|  |
| --- |
| **计算机专业类课程** |
| **实验报告** |
| **课程名称：软件技术基础综合课程设计**  **学　　院：计算机科学与工程学院**  **专　　业：计算机科学与技术（“互联网+”）**  **学生姓名：**  **学　　号：**  **指导教师：张栗粽** |
|  |
| **日　　期： 2022年 6月 16日** |

**电 子 科 技 大 学**

**实 验 报 告**

**实验二**

1. **实验名称： 多进程多用户文件一致性读写访问设计实现**
2. **实验学时：4**
3. **实验内容和目的：**

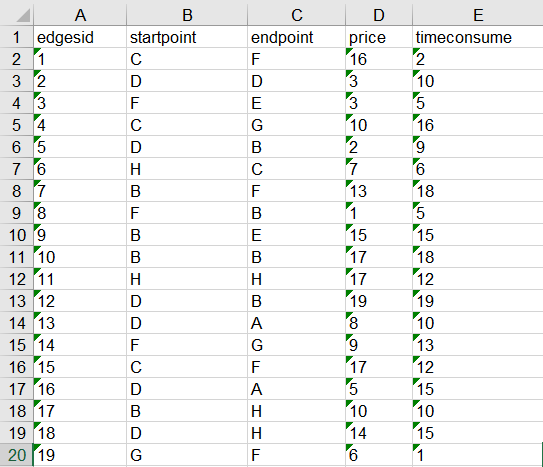
1、设计实现数据表的文件存储方式，能在文件中存储多张数据表，每张数据表可存储多条记录。实现指定表中记录的存储、读写和记录的简单查询与索引查询函数。能够实现单用户和进程对文件数据中记录的写入与查询。

2、实现多进程对单个文件中某表中的记录的互斥写入与查询访问操作，保证表中记录数据的一致性。

3、实现多用户对文件中记录数据的同时写入与查询一致性操作。

1. **实验步骤：**

（一）文件存储方式的数据库实现

基于excel表格优秀的数据存储与处理特点以及其与数据库的高度相似性，我们决定用excel表格来实现文件存储方式的数据库。根据数据库的常规关系模式，excel表统一规定格式：表头位于第一行，是属性；其下并列各个实体。如下图所示。

（二）对数据库的增删改查操作的实现

对数据库中数据的增删改查通过java的poi来实现操作。

实现增删改查的基本思路：在用户输⼊SQL语句后，程序通过调⽤实验三的SQL解析器对该语句进⾏解析。对解析出的结果按SELECT，DELETE，UPDATE，INSERT，CREATE TABLE五种类型分别送⼊对应的函数，并对模拟出的数据库进⾏增删改查操作。

(1)SELECT

对如下的sql查询语句：

select edges\_id from edgesinfo where edges\_id < '8' and start\_point = '南京';

我们的解析器解析出的结果为：

1;edges\_id;edgesinfo;edges\_id,start\_point;1,3;8,南京;1

以上解析结果不同部分由“;”分隔开来，共7部分分别代表：语句类型、需要展⽰的字段名、进⾏操作的表名、条件语句中查询的字段名、条件的类型（<,<=,>,like等类型）、条件的具体值、不同条件间的关系（and,or）。

由于其他类型的sql语句中也会出现条件判断，因此我们⾸先通过如下两个函数对where部分进⾏处理。

//对组合条件（如id > '7' and id < '25'）其中的一个进行筛选，返回符合条件的行号索引

**public** **static** StringBuffer QuerySpecifiedIndex(**int** sheetIndex, String name, **int** type, String condition) {

StringBuffer Index = **new** StringBuffer();

**while** (**true**) {

**try** {

**while**(**true**) {

//读取文件

**boolean** readFileSuccess = *readDataFromFile*();

**if** (!readFileSuccess) **continue**;

//初始化EXCEL中的sheet（表）、row（行）、cell（单元格）

HSSFSheet sheet = *workbook*.getSheetAt(sheetIndex - 1);

HSSFRow titleRow = sheet.getRow(0);

**int** lastCellNum = titleRow.getLastCellNum();

**boolean** judge1 = **false**;//判读是否找到符合条件的数据

**if**(name.equals("null")) {//为null则返回该表的所有字段名

**int** RowNum = sheet.getLastRowNum();//表中的行数

**for**(**int** i = 0; i < RowNum; i++ ) {

HSSFRow row = sheet.getRow(i+1);

HSSFCell flagCell = row.getCell(lastCellNum - 1);

String flag = flagCell.getStringCellValue();

**if** (flag.equals("0")) **continue**;//flag为0代表物品被删除

Index.append(i+1 + ",");

}

Index.deleteCharAt(Index.length()-1);//去除字符串末尾逗号

//对组合条件（如id > '7' and id < '25'）中的and或者or进行处理，并返回最终的索引

**private** **static** String SelectIndex(**int** sheetIndex, String name, **int**[] type, String condition,String relation) {

//对解析出的sql语句进行字符串处理

String[] names = name.split(",");//字段名按逗号分开

String[] conditions = condition.split(",");//条件值按逗号分开

String[] relations = relation.split(",");//关系值按逗号分开

//初始化存储索引的变量

String Indexs = **new** String();

List<String> indexsList = **new** ArrayList<>();

**int**[] orIndex = **new** **int**[100];

**int** t=0;

List<List<String>> indexsLists = **new** ArrayList<List<String>>();

**for**(**int** i=0; i<conditions.length; i++) {//对n个条件进行处理

List<String> index\_tempList = **new** ArrayList<>();

StringBuffer Index\_Modify = *QuerySpecifiedIndex*(sheetIndex, names[i], type[i], conditions[i] );//返回单一条件的索引

String Index\_Modify1 = Index\_Modify.toString();

**if**(Index\_Modify1.equals("null")) {//检测表中是否存在输入的字段名

System.***out***.println("字段名 "+names[i]+" 不存在");

**return** Indexs = "字段名" + names[i] + "不存在";

}

String indexString = Index\_Modify.toString();//StringBuffer转String

String[] indexStrings = indexString.split(",");//索引按字段名分开

**for**(**int** j=0; j<indexStrings.length; j++) {

index\_tempList.add(indexStrings[j]);

}

indexsLists.add(index\_tempList);

}

**if**(!relations[0].equals("null")) {

**for**(**int** i=0; i<relations.length; i++) {//对and和or进行逻辑处理

**if**(relations[i].equals("1")) {//and取交集

indexsLists.get(i+1).retainAll(indexsLists.get(i));

}

**else** {//or取并集

orIndex[t] = i;

t++;

}

}

}

indexsList = indexsLists.get(indexsLists.size()-1);

**for**(**int** i=0; i<t; i++) {

indexsList.removeAll(indexsLists.get(orIndex[i]));

indexsList.addAll(indexsLists.get(orIndex[i]));

}

**for**(**int** i=0; i<indexsList.size(); i++) {

Indexs += indexsList.get(i);//最终索引拼接

**if**(i!=indexsList.size())

Indexs += ",";//每个索引用逗号分隔

}

**return** Indexs;

}

第⼀个函数对单⼀条件如：id < '7' 进⾏处理，并返回符合条件的excel中的⾏号。第⼆个函数基于第⼀个函数的返回值对多个条件之间的“and”及“or”进⾏处理，并返回最终符合所有条件的索引。基于上述的⾏号索引，通过展⽰内容函数ShowData完成select类型sql语句的处理。

//输入最终的符合条件的索引，展示对应的数据

**private** **static** String ShowData(**int** sheetIndex, String name, String Index) {//三个变量分别为表索引，字段名，符合条件的行号索引

String showInfo = **new** String();

**try** {

**while**(**true**) {

**if**(Index.isEmpty()){//没有符合条件的数据

showInfo = "没有符合条件的数据。";

System.***out***.println("没有符合条件的数据。");

System.***out***.println();

**return** showInfo;

}

//对字段名和行号索引进行拆分

String[] Index1 = Index.split(",");

String[] name1 = name.split(",");

HSSFSheet sheet = *workbook*.getSheetAt(sheetIndex-1);//锁定查找的表

HSSFRow titleRow = sheet.getRow(0);//锁定title行

**int** lastCellNum = titleRow.getLastCellNum();//获得列数

**int**[] columnIndex = **new** **int**[name1.length];//确定对应字段名

**int**[] columnIndexJudge = **new** **int**[name1.length];//判断是否存在该字段名

**if**(name1[0].equals("all")) {//对应SELECT \* 即返回所有字段名

**int** CellNum = titleRow.getLastCellNum();

HSSFRow showTitleRow = sheet.getRow(0);

**for**(**int** j = 0; j < CellNum - 1; j++) {//用for循环输出一行中的所有数据

HSSFCell ShowCell = showTitleRow.getCell(j);

String value = ShowCell.getStringCellValue();

System.***out***.print(value + " ");

showInfo += value+" ";

}

showInfo = showInfo.trim();//去除首尾空格

showInfo += ",";//不同行数据用逗号分隔

System.***out***.println();

**for**(**int** i = 0; i < Index1.length; i++) {

HSSFRow ShowRow = sheet.getRow(Integer.*valueOf*(Index1[i]));//String型索引转int

**for**(**int** j = 0; j < CellNum - 1; j++) {

HSSFCell ShowCell = ShowRow.getCell(j);

String value = ShowCell.getStringCellValue();//获取单元格值

System.***out***.print(value + " ");

showInfo += value+" ";//拼接数据，不同单元格用空格分隔

}

showInfo = showInfo.trim();

showInfo += ",";

System.***out***.println();

}

showInfo = showInfo.substring(0, showInfo.length()-1);

System.***out***.println();

**return** showInfo;

}

**for**(**int** i = 0; i < lastCellNum; i++) {//适用于一般情况

HSSFCell titleCell = titleRow.getCell(i);

String title = titleCell.getStringCellValue();

**for**(**int** j = 0; j < name1.length; j++) {

**if**(title.equals(name1[j])) {

columnIndex[j] = i;//获得所需字段名的列索引

columnIndexJudge[j] = 1;//判断是否由列与输入的字段名对应，没有则返回字段名xxx不存在

}

}

}

**for**(**int** i = 0; i < name1.length; i++) {//返回对应的报错信息

**if**(columnIndexJudge[i] == 0) {

System.***out***.println("字段名 " + name1[i] + " 不存在");

showInfo += "字段名 " + name1[i] + " 不存在";

**return** showInfo;

}

}

**for**(**int** j = 0; j < columnIndex.length; j++) {//利用获得的列索引输出对应数据的title

HSSFCell titleCell = titleRow.getCell(columnIndex[j]);

String value = titleCell.getStringCellValue();

System.***out***.print(value + " ");

showInfo += value+" ";

}

showInfo = showInfo.trim();

showInfo += ",";

System.***out***.println();

**for**(**int** i = 0; i < Index1.length; i++) {//利用列索引获取对应数据的值

HSSFRow ShowRow = sheet.getRow(Integer.*valueOf*(Index1[i]));

**for**(**int** j = 0; j < columnIndex.length; j++) {

HSSFCell ShowCell = ShowRow.getCell(columnIndex[j]);

String value = ShowCell.getStringCellValue();

System.***out***.print(value + " ");

showInfo += value+" ";

}

showInfo = showInfo.trim();

showInfo += ",";

System.***out***.println();

}

System.***out***.println();

showInfo = showInfo.substring(0, showInfo.length()-1);//去除尾部逗号

**return** showInfo;

}

}**catch** (Exception e) {

e.printStackTrace();

**return** showInfo;

}

}

(2)DELET

delete语句同select语句类似，在对语句内的where部分进⾏处理并返回符合条件的⾏号索引后，通过DeleteData函数完成对应数据的删除。

此处由于Java在通过poi对xls格式的⽂件进⾏处理时⽆法实现真正物理意义上的删除（删除后的⾏还存 在，只是内容为空），不便于后续操作，因此我们为每⼀个实体即每⼀⾏添加属性flag。falg为1表⽰该 实体存在，flag为0表⽰该实体已被删除。由此避免该问题的产⽣。

//利用查找到的索引删除行

**private** **static** String DeleteData(**int** sheetIndex, String RowIndex\_String ) {

String showInfo = **new** String();

**try** {

**while**(**true**) {

**boolean** readFileSuccess = *readDataFromFile*();//读数据

String[] Index1 = RowIndex\_String.split(",");//分索引

**if** (!readFileSuccess) **continue**;

HSSFSheet sheet = *workbook*.getSheetAt(sheetIndex-1);//定表名

**for**(**int** i = 0; i < Index1.length; i++) {

**int** RowIndex = Integer.*valueOf*(Index1[i]);//锁定需要删除的目标行

HSSFRow deleteRow = sheet.getRow(RowIndex);

**int** lastCellNum = deleteRow.getLastCellNum();

HSSFCell deleteCell = deleteRow.getCell(lastCellNum - 1);

deleteCell.setCellValue("0");//将目标行的flag标签置为0（1代表存在，0代表删除）

}

*saveFile*();//保存数据

showInfo = "删除成功！";

System.***out***.println("删除成功！");

System.***out***.println();

**return** showInfo;

}

}**catch**(Exception e) {

e.printStackTrace();

**return** showInfo;

}

}

(3)UPDATE

查找下标过程类似，不必过多阐述。通过ModifyData函数对符合条件的数据进⾏修改：

//修改数据

**public** **static** String ModifyData(**int** sheetIndex, String name, String element, String RowIndex\_Modify) {

String showInfo = **new** String();

**try** {

**while**(**true**) {

**boolean** readFileSuccess = *readDataFromFile*();//读文件

**if** (!readFileSuccess) **continue**;

String[] Index1 = RowIndex\_Modify.split(",");//分索引

HSSFSheet sheet = *workbook*.getSheetAt(sheetIndex-1);//定表名

HSSFRow titleRow = sheet.getRow(0);//确定title行

String[] name1 = name.split(",");

String[] element1 = element.split(",");

**int**[] column\_Index = **new** **int**[name1.length];//存储对应字段名的列号

**for**(**int** i = 0; i < titleRow.getLastCellNum(); i++) {

**for**(**int** j = 0; j < name1.length; j++) {

HSSFCell cell = titleRow.getCell(i);

String value = cell.getStringCellValue();

**if**(value.equals(name1[j])) {//判断是否字段名和title相同

column\_Index[j] = i;//相同则列号存入

}

}

}

**for**(**int** i = 0; i < Index1.length; i++) {

HSSFRow modifyRow = sheet.getRow(Integer.*valueOf*(Index1[i]));//for循环对n行数据进行修改

**for**(**int** j = 0; j < column\_Index.length; j++) {

HSSFCell modifyCell = modifyRow.getCell(column\_Index[j]);

modifyCell.setCellValue(element1[j]);//修改单元格数据

}

}

*saveFile*();//保存数据

showInfo = "修改成功！";

**return** showInfo;

}

}**catch**(Exception e){

e.printStackTrace();

**return** showInfo;

}

}

(4)INSERT

在表中最后⼀⾏进⾏新建⾏的操作，并对各个单元格赋值：

//插入数据

**private** **static** String InsertData(**int** sheetIndex, String name, String element) {

String showInfo = **new** String();

**try** {

**while**(**true**) {

**boolean** readFileSuccess = *readDataFromFile*();//读文件

**if** (!readFileSuccess) **continue**;

String[] element1 = element.split(",");//将插入值分隔

String[] name1 = name.split(",");//插入字段名分隔

HSSFSheet sheet = *workbook*.getSheetAt(sheetIndex - 1);//定表名

HSSFRow row = sheet.getRow(0);

**int** lastRowNum = sheet.getLastRowNum();//初始化行数

**int** lastCellNum = row.getLastCellNum();//初始化列数

HSSFRow nextRow = sheet.createRow(lastRowNum + 1);

**int**[] columnIndex = **new** **int**[name1.length];

//返回字段名的列号，便于修改

**for**(**int** i = 0; i < lastCellNum; i++) {

HSSFCell titleCell = row.getCell(i);

String title = titleCell.getStringCellValue();

**for**(**int** j = 0; j < name1.length; j++) {

**if**(title.equals(name1[j])) {

columnIndex[j] = i;

}

}

}

**for** (**int** i = 0; i < lastCellNum; i++) {

HSSFCell addCell = nextRow.createCell(i);//最后一行新建行

**boolean** judge = **false**;

**for**(**int** j = 0; j < columnIndex.length; j++) {

**if**(i == lastCellNum - 1) {

addCell.setCellValue("1");//flag标签置为1

}

**else** **if**(i == columnIndex[j]) {

addCell.setCellValue(element1[j]);//为单元格赋值

judge = **true**;

}

}

**if**(!judge) {

addCell.setCellValue("null");//没有输入值的单元格为null

}

}

*saveFile*();//保存文件

showInfo = "增加成功！";

System.***out***.println("增加成功！");

System.***out***.println();

**return** showInfo;

}

}**catch**(Exception e) {

e.printStackTrace();

**return** showInfo;

}

}

(5)CREAT TABLE

对上述的创建表的语句，对excel⽂件进⾏新建sheet的操作，再对第⼀⾏添加各字段名：

//创建新表

**private** **static** String createTalbe(String tableName, String filedName) {

String showInfo = **new** String();

**try** {

**while**(**true**) {

**boolean** readFileSuccess = *readDataFromFile*();//读取文件

**if** (!readFileSuccess) **continue**;

String[] filedNames = filedName.split(",");//新表的字段名分隔

HSSFSheet createSheet = *workbook*.createSheet(tableName);//创建新表

HSSFRow firstRow = createSheet.createRow(0);//创建新表第一行

**for**(**int** i = 0; i < filedNames.length; i++) {//为第一行赋title名

HSSFCell createCell = firstRow.createCell(i);

createCell.setCellValue(filedNames[i]);

}

System.***out***.println("增加成功！");

showInfo = "增加成功！";

System.***out***.println();

*saveFile*();

**return** showInfo;

}

}

**catch** (Exception e) {

e.printStackTrace();

**return** showInfo;

}

}

（三）信号量实现互斥与同步

在本项目中，需要互斥的部分为读操作与写操作。在写操作发生时，读操作不能发生，其他写操作亦不能发生。而在读操作时，由于其本身不会对数据库内的数据产生影响，所以读操作本身不利用信号量上锁。而对写操作的限制通过一个读写锁实现。但是这里只对读写上锁并不能保证读操作与写操作内部的有序性，故而引入信号量queue来实现等待队列。举个例子，Reader到来拿走queue，接着拿走rw后归还queue；Writer到来拿走queue，接着被P(rw)阻塞；第二个Reader到来，上来就被P(queue)阻塞，它何时才能执行呢？必须等到第一个Reader归还rw，Writer得到rw执行完毕后，归还queue，第二个Reader才得以执行——queue使第二个Reader强制等待，形成一个“Reader-Writer-第二个Reader”的固定执行顺序——所谓“等待队列”。在以上思路理清之后，创建一个类来管理各种信号量。以上信号量的管理类如下：

**package** test1;

**import** java.util.concurrent.Semaphore;

**public** **class** SharedData {

**public** Semaphore rw = **new** Semaphore(1); // 读写锁（本质是个互斥锁）

**public** Semaphore mutex = **new** Semaphore(1); // 读者计数器 互斥锁

**public** Semaphore queue = **new** Semaphore(1); // 队列锁

**public** **int** readerCnt = 0; // 读者计数器

**public** **int** n = 100;

**public** Semaphore working = **new** Semaphore(n); //一次只能有n个进程在工作

}

而我们只需要对读操作和写操作在进行前后根据互斥与同步的关系加上信号量相关操作即可，如下图所示：

**try** {

// 第一个(cnt==0)读者进行P操作

*sharedData*.queue.acquire();

*sharedData*.mutex.acquire();

**if**(*sharedData*.readerCnt == 0){

*sharedData*.rw.acquire();

}

*sharedData*.readerCnt++;

*sharedData*.mutex.release();

*sharedData*.queue.release();

// 开始读 >\_<

……

// 最后一个(cnt==0)读者进行V操作

*sharedData*.mutex.acquire();

*sharedData*.readerCnt--;

**if**(*sharedData*.readerCnt == 0){

*sharedData*.rw.release();

}

w.displayMessage("查询完毕，输入任意值结束当前任务");

w.displayMessage(Thread.*currentThread*().getName() + " is stoping");

**while**(w.flag) {System.***out***.println("\r");};

Scanner input = **new** Scanner(w.Rmessage);

w.flag = **true**;

input.close();

*sharedData*.mutex.release();

// 睡一觉zzz

Thread.*sleep*(**new** Random().nextInt(2));

对写操作上锁如下：

**try** {

// 核心逻辑

*sharedData*.queue.acquire();

*sharedData*.rw.acquire();

……

w.displayMessage("删除完毕，输入任意值结束当前任务");

w.displayMessage(Thread.*currentThread*().getName() + " is stoping");

**while**(w.flag) {System.***out***.println("\r");};

Scanner input = **new** Scanner(w.Rmessage);

w.flag = **true**;

input.close();

*sharedData*.rw.release();

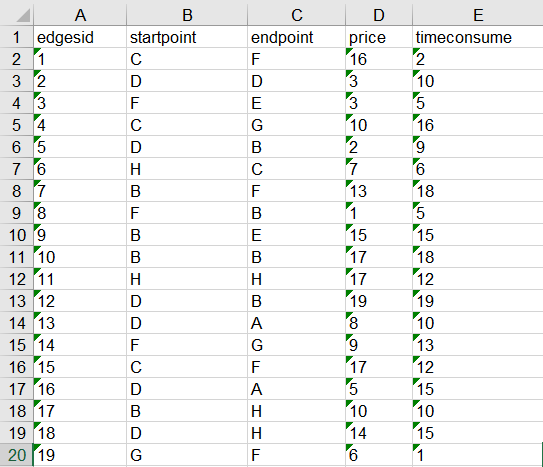
*sharedData*.queue.release();

// 睡一觉zzz

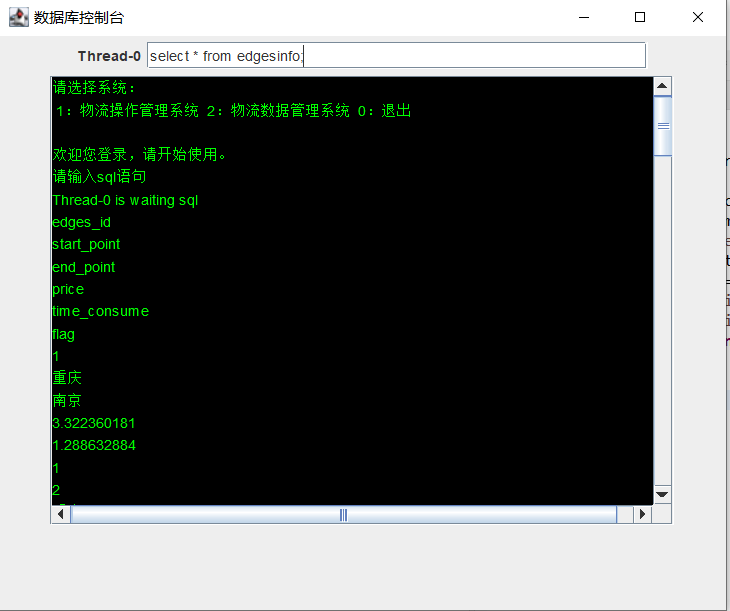
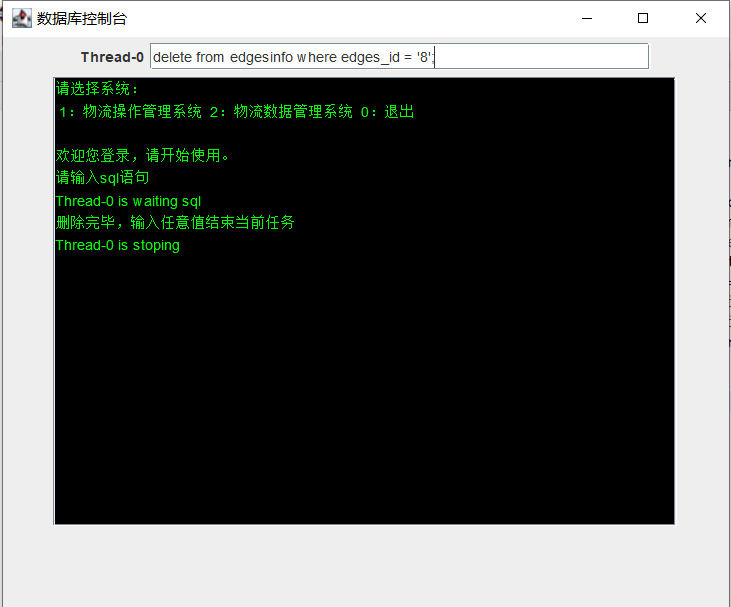
Thread.*sleep*(**new** Random().nextInt(2));

1. **实验结果分析：**

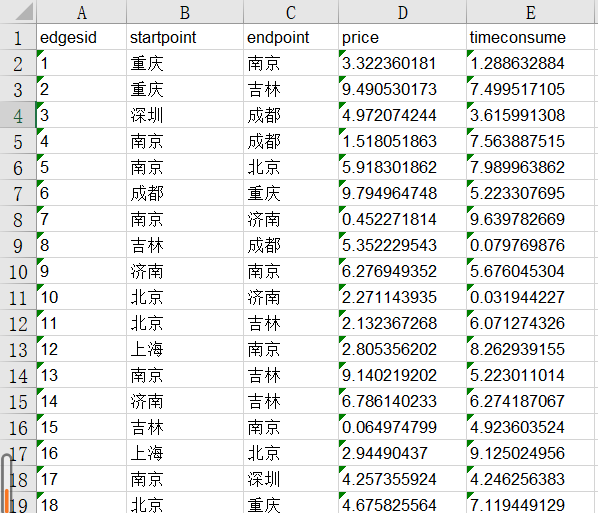
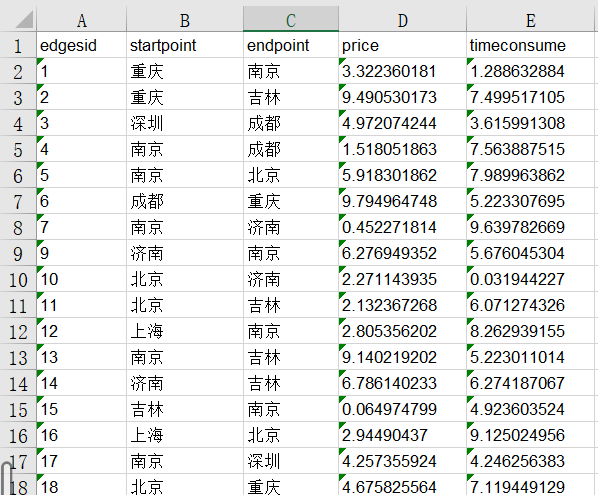
（一）文件存储方式的数据库实现

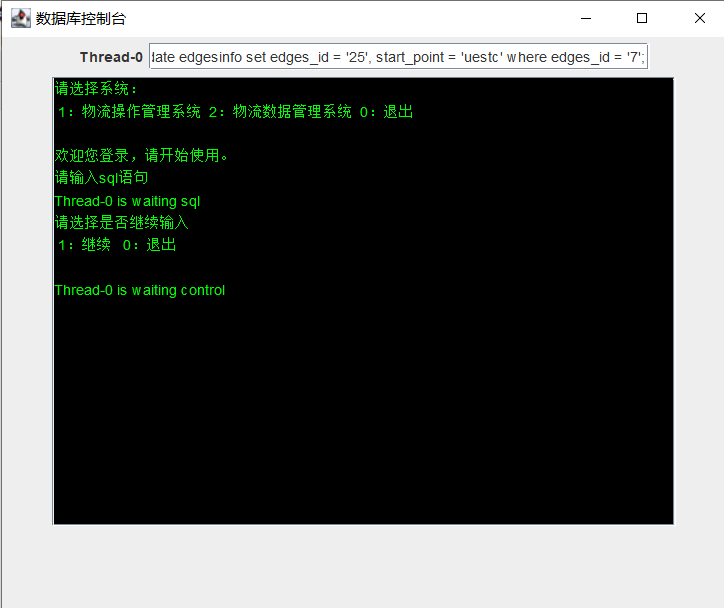
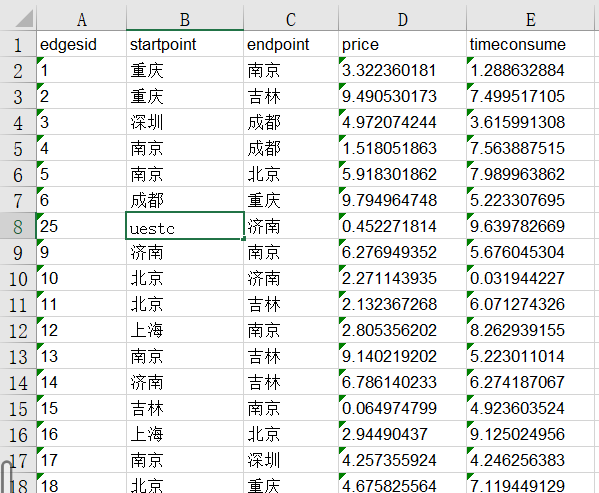
 存储数据库的excel表格如下所示，可见完成了存储的实现：

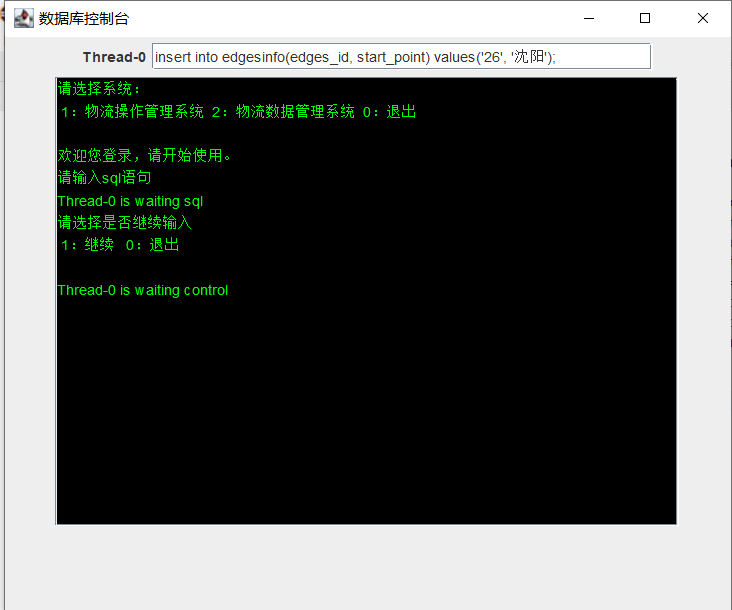
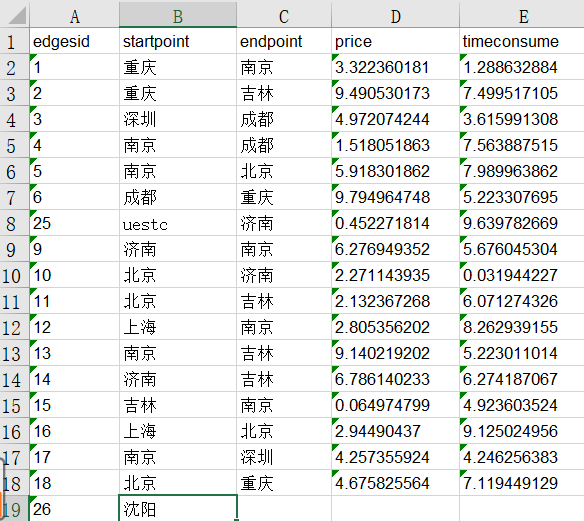
（二）对数据库的增删改查操作的实现

(1) SELECT语句，经过测试，可以在窗口输出需求的数据：

(2) DELET语句经过测试，可以在窗口输出运行结果，并在数据库中完成修改：

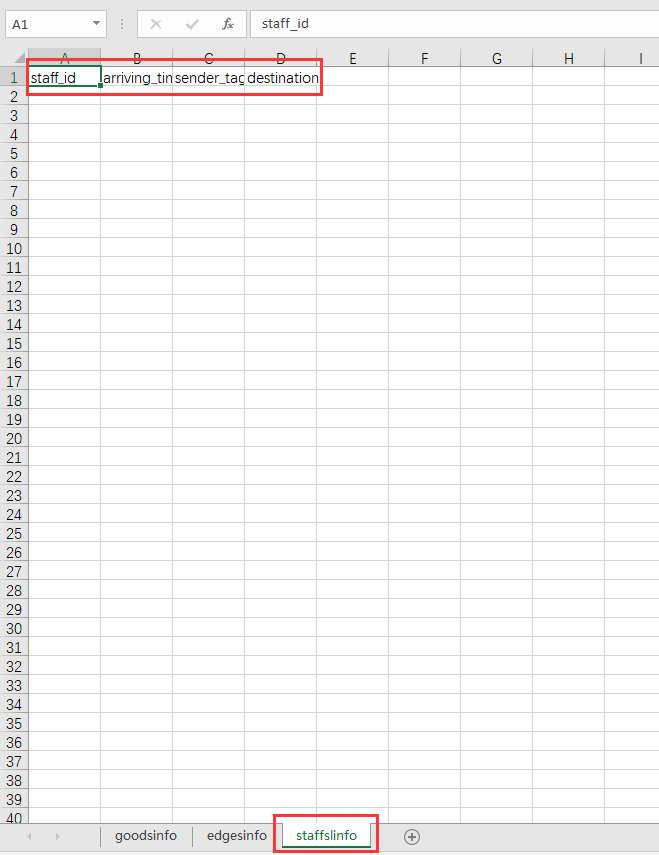


(3) UPDATE语句经过测试，可以在窗口输出运行结果，并在数据库中完成修改： 

(4) INSERT语句经过测试，可以在窗口输出运行结果，并在数据库中完成修改：

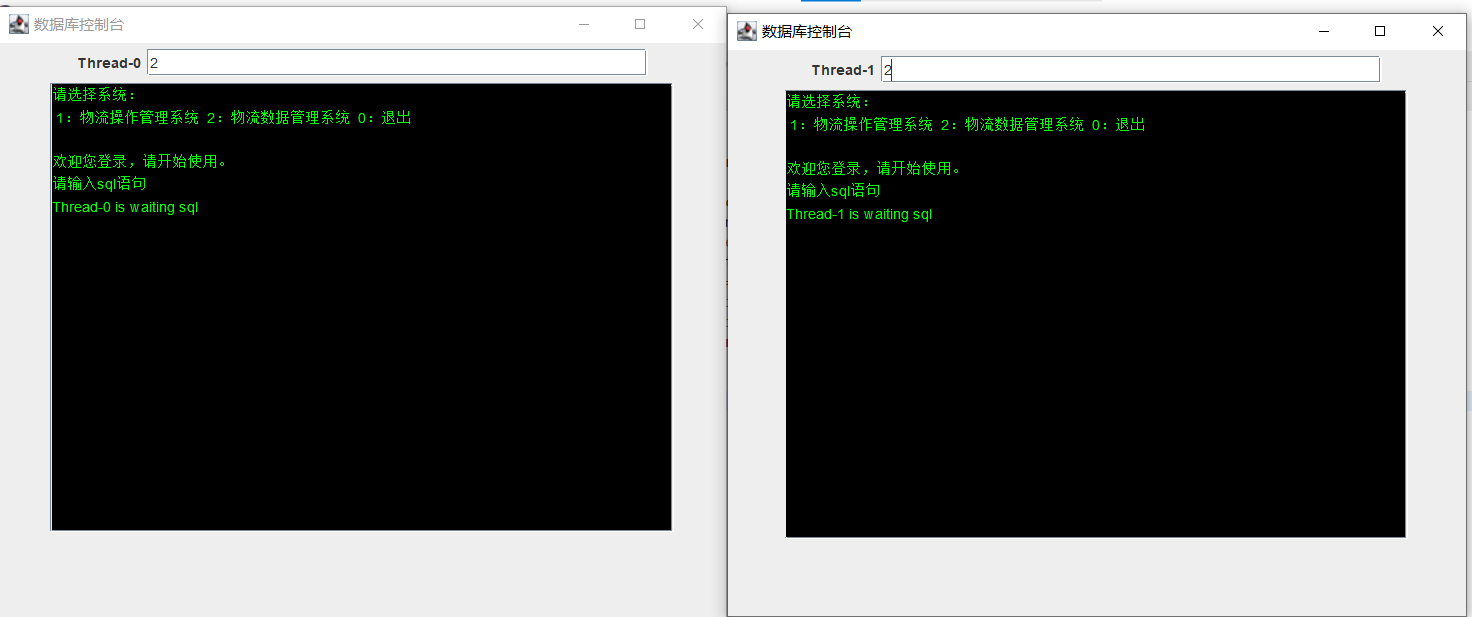
(5) CREATE TABLE 语句经过测试可以完成创建：

create table staffslinfo(staff\_id INT, arriving\_time int, sender\_tag CHAR(20), destination CHAR(20));

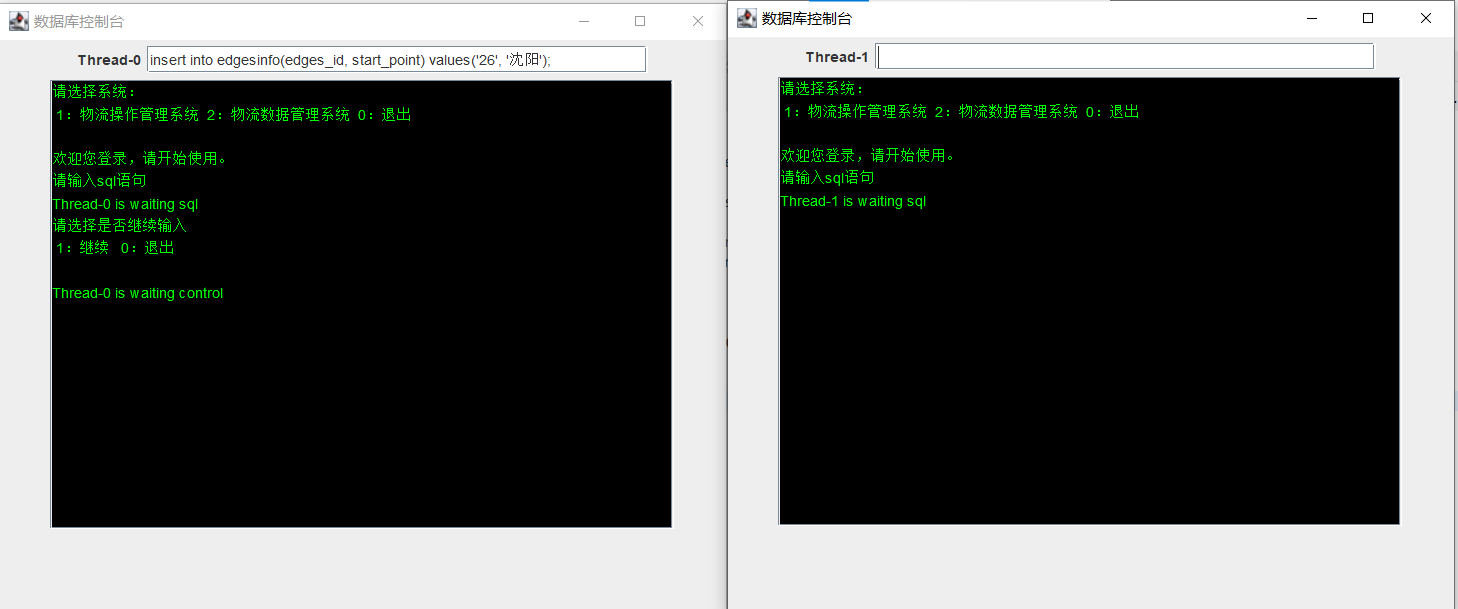


（三）信号量实现互斥与同步

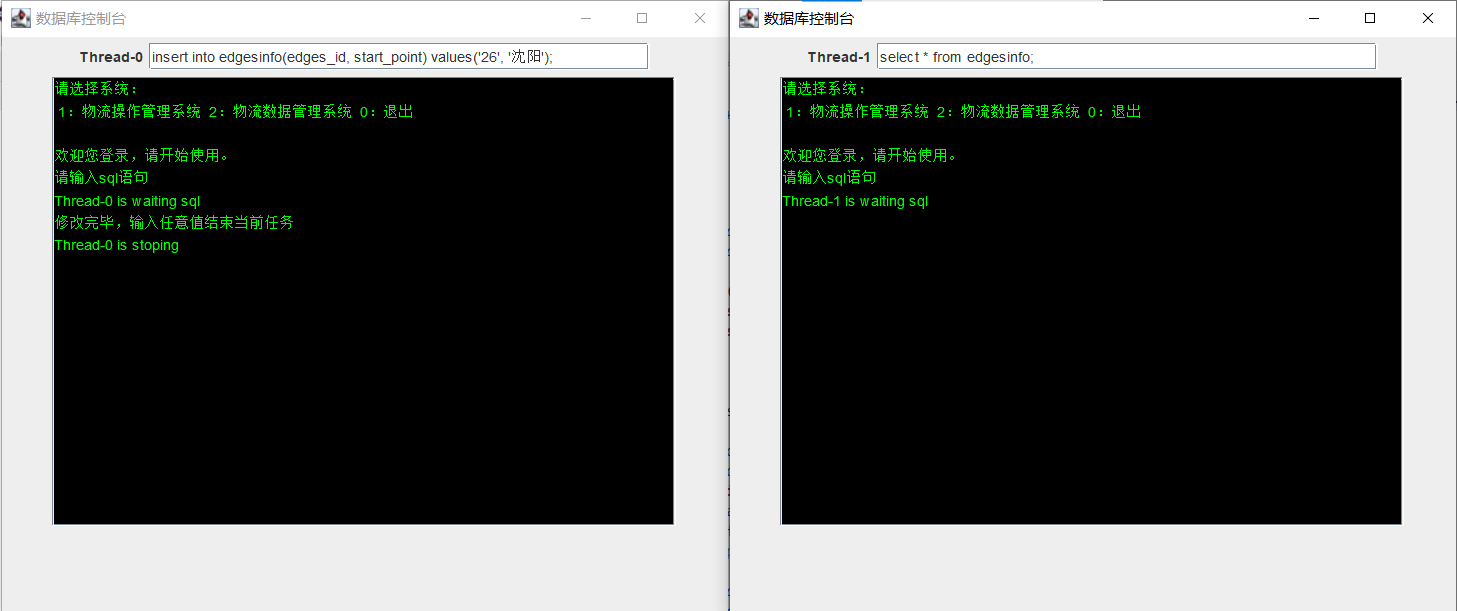
首先创建两个进程：



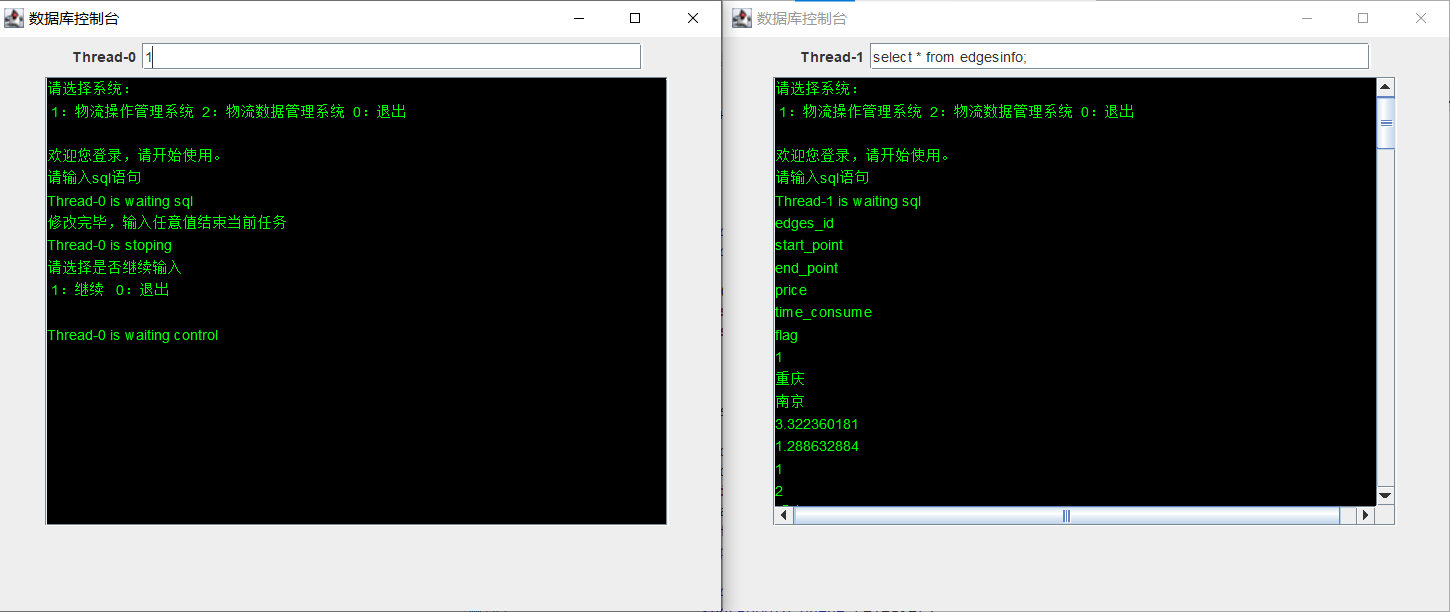
接着输入修改数据库的语句：



在修改结束后，不确认结束，此时输入查询语句，虽然有输入但是无法执行：



此时确认修改结束，查询操作得以执行：



基于以上结果，可以看出，经过信号量的设计，本系统的读写操作已经实现了同步与互斥。

1. **实验二结论、心得体会和改进建议：**

在本次实验中，我们在自己的努力下基于excel与java的poi和信号量等方法实现了数据库的本地存储、增删改查以及读写互斥和进程控制。这些在课堂上提及的理论知识似乎遥不可及，在落地之初咋一看也不容易实现。但是在我们的一步一步努力下，我们不仅完成了实现与设计 ，并且进一步加深了对课堂知识的理解与应用、受益匪浅。