

# **编译原理上机报告**

---

## **《DBMS的设计与实现》**

---

**邮箱：xwHuNichols(a.t.)163.com**

**完成时间：2018年6月11日**

# 1 项目概况

## 1.1 基本目标

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

实现了以下SQL语句及功能：

|    |                 |       |
|----|-----------------|-------|
| 1  | CREATE DATABASE | 创建数据库 |
| 2  | USE DATABASE    | 选择数据库 |
| 3  | CREATE TABLE    | 创建表   |
| 4  | SHOW TABLES     | 显示表名  |
| 5  | INSERT          | 插入元组  |
| 6  | SELECT          | 查询元组  |
| 7  | UPDATE          | 更新元组  |
| 8  | DELETE          | 删除元组  |
| 9  | DROP TABLE      | 删除表   |
| 10 | DROP DATABASE   | 删除数据库 |
| 11 | EXIT            | 退出系统  |

支持数据类型：INT、CHAR(N)

# 2 项目实现方案

## 2.1 逻辑结构与物理结构

### 逻辑结构

在数据库物理设计上，增加了数据库元数据 (database.dat) 和表元数据 (tables.dat)

database.dat

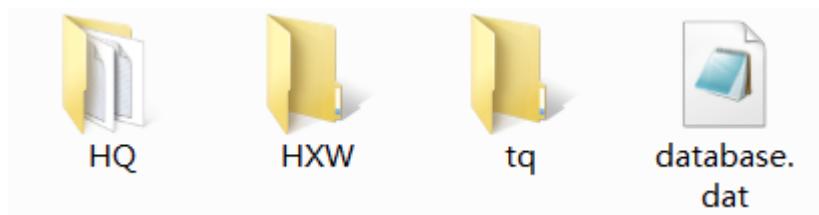
| 数据库名称 |
|-------|
| HXW   |
| TQ    |
| ..... |

tables.dat

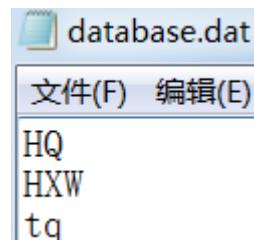
| 表名      | 字段    | 类型    | 长度    |
|---------|-------|-------|-------|
| Student | Sno   | CHAR  | 10    |
| Course  | Cno   | CHAR  | 5     |
| SC      | Grade | INT   | 4     |
| .....   | ..... | ..... | ..... |

## 物理结构

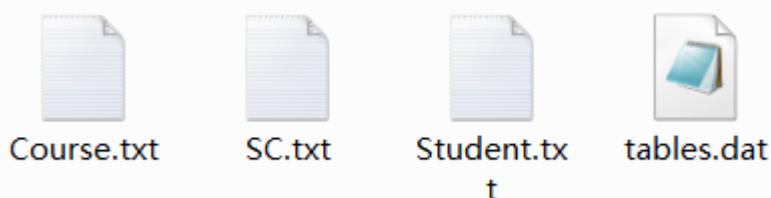
DBMS主目录信息如下：



其中，database.dat中的内容（包含已创建的数据库信息，分别对应相应的子目录）如下：



HQ数据库对应的目录信息（包含Student,Course,SC三个表，分别存放于Student.txt,Course.txt,SC.txt中）如下：



其中，tables.dat中的内容如下：

tables.dat - 记事本

| Student | Sno     | CHAR | 10 |  |
|---------|---------|------|----|--|
| Student | Sname   | CHAR | 10 |  |
| Student | Ssex    | CHAR | 2  |  |
| Student | Sage    | INT  | 4  |  |
| Course  | Cno     | CHAR | 5  |  |
| Course  | Cname   | CHAR | 10 |  |
| Course  | Cpno    | CHAR | 5  |  |
| Course  | Ccredit | INT  | 4  |  |
| SC      | Sno     | CHAR | 10 |  |
| SC      | Cno     | CHAR | 5  |  |
| SC      | Grade   | INT  | 4  |  |

## 优缺点

该数据库保存和读取信息十分方便，也便于根据文件内容调试程序。但该数据库无法存储大量且复杂的数据信息，优化率低下，只适合用于了解编译原理及操作系统的知识。

## 2.2 语法结构与数据结构

### CREATE

#### ① CREATE 语句的产生式语法结构

```

1 | createsql:CREATE TABLE table '(' fieldsdefinition ')' ';' 
2 |           |CREATE DATABASE ID '';

```

#### ② 非终结符数据结构

```

1 //create语法树的类型-----.
2 char *yych; //字面量
3 struct Createfieldsdef *cfdef_var;//字段定义
4 struct fieldType *fT;//type定义
5 struct Createstruct *cs_var;//整个create语句
6 //create语句中的字段定义-----.
7 struct Createfieldsdef{
8     char         *field;      //字段名称
9     enum TYPE    type;       //字段类型
10    int          length;     //字段长度
11    struct Createfieldsdef *next_fdef;//下一段
12 };
13 //type字段定义-----.
14 enum TYPE {CHAR, INT};
15 struct fieldType{
16     enum TYPE    type;       //字段类型
17     int          length;     //字段长度
18 };
19 //create语法树根节点-----.
20 struct Createstruct{
21     char *table;           //基本表名称

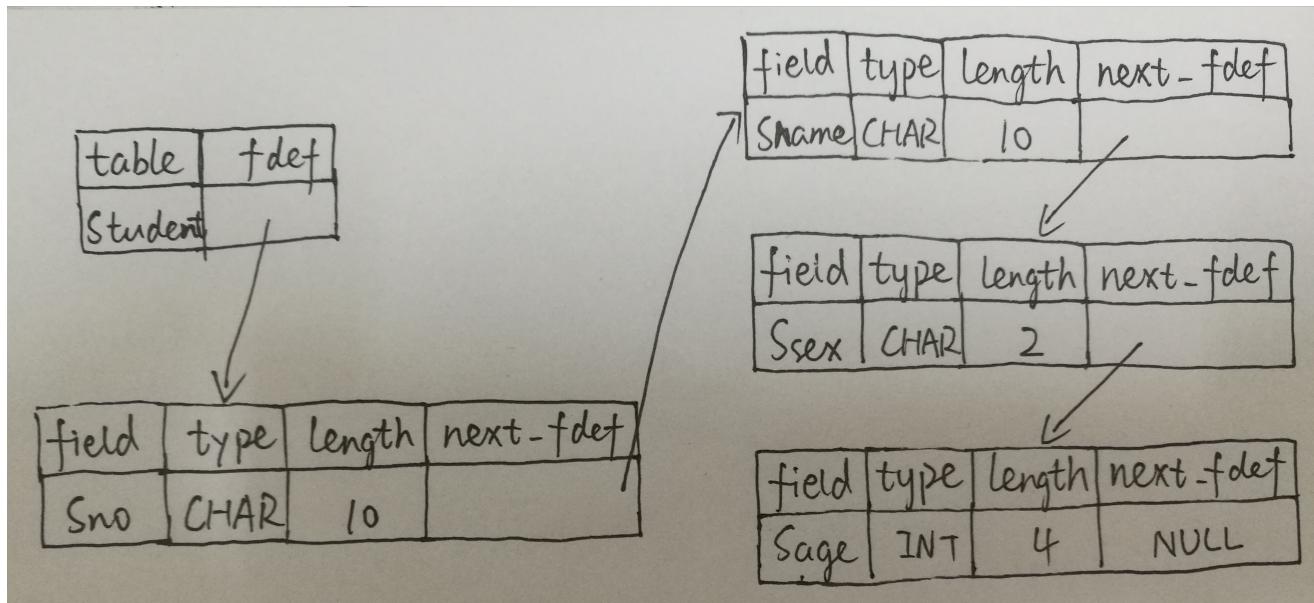
```

```
22     struct Createfieldsdef *fdef; //字段定义  
23 };
```

### ③ 实例

CREATE TABLE Student ( Sno CHAR(10) , Sname CHAR (10) , Ssex CHAR (2) , INT ) ;

对应的数据结构如下图：



## USE

### ① USE 语句的产生式语法结构

```
1 | usesql:USE ID ' ; '
```

### ② 实例

USE HQ;

## SHOW

### ① SHOW 语句的产生式语法结构

```
1 | showsql:SHOW DATABASE ' ; '  
2 | | SHOW TABLES ' ; '
```

### ② 实例

SHOW TABLES;

## SELECT

### ① SELECT 语句的产生式语法结构

```
1 | selectsql:SELECT fields_star FROM tables ' ; '  
2 | | | SELECT fields_star FROM tables WHERE conditions ' ; '
```

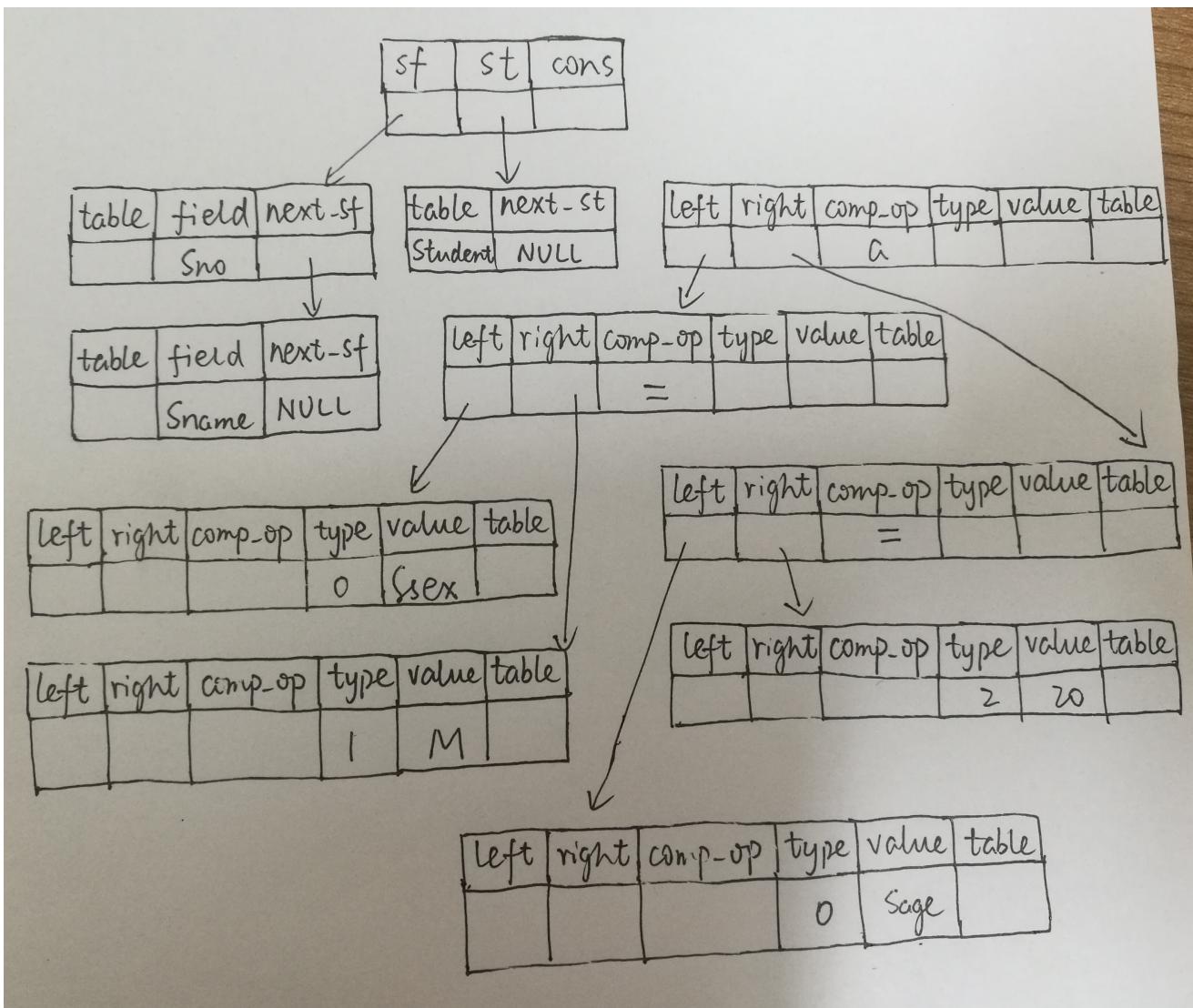
## ② 非终结符数据结构

```
1 //select语法树的类型-----
2     char op;                                //运算符
3     struct Selectedfields *sf_var;          //所选字段
4     struct Selectedtables *st_var;          //所选表格
5     struct Conditions *cons_var;           //条件语句
6     struct Selectstruct *ss_var;            //整个select语句
7 //select条件-----
8 struct Conditions{
9     struct Conditions *left; //左部条件
10    struct Conditions *right; //右部条件
11    char comp_op;             /* 'a'是and, 'o'是or, '<' , '>' , '=' , '!='
12    int type;                /* 0是字段, 1是字符串, 2是整数 */
13    char *value;              /* 根据type存放字段名、字符串或整数 */
14    char *table;               /* NULL或表名 */
15 };
16 //select语句中选中的字段-----
17 struct Selectedfields{
18     char *table;                  //字段所属表
19     char *field;                 //字段名称
20     struct Selectedfields *next_sf; //下一个字段
21 };
22 //select语句中选中的表-----
23 struct Selectedtables{
24     char *table;                 //基本表名称
25     struct Selectedtables *next_st; //下一个表
26 };
27 //select语法树的根节点-----
28 struct Selectstruct{
29     struct Selectedfields *sf;    //所选字段
30     struct Selectedtables *st;   //所选基本表
31     struct Conditions *cons;   //条件
32 };
```

## ③ 实例

SELECT Sno,Sname FROM Student WHERE Ssex='M' AND Sage=20;

对应的数据结构如下图：



## INSERT

### ① INSERT 语句的产生式语法结构

```

1 | insertsql:INSERT INTO table VALUES '(' values ')' ' ';
2 |           |INSERT INTO table '(' values ')' VALUES '(' values ')' ' ';

```

### ② 非终结符数据结构

```

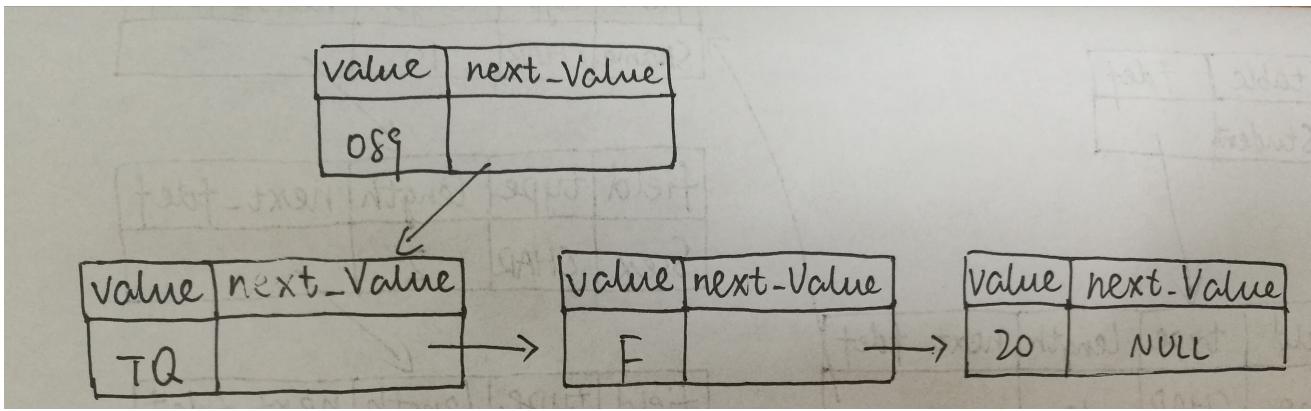
1 //insert的语法树的类型-----
2     struct insertValue *is_val;
3 //insert语法树根节点-----
4 struct insertValue {
5     char *value;           //插入值
6     struct insertValue *next_Value; //下一插入值
7 };

```

### ③ 实例

`INSERT INTO Student VALUES ( '089' , 'TQ' , 'F' , 21 );`

对应的数据结构如下图：



## DELETE

### ① DELETE 语句的产生式语法结构

```

1 | deletesql:DELETE FROM table ';'
2 |           |DELETE FROM table WHERE conditions ;'

```

### ② 实例

DELETE FROM Student;

## DROP

### ① DROP 语句的产生式语法结构

```

1 | dropsql:DROP TABLE ID ';'
2 |           |DROP DATABASE ID ;'

```

### ② 实例

DROP TABLE Student;

## UPDATE

### ① UPDATE 语句的产生式语法结构

```

1 | updatesql:UPDATE table SET sets ';'
2 |           |UPDATE table SET sets WHERE conditions ;'

```

### ② 非终结符数据结构

```

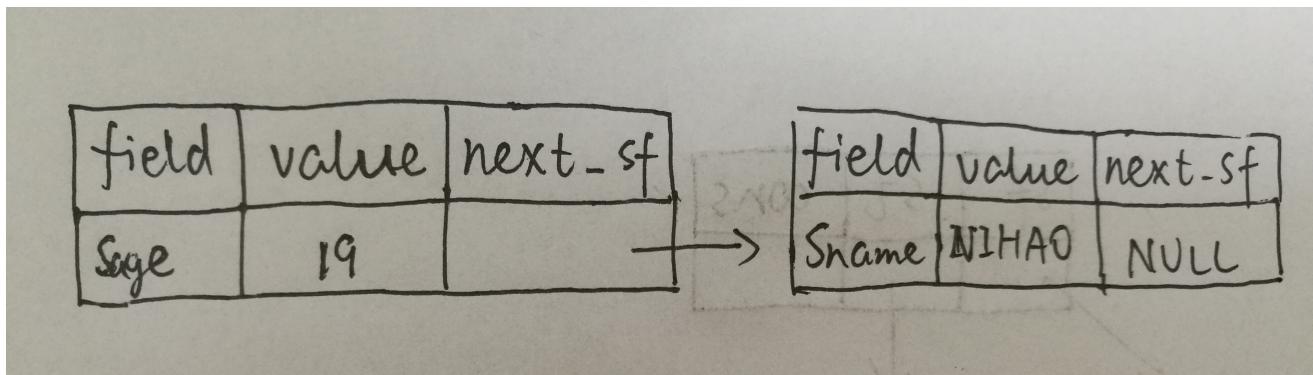
1 //update的语法树的类型-----
2     struct Setstruct *s_var;
3     //set语法树根节点-----
4 struct Setstruct{
5     char *field;           //需更改字段
6     char *value;           //更改值
7     struct Setstruct *next_sf; //需更改下一字段
8 };

```

### ③ 实例

```
UPDATE Student SET Sage=19,Sname='NIHAO';
```

对应的数据结构如下图：



EXIT

① EXIT 语句的产生式语法结构

```
1 | exitsql:EXIT ';
```

② 实例

```
EXIT;
```

## 2.3 执行流程

CREATE

```
void createDB()
```

① 函数说明：创建指定名字的数据库文件夹

② 输入参数：无

③ 输出参数：无

④ 执行流程：用户输入创建数据库的SQL语句，调用mkdir()函数在当前工作目录下的sql文件夹内创建用户指定的数据库文件夹。

```
void createTable(struct Createstruct *cs_root)
```

① 函数说明：创建指定名字的表文件(.txt)

② 输入参数： struct Createstruct \*cs\_root: create语法树的数据结构

③ 输出参数：无

④ 执行流程：用户输入创建表的SQL语句，在选定的数据库目录下创建指定的表文件(.txt)，并且将创建表的字段内容添加到元数据中 ( tables.dat )。

USE

```
void useDB()
```

① 函数说明：进入指定数据库文件夹内

② 输入参数：无

③ 输出参数：无

④ 执行流程：用户输入使用数据库的SQL语句，调用\_chdir( )函数将工作目录转到指定数据库的目录下。

## SHOW

### void getDB()

① 函数说明：获得所有已创建数据库的名字

② 输入参数：无

③ 输出参数：无

④ 执行流程：用户输入展示数据库的SQL语句，从元数据 ( database.dat ) 中读出并打印所有已创建的数据库的名字。

### void getTable()

① 函数说明：获得所有已创建表的内容

② 输入参数：无

③ 输出参数：无

④ 执行流程：用户输入展示表的SQL语句，从选定的数据库的元数据 ( tables.dat ) 中读出所有已创建表的内容。

## SELECT

### void selectCdt(struct Selectedfields \*fieldRoot, struct Selectedtables \*tableRoot, struct Conditions \*conditionRoot)

① 函数说明：打印指定表内所有满足条件的元组信息

② 输入参数： struct Selectedfields \*fieldRoot 所查询的字段的根节点

struct Selectedtables \*tableRoot 所查询的表的根节点

struct Conditions \*conditionRoot 查询条件的根节点

③ 输出参数：无

④ 执行流程： 用户输入带条件查询的SQL语句，将所有需要查询的表的所有元组做笛卡尔乘积，相当于合并成一张大表，然后每次将一行元组值传入Condition( )函数内判断是否满足条件，返回1满足条件打印相关信息，返回0不满足条件。

### void selectNoCdt(struct Selectedfields \*fieldRoot, struct Selectedtables \*tableRoot)

① 函数说明：打印指定表内所有元组信息

② 输入参数： struct Selectedfields \*fieldRoot 所查询的字段的根节点

struct Selectedtables \*tableRoot 所查询的表的根节点

③ 输出参数：无

④ 执行流程： 用户输入不带条件的SQL语句，若查询字段的根节点为NULL，说明需要查询表内所有信息，将字段值以及所有元组值存放于数组中并打印；若查询字段为特定字段，将特定字段和特定字段下的元组值存放于数组中，并打印。

```
int CDTSearch(struct Conditions *conditionRoot, int totField, char allField[][100], char value[][100])
```

① 函数说明：递归函数，判断当前输入的元组是否满足condition条件

② 输入参数：struct Conditions \*conditionRoot 查询条件根节点

int totField 所有字段个数

char allField[][100] 保存所有字段值

char value[][100] 保存当前行的元组值

③ 输出参数：返回1，表示当前输入的元组满足条件；返回0，不满足

④ 执行流程：将条件根节点，所有字段个数，所有字段值及元组值传入此函数，该函数根据条件根节点所在的链表判断（若conditions的判断符是'a'，则递归return (conditionRoot->left, ....)&&(conditionRoot->Right, ....)；若conditions的判断符是'o'，则递归return (conditionRoot->left, ....)|||(conditionRoot->Right, ....)）：若condition左中右部分分别是字段、'=、字段，则是判断同字段下的属性值是否相同，在函数内先匹配到左部的table1.field1，然后匹配右部的table2.field2，再判断各自字段下的属性值是否相同，返回相应的值；若condition左中右部分分别是字段、'=|>|<|'!、字面量，则判断各自字段下的属性值是否满足条件，并返回相应的值。

```
void hanle(int read_line[],int count_value[],int all_table,int index)
```

① 函数说明：笛卡尔乘积算法

② 输入参数：int read\_line[ i ] 当前所读到表 i 的行号，从1开始

int count\_value[ i ] 当前表 i 的最大行号

int all\_table 表格个数

int index 当前所指的表，从0开始

③ 输出参数：无

④ 执行流程：

I 将最后一个表当前所读到的行号加1；

II 判断当前读到的行号是否大于最大行号，大于则表示当前表读完，将read\_line[ index ]=1,重新指向第一行将index--，让它指向下一个表，使read\_line[ index ]++,向下移一行；

II.1 若insex>=0，说明还有表未读完，则进行递归

II.2 将index重新指向最后一个表

```
void freeCdt(struct Conditions *conditionRoot)
```

① 函数说明：释放创建的条件根节点指针

② 输入参数：struct Conditions \*conditionRoot 条件根节点

③ 输出参数：无

④ 执行流程：若当前节点不为空，让当前根节点指向它的下一节点，释放当前节点。

## INSERT

```
void insertOrder(char *tableName,struct insertValue *values)
```

① 函数说明：按建表时字段的既定顺序插入元组

② 输入参数：char \*tableName 字符指针，值为需要插入的表名

struct insertValue \*values 插入元组的根节点

③ 输出参数：无

④ 执行流程：找到需要插入元组的表，打开相应的文件(.txt)，将插入的元组值保存在数组内，在文件末尾添加进去，若暂时未想好填的值，可填NULL。

**void insertNoOder(char \*tableName, struct insertValue \*valuesNames, struct insertValue \*values)**

① 函数说明：按用户指定的字段顺序插入元组

② 输入参数：char \*tableName 字符指针，值为需要插入的表名

struct insertValue \*valuesNames 用户给定顺序的字段根节点

struct insertValue \*values 用户插入元组的根节点

③ 输出参数：无

④ 执行流程：找到需要插入元组的表，打开相应的文件(.txt)，将指定顺序的字段和插入的元组值分别保存在不同的数组内，并匹配到表内的所有字段，将元组值插入到相应的位置下。若用户给定的字段数少于表内所有字段数，则会在未指定的字段下补NULL，插入过程中，指定字段和插入元组个数必须相同。

## DELETE

**void deleteAll(char \*tableName)**

① 函数说明：删除指定的表内数据

② 输入参数：char \*tableName 字符指针，值为删除数据的表名

③ 输出参数：无

④ 执行流程：找到需要删除数据的表，打开相应的文件(.txt)，将表内字段数和字段值读出，存放在数组内，调用 fopen函数(参数为'w')，将表重写，写入字段数和字段值。

**void deleteCdt(char \*tableName, struct Conditions \*conditionRoot)**

① 函数说明：删除表内满足条件的字段的元组

② 输入参数：char \*tableName 字符指针，值为删除数据的表名

struct Conditions \*conditionRoot 删除条件的根节点

③ 输出参数：无

④ 执行流程：找到需要删除数据的表，打开相应的文件(.txt)，读出所有内容，对表进行重写入字段数和字段值，每次将表内一行数据存放于数组内，将其传入condition函数，判断当前行是否满足删除条件，满足则不写入当前行的数据，否则写入。

## DROP

**void dropDB()**

① 函数说明：删除指定的数据库文件夹

② 输入参数：无

③ 输出参数：无

④ 执行流程：调用system函数，参数为 "rd databaseName"，其中databaseName是数据库文件夹的名字。

### **void dropTable(char \*tableName)**

① 函数说明：删除指定的表文件

② 输入参数：char \*tableName 字符指针，其值为要删除的表名字

③ 输出参数：无

④ 执行流程：调用system函数，参数为 "del tableName"，其中tableName为要删除的表文件名。

### **UPDATE**

#### **void updateAll(char \*tableName,struct Setstruct \*setRoot)**

① 函数说明：更新所有选定字段的元组值

② 输入参数：char \*tableName 字符指针，其值为要更新的表名字

struct Setstruct \*setRoot 要更新的字段的根节点

③ 输出参数：无

④ 执行流程：打开需要更新的表文件(.txt)，将所有内容读出，保存在数组内，并将表文件重写，需要更新的字段值及更新的元组值分别保存在数组内，匹配更新字段和原表内字段，将更新值写入到相应位置下，否则写入旧值。

#### **void updateCdt(char \*tableName,struct Setstruct \*setRoot,struct Conditions \*conditionRoot)**

① 函数说明：更新所有满足条件选定字段的元组值

② 输入参数：char \*tableName 字符指针，其值为要更新的表名字

struct Setstruct \*setRoot 要更新的字段的根节点

struct Conditions \*conditionRoot 更新条件根节点

③ 输出参数：无

④ 执行流程：打开需要更新的表文件(.txt)，将所有内容读出，保存在数组内，并将表文件重写，需要更新的字段值及更新的元组值分别保存在数组内，每次读入一行数据，传入condition函数，判断是否满足更新条件，若满足则匹配更新字段和原表内字段，将更新值写入到相应位置下，否则写入旧值。

### **EXIT**

#### **exit(0)**

## **2.4 测试功能**

### **测试1**

**输入：**

CREATE DATABASE HQ;

CREATE DATABASE HXW;

```
creaTE DAtaBaSE tq;
```

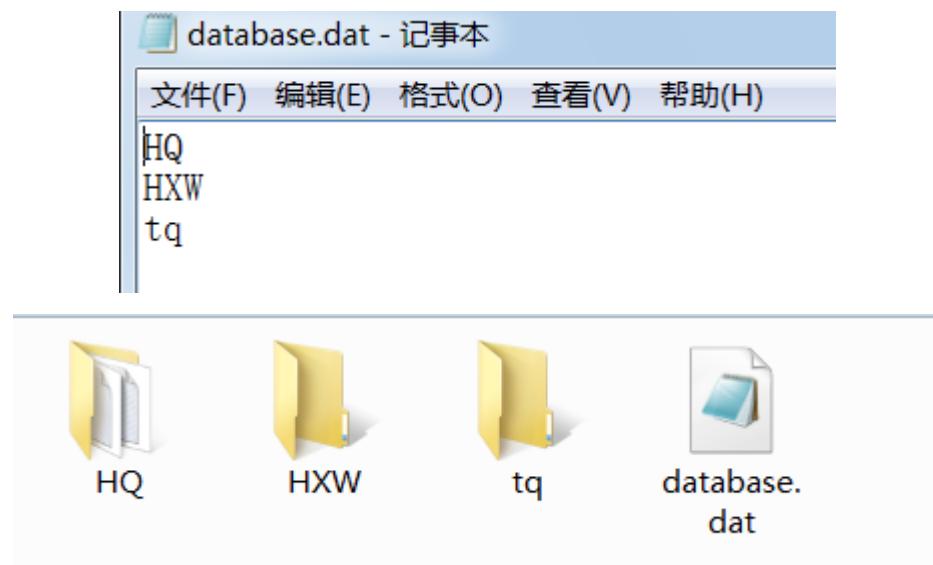
输出：

```
*****
*          Welcome to MySQL!
*
*****  
MySQL>CREATE DATABASE HQ;  
Create database HQ succeed!  
MySQL>CREATE DATABASE HXW;  
Create database HXW succeed!  
MySQL>creaTE DAtaBaSE tq;  
Create database tq succeed!
```

测试2

输入： SHOW DATABASES;

输出：



测试3

输入：

```
USE HQ;
```

输出：

```
MySQL>USE HQ;  
Current database:HQ
```

## 测试4

输入：

```
create table Student (Sno CHAR(10),Sname CHAR(10),Ssex CHAR(2),Sage INT);
CREATE TABLE Course (Cno CHAR(5),Cname CHAR(10),Cpno CHAR(5),Ccredit INT); create TABLE SC (Sno
CHAR(10),Cno CHAR(5),Grade INT); create table test (Sdept CHAR(10),Cpno CHAR(5),NUMBER INT);
```

输出：

```
MySQL>create table Student (Sno CHAR(10), Sname CHAR(10), Ssex CHAR(2), Sage INT);
Create succeed!
MySQL>CREATE TABLE Course (Cno CHAR(5), Cname CHAR(10), Cpno CHAR(5), Ccredit INT);
Create succeed!
MySQL>create TABLE SC (Sno CHAR(10), Cno CHAR(5), Grade INT);
Create succeed!
MySQL>create table test (Sdept CHAR(10), Cpno CHAR(5), NUMBER INT);
Create succeed!
```

|  |                |        |
|--|----------------|--------|
|  Course.txt   | 2018/6/13 0:01 | 文本文档   |
|  SC.txt       | 2018/6/13 0:01 | 文本文档   |
|  Student.txt | 2018/6/13 0:01 | 文本文档   |
|  tables.dat | 2018/6/13 0:01 | DAT 文件 |
|  test.txt   | 2018/6/13 0:01 | 文本文档   |

## 测试6

输入：

```
show tables;
```

输出：

tables.dat - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
Student Sno CHAR 10
Student Sname CHAR 10
Student Ssex CHAR 2
Student Sage INT 4
Course Cno CHAR 5
Course Cname CHAR 10
Course Cpno CHAR 5
Course Ccredit INT 4
SC Sno CHAR 10
SC Cno CHAR 5
SC Grade INT 4
test Sdept CHAR 10
test Cpno CHAR 5
test NUMBER INT 4
```

### 测试7

输入：

```
drop table test;
```

输出：

```
MySQL>drop table test;
Delete table Succeed!
```

|             |                |        |
|-------------|----------------|--------|
| Course.txt  | 2018/6/13 0:01 | 文本文档   |
| SC.txt      | 2018/6/13 0:01 | 文本文档   |
| Student.txt | 2018/6/13 0:01 | 文本文档   |
| tables.dat  | 2018/6/13 0:09 | DAT 文件 |

### 测试8

输入：

```
insert into Student values ('121','LY','M',20);
```

输出：

```
MySQL>insert into Student values ('121','LY','M',20);
Insert succeed!
```

### 测试9

输入:

```
INSERT INTO Student (Ssex,Sage,Sno,Sname) values('F',19,'124','LC');
```

输出:

```
|MySQL>INSERT INTO Student (Ssex, Sage, Sno, Sname) values(' F ', 19, ' 124 ', ' LC ') ;  
|Insert succeed!
```

测试10

输入:

```
update SC set Grade=55 where Sno='125' AND Cno='3';
```

输出:

```
MySQL>update SC set Grade=55 where Sno=' 125 ' AND Cno=' 3 ' ;  
|已复制 1 个文件。  
|Update succeed!
```

测试11

输入:

```
update Course set Ccredit=4 where Cno='1' OR Cno='3' OR Cno ='5';
```

输出:

```
MySQL>update Course set Ccredit=4 where Cno=' 1 ' OR Cno=' 3 ' OR Cno = ' 5 ' ;  
|已复制 1 个文件。  
|Update succeed!
```

测试12

输入:

```
select * from Student;
```

输出:

```
MySQL>select * from Student;  
Sno Sname Ssex Sage  
121 LY M 18  
122 WM F 20  
123 ZL M 17  
124 LC F 17  
125 FA M 17  
5 Records Selected!
```

### 测试13

输入：

```
select Sno from Student where ((Sage=17));
```

输出：

```
MySQL>select Sno from Student where ((Sage=17));
Sno
123
124
125
3 Records Selected!
```

### 测试14

输入：

```
select Student.Sno,Student.Sname,SC.Cno,Course.Cname from Student,SC,Course where
Student.Sno=SC.Sno AND Course.Cno=SC.Cno AND Student.Sage=17 AND SC.Grade>70;
```

输出：

```
MySQL>select Student.Sno, Student.Sname, SC.Cno, Course.Cname from Student, SC, Course where S
o=SC.Cno AND Student.Sage=17 AND SC.Grade>70;
Student.Sno      Student.Sname      SC.Cno      Course.Cname
123            ZL                1          DB
123            ZL                4          OPR
123            ZL                5          SJ
124            LC                1          DB
124            LC                2          MATH
124            LC                3          IFM
124            LC                4          OPR
124            LC                6          DSP
125            FA                1          DB
125            FA                2          MATH
10 Records Selected!
```

### 测试15

输入：

```
delete from SC;
```

输出：

```
MySQL>delete from SC;
Delete data Succeed!
```

```
SC.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
3
Sno      Cno      Grade
```

### 测试16

输入：

```
delete from Student where Ssex='M' AND Sage=17;
```

输出：

```
MySQL>delete from Student where Ssex='M' AND Sage=17;
已复制          1 个文件。
Delete data succeed!
```

```
Student.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
4
Sno      Sname    Ssex     Sage
121      LY        M         18
122      WM        F         20
124      LC        F         17
```

### 测试17

输入：

```
exit;
```

输出：

```
MySQL>exit;
Glad to see you again!
```

## 3总结与未来工作

### 3.1 未完成功能

在创建表，书写查询条件时，表名、字段名以及查询值不能以数字开头。

### **3.2 未来实现方案**

在lex中定义新的ID，该ID可以由大写字母，小写字母和数字混合组成。