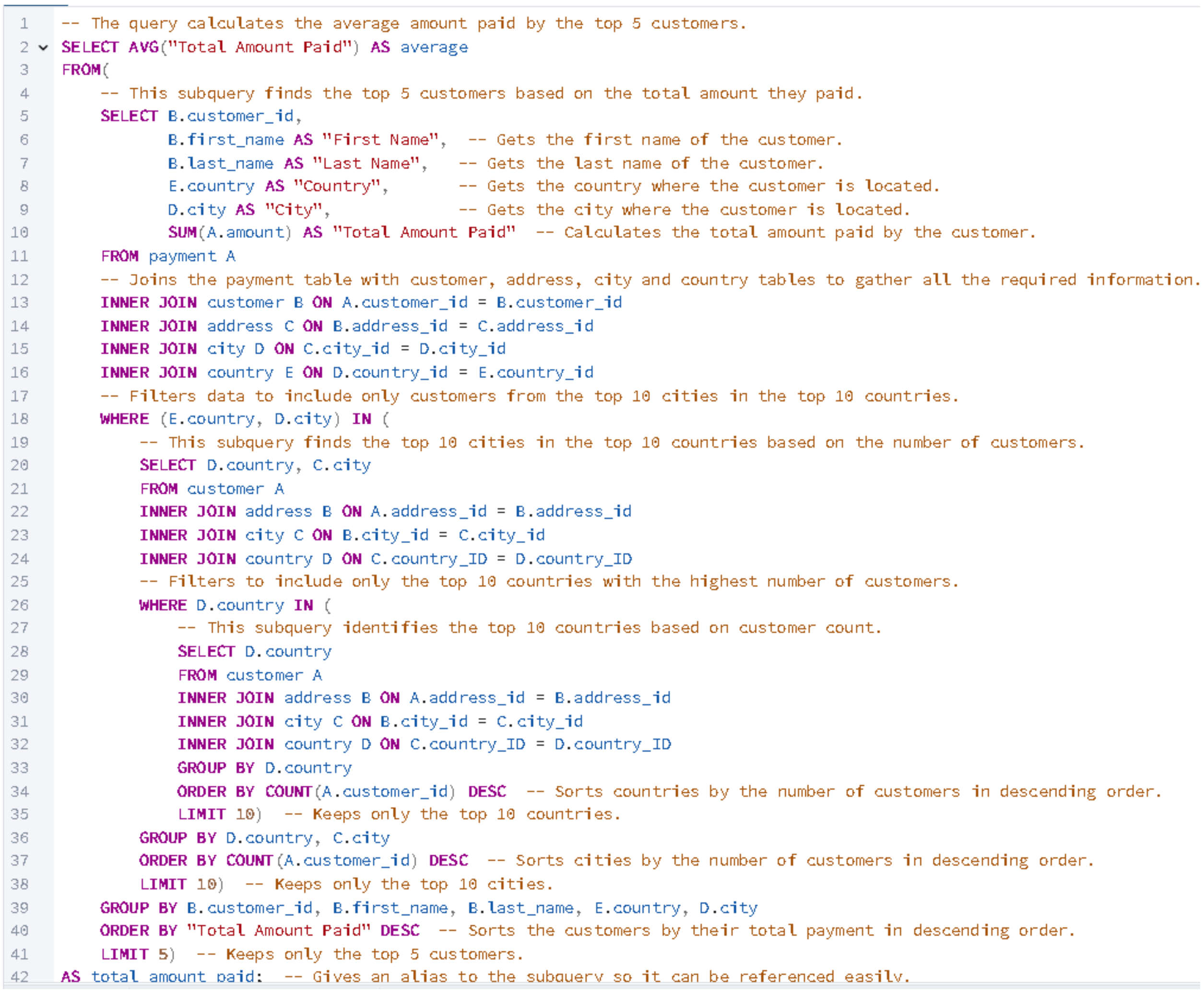
Exercise 3.8 Performing Subqueries

1. **To find the average amount paid by the top 5 customers**

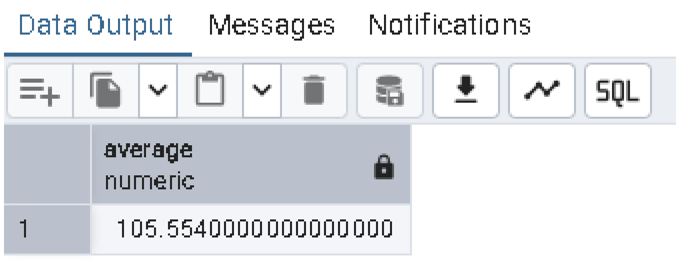
Query :

```
SELECT AVG("Total Amount Paid") AS average
FROM(
SELECT B.customer_id,
B.first_name AS "First Name",
B.last_name AS "Last Name",
E.country AS "Country",
D.city AS "City",
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 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)
GROUP BY B.customer_id, B.first_name, B.last_name, E.country, D.city
ORDER BY "Total Amount Paid" DESC
LIMIT 5)
AS total_amount_paid;
```



```sql
-- The query calculates the average amount paid by the top 5 customers.
SELECT AVG("Total Amount Paid") AS average
FROM(
    -- This subquery finds the top 5 customers based on the total amount they paid.
    SELECT B.customer_id,
           B.first_name AS "First Name",  -- Gets the first name of the customer.
           B.last_name AS "Last Name",   -- Gets the last name of the customer.
           E.country AS "Country",       -- Gets the country where the customer is located.
           D.city AS "City",             -- Gets the city where the customer is located.
           SUM(A.amount) AS "Total Amount Paid"  -- Calculates the total amount paid by the customer.
    FROM payment A
    -- Joins the payment table with customer, address, city and country tables to gather all the required information.
    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
    -- Filters data to include only customers from the top 10 cities in the top 10 countries.
    WHERE (E.country, D.city) IN (
        -- This subquery finds the top 10 cities in the top 10 countries based on the number of customers.
        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
        -- Filters to include only the top 10 countries with the highest number of customers.
        WHERE D.country IN (
            -- This subquery identifies the top 10 countries based on customer count.
            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  -- Sorts countries by the number of customers in descending order.
            LIMIT 10)  -- Keeps only the top 10 countries.
        GROUP BY D.country, C.city
        ORDER BY COUNT(A.customer_id) DESC  -- Sorts cities by the number of customers in descending order.
        LIMIT 10)  -- Keeps only the top 10 cities.
    GROUP BY B.customer_id, B.first_name, B.last_name, E.country, D.city
    ORDER BY "Total Amount Paid" DESC  -- Sorts the customers by their total payment in descending order.
    LIMIT 5)  -- Keeps only the top 5 customers.
AS total_amount_paid;  -- Gives an alias to the subquery so it can be referenced easily.
```

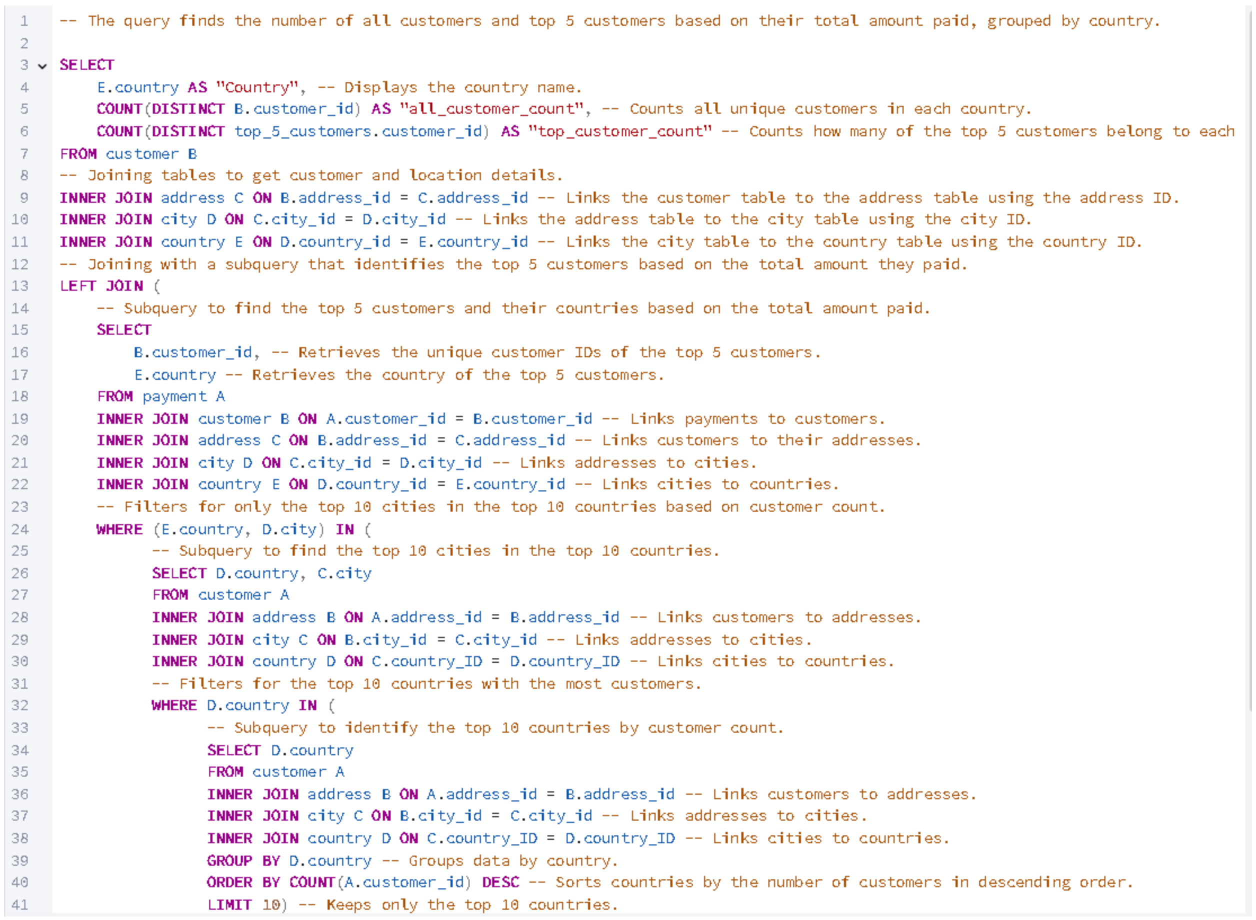


```
   average
   numeric
1  105.5540000000000000
```

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

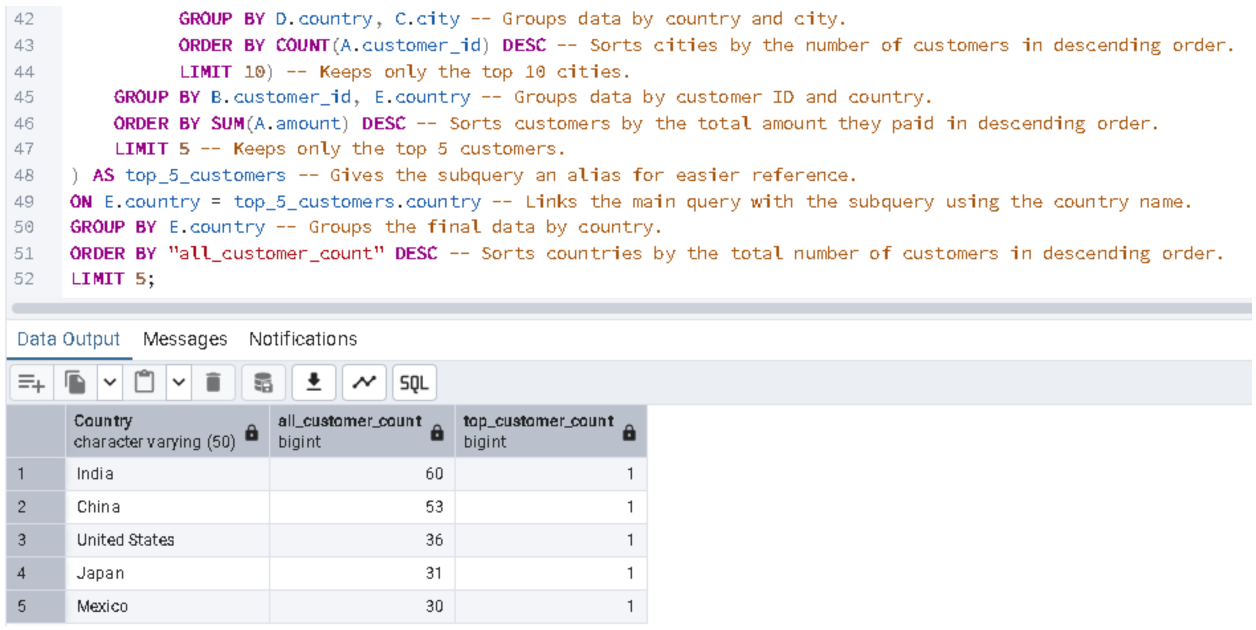
Query :

```
SELECT
    E.country AS "Country",
    COUNT(DISTINCT B.customer_id) AS "all_customer_count",
    COUNT(DISTINCT top_5_customers.customer_id) AS
"top_customer_count"
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 (
    -- Inner query to identify top 5 customers based on total amount paid
    SELECT
        B.customer_id,
        E.country
    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 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)
    GROUP BY B.customer_id, E.country
    ORDER BY SUM(A.amount) DESC
    LIMIT 5
) AS top_5_customers
ON E.country = top_5_customers.country
GROUP BY E.country
ORDER BY "all_customer_count" DESC
LIMIT 5;
```



```sql
-- The query finds the number of all customers and top 5 customers based on their total amount paid, grouped by country.

SELECT
    E.country AS "Country", -- Displays the country name.
    COUNT(DISTINCT B.customer_id) AS "all_customer_count", -- Counts all unique customers in each country.
    COUNT(DISTINCT top_5_customers.customer_id) AS "top_customer_count" -- Counts how many of the top 5 customers belong to each
FROM customer B
-- Joining tables to get customer and location details.
INNER JOIN address C ON B.address_id = C.address_id -- Links the customer table to the address table using the address ID.
INNER JOIN city D ON C.city_id = D.city_id -- Links the address table to the city table using the city ID.
INNER JOIN country E ON D.country_id = E.country_id -- Links the city table to the country table using the country ID.
-- Joining with a subquery that identifies the top 5 customers based on the total amount they paid.
LEFT JOIN (
    -- Subquery to find the top 5 customers and their countries based on the total amount paid.
    SELECT
        B.customer_id, -- Retrieves the unique customer IDs of the top 5 customers.
        E.country -- Retrieves the country of the top 5 customers.
    FROM payment A
    INNER JOIN customer B ON A.customer_id = B.customer_id -- Links payments to customers.
    INNER JOIN address C ON B.address_id = C.address_id -- Links customers to their addresses.
    INNER JOIN city D ON C.city_id = D.city_id -- Links addresses to cities.
    INNER JOIN country E ON D.country_id = E.country_id -- Links cities to countries.
    -- Filters for only the top 10 cities in the top 10 countries based on customer count.
    WHERE (E.country, D.city) IN (
          -- Subquery to find the top 10 cities in the top 10 countries.
          SELECT D.country, C.city
          FROM customer A
          INNER JOIN address B ON A.address_id = B.address_id -- Links customers to addresses.
          INNER JOIN city C ON B.city_id = C.city_id -- Links addresses to cities.
          INNER JOIN country D ON C.country_ID = D.country_ID -- Links cities to countries.
          -- Filters for the top 10 countries with the most customers.
          WHERE D.country IN (
                -- Subquery to identify the top 10 countries by customer count.
                SELECT D.country
                FROM customer A
                INNER JOIN address B ON A.address_id = B.address_id -- Links customers to addresses.
                INNER JOIN city C ON B.city_id = C.city_id -- Links addresses to cities.
                INNER JOIN country D ON C.country_ID = D.country_ID -- Links cities to countries.
                GROUP BY D.country -- Groups data by country.
                ORDER BY COUNT(A.customer_id) DESC -- Sorts countries by the number of customers in descending order.
                LIMIT 10) -- Keeps only the top 10 countries.
```



```sql
          GROUP BY D.country, C.city -- Groups data by country and city.
          ORDER BY COUNT(A.customer_id) DESC -- Sorts cities by the number of customers in descending order.
          LIMIT 10) -- Keeps only the top 10 cities.
    GROUP BY B.customer_id, E.country -- Groups data by customer ID and country.
    ORDER BY SUM(A.amount) DESC -- Sorts customers by the total amount they paid in descending order.
    LIMIT 5 -- Keeps only the top 5 customers.
) AS top_5_customers -- Gives the subquery an alias for easier reference.
ON E.country = top_5_customers.country -- Links the main query with the subquery using the country name.
GROUP BY E.country -- Groups the final data by country.
ORDER BY "all_customer_count" DESC -- Sorts countries by the total number of customers in descending order.
LIMIT 5;
```

	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

3. Writing SQL queries, especially with subqueries, has been a challenging experience for me as someone with limited SQL knowledge. While subqueries can be very useful, I found them quite confusing and difficult to manage when working on complex tasks. Subqueries are helpful when you need to filter or calculate data from multiple tables or create temporary results to work with. However, in my case, writing long and nested subqueries felt overwhelming, especially because the query became hard to read and interpret. I had to carefully match the keys and ensure everything was joined correctly, which took a lot of trial and error.

   I believe subqueries are necessary in situations where you need up-to-date results or when creating temporary logic that doesn't require additional database objects like tables or views. But for someone like me, who has just started learning SQL, using tools like CTEs (Common Table Expressions), views, or even temporary tables would have made things much simpler. Using a view, for instance, would have been as simple as referencing a single name instead of copying 20+ lines of code repeatedly.

   I understand that subqueries are efficient in certain cases, especially when working with real-time data, but for complex problems like this (for me), they became tricky and prone to errors. It would be great to have an easier approach for such tasks to avoid mistakes, as even a small error in a subquery can change the entire result.