# 事务

## 什么是事务

是指用户定义的一个数据库操作系列，这些操作要么全部执行，要么全部不执行，是一个不可分割的工作单位。

## 事务的四个特性

**原子性(Atomic)**：事务必须是原子工作单元；对于其数据修改，要么全都执行，要么全都不执行。通常，与某个事务关联的操作具有共同的目标，并且是相互依赖的。如果系统只执行这些操作的一个子集，则可能会破坏事务的总体目标。原子性消除了系统处理操作子集的可能性。

**一致性(Consistency)**：事务的一致性指的是在一个事务执行之前和执行之后数据库都必须处于一致性状态。这种特性称为事务的一致性。假如数据库的状态满足所有的完整性约束，就说该数据库是一致的。

**隔离性(Isolation)**：由并发事务所作的修改必须与任何其它并发事务所作的修改隔离。事务查看数据时数据所处的状态，到底是另一个事务执行之前的状态还是中间某个状态，相互之间存在什么影响，是可以通过隔离级别的设置来控制的。

**持久性(Durability)**：事务结束后，事务处理的结果必须能够得到固化，即写入数据库文件中即使机器宕机数据也不会丢失，它对于系统的影响是永久性的。

## 事务并发控制

如果不对事务进行并发控制可能会产生以下情况：

**第一类丢失更新（Update Lost）**：此种更新丢失是因为回滚的原因，所以也叫回滚丢失。此时两个事务同时更新count，两个事务都读取到100，事务一更新成功并提交，count=100+1=101，事务二出于某种原因更新失败了，然后回滚，事务二就把count还原为它一开始读到的100，此时事务一的更新就这样丢失了。

**脏读（Dirty Read）**：此种异常时因为一个事务读取了另一个事务修改了但是未提交的数据。举个例子，事务一更新了count=101，但是没有提交，事务二此时读取count，值为101而不是100，然后事务一出于某种原因回滚了，然后第二个事务读取的这个值就是噩梦的开始。

**不可重复读（Not Repeatable Read）**：此种异常是一个事务对同一行数据执行了两次或更多次查询，但是却得到了不同的结果，也就是在一个事务里面你不能重复（即多次）读取一行数据，如果你这么做了，不能保证每次读取的结果是一样的，有可能一样有可能不一样。造成这个结果是在两次查询之间有别的事务对该行数据做了更新操作。举个例子，事务一先查询了count，值为100，此时事务二更新了count=101，事务一再次读取count,值就会变成101，两次读取结果不一样。

**第二类丢失更新（Second Update Lost）**：此种更新丢失是因为更新被其他事务给覆盖了，也可以叫覆盖丢失。举个例子，两个事务同时更新count，都读取100这个初始值，事务一先更新成功并提交，count=100+1=101，事务二后更新成功并提交，count=100+1=101,由于事务二count还是从100开始增加，事务一的更新就这样丢失了。

**幻读（Phantom Read）**：幻读和不可重复读有点像，只是针对的不是数据的值而是数据的数量。此种异常是一个事务在两次查询的过程中数据的数量不同，让人以为发生幻觉，幻读大概就是这么得来的吧。举个例子，事务一查询order表有多少条记录，事务二新增了一条记录，然后事务一查了一下order表有多少记录，发现和第一次不一样，这就是幻读。

## 事务隔离级别

**读未提交（Read Uncommitted）**：该隔离级别指即使一个事务的更新语句没有提交,但是别的事务可以读到这个改变，几种异常情况都可能出现。极易出错，没有安全性可言，基本不会使用。

**读已提交（Read Committed）**：该隔离级别指一个事务只能看到其他事务的已经提交的更新，看不到未提交的更新，消除了脏读和第一类丢失更新，这是大多数数据库的默认隔离级别，如Oracle,Sqlserver。

**可重复读（Repeatable Read）**：该隔离级别指一个事务中进行两次或多次同样的对于数据内容的查询，得到的结果是一样的，但不保证对于数据条数的查询是一样的，只要存在读改行数据就禁止写，消除了不可重复读和第二类更新丢失，这是Mysql数据库的默认隔离级别。

**串行化（Serializable）**：意思是说这个事务执行的时候不允许别的事务并发执行.完全串行化的读，只要存在读就禁止写,但可以同时读，消除了幻读。这是事务隔离的最高级别，虽然最安全最省心，但是效率太低，一般不会用。

# 锁

事务隔离级别是通过锁实现的。

## 锁的分类

一般可以分为两类，一个是悲观锁，一个是乐观锁，悲观锁一般就是我们通常说的数据库锁机制，乐观锁一般是指用户自己实现的一种锁机制，比如hibernate实现的乐观锁甚至编程语言也有乐观锁的思想的应用。

### 悲观锁

**悲观锁**：顾名思义，就是很悲观，它对于数据被外界修改持保守态度，认为数据随时会修改，所以整个数据处理中需要将数据加锁。悲观锁一般都是依靠关系数据库提供的锁机制，事实上关系数据库中的行锁，表锁不论是读写锁都是悲观锁。

#### ****悲观锁按照使用性质划分：****

**共享锁（Share locks简记为S锁）**：也称读锁，事务A对对象T加s锁，其他事务也只能对T加S，多个事务可以同时读，但不能有写操作，直到A释放S锁。

如果两个事务对同一个资源上了共享锁，事务A 想更新该数据，那么它必须等待 事务B 释放其共享锁。

**排它锁（Exclusivelocks简记为X锁）**：也称写锁，事务A对对象T加X锁以后，其他事务不能对T加任何锁，只有事务A可以读写对象T直到A释放X锁。

**更新锁（简记为U锁）**：用来预定要对此对象施加X锁，它允许其他事务读，但不允许再施加U锁或X锁；当被读取的对象将要被更新时，则升级为X锁，主要是用来防止死锁的。因为使用共享锁时，修改数据的操作分为两步，首先获得一个共享锁，读取数据，然后将共享锁升级为排它锁，然后再执行修改操作。这样如果同时有两个或多个事务同时对一个对象申请了共享锁，在修改数据的时候，这些事务都要将共享锁升级为排它锁。这些事务都不会释放共享锁而是一直等待对方释放，这样就造成了死锁。如果一个数据在修改前直接申请更新锁，在数据修改的时候再升级为排它锁，就可以避免死锁。

#### ****悲观锁按照作用范围划分：****

**行锁**：锁的作用范围是行级别，数据库能够确定那些行需要锁的情况下使用行锁，如果不知道会影响哪些行的时候就会使用表锁。举个例子，一个用户表user，有主键id和用户生日birthday当你使用update … where id=?这样的语句数据库明确知道会影响哪一行，它就会使用行锁，当你使用update … where birthday=?这样的的语句的时候因为事先不知道会影响哪些行就可能会使用表锁。

**表锁**：锁的作用范围是整张表。

### 乐观锁

**乐观锁**：顾名思义，就是很乐观，每次自己操作数据的时候认为没有人回来修改它，所以不去加锁，但是在更新的时候会去判断在此期间数据有没有被修改，需要用户自己去实现。既然都有数据库提供的悲观锁可以方便使用为什么要使用乐观锁呢？对于读操作远多于写操作的时候，大多数都是读取，这时候一个更新操作加锁会阻塞所有读取，降低了吞吐量。最后还要释放锁，锁是需要一些开销的，我们只要想办法解决极少量的更新操作的同步问题。换句话说，如果是读写比例差距不是非常大或者你的系统没有响应不及时，吞吐量瓶颈问题，那就不要去使用乐观锁，它增加了复杂度，也带来了额外的风险。

#### ****乐观锁实现方式：****

**版本号（记为version）**：就是给数据增加一个版本标识，在数据库上就是表中增加一个version字段，每次更新把这个字段加1，读取数据的时候把version读出来，更新的时候比较version，如果还是开始读取的version就可以更新了，如果现在的version比老的version大，说明有其他事务更新了该数据，并增加了版本号，这时候得到一个无法更新的通知，用户自行根据这个通知来决定怎么处理，比如重新开始一遍。这里的关键是判断version和更新两个动作需要作为一个原子单元执行，否则在你判断可以更新以后正式更新之前有别的事务修改了version，这个时候你再去更新就可能会覆盖前一个事务做的更新，造成第二类丢失更新，所以你可以使用update … where … and version=”old version”这样的语句，根据返回结果是0还是非0来得到通知，如果是0说明更新没有成功，因为version被改了，如果返回非0说明更新成功。

**时间戳（timestamp）**：和版本号基本一样，只是通过时间戳来判断而已，注意时间戳要使用数据库服务器的时间戳不能是业务系统的时间。

**待更新字段**：和版本号方式相似，只是不增加额外字段，直接使用有效数据字段做版本控制信息，因为有时候我们可能无法改变旧系统的数据库表结构。假设有个待更新字段叫count,先去读取这个count,更新的时候去比较数据库中count的值是不是我期望的值（即开始读的值），如果是就把我修改的count的值更新到该字段，否则更新失败。java的基本类型的原子类型对象如AtomicInteger就是这种思想。

**所有字段**：和待更新字段类似，只是使用所有字段做版本控制信息，只有所有字段都没变化才会执行更新。

#### ****乐观锁几种方式的区别：****

新系统设计可以使用version方式和timestamp方式，需要增加字段，应用范围是整条数据，不论那个字段修改都会更新version,也就是说两个事务更新同一条记录的两个不相关字段也是互斥的，不能同步进行。旧系统不能修改数据库表结构的时候使用数据字段作为版本控制信息，不需要新增字段，待更新字段方式只要其他事务修改的字段和当前事务修改的字段没有重叠就可以同步进行，并发性更高。