Buffer manager 设计报告

--计算机科学与技术学院 计算机科学与技术专业 1405 班 沈栋 3140102265

1. 模块概述

Minisql 作为一个数据库关键的是对数据的处理，而内存中的数据处理快，但是容量有限，而磁盘中数据的容量大，但处理慢，如何利用着而 Buffer manager 就是在管理内存 和磁盘中间的数据交互。提高数据处理的效率

2. 主要功能

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) 向 Recorder Manager、Index Manager 提供供写入的块。

3. 对外提供的接口

```
初始化：void initialize(int );

读入指定文件 table 或者 index 的 block：
int malloc_block(int& biggestnum);
int malloc_block(int &, int &);

为 table 或者 index 创建一个新的 block:
int create_newblock_table(Table &T);
int    create_newblock_index(Index &I);

将 index 或者 block 的块写回磁盘
bool write_block_record(int &biggestnum,int &index);
```

```
bool write_block_index(int &biggestnum, int &index);

将缓冲区的 block 写回磁盘中
bool write_block_index(int &biggestnum, int &index);

得到正确插入地址
int    get_insert_position(int &biggestnum,int &type);

检测该 block 是否在内存中
int check_block_memory(int &biggestnum,int &type);
更新 LRU:
int    update_LR(int& first)
```

4. 设计思路

a) Buffer 的关键是缓冲块的管理，在这个实验中我们采用了 LRU 算法，替换最远被访问的块

b) 整体的数据构造是定义了一个 buffer 类，类里面有 block 的结构体的数组，用来存储信息

c) 具体构造可以见 5

d) 在程序启动的时候实例化一个 buffer，泽 buffer 里面的 block 数组就是每一个具体的 block

e) 每次 index 和 record 去申请一个 block 的时候，若其在内存中，直接返回编号，通过编号访问 block，若不在内存中，分 2 种情况

f) 内存满了 ,根据 LRU ,替换一个 ,将替换的写入文件中 ,将对应的下标返回 igei 调用者

g) 没满，按顺序给下一个 block，并且将 blokc_valid_number++;

h) 每次程序结束的时候或者 block 被替换的时候会将所有写过的 block 写入磁盘

i)

5. 整体架构：

内存中的构架

Buffer->Block,buffer 这样一个类中有多个 block 来作为内存的基本单位来交互管理

Block 的节点实现

```
typedef struct node Block;
struct node{
 //string filename;  //index or recod   1 表示 index，  2 表示 record
 int biggestnum;          // block 的实际编号，文件编号+block 的偏移量
 int blockOffset;     //which block offset in file，  写回文件中要用
 int type;            //标记是 index 还是 record，写回内存中要用，  type=1 是 index，type=2 是 record
 int blockindex;          //block index in memory     内存中的第几号 block
 bool isWritten;          //是否被修改过，如果被修改过就要被写会内存
 int vaild_number;    //block  中有效记录到哪里
 char block[block_size];
};
```

Buffer 的实现

```
class buffer{
public:
 string filename;

 Block b[max_number_block_memory];
 int valid_numer_block;
      ..............
private:
 int LRUvalue[max_number_block_memory];//用于实现 LRU 算法,the lower, the better
```

6. 关键函数和代码

关键函数是去磁盘中读入和写回磁盘以及 LRU 算法的实现

从磁盘中读入

int malloc_block(int& biggestnum);

判断是否在内村中，在直接返回编号，

不在

调用 get_insert_position 函数得到可以写的块编号

如果该块为脏，就将其写会磁盘

根据对应的 biggestnum 得到 filename

打开 file

根据 biggestnum 得到块的偏移量*块的大小移动文件指针到对应位置

读入数据到对应的块里

返回块的编号

int malloc_block(int &, int &); //重载函数，针对 index 的磁盘读，实现思路一样

int get_insert_position(int &biggestnum,int &type)

查看该 block 是否在内存中，不在的话直接返回下一个内存中 block 编号，作为输出，同时更新 LRU

不然的话找到 LRU 最小的编号，并将 LRU 更新

bool write_block_record(int &biggestnum,int &index);

根据对应的 biggestnum 得到 filename

打开 file

根据 biggestnum 得到块的偏移量*块的大小移动文件指针到对应位置

写入数据到对应的块里

返回写入状态

bool write_block_index(int &biggestnum,int &index);

实现思路与 write_block_record 一样

bool write_block_index(int &biggestnum, int &index);

遍历一遍内存，将所有的脏数据写出去，调用前面 2 个函数

int update_LR(int& first);

将下标 first 的 LRU 置 1，其他都是加 1