3.9 Common Table Expressions

Rewriting the queries from task 3.8 as CTEs

To find the average amount paid by the top 5 customers

QUERY :

```
 WITH top_locations AS (
    SELECT D.country, C.city
    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
    WHERE D.country IN (
        SELECT D.country
        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
        ORDER BY COUNT(A.customer_id) DESC
        LIMIT 10
    )
    GROUP BY D.country, C.city
    ORDER BY COUNT(A.customer_id) DESC
    LIMIT 10
),
```

```
top_customers AS (
    SELECT B.customer_id,
        SUM(A.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 (E.country, D.city) IN (SELECT * FROM top_locations)
    GROUP BY B.customer_id
    ORDER BY total_amount_paid DESC
    LIMIT 5
)
SELECT AVG(total_amount_paid) AS average
FROM top_customers;
```

```sql
-- Step 1: Identify the top 10 countries and cities with the highest number of customers
WITH top_locations AS (
    SELECT
        D.country,
        C.city
    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
    WHERE D.country IN (
        -- Subquery: Find the top 10 countries based on the total number of customers
        SELECT
            D.country
        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
        ORDER BY COUNT(A.customer_id) DESC
        LIMIT 10
    )
    -- Group by country and city, and order by the number of customers to pick the top 10 cities
    GROUP BY D.country, C.city
    ORDER BY COUNT(A.customer_id) DESC
    LIMIT 10
),

-- Step 2: Identify the top 5 customers based on the total amount they paid in the top locations
top_customers AS (
    SELECT
        B.customer_id,
        SUM(A.amount) AS total_amount_paid -- Calculate the total payment made by each customer
    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
    -- Filter for customers from the top locations identified in the first CTE
    WHERE (E.country, D.city) IN (SELECT * FROM top_locations)
    GROUP BY B.customer_id -- Group by customer to calculate total payments
    ORDER BY total_amount_paid DESC -- Order by the highest amount paid
    LIMIT 5 -- Pick the top 5 customers
)

-- Step 3: Calculate the average total amount paid by the top 5 customers
SELECT
    AVG(total_amount_paid) AS average -- Find the average of the total payments
FROM top_customers;
```

Data Output | Messages | Notifications

	average numeric
1	105.5540000000000000

Explanation :

1. First, I created a CTE called **top_locations** to get the top 10 countries and cities with the most customers. This step grouped the data by country and city and used filtering to focus on the top 10 countries with the highest customer count.
2. Second, I created another CTE, top_**customer**s to identify the top 5 customers based on the total amount they paid. Here, I joined all the necessary tables (payment, customer, address, city and country) and used the first CTE to filter the data to include only those top countries and cities.
3. The main query calculates the average amount paid by the top 5 customers from the second CTE.

To find out how many of the top 5 customers identified in the above step are based within each country

QUERY :

```
-- Identify the top 10 countries and cities with the most customers
WITH top_locations AS (
    SELECT
        D.country,
        C.city
    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
    WHERE D.country IN (
        -- Find the top 10 countries with the most customers
        SELECT D.country
```

```
        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
        ORDER BY COUNT(A.customer_id) DESC
        LIMIT 10
    )
    GROUP BY D.country, C.city
    ORDER BY COUNT(A.customer_id) DESC
    LIMIT 10
),

-- Find the top 5 customers based on the total amount paid in the top locations
top_customers AS (
    SELECT
        B.customer_id,
        E.country,
        SUM(A.amount) AS total_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 (E.country, D.city) IN (
        SELECT * FROM top_locations
    )
    GROUP BY B.customer_id, E.country
```

```
    ORDER BY total_paid DESC
    LIMIT 5
)

-- Combine all customer data with the top customers and calculate the required counts
SELECT
    E.country AS "Country",
    COUNT(DISTINCT B.customer_id) AS "all_customer_count", -- Total unique customers in the country
    COUNT(DISTINCT top_customers.customer_id) AS "top_customer_count" -- Top 5 customers in the country
FROM customer B
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
LEFT JOIN top_customers
ON E.country = top_customers.country
GROUP BY E.country
ORDER BY "all_customer_count" DESC
LIMIT 5;
```

Query | Query History

```sql
-- Identify the top 10 countries and cities with the most customers
WITH top_locations AS (
    SELECT
        D.country,
        C.city
    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
    WHERE D.country IN (
        -- Find the top 10 countries with the most customers
        SELECT D.country
        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
        ORDER BY COUNT(A.customer_id) DESC
        LIMIT 10
    )
    GROUP BY D.country, C.city
    ORDER BY COUNT(A.customer_id) DESC
    LIMIT 10
),

-- Find the top 5 customers based on the total amount paid in the top locations
top_customers AS (
    SELECT
        B.customer_id,
        E.country,
        SUM(A.amount) AS total_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 (E.country, D.city) IN (
        SELECT * FROM top_locations
    )
    GROUP BY B.customer_id, E.country
    ORDER BY total_paid DESC
    LIMIT 5)
```

Data Output | Messages | Notifications

SQL

	Country character varying (50)	all_customer_count bigint	top_customer_count bigint
1	India	60	1
2	China	53	1
3	United States	36	1
4	Japan	31	1
5	Mexico	30	1

Explanation :

1. First, I created a CTE named top_locations to find the top 10 countries and cities with the highest number of customers. This step groups the data by countries and cities, filters the top countries, and orders them by customer count.
2. Second, I created another CTE called top_customers to identify the top 5 customers with the highest total payments in the previously identified top locations.
3. Finally, I combined the data from all customers with the top customers using a LEFT JOIN and calculated the total unique customer count (all_customer_count) and the count of top customers (top_customer_count) for each country.

Comparing the Performance of CTEs and Subqueries

1st CTE :

	QUERY PLAN text
1	Aggregate (cost=127.61..127.62 rows=1 width=32)
2	-> Limit (cost=127.58..127.58 rows=2 width=36)
3	-> Sort (cost=127.58..127.58 rows=2 width=36)
Total rows: 65	Query complete 00:00:00.110

Sub Query :

	QUERY PLAN text
1	Aggregate (cost=127.62..127.63 rows=1 width=32)
2	-> Limit (cost=127.59..127.59 rows=2 width=67)
3	-> Sort (cost=127.59..127.59 rows=2 width=67)
Total rows: 65	Query complete 00:00:00.151

2nd CTE :

	QUERY PLAN text
1	Limit (cost=229.82..229.83 rows=5 width=25)
2	-> Sort (cost=229.82..230.09 rows=109 width=25)
3	Sort Key: (count(DISTINCT b.customer_id)) DESC
Total rows: 88	Query complete 00:00:00.112

Sub Query :

	QUERY PLAN text
1	Limit (cost=229.82..229.83 rows=5 width=25)
2	-> Sort (cost=229.82..230.09 rows=109 width=25)
3	Sort Key: (count(DISTINCT b.customer_id)) DESC
Total rows: 88	Query complete 00:00:00.112

I didn’t find much difference in the run time between the CTEs and the subqueries because the queries weren’t very large.

CTEs are great for organizing and simplifying queries, especially when you want to break them into smaller, more readable parts. However, in

older versions of PostgreSQL (before version 12), CTEs can be slower because the database processes them as separate steps, making them less flexible for optimization. Subqueries, on the other hand, are inline, allowing the database to optimize them more freely, which can make them faster in some cases.

For small datasets like mine, the difference isn't noticeable. But for larger datasets or more complex queries, it's important to compare both methods to make sure your query runs efficiently. Depending on the database version and the specific query, one method may be better than the other.

Challenges faced when replacing the subqueries with CTEs

When I tried replacing subqueries with CTEs, one of the main challenges I faced was ensuring that the logic stayed consistent and produced the same results. Subqueries are more directly integrated into the main query, so when switching to CTEs, I had to break down the logic into separate parts. This sometimes made the process feel less intuitive, as I needed to carefully plan the steps and structure the query differently.

Since CTEs require defining each step explicitly, it sometimes felt harder to keep track of how one part connected to the next, especially with multiple CTEs in the same query. This made debugging and refining the query more time-consuming. It also took me some time to adjust to the new structure and avoid redundant operations, as CTEs don't flow as seamlessly as subqueries in certain cases. Overall, while CTEs can make queries more readable in the end, I found them tricky and difficult to write.