# 《DBMS》实验报告一

**报告人：**冷友方 王大伟 罗传文 赵丹 杨婧如

**学院：**信息学院

**日期：**2016年10月26日

## 一、数据文件的组织

在我们实验中使用了页式组织方式。

系统文件：System.dbf 包含多个Datafile，这里的Datafile是数据文件，每个Datafile对应一个表，表名和datafile相同。

系统文件里面存的是系统的全局信息，包括Page Size， Buffer Size，Filenumber，Datafile等。

* Page Size：块的大小，系统默认使用8K。
* Buffer Size：缓冲区的大小，为块的整数倍，系统使用的是块大小的32倍。
* Filenumber：整个数据库所包含的数据文件个数，系统默认最大为100。
* Dadafile: 是一个结构体，包含文件的ID和Name。

系统文件的组织结构如图1所示。

每个Datafile的第一块为文件头，文件头包含PageNumber，FreePageNumber，Page的位图，使用为1，不使用为0。

* Pagenumber：当前文件的总块数。
* FreePageNumber：空闲块数。
* 每个Page包含头信息和数据信息。

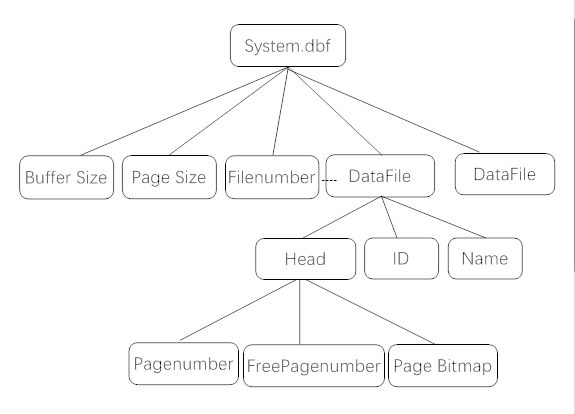


图1 系统文件的组织方式

## 二、缓冲区管理

缓冲区的页面是使用连续的内存块作为整个数据库的全局缓冲区。每块缓冲区的大小和数据文件的Page Size大小相等，也为8K。

当所使用的块数未达到总数时，直接分配一块新的Page。当达到的总数时，则需要做页面替换，缓冲区的替换方式为先进先出，用循环队列的方式实现。

循环队列的方式如图2所示：

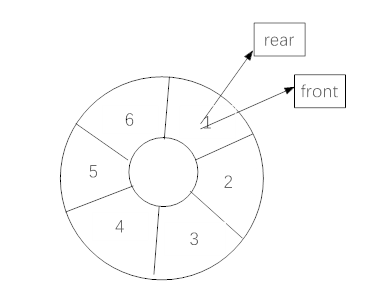


图2 缓冲区的替换方式，循环队列

## 三、空闲空间管理

这里是文件的空闲管理。一个DataFile包含多个Page，每个Page的使用情况在文件头中描述，包含 fileId， pageOfFile，freeCount，isPageFree[Max\_Page\_Per\_File]。

* fileId： 文件的ID号
* pageOfFile： 文件中包含的page数
* freeCount： 文件中可用的文件数
* isPageFree[Max\_Page\_Per\_File]： page的位图数组，只存储0或者1

表1 文件空闲块管理示意图

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| fileId=1, pageOfFile=21, freeCount=15, isPageFree=[11111000000............] | | | | | | |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

## 四、代码及函数列表

**Structfile.h 文件结构头文件**

**Memblock.h 缓冲区管理头文件**

**Datafiles.c 文件管理源文件**

* isFileExist 函数 判断文件是否存在
* createSysFile 函数 创建系统文件
* saveSysFile 函数 保存系统文件
* readSysFile 函数 读系统文件
* createDataFile 函数 创建数据文件
* getFreePage 函数 获得空闲页
* writeNewPageToFile函数 往文件写新页
* readPageFromFile 函数 从文件读一页
* displayDBInfo 函数 显示数据库信息
* initDataFile 函数 初始化数据文件

**Buffmgr.c 缓冲区管理源文件**

* initQueue函数 初始化队列
* EnterQueue函数 入队
* DelQueue函数 出队
* allocateBuff函数 分配缓冲区
* queryFreeBuff函数 查询空闲缓存
* buffSwith 函数 缓冲区置换
* initMemory函数 初始化缓冲区

## 实验中遇到的问题

### 5.1 数据文件的自动扩展

实验中，考虑到了初使化文件时指定大小，如果需要存放的数据大于当前大小时，数据文件并不能自动扩展大小。

**改进方法：**因为我们的设计中，每个表存放在一个文件中，可以为每个文件设置一个auto\_extent 参数，指定是否可以自动扩展，这样，当需要额外的空间时，就可以用文件操作的fseek函数来进行文件的扩展。

### 5.2 空闲空间回收

本实验中，设计用“高水平(High water mark)”来实现空闲空间的回收：高于高水位之上的空间全部是未使用空间，可以直接回收。但是，如果表中存在大量的Delete操作时，高水位之下也会出现很多空闲空间，此时就会出现文件碎片，对文件碎片的回收则需要大量的块移动，消耗大量的I/O操作，由于块移动实现较为复杂，相当于整个文件的重组，所以实验中并未实现文件重组方式的空闲空间回收。

### 5.3 内存空间的定位问题

在内存中进行指针移动时，由于指针一次只能移动一个数据类型大小的位置，例如定义int \*p时，p是一个指向int类型的指针，当执行p++时，指针p向前移动一个Int大小的位置，即4字节，但是不能实现每1个字节的移动，即不能实现内存的精确定位。而且内存中需要存放的数据类型也不能预先知道，需要根据用户的输入来确定存放类型。

**解决方法：**分配缓冲区时，统一使用char[buff\_size]来预先分配缓冲区，由于char刚好占用一个字节，即可以实现字节粒度的内存移动；另外，使用char分配内存后，也可以存放其它类型的数据，例如可以使用4个char存放int类型的数据，存放完成之后，指针向前移动4个位置，即可存放其它类型的数据，这样即可实现多类型的数据。

### 5.4 如何确定数据文件中的不同数据类型

每一个文件中存放的元组类型千差万别，如何确定每一个位置存放的数据类型是一个问题。

**解决方法：**为每一类元组（或一个表）保存一个数据字典，存放在系统文件中，通过元组的大小和元组内部数据类型的大小，通过偏移量即可确定元组中任意变量的位置。