Answers 3.9

STEP 1

QUERY #1

Query | Query History

```sql
WITH top_cities AS
(SELECT city.city
FROM customer
INNER JOIN address ON customer.address_id = address.address_id
INNER JOIN city ON address.city_id = city.city_id
INNER JOIN country ON city.country_id = country.country_id
GROUP BY city.city, country.country
ORDER BY COUNT(customer.customer_id) DESC
LIMIT 10),
top_customers AS
(SELECT customer.customer_id, customer.first_name, customer.last_name, city.city, country.country, SUM(payment.amount) AS total_amount_paid
FROM payment
INNER JOIN customer ON payment.customer_id = customer.customer_id
INNER JOIN address ON customer.address_id = address.address_id
INNER JOIN city ON address.city_id = city.city_id
INNER JOIN country ON city.country_id = country.country_id
WHERE city.city IN (SELECT city FROM top_cities)
GROUP BY customer.customer_id, customer.first_name, customer.last_name, city.city, country.country
ORDER BY total_amount_paid DESC
LIMIT 5)
SELECT AVG(top_customers.total_amount_paid) AS average
FROM top_customers
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1

	average numeric
1	108.5500000000000000

CODE:

```
WITH top_cities AS

(SELECT city.city

FROM customer

INNER JOIN address ON customer.address_id = address.address_id

INNER JOIN city ON address.city_id = city.city_id

INNER JOIN country ON city.country_id = country.country_id

GROUP BY city.city, country.country

ORDER BY COUNT(customer.customer_id) DESC

LIMIT 10),

top_customers AS

(SELECT customer.customer_id, customer.first_name, customer.last_name, city.city, country.country,
SUM(payment.amount) AS total_amount_paid

FROM payment

INNER JOIN customer ON payment.customer_id = customer.customer_id

INNER JOIN address ON customer.address_id = address.address_id

INNER JOIN city ON address.city_id = city.city_id

INNER JOIN country ON city.country_id = country.country_id

WHERE city.city IN (SELECT city FROM top_cities)

GROUP BY customer.customer_id, customer.first_name, customer.last_name, city.city, country.country

ORDER BY total_amount_paid DESC

LIMIT 5)

SELECT AVG(top_customers.total_amount_paid) AS average

FROM top_customers
```

EXPLANATION:

First CTE (top_cities):

- Identifies the top 10 cities that have the highest number of customers.
- Uses COUNT(customer.customer_id) to count the number of customers in each city.
- Sorts cities in descending order.
- Uses LIMIT 10 to return only the top 10 cities

Second CTE (top_customers):

- Identifies the top 5 customers who have made the highest total payments.
- Only considers customers from the top 10 cities.
- Uses SUM(payment.amount) to calculate each customer's total payments.
- Groups by customer details (customer_id, first_name, last_name, city, country).
- Sorts by total_amount_paid DESC.
- Uses LIMIT 5 to keep only the top 5 customers.

Main query:

- Takes the total_amount_paid from the top_customers CTE.
- Computes the average total payment among these top 5 customers using AVG(top_customers.total_amount_paid).

QUERY #2:

Query | Query History

```
WITH top_cities AS
(SELECT city.city
FROM customer
INNER JOIN address ON customer.address_id = address.address_id
INNER JOIN city ON address.city_id = city.city_id
INNER JOIN country ON city.country_id = country.country_id
GROUP BY city.city, country.country
ORDER BY COUNT(customer.customer_id) DESC
LIMIT 10),
top_customers AS
(SELECT customer.customer_id, customer.first_name, customer.last_name, city.city, country.country, SUM(payment.amount) AS total_amount_paid
FROM payment
INNER JOIN customer ON payment.customer_id = customer.customer_id
INNER JOIN address ON customer.address_id = address.address_id
INNER JOIN city ON address.city_id = city.city_id
INNER JOIN country ON city.country_id = country.country_id
WHERE city.city IN (SELECT city FROM top_cities)
GROUP BY customer.customer_id, customer.first_name, customer.last_name, city.city, country.country
ORDER BY total_amount_paid DESC
LIMIT 5)
SELECT country.country,
COUNT(DISTINCT customer.customer_id) AS all_customer_count,
COUNT(DISTINCT top_customers.customer_id) AS top_customer_count
FROM customer
INNER JOIN address ON customer.address_id = address.address_id
INNER JOIN city ON address.city_id = city.city_id
INNER JOIN country ON city.country_id = country.country_id
LEFT JOIN top_customers ON customer.customer_id = top_customers.customer_id
GROUP BY country.country
ORDER BY top_customer_count DESC
```

OUTPUT:

Data Output | Messages | Notifications

Showing rows: 1 to 108

	country character varying (50)	all_customer_count bigint	top_customer_count bigint
1	Turkey	15	1
2	United Kingdom	9	1
3	United States	36	1
4	India	60	1
5	Zambia	1	1
6	Argentina	13	0
7	Armenia	1	0
8	Austria	3	0
9	Azerbaijan	2	0
10	Bahrain	1	0
11	Bangladesh	3	0
12	Belarus	2	0
13	Bolivia	2	0
14	Brazil	28	0
15	Brunei	1	0
16	Bulgaria	2	0
17	Cambodia	2	0
18	Cameroon	2	0
19	Canada	5	0
20	Chad	1	0
21	Chile	3	0

Total rows: 108 | Query complete 00:00:00.153

CODE:

```
WITH top_cities AS
(SELECT city.city
FROM customer
INNER JOIN address ON customer.address_id = address.address_id
INNER JOIN city ON address.city_id = city.city_id
INNER JOIN country ON city.country_id = country.country_id
GROUP BY city.city, country.country
ORDER BY COUNT(customer.customer_id) DESC
LIMIT 10),
top_customers AS
(SELECT customer.customer_id, customer.first_name, customer.last_name, city.city, country.country,
SUM(payment.amount) AS total_amount_paid
FROM payment
INNER JOIN customer ON payment.customer_id = customer.customer_id
INNER JOIN address ON customer.address_id = address.address_id
INNER JOIN city ON address.city_id = city.city_id
INNER JOIN country ON city.country_id = country.country_id
WHERE city.city IN (SELECT city FROM top_cities)
GROUP BY customer.customer_id, customer.first_name, customer.last_name, city.city, country.country
ORDER BY total_amount_paid DESC
LIMIT 5)
SELECT country.country,
COUNT(DISTINCT customer.customer_id) AS all_customer_count,
COUNT(DISTINCT top_customers.customer_id) AS top_customer_count
FROM customer
INNER JOIN address ON customer.address_id = address.address_id
INNER JOIN city ON address.city_id = city.city_id
INNER JOIN country ON city.country_id = country.country_id
LEFT JOIN top_customers ON customer.customer_id = top_customers.customer_id
GROUP BY country.country
ORDER BY top_customer_count DESC
```

EXPLANATION:

First CTE (top_cities):

- Same as in Query 1.
- Finds the top 10 cities based on customer count.
- Used later to filter customers.

Second CTE (top_customers):

- Same as in Query 1.
- Identifies top 5 high paying customers from the top 10 cities.

Main Query:

- Counts all customers per country using COUNT(DISTINCT customer.customer_id).
- Counts top 5 high paying customers per country using COUNT(DISTINCT top_customers.customer_id).
- Uses LEFT JOIN top_customers so that customers who are not in top_customers will have NULL values.
- Groups results by country.country.
- Orders by top_customer_count DESC to see which countries have the most high paying customers.

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

QUERY #1

CTE ANALYSIS:

Query | Query History

```
1  EXPLAIN ANALYZE
2  WITH top_cities AS
3  (SELECT city.city
4  FROM customer
5  INNER JOIN address ON customer.address_id = address.address_id
6  INNER JOIN city ON address.city_id = city.city_id
7  INNER JOIN country ON city.country_id = country.country_id
8  GROUP BY city.city, country.country
9  ORDER BY COUNT(customer.customer_id) DESC
10 LIMIT 10),
```

Data Output | Messages | Notifications

Showing rows: 1 to 51 | Page No: 1 of 1

	QUERY PLAN text
40	Buckets: 1024 Batches: 1 Memory Usage: 14kB
41	-> Seq Scan on country country_1 (cost=0.00..2.09 rows=109 width=13) (actual time=0.011..0.024 rows=109 loops=1)
42	-> Index Scan using idx_fk_city_id on address (cost=0.28..0.37 rows=1 width=6) (actual time=0.003..0.003 rows=1 loops=11)
43	Index Cond: (city_id = city.city_id)
44	-> Index Scan using idx_fk_address_id on customer (cost=0.28..0.38 rows=1 width=19) (actual time=0.002..0.003 rows=1 loops=12)
45	Index Cond: (address_id = address.address_id)
46	-> Index Scan using country_pkey on country (cost=0.14..0.18 rows=1 width=13) (actual time=0.002..0.002 rows=1 loops=12)
47	Index Cond: (country_id = city.country_id)
48	-> Index Scan using idx_fk_customer_id on payment (cost=0.29..1.53 rows=24 width=8) (actual time=0.003..0.008 rows=23 loops=12)
49	Index Cond: (customer_id = customer.customer_id)
50	Planning Time: 3.920 ms
51	Execution Time: 2.368 ms

Planning time: 3.920 ms

Execution time: 2.368 ms

QUERY #1

SUBQUERY ANALYSIS:

Query | Query History

```
1  EXPLAIN ANALYZE
2  SELECT AVG(total_amount_paid.total_amount_paid) AS average
3  FROM
4  (SELECT customer.customer_id, customer.first_name, customer.last_name, city.city, country.country, SUM(payment.amount) AS total_amount_paid
5  FROM payment
6  INNER JOIN customer ON payment.customer_id = customer.customer_id
7  INNER JOIN address ON customer.address_id = address.address_id
8  INNER JOIN city ON address.city_id = city.city_id
9  INNER JOIN country ON city.country_id = country.country_id
```

Data Output | Messages | Notifications

Showing rows: 1 to 51 | Page No: 1

	QUERY PLAN text
39	-> Hash (cost=2.09..2.09 rows=109 width=13) (actual time=0.049..0.049 rows=109 loops=1)
40	Buckets: 1024 Batches: 1 Memory Usage: 14kB
41	-> Seq Scan on country country_1 (cost=0.00..2.09 rows=109 width=13) (actual time=0.011..0.024 rows=109 loops=1)
42	-> Index Scan using idx_fk_city_id on address (cost=0.28..0.37 rows=1 width=6) (actual time=0.003..0.003 rows=1 loops=11)
43	Index Cond: (city_id = city.city_id)
44	-> Index Scan using idx_fk_address_id on customer (cost=0.28..0.38 rows=1 width=19) (actual time=0.002..0.003 rows=1 loops=12)
45	Index Cond: (address_id = address.address_id)
46	-> Index Scan using country_pkey on country (cost=0.14..0.18 rows=1 width=13) (actual time=0.002..0.002 rows=1 loops=12)
47	Index Cond: (country_id = city.country_id)
48	-> Index Scan using idx_fk_customer_id on payment (cost=0.29..1.53 rows=24 width=8) (actual time=0.003..0.008 rows=23 loops=12)
49	Index Cond: (customer_id = customer.customer_id)
50	Planning Time: 3.327 ms
51	Execution Time: 2.438 ms

Total rows: 51 | Query complete 00:00:00.077

Planning time: 3.327 ms

Execution time 2.438 ms

QUERY #2

CTE ANALYSIS:

Query | Query History

```
1  EXPLAIN ANALYZE
2  WITH top_cities AS
3  (SELECT city.city
4  FROM customer
5  INNER JOIN address ON customer.address_id = address.address_id
6  INNER JOIN city ON address.city_id = city.city_id
7  INNER JOIN country ON city.country_id = country.country_id
8  GROUP BY city.city, country.country
```

Data Output | Messages | Notifications

Showing rows: 1 to 79 | Page No: 1

	QUERY PLAN text
66	-> Seq Scan on city city_2 (cost=0.00..11.00 rows=600 width=15) (actual time=0.005..0.061 rows=600 loops=1)
67	-> Hash (cost=2.09..2.09 rows=109 width=13) (actual time=0.031..0.031 rows=109 loops=1)
68	Buckets: 1024 Batches: 1 Memory Usage: 14kB
69	-> Seq Scan on country country_2 (cost=0.00..2.09 rows=109 width=13) (actual time=0.006..0.015 rows=109 loops=1)
70	-> Index Scan using idx_fk_city_id on address address_1 (cost=0.28..0.37 rows=1 width=6) (actual time=0.002..0.002 rows=1 loops=11)
71	Index Cond: (city_id = city_1.city_id)
72	-> Index Scan using idx_fk_address_id on customer customer_1 (cost=0.28..0.38 rows=1 width=19) (actual time=0.001..0.002 rows=1 loops=12)
73	Index Cond: (address_id = address_1.address_id)
74	-> Index Scan using country_pkey on country country_1 (cost=0.14..0.18 rows=1 width=13) (actual time=0.001..0.001 rows=1 loops=12)
75	Index Cond: (country_id = city_1.country_id)
76	-> Index Scan using idx_fk_customer_id on payment (cost=0.29..1.53 rows=24 width=8) (actual time=0.004..0.010 rows=23 loops=12)
77	Index Cond: (customer_id = customer_1.customer_id)
78	Planning Time: 3.615 ms
79	Execution Time: 3.909 ms

Total rows: 79 | Query complete 00:00:00.081

Planning time: 3.615 ms

Execution Time: 3.909 ms

QUERY #2

SUBQUERY ANALYSIS

Query | Query History

```
1  EXPLAIN ANALYZE
2  SELECT country.country, COUNT(DISTINCT customer.customer_id) AS all_customer_count, COUNT(DISTINCT top_5_customers.customer_id) AS top_custo
3  FROM customer
4  INNER JOIN address ON customer.address_id = address.address_id
```

Data Output | Messages | Notifications

Showing rows: 1 to 79 | Page No:

	QUERY PLAN text
63	-> Seq Scan on address address_2 (cost=0.00..14.03 rows=603 width=6) (actual time=0.005..0.083 rows=603 loops=1)
64	-> Hash (cost=11.00..11.00 rows=600 width=15) (actual time=0.177..0.178 rows=600 loops=1)
65	Buckets: 1024 Batches: 1 Memory Usage: 37kB
66	-> Seq Scan on city city_2 (cost=0.00..11.00 rows=600 width=15) (actual time=0.005..0.075 rows=600 loops=1)
67	-> Hash (cost=2.09..2.09 rows=109 width=13) (actual time=0.035..0.036 rows=109 loops=1)
68	Buckets: 1024 Batches: 1 Memory Usage: 14kB
69	-> Seq Scan on country country_2 (cost=0.00..2.09 rows=109 width=13) (actual time=0.007..0.017 rows=109 loops=1)
70	-> Index Scan using idx_fk_city_id on address address_1 (cost=0.28..0.37 rows=1 width=6) (actual time=0.002..0.003 rows=1 loops=11)
71	Index Cond: (city_id = city_1.city_id)
72	-> Index Scan using idx_fk_address_id on customer customer_1 (cost=0.28..0.38 rows=1 width=19) (actual time=0.002..0.002 rows=1 loops=12)
73	Index Cond: (address_id = address_1.address_id)
74	-> Index Scan using country_pkey on country country_1 (cost=0.14..0.18 rows=1 width=13) (actual time=0.001..0.001 rows=1 loops=12)
75	Index Cond: (country_id = city_1.country_id)
76	-> Index Scan using idx_fk_customer_id on payment (cost=0.29..1.53 rows=24 width=8) (actual time=0.004..0.011 rows=23 loops=12)
77	Index Cond: (customer_id = customer_1.customer_id)
78	Planning Time: 4.367 ms
79	Execution Time: 4.733 ms

Total rows: 79 | Query complete 00:00:00.064

Planning time: 4.367 ms

Execution time: 4.733 ms

To evaluate performance, we run both the original subquery-based queries and the CTE based versions using EXPLAIN ANALYZE.

Expected Result for Subqueries:

- PostgreSQL executes the subquery multiple times, increasing execution cost.
- Potential optimization issues if the same subquery is executed repeatedly.

Expected result for CTEs:

- CTEs prevent repeated execution of the same logic, reducing query cost.
- CTE result is stored temporarily, improving efficiency.
- Performance improves in queries that reuse the same data.

COMPARISON:

	QUERY #1		**QUERY #2**	
	Planning	**Execution**	**Planning**	**Execution**
CTE	3.920 ms	**2.368 ms**	3.615 ms	**3.909 ms**
SUBQUERY	3.327 ms	2.438 ms	4.367 ms	4.733 ms

Which Approach is Better?

CTEs generally perform better when the same subquery logic is used multiple times, because PostgreSQL can evaluate them once and reuse the result.

Subqueries might be faster in simpler cases, but for complex queries, CTEs improve both readability and performance.

CTEs are the best choice in this case, as they allow reusing query results without recalculating them multiple times.

STEP 3: Challenges faced when replacing your subqueries with CTEs.

1. One of the main challenges I faced when replacing subqueries with CTEs was restructuring the query logic while maintaining accuracy. With subqueries, everything was nested, and the dependencies were clear within each section. However, when using CTEs, I had to break the logic into separate, reusable steps while ensuring that each CTE correctly fed into the next part of the query.

2. Another challenge was performance optimization. While CTEs improve readability, they can sometimes lead to performance issues if not handled properly. I had to consider whether the CTEs would be recomputed multiple times and if using WITH would be beneficial. Additionally, adjusting joins, especially ensuring that the final counts in Query 2 remained correct, required extra attention, as changing from subqueries to CTEs sometimes altered how NULL values were handled.