编译原理上机报告

《DBMS 的设计与实现》

学号： 20009200601　　姓名： 杜新鹏

手机： 15666141760　邮箱： 1363616727@qq.com

阿里云登录账号：

完成时间：　2023 年 6 月 10 日

目　　录

1. 项目概况

1.1 基本目标

根据 Lex 和 Yacc 设计并实现一个 DBMS 原型系统，可以接受基本的 SQL 语句，对其进行词法分析、语法分析，然后解释执行 SQL 语句，完成对数据库文件的相应操作，实现 DBMS 的基本功能。

1.2 完成情况

基本实现了测试用例文件中的所有语句的功能。

实现的语句（实现了输入为大小写不敏感）：

1. 数据库的创建、显示和删除

CREATE DATABASE

SHOW DATABASES

DROP DATABASE

USE DATABASE

2. 数据表的创建、显示和删除

CREATE TABLE

SHOW TABLES

DROP TABLE

3. SQL 插入指令（指定列或按顺序）

INSERT INTO VALUES

4. 数据库单表查询 （WHERE 复合条件判断、选定指定列）

SELECT SNAME,SAGE FROM STUDENT WHERE SAGE=21;

5. 数据库多表查询

SELECT fields_star FROM tables

6. 记录的删除

DELETE FROM table WHERE conditions

7. 记录指定字段更新

UPDATE table SET updates WHERE conditions

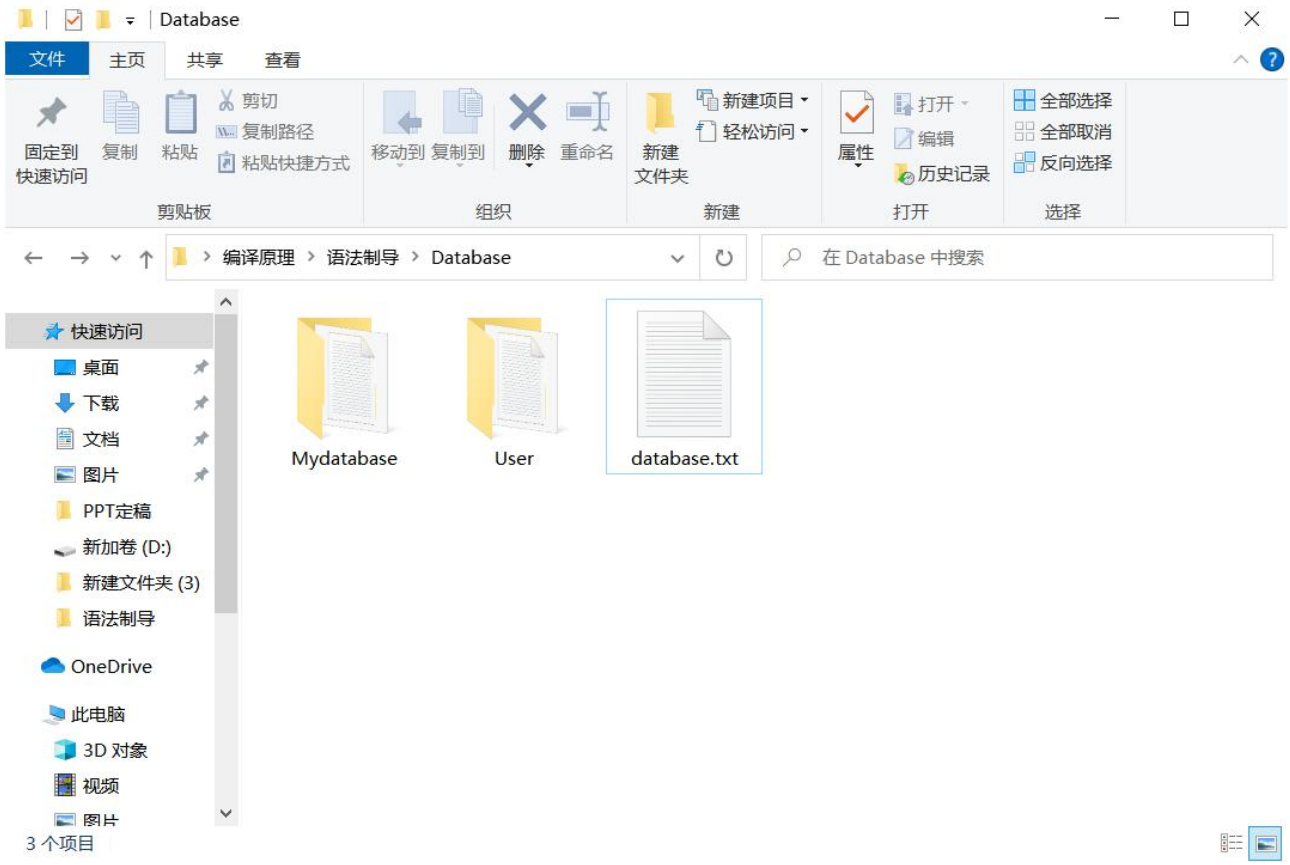
2.项目实现方案

2.1 逻辑结构与物理结构

为了要实现增加的 SQL 语句，增加了哪些元数据（例如，如何保存视图定义、安全权限、索引数据），元数据的逻辑结构是什么样的（可用表格来说明），数据库的物理结构是什么样的（整体数据库文件、页、内部组成等等，不单单是自己系统涉及到的一部分数据库结构），需要用图形来说明物理结构，并且包含数据保存实例。

由于时间以及个人水平的限制，我并没有采用更加贴近商业上真实情况的数据库实现，而是使用文件系统来记录数据库。

当创建了一个新的数据库后，程序会在指定 Database 文件夹中创建一个 database.txt 文本文件，该文本文件用于记录创建了哪些数据库。当执行 Create database 语句时，程序会在 Database 文件夹中创建对应的数据库文件夹，如下图所示：



数据库所对应的文件夹中也有一个文本文档，可以用来记录所创建的表的逻辑结构，例如：

用户元数据的逻辑结构

表名	列 1 及其类型	列 2 及其类型	...
User	Name Char(20)	No INT	...
Student	Name Char(20)	Sno INT	...

…	…	…	…

如果在数据库中建立了表，那么表的数据结构与数据库中的 txt 文件夹相类似，但是只需要将其值按照数据库 txt 文件中所记录的表的格式列一一对应即可。如下图所示：

表名	**列 1 及其类型**	**列 2 及其类型**	**…**
User	Li Ping	125	…
User	Wang Li	56	…
…	…	…	…

Student	Lu Ban	2003	…
Student	Su Kang	2009	…
…	…	…	

所创建的表的内容放在数据库中与表同名的 txt 文件中。

该结构的优点：实现起来相对很简单，同时便于编码的单元测试，输入输出皆为直接可读和可写的文件，便于验证代码的正确性。

该结构的缺点：结构较为简陋，数据不够安全，很容易造成数据泄漏，不适用于实际的应用；该实现的存储结构以行为单位，对于存储空间是一种巨大的浪费；由于对文件的读写操作效率通常比较低，因此该结构的具体效率是比较低的。

2.2 语法结构与数据结构

逐条一一说明增加 SQL 语句的语法结构与数据结构。语法结构用产生式说

明，其中非终结符的属性用结构体表示，所用数据结构需要说明，并且最后用图形说明整个 SQL 语句的数据结构。

如下所示：

（1）CREATE 语句的产生式语法结构：

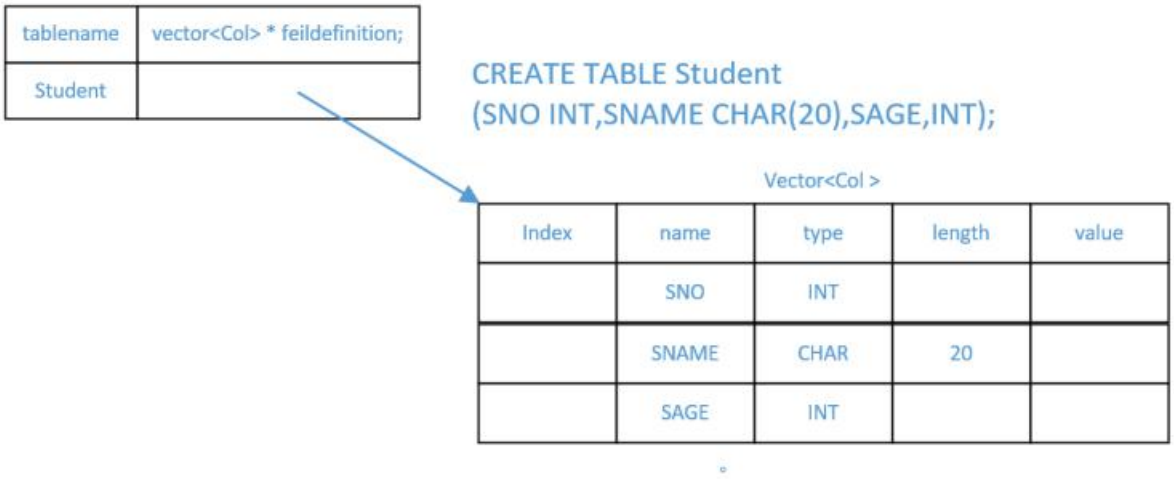
createsql: CREATE TABLE table '(' fieldsdefinition ')' ';'

非终结符 fieldsdefinition 的数据结构：

```
struct table_field_node {
    char * field_name;
    enum type {
        INT, STRING
    }type;
    int len;
    int offset;
    table_field_node * next = nullptr;
};
```

例句：CREATE TABLE Student(SNO INT,SNAME CHAR(20),SAGE,INT);

数据结构如下图：



（2）INSERT 语句的产生式语法结构：

insertsql: INSERT INTO table VALUES '(' values ')' ';'

| INSERT INTO table '(' fields ')' VALUES '(' values ')' ';'

非终结符 insertsql 的数据结构：

```
struct insert_node {
    table_node * table = nullptr;
    table_field_node * field = nullptr;
    values_node * values = nullptr;
};
```

非终结符 fields 的数据结构

```
struct table_field_node {
    char * field_name;
    enum type {
        INT, STRING
```

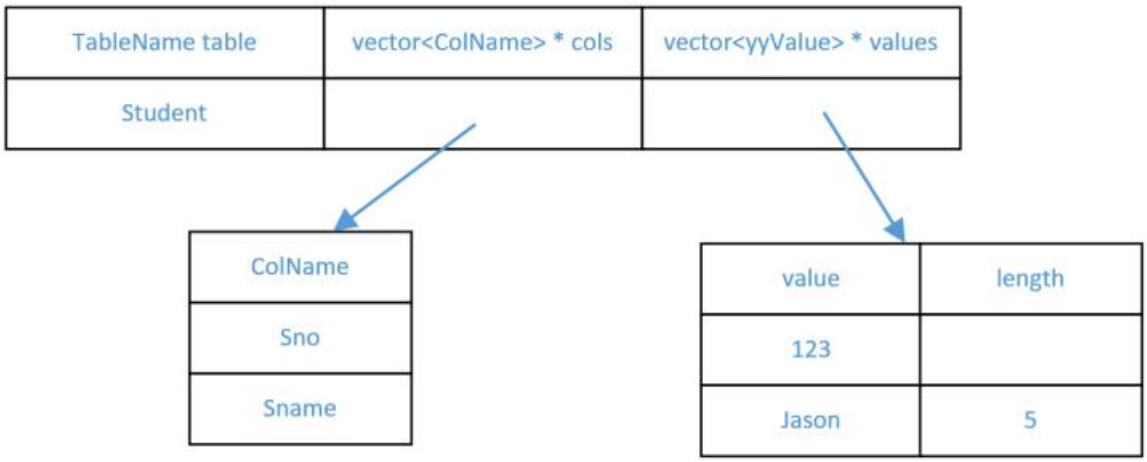
```
    }type;
    int len;
    int offset;
    table_field_node * next = nullptr;
};
```

非终结符 values 的数据结构：

```
struct values_node {
    enum type {
        INT, STRING
    }type;
    int intval;
    char * chval;
    values_node * next = nullptr;
};
```

例句：INSERT INTO STUDENT (Sno,Sname) VALUES (123,”Jason”);

数据结构如下图：



（3）SELECT 语句的产生式语法结构：

selectsql: SELECT fields_star FROM tables ';'

| SELECT fields_star FROM tables WHERE conditions ';'

;

非终结符 fields_star 的产生式语法结构：

fields_star: table_fields { $$ = $1;}

| '*'

;

非终结符 fields_star 的数据结构：

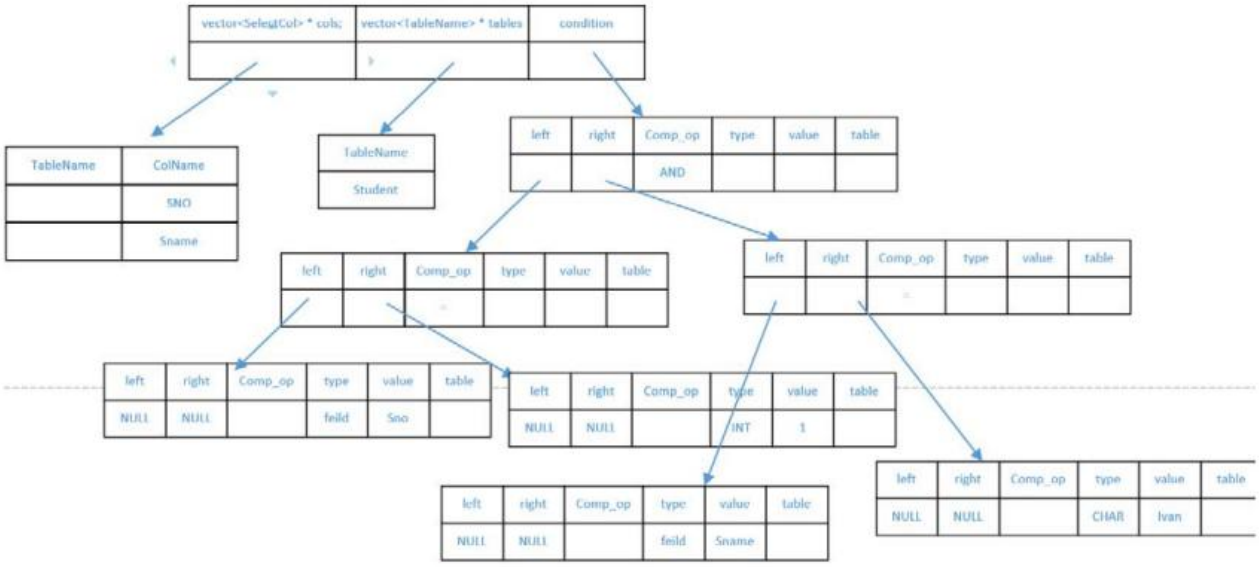
```
struct table_field_node {
    char * field_name;
    enum type {
        INT, STRING
    }type;
```

```
    int len;
    int offset;
    table_field_node * next = nullptr;
};
```

非终结符 conditions 的数据结构：

```
struct condexp_node {
    condexp_node * left = nullptr;
    enum op {
        AND, OR, EQ, G, B, NOT
    }op;
    condexp_node * right = nullptr;
    enum type {
        INT, STRING, COLUM, LOGIC
    }type;
    int intval;
    char * chval;
};
```

例句：SELECT SNO,SNAME FROM Student Where Sno=1 AND Sname=“Ivan”;



（4）DELETE 语句产生式语法结构：

例句：DELETE FROM Student WHERE Sno=1;

deletesql: DELETE FROM table ';'

| DELETE FROM table WHERE conditions ';'

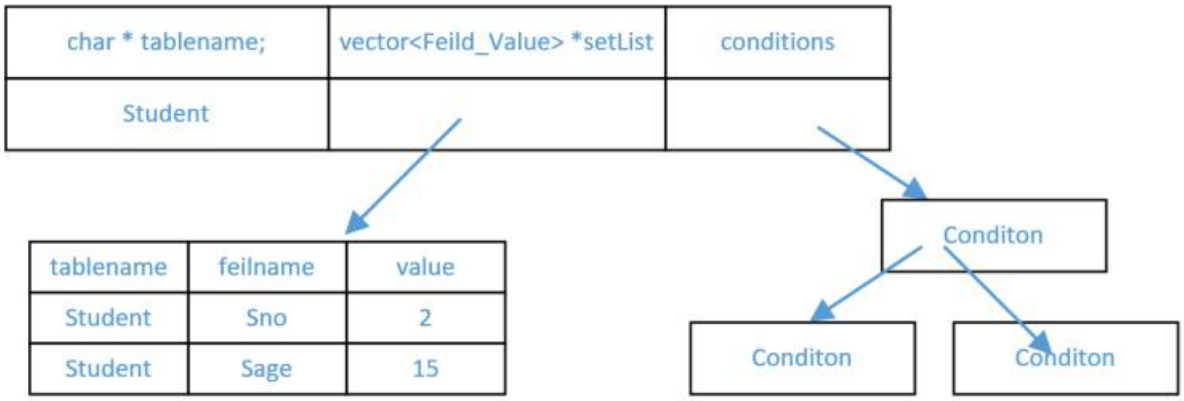
DELETE 语句数据结构同 SELECT 语句

（5）UPDATE 语句产生式语法结构

updatesql: UPDATE table SET updates WHERE conditions ';'

UPDATE 语句数据结构同 SELECT 语句

例句：UPDATE Student SET Sno=2,Sage=15 WHERE conditions ;



对应的数据结构如下图所示：

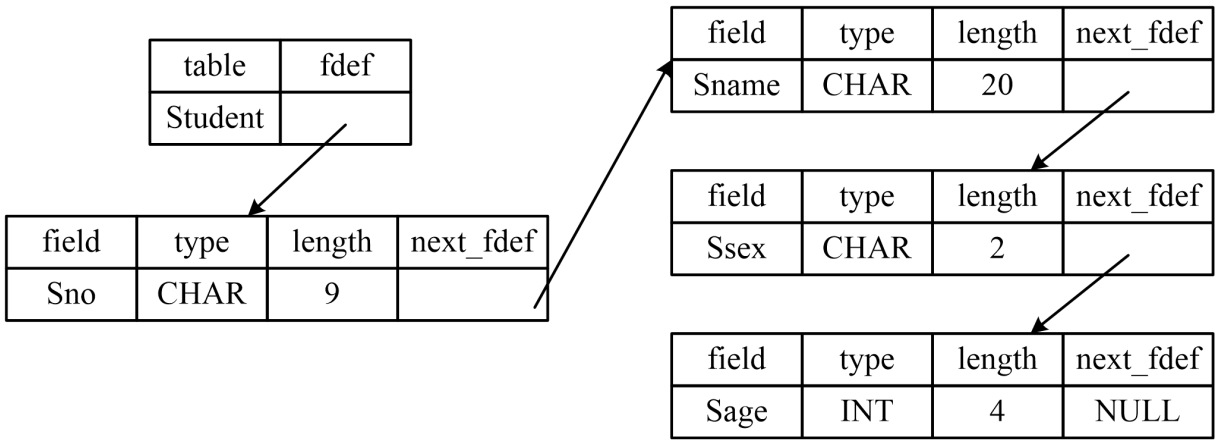


图 2 数据结构图

2.3 执行流程

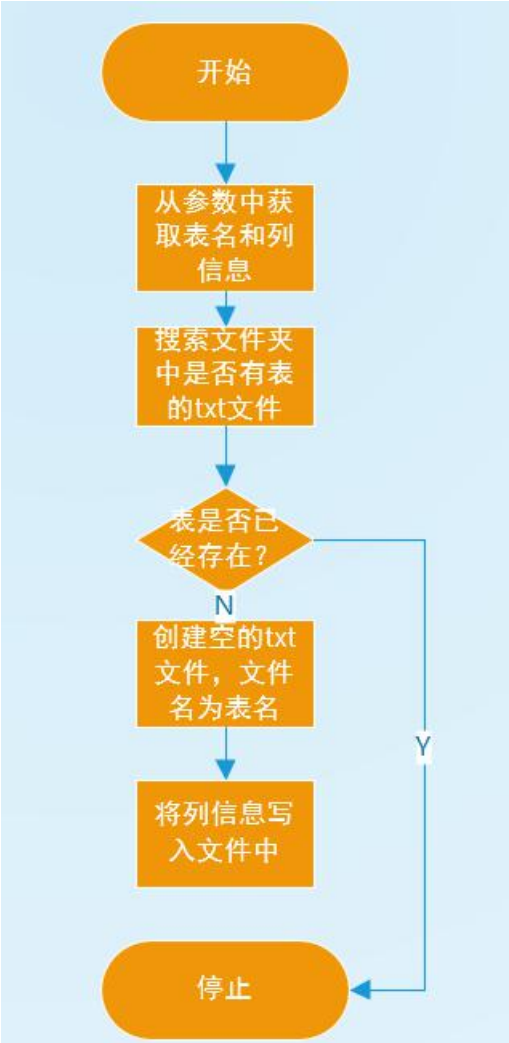
函数名称：bool My_CreateTable(CreateRootValue * createInfo);

函数说明：根据传入的参数中包含的表信息创建表

输入参数：createInfo 非终结符 createsql 的属性值

输出参数：false 表示错误，true 表示创建成功

执行流程：



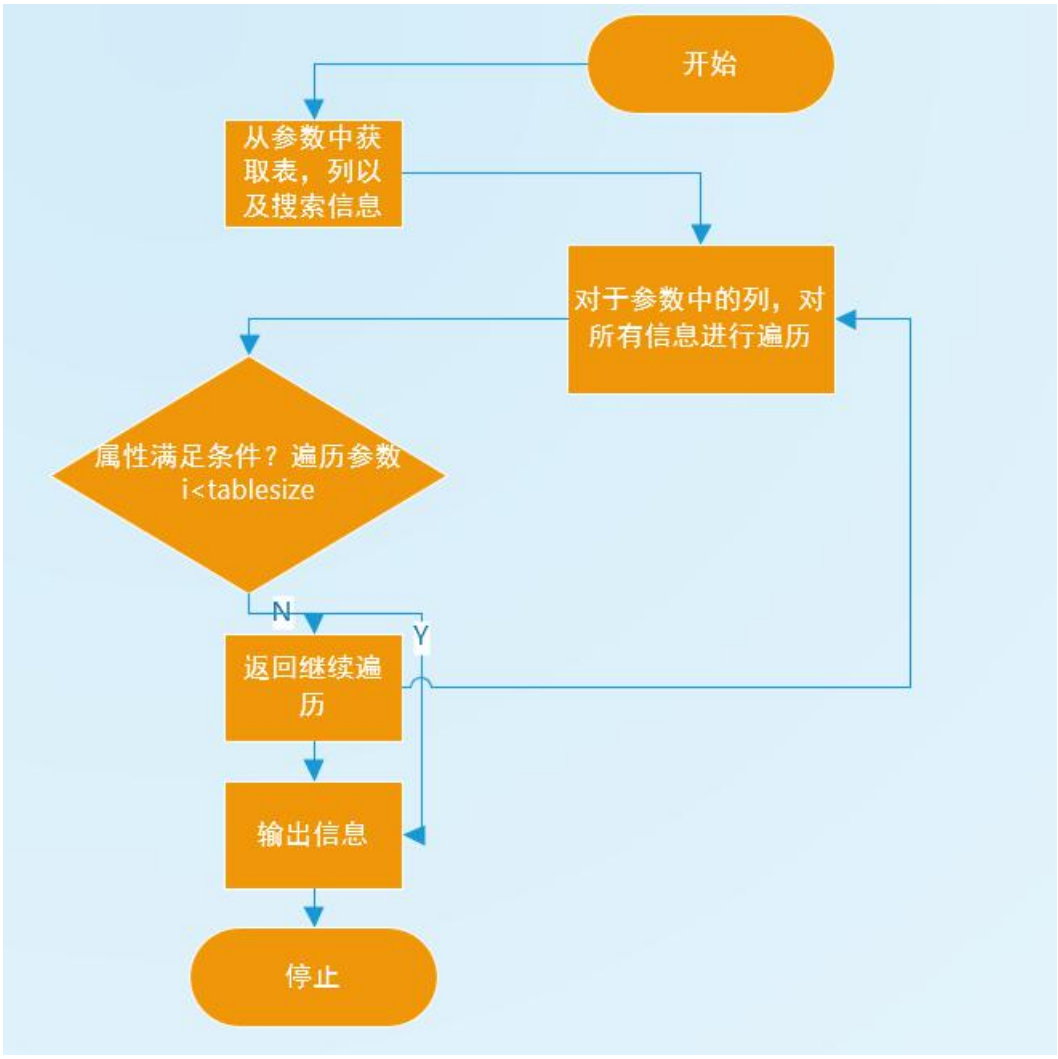
函数名称：bool My_Select(SelectRootValue * selectInfo);

函数说明：select 输入的信息并且将结果打印出来

输入参数：**selectInfo** 非终结符 selectsql 的属性值

输出参数：false 表示操作失败，true 表示操作成功

执行流程：



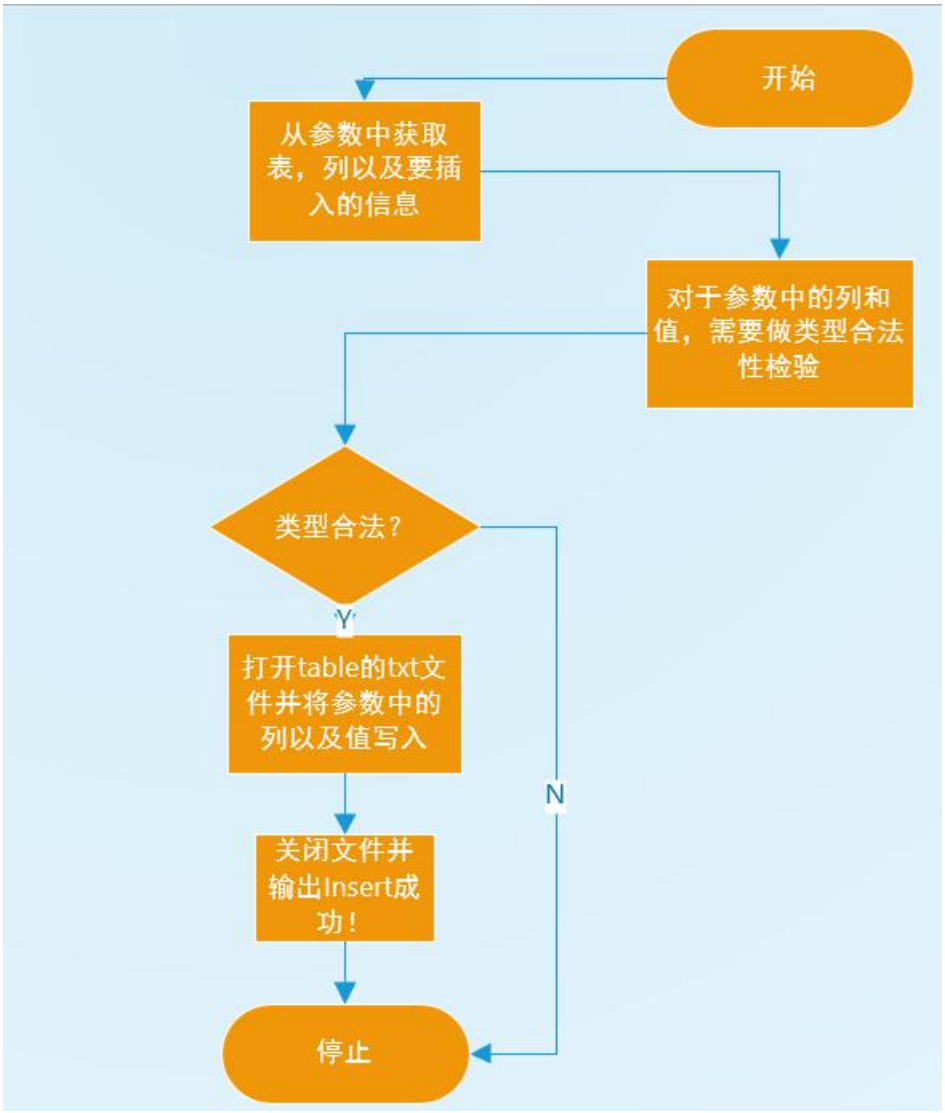
函数名称：bool My_Insert(InsertRootValue * insertInfo);

函数说明：执行 Insert 操作，Insert 信息为参数 insertInfo

输入参数：**insertInfo** 非终结符 insertsql 的属性值

输出参数：false 表示操作失败，true 表示操作成功

执行流程：



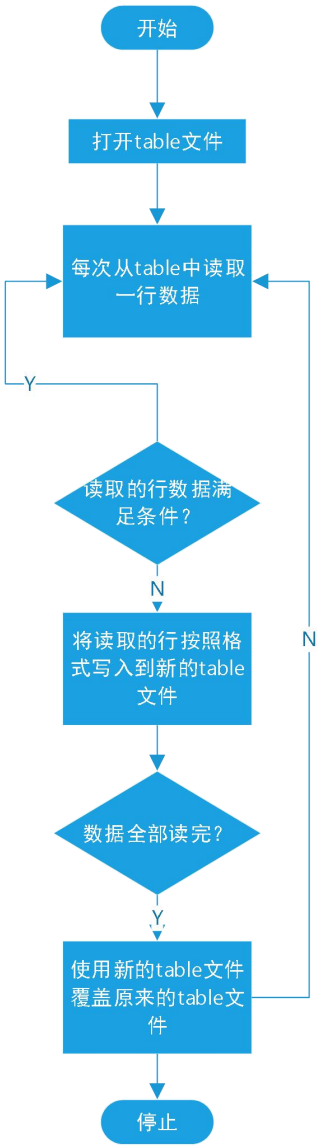
函数名称：bool My_Delete(DeleteRootValue * deleteInfo);

函数说明：根据表中符合条件的信息对一个表中的列进行删除

输入参数：**deleteInfo** 非终结符 deletesql 的属性值

输出参数：false 表示操作失败，true 表示操作成功

执行流程：



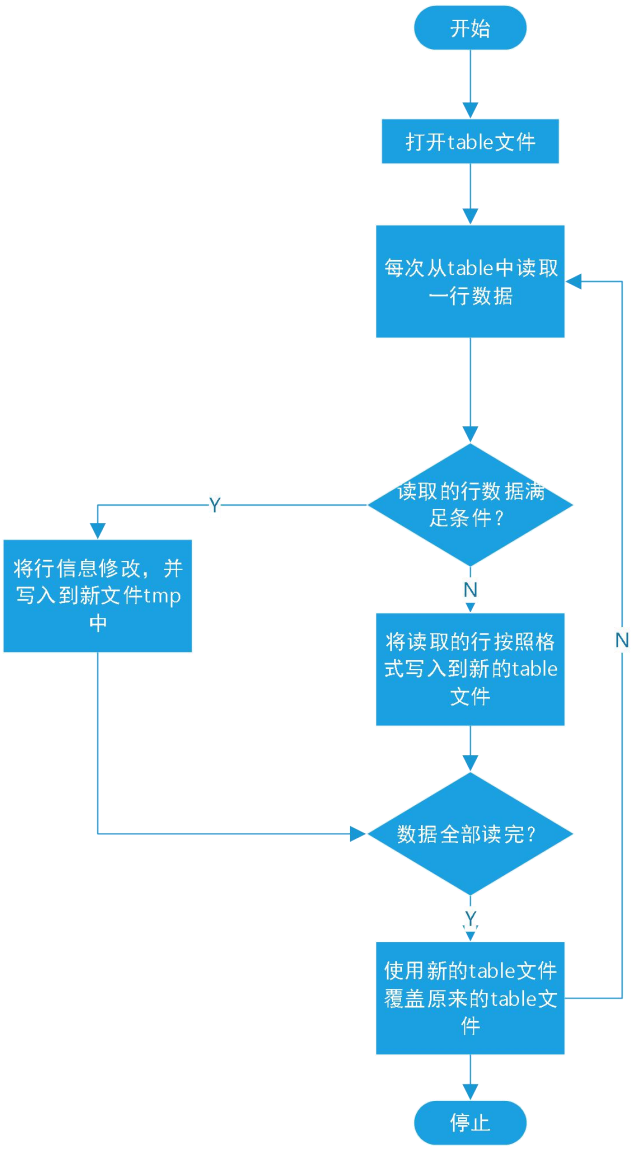
函数名称：bool My_Update(UpdateRootValue * updateInfo);

函数说明：根据表中符合条件的信息对一个表中的列进行修改

输入参数：**updateInfo** 非终结符 updatesql 的属性值

输出参数：false 表示操作失败，true 表示操作成功

执行流程：



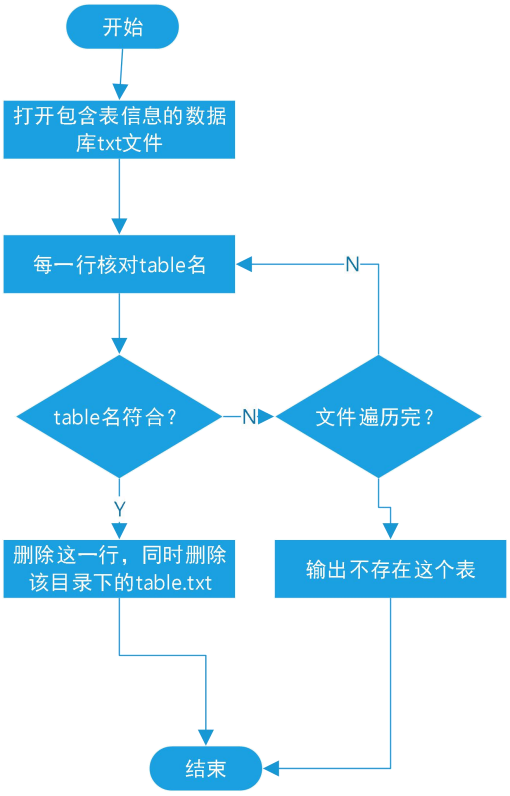
函数名称：bool My_DropTable(char * tablename);

函数说明：删除表

输入参数：tablename 为要删除的表名

输出参数：false 表示操作失败，true 表示操作成功

执行流程：



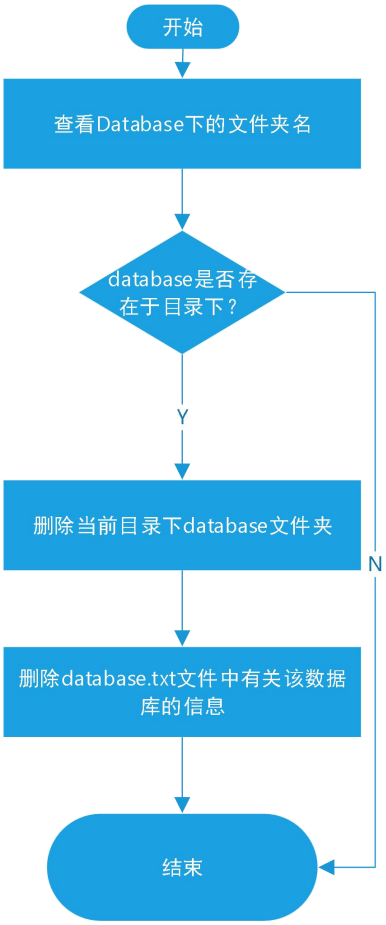
函数名称 bool My_CreateDataBase(char * database);

函数说明：创建数据库

输入参数：为要建立的数据库的名字

输出参数：false 表示操作失败，true 表示操作成功

执行流程：



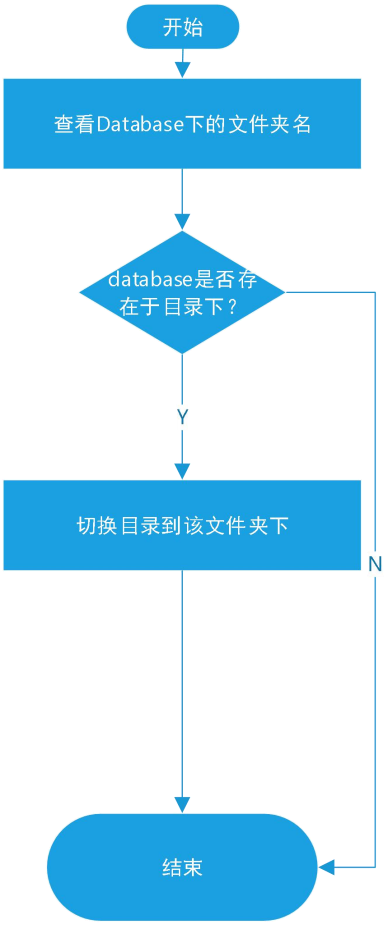
函数名称：bool My_UseDataBase(char * database);

函数说明：切换数据库

输入参数：为要切换的数据库的名字

输出参数：false 表示操作失败，true 表示操作成功

执行流程：



2.4 功能测试

按照测试 txt 文件中的语句测试所实现的 SQL 语句的基本功能，框架如下：

测试 1

输入：

```
CREATE DATABASE XJGL;

CREATE DATABASE JUST_FOR_TEST;

CREATE DATABASE JUST_FOR_TEST;
```

输出：

```
mysql> CREATE DATABASE XJGL;

mysql>CREATE DATABASE JUST_FOR_TEST;

mysql>CREATE DATABASE JUST_FOR_TEST;
该数据库已存在
```

测试 2

输入：

```
SHOW DATABASES;

DROP DATABASE JUST_FOR_TEST;

SHOW DATABASES;
```

输出：

```
mysql>SHOW DATABASES;
xjgl
just_for_test

mysql>DROP DATABASE JUST_FOR_TEST;

mysql>SHOW DATABASES;
xjgl

mysql>USE XJGL;
切换成功，当前数据库:xjgl

mysql>
```

测试 3

输入：

USE XJGL;

输出：

```
mysql>USE XJGL;
切换成功，当前数据库:xjgl
```

测试 4

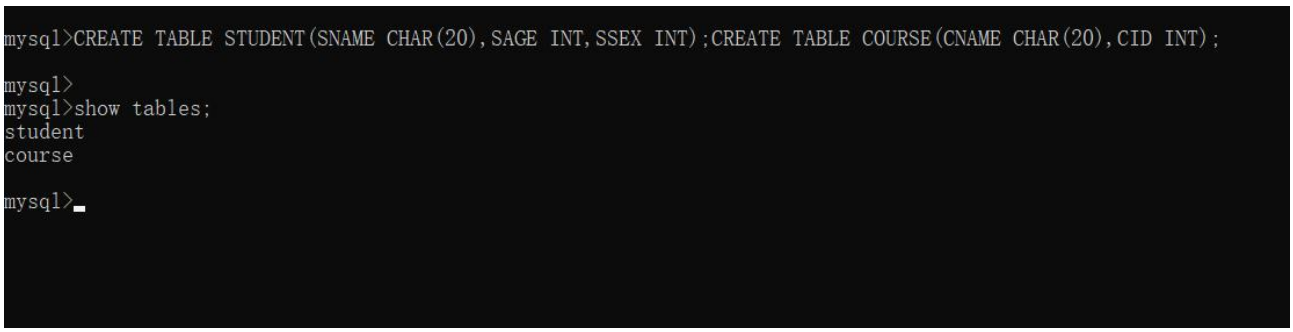
输入：

CREATE TABLE STUDENT(SNAME CHAR(20),SAGE INT,SSEX INT);

CREATE TABLE COURSE(CNAME CHAR(20),CID INT);

SHOW TABLES;

输出：



测试 5

输入：

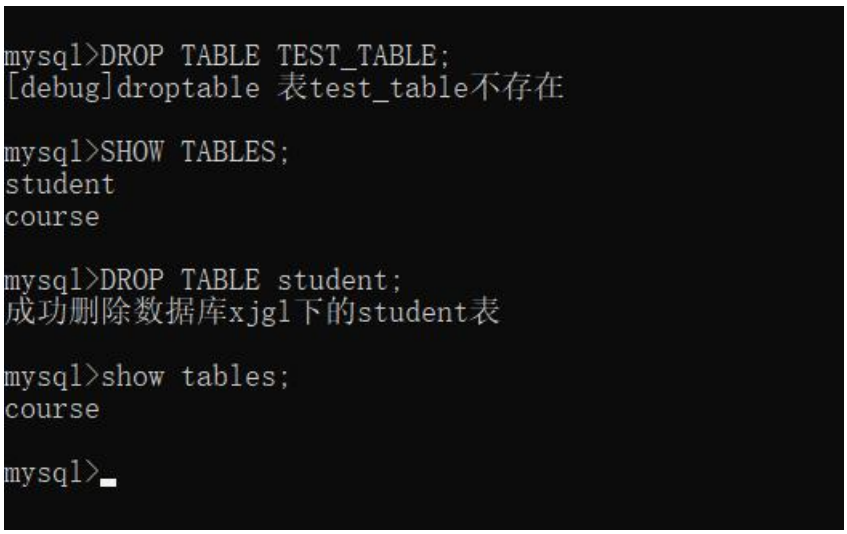
DROP TABLE TEST_TABLE;

SHOW TABLES;

DROP TABLE STUDENT;

SHOW TABLES;

输出：



测试 6

输入：

CREATE TABLE STUDENT(SNAME CHAR(20),SAGE INT,SSEX INT);

INSERT INTO STUDENT(SNAME,SAGE,SSEX) VALUES ('ZHANGSAN',22,1);

INSERT INTO STUDENT VALUES ('LISI',23,0);

INSERT INTO STUDENT(SNAME,SAGE) VALUES ('WANGWU',21);

INSERT INTO STUDENT VALUES ('ZHAOLIU',22,1);

INSERT INTO STUDENT VALUES ('XIAOBAI',23,0);

INSERT INTO STUDENT VALUES ('XIAOHEI',19,0);

INSERT INTO COURSE(CNAME,CID) VALUES ('DB',1);

INSERT INTO COURSE (CNAME,CID) VALUES('COMPILER',2);

insert into course (CNAME,CID) VALUES('C',3);

输出：

```
mysql>INSERT INTO STUDENT(SNAME,SAGE,SSEX) VALUES ('ZHANGSAN',22,1);
mysql>INSERT INTO STUDENT VALUES ('LISI',23,0);
mysql>INSERT INTO STUDENT(SNAME,SAGE) VALUES ('WANGWU',21);
mysql>INSERT INTO STUDENT VALUES ('ZHAOLIU',22,1);
mysql>INSERT INTO STUDENT VALUES ('XIAOBAI',23,0);
mysql>INSERT INTO STUDENT VALUES ('XIAOHEI',19,0);
mysql>INSERT INTO COURSE(CNAME,CID) VALUES ('DB',1);
mysql>INSERT INTO COURSE (CNAME,CID) VALUES('COMPILER',2);
mysql>insert into course (CNAME,CID) VALUES('C',3);
mysql>
```

测试 6

输入：

SELECT SNAME,SAGE,SSEX FROM STUDENT;

输出：

```
mysql>SELECT SNAME,SAGE,SSEX FROM STUDENT;

+-----------+------+------+
| SNAME     | SAGE | SSEX |
+-----------+------+------+
| ZHANGSAN  |   22 |    1 |
| LISI      |   23 |    0 |
| WANGWU    |   21 | null |
| ZHAOLIU   |   22 |    1 |
| XIAOBAI   |   23 |    0 |
| XIAOHEI   |   19 |    0 |
+-----------+------+------+

mysql>
```

测试 7

输入：

SELECT SNAME,SAGE FROM STUDENT;

SELECT * FROM STUDENT;

输出：

```
mysql>SELECT SNAME,SAGE FROM STUDENT;

+-----------+------+
| SNAME     | SAGE |
+-----------+------+
| ZHANGSAN  |   22 |
| LISI      |   23 |
| WANGWU    |   21 |
| ZHAOLIU   |   22 |
| XIAOBAI   |   23 |
| XIAOHEI   |   19 |
+-----------+------+

mysql>SELECT * FROM STUDENT;
+-----------+------+------+
| SNAME     | SAGE | SSEX |
+-----------+------+------+
| ZHANGSAN  |   22 |    1 |
| LISI      |   23 |    0 |
| WANGWU    |   21 | null |
| ZHAOLIU   |   22 |    1 |
| XIAOBAI   |   23 |    0 |
| XIAOHEI   |   19 |    0 |
+-----------+------+------+

mysql>
```

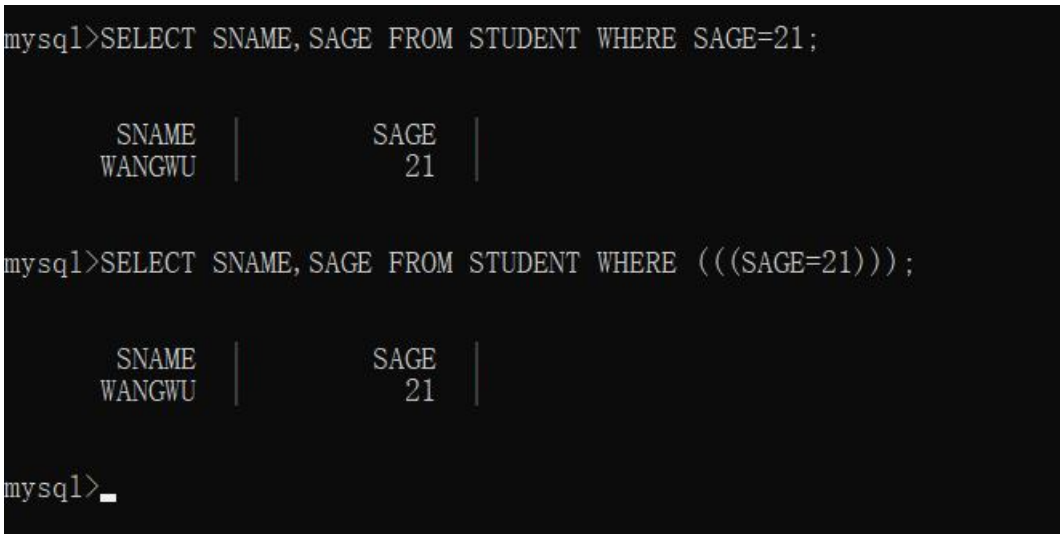
测试 8

输入：

SELECT SNAME,SAGE FROM STUDENT WHERE SAGE=21;

SELECT SNAME,SAGE FROM STUDENT WHERE (((SAGE=21)));

输出：



测试 9

输入：

SELECT SNAME,SAGE FROM STUDENT WHERE (SAGE>21) AND (SSEX=0);

SELECT SNAME,SAGE FROM STUDENT WHERE (SAGE>21) OR (SSEX=0);

SELECT * FROM STUDENT WHERE SSEX!=1;

输出：

```
mysql>SELECT SNAME,SAGE FROM STUDENT WHERE (SAGE>21) AND (SSEX=0);

+-------+------+
| SNAME | SAGE |
+-------+------+
| LISI  |   23 |
| XIAOBAI |   23 |
+-------+------+

mysql>SELECT SNAME,SAGE FROM STUDENT WHERE (SAGE>21) OR (SSEX=0);

+-------+------+
| SNAME | SAGE |
+-------+------+
| ZHANGSAN |   22 |
| LISI  |   23 |
| ZHAOLIU |   22 |
| XIAOBAI |   23 |
| XIAOHEI |   19 |
+-------+------+

mysql>SELECT * FROM STUDENT WHERE SSEX!=1;
+-------+------+------+
| SNAME | SAGE | SSEX |
+-------+------+------+
| LISI  |   23 |    0 |
| XIAOBAI |   23 |    0 |
| XIAOHEI |   19 |    0 |
+-------+------+------+

mysql>
```

测试 10

输入：//多表查询

SELECT * FROM STUDENT;SELECT * FROM COURSE;

输出：

```
mysql>SELECT * FROM STUDENT;SELECT * FROM COURSE;
+-------+------+------+
| SNAME | SAGE | SSEX |
+-------+------+------+
| ZHANGSAN |   22 |    1 |
| LISI     |   23 |    0 |
| WANGWU   |   21 | null |
| ZHAOLIU  |   22 |    1 |
| XIAOBAI  |   23 |    0 |
| XIAOHEI  |   19 |    0 |
+-------+------+------+

mysql> +----------+-----+
| CNAME    | CID |
+----------+-----+
| DB       |   1 |
| COMPILER |   2 |
| C        |   3 |
+----------+-----+

mysql>
```

测试 11

输入：

select * from student,course;

SELECT * FROM STUDENT,COURSE WHERE (SSEX=0) AND (CID=1);

输出：

```
mysql>select * from student,course;
+-------+------+------+
| SNAME | SAGE | SSEX |
+-------+------+------+
| ZHANGSAN |   22 |    1 |
| LISI     |   23 |    0 |
| WANGWU   |   21 | null |
| ZHAOLIU  |   22 |    1 |
| XIAOBAI  |   23 |    0 |
| XIAOHEI  |   19 |    0 |
+-------+------+------+

+----------+-----+
| CNAME    | CID |
+----------+-----+
| DB       |   1 |
| COMPILER |   2 |
| C        |   3 |
+----------+-----+


mysql>SELECT * FROM STUDENT,COURSE WHERE (SSEX=0) AND (CID=1);
+-------+------+------+
| SNAME | SAGE | SSEX |
+-------+------+------+
属性CID不存在属性CID不存在属性CID不存在属性CID不存在属性CID不存在属性CID不存在
+-------+-----+
| CNAME | CID |
+-------+-----+
属性SSEX不存在属性SSEX不存在属性SSEX不存在

mysql>
```

测试 12

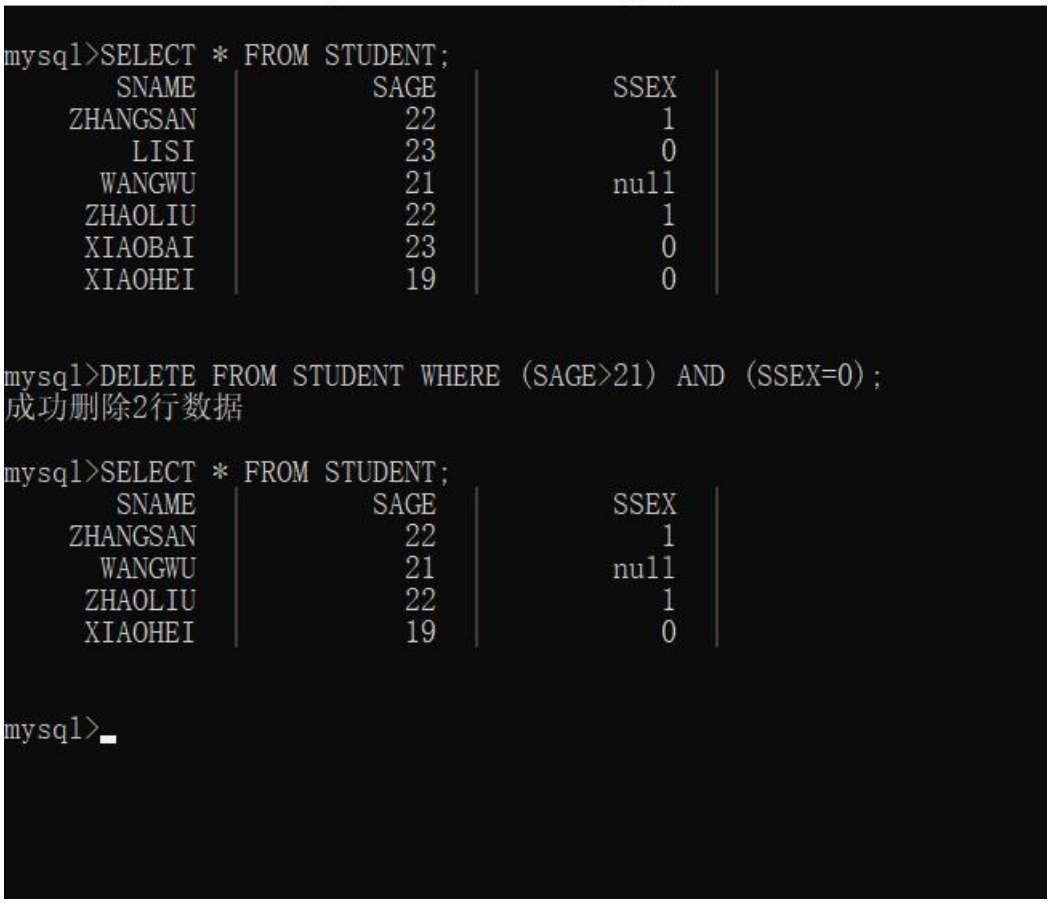
输入：

SELECT * FROM STUDENT;

DELETE FROM STUDENT WHERE (SAGE>21) AND (SSEX=0);

SELECT * FROM STUDENT;

输出：



测试 13

输入：

//测试 UPDATE

SELECT * FROM STUDENT;

UPDATE STUDENT SET SAGE=21 WHERE SSEX=1;

SELECT * FROM STUDENT;

UPDATE STUDENT SET SAGE=27,SSEX=1 WHERE SNAME='ZHANGSAN';

SELECT * FROM STUDENT;

输出：

```
mysql>SELECT * FROM STUDENT;
+-------+------+------+
| SNAME | SAGE | SSEX |
+-------+------+------+
| ZHANGSAN | 22 | 1 |
| WANGWU | 21 | null |
| ZHAOLIU | 22 | 1 |
| XIAOHEI | 19 | 0 |
+-------+------+------+

mysql>UPDATE STUDENT SET SAGE=21 WHERE SSEX=1;
成功更新2行数据

mysql>SELECT * FROM STUDENT;
+-------+------+------+
| SNAME | SAGE | SSEX |
+-------+------+------+
| ZHANGSAN | 21 | 1 |
| WANGWU | 21 | null |
| ZHAOLIU | 21 | 1 |
| XIAOHEI | 19 | 0 |
+-------+------+------+

mysql>UPDATE STUDENT SET SAGE=27,SSEX=1 WHERE SNAME='ZHANGSAN';
成功更新1行数据
成功更新1行数据

mysql>SELECT * FROM STUDENT;
+-------+------+------+
| SNAME | SAGE | SSEX |
+-------+------+------+
| ZHANGSAN | 27 | 1 |
| WANGWU | 21 | null |
| ZHAOLIU | 21 | 1 |
| XIAOHEI | 19 | 0 |
+-------+------+------+

mysql>
```

3.总结与未来工作

3.1 未完成功能

SQL 还有很多的功能，比如 grade，exists，还有 select 的多层嵌套使用，在本次实验中都由于时间关系等原因没有实现。这些实现会更加复杂，更具有挑战性。此外，我所涉及的 DBMS 系统是基于文件系统的而不是段页式管理这种比较实用的物理结构，这也是一个未完成的功能。

3.2 未来实现方案

在未来如果有时间和机会我会着重于以下几个方面来对该 DBMS 系统进行改进：

1.改进查询性能：优化查询引擎和索引结构，以提高查询性能。可以考虑实现更高效的查询优化和执行策略，如索引优化、查询重写和并行执行。

2.支持更多数据类型和数据结构：考虑扩展您的 DBMS 工具以支持更多种类的数据类型和数据结构。这样可以增加工具的灵活性和适用性，满足更广泛的应用需求。

3.引入缓存机制：使用缓存机制可以提高数据访问速度。您可以实现一个缓存层，将经常访问的数据或查询结果缓存到内存中，以减少文件系统的访问次数，提高响应速度。