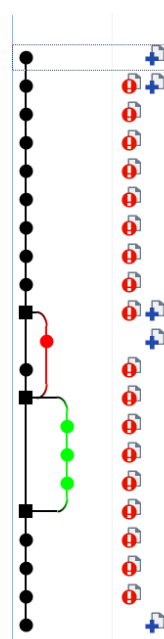



SimpleDB Lab1 实验报告4

Commit 记录


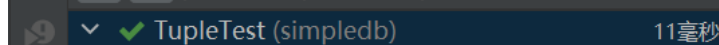
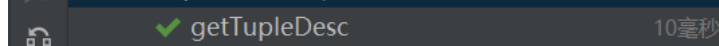
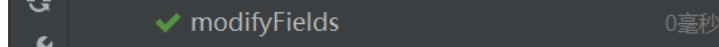
工作树变更				
		<div>master github/master origin/HEAD origin/master Lab1 finish and add...</div> <div>Lab1 finish test</div> <div>Lab1_exercise6 finish test</div> <div>Lab1_exercise5 finish test</div> <div>Lab1_exercise4 finish test</div> <div>correct Lab1_exercise1</div> <div>correct Lab1_exercise1</div> <div>correct Lab1_exercise1</div> <div>implement Lab1_exercise3 without test</div> <div>Merge remote-tracking branch 'github/master'</div> <div>Create README.md</div> <div>format code</div> <div>Merge branch 'main' into 'master'</div> <div>Lab1_exercise2 pass</div> <div>Lab1_exercise1 pass</div> <div>Lab1_exercise1 pass</div> <div>Lab1_exercise1 pass</div> <div>init</div> <div>add out directory to .gitignore</div> <div>Update .gitignore</div> <div>SimpleDB NKU 2020</div>	<div>NKULYX</div> <div>NKULYX</div> <div>NKULYX</div> <div>NKULYX</div> <div>NKULYX</div> <div>NKULYX</div> <div>G12LeaLiu20</div> <div>TTATT</div> <div>TTATT</div> <div>TTATT</div> <div>TTATT</div> <div>TTATT</div> <div>TTATT</div> <div>TTATT</div> <div>TTATT</div> <div>TTATT</div> <div>TTATT</div> <div>TTATT</div> <div>NKU-DBIS-DB</div> <div>NKU-DBIS-DB</div> <div>NKU-DBIS-DB</div>	<div>2022/3/26 12:07:51</div> <div>2022/3/26 12:05:15</div> <div>2022/3/19 18:17:12</div> <div>2022/3/19 17:40:04</div> <div>2022/3/18 1:15:51</div> <div>2022/3/17 23:47:01</div> <div>2022/3/17 22:43:14</div> <div>2022/3/16 0:07:22</div> <div>2022/3/13 16:20:31</div> <div>2022/3/13 11:41:10</div> <div>2022/3/13 11:36:09</div> <div>2022/3/13 11:40:12</div> <div>2022/3/13 11:11:17</div> <div>2022/3/13 11:07:47</div> <div>2022/3/13 0:47:12</div> <div>2022/3/13 0:41:14</div> <div>2022/3/12 23:55:40</div> <div>2022/3/10 21:42:03</div> <div>2021/3/19 16:55:07</div> <div>2020/3/6 15:33:40</div> <div>2020/3/6 12:19:48</div>

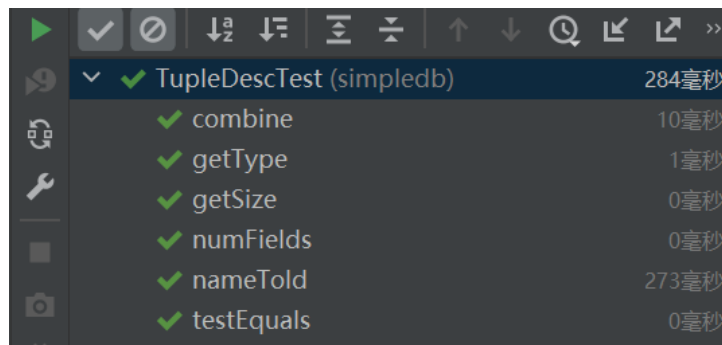
实验设计思路

Exercise 1

在 Exercise 1 中，实验要求我们实现 `TupleDesc.java` 和 `Tuple.java` 两个类，并通过单元测试 `TupleDescTest.java` 和 `TupleTest.java`。根据实验介绍内容以及代码内的注释，我们可以理解，`TupleDesc.java` 要实现的就是描述一张表的模式中相关属性的信息，包括属性的名称和类型。而 `Tuple.java` 中则是描述的每一个元组的内容。

在 `TupleDesc.java` 中，为了能够保存对于模式中属性的描述，我们采用了 `ArrayList` 来存储，存储的每一个元素是已经实现好的 `TDItem`，这个类中包含两个可以访问的属性，分别用来描述属性的名称和类型。对于所有的 `get` 方法，都需要首先判断所访问的下标是否合法，如果是合法下标再去进行搜索，如果并不是合法下标，则需要抛出 `NoSuchElementException` 的异常。而在所有 `equals` 方法中，都需要首先判断传入的参数是否为空，判断是否和当前的类型一致，在这些条件都满足的情况下，才能将 `Object` 对象强制转化成我们所需要的类型，再进一步标胶内部的属性信息是否相同。

	✓ TupleTest (simpledb)	11毫秒
	✓ getTupleDesc	10毫秒
	✓ modifyFields	0毫秒
	✓ modifyRecordId	1毫秒



Method	Time (ms)
✓ TupleDescTest (simplifiedb)	284
✓ combine	10
✓ getType	1
✓ getSize	0
✓ numFields	0
✓ nameTold	273
✓ testEquals	0

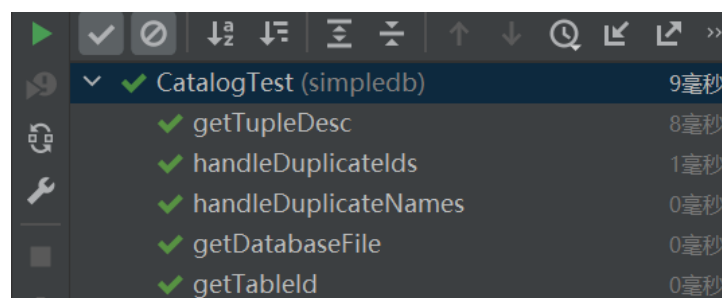
Exercise 2

在 Exercise 2 中，实验要求我们实现 `Catalog.java` 这个类，这个类是整个数据库中，数据表和模式的一个目录，在整个项目中采用单例模式，即可以认为这是一个全局变量，所有的访问都需要对这一个变量进行。

为了能够存储数据表和模式，我们在这个类中创建了一个 `Table` 内部类，用来辅助描述表的信息，其中包含了表所在的文件，表的名称，表中的主键以及描述这个表的一个ID。考虑到目录中的信息可能会经常发生增删，为了能够高效完成增删，采用了 `LinkedList` 来存储 `Table`，并且由于这个类需要能够提供一个遍历 `tableId` 的迭代器，因此又增加了一个描述 `tableId` 的 `LinkedList`，这个链表与存储 `Table` 的链表在位置上是——对应的关系。

在向目录中插入一张表的时候，首先考虑到如果表的名字为空，则只能通过 `tableId` 来判断当前的表是否和目录中的某一个表一致，如果存在一致，则用新的信息覆盖掉之前的信息。如果不存在，则在目录的最后插入这张新表，并且要将 `tableId` 同样加入到对应的链表中。而当考虑表的名称和标号都是用来描述表的信息的，因此在名字不为空的情况下，需要同时考虑这两个因素，任何一个因素发生冲突都需要覆盖掉当前目录中对应位置的信息。

同样在所有的 `get` 方法中，都需要判断当前的下标是否是合法位置，否则需要抛出 `NoSuchElementException` 异常。



Method	Time (ms)
✓ CatalogTest (simplifiedb)	9
✓ getTupleDesc	8
✓ handleDuplicateIds	1
✓ handleDuplicateNames	0
✓ getDatabaseFile	0
✓ getTableId	0

Exercise 3

在 Exercise 3 中，实验要求我们实现 `BufferPool.java` 这个类，这个类是用来描述数据库的缓冲区的状态的，在整个项目中同样采取单例模式，即所有对缓冲区的访问都需要对同一个变量进行操作。

缓冲区负责缓存最近从硬盘中访问到的数据文件，方便下一次访问时快速读取。为了规范，所有的方法都需要通过缓冲区来读取和访问数据文件，因为只有通过缓冲区读取，才能够将数据文件从硬盘加载到缓冲区中，否则数据文件并没能成功进入缓存，下一次访问开销将还是很大。

在缓冲区中描述了每个页的大小，以及缓冲区所能缓存的最大页数。为了能够方便存储和查找缓冲区中缓存的页，考虑采用 `HashMap<PageId, Page>` 来存储页，这样就能够近似 $O(1)$ 的时间复杂度下查询到在缓冲区中的页信息。

在实现 `getPage` 方法的时候，首先根据 `PageId` 查找当前页是否在缓冲区中，如果在则直接返回，如果不在，则需要通过 `Catalog` 去读取这个页面，并将它加载到缓冲区中，这里要注意，`Catalog` 的获取要通过 `Database.getCatalog()` 方法，获取到全局唯一的单例变量。而在 Lab1 中，我们并不需要考虑缓冲区满的情况，这个时候我们只需要抛出异常即可，在后续的实验，会完成页置换的方法。

Exercise 4

在 Exercise 4 中，要求我们实现 `HeapPageId.java` `RecordID.java` 和 `HeapPage.java` 三类，`HeapPageId.java` 中主要描述了某一个页所属的表以及在表中的位置信息。`RecordID.java` 中主要描述了某一个 tuple 所属的页和在页中的位置信息。而 Lab1 在 `HeapPage.java` 中只需要实现一些简单的数据信息的获取方法。

由于在 `BufferPool.java` 使用了 `PageId.java` 作为 key，因此需要在 `HeapPageId.java` 类中实现 `hashCode` 方法，这里直接调用了 Java 在 `Object` 类中提供的 `hash` 方法，将 `tableId` 和 `pageNum` 两个变量同时加入到 `hashCode` 的计算中。

在 `HeapPage.java` 中，计算一个页中有多少元组的时候，参考公式

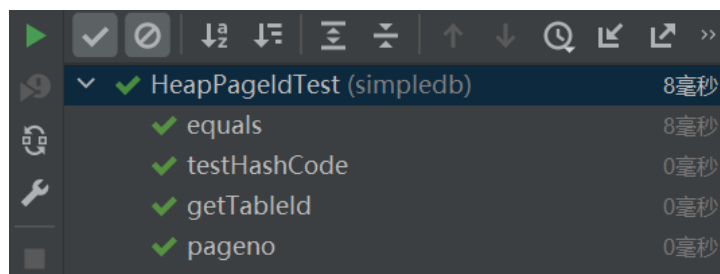
$$\left\lfloor \frac{PageSize \times 8}{TupleDecs \times 8 + 1} \right\rfloor$$

在计算头部大小的时候，将每8个bit打包成一个byte进行存储即可，参考公式

$$\left\lceil \frac{slotNum}{8} \right\rceil$$

在确定第 i 个 slot 是否被占用的时候，需要做一定的转化，首先计算 $i/8$ 确定他在 head 的第几个 byte 中，然后再根据 $i \bmod 8$ 来计算在这个 byte 中的第几个位置，将这个 byte 右移后与 1 做与，判断是否为 1 即可。

在获取 `Tuple` 的迭代器时，要注意只需要返回不为空的 `Tuple`，也就是说，在遍历 `tuples` 数组的时候，要根据下标利用函数 `isSlotUsed` 判断这个位置是否有 `Tuple`，如果有再将其加入到一个临时的链表中，最后返回这个链表的一个迭代器即可。



▶	✓	⊘	↓↑	↓↑	≡	÷	↑	↓	🔍	↶	↷	»
🔄	✓	RecordIdTest (simplifiedb)	9毫秒									
🔧	✓	equals	8毫秒									
	✓	tupleno	1毫秒									
	✓	hCode	0毫秒									
	✓	getPageId	0毫秒									

▶	✓	⊘	↓↑	↓↑	≡	÷	↑	↓	🔍	↶	↷	»
🔄	✓	HeapPageReadTest (simplifiedb)	12毫秒									
🔧	✓	getId	12毫秒									
	✓	getSlot	0毫秒									
	✓	getNumEmptySlots	0毫秒									
	✓	testIterator	0毫秒									

Exercise 5

在 Exercise 5 中，实验要求我们实现 `HeapFile.java` 这个类，在这个类中保存了在磁盘中的文件和模式的属性信息。在 Lab1 中只要求实现 `readPage` 和 `iterator` 方法。

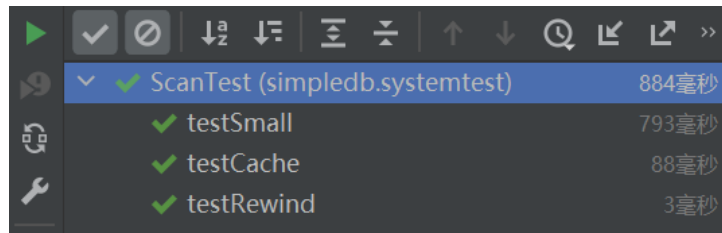
由于 `readPage` 方法是底层提供从磁盘文件中读取文件内容的方法，而 `BufferPool` 中也是通过调用 `DBFile.readPage` 来读取页内容的，因此，我们需要在这个类中实现从磁盘文件中读取内容的方法。考虑采用 `RandomAccessFile` 来实现，因为这个方法能够在文件中实现随机读取，在实验中我们只需要读取特定的一个页的内容，因此需要能够随机访问。根据 `PageId` 中提供的方法获取到这个页在文件中的位置，再通过每个页的大小来确定这个页在文件中的起始位置。然后将页从文件中读入内存。这个过程在 `BufferPool` 调用相关方法的时候，就能够实现将页加载到缓冲区。

根据文档和注释，我们可以得知，`iterator` 方法想让我们实现一个 `DbFileIterator`，通过这个迭代器来访问整个文件中的 `Tuple`。考虑到一个文件中会有多个页，而每个页都可以通过 `iterator` 方法获取到遍历这个页中 `Tuple` 的迭代器，因此考虑采用逐页访问的方法，即每次依次获取文件中的页信息，然后通过 `iterator` 方法来访问这个页中的全部 `Tuple`，如果访问完毕，则判断时候还有下一页，如果有下一页，则获取下一页的 `iterator` 然后继续遍历，直到遍历完所有页的所有 `Tuple`。具体实现方法可以查阅代码中 `iterator` 方法中的匿名内部类。

▶	✓	⊘	↓↑	↓↑	≡	÷	↑	↓	🔍	↶	↷	»
🔄	✓	HeapFileReadTest (simplifiedb)	738毫秒									
🔧	✓	getId	718毫秒									
	✓	readPage	5毫秒									
	✓	getTupleDesc	2毫秒									
	✓	numPages	2毫秒									
	✓	testIteratorBasic	5毫秒									
	✓	testIteratorClose	6毫秒									

Exercise 6

在 Exercise 6 中，要求我们实现 `SeqScan` 方法，来将 Lab1 的全部功能串联起来进行测试。实验中具体实现的方法都可以通过调用之前的方法来简单实现，并没有复杂的逻辑，再次不再赘述了。



✓	✓ ScanTest (simplifiedb.systemtest)	884毫秒
✓	testSmall	793毫秒
✓	testCache	88毫秒
✓	testRewind	3毫秒

重难点问题

在 Exercise 5 中，如何实现 `iterator` 方法是在 Lab1 中比较困难的部分，需要我们自己实现一个辅助的类。考虑到一个文件中会有多个页，而每个页都可以通过 `iterator` 方法获取到遍历这个页中 `Tuple` 的迭代器，因此考虑采用逐页访问的方法，即每次依次获取文件中的页信息，然后通过 `iterator` 方法来访问这个页中的全部 `Tuple`。

如何获取页，考虑采用 `Database.getBufferPool().getPage()` 方法，传入 `transactionId`, `heapPageId` 和一个 `Permission` 权限来获取一个指定的页。`HeapPageId` 是通过文件的 ID 和当前想要读取的页码来构造的。然后可以调用 `HeapPage` 中的 `iterator` 方法获取能够遍历 `Tuple` 的迭代器。

对于 `hasNext` 方法，需要做两方面的判断，首先判断当前持有的 `tupleIterator` 的 `hasNext` 方法是否返回 `true`，如果返回 `true`，则方法返回 `true`，如果返回值为 `false`，则需要判断是否还有下一页，如果还有下一页，要更新当前的页为下一页，并重新获得该页的 `iterator`，并返回 `tupleIterator` 的 `hasNext` 方法的返回值。直到所有的页也都访问完，则返回一个 `false`。

```
1 public DbFileIterator iterator(TransactionId tid) {
2     // some code goes here
3     return new DbFileIterator() {
4
5         int currentPageIndex;
6         HeapPage currentPage;
7         Iterator<Tuple> tupleIterator;
8
9         @Override
10        public void open() throws DbException,
11        TransactionAbortedException {
12            this.currentPageIndex = 0;
13            this.currentPage =
14            (HeapPage)Database.getBufferPool().getPage(
15                tid,
16                new HeapPageId(getId(), currentPageIndex),
17                Permissions.READ_ONLY);
18            this.tupleIterator = this.currentPage.iterator();
19        }
20    }
21 }
```

```

19         @Override
20         public boolean hasNext() throws DbException,
TransactionAbortedException {
21             if(this.tupleIterator == null){
22                 return false;
23             }
24             if(this.tupleIterator.hasNext()){
25                 return true;
26             }else {
27                 if(this.currentPageIndex != numPages() -1){
28                     this.currentPageIndex++;
29                     this.currentPage =
(HeapPage)Database.getBufferPool().getPage(
30                         tid,
31                         new
HeapPageId(getId(),currentPageIndex),
32                         Permissions.READ_ONLY);
33                     this.tupleIterator =
this.currentPage.iterator();
34                     return this.tupleIterator.hasNext();
35                 }else{
36                     return false;
37                 }
38             }
39         }
40
41         @Override
42         public Tuple next() throws DbException,
TransactionAbortedException, NoSuchElementException {
43             if(this.tupleIterator == null){
44                 throw new NoSuchElementException();
45             }
46             if(this.tupleIterator.hasNext()){
47                 return this.tupleIterator.next();
48             }
49             throw new NoSuchElementException();
50         }
51
52         @Override
53         public void rewind() throws DbException,
TransactionAbortedException {
54             open();
55         }
56
57         @Override
58         public void close() {
59             this.tupleIterator = null;
60             this.currentPage = null;
61         }
62     };
63 }

```

