**[并发事务的丢失更新及及其处理方式](http://www.cnblogs.com/doucheyard/p/5662171.html)**

    在事务的隔离级别内容中，能够了解到两个不同的事务在并发的时候可能会发生数据的影响。细心的话可以发现事务隔离级别章节中，脏读、不可重复读、幻读三个问题都是由事务A对数据进行修改、增加，事务B**总**是在做读操作。如果两事务都在对数据进行修改则会导致另外的问题：**丢失更新**。

**1、丢失更新的定义及产生原因。**

**第一类丢失更新**(回滚丢失，Lost update)  **（通过设置隔离级别可以防止 Repeatable Read）**

    A事务撤销时，把已经提交的B事务的更新数据覆盖了。这种错误可能造成很严重的问题，通过下面的账户取款转账就可以看出来：

|  |  |  |
| --- | --- | --- |
| 时间 | 取款事务A | 转账事务B |
| T1 | **开始事务** |  |
| T2 |  | **开始事务** |
| T3 | 查询账户余额为1000元 |  |
| T4 |  | 查询账户余额为1000元 |
| T5 |  | 汇入100元把余额改为1100元 |
| T6 |  | **提交事务** |
| T7 | 取出100元把余额改为900元 |  |
| T8 | **撤销事务** |  |
| T9 | **余额恢复为1000** **元（丢失更新）** |  |

A事务在撤销时，“不小心”将B事务已经转入账户的金额给抹去了。

  SQL92没有定义这种现象，**标准定义的所有隔离界别都不允许第一类丢失更新发生。**

**第二类丢失更新** (覆盖丢失/两次更新问题，Second lost update) A事务覆盖B事务已经提交的数据，造成B事务所做操作丢失：

|  |  |  |
| --- | --- | --- |
| 时间 | 转账事务A | 取款事务B |
| T1 |  | **开始事务** |
| T2 | **开始事务** |  |
| T3 |  | 查询账户余额为1000元 |
| T4 | 查询账户余额为1000元 |  |
| T5 |  | 取出100元把余额改为900元 |
| T6 |  | **提交事务** |
| T7 | 汇入100元 |  |
| T8 | **提交事务** |  |
| T9 | **把余额改为1100** **元（丢失更新）** |  |

   上面的例子里由于支票转账事务覆盖了取款事务对存款余额所做的更新，导致银行最后损失了100元，相反如果转账事务先提交，那么用户账户将损失100元。

     第二类丢失更新，实际上和不可重复读是同一种问题。

有些系统中第二类丢失更新可能就影响很大了，举个简单的例子：  
财务系统加工资，若公司本次调薪决定给员工张三加1k人民币，财务部两名操作人员A和B，过程情况若是这样的：  
1）A操作员在应用系统的页面上查询出张三的薪水信息，然后选择薪水记录进行修改，打开修改页面但A突然有事离开了，页面放在那没有做任何的提交。  
2）这时候B操作员同样在应用中查询出张三的薪水信息，然后选择薪水记录进行修改，录入增加薪水额1000，然后提交了。  
3）这时候A操作员回来了，在自己之前打开的薪水修改页面上也录入了增加薪水额1000，然后提交了。  
其实上面例子操作员A和B只要一前一后做提交，悲剧就出来了。后台修改薪水的sql：update 工资表 set salary = salary + 增加薪水额 where staff\_id = ‘员工ID’。这个过程走下来后结果是：张三开心了这次涨了2k，操作员A和B都郁闷了。

解决思路：  
       基本两种思路，一种是悲观锁，另外一种是乐观锁；  
简单的说就是一种假定这样的问题是高概率的，最好一开始就锁住，免得更新老是失败；另外一种假定这样的问题是小概率的，最后一步做更新的时候再锁住，免得锁住时间太长影响其他人做有关操作。  
  
解决方案1（悲观锁）  
 a.传统的悲观锁法（不推荐）：  
         以上面的例子来说明，在弹出修改工资的**页面初始化**时（这种情况下一般会去从数据库查询出来），在这个初始化查询中使用select ...for update nowait， 通过添加for update nowait语句，将这条记录锁住，避免其他用户更新，从而保证后续的更新是在正确的状态下更新的。然后在保持这个链接的状态下，在做更新提交。当然这个有个前提就是要保持链接，就是要对链接要占用较长时间，这个在现在web系统高并发高频率下显然是不现实的。  
     
b.现在的悲观锁法（推荐优先使用）：页面提交前加入悲观锁，不允许其他事务动此记录  
        在修改工资这个**页面做提交**时先查询下，当然这个查询必须也要加锁（select ...for update nowait），有人会说，在这里做个查询确认记录是否有改变不就行了吗，是的，是要做个确认，**只是你不加for update就不能保证你在查询到更新提交这段时间里这条记录没有被其他会话更新过，**所以这种方式也需要在查询时锁定记录，保证在这条记录没有变化的基础上再做更新，若有变化则提示告知用户。  
 **页面做提交**时先加锁查询下,不要一开始就加锁，增加效率。  
解决方案2（乐观锁）  
a.旧值条件（前镜像）法：  
       就是在sql更新时使用旧的状态值做条件，SQL大致如下 Update table set col1 = newcol1value, col2 = newcol2value…. where col1 = oldcol1value and col2 = oldcol2value….，在上面的例子中我们就可以把当前工资作为条件进行更新，如果这条记录已经被其他会话更新过，则本次更新了0行，这里我们应用系统一般会做个提示告知用户重新查询更新。这个取哪些旧值作为条件更新视具体系统实际情况而定。（这种方式有可能发生阻塞，如果应用其他地方使用悲观锁法长时间锁定了这条记录，则本次会话就需要等待，所以使用这种方式时最好统一使用乐观锁法。ABA问题）  
  
b.使用版本列法（推荐优先使用）：  
       其实这种方式是一个特殊化的前镜像法，就是不需要使用多个旧值做条件，只需要在**表上加一个版本列**，这一列可以是NUMBER或 DATE/TIMESTAMP列，加这列的作用就是用来记录这条数据的版本（在表设计时一般我们都会给每个表增加一些NUMBER型和DATE型的冗余字段，以便扩展使用，这些冗余字段完全可以作为版本列用），在应用程序中我们每次操作对版本列做维护即可。在更新时我们把**上次版本作为条件进行更新**。  
  
c.使用校验和法（不推荐）  
  
d.使用ORA\_ROWSCN法（不推荐）       
  
**结论：**  
        综上所述，我们对丢失更新问题建议**采取上面的悲观锁b方法或乐观锁b方法**（红字体已标注），其实这两种方式的本质都一样，都是在更新提交时做一次查询确认在更新提交，我个人觉得都是乐观的做法，区别在于悲观锁b方法是通过**select..for  
update方式，这个可能会导致其他会话的阻塞，而乐观锁b方法需要多一个版本列的维护。**  
        个人建议：在**用户并发数比较少且冲突比较严重的应用系统中选择悲观锁b方法，其他情况首先乐观锁版本列法。**