Writeup for lab3 **Yuwei Wu**

CS392 Database Management System ACM Class, Zhiyuan College, SJTU

Prof. **Feifei Li** Due Date: May 23, 2019

Submit Date: May 23, 2019

Design decisions about lab3

- When implementing the *Join* operator, I just write two nested loops. The first loop is to iterate through the second child and every time a matching tuple is found, it will be returned as next tuple iterated. And when there is no next element in the second child, just iterate into next tuple of the first child and rewind the second child to do the iteration again util there is no next element in the first child.
- When implementing the *HashEquiJoin* operator, I first build up a map every time the operator is open. Specifically, a field-to-tuple-array map is built through which I can get an array of tuples of the first child with equal field value. And the iteration is to iterate through the child 2. Every time I fetch an element of child 2, I then iterate through the array of tuples of child 1 with equal field value and return the merged tuple.
- For the Aggregator part, I just keep in record a field-to-aggregated-value map. Tuples to be merged into group are aggregated if they have the same group-by field value. And this map is enough to keep record for these aggregated value. To compute the Avg operator, a count operator is needed.
- The other parts are quite straight-forward.

API changes

- Add static method **Tuple joinTuples(Tuple tuple1, Tuple tuple2, TupleDesc tupleDesc)** to *Tuple* class. And in that way, to join two tuple *tuple1*, *tuple2* with merged tupleDesc, I just simply call the method **Tuple.joinTuple(tuple1, tuple2, mergedTupleDesc)**.
- Add method **TupleIterator getTupleIter(TupleDesc desc)** to *Aggregator* class. To obtain a tupleIterator that has the required TupleDesc in *Aggregate* class, just call **Aggregator.iterator()** is not enough, as *Aggregator* does know the aggregateFieldName. So I just add an additional method in the *Aggregator* class to return a tuple iterator with given tupleDesc.

Missing or incomplete elements

- I didn't do any of the query optimization part.

Time spent and difficulties

- I spent two whole days working on lab3.
- I found the part that needs to implement *HashEquiJoin* operator difficult as it costs me some time to figure out what is the difference between nested join operation and hash join operation.
- And I meet bug in the part of previous labs. Problems will occur in **HeapPage.iterator()** if deletion and iteration are done at the same time. I always call the **getNumEmptySlots()** to calculate how many slots left to read in page iteration, but the number will change every time a delation is done, which results in part of tuples in a page has no chance to be found. So I just store the initial number of empty slots in constructor to ensure the number will not be changed and iteration goes through all the tuples.