Lab 1 Report

Group Members:

Yee Yi Xian	1004601
Rebecca Ling	1004413

Design Decisions and Code Explanation

Below we explain our design decisions and the ~~important~~ non-trivial part of our code, as split into the classes we were required to implement for Lab 1

Catalog

- We implemented a helper class called Table which contains the dbFile, tableName and pkeyField. This allowed us to implement less concurrentHashMaps
- We used 2 ConcurrentHashMaps,
    - Map between tableId and Table to easily retrieve the table tagged to the tableid, so that we can access the tableName, dbFile and pkeyField.
    - Map between tableName and tableId to easily retrieve the tableId, through calling the getTableId method.

Tuple and TupleDesc

- These classes were implemented to help ensure that DbFile.java can identify each tuple and table schema.To ensure we were able to calculate the page size, slot number and page header size easily, we ensured that the data types used were consistent.

BufferPool

- We used a ConcurrentHashMap between page and pageId so that it is easier to retrieve specific pages with the pageId.

HeapPage, HeapPageId, RecordId

- These classes were implemented to help the HeapFile to be able to iterate through the pages more easily, as each page is uniquely tagged with a hashcode.

- Hashcode requires a unique id multiplied by a constant, preferably an odd prime number, and adding another constant. This is to ensure that each hashcode is unique. In this case we used 37 as the odd prime number.
    - HeapPageId: `tableId * 37 + pageNumber`
    - recordId: `pid.hashCode * 37 + tupleNumber`
- For the iterator method in HeapPage, we iterated through the array of tuples of the page and added all the tuples into a new list, which was returned as the iterator. This is to remove all the empty slots that are not used between the used slots, so that it is easier to iterate through the array of tuples.

HeapFile

- In order to read or write data from disk, we needed to access the table which is stored in a HeapFile object. Each HeapFile object has multiple pages, of which consists of a header that contains a bitmap with one bit per tuple slot
- To implement the readPage() method, we had to use RandomAccessFile in order to find a specific page on the disk to read. As long as the offset pointer was within the index of the RandomAccessFile, we would seek, read and then close the RandomAccessFile to successfully read the page.
- We also created a private class HeapFileIterator which implements the DbFileIterator interface inside HeapFile.java to allow the Operator class to parse individual tuples.

SeqScan

- We implemented a new constructor in TupleDesc, so that it will be able to take in an array of TDItems as the input and create a TupleDesc. The new constructor allowed us to be able to implement the getTupleDesc method easily in SeqScan, such that we can create a new array of TDItems and return a TupleDesc with by calling the constructor on the array of TDItems

Missing or incomplete elements

We managed to successfully implement everything as required in Lab 1. Yay!