Yuan Wei

BUID: U52001624

- Describe any design decisions you made, including methods for selectivity estimation, join ordering.
- Discuss and justify any changes you made to the API.
- Describe any missing or incomplete elements of your code.
- Describe how long you spent on the assignment, and whether there was anything you found particularly difficult or confusing.

Exercise 1

for the parameter, beside the parameter in the constructor, # of tuples and a interval has also be declared for a convinience compute on index and following estimatiom. an int array named buckets was also used to record the # of tuples in each interval.

we need to get the index for estimated value,

```
private int getIndexByValue(int v) {
         return v == max ? this.buckets.length - 1 : (int)((v - this.min) /
this.interval);
    }
```

as we conclued in lab instruction, there are 6 cases to estimate. However, these case can be take as eqution denoted by EQUALS and LESS_THAN

```
case LESS_THAN_OR_EQ:
    return estimateSelectivity(Predicate.Op.LESS_THAN, v + 1);
case GREATER_THAN:
    return 1.0 - estimateSelectivity(Predicate.Op.LESS_THAN, v + 1);
case GREATER_THAN_OR_EQ:
    return 1.0 - estimateSelectivity(Predicate.Op.LESS_THAN, v);
case NOT_EQUALS:
    return 1.0 - estimateSelectivity(Predicate.Op.EQUALS, v);
```

It took me 3 hours to fully understande and implemented the logic of hitstgram estimation

Exercise 2

For the Selectivity Esitimation, we have to differience 2 cases : IntHistgram and StringHistgram which was implemented in previous;

The estimated Cardinality is

$$numOfBaseTuples \times selectivityFactor$$

the scan cost is

$$numOfBasePages \times costPerPageIO$$

It takes me 1 hour to understand and implement.

Exercise 3 & Exercise 4

for the implementation of estimateJoinCost, we estimate the cost of given left input, right input, and their cost of scan. I applied a Nested Loop Join here so the estimated cost is:

$$cost1 + card1 \times cost2 + card1 \times card2$$

for table Join Cardinality estimation

based on the principle provided:

> - For equality joins, 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.
> - For equality joins when there is no primary key, it's hard to say much about what the size of the output is -- it could be the size of the product of the cardinalities of the tables (if both tables have the same value for all tuples) -- or it could be 0. It's fine to make up a simple heuristic (say, the size of the larger of the two tables).
> - For range scans, it is similarly hard to say anything accurate about sizes. The size of the output should be proportional to the sizes of the inputs. It is fine to assume that a fixed fraction of the cross-product is emitted by range scans (say, 30%). In general, the cost of a range join should be larger than the cost of a non-primary key equality join of two tables of the same size.

so we consider these conditions

```
if NOT RangeScan
  if(hasT1Key)
    return card2
  if(hast2Key)
    return card1
  else
    return max(card1,card2)
else
  return 0.3 * card1 * card2;
```

The intuitive think is we simplly use the nested loop to find the minimum cost of the Joint. However, in the code it provided a effiecient way using a PlanCache. this method is called

For exercise 4, it provided the method computeCostAndCardOfSubplan which works with PlanCache for savuing the subsets of join in Selinger Algorithm. So we just iterate the LogicalNode in nodeSet and maintain a minimum cost.

to finish this code, i spent 2 days on it. since i found my code TLE. Then i tried to debug them, suprisely found the implementation on my Join.java in LAB3 has some bug.