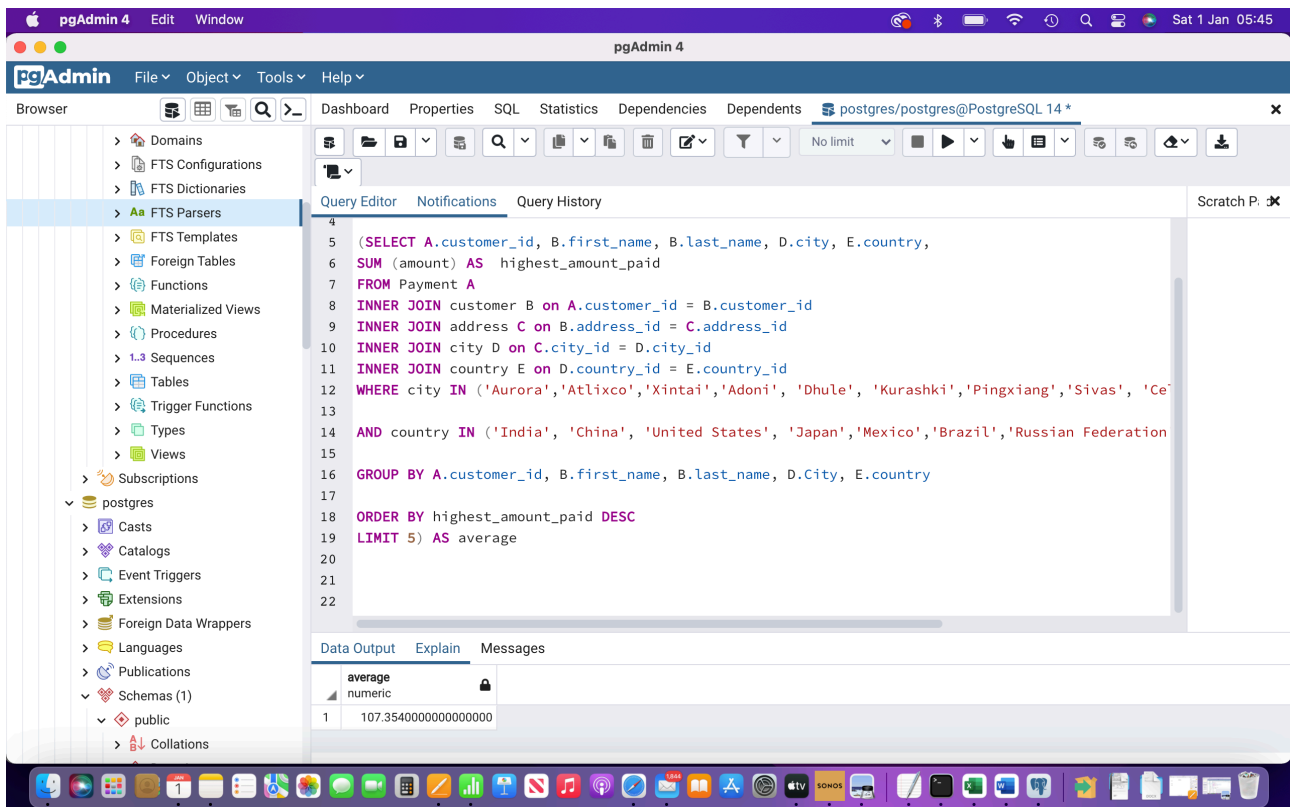


EXERCISE 3.8

Part 1.

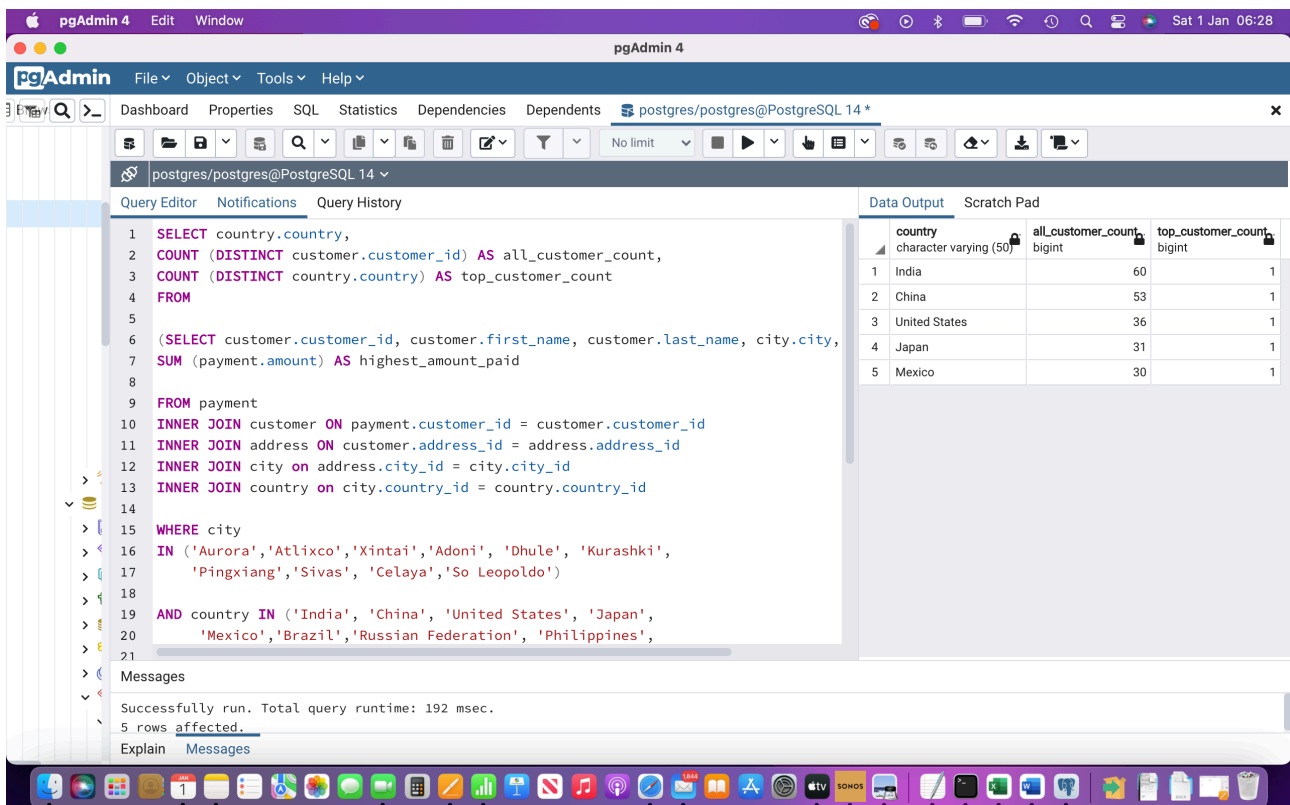


The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure, including the 'public' schema. The central pane shows a SQL query in the Query Editor. The query is a complex JOIN statement that filters data by city and country, calculates the highest amount paid for each customer, and returns the top 5 results ordered by highest amount paid in descending order. The output is displayed in the Data Output pane at the bottom right.

```
4
5 (SELECT A.customer_id, B.first_name, B.last_name, D.city, E.country,
6 SUM (amount) AS highest_amount_paid
7 FROM Payment A
8 INNER JOIN customer B on A.customer_id = B.customer_id
9 INNER JOIN address C on B.address_id = C.address_id
10 INNER JOIN city D on C.city_id = D.city_id
11 INNER JOIN country E on D.country_id = E.country_id
12 WHERE city IN ('Aurora','Atlixco','Xintai','Adoni', 'Dhule', 'Kurashki','Pingxiang','Sivas', 'Ce
13
14 AND country IN ('India', 'China', 'United States', 'Japan','Mexico','Brazil','Russian Federation
15
16 GROUP BY A.customer_id, B.first_name, B.last_name, D.City, E.country
17
18 ORDER BY highest_amount_paid DESC
19 LIMIT 5) AS average
20
21
22
```

average
107.354000000000000000

Part 2. (a) - ***OUTPUT is on the right hand side and not bottom due to size of SYNTAX.



The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure. The central pane shows a SQL query in the Query Editor. The query is a complex JOIN statement that filters data by city and country, calculates the highest amount paid for each customer, and returns the top 5 results ordered by highest amount paid in descending order. The output is displayed in the Data Output pane at the bottom right.

```
1 SELECT country.country,
2 COUNT (DISTINCT customer.customer_id) AS all_customer_count,
3 COUNT (DISTINCT country.country) AS top_customer_count
4 FROM
5
6 (SELECT customer.customer_id, customer.first_name, customer.last_name, city.city,
7 SUM (payment.amount) AS highest_amount_paid
8
9 FROM payment
10 INNER JOIN customer ON payment.customer_id = customer.customer_id
11 INNER JOIN address ON customer.address_id = address.address_id
12 INNER JOIN city on address.city_id = city.city_id
13 INNER JOIN country on city.country_id = country.country_id
14
15 WHERE city
16 IN ('Aurora','Atlixco','Xintai','Adoni', 'Dhule', 'Kurashki',
17 'Pingxiang','Sivas', 'Celaya','So Leopoldo')
18
19 AND country IN ('India', 'China', 'United States', 'Japan',
20 'Mexico','Brazil','Russian Federation', 'Philippines',
21
```

country	all_customer_count	top_customer_count
1 India	60	1
2 China	53	1
3 United States	36	1
4 Japan	31	1
5 Mexico	30	1

Messages

Successfully run. Total query runtime: 192 msec.
5 rows affected.

Part 2 (b)

The screenshot shows the pgAdmin 4 interface with a SQL query editor and a data output window. The query is a complex SQL statement involving multiple tables and joins. The data output window shows the results of the query, which are 5 rows of data.

```
17 'Pingxiang','Sivas','Celaya','So Leopoldo')
18
19 AND country IN ('India','China','United States','Japan',
20 'Mexico','Brazil','Russian Federation','Philippines',
21 'Turkey','Indonesia')
22
23 GROUP BY customer.customer_id, customer.first_name, customer.last_name, city.City
24
25 ORDER BY highest_amount_paid DESC
26 LIMIT 5) AS top_customer_count
27
28 LEFT JOIN customer ON customer.customer_id = customer.customer_id
29 LEFT JOIN address ON customer.address_id = address.address_id
30 LEFT JOIN city ON address.city_id = city.city_id
31 LEFT JOIN country ON city.country_id = country.country_id
32 GROUP BY country.country
33 ORDER BY COUNT(country.country) DESC
34
35 LIMIT 5
36
```

country	all_customer_count	top_customer_count
India	60	1
China	53	1
United States	36	1
Japan	31	1
Mexico	30	1

Messages

Successfully run. Total query runtime: 192 msec.
5 rows affected.

Part 3a)

Step 1 could have been achieved without a subquery.

Step 2 required me to combine related information from different tables and therefore would be efficient.

Part 3 b)

Subqueries are beneficial in dividing complex queries into logical steps and have the added benefit of being efficient .