数据库系统概论

2014110099 黄锦畦 2011011239 王庆

**目录**

[1 系统结构设计 3](#_Toc409310762)

[1.1 系统结构 3](#_Toc409310763)

[1.2 查询解析模块 3](#_Toc409310764)

[1.3 系统管理模块 3](#_Toc409310765)

[1.4 记录管理模块 3](#_Toc409310766)

[2 主要模块设计原理 3](#_Toc409310767)

[2.1 记录管理模块 3](#_Toc409310768)

[2.1.1 文件的定义 4](#_Toc409310769)

[2.1.2 存储方法 4](#_Toc409310770)

[2.1.3 记录结构 4](#_Toc409310771)

[2.2 系统管理模块 4](#_Toc409310772)

[2.3 查询解析模块 4](#_Toc409310773)

[2.3.1 字符串分离 4](#_Toc409310774)

[2.3.2 Where语句解析 5](#_Toc409310775)

[2.3.3 赋值判断 5](#_Toc409310776)

[2.3.4 Select join 5](#_Toc409310777)

[3 主要模块接口说明 5](#_Toc409310778)

[3.1 记录管理模块 5](#_Toc409310779)

[3.1.1 内部接口 5](#_Toc409310780)

[3.1.2 外部接口 6](#_Toc409310781)

[3.2 系統管理模块 8](#_Toc409310782)

[3.2.1 内部接口 8](#_Toc409310783)

[3.2.2 外部接口 8](#_Toc409310784)

[3.3 查询解析模块 9](#_Toc409310785)

[3.3.1 内部成员变量 9](#_Toc409310786)

[3.3.2 外部接口 9](#_Toc409310787)

[3.4 索引模块 11](#_Toc409310788)

[4 实验结果 12](#_Toc409310789)

[4.1 文件存储 12](#_Toc409310790)

[4.2 系统管理语句 13](#_Toc409310791)

[4.2.1 CREATE DATABASE *orderDB*; 创建名为orderDB的数据库。 14](#_Toc409310792)

[4.2.2 DROP DATABASE orderDB; 删除名为orderDB的数据库。 14](#_Toc409310793)

[4.2.3 USE orderDB; 当前数据库切换为orderDB。 15](#_Toc409310794)

[4.2.4 SHOW TABLES; 列出当前数据库包含的所有表。 16](#_Toc409310795)

[4.2.5 CREATE TABLE customer 16](#_Toc409310796)

[4.2.6 DROP TABLE testTable; 17](#_Toc409310797)

[4.2.7 DESC TABLE customer; 17](#_Toc409310798)

[4.3 查询解析语句 18](#_Toc409310799)

[4.3.1 Insert 输入检测 18](#_Toc409310800)

[4.3.2 Delete删除数据 19](#_Toc409310801)

[4.3.3 Update更新数据 20](#_Toc409310802)

[4.3.4 Select选择数据 21](#_Toc409310803)

[5 小组分工 22](#_Toc409310804)

[6 参考文献以及github链接 22](#_Toc409310805)

[7 编程环境说明 22](#_Toc409310806)

# 系统结构设计

## 系统结构

本项目中的数据库管理系统由三个模块组成,分别为查询解析、系统管理和纪录管理。我们没有选用助教提供的页式文件系统,而是自己写了相关的系统,所以结构和要求的不太一样。

## 查询解析模块

该模块解析SQL语句,能将输入的SQL语句解析成关系代数表达式,并生成查询执行计划。该模块实现下列四个常见的语法选项，以及部分系统管理语句。

* INSERT
* DELETE
* UPDATE
* SELECT
* SHOW
* USE
* CREATE
* DESC
* DROP

## 系统管理模块

本项目要管理多个数据库，实现在各个数据库之间切换。该模块存储表，列的格式，协助查询解析模块访问文件系统，输出查询结果。

## 记录管理模块

该模块是DBMS的文件系统,管理存储数据库记录以及元数据的文件。文件系统是由我们小组自己编写, 其中参考了Microsoft SQL Server 2003 的结构, 实现了下列的文件管理：

* 新建文件、删除文件、打开文件、关闭文件。
* 创建数据库、删除数据库、切换数据库、列出现有的所有数据库以及其包含的所有表名。
* 创建表、删除表、列出现有的所有表以及其模式信息。
* 插入记录、删除记录、更新记录、获取属性值满足特定条件的记录。

# 主要模块设计原理

## 记录管理模块

从最底层的文件系统说起，这个模块包含了文件的定义和一些基本的存储方法。

### 文件的定义

我们把一个表独立为一个文件，一个数据库为一个文件夹，在这个基础上建立了一个文件系统。文件采用了页头和数据块的模式。文件的空间定在8192byte，页头占8byte，剩下的8184byte为数据块。页头由4个unsigned short组成，纪录了表的ID (FileID) ，页数 (PageID) ，第一个可以写入的位置 (firstFreeOffset) 和还剩多少空间 (freeCount) ；数据块则由char型构成。

### 存储方法

读写需要用二进制的格式，这样可以避免程序错误解析unsigned的数据，以及非二进制读写地址偶数位对齐的问题。在读取页的时候，必须先提供表的名字，亦即文件的名字，还有具体页数，用fseek函数读取指定页。这个系统十分直观快捷，页为子文件的机制避免了空页占据大量I/O时间。而为了在执行时提供更高的效率，在程序执行的时候，除非缓存区已经溢出，否则一切的I/O操作都会在结束的时候才会执行。缓存区由一个C++ Map Container管理，以rowID (由表的ID 和页数组成) 作为键值。缓存区没有该rowID的时候才读取，容量不够才写入。

### 记录结构

记录的结构十分简单，先是由一个bool表达该列数据是否为null，接着就是该列的数据。在所有数据都表达后，有一个offset的unsigned short，该数值表达了该项数据是否已被删除。我们设计了一个以offset来判断存在与否的系统。首先，整个文件都会以最大值来初始化。当一项数据存在/没有被删除时，offset必定为unsigned short的最大值。删除纪录时，依据该纪录的位置，更新firstFreeOffset。要是该纪录比现有的第一个空余空间靠前，就把firstFreeOffset覆盖，并把本来的firstFreeOffset存入该纪录的offset。这样，删除的纪录就成为了一个单向的链表，链表的最后会以最大值来表示。插入一个数据时，会从页头读取第一个空余空间的位置并覆盖，再用第一个空余空间的offset补上firstFreeOffset。我们利用归纳法，证明了这个纪录系统可以准确的表达纪录的状态。这个系统的好处是节省了一个isDeleted的bool，坏处则是读取单一纪录的时间可能变得极长。但是数据库的用途基本不会只读取一个纪录，而是必须遍历一次表。这个时候，我们就可以透过编程的方法，使用内存实现链表处理，而不是外存多余的bool，来达到同样的O(n)效果。

## 系统管理模块

参考了SQL server 管理文件的手法，每一个数据库都会建立sysObject和sysColumn来管理表和解释数据。sysObject文件存储了所有表的属性，包括名字，ID和拥有的列。sysColumn则纪录所有列的讯息，包括所属的表ID，列ID，名字，可否为null，类型，长度和在纪录中的相对位置。配合一些C++对数据类形的巧妙处理，这些讯息可以用来解析原始的数据。

## 查询解析模块

### 字符串分离

读取包含SQL语句的执行文本，认为分号为一条语句的结束。由于语句解析部分非常冗杂，该解析语句要求输入语句必须严格分离，否则认为该语句错误。

比如：update publisher set nation = 'CNA' , name = 'DATABASE' where nation = 'PRC';

通过识别语句第一个关键字进入不同类型语句的解析模块。

### Where语句解析

Select join的where语句解析和其余的where分类处理。Select join的where语句解析，操作符左边是attr，右边可能是attr也可能是属性对应的比较值。其余的where分类处理左边是attr，右边是属性对应的比较值。

### 赋值判断

该数据库管理系统目前只支持int和varchar操作。可以判断输入的字符串是不是数字以及varchar字符串有没有超出范围长度。

### Select join

分别找出tableA和tableB符合要求的值，将其存到内存，根据tableA和tableB的连接要求筛选出符合要求的记录，将其要求输出的列输出。

# 主要模块接口说明

## 记录管理模块

记录管理模块在文件dbManage.h以及dbManage.cpp中定义，包含用于文件的内部操作的接口，包括：创建文件、删除文件、打开文件、关闭文件、读文件、写文件等。同时，随着用户对数据的读写操作，相应的文件头也需要进行更新，因而也需要设计接口实现这些服务。

### 内部接口

* 接口：void pop();

功能：从缓存区弹出一个页面，并实现写入。

* 接口：void push(USRT FileID, USRT PageID, Node\* buffer);

功能：把文件放到缓存区。

* 接口：Node\* findPage(USRT FileID, USRT PageID);

功能：从缓存区裡寻找文件。

* 接口：Node\* readPage(USRT FileID, USRT pageID);

功能：读取文件到缓存区。

* 接口：bool insertData(SysObject\* table, char record[], int index);

功能：实现在缓存区裡插入纪录。

* 接口：bool deleteData(SysObject\* table, USRT pageid, USRT offset, int index);

功能：实现在缓存区裡删除纪录，实现了删除链表。

* 接口：void writePageToFile(USRT pageid, Page\* pagedata, string tablename);

功能：实现写入文件。

* 接口：bool readPageFromFile(USRT pageid, Page\* pagedata, string tablename);

功能：实现读取文件。

### 外部接口

* 接口：DBManager(string dbname);

功能：建立一个DBManager。

* 接口：~DBManager();

功能：删除DBmanager时必须把所有缓存区裡的页处理掉。

* 接口：void flush();

功能：把所有缓存区裡的页处理掉。

* 接口：void createDataBase(string databaseName);

功能：建立数据库。

* 接口：bool switchDataBase(string databaseName);

功能：切换数据库。

* 接口：void dropDataBase(string databaseName);

功能：删除数据库。

* 接口：void createTable(string tablename, UINT colNum, string colName[], BYTE

type[], USRT length[], bool nullable[]);

功能：用得到的参数建立一个表，更新相关的sysObject和sysColumn。

* 接口：void dropTable(string tableName);

功能：删除一个表

* 接口：bool insertRecord(RecordEntry \*input,string colName[], string

tableName);

功能：插入数据。

* 接口：bool updateRecord(string tableName,BYTE \*\*Value,string \*colName,BYTE \*type,BYTE \*len,BYTE \*op, BYTE condCnt);

功能：更新数据。

* 接口：bool deleteRecord(string tableName,BYTE \*\*Value,string \*colName,BYTE

\*type,BYTE \*len,BYTE \*op, BYTE condCnt);

功能：删除数据。

* 接口：vector<RecordEntry\*> findRecord(string tableName,BYTE \*\*Value,string \*colName, BYTE \*type, BYTE \*len,BYTE \*op, BYTE condCnt, string \*showColName, int showNum);

功能：寻找符合条件的数据。

* 接口：vector<RecordEntry\*> getFindRecord(string tableName,BYTE \*\*Value,string \*colName, BYTE \*type, BYTE \*len,BYTE \*op, BYTE condCnt);

功能：寻找符合条件的数据。

* 接口：void combine(vector<RecordEntry\*> &result, const vector<RecordEntry\*> &A, const vector<RecordEntry\*> &B, BYTE colA,BYTE colB,BYTE tarAIndex);

功能：结合两个表，寻找符合条件的数据。

* 接口：RecordEntry\* getRecord(string tableName, USRT offset, USRT pageid);

功能：寻找指定的数据。

* 接口：bool checkRecordAvaliable(string tableName, USRT offset, USRT pageid);

功能：检查数据是否已被删除。

* 接口：void printRecord(string tableName,BYTE colnum,string \*colName,USRT offset,USRT pageid);

功能：输出指定的数据。

* 接口：void printDatabase();

功能：输出数据库内所有表的属性。

* 接口：void printTable(string tableName);

功能：输出指定表的属性。

* 接口：void printTables();

功能：输出所有表的名字。

* 接口：vector<SysColumn\*> getTableAttr(string tableName);

功能：寻找一个表的所有列。

* 接口：SysColumn\* getTableColumn(string tableName, string columnName);

功能：给予表和列的名字，返回列的属性。

* 接口：bool checkTableColumn(string tableName, string columnName);

功能：检寻列是否存在表内。

## 系統管理模块

系统管理模块在文件systemManage.h，systemManage.cpp，dataUtility.h以及dataUtility.cpp中定义，包含处理sysObject和sysColumn文件和处理数据类型之间的转换。

### 内部接口

* 接口：TYPE\_ID getNewID();

功能：确保所有ID的独立性。

### 外部接口

* 接口：SysManager();

功能：建立system Manager。

* 接口： ~SysManager();

功能：删删system Manager时必须处理好缓存区的数据。

* 接口： void loadSysfile();

功能：从数据库文件sysFile中读取数据库的资讯。

* 接口： bool createTable(string name);

功能：创建一个表。

* 接口：SysObject \*findTable(string name);

功能：返回一个表的属性。

* 接口：string findTableName(TYPE\_ID tableID);

功能：通过ID寻找表的名字。

* 接口：bool dropTable(string name);

功能：删除表。

* 接口：bool insertColumn(string name,string tableName,BYTE xtype,TYPE\_OFFSET length,bool nullable,TYPE\_OFFSET index);

功能：从表中新增一个列。

* 接口：SysColumn \*findColumn(string name,string tablename);

功能：通过列和表的名字返回列的资讯。

* 接口：SysColumn \*findColumn(TYPE\_ID colID);

功能：通过ID返回列的资讯。

* 接口：TYPE\_OFFSET getRecordLength(string tableName);

功能：得到一个纪录的长度。

* 接口：void setName(string dbName);

功能：设定system Manager对应的数据库。

* 接口：vector<string> getAllTable();

功能：得到数据库内所有表的名字。

* 接口：void print();

功能：输出数据库内所有表的属性。

* 接口：void printTable(string tableName);

功能：输出指定表的属性。

* 接口：void printTables(string dbName);

功能：输出所有表的名字。

* 接口：vector<SysColumn\*> getTableAttr(string tablename);

功能：寻找一个表的所有列。

* 接口：void flush();

功能：清洗缓存区，并储存所有数据。

* 接口：void drop();

功能：清洗缓存区，放弃所有数据。

## 查询解析模块

查询模块在文件wq\_parser.h，wq\_parser.cpp中定义实现，正确解析语句之后调用DBManager类中的接口实现相关sql语句。

### 内部成员变量

* 变量：DBManager\* currentDb;

功能：用来调用DBmanager类下的各种执行函数

* 变量：vector<columnRequire> table1Require;

功能：第一个表的值需要满足条件的vector， attr op value

* 变量：vector<columnRequire> table2Require;

功能：第二个表的值需要满足条件的vector，attr op value

* 变量：vector<tableJoinRequire> twoTableJoinRequire;

功能：两表链接时，链接要求，table1.attr1 op table2.attr2

* 变量：vector<string> table1ShowColumn;

功能：select第一个表需要显示列的名字的vector

* 变量：vector<string> table2ShowColumn;

功能：select第二个表需要显示列的名字的vector

* 变量：string table1Name;

功能：两表链接时第一个表的名字

* 变量：string table2Name;

功能：两表链接时第二个表的名字

### 外部接口

* 接口：parser(string dbname);

功能：初始化DBmanager，数据库的名称

* 接口：~parser();

功能：释放DBManager类变量

* 接口：void testParse();

功能：提示用户输入sql执行文件，输入quit表示退出

* 接口：void BatchSqlInFile(char\* filename);

功能：处理每一个sql执行文件，逐行读取，遇到；表示一条语句结束，调用解析函数

* 接口：void splitStr(char\* str, vector<string>\* res);

功能：将输入的一行字符串，以空格分离，单引号之间的数据不处理

* 接口：bool parserOneCommand(vector<string> commands);

功能：解析每条语句的第一个关键词，进入该关键词对应的解析函数，比如delete，进入parserDelete函数进行该语句的解析

* 接口：bool parserCreate(vector<string> commands);

功能：解析create语句，database和table分类处理，简单验证语句的正确性。创建table没有将primary key记录下来。

* 接口：bool parserDrop(vector<string> commands);

功能：解析drop语句，database和table分类处理

* 接口：bool parserDesc(vector<string> commands)

功能：简单验证语句正确行，调用printTable接口

* 接口：bool parserDelete(vector<string> commands);

功能：解析delete语句，调用parserwhere函数解析delete条件

* 接口：bool parserUpdate(vector<string> commands);

功能：解析update语句，调用parserwhere函数解析update条件，调用parserset函数解析update的set部分，where和set的信息push到table1Require（set的op为SET）

* 接口：bool parserSelect(vector<string> commands);

功能：解析select语句，判断是一个表还是两个表，调用不同的函数

* 接口：bool parserOneTableSelect(vector<string> commands);

功能：解析从一个表select语句

* 接口：bool parserTwoTableSelect(vector<string> commands);

功能：解析从两个表select语句，链接表解析实现

* 接口：bool parserTwoTableWhere(vector<string> commands);

功能：链接表select语句的where解析，op右侧可能是表的属性

* 接口：bool parserShowTable(vector<string> commands);

功能：解析show tables语句

* 接口：bool parserCreateColumn(vector<string> columnInfo, tableColumn\* columnInfos);

功能：解析create table的各个列信息

* 接口：bool parserUse(vector<string> commands);

功能：解析use 语句，切换数据库

* 接口：bool parserInsert(vector<string> commands);

功能：解析insert数据，只支持insert整个columns的形式

* 接口：bool parserWhere(vector<string> commands, string tablename);

功能：解析where条件

* 接口：bool parserSet(vector<string> commands, string tablename);

功能：解析update的set部分

* 接口：bool checkNameAvaliable(string s);

功能：解析该字符串是否符合数据库中变量名的要求A-Za-z和’\_’

* 接口：bool isCmp(char c);

功能：判断该char是不是 > < =

* 接口：bool isEnglishAlphabet(char c);

功能：判断该char是不是A-Za-z

* 接口：bool isOpt(string s);

功能：判断该字符串是不是操作符，支持的操作符有= != > >= < <=

* 接口：bool isDig(char c);

功能：判断该char是不是数字

* 接口：int getType(string s);

功能：返回int或者varchar的全局定义数值

* 接口：int getOpt(string s);

功能：返回操作符的全局定义数值

* 接口：bool checkStingIsInt(string s);

功能：判断该字符串是不是数字

* 接口：string getKeyWords(int keyvalue);

功能：根据全局定义数值返回对应的字符串

* 接口bool checkKeyWord(string s, int keyvalue);

功能：判断该字符串的意义对应全局定义数据是不是keyvalue

* 接口：bool checkColumnsValue(vector<SysColumn\*> sysColumns, vector<string> datas);

功能：判断所有列对应的值是否符合数据类型以及长度的要求

* 接口bool checkOneColumnValue(SysColumn sysColumn, string data);

功能：判断一列对应的值是否符合数据类型以及长度的要求

## 索引模块

基于内存的B+tree索引，设计主要程序btree.h和btree.cpp

内部接口：

* 接口：insert\_into\_leaf(node<T>\*, T, record<T>\*);

功能：插入叶子节点

* 接口：insert\_into\_leaf\_after\_splitting(node<T>\*, T, record<T>\*);

功能：插入分离后叶子节点

* 接口：insert\_into\_node(node<T>\*, int, T key, node<T>\*);

功能：插入节点

* 接口：insert\_into\_node\_after\_splitting(node<T>\*, int, T, node<T>\*);

功能：插入分离后节点

* 接口：insert\_into\_parent(node<T>\*, T, node<T>\*);

功能：插入父节点

* 接口：get\_neighbor\_index(node<T>\* n);

功能：获取邻居节点的下标

* 接口：remove\_entry\_from\_node(node<T>\*, int, node<T>\*);

功能：移除节点

* 接口：adjust\_root();

功能：调整根节点

* 接口：coalesce\_nodes(node<T>\*, node<T>\*, int, T);

功能：合并节点

* 接口：redistribute\_nodes(node<T>\*, node<T>\*, int, int, T);

功能：节点重新分布

* 接口：delete\_entry(node<T>\*, T, void \*);

功能：删除

* 接口：record<T>\* find(T key);

功能：查找

外部接口：

* 接口：node<T>\* search(T e);

功能：查找节点

* 接口：bool insert(T, T);

功能：插入节点

* 接口：bool remove(T);

功能：移除节点

* 接口：void destroy\_tree\_nodes(node<T>\*);

功能：释放节点

* 接口：void enqueue(node<T>\*);

功能：将节点加入队列

* 接口：node<T> \* dequeue();

功能：移除队列

* 接口：int path\_to\_root(node<T>\*);

功能：节点到根节点的路径

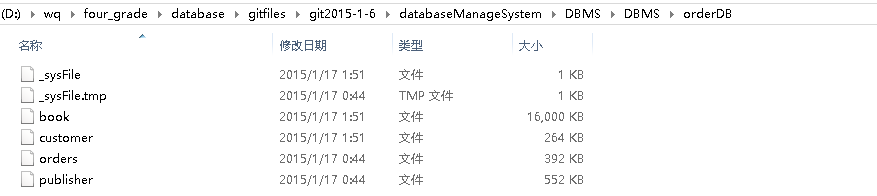
* 接口：void print\_tree();

功能：输出树的信息和结构

# 实验结果

## 文件存储

每一个数据库存储对应一个文件夹，每一张表的数据存储对应该文件夹中的一个文件，每个数据库的系统管理文件对应一个文件。



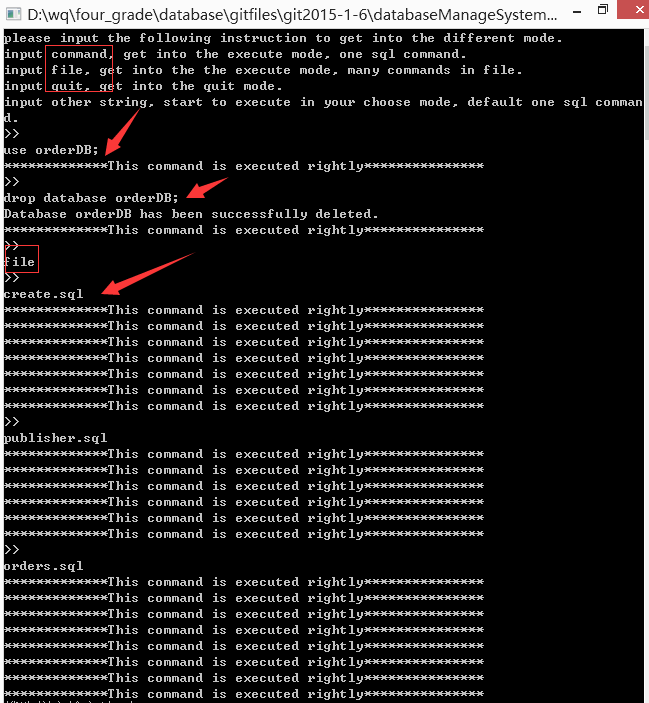
## 系统管理语句

使用VS2012运行DBMS工程的main程序，出现以下界面：

默认命令行指令输入模式，即你可以直接输入；结尾的sql语句，输入回车执行该语句。

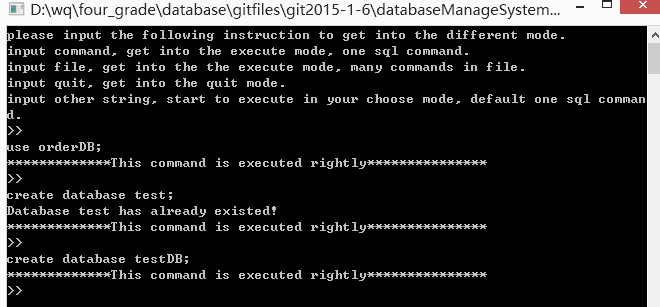
如果你需要执行存有sql语句的文件，需要先输入file，回车切换到文件指令输入模式，然后输入文件名，将会执行文件中的sql语句。

如果你想要退出当前程序请输入quit，回车即可退出。



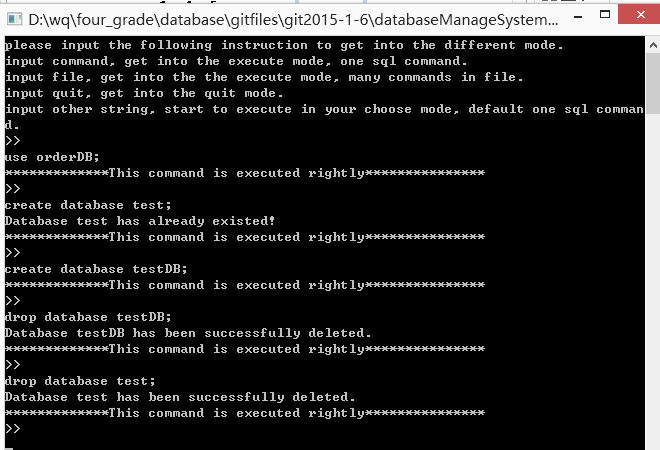
（默认的database为名为test的数据库，关键词的输入支持大小写混杂【比如：可以识别CreatE为CREATE关键词】）

### CREATE DATABASE *orderDB*; 创建名为orderDB的数据库。



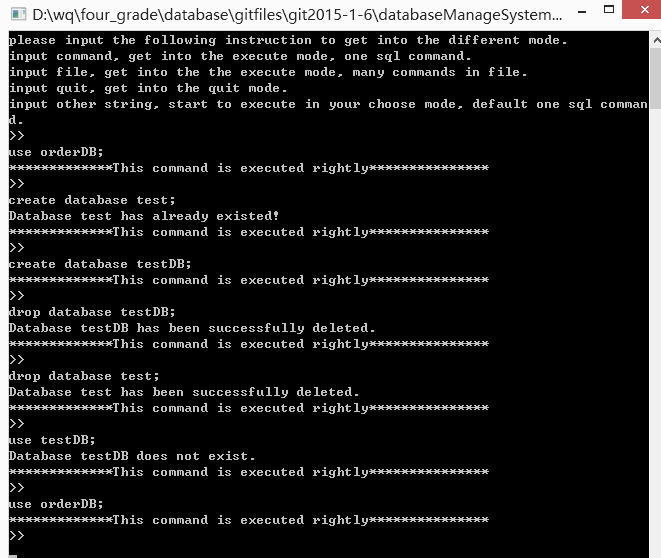


### DROP DATABASE orderDB; 删除名为orderDB的数据库。

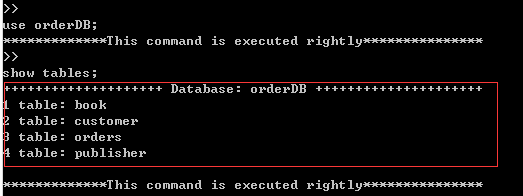




### USE orderDB; 当前数据库切换为orderDB。



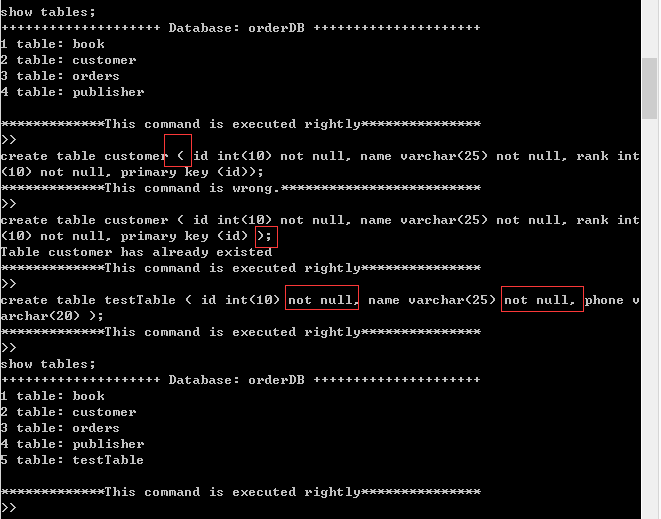
### SHOW TABLES; 列出当前数据库包含的所有表。



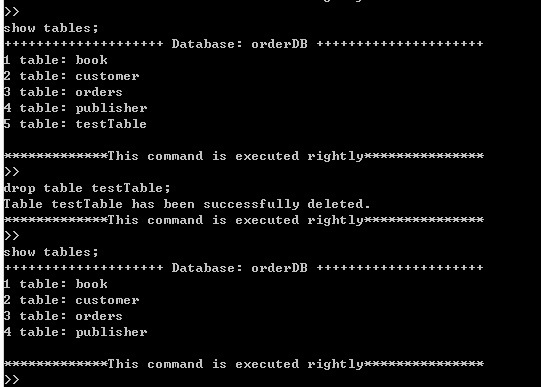
### CREATE TABLE customer

( id int(10) NOT NULL,name varchar(25) NOT NULL,rank int(10) NOT NULL,PRIMARY KEY(id) );创建名为customer的表，它包含三个字段id、name和rank，其中id是主键。这三个字段的数据类型分别为整型、字符串和整型，并且都不允许为空。

已经创建过了，那创建其他表验证正确。因为解析语句写的比较糙，要求create table 的 括号必须( 和):空格分离。

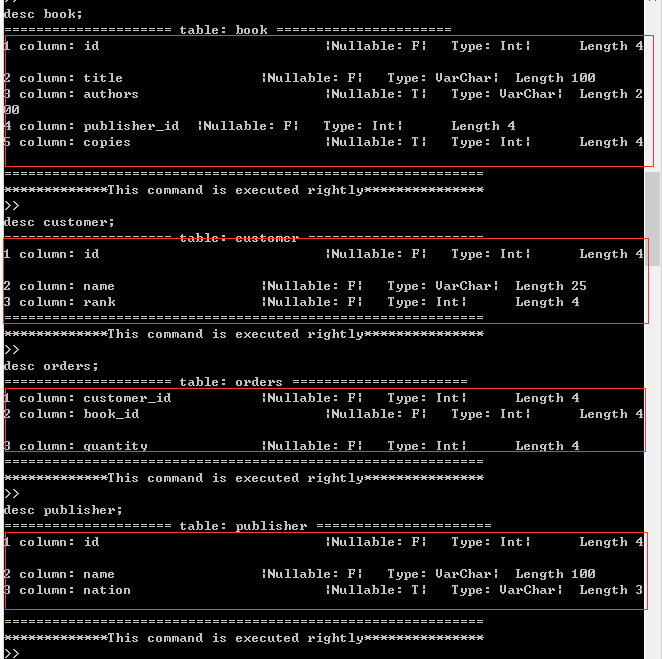


### DROP TABLE testTable;



### DESC TABLE customer;

显示不够友好，但是结果是对滴！



## 查询解析语句

### Insert 输入检测

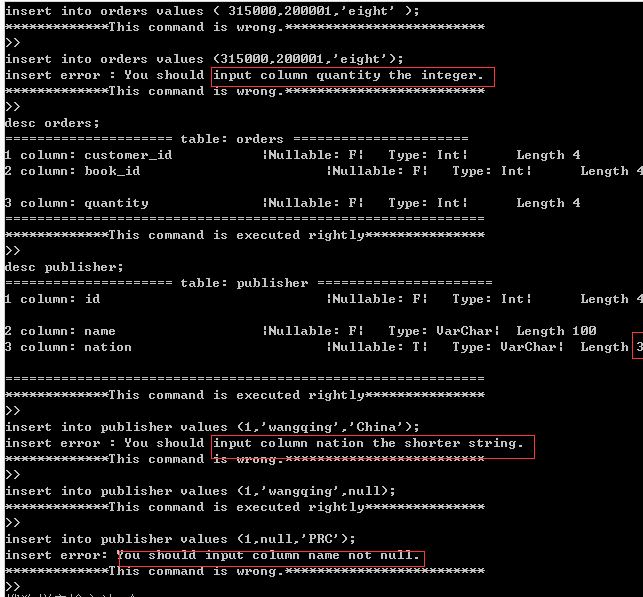
框中的语句，执行结果如下：

输入sql文件很多insert，执行成功！可以检测基本错误：

如果要求输入int，但是数据类型不是int，报错。

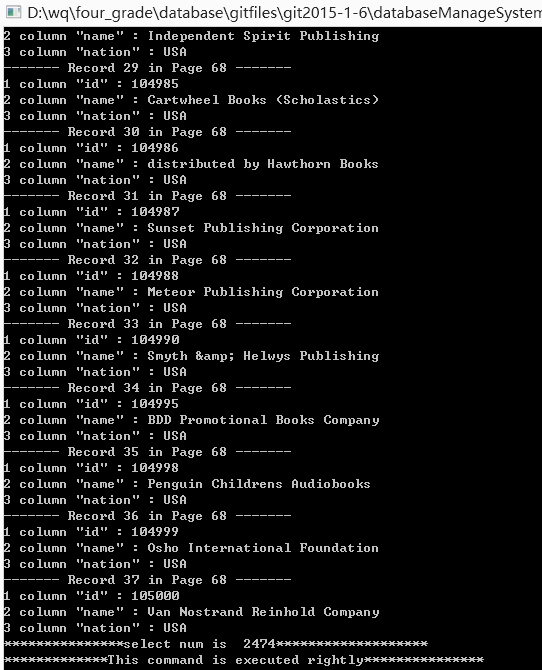
如果要求输入字符串，但是字符串的长度超出限制，报错。

如果不能是null，但是输入null，报错。【代码中只对varchar的null进行了处理，如果int输入null，解析语句错误。】

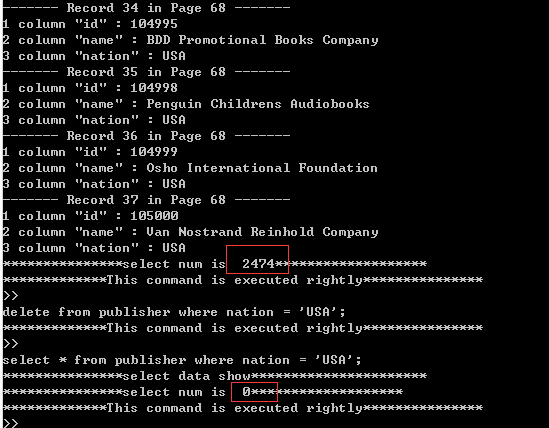


### Delete删除数据

执行select \* from publisher where nation = ‘USA’;

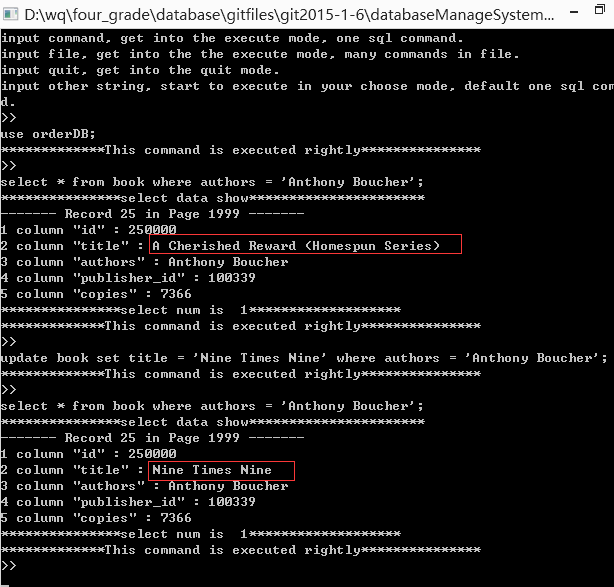


执行delete操作之后进行select，select不到，证明删除成功。



### Update更新数据

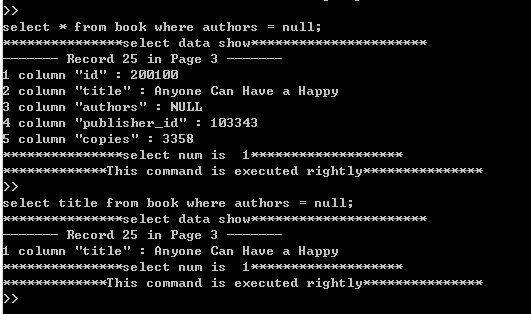
有图有真相，update成功。



### Select选择数据

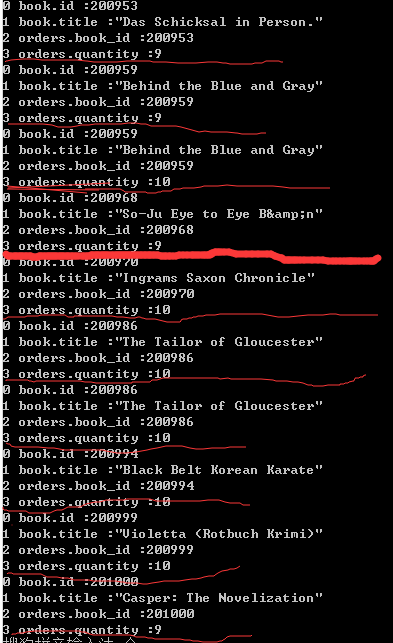
#### Select选择显示满足条件的部分语句

SELECT *title* FROM *book* WHERE *authors = null*; 因为没有区分=和is的区别，所以op只识别6种（= != > < >= <=）

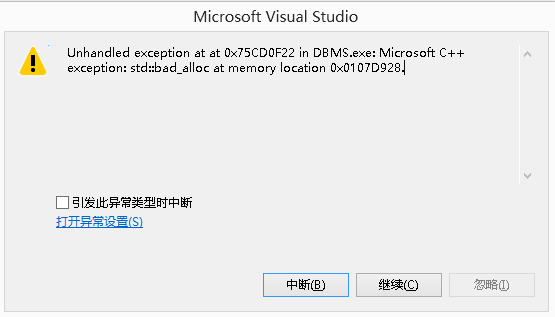


#### Select Join 数据

执行select book.id , book.title, orders.book\_id , orders.quantity from book , orders where book.id = orders.book\_id and orders.quantity > 8;



执行结果正确，确定的数目本来会在最后显示，但是在运行的过程中报错：



因为new不出来新的内存，所以出现bug。因为没有索引，所以联合检索的方法是暴力搜索将符合要求的表1中的m个记录和表2中的n个记录，存在内存，然后检索联合条件，将符合条件的找出。

# 小组分工

黄锦畦：系统管理、记录管理（删除数据）、文件系统

王庆：查询解析、记录管理（插入、更新、查找数据【两个表的简单链接】）

# 参考文献以及github链接

项目链接：<https://github.com/databasesystem/databaseManageSystem>

参考网址：<http://www.cplusplus.com/>

<http://www.w3cschool.cn/sql_update.html>

# 编程环境说明

Visual Studio 2012编辑器

使用C++和C语言编写

支持Windows平台，不能实现跨平台

编译选择：Debug模式，因为Release模式因为VS2012的问题，会认为很多函数的调用不安全，不能正常编译！