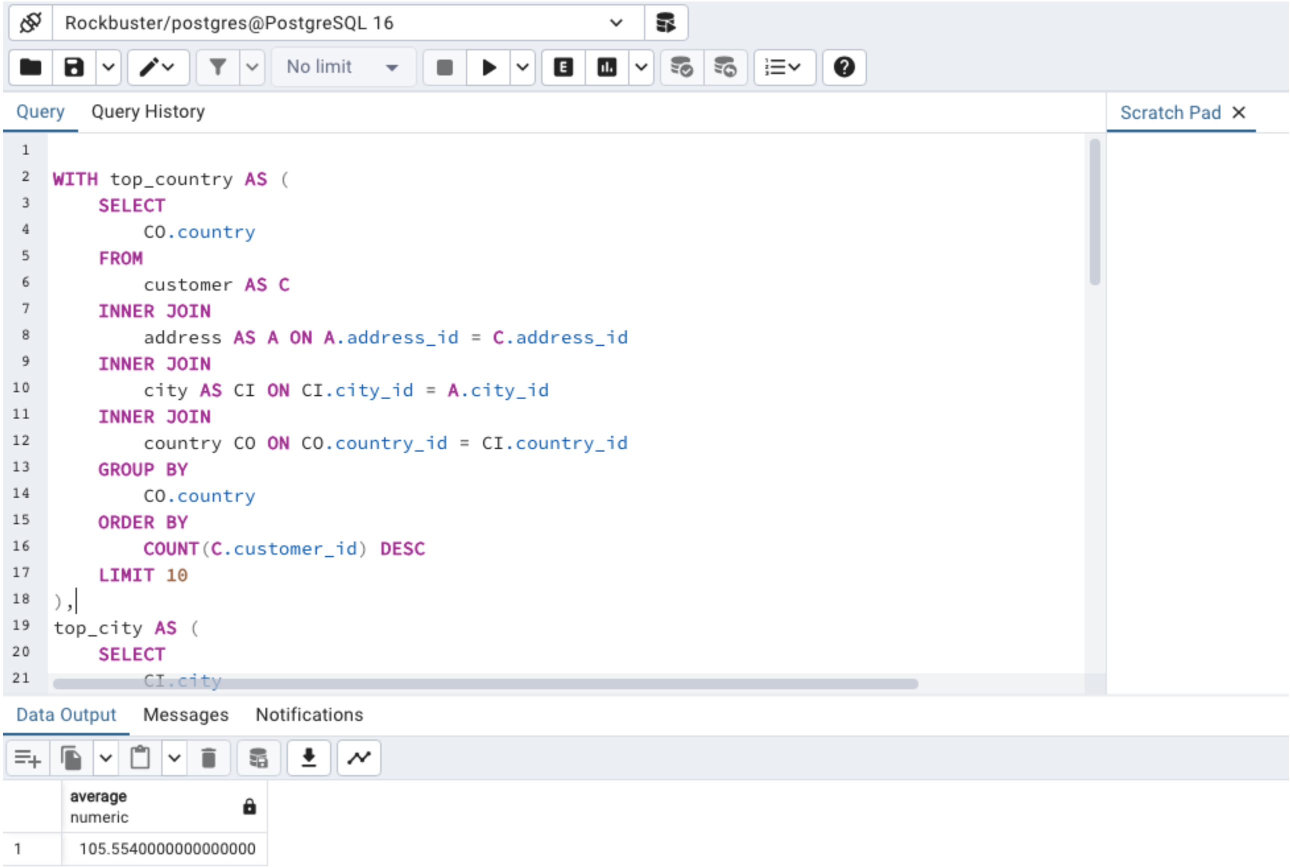
TASK 3.9

Common Table Expressions

STEP 1

```
-- Common Table Expression (CTE) to find the top 10 countries with the most customers
WITH top_country AS (
  SELECT
    CO.country
  FROM
    customer AS C
  INNER JOIN
    address AS A ON A.address_id = C.address_id
  INNER JOIN
    city AS CI ON CI.city_id = A.city_id
  INNER JOIN
    country CO ON CO.country_id = CI.country_id
  GROUP BY
    CO.country
  ORDER BY
    COUNT(C.customer_id) DESC
  LIMIT 10
),
-- Common Table Expression (CTE) to find the top 10 cities within the top 10 countries
top_city AS (
  SELECT
    CI.city
  FROM
    customer AS C
  INNER JOIN
    address AS A ON A.address_id = C.address_id
  INNER JOIN
    city AS CI ON CI.city_id = A.city_id
  INNER JOIN
    country CO ON CO.country_id = CI.country_id
  WHERE
    CO.country IN (SELECT * FROM top_country)
  GROUP BY
    CO.country, CI.city
  ORDER BY
    COUNT(C.customer_id) DESC
  LIMIT 10
),
-- Common Table Expression (CTE) to calculate the total amount paid by the top 5 customers in the top cities
total_amount_paid AS (
  SELECT
    C.customer_id,
    C.first_name,
    C.last_name,
    CO.country,
    CI.city,
    SUM(P.amount) AS total_amount_payment
  FROM
    payment AS P
  INNER JOIN
    customer AS C ON C.customer_id = P.customer_id
  INNER JOIN
    address AS A ON A.address_id = C.address_id
  INNER JOIN
    city AS CI ON CI.city_id = A.city_id
  INNER JOIN
    country CO ON CO.country_id = CI.country_id
  WHERE
    CI.city IN (SELECT * FROM top_city)
  GROUP BY
    C.customer_id, CO.country, CI.city
  ORDER BY
    total_amount_payment DESC
  LIMIT 5
)
-- Final query to calculate the average total amount paid by the top 5 customers in the top cities
SELECT
  AVG(total_amount_payment) AS average
FROM
  total_amount_paid;
```



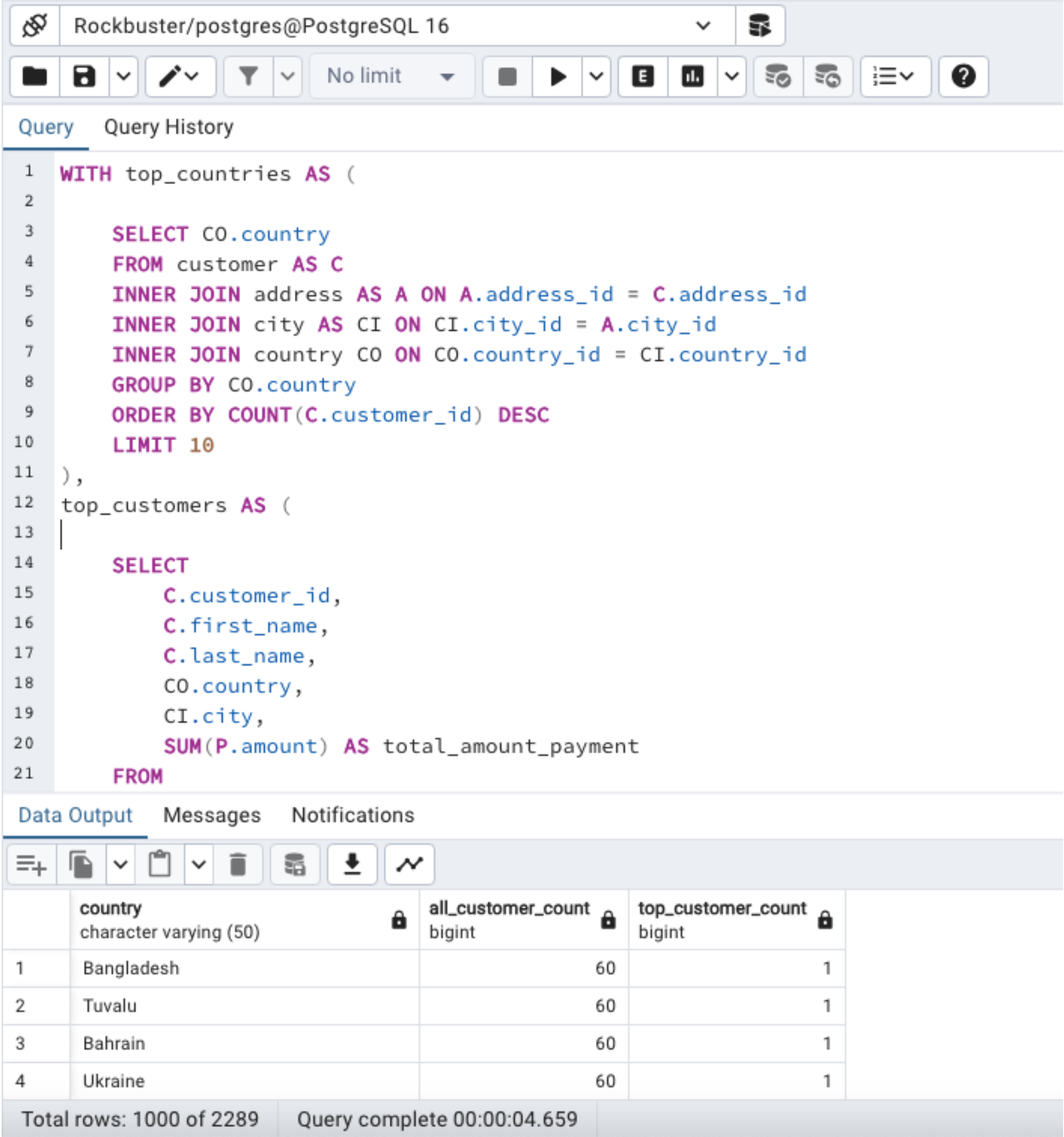
```

WITH top_country AS (
    SELECT
        CO.country
    FROM
        customer AS C
    INNER JOIN
        address AS A ON A.address_id = C.address_id
    INNER JOIN
        city AS CI ON CI.city_id = A.city_id
    INNER JOIN
        country CO ON CO.country_id = CI.country_id
    GROUP BY
        CO.country
    ORDER BY
        COUNT(C.customer_id) DESC
    LIMIT 10
),
top_city AS (
    SELECT
        CI.city
```

```
  average
  numeric
1 105.5540000000000000
```

1.a

```
WITH top_countries AS (
  -- CTE to find the top 10 countries with the most customers
  SELECT CO.country
  FROM customer AS C
  INNER JOIN address AS A ON A.address_id = C.address_id
  INNER JOIN city AS CI ON CI.city_id = A.city_id
  INNER JOIN country CO ON CO.country_id = CI.country_id
  GROUP BY CO.country
  ORDER BY COUNT(C.customer_id) DESC
  LIMIT 10
),
top_customers AS (
  -- CTE to calculate the total amount paid by the top 5 customers in the top cities of top
countries
  SELECT
    C.customer_id,
    C.first_name,
    C.last_name,
    CO.country,
    CI.city,
    SUM(P.amount) AS total_amount_payment
  FROM
```

```
    payment AS P
  INNER JOIN
    customer AS C ON C.customer_id = P.customer_id
  INNER JOIN
    address AS A ON A.address_id = C.address_id
  INNER JOIN
    city AS CI ON CI.city_id = A.city_id
  INNER JOIN
    country CO ON CO.country_id = CI.country_id
  WHERE
    CI.city IN (
      SELECT CI.city
      FROM customer AS C
      INNER JOIN address AS A ON A.address_id = C.address_id
      INNER JOIN city AS CI ON CI.city_id = A.city_id
      INNER JOIN country CO ON CO.country_id = CI.country_id
      WHERE CO.country IN (SELECT * FROM top_countries)
      GROUP BY CO.country, CI.city
      ORDER BY COUNT(C.customer_id) DESC
      LIMIT 10
    )
  GROUP BY
    C.customer_id, CO.country, CI.city
  ORDER BY
    total_amount_payment DESC
  LIMIT 5
)
-- Final query to get the count of all customers and top customers in each country
SELECT
  CO.country,
  COUNT(DISTINCT C.customer_id) AS all_customer_count,
  COUNT(DISTINCT top_customers.customer_id) AS top_customer_count
FROM
  customer AS C
INNER JOIN
  address AS A ON A.address_id = C.address_id
INNER JOIN
  city AS CI ON CI.city_id = A.city_id
INNER JOIN
  country CO ON CO.country_id = CI.country_id
LEFT JOIN
  top_customers ON top_customers.country = CO.country
GROUP BY
  CO.country
ORDER BY
  top_customer_count DESC;
```



```
WITH top_countries AS (

    SELECT CO.country
    FROM customer AS C
    INNER JOIN address AS A ON A.address_id = C.address_id
    INNER JOIN city AS CI ON CI.city_id = A.city_id
    INNER JOIN country CO ON CO.country_id = CI.country_id
    GROUP BY CO.country
    ORDER BY COUNT(C.customer_id) DESC
    LIMIT 10
),
top_customers AS (

    SELECT
        C.customer_id,
        C.first_name,
        C.last_name,
        CO.country,
        CI.city,
        SUM(P.amount) AS total_amount_payment
    FROM
```

	country character varying (50)	all_customer_count bigint	top_customer_count bigint
1	Bangladesh	60	1
2	Tuvalu	60	1
3	Bahrain	60	1
4	Ukraine	60	1

Total rows: 1000 of 2289 Query complete 00:00:04.659

First, I identified the main components of the original query, including finding the top countries, calculating total payments by top customers, and counting all customers per country. Then, I transformed each component into a Common Table Expression (CTE), breaking down the logic into more manageable parts. Finally, I joined these CTEs together to retrieve the desired information, ensuring readability and maintainability.

STEP 2

Comparing the performance of the original query and the one using CTEs depends on various factors such as database size, indexing, and server resources. Generally, the query with CTEs may perform better due to its modular and structured nature, allowing for better query optimization and reuse of intermediate results. However, in some cases, the performance difference may be negligible or even favor the original query, especially if the CTEs introduce unnecessary processing overhead.
To compare the costs of the queries, I would use the EXPLAIN command to generate query plans for each one, which provides an estimated cost based on factors like table scans, joins, and index usage. Then, I would run both queries in pgAdmin 4 to obtain their execution times in milliseconds.

Let’s see:

QUERY 1 from 3.8:

```
Aggregate (cost=166.06..166.07 rows=1 width=32)
Query complete 00:00:00.219
```

QUERY 1 as CTEs:

```
Aggregate (cost=166.06..166.07 rows=1 width=32)
Query complete 00:00:00.258
```

QUERY 2 from 3.8:

```
"Sort (cost=270.33..270.60 rows=109 width=25)"
Query complete 00:00:00.468
```

QUERY 2 as CTEs:

```
Sort (cost=270.24..270.51 rows=109 width=25)
Query complete 00:00:04.659
```

I find the results somewhat unexpected. The decision to use either Common Table Expressions (CTEs) or subqueries should rely on performance testing and thorough analysis tailored to individual cases.

STEP 3:

Personally,transitioning from using subqueries to employing Common Table Expressions (CTEs) presented a few challenges. Initially, understanding the syntax and structure of CTEs required some effort, especially grasping how to define and reference them within the query. Additionally, translating the logic from subqueries to CTEs while maintaining the integrity and functionality of the original query posed a bit of a learning curve. Ensuring that the CTEs were properly scoped and sequenced to produce the desired results took some trial and error. Moreover, managing the nesting of CTEs within the main query while keeping the code organized and readable proved to be another hurdle. However, through experimentation and practice, I gradually became more comfortable with leveraging CTEs as a powerful tool for improving query readability, modularity, and performance. Overall, while there were challenges in the transition process, the experience provided valuable insights into the versatility and efficiency of CTEs in SQL query optimization.