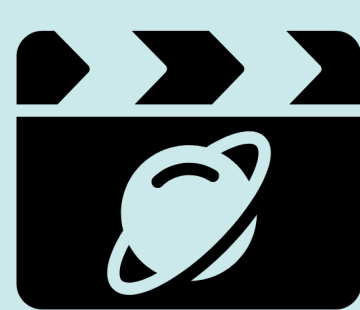
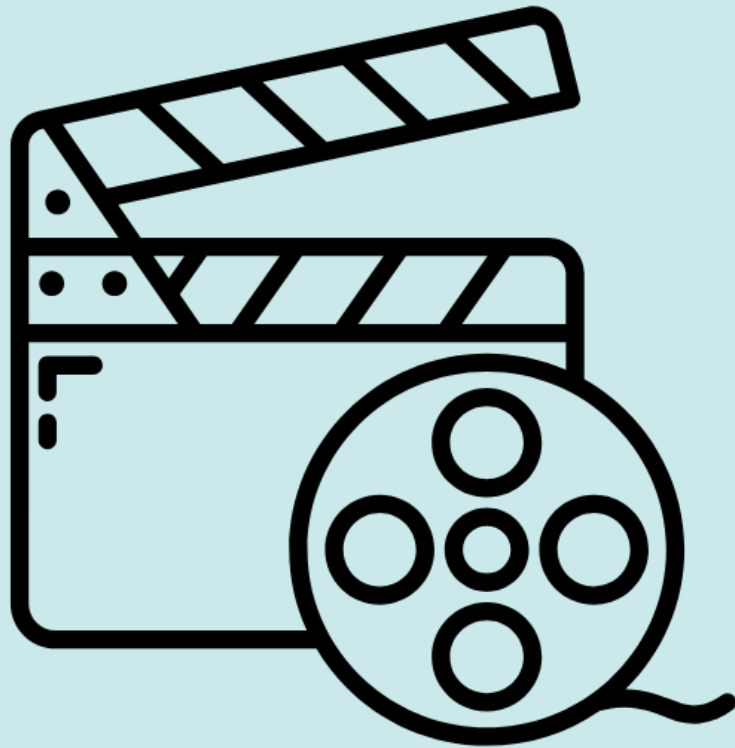


COMMON TABLE EXPRESSIONS (CTEs)





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

QueryQuery History

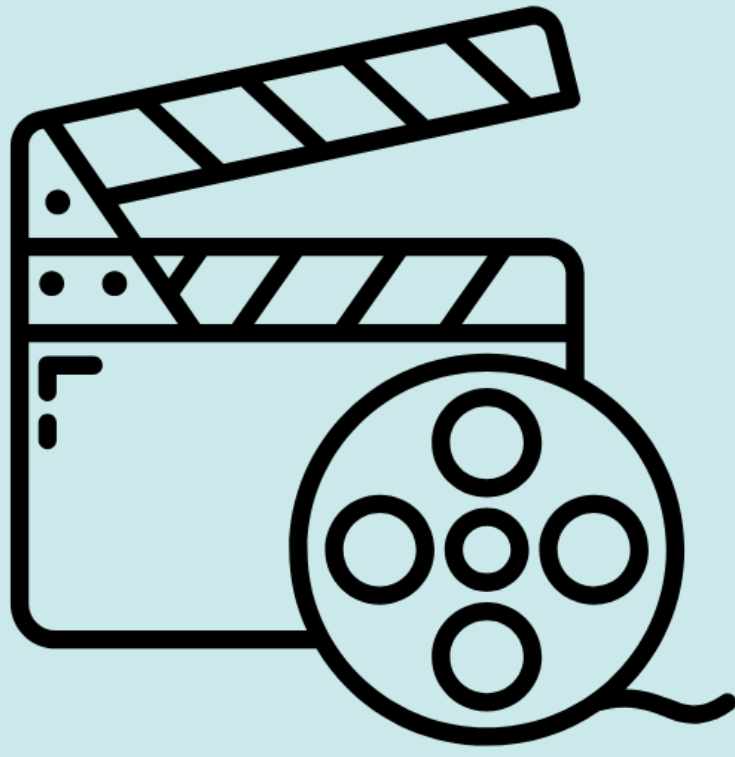
```
1  -- Step 1: Calculate the total amount paid by each customer, along with their name, city, and country
2  WITH customer_payments AS (
3      SELECT
4          A.customer_id,
5          A.first_name,
6          A.last_name,
7          D.country,
8          C.city,
9          SUM(P.amount) AS total_amount_paid -- total amount paid by the customer
10     FROM customer A
11     JOIN address B ON A.address_id = B.address_id -- link customer to their address
12     JOIN city C ON B.city_id = C.city_id -- link address to city
13     JOIN country D ON C.country_id = D.country_id -- link city to country
14     JOIN payment P ON A.customer_id = P.customer_id -- link customer to their payments
15     GROUP BY A.customer_id, A.first_name, A.last_name, D.country, C.city
16 ),
17
18 -- Step 2: Find the top 10 countries with the most customers
19 top_countries AS (
20     SELECT
21         D.country -- country name
22     FROM customer A
23     JOIN address B ON A.address_id = B.address_id
24     JOIN city C ON B.city_id = C.city_id
25     JOIN country D ON C.country_id = D.country_id
26     GROUP BY D.country -- group by country
27     ORDER BY COUNT(A.customer_id) DESC -- sort by number of customers in descending order
28     LIMIT 10 -- take only the top 10 countries
29 ),
30
31 -- Step 3: Find the top 10 cities within those countries, based on customer count
32 top_cities AS (
33     SELECT
34         C.city -- city name
35     FROM customer A
36     JOIN address B ON A.address_id = B.address_id
37     JOIN city C ON B.city_id = C.city_id
38     JOIN country D ON C.country_id = D.country_id
39     WHERE D.country IN (SELECT country FROM top_countries) -- only include cities from top countries
40     GROUP BY D.country, C.city -- group by both country and city
41     ORDER BY COUNT(A.customer_id) DESC -- sort by number of customers per city
42     LIMIT 10 -- take top 10 cities
43 ),
44
45 -- Step 4: From the top cities, get the top 5 customers based on total amount paid
46 top_customers AS (
47     SELECT *
48     FROM customer_payments
49     WHERE city IN (SELECT city FROM top_cities) -- only include customers from top cities
50     ORDER BY total_amount_paid DESC -- sort by spending amount (highest first)
51     LIMIT 5 -- take top 5 paying customers
52 )
53
54 -- Step 5: Calculate the average amount paid by these top 5 customers
55 SELECT
56     AVG(total_amount_paid) AS average_amount_paid -- final result: average amount paid by top 5
57 FROM top_customers;
```

Data OutputMessagesExplain XNotifications

Showing rows: 1 to 1

Page No: 1 of 1

	average_amount_paid numeric
1	105.5540000000000000



Step 1B: Find out how many of the top 5 customers you identified in step 1 are based within each country.

QueryQuery History


```
1  -- Step 1: Calculate total amount paid by each customer
2  WITH customer_totals AS (
3      SELECT
4          p.customer_id,                -- customer ID
5          SUM(p.amount) AS total_sum    -- total amount paid by this customer
6      FROM payment p
7      GROUP BY p.customer_id           -- group by customer to get total per customer
8  ),
9
10 -- Step 2: Calculate the average total amount paid across all customers
11 avg_total_payment AS (
12     SELECT
13         AVG(total_sum) AS avg_payment -- average total amount paid
14     FROM customer_totals
15 ),
16
17 -- Step 3: Identify top customers (those who paid more than the average)
18 top_customers AS (
19     SELECT
20         ct.customer_id                -- customer ID
21     FROM customer_totals ct
22     CROSS JOIN avg_total_payment atp  -- include average value for comparison
23     WHERE ct.total_sum > atp.avg_payment -- only keep customers above the average
24 ),
25
26 -- Step 4: Combine top customer info with their country
27 customer_country AS (
28     SELECT
29         a.customer_id,
30         d.country                    -- country of the customer
31     FROM customer a
32     JOIN address b ON a.address_id = b.address_id
33     JOIN city c ON b.city_id = c.city_id
34     JOIN country d ON c.country_id = d.country_id
35 ),
36
37 -- Step 5: Count total and top customers per country
38 country_stats AS (
39     SELECT
40         cc.country,                  -- country name
41         COUNT(DISTINCT cc.customer_id) AS all_customer_count, -- total customers in country
42         COUNT(DISTINCT tc.customer_id) AS top_customer_count  -- top customers from that country
43     FROM customer_country cc
44     LEFT JOIN top_customers tc
45         ON cc.customer_id = tc.customer_id -- check if customer is a "top customer"
46     GROUP BY cc.country                 -- group by country
47 )
48
49 -- Step 6: Return the top 10 countries with the most top customers
50 SELECT
51     country,
52     all_customer_count, -- total number of customers
53     top_customer_count  -- number of top customers (paid above average)
54 FROM country_stats
55 ORDER BY top_customer_count DESC -- sort by number of top customers
56 LIMIT 10;                       -- return top 10 countries
57
```

Data OutputMessagesExplain XNotifications

Showing rows: 1 to 10

Page No: 1 of 1

	country character varying (50)	all_customer_count bigint	top_customer_count bigint
1	India	60	26
2	China	53	25
3	United States	36	16
4	Japan	31	14
5	Russian Federation	28	13
6	Brazil	28	12
7	Mexico	30	11
8	Philippines	20	11
9	Taiwan	10	7
10	Turkey	15	7

Data Immersion	Achievement III		
	SQL for Data Analysts		
	Task 3.8 By Ola Gaffarova	Table of content Page 4 of 5	Made for Rockbuster Stealth

Step 2:

I initially expected the subquery version to perform similarly or even slightly better than the CTE version, since inline subqueries often allow PostgreSQL to optimize the execution path more freely. However, in this case, the subquery version— which nests a CASE WHEN condition containing two subselects (one of them with a nested AVG() of a grouped subquery)— resulted in significantly higher planning and execution costs. This is because PostgreSQL evaluates those subqueries repeatedly for each row, which adds considerable overhead.

In contrast, the CTE version calculates total payments and the average once, in isolated steps, and then joins the results. This avoids redundant computation and leads to improved performance and efficiency. While PostgreSQL may materialize CTEs (treating them as temporary result sets), which can sometimes limit query planner optimizations, in this scenario, the clarity, modularity, and pre-aggregation provided by CTEs outweighed those drawbacks.

That said, I believe there are cases where the performance of a CTE is nearly identical to that of a subquery, and other cases where CTEs provide a substantial improvement. Therefore, it is important to test and analyze each query individually, using tools like EXPLAIN ANALYZE, rather than assuming one approach is always better than the other.

SUBQUERY STEP1A

Data Output	Messages	Explain	×	Notifications
Showing rows: 1 to 65 Page No: 1 of 1				
QUERY PLAN text				
1	Aggregate (cost=1078.11..1078.12 rows=1 width=32)			

CTE STEP1A

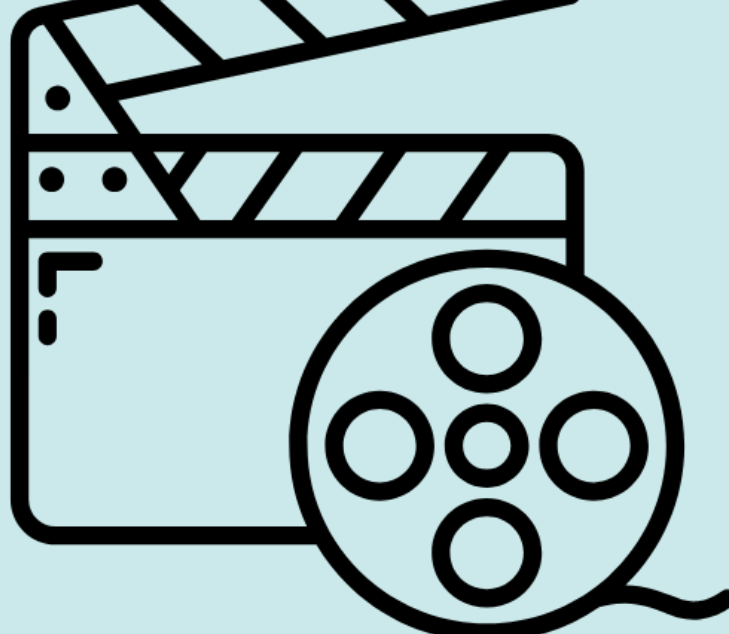
Data Output	Messages	Explain	×	Notifications
Showing rows: 1 to 66 Page No: 1 of 1				
QUERY PLAN text				
1	Aggregate (cost=1125.02..1125.03 rows=1 width=32)			

SUBQUERY STEP1B

Data Output	Messages	Explain	×	Notifications
Showing rows: 1 to 31 Page No: 1 of 1				
QUERY PLAN text				
1	Limit (cost=38213.96..38213.98 rows=10 width=25)			

CTE STEP1B

Data Output	Messages	Explain	×	Notifications
Showing rows: 1 to 32 Page No: 1 of 1				
QUERY PLAN text				
1	Limit (cost=475.77..475.80 rows=10 width=25)			

Data Immersion	Achievement III		
	SQL for Data Analysts		
	Task 3.8 By Ola Gaffarova	Table of content Page 5 of 5	Made for Rockbuster Stealth

Step 3:

Replacing subqueries with CTEs was helpful for breaking down the logic into manageable, readable sections. It made the query structure easier to understand, especially when working with multi-step filtering (e.g. top countries → top cities → top customers). Each block could be explained and debugged independently.