**Interface And API 说明文档**

作者：软工1601 范佳臣 3160102299

# 一、主要功能

## 1.1 Interface主要功能

Interface负责用户使用界面：

1. 为用户提供使用界面（登录界面及语句输入界面）；
2. 对输入语句的编译；
3. 输出执行语句结果（报错或正确执行结果）。

## 1.2 API主要功能

API负责在执行语句成功编译后，使用接口调用各模块：

1. 在编译完成后调用相应模块；
2. 获取各模块执行模块，返回信息供interface输出告知用户。

# 二、模块总体设计思路

我们将user的账户和密码作为常量储存在内存中，只有在输入账户以及密码完全与预匹配的的状况下，可进入interface的功能模块进行sql语句的输入执行操作。

语句可单行或多行输入，输入的换行符和制表符会自动转换为空格，在语句读入的检测中，若判断已输入’;’的状况下，我们将其视为用户完整输入一条指令，并对其进行编译。我们根据保留字以及空格对输入的语句进行分段，并最终逐段进行判定，将正确语句的相应信息存入对应的类中，最终将编译抽象出的语句中的信息发送给相应的模块进行调用。以此来实现interface与其他各模块之间的联系。

# 三、具体实现

## 3.1 数据结构定义

a)表模式信息结构体(class Table)

**class** **Table**

{

**public**:

string table\_name; *// name of table*

int column\_num; *// the number of table's attributes*

string column\_name[COLUMNMAXSIZE]; *// string array to store the name of data*

ColumnType column\_type[COLUMNMAXSIZE]; *// ColumnType array to store the type of data*

int string\_length[COLUMNMAXSIZE]; *// int array to store the max length of the char*

bool is\_unique[COLUMNMAXSIZE]; *// bool array to store the check of attributes*

string primary\_key; *// the the name of the primary key*

int exist\_index\_num;

string all\_index\_name[COLUMNMAXSIZE];

};

b)通过class Table派生出的各种类，传输各类sql语句的编译信息。

1）创建表语句结构体(class CreateTable)

**class** **CreateTable** :**public** Table

{

**public**:

string primary\_key; *// the the name of the primary key*

};

2)删除表语句结构体(class DropTable)

**class** **DropTable**:**public** Table

{

};

3)建立索引语句结构体(class CreateIndex)

**class** **CreateIndex** :**public** Table

{

**public**:

string index\_name; *// the name of index we need to create*

string index\_column\_name; *// the name of attributes used to create index*

};

4)删除索引语句结构体(class DropIndex)

**class** **DropIndex** :**public** Table

{

**public**:

string index\_name; *// the name of index we need to drop*

};

5)插入信息语句结构体(class Insert)

**class** **Insert** :**public** Table

{

**public**:

string row\_value[COLUMNMAXSIZE]; *// string array to store the date*

int value\_num; *// the number of the insert data*

};

6)删除元祖语句结构体(class Delete)

**class** **Delete** :**public** Table

{

**public**:

int condition\_num; *// the num of the condition*

vector<string> condition\_name; *// the name of attributes*

vector<string> condition\_op; *// the operator = <> < > <= >=*

vector<string> condition\_value; *// the value need to compare*

};

7)挑选元组语句结构体(class Select)

**class** **Select** :**public** Table

{

**public**:

int condition\_num; *// the num of the condition*

vector<string> condition\_name; *// the name of attributes*

vector<string> condition\_op; *// the operator = <> < > <= >=*

vector<string> condition\_value; *// the value need to compare*

};

## 3.2 主要函数及功能描述

a) 登录界面实现函数

void logInterface()

{

bool isLog = false;

string user\_name, user\_password;

**while** (!isLog)

{

cout << "Enter the user name: ";

getline(cin, user\_name);

cout << "Enter the password: ";

getline(cin, user\_password);

**if** (user\_name == root\_name && user\_password == root\_password)

{

isLog = true;

}

**else**

{

cout << "Wrong user name or password" << endl;

}

}

cout << "Welcome to the MiniSQL. All commands end with ';'" << endl;

}

具体实现：

分别读取两行数据user\_name, user\_password,再分别将这两个变量中的数据与我们储存的root账户信息相匹配，成功则进入正常执行界面，不成功则清空两数据储存信息，并再次读取。

b)功能界面实现函数

void execInterface()

{

bool isQuit = false;

string temp;

**const** string space(" ");

**while** (!isQuit)

{

**try**

{

string sql\_string;

cout << "**\n**MiniSQL> ";

getline(cin, sql\_string);

**while** (!isSqlEnd(sql\_string))

{

cout << " -> ";

getline(cin, temp);

sql\_string += space;

sql\_string += temp;

}

**const** int operation\_type = sqlConvert(sql\_string);

**if** (operation\_type == 60)

{

isQuit = true;

cout << "You quit the MiniSQL." << endl;

}

}

**catch** (...)

{

cout << "Some thing was wrong. Windows is looking for solutions...";

}

}

}

具体实现:

不断读取每行输入语句信息，并将其中的换行符制表符替换为空格，最后当读取到用户输入了‘;’则判定为语句输入结束执行编译模块。

c)具体语句的编译模块（因代码量较大，只进行文字说明）

*//对输入sql语句的翻译模块*

int sqlConvert(string &sql);

*//输入首个保留字为create的sql语句的翻译模块*

int createConvert(string &sql);

*//对create index语句的翻译模块*

int createIndexConvert(string &sql);

*//对create table语句的翻译模块*

int createTableConvert(string &sql);

*//create table中输入属性的翻译模块*

int createTableConditionConvert(string &sql, CreateTable &CT);

*//create table中输入主键的翻译模块*

int createTablePrimaryConvert(string &sql, CreateTable &CT);

*//输入首个保留字为delete的sql语句的翻译模块*

int deleteConvert(string &sql);

*//delete中输入条件的翻译模块*

int deleteConditionConvert(string &sql, Delete &D);

*//输入首个保留字为drop的sql语句的翻译模块*

int dropConvert(string &sql);

*//对drop table语句的翻译模块*

int dropTableConvert(string &sql);

*//对drop index语句的翻译模块*

int dropIndexConvert(string &sql);

*//对insert语句的翻译模块*

int insertConvert(string &sql);

*//insert中输入插入值的翻译模块*

int insertConditionConvert(string &sql, Insert &I);

*//对select语句的翻译模块*

int selectConvert(string &sql);

*//select中输入条件的翻译模块*

int selectConditionConvert(string &sql, Select &S);

具体实现：

根据空格或者各种保留字对输入的整个语句字符串进行分段匹配，再分别根据不同的语法定义，分别对不同的语句抽象出应保留的信息储存在对应的数据结构中，在每次单个语句成功编译完成之后，调用API接口根据不同的语句传输不同的数据结构。

## 3.3 API接口调用

*// 从catalog management中获取表的所有信息，table name已实现储存在输入中*

bool getTableInfo(Table &);

*// 执行create table，最后catalog management返回格式化结果输出字符串*

string catalogExecute(CreateTable &);

*// 执行drop table，最后catalog management返回格式化结果输出字符串*

string catalogExecute(**const** DropTable&);

bool recordExecute(DropTable &);

bool indexExecute(DropTable &);

*// 执行create index，最后index management返回格式化结果输出字符串*

bool catalogExecute(CreateIndex &);

string indexExecute(CreateIndex &);

*// 执行drop index，最后index management返回格式化结果输出字符串*

bool catalogExecute(DropIndex &);

string indexExecute(DropIndex &);

*// 执行insert，最后record management返回格式化结果输出字符串*

bool catalogExecute(Insert &);

string recordExecute(Insert &);

*// 执行delete，最后record management返回格式化结果输出字符串*

bool catalogExecute(Delete &);

string recordExecute(Delete &);

*// 执行select，最后record management返回格式化结果输出字符串*

bool catalogExecute(Select &);

string recordExecute(Select &);

在成功完成编译后，根据执行语句的类型调用不同的api接口，最后由返回string类型接口返回输出结果，告知用户执行结果。