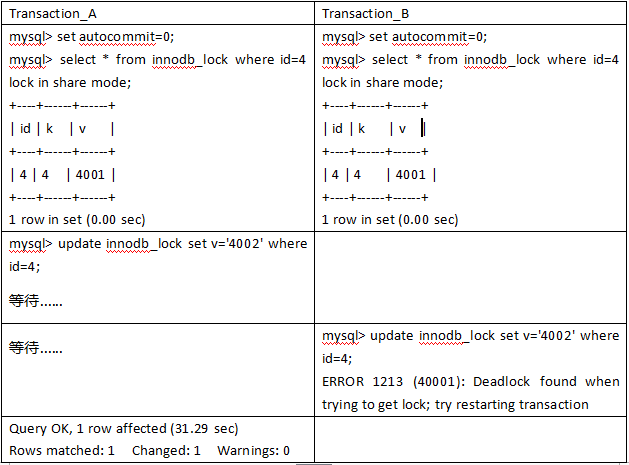
+----+------+------+

| 4 | 4 | 4001 |

+----+------+------+

1 row in set (9.53 sec)

**共享锁**

共享锁，也称读锁，多用于判断数据是否存在，多个读操作可以同时进行而不会互相影响。当如果事务对读锁进行修改操作，很可能会造成死锁。如下图所示。  


# Transaction\_A

mysql> set autocommit=0;

mysql> select \* from innodb\_lock where id=4 lock in share mode;

+----+------+------+

| id | k | v |

+----+------+------+

| 4 | 4 | 4001 |

+----+------+------+

1 row in set (0.00 sec)

mysql> update innodb\_lock set v='4002' where id=4;

Query OK, 1 row affected (31.29 sec)

Rows matched: 1 Changed: 1 Warnings: 0

# Transaction\_B

mysql> set autocommit=0;

mysql> select \* from innodb\_lock where id=4 lock in share mode;

+----+------+------+

| id | k | v |

+----+------+------+

| 4 | 4 | 4001 |

+----+------+------+

1 row in set (0.00 sec)

mysql> update innodb\_lock set v='4002' where id=4;

ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction

**分析行锁定**

通过检查InnoDB\_row\_lock 状态变量分析系统上的行锁的争夺情况 show status like 'innodb\_row\_lock%'

mysql> show status like 'innodb\_row\_lock%';

+-------------------------------+-------+

| Variable\_name | Value |

+-------------------------------+-------+

| Innodb\_row\_lock\_current\_waits | 0 |

| Innodb\_row\_lock\_time | 0 |

| Innodb\_row\_lock\_time\_avg | 0 |

| Innodb\_row\_lock\_time\_max | 0 |

| Innodb\_row\_lock\_waits | 0 |

+-------------------------------+-------+

innodb\_row\_lock\_current\_waits: 当前正在等待锁定的数量  
innodb\_row\_lock\_time: 从系统启动到现在锁定总时间长度；非常重要的参数，  
innodb\_row\_lock\_time\_avg: 每次等待所花平均时间；非常重要的参数，  
innodb\_row\_lock\_time\_max: 从系统启动到现在等待最常的一次所花的时间；  
innodb\_row\_lock\_waits: 系统启动后到现在总共等待的次数；非常重要的参数。直接决定优化的方向和策略。

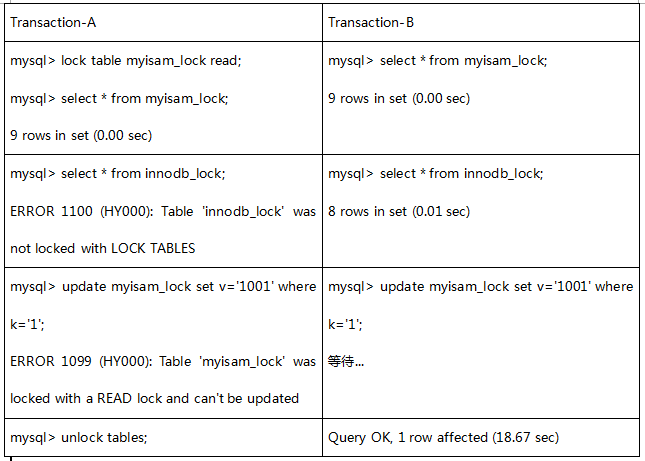
**行锁优化**

1 尽可能让所有数据检索都通过索引来完成，避免无索引行或索引失效导致行锁升级为表锁。  
2 尽可能避免间隙锁带来的性能下降，减少或使用合理的检索范围。  
3 尽可能减少事务的粒度，比如控制事务大小，而从减少锁定资源量和时间长度，从而减少锁的竞争等，提供性能。  
4 尽可能低级别事务隔离，隔离级别越高，并发的处理能力越低。

**表锁**

表锁的优势：开销小；加锁快；无死锁  
表锁的劣势：锁粒度大，发生锁冲突的概率高，并发处理能力低  
加锁的方式：自动加锁。查询操作（SELECT），会自动给涉及的所有表加读锁，更新操作（UPDATE、DELETE、INSERT），会自动给涉及的表加写锁。也可以显示加锁：  
共享读锁：lock table tableName read;  
独占写锁：lock table tableName write;  
批量解锁：unlock tables;

**共享读锁**

对MyISAM表的读操作（加读锁），不会阻塞其他进程对同一表的读操作，但会阻塞对同一表的写操作。只有当读锁释放后，才能执行其他进程的写操作。在锁释放前不能取其他表。  


Transaction-A

mysql> lock table myisam\_lock read;

Query OK, 0 rows affected (0.00 sec)

mysql> select \* from myisam\_lock;

9 rows in set (0.00 sec)

mysql> select \* from innodb\_lock;

ERROR 1100 (HY000): Table 'innodb\_lock' was not locked with LOCK TABLES

mysql> update myisam\_lock set v='1001' where k='1';

ERROR 1099 (HY000): Table 'myisam\_lock' was locked with a READ lock and can't be updated

mysql> unlock tables;

Query OK, 0 rows affected (0.00 sec)

Transaction-B

mysql> select \* from myisam\_lock;

9 rows in set (0.00 sec)

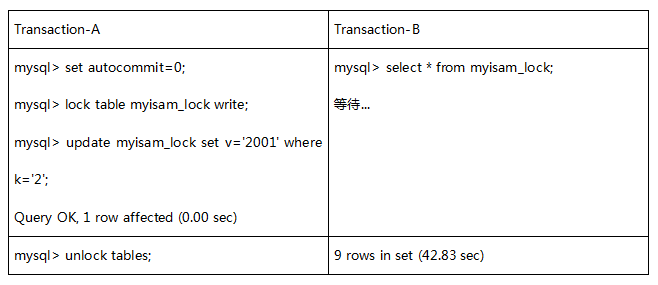
mysql> select \* from innodb\_lock;

8 rows in set (0.01 sec)

mysql> update myisam\_lock set v='1001' where k='1';

Query OK, 1 row affected (18.67 sec)

**独占写锁**

对MyISAM表的写操作（加写锁），会阻塞其他进程对同一表的读和写操作，只有当写锁释放后，才会执行其他进程的读写操作。在锁释放前不能写其他表。  


Transaction-A

mysql> set autocommit=0;

Query OK, 0 rows affected (0.05 sec)

mysql> lock table myisam\_lock write;

Query OK, 0 rows affected (0.03 sec)

mysql> update myisam\_lock set v='2001' where k='2';

Query OK, 1 row affected (0.00 sec)

mysql> select \* from myisam\_lock;

9 rows in set (0.00 sec)

mysql> update innodb\_lock set v='1001' where k='1';

ERROR 1100 (HY000): Table 'innodb\_lock' was not locked with LOCK TABLES

mysql> unlock tables;

Query OK, 0 rows affected (0.00 sec)

Transaction-B

mysql> select \* from myisam\_lock;

9 rows in set (42.83 sec)

总结：**表锁，读锁会阻塞写，不会阻塞读。而写锁则会把读写都阻塞**。

**查看加锁情况**

show open tables; 1表示加锁，0表示未加锁。

mysql> show open tables where in\_use > 0;

+----------+-------------+--------+-------------+

| Database | Table | In\_use | Name\_locked |

+----------+-------------+--------+-------------+

| lock | myisam\_lock | 1 | 0 |

+----------+-------------+--------+-------------+

**分析表锁定**

可以通过检查table\_locks\_waited 和 table\_locks\_immediate 状态变量分析系统上的表锁定：show status like 'table\_locks%'

mysql> show status like 'table\_locks%';

+----------------------------+-------+

| Variable\_name | Value |

+----------------------------+-------+

| Table\_locks\_immediate | 104 |

| Table\_locks\_waited | 0 |

+----------------------------+-------+

table\_locks\_immediate: 表示立即释放表锁数。  
table\_locks\_waited: 表示需要等待的表锁数。此值越高则说明存在着越严重的表级锁争用情况。

此外，MyISAM的读写锁调度是写优先，这也是MyISAM不适合做写为主表的存储引擎。因为写锁后，其他线程不能做任何操作，大量的更新会使查询很难得到锁，从而造成永久阻塞。

**什么场景下用表锁**

InnoDB默认采用行锁，在未使用索引字段查询时升级为表锁。MySQL这样设计并不是给你挖坑。它有自己的设计目的。  
即便你在条件中使用了索引字段，MySQL会根据自身的执行计划，考虑是否使用索引(所以explain命令中会有possible\_key 和 key)。如果MySQL认为全表扫描效率更高，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。

第一种情况：**全表更新**。事务需要更新大部分或全部数据，且表又比较大。若使用行锁，会导致事务执行效率低，从而可能造成其他事务长时间锁等待和更多的锁冲突。

第二种情况：**多表级联**。事务涉及多个表，比较复杂的关联查询，很可能引起死锁，造成大量事务回滚。这种情况若能一次性锁定事务涉及的表，从而可以避免死锁、减少数据库因事务回滚带来的开销。

**页锁**

开销和加锁时间介于表锁和行锁之间；会出现死锁；锁定粒度介于表锁和行锁之间，并发处理能力一般。只需了解一下。

**总结**

1 InnoDB 支持表锁和行锁，使用索引作为检索条件修改数据时采用行锁，否则采用表锁。  
2 InnoDB 自动给修改操作加锁，给查询操作不自动加锁  
3 行锁可能因为未使用索引而升级为表锁，所以除了检查索引是否创建的同时，也需要通过explain执行计划查询索引是否被实际使用。  
4 行锁相对于表锁来说，优势在于高并发场景下表现更突出，毕竟锁的粒度小。  
5 当表的大部分数据需要被修改，或者是多表复杂关联查询时，建议使用表锁优于行锁。  
6 为了保证数据的一致完整性，任何一个数据库都存在锁定机制。锁定机制的优劣直接影响到一个数据库的并发处理能力和性能。