

# 数据库课程设计报告

马云雷 5080309177（组长）

刘通 5080309449

安承男 5080309056

## 目录

数据库课程设计报告 .....	1
0. 引言 .....	3
1. 词法分析 .....	3
2. 语法分析 .....	4
3. 文件系统: .....	9
a) 文件系统概述: .....	9
b) 索引管理: .....	10
c) 使用索引管理一个表的 key、foreign key。 .....	11
4. 关系代数: .....	12
5. 逻辑查询计划: .....	12
Create 语句: .....	13
Select 语句 .....	14
Insert 语句: .....	14
Delete 语句: .....	15
Update 语句: .....	15
Alter 语句: .....	15
Describe 语句: .....	15
Grant 语句 .....	15
6. 物理查询计划 .....	16
a) 计算 Where 子句中 bool 语句的值 .....	16
b) Describe 的实现。 .....	17
c) Create database .....	18
d) Create table .....	18
e) Create index .....	18
f) Create view .....	18
g) Select (Sigma) 操作 .....	19
h) Project 操作 .....	19
i) Order 操作 .....	19
j) Gamma 操作 .....	20
k) Cross join .....	20
l) Insert .....	20
m) Delete 操作 .....	21
n) Update 操作 .....	21
o) Alter 操作 .....	21
p) Grant 操作 .....	21
q) Null 操作 .....	22
r) View 的替换和子查询 .....	22
s) 完整性约束 .....	22
7. C/S 架构 .....	22
8. 多线程, 多用户 .....	23
9. 用户接口设计 .....	23
10. 测试 .....	26

11、 总结 .....	27
--------------	----

## 0. 引言

本文档是上海交通大学计算机科学与技术系数据库课程设计最终报告。

本次课程设计完成了一个 DBMS, 其中实现了一个 DBMS 的大部分功能, 包括 create , insert, update, delete, select, grant, drop, alter, describe 等等。

Create: 创建数据库、表, 索引, 视图。

Insert: 向表中插入数据。

Update: 更新数据。

Delete: 删除数据, 支持 where.

Select: 从表、视图中查询数据, 支持高级特性, 比如, where, group, order, distinct。

Grant: 用户权限管理。

Drop: 删除数据库、表, 索引、视图。

Alter: 向表中添加或者删除一个列。

Describe: 表述一个表的信息。

其他的高级特性包括:

聚集函数: max, min, count, sum, avg。

Select: 中的运算。

别名: as name。

多重排序, 两种方向的排序: asc、desc。

Key: 键。

Foreign key: 外键。

约束: check 语句, foreign key 4 种约束。

Where 子句: and、or、exists, in, all、any, like escape。

本报告分为以下几个部分:

- 1: 词法分析;
- 2: 语法分析;
- 3: 文件系统;
- 4: 关系代数表达式
- 5: 逻辑查询计划;
- 6: 物理查询计划;

## 1. 词法分析

词法分析主要是提取 sql 语句中的关键字, 运算符,。

关键字包括:

"ALTER"	"ADD"	"ALL"	"AND"	"ANY"	"AS"
"ASC"	"BY"	"CREATE"	"DELETE"	"DESC"	"DROP"
"DATABASE"	"ESCAPE"	"EXISTS"	"FROM"	"INDEX"	"INSERT"

"INTO"	"KEY"	"LIKE"	"NOT"	"NULL"	"ON"
"OR"	"ORDER"	"PRIMARY"	"SET"	"SELECT"	"TABLE"
"UPDATE"	"VIEW"	"VALUES"	"WHERE"	"AUTO"	"CHECK"
"UNION"	"USE"	"HAVING"	"DEFAULT"	"GROUP"	"INCREMENT"
"DISTINCT"	"IN"	"AVG"	"COUNT"	"MIN"	"MAX"
"SUM"	"INT"	"FLOAT"	"CHAR"	"BOOLEAN"	

上边的关键字在写 lex 的时候还考虑了大小写的问题，即我们的词法分析器还支持大小写。

运算符：

=、<>、<、>、<=、>=、+、-、\*、/、%、,、.、;、.、(、)

变量名（正则表达式）：

$[a-zA-Z]([0-9a-zA-Z]|\_)*$

常量（正则表达式）：

digit=[0-9]

notQuote=[^']

doubleQuote=[' ']{2}

int={digit}+

float={digit}+[.]{digit}+

str={notQuote}\*

LineTerminator = \r|\n|\r\n

WhiteSpace = {LineTerminator} | [ \t\f]

## 2. 语法分析

优先级处理：

precedence left OR;

precedence left AND, UNION;

precedence nonassoc LT, LE, GT, GE, EQ, NEQ;

precedence left PLUS, MINUS;

precedence left TIMES, DIVIDE, MOD;

precedence left DOT;

构建语法树：

多条 sql 语句：

sql\_list::=sql

|sql sql\_list

每一个 sql 语句以分号分隔：

sql ::=create:c SEMICOLON

|select:s SEMICOLON

|insert:i SEMICOLON

```

|delete:d SEMICOLON
|update:u SEMICOLON
|drop:dr SEMICOLON
|alter:a SEMICOLON
|describe:d SEMICOLON
|grant:g SEMICOLON
|use:u SEMICOLON

```

1、Create 语句,create table, database ,index, view:

```

create ::=CREATE:C TABLE:T NAME:N LPAREN:LP create_element_list:c RPAREN:RP
        |CREATE:C DATABASE:D NAME:N
        |CREATE:C INDEX:I NAME:N1 ON:O NAME:N2 LPAREN:L NAME:N3
RPAREN:R |CREATE:C VIEW NAME:N AS:A select:s

```

创建表是的列定义:

```

create_element_list::=create_element:c
        |create_element:c1 COMMA:C create_element_list:c

```

每一个可能是列定义,也可能是 check 语句,还可能是 key 定义,或者 foreign key 定义:

```

create_element::=column_definition:c
        |CHECK LPAREN bool_expr:b RPAREN
        |PRIMARY KEY LPAREN namelist:n RPAREN
        |FOREIGN KEY LPAREN NAME RPAREN REFERENCES NAME: LPAREN
NAMERPAREN

```

列定义:

```

column_definition::=NAME data_type null_or_not default_value auto_increasement primary_key
        | NAME:i data_type:dty CHECK LPAREN bool_expr:b RPAREN

```

Check 语句:

Check 语句是一个 bool 的表达式。

```

bool_expr::=value:v1 cop:c value:v2
        |bool_expr:b1 AND bool_expr:b2
        |bool_expr:b1 OR bool_expr:b2
        |EXISTS LPAREN select:s RPAREN
        |NOT EXISTS LPAREN select:s RPAREN
        |value:v IN LPAREN select:s RPAREN
        |value:v NOT IN LPAREN select:s RPAREN
        |value:v cop:c ANY LPAREN select:s RPAREN
        |value:v cop:c ALL LPAREN select:s RPAREN
        |LPAREN bool_expr:b RPAREN
        |value:v LIKE STRING:S
        |value:v LIKE STRING:S ESCAPE STRING:S2
        |value:v NOT LIKE STRING:S

```

Default value 是词法分析出来的常量,包括整数,字符串,浮点数 boolean 类型的常量

2、select 语句:

```

select ::= SELECT distinct_or_not:d select_expr:s
        from_clause:f

```

where\_clause:w

group\_clause:g

having\_clause:h

order\_clause:o

select 语句中的一些子句可能是空的。

distinct\_or\_not ::=

| DISTINCT

From 子句:

from\_clause ::=

| FROM table\_ref\_list:trl

table\_ref\_list :: = (table\_ref) +

Where 子句

where\_clause ::=

| WHERE bool\_expr:b

Group 子句

group\_clause ::=

| GROUP BY col\_name:c

Having 子句:

having\_clause ::=

| HAVING bool\_expr

Order 子句

order\_clause ::=

| ORDER BY order\_list

order\_list ::= col\_name:n asc\_or\_desc:a

| col\_name:n asc\_or\_desc:a COMMA order\_list:o

asc\_or\_desc ::=

| ASC

| DESC

3:insert 语句:

insert ::= INSERT:i INTO NAME VALUES LPAREN const\_value\_list RPAREN

| INSERT:i INTO NAME:N VALUES LPAREN select:s RPAREN

| INSERT:i INTO NAME:N LPAREN namelist:nl RPAREN VALUES LPAREN select:s

RPAREN

| INSERT:i INTO NAME:N LPAREN namelist:nl RPAREN VALUES LPAREN

const\_value\_list:cl RPAREN

4: delete 语句:

delete ::= DELETE:D FROM NAME:N

| DELETE:D FROM NAME:N WHERE bool\_expr:b

5: UPDATE 语句:

update::=UPDATE:U NAME:N SET assignlist:a WHERE bool\_expr:b

|UPDATE:U NAME:N SET assignlist:a

6: drop 语句:

drop::=DROP:D TABLE namelist:n

|DROP:D VIEW NAME:N

|DROP:D INDEX NAME:N1 ON NAME:N2

|DROP:D DATABASE NAME:N

7alter 语句:

alter::=ALTER:A TABLE NAME:N ADD NAME:N1 data\_type:dt

|ALTER:A TABLE NAME:N DROP NAME:N1

8: describe 语句:

describe ::= DESCRIBE namelist:n

9: grant 语句:

grant::=GRANT privileges:p ON namelist:n1 TO namelist:n2

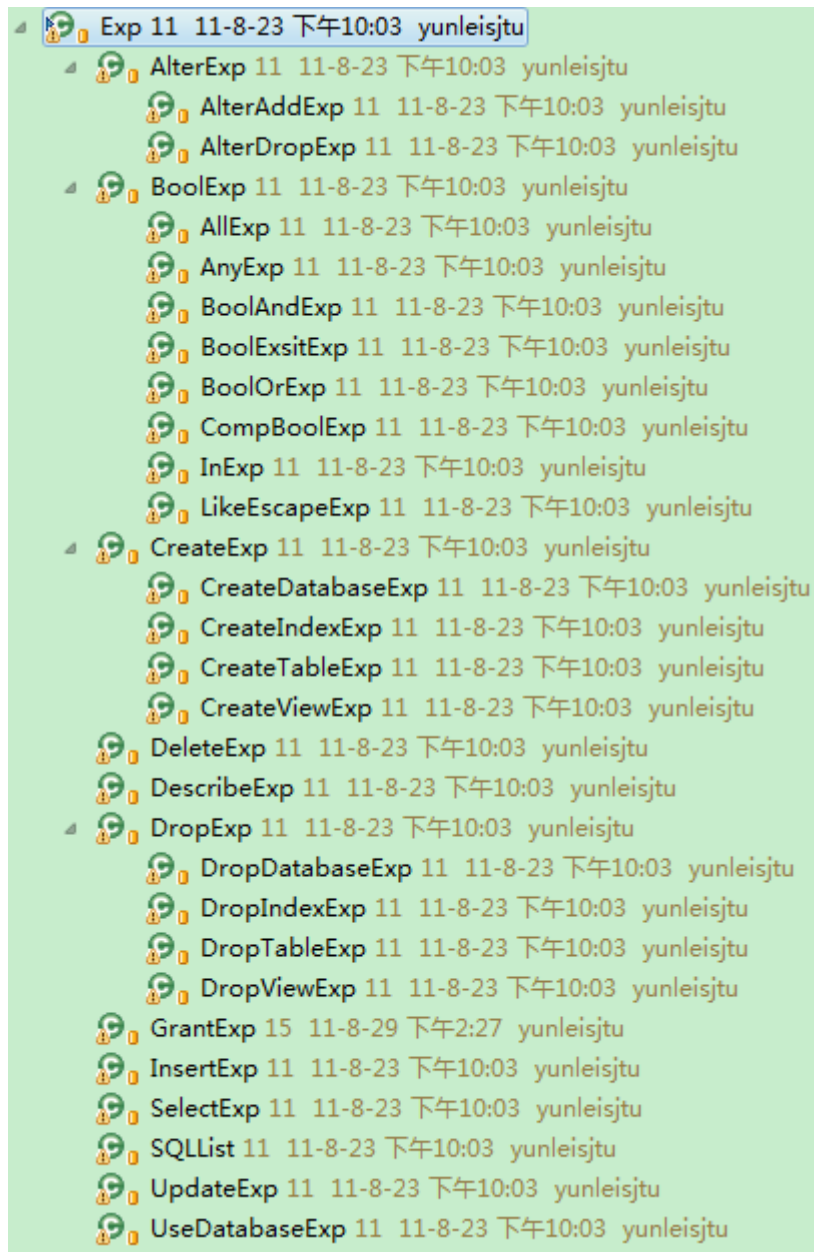
|GRANT privileges:p ON namelist:n1 TO namelist:n2 WITH GRANT OPTION

10: use database:

use ::=USE NAME

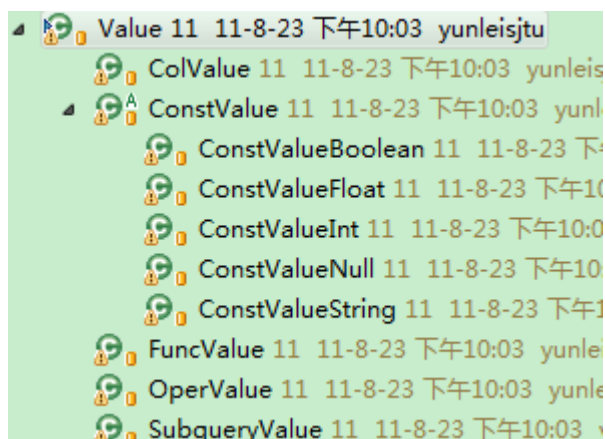
Absyn 包中存放所有的非终结符节点:

节点的继承关系如图:



图中的这些节点都是继承与 Exp 这个虚类。

除此之外，凡是设计运算的地方（变量或常量），都是继承于 value 的节点：



Colvalue 是一个列变量，包含一个列名，计算时从元组中获得它的值。

ConstValue 是常量，还分为 boolean, float, int, null, string, 这些是由词法分析器获得的。

FuncValue 是聚集函数，包含一个列名。

Opvalue 的定义是 (value1 op value2), 表示 value 的四则运算。

SubqueryValue 是子查询。包含一个 select 子句。

抽象语法树正是由这些节点构成的。

### 3. 文件系统:

#### a) 文件系统概述:

文件系统提供数据库对底层文件的操作。

数据库的文件格局是这样布置的:

一个数据库是一个文件夹;

一个表有两个文件: 属性属性 (“attr”后缀文件) 和数据文件 (“data”后缀文件)。

数据文件格式: 一行以“;”结尾, 一列以“,”结尾。不过由于数据结构的限制, 也是为了插入和删除方便, 每一行的长度是固定的。

一个数据库的所有的 view, 存放在一个文件 view.list 中;

一个 table 的索引, 存放在以 table 名字命名的 tablename.index 文件中。

View, 索引, 以及表的属性都是序列化的 java 对象。

数据长度定义:

```
public static int INTSIZE=11;
```

```
public static int BOOLSIZE=5;
```

```
public static int FLOATSIZE=16;
```

```
char 类型的变量有另外的一个变量定义。
```

DBInfo 包下的 DBMani.java 提供了一系列函数:

判断 db 是否存在:

```
public static boolean getDB(Symbol name)
```

获得特权链表

```
public static PrioList getPrioList()
```

保存特权链表

```
public static void putPrioLit(PrioList pl)
```

判断某个表是否存在

```
public static boolean hasTable(String db,String tablename)
```

判断某个用户是否存在

```
public static boolean checkUser(String name,String pw)
```

删除一个用户

```
public static void deleteUser(String name,String pw)
```

添加一个用户

```
public static void addUser(String name,String pw)
```

添加 index 的 info

```
public static void addIndexInfo(String database,String tablename,String col)
```

删除 index 的 info

```
public static void removeIndexInfo(String database,String tablename,String col)
```

保存 index 链表

```
public static void putIndexList(String database,IndexList il)
```

获得 index 链表

```
public static IndexList getIndexList(String database )
```

获得文件大小

```
public static long getfilesize(String database,String tablename)
```

读一个数据文件的随机一部分

```
public static String read(String name, int index, int num)
```

读数据文件的全部数据。

```
public static String readFile(String database,String tablename)
```

删除一个 table 的数据文件

```
public static void delteFile(String database,String tablename)
```

创建 table 的文件，包括属性文件和

```
public static void createNewTableFile(String dbname,String tablename) throws IOException
```

创建一个数据库

```
public static boolean createDB(String dbname)
```

获得 view 链表

```
public static ViewList getViewList(String db)
```

保存 view 链表

```
public static void putViewList(String db,ViewList vl)
```

获得属性链表

```
public static Alge.AttrList getAttrList(String db,String table)
```

获得视图定义

```
public static SelectExp getViewDef(String db,String table)
```

## b) 索引管理：

索引采用我们自己在 hashmap 基础上编写的 HashIndex 数据结构：

```
public class IndexMap extends Hashtable<Object , Object> implements Serializable
```

```
public class HashIndex
```

```
public String database;
public String table ;
public String col;
public String indexname;
public IndexMap hash;
```

```
}
```

Indexmao 可以序列化到磁盘上。此数据结构在索引管理、key 管理、以及 foreign key 管理上起到了很大的作用。

HashIndex 类给索引进行了一场封装，包括读写索引，查询键值，删除键值，添加位置，删除位置。等等操作：

某个键值对应一个 list，这个 list 记录的是一系列的位置信息，表明这个键值出现在了哪个位置上。

原理如下：

key1   pos1,pos2 pos3	
key2	
key3   pos5,pos6,po8,.....	

保存数据：     public void putData()

读取数据     public void getData()

查询某个键值上的数据数量     public int   posSize(Object key)

删除某个键值上的某个数据     public void deletePos(Object key,Object o)

查询是否包含某个键值     public boolean hasKey(Object key)

查询某个键值上是否包含某个位置     public boolean containPos(Object key,Object o)

### c) 使用索引管理一个表的 key、foreign key。

如果某个表中含有 key。则为这个表建立一个索引，索引的名字是 tablename.key.index。

利用索引来管理 key。

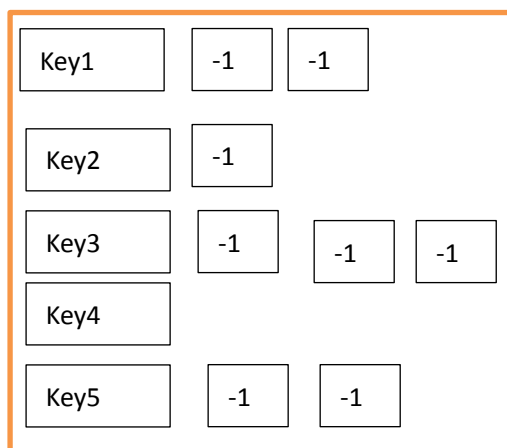
当插入某个键值时，只需要插入一个位置 (-1)；这是个特殊的位置。而对于键值管理来说，没必要知道一个键值的位置。

初始时，某个键值上只有一个位置。当这个键值被 foreign key 引用时，就会在这个键值上再插入一个位置 (-1)；如果一个 foreign key 删除，那么这个键值上的位置就会减一。也就是：

一个键值被 foreign key 引用的次数= 这个键值上的位置数目-1；

这样，当删除一个键值时，就能否很方便的检查这个键值是否被 foreign key 引用，以及被引用的次数。当一个键值被引用次数大于 0 的时候（即该键值的对应的位置大于 1），这个键值无法被删除，由此很方便的实现完整性约束。

原理图如下：



## 4. 关系代数:

从抽象语法树到关系代数是很重要的一步,, 因为对于单个的 sql 语句是很复杂的, 必须把这个操作拆分成更简单的操作组合。比如 select 这个语句, 最复杂的 select 语句包含了 where 子句, order 子句, group 子句, having 子句, distinct 条件。而且 select 这个语句经常作为子查询出现。因此必须能够实现递归。

关系代数包括:

Projection: 从一个表中选择若干个列;

Selection: 根据一个条件选择部分符合条件的元组。

Join: 把两个表进行两两连接。

Duplicate elimination:消除重复。

Aggregation Operator: max, min, count, sum, avg。

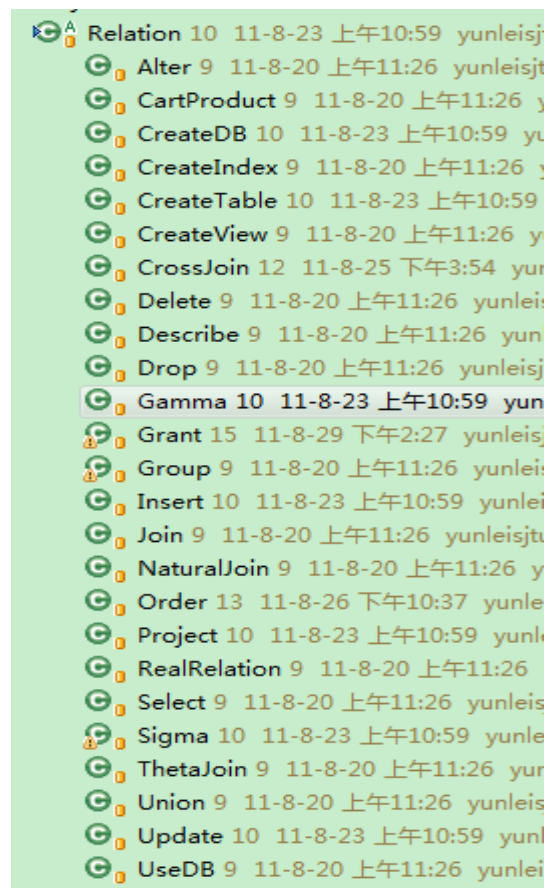
Grouping: 分组

一个完整的 select 可以语句可以用上述关系代数进行操作, 而对于其他的数据修改操作比如 insert, update, 这可以进行类似于语法树的翻译。如果涉及到子查询, 则递归翻译 select。

## 5. 逻辑查询计划:

逻辑查询计划就是把语法树转化成关系代数, 所有的逻辑查询操作都位于 Alge 包下。该部分的代码位于: Semant/Semant.java 中。

首先看逻辑查询计划的层次结构。



所有的逻辑查询操作都继承于 **Relation** 这个虚类。

这个虚类存放着一些公共的成员变量：

**AttrList attrlist;**

**String result;**

每一个操作的返回值存放在这两个域中，第一个用于存放返回的结果表的属性列表，第二个存放表的数据，或者提示信息。

从抽象语法树到逻辑查询计划，中间还要进行类型检查，主要进行的检查包括：

数据库、表、索引、视图是否存在；

表的某个域是否存在；

要插入的数据的类型是否符合定义；

要修改的数据是否符合定义；

程序的入口为：

```
public Relation transExp(Exp e)
{
    Relation result = null;
    if(e instanceof CreateExp)
        result= transExp((CreateExp)e);
    if(e instanceof SelectExp )
        result= transExp((SelectExp)e);
    if(e instanceof InsertExp )
        result= transExp((InsertExp)e);
    if(e instanceof DeleteExp )
        result= transExp((DeleteExp)e);
    if(e instanceof UpdateExp )
        result= transExp((UpdateExp)e);
    if(e instanceof DescribeExp)
        result= transExp((DescribeExp)e);
    if(e instanceof DropExp )
        result= transExp((DropExp)e);
    if(e instanceof GrantExp )
        result= transExp((GrantExp)e);
    if(e instanceof UseDatabaseExp )
        result= transExp((UseDatabaseExp)e);
    if(e instanceof AlterExp )
        result= transExp((AlterExp)e);
    return result;
}
```

## Create 语句：

**Create database** 检查数据库是否存在，并且把当前的数据库设置为新创建的这个数据库。

**Create table:** 检查 **table** 是否存在，并且根据声明语句进行两趟扫描。第一遍，扫描列定义，

并且把这个新的列加入到链表中。第二遍扫描，处理 **key** 定义，**foreign key** 定义，以及 **check** 语句。

**Create view** 以及 **create index**，都是先检查相应的视图或者索引是否存在。然后生成关系代数。

## Select 语句

首先根据 **from** 子句中的 **table** 生成一个 **crossjoin**，如果其中包含视图，就进行替换，就是全连接，连接的每一个节点要么是一个子查询，要么是一个 **RealRelation**（即存储在磁盘上的已知表）。

然后检查 **select** 选择的列名知否在 **crossjoin** 中存在。

然后检查 **where** 子句中涉及到的列名是否存在。

检查 **order** 子句中出现的列名是否存在。

检查 **group** 子句中出现的列名是否存在。

检查 **having** 子句中出现的列名是否存在。

最后是生成关系代数。

首先生成 **Sigma**（对应于 **select**），也就是根据 **where** 子句的 **bool** 表达式和 **from** 子句的 **crossjoin** 生成一个选择语句。

然后生成 **Project** 子句，也就是从一个表中选择某些列，在这一步，如果 **group** 子句不为空，也要加上 **group** 子句的内容，因为 **group** 的选项会影响 **project** 的结果。

如果存在 **order** 子句的话，再生成 **order** 代数表达式；

如果存在 **distinct** 条件的话，再生成 **Gamma** 代数表达式。

所以如果一个 **select** 句子包含了所有的子句的话，结构应该都是这样的：

$(\text{RealRelation} \mid \text{SubQuery}) \rightarrow \text{Sigma} \rightarrow \text{Project} \rightarrow \text{Order} \rightarrow \text{Gamma}$ 。

这个过程可以递归实现。由于在很多 **sql** 语句中存在递归调用，因此上述过程会重复调用。

## Insert 语句：

首先根据要插入的列名（如果没有提供列名，则默认是每一列都要插入，首先从表的属性中提取出所有的列名），从表的属性中找到对应的属性，检查他的类型，然后检查要插入的常量的类型，查看两者是否一致。

**Insert** 语句还有一种情况就是涉及到 **select** 子查询，把 **select** 子查询的结果作为常量插入到表中。这时要先递归分析 **select** 能够返回的结果，然后一一检查这个结果的类型是否和表中的属性相符和。

最后根据（**name list** 和 **const value list**）或（**name list** 和 **select**）生成关系代数。

## Delete 语句:

Delete 语句主要检查 where 子句中出现的所有的列名是否存在,然后根据表明和 where 子句中的 bool 表达式生成关系代数。

## Update 语句:

Update 语句要检查 table 是否存在,然后检查赋值语句,检查赋值语句中出现的列名是否存在,然后根据列名获得属性,并对比赋值语句右侧的常量表达式的类型判断两者是否相一致。并且判断 where 子句中出现的列名的合法性。最后根据以上条件生成关系代数。

## Alter 语句:

Alter 语句分 alter add 和 alter drop。如果是 alter add,只需要直接生成关系代数表达式就可以了。

如果是 alter drop,要首先检查一下希望 drop 的列名是否存在,如果不存在的话,报错返回。

## Describe 语句:

对于 describe 语句中的每一个表的名字,要分别检查其是否存在。然后生成关系代数。

## Grant 语句

Grant 语句首先要检查用户是否存在,当前用户是否存在赋予其他用户权限的权限 (with grant option)。并计算好权限,生成 Grant 关系代数。

## 6. 物理查询计划

在前一阶段逻辑查询计划的基础上，物理查询计划主要是对磁盘数据库的操作，执行，以及最终返回结果。

在物理查询计划中有一些很基础的操作，比如，根据一个列名，来查找它的值；还有判断一个 where 子句中 bool 语句的值，这些操作有很大的重复性，我们将首先阐述这些操作

### a) 计算 Where 子句中 bool 语句的值

计算一个 bool 语句是否满足条件的情形是：给定一个元祖，和它的属性，还有一个 bool 表达式，要求计算 true or false？

函数声明如下：

public boolean calBoolExp(BoolExp boolexp, List<Attr> attrlist, String [] values)

参数依次为：bool 表达式，属性列表，一条元组。

由于 bool 表达式包括：and, or, all, any, in, exist, like, compare。所以需要有一个分发的过程：

```

    if(boolexp instanceof AllExp)
        return calBoolExp((AllExp)boolexp, attrlist, values);
    if(boolexp instanceof AnyExp)
        return calBoolExp((AnyExp)boolexp, attrlist, values);
    if(boolexp instanceof BoolAndExp)
    {
        BoolAndExp bae=(BoolAndExp)boolexp;
        return calBoolExp(bae.exp1, attrlist,
values)&&calBoolExp(bae.exp2, attrlist, values);
    }
    if(boolexp instanceof BoolOrExp)
    {
        BoolOrExp bae=(BoolOrExp)boolexp;
        return calBoolExp(bae.exp1, attrlist,
values)||calBoolExp(bae.exp2, attrlist, values);
    }
    if(boolexp instanceof BoolExsitExp)
        return calBoolExp((BoolExsitExp)boolexp, attrlist, values);
    if(boolexp instanceof CompBoolExp)
        return calBoolExp((CompBoolExp)boolexp, attrlist, values);
    if(boolexp instanceof InExp)
        return calBoolExp((InExp)boolexp, attrlist, values);
    if(boolexp instanceof LikeEscapeExp)
        return calBoolExp((LikeEscapeExp)boolexp, attrlist, values);
    return false;

```

## i. All exp 和 any exp 的实现

allexp 的组成是: value Comp subquery;

value 可能是一个常量, 也可能是一个列名, 如果是一个列名, 则要根据参数中的属性列表找到它的位置, 再从元组值中获得真正的值。

对于 subquery, 要首先递归计算他的值, all 和 any 中的 subquery 的返回值能且仅能包含一行, 然后根据 value 的值, 对 subquery 的每一行调用比较函数。

All exp 是当所有的比较都返回 true 时才返回 true。否则返回 false。

Any exp 是只要某一行的比较返回 true, 就返回 true, 在最后返回 false。

## ii. Exsits 的实现

对于 exists 运算, 要首先递归计算子查询。根据子查询的返回结果是否空和要求存在或者不存在返回相应的值。

## iii. In exp 的实现

In exp 的格式为: value in subselect

对于 in exp, 同样首先得计算子查询, 然后根据 value 是常量, 或者列名, 计算获得真正的值, 然后逐个的进行比较查看 value 是否在子查询的结果中。

## iv. Compare exp 的实现

Compare exp 的格式为 value1 cop value2;

同样的, 根据上边的叙述, 根据 value1 和 value2 获得真正的值;

如果他们是整数, 就调用整数的比较函数, 如果是字符串, 就调用字符串的比较函数, 如果是 float 类型, 就调用 float 的比较函数。

上边是一些基础的操作, 接下来将逐个的介绍各个代数表达式的物理查询计划。

## b) Describe 的实现。

Describe 主要涉及如何存储一个表的属性。

在逻辑查询计划中已经提到, 一个表的属性存放在一个 attr 类型的变量里边。

Attr 的定义如下:

```
public class Attr implements Serializable{
    public String tableName;
    public String name;
    public Type type;
```

```

    public boolean not_null;
    public Absyn.ConstValue defaultValue;
    public boolean auto_incre;
    public boolean key;
    /**check**/
    public Absyn.BoolExp check;

    /***/
    public boolean fk;
    public ColName foreign_key;
}

```

一个 attr 变量，要么存放一个 check，其余的为 null。要么存放一个列的属性，check 为 null。

如果是一个 fk 的话，fk 为 true，并且 foreign\_key 指向另一个表中的列的名字。

一个表是由一系列的 attr 变量通过链表组织起来的：

```

public class AttrList implements Serializable{
    public Attr attr;
    public AttrList next;
}

```

该链表实现了序列化接口，可以写在磁盘上。

Describe 的作用，就是从磁盘上读出结果，然后把结果输出。

## c) Create database

Create database 比较简单，就是创建一个新的文件夹，然后把当前的数据库设为这个数据库就可以了。

## d) Create table

为表创建两个文件 “.attr”文件和 “.data”文件，前者保存属性，后者保存数据。并且把 table 的属性链表保存在属性文件中。

## e) Create index

创建索引文件，并且读出表的内容，为索引添加位置信息。

## f) Create view

View 的信息也保存在磁盘上，每一个 view 保存在下边一个类中：

```

public class View implements Serializable{

```

```

    public String name;
    public Absyn.SelectExp select;
}

```

同样这个 view 也保存在一个链表中。当用户需要时，可以从这个链表中查询到这个 view，并且把 view 的名字替换为一个子查询。

## g) Select (Sigma) 操作

该关系代数的作用是根据一个 bool 表达式选择符合条件的元组作为结果。

首先从 子关系（子查询或者存储表）中获得原始的元组，然后对于每一个元组，调用

```
this.calBoolExp(sigma.condition.boolExp, list, rows[i].split(","))
```

如果返回值为 true，就把该行加入到 sigma 的结果中去。

## h) Project 操作

该关系代数的作用是，从子关系中选择某几列作为结果。

首先要判断是否含有聚集函数，如果含有聚集函数，并且含有既不在聚集函数中又不在 group 中，认为这是错误的。

project 要区分是否含有 group 子句，如果含有这个子句的话，还要处理聚集函数的操作。

如果含有 group 操作：

首先建立一个索引，把 group 的关键字作为索引的键值。

然后，对于每一条元组进行分别的处理，如果要选择的是某一列，或者某些列的四则运算的结果，那么直接加入结果；如果选择的是聚集函数，那么不管这个聚集函数到底是那种类型的，都要根据聚集函数中的列作四种计算：max，min，count，sum。然后把这四种结果加入以 group 为 key 的另一个索引。

当所有的元组处理完毕之后，再判断所选择的列是什么。如果是普通的列或者列的四则运算，那么直接加入结果，如果是聚集函数，那么根据聚集的类型把相应的聚集结果加入结果中去。

如果不含有 group 操作，则运算会稍微简单一些，但还是要注意聚集函数，和上边类似，也要进行聚集操作，然后其他的操作类似。在最终完成之后，判断如果含有聚集操作，则把聚集结果加入最终结果，否则仅仅是普通的结果。

Project 能够选择的列包括：一个列名，一个常量，还有列名和常量的四则运算，聚集函数。

如果要选择的列是四则运算：OperValue，则调用函数：

```

public Integer getOpValue(OperValue ov, List<Attr> attrlist, String []
values) throws Exception

```

进行递归的深度解析，找到这个 OperValue 的最终结果。

## i) Order 操作

Order 是对元组进行排序，支持多重排序。具体实现方式如下：

因为要支持多重的排序，所以使用的算法是桶排序算法（基数排序）。

首先，根据第一个排序关键字，利用 **hash** 散列，把所有的元组散列到若干个桶中；把桶中的数据按照关键字的大小收集起来（如果是 **asc** 排序，就从小到大，如果是 **desc**，从大到小收集）；

把收集起来的数据按照下一个排序关键字重复以上过程。

## j) Gamma 操作

Gamma 是 **distinct** 操作，是把元组中重复的部分去掉。

实现方法是通过 **hash** 映射，把整个元组作为一个关键字进行散列。当发生碰撞时不操作。最后再把所有的数据收集起来，就形成了唯一的元组集。

## k) Cross join

该关系代数是求两个关系的积。

对于第一个关系的每一行，和另一个关系的每一行形成一个新的关系。

## l) Insert

Insert 操作分为四种情况：

Insert into tablename values( const values);

Insert into tablename values( subsuquery);

Insert into tablename ( names) values( const values);

Insert into tablename ( names) values( subsuquery);

对于前两种，只需要从系统中先获得表的所有的列的名字，然后就可以变成第二种形式。

如果插入的是常量：

对于表的属性列表中的每一个列，如果在 **const value** 中提供了值，那么把这个常量插入结果，如果 **const value** 没有提供值，那么从属性的 **default value** 中获得值。但如果这个属性是一个 **key value**，则不能插入这个缺省的值。**Key** 和 **foreign key** 的检查会推迟到最后、插入数据库之前。

如果插入的是子查询：

首先递归执行子查询，获得子查询的结果。然后对子查询的每一行进行上述的操作。

如果存在对于该表的索引，要向索引插入所有的新值。

**Key** 的检查：如果某列是 **key**，要查看 **key** 的索引，如果该键值存在。则插入失败。

**Fk** 的处理：如果某一列是 **fk**，则拿到 **fk** 的索引，并增加该 **fk** 的引用次数。

当上述操作完成后，要把数据写入磁盘。首先计算元组的长度。每一种类型的变量的长度定

义已经在上边给出。然后再加上列的数目（每一列后边加上一个“,”），最后加上 1（“;”）。得到元组的长度，插入数据库。

## m) Delete 操作

Delete 操作要对于每一行元组进行计算 bool 条件的值,当 bool 为真的时候才能删除这一样。

如果 bool 条件是 null，这将全部元组删除。

删除时要检查 key 和 fk。

如果某一列是 key，这要检查 key 的引用次数（key 的索引中的位置的数目），如果引用次数大于 1，表明有 fk 引用到这个 key。不能够删除。

如果某一列是 fk，要首先获得引用的 key 的索引，然后把对应引用次数减一。

## n) Update 操作

如果 update 的 bool 条件是 null，则要把全部的元组进行更新。

如果 bool 条件不为 null，则要对每一行元组进行计算判断条件，对于符合条件的元组才能进行更新操作。

除了 bool 条件之外，还要检查 key 和 foreign key 是否符合条件。

如果某一列的是一个已经存在的 key，更新失败。

如果某一列是一个 fk，要先把原来 fk 引用的外部键的次数减一，并且把新的 fk 引用的外部键的次数加一。如果新的 fk 引用一个不存在的 key，更新失败。在这一系列步骤的操作顺序是：

先检查新的 fk 是否存在，如果存在才能继续进行；

然后删除旧的 fk；

最后添加新的 fk 引用。

## o) Alter 操作

Alter 操作分为添加一个新列和删除一个列。

如果添加一个新列，则增加一个新的属性到属性列表中，并且对于每一个元组，增加一个“null,” 值到表中。

如果删除一个列，则需要检查引用完整性：如果这个列被引用为 foreign key，则不能够删除这一列。

## p) Grant 操作

所有的系统信息存放在 /根目录/system 文件夹下。

每个用户对与一个表的权限存放在一个数据结构中：

```
public class UserPrio implements Serializable{
    private static final long serialVersionUID = 11L;
    public String username;
    public String tablename;
```

```
int priority;  
public static int SELECT=1;  
public static int INSERT=2;  
public static int UPDATE=4;  
public static int GRANT=8;  
}
```

Static 常量是权限定义，真正存储在系统中的是一个整数常量。

当授予新的用户权限时，首先找到当前用户的权限，判断当前用户是否有（grant option）权限，然后把当前用户的权限和要授予的权限进行逻辑与操作。把新的用户权限插入链表保存在数据库中。

## q) Null 操作

当进行数据运算的时候，如果碰到 null，则认为整个表达式都是 null 值。

当进行聚集运算的时候，不把 null 统计在内，但是 count（\*）会把 null 统计在内。

## r) View 的替换和子查询

当进行取表操作时，如果发现当前的名字不是一个表，就去 view list 中寻找以这个名字命名的视图。获得该视图的定义，然后递归解析该视图，并且替换掉原来的视图名字。

## s) 完整性约束

完整性约束主要在修改数据时才能够影响到完整性。Insert, delete, update 时都要检查 key, fk 等等，具体的方法上边已经详细的描述了。

# 7. C/S 架构

本 DBMS 系统采用 Client、Server 架构，具体实现为：

- Client 端登录：

用户登录时，Client 端获得用户名、密码和数据库名，并通过 Socket 协议将这三项信息发送给 Server 端，Server 端检查用户名和密码是否正确且数据库是否存在即是否有使用该数据库的权限。如果以上全部没问题，Server 端返回“yes”给 Client 端，否则返回“no”。

- Client 端操作：

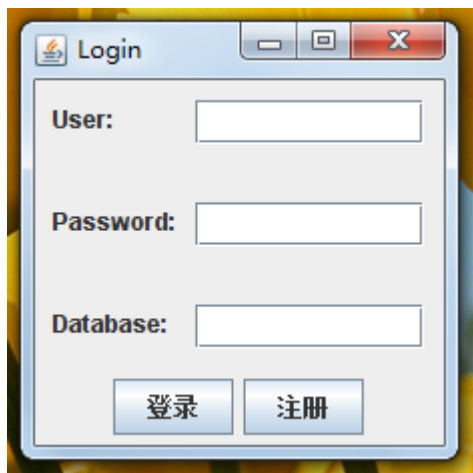
用户成功登录后，Client 端可以获得用户的操作指令，并发送给 Server 端，Server 端执行指令，并将结果返回给 Client 端，Client 端分析后输出。Server 端给每个用户创建一个线程并行的执行各个用户的指令，用户退出时，对应的线程结束。

## 8. 多线程，多用户

当有一个用户请求连接时，就给该用户创建一个线程，用于接收请求，直到用户退出时才能够结束线程。

## 9. 用户接口设计

(一) 登录界面：

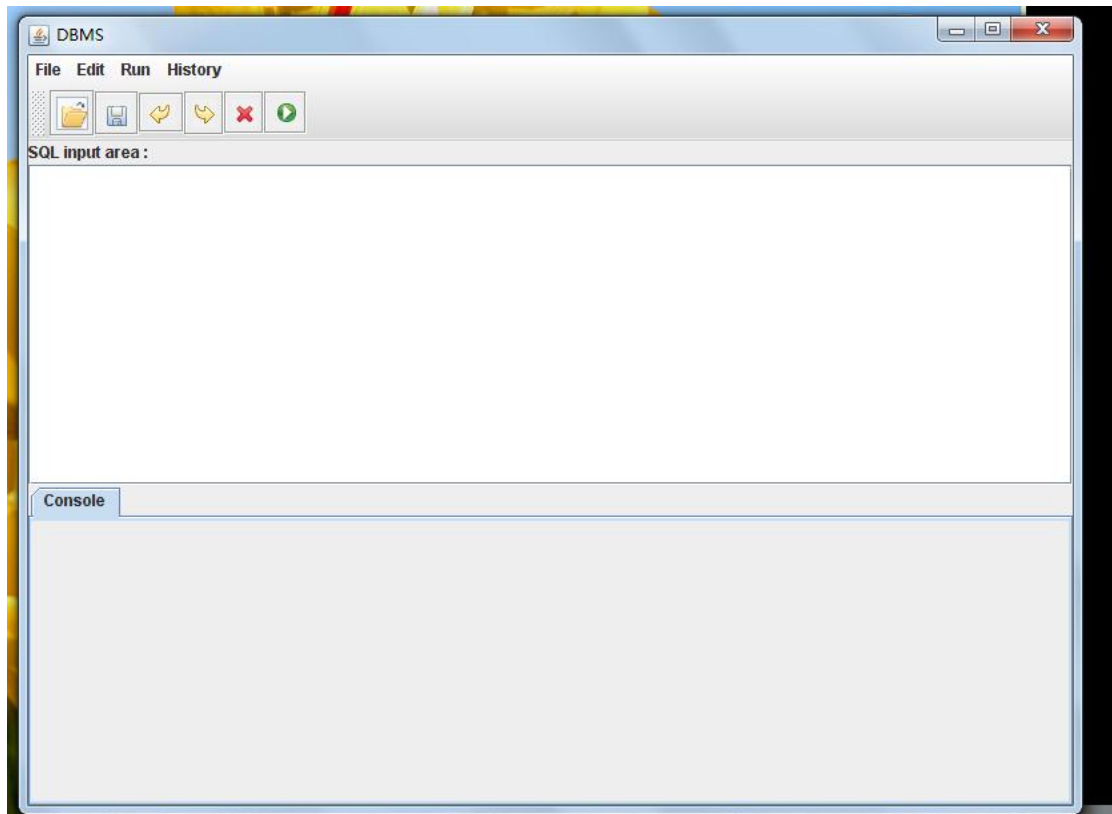
A screenshot of a 'Login' window. It has a title bar with 'Login' and standard window controls. The window contains three text input fields labeled 'User:', 'Password:', and 'Database:'. At the bottom, there are two buttons labeled '登录' (Login) and '注册' (Register).A screenshot of a 'Register' window. It has a title bar with 'Register' and standard window controls. The window contains three text input fields labeled 'User:', 'Password:', and 'Confirm:'. At the bottom, there are two buttons labeled '提交' (Submit) and '取消' (Cancel).

用户首先进入登录界面。若要登录，则输入用户名和密码并选择要访问哪个数据库，点击“登录”按钮，服务器端检查用户名和密码是否正确，若正确则显示 DBMS 的主界面，否则提示错误。若要注册，则直接点击“注册”按钮，然后显示注册界面。

(二) 注册界面：

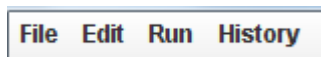
注册界面比较简单，用户只需填写用户名、密码和确认密码，点击“提交”按钮，即可注册，注册后返回登录界面。若点击“取消”按钮，则不注册直接返回登录界面。

(三) 主界面：

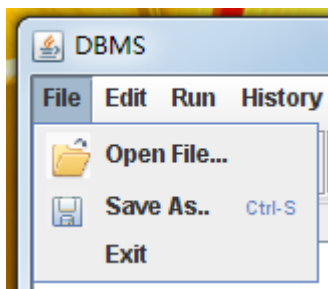


主界面功能介绍：

菜单栏：



- File

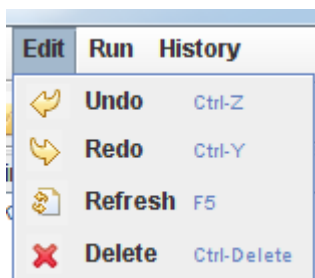


Open File: 打开文件，将文件内容复制到 SQL 语句输入区

Save As: 将 SQL 语句输入区的内容复制到某个文件中

Exit: 退出 DBMS 系统

- Edit



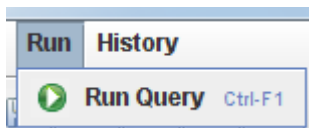
Undo: 将当前指令的上一条指令的内容复制到 SQL 语句输入区

Redo: 将当前指令的下一条指令的内容复制到 SQL 语句输入区

Refresh: 刷新输出结果, 即将当前指令重新执行一遍

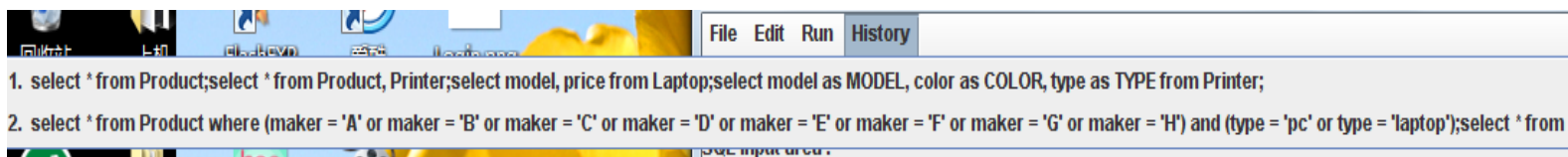
Delete: 删除 SQL 语句输入区中的所有内容

- Run



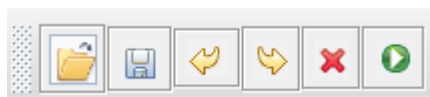
Run Query: 执行 SQL 语句输入区中的内容, 该内容变为当前指令

- History



显示只执行过的所有指令, 单击某条指令, 它的内容复制到 SQL 语句输入区

工具栏:



工具栏上的按钮依次为:

打开文件

保存 SQL 语句

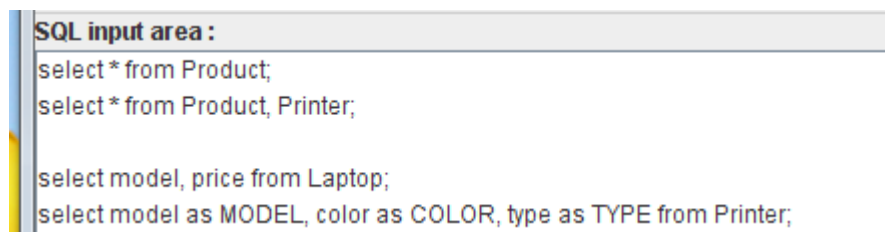
将上一条指令显示在 SQL 语句输入区中

将下一条指令显示在 SQL 语句输入区中

删除 SQL 语句输入区中的内容

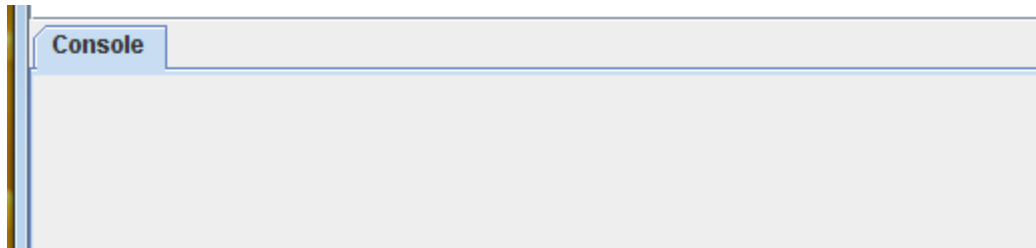
执行 SQL 语句输入区中的指令

SQL 语句输入区:

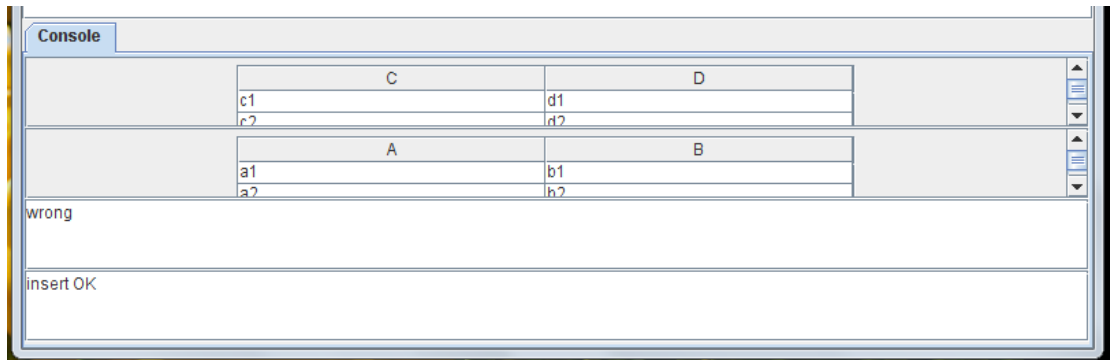


在此输入想要执行的 SQL 语句, 可以同时输入多条语句

执行结果显示区:



点击 Run Query 后，此处显示执行结果或错误提示，可以同时显示多条语句的结果



## 10. 测试

测试实例在 testcase 文件夹下，测试分以下各个项目：

- 1、创建 db 和删除 db
- 2、创建表和删除表
- 3、Describe 描述表
- 4、简单插入表
- 5、Check 约束
- 6、Key 约束
- 7、Foreign key 约束
- 8、Distinct
- 9、Update
- 10、View 测试
- 11、Like escape
- 12、Alter
- 13、添加索引
- 14、AVG sum, count
- 15、Min, max
- 16、子查询比较语句
- 17、Group 测试
- 18、Order 测试
- 19、特殊 select 测试 (select 中有四则运算)
- 20、In 测试
- 21、Exsit 测试
- 22、Grant 测试
- 23、Null 测试

## 11、 总结

通过此次的课程设计，对数据库的实现有了更好的了解。感谢一起完成工作的同学。