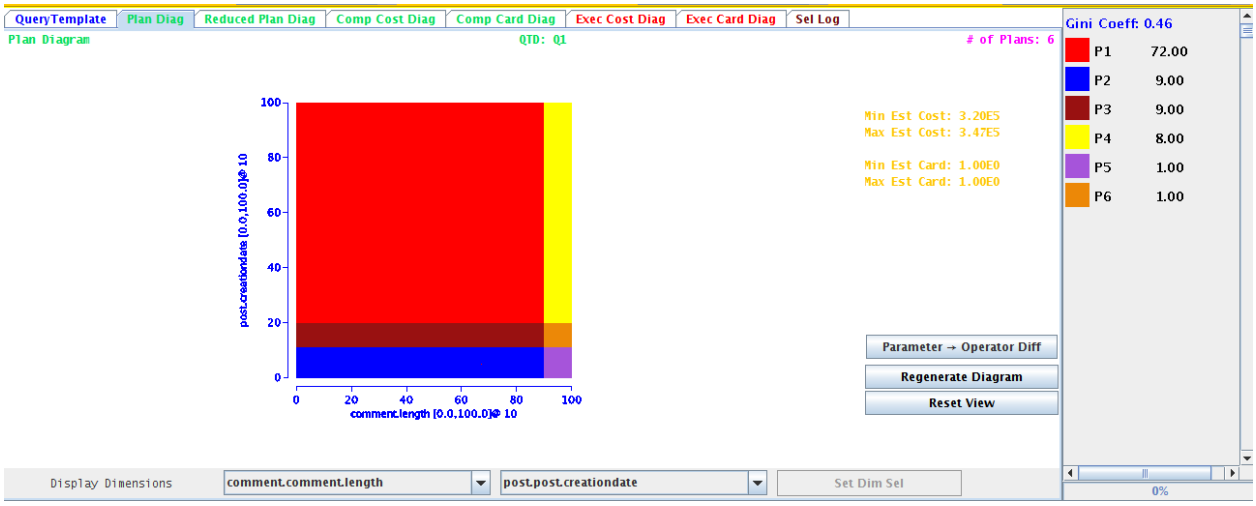
COL362 - Assignment 2

Harshit Mawandia - 2020CS10348 Kartik Sharma - 2020CS10351

In Picasso, the plan diagram is a visual representation of the different execution plans that PostgreSQL could use to generate the output for a given query. Each plan corresponds to a specific combination of query optimization techniques that are used by PostgreSQL. To explore and compare these different plans, Picasso generates a cost diagram and a cardinality diagram that show estimated execution costs and result cardinalities for each plan. We intend to use this query visualization software to optimize our queries further!

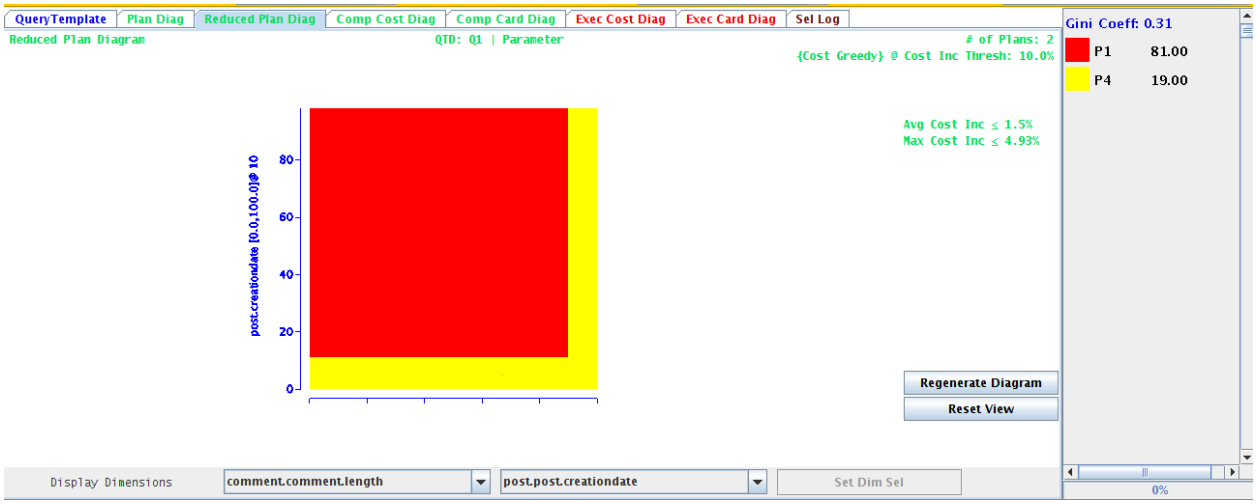
Query 1.

a) Exact Plan Diagram:



The plan diagram is a visual representation of the different execution plans that PostgreSQL could use to generate the output for a given query. It shows the different options available for optimizing the query and provides a starting point for evaluating and comparing these options.

Reduced Plan Diagram:

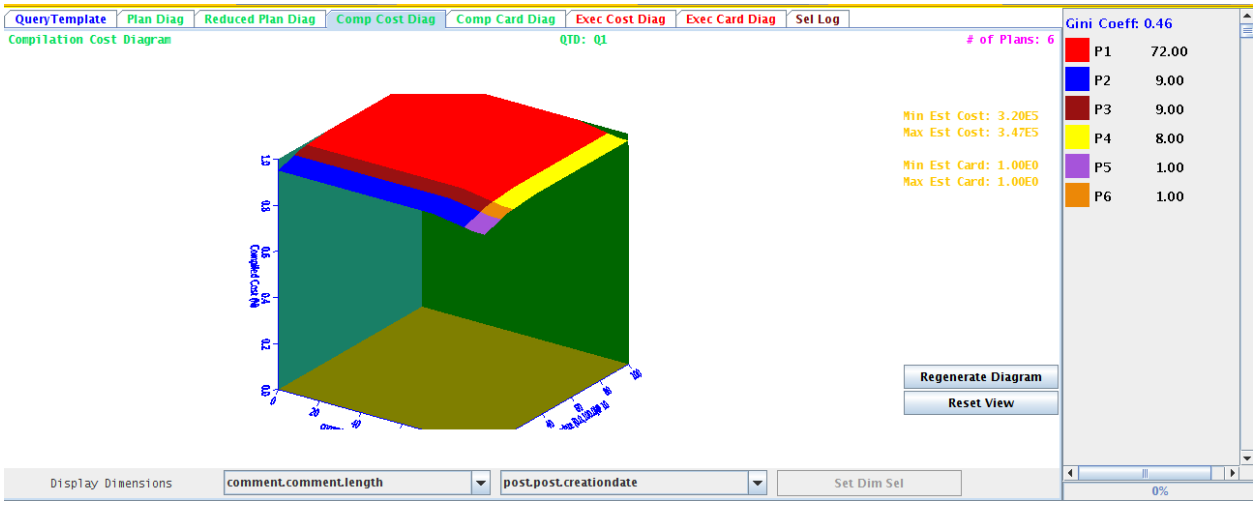


Exact plan diagram for the query shows the various plans that Picasso chooses to execute depending on the different values of the variables.

We see that there are 6 different plans for the execution of query.

1. for CreationDate = [20,100] and commentlength = [0,90]
2. for CreationDate = [0,10] and commentlength = [0,90]
3. for CreationDate = [10,20] and commentlength = [0,90]
4. for CreationDate = [20,100] and commentlength = [90,100]
5. for CreationDate = [0,10] and commentlength = [90,100]
6. for CreationDate = [10,20] and commentlength = [90,100]

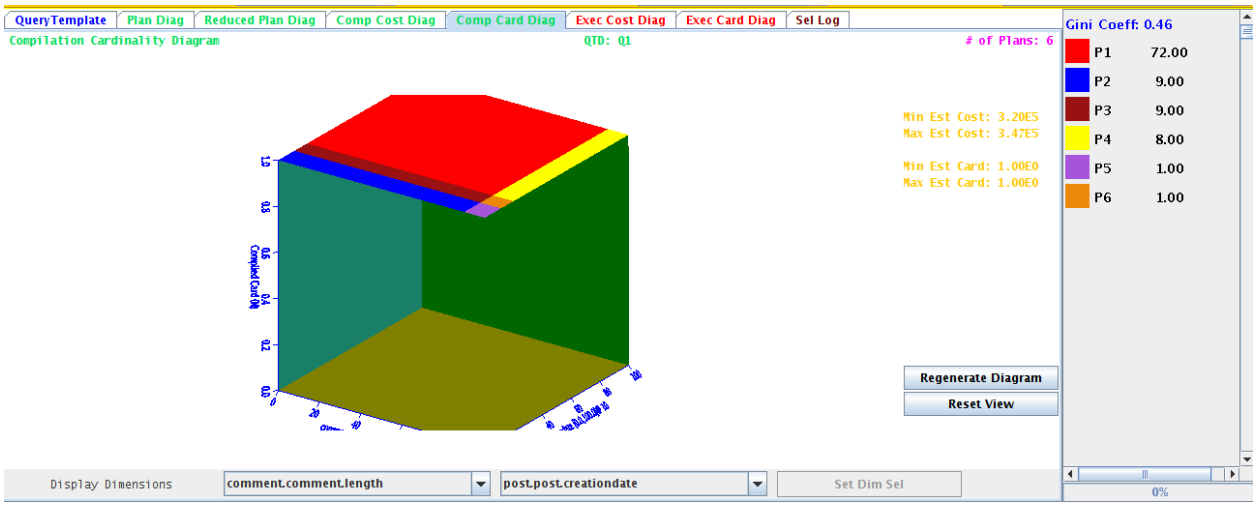
b) Compiled Cost Diagram



This diagram shows compilation cost for each of the plans over a range of values of variables. The cost diagram is a graphical representation of the estimated costs associated with each execution plan option in the plan diagram. It helps users evaluate and compare the performance

of different query optimization strategies in terms of their associated costs, allowing them to make informed decisions about which plan to choose.

c) Cardinality Diagram

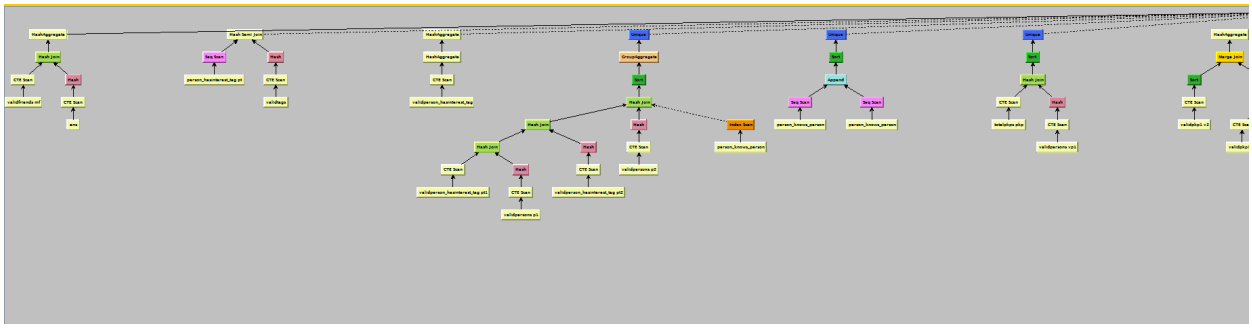


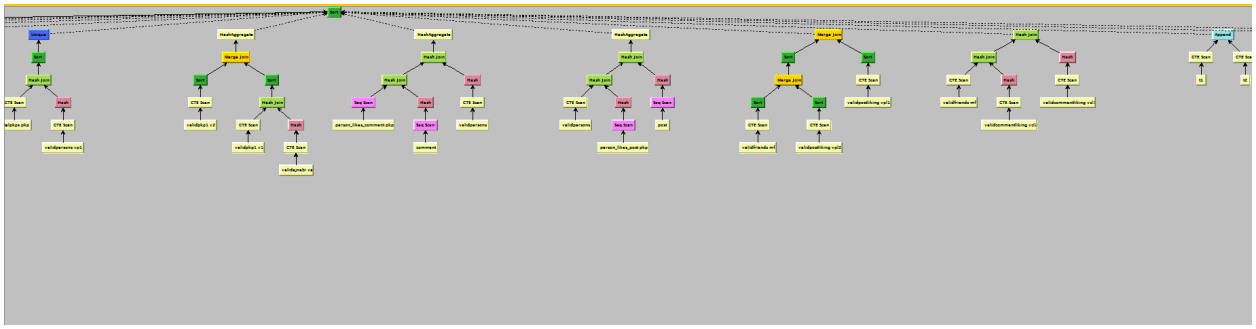
For all 6 plans of execution the cardinality remains the same for all values of the variables. The cardinality diagram is a visual representation of the expected number of rows or records returned by each execution plan option in the plan diagram. It helps users evaluate and compare the performance of different query optimization strategies in terms of the amount of data they will return, allowing them to make informed decisions about which plan to choose.

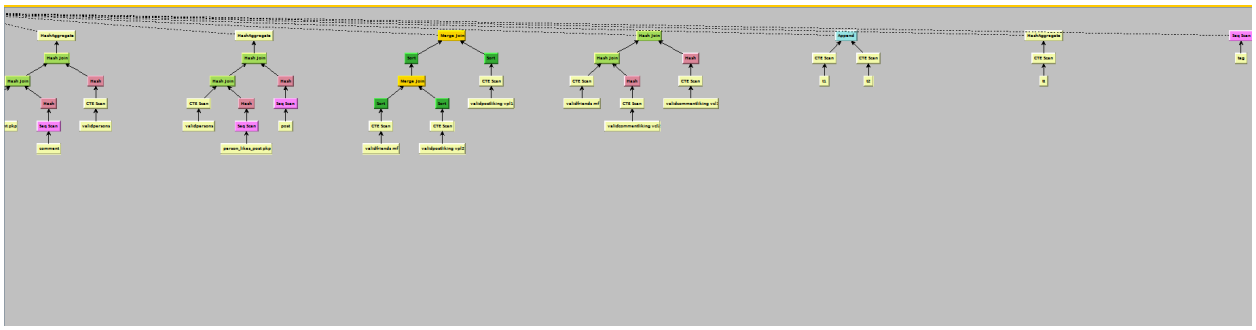
d) Plan Trees

The schematic plan-tree diagram is a tree diagram that illustrates the structure of a selected execution plan option from the plan diagram, detailing the sequence of steps involved in executing the query or set of queries. It helps users understand the internal workings of different query optimization strategies and identify areas for further optimization by varying attributes of the query

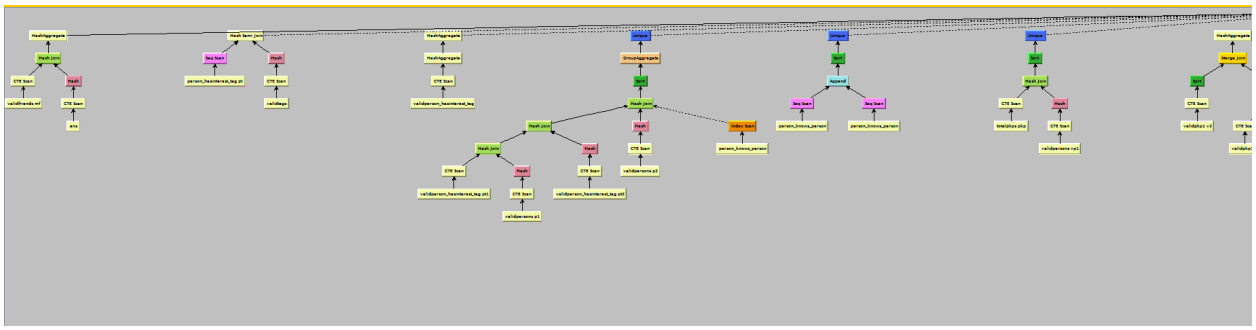
Plan 1:

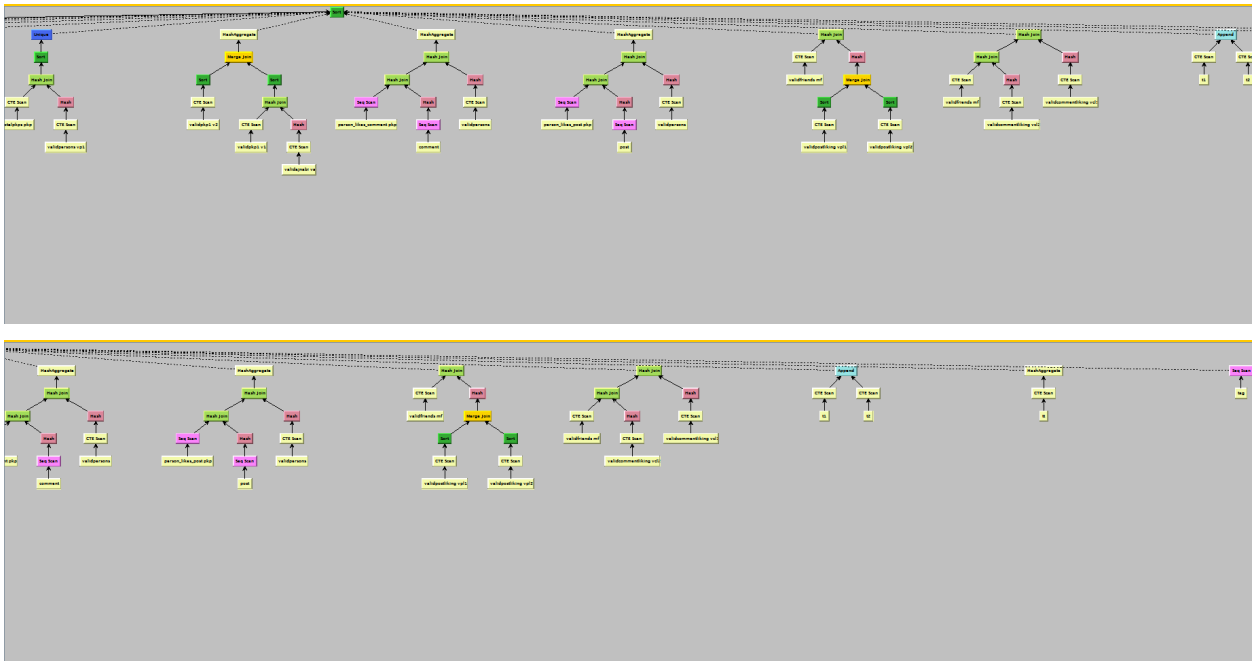




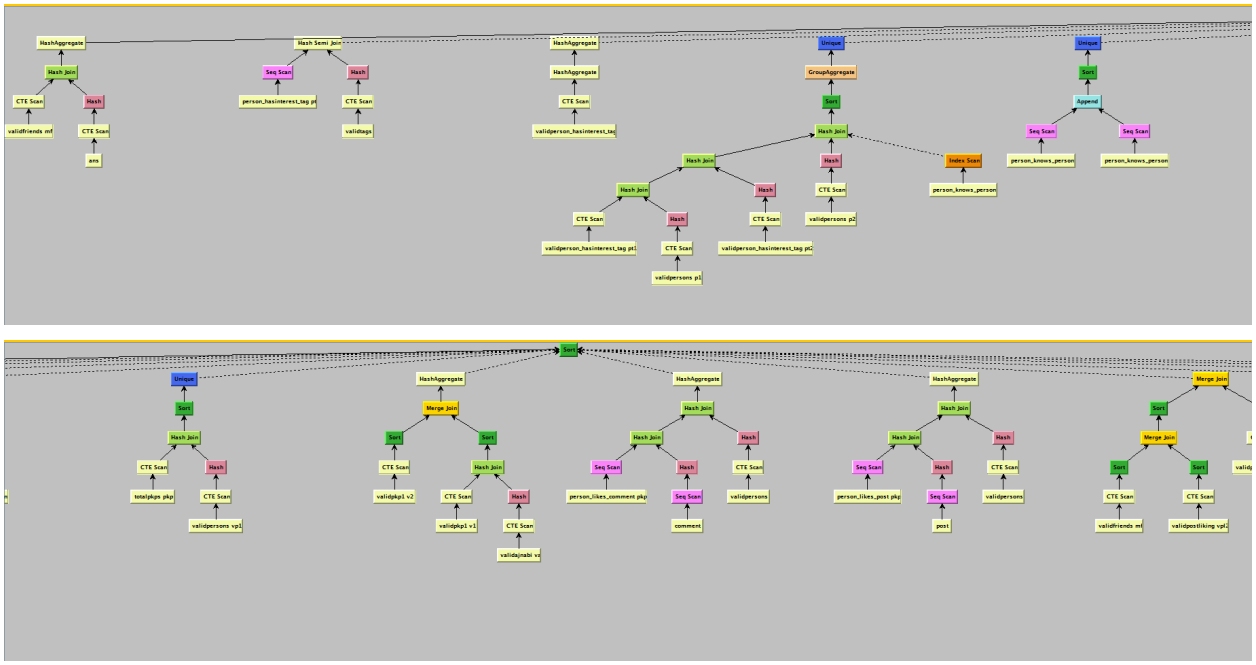


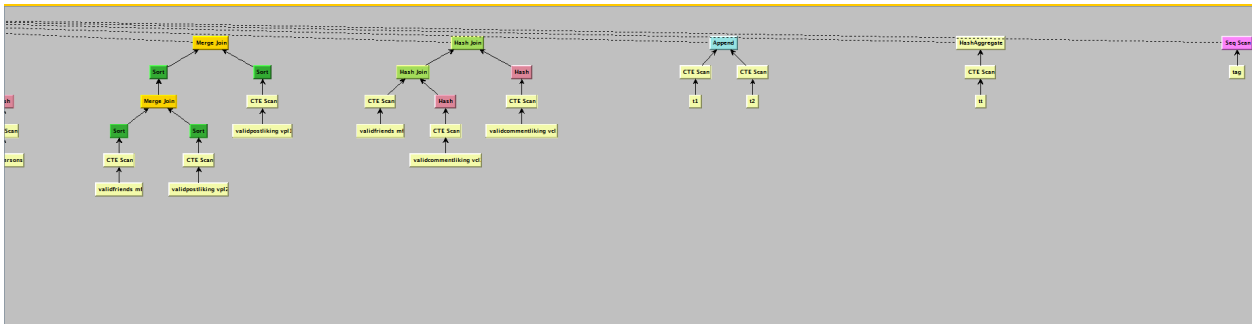
Plan 2:



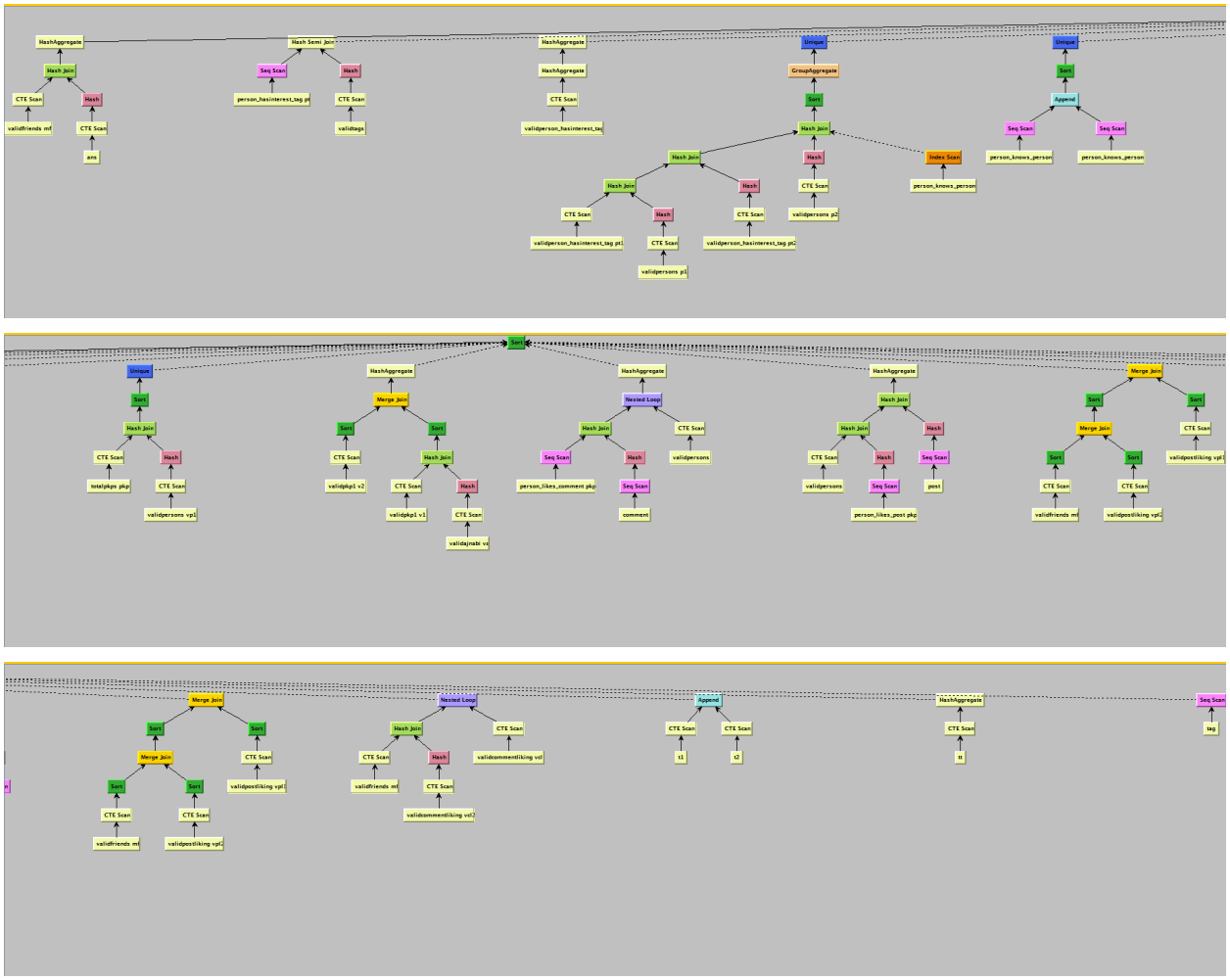


Plan 3:

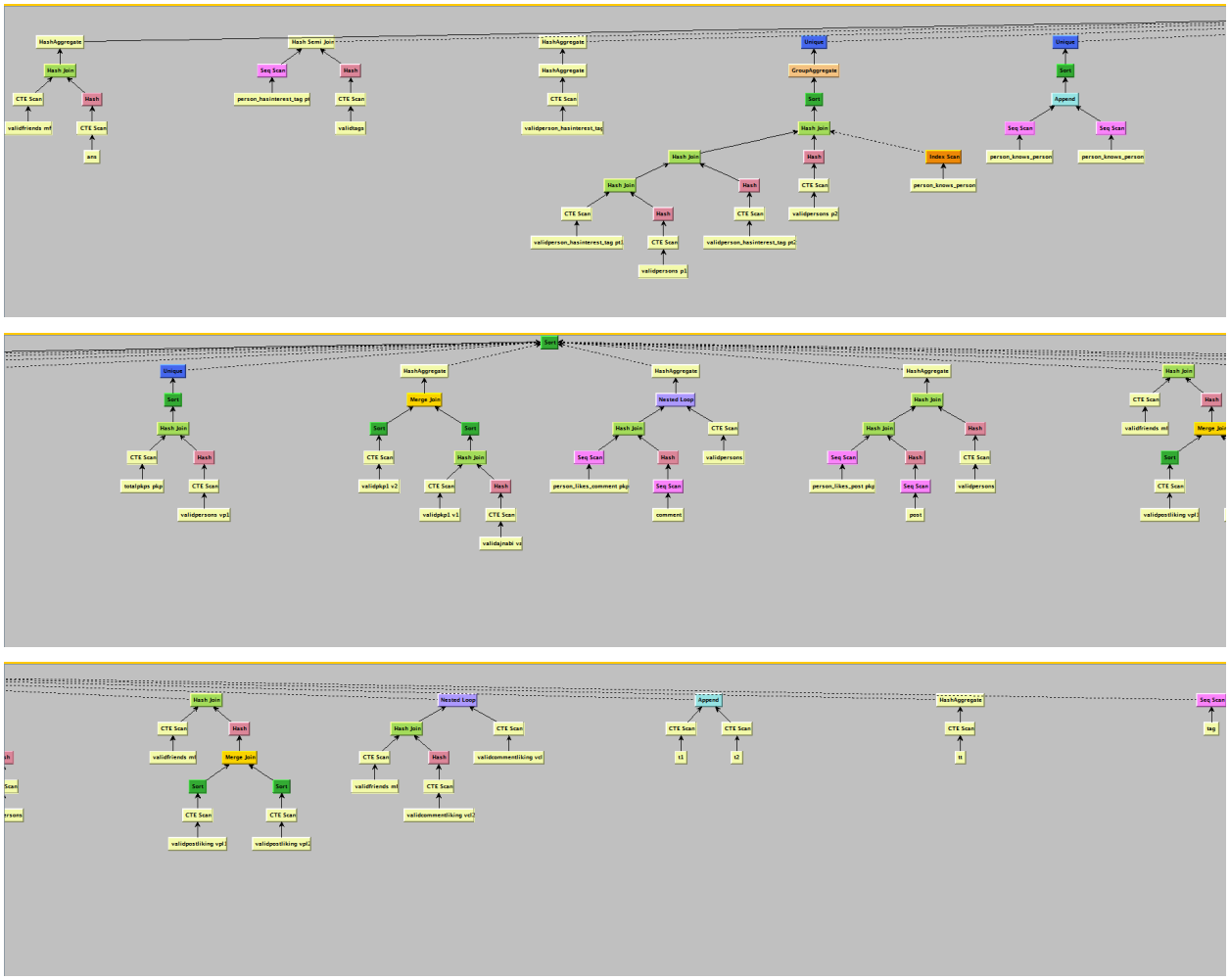




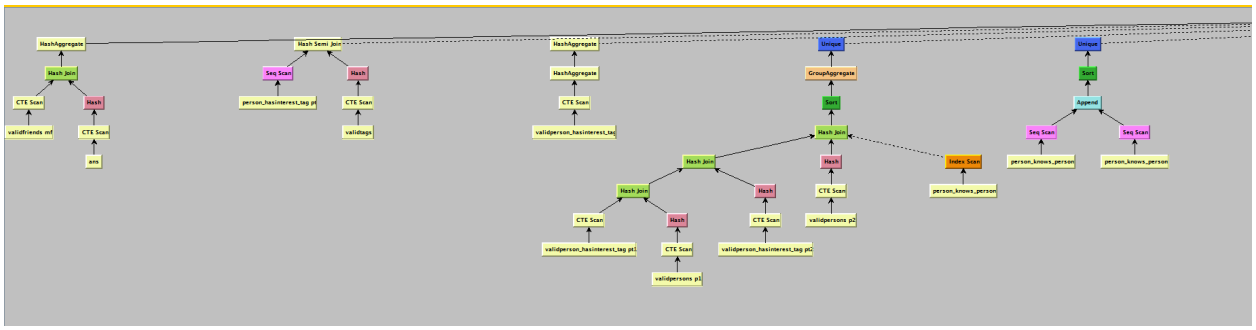
Plan 4:

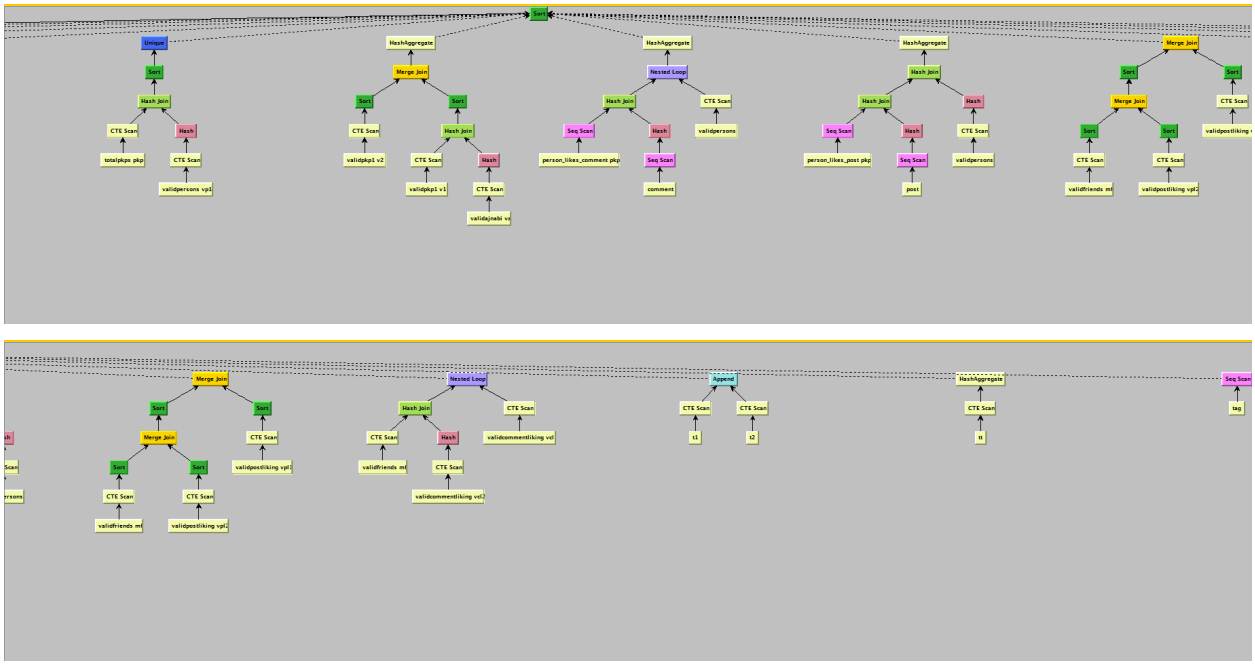


Plan 5:



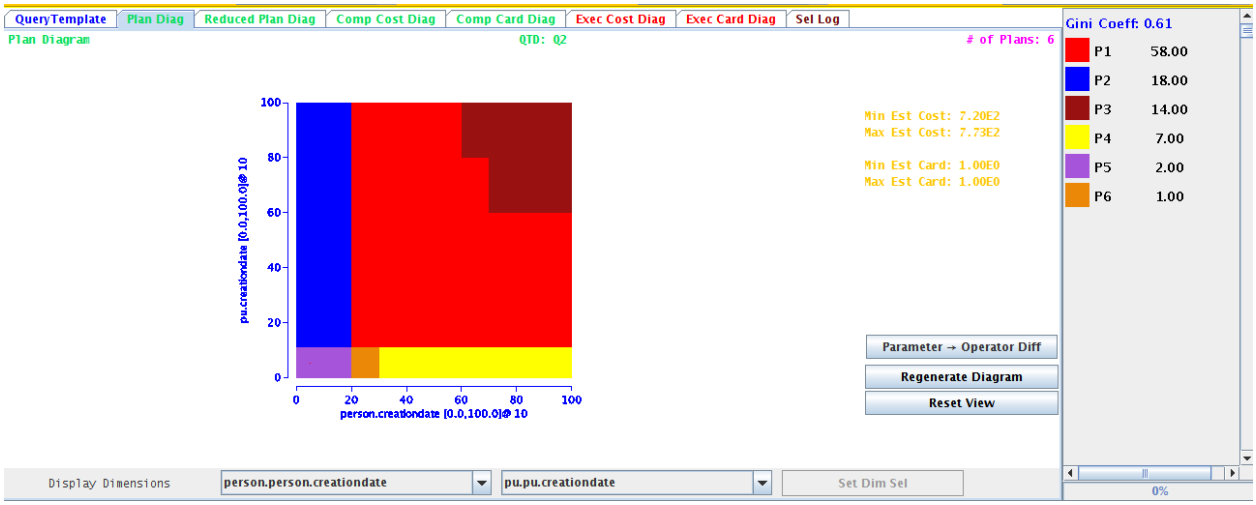
Plan 6:

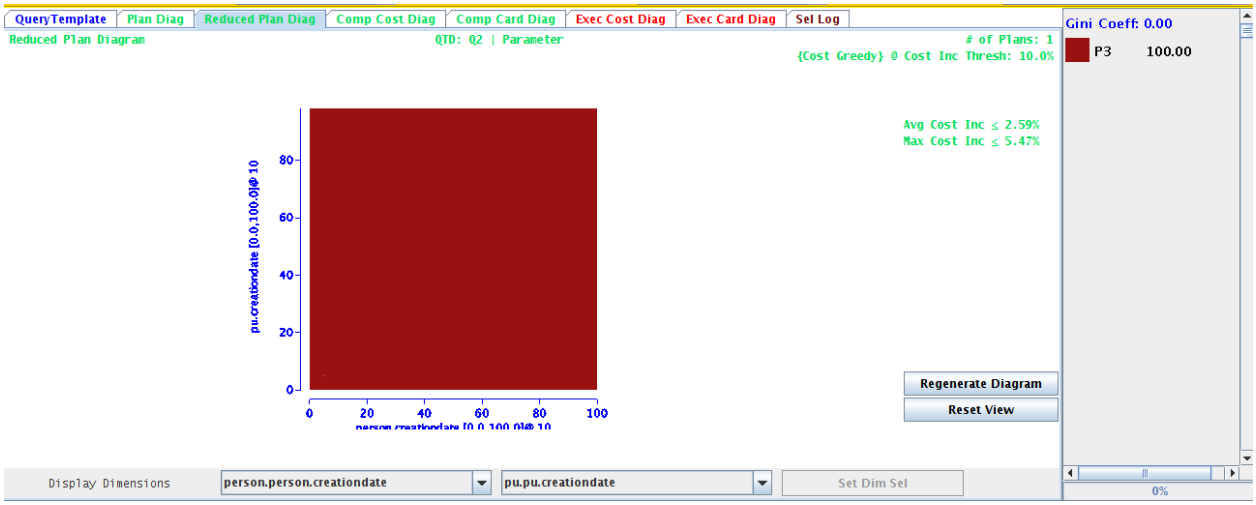




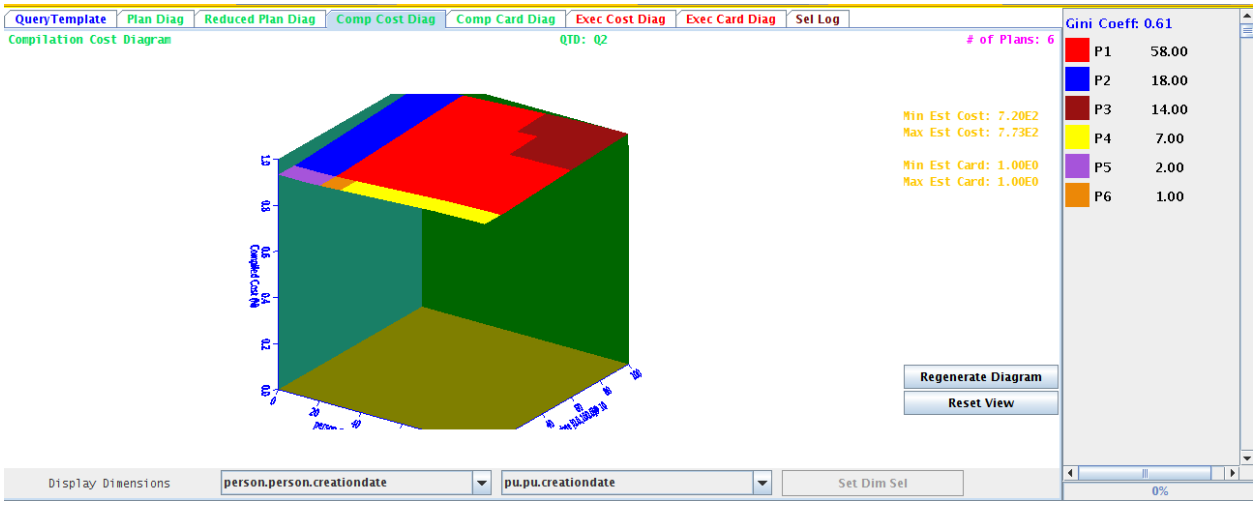
Query 2.

a) Exact Plan Diagram:



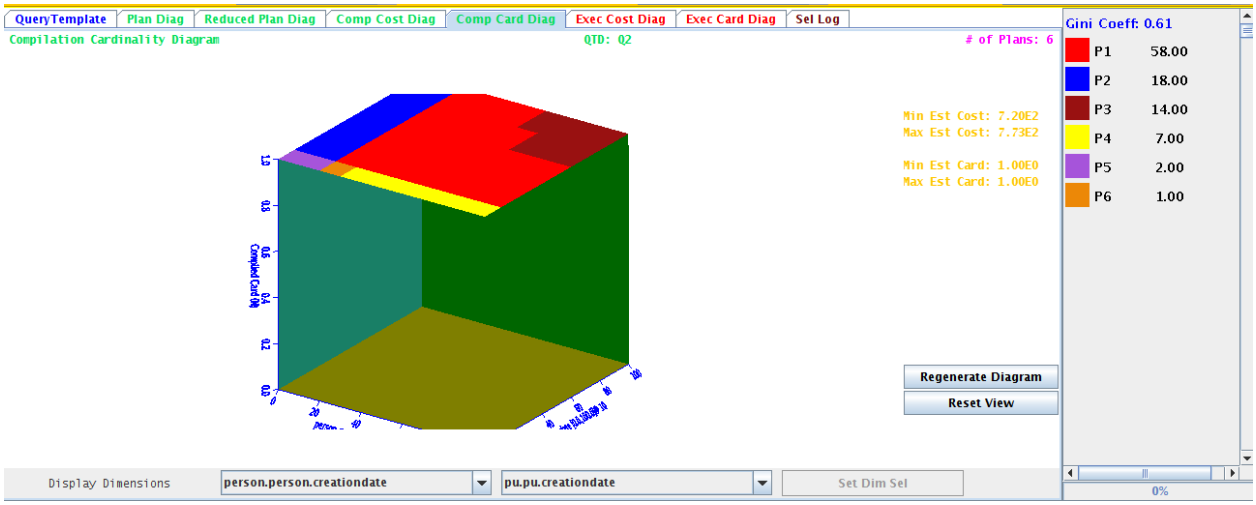


b) Compiled Cost Diagram:



The compilation cost remains the same for all 6 execution plans across all values of the variables.

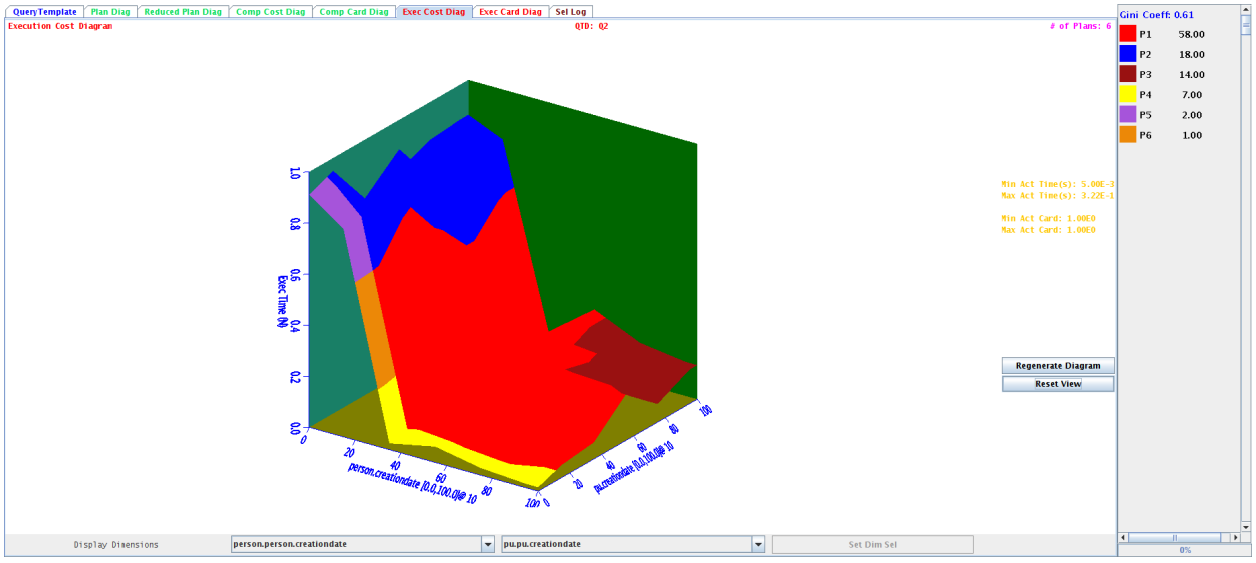
c) Cardinality Diagram:



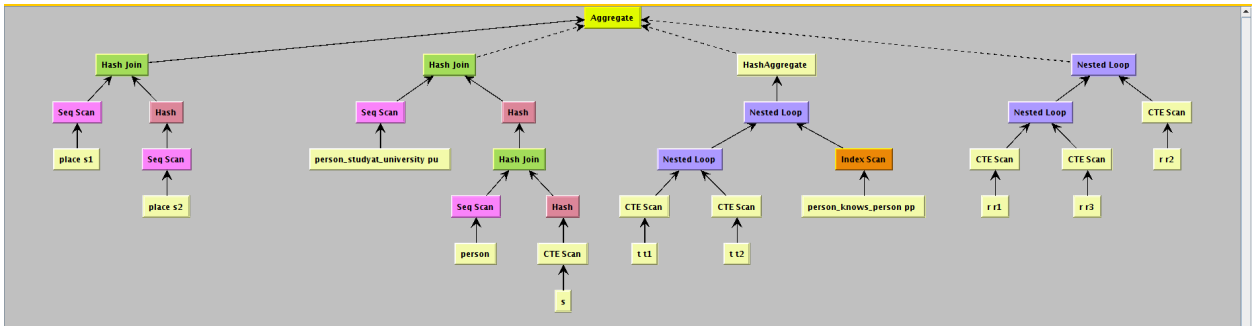
The cardinality remains the same across all 6 plans for all the values of the variables.

d) Plan Trees

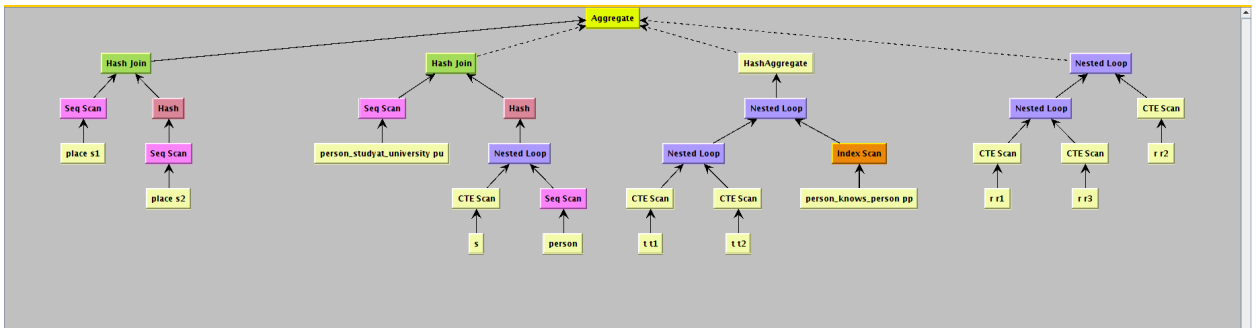
Execution Cost Diagram:



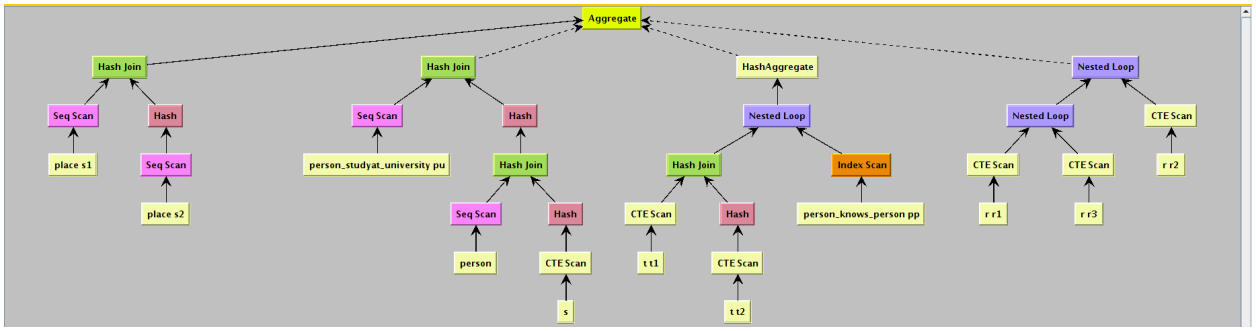
Plan 1:



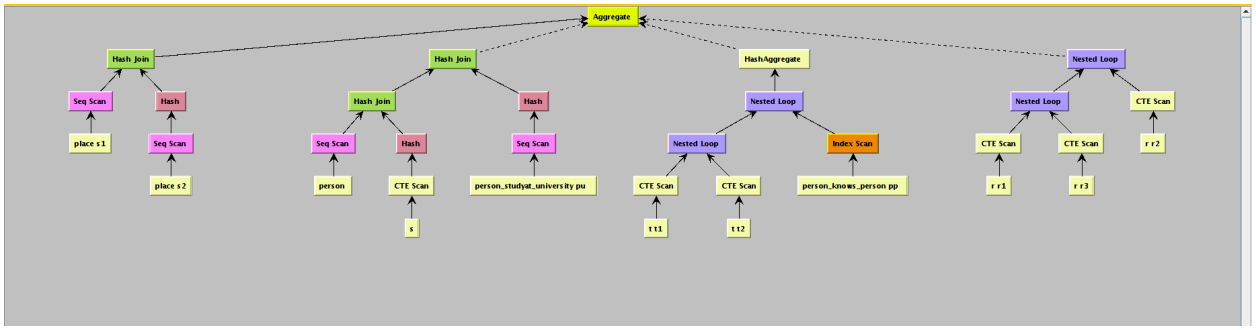
Plan 2:



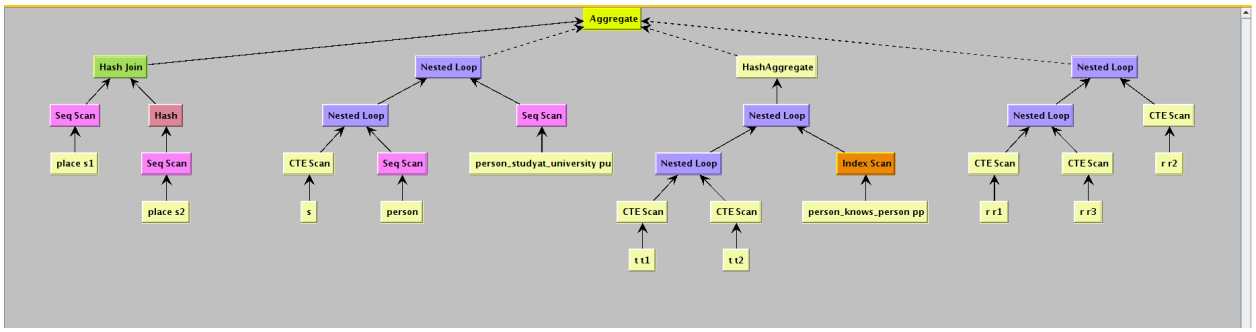
Plan 3:



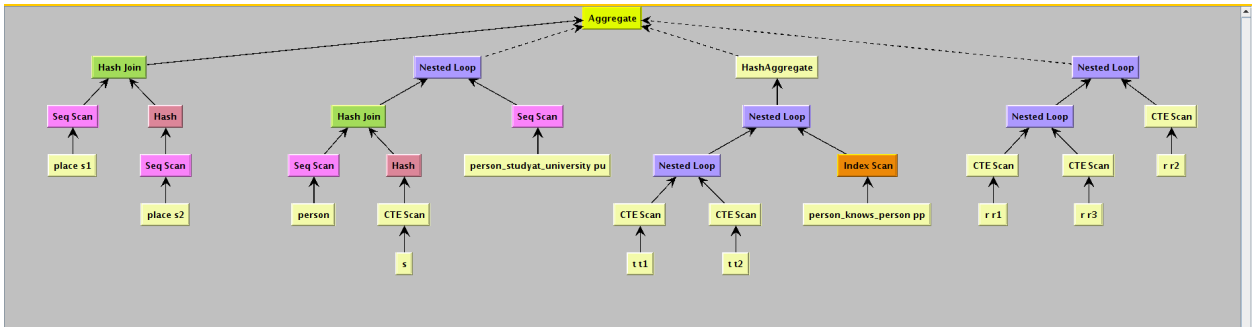
Plan 4:



Plan 5:

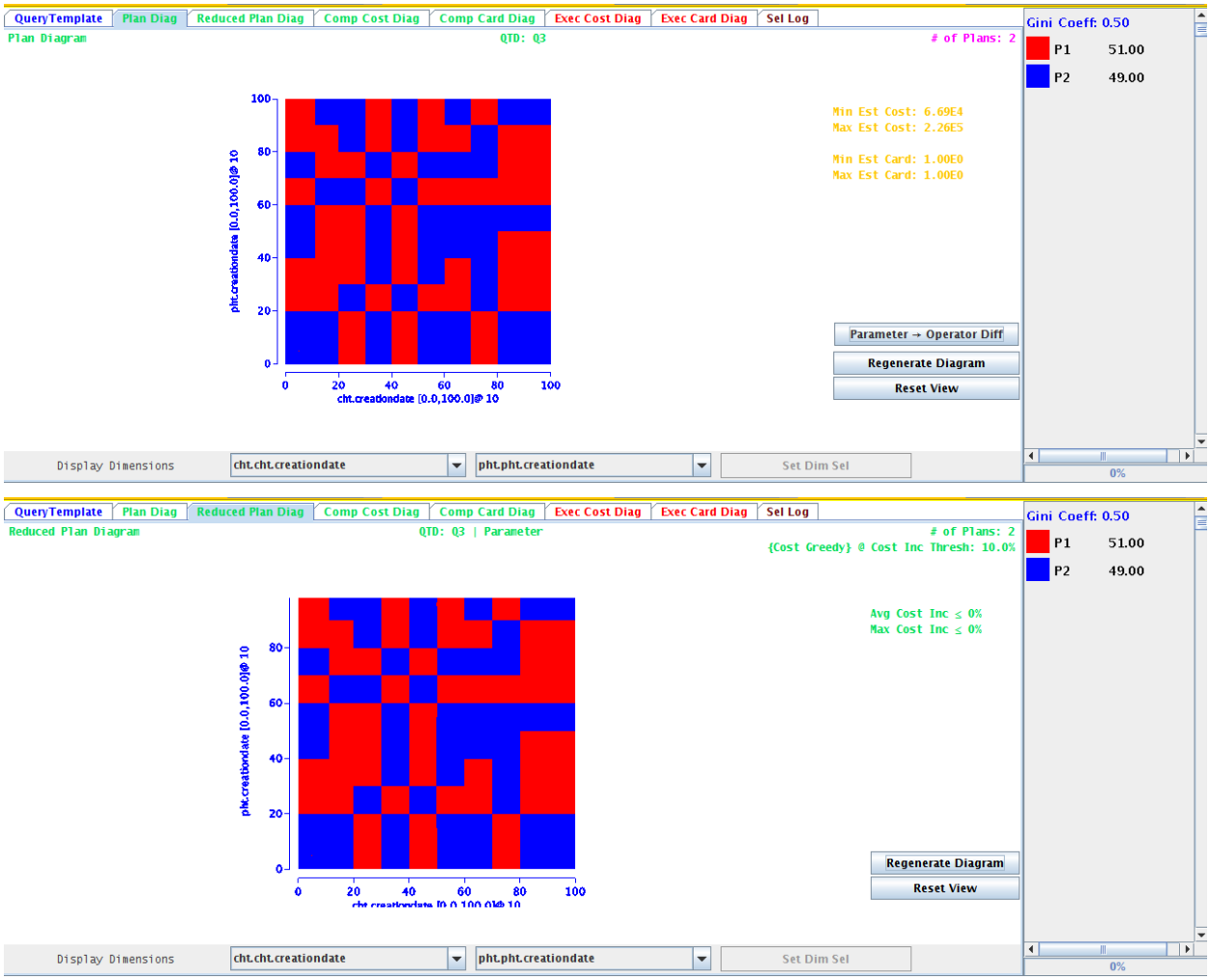


Plan 6:



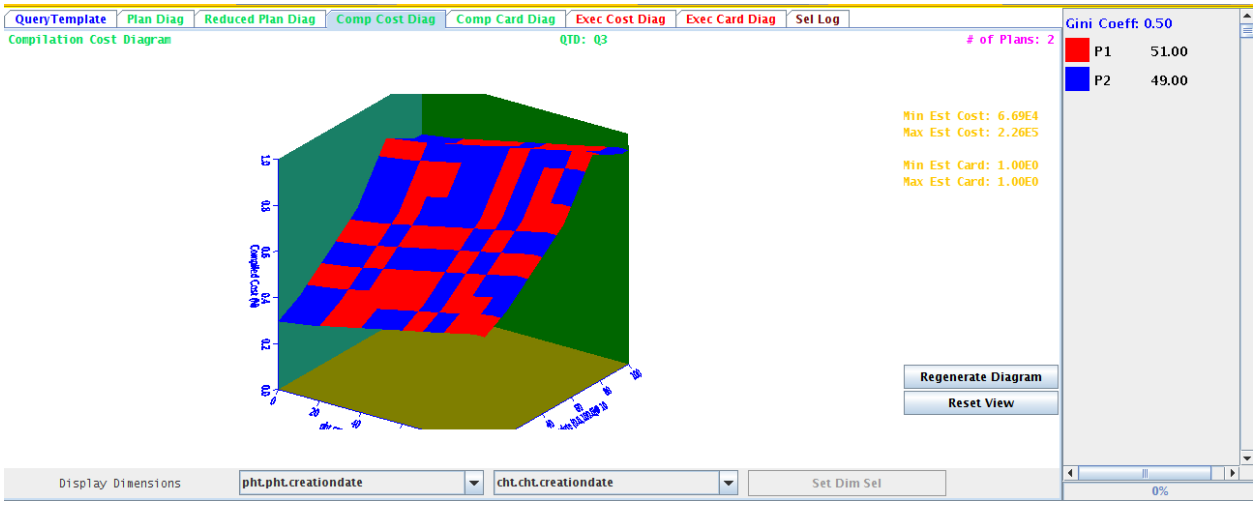
Query 3.

a) Exact Plan Diagram:



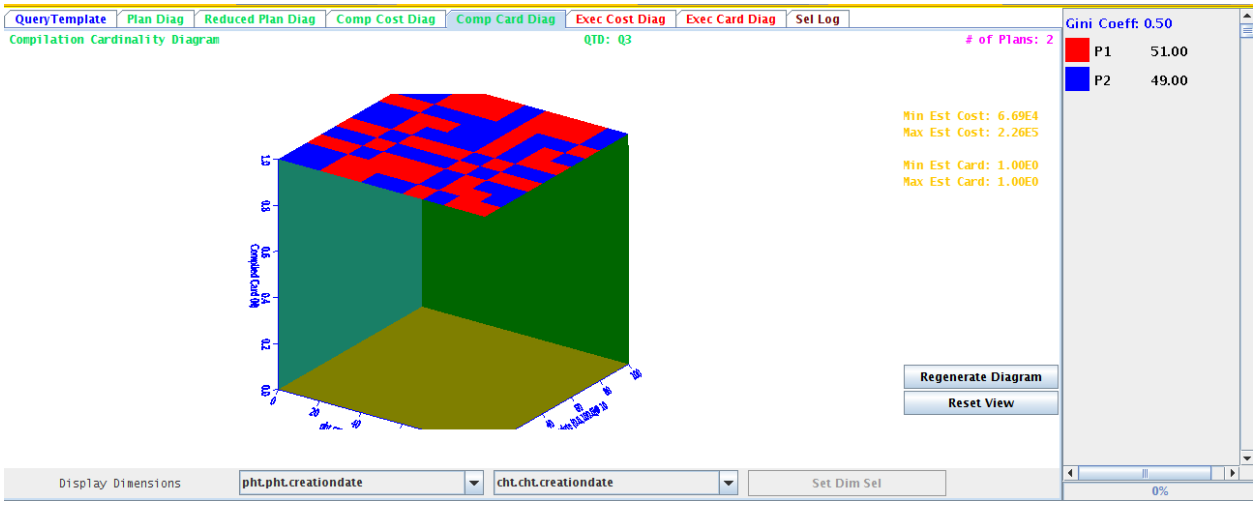
We see that there are 2 execution plans for the query, which toggle among themselves as different values of the variables.

b) Compiled Cost Diagram:



We see that as the values of the Picasso varies variables increase, the compilation cost also increases.

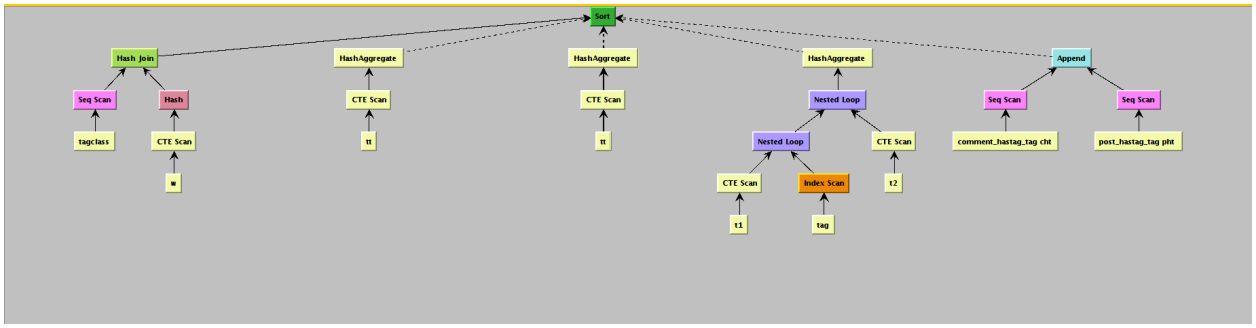
c) Cardinality Diagram:



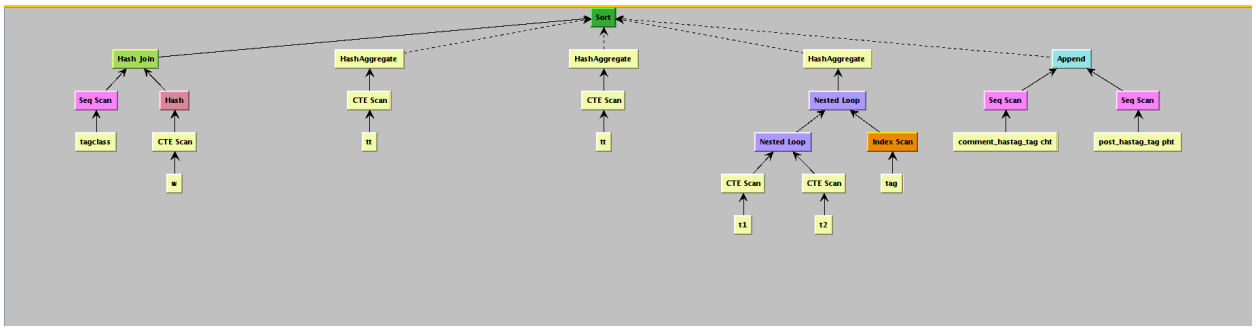
The cardinality remains the same for all values of the variables for both plans of executions.

d) Plan Trees

Plan 1:

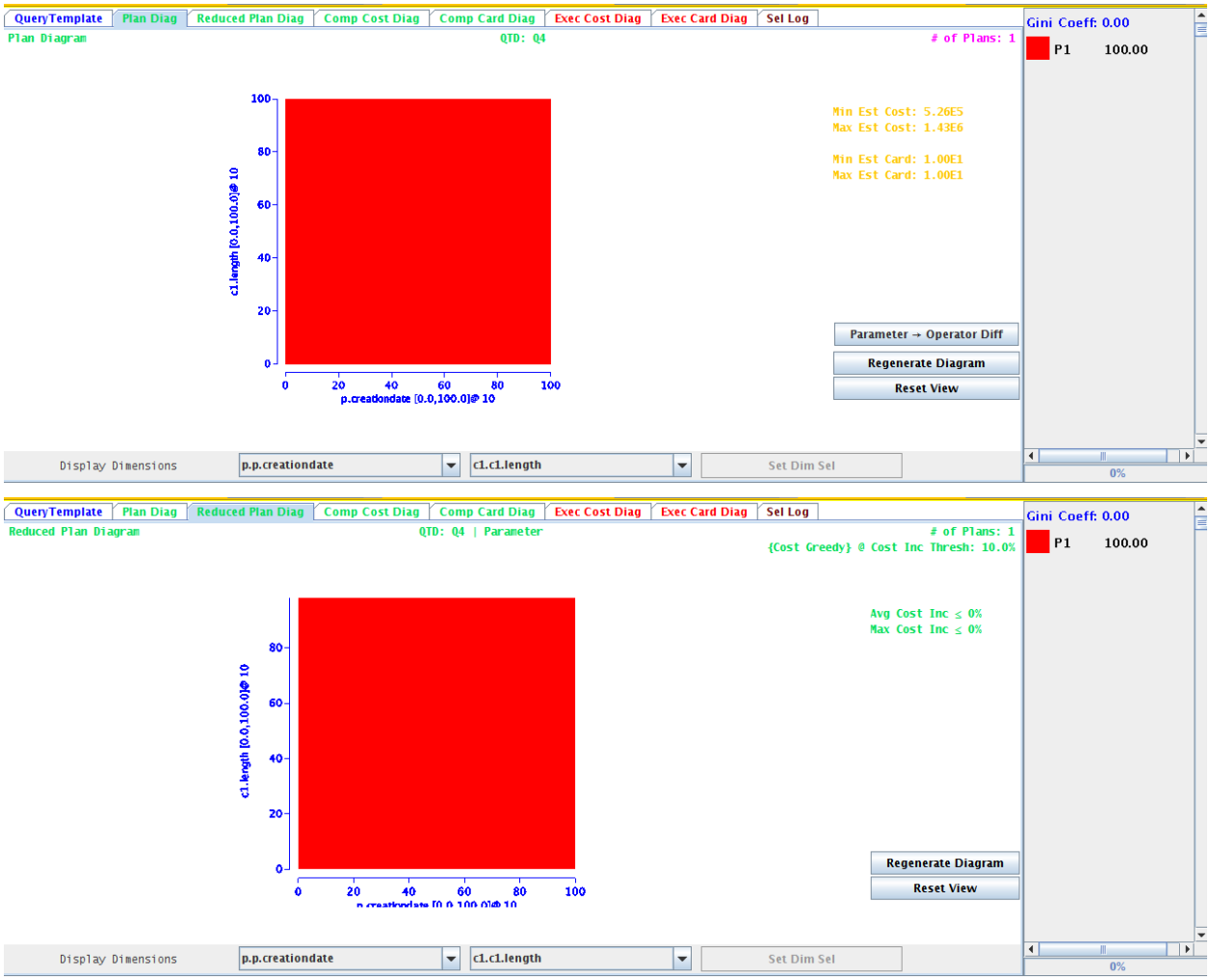


Plan 2:



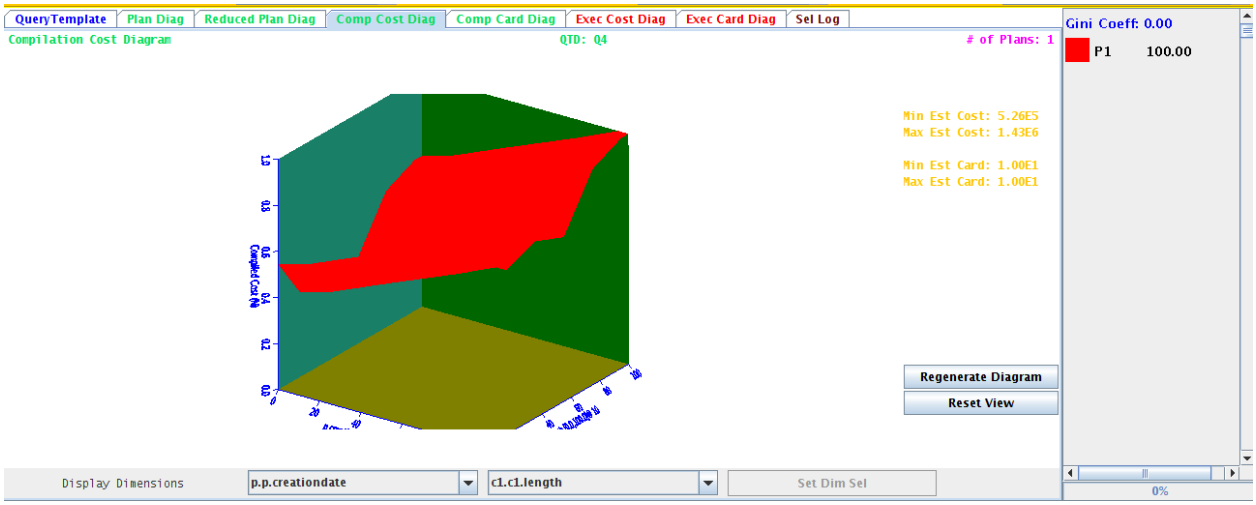
Query 4.

a) Exact Plan Diagram:



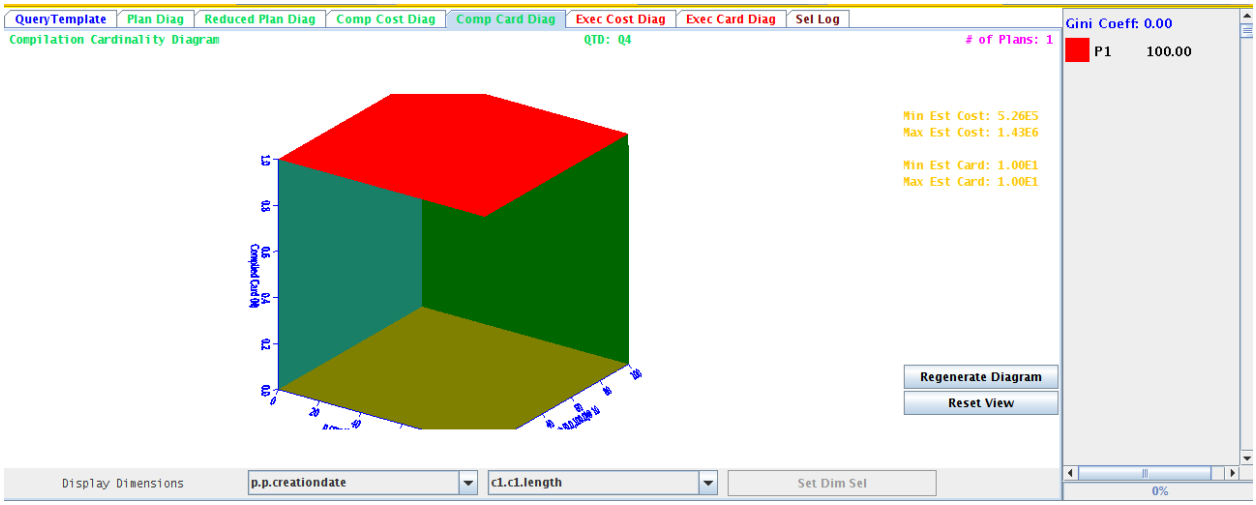
We see that there is only one plan of execution for all values of the both the variables.

b) Compiled Cost Diagram:



Although the trends are irregular at points, we can see the compilation cost increases with the increase in values of the variables approximately

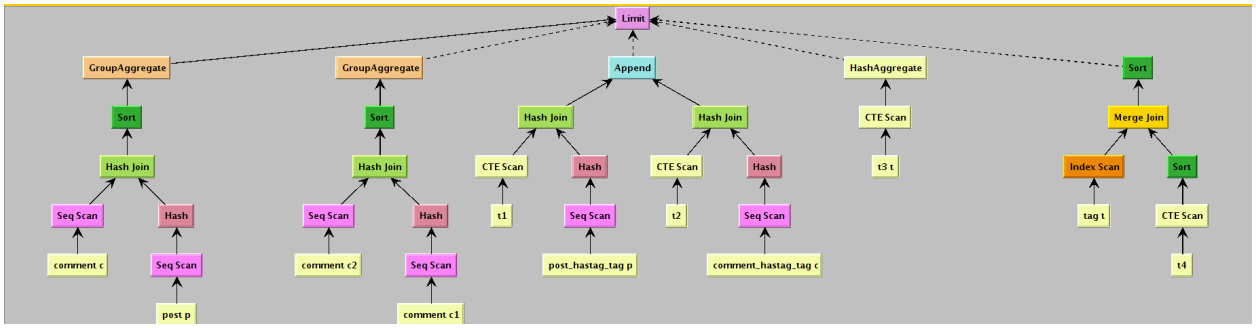
c) Cardinality Diagram:



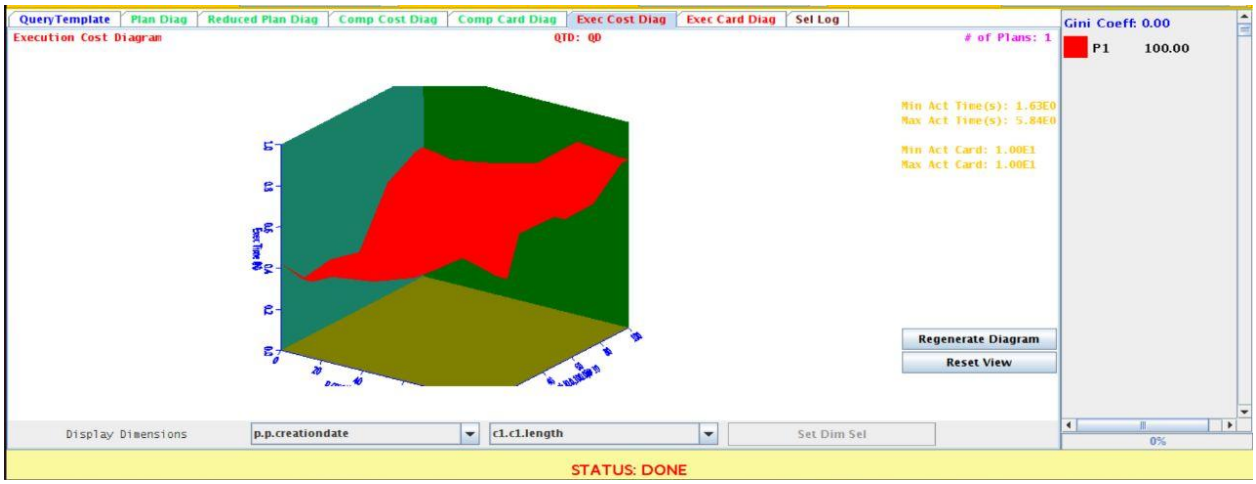
The cardinality remains same across all the values of the variables.

d) Plan Trees

Plan 1:

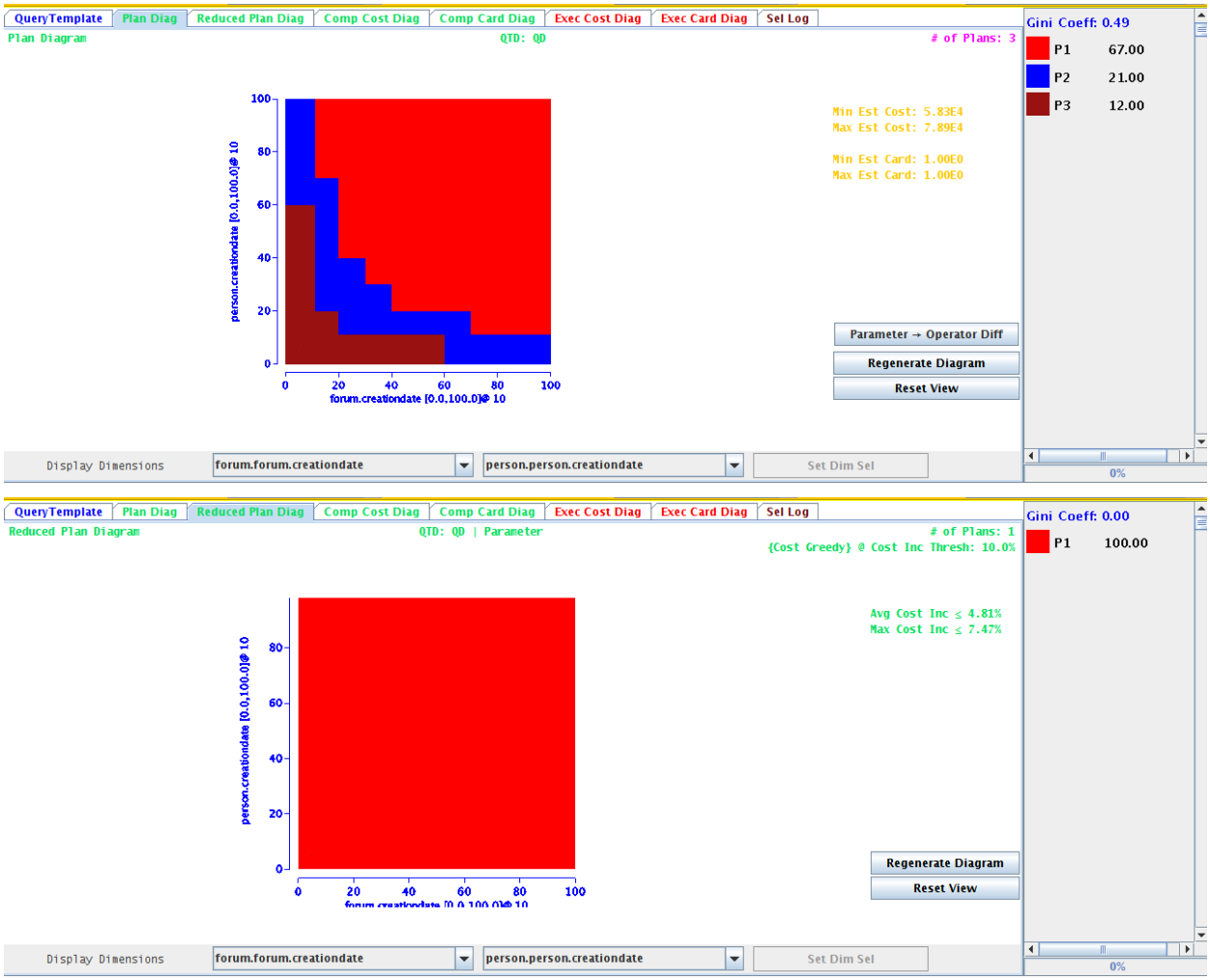


e) Execution Cost Diagram:



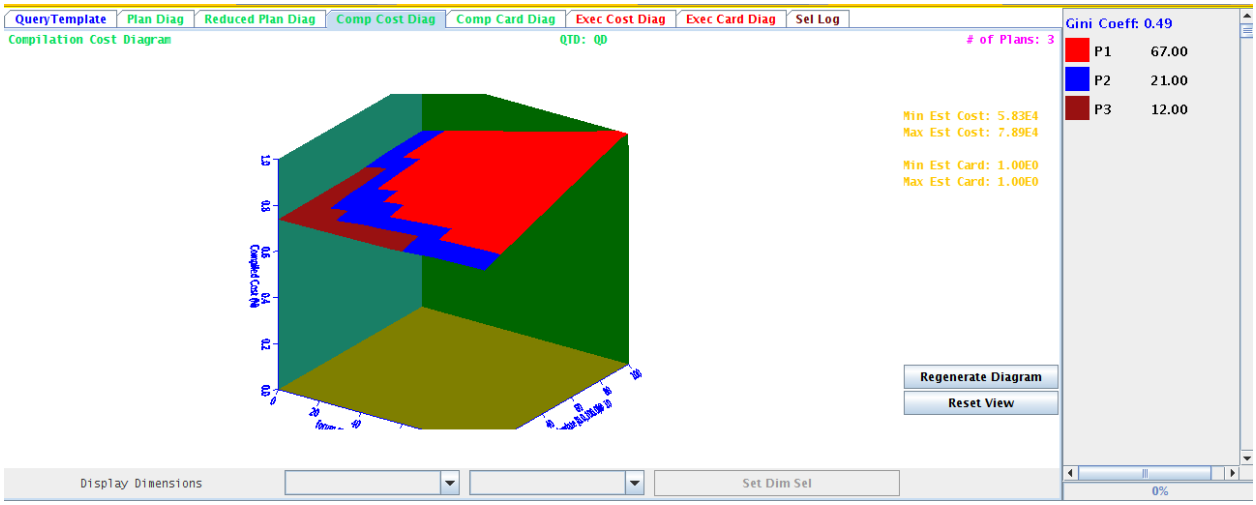
Query 5.

a) Exact Plan Diagram:



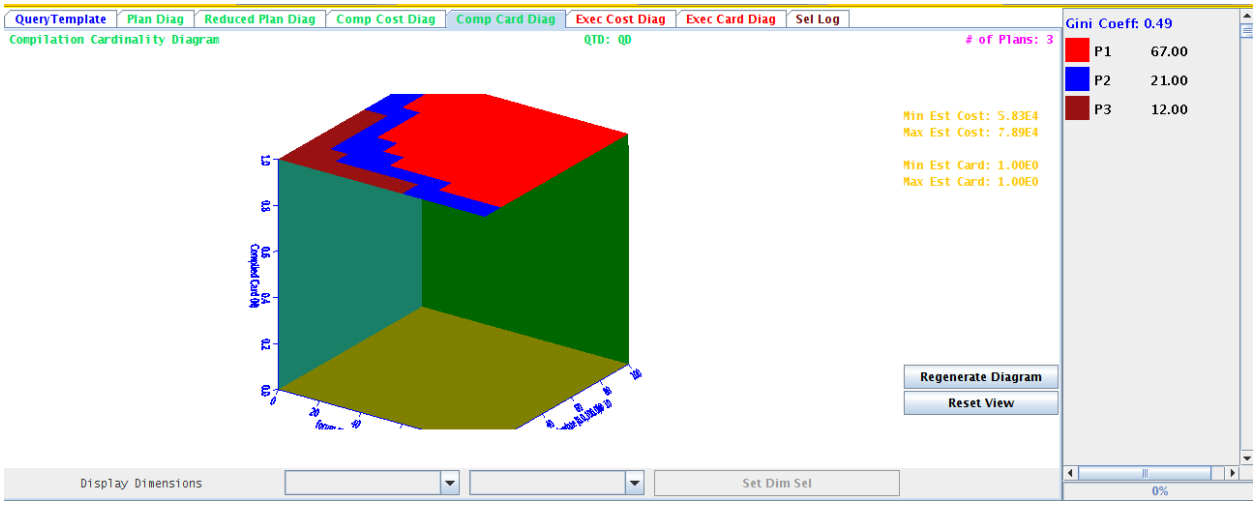
We see that there are 3 different plans of executions, most significant portion of which is P1 which occurs for 67% of the range of variables, and occurs for higher values of both x and y axis.

b) Compiled Cost Diagram:



We see that as the values of the both the variables increases, the compilation cost also increases. For lower values of the variables,account creation date and forum creation date, the cost for compilation is less.

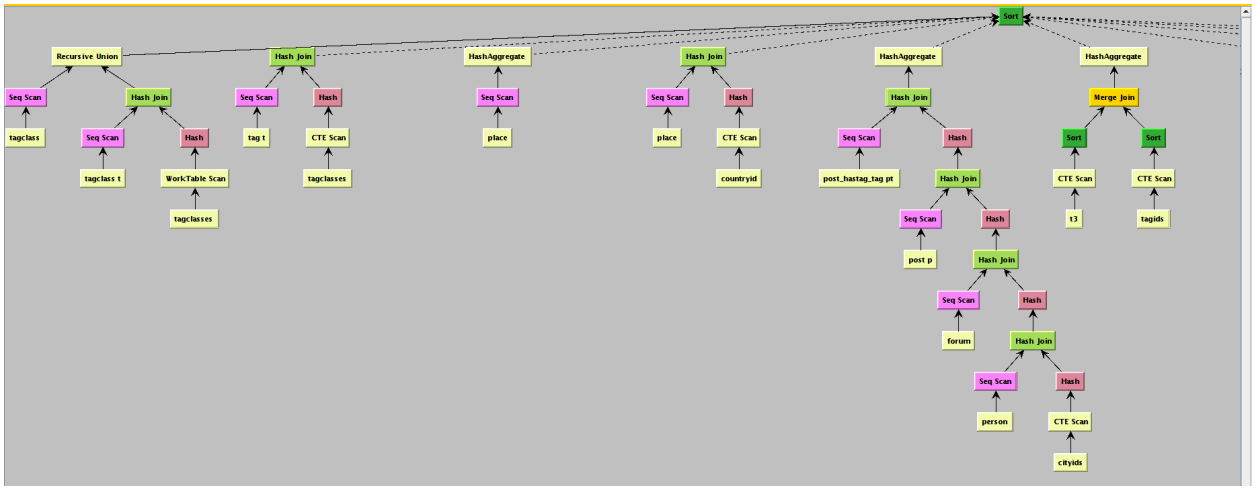
c) Cardinality Diagram:

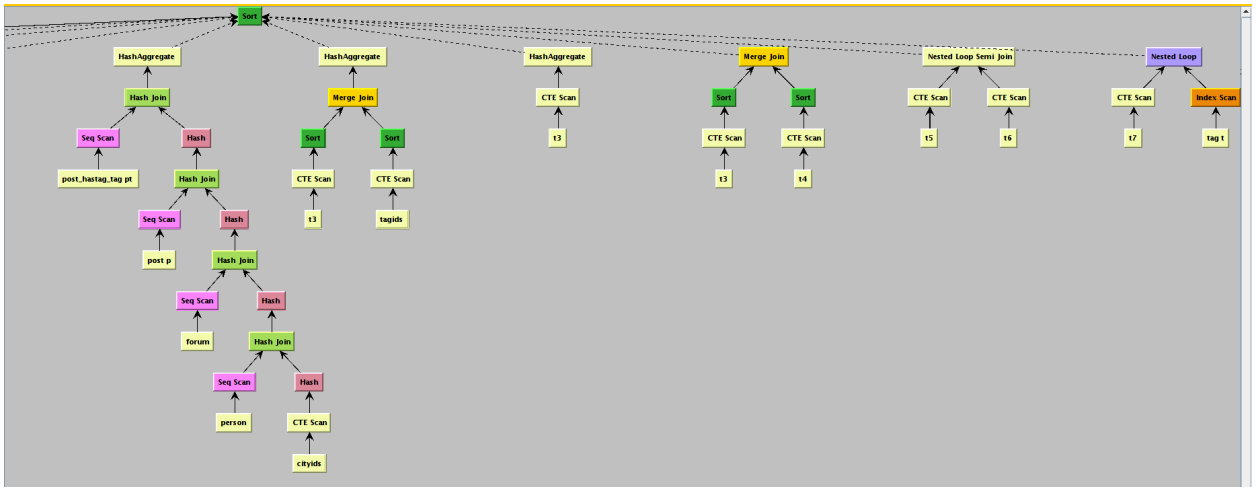


The cardinality remains the same for all 3 plans of execution across all the values of the two variables.

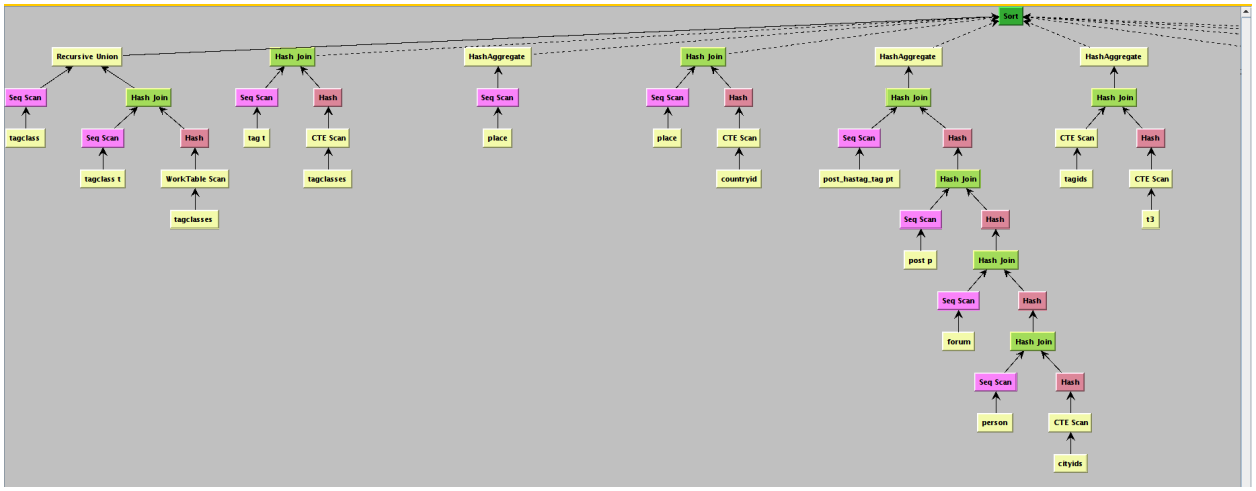
d) Plan Tree

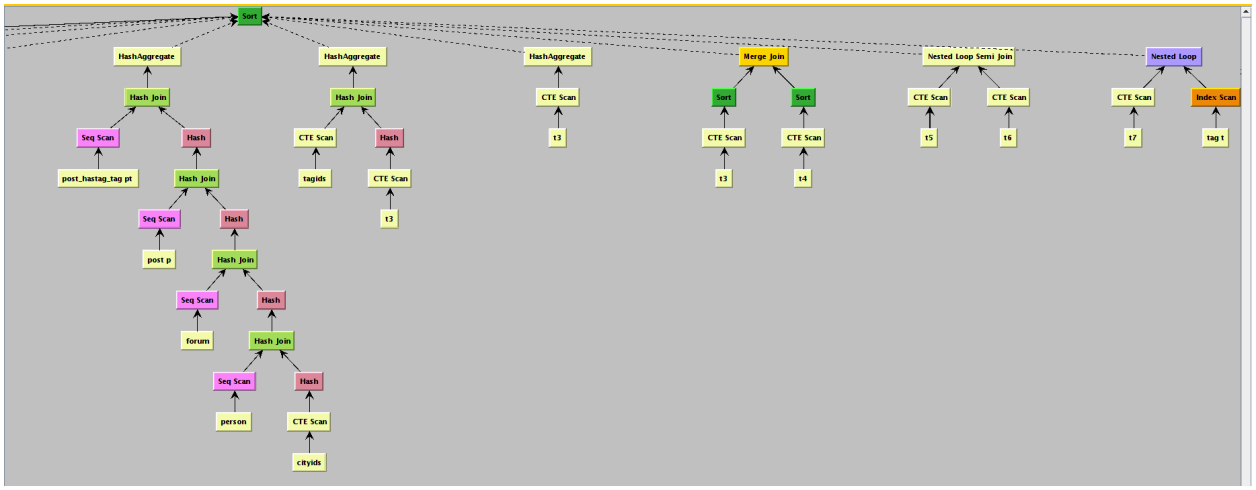
Plan 1:


Sort
Recursive Union
Hash Join
HashAggregate
Hash Join
HashAggregate
HashAggregate
Seq Scan
Hash Join
Seq Scan
Hash
Seq Scan
Hash
Seq Scan
Hash
Hash Join
Merge Join
tagclass
Seq Scan
Hash
tag t
CTE Scan
place
Seq Scan
Hash
Seq Scan
Hash
Sort
Sort
tagclass t
WorkTable Scan
tagclasses
place
CTE Scan
countryid
post_hastag_tag pt
Hash Join
CTE Scan
CTE Scan
tagclasses
tagclasses
Seq Scan
Hash
t3
tagids
post p
Hash Join
Seq Scan
Hash
forum
Hash Join
Seq Scan
Hash
person
CTE Scan
cityids

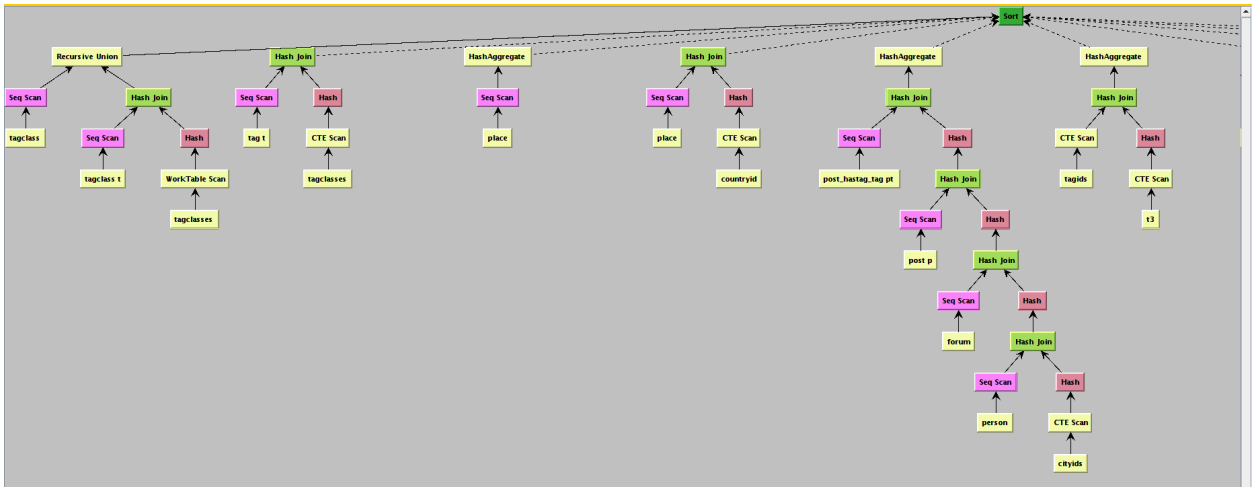

Sort
HashAggregate
HashAggregate
HashAggregate
Merge Join
Nested Loop Semi Join
Nested Loop
Hash Join
Merge Join
CTE Scan
Sort
Sort
CTE Scan
CTE Scan
CTE Scan
Index Scan
Seq Scan
Hash
Sort
Sort
t3
CTE Scan
CTE Scan
t5
t6
t7
tag t
post_hastag_tag pt
Hash Join
CTE Scan
CTE Scan
t3
tagids
t3
t4
Seq Scan
Hash
post p
Hash Join
Seq Scan
Hash
forum
Hash Join
Seq Scan
Hash
person
CTE Scan
cityids

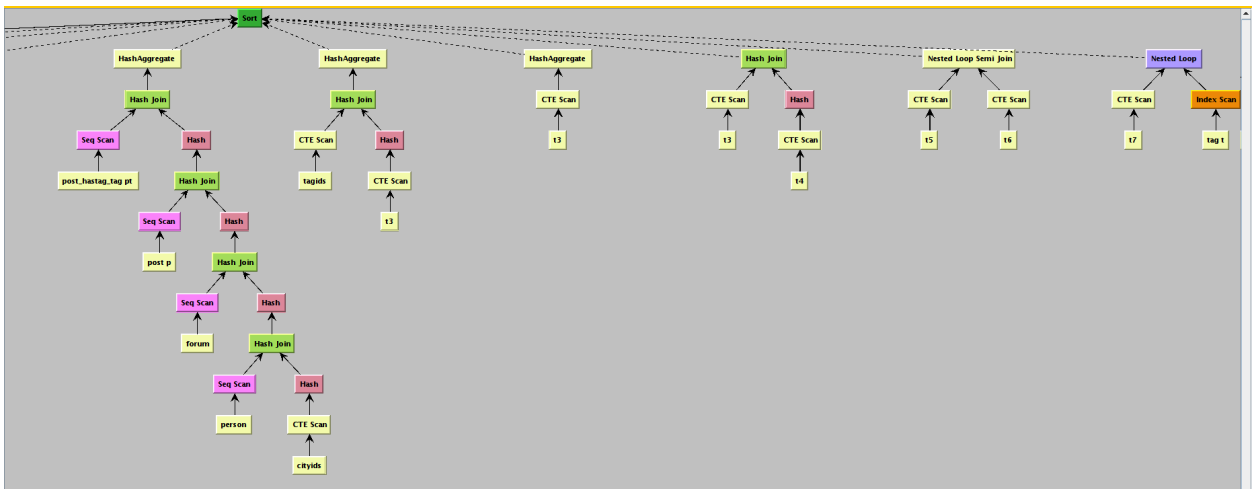
Plan 2:


Sort
Recursive Union
Hash Join
HashAggregate
Hash Join
HashAggregate
HashAggregate
Seq Scan
Hash Join
Seq Scan
Hash
Seq Scan
Hash
Hash Join
Hash Join
tagclass
Seq Scan
Hash
tag t
CTE Scan
place
Seq Scan
Hash
Seq Scan
Hash
CTE Scan
Hash
tagclass t
WorkTable Scan
tagclasses
place
CTE Scan
countryid
post_hastag_tag pt
Hash Join
tagids
CTE Scan
tagclasses
Seq Scan
Hash
t3
post p
Hash Join
Seq Scan
Hash
forum
Hash Join
Seq Scan
Hash
person
CTE Scan
cityids

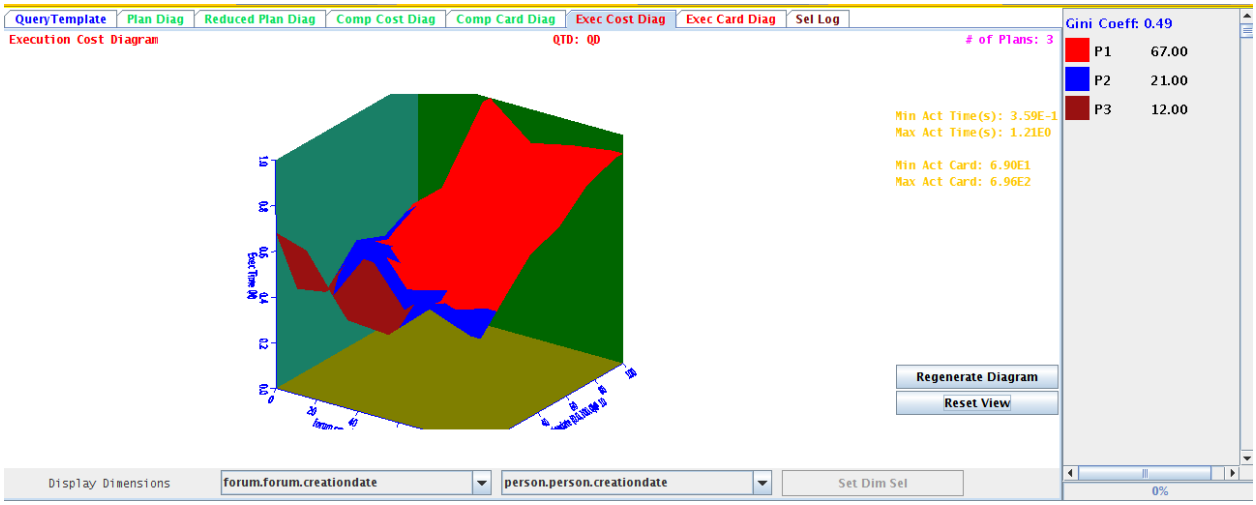

Sort
HashAggregate
HashAggregate
HashAggregate
Merge Join
Nested Loop Semi Join
Nested Loop
Hash Join
Hash Join
CTE Scan
Sort
Sort
CTE Scan
CTE Scan
CTE Scan
CTE Scan
Index Scan
Seq Scan
Hash
CTE Scan
Hash
t3
CTE Scan
CTE Scan
t5
t6
t7
tag t
post_hastag_tag pt
Hash Join
tagids
CTE Scan
t3
t3
t4
Seq Scan
Hash
post p
Hash Join
Seq Scan
Hash
forum
Hash Join
Seq Scan
Hash
person
CTE Scan
cityids

Plan 3:


Sort
Recursive Union
Hash Join
HashAggregate
Hash Join
HashAggregate
HashAggregate
Seq Scan
Hash Join
Seq Scan
Hash
Seq Scan
Hash Join
Seq Scan
Hash
Hash Join
Hash Join
tagclass
Seq Scan
Hash
tag t
CTE Scan
Seq Scan
Hash
CTE Scan
Hash
Seq Scan
Hash
Hash Join
Hash Join
tagclass t
WorkTable Scan
tagclasses
tagclasses
place
place
CTE Scan
countryid
post_hastag_tag pt
Hash Join
tagids
CTE Scan
t3
Seq Scan
Hash
post p
Hash Join
Seq Scan
Hash
forum
Hash Join
Seq Scan
Hash
person
CTE Scan
cityids


Sort
HashAggregate
HashAggregate
HashAggregate
Hash Join
Nested Loop Semi Join
Nested Loop
Hash Join
Hash Join
CTE Scan
CTE Scan
Hash
CTE Scan
CTE Scan
CTE Scan
Index Scan
Seq Scan
Hash
CTE Scan
Hash
t3
t3
CTE Scan
t5
t6
t7
tag t
post_hastag_tag pt
Hash Join
tagids
CTE Scan
t4
t3
Seq Scan
Hash
post p
Hash Join
Seq Scan
Hash
forum
Hash Join
Seq Scan
Hash
person
CTE Scan
cityids

e) Execution Cost Diagram:



We see that as the values of the account creation date and forum create date increases, the execution cost increases. While in lower values of the variable we see irregular trends in the execution cost diagrams, especially for Plans 2 and 3 of execution.

Optimizations

1. Our first naive implementation took several minutes to get the output. This was because we were joining very large tables like person X person or personknowsperson X personknowsperson without any pre processing and filtering. We established a filter that any person in our output should be liking at least k tags from the tag list. This helped us reduce the table sizes substantially and saved a lot of execution time. Subsequent reordering of operations and using distinct and union only when absolutely required also helped us bring down the execution time to 7.2s on our machines.
2. In our first attempt, we naively joined all the relations without thinking much about the execution time and consequently the run time was about 1.2s. Then we realized that it was a bad way as there were many other constraints as well which could help bring the size of intermediate relations down. So, we reordered the joins and constraints and separated them to get a whopping 20ms execution time for this query on our machines.
3. We first optimized this query by just reordering some relation joins and constraints to ~500ms. In other queries, using index table didn't gave us a lot of performance improvement, but here it did! We were able to bring down the execution time to ~200ms as the tables 'comment_hastag_tag' and 'post_hastag_tag' as these were relatively large and had repetive operations for their creationdate attribute. The index table helped improving the query performance.
4. In the naive implementation, we were first using count operator, and then in the next table taking the select operator, but in the optimised queries we first use the select operator

using count and then creating the table for the next use. This reduced the execution time by almost 1200ms as the comments and post tables are large.

5. In our first attempt, we naively joined the persons table with forum and then used the country filter on the person to get the desired final table. But in the optimised query, we first fiter the people using the country ID and then join it with the forums table to get the desired table which reduced the execution time by almost ~500ms.

README

The execution cost diagram have been made for Q2, Q4 and Q5.
Please find the images attached for each query in the graph folder. And the picasso version of the query to get these outputs in the picasso_diagram.sql file.
