DataBase-Lab2-Report

-郭培宇
-521120910259

1. Designing Part

1.1 Exercise 1

在 Lab 1 中，我们没有正确的根据 BufferPool 构造函数中定义的 numPages 对 BufferPool 中缓存的最大页面数量进行限制。在本次 lab2 的 Lab2 中，我们将主要实现一个一种页面驱逐策略。具体方法是在缓冲池的页面数量超过'numpages'设定的上限时，应该在加载下一个页面加入缓冲池之前驱逐一个池内的页面以维持缓冲池的数量限制。具体的驱逐策略并未特定规定，我的具体实现方法是将进入缓存池的页面放入链接哈希表，最后需要驱逐时按照 LinkedHashSet 的插入顺序或访问顺序进行脏页驱逐，步骤是：

① 创建一个静态类，用于表示缓冲池中的每个页面项目，具体包裹页面标识符 pid、事务标识符 tid、权限 perm 和页面对象 page 这四个成员变量，并在最后提供一个构造函数以初始化以上四个成员变量。

② 使用 LinkedHashMap 来代替原 BufferPool.java 文件中导入的 ConcurrentHashMap 来存储页面和页面项目，因为前者会按照插入顺序或访问顺序来维护键值对的顺序，而后者不保证顺序，因此我选择了 LinkedHashMap。

③在 GetPage,InsertTuple,DeleteTuple 和 TransactionComplete 等函数中补全 throw 关键句中的异常处理，比如 getpage 中获取指定 pid 的页面，若缓冲池满了则调用 evictpage 函数进行脏页驱逐。

④ 补全 evictpage 函数：使用了 LinkedHashMap 的 keySet() 方法来获取页面标识符的集合，然后调用 stream() 方法进行流操作，通过 findFirst() 方法获取集合中的第一个元素。

⑤ 详细定义 flushAllPages 和 flushPages 函数，具体来说 flushallpages 函数获取缓存池中所有页面的标识符集合（使用 keySet() 方法），然后对这个集合进行遍历。对于每个页面标识符 pid，都会调用 flushPage(pid) 方法来将该页面刷新到磁盘上。如遇到异常则打印异常信息，同时不中断后续页面刷新操作。

以上使得我在 `flushPage` 和 `flushAllPages` 方法不会从缓冲池中驱逐页面，并保证唯一应该从缓冲池中删除页面的方法是 `evictPage` ，它在驱逐的脏页面上调用 `flushPage`。

1.2 Exercise 2

在接下来的 exercise 2 ——> 5 中，我们需要修改的函数和代码主要在`BTreeFile.java`，这是实现 B+Tree 的核心文件。与 HeapFile 不同，BTreeFile 由四种不同的页面组成。本 exercise 的任务是实现 `BTreeFile.java` 中的 `findLeafPage()` 函数。该函数用于在给定特定键值的情况下查找适当的叶页，并用于搜索和插入。

我的具体思路是首先通过 getpage 获取指定页面，然后判断是否是叶子页面，若是则返回，不是则继续查找。在继续查找的过程中首先查找是否有下一个条目，无则返回 null，有则通过迭代器从当前页面的子节点中找到下一个该查找的页面（具体逻辑是遍历当前页面的条目，找到第一个 key 大于等于给定值的条目然后返回这个条目对应的左子节点，若不符合条件则返回当前页面的最右子节点），若找到下一个页面则继续递归调用 findLeafPage 方法继续向下寻找。

1.3 Exercise 3

此 exercise 中我们要实现 B+树中元素的插入，需要修改的是 splitLeafPage()和 splitInternalPage()这两个方法。我的思路是在 splitLeafPage 函数中：

首先，计算出需要移动到新页面的元素个数。获取父页面，并创建一个新的空白叶子页面。将叶子页面中的一半元素移动到新页面中，并调整相应的指针，包括删除原页面中的元素并将其插入新页面，更新父页面的中间键，更新新页面的父指针以及更新邻居叶子页面的指针。最后根据给定的键值判断需要插入的页面，并返回该页面。

在 splitInternalPage 函数中：

类似于 splitLeafPage 的计算、调整相应的指针、更新新页面的父指针、最后根据给定的键值判断需要插入的页面，并返回该页面。唯一的区别是 splitLeafPage 方法需要更新邻居叶子页面的指针，而本方法不需要。

1.4 Exercise 4

这里我们要实现 B+树中页面的重新分配，我们需要修改 stealFromLeafPage，stealFromLeftInternalPage，stealFromRightInternalPage 这三个函数，其中思路几乎相同：

首先，计算当前页面和兄弟（左兄弟、右兄弟）页面的大小，并确保当前页面的大小小于最大元组数的一半，而目标页面的大小大于最大元组数的一半。然后：

stealFromLeafPage：从兄弟页面中选择一定数量的元组，移动到当前页面，并更新父条目的键值为当前页面或兄弟页面的最后一个元组的键值。

stealFromLeftInternalPage：向当前页面插入父条目，删除左兄弟页面的部分条目，并将它们插入当前页面。更新父条目的键值为左兄弟页面的中间键，并更新左兄弟页面的中间键。

stealFromRightInternalPage：向当前页面插入父条目，删除右兄弟页面的部分条目，并将它们插入当前页面。更新父条目的键值为右兄弟页面的第一个键，并更新右兄弟页面的第一个键。

1.5 Exercise 5

我们需要修改 mergeLeafPages()和 mergeInternalPages()两个方法，这两个方法也类似，比如首先确保左页面和右页面的元组总数不超过左页面的最大元组数。然后，将右页面的所有元组移动到左页面并将其从右页面中删除。接着，删除父页面中的条目，并更新左页面，这里两个方法唯一的区别是前者是将右兄弟指针改为右页面的右兄弟，而后者是更新左页面中所有子页面的夫指针。最后，将右页面标记为空页，以便重用。

2 实验心得

此次 lab2 我认为难度较大的过程仍然是学习 java 语言的编译规范及相关 Package 的熟悉，因为就像在上次 lab 中所说的本人因为是金融计算机专业只系统学习过 c++及自学过 python，对于 java 本人需要在作业推进的过程中不断学习 java 相关的知识。

其次，是了解 B+树的具体原来和 B+树的操作规范，B+树本身是个高效但复杂的数据结构，其中对于基于此数据结构的数据库的任何操作都必须合法且符合 B+树规范，比如 readme 文档中的”A non-root node cannot be less than half full.”这类规范都是 exercise2 到 5 的 java 代码编写过程中需要不断注意且测试的。

但是经过此次 lab，我认为我对于基于 B+树结构实现的数据库的概念又加深了一些，比如该类数据库的具体实现方法和底层逻辑，结合 java 的数据库操作的具体函数及方法的完善让我对实际 database 的认知逐渐清晰。