编译原理上机报告

《DBMS 的设计与实现》

学号：15030199012　　姓名：朱仁博

手机：13032978342　　邮箱：rbzhu@stu.xidian.edu.cn

完成时间：2018 年　6 月　20 日

目　　录

1. 项目概况

1.1 基本目标

使用 Lex、Yacc、VC++6.0（C 语言），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

已经实现的功能、语句。

（1）CREATE DATABASE　　创建数据库

（2）SHOW DATABASES　　显示所有数据库

（3）DROP DATABASE　　删除数据库

（4）USE DATABASE　　选择数据库

（5）CREATE TABLE　　创建表

（6）SHOW TABLES　　显示所有表名

（7）DROP TABLE　　删除表

（8）INSERT　　插入元组

（9）SELECT　　查询元组

（10）UPDATE　　更新元组

（11）DELETE　　删除元组

（12）EXIT　　退出系统

2. 项目实现方案

2.1 逻辑结构与物理结构

重要的元数据：数据库 Database、表 Table、约束 Constraint、外码 ForeignKey、默认值 Default、索引 Index、视图 View、权限 Authority。

表 1 数据库 Database 的逻辑结构

列名	说明	类型
Users	用户	Char **
Tables	表	Table *
Logs	日志	Char **
…	…	…

表 2 表 Table 的逻辑结构

列名	说明	类型
Fields	列名	Char **
Values	值	Value *
Constraints	约束	Constraint *
Indexs	索引	Index *
Authorities	用户权限	Authority *

表 3 约束 Constraint 的逻辑结构（没做）

列名	说明	类型
Type	约束类型：NOT NULL、UNIQUE 、 PRIMARY KEY、FOREIGN KEY、CHECK、DEFAULT	Int
PrimaryKey	主码	Char *
ForeignKeys	外码	ForeignKey *
Defaults	默认值	Defalut *

表 4 外码 Foreignkey 的逻辑结构（没做）

列名	说明	类型
Field	列名	Char *
Table	被参照表名	Char *
Field_	被参照列名	Char *

表 5 默认值 Default 的逻辑结构（没做）

列名	说明	类型
Field	列名	Char *
DefalutValue	默认值	Value

表 6 索引 Index 的逻辑结构（没做）

列名	说明	类型
Type	索引类型：CLUSTERED、NONCLUSTERED…	Int
Fields	索引的列	Char **

表 7 视图 View 的逻辑结构（没做）

列名	说明	类型
ViewName	视图名称	Char *
ViewFields	视图列：包括选择的表、列、新列名	ViewField *
Tables	选择的表	Table *
Conditions	判断条件	Condition *
GroupBy	聚合函数	Char *
OrderBy	排序函数	Char *
Desc	是否逆序	Boolean

表 8 权限 Authority 的逻辑结构（没做）

列名	说明	类型
Type	权限类型	Int
Users	用户	Char **

物理结构：

（1）DBMS 中有 3 个数据库：database1、database2 和 database3。

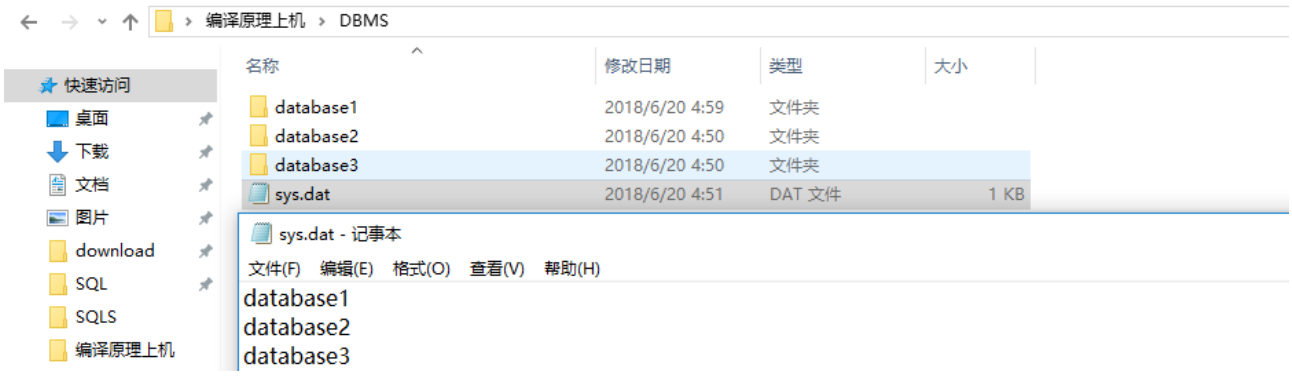

图 1 databses 物理结构

（2）database1 中有两个表：table1 和 table2，views 文件夹存放所有视图。

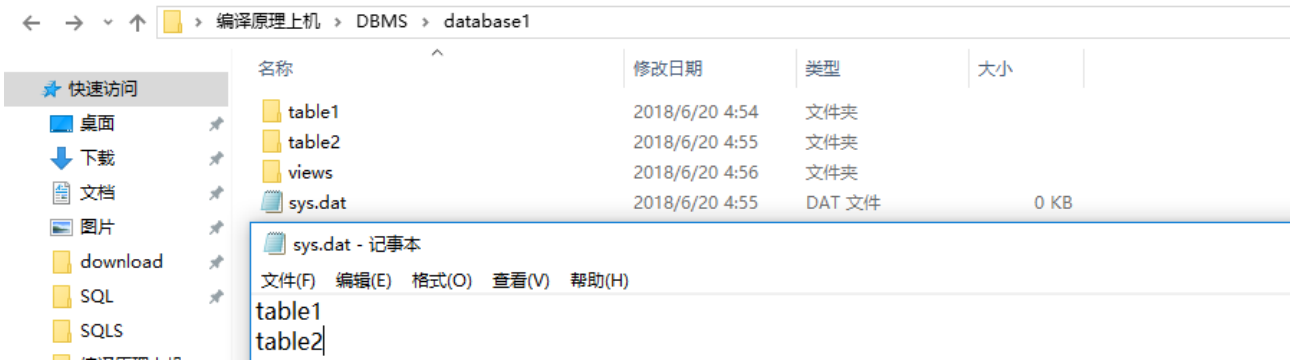

图 2 tables 物理结构

（3）在 table1 中，fields.txt 存放该表的列名，values.txt 存放该表的所有元组，constrains 文件夹存放该表的所有约束，indexs 文件夹存放该表的所有索引，authorities 文件夹存放该表的所有约束。其表结构对应的 SQL 语句：

```
CREATE TABLE table1
(
   name CHAR(20) NOT NULL DEFAULT '未命名',
   sno INT NOT NULL UNIQUE,
   age INT,
   PRIMARY KEY (name),
   FOREIGN KEY (sno) REFERENCE table2(sno),
   CHECK (age > 10),
);
```

```
CREATE CLUSTERED INDEX ix1 ON table1(name);
CREATE NONCLUSTERED INDEX ix2 ON table1(name,age);
GRANT SELECT ON table1 TO PUBLIC;
REVOKE DELETE ON table1 FROM PUBLIC;
GRANT DELETE ON table1 TO user1,user2,user4;
```

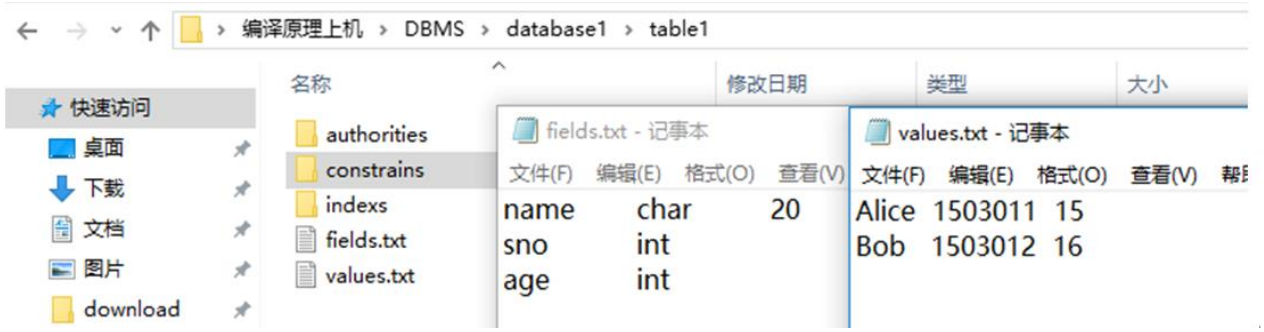

图 3 table 物理结构

图 4 constrains 物理结构

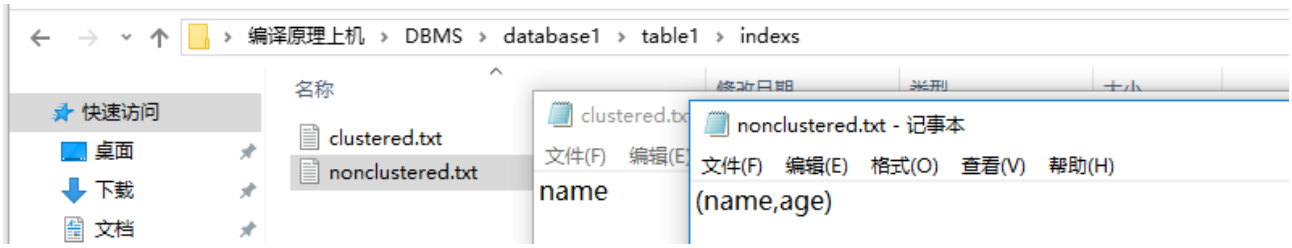

图 5 indexs 物理结构

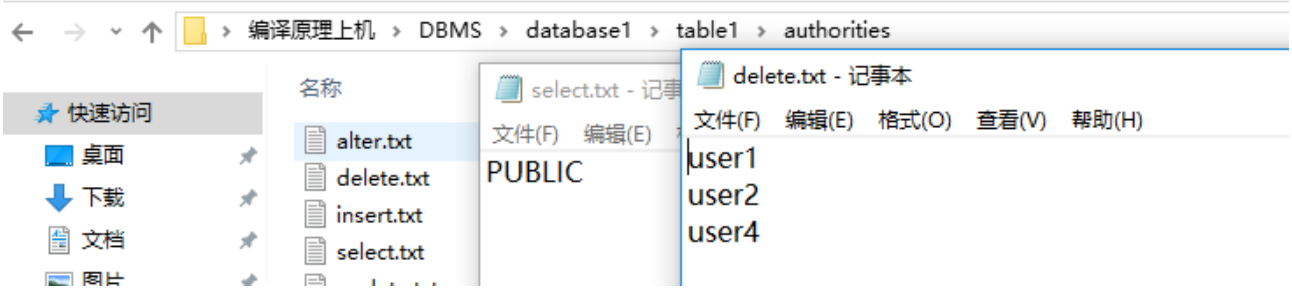

图 6 authorities 物理结构

（4）在 views 中，该数据库中有两个视图：view1 和 view2。

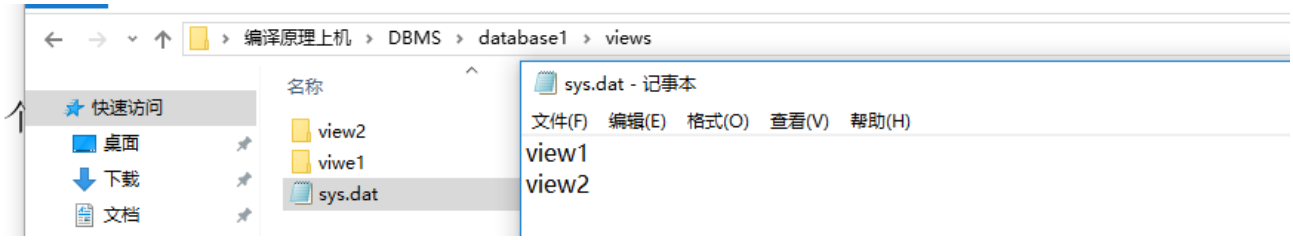

图 7 views 物理结构

（5）在 view1 中，对应 SQL 语句：

```
CREATE VIEW view1 AS
SELECT name AS new_name,sno
FROM table1
WHERE age > 15
ORDER BY sno
```

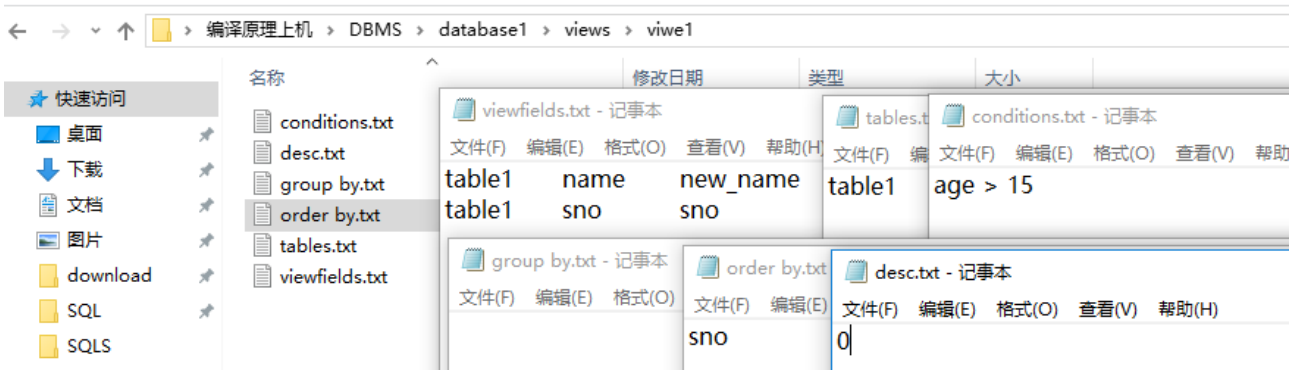

图 8 view 物理结构

物理结构优点：树形结构，逻辑清晰，几乎不会发生不同语句的命名冲突。

物理结构缺点：文件数量较多，SQL 语句执行效率受 IO 影响。

注：由于时间原因，以上的诸多功能没有实现，故实际使用的物理结构为 PPT 中的物理结构。

2.2 语法结构与数据结构

（1）CREATE DATABSE

产生式的语法结构：

```
createdb:        CREATE DATABASE ID ';'
```

非终结符 createdb 的属性使用如下结构说明：

```
struct Createdbstruct                    //1.createdb 语法树根节点
{
    char *database;
};
```

实例 CREATE DATABASE zhu1;

（2）SHOW DATABASES

产生式的语法结构：

```
showdb:          SHOW DATABASES ';'
```

实例 SHOW DATABASES;

（3）DROP DATABASE

产生式的语法结构：

```
dropdb:          DROP DATABASE ID ';'
```

非终结符 dropdb 的属性使用如下结构说明：

```
struct Dropdbstruct                      //3.dropdb 语法树根节点
{
    char *database;
};
```

实例 DROP DATABASE zhu2;

（4）USE DATABASE

产生式的语法结构：

```
usedb:          USE DATABASE ID ';'
```

非终结符 usedb 的属性使用如下结构说明：

```
char nowpath[1024] = "";
struct Usedbstruct                          //4.usedb 语法树根节点
{
    char *database;
};
```

实例 USE DATABASE zhu1;

（5）CREATE TABLE

产生式的语法结构：

```
createsql:      CREATE TABLE ID '(' field_types ')' ';'
field_types:  field_type | field_types ',' field_type
field_type:     ID type
type:           CHAR '(' NUMBER ')' | INT
```

非终结符 createsql 的属性使用如下结构说明：

```
struct Createstruct                         //5.createtb 语法树根节点
{
    char *table;
    struct Createfieldsdef *fdef;
};
```

非终结符 field_type 和 field_types 的属性使用如下结构说明：

```
struct Createfieldsdef //create 语句中的字段
{
    char *field;        //列名
    int type;           //1：字符串，2：整数
    int length;             //若 type 为 1，字符长
    struct Createfieldsdef *next_fdef;
};
```

实例 CREATE TABLE stu(sname CHAR(20),sno INT,age INT);对应的数据结构：

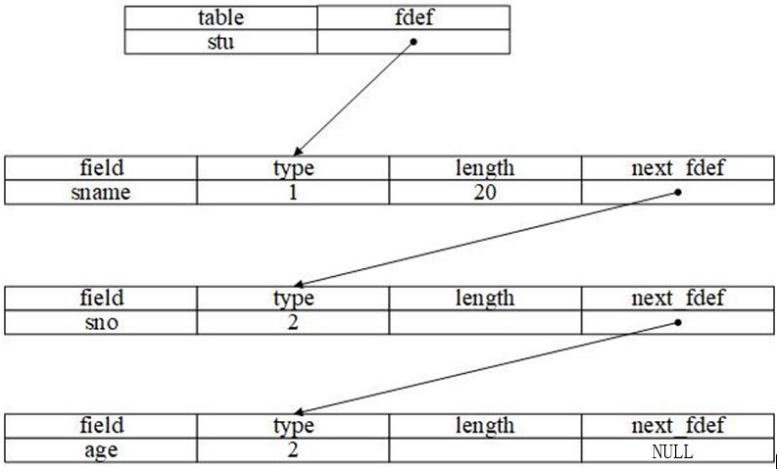

图 9 CREATE TABLE 数据结构图

（6）SHOW TABLES

产生式的语法结构：

showtb: SHOW TABLES ';'

实例：SHOW TABLES;

（7）DROP TABLE

产生式的语法结构：

droptb: DROP TABLE ID ';'

非终结符 droptb 的属性使用如下结构说明：

```
struct Droptbstruct                    //7.droptb 语法树根节点
{
    char *table;
};
```

实例 DROP TABLE xx;

（8）INSERT

产生式的语法结构：

```
insertsql:      INSERT INTO ID '(' values ')' ';'
values:         value | values ',' value
value:          STRING | NUMBER
```

非终结符 insertsql 的属性使用如下结构说明：

```
struct Insertsturct                    //8.insert 语法树根节点
{
    char *table;
    struct Values *v;
};
```

非终结符 value 和 values 的属性使用如下结构说明：

```
struct Values                //元组的字段
{
    char *table;
    int type;
    char *string;
    struct Values *next_v;
};
```

实例 INSERT INTO stu('name1',1503001,15); 对应的数据结构：

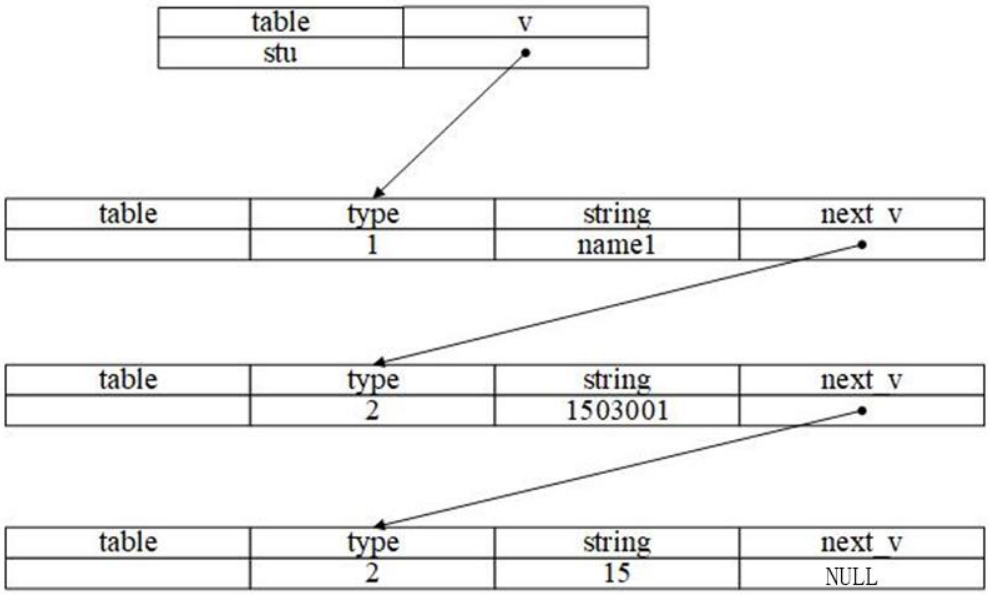

图 10 INSERT 数据结构图

（9）SELECT

产生式的语法结构：

```
selectsql:      SELECT fields_star FROM tables ';'
                | SELECT fields_star FROM tables WHERE conditions ';'
fields_star:    table_fields | '*'
table_fields:   table_field | table_fields ',' table_field
table_field:    ID | ID '.' ID
tables:         tables ',' ID | ID
conditions:     condition | '(' conditions ')' | conditions AND conditions | conditions OR conditions
condition:      table_field op table_field | table_field op STRING    | table_field op NUMBER
op:             '=' | '<' | '>' | '!'
```

非终结符 selectsql 的属性使用如下结构说明：

```
struct Selectstruct                     //9.select 语法树根节点
{
    struct Selectedfields *sf;
    struct Selectedtables *st;
    struct Conditions *cons;
};
```

非终结符 fields_star、table_fields 和 table_field 的属性使用如下结构说明：

```
struct Selectedfields   //选择的字段
{
    char *table;
    char *field;
    struct Selectedfields *next_sf;
};
```

非终结符 tables 的属性使用如下结构说明：

```
struct Selectedtables  //选择的表
{
    char *table;
    struct Selectedtables *next_st;
};
```

非终结符 condition 和 conditions 的属性使用如下结构说明：

```
struct Conditions           //条件
{
    struct Conditions *left;
    struct Conditions *right;
    char op;            //'a'：and，'o'：or，'!'：!=
    int type;           //0：字段，1：字符串，2：整数
```

```
    char *value;
    char *table;
};
```

实例：SELECT stu.sno FROM class,stu WHERE class.cno = 1 and class.sno = stu.sno ; 对应的数据结构：

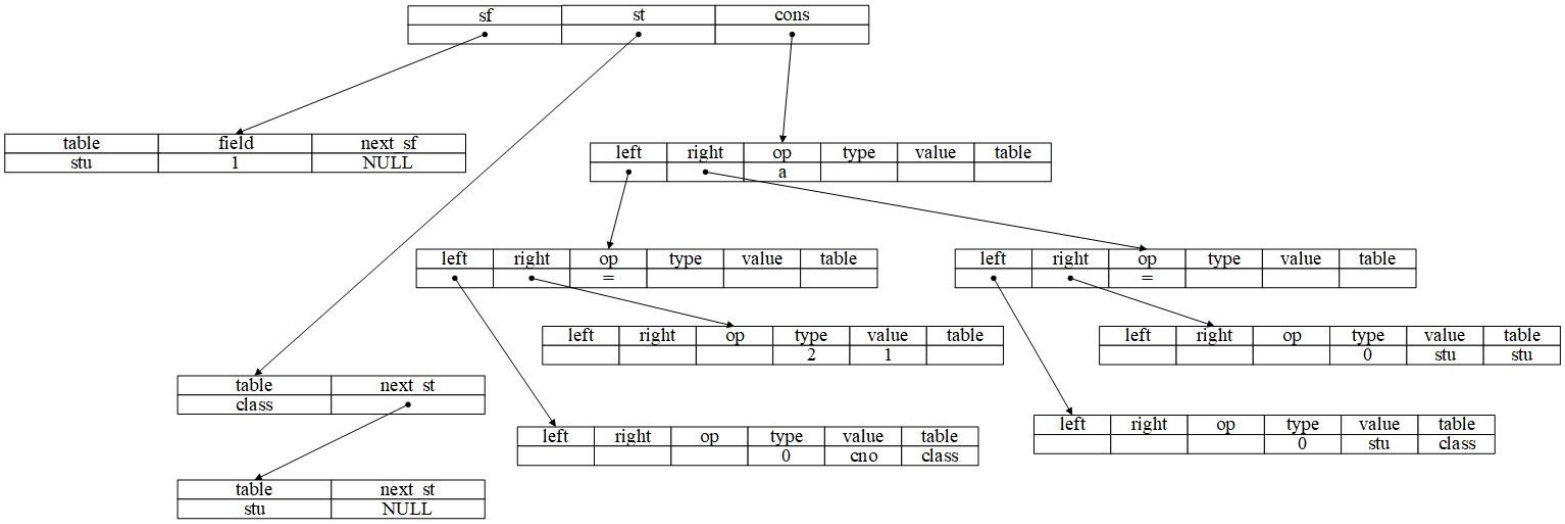

图 11 SELECT 数据结构图

（10） UPDATE

产生式的语法结构：

```
updatesql:      UPDATE ID SET changes WHERE conditions ';'
changes:    change | changes ',' change
change:     ID '=' STRING | ID '=' NUMBER
conditions: condition | '(' conditions ')' | conditions AND conditions | conditions OR conditions
condition:      table_field op table_field | table_field op STRING    | table_field op NUMBER
op:             '=' | '<' | '>' | '!'
```

非终结符 updatesql 的属性使用如下结构说明：

```
struct Updatastruct                     //10.update 语法树根节点
{
    char *table;
    struct Changestruct *ch;
    struct Conditions *cons;
};
```

非终结符 change 和 changes 的属性使用如下结构说明：

```
struct Changestruct         //修改的内容
```

```
{
    char *string;
    int type;
    char *value;
    struct Changestruct *next_ch;
};
```

非终结符 condition 和 conditions 的属性使用见 SELECT。

实例 UPDATE stu SET age = 20 WHERE sname = 'name1';对应的数据结构：

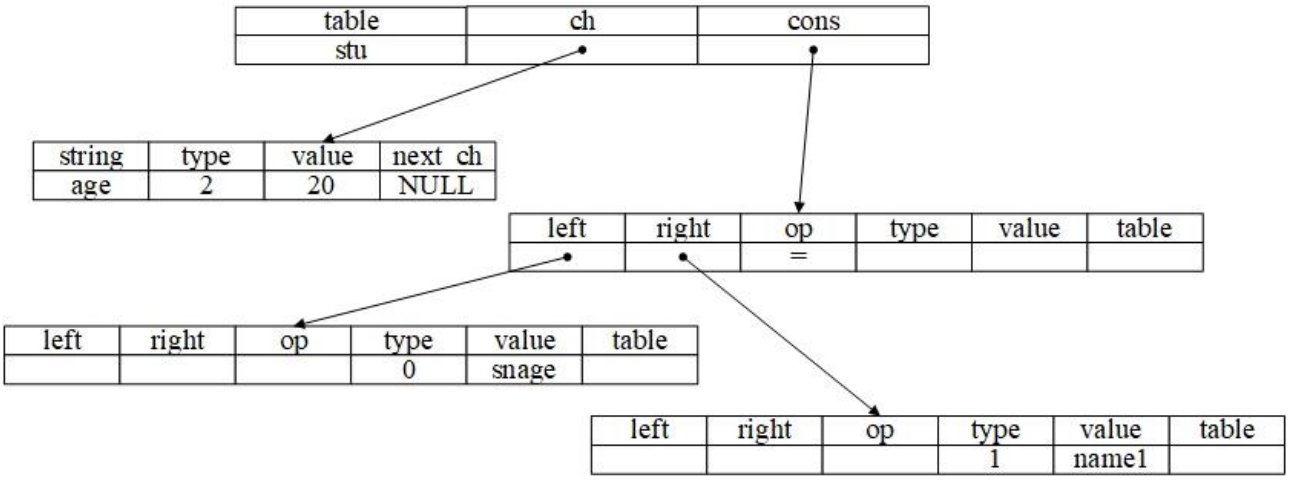

图 12 UPDATE 数据结构图

（11） DELETE

产生式的语法结构：

```
deletesql:      DELET FROM ID WHERE conditions ';'
conditions:     condition | '(' conditions ')' | conditions AND conditions | conditions OR conditions
condition:      table_field op table_field | table_field op STRING    | table_field op NUMBER
op:             '=' | '<' | '>' | '!'
```

非终结符 usedb 的属性使用如下结构说明：

```
struct Deletestruct                    //11.delete 语法树根节点
{
    char *tables;
    struct Conditions *cons;
};
```

非终结符 condition 和 conditions 的属性使用见 SELECT。

实例 DELETE FROM stu WHERE age < 20; 对应的数据结构：

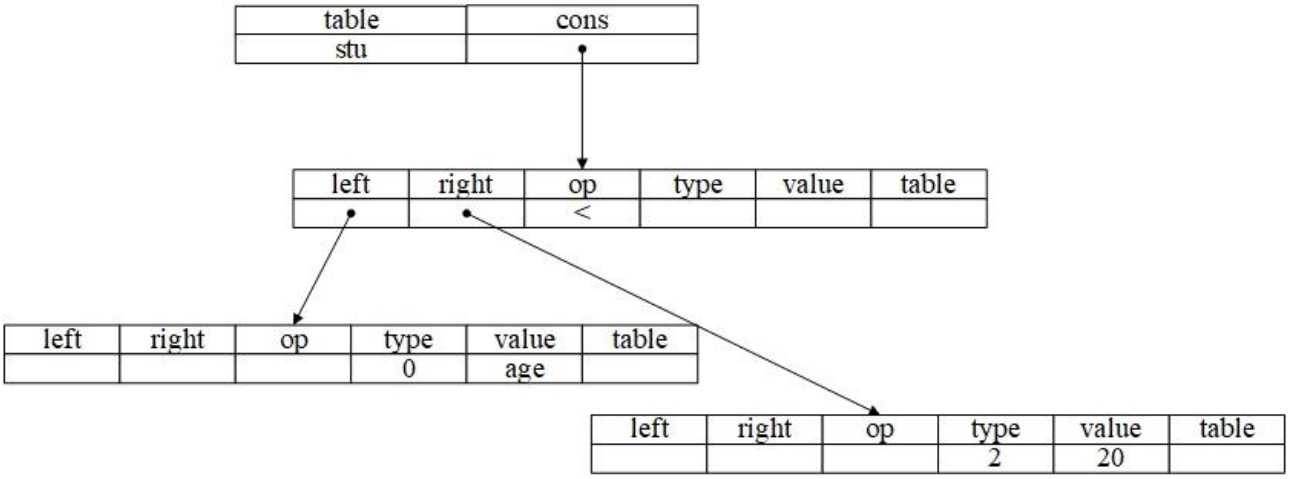

图 13 DELETE 数据结构图

（12） EXIT

产生式的语法结构：

exit: EXIT ';'

实例：EXIT;

（13） 整体

产生式的语法结构：

statements: statements statement | statement | error '/n';
statement: createdb | dropdb | showdb | exit | usedb | createsql | showtb | droptb | insertsql | selectsql | updatesql | deletesql;

非终结符 statements 的属性使用如下结构说明：

```
struct Statements                    //13.总根节点
{
    int type;
    void * sql;
    struct statements *next_s;
};
```

数据结构：

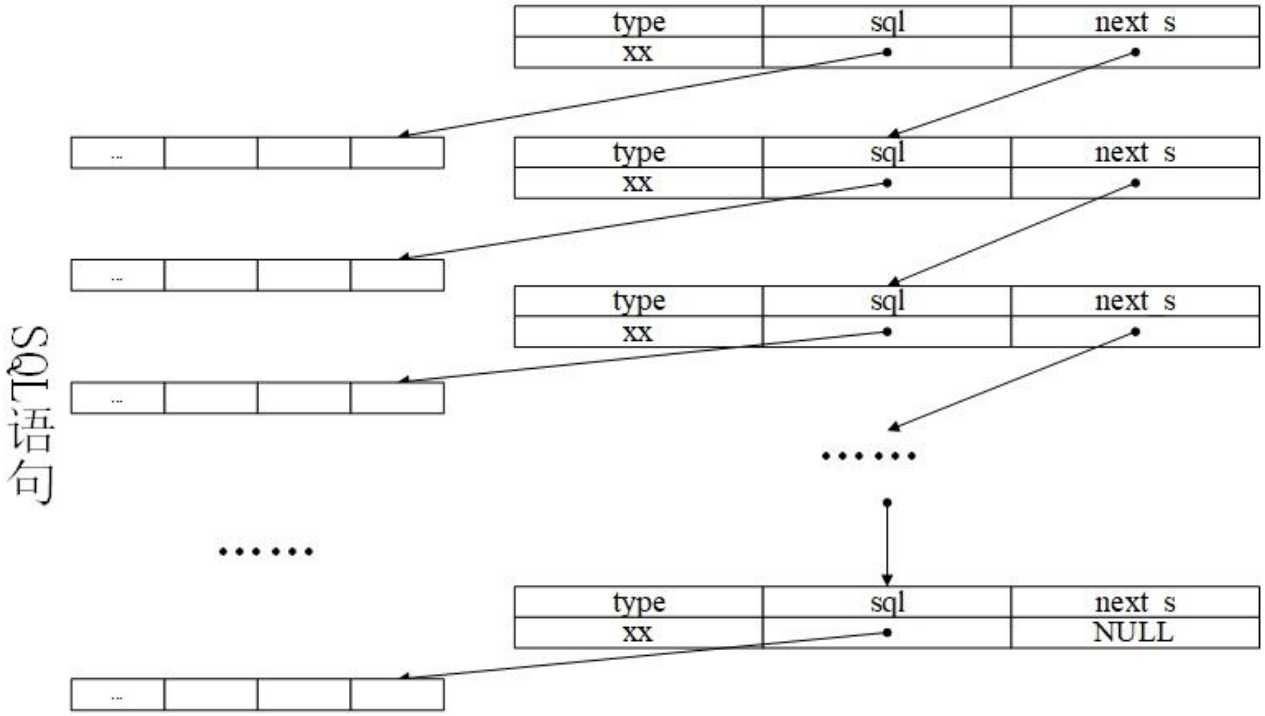

图 14 整体数据结构图

2.3 执行流程

（1）CREATE DATABASE

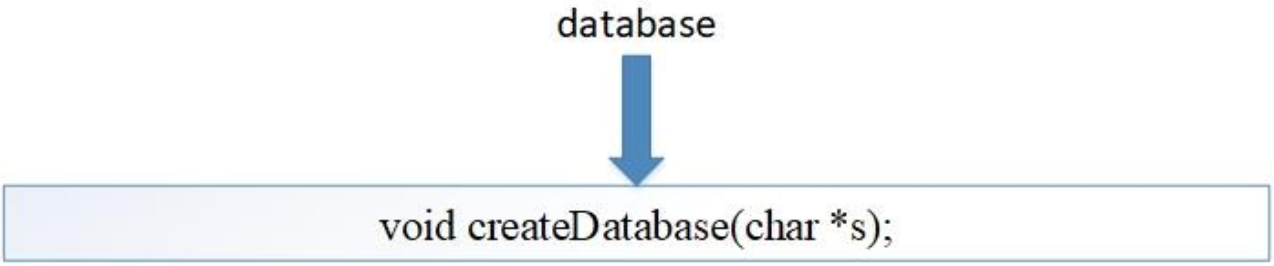

图 15 CREATE 流程图

函数名称：void createDatabase(char *s)

函数说明：创建数据库

输入参数：database 名称

输出参数：无

执行流程： ①判断数据库是否已存在，若已存在，结束。

②创建相关数据库文件夹，更新 sys.dat 文件。

（2）SHOW DATABSES

图 16 SHOW DATABASES 流程图

函数名称：void showDatabases()

函数说明：显示所有数据库

输入参数：无

输出参数：无

执行流程：从 sys.dat 文件中读取所有数据库名称，并打印。

（3）DROP DATABASE

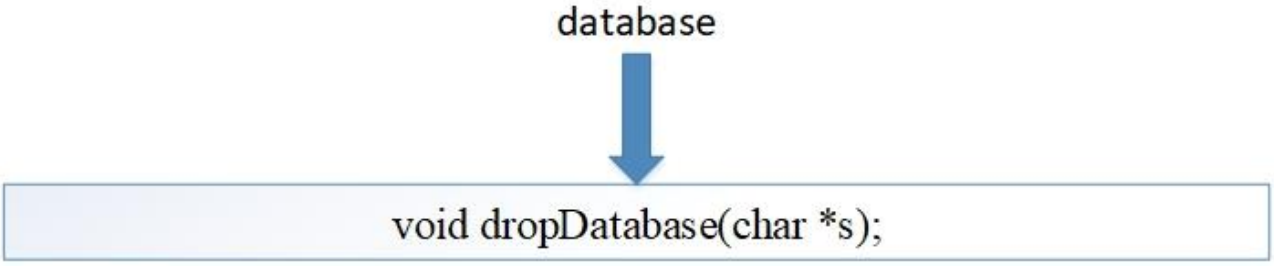

图 17 DROP DATABASE 流程图

函数名称：void dropDatabase(char *s)

函数说明：删除数据库

输入参数：数据库名称

输出参数：无

执行流程： ①判断数据库是否已存在，若不存在，结束。

②删除该数据库文件夹中的 sys.dat 文件。

③删除该数据库文件夹中的 txt 文件。

④删除该数据库文件夹。

⑤更新 sys.dat 文件。

（4）USE DATABASE

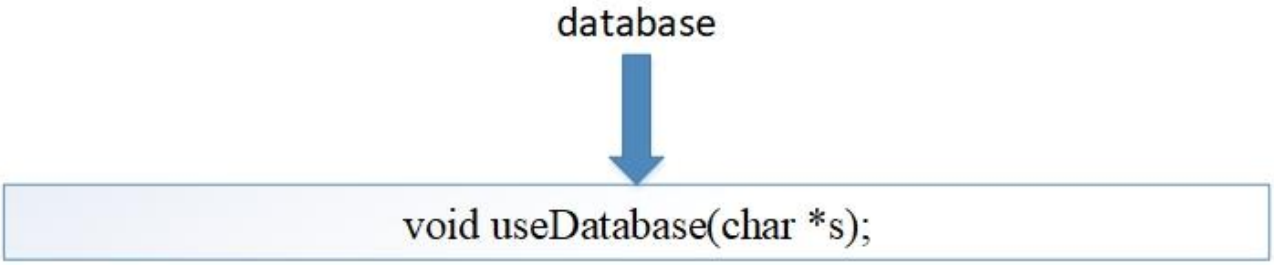

图 18 USE DATABASE 流程图

函数名称：void useDatabase(char *s)

函数说明：使用数据库

输入参数：数据库名称

输出参数：无

执行流程： ①判断数据库是否已存在，若不存在，结束。

②更改当前路径为数据库文件夹。

（5）CREATE TABLE

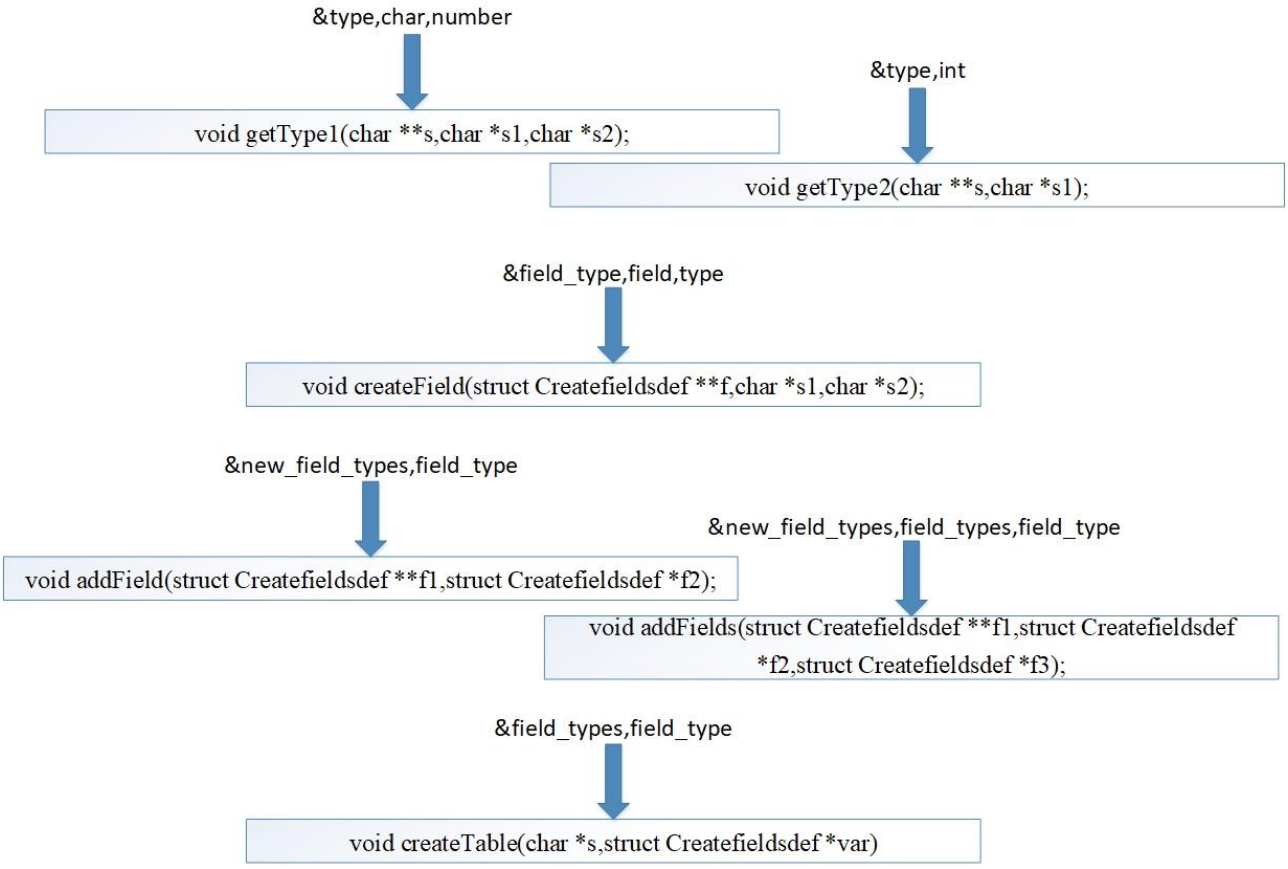

图 19 CREATE TABLE 流程图

整体流程：

①读入类型，结合列名生成字段。

②生成累计字段。

③使用表名和累计字段新建表。

函数名称：void getType1(char **s,char *s1,char *s2)

函数说明：读入类型

输入参数：保存结果的指针地址，字符串 char，字符串数字

输出参数：无

执行流程：将新串“s1(s2)”的地址赋给 s。

函数名称：void getType2(char **s,char *s1)

函数说明：读入类型

输入参数：字符串 int

输出参数：无

执行流程：将新串“s1”的地址赋给 s。

函数名称：void createField(struct Createfieldsdef **f,char *s1,char *s2)

函数说明：生成字段

输入参数：保存结果的指针地址，列名，类型

输出参数：无

执行流程：用列名与类型构造结构体，将地址赋给 f。

函数名称：void addField(struct Createfieldsdef **f1,struct Createfieldsdef *f2)

函数说明：单个字段生成累计字段

输入参数：保存结果的指针地址，字段

输出参数：无

执行流程：将字段地址赋给 f1。

函数名称：void addFields(struct Createfieldsdef **f1,struct Createfieldsdef *f2,struct Createfieldsdef *f3)

函数说明：累计字段和单个字段生成累计字段

输入参数：保存结果的指针地址，累计字段，单个字段

输出参数：无

执行流程：将单个字段串在累计字段之后，并把累计字段的地址赋给 f1。

函数名称：void createTable(char *s,struct Createfieldsdef *var)

函数说明：用表名和累计字段生成表

输入参数：表名，累计字段

输出参数：无

执行流程： ①判断是否在一个数据库中，若否，则结束。

②判断该表是否已存在，若已存在，则结束。

③判断累计字段中是否有重复，若有，则结束。

④创建表的 txt 文件，更新 sys.dat 文件。

（6）SHOW TABLES

图 20 SHOW TABLES 流程图

函数名称：void showTables()

函数说明：显示当前数据库中所有表名

输入参数：无

输出参数：无

执行流程： ①判断是否在一个数据库中，若否，则结束。

②从 sys.dat 文件中读取所有表名，去重并打印。

（7）DROP TABLE

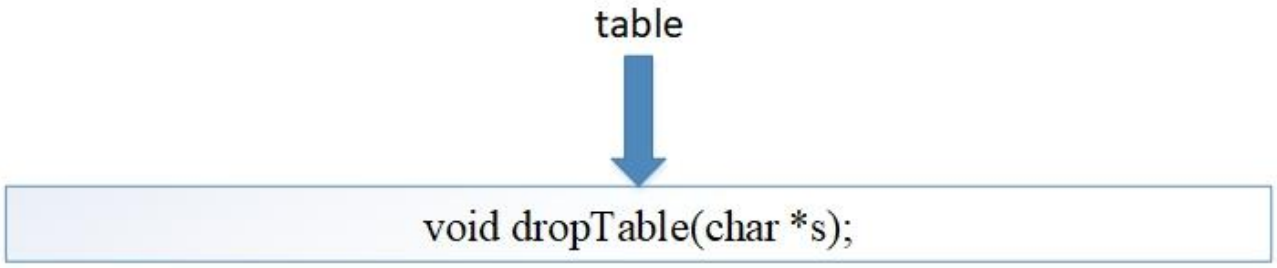

图 21 DROP TABLE 流程图

函数名称：void showTables()

函数说明：删除表

输入参数：无

输出参数：无

执行流程： ①判断是否在一个数据库中，若否，则结束。

②删除该表的 txt 文件。

③从 sys.dat 中删除该表的信息。

（8）INSERT

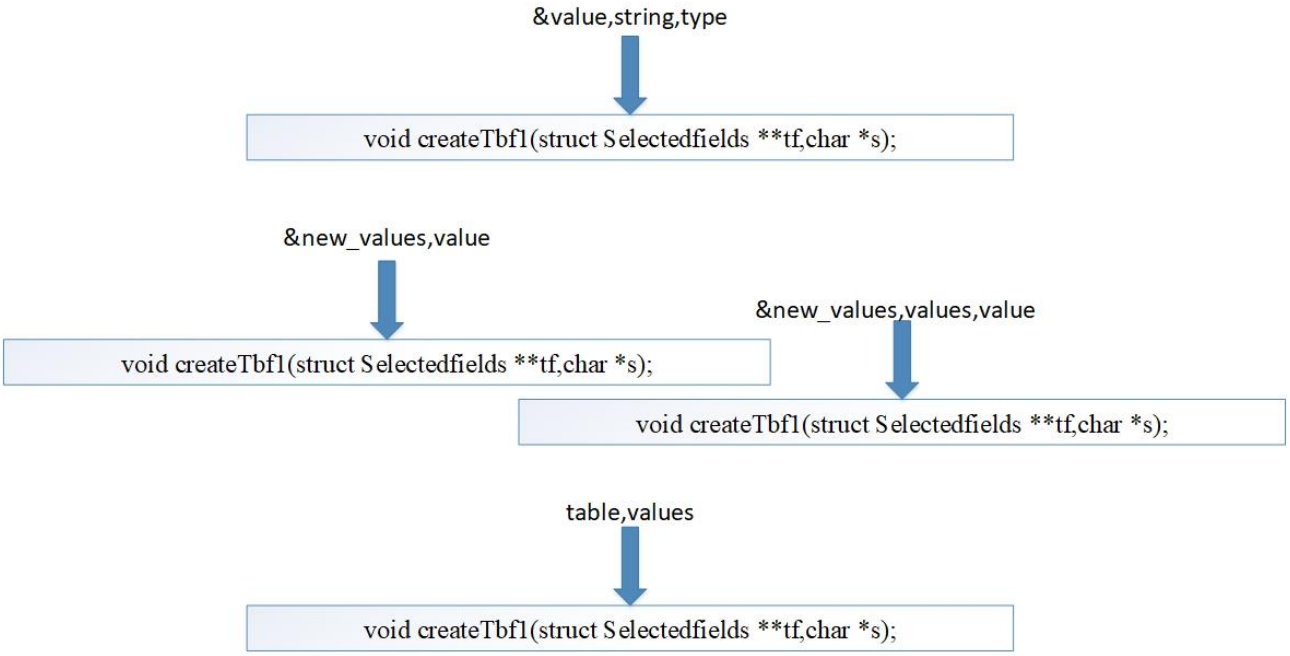

图 22 INSERT 流程图

整体流程：

①获得元素。

②生成元组。

③将元祖插入到表中。

函数名称：void getValue(struct Values **v,char *s,int type)

函数说明：获得元素

输入参数：保存结果的指针地址，值的字符串，类型

输出参数：无

执行流程：用类型和字符串生成元素，将地址赋给 v。

函数名称：void addValue(struct Values **v1,struct Values *v2)

函数说明：单个元素生成元组

输入参数：保存结果的指针地址，元素

输出参数：无

执行流程：将元素地址赋给 v1。

函数名称：void addValues(struct Values **v1,struct Values *v2,struct Values *v3)

函数说明：单个元素和元组生成元组

输入参数：保存结果的指针地址，元组，元素

输出参数：无

执行流程：将元素串在元组后面，并把元组地址赋给 v1。

函数名称：void insertValues(char *s,struct Values *val)

函数说明：将元组插入到表中

输入参数：表名，元组

输出参数：无

执行流程： ①判断是否在一个数据库中，若否，则结束。

②判断该表是否已存在，若不存在，则结束。

③读取表中各列类型，若元组个数不符或类型不符，则结束。

④将数据写入 txt 文件中。

（9）SELECT

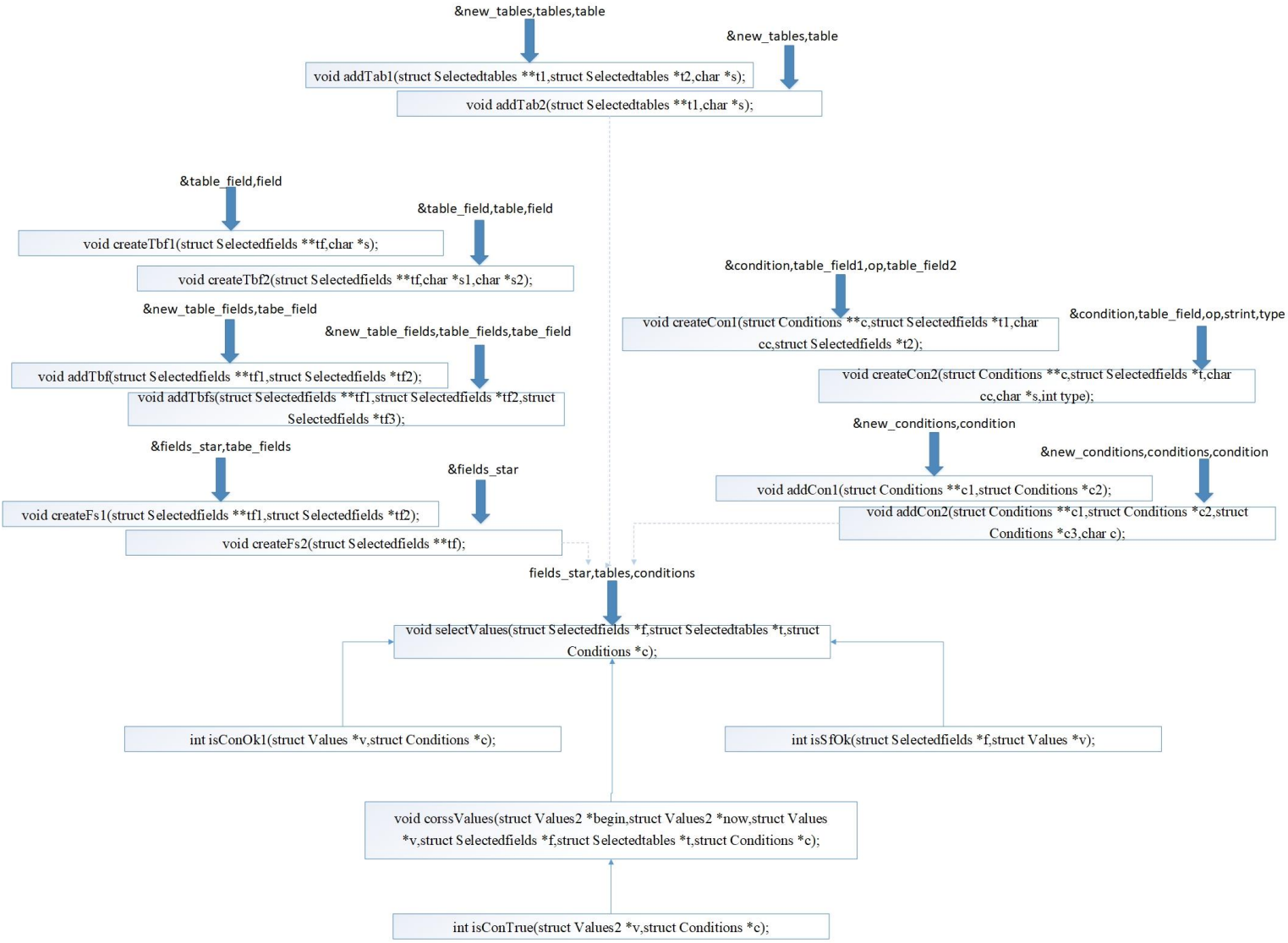

图 23 SELECT 流程图

整体流程：

① 构造累计选择列。

a) 用列名或列名和表明构造选择的列。

b) 构造累计选择列。

c) 合并累计选择列和*的结构。

② 构造累计选择表。

a) 用表或累计表构造累计表。

③ 构造累计选择条件。

a) 用比较符号构造选择条件。

b) 用 and 或 or 构造累计选择条件。

④ 用累计选择列、累计选择表、累计选择条件查询结果。

a) 判断累计选择表中的选择表是否都存在。

b) 判断累计选择列是否符合要求。

c) 判断累计查询条件是否符合要求。

d) 累计选择表做叉乘。

e) 对叉乘的每一元素，判断是否符合累计查询结果。

f) 若符合，输出累计选择列中元素对应的值。

函数名称： void addTab1(struct Selectedtables **t1,struct Selectedtables *t2,char *s)

函数说明：单个表和累计选择表生成累计选择表

输入参数：保存结果的指针地址，累计表，表名

输出参数：无

执行流程：把单个选择表串在累计选择表之后，并将累计表的地址赋给 t1。

函数名称：void addTab2(struct Selectedtables **t1,char *s)

函数说明：单个表生成累计选择表

输入参数：保存结果的指针地址，表名

输出参数：无

执行流程：用单个选择表构建累计选择表结构，并将地址赋给 t1。

函数名称：void createTbf1(struct Selectedfields **tf,char *s)

函数说明：列名生成选择列结构

输入参数：保存结果的指针地址，列名

输出参数：无

执行流程：用列名构建选择列结构，并将地址赋给 tf。

函数名称：void createTbf2(struct Selectedfields **tf,char *s1,char *s2)

函数说明：表名列名生成选择列结构

输入参数：保存结果的指针地址，表名，列名

输出参数：无

执行流程：用表名和列名构建选择列结构，并将地址赋给 tf。

函数名称：addTbf(struct Selectedfields **tf1,struct Selectedfields *tf2)

函数说明：单个选择列生成累计选择列

输入参数：保存结果的指针地址，列名

输出参数：无

执行流程：用单个选择列构建累计选择列结构，并将地址赋给 tf1。

函数名称：void addTbfs(struct Selectedfields **tf1,struct Selectedfields *tf2,struct Selectedfields *tf3)

函数说明：选择列和累计选择列生成累计选择列

输入参数：保存结果的指针地址，累计选择列，列名

输出参数：无

执行流程：用选择列和累计选择列构建累计选择列结构，并将地址赋给 tf1。

函数名称：void createFs(struct Selectedfields **tf1,struct Selectedfields *tf2)

函数说明：赋值累计选择列

输入参数：保存结果的指针地址，累计选择列

输出参数：无

执行流程：将累计选择列 tf2 的地址赋给 tf1。

函数名称：void createCon1(struct Conditions **c,struct Selectedfields *t1,char cc,struct Selectedfields *t2)

函数说明：两个变量构建判断条件

输入参数：保存结果的指针地址，变量选择列，比较符，变量选择列

输出参数：无

执行流程：用两个变量和比较符构建选择条件结构体，并将地址赋给 c。

函数名称：void createCon2(struct Conditions **c,struct Selectedfields *t,char cc,char *s,int type)

函数说明：左变量和右值构建判断条件

输入参数：保存结果的指针地址，变量选择列，比较符，值字符串，类型

输出参数：无

执行流程：用左变量、右值和比较符构建判断条件结构体，并将地址赋给 c。

函数名称：void addCon1(struct Conditions **c1,struct Conditions *c2)

函数说明：赋值判断条件

输入参数：保存结果的指针地址，判断条件

输出参数：无

执行流程：将判断条件 c2 的地址赋给 c1。

函数名称：void addCon2(struct Conditions **c1,struct Conditions *c2,struct Conditions *c3,char c)

函数说明：合并两个判断条件构建判断条件

输入参数：保存结果的指针地址，判断条件，判断条件，关联符号

输出参数：无

执行流程：用两个判断条件和关联符号构造判断条件，并将地址赋给 c1。

函数名称：int isConOk1(struct Values *v,struct Conditions *c)

函数说明：判断判断条件是否符合规则（变量存在且唯一、类型匹配）

输入参数：选择表中的所有列，判断条件

输出参数：0 或 1

执行流程： ①若判断条件为空，则返回 1。

②若关联符为 and 或 or，判断左右节点，左右均符合则返回 1。

③若左节点表名为空，则在所有列中找左节点的变量名，若找不到，则返回 0；若找到多个，则返回 0；否则确定左节点表名。若左节点表名不为空，则在所有列中找左节点对应的表名和变量名，若找不到，则返回 0。

④若右节点类型为变量，则对右节点做相同于④的操作，若未返回 0，比较左节点与找到的列的类型，若类型不匹配，则返回 0；若右节点类型为数值，比较左右节点的类型，若类型不匹配，则返回 0。其他情况返回 1。

函数名称：int isSfOk(struct Selectedfields *f,struct Values *v)

函数说明：判断累计选择列是否符合规则（变量存在且唯一）

输入参数：累计选择列，选择表中的所有列

输出参数：0 或 1

执行流程： ①循环判断累计选择列中的每一选择列

②若选择列中的表名为空，则在所有列中找选择列的变量名，若找不到，则返回 0；若找到多个，则返回 0。

③若选择列中的表名不为空，则在所有列中找选择列对应的表名和变量名，若不存在，则返回 0。其它情况返回 1。

函数名称：isConTrue(struct Values2 *v,struct Conditions *c)

函数说明：判断叉乘结果的一行是否符合累计判断条件

输入参数：叉乘结果的一行，累计判断条件

输出参数：0 或 1

执行流程： ①若累计判断条件为空，则返回 1。

②若关联符号为 and，则判断左右节点，左右均符合则返回 1。

③若关联符号为 or，则判断左右节点，其中一个符合则返回 1。

④在叉乘结果中找左节点的变量对应的类型和值。

⑤若右节点的类型为变量，则在叉乘结果中找右节点的变量对应的类型和值。

⑥根据对应的类型和比较操作符，返回结果。

函数名称：void corssValues(struct Values2 *begin,struct Values2 *now,struct Values *v,struct Selectedfields *f,struct Selectedtables *t,struct Conditions *c)

函数说明：累计选择表的叉乘递归，最后一层打印所有符合条件的结果

输入参数：叉乘过程或结果中每个变量值的结构体的首，叉乘过程或结果中每个变量值的结构体的当前位置，选择表中的所有列，累计选择列，累计选择表的当前位置，累计判断条件

输出参数：无

执行流程： ①若累计选择表当前位置不为空，则在叉乘结果的当前节点不断加入累计选择表当前位置对应的文件中的列。加入完毕后，累计选择表当前位置向前移动，继续递归调用该函数。

②若累计选择表当前位置为空，则判断当前叉乘结果的该行是否符合累计判断条件，若符合，则判断累计选择列是否为空，若为空，则输出叉乘结果改行的每个变量的值；否则按顺序输出在累计选择列中存在的变量的值。

函数名称： void selectValues(struct Selectedfields *f,struct Selectedtables *t,struct Conditions *c)

函数说明：select 主函数

输入参数：累计选择列，累计选择表，累计判断条件

输出参数：无

执行流程： ①判断当前是否在一个数据库中，若否，则结束。

②判断累计选择表中的选择表是否都存在，若否，则结束。

③提取累计选择表中的所有列。

④判断累计选择列和累计选择条件中的变量是否都存在且唯一、累计判断条件左右节点类型是否相符，若有一不符合，则结束。

⑤将累计选择表叉乘，对结果的每一行，判断是否符合累计判断条件，若符合，则打印。

（10） UPDATE

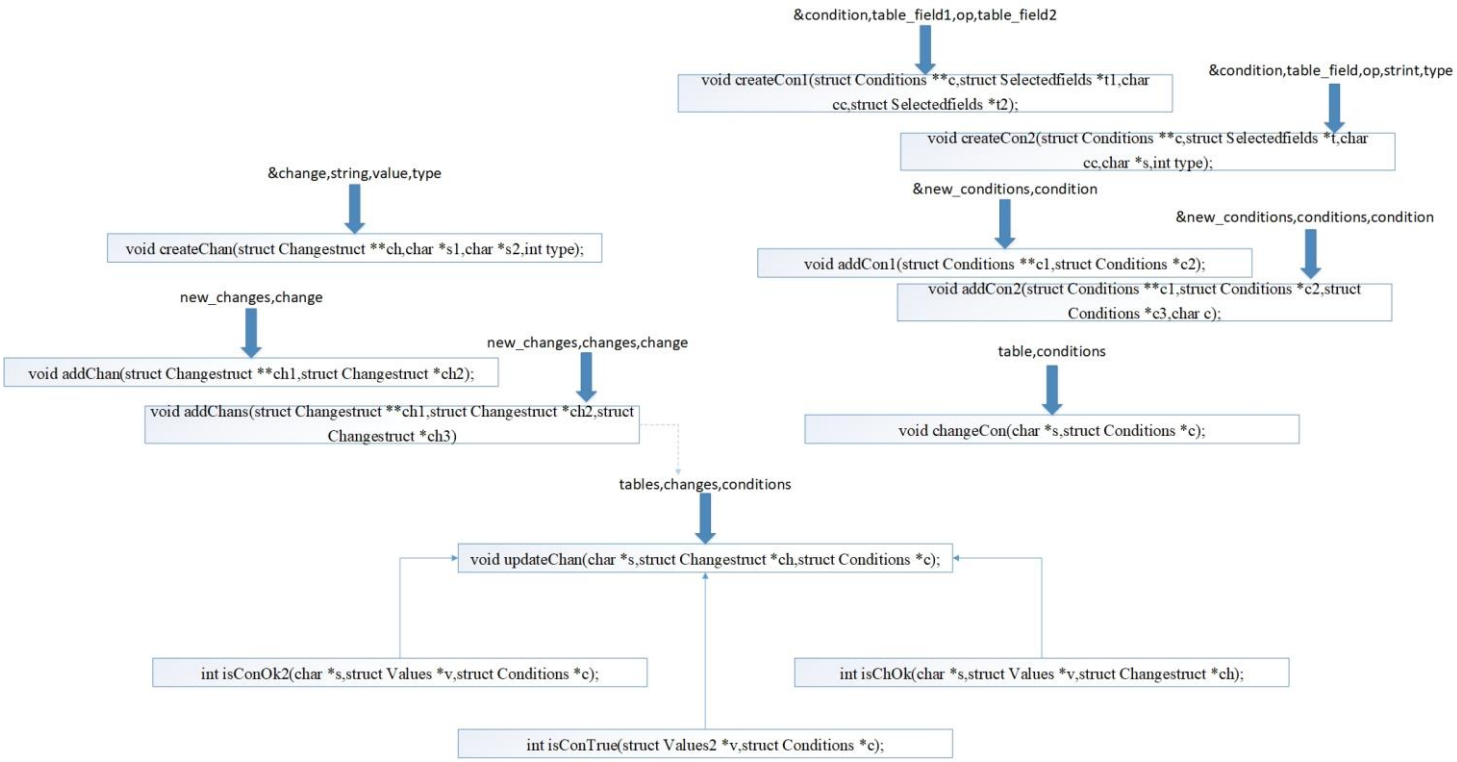

图 24 UPDATE 流程图

整体流程：

① 构造累计更新值。

a) 构造更新列结构体。

b) 构造累计更新值列构体。

② 构造累计选择条件。

a) 用比较符号构造选择条件。

b) 用 and 或 or 构造累计选择条件。

③ 用累计选择条件判断表中每一行，若符合，用累计更新列更新对应值。

函数名称：void createChan(struct Changestruct **ch,char *s1,char *s2,int type)

函数说明：构造更新列

输入参数：保存结果的指针地址，列名，值字符串，类型

输出参数：无

执行流程：用列名和值构造更新列，并将地址赋给 ch。

函数名称：void addChan(struct Changestruct **ch1,struct Changestruct *ch2)

函数说明：更新列构造累计更新列

输入参数：保存结果的指针地址，更新列

输出参数：无

执行流程：将更新列的地址赋给 ch。

函数名称： void addChans(struct Changestruct **ch1,struct Changestruct *ch2,struct Changestruct *ch3)

函数说明：更新列和累计更新列构造累计更新列

输入参数：保存结果的指针地址，更新列，累计更新列

输出参数：无

执行流程：将更新列串在累计更新列之后，并将累计更新列的地址赋给 ch。

函数名称：isConOk2(char *s,struct Values *v,struct Conditions *c)

函数说明：判断累计判断条件是否符合规则（表名一致、变量存在、类型匹配）

输入参数：表名，表中所有列，累计判断条件

输出参数：0 或 1

执行流程： ①若关联符为 and 或 or，判断左右节点，左右均符合则返回 1。

②若表名与左节点表名不同，则返回 0。

③若表中列中不存在左节点的变量，则返回 0。

④若右节点类型为变量，对其做同③操作，若未返回 0，则比较左节点与右节点的类型，若不同则返回 0。

⑤其他情况返回 1。

函数名称：int isChOk(char *s,struct Values *v,struct Changestruct *ch)

函数说明：判断累计更新列是否符合规则（变量存在、类型匹配）

输入参数：表名，表中所有列，累计更新列

输出参数：0 或 1

执行流程： ①对累计更新列中的每一个更新列做以下操作。

②若更新列中的变量在表中所有列中不存在，则返回 0。

③若更新列中的变量和更新值的类型不一致，则返回 0。

④若更新列均未返回 0，则返回 1。

函数名称：void changeCon(char *s,struct Conditions *c)

函数说明：填充累计判断条件中的表名

输入参数：表名，累计判断条件

输出参数：0 或 1

执行流程： ①若关联符为 and 或 or，则修改左右节点。

②用表名填充将表名为空的节点。

函数名称：void updateChan(char *s,struct Changestruct *ch,struct Conditions *c)

函数说明：update 主函数

输入参数：表名，累计更新列，累计判断条件

输出参数：无

执行流程： ①判断当前是否在一个数据库中，若否，则结束。

②判更新的表是否存在，若否，则结束。

③提取表中的所有列，判断累计更新列和累计判断条件是否符合要求，若存在不符合，则结束。

④对表中的每一行，判断累计判断条件是否成立，若成立则用累计更新列更新这一行；若不成立，则不改变这一行。

（11） DELETE

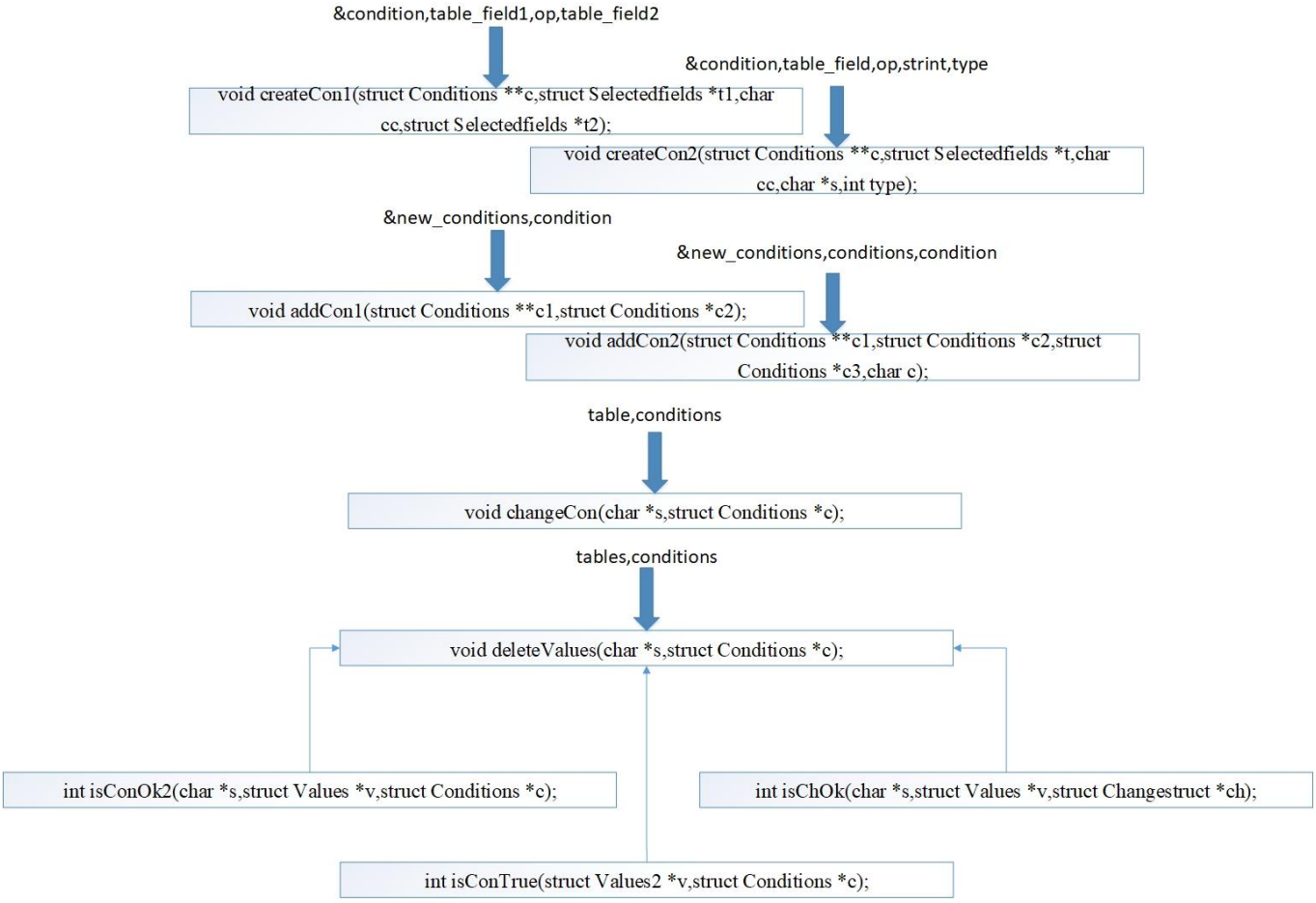

图 25 DELETE 流程图

整体流程：

① 构造累计判断条件。

a) 用比较符号构造选择条件。

b) 用 and 或 or 构造累计选择条件。

② 用累计判断条件判断表中每一行，若符合，则删除。

函数名称：void deleteValues(char *s,struct Conditions *c)

函数说明：delete 主函数

输入参数：表名，累计更新列，累计判断条件

输出参数：无

执行流程： ①判断当前是否在一个数据库中，若否，则结束。

②判断更新的表是否存在，若否，则结束。

③提取表中的所有列，判断累计判断条件是否符合要求，若否，则结束。

④对表中的每一行，若累计判断条件成立，则删除。

（12） EXIT

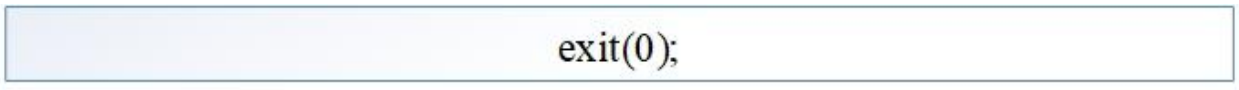

图 26 EXIT 流程图

2.4 功能测试

（1）CREATE DATABASE zhu1;

（2）CREATE DATABASE zhu2;

（3）SHOW DATABASES;

（4）CREATE DATABASE zhu2; //创建已存在的数据库，报错

（5）DROP DATABASE zhu2;

（6）SHOW DATABASES;

```
SQL [1]: CREATE DATABASE zhu1;
create database successed.
SQL [2]: CREATE DATABASE zhu2;
create database successed.
SQL [3]: SHOW DATABASES;
zhu1
zhu2
SQL [4]: CREATE DATABASE zhu2;
database already exist.
SQL [5]: DROP DATABASE zhu2;
drop database successed.
SQL [6]: SHOW DATABASES;
zhu1
```

图 27 功能测试 1

（7）USE DATABASE zhu1;

（8）CREATE TABLE class(cno INT,sno INT);

（9）CREATE TABLE stu(sname CHAR(20),sno INT,age INT);

（10） CREATE TABLE cs(sno INT,cname CHAR(20),grade INT);

（11） CREATE TABLE xxx(aaa int);

（12） CREATE TABLE class(aa INT); //创建已存在的表，报错

（13） CREATE TABLE xx(name int,name char(10)); //创建表有同名列，报错

（14） SHOW TABLES;

（15） DROP TABLE xxx;

（16） DROP TABLE yyy; //删除不存在的表，报错

（17） SHOW TABLES;

```
SQL [7]: USE DATABASE zhu1;
now in database zhu1.
SQL [8]: CREATE TABLE class(cno INT,sno INT);
create table successed.
SQL [9]: CREATE TABLE stu(sname CHAR(20),sno INT,age INT);
create table successed.
SQL [10]: CREATE TABLE cs(sno INT,cname CHAR(20),grade INT);
create table successed.
SQL [11]: CREATE TABLE xxx(aaa int);
create table successed.
SQL [12]: CREATE TABLE class(aa INT);
table already exist.
SQL [13]: CREATE TABLE xx(name int,name char(10));
create table error.
SQL [14]: SHOW TABLES;
class
stu
cs
xxx
SQL [15]: DROP TABLE xxx;
drop table successed.
SQL [16]: DROP TABLE yyy;
table not exist.
SQL [17]: SHOW TABLES;
class
stu
cs
```

图 28 功能测试 2

（18） INSERT INTO class(1,1503001);

（19） INSERT INTO class(1,1503002);

（20） INSERT INTO class(2,1503003);

（21） INSERT INTO class(2,1503004);

（22） INSERT INTO stu('name1',1503001,15);

（23） INSERT INTO stu('name2',1503002,16);

（24） INSERT INTO stu('name3',1503003,17);

（25） INSERT INTO stu('name4',1503004,18);

（26） INSERT INTO cs(1503001,'math',99);

（27） INSERT INTO cs(1503001,'cs',88);

（28） INSERT INTO cs(1503001,'che',77);

（29） INSERT INTO cs(1503002,'math',78);

（30） INSERT INTO cs(1503003,'cs',50);

（31） INSERT INTO xxx(1,1); //插入元组到不存在的表，报错

（32） INSERT INTO stu(11,101,11); //插入类型不匹配，报错

（33） INSERT INTO stu('name1',101,11,12); //插入数量不匹配，报错

```
SQL [18]: INSERT INTO class(1,1503001);
insert successed.
SQL [19]: INSERT INTO class(1,1503002);
insert successed.
SQL [20]: INSERT INTO class(2,1503003);
insert successed.
SQL [21]: INSERT INTO class(2,1503004);
insert successed.
SQL [22]: INSERT INTO stu('name1',1503001,15);
insert successed.
SQL [23]: INSERT INTO stu('name2',1503002,16);
insert successed.
SQL [24]: INSERT INTO stu('name3',1503003,17);
insert successed.
SQL [25]: INSERT INTO stu('name4',1503004,18);
insert successed.
SQL [26]: INSERT INTO cs(1503001,'math',99);
insert successed.
SQL [27]: INSERT INTO cs(1503001,'cs',88);
insert successed.
SQL [28]: INSERT INTO cs(1503001,'che',77);
insert successed.
SQL [29]: INSERT INTO cs(1503002,'math',78);
insert successed.
SQL [30]: INSERT INTO cs(1503003,'cs',50);
insert successed.
SQL [31]: INSERT INTO xxx(1,1);
table not exist.
SQL [32]: INSERT INTO stu(11,101,11);
insert error.
SQL [33]: INSERT INTO stu('name1',101,11,12);
insert error.
```

图 29 功能测试 3

（34） SELECT * FROM class;

（35） SELECT * FROM stu;

（36） SELECT * FROM cs;

（37） SELECT * FROM cs WHERE cs.sno = 1503001;

（38） SELECT stu.sno FROM class,stu WHERE class.cno = 1 and class.sno = stu.sno ;

（39） SELECT * FROM stu WHERE age > 16 and age != 20;

（40） SELECT * FROM stu WHERE sno = 1503004 or age > 10 and age < 15;

（41） SELECT class.cno,stu.sno,stu.sname,stu.age,cs.cname,cs.grade FROM class,stu,cs WHERE class.sno = stu.sno and stu.sno = cs.sno;

```
SQL [35]: SELECT * FROM stu;
(stu.sname, stu.sno, stu.age)
(name1, 1503001, 15)
(name2, 1503002, 16)
(name3, 1503003, 17)
(name4, 1503004, 18)
SQL [36]: SELECT * FROM cs;
(cs.sno, cs.cname, cs.grade)
(1503001, math, 99)
(1503001, cs, 88)
(1503001, che, 77)
(1503002, math, 78)
(1503003, cs, 50)
SQL [37]: SELECT * FROM cs WHERE cs.sno = 1503001;
(cs.sno, cs.cname, cs.grade)
(1503001, math, 99)
(1503001, cs, 88)
(1503001, che, 77)
SQL [37]: SELECT * FROM cs WHERE cs.sno = 1503001;
(cs.sno, cs.cname, cs.grade)
(1503001, math, 99)
(1503001, cs, 88)
(1503001, che, 77)
SQL [38]: SELECT stu.sno FROM class, stu WHERE class.cno = 1 and class.sno = stu.sno
(stu.sno)
(1503001)
(1503002)
SQL [39]: SELECT * FROM stu WHERE age > 16 and age != 20;
(stu.sname, stu.sno, stu.age)
(name3, 1503003, 17)
(name4, 1503004, 18)
SQL [40]: SELECT * FROM stu WHERE sno = 1503004 or age > 10 and age < 15;
(stu.sname, stu.sno, stu.age)
(name4, 1503004, 18)
SQL [41]: SELECT class.cno, stu.sno, stu.sname, stu.age, cs.cname, cs.grade FROM class, stu, cs WHERE class.sno = stu.sno and s
tu.sno = cs.sno;
(class.cno, stu.sno, stu.sname, stu.age, cs.cname, cs.grade)
(1, 1503001, name1, 15, math, 99)
(1, 1503001, name1, 15, cs, 88)
(1, 1503001, name1, 15, che, 77)
(1, 1503002, name2, 16, math, 78)
(2, 1503003, name3, 17, cs, 50)
```

图 30 功能测试 4

（42） SELECT xx FROM stu; //选择的列不存在，报错

（43） SELECT sno FROM stu,cs; //存在同名列，未指定表，报错

（44） SELECT aa.sno FROM stu; //选择的选择的表不匹配，报错

（45） SELECT * FROM stu WHERE aa.sno > 1; //判断的表不存在，报错

（46） SELECT * FROM stu WHERE sname = 1; //判断的类型不匹配，报错

```
SQL [42]: SELECT xx FROM stu;
sleect error.
SQL [43]: SELECT sno FROM stu,cs;
sleect error.
SQL [44]: SELECT aa.sno FROM stu;
sleect error.
SQL [45]: SELECT * FROM stu WHERE aa.sno > 1;
sleect error.
SQL [46]: SELECT * FROM stu WHERE sname = 1;
sleect error.
```

图 31 功能测试 5

（47） SELECT * FROM stu;

（48） UPDATE stu SET age = 20 WHERE sname = 'name1';

（49） UPDATE stu SET sno = 11111,age = 22 WHERE sname = 'name2';

（50） SELECT * FROM stu;

（51） UPDATE stu SET age = '10' WHERE sname = 'name1';//更新类型不匹配

（52） UPDATE stu SET xx = 10 WHERE sname = 'name1';//更新列不存在

```
SQL [47]: SELECT * FROM stu;
(stu.sname, stu.sno, stu.age)
(name1, 1503001, 15)
(name2, 1503002, 16)
(name3, 1503003, 17)
(name4, 1503004, 18)
SQL [48]: UPDATE stu SET age = 20 WHERE sname = 'name1';
update successed.
SQL [49]: UPDATE stu SET sno = 11111,age = 22 WHERE sname = 'name2';
update successed.
SQL [50]: SELECT * FROM stu;
(stu.sname, stu.sno, stu.age)
(name1, 1503001, 20)
(name2, 11111, 22)
(name3, 1503003, 17)
(name4, 1503004, 18)
SQL [51]: UPDATE stu SET age = '10' WHERE sname = 'name1';
update error.
SQL [52]: UPDATE stu SET xx = 10 WHERE sname = 'name1';
update error.
```

图 32 功能测试 6

（53） SELECT * FROM stu;

（54） DELETE FROM stu WHERE age < 20;

（55） SELECT * FROM stu;

（56） EXIT;

```
SQL [53]: SELECT * FROM stu;
(stu.sname, stu.sno, stu.age)
(name1, 1503001, 20)
(name2, 11111, 22)
(name3, 1503003, 17)
(name4, 1503004, 18)
SQL [54]: DELETE FROM stu WHERE age < 20;
delete successed.
SQL [55]: SELECT * FROM stu;
(stu.sname, stu.sno, stu.age)
(name1, 1503001, 20)
(name2, 11111, 22)
SQL [56]: EXIT;
Press any key to continue
```

图 33 功能测试 7

2.5 未完成功能

（1）约束 CONSTRAIN

（2）外码 FOREIGNKEY

（3）默认值 DEFAULT

（4）INSERT 选择性插入某些列

（5）修改 ALTER

（6）索引 INDEX

（7）视图 VIEW

（8）权限 GRANT、DENY、REVOKE、AUTHORITY

（9）嵌套查询

2.6 未来实现方案

（1）约束 CONSTRAIN

约束条件在创建表或修改表的时候增减，物理结构见前文，重要的是加了约束之后，插入和更新操作需要先判断一下是否满足条件。

（2）外码 FOREIGNKEY

外码在创建表或修改表的时候增减，物理结构见前文，创建或修改外码时，需要判断外码类型是否相同，以及在被参考的列不存在的时候的异常处理。

（3）默认值 DEFAULT

默认值在创建或修改表的时候增减，物理结构见前文，创建或修改默认值时，需判断默认值的类型与列的类型是否相同。

（4）INSERT 选择性插入某些列

选择性地插入元组，需要将未选择的列置为默认值，若默认值未设置，则为NULL，这会影响判断条件的判断，所以在判断条件处理时需要特殊处理。

（5）修改 ALTER

修改操作可以修改列名、列的类型、约束、外码、默认值，在（1）-（3）实现玩之后，增加这个比较简单。需要特殊处理一下修改后原有值与表冲突的异常情况。

（6）索引 INDEX

索引的语句处理比较简单，见前文的物理结构设计，但是真正实现索引，需要用重新设计元组的存储结构以及元组的存放顺序。若有多个索引，可能需要复制多份；或者采用指针的方式，采用多套不同顺序指针。

（7）视图 VIEW

可以采用一种基本的方法，储存创建视图时的 SQL 语句中的关键值，在使用视图时执行一次查询操作，这种方法在使用视图频繁时，效率比较低；或者将也视为一张表，更新相关表的时候，同时更新视图，这种方法在使用视图不频繁时，会增加额外负担。

（8）权限 GRANT、DENY、REVOKE、AUTHORITY

首先要处理三种权限处理的语句，物理结构见前文。还需要在每次使用数据库时，增加一个用户登录的操作，使每次在数据库的操作都有一个用户，以确定用户对每张表的各种权限，在执行 SQL 语句前，先判断用户权限。

（9）嵌套查询

将查询写成递归的形式，储存每一层嵌套的查询结果，返回给外部一层。最内部的层依然采用从文件读取数据。

2.7 总结

花了比较长的时间终于完成了这次实验内容，这次作业使用了 Lex 和 Yacc 来实现与词法分析和语法分析，节省了很多时间。采用的语言为 C 语言，相比与 C++、JAVA 和 Python，C 语言比较纯粹，但需要画更多的时间去自己造轮子。整体来看，词法规则和递推式的构造比较简单，最困难的是，判断条件的实现和判断处理各种语义错误的情况，需要考虑很多种情况。DEBUG 的过程比较煎熬，需要耐心与细心。

各种功能的实现采用的还是纯暴力的方法，由于使用的是 C 语言，想要优化的话比较花费时间，需要自己构造各种数据结构将各种 O(n)优化到 O(lgn)，临近期末，时间紧迫，故没有实现。

通过这次实验，我对编译原理有了更加深入的理解，深知完成一个完整的编译器是多么的不易。并且，重新拾起了 SQL 语言和 C 语言的指针等各种比较底层的知识，是历练也是成长。

如果以后有机会，希望能够实现一种自己的语言，不求有多强大，只需满足自己日常娱乐就好。