Ming Hill
CS 660
Professor Athanassoulis
9 Dec 2023

PA4 Write Up

Worked with Abdelazim

For most of the project, we did not have to make any major design decisions, given that the instructions were pretty straightforward. That being said, it was only the estimateSelectivity method that we made a pretty significant design decision on. A lot of the calculations that we computed for this method were given to us in the README, and we just had to implement them ourselves. Initially, we want to check if the totalValue in our IntHistogram class has less than one value. In this case, it would cause issues to our program when dividing a number by 0, therefore, we just returned 0. Following that, we had to make sure that the value that we were estimating fell between the minimum and maximum values of the histogram. This is essentially 2 edge cases that we had to keep in mind. To find the estimation, we first had to assign some variables, including the index, which was the bucket within the histogram that we were choosing. The width signifies the range of values within each bucket. Finally, we created a bucketStart and bucketEnd variable, which gives us the start and end of the indexed bucket. All of these calculations were given to us in the README. Finally, we used case checking to see which Operation was needed to give the correct estimation of selectivity. For the JoinOptimizer class, once again, everything was pretty straightforward and was given to us in the README. To estimate the join cost, we used the formula given:

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

We just replaced the scancost and ntups with card1, card2 and cost1 and cost2.
For estimating JoinCardinatiy, we understand that there are different cases depending on the presence of primary keys, and the type of operation that was performed (ranged query vs equality qeurry). Therefore, if the operation is a range query, we used 0.3 which is an estimation of the cross-product, and for equality searches, we first check for the presence of primary keys, and depending on that, we return the respective card. If they both have primary keys, then we return the minimum of the two, and if not, we return the one without the primary key.

We did not make any changes to the API and we included all of the elements in our code.

For this assignment, it was much easier compared to the 2 previous assignments that we had. We felt like the README that was given as the instruction was very helpful in understanding the functionality of the methods we were assigned to implement and gave us a strong foundation to begin our coding. We probably spent a total of 6-8 hours on this entire assignment. Everything for

this assignment was pretty straightforward, however, the only issue we really ran into was the sparse case when creating the IntHistogram.

I collaborated with Abdelazim for this assignment. Since this project was not very intense, we pretty much worked on the entire assignment together.