Lab 3 Writeup
Fei Peng, 404406320 && Shijian Zheng, 804514110

***We finish Lab 3 in pair and each of us did three exercises. One grace day is consumed for both of us.**

Design:

Exercise 1:

Step 3: simpledb.Parser.handleQueryStatement()

First, get the logical plan of the query lp by invoking parseQueryLogicalPlan(tId, s).
Second, get the physical plan by invoking lp.physicalPlan(tId, TableStats.getStatsMap(), explain).
Then set the logical and physical plan to the query.

Step 4: simpledb.Parser.parseQueryLogicalPlan()

Parse the different clauses of the query into a logical plan.

Step 5: simpledb.logicalPlan.physicalPlan()

build the physical plan using the following step:
1. Put table alias and iterator of a transaction into subplan map.
2. Set up statsMap and filterSelectivities for tables to store the statistic information and selectivity of tables.
3. Put filter into the subplan map.
4. Join two tables in subplan using the best order and get the iterator.
5. Walk through the select list, to determine order in which to project output fields.

Step 6: simpledb.Query.execute()

Print out the result of the query.

Exercise 2:

1. Calculate the **width** of bucket and create a array **bucklist** to represent the buckets.
2. When add value, find the right bucket and plus one on that bucket.
3. Calculate selectivity of various operators respectively base on a specific value in estimateSelectivity() method.

Exercise 3:

1. I use 2 hash map, **intMap** and **StrMap** to represent histograms for int fields and string histograms for string fields.
2. In the constructor, I build the string histogram first because it doesn't need the min and max value. For int histogram, I use a minMap and maxMap to store the min and max value for each int field. I iterator the table 2 times, the first one is to find the min and max value of each int field, the second one build the int histogram and fill in the values.

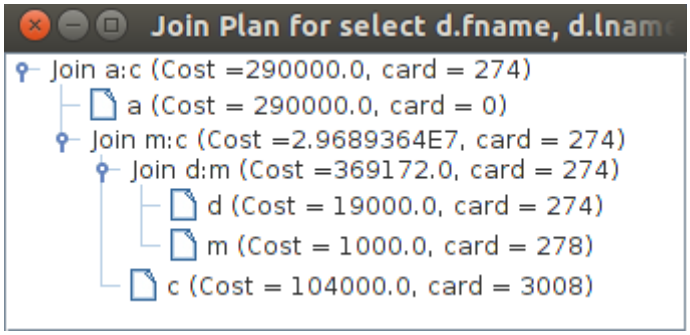
Exercise 4:

1. Calculate the join cost base on formular joincost(t1 join t2)= scancost(t1)+scancost(t1)*scancostt2+ntups(t1)+ntups(t2)
2. The equality join is implement base on the description, for equality join, when one of the attributes is a primary key, the number of tuples produced by the join cannot be larger than the cardinality of the non-primary key attribute, when no primary key or both primary key, the number of tuple produced should less or equal to the larger cardinality. For range scan, the size of the out put is 0.3 of the cross-product.

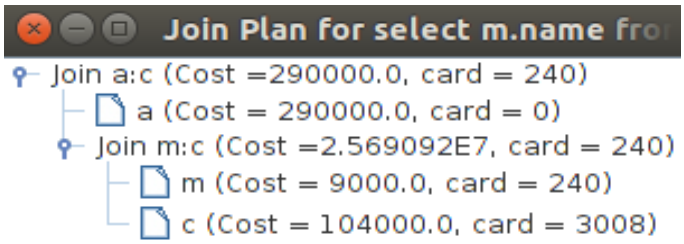
Exercise 5:

1. This method is implement base on the given pseudo code, the basic idea is using DP to get the best order for every length, which is stored in the plan cache.

Exercise 6:
1. With 0.1% datasets, the resulted order is a join ((d join m) join c) , a like the below figure.



2. I test with the query “select m.name from Actor a, Casts c, Movie m where a.id=c.pid and c.mid=m.id and a. fname='John' and a.lname='Spicer';”, which select all the movies John Spicer played a role in. The result is like following, a join (m join c)is the best plan with cost 290000 and no other permutations can have better cost, a is very likely in the left most place of a join plan since it has 0 cardinality, which cause the term card1*cost2 equals to 0;



API Changes
Nothing.

Difficulties:
1. In the IntHistogram class, it took me some time to calculate the correct bonder for GREATER_THAN_OR_EQ and LESS_THAN_OR_EQ.
2. I use arraylist to store tables in catalog, which turns out not good option. The test case add table twice (the first one is the time heapfile created, with a random number as table name), that cause two tables with same table id but different table name stored in the catalog. I spend a long time to figure it out.