Jason Tieu
304047667

Lab 2 Writeup

- **Describe any design decisions you made, including your choice of page eviction policy. If you used something other than a nested-loops join, describe the tradeoffs of the algorithm you chose.**

  -For IntegerAggregator, I used HashMaps to store important values I needed for the operators. I had separate HashMaps that mapped the group-by field to a field with the aggregate value. I also had a HashMap that mapped group-by field to a private helper class called AVGinfo that stored ints for the current sum and count for when the average operator was used. I used a similar design with a HashMap for the StringAggregator.

  -For the Aggregate class, I had an Aggregator member that was instanced as a IntAggregator or StringAggregator depending on what the field type of the column we were computing was. I also had a DbIterator that was set to the Aggregator's iterator.

  -For the HeapPage class:

    - For deleteTuple(), I ran a loop that iterated through the tuple array to look for the one to be deleted. Once I found it, I set it's RecordId to null and marked it's header slot as unused. I then set that tuple in the tuple array to null.

    - For insertTuple(), I ran a loop to find the first slot that is not used. Once I found a slot, I added the tuple to the proper index in the tuple array and marked it's corresponding header slot as used. I also properly set the tuple's RecordId by creating a new RecordId consisting of the PageId and the tuple number (index of where in the tuple array);

    - For markDirty, I simply stored private data members: a boolean to see if the page is dirty or not, and a TransactionId to save the latest TransactionId that did the dirtying.

    - For isDirty(), I returned the TransactionId that I saved or null if the page hasn't been dirtied.

    -For markSlotUsed, I figured out the correct header byte and the appropriate bit index I had to set. Then according to whether or not the boolean was true or false, I either set or cleared the bit.

  - I decided to use a nested-loops join for fetchNext.

  -For the HeapFile class:

    - For writePage() I created a RandomAccessFile and sought the page with the correct offset. I then wrote to it by getting the page's data.

- For insertTuple(), I iterated over every page in the HeapFile and found the first one that had an empty slot. Then I inserted that tuple by calling HeapPage's insertTuple() method and made an ArrayList of the pages modified. If there was no empty page, I created an empty page and wrote it to disk with the writePage() function.

- For deleteTuple(), I used Database.getBufferPool() to get the page with the tuple and deleted said tuple from the found page. I then set the newly deleted tuple's RecordId to null to signify that it was no longer stored on any page.

- For the BufferPool class, I used similar design for the insertTuple and deleteTuple methods as I did for in HeapFile and HeapPage. The notable thing was I made a HashMap that mapped PageId to Page and inserted into the HashMap every time I called insertTuple. I used that HashMap in the flushPage() method by getting the correct Page using the PageId from the arguments. I would write the page to disk if it was dirty.

- For my eviction policy, I simply evicted the first page of a list of pages I stored in my BufferPool instance. Although it may not be efficient, I found that I helped me pass the systemtest. Also, I did this with the LRU replacement policy in mind. The first page of the list would be the oldest one to have been inserted, so I felt it was adequate to evict the oldest page.

- **Discuss and justify any changes you made to the API.**
  I did not make any changes to the API and stuck with the skeleton code that was provided.
- **Describe any missing or incomplete elements of your code.**
  I was able to pass the systemtest successfully. I do not think I left out any methods or classes that were required to be completed for this lab. One part that I felt was incomplete or possibly inefficient was my eviction policy. Even though I was able to successfully pass the EvictionTest system test, I decided to just always evict the first page in a list of pages that I stored.
- **Describe how long you spent on the lab, and whether there was anything you found particularly difficult or confusing.**
  I spent roughly 10 hours on this lab. I had trouble fully understanding how to implement methods such as fetchNext for operators such as Join, Delete, and Insert but was eventually able to figure it out. Also, I would not consider myself fully comfortable with Java yet, so I had to continually research on what I could and could not do. Those are of my own fault, but they made the lab tougher nonetheless.