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

|  |
| --- |
| 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

|  |
| --- |
| 硬件设备：Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz  软件系统：Windows 11 22H2、Ubuntu 20.04.4 LTS  开发工具：Visual Studio Code 1.65.2 |

三、实验过程

|  |
| --- |
| **实验过程及实现方法：**  buffer.cpp文件中的方法介绍。   1. **BufMgr(const int bufs)**：BufMgr类的构造函数。为缓冲池分配一个包含bufs个页面的数组，并为缓冲池的BufDesc表分配内存。当缓冲池的内存被分配后，缓冲池中所有页框的状态被置为初始状态。接下来，将记录缓冲池中当前存储的页面的哈希表被初始化为空。本实验已经实现了该构造函数。 2. **~BufMgr()**：BufMgr类的析构函数。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。 3. **void advanceClock()**：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。 4. **void allocBuf(FrameId& frame)**：使用时钟算法分配一个空闲页框。如果页框中的页面是脏的，则需要将脏页先写回磁盘。如果缓冲池中所有页框都被固定了(pinned)，则抛出BufferExceededException异常。allocBuf()是一个私有方法，它会被下面介绍的readPage()和allocPage()方法调用。请注意，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。 5. **void readPage(File\* file, const PageId PageNo, Page\*& page)**：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，则通过参数page返回指向该页面所在的页框的指针；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。根据lookup()的返回结果，我们处理以下两种情况。   –情况1: 页面不在缓冲池中。在这种情况下，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，Set()会将页面的pinCnt置为1。最后，通过参数page返回指向该页框的指针。  –情况2: 页面在缓冲池中。在这种情况下，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。   1. **void unPinPage(File\* file, const PageId PageNo, const bool dirty)**：将缓冲区中包含(file, PageNo)表示的页面所在的页框的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。如果该页面不在哈希表中，则什么都不用做。 2. **void allocPage(File\* file, PageId& PageNo, Page\*& page)**：首先调用file->allocatePage()方法在file文件中分配一个空闲页面，file->allocatePage()返回这个新分配的页面。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。 3. **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。 4. **void flushFile(File\* file)**：扫描bufTable，检索缓冲区中所有属于文件file的页面。对每个检索到的页面，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。如果文件file的某些页面被固定住(pinned)，则抛出PagePinnedException异常。如果检索到文件file的某个无效页，则抛出BadBufferException异常。   下面我们将实现buffer.cpp文件中的方法。   1. **~BufMgr()**：此函数为析构函数。扫描每个页框，若当前页框的valid值为true，则调用flushFile函数将可能的脏页写回磁盘。再将bufDescTable、bufPool和hashTable删除。 2. **void advanceClock()**：clockHand加1并将结果对numBufs取模即可。 3. **void allocBuf(FrameId& frame)**：定义bufPinned表示固定的页框的数量，在每次遇到固定的页框时更新。首先调用advanceClock函数更新时钟指针，若当前指向的页框的valid为false则直接返回当前的clockHand。否则，查看refbit，若为true，则将其置为false，进入下一轮循环。否则，若当前页框pinCnt为0，则先将脏页写回，再返回当前clockHand。否则，更新bufPinned，当bufPinned达到numBufs时，抛出BufferExceededException异常。返回前，若clockHand所指的页框的valid为true，则调用hashTable->remove将该页面从哈希表中删除。 4. **void readPage(File\* file, const PageId PageNo, Page\*& page)**：首先调用hashTable->lookup查找缓存中是否存在该页面。若有，则将refbit置为true，pinCnt加1。否则，调用allocBuf分配一个页框frame，调用file->readPage获取该页面并存到bufPool[frame]中，调用hashTable->insert插入哈希表中，调用bufDescTable[frame].Set设置页框状态。最后，将bufPool[frame]的地址给page。 5. **void unPinPage(File\* file, const PageId PageNo, const bool dirty)**：调用hashTable->lookup尝试再哈希表中查找该页面，若没有该页面，则直接返回。否则，若当前页面的pinCnt为0，则抛出PageNotPinnedException。否则，将pinCnt减1，更新dirty状态。 6. **void allocPage(File\* file, PageId& PageNo, Page\*& page)**：调用file->allocatePage分配一个空闲页面，并调用page\_number获取该页面的编号。调用allocBuf分配页框frame，并调用hashTable->insert将新页面插入哈希表中。调用bufDescTable[frame].Set设置该页框状态，并将页面存在bufPool[frame]中。最后将bufPool[frame]的地址给page。 7. **void disposePage(File\* file, const PageId pageNo)**：调用hashTable->lookup查找该页面是否在缓存区中，若在则调用bufDescTable[frame].Clear清除所在页框，并调用hashTable->remove从哈希表中删除该页面。最后调用file->deletePage将该页面删除。 8. **void flushFile(File\* file)**：扫描每个页框，若当前页框的页面属于file，则依次进行如下判断。若页框状态valid为false，则抛出BadBufferException。若pinCnt不为0，则抛出PagePinnedException。若dirty为true，则调用file->writePage将bufPool中的数据写回磁盘，并将dirty置为false。再调用hashTable->remove从哈希表中删除该页面，最后调用bufDescTable[frame].Clear将该页框状态删除。   **实验结果：**    如图所示，实现的实验代码顺利通过了所有的测试，完成了实验。 |

四、实验结论

|  |
| --- |
| 通过本次对缓存区管理器的实现，我更深刻地了解了数据库管理系统的存储管理器和缓冲区管理器的工作原理，体会到了数据库管理系统中独特地魅力。也通过使用C++面向对象程序设计方法实现了缓冲区管理器，对面向对象程序设计方法有了更深的理解。 |