Step 1: Answer the business questions from steps 1 and 2 of task 3.8 using CTEs

- WITH total_amount_paid_cte (customer_id,full_name,country,city,total_payment)

```
AS      (SELECT
        c.customer_id,
        CONCAT(c.first_name,' ',c.last_name) AS full_name,
        co.country AS country,
        ci.city AS city,
        SUM(p.amount) AS total_payment
        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 AS co ON co.country_id = ci.country_id
        INNER JOIN payment AS p ON p.customer_id = c.customer_id
        WHERE ci.city IN ('Aurora', 'Atlixco', 'Xintai', 'Adoni', 'Dhule (Dhulia)','Kurashiki',
'Pingxiang', 'Sivas', 'Celaya', 'So Leopoldo')
        GROUP BY 1,2,3,4--,5,6
        ORDER BY total_payment DESC
        LIMIT 5)
SELECT
ROUND(AVG(total_payment), 2) AS average_amount_paid
FROM total_amount_paid_cte
```

	average_amount_paid numeric
1	107.35

- WITH TopCustomers_cte AS (

```
SELECT
    c.customer_id,
    CONCAT(c.first_name, ' ', c.last_name) AS full_name,
    co.country AS country,
    ci.city AS city,
    SUM(p.amount) AS total_payment
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 AS co ON co.country_id = ci.country_id
    INNER JOIN payment AS p ON p.customer_id = c.customer_id
    WHERE ci.city IN ('Aurora', 'Atlixco', 'Xintai', 'Adoni', 'Dhule (Dhulia)',
                      'Kurashiki', 'Pingxiang', 'Sivas', 'Celaya', 'So Leopoldo')
    GROUP BY c.customer_id, co.country, ci.city
    ORDER BY total_payment DESC
    LIMIT 5
),
TopCustomersCount_cte AS (
    SELECT
        abc.country AS abccountry,
        COUNT(DISTINCT abc.customer_id) AS top_customer_count
    FROM TopCustomers_cte AS abc
    GROUP BY abc.country
)
SELECT
    co.country AS country,
    COUNT(DISTINCT c.customer_id) AS all_customer_count,
    COALESCE(MAX(abcd.top_customer_count), 0) 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 AS co ON co.country_id = ci.country_id
LEFT JOIN TopCustomersCount_cte AS abcd ON abcd.abccountry = co.country
GROUP BY co.country
ORDER BY all_customer_count DESC
```

	country character varying (50)	all_customer_count bigint	top_customer_count bigint
1	India	60	1
2	China	53	0
3	United States	36	1
4	Japan	31	0
5	Mexico	30	2
6	Brazil	28	0
7	Russian Federation	28	0
8	Philippines	20	0
9	Turkey	15	1

Write 2 to 3 sentences explaining how you approached this step, for example, what you did first, second, and so on.

To approach this step, I first reviewed the queries from steps 1 and 2 and identified the subqueries that could be replaced with Common Table Expressions (CTEs). I then rewrote each query using CTEs to improve readability and structure, ensuring that the logic and results remained consistent. Finally, I tested the CTE-based queries to confirm their correctness and copied the outputs for inclusion in the answers document.

Step 2: Compare the performance of your CTEs and subqueries.

Which approach do you think will perform better and why?

CTEs are generally preferable due to their readability, reusability, and structured approach. They are particularly effective in scenarios where intermediate results need to be referenced multiple times. Subqueries, on the other hand, might perform slightly better in simpler cases where modularity and reusability are not as critical.

In this scenario, the performance difference is negligible because of the dataset size and query structure. However, for better maintainability and scalability, the CTE-based approach is more suitable.

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

- CTE

```
QUERY PLAN
text
Sort  (cost=168.07..168.34 rows=109 width=25)
  Sort Key: (count(DISTINCT c.customer_id)) DESC
  ->  GroupAggregate  (cost=157.30..164.38 rows=109 width=25)
        Group Key: co.country
        ->  Sort  (cost=157.30..158.80 rows=599 width=21)
              Sort Key: co.country, c.customer_id
              ->  Hash Left Join  (cost=108.25..129.67 rows=599 width=21)
                    Hash Cond: ((co.country)::text = (abcd.abccountry)::text)
                    ->  Hash Join  (cost=43.52..63.30 rows=599 width=13)
                          Hash Cond: (ci.country_id = co.country_id)
                          ->  Hash Join  (cost=40.07..58.22 rows=599 width=6)
                                Hash Cond: (a.city_id = ci.city_id)
                                ->  Hash Join  (cost=21.57..38.14 rows=599 width=6)
                                      Hash Cond: (c.address_id = a.address_id)
                                      ->  Seq Scan on customer c  (cost=0.00..14.99 rows=599 width=6)
                                      ->  Hash  (cost=14.03..14.03 rows=603 width=6)
                                            ->  Seq Scan on address a  (cost=0.00..14.03 rows=603 width=6)
                                ->  Hash  (cost=11.00..11.00 rows=600 width=6)
                                      ->  Seq Scan on city ci  (cost=0.00..11.00 rows=600 width=6)
                          ->  Hash  (cost=2.09..2.09 rows=109 width=13)
                                ->  Seq Scan on country co  (cost=0.00..2.09 rows=109 width=13)
                    ->  Hash  (cost=64.67..64.67 rows=5 width=17)
                          ->  Subquery Scan on abcd  (cost=64.53..64.67 rows=5 width=17)
                                ->  GroupAggregate  (cost=64.53..64.62 rows=5 width=17)
                                      Group Key: abc.country
                                      ->  Sort  (cost=64.53..64.55 rows=5 width=13)
                                            Sort Key: abc.country, abc.customer_id
                                            ->  Subquery Scan on abc  (cost=64.41..64.48 rows=5 width=13)
                                                  ->  Limit  (cost=64.41..64.43 rows=5 width=86)
                                                        ->  Sort  (cost=64.41..65.02 rows=244 width=86)
                                                              Sort Key: (sum(p.amount)) DESC
                                                              ->  HashAggregate  (cost=57.31..60.36 rows=244 width=86)
                                                                    Group Key: c_1.customer_id, co_1.country, ci_1.city
                                                                    ->  Nested Loop  (cost=18.16..54.87 rows=244 width=28)
                                                                          ->  Hash Join  (cost=17.88..37.14 rows=10 width=22)
                                                                                Hash Cond: (ci_1.country_id = co_1.country_id)
                                                                                ->  Nested Loop  (cost=14.43..33.66 rows=10 width=15)
                                                                                      ->  Hash Join  (cost=14.15..29.77 rows=10 width=15)
                                                                                            Hash Cond: (a_1.city_id = ci_1.city_id)
                                                                                            ->  Seq Scan on address a_1  (cost=0.00..14.03 rows=603 width=6)
                                                                                            ->  Hash  (cost=14.03..14.03 rows=10 width=15)
                                                                                                  ->  Seq Scan on city ci_1  (cost=0.03..14.03 rows=10 width=15)
                                                                                                        Filter: ((city)::text = ANY ('{Aurora,Atlixco,Xintai,Adoni,'Dhule (Dhulia)',Kurashiki,Pingxiang,Sivas,Celaya,'So Leopoldo'}'::text[]))
                                                                                      ->  Index Scan using idx_fk_address_id on customer c_1  (cost=0.28..0.38 rows=1 width=6)
Total rows: 49   Query complete 00:00:00.032                                                                                                    CRLF   Ln 103, Col 8
```

- Subquery

```
QUERY PLAN
text
Sort  (cost=169.90..170.17 rows=109 width=25)
  Sort Key: (count(DISTINCT c.customer_id)) DESC
  ->  GroupAggregate  (cost=159.13..166.21 rows=109 width=25)
        Group Key: co.country
        ->  Sort  (cost=159.13..160.63 rows=599 width=21)
              Sort Key: co.country, c.customer_id
              ->  Hash Left Join  (cost=110.08..131.50 rows=599 width=21)
                    Hash Cond: ((co.country)::text = (abcd.abccountry)::text)
                    ->  Hash Join  (cost=43.52..63.30 rows=599 width=13)
                          Hash Cond: (ci.country_id = co.country_id)
                          ->  Hash Join  (cost=40.07..58.22 rows=599 width=6)
                                Hash Cond: (a.city_id = ci.city_id)
                                ->  Hash Join  (cost=21.57..38.14 rows=599 width=6)
                                      Hash Cond: (c.address_id = a.address_id)
                                      ->  Seq Scan on customer c  (cost=0.00..14.99 rows=599 width=6)
                                      ->  Hash  (cost=14.03..14.03 rows=603 width=6)
                                            ->  Seq Scan on address a  (cost=0.00..14.03 rows=603 width=6)
                                ->  Hash  (cost=11.00..11.00 rows=600 width=6)
                                      ->  Seq Scan on city ci  (cost=0.00..11.00 rows=600 width=6)
                          ->  Hash  (cost=2.09..2.09 rows=109 width=13)
                                ->  Seq Scan on country co  (cost=0.00..2.09 rows=109 width=13)
                    ->  Hash  (cost=66.50..66.50 rows=5 width=17)
                          ->  Subquery Scan on abcd  (cost=66.36..66.50 rows=5 width=17)
                                ->  GroupAggregate  (cost=66.36..66.45 rows=5 width=17)
                                      Group Key: abc.country
                                      ->  Sort  (cost=66.36..66.38 rows=5 width=13)
                                            Sort Key: abc.country, abc.customer_id
                                            ->  Subquery Scan on abc  (cost=66.24..66.31 rows=5 width=13)
                                                  ->  Limit  (cost=66.24..66.26 rows=5 width=86)
                                                        ->  Sort  (cost=66.24..66.85 rows=244 width=86)
                                                              Sort Key: (sum(p.amount)) DESC
                                                              ->  HashAggregate  (cost=58.53..62.19 rows=244 width=86)
                                                                    Group Key: c_1.customer_id, concat(c_1.first_name, ' ', c_1.last_name), co_1.country, ci_1.city
                                                                    ->  Nested Loop  (cost=18.16..55.48 rows=244 width=60)
                                                                          ->  Hash Join  (cost=17.88..37.14 rows=10 width=35)
                                                                                Hash Cond: (ci_1.country_id = co_1.country_id)
                                                                                ->  Nested Loop  (cost=14.43..33.66 rows=10 width=28)
                                                                                      ->  Hash Join  (cost=14.15..29.77 rows=10 width=15)
                                                                                            Hash Cond: (a_1.city_id = ci_1.city_id)
                                                                                            ->  Seq Scan on address a_1  (cost=0.00..14.03 rows=603 width=6)
                                                                                            ->  Hash  (cost=14.03..14.03 rows=10 width=15)
                                                                                                  ->  Seq Scan on city ci_1  (cost=0.03..14.03 rows=10 width=15)
                                                                                                        Filter: ((city)::text = ANY ('{Aurora,Atlixco,Xintai,Adoni,"Dhule (Dhulia)",Kurashiki,Pingxiang,Sivas,Celaya,"So Leopoldo"}'::text[]))
                                                                                      ->  Index Scan using idx_fk_address_id on customer c_1  (cost=0.28..0.38 rows=1 width=19)
Total rows: 49   Query complete 00:00:00.034                                                                                                                    CRLF   Ln 45, Col 10
```

Did the results surprise you? Write a few sentences to explain your answer

Yes, the results did surprise me. I expected the CTE approach to perform significantly better due to its structured design and reusability. However, the execution times for both queries were almost identical, showing that the database engine optimizes both CTEs and subqueries efficiently for this dataset size and complexity. This highlights that the performance difference is often minimal in simpler scenarios, even though I still find CTEs more beneficial for readability and maintainability.

Step 3:

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

When replacing subqueries with CTEs, the main challenge was ensuring that the rewritten queries produced the same results. Subqueries are directly embedded in the main query, so breaking them into separate CTEs required careful attention to the logic and how each part connected. It took extra effort to test the changes and confirm everything worked correctly.

Another difficulty was understanding how CTEs are processed compared to subqueries. While CTEs made the queries easier to read and manage, I had to test them to ensure they performed well. Overall, it was a bit tricky at first, but it helped me better understand how to structure queries.