SUTD 50.043 Database SimpleDB Lab1 Writeup Report

Group 29
Xu Muzi 1005641
Nigel Mun 1005031

Implementation Description

For the majority of the implementations for the functions and methods, I have simply followed the guided comments given, both in the handout document as well as the commented guide in the code itself.

Exercise 1
There are no main notable changes from the intended code, and I have just followed instructions in the comments.

Exercise 2
I have added a custom additional class called *CatalogTable* in *src/java/simpledb/common/CatalogTable.java*. This class is used in *src/java/simpledb/common/Catalog.java* in the variable dbFiles which is a ConcurrentHashMap<Integer, CatalogTable>. This allows the code in *Catalog.java* to be much readable, using methods in *CatalogTable.java* such as *getCatalogFile(), getCatalogName()* and *getCatalogPrimaryKey()* that is used in many of *Catalog.java* methods. If this method is not implemented, then I would have to use many *ConcurrentHashMap* instead for each of the get functions in *Catalog.java*, which may be more messy.

Exercise 3
The function *getPage()* in *BufferPool.java* is not fully implemented yet as functions such as acquiring lock or releasing lock and evicting page is not implemented yet for this lab. The unit test cases does not seem to fail for now, hence I believe that what I implemented in *getPage()* should be sufficient for now.

I used *ConcurrentHashMap* instead of a *HashMap* in *Catalog.java* and *BufferPool.java* and because it is synchronised and thread-safe, and exceptions may occur if many threads are performing adding/modifying to the object. In my opinion, I feel that this implementation is the safer option.

Exercise 4
For *HeapPageID.java* and *RecordID.java*, for the *hashCode()* method, I have implemented *Object.hash()* which requires an import of *java.util.Objects*. I have found this implementation online, where *Objects.hash()* can take one or more objects and provides a hashcode for them, which is what is required in the instructions.

Exercise 5

For HeapFile.java, I implemented the *MyDbFileIterator* class by extending the helper abstract class *AbstractDbFileIterator* and included several if conditions to check for iterator == null situation as if it is the case, *iterator.hasNext()* would not be able to pass through.
For readPage method, '*RandomAccess*' is used to read the bytes of a specific page from the file, and then read the bytes into a byte array. I then create a new HeapPage object using the HeapPageId from the PageId parameter and the byte array for the page data and return the result
The *insertTuple* method iterates over all pages of the heap file to find one with an empty slot. If found, the method inserts the tuple into the page and returns a singleton list containing the modified page. If no page with an empty slot is found, the method creates a new page, inserts the tuple into it, writes the new page to disk, and returns a singleton list containing the new page.
The *deleteTuple* method takes a tuple as an argument and deletes it from the corresponding page. The method returns a singleton list containing the modified page.

Exercise 6

For SeqScan.java, the *getTupleDesc()* method retrieves the TupleDesc of the table from the Catalog using the tableid, and creates a new TupleDesc with the same number of fields, but with field names that are prepended with the tableAlias. The *hasNext(), next(), close(),* and *rewind()* methods simply delegate to an internal iterator object.

There has not been any changes to API, and all the methods support the current API. Overall, there are no missing or incomplete implementations as all unit tests and system test for lab1 have passed.