PA 4 WriteUp - Abdelazim Lokma

For this Programming Assignment, I worked with Ming Hill, implementing all functions collaboratively. We worked together in person, completing the coding tasks side by side on either my laptop or his. At the end of each session, the person with the most up to date codebase would push their changes to our shared GitHub Repository. This approach allowed both of us to access and review our work independently without relying on the other person's device.

Estimate Selectivity:

The goal of the `IntHistogram::estimateSelectivity` function is to determine the selectivity of predicates using a histogram-based approach. We felt that implementing a switch statement to handle all the different predicate types help keep the code concise. We first check if the value `v` falls outside the range of the histogram. If so, we return 0 because the value is not within the histogram's range. Selectivity is determined by calculating the count of values in the target bucket and the additional contributions from previous (less than / less than or =) or later buckets (greater than / greater than or =), depending on the predicate type.

Estimate Join Cost:

This function was very easy to implement, it is designed to estimate the cost associated with performing a join operation between two relations. We calculate the cost of reading data from the second relation for each tuple in the first relation. After which we calculated the CPU cost as the product of the cardinalities of both relations (`card1` and `card2`). finally we returned the join cost, which is the product of the cost of reading the first relation + the cost of reading the second relation for each tuple in the first relation + the cpu cost of the join operation.

```
joincost(t1 join t2) = scancost(t1) + ntups(t1) x scancost(t2) // IO cost
+ ntups(t1) x ntups(t2)  // CPU cost
```

Estimating Join Cardinality:

For this function, we kept its implementation very simple. In the case of an equality join with primary keys, we return the cardinality of the smaller table. As each row in the

smaller table can match at most one row in the larger table. If only one of the tables has a primary key, we return the cardinality of the second table, as each row in the first table uniquely identifies a corresponding row in the second table. For non-equality joins, we simply estimate the join cardinality as 30% of the product of the cardinalities of the two tables. This simple heuristic is made on the assumption that the join condition will reduce the resulting table size by a large amount.