Lab 2 Write-Up

Our major design decisions in this lab were centered around implementation choices or policies. Firstly, our eviction policy is random and works by using the keySet() function for concurrent hashmap bp_map to go over the different pages in the bufferpool. We try to remove the first page we access (flushes the page to disk, removes the key and associated value from bp_map and marks that a page has been evicted using boolean evicted). If we cannot remove it then we throw and catch an error but continue to try for consecutive pages. If we cannot remove any page (i.e evicted is false) then we throw an error. For the joins, we used simple nested loops because we understood their implementation a bit more naturally. We start from the first tuple on the left and go through each tuple on the right looking for required field matches. If we have a match then JoinTuples() concatenates them and the right iterator moves to the next tuple. We run until we have completed the inner and outer loops. We also decided to stick with the implementation of extending operator class instead of using OpIterator because it was given as such and thus easier to implement.

We have not made any API changes. However, we have corrected the two issues with our lab 1 code. We added the condition to check if typeAr.length == fieldAr.length in the constructor of TupleDesc and if typeAr.length>=1. We also had a failure in HeapFileReadTest which we debugged and fixed by adding a needed condition (curr_pgNo > -2) in the readNext() function of our HeapFileIterator.

Some of the issues that we fixed are the following. We changed our implementation of numPages from lab1 so that instead of calculating numPages in the constructor of HeapFile, we moved it to the function numPages() so that it would be updated properly. Moreover, error handling in general was a bit tricky because sometimes the conditions under which a function would fail were not obvious to us, and it required some time to figure out if the errors were already being dealt with in the other functions we were calling within the function. We also had to spend some time debugging a NoSuchElement error when we tried to run a simple select query in 3.7, and we realized we needed to fix how we were checking string equalities. Finally, we also had a random value printed onto the console, but we found and debugged the println() that was the root of the issue.

There are some issues that we faced while testing out other queries in 3.7 that we did not get to fix. We tried an aggregate function, such as “select sum(data.f2) from data group by data.f1”, and the correct data was printed on the console. However, the query plan that is printed before the data throws an InvocationTargetException in Parser.java, which is caused by NullPointerException at upBarPosition inside QueryPlanVisualizer.java. Based on the PDF, we should not run into any errors with this function, which has one aggregate column and one group by. But because this query plan visualization is beyond our scope of the project, we decided to leave it at that. Also, we had an issue with deleting (for example, delete from data where f1=5;). The correct number of tuples were being deleted (so in this case, 2), but the problem is that the ones that follow that tuple are not being read or printed. We suspected that this was due to making the deleted tuple null inside HeapPage.deleteTuple() which made the subsequent tupleIterator to assume that this null is the end of what it should read. So we tried commenting out that line, but this led to just erasing the last tuple every time regardless of the where condition. We created a strdata.txt and strcatalog.txt to see if everything works fine on String-type data, but the same errors were

found. Other than these two errors, we were able to see expected results for String-type data as well.

For this lab, we had much more familiarity with Java, the given code, and how simpledb is interconnected, so we were able to perform our tasks much more independently and efficiently, taking a total of about 60 hours. The code passes all the unit and system tests, and gives expected outputs in 3.6. Overall, the process was much smoother than for lab1.