《高级数据库系统》课程实验

~

实验目的

在这个项目中，我们将实现一个简单的存储和缓冲区管理器。将涉及到缓冲区和帧大小、缓冲区和帧存储结构、页面格式、页面存储结构、文件格式、缓冲技术、哈希技术、文件存储结构以及磁盘空间和缓冲区模块的接口功能。特定技术是从课堂中涉及的与缓冲区和存储管理器相关的材料中选择的。

实验环境

硬件：

CPU	**1.8 GHz Intel Core i5**
内存	**8 GB 1600 MHz DDR3**

软件：

操作系统	**macOS Mojave**
开发工具	**Xcode Version 11.2.1 (11B500)**
开发语言	**c++**

实验设计

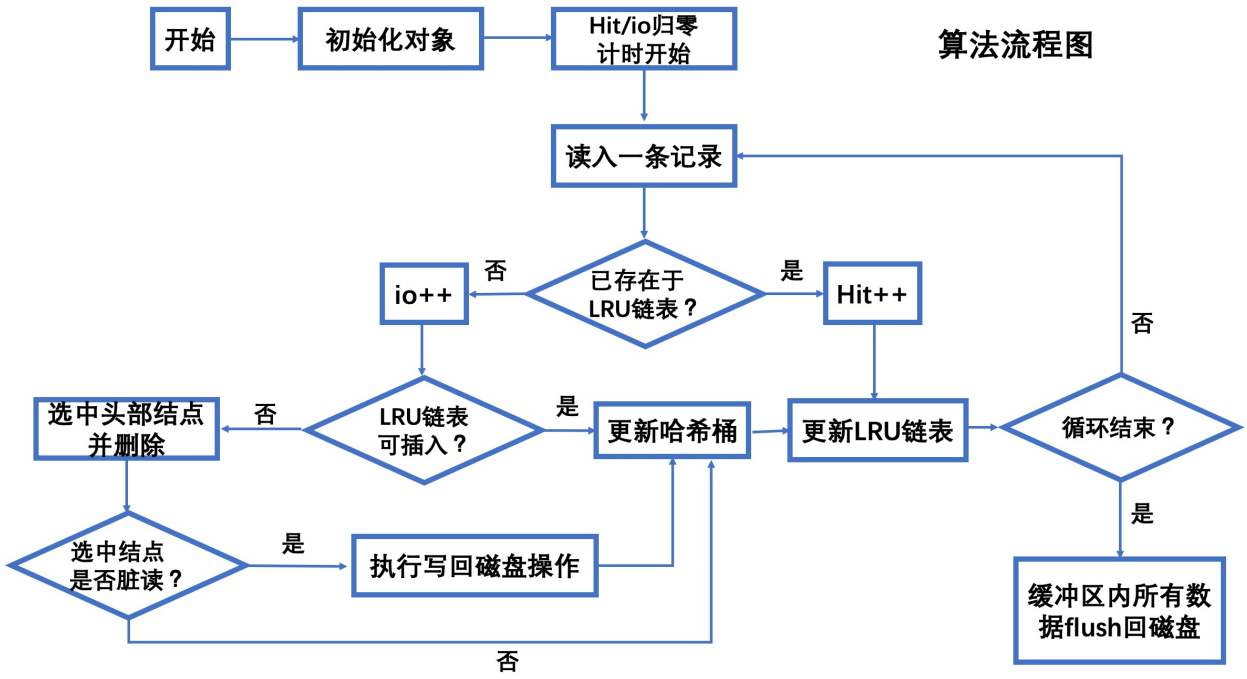
#define PAGEREF 500000

#define DEFBUFSIZE 1024 //2048、4096

#define FRAMESIZE 4096

#define PAGESIZE 4096

主程序总体结构如下：



1.初始化tracesetup对象，并读入data-5w-50w-zipf.txt文件，并将每条记录的读写信息和page_id存入预定义的大小为500000的trace数组中。

2.初始化BMgr对象，并对对象内部的私有数组ftop和ptof进行初始化。其中ptof存储指向BCB的指针数组；ftop存储在该frame_id中存储的page_id的信息。两者大小都为1024。

3.初始化DSMgr对象，增添一块大小为4096*(1024+2)字节的内存空间，并将其中的内容写入data.dbf文件中，初始化da ta.dbf。

4.初始化LRU对象及其内部私有的头部指针和尾部指针。在一个LRU对象中，头部指针和尾部指针不存储实际数据，只是为了便于算法设计。

5.io及hit归零，开始计时。

6.遍历trace数组的每一条记录record。对于一条记录来说，判断其是否已存在于LRU双向链表中：

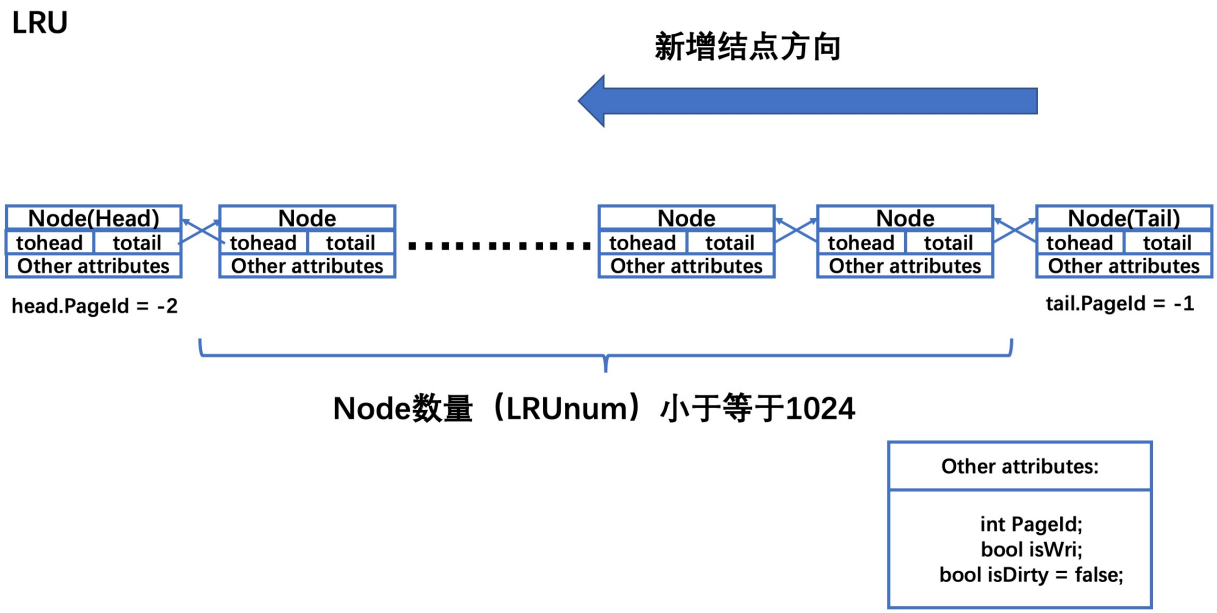
如果不存在，则io++，并再判断当前LRU是否是可以插入的（即长度小于1024），如果可以插入，则对链表进行新增结点的操作，并对结点进行赋值等操作，再判断该记录是读还是写来对结点进行脏读处理，同时也将该结点对应的page_id通过哈希函数映射为frame_id，通过frame_id将该数据块更新入对应缓冲区，并更新ftop数组；如果不可以插入，则先选定需要被置换出的结点，判断该结点是否为脏读，不是的话就不需要写回磁盘，是的话再写回磁盘即可，之后在BCB和LRU中删除该结点，引进新增结点，同样进行脏读判断，并更新ftop数组。

如果一条记录对应的page_id已经存在于LRU双向链表中，hit++，抽提到队尾，再次进行脏读判断。

7.所有记录遍历完成后，将LRU内存留的所有结点对应的内容全部再写回到磁盘中，计时结束。

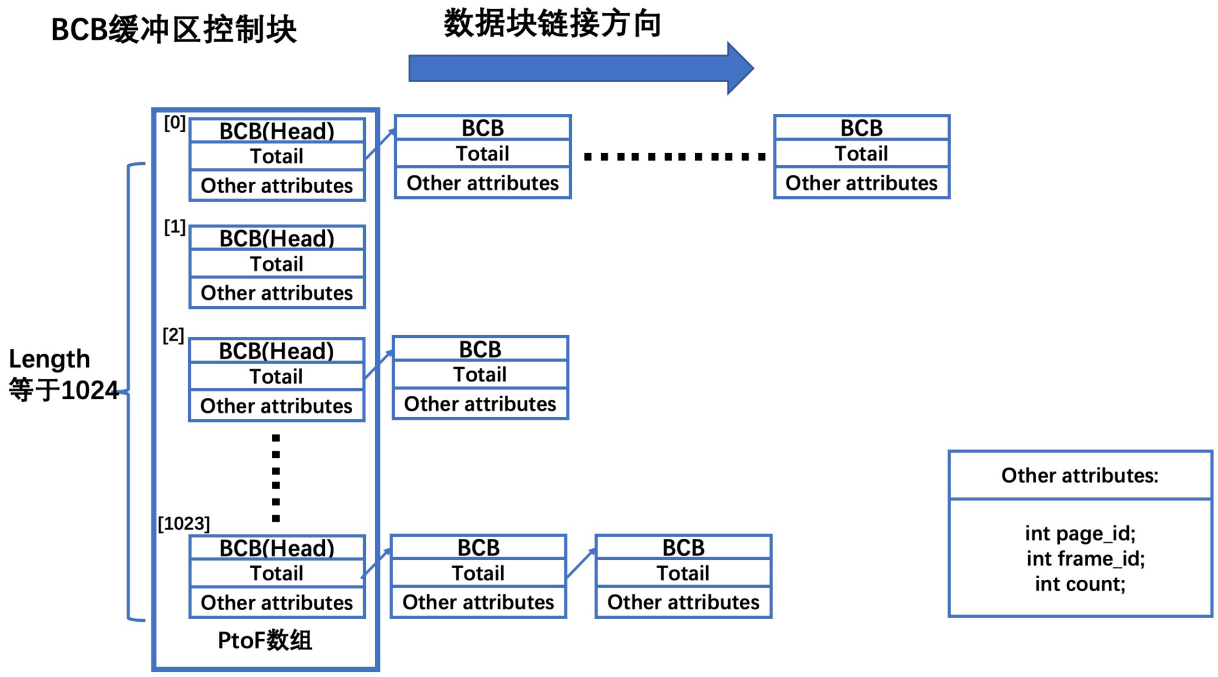
主要数据结构

LRU双向链表结构



一个LRU对象包括了诸如插入结点、丢弃结点、判断结点是否可插入等一些公共方法，同时也包含一个头部结点(page_id=-2)和一个尾部结点(page_id=-1)用于标记临界。一个节点包含着指向头部的下一个指针(tohead)、指向尾部的上一个指针(totail)，还有page_id、是否读写的isWri以及判断该结点是否为脏数据的isDirty。

BCB单向链表结构



缓冲区控制块中，为了执行方便，控制ptof哈希映射表数组中的元素类型与将要链接的新的数据块的类型一致，该类型(BCB)中的属性有指向下一个数据块的指针(Totail)、代表每个数据块的page_id、frame_id，还有当前frame中的数据块数量count(仅对数组中的BCB进行操作)。数组中的每个BCB指向一个特定的哈希桶，在整个记录遍历的过程中，使用哈希函数将数据块页号映射为frame_id，进入特定的哈希桶，链入哈希桶中单向链表的尾部。

设计实现

类的实现

本次实验一共设计实现了4个大类

类名	功能
tracesetup	初始化存储记录的数组trace[500000]，并读入全部的记录数据
LRU	初始化LRU链表，并提供方法便于对LRU双向链表进行操作
BMgr	初始化存储哈希桶的BCB数组，并提供方法便于对缓冲区控制块进行操作
DSMgr	初始化文件操作的私有对象，提供数据读写操作的方法

data.dbf

data.dbf文件存储该缓冲系统中产生的所有数据块，大小为4096*50000=195MB大小，因此能够存储全部的记录信息，每条记录信息由页号作为偏移量分隔开，每个数据块都占用4096字节。

首先，在程序运行之前先创建data.dbf并向data.dbf文件中写入全部的数据；

之后，在程序运行中，如果出现脏读数据，则会将对应页号的数据块进行覆写；

在程序运行结束时，将所有残存在缓冲区中的数据块全部写回至data.dbf。

算法实现细节

1.本实验开始时一次性读取所有的记录数据存于数组中，避免实验进行时反复对数据文件进行操作。

2.对LRU算法使用双向链表进行操作，对于BCB缓冲区控制块使用单向链表进行操作。

3.只有当要被置换出去的数据块是脏读数据时才会执行写回磁盘的操作，非脏读数据不执行写回，因为实际上该数据在操作前后没有变化；但在算法结束后，会将残留在缓冲区中的数据（不管脏读与否）全部flush回磁盘。

4.为了正常运行脚本，请手动将位于**tracesetup.hpp**、**DSMgr.hpp**中的**文件路径变量**设置成为有效路径。

实验结果

整个算法的运行时间大概在**60～70s**之间，最终hit数为**169565**,hit rate为**33.91%**，io数为**330435**，buffer size**为1024.**

```
处理记录序号: 499996
处理记录序号: 499997
处理记录序号: 499998
处理记录序号: 499999
算法运行时间: 63.7881s
buffer size: 1024
hit数: 169565
hit rate: 33.913%
io数: 330435
Program ended with exit code: 0
```

data.dbf文件的大小在整个程序运行过程中始终维持在4096*50000=195MB大小。

```
-rw-r--r--@ 1 erik  staff       2633 12  2 21:18 BMgr.cpp
-rw-r--r--@ 1 erik  staff       1388 12  2 20:38 BMgr.hpp
-rw-r--r--@ 1 erik  staff       1477 12  2 23:18 DSMgr.cpp
-rw-r--r--@ 1 erik  staff        984 12  2 23:04 DSMgr.hpp
-rw-r--r--@ 1 erik  staff       1776 12  2 21:18 LRU.cpp
-rw-r--r--@ 1 erik  staff        737 12  2 20:28 LRU.hpp
-rw-r--r--@ 1 erik  staff    3690270 11 11 10:20 data-5w-50w-zipf.txt
-rw-r--r--  1 erik  staff  204800000 12  2 23:18 data.dbf
-rw-r--r--@ 1 erik  staff       3371 12  2 21:25 main.cpp
-rw-r--r--@ 1 erik  staff        699 12  1 17:00 tracesetup.cpp
-rw-r--r--@ 1 erik  staff       1070 12  2 22:33 tracesetup.hpp
```

除此之外，还测试了当buffer size为**2048**时的情况：

```
处理记录序号: 499996
处理记录序号: 499997
处理记录序号: 499998
处理记录序号: 499999
算法运行时间: 70.4061s
buffer size: 2048
hit数: 209569
hit rate: 41.9138%
io数: 290431
Program ended with exit code: 0
```

当buffer size为**4096**时的情况：

总结

这次实验不仅让我对数据库的缓冲区系统有了更加深入的认识，还让我对哈希技术、LRU算法等有了全新的了解，明白了数据库底层的一些实现细节，这对我以后的工作和学习有着很大的帮助。