**对于本组代码的一些说明**

**一、结构：**

我们利用三个类与一个映射来组织整个数据库的结构。Database类代表包含若干表格的一个数据库，Table类表示数据库中的一个表格，Column类表示表格中的某一列。

在main.cpp中，我们用一个名为database的map来管理现有的数据库，利用map的特性可以用数据库名访问数据库的指针，同时利用指针now可以很方便地在各数据库之间切换。

Database类中的向量dvalue，则包含了一个数据库中的所有表格的指针，在建立表格时我们已经根据表格名的字典序给他们进行了排序，同时重载了[]以便于用Database对象+[表格名]访问表格指针。

Table类中的向量tvalue，则包含了表格中每一列的指针，它们按照插入顺序从前至后排序。同样地，利用Table对象+[列名]可以访问列指针。另外，对于表格中每一行的数据，我们根据primary\_key列元素的大小（或字典序）从小到大进行了排序。

Column类中的向量cvalue包含了该列所有的元素，这里所有数据类型均使用string存储，可以使用Column对象+[行数]访问该列某一行的元素。

而主函数则是用getline读入当前命令，判断命令类型，然后调用相应的模块执行命令（命令接受需求文件中的形式，接受多余换行，接受行末、WHERE前面及逗号后面的多余空格，不接受其他情况的多余空格，涉及到命令输入错误（字符数不变）、输入的数据库、表格或列名不存在及NOT\_NULL元素未填充的情况会报错）。代码中有很多“魔数”，均为截取子串所需，请谅解。

关于代码的具体实现，请阅读cpp中的注释。若出现注释因IDE不同而乱码的问题，请阅读“代码副本文件夹”中的相应txt文件。

**二、封装与接口：**

**2.1 Database类：**

database.h中对类的定义如下（函数的具体功能和实现均在cpp文件中）：

class Database {

string dname; //数据库名

vector<Table\*> dvalue; //包含数据库所有表格指针的向量

public:

Database(string a); //初始化列表，字符串a为数据库名

~Database(); //析构函数，删除dvalue中的指针释放内存

Table\* operator[] (const string& a); //[]的重载，用表格名访问表格指针

string getname() const; //获得数据库名的接口

int getsize(); //获得数据库里表格数的接口

void create(const string& a,const string& b); //在库里创建表格的函数，a表示表格名，b为表格的主键

void show(); //展示数据库所包含的表格

bool find\_table(const string& a); //传入表格名判断库里是否有该表

void del(const string& a); //通过表格名删除表格

};

**2.2 Table类：**

table.h对类的定义如下（函数的具体功能和实现均在cpp文件中）：

class Table {

string tname; //表格名

string primary\_key; //表格主键名

vector<Column\*> tvalue; //包含表格所有列指针的向量

public:

Table(string a,string b); //构造函数，a为表名，b为主键名

~Table(); //析构函数，释放tvalue内指针的内存

string getname() const; //获得表格名的接口

string getprime () const; //获得表格主键名的接口

int getsize(); //获得表格列数的接口

friend class Database; //方便Database类访问Table数据

Column\* operator[] (const string& a); //重载[]，便于以列名访问列的指针

void create(const string& a,const string& b,bool c); //在表格中添加列，a为列名，b为列变量的类型名，c表示该列元素是否可为NULL

void show\_all(const vector<bool>& check); //展示整个表格数据，向量vector由whereClauses生成，表示哪些列符合条件，下同

void show\_one(const string& cname,const vector<bool>& check); //给出列名展示某些列的符合条件的数据

void show\_column(); //展示制表信息

bool find\_column(const string& a); //给定列名判断一个列是否在表中

bool null\_check(const string& a); //在插入操作中，给定一个列名，在该列为Not Null时，判断它是否被插入，否则报错

void default\_fill(); //在插入操作中，对没有插入数据的列用缺省值NULL填充

void del\_row(const vector<bool>& check); //删除符合条件的行

void swap\_row(int a,int b); //给行数交换某两行（行数从0开始）

void sort\_prime(); //根据主键从大到小给表格的行排序

void update\_row(string cname,string value,const vector<bool>& check); //修改表中数据，cname表示列名，value表示要填入的值

vector<bool> whereClauses(const string& str); //把一个wherecluause处理为bool向量，其中check[i]为true表示该行符合条件，反之同理

bool whereclauses\_work(const int& i, const string& str); //为实现whereClauses函数定义的辅助函数，详见table.cpp

};

**2.3 Column类：**

column.h中对类的定义如下（函数的具体功能和实现均在cpp文件中）：

class Column {

string cname; //列名

string type; //列存储的变量类型名

bool can\_be\_null; //表示该列数据是否可为空

vector<string> cvalue; //包含列所有行数据的向量

public:

Column(string a,string b,bool c); //构造函数，a为列名，b为列存储的变量类型名，c表示该列数据是否可为NULL

~Column(); //析构函数

string getname() const; //获得列名的接口

string gettype() const; //获得列存储变量类型的接口

bool can\_null() const; //获得该列元素是否可为NULL的接口

int getsize(); //获得该列元素个数，即表格行数的接口

void insert(const string& a); //在列中插入元素，a为待插入的值

void del(int s); //删除列中元素，s为行数

void update(int s,const string& a); //修改列中元素，s为行数，a为目标值

friend class Table; //便于Table访问Column数据

string& operator[] (int s); //重载[]，便于用Column对象+[行数]访问相应位置的元素

};

**三、关于whereClauses：**

在使用whereclauses时，本组先将WHERE之后的字符串截取出来，然后对本table中所有行进行枚举处理，判断其是否符合条件，最终返回一个包含所有行的vector<bool>，记录每一行的结果是true还是false。对单行进行处理时，会把需要处理的字符串按逻辑运算符打散(先OR后AND)成前后两部分然后对每一部分递归处理两段子串，再将子串的计算结果采用相应的逻辑运算符连接以得到结果。当处理的字符串不可继续打散递归时，则寻找比较运算符，并对其前后的部分进行处理，寻找列名的关键字，找到对应列，将列的类型代入，并进行比较得到结果。最后组合所有结果，就可以得到单行的结果。对于NULL，考虑其特性(仅在NULL or true时为1，其余时结果均为NULL)，将其视作默认false的情况考虑。