A simple writeup for LAB 1

I followed the sequence of exercises 1 to 6 and finished LAB1. Below is my thoughts on why and how I made some decisions on the structure in the LAB during my process of dealing with the LAB.

Exercise 1

TUPLE.JAVA

1. I referred to the internet and chose the appropriate container java.util.concurrent.CopyOnWriteArrayList. it's safe when reading the fields and writing the fields.
2. I added member td and rid, which records the schema of tuple and position of tuple at disk. And most importantly, I added the container fields.
3. I overrode the equals function to judge the equivalence of two Tuple objects. It will judge the type of the object and compare the two obj attr by attr to ensure that they are identical.

TUPLEDESC.JAVA

1. I instantiated a tupleDesc Item list with the container I used above. It stores a tupleDesc's items. Every item is made up of fieldname and fieldtype.
2. In **indexForFieldName()** function, I considered that when the two tuple is merged, there may be an alias before the field name. So I deleted the name before".".
3. In **hashCode(),** I made the decision to define the hashcode as the addition of every item's string hash. To avoid eviction, I multiplied it by my favorite number 87.

Exercise 2

CATALOG.JAVA

1. I created a tool class Table, with private member Dbfile, name, pkfield and construction function. This implements all the attributes that a table needs.
2. To be threadsafe, I used ConcurrentHashMap to create two hash maps, which are fileId 2 table and name 2 id. The reason to use hashmap is referred from the internet, which is a good way to save storage spaces and boost access efficiency.

Exercise 3

BUFFERPOOL.JAVA

1. The instance of buffPool is chosen as a ConcurrentHashMap, which can promise the one-on-one pageid to page reflection.
2. In **getPage()**, I used the API Database.getCatalog().getDatabaseFile to create new databasefile if pid is not in bufferpool but in storage.

Exercise 4

BUFFERPOOL.JAVA

1. **Hashcode()**is defined to left move 10bit plus table id.

HEAPPAGE.JAVA

RecordId.java

1. In **hashCode()**, notify that a tuple can be unique after three attributes are affirmed: tableid, pagenumber and tuplenumber.

Exercise 5

HeapFile.java

1. I wrote a inner private class HeapFileIterator using DbFileIterator API, which will be used in the **iterator()** below. The heapFileIterator is actually a tupleIterator that can iterate throu the tuples inside the HeapFile. The most important part in the class is tupleIterator, which I used the API HeapPage.iterator to implement it.
2. In **readPage()**, I usedrandomAccessFile to open the file more flexibly. In the function, I also checked whether the page is out of range of heapfile, whether the page has reached the end of file.

Exercise 6

SeqScan.java

1. Defined tableid, tableAlias and dbfileIterator member.

Missing or incomplete elements

1. I think my way to create hash may be inefficient or may have evicted with other hash code in some cases. But so far I haven't come up with a new method to create hash. Maybe I should define a brand new hash function.
2. In some cases, I may have forgotten to judge many null or memory overflow or memory leak error, which may make my simpleDB not safe in some extreme use.
3. In some functions, the time complexity or space complexity are not pondered enough, which may result in intolerable running time or frequent memory exchange.

Time spent on the LAB

I spent about 36 hours in total. The most hard part is that I am not familiar with Java language. I need to search on the internet about which type of containers to use or look through the files to find which API to implement. On the contrary, figuring out the configuration of the database is rather easier for me.