|  |  |
| --- | --- |
| 成绩评定 |  |
| 教师签名 |  |

**嘉应学院 计算机学院**

**数据库课程设计**

**实验报告**

|  |  |
| --- | --- |
| **课程名称：** | **数据库课程设计** |
| **开课学期：** | **2018-2019第一学期** |
| **班 级：** | **软件工程1602** |
| **指导老师：** |  |
| **实验题目：** | **实验13： 事务的使用** |
| **学 号：** | **161060009** |
| **姓 名：** | **黄佳俊** |

实验13 数据库事务设计

一、实验目的

通过实验，掌握数据库管理系统中事务的概念及设计方法。

二、实验原理

事务是作为单个逻辑工作单元执行的一系列操作。一个逻辑工作单元必须有四个属性，称为 ACID（原子性、一致性、隔离性和持久性）属性，只有这样才能成为一个事务：

原子性（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。）；

一致性（事务在完成时，必须使所有的数据都保持一致状态。在相关数据库中，所有规则都必须应用于事务的修改，以保持所有数据的完整性。事务结束时，所有的内部数据结构（如 B 树索引或双向链表）都必须是正确的。）；

隔离性（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，要么是另一并发事务修改它之前的状态，要么是另一事务修改它之后的状态，事务不会查看中间状态的数据。这称为可串行性，因为它能够重新装载起始数据，并且重播一系列事务，以使数据结束时的状态与原始事务执行的状态相同。）；

持久性（事务完成之后，它对于系统的影响是永久性的。该修改即使出现系统故障也将一直保持。）

mysql 事务定义

START TRANSACTION;

（事务体）

COMMIT; (提交）

ROLLBACK ; （回滚）

三、实验环境

操作系统： win7

开发环境：数据库管理服务器 MYSQL 5.5

四、实验内容方法

(一）从业务逻辑中抽出事务

根据事务的ACID 特性，找出那些是要么全做、要么全不做的，密切关联的业务（原子性），导致数据库发生改变的（要求一致性）（如一系列的SQL修改操作，不是查询），，不受干扰，永久发生作用的业务。

注意

\*\* 不能将没有关联的业务设计为一个事务。

\*\* 不能设计过长、过于复杂的事务

例如商贸活动很多中有， a.商场采购产品，b.供应商提供产品， c 客户浏览产品，d. 商场年终盘点，e.创建订单信息，f.订购产品，g.运输商发货,h.更新库存...

事务一：

b.供应商提供产品

c 客户浏览产品

e.创建订单

f.订购产品

h.更新库存

事务二：

e.创建订单

f.订购产品

g.运输商发货

h.更新库存

事务三：

e.创建订单

f.订购产品

h.更新库存

分析哪个事务设计是合适的。

（二）在存储过程中实现事务

1.用老师提供的数据库脚本trade2\_script.txt 创建数据库trade2

2. 创建存储过程 transp ，检验事务效果-破坏性实验

Delimiter //

Create Procedure transp()

Begin

Start Transaction; /\* 开始事务 \*/

/\* e.创建订单 订单号 11088， 客户编码ALFKI，客户编号 6，订购时间 '2018-12-17' \*/

insert into orders (OrderID,CustomerID,EmployeeID,OrderDate) values(11088,'ALFKI',6,'2018-12-17');

/\* f.订购产品： 订单ID=11088, 产品ID=13，单价=6，数量=10，折扣=0.1 \*/

insert into orderdetails values(11088,13,6,10,0.1);

/\* h.更新库存 \*/

update products set unitsinstock = unitsinstock - 10 where productid=13;

/\* f.订购产品：订单ID=11088, 产品ID=18，单价=62.5，数量=15，折扣=0.13 \*/

insert into orderdetails values(11088,18,62.5,15,0.13);

/\* h.更新库存 \*/

update products set unitsinstock = unitsinstock - 15 where productid=E18; /\* 破坏性 错误无 E18\*/

commit; /\* 提交 \*/

end //

Delimiter ;

测试事务

测试用例： 在更新库存中出错 productid=E18,

预期：在订单表中撤销订单号及相关信息， 在订单细节表中撤销订购的产品信息，在产品表中恢复原来的库存。

执行存储过程之前（事务发生前 检查产品库存： productid 13 库存 24; productid18 库存 42;

select unitsinstock from products where productid=13;

select unitsinstock from products where productid=18;

调用存储过程 call transp();

检查与预期是否一致？将会报错，但是将出现不一致的情况。

select \* from orderdetails where orderid=11088 and productid=13;

select \* from orderdetails where orderid=11088 and productid=18;

不一致需要逐条小心地恢复 (还必须考虑删除顺序）

update products set unitsinstock = unitsinstock + 10 where productid=13;

delete from orderdetails where orderid=11088;

delete from orders where orderid=11088;

3. 错误回滚 rollback

加入一条 错误处理标记 declare CONTINUE HANDLER FOR SQLEXCEPTION SET t\_err = 1;

if t\_err =1 then

Rollback; /\* 有sql 错误 回滚 \*/

else

commit; /\* 无 sql 错误 提交 \*/

end if;

将存储过程改造为：

Delimiter //

Create Procedure transp1()

Begin

declare t\_err int default 0;

declare CONTINUE HANDLER FOR SQLEXCEPTION SET t\_err = 1;

Start Transaction;

insert into orders (OrderID,CustomerID,EmployeeID,OrderDate) values(11088,'ALFKI',6,'2018-12-10');

/\* 订单ID=11088, 产品ID=13，单价=6，数量=10，折扣=0.1 \*/

insert into orderdetails values(11088,13,6,10,0.1);

update products set unitsinstock = unitsinstock - 10 where productid=13;

/\* 订单ID=11088, 产品ID=18，单价=62.5，数量=15，折扣=0.13 \*/

insert into orderdetails values(11088,18,62.5,15,0.13);

update products set unitsinstock = unitsinstock - 15 where productid=E18; /\* 错误无 E18\*/

if t\_err =1 then

Rollback; /\* 有sql 错误 回滚 \*/

else

commit; /\* 无 sql 错误 提交 \*/

end if;

end //

Delimiter ;

执行此存储过程

call transp1();

按上面的方法测试事务。

（三） 在Java JDBC 实现事务

1.安装JDBC驱动

将老师提供的JDBC驱动添加到 classpath 中。

2. 在Java 实现上述事务 Jdbc\_tran1.java

注意点： 1.关闭自动提交 conn.setAutoCommit(false);

2. 一定要捕获错误，并且作Rollback处理！ conn.rollback();

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.Date;

import java.sql.ResultSet;

import java.sql.Statement;

public class Jdbc\_tran1 {

public static void addOrderDetail1(int orderid, String customid, int empid,String dayt,

int productid,double price, int num1, double disc) {

Connection conn = null;

Statement state = null; //声明一个Statement对象

try {

String dbURL = "jdbc:mysql://localhost:3306/trade2";

String user = "root";

String password = "";

conn = DriverManager.getConnection(dbURL, user, password);

conn.setAutoCommit(false);

state=conn.createStatement();

//插入SQL

// insert into orders (OrderID,CustomerID,EmployeeID,OrderDate) values(11088,'ALFKI',6,'2018-12-10');

String S\_sql ="insert into orders (OrderID,CustomerID,EmployeeID,OrderDate) values( " +

" '"+orderid+"' , '"+ customid +"' , '"+ empid +"' , '"+ dayt +"')";

int rowsql = state.executeUpdate(S\_sql);

String S\_sql1 ="insert into orderdetails values ( " +

" '"+orderid+"' , '"+ productid +"' , '"+ price+"' , '"+ num1 +"' , '"+ disc +"' )";

int rowsql1 = state.executeUpdate(S\_sql1);

// 更新语

String S\_sql2 ="update Products set UnitsInStock = UnitsInStock - '"+ num1 +"' where ProductID='"+ productid +"' ";

int rowsql2 = state.executeUpdate(S\_sql2);

/\*

可以用下列方法，强制回滚

if((rowsql==1) && (rowsql1==1) && (rowsql2==1)){

conn.commit();

} else {

conn.rollback();

}

\*/

conn.commit();

} catch (SQLException ex) {

// rollback 事务

try{

if(conn != null)

conn.rollback();

}catch(SQLException e){

System.out.println(e.getMessage());

}

System.out.println(ex.getMessage());

} finally {

try {

if(state!= null) state.close();

if(conn != null) conn.close();

} catch (SQLException e) {

System.out.println(e.getMessage());

}

}

}

public static void main(String[] args) {

/\* 订单ID=11089, 产品ID=18，单价=62.5，数量=15，折扣=0.13 \*/

addOrderDetail1(11089, "ALFKI",6,"2018-12-11",

18,62.5,8,0.13);

}

}

执行 java Jdbc\_tran1.java

检查是否生效

破坏性实验-破坏完整性

\*\* //破坏完整性 orderdetails 的产品id, productid 参照 products表的productid, 当orderdetails 表中试图添加产品表中没有的产品id时（如88），会不会所有都被回滚。

addOrderDetail1(11090, "ALFKI",6,"2018-11-22",

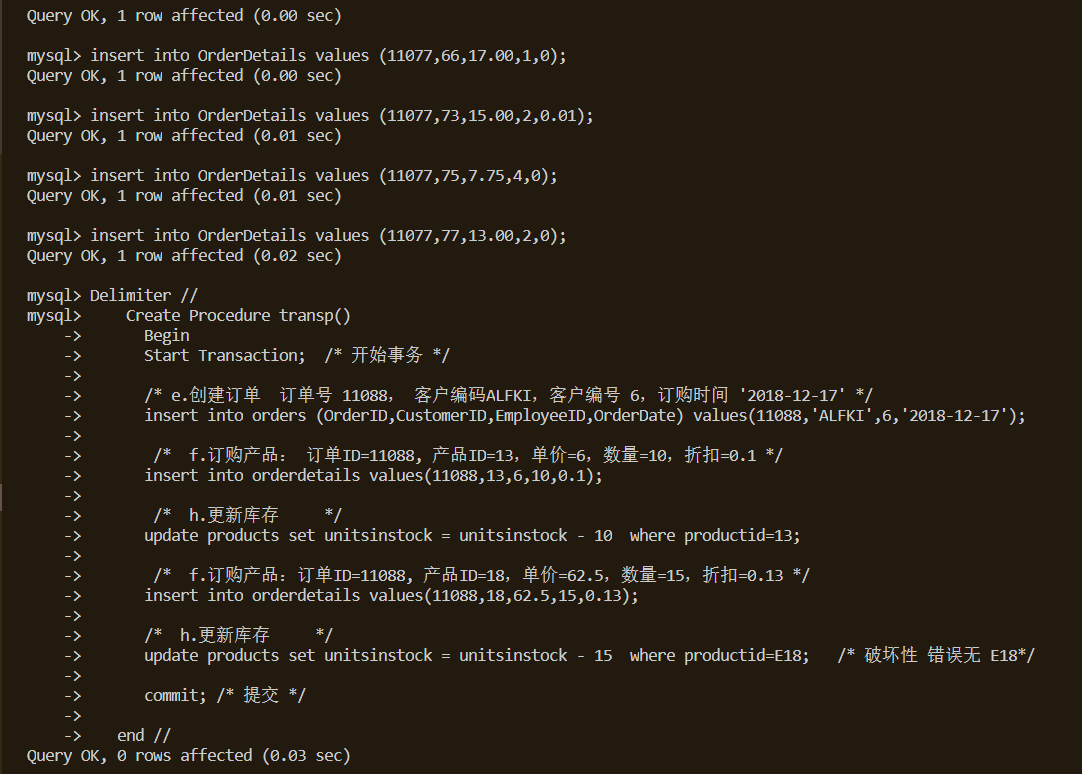
88,62.5,8,0.13);

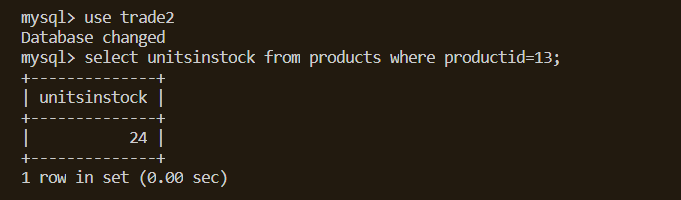
五、实验扩展

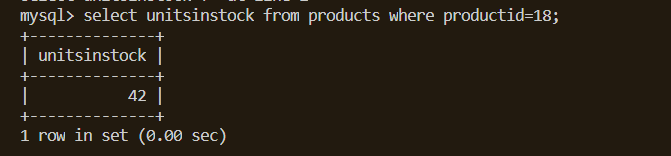
将事务设计的思想应用到实际项目中。

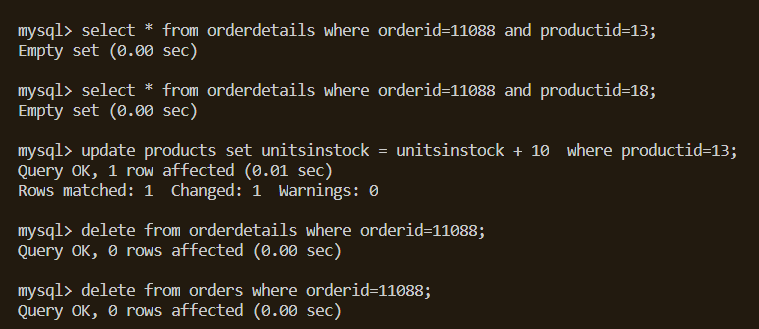
1. 实验结果：

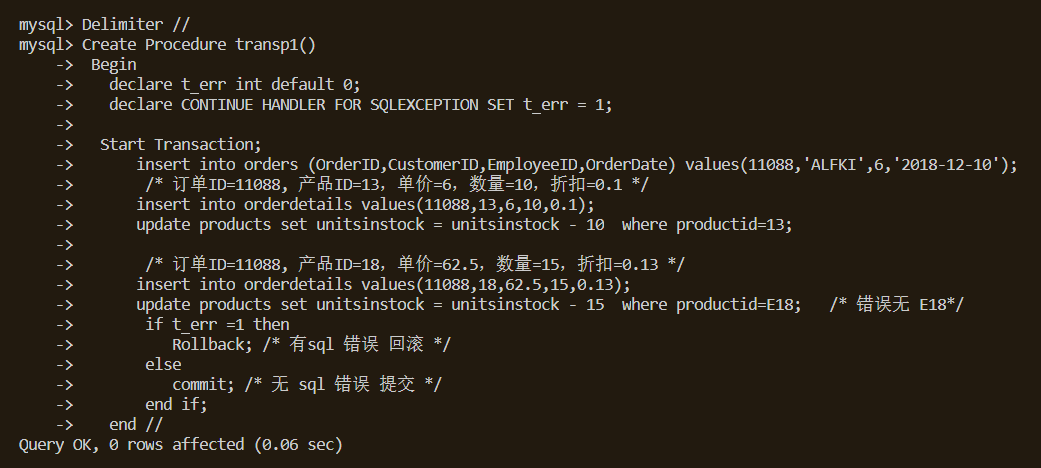
业务逻辑的事务使用











1. jdbc中的Java的事务处理。

执行代码如下：

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.Date;

import java.sql.ResultSet;

import java.sql.Statement;

public class Jdbc\_tran1 {

public static void addOrderDetail1(int orderid, String customid, int empid, String dayt, int productid,

double price, int num1, double disc) {

Connection conn = null;

Statement state = null; // 声明一个Statement对象

try {

String dbURL = "jdbc:mysql://localhost:3306/trade2";

String user = "root";

String password = "";

conn = DriverManager.getConnection(dbURL, user, password);

conn.setAutoCommit(false);

state = conn.createStatement();

// 插入SQL

// insert into orders (OrderID,CustomerID,EmployeeID,OrderDate)

// values(11088,'ALFKI',6,'2018-12-10');

String S\_sql = "insert into orders (OrderID,CustomerID,EmployeeID,OrderDate) values( " + " '" + orderid

+ "' , '" + customid + "' , '" + empid + "' , '" + dayt + "')";

int rowsql = state.executeUpdate(S\_sql);

String S\_sql1 = "insert into orderdetails values ( " + " '" + orderid + "' , '" + productid + "' , '"

+ price + "' , '" + num1 + "' , '" + disc + "' )";

int rowsql1 = state.executeUpdate(S\_sql1);

// 更新语

String S\_sql2 = "update Products set UnitsInStock = UnitsInStock - '" + num1 + "' where ProductID='"

+ productid + "' ";

int rowsql2 = state.executeUpdate(S\_sql2);

/\*

\* 可以用下列方法，强制回滚 if((rowsql==1) && (rowsql1==1) && (rowsql2==1)){ conn.commit();

\* } else { conn.rollback(); }

\*/

conn.commit();

} catch (SQLException ex) {

// rollback 事务

try {

if (conn != null)

conn.rollback();

} catch (SQLException e) {

System.out.println(e.getMessage());

}

System.out.println(ex.getMessage());

} finally {

try {

if (state != null)

state.close();

if (conn != null)

conn.close();

} catch (SQLException e) {

System.out.println(e.getMessage());

}

}

}

public static void main(String[] args) {

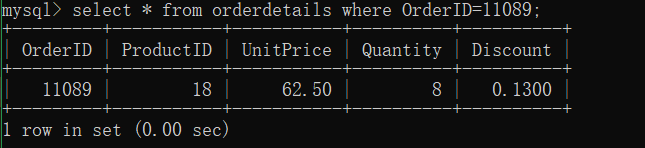
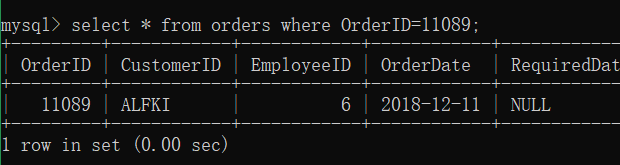
/\* 订单ID=11089, 产品ID=18，单价=62.5，数量=15，折扣=0.13 \*/

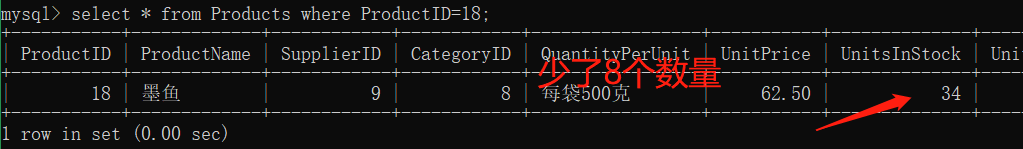
addOrderDetail1(11089, "ALFKI", 6, "2018-12-11", 18, 62.5, 8, 0.13);

}

}

通过run执行后查询结果：



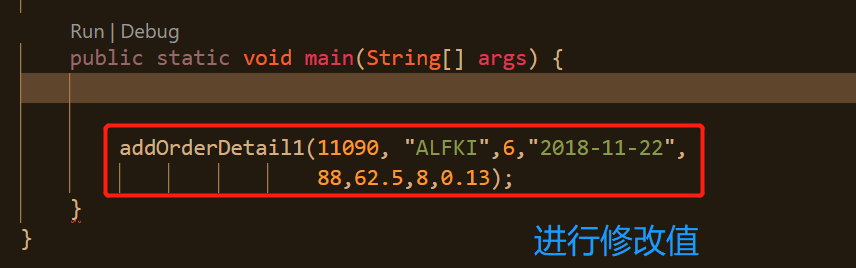


执行成功的。

破坏完整性 orderdetails 的产品id, productid 参照 products表的productid, 当orderdetails 表中试图添加产品表中没有的产品id时（如88），会不会所有都被回滚。

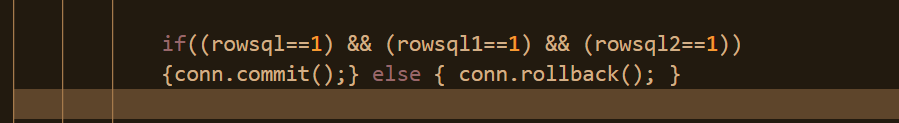
addOrderDetail1(11090, "ALFKI",6,"2018-11-22",

88,62.5,8,0.13);



执行run后，可知道，本来就有一个回滚条件可知，

因为条件是‘与’条件，



只要有一处不成功执行，就会全部回滚，即（conn.rollback()被执行）！！！

**扩展实验，**

本实践项目是执行了进销存中得一些事务

情况如下：

添加入库信息

// 在事务中添加入库信息

    public static boolean insertRukuInfo(TbRukuMain ruMain) {

        try {

            boolean autoCommit = conn.getAutoCommit();

            conn.setAutoCommit(false);

            // 添加入库主表记录

            insert("insert into tb\_ruku\_main values('" + ruMain.getRkId() + "','" + ruMain.getPzs() + "',"

                    + ruMain.getJe() + ",'" + ruMain.getYsjl() + "','" + ruMain.getGysname() + "','"

                    + ruMain.getRkdate() + "','" + ruMain.getCzy() + "','" + ruMain.getJsr() + "','" + ruMain.getJsfs()

                    + "')");

            Set<TbRukuDetail> rkDetails = ruMain.getTabRukuDetails();

            for (Iterator<TbRukuDetail> iter = rkDetails.iterator(); iter.hasNext();) {

                TbRukuDetail details = iter.next();

                // 添加入库详细表记录

                insert("insert into tb\_ruku\_detail values('" + ruMain.getRkId() + "','" + details.getTabSpinfo() + "',"

                        + details.getDj() + "," + details.getSl() + ")");

                // 添加或修改库存表记录

                Item item = new Item();

                item.setId(details.getTabSpinfo());

                TbSpinfo spInfo = getSpInfo(item);

                if (spInfo.getId() != null && !spInfo.getId().isEmpty()) {

                    TbKucun kucun = getKucun(item);

                    if (kucun.getId() == null || kucun.getId().isEmpty()) {

                        insert("insert into tb\_kucun values('" + spInfo.getId() + "','" + spInfo.getSpname() + "','"

                                + spInfo.getJc() + "','" + spInfo.getCd() + "','" + spInfo.getGg() + "','"

                                + spInfo.getBz() + "','" + spInfo.getDw() + "'," + details.getDj() + ","

                                + details.getSl() + ")");

                    } else {

                        int sl = kucun.getKcsl() + details.getSl();

                        update("update tb\_kucun set kcsl=" + sl + ",dj=" + details.getDj() + " where id='"

                                + kucun.getId() + "'");

                    }

                }

            }

            conn.commit();

            conn.setAutoCommit(autoCommit);

        } catch (SQLException e) {

            e.printStackTrace();

            return false;

        }

        return true;

    }

// 在事务中添加入库退货信息

    public static boolean insertRkthInfo(TbRkthMain rkthMain) {

        try {

            boolean autoCommit = conn.getAutoCommit();

            conn.setAutoCommit(false);

            // 添加入库退货主表记录

            insert("insert into tb\_rkth\_main values('" + rkthMain.getRkthId() + "','" + rkthMain.getPzs() + "',"

                    + rkthMain.getJe() + ",'" + rkthMain.getYsjl() + "','" + rkthMain.getGysname() + "','"

                    + rkthMain.getRtdate() + "','" + rkthMain.getCzy() + "','" + rkthMain.getJsr() + "','"

                    + rkthMain.getJsfs() + "')");

            Set<TbRkthDetail> rkDetails = rkthMain.getTbRkthDetails();

            for (Iterator<TbRkthDetail> iter = rkDetails.iterator(); iter.hasNext();) {

                TbRkthDetail details = iter.next();

                // 添加入库详细表记录

                insert("insert into tb\_rkth\_detail values('" + rkthMain.getRkthId() + "','" + details.getSpid() + "',"

                        + details.getDj() + "," + details.getSl() + ")");

                // 添加或修改库存表记录

                Item item = new Item();

                item.setId(details.getSpid());

                TbSpinfo spInfo = getSpInfo(item);

                if (spInfo.getId() != null && !spInfo.getId().isEmpty()) {

                    TbKucun kucun = getKucun(item);

                    if (kucun.getId() != null && !kucun.getId().isEmpty()) {

                        int sl = kucun.getKcsl() - details.getSl();

                        update("update tb\_kucun set kcsl=" + sl + " where id='" + kucun.getId() + "'");

                    }

                }

            }

            conn.commit();

            conn.setAutoCommit(autoCommit);

        } catch (SQLException e) {

            e.printStackTrace();

            return false;

        }

        return true;

    }

// 在事务中添加销售信息

    public static boolean insertSellInfo(TbSellMain sellMain) {

        try {

            boolean autoCommit = conn.getAutoCommit();

            conn.setAutoCommit(false);

            // 添加销售主表记录

            insert("insert into tb\_sell\_main values('" + sellMain.getSellId() + "','" + sellMain.getPzs() + "',"

                    + sellMain.getJe() + ",'" + sellMain.getYsjl() + "','" + sellMain.getKhname() + "','"

                    + sellMain.getXsdate() + "','" + sellMain.getCzy() + "','" + sellMain.getJsr() + "','"

                    + sellMain.getJsfs() + "')");

            Set<TbSellDetail> rkDetails = sellMain.getTbSellDetails();

            for (Iterator<TbSellDetail> iter = rkDetails.iterator(); iter.hasNext();) {

                TbSellDetail details = iter.next();

                // 添加销售详细表记录

                insert("insert into tb\_sell\_detail values('" + sellMain.getSellId() + "','" + details.getSpid() + "',"

                        + details.getDj() + "," + details.getSl() + ")");

                // 修改库存表记录

                Item item = new Item();

                item.setId(details.getSpid());

                TbSpinfo spInfo = getSpInfo(item);

                if (spInfo.getId() != null && !spInfo.getId().isEmpty()) {

                    TbKucun kucun = getKucun(item);

                    if (kucun.getId() != null && !kucun.getId().isEmpty()) {

                        int sl = kucun.getKcsl() - details.getSl();

                        update("update tb\_kucun set kcsl=" + sl + " where id='" + kucun.getId() + "'");

                    }

                }

            }

            conn.commit();

            conn.setAutoCommit(autoCommit);

        } catch (SQLException e) {

            e.printStackTrace();

            return false;

        }

        return true;

    }

// 在事务中添加销售退货信息

    public static boolean insertXsthInfo(TbXsthMain xsthMain) {

        try {

            boolean autoCommit = conn.getAutoCommit();

            conn.setAutoCommit(false);

            // 添加销售退货主表记录

            insert("insert into tb\_xsth\_main values('" + xsthMain.getXsthId() + "','" + xsthMain.getPzs() + "',"

                    + xsthMain.getJe() + ",'" + xsthMain.getYsjl() + "','" + xsthMain.getKhname() + "','"

                    + xsthMain.getThdate() + "','" + xsthMain.getCzy() + "','" + xsthMain.getJsr() + "','"

                    + xsthMain.getJsfs() + "')");

            Set<TbXsthDetail> xsthDetails = xsthMain.getTbXsthDetails();

            for (Iterator<TbXsthDetail> iter = xsthDetails.iterator(); iter.hasNext();) {

                TbXsthDetail details = iter.next();

                // 添加销售退货详细表记录

                insert("insert into tb\_xsth\_detail values('" + xsthMain.getXsthId() + "','" + details.getSpid() + "',"

                        + details.getDj() + "," + details.getSl() + ")");

                // 修改库存表记录

                Item item = new Item();

                item.setId(details.getSpid());

                TbSpinfo spInfo = getSpInfo(item);

                if (spInfo.getId() != null && !spInfo.getId().isEmpty()) {

                    TbKucun kucun = getKucun(item);

                    if (kucun.getId() != null && !kucun.getId().isEmpty()) {

                        int sl = kucun.getKcsl() + details.getSl();

                        update("update tb\_kucun set kcsl=" + sl + " where id='" + kucun.getId() + "'");

                    }

                }

            }

            conn.commit();

            conn.setAutoCommit(autoCommit);

        } catch (SQLException e) {

            e.printStackTrace();

        }

        return true;

    }