

# Lab 1 report

姓名：戚晓睿

学号：1811412

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

GitCommitPic: see it below

Branch: master ▾

Commits on Mar 15, 2020

Lab 1 finished.  
NickSkyyy committed yesterday

Change some structures. ...  
NickSkyyy committed yesterday

1cc1197

ae6dd15

Commits on Mar 14, 2020

lab1 without 2.6 testSmall. ...  
NickSkyyy committed 2 days ago

2.5 finished!!!  
NickSkyyy committed 2 days ago

6eeb433

12a83be

Commits on Mar 12, 2020

2.5 without iterator. ...  
NickSkyyy committed 4 days ago

5938bd1

Commits on Mar 11, 2020

2.4 without HeapPage. ...  
NickSkyyy committed 5 days ago

867888d

Commits on Mar 8, 2020

2.3 finished. ...  
NickSkyyy committed 8 days ago

7257486

Commits on Mar 7, 2020

2.2 finished without TupleDesc.hashCode  
NickSkyyy committed 9 days ago

5a57dfe

Commits on Mar 6, 2020

First video class trail.  
NickSkyyy committed 10 days ago

Update .gitignore  
NKU-DBIS-DB committed 10 days ago

SimpleDB NKU 2020  
NKU-DBIS-DB committed 10 days ago

067881a

Verified6a0540e

1aa5633

Newer

Older

滚动鼠标轴或单击，开始截长图

## 1 设计思路

Lab1设计的是数据库比较基础的部分，初步建立起了Schema (TupleDesc.java)、Tuple (Tuple.java)、Table (HeapFile.java)等基本元素，以及为后续操作提供方便的全局BufferPool (BufferPool.java)、Catalog (Catalog.java)等。

由于是初上手SimpleDB的实现，在设计过程中将侧重点放在了实现功能上面，用简单的数据结构和较为有效的算法实现了Lab1的所有内容。

第二部分将详细地分块介绍每一个实现的内容、难点与自己的思考。

## 2 详细介绍

### 2.1 TupleDesc & Tuple

建立基础的Schema和Tuple，实现起来较为简单，没有特别难的地方。主要介绍两个方法：TupleDesc.merge， TupleDesc.equals.

#### 2.1.1 TupleDesc.merge

```
1 public static TupleDesc merge(TupleDesc td1, TupleDesc td2)
```

考虑到传入的td有可能为null，空指针操作十分危险，在这里设计了长度的分类判断。

```
1 if (td1 == null)
2     len = td2.numFields();
3 else if (td2 == null)
4     len = td1.numFields();
5 else
6     len = td1.numFields() + td2.numFields();
```

len代表了merge之后的TupleDesc长度。

#### 2.1.2 TupleDesc.equals

上学期学了java，这一部分只提一下对Object的处理。

```
1 if (o instanceof TupleDesc) {
2     TupleDesc td = (TupleDesc)o;
3     ...
4 }
```

instanceof对Object进行判断，如果是TupleDesc的实体类对象，则进行第2行的强制转换，之后才能进行两个同样对象间的比较。

## 2.2 Catalog

在一开始设计的时候我使用的下面的数据结构：

```
1 private Map<String, Pair<DbFile, String>> tables; // table name to table &
   key
2 private Map<Integer, String> ids; // id to table name
```

但是在后面的实践过程中发现Pair的加入并没有起到很好的帮助，反而在取出内容的过程中导致操作繁琐，于是将Pair对应项拆开变成下面的三个HashMap：

```

1 private Map<String, DbFile> tables; // table name to table
2 private Map<String, String> keys; // table name to key
3 private Map<Integer, String> ids; // id to table name

```

这样对应到每一个需要的Value取值就不会像使用Pair那样充满限制。java自带的HashMap封装很好，Key和Value之间可以互相取到，非常方便。

## 2.3 BufferPool

配合HeapFile，这里是Lab1中最难理解和实现的地方之一。在首次撰写的时候，仅仅完成了小部分功能。在打好内部判断框架之后，就继续往下写了。

结合后面的练习，我们可以知道：BufferPool是**缓冲池**，任何在**磁盘中取出Page**的操作都需要经过BufferPool.getPage的调用，而不能使用其他类的其他方法。每当取出一个Page时，需要在BufferPool里面定位该Page的位置。由于仍然是一个配对匹配问题，这里使用的数据结构还是**HashMap**。

### 2.3.1 BufferPool.getPage

```

1 public Page getPage(TransactionId tid, PageId pid, Permissions perm) throws
   TransactionAbortedException, DbException {
2     ...
3     if (pool.size() + 1 > numPages)
4         throw new DbException("no more space");
5     HeapFile hf =
6         (HeapFile)Database.getCatalog().getDatabaseFile(pid.getTableId());
7     HeapPage hp = (HeapPage)hf.readPage(pid);
8     pool.put(pid, hp);
9     return hp;
10 }

```

第4行是对多于缓冲池最大页数操作的处理，我采用的是抛出异常的方式，提示用户当前缓冲池内无剩余可用空间。

第6-8行是对新Page的添加操作，对应方法的详细介绍将在下面给予说明。

## 2.4 HeapPageId, RecordId & HeapPage

前两项有关Id的文件比较好写。HeapPage部分，由于大量的公式和提示信息都写在了.md帮助文档里，写的时候比较顺利（但自己一开始没有好好看给的文档，于是在HeapPage.isSlotUsed方法上卡了很久，通过Debug才找到了header的位移判断操作，可见**好好读题有多么重要！**）

主要介绍两个方法：hashCode，HeapPage.isSlotUsed。

### 2.4.1 hashCode

这个地方存疑，暂时写进报告。HeapPageId和RecordId两个类都有hashCode方法，但并没有在Lab1中过多使用。由于是自己设计哈希编码，暂时使用了较为简单的加法处理和质数处理。

```

1 return 3 * tableId + pageNum;
2 return 3 * pid.hashCode() + tupleNum;

```

### 2.4.2 HeapPage.isSlotUsed

强调一下：以后做作业**一定好好读题目好好读题目好好读题目**。哭泣Debug好久才想到了位移处理的办法，不过好在最后自己是想出来了，在仔细回看帮助文档的时候也印证了自己方法的正确性。如果不给提示的话，这一部分，加上前面的**getNumTuples**和**getHeaderSize**，三个方法的实现也可以算是难点之一了。

```
1 public boolean isSlotUsed(int i) {
2     if ((header[i / 8] >> (i % 8) & 1) == 1)
3         return true;
4     return false;
5 }
```

## 2.5 HeapFile

整个Lab1最难的地方。事后分析一下：其一，**构造函数传参少**（只包含File和TupleDesc）是让我们觉得无从下手的首要因素；其二，**前面大部分练习改动较少**，使得在这个练习的时候，不敢做过多的改动（比如，不敢设计新的实体类实现DbFileIterator接口）。

主要介绍两个方法：HeapFile.readPage, HeapFile.iterator.

### 2.5.1 HeapFile.readPage

这里是BufferPool调用的方法，实现对磁盘中指定页的提取。我的理解是，在这里pid（HeapPageId）就和普通的int没有什么区别，是Page的标号，从0开始不断增加。帮助文档中给了提示要使用**RandomAccessFile**方法实现对文件的随机位置访问，简单查一下java的开发说明不难写出。

```
1 try {
2     RandomAccessFile raf = new RandomAccessFile(f, "r");
3     raf.seek(pid.getPageNumber() * BufferPool.getPageSize());
4     raf.read(data, 0, data.length);
5     p = new HeapPage((HeapPageId)pid, data);
6 } catch (Exception e) {
7     e.printStackTrace();
8 }
```

### 2.5.2 HeapFile.iterator

**Lab1杀手，没有之一**。返回一个接口，对于前面的较少改动来讲，这一部分则是大下功夫。加之前面练习的很多iterator<>方法都是使用java封装好的List或者Map进行返回值处理，这一部分刚上手毫无头绪，采用了简单粗暴的方法：

```
1 return new DbFileIterator{
2     ...
3 };
```

实践过程中发现问题。Unit Test过程中发现这样的方式无法存储下列信息：

```
1 private int curPage;
2 private Iterator<Tuple> it;
```

curPage代表当前页位置，it代表当前页内指向数据的指针。如果采用一开始设计的方式，**将二者存放在HeapFile类中是行不通的**。在换页以及DbFileIterator.close的操作上会频繁报错。

于是该换思路，设计新的类**HeapFileIterator实现DbFileIterator接口**，将上述信息保存下来，随着程序进程不断更新，以达到预期的效果。

由于改动较多，直接附上全部代码：

```
1 private class HeapFileIterator implements DbFileIterator {
2     private TransactionId tid;
3     private int curPage;
4     private Iterator<Tuple> it;
5     public HeapFileIterator(TransactionId tid) {
6         this.tid = tid;
7     }
8     @Override
9     public void open() throws DbException, TransactionAbortedException {
10         curPage = 0;
11         HeapPageId pid = new HeapPageId(getId(), curPage);
12         HeapPage hp = (HeapPage)Database.getBufferPool().getPage(tid, pid,
Permissions.READ_ONLY);
13         it = hp.iterator();
14     }
15
16     @Override
17     public boolean hasNext() throws DbException,
TransactionAbortedException {
18         if (it == null)
19             return false;
20         if (it.hasNext())
21             return true;
22         if (curPage + 1 < f.length() / BufferPool.getPageSize()) {
23             curPage++;
24             HeapPageId pid = new HeapPageId(getId(), curPage);
25             HeapPage hp = (HeapPage)Database.getBufferPool().getPage(tid,
pid, Permissions.READ_ONLY);
26             it = hp.iterator();
27             return it.hasNext();
28         }
29         else
30             return false;
31     }
32     @Override
33     public Tuple next() throws DbException, TransactionAbortedException,
NoSuchElementException {
34         if (!hasNext())
35             throw new NoSuchElementException();
36         return it.next();
37     }
38     @Override
39     public void rewind() throws DbException, TransactionAbortedException {
40         open();
41     }
42     @Override
43     public void close() {
44         curPage = 0;
45         it = null;
46     }
47 }
```

其中比较复杂和麻烦的地方是17-31行的hasNext方法实现，这个地方要考虑到换页的判断，很多时候一个不小心就会报null pointer异常。卡了很久，而变量curPage为这个方法提供了不少便利。

## 2.6 SeqScan

有了HeapFile的历练，这个练习的实现在HeapFile的基础上稍作修改和调整就可以了，而且随着练习的深入，思路和框架不断清晰，实现起来也是比较快的。

这个地方有一个小插曲。我在跑unit test和system test的时候，如果使用run，会出现“**Failed to start test**”或者“**No tests were found**”的报错，但是使用debug就没有问题。

这个问题一直没有得到解决，我一直用的debug方法通过的所有测试。直到我完成了Lab1的所有练习，网上查阅了相关资料，发现了原因：**金山词霸和IDEA无法兼容，如果两个软件同时开启就会导致IDEA的test有时无法正常运行!!!**在我关闭了金山词霸之后，run test恢复了正常。