《数据库课程设计》（CS372）

实验报告

SELECT NAME, ID, CLASS FROM GROUP_6 ORDER BY ID		
NAME	**ID**	**CLASS**
王飞	5080309436	F0803011
陈斌	5080309505	F0803014
王晓东	5080309517	F0803014

上海交通大学计算机科学与工程系

2011 年 8 月

目录

1 实验概述

1.1 实验目的

使学生加深对《数据库原理》课程中学到的基本概念、基本原理和基本技术的理解；提供一个让学生综合应用所学程序设计、操作系统、编译原理和软件工程等方面知识的机会；培养学生独立自主学习、分析和解决问题的能力；增强学生进行大型程序设计的实践能力。

1.2 实验环境

Windows，java

1.3 系统设计

（1）存储管理器：在磁盘上按关系数据模型存储数据，并支持高效的访问（如索引）；
（2）缓冲管理器：管理内存中的缓冲区，专用于与磁盘之间的数据 I/O；
（3）查询处理器：编译 SQL 语句，生成查询计划，优化查询计划；
（4）执行引擎：执行查询计划；
（5）用户界面：接受用户的 SQL 语句，显示返回结果；图形用户界面，命令行界面。

2 查询处理模块(QueryManager)

2.1 词法分析(Lexer)

2.1.1 Flex 文件说明：
定义空白符、标识符、数字、用户自定义变量等

```
delim=    [ \t\n\f\r]
alpha=    [A-Za-z]
digit=    [0-9]
notquote = [^']
doublequote = [']{2}
id    =    {alpha}({alpha}|{digit}|_)*
intvalue    =    {digit}+
REALVALUE = {digit}+[.]{digit}+
ws    =    {delim}+
str = {notquote}*
```

关键字为了适应大小写匹配，重新定义。如：

```
select = [S|s][E|e][L|l][E|e][C|c][T|t]
```

delete = [D|d][E|e][L|l][E|e][T|t][E|e]

词法分析中定义了两种状态，包括<YYINITIAL>初始状态和<STRINGSTATE>字符串状态。

< YYINITIAL > "\'"　{stringvalue.setLength(0);yybegin(STRINGSTATE);}

碰到’转入<STRINGSTATE>状态进行解析

```
<STRINGSTATE>
{
    {doublequote}
        {stringvalue.append("'");}
    ['']
        {yybegin(YYINITIAL);return tok(sym.STRINGVALUE,stringvalue.toString());}
    {str}
        {stringvalue.append(yytext());}

}
```

读入字符串，碰到’返回<YYINITIAL>状态，继续进行词法分析

```
%eofval{
    {
        if (yystate()==STRINGSTATE)    throw new Exception("String error");
        return tok(sym.EOF, null);
    }
%eofval}
```

词法分析解析后，如果还在<STRINGSTATE>状态，说明出错。

2.1.2 细节问题

状态转换，转入<STRINGSTATE>后处理。

关键字大小写匹配。

2.2 语法分析(Parser)

本数据库支持的数据定义语言(DDL)和数据操纵语言(DML)如下:

SQL 语句	类型	举例	说明
CREATE TABLE	DDL	create table t(a int, b varchar(10));	创建表
DROP TABLE	DDL	drop table t, u;	删除表
CREATE VIEW	DDL	create view vtest as select * from test;	创建视图
DROP VIEW	DDL	drop view vtest;	删除视图
SELECT	DML	select t.a from t as test where t.a = 4;	按照条件查询
INSERT	DML	insert into Product values('A',1001,'pc');	插入记录
DELETE	DML	delete from t where a = 1	删除记录
UPDATE	DML	update table t (a 1, b 2) where c = 3	更新记录

2.2.1 语法规约(Grammar Specification)

实现了 SQL 语法的一个简单子集，主要的语法规约举例如下：

```
program ::= sql_exp_list;
sql_exp_list ::= sql_exp | sql_exp sql_exp_list;
/* create expression */
create_exp ::= create_table_exp | create_view_exp | create_index_exp;
/* select expression */
select_exp ::= select_op | select_bin_exp | LPAREN select_exp RPAREN;
/* insert expression */
insert_exp ::= INSERTINTO ID VALUES LPAREN value_exp_list RPAREN
             | INSERTINTO ID LPAREN id_list RPAREN VALUES LPAREN value_exp_list RPAREN;
/* delete expression */
delete_exp ::= DELETE FROM ID WHERE where_cond_exp;
/* update expression */
update_exp ::= UPDATE ID SET update_column_list WHERE where_cond_exp;
/* drop expression */
drop_exp ::= DROPINDEX id_list | DROPTABLE id_list | DROPVIEW id_list;
/* alter expression */
alter_exp ::= ALTERTABLE ID ADD ID data_type | ALTERTABLE ID ALTERCOLUMN ID data_type
             | ALTERTABLE ID DROPCOLUMN ID;
```

2.2.2 抽象语法树(Absyn)

根据语法规约表达式左侧的非终结符可以构造出抽象语法树的节点，使用 visual class generator 可以根据简单的规约描述自动生成节点类，通过几个主要的节点描述如下：

```
CreateTableExp ::= "String" : name
                   ColumnDefList : list
CreateViewExp ::= "String" : name
                  SelectExp : selectExp
CreateIndexExp ::= "String" : iName
                   "String" : tName
                   IdList : list
SelectOp ::= "boolean" : selectDist
            "boolean" : selectAll
              SelectColumnList : cList
              FromTableList : tList
              WhereExp : whereExp
SelectBinExp ::= "boolean" : unionAll
                 SelectExp : left
                 SelectExp : right
```

```
InsertExp ::= "String" : name
              IdList : iList
              ValueExpList : vlist
ValueExpList ::= ValueExp : head
                 ValueExpList : tail
DeleteExp ::= "String" : name
              WhereCondExp : cond
UpdateExp ::= "String" : name
              UpdateColumnList : list
              WhereCondExp : cond
DropExp ::= "int" : type
            IdList : list
AlterExp ::= "int" : type
            "String" : tName
            "String" : cName
             DataType : dataType
```

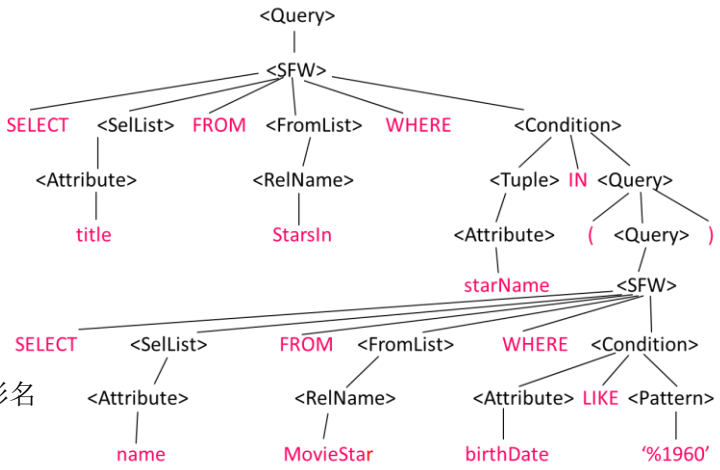
2.3 执行计划生成与优化(ExecutionPlan)

对于 SQL 语句中存在 SFW 类型语句，语法分析得到语法树非常庞大，where 中存在 and、or、子查询，会相互嵌套难以分析运行，切不经优化代价太大，所以语法分析树要转换成查询计划。

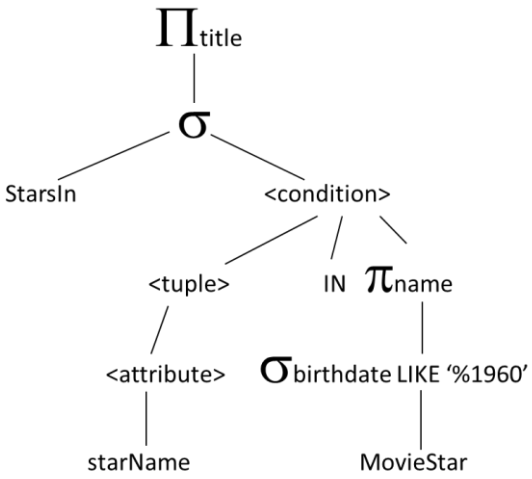
存在子查询情况：

```
SELECT title
FROM StarsIn
WHERE starName IN (
        SELECT name
        FROM MovieStar
        WHERE birthdate LIKE '%1960'
);
```

以上 SQL 语句为找出 1960 年出生的明星的电影名

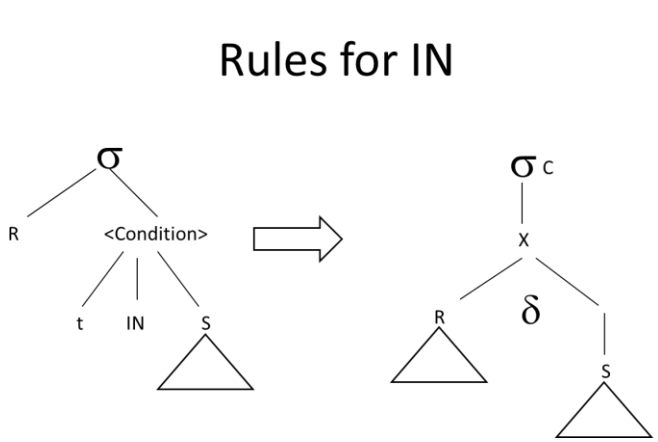


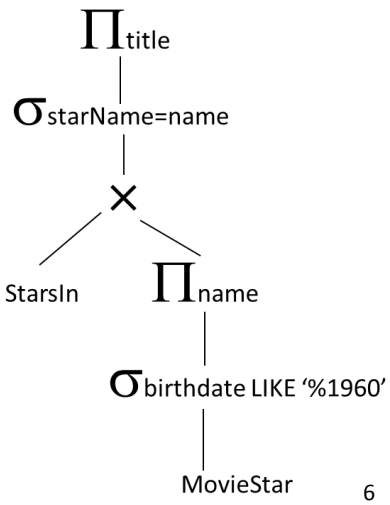
把语法分析树转换成关系代数形式(即逻辑查询计划)中间形式为：



使用 σ 暂时代替σ_c把子查询表示出来

更具 In 规则：转换成逻辑查询计划：

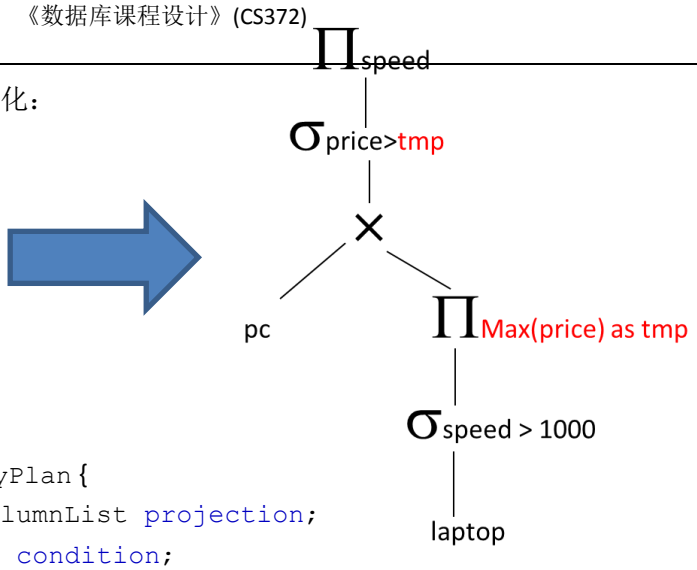




针对对比语句的子查询，做了优化：

如：

```
SELECT speed
FROM pc
WHERE price > all (SELECT price
      From laptop
      WHERE speed >1000)
```



具体实现：

类的主要数据结构：

```
public class LogicalQueryPlan{
    public Absyn.SelectColumnList projection;
    public Absyn.WhereExp condition;
    public Absyn.FromTableList production;
    public ArrayList<LogicalQueryPlan> plans = new
           ArrayList<LogicalQueryPlan>();
    public ArrayList<Absyn.WhereExp> conditionlist = new
          ArrayList<Absyn.WhereExp>();
}
```

类型主要使用语法数的结点来储存，但是形式改变。

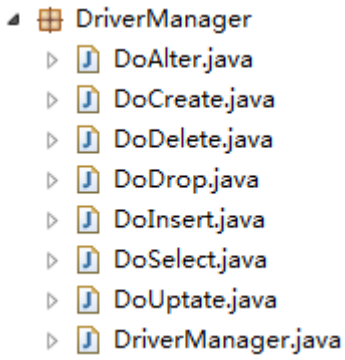
使用ArrayList存储多个子查询。

同时逻辑查询把Where中多个嵌套的条件进行分析转换成有次序执行的conditionlist

物理执行时，只需要递归执行 LogicalQueryPlan，把 production 和 plans 做叉积，根据 condition 判断条件，最后 projection 选择属性即可。

3 执行引擎(DriverManager)

3.1 实现结构



本模块首先启动 RecordManager，然后接收 SQL 语句序列，并传递给 QueryManager，得到其反馈的执行计划，对不同类型的执行计划建立不同的执行子引擎，各执行子引擎通过调用 RecordManager 提供的接口来完成执行计划。

3.2 API 举例

使用举例	说明
DriverManager dm = new DriverManager();	启动执行引擎
dm.runSqlFile(fileName);	执行 SQL 源文件
dm.runSqlList(sqlList);	执行 SQL 语句序列
dm.runSql(sql);	执行一条 SQL 语句

4 记录管理模块&&数据字典模块

4.1 设计思想

该部分在底层磁盘读写模块以及缓冲管理模块的基础上完成数据库的格式化存储。在虚拟磁盘上可以创建多个数据库，而每个数据库即对应一个记录管理模块。
记录管理模块主要包括以下存储区和机制：
1、超块 存储整个数据库的全局信息，如用户名和密码，数据库大小等等
2、位图 用来标记该数据库中的空闲块
3、数据区
a. 数据字典区 存储了所有 Table 的属性信息
b. 数据区 存储每个 Table 对应的 Tuple 的信息
c. VARCHAR 区 当数据区的 tuple 含有类型为 VARCHAR 的变长数据时，在该区存储

4.2 超块中信息

```
String userName;  用户名
String userPasswd;  用户登录密码
int sizeOfDatabase;     数据库大小（字节数）
int blkSize    每块大小（字节数）
int blkCount;      每块的页面数
int nextFreeBlk;  下一个空闲块的块号
int realBlkSize    每块的实际字节数
int nextFreeTableStruct;  插入时下一个表区位置
int nextFreeVchar;      下一个VARCHAR位置
int bitmapSize;    位图大小
```

blkCount*blkSize 为每块的实际大小，blkSize=4K，也就是说数据库支持不同大小的页面划分方法，为 4K 的整数倍
系统初始化的时候，会将超块的信息读取到模块内部缓存

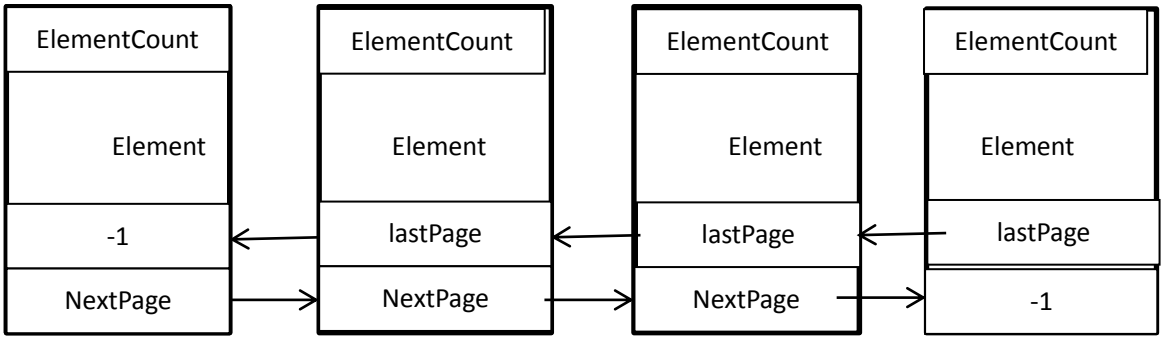
4.3 位图管理

位图大小根据数据库的大小动态确定，维护一个全局变量 nextFreeBlk 表示下一次分配时的空闲块。当空闲块被分配以后，nextFreeBlk 将被改变，指向下一个空闲块。如果被释放的空闲块小于 nextFreeBlk，则将 nextFreeBlk 调整为刚释放的空闲块。
系统初始化的时候，会将位图的信息读取到模块内部缓存

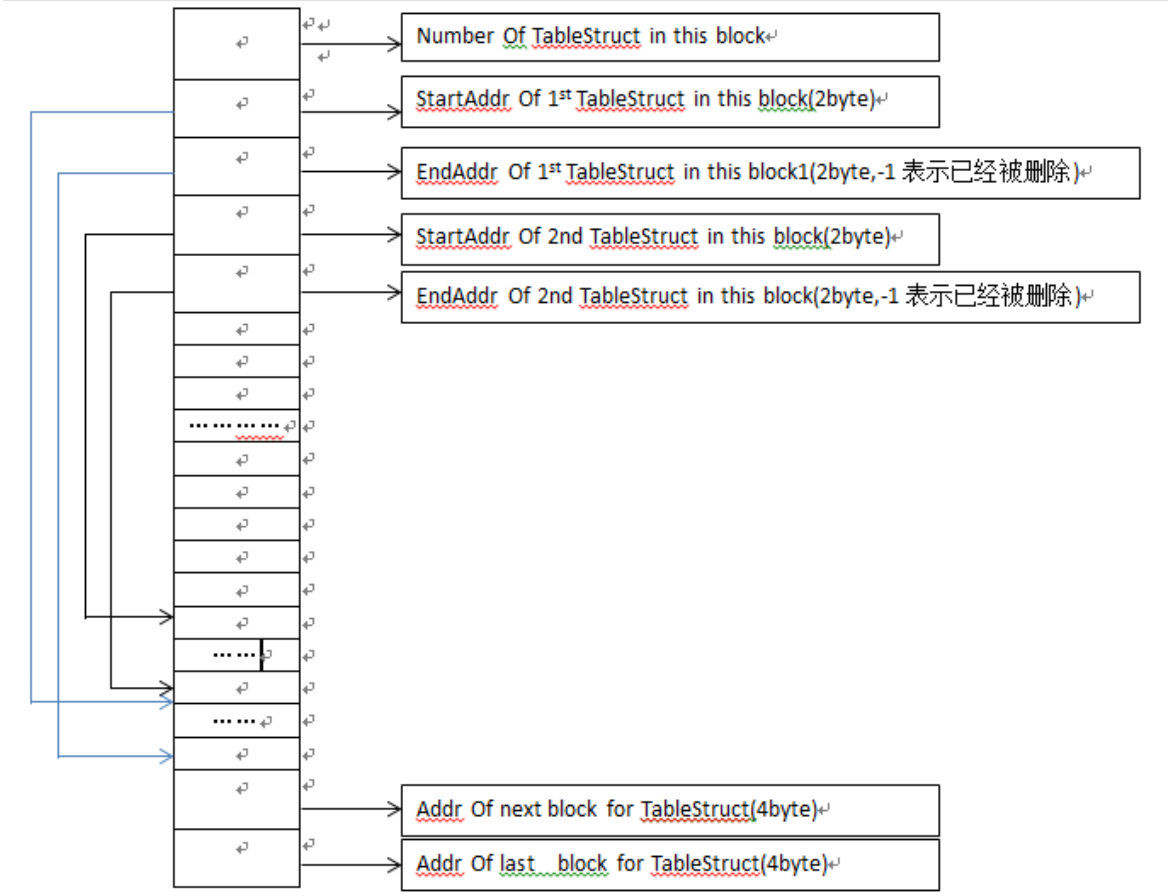
4.4 数据区

a. 数据字典区、b.数据区、c. VARCHAR 区全部使用链表存储

块间链表如下图所示：



块内信息存储示例，以数据字典区为例（TableStruct 为表的属性等信息），其它存储类似：



4.4.1 数据字典区

整个数据字典区各页之间用链表存起来。系统初始化的时候，会将数据字典的信息读取到 Hashmap 中作为缓存。数据由于实现了增加列属性，所以一个 Table 对应多个表区（TableFeature），前一个表区通过 nextTable 指向下一个表区的字节地址，表区不存在则为-1

```
Table数据字典表区数据结构，一个Table可能有多个表区组成
public class TableFeature {
        TableInfo tInfo;
        int firstPage = -1;//First page
        int curPos = -1;// The addr to insert a new record
        int nextTable = -1;// Whether the table has other columns
        int tableCount = 1; //增加列以后，table数目
        int size; //tInfo 中 column 数目
}

每一个表区属性的数据结构
public class TableInfo {
    public String tableName;
    public AttrDataType[] attrDataTypes;
    public String[] attrNames;
    public String[] attrDefaultValues;
    public int[] attrConstraints;
}
```

4.4.2 数据记录区

每个 Table 表区都会在该数据区对应一个链表，该链表包含了 Table 表区的所有 Tuple 页面。一个 Table 可以对应多个表区，在读取 Tuple 的时候，需要将多个表区的数据组合起来，组成一个 Table 对应的真正 Tuple

4.4.3 VARCHAR 区

整个 VARCHAR 区的所有页面之间用一个链表连接起来。

4.4.4 特殊类型处理

Decimal(a,b): 存为两个 long，支持最长整数部分和小数部分共 38 位的 BigDecimal
VARCHAR: 在 Tuple 里存储分配的 VARCHAR 的字节地址

Null: 对于 CHAR(i)类型，存储时第一个字节保留，后面才是真正数据。当保留字节是'&'时，为 NULL。对于其他类型，第一个字节是'&'，就是 NULL

5 磁盘管理模块(DiskManager)

包括在一个磁盘内创建多个数据库，对数据库打开、读、写、关闭和删除操作，并且具有缓存功能。磁盘的第一块存储整个磁盘的信息，包括磁盘的大小，里面所具有的数据库名、数据库大小、数据库所在磁盘的绝对路径。在磁盘中以块为单位进行操作，一个块4KB

DiskManager.java提供上层接口：

boolean CreateFile(String fileName, int filesize) 创建数据库
boolean DestroyFile(String fileName) 删除数据库
MYFile OpenFile(String fileName) 打开数据库，返回句柄
boolean CloseFile(MYFile fileHandle) 关闭数据库
boolean WritePage(int page, String bytebuffer) 写块
boolean ReadPage(int page, byte[] bytebuffer) 读块
boolean WriteData(int byteNum, String data) 写数据

5.1 底层磁盘读写模块(I/O Manager)

在Windows平台下，使用fsutil file createnew <filename> <length>创建一个磁盘，初始内容为0；使用java的RandomAccessFile类对磁盘进行读写。

```
RandomAccessFile $disk = new RandomAccessFile($path, "rw");
```

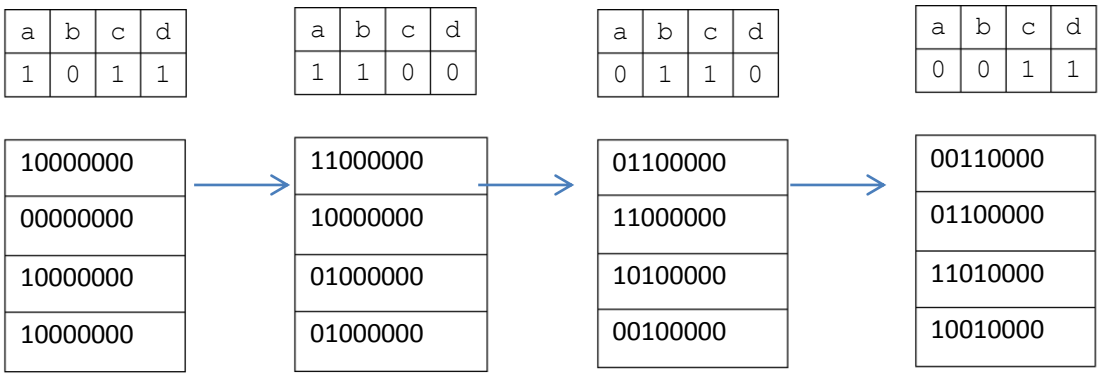
从$path路径中以读写的方式读入初始磁盘，
使用RandomAccessFile的
$disk .seek(place)定位到具体某个byte(操作代价较大，所以尽量使用缓存防止经常对磁盘进行读写。)
$disk .read(buffer)从当前定位从磁盘把buffer长度的数据写入buffer。
$disk .write(buffer)从当前定位把buffer长度的数据写入磁盘。

5.2 缓冲管理模块(BufferManager)

BufferManager.java
根据要求使用了1M缓存，由于文件系统定义一个块为4KB，所以缓存用二维数组
`public byte $diskbuffer[256][4096]` 存储。
缓存原理：

需要写磁盘或者读磁盘的时候，先去缓存查找是否存在所需数据，不存在时候发生缺页，去磁盘上读入数据，修改数据也写入缓存，定时把已修改缓存写回磁盘。

页面替换，利用老化算法，每个时钟周期标志位(本工程使用16位)右移一次。
被使用过则最高位置1，否则置0
如：对于a、b、c、d四块使用情况,第一个表格相应块号下面1表示这个时钟周期被使用。



只要在标记中找出二进制数值(无符号数)最小的数的页面进行替换。

6 参考文献

《数据库基础教程》
《数据库系统实现》