更新丢失是指并发下两次更新同时进行，后一次更新覆盖了前一次更新的情况，更新丢失是数据没有保证一致性导致的。  
举个栗子：

* 用户A在银行卡有100元钱，某一刻用户B向A转账50元（称为B操作），同时有用户C向A转账50元（称为C操作);
* B操作从数据库中读取他此时的余额100，计算新的余额为100+50=150
* C操作也从数据库中读取他此时的余额100，计算新的余额为100+50=150
* B操作将balance=150写入数据库,之后C操作也将balance=150写入数据库
* 最终A的余额变为150

上面的例子，A同时收到两笔50元转账，最后的余额应该是200元，但却因为并发的问题变为了150元，原因是B和C向A发起转账请求时，同时打开了两个数据库会话，进行了两个事务，后一个事务拿到了前一个事务的中间状态数据，导致更新丢失。  
常用的解决思路有两种：

* 加锁同步执行
* update前检查数据一致性

**悲观锁**

顾名思义，悲观锁在读取数据的时候都会认为会有别人去修改，于是在取数据的时候会对当前数据加一个锁,在操作结束前，不允许其余操作更改。要注意悲观锁和乐观锁都是业务逻辑层次的定义，不同的设计可能会有不同的实现。在mysql层常用的悲观锁实现方式是加一个排他锁。  
**排他锁**  
查阅资料很多对排他锁的解释是：“排他锁通过在事务中使用select xx for update语句来实现，排他锁会在当前行加一个行级锁，在当前事务提交前，其余事务无法进行update操作。”

然而实际上并不是这样，实际上是加了排他锁的数据，在释放锁（事务结束）之前其他事务不能再对该数据加锁  
排他锁之所以能阻止update,delete等操作是因为update，delete操作会自动加排他锁

也就是说即使加了排他锁也无法阻止select操作。而select XX for update 语法可以对select 操作加上排他锁。所以为了防止更新丢失可以在select时加上for update加锁 这样就可以阻止其余事务的select for update(但注意无法阻止select)  
example:

**begin**;

**select** \* **from** **account** **where** **id** = 1 **for** **update**;

**update** **account** **set** balance=150 **where** **id** =1;

**commit**;

这样在B操作提交前,C操作无法获得排他锁，从而避免对account的重复更新导致的更新丢失。

**乐观锁**

乐观锁是指在获取数据时候不加锁，乐观的认为操作不会有冲突，在update的时候再去检查冲突。  
example：

**begin**;

**select** balance **from** **account** **where** **id**=1;

*-- 得到balance=100;然后计算balance=100+50=150*

**update** **account** **set** balance = 150 **where** **id**=1 **and** balance = 100;

**commit**;

如上，如果sql在执行的过程中发现update的affected为0 说明balance不等于100即该条数据有被其余事务更改过，此时业务上就可以返回失败或者重新select再计算