# Database System Implementation

## Project 4 -1 – Statistical Estimation for Query Optimizer

Developed by –

Ghodkari Chowdary, Raghunatha Rao - UFID: 6218-1051
Mullapudi, Aseesh - UFID: 9175-1971

Aim of this project is to implement a Statistical Estimation which is useful for Query Optimizer in our main Database Management System.

We have implemented the Statistical Estimation by completing the required functions. Completed code is present in Statistics.cc and Statistics.h, has been tested and results have been summarized.

# Implementation Details:

## Statistics file format (Statistics.txt):

The format of the Statistics.txt is as below:

<number of relations after CNF is applied>
<RelationName>#<Estimated cost after performing join or select operation>
<number of attributes with their relation>
<RelationName> <AttributeName> <DistinctTuplecount>

E.g:
1 (noof relations after join performed)
lineitem_part#21433 (join is appended by '_' 21433 is estimated cost)
5 (noof attributes after join performed)
lineitem_part p_partkey 200000(joinrelation and attribute and distinct tuple count)
lineitem_part p_container 40
lineitem_part l_partkey 200000
lineitem_part l_shipinstruct 4
lineitem_part l_shipmode 7

## Statistics Class Functions:

1) ***void AddRel (char *relName, int numTuples)***

   This function adds another base relation into the structure. The parameter set tells the statistics object what the name and size of the new relation is. The size is given in terms of the number of tuples. We are maintaining this information in a hash structure which is helpful for faster lookups.

2) ***void AddAtt (char *relName, char *attName, int numDistincts)***

This function adds an attribute to one of the base relations in the structure. The parameter set tells the Statistics object what the name of the attribute is, what relation the attribute is attached to, and the number of distinct values that the relation has for the attribute. If the numDistincts is initially passed in as a -1, then the number of distinct is assumed to be equal to the number of tuples in the associated relation. We are maintaining this information in hash structure which is helpful for faster lookups.

**3)** ***void CopyRel (char *oldName, char *newname)***

This function produces a copy of the relation including all its attributes and all its statistics and stores it under the new name. This is useful to clone a relation and its attributes.

**4)** ***void Read (char *fromWhere)***

This function allows the Statistics file object to read itself back from a text file Statistics.txt.

***5) void Write (char *fromWhere)***

This function allows the Statistics file object to write itself to a text file Statistics.txt.

**6)** ***void Apply (struct AndList *parseTree, char **relNames, int numToJoin)***

This function calculates the estimated number of tuples required to compute a CNF (join or select operations) and state of the statistics object is updated with the estimation which help the query optimizer in later stage to choose the optimal way to implement the query. Since this is function is same as the estimate function, we are internally calling the estimate function and the apply operation is controlled by Boolean flag in the estimate function which applies the estimated values to relation and updates the hash structures.

**7)** ***int Estimate (struct AndList *parseTree, char **relNames, int numToJoin)***

This function computes the number of tuples that would result in a join or select operations over the relations in the hash structures and returns an integer to the caller function. This function does not allow change to the current

statistics object unless this function is called from the Apply which is being controlled by a Boolean flag.

## 8) gtest.cc:

This file implements google tests for 4 testcases using the functions Estimate, Read, Write and Apply.

**Instructions to Run the Code:**

Ensure the directory contains the heap files generated from the Project1 or sorted files from Project2 so that code has access to Bin files, or you can even use the a2test.cc file to compile it and run to generate the sorted bin files.

**If you want to generate sorted files in this same directory**:

Steps:

1) make a2test.out
2) ./a2test.out

Follow the options to generate the sorted bin files.

**To run the overall project, follow the below steps:**

**Compile and run test.cc:**

Ensure there is Catalog file in the root folder as well

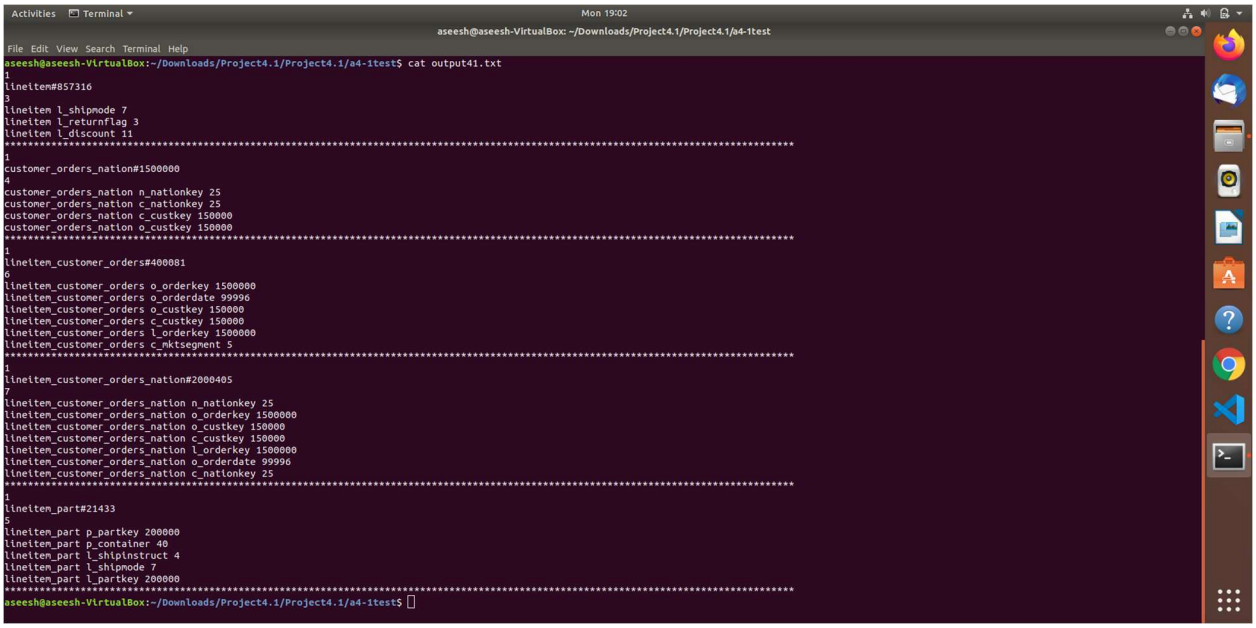
1. make
2. ./a4-1.out [0-11]

**To run the test cases**

1. make
2. ./runTestCases.sh

File output41.txt is generated

The screenshot of the output41.txt is below

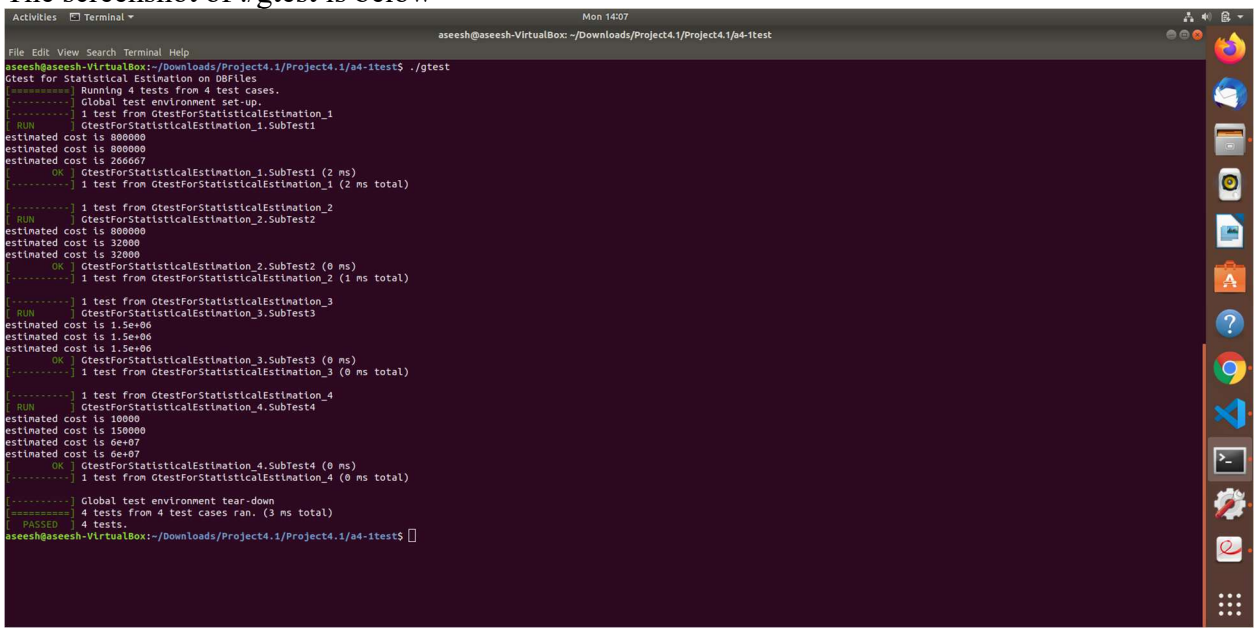



## Instructions to run gtests:

**Compile and run code gtest.cc:**

1. make gtest
2. ./gtest

The screenshot of ./gtest is below




Some Bugs Identified:

- Attributes in test case q3 and q11 are mistyped
- In q4, wrong relation names were passed in the arguments, corresponding changes done
- Attribute o_orderdate is added to q5, q10 and l_receiptdate added to q7