1. Experimental Procedure

Exercise 1: IntHistogram.java

The core method in IntHistogram is `estimateSelectivity()`. Selectivity is defined as the proportion of tuples in the result set, after applying a predicate to a table, relative to the original number of tuples in the table.

Exercise 2: TableStats.java

`TableStats`: The `process` function is used to calculate the number of tuples in a table, determine the maximum and minimum values for each integer column, and compute a histogram for each column.

Exercise 3: Join Cost Estimation

The join cost is composed of `scan(t1) + card(t1)*scan(t2) + card(t1)*card(t2)`. The first two components are IO costs, and the third component is the computational cost of looping.

Exercise 4: Join Ordering

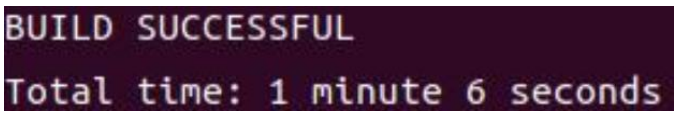
For Exercise 4, the task is to generate an optimized order of joins. Based on the cost calculation formula provided above, different orders of joins have different costs. `orderJoins()` returns the optimal join order based on given statistics for each table and the selectivity of each table.

Bonus Exercises

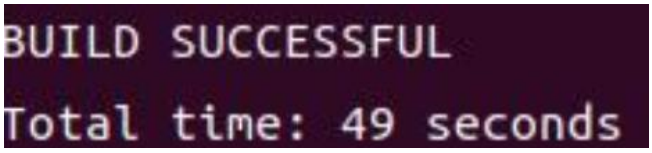
Mainly optimize `enumerateSubsets()` to enhance speed, which relies on another function `getSubsetIndex()`. Noting that Join can query subsets of different sizes, `getSubsetIndex()` uses bitwise operations to precompute all indices for subsets of size s<n. Here, two nested loops are used: the outer loop ranges from 1 to (1<<n) - 1, enumerating all 2^n-1 non-empty subsets; the inner loop ranges from 0 to n-1, enumerating the indices of the selected elements. Results are stored in a list, thus only calculated once, which speeds up computation.

2.Run Result:

Ant test



Ant systemtest:



3.Time consumed:

This task is really difficult for me. It used up all my Labor Day Holiday. There is much knowledge that I haven't learned before in this test, so I have to learn first.