**一：事务（Transaction）关系型数据库 RDBMS**

**1.1：事务的定义**

事务（transaction）：是将一系列数据操作捆绑成一个整体进行统一管理。

如果某个事务执行成功，那么事务中进行的所有数据更改都会提交，永久的保存在数据库中！

如果某个事务中途取消或者执行失败，那么事务中进行的所有数据更改都不会提交，会恢复到事务之前的状态！

简单来说：事务就是一条或者多条sql语句的集合！

**1.2：事务的特性（ACID）**

原子性 (Atomicity)：在事务中的操作，要么都执行，要么都不执行！

一致性（Consistency）：事务必须保证数据库从一个一致性的状态变成另一个一致性的状态！

隔离性（Isolation）:每个事务之间互不干扰！哪怕是并发执行也不干扰！

持久性（Durability）:事务一旦被改变，那么对数据库中数据的影响是永久性的！

**1.3：小例子**

jdbc版本事务回滚

**public void** fromToBank(String name, **double** money) {  
 **try** {  
 String sql = **"UPDATE bank SET money=money+? WHERE `name`=?"**; *//sql语句* Object[] params = {money, name}; *//sql传参* **int** num = executeUpdate(sql, params);  
 **if** (num > 0) {  
 System.***out***.println(**"成功"**); *//判断是否成功* } **else** {  
 System.***out***.println(**"失败"**);  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
}

@Test  
**public void** text() **throws** Exception {  
 **try** {  
 **super**.**con** = getConnection(); *//连接数据库的方法* **con**.setAutoCommit(**false**); *//关闭自动提交事务* fromToBank(**"小黑"**, -1000);  
 **if** (**true**) { *//判断是否回滚*

**throw new** Exception(**"转账失败========"**);  
 }fromToBank(**"小白"**, 1000);  
 **con**.commit();  *//提交事务* } **catch** (Exception e) {  
 e.printStackTrace();  
 **try** {  
 **con**.rollback(); *//事务回滚*  
 } **catch** (SQLException e1) {  
 e1.printStackTrace();  
 }  
  
 }  
}

sql版本的 事务回滚

-- 取消自动提交事务

SET autocommit=0;

BEGIN; -- 开启事务

UPDATE teacher SET sal=sal-1000 WHERE `NAME`='小白1';

SAVEPOINT ponit1; -- 设置回滚点

UPDATE teacher SET sal=0 WHERE `NAME`='小白1';

ROLLBACK TO ponit1; -- 返回到指定的回滚点 ROLLBACK; -- 回滚 直接到达最初的位置

COMMIT; -- 提交事务

**1.4：使用jdbc和事务模拟转账操作！**

注意点：

01.connection连接必须是同一个

02.connnection.setAutoCommit(fasle) //关闭自动提交

03.自已模拟出现一个异常，在catch代码块中增加connection.rollback()

04.connnection.commit（）提交事务

1.5：事务的书写原则

01.事务尽可能简短

因为事务启动后，数据库会专门保留资源来保证事务的ACID特定！如果事务繁多会影响我们的系统性能！

02.事务中涉及到的数据量尽量小

当高并发的情况产生时，事务操作的数据量越小，各个事务之前对数据的争夺就越小！

03.查询数据的时候尽量不要使用事务

因为查询不会对数据库的数据做改变！

04.事务操作过程中，尽量不要出现用户输入等待的情况

如果等待时间过长，会占用系统资源，导致系统崩溃！

**二：事务隔离级别**

int num=0;

num=10;

**2.1.事务隔离级别的引入**

在数据库操作中，为了有效保证并发读取数据的正确性，提出的事务隔离级别。

更新丢失

两个事务都同时更新一行数据，一个事务对数据的更新把另一个事务对数据的更新覆盖了。这是因为系统没有执行任何的锁操作，因此并发事务并没有被隔离开来。

脏读

一个事务读取到了另一个事务未提交的数据操作结果。这是相当危险的，因为很可能所有的操作都被回滚。

不可重复读

不可重复读（Non-repeatable Reads）：一个事务对同一行数据重复读取两次，但是却得到了不同的结果。

包括以下情况：

（1）虚读：事务T1读取某一数据后，事务T2对其做了修改，当事务T1再次读该数据时得到与前一次不同的值。

（2） 幻读（Phantom Reads）：事务在操作过程中进行两次查询，第二次查询的结果包含了第一次查询中未出现的数据或者缺少了第一次查询中出现的数据（这里并不要求两次查询的SQL语句相同）。这是因为在两次查询过程中有另外一个事务插入数据造成的。

**2.2：什么是事务隔离级别**

针对于上诉所阐述的问题，数据库给我们提供一个解决的方案===》 事务隔离级别

**2.3：事务隔离级别的分类**

未授权读取

也称为读未提交（Read Uncommitted）：允许脏读取，但不允许更新丢失。如果一个事务已经开始写数据，则另外一个事务则不允许同时进行写操作，但允许其他事务读此行数据。该隔离级别可以通过“排他写锁”实现。

授权读取

也称为读提交（Read Committed）：允许不可重复读取，但不允许脏读取。这可以通过“瞬间共享读锁”和“排他写锁”实现。读取数据的事务允许其他事务继续访问该行数据，但是未提交的写事务将会禁止其他事务访问该行。

可重复读取（Repeatable Read）

可重复读取（Repeatable Read）：禁止不可重复读取和脏读取，但是有时可能出现幻读数据。这可以通过“共享读锁”和“排他写锁”实现。读取数据的事务将会禁止写事务（但允许读事务），写事务则禁止任何其他事务。

序列化（Serializable）

序列化（Serializable）：提供严格的事务隔离。它要求事务序列化执行，事务只能一个接着一个地执行，不能并发执行。仅仅通过“行级锁”是无法实现事务序列化的，必须通过其他机制保证新插入的数据不会被刚执行查询操作的事务访问到。

隔离级别越高，越能保证数据的完整性和一致性，但是对并发性能的影响也越大。对于多数应用程序，可以优先考虑把数据库系统的隔离级别设为Read Committed。它能够避免脏读取，而且具有较好的并发性能。尽管它会导致不可重复读、幻读和第二类丢失更新这些并发问题，在可能出现这类问题的个别场合，可以由应用程序采用悲观锁或乐观锁来控制。

注意点：

事务隔离级别越高，效率越低，但是安全性越高！

**2.4：mysql事务隔离级别演示**

01.查询mysql默认的事务隔离级别

SELECT @@tx\_isolation;

02.修改事务隔离级别为读未提交（ReadUncommitted）

SET tx\_isolation="Read-Uncommitted"

03.演示脏读

cmd命令行：

begin;

修改数据

在小海豚中查询，发现能看到没有提交的数据！

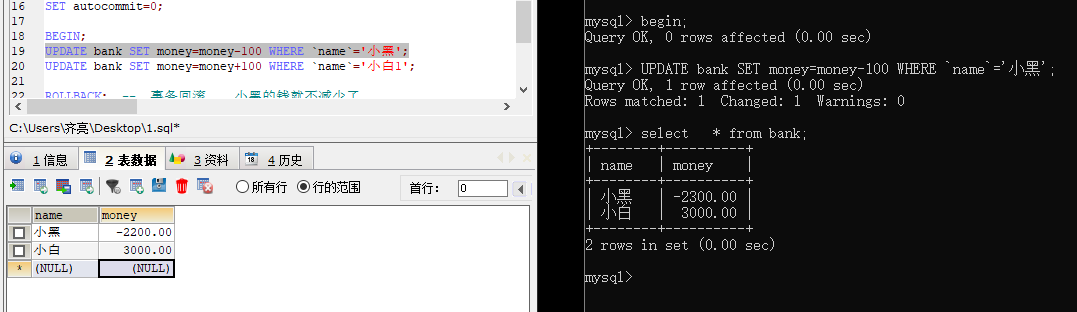
04.演示更新丢失

cmd命令行：

begin;

修改数据

在小海豚中也修改同一行数据，发现老转圈（等待）



cmd运行时和sqlyong是分开的两个没有任何关系