

Lab 3 Report

姓名：戚晓睿


学号：1811412



Github: <https://github.com/NickSkyyy/SimpleDB>

GitCommitPic: see it below

Commits on May 3, 2020


extra credit finished.



 NickSkyyy committed 12 hours ago

 0589b0c 

Commits on May 2, 2020


try to deal with extra credit.



 NickSkyyy committed yesterday

 1ace5a9 

Commits on Apr 28, 2020


4 finished.



 NickSkyyy committed 5 days ago

 9777e81 

Commits on Apr 27, 2020


3 error in system test 4.



 NickSkyyy committed 7 days ago

 30e2c25 


Commits on Apr 25, 2020



4 steal internal page finished.

 NickSkyyy committed 8 days ago

 aef8ce8 


4 steal leaf page finished.



 NickSkyyy committed 8 days ago

 9df370d 


Commits on Apr 21, 2020



3 finished.

 NickSkyyy committed 12 days ago


 2b3a2e0 



2 error in system test.

 NickSkyyy committed 12 days ago

 289fa6d 


3 null pointer error.



 NickSkyyy committed 12 days ago

 da9b14f 

Commits on Apr 20, 2020


3 check splitInternalPage.



 NickSkyyy committed 13 days ago

 525e724 


Commits on Apr 19, 2020



2 finished.

 NickSkyyy committed 14 days ago

 ed2ef97 


lab 3 start. ...



 NickSkyyy committed 14 days ago

 d0559e4 

Commits on Apr 10, 2020

Lab 2

 NickSkyyy committed 23 days ago

 5b42ad4 

滚动鼠标轴或单击，开始截长图

Browse the repository at this point in the history

1 设计思路

Lab3 实现的是对整个B树的处理，包括查找、插入和删除三类操作。相较前两个 Lab 而言，这一次比较系统化，难度上升体现在对于整体B树结构的理解和把握上。通读源代码提供的各种方法有助于更好实现本次 Lab 的任务。

在实现过程中，保持**less is more**的思想，以掌握整体B树结构为前提，按需求将方法中的代码量控制到最少，尽量减少不必要的内容。一者方便自己更清晰地进行 debug，二者可以有效提升程序效率。

2 详细介绍

2.1 Search

最基本的B树查询操作实现，放在本次 Lab 第一的位置上。但由于刚上手，写起来的时候也是遇到了不小的麻烦。由于之前在写 `BufferPool.java` 的时候专注于写 `HeapFile` 等内容的实现，导致忽略了 `DbFile` 下 `BTreeFile` 的子类实现，因此优先将 `BufferPool` 的结构进行了优化，提升其泛化能力，只返回对应的抽象类(`DbFile`, `Page`)，而对具体内容的判断放在了对应子类的实现里面。这里只取 `BufferPool.getPage` 作为例子：

```
1 public Page getPage(TransactionId tid, PageId pid, Permission perm) throws
   TransactionAbortedException, DbException {
2     ...
3     DbFile dbf = Database.getCatalog().getDatabaseFile(pid.getTableId());
4     Page p = dbf.readPage(pid);
5     pool.put(pid, p);
6     order.add(pid);
7     return p;
8 }
```

由于B树中节点类型众多，总共有4种 `Page`，后面我将用以下缩写代替其页类型，分别是：

```
1 BTreeLeafPage btlp;
2 BTreeInternalPage btip;
3 BTreeHeaderPage bthp;
4 BTreeRootPtrPage btrpp;
```

因此需要用到多重 `if` 语句来判断，在 `BTreeFile.findLeafPage` 方法中针对不同的页类型逐一实现。叶子结点直接返回，根节点则返回类下记录的第一个 `btip`，而对应的 `btip` 则根据给定的字段进行二分查找。

```
1 if (f == null)
2     return findLeafPage(tid, dirtypages, bte.getLeftChild(), perm, f);
```

值得注意的是，当传入参数 `f` 为空字段时，这里返回的内容是**最左端**(`left-most`)的叶子结点，并非 `null` 值。

2.2 Insert

对B树的插入操作，核心处理插入过程中的节点分裂问题。对两类页类型 `btip`, `btlp` 需要分开进行处理。尽管如此，两者的共性为：插入都会导致新页面的生成，将移动原页面一半的 `tuples/entries` 进入到新的页面当中，并将对应的 `middle` 元素置入 `parent` 当中。

不同点在于，叶子结点需要**记录兄弟信息**，因此需要格外注意，我在后面 `delete` 操作的时候正是忘记了更新这个信息，导致 debug 卡了3天时间。考虑到叶子结点间有这样的关系：`leftSibling - page`
`- rightSibling`

```

1  if (page.getRightSiblingId() != null) {
2      BTreeLeafPage right = (BTreeLeafPage)getPage(tid, dirtypages,
page.getRightSiblingId(), Permissions.READ_WRITE);
3      right.markDirty(true, tid);
4      dirtypages.put(right.getId(), right);
5      right.setLeftSiblingId(bt1p.getId());
6      bt1p.setRightSiblingId(right.getId());
7  }

```

这段代码有效处理了原本存在的**右兄弟非空**的现象，细节不容忽视。后面则是比较简单的复制一半内容的操作，注意一定要**先删除后插入**，否则会出现对应内容项不匹配的错误。

```

1  Tuple t = it.next();
2  page.deleteTuple(t);
3  bt1p.insertTuple(t);
4  ...
5  updateParentPointers(tid, dirtypages, parent);

```

上面代码的第5行，是叶子结点进行的父节点更新操作，此处为了将新生成的 page 连接到对应的 parent 当中。

对于 bt1p 来讲，插入操作需要额外考虑到的**仅有对于新生成节点的更新操作**，移动到 page 当中的 entry 所表示的儿子节点集合相当于将自己的父亲节点更改为 page，代码如下：

```

1  parnet.insertEntry(new BTreeEntry(upper.getKey(), page.getId(),
bt1p.getId()));
2  updateParentPointers(tid, dirtypages, bt1p);

```

2.3 Delete

处理B树的删除操作，核心解决因删除导致的节点合并、数据借用问题，这一部分相较于前两个处理的方法更多，要考虑的情况更丰富，是本次 Lab 中最麻烦的一个练习。正如在2.2中提到的，由于忘记考虑叶子结点的 sibling 处理，出现了 debug 3天不成功的结局:3 sad 和插入处理一样，bt1p, bt1p 两种页类型需要分别考虑。

2.3.1 Steal

根据源代码要求，叶子结点通过 isRightSibling 的布尔值判断从左兄弟还是右兄弟借用数据。

```

1  int cnt = page.getNumTuples(), mark = (page.getNumTuples() +
sibling.getNumTuples()) / 2;

```

起初自己的思路出现了小错误，误将 getNumTuples 的地方理解为 getMaxTuples，导致测试失败。之后了解到方法采用的思路是**整体平分**达到节点内平衡，解决了此问题。这里，对于奇数偶数的判定比较宽松，我采取的**恒上提左侧 reverse 的第一个数据**作为 uppper key。

对于 bt1p，设计了两个方法，分别从左侧和右侧进行数据借用。拆分成两个方法的原因，个人认为，是源代码提供的方法中设计了对应的**两套针对左右不同节点的删除方法**，因此这里将方法拆开，也方便源代码提供方法的调用。以从左侧借用数据为例子：

```

1  Iterator<BTreeEntry> it = leftSibling.reverseIterator();
2  BTreeEntry down = new BTreeEntry(parentEntry.getKey(), null, null);
3  down.setLeftChild(it.next().getRightChild());
4  it = page.iterator();
5  down.setRightChild(it.next().getLeftChild());
6  page.insertEntry(down);

```

我的处理是**优先下放父亲节点中的数据**，再以+1数据量的数据进行平分处理，最后更新父亲节点和新生成节点的更新处理。

2.3.2 Merge

下面讲合并操作，两种页类型的共性是：都进行从右到左的合并，合并完成后将右侧页面设置为空值，以便进行新的循环使用。

对于叶子结点，合并的时候**一定要记得更新曾有的兄弟节点信息**，

```

1  if (rightPage.getRightSiblingId() != null) {
2      BTreeLeafPage btlp = (BTreeLeafPage)getPage(tid, dirtyPages,
3          rightPage.getRightSiblingId(), Permissions.READ_WRITE);
4      leftPage.setRightSiblingId(btlp.getId());
5      btlp.setLeftSiblingId(leftPage.getId());
6  }
7  else
8      leftPage.setRightSiblingId(null);

```

对于 btlp，首先需要**下放父亲节点中的数据**，这与叶子节点中对**父亲中 entry 的删除是不同的**：

```

1  // btlp
2  deleteParentEntry(tid, dirtyPages, leftPage, parent, parentEntry);
3  // btlp
4  BTreeEntry down = new BTreeEntry(parentEntry.getKey(), null, null);
5  Iterator<BTreeEntry> it = leftPage.reverseIterator();
6  down.setLeftChild(it.next().getRightChild());
7  it = rightPage.iterator();
8  down.setRightChild(it.next().getLeftChild());
9  leftPage.insertEntry(down);

```

3 附加

3.1 DirtyPages

根据和同学的讨论发现，本次 Lab 中对于 dirtyPages 的处理似乎不影响测试的通过，但是读完源代码后，个人认为 dirtyPages 的处理**对于整个系统的效率**是有不可或缺的影响的，每次 getPage 的时候先对 dirtyPages 进行查询操作，等等细节体现出这一部分的作用。

在本次 Lab 中，个人对 dirtyPages 的处理如下：每次操作后产生的新页、更新的脏页、被借用数据后减少信息的脏页等，我都**加入进了提供的 HashMap**中。由于源代码提供的方法中都有对 dirtyPages 的处理，因此在自己的代码中只有对 dirtyPage 的增添。

这里暂做此类处理，后续 Lab 中如果发现其他的内容在做更改。

3.2 BTreeReverseScan

附加题整体难度适中，在帮助文档里面给予了充足的提示：在 BTreeFile 中加入 reverseIterator 方便后续操作，设计与源代码提供相反的 get right-most page 的取页方法、reverseScan 等新的反向遍历类。

一些核心点是，findLeafPage 中添加新的布尔变量 isReverse，以此来判定是否需要反向遍历；和源代码提供的方法一样，当传参字段 field 为空值时，返回 right-most page。

```
1 private BTreeLeafPage findLeafPage(TransactionId tid, HashMap<PageId, Page>
  dirtypages, BTreePageId pid, Permissions perm, Field f, boolean isReverse)
  throws DbException, TransactionAbortedException {
2     ...
3     Iterator<BTreeEntry> it = btip.reverseIterator();
4     while (it.hasNext()) {
5         BTreeEntry bte = it.next();
6         if (f == null)
7             return findLeafPage(tid, dirtypages, bte.getRightChild(), perm,
  f, isReverse);
8     ...
9     }
10    ...
11 }
12 public BTreeLeafPage findLeafPage(TransactionId tid, BTreePageId pid,
  Permissions perm, Field f, boolean isReverse) throws DbException,
  TransactionAbortedException {}
```

除上述方法以外，新设计的 indexReverseIterator，reverseIterator 都是将源代码中的复制并稍加改动，获取左儿子/兄弟节点的地方，更改为右儿子/兄弟，LESS和GREATER适用范围进行互换等等。下面举一部分例子作为说明：

```
1 // BTreeFileReverseIterator.readNext
2 while (it == null && curp != null) {
3     BTreePageId nextp = curp.getLeftSiblingId();
4     ...
5     it = curp.reverseIterator();
6     ...
7 }
8 // BTreeSearchReverseIterator.open
9 if(ipred.getOp() == Op.EQUALS || ipred.getOp() == Op.LESS_THAN ||
  ipred.getOp() == Op.LESS_THAN_OR_EQ) {
10     curp = f.findLeafPage(tid, root, Permissions.READ_ONLY,
  ipred.getField());
11 }
12 else {
13     curp = f.findLeafPage(tid, root, Permissions.READ_ONLY, null, true);
14 }
```

在 BTreeReverseScanTest 测试类中设计到比对数据的内容，有两种处理方式。一种是将源代码提供的 compare 类进行更改，从升序变为降序比较，另外一中方式是保持原数列采样不变，通过翻转指针的方式进行比较。我采用的方法是后者。

对于附加题增添的新内容，在源代码中增加新类 BTreeReverseScan.java；在 systemtest 里面加入 BTreeReverseScanTest.java。