

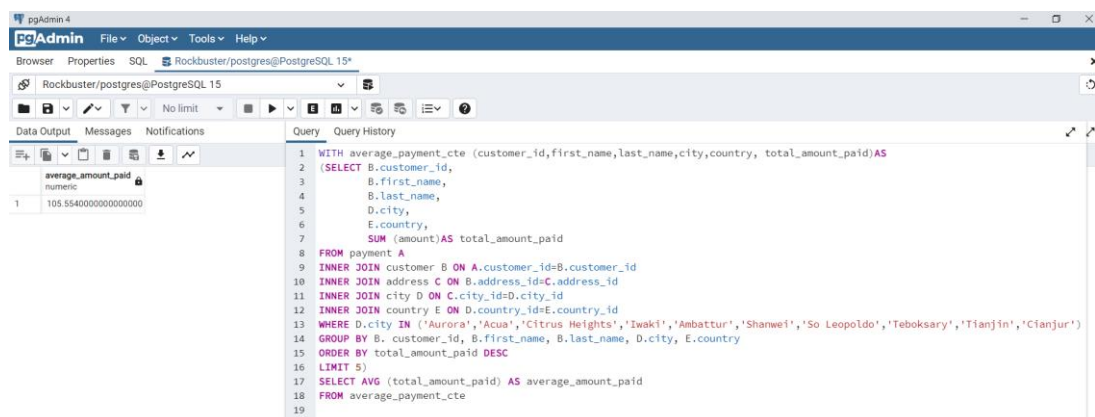
Task 9. Common Table Expressions

Answers 3.9

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

- CTE (converted from subquery on step 1): **Objective: find the average amount paid by the top 5 customers**

```
WITH average_payment_cte
(customer_id,first_name,last_name,city,country, total_amount_paid)AS
(SELECT B.customer_id,
      B.first_name,
      B.last_name,
      D.city,
      E.country,
      SUM (amount)AS total_amount_paid
FROM payment A
INNER JOIN customer B ON A.customer_id=B.customer_id
INNER JOIN address C ON B.address_id=C.address_id
INNER JOIN city D ON C.city_id=D.city_id
INNER JOIN country E ON D.country_id=E.country_id
WHERE D.city IN ('Aurora','Acua','Citrus
Heights','Iwaki','Ambattur','Shanwei','So
Leopoldo','Teboksary','Tianjin','Cianjur')
GROUP BY B. customer_id, B.first_name, B.last_name, D.city, E.country
ORDER BY total_amount_paid DESC
LIMIT 5)
SELECT AVG (total_amount_paid) AS average_amount_paid
FROM average_payment_cte
```



- CTE (converted from subquery on step 2): **Objective: Find out how many of the top 5 customers are based within each country.**

```

WITH top_customer_count_cte (customer_id, first_name, last_name, city, country,
total_amount_paid)AS
  (SELECT B.customer_id,
    B.first_name,
      B.last_name,
      D.city,
      E.country,
      SUM (amount)AS total_amount_paid
    FROM payment A
    INNER JOIN customer B ON A.customer_id=B.customer_id
    INNER JOIN address C ON B.address_id=C.address_id
    INNER JOIN city D ON C.city_id=D.city_id
    INNER JOIN country E ON D.country_id=E.country_id
    WHERE D.city IN ('Aurora','Acua','Citrus Heights','Twaki','Ambattur','Shanwei','So
Leopoldo','Teboksary','Tianjin','Cianjur')
    GROUP BY B. customer_id, B.first_name, B.last_name, D.city, E.country
    ORDER BY total_amount_paid DESC
    LIMIT 5),
  customer_count_cte AS
  (SELECT DISTINCT D.country,
    COUNT(DISTINCT A.customer_id)AS all_customer_count,
    COUNT (DISTINCT D.country) 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
    GROUP BY D.country)
  SELECT DISTINCT D.country,
    COUNT(DISTINCT A.customer_id)AS all_customer_count,
    COUNT (top_customer_count_cte.customer_id) 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_customer_count_cte ON D.country=top_customer_count_cte.country
    GROUP BY D.country
    ORDER BY top_customer_count DESC
    LIMIT 5

```

country	all_customer_count	top_customer_count
India	60	60
China	53	53
United States	36	36
Japan	31	31
Mexico	30	30

```

1 WITH top_customer_count_cte (customer_id, first_name, last_name, city, country, total_amount_paid) AS
2 (SELECT B.customer_id,
3  B.first_name,
4  B.last_name,
5  D.city,
6  E.country,
7  SUM (amount) AS total_amount_paid
8  FROM payment A
9  INNER JOIN customer B ON A.customer_id=B.customer_id
10 INNER JOIN address C ON B.address_id=C.address_id
11 INNER JOIN city D ON C.city_id=D.city_id
12 INNER JOIN country E ON D.country_id=E.country_id
13 WHERE D.city IN ('Aurora', 'Acua', 'Citrus Heights', 'Iwaki', 'Ambattur', 'Shanwei', 'So Leopoldo', 'Teboksary', 'Tianjin')
14 GROUP BY B.customer_id, B.first_name, B.last_name, D.city, E.country
15 ORDER BY total_amount_paid DESC
16 LIMIT 5),
17 customer_count_cte AS
18 (SELECT DISTINCT D.country,
19  COUNT (DISTINCT A.customer_id) AS all_customer_count,
20  COUNT (DISTINCT D.country) AS top_customer_count
21 FROM customer A
22 INNER JOIN address B ON A.address_id = B.address_id
23 INNER JOIN city C ON B.city_id = C.city_id
24 INNER JOIN country D ON C.country_id = D.country_id
25 GROUP BY D.country)
26 SELECT DISTINCT D.country,
27  COUNT (DISTINCT A.customer_id) AS all_customer_count,
28  COUNT (top_customer_count_cte.customer_id) AS top_customer_count
29 FROM customer A

```

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

The first query was relatively easy. I first copy pasted the query from step 1 and added a CTE syntax to it, using the previous sub-query as the CTE, then followed by the main statement.

The second query took me quite a long time to get to.

First, I copy pasted the subquery from step 2/task 3.8, then I identified the part that should go into my CTE, however, I realized I would need more than one CTE, to reflect the whole query. By doing some research online, I realized we can have two CTE after WITH (the first references the second), so I used two CTE.

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

1. Which approach do you think will perform better and why?

Despite the challenges I'm facing with query two, I find the CTEs work better, particularly in the case of query two, because the code is easier to interpret.

2. Compare the costs of all the queries by creating query plans for each one.
3. The EXPLAIN command gives you an *estimated* cost. To find out the actual speed of your queries, run them in pgAdmin 4. After each query has been run, a pop-up window will display its speed in milliseconds.

Explain Query 1/sub-query:

QUERY PLAN	Query
1 Aggregate (cost=64.45..64.46 rows=1 width=32)	1 EXPLAIN SELECT AVG (total_amount_paid) AS average
2 → Limit (cost=64.37..64.39 rows=5 width=67)	2 FROM
3 → Sort (cost=64.37..64.98 rows=243 width=67)	3 (SELECT B.customer_id,
4 Sort Key: (sum(a.amount)) DESC	4 B.first_name,
5 → HashAggregate (cost=57.30..60.34 rows=243 width=67)	5 B.last_name,
6 Group Key: b.customer_id, d.city, e.country	6 D.city,
7 → Nested Loop (cost=18.16..54.87 rows=243 width=41)	7 E.country,
8 → Hash Join (cost=17.88..37.14 rows=10 width=35)	8 SUM (amount) AS total_amount_paid
9 Hash Cond: (d.country_id = e.country_id)	9 FROM payment A
10 → Nested Loop (cost=14.43..33.66 rows=10 width=28)	10 INNER JOIN customer B ON A.customer_id=B.customer_id
11 → Hash Join (cost=14.15..29.77 rows=10 width=15)	11 INNER JOIN address C ON B.address_id=C.address_id
12 Hash Cond: (c.city_id = d.city_id)	12 INNER JOIN city D ON C.city_id=D.city_id
13 → Seq Scan on address c (cost=0.00..14.03 rows=603 width=6)	13 INNER JOIN country E ON D.country_id=E.country_id
14 → Hash (cost=14.03..14.03 rows=10 width=15)	14 WHERE D.city IN ('Aurora','Acua','Citrus Heights','Iwakii','Ambattur','Shanwei','So Leopoldo','Teboksary',
15 → Seq Scan on city d (cost=0.03..14.03 rows=10 width=15)	15 GROUP BY B. customer_id, B.first_name, B.last_name, D.city, E.country
16 Filter: ((city)::text = ANY ((Aurora,Acua,Citrus Heights,Iwakii,Ambattur,Shanwei,So Leopoldo,Teboksary)::text))	16 ORDER BY total_amount_paid DESC
17 → Index Scan using idx_fk_address_id on customer b (cost=0.2..0.2 rows=1 width=6)	17 LIMIT 5) AS total_amount_paid;
18 Index Cond: (address_id = c.address_id)	18
19 → Hash (cost=2.09..2.09 rows=109 width=13)	19
20 → Seq Scan on country e (cost=0.00..2.09 rows=109 width=13)	20
21 → Index Scan using idx_fk_customer_id on payment a (cost=0.2..0.2 rows=1 width=6)	21
	22

Cost: "Aggregate (cost=64.45..64.46 rows=1 width=32)"

The speed was 45 miliseconds

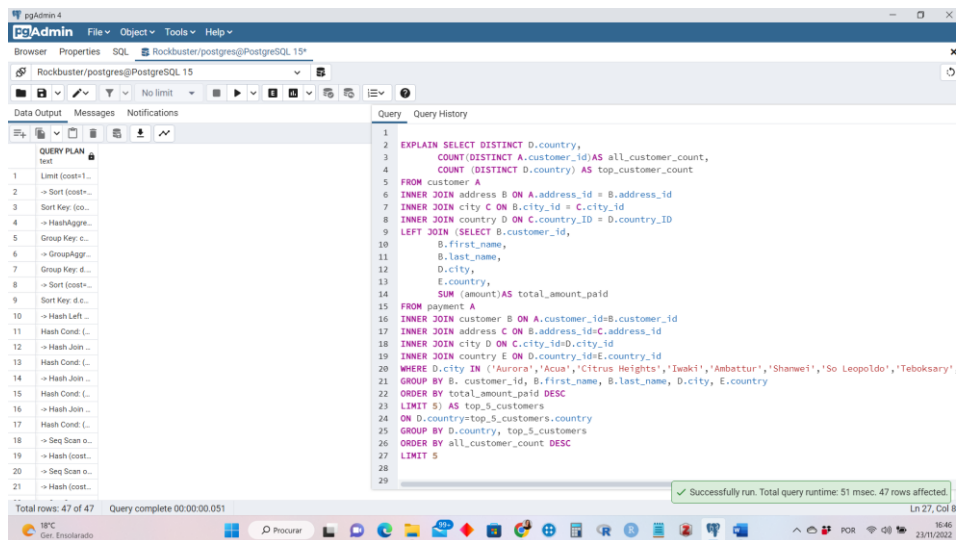
Explain Query 1/CTE:

QUERY PLAN	Query
1 Aggregate (c...	1 EXPLAIN WITH average_payment_cte (customer_id,first_name,last_name,city,country, total_amount_paid)AS
2 → Limit (cost...	2 (SELECT B.customer_id,
3 → Sort (cost...	3 B.first_name,
4 Sort Key: (su...	4 B.last_name,
5 → HashAggre...	5 D.city,
6 Group Key: b...	6 E.country,
7 → Nested Lo...	7 SUM (amount)AS total_amount_paid
8 → Hash Join...	8 FROM payment A
9 → Hash Join...	9 INNER JOIN customer B ON A.customer_id=B.customer_id
10 → Hash Join...	10 INNER JOIN address C ON B.address_id=C.address_id
11 → Hash Join...	11 INNER JOIN city D ON C.city_id=D.city_id
12 → Hash Join...	12 INNER JOIN country E ON D.country_id=E.country_id
13 → Hash Cond...	13 WHERE D.city IN ('Aurora','Acua','Citrus Heights','Iwakii','Ambattur','Shanwei','So Leopoldo','Teboksary',
14 → Seq Scan o...	14 GROUP BY B. customer_id, B.first_name, B.last_name, D.city, E.country
15 → Seq Scan o...	15 ORDER BY total_amount_paid DESC
16 Filter: ((city)...	16 LIMIT 5)
17 → Index Scan...	17 SELECT AVG (total_amount_paid) AS average_amount_paid
18 Index Cond: (...)	18 FROM average_payment_cte
19 → Hash (cost...	19
20 → Seq Scan o...	20

Its speed was 55 miliseconds

Cost: "Aggregate (cost=64.45..64.46 rows=1 width=32)"

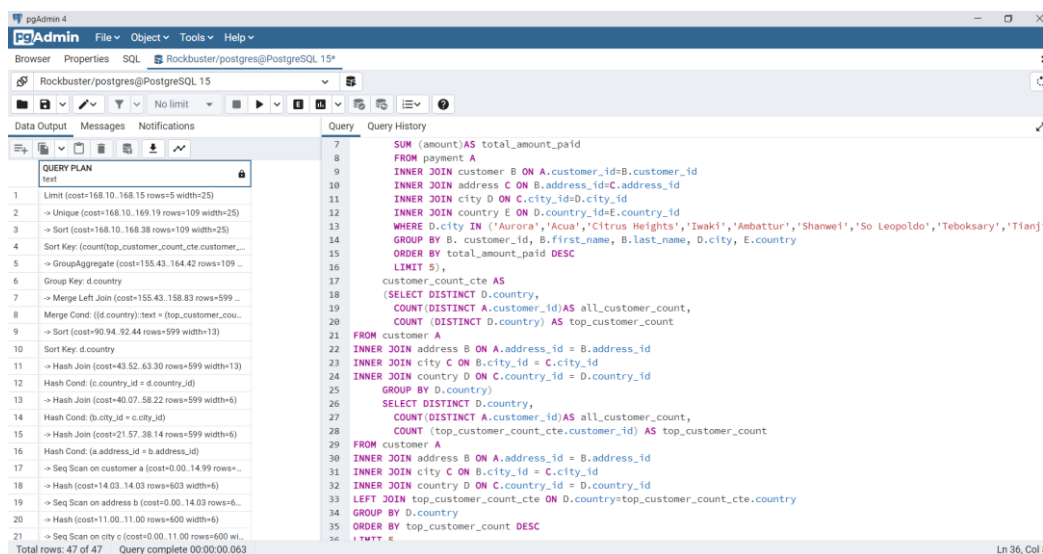
Explain Query 1/sub-query:



51 Miliseconds

Cost: "Limit (cost=189.48..189.49 rows=5 width=84)"

Explain Query 2/CTE:



Its speed was 56 milliseconds (less than query 1/CTE)

Cost: "Limit (cost=168.10..168.15 rows=5 width=25)"

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

Query 1 had a lower cost than Query 2, which is to be expected since the later is much more complex.

In Query 1, there are no major differences between using CTE or sub-queries in terms of cost, and there is a minor difference in the time use, which was longer in the case of the CTE query. Yet, for Query 2, using the CTE reduced the cost, while the time was almost the same as in the sub-query version. These results are not surprising and also show that the differences in time and cost will vary case by case.

Step 3: Write 1 to 2 paragraphs on the challenges you faced when replacing your subqueries with CTEs.

The main challenge I've faced, particularly with the second query was to identify the items that need to be included in the CTEs. It was also challenging to understand how to reference the CTE in the SELECT statement, when I realized I would need more than one CTE to account for the different aspects of the sub-query. The fact that the sub-query version of Query 2 was embedded as LEFT join, with FROM, made it really challenging to understand how it could be converted into a CTE. I think that this requires a lot of practice, and it is not something that can be learned easily. In short, the more complex the sub-query the more difficult it becomes to turn it into a CTE query. But I find the CTE much easier to use, and with a syntax that is more easily understandable.