**Task 1**



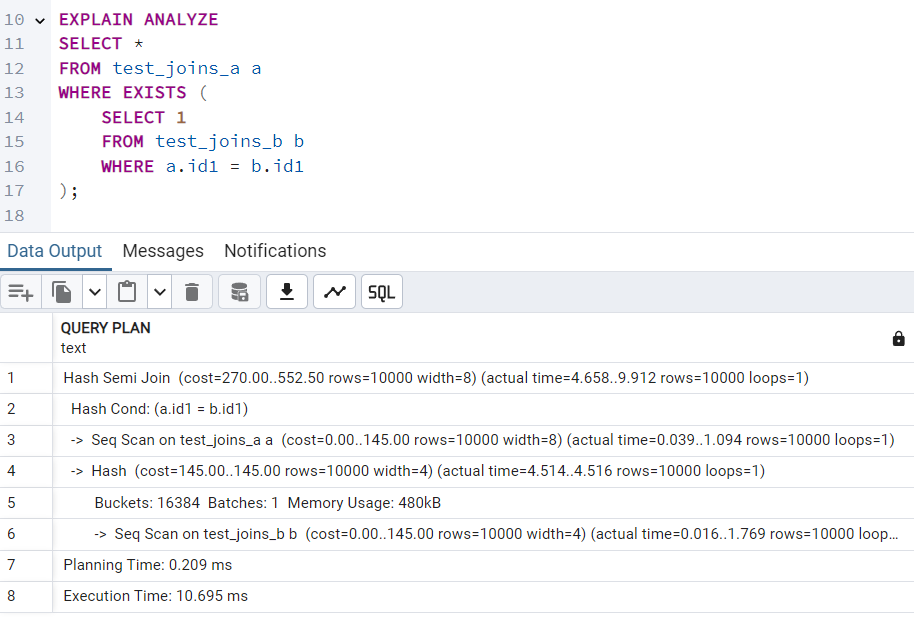
* Query 1 (a.id1 > b.id1) uses a Nested Loop Join due to the inequality condition, preventing other join optimizations.
* Query 2 (CROSS JOIN) inherently requires a Nested Loop Join because it generates a Cartesian product.
* Performance Concern: Both queries have a complexity of O(n²), which can be slow for large datasets.

**Task 2**

PostgreSQL typically avoids using a Hash Join for inequality conditions (>) because Hash Joins are optimized for equality conditions (=). However, we can try to force it using the SET enable\_nestloop = off; to prevent the planner from choosing a Nested Loop Join, allowing other join types to be considered.

****

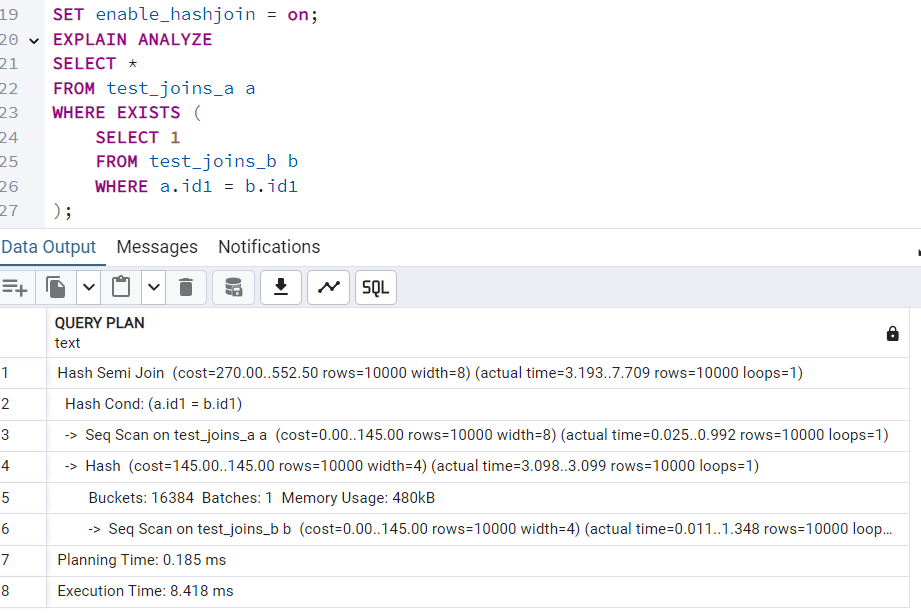
A **Semi-Join** returns rows from the left table (test\_joins\_a) where **matching rows exist** in the right table (test\_joins\_b), but it **does not return duplicates from the right table**.  
To encourage a **Hash Semi Join**, we use EXISTS or IN with equality conditions:

****

To check how PostgreSQL behaves without Hash Join, disable it using:

****

Now, re-enable Hash Join:

****

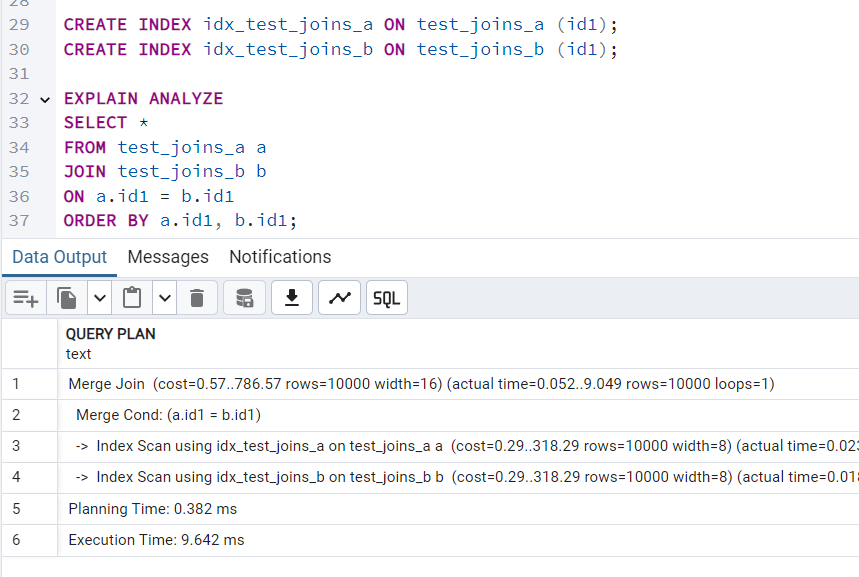
**Conclusion**

* Forcing Hash Join for > conditions is difficult because Hash Joins work best with equality conditions.
* A Hash Semi Join can be encouraged using EXISTS or IN with equality conditions.
* Disabling Hash Join forces PostgreSQL to fall back on less efficient alternatives, emphasizing the importance of join selection.
* Re-enabling Hash Join improves performance for queries benefiting from hash-based lookups.

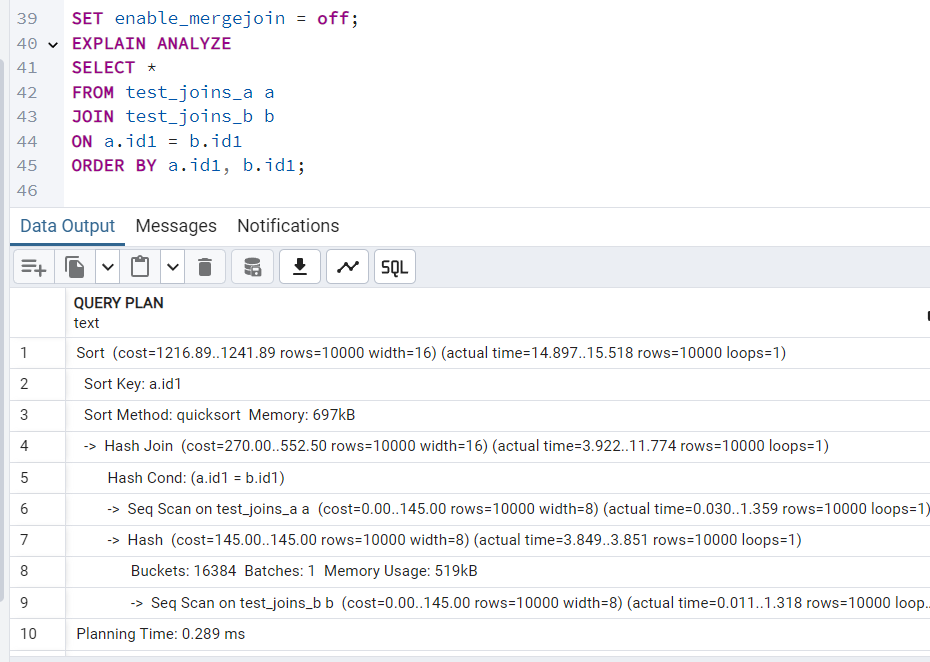
**Task 3**

**To ensure PostgreSQL selects a Merge Join, we:**

* Use an equality condition (=).
* Ensure both tables have an index on id1.
* The ORDER BY ensures the planner recognizes sorted input.

****

Disabling Merge Join

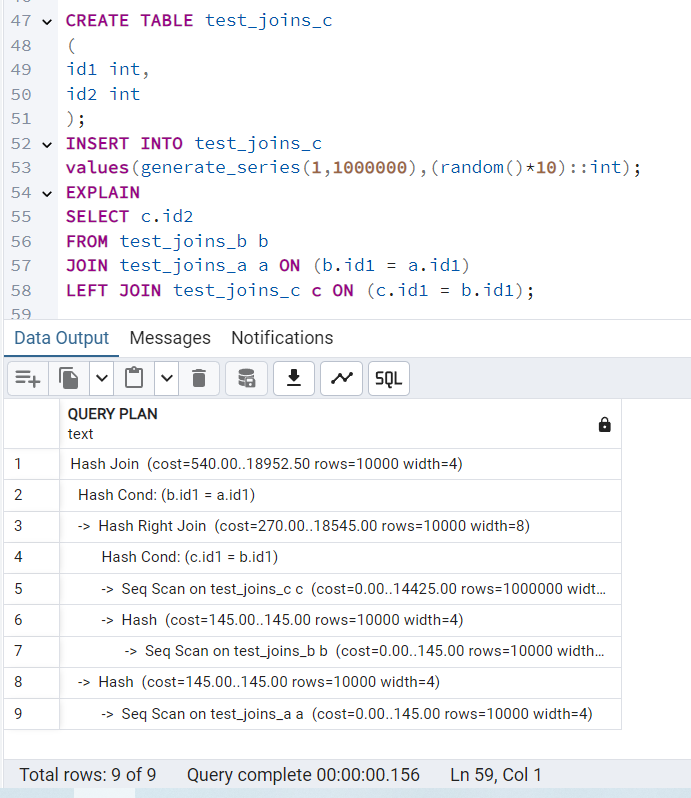
****

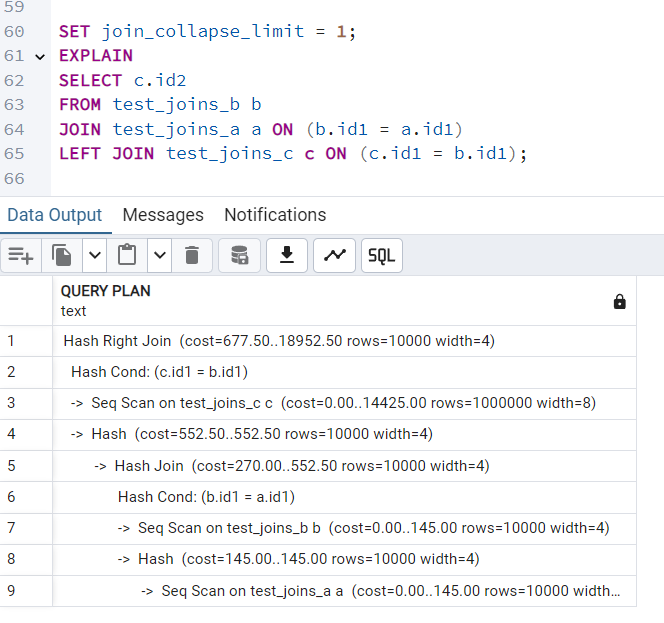
Re-enabling Merge Join

****

**Task 4**

PostgreSQL automatically determines the best join order based on table sizes, indexes, and conditions.

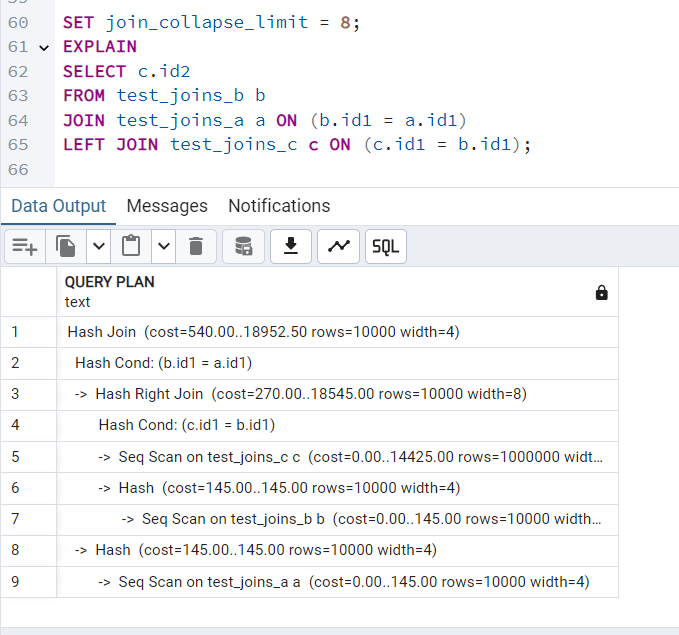
****

****

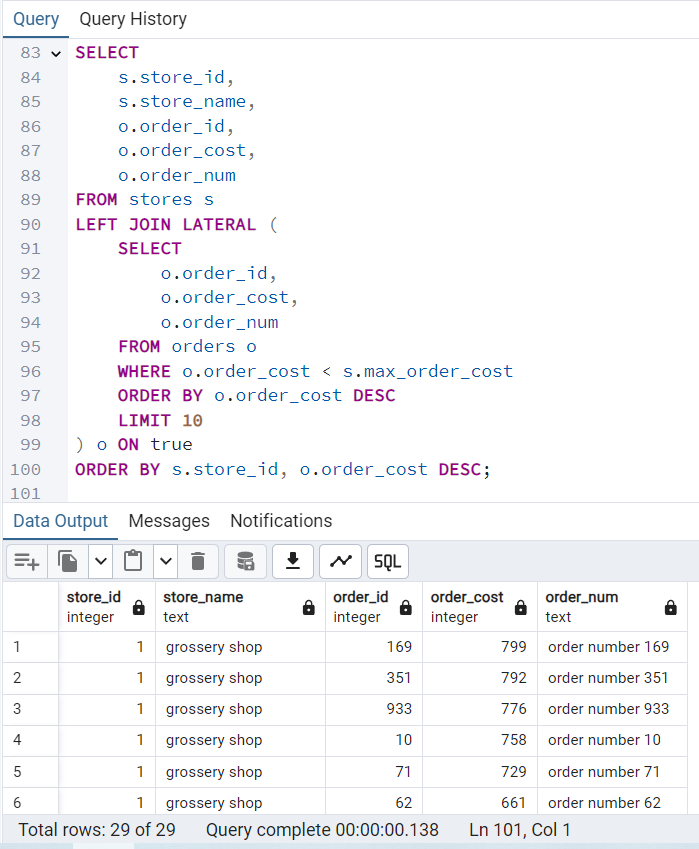
**What Changes?**

* When join\_collapse\_limit = 1, PostgreSQL executes joins in the written order without trying to optimize.
* This might increase query execution time because it disables automatic reordering.
* Instead of optimizing JOIN (A, B) before LEFT JOIN C, it strictly follows the query structure.

Resetting back to 8 restores optimal execution plans.



**Task 5**

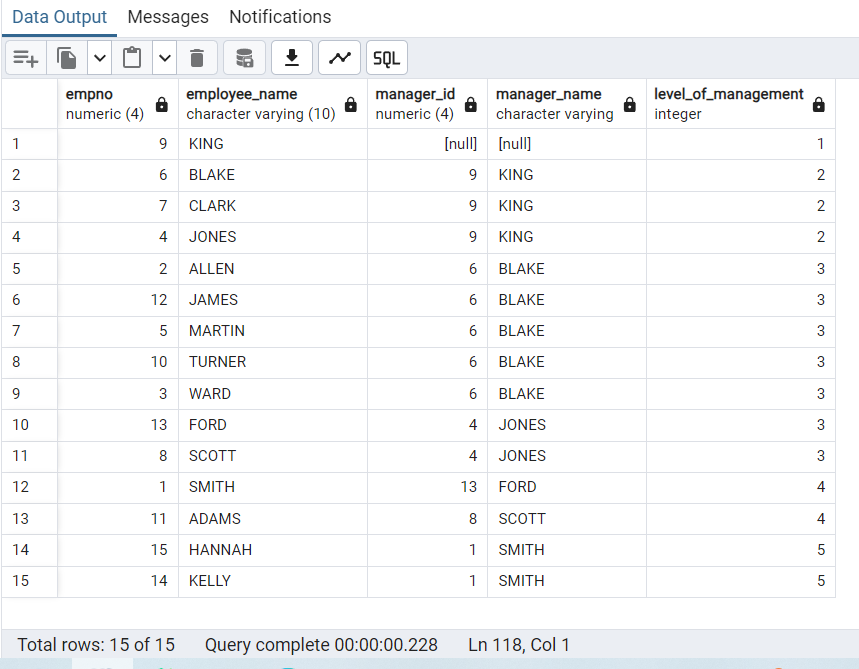
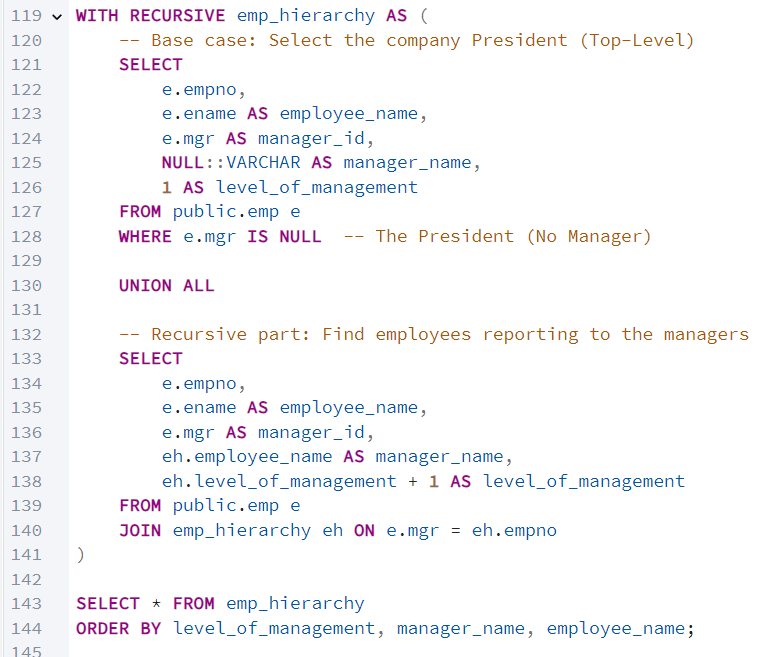
****

A LATERAL JOIN in SQL allows a subquery in the FROM clause to reference columns from preceding tables in the same FROM clause. It's useful when you need to:

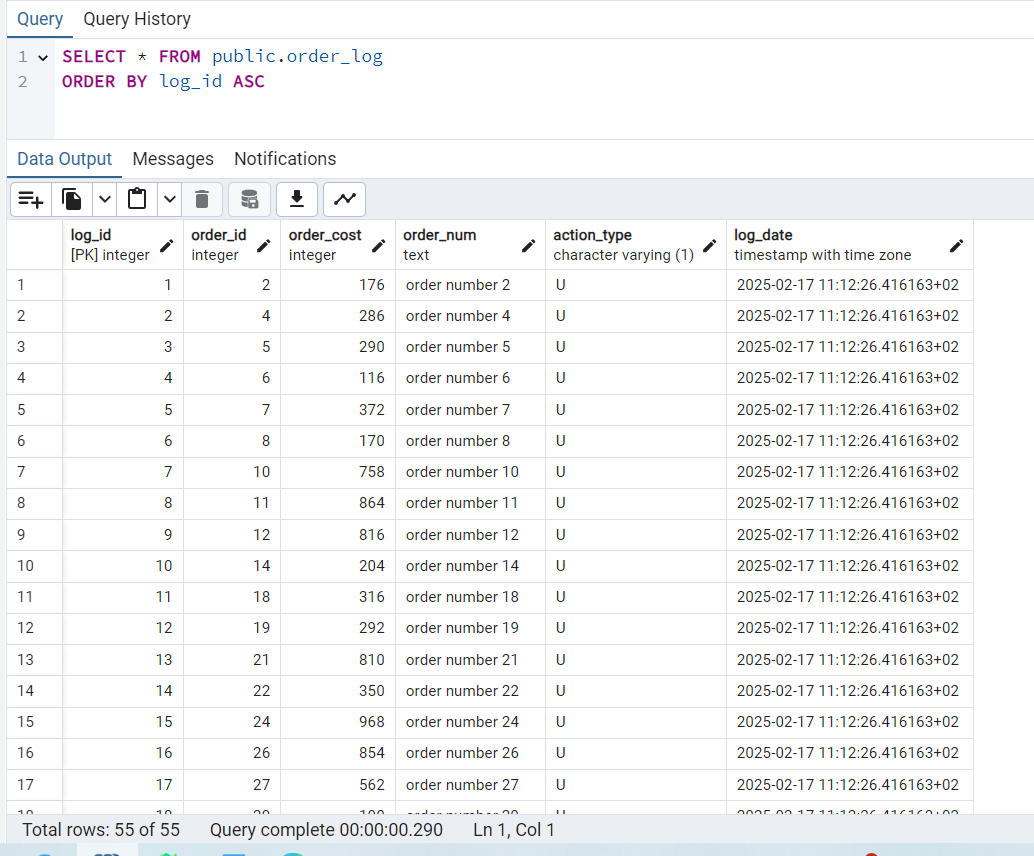
* When a subquery needs values from the outer query
* Fetch top-N results per group.

Alternative Ways to Write the Query without LATERAL JOIN use ROW\_NUMBER() to assign a rank to orders per store and filter only the **top 10 per store.**

**Task 6.**



**Task 7.**

****