CS33503数据库系统实验

实验检查记录

|  |  |  |  |
| --- | --- | --- | --- |
| 实验结果的正确性(60%) |  | 表达能力(10%) |  |
| 实验过程的规范性(10%) |  | 实验报告(20%) |  |
| 加分(5%) |  | 总成绩(100%) |  |

实验报告

一、实验目的（介绍实验目的）

|  |
| --- |
| 1. 掌握数据库管理系统的存储管理器的工作原理。  2. 掌握数据库管理系统的缓冲区管理器的工作原理。  3. 使用C++面向对象程序设计方法实现缓冲区管理器。 |

二、实验环境（介绍实验使用的硬件设备、软件系统、开发工具等）

|  |
| --- |
| 1. 硬件设备：Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.71 GHz; 8GB RAM  2. 软件系统：Windows 10  3. 开发工具：8.0.28 MySQL Community Server – GPL |

三、实验过程（介绍实验过程、设计方案、实现方法、实验结果等）

|  |
| --- |
| 1. 阅读源代码  本次实验需要实现一个缓冲区管理器，主要由3个类实现，分别为BufMgr、BufDesc和BufHashTbl，缓冲区管理器的底层涉及到文件类（File）和页面类（Page）。在实验源代码中，已经给出了File、Page和各种异常的定义与实现，BufMgr、BufDesc和BufHashTbl的全部定义和大部分实现也都已经给出，仅需实现BufMgr中的几个方法。  为了实现缓冲区管理器，下面对BufMgr、BufDesc和BufHashTbl这几个类进行分析。   1. BufHashTbl：   该类采用链表法实现了一个哈希表，能将一个文件句柄（File\*）和页号（PageId）的组合与一个缓冲池中的页框号（FrameId）进行映射，同时提供了插入、查找、删除等方法。  其定义如下：      其中hashBucket定义如下：     1. BufDesc：   该类用于记录缓冲池中每个页框的状态，BufMgr是它的友元。该类能记录当前页框的页框号、对应的文件句柄和页号、引用次数、是否被修改、是否有效和“最近”是否被访问等信息，并提供设置、清除和打印信息等方法。  其定义如下：     1. BufMgr：   该类为缓冲区管理器的核心，能记录时钟算法当前指向的页框、缓冲区大小、缓冲区各个页框状态、缓冲区中页面所组成的哈希表等信息，并提供读取页面、解除页面固定、分配页面、删除页面、清除缓冲区等方法。  其定义如下：    2. 实现buffer.cpp中的方法  (a) ~BufMgr()：  该方法先将缓冲池中所有脏页写回磁盘，然后释放缓冲池、BufDesc表和哈希表占用的内存。  (b) void advanceClock()：  该方法顺时针旋转时钟算法中的表针，将其指向缓冲池中下一个页框。  (c) void allocBuf(FrameId& frame)：  该方法使用时钟算法分配一个空闲页框。如果页框状态valid为false，则可直接分配；如果refbit被设为true，则需要将其改为false并寻找下个页框；如果页框被固定住了，则将固定住的页框数加一，当缓冲池中所有页框都被固定了的时候，抛出BufferExceededException异常；如果页框中的页面是脏的，则需要将脏页先写回磁盘。如果被分配的页框中包含一个有效页面，则必须将该页面从哈希表中删除。最后，分配的页框的编号通过参数frame返回。   1. void readPage(File\* file, const PageId PageNo, Page\*& page)：   该方法将一个页面读入缓冲池中。首先调用哈希表的lookup()方法检查待读取的页面(file, PageNo)是否已经在缓冲池中。  如果该页面已经在缓冲池中，先将页框的refbit置为true，并将pinCnt加1，最后通过参数page返回指向该页框的指针。  如果该页面不在缓冲池中，则哈希表的lookup()方法会抛出HashNotFoundException异常。在这种情况下，捕捉该异常，调用allocBuf()方法分配一个空闲的页框。然后调用file->readPage()方法将页面从磁盘读入刚刚分配的空闲页框。接下来，将该页面插入到哈希表中，并调用Set()方法正确设置页框的状态。最后通过参数page返回指向该页框的指针。   1. void unPinPage(File\* file, const PageId PageNo, const bool dirty)：   该方法将一个在缓冲区中的页面解除固定。首先调用哈希表的lookup()方法检查待读取的页面(file, PageNo)是否已经在缓冲池中。  如果该页面已经在缓冲池中，将其所在页框的pinCnt值减1。如果参数dirty等于true，则将页框的dirty位置为true。如果pinCnt值已经是0，则抛出PAGENOTPINNED异常。  如果该页面不在缓冲池中，则什么都不用做。   1. void allocPage(File\* file, PageId& PageNo, Page\*& page)：   该方法分配一个页面并将其放入缓冲池中。首先调用file->allocatePage()方法在file文件中分配一个空闲页面。然后调用allocBuf()方法在缓冲区中分配一个空闲的页框。接下来在哈希表中插入一条项目，并调用Set()方法正确设置页框的状态。该方法既通过pageNo参数返回新分配的页面的页号，还通过page参数返回指向缓冲池中包含该页面的页框的指针。   1. void disposePage(File\* file, const PageId pageNo)：   该方法从文件file中删除页号为pageNo的页面。在删除之前，如果该页面在缓冲池中，需要将该页面所在的页框清空并从哈希表中删除该页面。   1. void flushFile(File\* file)：   该方法清除缓冲池中所有属于文件file的页面。对每个检索到的属于文件file的页面，如果是无效页，则抛出BadBufferException异常；如果被固定住，则抛出PagePinnedException异常；其余页面进行如下操作：(a) 如果页面是脏的，则调用file->writePage()将页面写回磁盘，并将dirty位置为false；(b) 将页面从哈希表中删除；(c) 调用BufDesc类的Clear()方法将页框的状态进行重置。  3. 在main.cpp中添加测试用例  实验源代码中包含6个测试用例：test1将内容写入页面后再读出来，测试前后内容是否一致；test2将多个文件交叉进行写与读的操作，测试前后内容是否一致；test3读一个不存在的页，测试是否会报错；test4对一个未被固定住的页面进行解除固定操作，测试是否会报错；test5在缓冲池已满的情况下申请分配一个页面并放入缓冲池，测试是否会报错；test6对缓冲池中尚未解除固定的页面进行清除，测试是否会报错。  为了使测试更充分，下面添加3个测试用例：  Test7读取一个不在缓冲池中的页面，待其进入缓冲池中后，再次读取，测试其两次读取是否都能正常执行，且pinCnt值为2，代码如下：    Test8读取一个已被删除的页面，测试是否会报错，代码如下：    Test9在缓冲池中有多个文件对应页面且缓冲池已满时，分配页面，测试是否会报错，清除其中一个文件的页面，再分配页面，测试页面分配是否成功，且页面清除数量是否正确，代码如下：    4. 实验结果  首先运行make编译项目，然后运行./badgerdb\_main，结果如下：    可以看到，原有测试用例和添加的测试用例全部通过。 |

四、实验结论（总结实验发现及结论）

|  |
| --- |
| 通过本次实验，我对缓冲区管理器的工作原理有了更深刻的认识与理解，掌握了基于时钟算法的缓冲区页面替换策略，对基于C++语言的编程进行了首次尝试，提升了代码能力，也在阅读源代码的同时，了解了BadgerDB各个类的定义与实现，掌握了数据库管理系统里面存储管理器的工作原理。 |