3.9: Common Table Expressions

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

1. *Rewrite your queries from steps 1 and 2 of task 3.8 as CTEs.*
2. *Write 2 to 3 sentences explaining how you approached this step, for example, what you did first, second, and so on.*

Firstly, as in task 3.8 in Step 1, I located the data I needed in the ERD (customer data and the amount paid). Secondly, I generated the list of customer_id, first_ name and last_name, country, city and total_amount_paid. Thirdly, I defined the CTE and wrote the main statement to retrieve the average amount paid by the top 5 customers and in Step 2 - top 5 customers based within each country.

```
WITH average_amount_paid_cte (customer_id, first_name,
last_name, address, city, total_amount_paid)
AS (SELECT A.customer_id,
        A.first_name,
        A.last_name,
        C.city,
        D.country,
        SUM (E.amount) AS total_amount_paid
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
GROUP BY 1,2,3,4,5
ORDER BY total_amount_paid DESC
LIMIT 5)
SELECT AVG (total_amount_paid)
FROM average_amount_paid_cte
```

	avg numeric
1	199.1920000000000000

Query from Step 2 of task 3.8 as CTE (Find out how many of the top 5 customers are based within each country), using CTE:

```
WITH top_customer_count_cte AS
(SELECT A.customer_id,
        A.first_name,
        A.last_name,
        D.country,
        C.city,
```

```
        SUM (E.amount) AS total_amount_paid
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
GROUP BY A.customer_id, D.country, C.city
ORDER BY total_amount_paid DESC
LIMIT 5),
all_customer_count_cte AS
(SELECT DISTINCT (D.country),
       COUNT(DISTINCT A.customer_id) AS all_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 D.country,
COUNT (DISTINCT A.customer_id) AS all_customer_count,
COUNT (DISTINCT 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
```

	country character varying (50)	all_customer_count bigint	top_customer_count bigint
1	Netherlands	5	1
2	Runion	1	1
3	Belarus	2	1
4	Brazil	28	1
5	United States	36	1

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

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

For me personally, writing subqueries is easier, than writing CTE. However, it's said in the exercise, that CTE makes life much easier. The performance should be proved by checking the cost of each approach.

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

QUERY	COST
3.8 step 1 (subquery)	(cost=64.49..64.50 rows=1 width=32)
3.8 step 2 (subquery)	(cost=1177.64..1177.69 rows=5 width=84)
3.9 step 1 (CTE)	(cost=1046.43..1046.44 rows=1 width=32)
3.9 step 2 (CTE)	(cost=1150.08..1150.35 rows=109 width=25)

3.8 step 1:

	QUERY PLAN text
1	Aggregate (cost=64.49..64.50 rows=1 width=32)
2	-> Limit (cost=64.41..64.43 rows=5 width=67)
3	-> Sort (cost=64.41..65.02 rows=244 width=67)
4	Sort Key: (sum(e.amount)) DESC
5	-> HashAggregate (cost=57.31..60.36 rows=244 width=67)
6	Group Key: a.customer_id, d.country, c.city
7	-> Nested Loop (cost=18.16..54.87 rows=244 width=41)
8	-> Hash Join (cost=17.88..37.14 rows=10 width=35)
9	Hash Cond: (c.country_id = d.country_id)
10	-> Nested Loop (cost=14.43..33.66 rows=10 width=28)

3.8 step 2:

	QUERY PLAN text
1	Limit (cost=1177.64..1177.69 rows=5 width=84)
2	-> Unique (cost=1177.64..1183.09 rows=545 width=84)
3	-> Sort (cost=1177.64..1179.00 rows=545 width=84)
4	Sort Key: (count(DISTINCT d.customer_id)) DESC, a.country, (count(DISTINCT a.country))
5	-> GroupAggregate (cost=1139.93..1152.87 rows=545 width=84)
6	Group Key: a.country, top_5_customers.*
7	-> Sort (cost=1139.93..1141.43 rows=599 width=72)
8	Sort Key: a.country, top_5_customers.*
9	-> Hash Left Join (cost=1090.00..1112.30 rows=599 width=72)
10	Hash Cond: ((a.country)::text = (top_5_customers.country)::text)

3.9 step 1:

	QUERY PLAN text
1	Aggregate (cost=1046.43..1046.44 rows=1 width=32)
2	-> Limit (cost=1046.35..1046.36 rows=5 width=67)
3	-> Sort (cost=1046.35..1082.84 rows=14596 width=67)
4	Sort Key: (sum(e.amount)) DESC
5	-> HashAggregate (cost=621.47..803.92 rows=14596 width=67)
6	Group Key: a.customer_id, c.city, d.country
7	-> Hash Join (cost=66.00..475.51 rows=14596 width=41)
8	Hash Cond: (c.country_id = d.country_id)
9	-> Hash Join (cost=62.55..432.25 rows=14596 width=34)
10	Hash Cond: (b.city_id = c.city_id)

3.9 step 2:

	QUERY PLAN text
1	Sort (cost=1150.08..1150.35 rows=109 width=25)
2	Sort Key: (count(DISTINCT top_customer_count_cte.customer_id)) DESC
3	-> GroupAggregate (cost=1137.41..1146.39 rows=109 width=25)
4	Group Key: d.country
5	-> Merge Left Join (cost=1137.41..1140.81 rows=599 width=17)
6	Merge Cond: ((d.country)::text = (top_customer_count_cte.country)::text)
7	-> Sort (cost=90.94..92.44 rows=599 width=13)
8	Sort Key: d.country
9	-> Hash Join (cost=43.52..63.30 rows=599 width=13)
10	Hash Cond: (c.country_id = d.country_id)

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.*

QUERY	TOTAL QUERY RUNTIME
3.8 step 1 (subquery)	113 msec
3.8 step 2 (subquery)	65 msec
3.9 step 1 (CTE)	143 msec
3.9 step 2 (CTE)	97 msec

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

I'm not surprised that CTE takes longer time to fulfil the command. It takes longer to write CTE than to write subquery, maybe the same logic works with automated intelligence. Accordingly, subqueries have lower cost in comparison with CTE. The cost is almost the same in the 2nd case (3.9 step 2): I assume, it's because of the number of rows added (109 rows in CTE vs. 5 in subqueries). I did several iterations out of curiosity and turned out that each time the time of fulfillment of query was different. Evidently, it depends on amount of processes my computer is doing at the time of command in pgAdmin.

Step 3:

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

CTE for Step 1 was simple, I was even glad to know that I can substitute column names by numbers when grouping the results. The 2nd query was the whole challenge for me since I didn't know what 2nd CTE to define and then, how to combine all into the main query.