In Postgres, when we use explain with our select statement, the query plan is returned as output which has several rows:

The topmost row tells us the aggregate cost of the query. This overall cost helps to determine the overall efficiency of a query and the total no of rows as the output.

Each susbsequent row tells us the query proceeding how it is calculating and reaching the result.

Values in Rows:

### 1.Seq Scan (Sequential Scan)

* Indicates a full table scan.

2. Index Scan

* Uses an index to retrieve data efficiently.

3. Index Only Scan

* Uses an index only to retrieve data efficiently (happens when all the filters mentioned constitue an index).

4. Nested Loop (Join Strategy)

* Iterates over one table for each row in another table.

5. Hash Join

* Builds a hash table from one table and probes it with another.

6. Merge Join

* Sorts both tables and then merges them efficiently.

7. Sort

* Indicates explicit sorting.

8. Subquery Scan

* Executes a subquery and uses its results.

9. Filter

* Represents a WHERE condition applied after fetching rows.

10. Group Key:

* Tells the key on which the grouping is performed.

2.1 The value under 'rows' column in the output of EXPLAIN query and SELECT query after it are same. What does it mean?

Here, the rows tells us the no of rows that will be checked while running the query. The number of rows in the explain query and select query are same because the query is going to check all the rows which are present in the table since the indexing is not maintained there.

2.2 Is the SELECT query optimal? If no, how do we optimize it?

The select query is optimal but we need to index the column user\_id in our table so that the no of rows which are getting checked gets reduced and will also reduce the cost and time for running that query.

Q3.1 We decide to index columns in comments table to optimize the SELECT query. What column(s) will you index in which order?

We should create a composite index on (commentable\_type, commentable\_id, user\_id), since all three columns are involved in filtering the data.

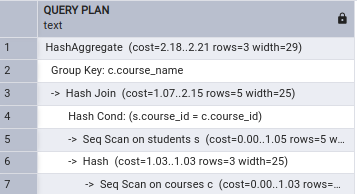
The reason is:

commentable\_type: Since it is a string column with low cardinality, filtering on it first will reduce the number of rows in a huge amount to be searched.

commentable\_id: After filtering by commentable\_type, the commentable\_id helps locate the specific entity.

user\_id: This refines the results to the specific user's comments on that entity.

4.1 EXPLAIN a SELECT query against one of your databases which employs an INNER JOIN between two tables. What does the output look like? What does the values under different columns mean? Do you get only one row in EXPLAIN's output?



This is the result I am getting after running the query:

EXPLAIN SELECT c.course\_name, COUNT(s.student\_id) AS total\_students

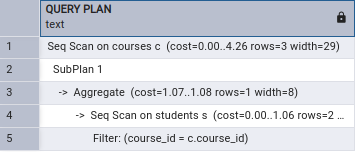
FROM courses c

INNER JOIN students s ON c.course\_id = s.course\_id

GROUP BY c.course\_name;

This query plan tells us that firstly it creates a hash on courses and students by sequential scan on the tables. Then, joins the tables on the condition s.course\_id = c.course\_id. Then, the grouping is happening on the basis of the course\_name and in the end the aggregate is created.

4.2 Form the same select query in above question using a subquery instead of a JOIN. What does the EXPLAIN output look like now? Which query is better and why?



This is the result when I perform the below query:

EXPLAIN SELECT c.course\_name,

(SELECT COUNT(\*) FROM students s WHERE s.course\_id = c.course\_id) AS total\_students

FROM courses c;

This query plan indicates that for a sequntial scan on courses c, we have another sequential scan on student which has a filter saying course\_id = c.course\_id. Sequential plan on student s aggregates as part of the subplan.

Looking at the result, we can come to a fact that JOIN is better since the cost of the subquery can go upto 4.26 compared to 2.21 for the JOIN query performing the same work.