1. **Answer the business questions from step 1 and 2 of task 3.8 using CTEs:**

**Average Amount Paid Query as a CTE:**

WITH top\_customers AS (

SELECT c.customer\_id, c.first\_name, c.last\_name, co.country, ct.city, SUM(p.amount) AS total\_amount\_paid

FROM customer AS c

JOIN address AS a ON c.address\_id = a.address\_id

JOIN city AS ct ON a.city\_id = ct.city\_id

JOIN country AS co ON ct.country\_id = co.country\_id

JOIN payment AS p ON c.customer\_id = p.customer\_id

WHERE ct.city IN (

SELECT ct.city

FROM customer AS c

JOIN address AS a ON c.address\_jd = a.address\_id

JOIN city AS ct ON a.city\_id = ct.city\_id

JOIN country AS co ON ct.country\_id = co.country\_id

GROUP BY ct.city

ORDER BY COUNT(c.customer\_id) DESC

LIMIT 10

)

GROUP BY c.customer\_id, c.first\_name, c.last\_name, co.country, ct.city

ORDER BY total\_amount\_paid DESC

LIMIT 5

)

SELECT t.customer\_id, t.first\_name, t.last\_name, t.country, t.city, t.total\_amount\_paid, AVG(t.total\_amount\_paid) AS average\_amount\_paid

FROM top\_customers AS t

GROUP BY t.customer\_id, t.first\_name, t.last\_name, t.country, t.city, t.total\_amount\_paid;

A screenshot of a computer

Description automatically generated

A screenshot of a table

Description automatically generated

In this version, I identified the subquery that retrieves the top 10 cities based on customer count and created a CTE called "top\_customers" for it. Then, I extracted the remaining portion of the original query, which retrieves the customer information and their total amount paid, and created a CTE called "top\_customers" for it. Finally, I modified the main query to reference the "top\_customers" CTE and performed the necessary grouping and aggregation to calculate the average amount paid.

**The top 5 customers are based within each country query as a CTE:**

WITH top\_cities AS (

SELECT ct.city

FROM customer AS c

JOIN address AS a ON c.address\_id = a.address\_id

JOIN city AS ct ON a.city\_id = ct.city\_id

JOIN country AS co ON ct.country\_id = co.country\_id

GROUP BY ct.city

ORDER BY COUNT(c.customer\_id) DESC

LIMIT 10

),

top\_5\_customers AS (

SELECT c.customer\_id, co.country, SUM(p.amount) AS total\_amount\_paid

FROM customer AS c

JOIN address AS a ON c.address\_id = a.address\_id

JOIN city AS ct ON a.city\_id = ct.city\_id

JOIN country AS co ON ct.country\_id = co.country\_id

JOIN payment AS p ON c.customer\_id = p.customer\_id

WHERE ct.city IN (SELECT city FROM top\_cities)

GROUP BY c.customer\_id, co.country

)

SELECT co.country, COUNT(DISTINCT c.customer\_id) AS all\_customer\_count, COUNT(DISTINCT tc.customer\_id) AS top\_customer\_count

FROM customer AS c

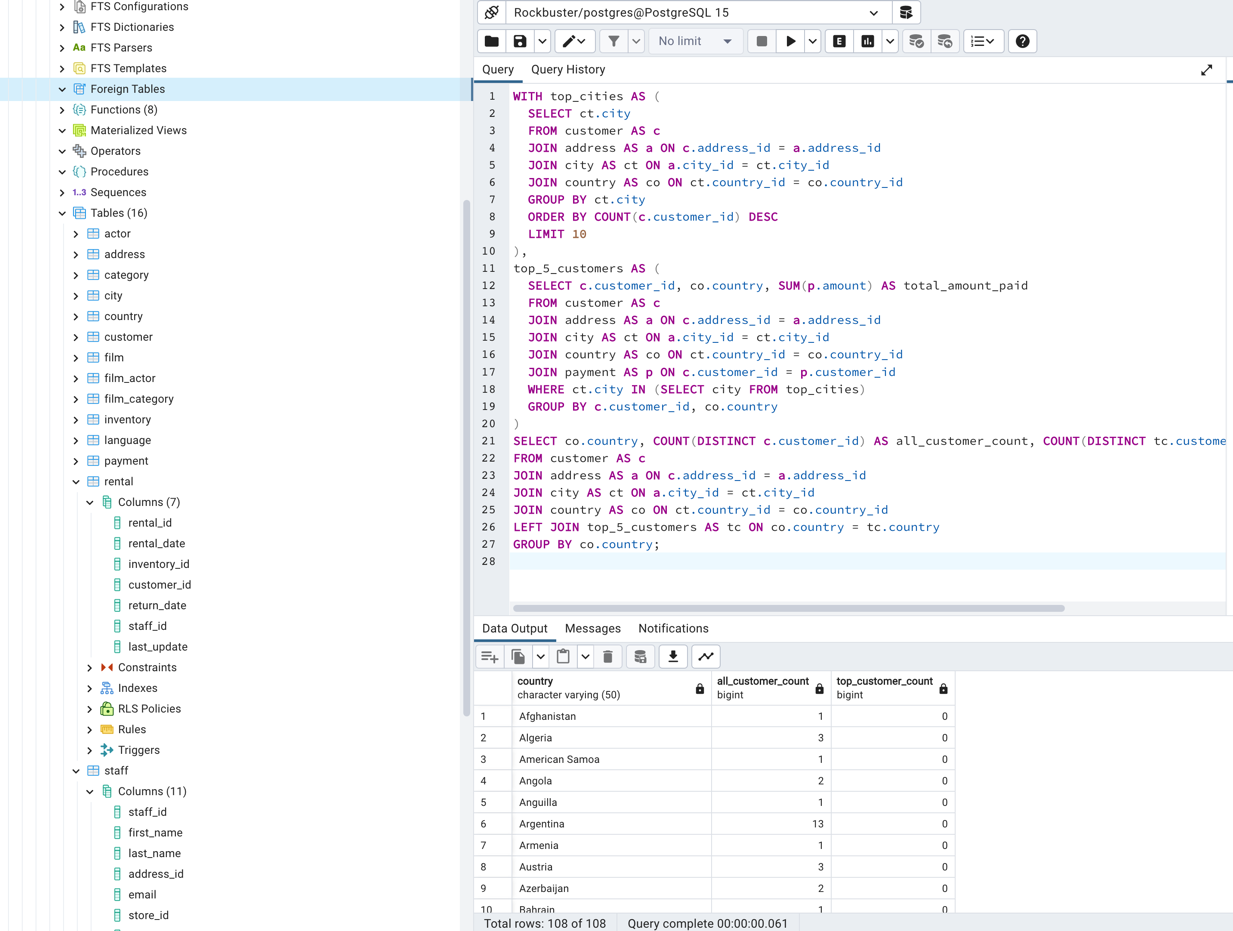
JOIN address AS a ON c.address\_id = a.address\_id

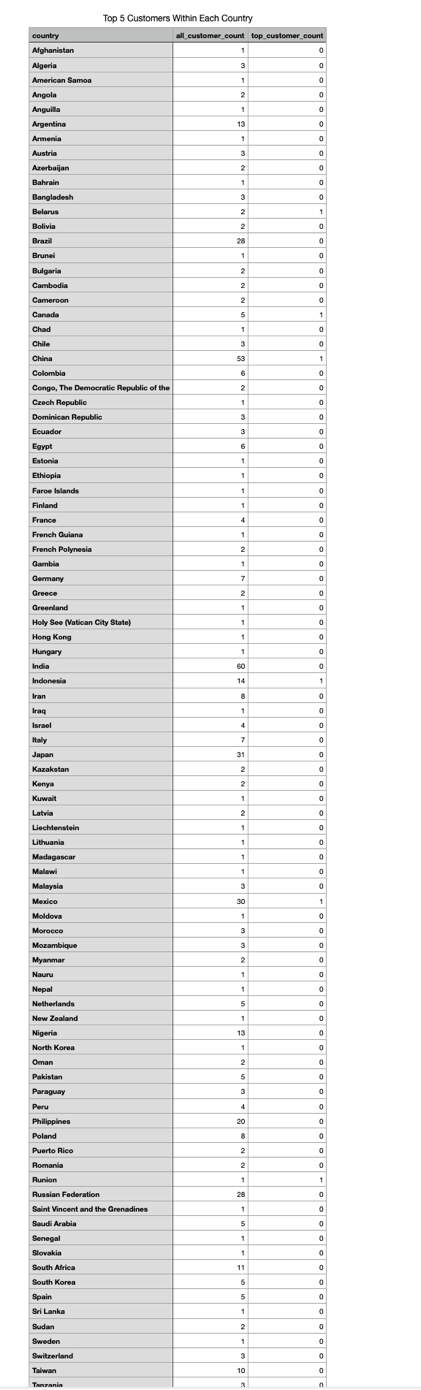
JOIN city AS ct ON a.city\_id = ct.city\_id

JOIN country AS co ON ct.country\_id = co.country\_id

LEFT JOIN top\_5\_customers AS tc ON co.country = tc.country

GROUP BY co.country;



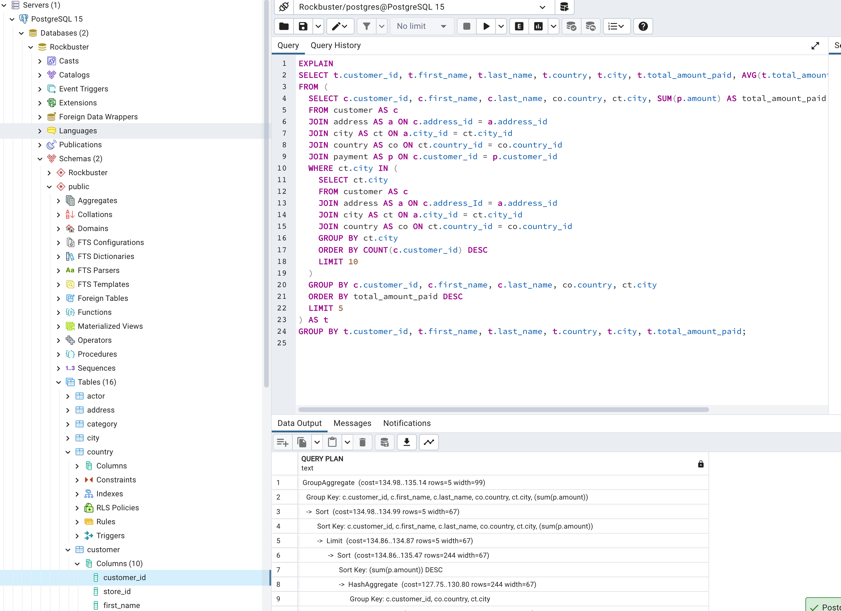


The provided CTE is designed to analyze the distribution of top customers within each country based on their total amount paid. The first CTE, "top\_cities," selects the top 10 cities with the highest customer count. This serves as a filter for subsequent calculations. The second CTE, "top\_5\_customers," joins various tables such as customer, address, city, country, and payment to obtain the total amount paid by each customer within the top cities. Finally, the main query performs a left join between the country table and the top\_5\_customers CTE to determine the customer count for both all customers and the top customers within each country. By grouping the results by country, the query provides insights into the distribution of top customers across different countries.

1. **Compare the performance of your CTEs and subqueries.**
2. *Which will perform better and why?*

* The performance of CTEs and subqueries can vary depending on the specific database system, query optimizer, data size, and the complexity of the queries. It is challenging to make a general statement about which will perform better in all scenarios. CTEs allow for better query optimization by providing a named temporary result set that can be optimized and reused within the query. This can reduce redundant computations and improve query execution time. Additionally, CTEs can enhance code readability and maintainability, making it easier for the optimizer to generate an efficient execution plan. On the other hand, subqueries may require the database system to execute the subquery multiple times, increasing the overall query execution time. Subqueries can also be more difficult for the optimizer to optimize efficiently, especially when dealing with complex nested subqueries.

1. *Compare the costs of all the queries by creating query plans for each one.*



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. *Did the results surprise you?*

* **Total rows: 44 of 44 and in Query complete 00:00:00.055** is incredible to note for such a long-written query. I appreciate being able to witness how fast the system is able to discover answers within a dataset so swiftly and precisely.

1. **Write 1-2 paragraphs on the challenges you faced when replacing your subqueries with CTEs.**

* The challenges when replacing subqueries with CTEs involve balancing performance considerations, understanding the impact on query optimization, and managing the scope and visibility of the CTEs withing the query or across multiple queries. Careful analysis, testing, and consideration of the specific database system and query requirements are necessary to overcome these challenges effectively. CTEs may negatively impact the query execution time making it important to analyze and compare the query plans and the performance of both subqueries and CTEs.