**详解mysql的for update** **背景**  
for update是在数据库中上锁用的，可以为数据库中的行上一个排它锁。当一个事务的操作未完成时候，其他事务可以读取但是不能写入或更新。

**for update的使用场景**

如果遇到存在高并发并且对于数据的准确性很有要求的场景，是需要了解和使用for update的。

比如涉及到金钱、库存等。一般这些操作都是很长一串并且是开启事务的。如果库存刚开始读的时候是1，而立马另一个进程进行了update将库存更新为0了，而事务还没有结束，会将错的数据一直执行下去，就会有问题。所以需要for upate 进行数据加锁防止高并发时候数据出错。

**记住一个原则：一锁二判三更新**

**for update如何使用**

使用姿势：

select \* from table where xxx for update

**for update的锁表**

InnoDB默认是行级别的锁，当有明确指定的主键时候，是行级锁。否则是表级别。

例子: 假设表foods ，存在有id跟name、status三个字段，id是主键，status有索引。

例1: (明确指定主键，并且有此记录，行级锁)

1. SELECT \* FROM foods WHERE id=1 FOR UPDATE;
2. SELECT \* FROM foods WHERE id=1 and name=’咖啡色的羊驼’ FOR UPDATE;

例2: (明确指定主键/索引，若查无此记录，无锁)

SELECT \* FROM foods WHERE id=-1 FOR UPDATE;

例3: (无主键/索引，表级锁)

SELECT \* FROM foods WHERE name=’咖啡色的羊驼’ FOR UPDATE;

例4: (主键/索引不明确，表级锁)

1. SELECT \* FROM foods WHERE id<>’3’ FOR UPDATE;
2. SELECT \* FROM foods WHERE id LIKE ‘3’ FOR UPDATE;

**for update的注意点**

1.for update 仅适用于InnoDB，并且必须开启事务，在begin与commit之间才生效。

2.要测试for update的锁表情况，可以利用MySQL的Command Mode，开启二个视窗来做测试。

**for update的疑问点**

当开启一个事务进行for update的时候，另一个事务也有for update的时候会一直等着，直到第一个事务结束吗？  
答：会的。除非第一个事务commit或者rollback或者断开连接，第二个事务会立马拿到锁进行后面操作。  
  
如果没查到记录会锁表吗？

答：会的。表级锁时，不管是否查询到记录，都会锁定表。

**for update 和 for update nowait区别（前者阻塞其他事务，后者拒绝其他事务）**

for update锁住表或者锁住行，只允许当前事务进行操作（读写），其他事务被阻塞，直到当前事务提交或者回滚，被阻塞的事务自动执行  
for update nowait 锁住表或者锁住行，只允许当前事务进行操作（读写），其他事务被拒绝，事务占据的statement连接也会被断开

## 悲观锁介绍

　　悲观锁是对数据被的修改持悲观态度（认为数据在被修改的时候一定会存在并发问题），因此在整个数据处理过程中将数据锁定。悲观锁的实现，往往依靠数据库提供的锁机制（也只有数据库层提供的锁机制才能真正保证数据访问的排他性，否则，即使在应用层中实现了加锁机制，也无法保证外部系统不会修改数据）。

使用场景举例

　　商品goods表中有一个字段status，status为1代表商品未被下单，status为2代表商品已经被下单，那么我们对某个商品下单时必须确保该商品status为1。假设商品的id为1。如果不采用锁，那么操作方法如下：

1. *//1.查询出商品信息*
2. select status from t\_goods where id=1;
3. *//2.根据商品信息生成订单*
4. insert into t\_orders (id,goods\_id) values (null,1);
5. *//3.修改商品status为2*
6. update t\_goods set status=2;

上面这种场景在高并发访问的情况下很可能会出现问题。前面已经提到，只有当goods status为1时才能对该商品下单，上面第一步操作中，查询出来的商品status为1。但是当我们执行第三步Update操作的时候，有可能出现其他人先一步对商品下单把goods status修改为2了，但是我们并不知道数据已经被修改了，这样就可能造成同一个商品被下单2次，使得数据不一致。所以说这种方式是不安全的。

## 使用悲观锁来实现

　　在上面的场景中，商品信息从查询出来到修改，中间有一个处理订单的过程，使用悲观锁的原理就是，当我们在查询出goods信息后就把当前的数据锁定，直到我们修改完毕后再解锁。那么在这个过程中，因为goods被锁定了，就不会出现有第三者来对其进行修改了。要使用悲观锁，我们必须关闭mysql数据库的自动提交属性。

1. set autocommit=0;
2. *//设置完autocommit后，我们就可以执行我们的正常业务了。具体如下：*
3. *//0.开始事务*
4. begin;/begin work;/start transaction; (三者选一就可以)
5. *//1.查询出商品信息*
6. select status from t\_goods where id=1 for update;
7. *//2.根据商品信息生成订单*
8. insert into t\_orders (id,goods\_id) values (null,1);
9. *//3.修改商品status为2*
10. update t\_goods set status=2;
11. *//4.提交事务*
12. commit;/commit work;

　　注：上面的begin/commit为事务的开始和结束，因为在前一步我们关闭了mysql的autocommit，所以需要手动控制事务的提交，在这里就不细表了。

　　上面的第一步我们执行了一次查询操作：select status from t\_goods where id=1 for update;与普通查询不一样的是，我们使用了select…for update的方式，这样就通过数据库实现了悲观锁。此时在t\_goods表中，id为1的那条数据就被我们锁定了，其它的事务必须等本次事务提交之后才能执行。这样我们可以保证当前的数据不会被其它事务修改。

　　注：需要注意的是，在事务中，只有SELECT ... FOR UPDATE 或LOCK IN SHARE MODE 相同数据时会等待其它事务结束后才执行，一般SELECT ... 则不受此影响。拿上面的实例来说，当我执行select status from t\_goods where id=1 for update;后。我在另外的事务中如果再次执行select status from t\_goods where id=1 for update;则第二个事务会一直等待第一个事务的提交，此时第二个查询处于阻塞的状态，但是如果我是在第二个事务中执行select status from t\_goods where id=1;则能正常查询出数据，不会受第一个事务的影响。