南開大學
Nankai University

南　　开　　大　　学

计　算　机　学　院

数据库系统实验报告

SimpleDB Lab3

姓名：韩佳迅

学号：2012682

年级：2020 级

专业：计算机科学与技术

2022 年 5 月 8 日

摘要

本次实验完成了 SimpleDB 的 Lab3, 并通过了测试。通过完成 Lab3 实现了数据库的 B+ Tree Index，重点在于理解 B+ 树的数据结构。

本报告中给出了每个 exercise 的具体设计和实现思路、重难点，借此实验和报告理清了 B+ 树的操作。

目录

一、 Lab3 总览

GitLab 仓库地址：https://gitlab.com/hanmaxmax/SimpleDB

Lab3 主要是实现 B+ 树索引。分为以下几个部分：

(一) Search

B+ 树索引查找的过程，实现了 BTreeFile.findLeafPage()

(二) Insert

在 B+ 树中插入新元素，需要实现拆分叶子节点和拆分中间节点两个方法。

(三) Delete

在 B+ 树中删除元素，需要实现分别叶节点和中间节点的从左/右兄弟 steal tuple 或 entry 的方法，以及合并叶子节点或中间节点的方法。

(四) Extra Credit

自己实现 BTreeReverseScan 的功能，需要添加 BTreeReverseScan.java，并在 BTreeFile.java 中添加相应的接口实现。同时还需要添加 BTreeReverseScanTest.java，对补充的反向扫描功能进行测试。

二、 设计思路及重难点

(一) Exercise 1

- **BTreeFile.findLeafPage()**

代码思路：

1. 判断该页是否为叶子页，如果是，则递归结束，直接返回；
2. 该页不是叶子，则先将该页转换为中间节点页 BTreeInternalPage；
3. 取该页的迭代器，对内部节点页中的 entry 进行迭代搜索
4. 搜索分为两种情况：

（1）如果要找的 field 是空，则直接找最左侧的叶子节点。此时一直递归寻找页迭代器的第一个迭代 entry 的最左孩子；

（2）如果要找的 field 非空，则需要迭代遍历该页的 entry，找到第一个大于（或等于）field 的 entry，然后递归其左孩子

5. 如果迭代到了最后一个页面，并且最后一个页面都不大于等于 field，则递归其右孩子

重难点：

Search 部分总体不难，重点要注意 field 为空的情况，在指导书中已经提到。

(二) Exercise 2

- **BTreeFile.splitLeafPage()**

代码思路:

1. 调用 BTreeFile 的 getEmptyPage，新建一个叶子 page;
2. 获取该满 page 的反向迭代器，从后往前迭代 tuple，将一半的 tuple 存入预先准备好的数组中;
3. 将 tuple 们从原 page 删除，插入到新的 page;
4. 取原 page 的中间值，为其创建新 entry，并将新 entry 插入到父节点中;
5. 更新叶子兄弟节点的指针

(1) 要先判断原节点有无右兄弟，若有，则更新其右兄弟的左指针指向新插入的节点;

(2) 设置插入的节点左指针指向原节点，右指针为原节点的原右节点;

(3) 设置原节点的右指针指向新插入的节点。

6. 更新脏页;
7. 更新父节点指针;
8. 返回应该插入具有给定键字段的元组的 page

- **BTreeFile.splitInternalPage()**

代码思路:

1. 调用 BTreeFile 的 getEmptyPage，新建一个中间节点 page;
2. 获取该满 page 的反向迭代器，从后往前迭代，将一半的 entry 存入预先准备好的数组中;
3. 将 entry 们从原 page 删除，插入到新的 page;
4. 取原 page 的中间值，为其创建新 entry，并将新 entry 插入到父节点中;
5. 将中间 entry 挤到父节点

(1) 获取将被挤到父节点的中间 entry

(2) 从 page 中删除该中间 entry

(3) 更新该中间 entry 的左右孩子分别为原 page 和新 page

(4) 将更新后的中间 entry 插入父节点，注意寻找父节点的时候调用的是 getParentWithEmptySlots()

(5) 更新原 page 和新 page 的指针

6. 更新脏页
7. 根据 field 去决定返回哪个页面

重难点:

拆分中间节点与叶子节点的不同之处:

1. 拆分叶子节点需要调整左右叶子兄弟的指针，而中间节点不用;
2. 拆分中间节点是把原 page 的中间 entry 挤到父节点中，并把其从原 page 中删除；而拆分叶子节点则是把它复制到父节点，原叶子节点仍存在该 entry。

(三) Exercise 3

- **BTreeFile.stealFromLeafPage()**

代码思路:

1. 获取该节点和它兄弟节点的元组数目，两者相加取一半为分配后的每个节点的元组数目，据此计算要从兄弟节点 steal 的元组数目；

2. 获取迭代器：根据该节点是从左兄弟 steal 还是右兄弟 steal，进行如下区分：

（1）如果该节点的兄弟是右节点（isRightSibling），则从右兄弟 steal，迭代器取正向的

（2）否则从左兄弟 steal，迭代器取反向的

3. 将要移动（steal）的元组们从兄弟节点 delete，再在自己节点 insert；

4. 更新父节点指向这两个叶子的 entry 的 key 值

（1）如果该节点是左节点，则取它的 reverseIterator() 迭代的第一个值

（2）否则，取它的左兄弟的 reverseIterator() 迭代的第一个值

- **BTreeFile.stealFromLeftInternalPage()**

代码思路：

1. 获取该节点和它兄弟节点的 entry 数目，两者相加取一半为分配后的每个节点的 entry 数目，据此计算要从兄弟节点 steal 的 entry 数目；

2. 获取迭代器：由于是从左兄弟 steal，所以取一个反向的迭代器；

3. 将要移动（steal）的元组们从兄弟节点 delete，再在自己节点 insert；

对于每一个要移动的 entry：

（1）调用 deleteKeyAndRightChild() 把该 entry 从左兄弟节点移走

（2）把 steal 过来的 entry 的 key 设为父 entry 的 key，相当于父节点 entry 下移，左兄弟 entry 上移

（3）entry 从左边被 steal 到右边后，它原来的右孩子变成左孩子，新的左孩子是原右节点（this page）的第一个最左侧的孩子

4. 更新已移动页面的父指针

5. 更新父节点中指向这两个叶子的 entry 的 key，相当于将原父节点的该 entry 下移，原左兄弟 entry 上移

6. 添加脏页

- **BTreeFile.stealFromRightInternalPage()**

代码思路：

与 stealFromLeftInternalPage 思路一致，相当于一个镜面操作，把左右互换即可。（注意这里取的是右兄弟的正向迭代器）

- **BTreeFile.mergeLeafPages()**

代码思路：（删除右节点，统一合并到左节点）

1. 获取要移动的右节点元组数，并取一个数组暂存要移动的元组们；

2. 获取原右节点的正向迭代器，迭代右节点的元组们；

3. 标记脏页

4. 在 B+ 树里面删除父节点的对应 entry

6. 合并节点（从右节点删除 tuple，从左节点插入 tuple）

7. 更新兄弟节点的指针

（1）若原右节点的右兄弟非空，则获取原右节点的右兄弟，并让原右兄弟的左指针指向合并后的节点

（2）然后让合并后节点的右指针指向原右节点的右兄弟

8. 把原右节点置空

- **BTreeFile.mergeInternalPages()**

代码思路：（删除右节点，统一合并到左节点）

1. 获取要移动的右节点 entry 数，并取一个数组暂存要移动的 entry 们；
2. 获取原右节点的正向迭代器，迭代右节点的 entry 们；
3. 标记脏页
4. 在 B+ 树里面删除父节点的对应 entry
5. 将原来父节点里对应的 entry 拉下来到新节点中间

（1）设置原来的父 entry 的左孩子为原左节点的最右孩子

（2）设置原来的父 entry 的右孩子为原右节点的最左孩子

（3）将该 entry 插入到合并后的节点里面

6. 合并节点（从右节点删除 entry，从左节点插入 entry）
7. 更新父节点指针
8. 把原右节点置空

重难点：

steal from 叶子节点和中间节点的区别：

（1）叶子节点的重新分配是将元素直接从兄弟节点挪过来，然后删除父节点中对应的 entry；

而中间节点的重新分配可以理解为是先把父节点的 entry 值拉下来到自己节点，再把兄弟节点的 entry 上移到父节点。

（2）对中间节点重新分配的时候还要注意每个 entry 在移动的时候，需要先更改其左右孩子，再进行挪动。

合并中间节点与合并叶子节点的区别：

（1）leaf 存的是真正的数据，而中间节点存的只是索引 entry，所以对中间节点 merge 时，要把父节点的 entry 拉下来，相当于最后的节点 = 左 + 父 entry+ 右，然后再删除父节点的 entry；而对叶子节点 merge 时，直接删除父节点的 entry，并不需要将它拉下来。

（2）对叶子节点的合并，需要更改其叶子兄弟节点的指针，而对中间节点则不用。

（四） Exercise 4 （思考题）

在这一部分，实现了反向扫描的功能：

1. 在 BTreeFile.java 添加 ReversefindLeafPage，具体实现思路与 findLeafPage 相同，区别在于循环遍历从大于等于改成小于等于，找左孩子改为找右孩子，field 为 null 时，改成返回最右侧孩子。

2. 在 BTreeFile.java 添加反向迭代器 BTreeReverseSearchIterator 和 BTreeFileReverseIterator 以及它们俩的对应接口函数。这两个迭代器是基于 BTreeSearchIterator 和 BTreeFileIterator 实现的，与 ReversefindLeafPage 一样，需要将原来的“左”“右”互换，将大于与小于互换。

3. 在项目中添加 BTreeReverseScan.java，注意这里要调用我们在 BTreeFile.java 中实现的反向迭代器。并且添加 BTreeReverseScanTest.java 测试代码，在测试代码中，调用我们实现的 BTreeReverseScan 类，并且注意要把大小比较反向。

最后测试结果如图：

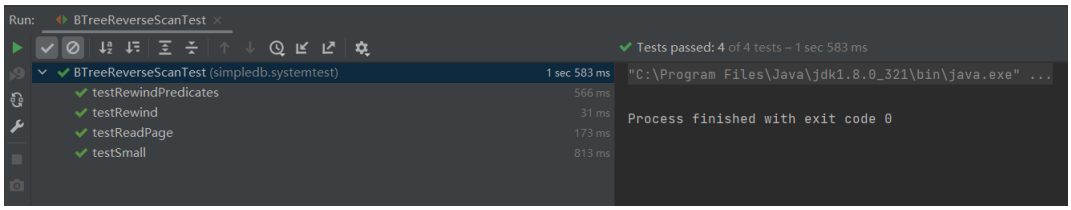

图 1: 提交记录

三、 Git Commit History

GitLab 仓库地址：https://gitlab.com/hanmaxmax/SimpleDB

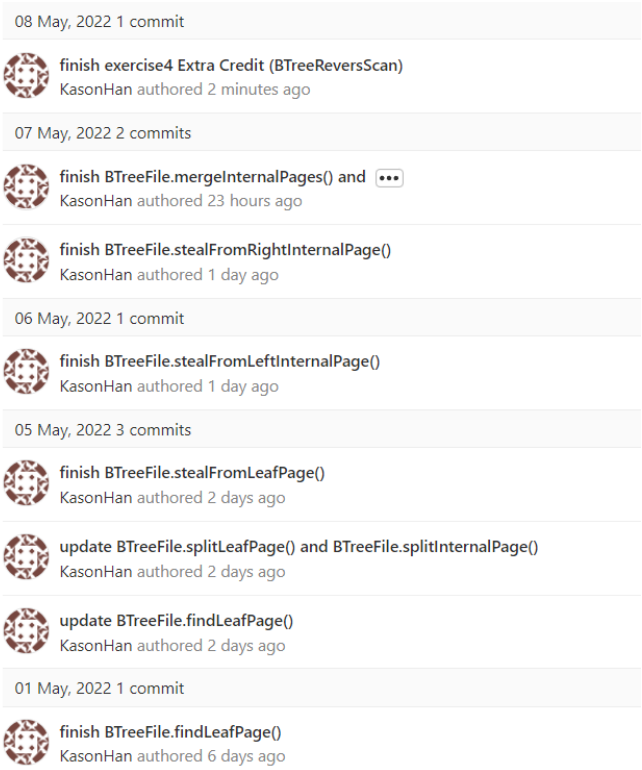

图 2: 提交记录

四、 改动部分

在项目中添加了新的文件 BTreeReverseScan.java、BTreeReverseScanTest.java。