Daniel Yao 004-182-464

Ming Lu 904-035-039

CS143 Lab 1 write-up

Overall this lab took my partner and me about a week of time to complete. We did not change any APIs, but we did add an additional class to aid the implementation of the lab. We decided to follow the suggested steps while implementing this lab. No part of the lab is particularly difficult or confusing. However debugging of the lab did take some time.

The rest of this write up will discuss in detail how we implemented 10 classes (Tuple.java, TupleDesc.java, Catalog.java, BufferPool.java, HeapFile.java, HeapPageId.java, RecordID.java, HeapPage.java, SeqScan.java, and HeapFileIter.java (created class).

Tuple.java

This class was straight forward, we created 3 private variables of type Filed[], TupleDesc, and RecordId, because each tuple has a schema type, a field based on that type, and its ID which we use to identify it. The rest was implementing every required method for the class.

TupleDesc.java

This class is also straight forward since the all we really needed to do was code a way to keep track of all the TDItem objects that's given. We used an ArrayList<TDItem> to accomplish the task, because later on in the class requires usage of iterators which hints at the usage of lists. Most of the logic required in the methods were just for loops to loop through all element in the ArrayList and do comparison, return values, etc on each object in the array. We never implemented the method hashCode() as it was related to HashMaps which is unnecessary for lab 1.

Catalog.java

This class was very easy to implement, since its hardest part, loadSchema, was given to us. Since the job of the Catalog class is to act as a dictionary, keeping track of all tables in the database. It's very clear to see that it needs Lists to keep track of tables, their names, their keys, and their ids. Such, we made 4 ArrayList (because iterators is needed and it's natural to use an array type structure) variables of type DbFile, String, String, and Integer. Rest of the programming is literally just initializing the lists, writing accessor methods and setter methods, and using ArrayList functions.

BufferPool.java

The implementation for this class is very simple for lab 1. We had to write a constructor, 2 accessor methods and 1 setter method, since most of its methods are not needed for lab 1. We used HashMap data structure to help implement the methods in this class because we saw the import ConcurrentHashMap, and thought it was a hint to do so. Furthermore, because the role of the BufferPool is to act as a caching class, at which HashMaps excel.

HeapFile.java

Our implementation for this class took a little more time than for the other classes. We created 3 variables for this class, a TupleDesc, a File, and an ArrayList<Page>. We chose to do so because HeapFile.java is a collection of tuples, all have a TupleDesc property, and is backed up on disk by File. The hardest part for this class was readPage. Calculating the correct index and having java to read the number of bytes properly took some time. Furthermore, we attempted to use java method read(b, offset, length) at first, which for some reason caused undefined behaviors. We resorted to using two separate functions – skipBytes(# bytes to skip), read(data) – instead.

HeapPageId.java

This class was easy for us to implement because it told us the two parameters needed for the class, and most of the methods we needed to write were accessor methods. The only less trivial method in the class was the hashCode() method. We decided to create a unique hashcode for each page by taking advantage of the fact that the String class has a .hascode() method and that each tableID is unique as well as its page numbers. So we attempted to create a unique hashcode for each table and its page by converting the string concatenation of the tableId and pgNo into string, then called hashCode() on it.

RecordId.java

This class was easy. It's practically identical to HeapPageId.java, in terms of what needed to be implemented. Took no time at all.

HeapPage.java

Most of the methods in this class was already implemented for us. The harder parts of this class was the isSlotUsed(int i) method which required some bitwise manipulation, to determine whether the ith record is empty or not.

SeqScan.java & HeapFileIter.java

We decided to create a HeapFileIter.java class for the SeqScan.java class because java compiler gave warnings/errors when we attempted casting. So what we did was have HeapFileIter implement DbFileIterator and handle the meat of the code, and have SeqScan.java also implement DbFileIterator, but has a HeapFileIter member to call HeapFileIter's methods. So HeapFileIter acts as a wrapper function for SeqScan.

Because the class required us to be able to rewind an iterator to the start, we decided to use a ListIterator<Tuple> type variable, as it is a subclass of Java's iterator variable that has hasPrevious() and previous() function for us to use. Aside from that, most of this class was checking if passed arguments are valid and if they are, attempt to check if there is another tuple in the iterator if so, attempt to open it, etc.