Omid Soltani
Data Analytics Immersion – Task 3.8

Step 1

Query | Query History

```
SELECT AVG(total_amount_paid) AS average
FROM
    (SELECT
        customer.customer_id,
        customer.first_name,
        customer.last_name,
        country.country,
        city.city,
        SUM(payment.amount) AS total_amount_paid
    FROM
        customer
    JOIN
        payment ON customer.customer_id = payment.customer_id
    JOIN
        address ON customer.address_id = address.address_id
    JOIN
        city ON address.city_id = city.city_id
    JOIN
        country ON city.country_id = country.country_id
    WHERE
        city.city IN ('Aurora', 'London', 'Abu Dhabi', 'Adana', 'Addis Abeba')
    GROUP BY
        customer.customer_id, customer.first_name, customer.last_name, country.country, city.city
    ORDER BY
        total_amount_paid DESC
    LIMIT
        5);
```

Data Output | Messages | Notifications

	average numeric
1	96.1580000000000000

The query is designed to calculate the average total amount paid by the top 5 customers from the specified cities:

1. The inner SELECT statement retrieves relevant information about customers, including their IDs, names, country, city, and the total amount they paid. It uses JOIN operations to connect tables such as ‘customer’, ‘payment’, ‘address’, ‘city’, and ‘country’.
2. The WHERE clause filters the results to include only customers from specific cities: 'Aurora', 'London', 'Abu Dhabi', 'Adana', and 'Addis Abeba'.
3. The GROUP BY clause groups the results by customer ID, first name, last name, country, and city.
4. The ORDER BY clause arranges the results based on the total amount paid in descending order.
5. The LIMIT clause restricts the output to the top 5 customers.
6. The outer SELECT statement calculates the average of the total amount paid by the top 5 customers using the AVG function and aliases the result as 'average'.

Omid Soltani
Data Analytics Immersion – Task 3.8

Step 2

Query | Query History

```sql
SELECT D.country,
       COUNT(DISTINCT A.customer_id) AS all_customer_count,
       COUNT(DISTINCT top_5_customers.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
  (SELECT A.customer_id,
          A.first_name,
          A.last_name,
          D.country_id,
          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
   WHERE C.city IN ('Aurora',
                    'Acua',
                    'Citrus Heights',
                    'Iwaki',
                    'Ambattur',
                    'Shanwei',
                    'So Leopoldo',
                    'Teboksary',
                    'Tianjin',
                    'Cianju')
   GROUP BY A.customer_id,
            A.first_name,
            A.last_name,
            D.country_id,
            C.city
   ORDER BY total_amount_paid DESC
   LIMIT 5) top_5_customers ON A.customer_id = top_5_customers.customer_id
GROUP BY D.country
ORDER BY all_customer_count DESC
LIMIT 5;
```

Data Output | Messages | Notifications

	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

Total rows: 5 of 5 | Query complete 00:00:00.129

In the given scenario, steps 1 and 2 involve aggregating data and counting the number of customers both globally and within specific criteria, such as the top 5 cities. While it's technically possible to achieve similar results without using subqueries by employing advanced techniques like window functions or complex joins, subqueries provide a more straightforward and readable solution.

Subqueries are particularly useful when dealing with nested queries or scenarios where a result set depends on the outcome of another query. They enhance code readability, simplify complex logic, and enable a step-by-step approach to problem-solving. In this case, using a subquery allows for a clearer separation of concerns, making it easier to understand and maintain the overall query structure.