Answers 3.9.

Step 1) queries from steps 1 and 2 of task 3.8 as CTE

1a: Average total amount paid by the Top 5 customers

```
WITH top_countries AS (
    SELECT D.country
    FROM customer A
    INNER JOIN address B ON A.address_id = B.address_id
    INNER JOIN city C ON B.city_id = C.city_id
    INNER JOIN country D ON C.country_id = D.country_id
    GROUP BY D.country
    ORDER BY COUNT(A.customer_id) DESC
    LIMIT 10
),
top_cities AS (
    SELECT C.city
    FROM customer A
    INNER JOIN address B ON A.address_id = B.address_id
    INNER JOIN city C ON B.city_id = C.city_id
    INNER JOIN country D ON C.country_id = D.country_id
    WHERE D.country IN (SELECT * FROM top_countries)
    GROUP BY D.country, C.city
    ORDER BY COUNT(A.customer_id) DESC
    LIMIT 10
),
top_customers AS (
    SELECT SUM(E.amount) AS total_payment
    FROM customer A
    INNER JOIN address B ON A.address_id = B.address_id
    INNER JOIN city C ON B.city_id = C.city_id
    INNER JOIN country D ON C.country_id = D.country_id
    INNER JOIN payment E ON A.customer_id = E.customer_id
    WHERE C.city IN (SELECT * FROM top_cities)
    GROUP BY A.first_name, A.last_name, C.city, D.country
    ORDER BY total_payment DESC
    LIMIT 5
)
SELECT AVG(total_payment) AS avg
FROM top_customers;
```

Output Average: 105.55

1b:

Query:

```
WITH top_countries AS (
    SELECT D.country
    FROM customer A
    INNER JOIN address B ON A.address_id = B.address_id
```

```
    INNER JOIN city C ON B.city_id = C.city_id
    INNER JOIN country D ON C.country_id = D.country_id
    GROUP BY D.country
    ORDER BY COUNT(A.customer_id) DESC
    LIMIT 10
),
top_cities AS (
    SELECT C.city
    FROM customer A
    INNER JOIN address B ON A.address_id = B.address_id
    INNER JOIN city C ON B.city_id = C.city_id
    INNER JOIN country D ON C.country_id = D.country_id
    WHERE D.country IN (SELECT * FROM top_countries)
    GROUP BY D.country, C.city
    ORDER BY COUNT(A.customer_id) DESC
    LIMIT 10
),
top_customers AS (
    SELECT SUM(E.amount) AS total_payment,
           D.country
    FROM customer A
    INNER JOIN address B ON A.address_id = B.address_id
    INNER JOIN city C ON B.city_id = C.city_id
    INNER JOIN country D ON C.country_id = D.country_id
    INNER JOIN payment E ON A.customer_id = E.customer_id
    WHERE C.city IN (SELECT * FROM top_cities)
    GROUP BY A.first_name, A.last_name, C.city, D.country
    ORDER BY total_payment DESC
    LIMIT 5
)
SELECT D.country,
       COUNT(DISTINCT A.customer_id) AS all_customer_count,
       COUNT(DISTINCT top_customers) AS top_customer_count
FROM customer A
INNER JOIN address B ON A.address_id = B.address_id
INNER JOIN city C ON B.city_id = C.city_id
INNER JOIN country D ON C.country_id = D.country_id
LEFT JOIN top_customers ON D.country = top_customers.country
GROUP BY D.country
ORDER BY top_customer_count DESC;
```

Output :

country	all_customer_count	top_customer_count
Mexico	30	1
India	60	1
China	53	1
United States	36	1
Japan	31	1
Argentina	13	0
Armenia	1	0

....

1c) Explanation:

- To calculate the average payment of the top 5 customers, I started by identifying the top 10 countries with the most customers using the top_countries CTE, grouping by country and ordering by customer count. Next, I created a second CTE, top_cities, to focus on the top 10 cities within those countries by filtering for cities located in the identified top countries and grouping by both country and city. Finally, I used the top_customers CTE to determine the top 5 customers by total payments within those cities. This allowed me to calculate the average payment of the top 5 customers in the final query, ensuring a logical, hierarchical approach to narrowing down the dataset.
- To find the number of customers and top customers by country, I reused the CTEs top_countries and top_cities to define the scope of analysis, limiting the results to top-performing regions and cities. Then, I created the top_customers CTE to calculate the payments of the top 5 customers within these cities. In the main query, I joined the overall customer data with the top_customers CTE using a LEFT JOIN on the country column. This allowed me to calculate the total number of customers (all_customer_count) and the number of top customers (top_customer_count) for each country. The results were grouped by country and sorted to provide a clear breakdown of customer distributions across regions.

Step 2)

a)

The CTE approach performs better for these specific queries, as seen in the lower execution times for both Query 1 (120 ms vs. 138 ms) and Query 2 (189 ms vs. 148 ms). This result suggests that PostgreSQL optimized the CTEs more efficiently by reusing intermediate results and avoiding redundant recalculations. Additionally, CTEs can simplify query logic and make it easier for the database to materialize intermediate results effectively. While subqueries are inline and do not require materialization, the complexity of nested subqueries can introduce additional processing overhead, which is evident in this case.

b-c) Comparing the costs and speeds

Comparison of the performance of CTEs and subqueries:

Query #	**Approach**	**Cost**	**Estimated Time**
Query 1 (Step 1)	CTE	166.68	120 ms
Query 1 (Step 1)	Subquery	166.68	138 ms
Query 2 (Step 2)	CTE	271.21	189 ms
Query 2 (Step 2)	Subquery	271.21	148 ms

d) The results were somewhat unexpected, as subqueries are generally believed to perform faster for simpler operations due to their inline execution. However, in these queries, the CTEs outperformed subqueries in terms of execution time. This could be attributed to PostgreSQL's ability to materialize and optimize CTEs efficiently for reuse, especially in complex queries where intermediate results are utilized multiple times. This highlights the importance of testing both approaches to determine the optimal one for a given query, as theoretical assumptions do not always align with real-world performance outcomes.

Step 3)

- One of the primary challenges in replacing subqueries with CTEs was restructuring the logic to separate the query into distinct steps while maintaining its overall functionality. Subqueries are often inline and embedded directly within the main query, which allows them to be referenced where needed. When transitioning to CTEs, it was necessary to extract these subqueries, define them independently using the WITH clause, and ensure they were properly referenced in the main query. This process required careful attention to detail, especially when dealing with nested subqueries, as each intermediate step had to be clearly defined and logically connected.
- Another challenge was managing the additional complexity introduced by naming and structuring multiple CTEs. While CTEs enhance query readability for complex datasets, ensuring that each CTE's alias and column names were consistent with the original subquery was critical to avoiding errors. Additionally, understanding the

potential impact of materialization (temporary storage of CTE results) on query performance required close monitoring of the execution plan to ensure that the CTE-based approach did not inadvertently increase computational overhead. Despite these challenges, using CTEs ultimately made the queries more modular and easier to debug and maintain.