



南開大學  
Nankai University

计算机学院  
SimpleDB 实验报告

# Lab1

姓名：朱霞洋

学号：2113301

专业：计算机科学与技术

2023 年 3 月 18 日

## 摘要

lab1 实现数据库基本的存储逻辑结构，具体结构如图4.10所示。

代码链接: [朱霞洋的 Gitlab 仓库](#)

## 目录

<b>1</b>	<b>设计思路</b>	<b>2</b>
1.1	Exercise1 Fields and Tuples . . . . .	2
1.1.1	Tuple . . . . .	2
1.1.2	TupleDesc . . . . .	2
1.2	Exercise2 Catalog . . . . .	3
1.3	Exercise3 BufferPool . . . . .	4
1.4	Exercise4 HeapPage . . . . .	4
1.4.1	RecordId . . . . .	5
1.4.2	HeapPageId . . . . .	5
1.4.3	HeapPage . . . . .	5
1.5	Exercise5 HeapFile . . . . .	6
1.6	Exercise6 Operator . . . . .	7
<b>2</b>	<b>重难点</b>	<b>7</b>
2.1	Tuples, DbFiles 等类的 Iterator 实现 . . . . .	7
2.2	BufferPool 中的 getPage() 函数 . . . . .	8
2.3	HeapFile 中的 readPage() 函数 . . . . .	8
<b>3</b>	<b>改动部分</b>	<b>8</b>
3.1	Catalog 中新建 Table 类 . . . . .	8
3.2	HeapFile 中为实现 Iterator 新建 getTuples 函数 . . . . .	8
3.3	Exercise5 中完善 BufferPool 的 getPage() 函数 . . . . .	8
<b>4</b>	<b>Test 结果</b>	<b>8</b>

# 1 设计思路

## 1.1 Exercise1 Fields and Tuples

Exercise1 中完成 Tuple 与 TupleDesc 两个文件的实现，其中 Tuple（元组）是数据库中的数据，它由多个 Field 构成，每个 Field 记录该数据的一个属性，此外每个 Tuple 有对应的 RecordId；而 TupleDesc 则是数据库中的 schema，记录了数据存储的模式，包括每块 Field 的名称（fieldName）和数据类型（fieldType），如图：1.3所示

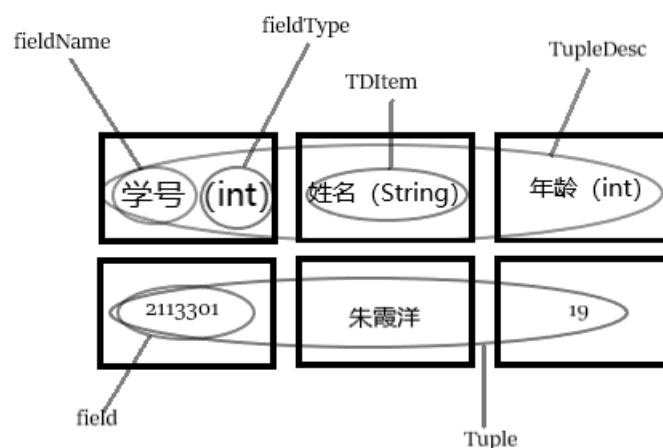


图 1.1: Tuple and TupleDesc

### 1.1.1 Tuple

在 Tuple 中用 ArrayList 容器来记录每一个 field，可以做到快速的随机访问并且其长度可以灵活变化，同时可以便于读写各个 field；此外，在实现 Tuple 的迭代器 iterator 时，可以直接返回一个 ArrayList 的 iterator。

```

1 private ArrayList<Field> Fields=new ArrayList<>();
2 ...
3 public Iterator<Field> fields(){
4     return Fields.iterator();
5 }

```

在 Tuple 中要实现的方法主要是各个字段（field）以及对应 RecordId 的读与写，这些方法实现较为简单，具体可见 Gitlab 仓库，其中较为重要的是迭代器 iterator 的实现，在读取数据会将会用到，该方法的实现直接调用了 ArrayList 的迭代器，如上方的代码所示。

### 1.1.2 TupleDesc

TupleDesc 中构造了一个 TDItem 辅助类，每个 TDItem 是一个 Field 的信息。例如学号即为一个 TDItem，fieldName 为”学号”，fieldType 为 int。

该部分用 ArrayList 来记录各个 TDItem，能快速随机访问，且长度灵活，在实现 TDItem 的迭代器时也能直接返回一个 ArrayList 的 iterator。

该部分主要实现的方法包括各个 field 信息的读与写，较为简单，其余比较重要的方法有：

- 迭代器 iterator  
用于遍历 TDItem，直接返回一个 ArrayList 的 iterator 即可；
- 获取 Tuple 大小 getSize()  
用 getLen() 函数获取每个 field 的大小，相加得到 TupleSize；
- 合并两个表 merge()  
该函数用于合并两个表时使用，将两个表的 TDItem 连接为一个 TDItem 数组，注意两个表可能有相同的 TDItem；
- 判断相等函数 equals()  
判断两个表的 Schema 是否相同，注意 TDItem 中 fieldName 为 null 的情况。

## 1.2 Exercise2 Catalog

Catalog 是数据库中的目录，管理着数据库中所有的 table，是 DataBase 中的单例对象，可以用 DataBase.getCatalog() 访问。在实现这个类时，构造了一个 Table 辅助类。每个 table 有一个 DbFile (Lab1 中是 HeapFile)，是数据库磁盘文件的接口，记录着 table 中数据的信息，此外还有一个表名 name 和表的主键名称 pkeyField。

```

1  public class Table{           //create a helper class Table to better understand how catalog works
2      private DbFile file;
3      private String name;
4      private String pkeyField; //each table contains a Dbfile, a name ,and a pkeyField
5      ...
6  }
```

DbFile 中有 getId() 方法，可以获取此 Dbfile 对应表的 tableid。这个 id 由哈希方法生成，在后续 Exercise 中会涉及，因此 Catalog 中也使用 HashMap 记录 tableid 和 table、表名 name 和 table、tableid 和主键的映射关系。

```

1  private HashMap<Integer,Table> idToTable; //crate a HashMap so we can easily find table via its id
2  private HashMap<String,Integer> nameToId; //enable us to find id via its table name
3  private HashMap<Integer,String> idToPkey; //enable us to find pkey via its id directly
```

在 Catalog 中要实现的重要的方法有：

- 添加表 addTable()  
即往 Catalog 中添加一个新的表，每添加一个表要往三个哈希表中添加新的映射关系；而如果 Catalog 中已经存有该表（有相同的表名），则更新表的信息，删除先前的表，插入新的表。
- 清空 Catalog clear()  
清空三个哈希表即可。
- tableId 的迭代器  
调用 Hashmap idToTable 中的 keyset.iterator() 即可。

此外要实现的方法主要是对 tableid 等信息的读取，只用在 HashMap 中进行读取并返回即可。

### 1.3 Exercise3 BufferPool

BufferPool 是 SimpleDb 数据库的缓存组件, 是整个系统最核心的组件, 任何地方访问一个 page 都需要通过 `bufferPool.getPage()` 方法, 在读取数据时会先在缓冲池 BufferPool 中读取, 如果没有目标 Page 则会从磁盘中读取并放入 BufferPool 中。

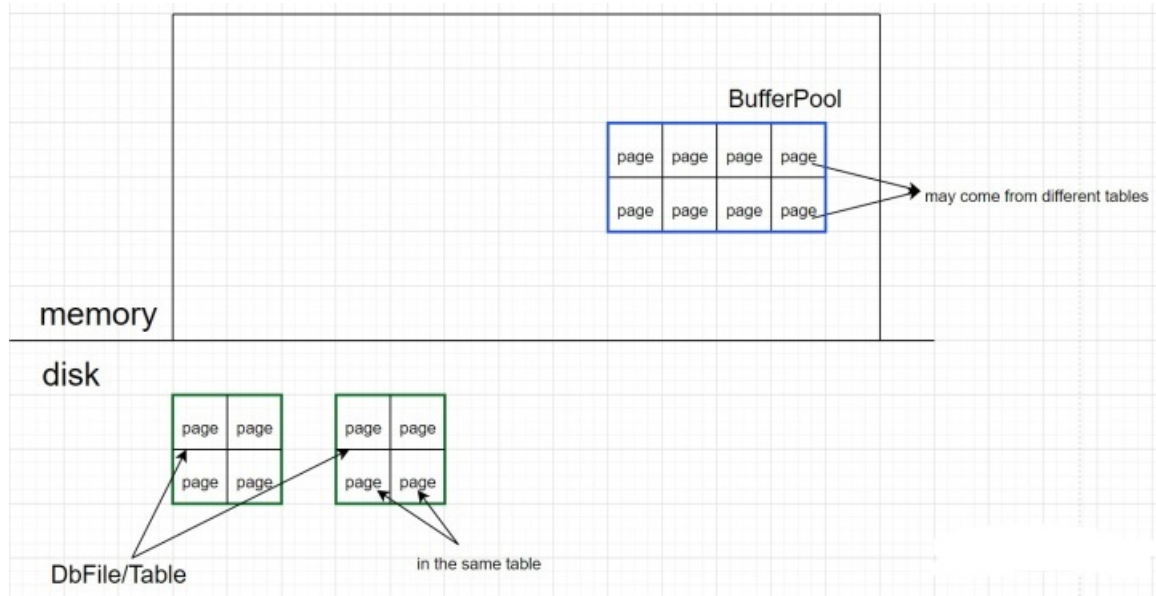


图 1.2: BufferPool

BufferPool 也是数据库中的单例对象, 可以用 `DataBase.getBufferPool()` 访问。在 Exercise2 中, 只用实现 BufferPool 的构造函数与 `getPage()` 函数。 `getPage` 函数将在 Exercise6 的 Seqscan 中调用, 通过每个 Page 的 PageId 来在 BufferPool 中读取 Page, 因此在 BufferPool 中也构建一个 PageId 和 Page 的哈希表, 能够快速用 id 找到对应 Page。注: Exercise2 不要求实现当 BufferPool 中没有对应 id 的 Page 的情况, 大致代码如下:

```

1  public Page getPage(TransactionId tid, PageId pid, Permissions perm)
2      throws TransactionAbortedException, DbException {
3      if(idToPage.containsKey(pid)){ //if Page with id pid does exist, return the page
4          return idToPage.get(pid);
5      }
6      else {
7          throw new DbException("error"); //implement an eviction policy in the future labs
8      }

```

### 1.4 Exercise4 HeapPage

在该部分要实现 HeapPage、HeapPageId、RecordId 三个类, 是 Lab1 中的重难点。

HeapPage 是内存 BufferPool 中的一个页, 包含某个 table 中的一部分 (若干个 Tuple 和 Tuple-Desc); HeapPageId 通过 tableid 和 pageNo 来标识一个 HeapPage, 这是在内存 BufferPool 中寻找 HeapPage 的依据; RecordId 则是 Page 中某个 Tuple 的标识, 是此前实现的 Tuple 中的成员变量。

### 1.4.1 RecordId

RecordId 是 Tuple 的标识，记录了 Tuple 所在 Page 的 PageId (成员变量 pid)，以及自身在 Page 中的序号 (成员变量 tupleNo)。该部分要实现的方法较为简单，比较重要的有 RecordId 的判断相等 equals() 方法和 hashCode() 方法。此处 hashCode() 要返回一个 RecordId 的哈希码，直接使用 Objects.hash(pid,tupleno) 得到。

### 1.4.2 HeapPageId

HeapPageId 是接口 PageId 的实现。储存了 HeapPage 所在 HeapFile 的 tableid (tableId)，以及自身在 HeapFile 中的序号 (pageNo)。实现也较为简单，主要是构造函数一些信息的读取，比较重要的是 equals() 方法和 hashCode() 方法，用 Objects.hash(tableId,pageNo)。

### 1.4.3 HeapPage

每个 HeapPage 是一个 HeapFile 的一块，记录了其对应的 PageId，其存储的数据的 TupleDesc 和一系列 Tuples。在存储结构上，每个 HeapPage 有一个 headers 数组和若干 Slots。Headers 数组的每一位记录了对应 Slot 的使用情况，如果该位为 1，则对应 Slot 存储了一个 Tuple 数据，而如果该位为 0，则对应 Slot 没有存储数据，如图：

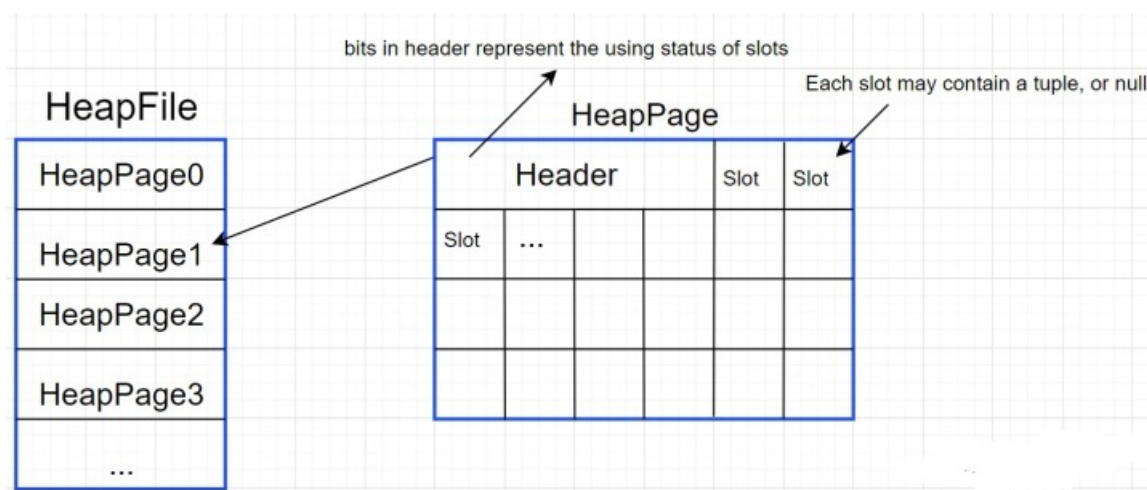


图 1.3: The Storage Architecture of HeapPage

该部分要实现的最重要的几个方法为：

- 判断 Slot 是否为空 isSlotUsed()

此处要注意 Java 中数据在内存中为大端字节序，要根据此来实现方法。最终使用位运算来判断 Headers 中的第 i 个 bit 是否为 1，如果为 1 则 return true，Slot 存有数据，否则 return 0；

- 获取空白 Slot 的数目 getNumEmptySlots()

反复调用 isSlotUsed()，得到总的空白 Slot 数即可。

- HeapPage 中 Tuples 的迭代器

由于 HeapPage 中并非每一个 Slot 都存有 Tuples，因此在实现迭代器时要判断 Slot 是否为空，最终得到一个 Tuple 的迭代器。

- 获取当前 Page 存储的 Tuples 的数目 `getNumTuples()`

由于一个 Tuple 除了本身占据的空间 (`TupleDesc.getSize()`) 外, 还在 headers 中占有一个 bit 大小的标识位置空间, 则 Tuples 的数目为:

$$\text{floor}(\text{getPageSize}() / (\text{TupleDesc.getSize}() + 1))$$

## 1.5 Exercise5 HeapFile

Lab1 中用 HeapFile 来实现 DbFile, 每个 HeapFile 对应一个表, HeapFile 中包含了表的 Schema 即 TupleDesc 和表的 id 标识 tableid。当在 BufferPool 中找不到目标 HeapPage 时, 就会根据 PageId 获得对应的 tableid, 然后在磁盘上找到 HeapFile 再根据 PageId 读取对应的 HeapPage。

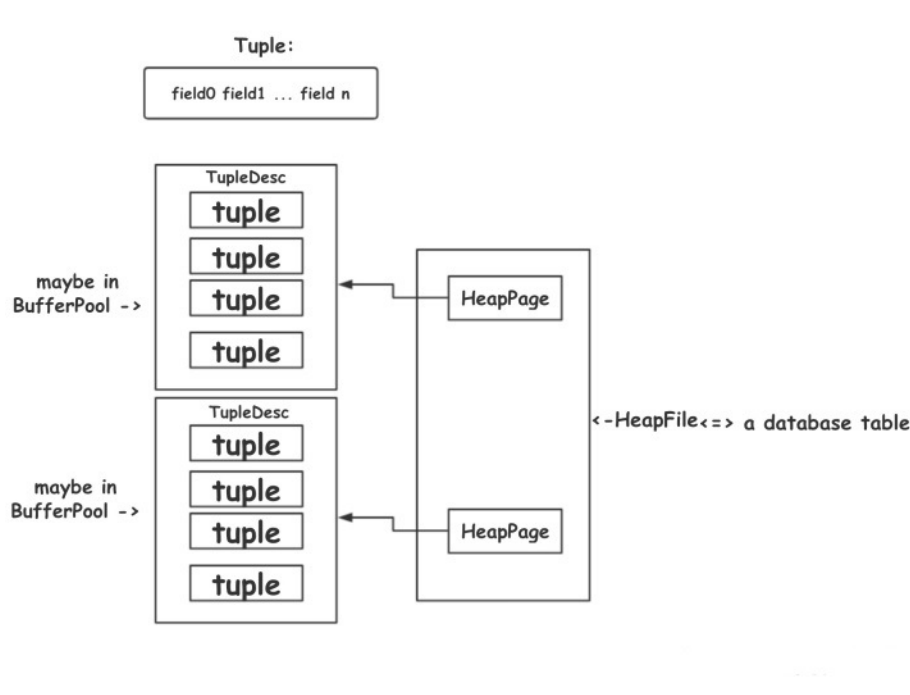


图 1.4: HeapFile 与各数据层次关系

HeapFile 中要实现的重要的方法有:

- 生成 tableid 的 `getId()` 函数

可以调用 `File` 的 `getAbsolutePath()` 方法得到 `File` 的存储位置 `Path`, 然后根据该 `Path` 生成 `HashCode`, 从而得到每一个 `HeapFile` 的哈希码

```
1 public int getId() {
2     return File.getAbsolutePath().hashCode();
3 }
```

- 读取一个 page 数据的 `readPage()` 函数

该函数会被 `BufferPool` 中的 `getPage()` 函数调用, 用于在一个 `HeapFile` 中读取一个 `HeapPage` 的数据。读取数据时, 首先算出该 `Page` 的偏移量:

```
1      int offset = BufferPool.getPageSize()*pid.getPageNumber();
```

然后用一个 RandomAccessFile 类的对象来进行随机访问文件, 用一个 getPageSize() 大小的 byte 数组来存储数据, 并由此生成一个 HeapPage 进行返回。

```
1      RandomAccessFile random=null;
2      byte[] data=new byte[bp.getPageSize()];
3      try{
4          random=new RandomAccessFile(File,"r");
5          random.seek(offset);
6          random.read(data,0,data.length);
7          page=new HeapPage((HeapPageId) pid,data);
8      }catch(IOException e){
9          e.printStackTrace();
10     }
```

- HeapFile 的迭代器

HeapFile 的迭代器将会返回该文件中的 Tuples 的一个迭代器, 因此又要在 HeapFile 的每一个 HeapPage 里进行迭代遍历, 不断返回每一页的 Tuples 的迭代器。在该部分新构造了一个 getTuples() 函数, 用于根据 HeapPageId 获取这一个 Page 的 Tuples 的迭代器。

```
1      public Iterator<Tuple> getTuples(HeapPageId pid) throws TransactionAbortedException,
           DbException {
2          HeapPage page=(HeapPage) Database.getBufferPool().getPage(tid,pid,Permissions.READ_ONLY);
           //find the page via pid
3          return page.iterator();           //return the tuples in the page with id pid
4      }
```

## 1.6 Exercise6 Operator

该部分实现了一个 Seqscan 类, 较为简单, 大部分工作是封装 Exercise5 中的 iterator。在 SimpleDB 中进行一些查询操作的过程中, 先在 root operator 上调用 getNext(), 之后在子节点上继续调用 getNext, 一直下去, 直到 leaf operators 被调用, 然后从硬盘上读取 tuples, 并在树结构上传递。

该部分比较值得注意的有:

- getTuples() 函数不能直接返回原 tupleDesc, 要为在上 fieldName 添加表的别名前缀, 即 alias.fieldName。此部分用字符串的拼接即可完成。
- 对子节点的 iterator 进行调用。

## 2 重难点

### 2.1 Tuples, DbFiles 等类的 Iterator 实现

其中 TupleDesc 中的 TDItem, Catalog 中的 tableid, HeapPage 中 Tuples 的迭代器实现较为容易, 使用了 ArrayList 来囊括对象然后返回一个 ArrayList 的迭代器即可;



而较为复杂的是 **HeapFile** 的迭代器，需要不断从每一个 Page 中返回 Tuples 的迭代器。

## 2.2 BufferPool 中的 getPage() 函数

getPage() 函数用于从 BufferPool 中返回一个 Page 的数据，详细见 1.3 BufferPool。

## 2.3 HeapFile 中的 readPage() 函数

readPage() 函数用于从一个 HeapFile 中根据目标 PageId 读取对应 Page 的数据，详细见 1.5 HeapFile。

# 3 改动部分

## 3.1 Catalog 中新建 Table 类

Catalog 管理着数据库中所有 table，因此构建了这个 Table 辅助类，方便理解，每个 table 有一个 DbFile (Lab1 中是 HeapFile)，是数据库磁盘文件的接口，记录着 table 中数据的信息。

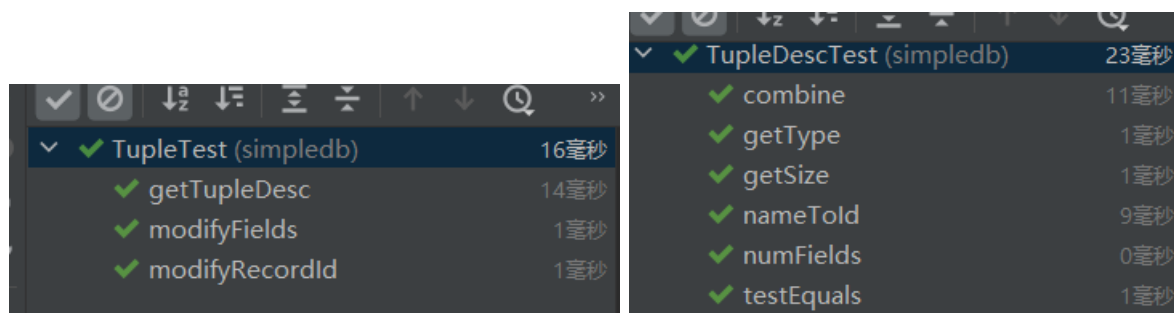
## 3.2 HeapFile 中为实现 Iterator 新建 getTuples 函数

HeapFile 的迭代器将在其每一个 HeapPage 中返回 Tuples 的迭代器，因此建立这个函数，可以根据 HeapPageId，返回这个 Page 中 Tuples 的迭代器。

## 3.3 Exercise5 中完善 BufferPool 的 getPage() 函数

在遍历 HeapFile 的每一个 HeapPage 时，需要现在 BufferPool 中找寻该 Page，如若没有则从磁盘读取。在 Exercise2 中没有实现当 BufferPool 没有对应页时的 getPage() 方法，而在 Exercise5 中为了遍历 Page 对其进行了完善。

# 4 Test 结果



✓ TupleTest (simpledb)	16毫秒
✓ getTupleDesc	14毫秒
✓ modifyFields	1毫秒
✓ modifyRecordId	1毫秒
✓ TupleDescTest (simpledb)	23毫秒
✓ combine	11毫秒
✓ getType	1毫秒
✓ getSize	1毫秒
✓ nameTold	9毫秒
✓ numFields	0毫秒
✓ testEquals	1毫秒

图 4.5: Exercise1

✓ CatalogTest (simplifiedb)	11毫秒
✓ handleDuplicateNames	9毫秒
✓ handleDuplicateIds	0毫秒
✓ getTableId	1毫秒
✓ getTupleDesc	1毫秒
✓ getDatabaseFile	0毫秒

图 4.6: Exercise2

✓ RecordIdTest (simplifiedb)	11毫秒	✓ HeapPageIdTest (simplifiedb)	11毫秒
✓ equals	10毫秒	✓ equals	9毫秒
✓ tupleno	0毫秒	✓ testHashCode	0毫秒
✓ getPageId	0毫秒	✓ getTableId	1毫秒
✓ hCode	1毫秒	✓ pageno	1毫秒
✓ HeapPageReadTest (simplifiedb)	17毫秒		
✓ getId	14毫秒		
✓ getSlot	1毫秒		
✓ testIterator	1毫秒		
✓ getNumEmptySlots	1毫秒		

图 4.7: Exercise4

✓ HeapFileReadTest (simplifiedb)	472毫秒
✓ getId	445毫秒
✓ testIteratorBasic	11毫秒
✓ testIteratorClose	9毫秒
✓ readPage	3毫秒
✓ numPages	2毫秒
✓ getTupleDesc	2毫秒

图 4.8: Exercise5

✓ ScanTest (simplifiedb.systemtest)	827毫秒
✓ testSmall	679毫秒
✓ testRewind	4毫秒
✓ testCache	144毫秒

图 4.9: ScanTest

```

BUILD SUCCESSFUL
Total time: 0 seconds
PS D:\soft\Java\javaProject\SimpleDB> java -classpath dist/simplifiedb.jar simplifiedb.test
1      1      1
2      2      2
3      4      4
PS D:\soft\Java\javaProject\SimpleDB>

```

图 4.10: Query Test