Minh Nguyen

Programming Assignment 3 Report

Simple DB

Professor George Kollios

I. Design decisions: you said to be a little bit more detailed. I'm not sure how detailed I should get, so it might be a little bit too detailed.

Exercise 1:

- Predicate.java and JoinPredicata.java are just skeleton codes. For the filter function, I get the field of the tuple and compare it with the other's field by calling the compare method.
- Filter.java is also skeleton code, but a bit different with fetchNext(). For fetchNext(), I iterate through the tuples, pass them in the filter function and call them on the Predicate.
- Join.java and HasEquiJoin.java. So for both classes, I use the exact same method, for merging the Tuples. I just simply create a new Tuple with the size of 2 merged tuples combined, loop through all the tuples in each and copy them into the new Tuple.

Exercise 2:

- IntegerAggregator.java. I wrote a support method, for mergeTupleIntoGroup, to check the aggregation operator and perform corresponding action with the field. The iterator() method I handled NO_GROUPING separately with the rest. I also created a separate arraylist for operation COUNT, each element has a value of 1.
- StringAggregator.java. StringAggregator is simpler than IntegerAggregator since it only accepts COUNT operation. Thus, for this class I did not need a support method like in IntegerAgg. The principles for iterator and mergeTupleIntoGroup are the same.
- Aggregate.java. I created an iterator to iterate through aggregations. For open(), I handle NO_GROUPING separately, and then I handle String and Integer Aggregator separately.

Exercise 3:

- HeapPage.java. I created a boolean variable for marking dirty and a TransactionId variable for dirty transactions. I use these 2 to complete the markdirty and isdirty method first. Then I completed markSlotUsed. Then I proceed to complete deleteTuple and insertTuple. For deleteTuple, I simply mark the slot of the deleting tuple used and erase it by putting it null in the tuple list. For insert I loop through to check if there is any empty slot, if not create a new one and insert the tuple in.

- HeapFile.java. In the process of doing insertTuple, I realized that I need writePage method, not necessarily need it, but I will use the same process, so I would better just write it and use it. With the same principles as I did for HeapPage, I loop through all the pages, if there is an empty slot, insert the tuple, else create a new one.

Exercise 4:

- Insert.java. fetchNext() is straightforward, iterate through and insert. I followed the instructions and was able to do this in a short amount of time.
- Delete.java. Similar to insert.java. Iterate through and delete, return number of deleted records.

II. Changes to the API:

I added some private variables and hashmaps that support my codes. Other than that, no changes were made.

III. Imcompletements:

None

IV. Feedback:

This programming assignment is straightforward. I spent around 15 hours on it (maybe I could have finished it in a shorter amount of time if I was fully focused). Most of my bugs are in the loops like index out of bounds, null elements, or it never reached the loops, etc... The skeletons' codes have similar principles. For HeapPage, those methods that we implemented in PA1 were very useful, and it did not take me long to revise the Byte and bit to use for markSlotUsed().

I spent the most amount of time on the Aggregators classes, around 9 hours (distracted), with the iterator() functions.

After programming assignment 2, programming assignment 3 is such a breather. Personally, I was having such a stressful week, which makes it hard for me to fully focus. I feel very lucky because this programming assignment was not as hard as the previous one. Overall, this assignment, similar to the previous ones, is very useful. It helps me fully understand how the joins and the aggregation operations work in detail.