数据库 第二阶段 架构说明

一.Table类

第二阶段的基本要求包括读写文件以及select中功能的增加。在保留原来代码的接口的同时，我们将Load、select操作放入Table类内，接受语句解析后的成分（例如列名等）作为参数，统一处理。

1.文件读取

```
void Table::LoadFile(std::string filename, std::vector<std::string> attrname)
```

接受文件名和列名作为参数，调用原有的 insert 接口储存数据。

2.select相关函数

```
void Table::SelectData(const std::vector<std::string>& attrName,//列名
    const std::vector<std::string>& attrNewName,//select ... as 新的列名
    int countpos, //count所在的位置
    const std::string& countAttr, //count后面的列名
    const std::vector<std::string>& groupby, //group后面的列名
    const std::string& orderbyAttr, //order后面的列名
    const std::string& orderbyCount, //order by count 后面的列名
    const std::string& Where, //where字句
    const std::string& filename)//输出文件名
```

（1）参数说明

```
SELECT student_num AS name, COUNT(score_id)
FROM score
INTO OUTFILE 'out'
WHERE score = 90
GROUP BY student_num
ORDER BY COUNT(score_id);
```

此时 attrName 中 储存 stu_name 和 COUNT(score_id),

attrNewName 中储存 name 和 COUNT(score_id),

countpos 的值为 1， countAttr 中储存 score_id ,

groupby 储存 student_num,

orderbyAttr 为空，orderbyCount 储存 score_id,

Where 中 储存 score = 90， filename 中储存 out。

（若语句为ORDER BY score_id，则orderbyAttr 中储存 score_id，orderbyCount 为空。）

（2）实现细节

为了使程序每部分承担功能不致过于繁杂，实现时将 count、 group by、order by 分离出来。

```
std::map<Data, int> Table::Count(std::string name)
```

Count 接受列名作为参数，返回 map<主键, 次数>。

```
void Table::Group(std::vector<Data>& SelectResult,
                  std::vector<std::string> groupby,
                  std::map<Data, int>& countResult,
                  std::string orderbyCount)
```

Group 接受筛选出的结果 SelectResult、分组的依据 groupby 作为参数。

同一组的数据只留下一个，其于删除；并记录该组原来有几行数据。

由于删除了一些数据，所以若 order by 子句中有 count，同样在这里处理。

countResult 是之前计数结果，在这里更新。orderbyCount 是排序的依据。

```
void Table::OrderAttr(std::vector<Data>& SelectResult, std::string orderbyAttr)
```

如果order by 后面没有 count 语句，则为列名，由此函数处理。

SelectData实现

- select后面支持 sin(stu_id*3+1) 之类的运算式，因此，首先调用 ALU 中的接口判断，并进行运算，将表达式以及运算结果作为临时的一列数据加入表中。
- 接下来调用原来的接口处理where语句，并将结果转存入SelectResult。
- 调用 count运算。
- 调用 group by 分组。
- 调用 order by 排序。
- 如果 filename 不为空则把输出定向到文件。
- 输出表头、内容。
- 关闭文件，删除临时列。

二、网络连接模块

本模块调用了socket中的相关模块实现局域网网络连接。通过宏定义判断操作系统，控制包含的头文件，并分别调用了windows和mac os平台上相关的API，使模块能够跨平台运行。

开始运行后，服务端将显示 ip 地址与端口号，客户端输入相应 ip 与端口号后连接。由客户端输入命令，服务端会将执行结果发送至客户端。

三、Command类

Command类负责对输入的语句进行解析，并调用底层数据结构类的相关接口进行操作。本类只处理输入，不负责输出，输出完全交由底层数据结构类处理。

整个类的框架与前一个小组第一阶段的代码基本相同。新的版本使用了正则表达式，大大提高了模块的鲁棒性；增加了一些新的函数以利于实现新功能，并改进了原有框架的一些不合理之处。

数据成员

buffer字符串，存储一行语句；

函数成员

构造对象时传入一行语句，初始化buffer.

FormatSQL对buffer进行预处理。调用正则表达式库，去除用户可能的非法输入，保证“, ( )”等运算符左右不存在空格。

举例说明FormatSQL处理前后读入语句的变化：

```
" INSERT INTO oop_info( stu_id, stu_name) VALUES (2018011343, "a") ;123"
    //处理后
"INSERT INTO oop_info(stu_id,stu_name)VALUES(2018011343,a)"
```

operate调用FormatSQL对读入语句进行预处理，读取语句的第一个单词调用对应函数。

Create，Drop，Select等11个函数对Statement处理后的语句进行进一步分析，调用其他类的相应接口。除了Select函数外，其它函数的实现与第一阶段大致相同。

此外，定义了4个用于字符串处理的函数.

toUpper函数将字符串中的小写字母转换成大写；

trim函数去除字符串头尾的空格；

split函数将字符串按照某一分隔符分解为一个字符串数组,并返回此字符串数组。 举例说明:

```
/*参数类型均为const& string*/
split(str="stu_id,stu_name,2018011343,a",sep=",");
/*vector<string>*/
return {"stu_id","stu_name","2018011343","a"};
```

getFirstSubstr函数将字符串按照某一分隔符分解成两部分, 并返回前一部分, 输入的字符串存储后一部分。举例说明:

```
/*前一个参数为string& 后一个为const char*/
getFirstSubstr(buffer="insert into",sep=" ");
/*buffer="into"*/
return "insert";
```

四、ALU模块

ALU类完成了算术、逻辑、比较表达式和数字函数的计算。

数据成员

string expression 存储一个表达式；

map<string,int> priority 储存运算符的优先级;

set<string> function 储存所有函数的名称;

regex operators 供IsALU调用，用正则表达式判断一个字符串是不是算式。

函数成员

构造对象时传入表达式，初始化expression.

由数据结构类调用IsALU函数判断某一特定字符串是不是表达式，再调用process函数进行计算。process函数返回一个储存了计算结果的vector<string>.

若select语句中不含"from tablename"，则由DatabaseMap类调用process()；否则,由相应的Table类调用process(table*,vector<Data>), 传入Table的指针和需要计算行的主键。

```
select 1+2,2+3;#由DatabaseMap两次调用ALU类：expression=1+2,expression=2+3
select Sin(id) from oop_info where id>1;
#expression=Sin(id),process(*table,Data).
```

ALUformat调用正则表达式对表达式进行处理使之规范化，以利于后续计算。

Transfer输入的原始中缀表达式转换为后缀表达式。

Calculate对后缀表达式进行计算，返回运算结果(至多保留六位小数)。若计算结果非法（如除以0），将返回NULL。

```
//原始字符串
1+2sin(pi ( )+2 )
//ALUformat处理后
1 + 2 Sin( pi( ) + id )
//Tranfer处理后（调用split按空格对字符串进行了拆分）
{1, 2, pi(,id,+,Sin(,+}
//Calculate返回到计算结果，假设id=2.
"1.090703"
```

DoubleToString函数将一个double类型的数据转换成字符串，并保留至多六位小数。

IsDouble函数用于判断一个字符串是否是浮点数。

split函数和toUpper函数与Command类的相同，定义在command.cpp中。

五、存档模块

存储方式

DatabaseMap层面：

用名为***DatabseNames***的文件存储一个整数N，表示数据库的总数；存储N个字符串，表示数据库的名称。

Database层面：

用数据库的名称命名一个文件，存储一个整数N，表示该数据库所含表格的数目；存储N个字符串，表示表格名称。

Table层面：

用表格的名称命名一个文件，存储两个整数Col，Row，依次表示表格属性的数目(列数)和表格数据的条数(行数)；

接下来存储Col行，每一行**4**个参数，依次为**属性名称、数据类型、是否非空、是否主键**，是与否用**y/n**来表示；

接下来存储表格的数据，每一行数据都**按照属性的名称在*map*中的排序存储。**

实现原理

首先创建名称为***DatsbaseNames***的文件，访问类***DatabaseMap***中的Map容器***dbs***，向文件里写入***dbs.size()***，接着通过迭代器向文件写入数据库的名称，每写入一个数据库的名称，都**以这个数据库的名称创建一个文件**，然后访问这个数据库存储表格的Map容器***Table_list***，向该文件写入***Table_list.size()***，接着迭代写入每个表格的名称，每写入一个表格名称，都**以这个表格的名称创建一个文件**，然后访问这个表格，类似上面获取数据库信息的方法按照**part1中的**存储顺序写入表格信息。

读档或加载的时候，**按照存档的顺序逆向进行**，并注意表格存储数据的读入要与属性(列)正确对应。

删除操作对已创建文件的影响

这个影响主要在于：删除表格的时候，以表格名称命名的文件也应该删除，这时用***remove（const char*）***来删除对应文件；**删除某个数据库**的时候，应该先删除这个数据库包含的表格，这时访问数据库存储表格的容器，利用表格名称删除对应文件，最后删除数据库对应的文件。

每一次操作后，都更新一次文档信息。