实验一《存储管理》实验报告

目录

组长：张恒

组员：陶友贤、修晔良、赵旺

1 实验内容

- ✧ 数据文件的组织
- ✧ 缓冲区管理
- ✧ 空闲空间管理

2 实验要求

2.1 数据文件的组织

a) 段页式组织方式

b) 支持基本数据类型，可不支持大对象数据

2.2 缓冲区管理

a) 缓冲区页面组织

b) 缓冲区查找

c) 缓冲区淘汰

2.3 空闲空间管理

a) 空闲空间组织

b) 空闲空间分配

c) 空闲空间回收

3 实验环境

a) 操作系统平台：windows 10

b) 编程语言：C++

c) 程序开发环境：Visual Studio 2013

4 实验原理

4.1 数据库管理系统

数据库管理系统(Database Management System)是一种操纵和管理数据库的大型软件，用于建立、使用和维护数据库，对数据库进行统一的管理和控制，以保证数据库的安全性和完整性。用户通过 DBMS 访问数据库中的数据，数据库管理员也通过 DBMS 进行数据库的维护工作。它可使多个应用程序和用户用不同的方法在同时或不同时刻去建立、修改和访问数据库。大部分 DBMS 提供数据定义语言 DDL（Data Definition Language）和数据操作语言 DML（Data Manipulation Language），供用户定义数据库的模式结构与权限约束，实现对数据的追加、删除等操作。

数据库管理系统是数据库系统的核心，是管理数据库的软件。数据库管理系统就是实现把用户意义下抽象的逻辑数据处理，转换成为计算机中具体的物理数据处理的软件。有了数据库管理系统，用户就可以在抽象意义下处理数据，而不必顾及这些数据在计算机中的布局和物理位置。

4.2 DBMS 的层次结构

根据处理对象的不同，DBMS 的层次结构由高级到低级依次为应用层、语言翻译处理层、数据存取层、数据存储层、操作系统。

- ✧ 应用层：DBMS 与终端用户和应用程序的界面层，处理的对象是各种各样的数据库应用。
- ✧ 语言翻译处理层：对数据库语言的各类语句进行语法分析、视图转换、授权检查、完整性检查等。
- ✧ 数据存取层：将上层的集合操作转为单记录操作。处理对象是单个元组。
- ✧ 数据存储层：处理的对象是数据页和系统缓冲区，即本次实验一的重点。
- ✧ 操作系统：DBMS 的基础，提供的存取原语和基本的存取方法通常是作为和 DBMS 存储层的接口。

4.3 数据组织、存储与管理

在数据存储层上，DBMS 要分类组织、存储和管理各种数据，包括数据字典、用户数据、存取路径等，需确定以何种文件结构和存取方式在存储级上组织这些数据、如何实现数据

之间的联系。数据组织和存储的基本目标是提高存储空间利用率，选择合适的存取方法提高存取效率。

4.4 缓冲区管理

密集的磁盘 I/O 操作是数据库引擎的一大特点。而完成磁盘 I/O 操作要消耗许多资源，且耗时较长。缓冲区管理是实现高效 I/O 操作的关键环节。缓冲区管理器负责将数据页或索引页从数据库磁盘文件读入缓冲区高速缓存中，并将修改后的页写回磁盘。缓冲区缓存中会保留一页，直到缓冲区管理器需要该缓冲区读入更多的数据。数据只有在被修改后才会重新写入磁盘。在将缓冲区高速缓存中的数据写回磁盘之前，可对其进行多次修改。

4.5 空闲空间管理

空闲空间管理主要是为了实现存储空间的分配和回收，DBMS 会为存储空间设置相应的结构以记住存储空间的使用情况，并配以相应算法方便地对存储空间进行分配和回收。常见的空闲空间管理方法主要有空闲表法、位示图法、空闲块链表法、链接索引块法。本次实验采用的是位视图法。

5　程序设计

5.1　数据组织方式

在我们所实现的 DBMS 中，采用的是“文件—段—页”的存储形式来对数据进行组织、存储等操作。

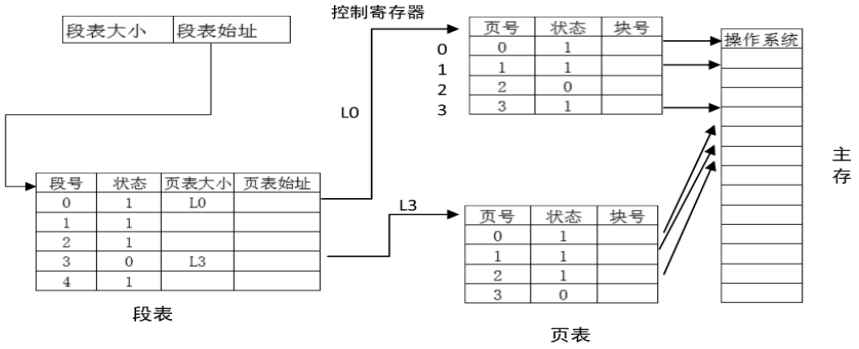
- 【文件】

在 DBMS 中，为当前数据库用户所创建的每个表都建立一个文件，每个文件的相关信息会记录在系统头部(struct DbInfo)，同时也会记录将当前的文件数量等参数记录在数据库头部(struct dbMetaHead)中。

初始建立文件时，会分配一个段中的一个页给文件。当删除一个文件时，需要回收它的文件号，以及它所占用的段里的每个页，即在空闲空间位示图中重置标记位。

- 段页式存储

段页式存储组织是分段式和分页式结合的存储组织方法，这样可充分利用分段管理和分页管理的优点。如下图所示



(1) 用分段方法来分配和管理虚拟存储器。程序的地址空间按逻辑单位分成基本独立的段，而每一段有自己的段名，再把每段分成固定大小的若干页。

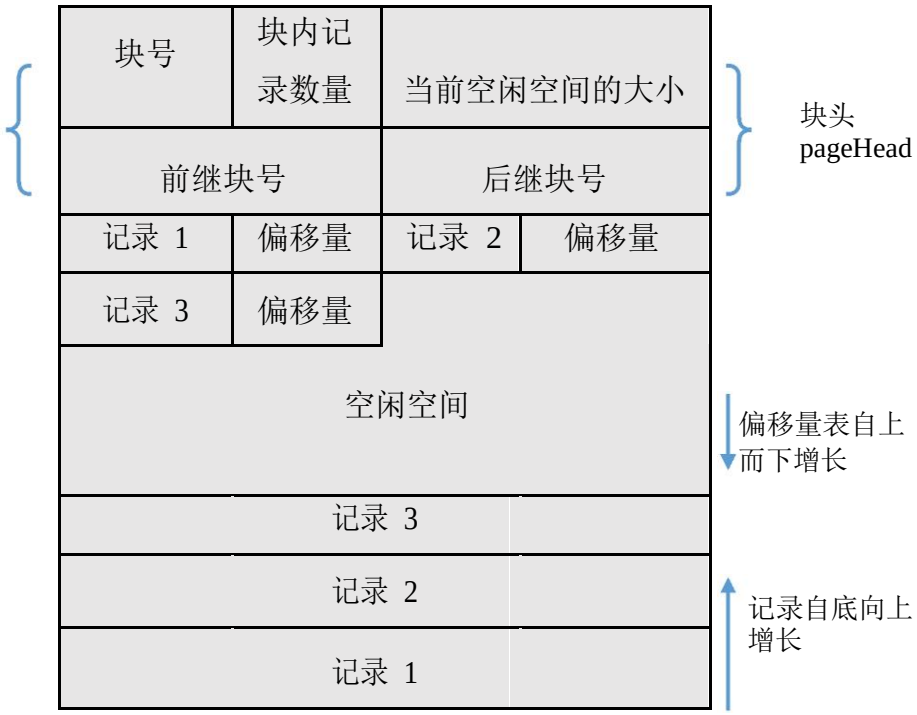
(2) 用分页方法来分配和管理实存。即把整个主存分成与上述页大小相等的存储块，可装入作业的任何一页。程序对内存的调入或调出是按页进行的。但它又可按段实现共享和保护

当向某一段中插入大量数据而导致空间不够用时，DBMS 会动态地为其扩展空间，即

再给这个段分配连续的空闲页，并将新页信息添加到段表最尾部，因此一个段所包含的页可能地址不连续。

- 【页/块】

页/块是 I/O 操作的最小单位，页和块是不同情况下的表述，实际大小都规定为 4KB，在磁盘中称为页，在内存、缓冲区中则称为块。每一页的组织方式如下所示，共包含三大部分：页头、偏移量表和记录。



其中，每块有一个固定大小的块头(struct pageHead)，记录了该块的块号、块包含的记录数量、当前的空闲空间大小，以及前继块号和后继块号。这样的定义是因为文件的大小（即所占用的页数）是动态增长的，因而在文件的相关信息中，DBMS 只需记录该文件的起始块号，然后通过读块头信息中的后继页号来找到下一块，以此类推，用这种顺序的方式来找到属于该文件的所有块。

从块尾开始，逐条插入记录，同时会为它分配一条偏移量表项，存储了它在当前页内的记录号和相对于页尾地址的偏移量大小，并将该偏移量表项从页头结尾处开始，依次向后面的空闲空间中插入。这样，通过偏移量表就可以读取页内的每条记录了。宏观地来看，只要给定块号和块内记录号，DBMS 就可以唯一地确定一个物理地址，进而找到一条记录。

- 【记录】

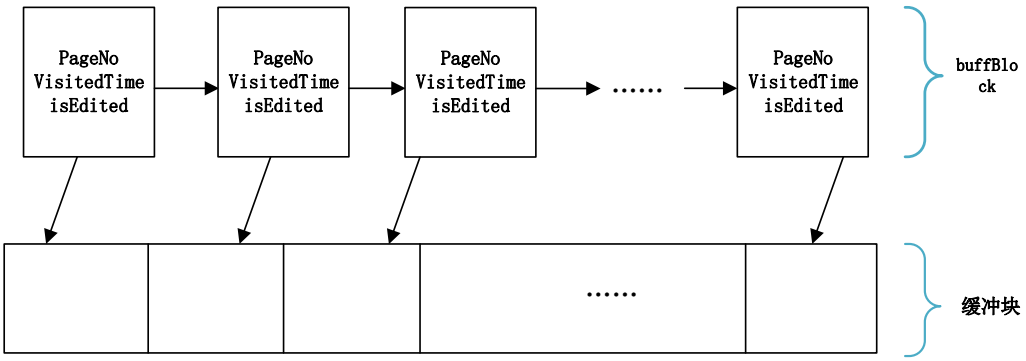
因为用户在数据库中创建表时会自行规定每条记录的模式，即每个字段的类型、大小等，计划是令 DBMS 将当前数据库中创建的所有表模式单独存储到一个文件中。用户在对数据进

行操作时，例如执行 select name from table1 的语句，首先会去找到 table1 对应的表模式，然后进行字段解析等相关操作，得到用户指定的字段，再去对数据进行读取等。

目前在进行读写文件时，会将所有数据预处理成字符串形式，再作为参数传入，然后 DBMS 会进行相关读写操作。但上述功能是一个数据库管理系统必须要实现的，因此也是我们接下来的重点。

5.2 缓冲区管理

DBMS 为每个数据库分配一定大小的内存作为缓冲区，内存的管理单元大小与磁盘存取粒度保持一致，即缓冲区块与页大小相同。另外，为了管理缓冲区，需要维护一个数组(类似于映射表），每个元素是一个描述块(struct buffBlock)，与一个缓冲区块相对应，每个描述块记录着其对应的缓冲区块中存放的页号、这一页停留在内存中的时间以及其是否被编辑过



定义结构体

```
//数据库系统的相关参数、空闲空间位示图、缓冲区、文件指针
struct dbMetaHead{
struct DbInfo desc;
unsigned long *FreeSpace_bitMap;
struct bufferPool buff;
FILE *datapath;
};
```

```
// 数据库系统的一些参数（有一些是固定值，但有一些会变）
struct DbInfo{
   Struct FileInfo fileInfo[MAX_FILE_NUM];
    Long fileSize;//文件中数据区的大小，默认 196MB
    Long pageSize;//每一个页的大小，默认为 4KB
    Long pageSum;//总共页数
    Long pageFree;//当前有多少可用的页
```

```
    Long bitMapAddr;//文件中空闲空间位示图的起始地址
    Long bitMapSize;//bitMap 的大小，以字节为单位
    Long dataAddStart;// 文件中数据区的起始位置
    int curFileNum;//目前有文件数目
};
```

```
  struct FileInfo{
      int fileType;   // 标识是否为索引文件，以及索引文
  件的类型（hash/b 树/顺序,线性）
      int fileID;   // 文件号
      int fileName;                  // 文件名
      int fileState;                  // 文件状态
      int fileSegNum; // 文件占用了多少段
      struct Segment segTable[SEG_NUM];//定义段
  };
```

```
/* (页表头部信息)
*/
struct pageMateHead{
      long pageNo;  // 页号
      int curRecordNum; // 当前该页存储的记录个数
      long prePageNo;// 前继页号
      long nextPageNo;// 后继页号
      long freeSpace;// 该页的空余空间大小
};

/* (页表中)
 * 块内的偏移量表从块的前端向后增长
 * 块内的记录是从后向前放置
 */
struct OffsetInPage{
      int recordID;
      int offset;   //该记录相对于块尾地址的偏移量
      bool isDeleted;
};
```

```
  /* (缓冲块信息)
  */
  struct bufferBlock{
      long pageNo;
      long visitTime;
      bool isEdited;
  };
```

```
/* (缓冲区信息)
 * 记录
*/
struct bufferPool{
    struct bufferBlock map[BUFFER_Size];
    char data[BUFFER_Size][PAGE_SIZE];
    long curTimeStamp;
};
```

5.3 相关函数与功能模块

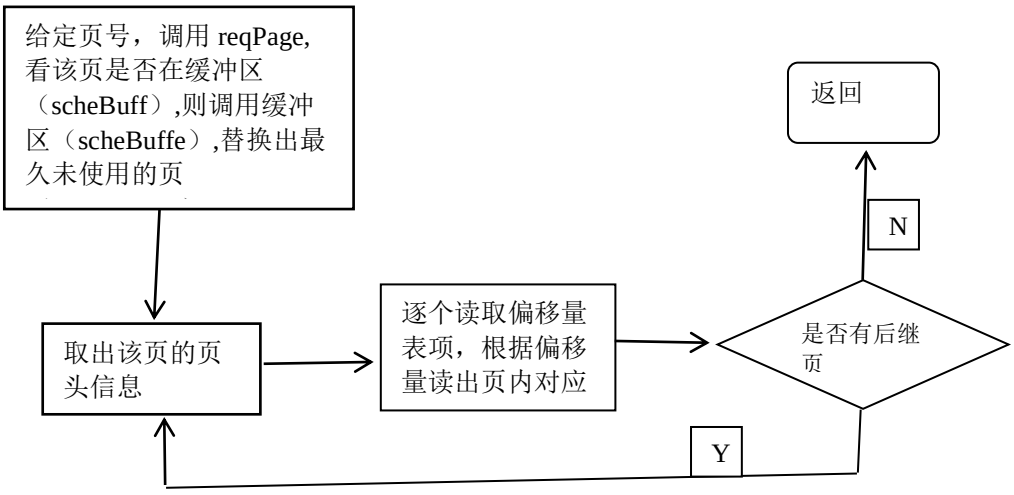
```
//storage.cpp
void createDbMetaHead(char *filename);
void init_database(struct DbMetaHead *, char *filename);
void showDbInfo(struct DbMetaHead *);
//segmentOpt.cpp
int createFile(struct DbMetaHead *, int type ,long reqPageNum);
int queryFileIndex(struct DbMetaHead *, int fid);
void writeFile(struct DbMetaHead *, int fid, int length, char *);
void readFile(struct DbMetaHead *, int fid, char *des);
void deleteFile(struct DbMetaHead *, int fid);
//bufOpt.cpp
int queryPage(struct DbMetaHead *head, long queryPageNo );
int replacePage(struct DbMetaHead *, int mapNo, long pageNo );
int scheBuff( struct DbMetaHead *head );
int reqPage( struct DbMetaHead *head, long queryPageNo );
//pageOpt.cpp
int getBit(unsigned long num, long pos);
int setBit( unsigned long *num, long pos, int setValue);
long allocatePage( struct DbMetaHead *head, long reqPageNum);
void recyOnePage( struct DbMetaHead *head ,long pageNo);
void recyAllPage(struct DbMetaHead *head);
```

● 数据库初始化（init_database）:从给定的文件中读取数据库的相关参数，若文件不存在，则创建一个数据库头（createDbMataHead）,并设置相关参数，写回文件。

● 创建文件（createFile）:给定文件类型可能为用户表文件，索引文件、存储表模式的文件，暂时是只考虑了用户文件，为其分配页的过程中，重点需要设置页头信息，记录前继、后继页号等，便于后序的文件管理。

● 向文件中写记录（writeFile）:给定文件号，在调用函数之前会对记录进行预处理，然后将其以字符串的形式传入，并给定长度，写入该文件的合适位置中。向指定文件中插入一条记录的流程如下：

● 读取文件中所有的记录（readFile）



● 删除文件（deleFile）:遍历文件的所有页，将其在空闲空间位示图 bitMap 中的标记位重置，收回文件号，更新数据库系统的头部参数。

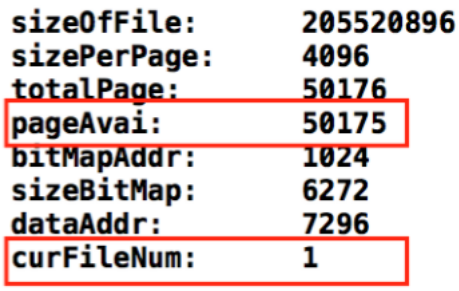
● 与单条记录相关的增删改查操作，待实现中…

6 实验结果

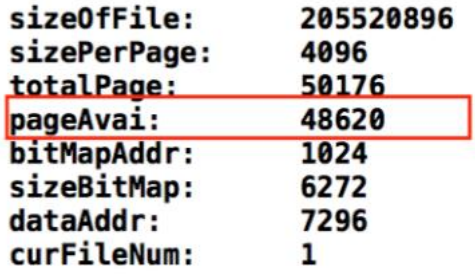
1. 初始化数据库。

```
sizeOfFile:      205520896
sizePerPage:     4096
totalPage:       50176
pageAvai:        50176
bitMapAddr:      1024
sizeBitMap:      6272
dataAddr:        7296
curFileNum:      0
```

2. 创建文件后，系统参数的变化



3. 插入了 20 万条记录后，文件 0 的一些参数，以及系统的一些参数变化



4. 按页读取文件 0 存储的全部记录

```
Struct employee{
    Long rid ;
   Char name[100];
   Int age;
   Int wage;
};
```

--------页头信息-------
页号：690
页的空余空间：6
该页当前存储的记录个数：135
偏移量表------记录内容

记录号	偏移量	是否删除	
0	18	0	94033abc9406399033
1	36	0	94034abc9406499034
2	54	0	94035abc9406599035
3	72	0	94036abc9406699036
4	90	0	94037abc9406799037
5	108	0	94038abc9406899038
6	126	0	94039abc9406999039
7	144	0	94040abc9407099040
8	162	0	94041abc9407199041
9	180	0	94042abc9407299042
10	198	0	94043abc9407399043
11	216	0	94044abc9407499044
12	234	0	94045abc9407599045
13	252	0	94046abc9407699046
14	270	0	94047abc9407799047
15	288	0	94048abc9407899048
16	306	0	94049abc9407999049
17	324	0	94050abc9408099050
18	342	0	94051abc9408199051
19	360	0	94052abc9408299052
20	378	0	94053abc9408399053
21	396	0	94054abc9408499054
22	414	0	94055abc9408599055
23	432	0	94056abc9408699056
24	450	0	94057abc9408799057
25	468	0	94058abc9408899058
26	486	0	94059abc9408999059
27	504	0	94060abc9409099060
28	522	0	94061abc9409199061
29	540	0	94062abc9409299062
30	558	0	94063abc9409399063
31	576	0	94064abc9409499064
32	594	0	94065abc9409599065
33	612	0	94066abc9409699066
34	630	0	94067abc9409799067
35	648	0	94068abc9409899068
36	666	0	94069abc9409999069
37	684	0	94070abc9410099070
38	702	0	94071abc9410199071
39	720	0	94072abc9410299072
40	738	0	94073abc9410399073
41	756	0	94074abc9410499074
41	756	0	94074abc9410499074
42	774	0	94075abc9410599075
43	792	0	94076abc9410699076
44	810	0	94077abc9410799077
45	828	0	94078abc9410899078
46	846	0	94079abc9410999079
47	864	0	94080abc9411099080
48	882	0	94081abc9411199081
49	900	0	94082abc9411299082
50	918	0	94083abc9411399083
51	936	0	94084abc9411499084
52	954	0	94085abc9411599085
53	972	0	94086abc9411699086
54	990	0	94087abc9411799087
55	1008	0	94088abc9411899088
56	1026	0	94089abc9411999089
57	1044	0	94090abc9412099090
58	1062	0	94091abc9412199091
59	1080	0	94092abc9412299092
60	1098	0	94093abc9412399093
61	1116	0	94094abc9412499094
62	1134	0	94095abc9412599095
63	1152	0	94096abc9412699096
64	1170	0	94097abc9412799097
65	1188	0	94098abc9412899098
66	1206	0	94099abc9412999099
67	1224	0	94100abc9413099100
68	1242	0	94101abc9413199101
69	1260	0	94102abc9413299102
70	1278	0	94103abc9413399103
71	1296	0	94104abc9413499104
72	1314	0	94105abc9413599105
73	1332	0	94106abc9413699106
74	1350	0	94107abc9413799107
75	1368	0	94108abc9413899108
76	1386	0	94109abc9413999109
77	1404	0	94110abc9414099110
78	1422	0	94111abc9414199111
79	1440	0	94112abc9414299112
80	1458	0	94113abc9414399113
81	1476	0	94114abc9414499114
82	1494	0	94115abc9414599115
83	1512	0	94116abc9414699116
84	1530	0	94117abc9414799117
85	1548	0	94118abc9414899118
86	1566	0	94119abc9414999119
87	1584	0	94120abc9415099120
88	1602	0	94121abc9415199121

7 实验感想

首先，从最开始的随机组队到现在实验一的基本完成，可以说我们团队真的是经历了很多，有过激烈的讨论与争执，更有代码成功运行时的喜悦与欢呼。从老师布置实验一开始，我们团队就已经开始讨论怎样去实现，从最开始的毫无头绪到渐渐摸索到了一些实现思路。在最开始的时候，我们主要讨论怎样去设计系统的各种数据结构，如：文件结构、缓冲区结构、块与记录数据结构等。因为我们团队认为文件、记录、块等数据结构的设计是整个数据库实现系统的基石。因此我们也在这部分讨论了很长时间，以及讨论各个数据结构之间的交互与通信。然后接下来的任务就是怎样具体实现的问题了。我们团队采用的方式主要是分组进行实现各个模块进而整合的方式进行。

虽然我们实验还有很多需要改进的的地方，但实验所要求的基本功能也都已经实现了。经历过实验一的这一过程，我们不仅编程能力得到了提升，而且对整个系统的实现也有了更加完整的把握。所以希望我们接下来的实验会做的更加的完善。