Khai Tran

SID: 1728872

CSE 444

This lab2 has three important parts such as Filter and Join (Predicate.java, JoinPredicate.java, Filter.java, Join.java), Aggregates (IntegerAggregator.java, StringAggregator.java, Aggregate.java), and HeapFile Mutability (HeapPage.java, HeapFile.java, Insert.java, Delete.java, and BufferPool.java). I implemented the Operators such as Filter, Join, Aggregate, Insert and Delete which are abstract Operator classes implement OpIterator which means iterating over the tuples, accordingly each other based define how they generate the tuples to return. The operators have one or two child OpIterator (s) which they can generate the tuples to return.

The Filter operator implements a selection which has a child and a Predicate using to filtrate and returns the tuples correctly.

We implements the join operator using a JoinPreciate has child0 and child1, when fetchNext() in Join was called the join operator will find the pairs of tuples t0-child0 and t1 -child1 and return the join of t0 and t1 by using a nested loops join. The target of nested loop join which find all joined tuples and add a list. The list is built in memory of hashtable carrying the tuples of child0, and then iterates over the tuples in child1 which match in the hashtable. I use a ArrayList<Tuple> for the hashtable. The key is the join field is all the tuples of join value. The simple nested loops join generates the joined tuple list by iterating all tuples t0 in child0 and the same for all tuples t1 in child1 by checking if t0 and t1 match to the joined list. Now, we have all matching tuples in a joined list which using this list of iterator to generate tuples.

The Aggregate operator implements aggregation with a single child of OpIterator and an Aggregator.Op which keep track of the aggregate value was computed merged tuples. Aggregator will construct an IntegerAggregator or a StringAggregator according to the type of data aggregating, and merge every tuple in the child, into the constructed aggregator, also give an iterator and returns a OpIterator to return the aggregate results. The Aggregator iterator to get the tuples to be returned. The important of implementation IntegerAggregator and StringAggregator is a hashtable to a group and an aggregate value. Aggregator with is no-grouping which hard to use a map has null keys. Alot of aggregate values are a single value such as MAX, but AVG has two values a count and a sum, therefore it is not easy to store an int for all aggregate types. I used a pair of Type ints to handle the problems. It will be easy to store with Aggregate values that just need either one or two value to be represented in the pair. With no groupings I added a field with a single aggregate value and the case of groups using a map from group to aggregate value.

The next two operators are Insert and Delete and to implement those above we had to work with HeapPage, HeapFile, BufferPool, Insert and Delete. In HeapPage inserting and deleting needed to update the header to change the state of the deleted or inserted tuple, and HeapFile implemented deleteTuple and insertTuple which taking a tuple insert or delete it and return a list of all pages that was dirtied. A dirtied page just makes its address different in memory and on

disk. Inserts and Deletes have to go through BufferPool, therefore pages can be added to the cache faster. We had to implement insertTuple and deleteTuple in BufferPool with access the correct DbFile to return back a list of pages affected pages to the cache.

I did not change anything to the API.

I think I do not have any missing element of my code. I believe that my code is correct.