Databases & SQL for Analysts

3.9: Common Table Expressions

Exercise 3.7

Write a query to find the top 5 customers in the top 10 cities who have paid the highest total amounts to Rockbuster. The customer team would like to reward them for their loyalty!

Original Query

```
SELECT * FROM (
        SELECT country.country,
                 city.city,
                 customer.customer_id,
                 SUM(payment.amount) as total_payment,
                 ROW_NUMBER() OVER ( PARTITION BY country.country, city.city
                            ORDER BY  SUM(payment.amount) DESC ) as top_customer
        FROM Customer
        INNER JOIN address ON customer.address_id = address.address_id
        INNER JOIN payment ON payment.customer_id = customer.customer_id
        INNER JOIN city ON city.city_id = address.city_id
        INNER JOIN country ON country.country_id = city.country_id
        WHERE city.city in(
                  SELECT top_cities_t.city
                  FROM (SELECT city.city as city,
                                          SUM(payment.amount) as total_payment_in_cities,
                                          ROW_NUMBER() OVER ( ORDER BY
SUM(payment.amount) DESC) as top_cities
                            FROM Customer
                            INNER JOIN address ON customer.address_id = address.address_id
                            INNER JOIN payment ON payment.customer_id =
customer.customer_id
                            INNER JOIN city ON city.city_id = address.city_id
                            INNER JOIN country ON country.country_id = city.country_id
                            GROUP BY city.city
                            ORDER BY top_cities) as top_cities_t
                  WHERE top_cities_t.top_cities < 11 )
        GROUP BY country.country, city.city, customer.customer_id
) top_customers_t
WHERE top_customer < 6
ORDER BY total_payment DESC) top_customers_t
WHERE top_customer < 6
ORDER BY total_payment DESC
```

Query with CTE

```
WITH aggregate_customer_amount_cte AS
(
        SELECT customer.customer_id as customer_id,
                    first_name,
                    last_name,
                     city,
                     country,
                     SUM(amount) as amount
        FROM Customer
        INNER JOIN payment USING (customer_id)
        INNER JOIN address USING (address_id )
        INNER JOIN city USING (city_id)
        INNER JOIN country USING (country_id)
        GROUP BY customer.customer_id,first_name,last_name,
             city, country
),
cities_ranking_cte  AS
(
SELECT city,
          RANK() OVER ( ORDER BY  SUM(amount) DESC ) as city_ranking
FROM aggregate_customer_amount_cte
GROUP BY city
),
customer_ranking AS
(
        SELECT  customer_id, city, country,first_name,
                    last_name, amount,
                            RANK() OVER ( PARTITION BY country, city
                            ORDER BY  amount DESC ) as top_customer
        FROM aggregate_customer_amount_cte
        INNER JOIN cities_ranking_cte USING (city)
        WHERE city_ranking < 11
)

SELECT customer_id, city, country,first_name,
           last_name, amount
FROM customer_ranking
WHERE top_customer < 6
ORDER BY amount DESC,  country, city
```

Step 1: Answer the business questions from steps 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. Copy-paste your CTEs and their outputs into your answers document.
3. Write 2 to 3 sentences explaining how you approached this step, for example, what you did first, second, and so on.

Step 1: Find the average amount paid by the top 5 customers.

```
WITH aggregate_customer_amount_cte (customer_id, first_name, last_name, city, country, amount
) AS
(
             SELECT customer.customer_id,
                         first_name,
                         last_name,
                  city,
                  country,
                SUM(amount) as amount
            FROM Customer

            INNER JOIN payment USING (customer_id)
            INNER JOIN address USING (address_id )
            INNER JOIN city USING (city_id)
            INNER JOIN country USING (country_id)
            GROUP BY customer.customer_id,first_name,last_name,
                city,country
),
cities_ranking_cte  AS
(
SELECT city,
               RANK() OVER ( ORDER BY  SUM(amount) DESC ) as city_ranking
FROM aggregate_customer_amount_cte
GROUP BY city
),
customer_ranking AS
(
            SELECT  customer_id, city, country,first_name,
                        last_name, amount,
                              RANK() OVER ( PARTITION BY country, city
                              ORDER BY  amount DESC ) as top_customer
            FROM aggregate_customer_amount_cte
            INNER JOIN cities_ranking_cte USING (city)
            WHERE city_ranking < 11
)

SELECT city, country, ROUND(AVG(amount),2) as avg_amount  FROM customer_ranking
WHERE top_customer < 6
GROUP BY country, city
ORDER BY avg_amount DESC,  country, city
```

Step 2: Find out how many of the top 5 customers are based within each country.

Your final output should include 3 columns:

- **“country”**
- **“all_customer_count”** with the total number of customers in each country
- **“top_customer_count**” showing how many of the top 5 customers live in each country

```
WITH aggregate_customer_amount_cte (customer_id, first_name, last_name, city, country, amount
) AS

(

        SELECT customer.customer_id,

                    first_name,

                    last_name,

            city,

            country,

            SUM(amount) as amount

        FROM Customer


        INNER JOIN payment USING (customer_id)

        INNER JOIN address USING (address_id )

        INNER JOIN city USING (city_id)

        INNER JOIN country USING (country_id)

        GROUP BY customer.customer_id,first_name,last_name,

            city,country

),

cities_ranking_cte  AS

(

SELECT city,

          RANK() OVER ( ORDER BY  SUM(amount) DESC ) as city_ranking

FROM aggregate_customer_amount_cte
GROUP BY city
),
customer_ranking AS
(
        SELECT  customer_id, city, country,first_name,
                last_name, amount,
                        RANK() OVER ( PARTITION BY country, city
                        ORDER BY  amount DESC ) as top_customer
        FROM aggregate_customer_amount_cte
        INNER JOIN cities_ranking_cte USING (city)
        WHERE city_ranking < 11
),
top_10_cities_top_5_customer AS (
        SELECT customer_id, city, country, first_name,
         last_name, amount
        FROM customer_ranking
        WHERE top_customer < 6
        ORDER BY amount DESC,  country, city
)
SELECT aggregate_customer_amount_cte.country,
            count(aggregate_customer_amount_cte.customer_id) as all_customer_count,
            count(top_10_cities_top_5_customer.customer_id) as top_customer_count
FROM aggregate_customer_amount_cte
LEFT JOIN top_10_cities_top_5_customer using (customer_id)
GROUP BY 1
ORDER BY top_customer_count DESC
```

Solution Approach

1. Create a temporary table for all the customer IDs, names, cities, countries and the total amount paid by each customer.
2. Create another temporary table for all the top 10 cities which paid the highest from the temporary table created in Step 1
3. Find the top 5 customers from the top 10 cities using step 1 and 2 temporary tables. Name it as top_5_customers

4. Join all three temporary tables to create aggregate values

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

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

   CTE can be reusable: One advantage of using CTE is reusable by design. Instead of declaring the same subquery in every place you need to use it, you can use CTE to define a temporary table once and then refer to it whenever needed.

   CTE can be more readable: Another advantage of CTE is CTE is moa re readable than Subqueries. Since CTE can be reusable, you can write less code using CTE than using a subquery.

   Performance: The CTE slightly outperforms the temporary table and table variable queries regarding overall duration.

2. Compare the queries' costs by creating plans for each.

The cost involved in both queries is almost the same, but the temporary table CTE took more time to execute compared subquery. Number sorting and aggregate function are more involved in a query than a subquery, and the result is leading to more performance costs in CTE.

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: using a subquery.	Sort (cost=3171.23..3174.88 rows=1460 width=50)
Explain Query 1: using CTEs	Sort (cost=5058.04..5061.69 rows=1460 width=268)
Explain Query 2: using a subquery	Sort (cost=3269.79..3270.06 rows=109 width=25)
Explain Query 2: using CTEs	Sort (cost=35598.44..35598.94 rows=200 width=134)

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

Surprisingly, the performance cost involved in CTEs is higher than Subqueries. Query 2 with CTE almost cost 35K, more than the subquery, which cost only 3K. The CTE query requires further optimisation for better results.

Step 3:

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

- Technical skills required in CTE are more compared to subqueries. Subqueries are similar to regular queries.
- The performance costs involved in different aggregate functions are different, for example ROW_NUMBER() and RANK().
- Finding the right query to reduce multiple calls to table requires skills.
- CTEs are easier to read than subqueries, making them more organised. Rewriting subqueries is complex task