**Buffer manager**

**Buffer Manager负责缓冲区的管理，主要功能有：**

1. 根据需要，读取指定的数据到系统缓冲区或将缓冲区中的数据写出到文件

2. 实现缓冲区的替换算法，当缓冲区满时选择合适的页进行替换

3. 记录缓冲区中各页的状态，如是否被修改过等

4. 提供缓冲区页的pin功能，及锁定缓冲区的页，不允许替换出去

为提高磁盘I/O操作的效率，缓冲区与文件系统交互的单位是块，块的大小应为文件系统与磁盘交互单位的整数倍，一般可定为4KB或8KB。

**实现的基本思想是：**

1. 定义buffer由50个block组成。每个block为4KB。
2. 当RecordManager或IndexManager需要取数据时，判断数据block是否已经在buffer中，如果在则直接返回block地址；如果不在，则从磁盘中将文件数据读入buffer，再返回block地址。
3. 在向buffer中读入数据时，要判断是否有空block，如果没有，需要使用LRU算法进行替换。
4. 要实现LRU算法，可以采用block\_list的形式，每次访问一个block，就将这个block插到链表头，这样链表尾部即为最近最少使用的block。

**类的设计如下：**

1. block类，包含了一个block的结构，以及将block数据写回到磁盘文件的函数writeback()。
2. file类，包含一个文件头的信息。
3. bufferMan类，继承block类，实现对buffer的管理。包含的函数有：

block \*findBlock():在buffer中找到一个空block或者用LRU算法替换出一个空block；

writetoBuffer(block \*block\_toBeWritten)：将一个block写入buffer中；

block \*getFromBuffer(block \*block\_toBeRead)：从buffer中读一个block；

file \*getFile(file \*file\_toBeRead)：从buffer中读一个文件；

CloseFile(file \*file\_toBeClosed)：依次释放要关闭的文件包含的block。

**伪代码：**

class block{

//block的结构：

int Num; //给每个block分配一个块号

bool dirty; //判断是否为脏块

bool written; //判断该块是否被修改过（修改过的block需要writeback）

bool lock; //判断block是否锁定

CTime uTime; //记录block最近一次使用的时间

char \*ch; //所有数据以char类型存入

file \*filename; //指向该block所属的文件

block \*next; //指向下一个block（构成block\_list）

public:

block(){

对block进行初始化

}

~block(){

释放block的内容

}

writeBack(){

如果块没有更新过，则释放块

如果块更新过，则将块中数据写入指向的文件，再释放块。

}

}

class file{

//file头的结构：

CString filename;

int type; //用一个int值来表示文件的类型是record还是index

int recordNum; //记录record的总数

int recordLength; //记录每条record的长度

block \*first; //指向文件第一个block

public:

file(){

对文件头进行初始化

}

~file(){

释放文件头

}

}

Class bufferMan: public block {

public:

//对数据块的操作：

block \*findBlock(){

如果有空块，则返回空块指针

如果没有足够空块，则用LRU算法寻找到上次使用时间最远的block，更新时间，将脏块的数据writeBack()，并返回替换后新的空块的指针

}

writetoBuffer(block \*block\_toBeWritten){

//将一个数据块写入buffer

findBlock()，将record或index文件的一个块的数据写入到空块中，更新块的信息。

}

block \*getFromBuffer(block \*block\_toBeRead){

要从buffer中读数据，先判断该数据块是否已经在buffer里。

如果该block在buffer中，直接返回block位置

如果不在buffer中，writetoBuffer()将该block写入buffer，返回该block的位置。

}

//对文件的操作：

file \*getFile(file \*file\_toBeRead){

如果该文件已经在buffer，则返回文件第一个block位置

如果文件不在buffer，检查文件是否存在

如果文件不存在，则newFile()

如果文件存在，则findBlock()，将文件数据依次读入空block，插在block链表头，然后返回文件第一个block位置。

}

CloseFile(file \*file\_toBeClosed){

将文件包含的块依次释放。

}

}