SQL Challenge

Submitted By:
Abhishek Chaubey

Contents:

SQL Order of Execution:

1. FROM and JOINs
The FROM clause, and subsequent JOINs are first executed to determine the total working set of data that is being queried. This includes subqueries in this clause, and can cause temporary tables to be created under the hood containing all the columns and rows of the tables being joined.

2. WHERE
Once we have the total working set of data, the first-pass WHERE constraints are applied to the individual rows, and rows that do not satisfy the constraint are discarded. Each of the constraints can only access columns directly from the tables requested in the FROM clause. Aliases in the SELECT part of the query are not accessible in most databases since they may include expressions dependent on parts of the query that have not yet executed.

3. GROUP BY
The remaining rows after the WHERE constraints are applied are then grouped based on common values in the column specified in the GROUP BY clause. As a result of the grouping, there will only be as many rows as there are unique values in that column. Implicitly, this means that you should only need to use this when you have aggregate functions in your query.

4. HAVING
If the query has a GROUP BY clause, then the constraints in the HAVING clause are then applied to the grouped rows, discard the grouped rows that don't satisfy the constraint. Like the WHERE clause, aliases are also not accessible from this step in most databases.

5. SELECT
Any expressions in the SELECT part of the query are finally computed.

6. DISTINCT
Of the remaining rows, rows with duplicate values in the column marked as DISTINCT will be discarded.

7. ORDER BY
If an order is specified by the ORDER BY clause, the rows are then sorted by the specified data in either ascending or descending order. Since all the expressions in the SELECT part of the query have been computed, you can reference aliases in this clause.

8. LIMIT / OFFSET
Finally, the rows that fall outside the range specified by the LIMIT and OFFSET are discarded, leaving the final set of rows to be returned from the query.

Set 1:

Question 1:

```sql

-- Q.1 Query all columns for all American cities in the CITY table with
-- populations larger than 100000. The countryCode for America is USA.


SELECT
        id,
        name,
        countrycode,
        district,
        population
FROM
        city
WHERE
        population > 100000
        AND
        countrycode = 'USA';

```

Question 2:

```sql

-- Q.2 Query the name field for all American cities in the CITY table with
-- populations larger than 120000. The countryCode for America is USA.


SELECT
        name
FROM
        city
WHERE
        population > 120000
        AND
        countrycode = 'USA';


```

Question 3:

```sql

-- Q.3 Query all columns (attributes) for every row in the CITY table.


SELECT
        id,
        name,
        countrycode,
        district,
        population
FROM
        city;

```

Question 4:

```sql

-- Q.4 Query all columns for a city in CITY with the id 1661.


SELECT
        id,
        name,
        countrycode,
        district,
        population
FROM
        city
WHERE
        id = 1661;


```

Question 5:

```sql

-- Q.5 Query all attributes of every Japanese city in the CITY table.
-- The countryCODE for Japan is JPN.


SELECT
        id,
        name,
        countrycode,
        district,
        population
FROM
        city
WHERE
        countrycode = 'JPN';


```

Question 6:

```sql

-- Q.6 Query the names of all the Japanese cities in the CITY table.
-- The countryCODE for Japan is JPN.


SELECT
        name
FROM
        city
WHERE
        countrycode = 'JPN';


```

Question 7:

```sql

-- Q.7 Query a list of CITY and STATE from the station table.


SELECT
        city,
        state
FROM
        station;

```

Question 8:

```sql

-- Q.8 Query a list of CITY names from STATION for cities that have an even ID number.
-- Print the results in any order, but exclude duplicates from the answer.


SELECT
        DISTINCT city
FROM
        station
WHERE
        id%2 = 0;

```

Question 9:

```sql

-- Q.9 Find the difference between the total number of CITY entries in the table and the
-- number of distinct CITY entries in the table.


SELECT
        (COUNT(city) -  COUNT(DISTINCT city)) AS difference
FROM
        station;

```

Question 10:

```sql

-- Q.10 Query the two cities in STATION with the shortest and longest CITY names, as well as their
-- respective lengths (i.e.: number of characters in the name). If there is more than one smallest
-- or largest city, choose the one that comes first when ordered alphabetically.


-- APPROACH 1:


(SELECT
        city,
        Length(city) AS len
FROM
        station
ORDER BY
        Length(city),
        city
LIMIT 1)

UNION ALL

(SELECT
        city,
        Length(city) AS len
FROM
        station
ORDER BY
        Length(city) DESC,
        city
LIMIT 1);



-- APPROACH 2:


WITH temp_city AS (
                    SELECT
                            city,
                            LENGTH(city) as len,
                            ROW_NUMBER() OVER (ORDER BY LENGTH(city), city) AS smallest,
                            ROW_NUMBER() OVER (ORDER BY LENGTH(city) DESC, city) AS largest
                    FROM
                            station
                )

SELECT
        city,
        len
FROM
        temp_city
WHERE
        smallest = 1
        OR
        largest = 1
ORDER BY
        len;

```

Question 11:

```sql

-- Q11. Query the list of CITY names starting with vowels (i.e., a, e, i, o, or u) from station.
-- Your result cannot contain duplicates.


-- Approach 1


SELECT
        DISTINCT city
FROM
        station
WHERE
        LEFT(city , 1) IN ('a','e','i','o','u');


-- Approach 2


SELECT
       DISTINCT city
FROM
        station
WHERE
        SUBSTR(city,1,1) IN ('A','E','I','O','U');


-- Approach 3


SELECT
        DISTINCT city
FROM
        station
WHERE
        city RLIKE '^[aeiouAEIOU]';


-- Approach 4


SELECT
        DISTINCT city
FROM
        station
WHERE
        city REGEXP '^[aeiouAEIOU]';


-- Approach 5


SELECT
        DISTINCT city
FROM
        station
WHERE
        city REGEXP '^[aeiou]';

```

Question 12:

```sql

-- Q12. Query the list of CITY names ending with vowels (a, e, i, o, u) from station.
-- Your result cannot contain duplicates.


-- Approach 1


SELECT
        DISTINCT city
FROM
        station
WHERE
        RIGHT(city , 1) IN ('a','e','i','o','u');


-- Approach 2


SELECT
        DISTINCT city
FROM
        station
WHERE
        SUBSTR(city,-1,1) IN ('A','E','I','O','U');


-- Approach 3


SELECT
        DISTINCT city
FROM
        station
WHERE
        city RLIKE '.*[aeiouAEIOU]$';


-- Approach 4


SELECT
        DISTINCT city
FROM
        station
WHERE
        city REGEXP '.*[aeiouAEIOU]$';


-- Approach 5


SELECT
        DISTINCT city
FROM
        station
WHERE
        city REGEXP '[aeiou]$';

```

Question 13:

```sql

-- Q13. Query the list of CITY names from station that do not start with vowels.
-- Your result cannot contain duplicates.


-- Approach 1


SELECT
        DISTINCT city
FROM
        station
WHERE
        LEFT(city , 1) NOT IN ('a','e','i','o','u');


-- Approach 2


SELECT
        DISTINCT city
FROM
        station
WHERE
        SUBSTR(city,1,1) NOT IN ('A','E','I','O','U');


-- Approach 3


SELECT
        DISTINCT city
FROM
        station
WHERE
        city NOT RLIKE '^[aeiouAEIOU]';


-- Approach 4


SELECT
        DISTINCT city
FROM
        station
WHERE
        city NOT REGEXP '^[aeiouAEIOU]';


-- Approach 5


SELECT
        DISTINCT city
FROM
        station
WHERE
        city NOT REGEXP '^[aeiou]';

```

Question 14:

```sql

-- Q14. Query the list of CITY names from station that do not end with vowels.
-- Your result cannot contain duplicates.


-- Approach 1


SELECT
        DISTINCT city
FROM
        station
WHERE
        RIGHT(city , 1) NOT IN ('a','e','i','o','u');


-- Approach 2


SELECT
        DISTINCT city
FROM
        station
WHERE
        SUBSTR(city,-1,1) NOT IN ('A','E','I','O','U');


-- Approach 3


SELECT
        DISTINCT city
FROM
        station
WHERE
        city NOT RLIKE '.*[aeiouAEIOU]$';


-- Approach 4


SELECT
        DISTINCT city
FROM
        station
WHERE
        city NOT REGEXP '.*[aeiouAEIOU]$';


-- Approach 5


SELECT
        DISTINCT city
FROM
        station
WHERE
        city NOT REGEXP '[aeiou]$';

```

Question 15:

```sql

-- Q15. Query the list of CITY names from station that either do not start with vowels or
-- do not end with vowels. Your result cannot contain duplicates.


-- Approach 1


SELECT
        DISTINCT city
FROM
        station
WHERE
        SUBSTR(city,1,1) NOT IN ('A','E','I','O','U')
        OR
        SUBSTR(city,-1,1) NOT IN ('A','E','I','O','U');


-- Approach 2


SELECT
        DISTINCT city
FROM
        station
WHERE
        city RLIKE '^[^aeiouAEIOU].*|.*[^AEIOUaeiou]$';

```

Question 16:

```sql

-- Q16. Query the list of CITY names from station that do not start with vowels and
-- do not end with vowels. Your result cannot contain duplicates.


-- Approach 1


SELECT
        DISTINCT city
FROM
        station
WHERE
        SUBSTR(city,1,1) NOT IN ('A','E','I','O','U')
        AND
        SUBSTR(city,-1,1) NOT IN ('A','E','I','O','U');


-- Approach 2


SELECT
        DISTINCT city
FROM
        station
WHERE
        city NOT REGEXP '^[aeiou]|[aeiou]$';


-- Approach 3


SELECT
        DISTINCT city
FROM
        station
WHERE
        city REGEXP '^[^aeiou].*[^aeiou]$';

```

Question 17:

```sql

-- Q.17 Write an SQL query that reports the products that were only sold in the first quarter of 2019.
-- That is, between 2019-01-01 and 2019-03-31 inclusive. Return the result table in any order.


-- Approach 1


SELECT
        p.product_id,
        p.product_name
FROM
        product p
INNER JOIN
        sales s
ON
        p.product_id = s.product_id
WHERE
        p.product_id
NOT IN (
        SELECT
                product_id
        FROM
                sales
        WHERE
                sales_date > '2019-03-31'
        )
GROUP BY
        p.product_id
HAVING
        MAX(s.sales_date) <='2019-03-31'
        AND
        MIN(s.sales_date) >='2019-01-01';


-- Approach 2


SELECT
        p.product_id,
        p.product_name
FROM
        product p
INNER JOIN
        sales s
ON
        p.product_id = s.product_id
GROUP BY
        p.product_id,
        p.product_name
HAVING
        MIN(s.sales_date) >= '2019-01-01' AND MAX(s.sales_date) <= '2019-03-31';

```

Question 18:

```sql

-- Q.18 Write an SQL query to find all the authors that viewed at least one of their own articles.
-- Return the result table sorted by id in ascending order.


SELECT
        DISTINCT a.author_id
FROM
        views a
JOIN
        views v
ON
        a.author_id = v.viewer_id
ORDER BY
        a.author_id;

```

Question 19:

```sql

-- Q.19 Write an SQL query to find the percentage of immediate orders in the table,
-- rounded to 2 decimal places.


SELECT
        ROUND(immediate_orders * 100 / total_orders, 2) AS immediate_orders_perct
FROM
        (
            SELECT
                    COUNT(
                            CASE
                            WHEN order_date = customer_preferred_delivery_date
                            THEN customer_id
                            END
                        )   AS immediate_orders,
                    COUNT(delivery_id) AS total_orders
            FROM
                    delivery
        ) temp ;

```

Question 20:

```sql

-- Q.20 Write an SQL query to find the ctr of each Ad. Round ctr to two decimal points.
-- Return the result table ordered by ctr in descending order and by ad_id in
-- ascending order in case of a tie.


SELECT
        ad_id,
        CASE
            WHEN (num_of_clicks * 100) / (num_of_clicks + num_of_views) IS NULL THEN 0
            ELSE ROUND((num_of_clicks * 100) / (num_of_clicks + num_of_views), 2)
            END AS ctr
FROM
        (
            SELECT
                    ad_id,
                    COUNT(
                            CASE
                            WHEN action = 'CLICKED' THEN ad_id
                            END
                        )   AS num_of_clicks,
                    COUNT(
                            CASE WHEN action = 'VIEWED' THEN ad_id
                            END
                        )   AS num_of_views
            FROM
                    ads
            GROUP BY
                    ad_id
        )   temp_ads
ORDER BY
        ctr DESC,
        ad_id ASC;

```

Question 21:

```sql

-- Q.21 Write an SQL query to find the team size of each of the employees.


SELECT
        employee_id,
        COUNT(employee_id) OVER(PARTITION BY team_id) AS team_size
FROM
        employee
ORDER BY
        employee_id;

```

Question 22:

```sql

-- Q.22 Write an SQL query to find the type of weather in each country for November 2019. The type of weather is:
-- • Cold if the average weather_state is less than or equal 15,
-- • Hot if the average weather_state is greater than or equal to 25, and
-- • Warm otherwise.
-- Return result table in any order.


SELECT
        c.country_name,
        CASE
            WHEN AVG(weather_state) <= 15 THEN 'COLD'
            WHEN AVG(weather_state) >= 25 THEN 'HOT'
            ELSE 'WARM'
            END AS avg_weather
FROM
        countries c
INNER JOIN
        weather w
ON
        c.country_id = w.country_id
WHERE
        day BETWEEN '2019-11-01' AND '2019-11-30'
GROUP BY
        c.country_id,
        c.country_name;

```

Question 23:

```sql

-- Q.23 Write an SQL query to find the average selling price for each product.
-- average_price should be rounded to 2 decimal places.


SELECT
        p.product_id,
        ROUND(SUM(p.price * u.units)/SUM(u.units), 2) AS average_price
FROM
        units_sold u
INNER JOIN
        prices p
ON
        p.product_id = u.product_id
WHERE
        u.purchase_date BETWEEN p.start_date AND p.end_date
GROUP BY
        p.product_id;

```

Question 24:

```sql

-- Q.24 Write an SQL query to report the first login date for each player.
-- Return the result table in any order.


-- Approach 1


WITH temp_activity AS (
                            SELECT
                                    player_id,
                                    event_date as first_login,
                                    ROW_NUMBER() OVER(PARTITION BY player_id ORDER BY event_date) as ranking
                            FROM
                                    activity
                    )

SELECT
        player_id,
        first_login
FROM
        temp_activity
WHERE
        ranking = 1;


-- Approach 2


SELECT
        player_id,
        first_login
FROM    (
            SELECT
                    player_id,
                    event_date as first_login,
                    ROW_NUMBER() OVER(PARTITION BY player_id ORDER BY event_date) ranking
            FROM
                    activity
        ) temp_activity
WHERE
        ranking = 1;

```

Question 25:

```sql

-- Q.25 Write an SQL query to report the device that is first logged in for each player.
-- Return the result table in any order.


-- Approach 1


WITH temp_activity AS (
                            SELECT
                                    player_id,
                                    device_id,
                                    ROW_NUMBER() OVER(PARTITION BY player_id ORDER BY device_id) ranking
                            FROM
                                    activity
                        )

SELECT
        player_id,
        device_id
FROM
        temp_activity
WHERE
        ranking = 1;


-- Approach 2


SELECT
        player_id,
        device_id
FROM    (
            SELECT
                    player_id,
                    device_id,
                    ROW_NUMBER() OVER(PARTITION BY player_id ORDER BY device_id) as ranking
            FROM
                    activity
        ) temp_activity
WHERE
        ranking = 1;

```

Question 26:

```sql

-- Q.26 Write an SQL query to get the names of products that have at least 100 units
-- ordered in February 2020 and their amount.


SELECT
        p.product_name,
        SUM(o.unit) AS total_unit_sold
FROM
        products p
INNER JOIN
        orders o
ON
        p.product_id = o.product_id
WHERE
        order_date BETWEEN '2020-02-01' AND '2020-02-28'
GROUP BY
        p.product_id
HAVING
        SUM(o.unit) >= 100;

```

Question 27:

```sql

-- Q.27 Write an SQL query to find the users who have valid emails.


SELECT
        user_id,
        name,
        mail
FROM
        users
WHERE
        mail LIKE '%@leetcode.com' AND mail REGEXP '^[a-zA-Z0-9][a-zA-Z0-9._-]*@[a-zA-Z0-9][a-zA-Z0-9._-]*\\.[a-zA-Z]{2,4}$';

```

Question 28:

```sql

-- Q.28 Write an SQL query to report the customer_id and customer_name of customers who have spent
-- at least $100 in each month of June and July 2020. Return the result table in any order.


SELECT
        customer_id,
        name
 FROM
        (
            SELECT
                    c.customer_id,
                    c.name,
                    EXTRACT(MONTH FROM o.order_date) AS month_extracted,
                    SUM(o.quantity * p.price) AS total_spent
            FROM
                    orders o
            INNER JOIN
                    customers c
            ON
                    c.customer_id = o.customer_id
            INNER JOIN
                    products p
            ON
                    p.product_id = o.product_id
            WHERE
                    o.order_date BETWEEN '2020-06-01' AND '2020-07-31'
            GROUP BY
                    c.customer_id,
                    c.name,
                    EXTRACT(MONTH FROM o.order_date)
        ) temp_customers

WHERE
        total_spent >= 100
GROUP BY
        customer_id
HAVING
        COUNT(customer_id) = 2;

```

Question 29:

```sql

-- Q.29 Write an SQL query to report the distinct titles of the kid-friendly movies streamed in June 2020.
-- Return the result table in any order.


SELECT
        DISTINCT c.title
FROM
        content c
INNER JOIN
        tv_program t
ON
        c.content_id = T.content_id
WHERE
        c.kids_content = 'Y'
        AND
        c.content_type = 'movies'
        AND
        T.program_date BETWEEN '2020-06-01' AND '2020-06-30';

```

Question 30:

```sql

-- Q.30 Write an SQL query to find the npv of each query of the Queries table.
-- Return the result table in any order.


SELECT
        DISTINCT n.id,
        n.year,
        n.npv
FROM
        queries q
INNER JOIN
        npv n
ON
        n.id = q.id
        AND
        n.year = q.year;

```

Question 31:

```sql

-- Q.31 Write an SQL query to find the npv of each query of the Queries table.
-- Return the result table in any order.


SELECT
        DISTINCT n.id,
        n.year,
        n.npv
FROM
        queries q
INNER JOIN
        npv n
ON
        n.id = q.id
        AND
        n.year = q.year;

```

Question 32:

```sql

-- Q.32 Write an SQL query to show the unique id of each user, If a user does not have a
-- unique id replace just show null. Return the result table in any order.


SELECT
        eu.unique_id,
        e.name
FROM
        employees e
LEFT JOIN
        employees_uni eu
ON
        e.id = eu.id
ORDER BY
        e.name;

```

Question 33:

```sql

-- Q.33 Write an SQL query to report the distance travelled by each user. Return the result table ordered by travelled_distance
-- in descending order, if two or more users travelled the same distance, order them by their name in ascending order.


SELECT
        u.name AS riders,
        COALESCE(SUM(r.distance), 0) AS distance_travelled
FROM
        users u
LEFT JOIN
        rides r
ON
        u.id = r.user_id
GROUP BY
        u.name
ORDER BY
        distance_travelled DESC,
        riders;

```

Question 34:

```sql

-- Q.34 Write an SQL query to get the names of products that have at least 100 units
-- ordered in February 2020 and their amount.


SELECT
        p.product_name,
        SUM(o.unit) AS total_unit_sold
FROM
        products p
INNER JOIN
        orders o
ON
        p.product_id = o.product_id
WHERE
        order_date BETWEEN '2020-02-01' AND '2020-02-28'
GROUP BY
        p.product_id
HAVING
        SUM(o.unit) >= 100;

```

Question 35:

```sql
-- Q.35 Write an SQL query to:
-- • Find the name of the user who has rated the greatest number of movies. In case of a tie,
-- return the lexicographically smaller user name.


SELECT
        u.name as user
FROM
        users u
INNER JOIN
        movie_rating mr
ON
        u.user_id = mr.user_id
GROUP BY
        u.user_id
ORDER BY
        COUNT(u.name) DESC,
        LENGTH(u.name)
LIMIT 1;


-- • Find the movie name with the highest average rating in February 2020. In case of a tie,
-- return the lexicographically smaller movie name.


SELECT
        m.title
FROM
        movies m
INNER JOIN
        movie_rating mr
ON
        m.movie_id = mr.movie_id
WHERE
        mr.created_at BETWEEN '2020-02-01' AND '2020-02-28'
GROUP BY
        m.movie_id
ORDER BY
        AVG(rating) DESC,
        m.title
LIMIT 1;

```

Question 36:

```sql

-- Q.36 Write an SQL query to report the distance travelled by each user. Return the result table ordered by travelled_distance
-- in descending order, if two or more users travelled the same distance, order them by their name in ascending order.


SELECT
        u.name AS riders,
        COALESCE(SUM(r.distance), 0) AS distance_travelled
FROM
        users u
LEFT JOIN
        rides r
ON
        u.id = r.user_id
GROUP BY
        u.name
ORDER BY
        distance_travelled DESC,
        riders;

```

Question 37:

```sql

-- Q.38 Write an SQL query to find the id and the name of all students who are enrolled
-- in departments that no longer exist. Return the result table in any order.


SELECT
        id,
        name
FROM
        students
WHERE
        department_id NOT IN (
                                SELECT
                                        id
                                FROM
                                        departments
                            );

```

Question 38:

```sql

-- Q.38 Write an SQL query to find the id and the name of all students who are enrolled
-- in departments that no longer exist. Return the result table in any order.


SELECT
        id,
        name
FROM
        students
WHERE
        department_id NOT IN (
                                SELECT
                                        id
                                FROM
                                        departments
                            );

```

Question 39:

```sql

-- Q.39 Write an SQL query to report the number of calls and the total call duration between
-- each pair of distinct persons (person1, person2) where person1 < person2.
-- Return the result table in any order.


-- Approach 1


SELECT
        CASE
            WHEN from_id < to_id THEN from_id
            ELSE to_id
            END AS person1,
        CASE
            WHEN from_id < to_id THEN to_id
            ELSE from_id
            END AS person2,
        COUNT(*) AS call_count,
        SUM(duration) AS total_duration
FROM
        calls
GROUP BY
        person1,
        person2;


-- Approach 2


SELECT
        least(from_id, to_id) AS person1,
        greatest(from_id,to_id) AS person2,
        COUNT(*) AS call_count,
        SUM(duration) AS total_duration
FROM
        calls
GROUP BY
        person1,
        person2;

```

Question 40:

```sql

-- Q.40 Write an SQL query to find the average selling price for each product.
-- average_price should be rounded to 2 decimal places.


SELECT
        p.product_id,
        ROUND(SUM(p.price * u.units)/SUM(u.units), 2) AS average_price
FROM
        units_sold u
INNER JOIN
        prices p
ON
        p.product_id = u.product_id
WHERE
        u.purchase_date BETWEEN p.start_date AND p.end_date
GROUP BY
        p.product_id;

```

Question 41:

```sql

-- Q.41 Write an SQL query to report the number of cubic feet of volume the inventory
-- occupies in each warehouse. Return the result table in any order.


SELECT
        w.name AS warehouse_name,
        SUM(p.length * p.width * p.height * w.units) AS volume
FROM
        warehouse w
INNER JOIN
        products p
ON
        w.product_id = p.product_id
GROUP BY
        w.name;

```

Question 42:

```sql
-- Q.42 Write an SQL query to report the difference between the number of
-- apples and oranges sold each day. Return the result table ordered by sale_date.


-- Approach 1


SELECT
        sale_date,
        difference
FROM
        (
            SELECT
                    sale_date,
                    sold_num - LEAD(sold_num, 1) OVER(PARTITION BY sale_date) AS difference
            FROM
                    sales
        )   temp_sales
WHERE
        difference IS NOT NULL
ORDER BY
        sale_date;


-- Approach 2


SELECT
        sale_date,
        SUM(
            CASE
            WHEN fruit = 'APPLES' THEN sold_num
            WHEN fruit = 'ORANGES' THEN -sold_num
            END
            ) AS difference
FROM
        sales
GROUP BY
        sale_date
ORDER BY
        sale_date;

```

```sql

-- Approach 3


SELECT
        s.sale_date,
        s.sold_num - ss.sold_num AS difference
FROM
        sales s
INNER JOIN
        sales ss
ON
        s.sale_date = ss.sale_date
WHERE
        s.fruit = 'APPLES' AND ss.fruit = 'ORANGES'
ORDER BY
        sale_date;


-- Approach 4


SELECT
        sale_date,
        SUM(IF(fruit = 'APPLES', sold_num, -sold_num)) AS difference
FROM
        sales
GROUP BY
        sale_date
ORDER BY
        sale_date;


-- Approach 5


SELECT
        sale_date,
        SUM(
            CASE
            WHEN fruit = 'APPLES' THEN sold_num
            ELSE -sold_num
            END
            ) AS difference
FROM
        sales
GROUP BY
        sale_date
ORDER BY
        sale_date;

```

Question 43:

```sql

-- Q.43 Write an SQL query to report the fraction of players that logged in again on the day after the day
-- they first logged in, rounded to 2 decimal places. In other words, you need to count the number of players
-- that logged in for at least two consecutive days starting from their first login date,
-- then divide that number by the total number of players.


-- Approach 1


WITH temp_activity AS (
                        SELECT
                                player_id,
                                LEAD(event_date, 1) OVER(PARTITION BY player_id ORDER BY event_date)  - event_date  AS difference
                        FROM
                                activity
            ),

    temp_activity2 AS (
                        SELECT
                                COUNT(DISTINCT player_id) AS players_count
                        FROM
                                temp_activity
                        WHERE
                                difference = 1
                        GROUP BY
                                player_id
            )

SELECT
        ROUND(COUNT(*) / (SELECT COUNT(DISTINCT player_id) FROM activity), 2) AS fraction
FROM
        temp_activity2;


-- Approach 2


WITH temp_activity AS (
                        SELECT
                                player_id,
                                event_date,
                                DATEDIFF(event_date,lag(event_date) OVER(PARTITION BY player_id ORDER BY event_date)) AS difference
                        FROM
                                activity
            ),

    temp_activity2 AS (
                        SELECT
                                COUNT(DISTINCT player_id) AS players_count
                        FROM
                                temp_activity
                        WHERE
                                difference = 1
                        GROUP BY
                                player_id
            )


SELECT
        ROUND(COUNT(*) / (SELECT COUNT(DISTINCT player_id) FROM activity), 2) AS fraction
FROM
        temp_activity2;

```

Question 44:

```sql

-- Q.44 Write an SQL query to report the managers with at least five direct reports.


-- Approach 1


SELECT
        name
FROM
        employee
WHERE
        id = (
                SELECT
                        e.manager_id
                FROM
                        employee e
                INNER JOIN
                        employee ee
                ON
                        e.manager_id = ee.id
                GROUP BY
                        e.manager_id
                HAVING
                        COUNT(e.manager_id) >= 5
            );


-- Approach 2


SELECT
        name
FROM
        employee
WHERE
        id = (
                SELECT
                        manager_id
                FROM
                        employee
                GROUP BY
                        manager_id
                HAVING
                        COUNT(manager_id) >= 5
            );

```

Question 45:

```sql

-- Q.45 Write an SQL query to report the respective department name and number of students majoring in
-- each department for all departments in the department table (even ones with no current students).
-- Return the result table ordered by student_number in descending order. In case of a tie, order
-- them by dept_name alphabetically

SELECT
        d.department_name,
        COUNT(s.student_id) AS number_of_students
FROM
        department d
LEFT JOIN
        student s
ON
        d.dept_id = s.dept_id
GROUP BY
        d.department_name
ORDER BY
        number_of_students DESC,
        department_name;

```

Question 46:

```sql

-- Q.46 Write an SQL query to report the customer ids from the Customer table that bought
-- all the products in the product table. Return the result table in any order.


SELECT
        customer_id
FROM
        customer
GROUP BY
        customer_id
HAVING
        COUNT(DISTINCT product_key) = (
                                        SELECT
                                                COUNT(DISTINCT product_key)
                                        FROM
                                                product
                                    );

```

Question 47:

```sql
-- Q.47 Write an SQL query that reports the most experienced employees in each project. In case of a tie,
-- report all employees with the maximum number of experience years. Return the result table in any order.


-- Approach 1


SELECT
        project_id,
        employee_id
FROM
        (
            SELECT
                    p.project_id,
                    e.employee_id,
                    DENSE_RANK() OVER(PARTITION BY p.project_id ORDER BY e.experience_years DESC) AS ranking
            FROM
                    employee e
            INNER JOIN
                    project p
            ON
                    e.employee_id = p.employee_id
        ) temp_employee
WHERE
        ranking = 1;


-- Approach 2


SELECT
        p.project_id,
        e.employee_id
FROM
        employee e
INNER JOIN
        project p
ON
        e.employee_id = p.employee_id
WHERE
        e.experience_years =
                            (
                                SELECT
                                        MAX(experience_years)
                                FROM
                                        employee
                            );

```

Question 48:

```sql

-- Q.48 Write an SQL query that reports the books that have sold less than 10 copies in the last year,
-- excluding books that have been available for less than one month from today. Assume today is 2019-06-23.
-- Return the result table in any order.


SELECT
        b.book_id, b.name
FROM
        books b
LEFT JOIN
        orders o
ON
        b.book_id = o.book_id
WHERE
        available_from < '2019-05-23'
        AND
        (o.dispatch_date BETWEEN '2018-06-23' AND '2019-06-23')
        OR
        dispatch_date IS NULL
GROUP BY
        b.book_id,
        b.name
HAVING
        COALESCE(SUM(o.quantity), 0) < 10;

```

Question 49:

```sql

-- Q.49 Write a SQL query to find the highest grade with its corresponding course for each student.
-- In case of a tie, you should find the course with the smallest course_id. Return the result table
-- ordered by student_id in ascending order.


-- Approach 1


SELECT
        student_id,
        course_id,
        grade
FROM
        (
            SELECT
                    student_id,
                    course_id,
                    grade,
                    ROW_NUMBER() OVER(PARTITION BY student_id ORDER BY grade DESC) AS ranking
            FROM
                    enrollments
        ) temp_enrollments
WHERE
        ranking = 1
order by
        student_id;


-- Approach 2


WITH temp_enrollments AS
            (
                SELECT
                        student_id,
                        course_id,
                        grade,
                        ROW_NUMBER() OVER(PARTITION BY student_id ORDER BY grade DESC) AS ranking
                FROM
                        enrollments
            )

SELECT
        student_id,
        course_id,
        grade
FROM
        temp_enrollments
WHERE
        ranking = 1
order by
        student_id;

```

Question 50:

```sql

-- Q.50 Write an SQL query to find the winner in each group. Return the result table in any order.


SELECT
        group_id,
        players as player_id
FROM    (
            SELECT
                    p.group_id,
                    CASE
                        WHEN first_player_goals > second_player_goals THEN first_player
                        WHEN first_player_goals < second_player_goals THEN second_player
                        WHEN first_player_goals = second_player_goals THEN IF(first_player < second_player, first_player, second_player)
                        END AS players,
                        MAX(IF(first_player_goals > second_player_goals, first_player_goals, second_player_goals)) AS goals,
                        ROW_NUMBER() OVER(PARTITION BY team_id ORDER BY MAX(IF(first_player_goals > second_player_goals, first_player_goals, second_player_goals)) DESC) AS ranking
            FROM
                    players p
            INNER JOIN
                    matches m
            ON
                    m.first_player = p.player_id
                    OR
                    m.second_player = p.player_id
            GROUP BY
                    p.group_id,
                    players
        ) temp_matches
WHERE
        ranking = 1;

```

Set 2:

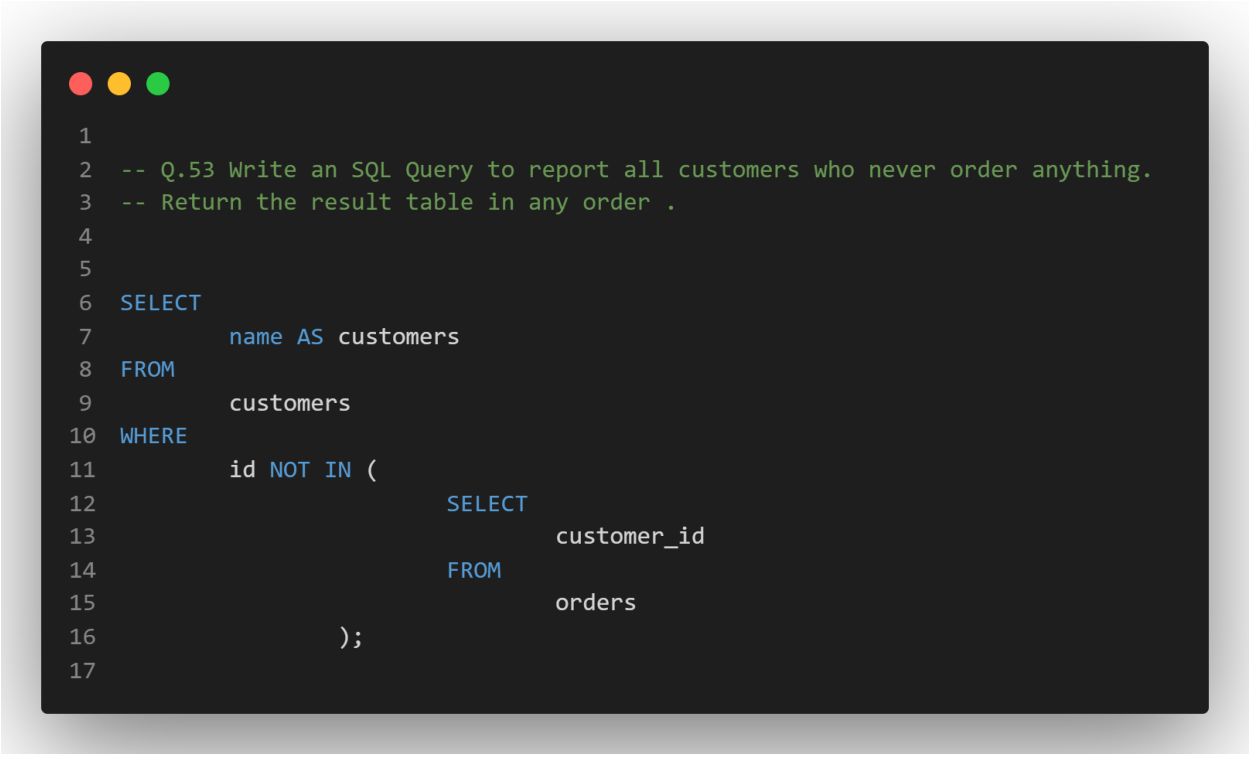
Question 51:

```sql

-- Q.51 Write an SQL Query to report the name, population, and area of the big countries.
-- Return the result table in any order .

SELECT
        name,
        population,
        area
FROM
        world
WHERE
        area > 3000000
        OR
        population > 25000000;

```

Question 52:

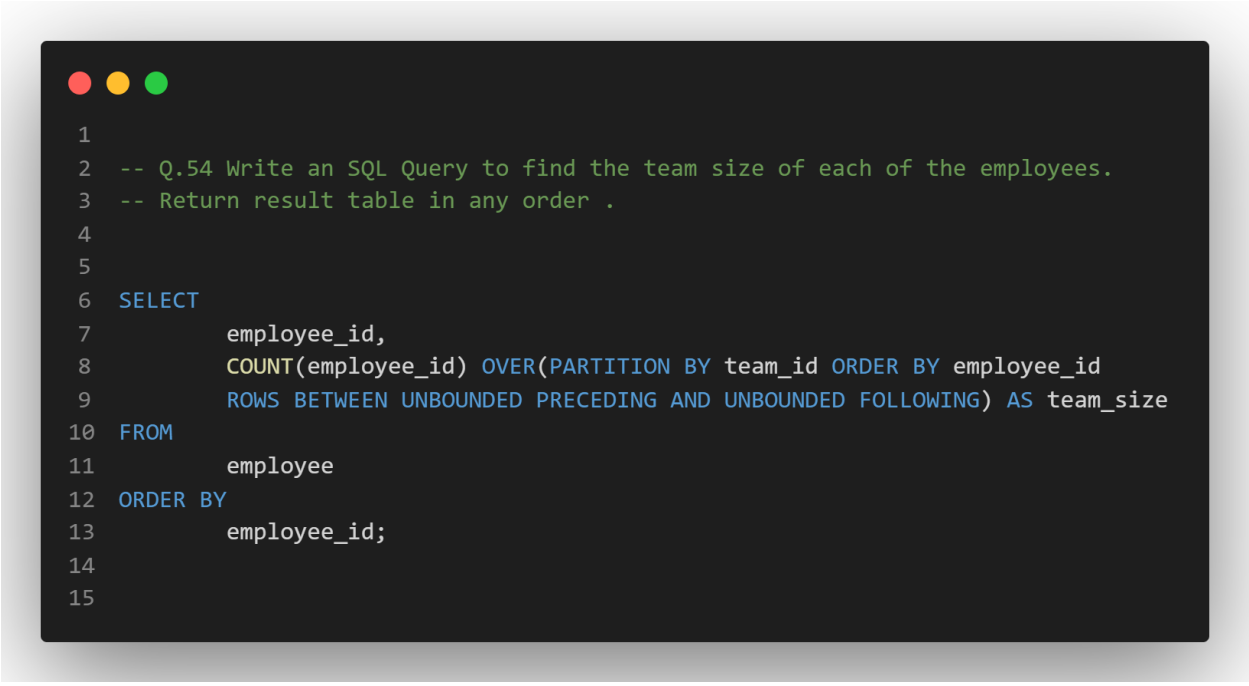
```sql

-- Q.52 Write an SQL Query to report the names of the customer that are not referred by the customer with id = 2.
-- Return the result table in any order.


SELECT
        name
FROM
        customer
WHERE
        refree_id <> 2
        OR
        refree_id IS NULL;

```

Question 53:



```sql

-- Q.53 Write an SQL Query to report all customers who never order anything.
-- Return the result table in any order .


SELECT
        name AS customers
FROM
        customers
WHERE
        id NOT IN (
                        SELECT
                                customer_id
                        FROM
                                orders
                );

```

Question 54:



```sql

-- Q.54 Write an SQL Query to find the team size of each of the employees.
-- Return result table in any order .


SELECT
        employee_id,
        COUNT(employee_id) OVER(PARTITION BY team_id ORDER BY employee_id
        ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS team_size
FROM
        employee
ORDER BY
        employee_id;


```

Question 55:

```sql

-- Q.55 Write an SQL Query to find the countries where this company can invest .
-- Return the result table in any order .


-- Approach 1


SELECT
        name AS country
FROM    (
                SELECT
                        c.name,
                        SUM(ca.duration) AS call_duration,
                        COUNT(c.country_code) AS number_of_calls
                FROM (
                        SELECT
                                id,
                                name,
                                CASE
                                WHEN LEFT(SUBSTR(phone_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone_number, 1,3), (LENGTH(SUBSTR(phone_number, 1,3))-1))
                                ELSE SUBSTR(phone_number, 1,3) END AS country_code
                        FROM
                                person
                ) temp_person
JOIN
        country c
ON
        temp_person.country_code = c.country_code
JOIN
        calls ca
ON
        temp_person.id = caller_id
GROUP BY
        c.name


UNION ALL



SELECT
        c.name,
        SUM(ca.duration) AS call_duration,
        COUNT(c.country_code) AS number_of_calls
FROM    (
                SELECT
                        id,
                        name,
                        CASE
                        WHEN LEFT(SUBSTR(phone_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone_number, 1,3), (length(SUBSTR(phone_number, 1,3))-1))
                        ELSE SUBSTR(phone_number, 1,3) END AS country_code
                FROM
                        person
        )       temp_person
JOIN
        country c
ON
        temp_person.country_code = c.country_code
JOIN
        calls ca
ON
        temp_person.id = ca.callee_id
GROUP BY
        c.name

        ) temp

GROUP BY
        name
HAVING
        SUM(call_duration)/SUM(number_of_calls) >  (SELECT AVG(duration) FROM calls);

```

```sql

-- Approach 2


 WITH temp_person AS (
                        SELECT
                                id,
                                name,
                                CASE
                                WHEN LEFT(SUBSTR(phone_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone_number, 1,3), (LENGTH(SUBSTR(phone_number, 1,3))-1))
                                ELSE SUBSTR(phone_number, 1,3) END AS country_code
                        FROM
                                person
                                 )


SELECT
        name AS country
FROM    (
        SELECT
                c.name,
                SUM(ca.duration) AS call_duration,
                COUNT(c.country_code) AS number_of_calls
        FROM
                temp_person
        JOIN
                country c
        ON
                temp_person.country_code = c.country_code
        JOIN
                calls ca
        ON
                temp_person.id = ca.caller_id
        GROUP BY
                c.name

UNION ALL

        SELECT
                c.name,
                SUM(ca.duration) AS call_duration,
                COUNT(c.country_code) AS number_of_calls
        FROM
                temp_person
        JOIN
                country c
        ON
                temp_person.country_code = c.country_code
        JOIN
                calls ca
        ON
                temp_person.id = ca.callee_id
        GROUP BY
                c.name

        )temp
GROUP BY
        name
HAVING
        SUM(call_duration)/SUM(number_of_calls) >  (SELECT AVG(duration) FROM calls);

```

Question 56:

```sql

-- Q.56 Write an SQL Query to report the device that is first logged in for each player.
-- Return the result table in any order.


SELECT
        player_id,
        device_id
FROM
        (
            SELECT
                    player_id,
                    device_id,
                    event_date,
                    ROW_number() OVER(PARTITION BY player_id ORDER BY event_date) ranking
            FROM
                    activity
        ) temp_activity
WHERE
        ranking = 1;

```

Question 57:

```sql

-- Q.57 Write an SQL Query to find the customer_number for the customer who has placed the largest number of orders.


WITH temp_orders AS (
                        SELECT
                                DISTINCT customer_number,
                                DENSE_RANK() OVER(ORDER BY total_orders DESC) AS ranking
                        FROM (
                                SELECT
                                        customer_number,
                                        COUNT(order_number) OVER(PARTITION BY customer_number) total_orders
                                FROM
                                        orders
                        )   temp_cust_details
                )

SELECT
        customer_number
FROM
        temp_orders
WHERE
        ranking = 1;

```

Question 58:

```sql

-- Q.58 Write an SQL Query to report all the consecutive available seats in the cinema.
-- Return the result table ordered by seat_id in ascending order.


SELECT
        DISTINCT c1.seat_id
FROM
        cinema c1
INNER JOIN
        cinema c2
ON
        ABS(c1.seat_id - c2.seat_id) = 1
        AND
        (c1.free = 1 AND c2.free = 1)
ORDER BY
        c1.seat_id;

```

Question 59:

```sql

-- Q.59 Write an SQL Query to report the names of all the salespersons who did not have any
-- orders related to the company with the name "RED".


SELECT
        name
FROM
        sales_person
WHERE
        sales_id NOT IN (
                            SELECT
                                    o.sales_id
                            FROM
                                    orders o
                            INNER JOIN
                                    company c
                            ON
                                    c.company_id = o.company_id
                            WHERE
                                    c.name = 'RED'
                        );

```

Question 60:

```sql

-- Q.60 Write an SQL Query to report for every three line segments whether they can form a triangle.
-- Return the result table in any order.


SELECT
        x,
        y,
        z,
        IF(x+y>z AND x+z>y AND y+z>x, 'YES','NO') AS is_triangle
FROM
        triangle;

```

Question 61:

```sql

-- Q.61 Write an SQL Query to report the shortest distance between any two points from the Point table.


SELECT
        MIN(ABS(c1.x - c2.x)) AS shortest_distance
FROM
        point c1
INNER JOIN
        point c2
WHERE
        c1.x!=c2.x;

```

Question 62:

```sql

-- Q.62 Write a SQL Query for a report that provides the pairs (actor_id, director_id) where the actor has
-- cooperated with the director at least three times. Return the result table in any order.


WITH temp_actor_director AS (
                                SELECT
                                        DISTINCT actor_id,
                                        director_id,
                                        DENSE_RANK() OVER(ORDER BY total_movies DESC) AS ranking
                                FROM (
                                        SELECT
                                                actor_id,
                                                director_id,
                                                COUNT(actor_id) OVER(PARTITION BY  actor_id, director_id) AS total_movies
                                        FROM
                                                actor_director
                                ) temp
                )

SELECT
        actor_id,
        director_id
FROM
        temp_actor_director
WHERE
        ranking = 1;

```

Question 63:

```sql

-- Q.63 Write an SQL Query that reports the product_name, year, and price for each sale_id in
-- the sales table. Return the resulting table in any order.


SELECT
        p.product_name,
        s.year,
        s.price
FROM
        sales s
INNER JOIN
        product p
ON
        p.product_id = s.product_id;

```

Question 64:

```sql

-- Q.64 Write an SQL Query that reports the average experience years of all the employees for each project,
-- rounded to 2 digits. Return the result table in any order.


-- Approach 1


SELECT
        p.project_id,
        ROUND(AVG(experience_years), 2) AS average_years
FROM
        employee e
INNER JOIN
        project p
ON
        p.employee_id = e.employee_id
GROUP BY
        project_id;


-- Approach 2


SELECT
        DISTINCT p.project_id,
        ROUND(AVG(experience_years) OVER(PARTITION BY project_id), 2) AS average_years
FROM
        employee e
INNER JOIN
        project p
ON
        p.employee_id = e.employee_id;

```

Question 65:

```sql

-- Q.65 Write an SQL Query that reports the best seller by total sales price, If there is a tie,
-- report them all. Return the result table in any order.


WITH temp_sales AS (
                SELECT
                        seller_id,
                        total_price,
                        DENSE_RANK() OVER (ORDER BY total_price DESC) ranking
                FROM
                    (
                        SELECT
                                s.seller_id,
                                SUM(s.quantity*p.unit_price) AS total_price
                        FROM
                                sales s
                        INNER JOIN
                                product p
                        ON
                            p.product_id = s.product_id
                        GROUP BY
                                seller_id
                    ) temp
             )


SELECT
        seller_id
FROM
        temp_sales
WHERE
        ranking = 1;

```

Question 66:

```sql

-- Q.66 Write an SQL Query that reports the buyers who have bought S8 but not iphone. Note that S8 and iphone
-- are products present in the product table. Return the result table in any order.


SELECT
        s.buyer_id
FROM
        sales s
INNER JOIN
        product p
ON
        p.product_id = s.product_id
WHERE
        p.product_name = 'S8'
        AND
        s.buyer_id NOT IN (
                                SELECT
                                        s.buyer_id
                                FROM
                                        sales S
                                INNER JOIN
                                        product P
                                ON
                                        s.product_id = p.product_id
                                WHERE
                                        p.product_name = 'Iphone'
                        );

```

Question 67:

```sql

-- Q.67 Write an SQL Query to compute the moving average of how much the customer paid in a seven days window
-- (i.e., current day + 6 days before). average_amount should be rounded to two decimal places.
-- Return result table ordered by visited_on in ascending order.


WITH temp_customer AS (
                        SELECT
                                visited_on,
                                SUM(amount) AS amount
                        FROM
                                customer
                        GROUP BY
                                visited_on
                ),

    temp_customer2 AS (
                        SELECT
                                visited_on,
                                SUM(amount) OVER(ORDER BY visited_on ROWS BETWEEN 6  PRECEDING AND CURRENT ROW) AS weekly_amount,
                                ROUND(AVG(amount) OVER(ORDER BY visited_on ROWS BETWEEN 6 PRECEDING AND CURRENT ROW), 2) AS average_amount,
                                DENSE_RANK() OVER(ORDER BY visited_on) as ranking
                        FROM
                                temp_customer
                )

SELECT
        visited_on,
        weekly_amount,
        average_amount
FROM
        temp_customer2
WHERE
        ranking > 6;

```

Question 68:

```sql

-- Q.68 Write an SQL Query to find the total score for each gender on each day.
-- Return the result table ordered by gender and day in ascending order.


SELECT
        gender,
        day,
        SUM(score_points) OVER(PARTITION BY gender ORDER BY day
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS total_points
FROM
        scores
order by
        gender,
        day;

```

Question 69:

```sql

-- Q.69 Write an SQL Query to find the start and end number of continuous ranges in the table logs.
--  Return the result table ordered by start_id.


SELECT
        MIN(log_id) AS start_id,
        MAX(log_id) AS end_id
FROM
        (
                SELECT
                        log_id,
                        DENSE_RANK() OVER(ORDER BY log_id - RN) AS ranking
                FROM
                    (
                        SELECT
                                log_id,
                                ROW_number() OVER(ORDER BY log_id) AS RN
                        FROM
                                logs
                    )   temp_log
        ) temp_log2
GROUP BY
        ranking
ORDER BY
        start_id;

```

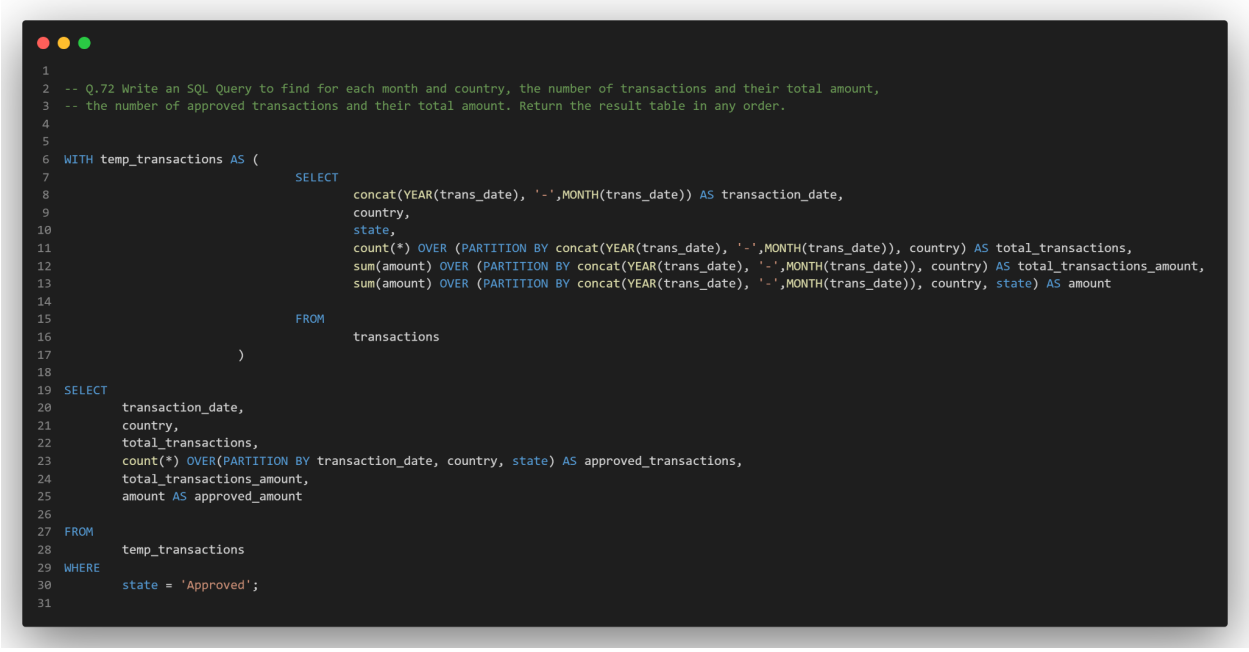
Question 70:

```sql

-- Q.70 Write an SQL Query to find the number of times each student attended each exam.
-- Return the result table ordered by student_id and subject_name.


WITH temp_student AS (
                        SELECT
                                student_id,
                                student_name,
                                subject_name
                        FROM
                                students,
                                subjects
                                        ),

    temp_student2 AS (
                        SELECT
                                student_id,
                                subject_name,
                                COUNT(*) AS times_attended_each_exam
                        FROM
                                exams
                        GROUP BY
                                student_id,
                                subject_name
                                        )

SELECT
        t.student_id,
        t.student_name,
        t.subject_name,
        COALESCE(times_attended_each_exam,0) AS attended_exams
FROM
        temp_student t
LEFT JOIN
        temp_student2 t2
ON
        t.student_id = t2.student_id
        AND
        t.subject_name = t2.subject_name
ORDER BY
        t.student_id,
        t.subject_name;

```

Question 71:

```sql

-- Q.71 Write an SQL Query to find employee_id of all employees that directly or indirectly
-- report their work to the head of the company. The indirect relation between managers will not exceed
-- three managers as the company is small. Return the result table in any order.


with recursive managers as (
                                SELECT
                                        employee_id,
                                        manager_id
                                FROM
                                        employees
                                WHERE
                                        employee_id = 1

                                UNION

                                SELECT
                                        e.employee_id,
                                        m.manager_id
                                FROM
                                        managers m
                                INNER JOIN
                                        employees e
                                ON
                                        e.manager_id = m.employee_id
                )

SELECT
        employee_id
FROM
        managers
WHERE
        employee_id <> manager_id;

```

Question 72:



```sql

-- Q.72 Write an SQL Query to find for each month and country, the number of transactions and their total amount,
-- the number of approved transactions and their total amount. Return the result table in any order.


WITH temp_transactions AS (
                                SELECT
                                        concat(YEAR(trans_date), '-',MONTH(trans_date)) AS transaction_date,
                                        country,
                                        state,
                                        count(*) OVER (PARTITION BY concat(YEAR(trans_date), '-',MONTH(trans_date)), country) AS total_transactions,
                                        sum(amount) OVER (PARTITION BY concat(YEAR(trans_date), '-',MONTH(trans_date)), country) AS total_transactions_amount,
                                        sum(amount) OVER (PARTITION BY concat(YEAR(trans_date), '-',MONTH(trans_date)), country, state) AS amount

                                FROM
                                        transactions
                        )

SELECT
        transaction_date,
        country,
        total_transactions,
        count(*) OVER(PARTITION BY transaction_date, country, state) AS approved_transactions,
        total_transactions_amount,
        amount AS approved_amount

FROM
        temp_transactions
WHERE
        state = 'Approved';

```

Question 73:

```sql

-- Q.73 Write an SQL Query to find the average daily percentage of posts that got
-- removed after being reported as spam, rounded to 2 decimal places.


WITH temp_action AS (
                        SELECT
                                action_date,
                                post_id,
                                COUNT(EXTRA) OVER(PARTITION BY action_date) num_post_reported_spam
                        FROM
                                actions
                        WHERE
                                extra = 'SPAM'
                )

SELECT
        ROUND(AVG(percentage), 2) AS avg_daily_percent
FROM
        (
                SELECT
                        action_date,
                        ROUND((COUNT(post_id)/num_post_reported_spam) * 100, 2) AS percentage
                FROM
                        temp_action
                WHERE
                        post_id IN (
                                        SELECT
                                                post_id
                                        FROM
                                                removals
                                )
                GROUP BY
                        action_date
        ) temp;

```

Question 74:

```sql

-- Q.74 Write an SQL query to report the fraction of players that logged in again on the day after the day
-- they first logged in, rounded to 2 decimal places. In other words, you need to count the number of players
-- that logged in for at least two consecutive days starting from their first login date,
-- then divide that number by the total number of players.


-- Approach 1


WITH temp_activity AS (
                        SELECT
                                player_id,
                                LEAD(event_date, 1) OVER(PARTITION BY player_id ORDER BY event_date)  - event_date  AS difference
                        FROM
                                activity
            ),

    temp_activity2 AS (
                        SELECT
                                COUNT(DISTINCT player_id) AS players_count
                        FROM
                                temp_activity
                        WHERE
                                difference = 1
                        GROUP BY
                                player_id
            )

SELECT
        ROUND(COUNT(*) / (SELECT COUNT(DISTINCT player_id) FROM activity), 2) AS fraction
FROM
        temp_activity2;


-- Approach 2


WITH temp_activity AS (
                        SELECT
                                player_id,
                                event_date,
                                DATEDIFF(event_date,lag(event_date) OVER(PARTITION BY player_id ORDER BY event_date)) AS difference
                        FROM
                                activity
            ),

    temp_activity2 AS (
                        SELECT
                                COUNT(DISTINCT player_id) AS players_count
                        FROM
                                temp_activity
                        WHERE
                                difference = 1
                        GROUP BY
                                player_id
            )


SELECT
        ROUND(COUNT(*) / (SELECT COUNT(DISTINCT player_id) FROM activity), 2) AS fraction
FROM
        temp_activity2;

```

Question 75:

```sql

-- Q.75 Write an SQL query to report the fraction of players that logged in again on the day after the day
-- they first logged in, rounded to 2 decimal places. In other words, you need to count the number of players
-- that logged in for at least two consecutive days starting from their first login date,
-- then divide that number by the total number of players.


-- Approach 1


WITH temp_activity AS (
                        SELECT
                                player_id,
                                LEAD(event_date, 1) OVER(PARTITION BY player_id ORDER BY event_date)  - event_date  AS difference
                        FROM
                                activity
            ),

    temp_activity2 AS (
                        SELECT
                                COUNT(DISTINCT player_id) AS players_count
                        FROM
                                temp_activity
                        WHERE
                                difference = 1
                        GROUP BY
                                player_id
            )

SELECT
        ROUND(COUNT(*) / (SELECT COUNT(DISTINCT player_id) FROM activity), 2) AS fraction
FROM
        temp_activity2;


-- Approach 2


WITH temp_activity AS (
                        SELECT
                                player_id,
                                event_date,
                                DATEDIFF(event_date,lag(event_date) OVER(PARTITION BY player_id ORDER BY event_date)) AS difference
                        FROM
                                activity
            ),

    temp_activity2 AS (
                        SELECT
                                COUNT(DISTINCT player_id) AS players_count
                        FROM
                                temp_activity
                        WHERE
                                difference = 1
                        GROUP BY
                                player_id
            )


SELECT
        ROUND(COUNT(*) / (SELECT COUNT(DISTINCT player_id) FROM activity), 2) AS fraction
FROM
        temp_activity2;

```

Question 76:

```sql

 -- Q.76 Write an SQL Query to find the salaries of the employees after applying taxes.
 -- Round the salary to the nearest integer.


WITH temp_salaries AS (
                        SELECT
                                company_id,
                                employee_id,
                                employee_name,
                                salary,
                                MAX(salary) OVER(PARTITION BY company_id) max_sal_per_company
                        FROM
                                salaries
                )

SELECT
        company_id,
        employee_id,
        employee_name,
        salary,
        ROUND(
                CASE
                    WHEN max_sal_per_company > 10000 THEN salary - (salary * 0.49)
                    WHEN max_sal_per_company BETWEEN 1000 AND 10000 THEN salary - (salary * 0.24)
                    ELSE salary
                END, 0) AS sal_after_tax_deduction
FROM
        temp_salaries;

```

Question 77:

```sql

-- Q.77 Write an SQL Query to evaluate the boolean expressions in Expressions table.
-- Return the result table in any order.


SELECT
        e.left_operand,
        e.operator,
        e.right_operand,
        CASE
                WHEN e.operator = '<' THEN IF(l.value < r.value, 'TRUE', 'FALSE')
                WHEN e.operator = '>' THEN IF(l.value > r.value, 'TRUE', 'FALSE')
                ELSE IF(l.value = r.value, 'TRUE', 'FALSE')
                END AS result
FROM
        expressions e
JOIN
        variables l
ON
        e.left_operand = l.name
JOIN
        variables r
ON
        e.right_operand = r.name;

```

Question 78:

```sql

-- Q.78 Write an SQL Query to find the countries where this company can invest .
-- Return the result table in any order .


-- Approach 1


SELECT
        name AS country
FROM    (
                SELECT
                        c.name,
                        SUM(ca.duration) AS call_duration,
                        COUNT(c.country_code) AS number_of_calls
                FROM (
                        SELECT
                                id,
                                name,
                                CASE
                                WHEN LEFT(SUBSTR(phone_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone_number, 1,3), (LENGTH(SUBSTR(phone_number, 1,3))-1))
                                ELSE SUBSTR(phone_number, 1,3) END AS country_code
                        FROM
                                person
                ) temp_person
JOIN
        country c
ON
        temp_person.country_code = c.country_code
JOIN
        calls ca
ON
        temp_person.id = caller_id
GROUP BY
        c.name


UNION ALL



SELECT
        c.name,
        SUM(ca.duration) AS call_duration,
        COUNT(c.country_code) AS number_of_calls
FROM    (
                SELECT
                        id,
                        name,
                        CASE
                        WHEN LEFT(SUBSTR(phone_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone_number, 1,3), (length(SUBSTR(phone_number, 1,3))-1))
                        ELSE SUBSTR(phone_number, 1,3) END AS country_code
                FROM
                        person
        )       temp_person
JOIN
        country c
ON
        temp_person.country_code = c.country_code
JOIN
        calls ca
ON
        temp_person.id = ca.callee_id
GROUP BY
        c.name

        ) temp

GROUP BY
        name
HAVING
        SUM(call_duration)/SUM(number_of_calls) >  (SELECT AVG(duration) FROM calls);

```

```sql

-- Approach 2


 WITH temp_person AS (
                        SELECT
                                id,
                                name,
                                CASE
                                WHEN LEFT(SUBSTR(phone_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone_number, 1,3), (LENGTH(SUBSTR(phone_number, 1,3))-1))
                                ELSE SUBSTR(phone_number, 1,3) END AS country_code
                        FROM
                                person
                                 )


SELECT
        name AS country
FROM    (
        SELECT
                c.name,
                SUM(ca.duration) AS call_duration,
                COUNT(c.country_code) AS number_of_calls
        FROM
                temp_person
        JOIN
                country c
        ON
                temp_person.country_code = c.country_code
        JOIN
                calls ca
        ON
                temp_person.id = ca.caller_id
        GROUP BY
                c.name

UNION ALL

        SELECT
                c.name,
                SUM(ca.duration) AS call_duration,
                COUNT(c.country_code) AS number_of_calls
        FROM
                temp_person
        JOIN
                country c
        ON
                temp_person.country_code = c.country_code
        JOIN
                calls ca
        ON
                temp_person.id = ca.callee_id
        GROUP BY
                c.name

        )temp
GROUP BY
        name
HAVING
        SUM(call_duration)/SUM(number_of_calls) >  (SELECT AVG(duration) FROM calls);

```

Question 79:

```sql

-- Q.79 Write a Query that prints a list of employee names (i.e.: the name attribute)
-- from the employee table in alphabetical order.


SELECT
        name
FROM
        employee
ORDER BY
        name;

```

Question 80:

```sql

-- Q.80 Write a Query to obtain the year-on-year growth rate for the total spend of each product for each year.


-- Approach 1


WITH temp_transactions AS (
                                SELECT
                                        product_id,
                                        transaction_date,
                                        spend AS curr_year_spend,
                                        LAG(spend,1,0) OVER w AS prev_year_spend,
                                        IFNULL(spend - LAG(spend,1) OVER w, 0) AS prev_curr_spend_diff
                                FROM
                                        user_transactions
                                WINDOW
                                        w AS (PARTITION BY product_id ORDER BY EXTRACT(YEAR FROM transaction_date))
                        )

SELECT
        product_id,
        curr_year_spend,
        ROUND(prev_year_spend, 2),
        IFNULL(ROUND((prev_curr_spend_diff * 100)/prev_year_spend,2),0) AS YOY
FROM
        temp_transactions;



-- Approach 2


WITH temp_transactions AS (
                                SELECT
                                        YEAR(STR_TO_date(transaction_date, '%m/%d/%Y')) AS YEAR_id,
                                        product_id,
                                        transaction_date,
                                        spend AS curr_year_spend,
                                        LAG(spend,1) OVER w AS prev_year_spend,
                                        spend - LAG(spend,1) OVER w AS prev_curr_spend_diff
                                FROM
                                        user_transactions
                                WINDOW
                                        w AS (PARTITION BY product_id ORDER BY EXTRACT(YEAR FROM transaction_date))
                        )

SELECT
        year_id,
        product_id,
        curr_year_spend,
        ROUND(prev_year_spend, 2),
        ROUND((prev_curr_spend_diff * 100)/prev_year_spend,2) AS YOY
FROM
        temp_transactions;


-- Approach 3


SELECT
        product_id,
        YEAR(STR_TO_date(transaction_date, '%m/%d/%Y')) AS YEAR_id,
        spend AS curr_year_spend,
        ROUND(LAG(spend,1,0) OVER w ,2) AS prev_year_spend,
        ROUND((spend - LAG(spend,1) OVER w ) * 100 /
                    LAG(spend,1) OVER w, 2)  AS YOY
FROM
        user_transactions
WINDOW
        w AS (PARTITION BY product_id ORDER BY EXTRACT(YEAR FROM transaction_date));

```

Question 81:

```sql

-- Q.81 Write a SQL Query to find the number of prime and non-prime items that can be stored
-- in the 500,000 square feet warehouse. Output the item type and number of items to be stocked.


-- Approach 1


WITH temp_inventory AS (
                        SELECT
                                item_type,
                                SUM(square_foot) AS square_foot_per_category,
                                COUNT(*) AS count_of_items
                        FROM
                                inventory
                        GROUP BY
                                item_type
                ),

    temp_inventory2 AS (
                        SELECT
                                (500000 - SUM(square_foot_per_category)*FLOOR(500000/SUM(square_foot_per_category))) AS area_left
                        FROM
                                temp_inventory
                        WHERE
                                item_type = 'PRIME_ELIGIBLE'
                ),

    temp_inventory3 AS (
                        SELECT
                                item_type,
                                CASE
                                        WHEN item_type = 'PRIME_ELIGIBLE'
                                                THEN FLOOR(500000/square_foot_per_category) * count_of_items
                                        WHEN item_type = 'NOT_PRIME'
                                                THEN FLOOR((SELECT area_left FROM temp_inventory2) / square_foot_per_category) * count_of_items
                                        END AS item_count
                        FROM
                                temp_inventory
                )

SELECT
        item_type,
        item_count
 FROM
        temp_inventory3;


```

```sql

-- Approach 2


WITH temp_inventory AS (
                        SELECT
                                item_type,
                                SUM(square_foot) AS square_foot_per_category,
                                COUNT(*) AS count_of_items,
                                CASE
                                        WHEN item_type = 'PRIME_ELIGIBLE' THEN FLOOR(500000/SUM(square_foot)) * COUNT(*)
                                        END AS prime_items_count
                        FROM
                                inventory
                        GROUP BY
                                item_type
                ),

    temp_inventory2 AS (
                        SELECT
                                (500000 - SUM(square_foot_per_category)*FLOOR(500000/SUM(square_foot_per_category))) AS area_left
                        FROM
                                temp_inventory
                        WHERE
                                item_type = 'PRIME_ELIGIBLE'
                ),

    temp_inventory3 AS (
                        SELECT
                                item_type,
                                CASE
                                        WHEN item_type = 'PRIME_ELIGIBLE'
                                                THEN prime_items_count
                                        WHEN item_type = 'NOT_PRIME'
                                                THEN FLOOR((SELECT area_left FROM temp_inventory2) / square_foot_per_category) * count_of_items
                                END AS item_count
                        FROM
                                temp_inventory
                )

SELECT
        item_type,
        item_count
FROM
        temp_inventory3;


```

Question 82:

```sql

-- Q.82 Write a Query to obtain the active user retention in July 2022.
-- Output the month (in numerical format 1, 2, 3) and the number of monthly active users (MAUs).


-- Approach 1


WITH temp_actions AS (
                        SELECT
                                user_id,
                                event_date,
                                event_type,
                                SUBSTR(event_date, 6, 2) - lag(SUBSTR(event_date, 6, 2)) OVER w AS difference
                        FROM
                                user_actions
                        WINDOW
                                w as (PARTITION BY user_id ORDER BY event_date)
                ),

        temp_actions2 AS (
                        SELECT
                                SUBSTR(event_date, 6, 2) AS months,
                                COUNT(user_id) AS monthly_active_users
                        FROM
                                temp_actions
                        WHERE
                                difference = 1 AND event_type IN ('LIKE', 'COMMENT', 'SIGN-IN')
                        GROUP BY
                                months
                                )

SELECT
        months,
        monthly_active_users
FROM
        temp_actions2;


-- Approach 2


 WITH temp_actions AS (
                        SELECT
                                user_id,
                                event_date,
                                event_type,
                                SUBSTR(event_date, 6, 2) - lag(SUBSTR(event_date, 6, 2)) OVER w AS difference
                        FROM
                                user_actions
                        WINDOW
                                w as (PARTITION BY user_id ORDER BY event_date)
                )


SELECT
        SUBSTR(event_date, 6, 2) AS months,
        COUNT(DISTINCT user_id) AS active_users
FROM
        temp_actions
WHERE
        difference = 1
        AND
        event_type IN ('LIKE', 'COMMENT', 'SIGN-IN')
GROUP BY
        months;

```

Question 83:

```sql

-- Q.83 Write a Query to report the median of searches made by a user.
-- Round the median to one decimal point.


 WITH temp_search_freq AS (
                                SELECT
                                        searches,
                                        num_users,
                                        ROW_NUMBER() OVER(ORDER BY searches) row_num,
                                        COUNT(*) OVER(ORDER BY searches ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) total_records
                                FROM
                                        search_frequency
                        ),

    temp_search_freq2 as (
                                SELECT
                                        searches,
                                        num_users,
                                        CASE
                                        WHEN total_records % 2 <>  0 THEN (
                                                                                SELECT
                                                                                        DISTINCT ROUND(SUM(searches) OVER w /
                                                                                        COUNT(*) OVER w,1)
                                                                                FROM
                                                                                        temp_search_freq
                                                                                WHERE
                                                                                        row_num = ROUND((total_records + 1) / 2, 0)
                                                                                WINDOW
                                                                                        w AS (ORDER BY searches ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)

                                                                                )

                                        WHEN total_records % 2 = 0 THEN (
                                                                                SELECT
                                                                                        DISTINCT ROUND(SUM(searches) OVER w /
                                                                                        COUNT(*) OVER w,1)
                                                                                FROM
                                                                                        temp_search_freq
                                                                                WHERE
                                                                                        row_num IN (total_records/2,(total_records/2)+1)
                                                                                WINDOW
                                                                                        w AS (ORDER BY searches ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
                                                                                )
                                        END AS median
                                FROM
                                        temp_search_freq
           )


SELECT
        DISTINCT median
FROM
        temp_search_freq2;

```

Question 84:

```sql

-- Q.84 Write a Query to update the Facebook advertisers status using the daily_pay table.
-- Advertiser is a two-column table containing the user id and their payment status based
-- on the last payment and daily_pay table has current information about their payment.
-- Only advertisers who paid will show up in this table.
-- Output the user id and current payment status sorted by the user id.


SELECT
        user_id,
        CASE
            WHEN user_id IN (SELECT user_id FROM daily_pay) THEN 'EXISTING'
            ELSE 'CHURN'
            END AS new_status
FROM
        advertiser
ORDER BY
        user_id;

```

Question 85:

```sql

-- Q.85 Write a SQL Query that calculates the total time that the fleet of
-- servers was running. The output should be in units of full days.


-- Approach 1


SELECT
                stop_time - start_time AS total_up_time
FROM
                (
                        SELECT
                                SUM(
                                        CASE
                                                WHEN session_status = 'start' then EXTRACT(DAY from STR_TO_DATE(status_time, '%m/%d/%y'))
                                                END
                                        )       AS start_time,
                                SUM(
                                        CASE
                                                WHEN session_status = 'stop' then EXTRACT(DAY from STR_TO_DATE(status_time, '%m/%d/%y'))
                                                END
                                        )       AS stop_time
                        FROM
                                        server_utilization
        ) temp_server_utilization;


-- Approach 2 (Works only in PostgreSQL)


WITH temp_server_utilization AS (
                                        SELECT
                                                server_id,
                                                status_time AS start_time,
                                                session_status,
                                                lead(status_time) OVER(ORDER BY server_id,status_time) AS end_time
                                        FROM
                                                server_utilization
                                )

SELECT
        EXTRACT(DAY FROM justify_hours(SUM(end_time - start_time))) as total_time
FROM
        temp_server_utilization
WHERE
        session_status = 'start';

```

Question 86:

```sql

-- Q.86 Sometimes, payment transactions are repeated by accident; it could be due to user error,
-- API failure or a retry error that causes a credit card to be charged twice.
-- Using the transactions table, identify any payments made at the same merchant with the
-- same credit card for the same amount within 10 minutes of each other. Count such repeated payments.


CREATE TABLE transactions(
        transaction_id INT,
        merchant_id INT,
        credit_card_id INT,
        amount INT,
        transaction_timestamp DATETIME
    );


INSERT INTO transactions VALUES
        (1,101,1,100,'2022-09-25 12:00:00'),
        (2,101,1,100,'2022-09-25 12:08:00'),
        (3,101,1,100,'2022-09-25 12:28:00'),
        (4,102,2,300,'2022-09-25 12:00:00'),
        (5,102,2,400,'2022-09-25 14:00:00');

-- Q.86 Sometimes, payment transactions are repeated by accident; it could be due to user error,
-- API failure or a retry error that causes a credit card to be charged twice.
-- Using the transactions table, identify any payments made at the same merchant with the
-- same credit card for the same amount within 10 minutes of each other. Count such repeated payments.


WITH temp_transactions AS (
                                SELECT
                                        merchant_id,
                                        credit_card_id,
                                        amount,
                                        transaction_timestamp,
                                        LAG(transaction_timestamp) OVER w AS prev_tran_timestamp,
                                        timestampdiff(MINUTE,LAG(transaction_timestamp) OVER w, transaction_timestamp) AS difference
                                FROM
                                        transactions
                                WINDOW
                                        w as (PARTITION BY credit_card_id ORDER BY  transaction_timestamp)
            )

SELECT
        COUNT(DISTINCT merchant_id) AS payment_count
FROM
        temp_transactions
WHERE
        difference <= 10;


-- Approach 2 (Works only in PostgreSQL)


WITH temp_transactions AS (
                                SELECT
                                        merchant_id,
                                        credit_card_id,
                                        amount,
                                        transaction_timestamp,
                                        LAG(transaction_timestamp) OVER w AS prev_tran_timestamp,
                                        EXTRACT(EPOCH FROM LAG(transaction_timestamp) OVER w -  transaction_timestamp) AS difference
                                FROM
                                        transactions
                                WINDOW
                                        w as (PARTITION BY credit_card_id ORDER BY  transaction_timestamp)
                        )

 SELECT
        COUNT(DISTINCT merchant_id) AS payment_count
FROM
        temp_transactions
 WHERE
        difference < 10;

```

Question 87:

```sql

-- Q.87 Write a SQL Query to find the bad experience rate in the first 14 days for new users who signed
-- up in June 2022. Output the percentage of bad experience rounded to 2 decimal places.


SELECT
        ROUND(
            SUM(
                CASE
                    WHEN status !='completed successfully'  THEN 1 ELSE 0
                    END
                )*100.0/count(*),2)  AS bad_experience_pct
FROM
        customers C
INNER JOIN
        orders O
ON
        o.customer_id = c.customer_id
WHERE
        o.order_timestamp < date_add(STR_TO_date(signup_timestamp, '%m/%d/%Y'), INTERVAL 14 DAY)
        AND
        MONTH(STR_TO_date(signup_timestamp, '%m/%d/%Y')) = 06
        AND
        YEAR(STR_TO_date(signup_timestamp, '%m/%d/%Y')) = 2022;

```

Question 88:

```sql

-- Q.88 Write an SQL Query to find the total score for each gender on each day.
-- Return the result table ordered by gender and day in ascending order.


SELECT
        gender,
        day,
        SUM(score_points) OVER(PARTITION BY gender ORDER BY day
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS total_points
FROM
        scores
order by
        gender,
        day;

```

Question 89:

```sql

-- Q.89 Write an SQL Query to find the countries where this company can invest .
-- Return the result table in any order .


-- Approach 1


SELECT
        name AS country
FROM    (
                SELECT
                        c.name,
                        SUM(ca.duration) AS call_duration,
                        COUNT(c.country_code) AS number_of_calls
                FROM (
                        SELECT
                                id,
                                name,
                                CASE
                                WHEN LEFT(SUBSTR(phone_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone_number, 1,3), (LENGTH(SUBSTR(phone_number, 1,3))-1))
                                ELSE SUBSTR(phone_number, 1,3) END AS country_code
                        FROM
                                person
                ) temp_person
JOIN
        country c
ON
        temp_person.country_code = c.country_code
JOIN
        calls ca
ON
        temp_person.id = caller_id
GROUP BY
        c.name


UNION ALL



SELECT
        c.name,
        SUM(ca.duration) AS call_duration,
        COUNT(c.country_code) AS number_of_calls
FROM    (
                SELECT
                        id,
                        name,
                        CASE
                        WHEN LEFT(SUBSTR(phone_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone_number, 1,3), (length(SUBSTR(phone_number, 1,3))-1))
                        ELSE SUBSTR(phone_number, 1,3) END AS country_code
                FROM
                        person
        )       temp_person
JOIN
        country c
ON
        temp_person.country_code = c.country_code
JOIN
        calls ca
ON
        temp_person.id = ca.callee_id
GROUP BY
        c.name

        ) temp

GROUP BY
        name
HAVING
        SUM(call_duration)/SUM(number_of_calls) >  (SELECT AVG(duration) FROM calls);


```

```sql

-- Approach 2


 WITH temp_person AS (
                        SELECT
                                id,
                                name,
                                CASE
                                WHEN LEFT(SUBSTR(phone_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone_number, 1,3), (LENGTH(SUBSTR(phone_number, 1,3))-1))
                                ELSE SUBSTR(phone_number, 1,3) END AS country_code
                        FROM
                                person
                                 )


SELECT
        name AS country
FROM    (
        SELECT
                c.name,
                SUM(ca.duration) AS call_duration,
                COUNT(c.country_code) AS number_of_calls
        FROM
                temp_person
        JOIN
                country c
        ON
                temp_person.country_code = c.country_code
        JOIN
                calls ca
        ON
                temp_person.id = ca.caller_id
        GROUP BY
                c.name

UNION ALL

        SELECT
                c.name,
                SUM(ca.duration) AS call_duration,
                COUNT(c.country_code) AS number_of_calls
        FROM
                temp_person
        JOIN
                country c
        ON
                temp_person.country_code = c.country_code
        JOIN
                calls ca
        ON
                temp_person.id = ca.callee_id
        GROUP BY
                c.name

        )temp
GROUP BY
        name
HAVING
        SUM(call_duration)/SUM(number_of_calls) >  (SELECT AVG(duration) FROM calls);

```

Question 90:

```sql

-- Q.90 Write an SQL Query to report the median of all the numbers in the database
-- after decompressing the numbers table. Round the median to one decimal point.


WITH RECURSIVE num_frequency (num,frequency, i) AS
                        (
                                SELECT
                                        num,
                                        frequency,1
                                FROM
                                        numbers

                                UNION ALL

                                SELECT
                                        num,
                                        frequency,
                                        i+1
                                FROM
                                        num_frequency
                                WHERE
                                        num_frequency.i < num_frequency.frequency
                        ),

        num_frequency2 AS (
                                SELECT
                                        num,
                                        frequency,
                                        row_number() OVER(ORDER BY num, frequency) AS row_num,
                                        COUNT(*) OVER(ORDER BY num, frequency ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS total_records
                                FROM
                                        num_frequency
                        )

SELECT
        DISTINCT CASE
                        WHEN total_records % 2 <>  0 THEN (
                                SELECT
                                        DISTINCT ROUND(SUM(num) OVER w /
                                        COUNT(*) OVER  w, 1)
                                FROM
                                        num_frequency2
                                WHERE
                                        row_num = ROUND((total_records + 1) / 2, 0)
                                WINDOW
                                        w as (ORDER BY  num, frequency ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING))

                    WHEN total_records % 2 = 0 THEN (
                                SELECT
                                        DISTINCT ROUND(SUM(num) OVER w /
                                        COUNT(*) OVER w, 1)
                                FROM
                                        num_frequency2
                                WHERE
                                        row_num IN (total_records/2,(total_records/2)+1)
                                WINDOW
                                        w as (ORDER BY  num, frequency ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING))
                    END AS median
FROM
        num_frequency2;

```

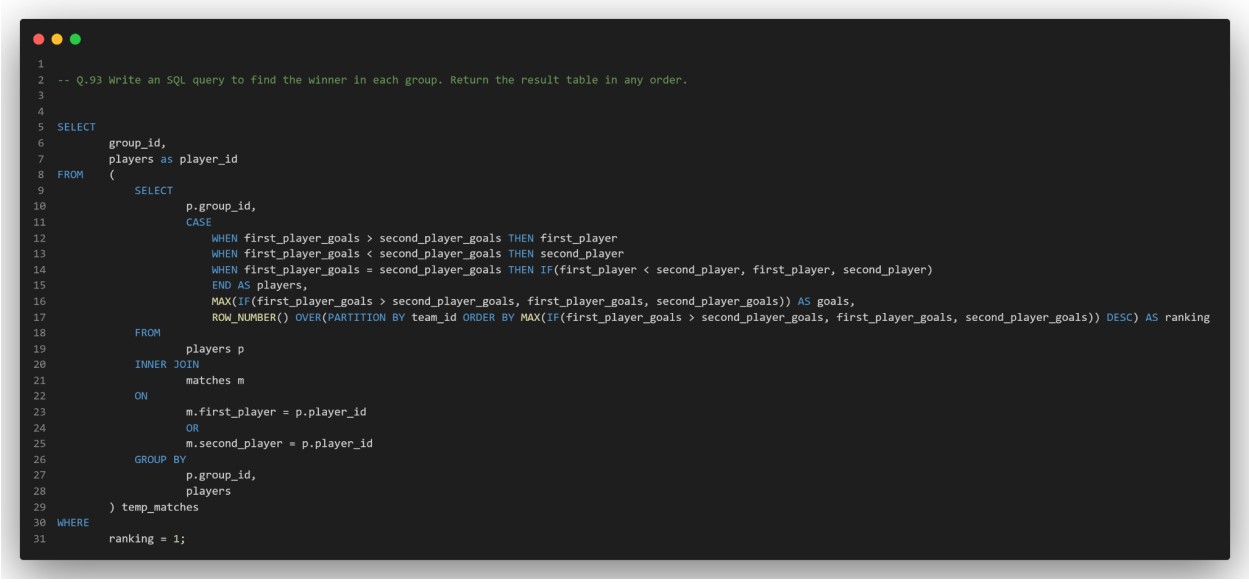
Question 91:

```sql

-- Q.91 Write an SQL Query to report the comparison result (higher/lower/same) of the average salary of
-- employees in a department to the companys average salary. Return the result table in any order.


WITH temp_comparison AS (
                                SELECT
                                        s.employee_id,
                                        e.department_id,
                                        s.amount,
                                        s.paydate,
                                        avg(amount) OVER (PARTITION BY MONTH(paydate) ORDER BY month(paydate), employee_id
                                                                ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) company_avg_salary,
                                        avg(amount) OVER (PARTITION BY MONTH(paydate), department_id order by month(paydate)
                                                                ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) department_avg
                                FROM
                                        salary s
                                INNER JOIN
                                        employee e
                                ON
                                        e.employee_id = s.employee_id
                        )

 SELECT
        DISTINCT DATE_FORMAT(paydate, '%Y-%m') AS pay_month,
        department_id,
        CASE
            WHEN company_avg_salary = department_avg THEN 'same'
            WHEN company_avg_salary > department_avg THEN 'lower'
            WHEN company_avg_salary < department_avg THEN 'higher'
            END AS comparison
 FROM
        temp_comparison;

```

Question 92:

```sql

-- Q.92 Write an SQL Query to report for each install date, the number of players
-- that installed the game on that day, and the day one retention.


-- Approach 1


SELECT
        a.event_date AS install_date,
        COUNT(a.player_id) AS installs,
        ROUND(COUNT(b.player_id) / COUNT(a.player_id), 2) AS day1_retention
FROM
        (
                SELECT
                        player_id,
                        MIN(event_date) AS event_date
                FROM
                        activity
                GROUP BY
                        player_id
        ) a
LEFT JOIN
        activity b
ON
        a.player_id = b.player_id
        AND
        a.event_date + 1 = b.event_date
GROUP BY
        a.event_date;


-- Approach 2


SELECT
        a1.event_date AS install_dt,
        COUNT(a1.player_id) AS installs,
        ROUND(COUNT(a3.player_id) / COUNT(a1.player_id), 2) AS day1_retention
FROM
        activity a1
LEFT JOIN
        activity a2
ON
        a1.player_id = a2.player_id
        AND
        a1.event_date > a2.event_date
LEFT JOIN
        activity a3
ON
        a1.player_id = a3.player_id
        AND
        DATEDIFF(a3.event_date, a1.event_date) = 1
WHERE
        a2.event_date IS NULL
GROUP BY
        a1.event_date;

```

Question 93:



```sql

-- Q.93 Write an SQL query to find the winner in each group. Return the result table in any order.


SELECT
        group_id,
        players as player_id
FROM    (
            SELECT
                    p.group_id,
                    CASE
                        WHEN first_player_goals > second_player_goals THEN first_player
                        WHEN first_player_goals < second_player_goals THEN second_player
                        WHEN first_player_goals = second_player_goals THEN IF(first_player < second_player, first_player, second_player)
                        END AS players,
                        MAX(IF(first_player_goals > second_player_goals, first_player_goals, second_player_goals)) AS goals,
                        ROW_NUMBER() OVER(PARTITION BY team_id ORDER BY MAX(IF(first_player_goals > second_player_goals, first_player_goals, second_player_goals)) DESC) AS ranking
            FROM
                    players p
            INNER JOIN
                    matches m
            ON
                    m.first_player = p.player_id
                    OR
                    m.second_player = p.player_id
            GROUP BY
                    p.group_id,
                    players
        ) temp_matches
WHERE
        ranking = 1;
```

Question 94:

```sql

-- Q.94 Write an SQL Query to report the students (student_id, student_name) being -- Quiet in all exams.
-- Do not return the student who has never taken any exam.


-- Approach 1


WITH temp_examination AS (
                                SELECT
                                        exam_id,
                                        student_id,
                                        score,
                                        max(score) OVER w AS highest,
                                        min(score) OVER w AS lowest
                                FROM
                                        exam
                                WINDOW
                                        w AS (PARTITION BY exam_id)
                        ),

     temp_examination1 AS (
                                SELECT
                                        DISTINCT student_id
                                FROM
                                        temp_examination
                                WHERE
                                        score IN (lowest, highest)
                                               )

    SELECT
            DISTINCT s.student_id,
            s.student_name
    FROM
            temp_examination
    INNER JOIN
            student s
    ON
            s.student_id = temp_examination.student_id
    WHERE
            s.student_id NOT IN (SELECT student_id FROM temp_examination1);


-- Approach 2


WITH temp_examination AS (
                                SELECT
                                        student_id,
                                        CASE
                                        WHEN score < max(score) OVER(PARTITION BY exam_id) AND score >min(score) OVER(PARTITION BY exam_id) THEN 0
                                        ELSE 1
                                        END AS category
                                FROM
                                        exam
                                ORDER BY
                                        student_id
            ),

    temp_examination1 AS (
                                SELECT
                                        student_id,
                                        SUM(category) AS high_low_count
                                FROM
                                        temp_examination
                                GROUP BY
                                        student_id
            )

SELECT
        s.student_id,
        s.student_name
FROM
        student s
INNER JOIN
        temp_examination1
ON
        s.student_id = temp_examination1.student_id
WHERE
        high_low_count = 0
ORDER BY
        temp_examination1.student_id;

```

Question 95:

```sql

-- Q.95 Write an SQL Query to report the students (student_id, student_name) being -- Quiet in all exams.
-- Do not return the student who has never taken any exam.


-- Approach 1


WITH temp_examination AS (
                                SELECT
                                        exam_id,
                                        student_id,
                                        score,
                                        max(score) OVER w AS highest,
                                        min(score) OVER w AS lowest
                                FROM
                                        exam
                                WINDOW
                                        w AS (PARTITION BY exam_id)
                        ),

     temp_examination1 AS (
                                SELECT
                                        DISTINCT student_id
                                FROM
                                        temp_examination
                                WHERE
                                        score IN (lowest, highest)
                                        )

    SELECT
            DISTINCT s.student_id,
            s.student_name
    FROM
            temp_examination
    INNER JOIN
            student s
    ON
            s.student_id = temp_examination.student_id
    WHERE
            s.student_id NOT IN (SELECT student_id FROM temp_examination1);


-- Approach 2


WITH temp_examination AS (
                                SELECT
                                        student_id,
                                        CASE
                                        WHEN score < max(score) OVER(PARTITION BY exam_id) AND score >min(score) OVER(PARTITION BY exam_id) THEN 0
                                        ELSE 1
                                        END AS category
                                FROM
                                        exam
                                ORDER BY
                                        student_id
            ),

    temp_examination1 AS (
                                SELECT
                                        student_id,
                                        SUM(category) AS high_low_count
                                FROM
                                        temp_examination
                                GROUP BY
                                        student_id
            )

SELECT
        s.student_id,
        s.student_name
FROM
        student s
INNER JOIN
        temp_examination1
ON
        s.student_id = temp_examination1.student_id
WHERE
        high_low_count = 0
ORDER BY
        temp_examination1.student_id;

```

Question 96:

```sql

-- Q.96 Write a query to output the user id, song id, and cumulative count of song plays as of 4 August 2022
-- sorted in descending order.

WITH streaming AS (
                        SELECT
                                user_id,
                                song_id,
                                song_plays
                        FROM
                                songs_history

                        UNION ALL

                        SELECT
                                user_id,
                                song_id,
                                count(*) AS song_plays
                        FROM
                                songs_weekly
                         WHERE
                                listen_time <= '08/04/2022 23:59:59'
                        GROUP by
                                user_id,
                                song_id
                )

SELECT
        user_id,
        song_id,
        SUM(song_plays) as song_plays
FROM
        streaming
GROUP BY
        user_id,
        song_id
ORDER BY
        song_plays DESC;

```

Question 97:

```sql

-- Q.97 Write a query to find the confirmation rate of users who confirmed their signups with text messages.
-- Round the result to 2 decimal places.


WITH temp_confirmation AS (
                                SELECT
                                        e.email_id,
                                        CASE
                                                WHEN signup_action = 'Confirmed' THEN 1
                                                END
                                                AS confirmed_users
                                FROM
                                        emails e
                                LEFT JOIN
                                        texts t
                                ON
                                        e.email_id = t.email_id
                                        AND
                                        t.signup_action = 'Confirmed'
                )


SELECT
        ROUND(SUM(confirmed_users)/COUNT(email_id),2) AS confirm_rate
FROM
        temp_confirmation;

```

Question 98:

```sql

-- Q.98 Calculate the 3-day rolling average of tweets published by each user for each date
-- that a tweet was posted. Output the user id, tweet date, and rolling averages rounded to 2 decimal places.


WITH temp_tweets AS (
                        SELECT
                                user_id,
                                tweet_date,
                                COUNT(tweet_id) AS tweets_count
                        FROM
                                tweets
                        GROUP BY
                                user_id,
                                tweet_date
                        ORDER BY
                                user_id,
                                tweet_date
            )

SELECT
        user_id,
        tweet_date,
        ROUND(avg(tweets_count)
        OVER(PARTITION BY user_id ORDER BY tweet_date
        ROWS BETWEEN 2 PRECEDING AND CURRENT ROW),2) AS rolling_avg_3days
FROM
        temp_tweets;

```

Question 99:

```sql

-- Q.99 Write a query to obtain a breakdown of the time spent sending vs. opening snaps
-- (as a percentage of total time spent on these activities) for each age group.


WITH temp_activities AS (
                                SELECT
                                        user_id,
                                        activity_type,
                                        sum(time_spent) time_spent,
                                        CASE
                                                WHEN activity_type = 'open' THEN sum(time_spent)
                                                ELSE 0
                                                END opening_snap,
                                        CASE
                                                WHEN activity_type = 'send' THEN sum(time_spent)
                                                ELSE 0
                                                END sending_snap
                                FROM
                                        activities
                                WHERE
                                        activity_type in ('open','send')
                                GROUP BY
                                        user_id,
                                        activity_type
                                ORDER BY
                                        user_id
                                        ),

    temp_activities2 AS (
                                SELECT
                                        user_id,
                                        SUM(opening_snap) time_sending,
                                        SUM(sending_snap) time_opening
                                FROM
                                        temp_activities
                                GROUP BY
                                        user_id
                        )

SELECT
        ab.age_bucket,
        ROUND(time_opening * 100.0 /(time_sending+time_opening), 2) AS send_perc,
        ROUND(time_sending * 100.0 /(time_sending+time_opening), 2) AS open_perc
FROM
        temp_activities2
INNER JOIN
        age_breakdown ab
ON
        ab.user_id = temp_activities2.user_id
ORDER BY
        ab.age_bucket;

```

Question 100:

```sql

-- Q.100 Write a query to return the IDs of these LinkedIn power creators in ascending order.


SELECT
        DISTINCT p.profile_id
FROM
        personal_profiles p
INNER JOIN
        employee_company ec
ON
        p.profile_id = ec.personal_profile_id
INNER JOIN
        company_pages c
ON
        ec.company_id = c.company_id
WHERE
        p.followers > c.followers
ORDER BY
        p.profile_id;

```

Set 3:

Question 101:

```sql

-- Q.101 Write an SQL query to show the second most recent activity of each user.
-- If the user only hAS one activity, return that one. A user cannot perform more than one activity at the same time.
-- Return the result table in any order.


WITH temp_activity AS (
                        SELECT
                                username,
                                activity,
                                start_date,
                                end_date,
                                ROW_NUMBER() OVER(PARTITION BY username ORDER BY start_date, end_date) row_num,
                                COUNT(*) OVER(PARTITION BY username ORDER BY start_date, end_date
                                                rows between unbounded preceding and unbounded following) total_activities
                        FROM
                                user_activity
            ),
    temp_activity2 AS (
                        SELECT
                                username,
                                activity,
                                start_date,
                                end_date,
                                IF(total_activities = 1 and row_num = 1, 2, row_num) AS ranking
                        FROM
                                temp_activity
            )

SELECT
        username,
        activity,
        start_date,
        end_date
FROM
        temp_activity2
WHERE
        ranking = 2;

```

Question 102:

```sql

-- Q.102 Write an SQL query to show the second most recent activity of each user.
-- If the user only hAS one activity, return that one. A user cannot perform more than one activity at the same time.
-- Return the result table in any order.


WITH temp_activity AS (
                        SELECT
                                username,
                                activity,
                                start_date,
                                end_date,
                                ROW_NUMBER() OVER(PARTITION BY username ORDER BY start_date, end_date) row_num,
                                COUNT(*) OVER(PARTITION BY username ORDER BY start_date, end_date
                                                rows between unbounded preceding and unbounded following) total_activities
                        FROM
                                user_activity
            ),
    temp_activity2 AS (
                        SELECT
                                username,
                                activity,
                                start_date,
                                end_date,
                                IF(total_activities = 1 and row_num = 1, 2, row_num) AS ranking
                        FROM
                                temp_activity
            )

SELECT
        username,
        activity,
        start_date,
        end_date
FROM
        temp_activity2
WHERE
        ranking = 2;

```

Question 103:

```sql

-- Q.103 Query the name of any student in students who scored higher than 75 Marks. Order your output
-- by the last three characters of each name. If two or more students both have names ending in the same
-- last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending id.


SELECT
        name
FROM
        students
WHERE
        marks > 75
ORDER BY
        RIGHT(name,3),
        id;

```

Question 104:

```sql

-- Q.104 Write a Query that prints a list of employee names (i.e.: the name attribute)
-- for employees in employee HAVING a salary greater than $2000 per month who have
-- been employees for less than 10 months. Sort your result by ascending employee_id.


SELECT
        name
FROM
        employee
WHERE
        salary > 2000
        AND
        months < 10
ORDER BY
        employee_id;

```

Question 105:

```sql

-- Q.105 Write a Query identifying the type of each record in the TRIANGLES table using its three side lengths.


SELECT
        a,
        b,
        c,
        CASE
                WHEN a + b <= c OR b + c <= a OR a + c <= b THEN 'NOT A TRIANGLE'
                WHEN a = b AND b = c THEN 'EQUILATERAL'
                WHEN a = b OR b = c OR c = a THEN 'ISSOCELES'
                WHEN a <> b AND b <> c THEN 'SCALEAN'
                END AS type_of_triangle
FROM
        triangles;

```

Question 106:

```sql

-- Q.106 Write a query calculating the amount of error (i.e.: actual - miscalculated average monthly salaries),
-- and round it up to the next integer.


SELECT
        ceil(avg(salary) - avg(replace(salary, '0', ''))) AS error
FROM
        employees;

```

Question 107:

```sql

-- Q.107 Write a query to find the maximum total earnings for all employees as
-- well as the total number of employees who have maximum total earnings.
-- Then print these values as 2 space-separated integers.


SELECT
        MAX(salary*months) as total_earnings,
        COUNT(*)
FROM
        employee
WHERE
        (salary*months) in (
                            SELECT
                                    MAX(months * salary)
                            FROM
                                    employee
                            );

```

Question 108:

```sql

-- Q.108 a. Query an alphabetically ordered list of all names in OCCUPATIONS, immediately followed
-- by the first letter of each profession AS a parenthetical (i.e.: enclosed in parentheses).


SELECT
        CONCAT(name, '(',substring(occupation, 1, 1),')') as `name(occupation)`
FROM
        occupations
ORDER BY
        name;

```

```sql

-- Q.108 b. WHERE [occupation_COUNT] is the number of occurrences of an occupation in OCCUPATIONS and [occupation]
-- is the lowerCASE occupation name. If more than one Occupation hAS the same [occupation_COUNT],
-- they should be ordered alphabetically.


SELECT
        CONCAT("There are a total of ",
        COUNT(*),' ', lower(occupation), 's.')  AS info
FROM
        occupations
GROUP BY
        occupation
ORDER BY
        COUNT(occupation),
        occupation;

```

Question 109:

```sql

-- Q.109 Pivot the Occupation column in OCCUPATIONS so that each Name is sorted alphabetically
-- and displayed underneath its corresponding Occupation. The output column headers should be
-- Doctor, Professor, Singer, and Actor, respectively.


SELECT
        MAX(CASE WHEN occupation = 'Doctor' then name END) AS Doctor,
        MAX(CASE WHEN occupation = 'Professor' then name END) AS Professor,
        MAX(CASE WHEN occupation = 'Singer' then name END) AS Singer,
        MAX(CASE WHEN occupation = 'Actor' then name END) AS Actor
FROM
        (
            SELECT
                    name,
                    occupation,
                    row_number() over(partition by occupation order by name) AS row_num
            FROM
                    occupations
        ) AS base
GROUP BY
        row_num;

```

Question 110:

```sql

-- Q.110 Write a query to find the node type of Binary Tree ordered by the value of the node.
-- Output one of the following for each node:
-- • Root: If node is root node.
-- • Leaf: If node is leaf node.
-- • Inner: If node is neither root nor leaf node.


SELECT
        n,
        CASE
            WHEN n NOT IN (SELECT DISTINCT p FROM bst WHERE p IS NOT NULL)  THEN 'Leaf'
            WHEN p IS NULL THEN 'Root'
            ELSE 'Inner'
            END AS type
FROM
        bst
ORDER BY
         n;

```

Question 111:

```sql

-- Q.111 Given the table schemas below, write a query to print the company_code,
-- founder name, total number of lead managers, total number of senior managers,
-- total number of managers, and total number of employees. Order your output by
-- ascending company_code.


SELECT
        c.company_code,
        c.founder,
        COUNT(DISTINCT lm.lead_manager_code),
        COUNT(DISTINCT sm.senior_manager_code),
        COUNT(DISTINCT m.manager_code),
        COUNT(DISTINCT e.employee_code)
FROM
        company c
INNER JOIN
        lead_manager lm
ON
        c.company_code = lm.company_code
INNER JOIN
        senior_manager sm
ON
        sm.lead_manager_code = lm.lead_manager_code
INNER JOIN
        manager m
ON
        m.senior_manager_code = sm.senior_manager_code
INNER JOIN
        employee e
ON
        e.manager_code = m.manager_code
GROUP BY
        c.company_code, c.founder
ORDER BY
        c.company_code;

```

Question 112:

```sql

-- Q.112 Write a query to print all prime numbers less than or equal to 1000.
-- Print your result on a single line, and use the ampersand () character as
-- your separator (instead of a space).


WITH RECURSIVE number_generation AS (
                                        SELECT
                                                1 num

                                        UNION ALL

                                        SELECT
                                                num + 1
                                        FROM
                                                number_generation
                                        WHERE
                                                num<1000
                    ),
                number_generation2 AS (
                                        SELECT
                                                n1.num AS numm
                                        FROM
                                                number_generation n1
                                        INNER JOIN
                                                number_generation n2
                                        WHERE
                                                n1.num % n2.num = 0
                                        GROUP BY
                                                n1.num
                                        HAVING
                                                COUNT(n1.num) = 2
                    )


SELECT
        group_concat(numm ORDER BY numm SEPARATOR '&') AS prime_numbers
FROM
        number_generation2;

```

Question 113:

```sql

-- Q.113 Write a query to print the pattern P(20).


WITH RECURSIVE generate_numbers AS
                                    (
                                        SELECT
                                                1 AS n

                                        UNION

                                        SELECT
                                                n+1
                                        FROM
                                                generate_numbers
                                        WHERE
                                                n<20
                                    )

SELECT
        repeat('*',n)
FROM
        generate_numbers;

```

Question 114:

```sql

-- Q.114 Write a query to print the pattern P(20).


WITH RECURSIVE generate_numbers AS
                                    (
                                        SELECT
                                                20 AS n

                                        UNION

                                        SELECT
                                                n-1
                                        FROM
                                                generate_numbers
                                        WHERE
                                                n>1
                                    )

SELECT
        repeat('*',n)
FROM
        generate_numbers;


```

Question 115:

```sql

-- Q.115 Query the name of any student in students who scored higher than 75 Marks. Order your output
-- by the last three characters of each name. If two or more students both have names ending in the same
-- last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending id.


SELECT
        name
FROM
        students
WHERE
        marks > 75
ORDER BY
        RIGHT(name,3),
        id;

```

Question 116:

```sql

-- Q.116 Write a Query that prints a list of employee names (i.e.: the name attribute)
-- from the employee table in alphabetical order.


SELECT
        name
FROM
        employee
ORDER BY
        name;

```

Question 117:

```sql

-- Q.117 Write a Query that prints a list of employee names (i.e.: the name attribute)
-- for employees in employee HAVING a salary greater than $2000 per month who have
-- been employees for less than 10 months. Sort your result by ascending employee_id.


SELECT
        name
FROM
        employee
WHERE
        salary > 2000
        AND
        months < 10
ORDER BY
        employee_id;

```

Question 118:

```sql

-- Q.118 Write a Query identifying the type of each record in the TRIANGLES table using its three side lengths.


SELECT
        a,
        b,
        c,
        CASE
                WHEN a + b <= c OR b + c <= a OR a + c <= b THEN 'NOT A TRIANGLE'
                WHEN a = b AND b = c THEN 'EQUILATERAL'
                WHEN a = b OR b = c OR c = a THEN 'ISSOCELES'
                WHEN a <> b AND b <> c THEN 'SCALEAN'
                END AS type_of_triangle
FROM
        triangles;

```

Question 119:

```sql

-- Q.119 Write a Query to obtain the year-on-year growth rate for the total spend of each product for each year.


-- Approach 1


WITH temp_transactions AS (
                                SELECT
                                        product_id,
                                        transaction_date,
                                        spend AS curr_year_spend,
                                        LAG(spend,1,0) OVER w AS prev_year_spend,
                                        IFNULL(spend - LAG(spend,1) OVER w, 0) AS prev_curr_spend_diff
                                FROM
                                        user_transactions
                                WINDOW
                                        w AS (PARTITION BY product_id ORDER BY EXTRACT(YEAR FROM transaction_date))
                        )

SELECT
        product_id,
        curr_year_spend,
        ROUND(prev_year_spend, 2),
        IFNULL(ROUND((prev_curr_spend_diff * 100)/prev_year_spend,2),0) AS YOY
FROM
        temp_transactions;



-- Approach 2


WITH temp_transactions AS (
                                SELECT
                                        YEAR(STR_TO_date(transaction_date, '%m/%d/%Y')) AS YEAR_id,
                                        product_id,
                                        transaction_date,
                                        spend AS curr_year_spend,
                                        LAG(spend,1) OVER w AS prev_year_spend,
                                        spend - LAG(spend,1) OVER w AS prev_curr_spend_diff
                                FROM
                                        user_transactions
                                WINDOW
                                        w AS (PARTITION BY product_id ORDER BY EXTRACT(YEAR FROM transaction_date))
                        )

SELECT
        year_id,
        product_id,
        curr_year_spend,
        ROUND(prev_year_spend, 2),
        ROUND((prev_curr_spend_diff * 100)/prev_year_spend,2) AS YOY
FROM
        temp_transactions;


-- Approach 3


SELECT
        product_id,
        YEAR(STR_TO_date(transaction_date, '%m/%d/%Y')) AS YEAR_id,
        spend AS curr_year_spend,
        ROUND(LAG(spend,1,0) OVER w ,2) AS prev_year_spend,
        ROUND((spend - LAG(spend,1) OVER w ) * 100 /
                    LAG(spend,1) OVER w, 2)  AS YOY
FROM
        user_transactions
WINDOW
        w AS (PARTITION BY product_id ORDER BY EXTRACT(YEAR FROM transaction_date));

```

Question 120:

```sql

-- Q.120 Write a SQL Query to find the number of prime and non-prime items that can be stored
-- in the 500,000 square feet warehouse. Output the item type and number of items to be stocked.


-- Approach 1


WITH temp_inventory AS (
                        SELECT
                                item_type,
                                SUM(square_foot) AS square_foot_per_category,
                                COUNT(*) AS count_of_items
                        FROM
                                inventory
                        GROUP BY
                                item_type
                ),

    temp_inventory2 AS (
                        SELECT
                                (500000 - SUM(square_foot_per_category)*FLOOR(500000/SUM(square_foot_per_category))) AS area_left
                        FROM
                                temp_inventory
                        WHERE
                                item_type = 'PRIME_ELIGIBLE'
                ),

    temp_inventory3 AS (
                        SELECT
                                item_type,
                                CASE
                                        WHEN item_type = 'PRIME_ELIGIBLE'
                                                THEN FLOOR(500000/square_foot_per_category) * count_of_items
                                        WHEN item_type = 'NOT_PRIME'
                                                THEN FLOOR((SELECT area_left FROM temp_inventory2) / square_foot_per_category) * count_of_items
                                        END AS item_count
                        FROM
                                temp_inventory
                )

SELECT
        item_type,
        item_count
 FROM
        temp_inventory3;


```

```sql

-- Approach 2


WITH temp_inventory AS (
                        SELECT
                                item_type,
                                SUM(square_foot) AS square_foot_per_category,
                                COUNT(*) AS count_of_items,
                                CASE
                                        WHEN item_type = 'PRIME_ELIGIBLE' THEN FLOOR(500000/SUM(square_foot)) * COUNT(*)
                                        END AS prime_items_count
                        FROM
                                inventory
                        GROUP BY
                                item_type
                ),

    temp_inventory2 AS (
                        SELECT
                                (500000 - SUM(square_foot_per_category)*FLOOR(500000/SUM(square_foot_per_category))) AS area_left
                        FROM
                                temp_inventory
                        WHERE
                                item_type = 'PRIME_ELIGIBLE'
                ),

    temp_inventory3 AS (
                        SELECT
                                item_type,
                                CASE
                                        WHEN item_type = 'PRIME_ELIGIBLE'
                                                THEN prime_items_count
                                        WHEN item_type = 'NOT_PRIME'
                                                THEN FLOOR((SELECT area_left FROM temp_inventory2) / square_foot_per_category) * count_of_items
                                END AS item_count
                        FROM
                                temp_inventory
                )

SELECT
        item_type,
        item_count
FROM
        temp_inventory3;

```

Question 121:

```sql

-- Q.121 Write a Query to obtain the active user retention in July 2022.
-- Output the month (in numerical format 1, 2, 3) and the number of monthly active users (MAUs).


-- Approach 1


WITH temp_actions AS (
                        SELECT
                                user_id,
                                event_date,
                                event_type,
                                SUBSTR(event_date, 6, 2) - lag(SUBSTR(event_date, 6, 2)) OVER w AS difference
                        FROM
                                user_actions
                        WINDOW
                                w as (PARTITION BY user_id ORDER BY event_date)
                ),

        temp_actions2 AS (
                        SELECT
                                SUBSTR(event_date, 6, 2) AS months,
                                COUNT(user_id) AS monthly_active_users
                        FROM
                                temp_actions
                        WHERE
                                difference = 1 AND event_type IN ('LIKE', 'COMMENT', 'SIGN-IN')
                        GROUP BY
                                months
                                )

SELECT
        months,
        monthly_active_users
FROM
        temp_actions2;


-- Approach 2


 WITH temp_actions AS (
                        SELECT
                                user_id,
                                event_date,
                                event_type,
                                SUBSTR(event_date, 6, 2) - lag(SUBSTR(event_date, 6, 2)) OVER w AS difference
                        FROM
                                user_actions
                        WINDOW
                                w as (PARTITION BY user_id ORDER BY event_date)
                )


SELECT
        SUBSTR(event_date, 6, 2) AS months,
        COUNT(DISTINCT user_id) AS active_users
FROM
        temp_actions
WHERE
        difference = 1
        AND
        event_type IN ('LIKE', 'COMMENT', 'SIGN-IN')
GROUP BY
        months;

```

Question 122:

```sql

-- Q.122 Write a Query to report the median of searches made by a user.
-- Round the median to one decimal point.


 WITH temp_search_freq AS (
                                SELECT
                                        searches,
                                        num_users,
                                        ROW_NUMBER() OVER(ORDER BY searches) row_num,
                                        COUNT(*) OVER(ORDER BY searches ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) total_records
                                FROM
                                        search_frequency
                        ),

    temp_search_freq2 as (
                                SELECT
                                        searches,
                                        num_users,
                                        CASE
                                        WHEN total_records % 2 <>  0 THEN (
                                                                                SELECT
                                                                                        DISTINCT ROUND(SUM(searches) OVER w /
                                                                                        COUNT(*) OVER w,1)
                                                                                FROM
                                                                                        temp_search_freq
                                                                                WHERE
                                                                                        row_num = ROUND((total_records + 1) / 2, 0)
                                                                                WINDOW
                                                                                        w AS (ORDER BY searches ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)

                                                                                )

                                        WHEN total_records % 2 = 0 THEN (
                                                                                SELECT
                                                                                        DISTINCT ROUND(SUM(searches) OVER w /
                                                                                        COUNT(*) OVER w,1)
                                                                                FROM
                                                                                        temp_search_freq
                                                                                WHERE
                                                                                        row_num IN (total_records/2,(total_records/2)+1)
                                                                                WINDOW
                                                                                        w AS (ORDER BY searches ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
                                                                                )
                                        END AS median
                                FROM
                                        temp_search_freq
           )


SELECT
        DISTINCT median
FROM
        temp_search_freq2;

```

Question 123:

```sql

-- Q.123 Write a Query to update the Facebook advertisers status using the daily_pay table.
-- Advertiser is a two-column table containing the user id and their payment status based
-- on the last payment and daily_pay table has current information about their payment.
-- Only advertisers who paid will show up in this table.
-- Output the user id and current payment status sorted by the user id.


SELECT
        user_id,
        CASE
            WHEN user_id IN (SELECT user_id FROM daily_pay) THEN 'EXISTING'
            ELSE 'CHURN'
            END AS new_status
FROM
        advertiser
ORDER BY
        user_id;

```

Question 124:

```sql

-- Q.124 Write a SQL Query that calculates the total time that the fleet of
-- servers was running. The output should be in units of full days.


-- Approach 1


SELECT
                stop_time - start_time AS total_up_time
FROM
                (
                        SELECT
                                SUM(
                                        CASE
                                                WHEN session_status = 'start' then EXTRACT(DAY from STR_TO_DATE(status_time, '%m/%d/%y'))
                                                END
                                        )       AS start_time,
                                SUM(
                                        CASE
                                                WHEN session_status = 'stop' then EXTRACT(DAY from STR_TO_DATE(status_time, '%m/%d/%y'))
                                                END
                                        )       AS stop_time
                        FROM
                                        server_utilization
        ) temp_server_utilization;


-- Approach 2 (Works only in PostgreSQL)


WITH temp_server_utilization AS (
                                        SELECT
                                                server_id,
                                                status_time AS start_time,
                                                session_status,
                                                lead(status_time) OVER(ORDER BY server_id,status_time) AS end_time
                                        FROM
                                                server_utilization
                                )

SELECT
        EXTRACT(DAY FROM justify_hours(SUM(end_time - start_time))) as total_time
FROM
        temp_server_utilization
WHERE
        session_status = 'start';

```

Question 125:

```sql

-- Q.125 Sometimes, payment transactions are repeated by accident; it could be due to user error,
-- API failure or a retry error that causes a credit card to be charged twice.
-- Using the transactions table, identify any payments made at the same merchant with the
-- same credit card for the same amount within 10 minutes of each other. Count such repeated payments.


WITH temp_transactions AS (
                                SELECT
                                        merchant_id,
                                        credit_card_id,
                                        amount,
                                        transaction_timestamp,
                                        LAG(transaction_timestamp) OVER w AS prev_tran_timestamp,
                                        timestampdiff(MINUTE,LAG(transaction_timestamp) OVER w, transaction_timestamp) AS difference
                                FROM
                                        transactions
                                WINDOW
                                        w as (PARTITION BY credit_card_id ORDER BY  transaction_timestamp)
            )

SELECT
        COUNT(DISTINCT merchant_id) AS payment_count
FROM
        temp_transactions
WHERE
        difference <= 10;


-- Approach 2 (Works only in PostgreSQL)


WITH temp_transactions AS (
                                SELECT
                                        merchant_id,
                                        credit_card_id,
                                        amount,
                                        transaction_timestamp,
                                        LAG(transaction_timestamp) OVER w AS prev_tran_timestamp,
                                        EXTRACT(EPOCH FROM LAG(transaction_timestamp) OVER w -  transaction_timestamp) AS difference
                                FROM
                                        transactions
                                WINDOW
                                        w as (PARTITION BY credit_card_id ORDER BY  transaction_timestamp)
                        )

 SELECT
        COUNT(DISTINCT merchant_id) AS payment_count
FROM
        temp_transactions
 WHERE
        difference < 10;

```

Question 126:

```sql

-- Q.126 Write a SQL Query to find the bad experience rate in the first 14 days for new users who signed
-- up in June 2022. Output the percentage of bad experience rounded to 2 decimal places.


SELECT
        ROUND(
            SUM(
                CASE
                    WHEN status !='completed successfully'  THEN 1 ELSE 0
                    END
                )*100.0/count(*),2)  AS bad_experience_pct
FROM
        customers C
INNER JOIN
        orders O
ON
        o.customer_id = c.customer_id
WHERE
        o.order_timestamp < date_add(STR_TO_date(signup_timestamp, '%m/%d/%Y'), INTERVAL 14 DAY)
        AND
        MONTH(STR_TO_date(signup_timestamp, '%m/%d/%Y')) = 06
        AND
        YEAR(STR_TO_date(signup_timestamp, '%m/%d/%Y')) = 2022;

```

Question 127:

```sql

-- Q.127 Write an SQL Query to find the total score for each gender on each day.
-- Return the result table ordered by gender and day in ascending order.


SELECT
        gender,
        day,
        SUM(score_points) OVER(PARTITION BY gender ORDER BY day
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS total_points
FROM
        scores
order by
        gender,
        day;

```

Question 128:

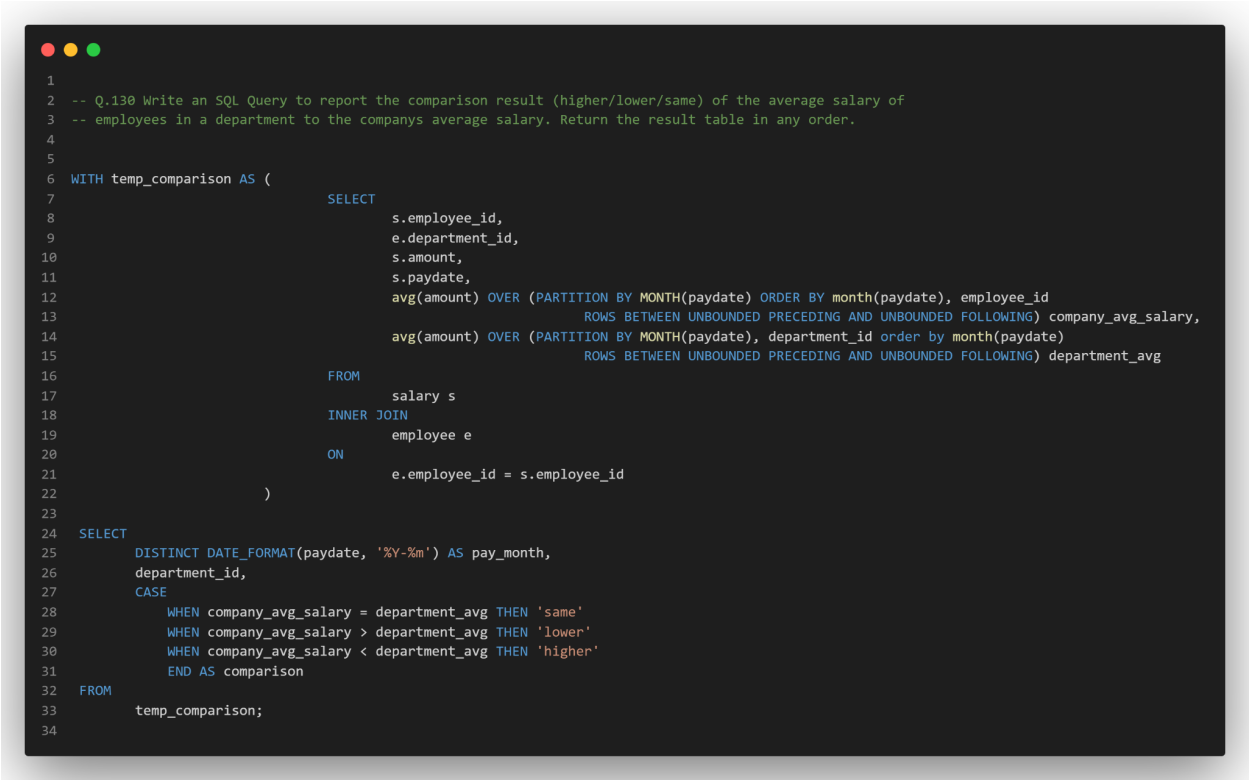
```sql

-- Q.128 Write an SQL Query to find the countries where this company can invest .
-- Return the result table in any order .


-- Approach 1


SELECT
        name AS country
FROM    (
                SELECT
                        c.name,
                        SUM(ca.duration) AS call_duration,
                        COUNT(c.country_code) AS number_of_calls
                FROM (
                        SELECT
                                id,
                                name,
                                CASE
                                WHEN LEFT(SUBSTR(phone_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone_number, 1,3), (LENGTH(SUBSTR(phone_number, 1,3))-1))
                                ELSE SUBSTR(phone_number, 1,3) END AS country_code
                        FROM
                                person
                ) temp_person
JOIN
        country c
ON
        temp_person.country_code = c.country_code
JOIN
        calls ca
ON
        temp_person.id = caller_id
GROUP BY
        c.name


UNION ALL


SELECT
        c.name,
        SUM(ca.duration) AS call_duration,
        COUNT(c.country_code) AS number_of_calls
FROM    (
                SELECT
                        id,
                        name,
                        CASE
                        WHEN LEFT(SUBSTR(phone_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone_number, 1,3), (length(SUBSTR(phone_number, 1,3))-1))
                        ELSE SUBSTR(phone_number, 1,3) END AS country_code
                FROM
                        person
        )       temp_person
JOIN
        country c
ON
        temp_person.country_code = c.country_code
JOIN
        calls ca
ON
        temp_person.id = ca.callee_id
GROUP BY
        c.name

        ) temp

GROUP BY
        name
HAVING
        SUM(call_duration)/SUM(number_of_calls) >  (SELECT AVG(duration) FROM calls);

```

```sql

-- Approach 2


 WITH temp_person AS (
                        SELECT
                                id,
                                name,
                                CASE
                                WHEN LEFT(SUBSTR(phone_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone_number, 1,3), (LENGTH(SUBSTR(phone_number, 1,3))-1))
                                ELSE SUBSTR(phone_number, 1,3) END AS country_code
                        FROM
                                person
                                 )


SELECT
        name AS country
FROM    (
        SELECT
                c.name,
                SUM(ca.duration) AS call_duration,
                COUNT(c.country_code) AS number_of_calls
        FROM
                temp_person
        JOIN
                country c
        ON
                temp_person.country_code = c.country_code
        JOIN
                calls ca
        ON
                temp_person.id = ca.caller_id
        GROUP BY
                c.name

UNION ALL

        SELECT
                c.name,
                SUM(ca.duration) AS call_duration,
                COUNT(c.country_code) AS number_of_calls
        FROM
                temp_person
        JOIN
                country c
        ON
                temp_person.country_code = c.country_code
        JOIN
                calls ca
        ON
                temp_person.id = ca.callee_id
        GROUP BY
                c.name

        )temp
GROUP BY
        name
HAVING
        SUM(call_duration)/SUM(number_of_calls) >  (SELECT AVG(duration) FROM calls);

```

Question 129:

```sql

-- Q.129 Write an SQL Query to report the median of all the numbers in the database
-- after decompressing the numbers table. Round the median to one decimal point.


WITH RECURSIVE num_frequency (num,frequency, i) AS
                        (
                                SELECT
                                        num,
                                        frequency,1
                                FROM
                                        numbers

                                UNION ALL

                                SELECT
                                        num,
                                        frequency,
                                        i+1
                                FROM
                                        num_frequency
                                WHERE
                                        num_frequency.i < num_frequency.frequency
                        ),

        num_frequency2 AS (
                                SELECT
                                        num,
                                        frequency,
                                        row_number() OVER(ORDER BY num, frequency) AS row_num,
                                        COUNT(*) OVER(ORDER BY num, frequency ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS total_records
                                FROM
                                        num_frequency
                        )

SELECT
        DISTINCT CASE
                        WHEN total_records % 2 <>  0 THEN (
                                SELECT
                                        DISTINCT ROUND(SUM(num) OVER w /
                                        COUNT(*) OVER  w, 1)
                                FROM
                                        num_frequency2
                                WHERE
                                        row_num = ROUND((total_records + 1) / 2, 0)
                                WINDOW
                                        w as (ORDER BY  num, frequency ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING))

                    WHEN total_records % 2 = 0 THEN (
                                SELECT
                                        DISTINCT ROUND(SUM(num) OVER w /
                                        COUNT(*) OVER w, 1)
                                FROM
                                        num_frequency2
                                WHERE
                                        row_num IN (total_records/2,(total_records/2)+1)
                                WINDOW
                                        w as (ORDER BY  num, frequency ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING))
                    END AS median
FROM
        num_frequency2;

```

Question 130:



```sql

-- Q.130 Write an SQL Query to report the comparison result (higher/lower/same) of the average salary of
-- employees in a department to the companys average salary. Return the result table in any order.


WITH temp_comparison AS (
                                SELECT
                                        s.employee_id,
                                        e.department_id,
                                        s.amount,
                                        s.paydate,
                                        avg(amount) OVER (PARTITION BY MONTH(paydate) ORDER BY month(paydate), employee_id
                                                          ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) company_avg_salary,
                                        avg(amount) OVER (PARTITION BY MONTH(paydate), department_id order by month(paydate)
                                                          ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) department_avg
                                FROM
                                        salary s
                                INNER JOIN
                                        employee e
                                ON
                                        e.employee_id = s.employee_id
                        )

 SELECT
        DISTINCT DATE_FORMAT(paydate, '%Y-%m') AS pay_month,
        department_id,
        CASE
            WHEN company_avg_salary = department_avg THEN 'same'
            WHEN company_avg_salary > department_avg THEN 'lower'
            WHEN company_avg_salary < department_avg THEN 'higher'
            END AS comparison
 FROM
        temp_comparison;

```

Question 131:

```sql

-- Q.131 Write an SQL Query to report for each install date, the number of players
-- that installed the game on that day, and the day one retention.


-- Approach 1


SELECT
        a.event_date AS install_date,
        COUNT(a.player_id) AS installs,
        ROUND(COUNT(b.player_id) / COUNT(a.player_id), 2) AS day1_retention
FROM
        (
                SELECT
                        player_id,
                        MIN(event_date) AS event_date
                FROM
                        activity
                GROUP BY
                        player_id
        ) a
LEFT JOIN
        activity b
ON
        a.player_id = b.player_id
        AND
        a.event_date + 1 = b.event_date
GROUP BY
        a.event_date;


-- Approach 2


SELECT
        a1.event_date AS install_dt,
        COUNT(a1.player_id) AS installs,
        ROUND(COUNT(a3.player_id) / COUNT(a1.player_id), 2) AS day1_retention
FROM
        activity a1
LEFT JOIN
        activity a2
ON
        a1.player_id = a2.player_id
        AND
        a1.event_date > a2.event_date
LEFT JOIN
        activity a3
ON
        a1.player_id = a3.player_id
        AND
        DATEDIFF(a3.event_date, a1.event_date) = 1
WHERE
        a2.event_date IS NULL
GROUP BY
        a1.event_date;

```

Question 132:

```sql

-- Q.132 Write an SQL query to find the winner in each group. Return the result table in any order.


SELECT
        group_id,
        players as player_id
FROM    (
            SELECT
                    p.group_id,
                    CASE
                        WHEN first_player_goals > second_player_goals THEN first_player
                        WHEN first_player_goals < second_player_goals THEN second_player
                        WHEN first_player_goals = second_player_goals THEN IF(first_player < second_player, first_player, second_player)
                        END AS players,
                        MAX(IF(first_player_goals > second_player_goals, first_player_goals, second_player_goals)) AS goals,
                        ROW_NUMBER() OVER(PARTITION BY team_id ORDER BY MAX(IF(first_player_goals > second_player_goals, first_player_goals, second_player_goals)) DESC) AS ranking
            FROM
                    players p
            INNER JOIN
                    matches m
            ON
                    m.first_player = p.player_id
                    OR
                    m.second_player = p.player_id
            GROUP BY
                    p.group_id,
                    players
        ) temp_matches
WHERE
        ranking = 1;

```

Question 133:

```sql

-- Q.133 Write an SQL Query to report the students (student_id, student_name) being -- Quiet in all exams.
-- Do not return the student who has never taken any exam.


-- Approach 1


WITH temp_examination AS (
                                SELECT
                                        exam_id,
                                        student_id,
                                        score,
                                        max(score) OVER w AS highest,
                                        min(score) OVER w AS lowest
                                FROM
                                        exam
                                WINDOW
                                        w AS (PARTITION BY exam_id)
                        ),

     temp_examination1 AS (
                                SELECT
                                        DISTINCT student_id
                                FROM
                                        temp_examination
                                WHERE
                                        score IN (lowest, highest)
                                               )

    SELECT
            DISTINCT s.student_id,
            s.student_name
    FROM
            temp_examination
    INNER JOIN
            student s
    ON
            s.student_id = temp_examination.student_id
    WHERE
            s.student_id NOT IN (SELECT student_id FROM temp_examination1);


-- Approach 2


WITH temp_examination AS (
                                SELECT
                                        student_id,
                                        CASE
                                        WHEN score < max(score) OVER(PARTITION BY exam_id) AND score >min(score) OVER(PARTITION BY exam_id) THEN 0
                                        ELSE 1
                                        END AS category
                                FROM
                                        exam
                                ORDER BY
                                        student_id
            ),

    temp_examination1 AS (
                                SELECT
                                        student_id,
                                        SUM(category) AS high_low_count
                                FROM
                                        temp_examination
                                GROUP BY
                                        student_id
            )

SELECT
        s.student_id,
        s.student_name
FROM
        student s
INNER JOIN
        temp_examination1
ON
        s.student_id = temp_examination1.student_id
WHERE
        high_low_count = 0
ORDER BY
        temp_examination1.student_id;

```

Question 134:

```sql

-- Q.134 Write an SQL Query to report the students (student_id, student_name) being -- Quiet in all exams.
-- Do not return the student who has never taken any exam.


-- Approach 1


WITH temp_examination AS (
                                SELECT
                                        exam_id,
                                        student_id,
                                        score,
                                        max(score) OVER w AS highest,
                                        min(score) OVER w AS lowest
                                FROM
                                        exam
                                WINDOW
                                        w AS (PARTITION BY exam_id)
                        ),

     temp_examination1 AS (
                                SELECT
                                        DISTINCT student_id
                                FROM
                                        temp_examination
                                WHERE
                                        score IN (lowest, highest)
                                               )

    SELECT
            DISTINCT s.student_id,
            s.student_name
    FROM
            temp_examination
    INNER JOIN
            student s
    ON
            s.student_id = temp_examination.student_id
    WHERE
            s.student_id NOT IN (SELECT student_id FROM temp_examination1);


-- Approach 2


WITH temp_examination AS (
                                SELECT
                                        student_id,
                                        CASE
                                        WHEN score < max(score) OVER(PARTITION BY exam_id) AND score >min(score) OVER(PARTITION BY exam_id) THEN 0
                                        ELSE 1
                                        END AS category
                                FROM
                                        exam
                                ORDER BY
                                        student_id
            ),

    temp_examination1 AS (
                                SELECT
                                        student_id,
                                        SUM(category) AS high_low_count
                                FROM
                                        temp_examination
                                GROUP BY
                                        student_id
            )

SELECT
        s.student_id,
        s.student_name
FROM
        student s
INNER JOIN
        temp_examination1
ON
        s.student_id = temp_examination1.student_id
WHERE
        high_low_count = 0
ORDER BY
        temp_examination1.student_id;

```

Question 135:

```sql

-- Q.135 Write an SQL query to show the second most recent activity of each user.
-- If the user only hAS one activity, return that one. A user cannot perform more than one activity at the same time.
-- Return the result table in any order.


WITH temp_activity AS (
                        SELECT
                                username,
                                activity,
                                start_date,
                                end_date,
                                ROW_NUMBER() OVER(PARTITION BY username ORDER BY start_date, end_date) row_num,
                                COUNT(*) OVER(PARTITION BY username ORDER BY start_date, end_date
                                                rows between unbounded preceding and unbounded following) total_activities
                        FROM
                                user_activity
            ),
    temp_activity2 AS (
                        SELECT
                                username,
                                activity,
                                start_date,
                                end_date,
                                IF(total_activities = 1 and row_num = 1, 2, row_num) AS ranking
                        FROM
                                temp_activity
            )

SELECT
        username,
        activity,
        start_date,
        end_date
FROM
        temp_activity2
WHERE
        ranking = 2;

```

Question 136:

```sql

-- Q.136 Write an SQL query to show the second most recent activity of each user.
-- If the user only hAS one activity, return that one. A user cannot perform more than one activity at the same time.
-- Return the result table in any order.


WITH temp_activity AS (
                        SELECT
                                username,
                                activity,
                                start_date,
                                end_date,
                                ROW_NUMBER() OVER(PARTITION BY username ORDER BY start_date, end_date) row_num,
                                COUNT(*) OVER(PARTITION BY username ORDER BY start_date, end_date
                                                rows between unbounded preceding and unbounded following) total_activities
                        FROM
                                user_activity
            ),
    temp_activity2 AS (
                        SELECT
                                username,
                                activity,
                                start_date,
                                end_date,
                                IF(total_activities = 1 and row_num = 1, 2, row_num) AS ranking
                        FROM
                                temp_activity
            )

SELECT
        username,
        activity,
        start_date,
        end_date
FROM
        temp_activity2
WHERE
        ranking = 2;

```

Question 137:

```sql

-- Q.137 Write a query calculating the amount of error (i.e.: actual - miscalculated average monthly salaries),
-- and round it up to the next integer.


SELECT
        ceil(avg(salary) - avg(replace(salary, '0', ''))) AS error
FROM
        employees;

```

Question 138:

```sql

-- Q.138 Write a Query identifying the type of each record in the TRIANGLES table using its three side lengths.


SELECT
        a,
        b,
        c,
        CASE
                WHEN a + b <= c OR b + c <= a OR a + c <= b THEN 'NOT A TRIANGLE'
                WHEN a = b AND b = c THEN 'EQUILATERAL'
                WHEN a = b OR b = c OR c = a THEN 'ISSOCELES'
                WHEN a <> b AND b <> c THEN 'SCALEAN'
                END AS type_of_triangle
FROM
        triangles;

```

Question 139:

```sql

-- Q.139 Write a Query identifying the type of each record in the TRIANGLES table using its three side lengths.


SELECT
        a,
        b,
        c,
        CASE
                WHEN a + b <= c OR b + c <= a OR a + c <= b THEN 'NOT A TRIANGLE'
                WHEN a = b AND b = c THEN 'EQUILATERAL'
                WHEN a = b OR b = c OR c = a THEN 'ISSOCELES'
                WHEN a <> b AND b <> c THEN 'SCALEAN'
                END AS type_of_triangle
FROM
        triangles;

```

Question 140:

```sql

-- Q.140 Write a Query identifying the type of each record in the TRIANGLES table using its three side lengths.


SELECT
        a,
        b,
        c,
        CASE
                WHEN a + b <= c OR b + c <= a OR a + c <= b THEN 'NOT A TRIANGLE'
                WHEN a = b AND b = c THEN 'EQUILATERAL'
                WHEN a = b OR b = c OR c = a THEN 'ISSOCELES'
                WHEN a <> b AND b <> c THEN 'SCALEAN'
                END AS type_of_triangle
FROM
        triangles;

```

Question 141:

```sql

-- Q.141 Write a query to find the node type of Binary Tree ordered by the value of the node.
-- Output one of the following for each node:
-- • Root: If node is root node.
-- • Leaf: If node is leaf node.
-- • Inner: If node is neither root nor leaf node.


SELECT
         n,
         CASE
             WHEN n NOT IN (SELECT DISTINCT p FROM bst WHERE p IS NOT NULL)  THEN 'Leaf'
             WHEN p IS NULL THEN 'Root'
             ELSE 'Inner'
             END AS type
FROM
         bst
ORDER BY
          n;

```

Question 142:

```sql

-- Q.142 Given the table schemas below, write a query to print the company_code,
-- founder name, total number of lead managers, total number of senior managers,
-- total number of managers, and total number of employees. Order your output by
-- ascending company_code.


SELECT
        c.company_code,
        c.founder,
        COUNT(DISTINCT lm.lead_manager_code),
        COUNT(DISTINCT sm.senior_manager_code),
        COUNT(DISTINCT m.manager_code),
        COUNT(DISTINCT e.employee_code)
FROM
        company c
INNER JOIN
        lead_manager lm
ON
        c.company_code = lm.company_code
INNER JOIN
        senior_manager sm
ON
        sm.lead_manager_code = lm.lead_manager_code
INNER JOIN
        manager m
ON
        m.senior_manager_code = sm.senior_manager_code
INNER JOIN
        employee e
ON
        e.manager_code = m.manager_code
GROUP BY
        c.company_code, c.founder
ORDER BY
        c.company_code;

```

Question 143:

```sql

-- Q.143 Write a query to output all such symmetric pairs in ascENDing order by the value of X.
-- List the rows such that X1 ≤ Y1.


WITH temp_functions AS (
                        SELECT
                                x,
                                y,
                                ROW_NUMBER() OVER (ORDER BY x, y) AS row_num
                        FROM
                                functions
                )

SELECT
        DISTINCT f1.x,
        f1.y
FROM
        temp_functions f1
INNER JOIN
        temp_functions f2
ON
        f1.x = f2.y
AND
        f1.y = f2.x
AND
        f1.row_num <> f2.row_num
WHERE
        f1.x <= f1.y
ORDER BY
        f1.x;

```

Question 144:

```sql

-- Q.144 Write a query to output the names of those students whose best friENDs got offered a higher
-- salary than them. Names must be ordered by the salary amount offered to the best friENDs.
-- It is guaranteed that no two students get the same salary offer.


SELECT
        s1.name
FROM
        friends f1
INNER JOIN
        students s1
ON
        f1.id = s1.id
INNER JOIN
        students s2
ON
        f1.friend_id = s2.id
INNER JOIN
        packages p1
ON
        f1.id = p1.id
INNER JOIN
        packages p2
ON
        f1.friend_id = p2.id
WHERE
        p1.salary < p2.salary
ORDER BY
        p2.salary;


```

Question 145:

```sql

-- Q.145 Write a query to print the respective hacker_id and name of hackers who achieved full scores for
-- more than one challenge. Order your output in descENDing order by the total number of challenges in
-- which the hacker earned a full score. If more than one hacker received full scores in the same number
-- of challenges, then sort them by ascending hacker_id.


SELECT
        h.hacker_id,
        h.name
FROM
        hackers h
INNER JOIN
        submissions s
ON
        h.hacker_id = s.hacker_id
INNER JOIN
        challenges c
ON
        s.challenge_id = c.challenge_id
INNER JOIN
        difficulty d
ON
        c.difficulty_level = d.difficulty_level
WHERE
        s.score = d.score
GROUP BY
        h.name, h.hacker_id
HAVING
        COUNT(s.score) > 1
ORDER BY
        COUNT(s.challenge_id) desc,
        h.hacker_id;

```

Question 146:

```sql

-- Q.146 Write a query to output the start and END dates of projects listed by the number of days it took
-- to complete the project in ascending order. If there is more than one project that have the same number
-- of completion days, then order by the start date of the project.


-- Approach 1


WITH project_start AS
                    (
                        SELECT
                                start_date,
                                ROW_NUMBER() OVER() AS ps_rownum
                        FROM
                                projects
                        WHERE
                                start_date not in (
                                                    SELECT
                                                            end_date
                                                    FROM
                                                            projects
                                                )
                    ),
    project_end AS
                (
                        SELECT
                                end_date,
                                ROW_NUMBER() OVER() AS pe_rownum
                        FROM
                                projects
                        WHERE
                                END_date not in (
                                                    SELECT
                                                            start_date
                                                    FROM
                                                            projects
                                                )
                )

SELECT
        project_start.start_date,
        project_end.end_date
FROM
        project_start
INNER JOIN
        project_end
on
        project_end.pe_rownum = project_start.ps_rownum
ORDER BY
        DATEDIFF(project_start.start_date, project_end.end_date) desc,
        project_start.start_date;

```

```sql

-- Approach 2


WITH temp_project AS (
                    SELECT
                            temp.start_date,
                            temp.end_date,
                            SUM(
                                CASE
                                    WHEN previous_end_date IS NULL THEN 0
                                    WHEN DAY(end_date) - DAY(previous_end_date) = 1 THEN 0
                                    ELSE 1
                                END
                                ) OVER(ORDER BY start_date) AS project_num
                    FROM (
                            SELECT
                                    start_date,
                                    end_date,
                                    LAG(end_date) OVER (ORDER BY start_date) AS previous_end_date
                            FROM
                                    projects
                    ) temp
)

SELECT
        MIN(start_date) AS project_start_date,
        MAX(end_date) as project_end_date
FROM
        temp_project
GROUP BY
        project_num
ORDER BY
        DAY(MAX(end_date))-DAY(MIN(start_date));

```

```sql

-- Approach 3


WITH temp_project AS (
                        SELECT
                                temp.start_date,
                                temp.end_date,
                                SUM(
                                    CASE
                                    WHEN previous_end_date IS NULL THEN 0
                                    WHEN DAY(end_date) - DAY(previous_end_date) = 1 THEN 0
                                    ELSE 1
                                    END
                                    )  over(order by start_date range between unbounded preceding and current row) AS project_num
                        FROM (
                                SELECT
                                        start_date,
                                        end_date,
                                        LAG(end_date) OVER (ORDER BY start_date) AS previous_end_date
                                FROM
                                        projects
                ) temp
            )

SELECT
        MIN(start_date) AS project_start_date,
        MAX(end_date) AS project_end_date
FROM
        temp_project
GROUP BY
        project_num
ORDER BY
        DAY(MAX(end_date))-DAY(MIN(start_date));

```

Question 147:

```sql

-- Q.147 In an effort to identify high-value customers, Amazon asked for your help to obtain data
-- about users who go on shopping sprees. A shopping spree occurs when a user makes purchASes on 3
-- or more consecutive days. List the user IDs who have gone on at leASt 1 shopping spree in ascending order.


SELECT
        DISTINCT user_id
FROM
    (
        SELECT
                user_id,
                transaction_date,
                rn,
                transaction_date :: date - rn::integer,
                COUNT(transaction_date :: date - rn::integer) OVER(PARTITION BY user_id) AS cn
        FROM
                (
                    SELECT
                            user_id,
                            transaction_date,
                            ROW_NUMBER() OVER(PARTITION BY user_id ORDER BY transaction_date) AS rn
                    FROM
                            transactions
                ) temp_transactions
    )temp_transactions1

WHERE cn >=3;


-- SOLVED IN POSTGRESQL

```

Question 148:

```sql

-- Q.148 You are given a table of PayPal payments showing the payer, the recipient, and the amount paid.
-- A two-way unique relationship is established WHEN two people sEND money back and forth. Write a query
-- to find the number of two-way unique relationships in this data.


WITH temp_payments AS (
                        SELECT
                                DISTINCT p1.payer_id,
                                p1.recipient_id
                        FROM
                                payments p1
                        INNER JOIN
                                payments p2
                        ON
                                p1.payer_id = p2.recipient_id
                        AND
                                p2.payer_id = p1.recipient_id
                        AND
                                p1.payer_id < p2.payer_id
        )

SELECT
        COUNT(*) unique_relationships
FROM
        temp_payments;

```

Question 149:

```sql

-- Q.149 Write a query to obtain the list of customers whose first transaction was valued at $50 or more.
-- Output the number of users.


WITH temp_transactions AS (
                            SELECT
                                    user_id,
                                    transaction_date,
                                    spend,
                                    ROW_NUMBER() OVER(PARTITION BY user_id ORDER BY transaction_date) row_num
                            FROM
                                    user_transactions
                    )

SELECT
        COUNT(DISTINCT user_id) as users
FROM
        temp_transactions
WHERE
        row_num = 1
        and
        spend >= 50;

```

Question 150:

```sql

-- Q.150 Write a query to obtain the SUM of the odd-numbered and even-numbered meASurements on a particular day,
-- in two different columns.


WITH temp_measurments AS (
                            SELECT
                                    measurement_value,
                                    measurement_time,
                                    ROW_NUMBER() OVER(PARTITION BY measurement_time::DATE ORDER BY measurement_time) row_num
                            FROM
                                    measurments
                )

SELECT
        measurement_time::DATE,,
        ROUND(SUM(
            CASE
                WHEN row_num % 2 <> 0 THEN measurment_value
                END),2) AS odd_value,
        ROUND(SUM(
            CASE
                WHEN row_num % 2 = 0 THEN measurment_value
                END),2) AS even_value
FROM
        temp_measurments
GROUP BY
        measurement_time::DATE,
ORDER BY
        measurement_time;


-- SOLVED IN POSTGRESQL

```

Question 151:

```sql

-- Q.151 In an effort to identify high-value customers, Amazon asked for your help to obtain data
-- about users who go on shopping sprees. A shopping spree occurs when a user makes purchASes on 3
-- or more consecutive days. List the user IDs who have gone on at leASt 1 shopping spree in ascending order.


SELECT
        DISTINCT user_id
FROM
    (
        SELECT
                user_id,
                transaction_date,
                rn,
                transaction_date :: date - rn::integer,
                COUNT(transaction_date :: date - rn::integer) OVER(PARTITION BY user_id) AS cn
        FROM
                (
                    SELECT
                            user_id,
                            transaction_date,
                            ROW_NUMBER() OVER(PARTITION BY user_id ORDER BY transaction_date) AS rn
                    FROM
                            transactions
                ) temp_transactions
    )temp_transactions1

WHERE cn >=3;

```

Question 152:

```sql

-- Q.152 The Airbnb Booking RecommENDations team is trying to understand the "substitutability" of two rentals
-- and whether one rental is a good substitute for another. They want you to write a query to find the unique
-- combination of two Airbnb rentals WITH the same exact amenities offered. Output the COUNT of the unique
-- combination of Airbnb rentals.


WITH temp_amenities AS (
                        SELECT
                                rental_id,
                                amenity,
                                COUNT(amenity) over(partition by rental_id) AS no_of_amenities
                        FROM
                                rental_amenities
            ),

temp_amenities2 AS (
                        SELECT
                                COUNT(*)
                        FROM
                                temp_amenities a
                        inner join
                                temp_amenities b
                        on
                                a.no_of_amenities = b.no_of_amenities
                        AND
                                a.amenity = b.amenity
                        AND
                                a.rental_id<>b.rental_id
                        GROUP BY
                                a.rental_id,
                                b.rental_id,
                                a.no_of_amenities
                        HAVING
                                COUNT(*) = a.no_of_amenities
            )


SELECT
        CEIL(COUNT(*)/2) as matching_airbnb
FROM
        temp_amenities2;

```

Question 153:

```sql

-- Q.153 Write a query to calculate the return on ad spend (ROAS) for each advertiser
-- across all ad campaigns. Round your answer to 2 decimal places, and order your
-- output by the advertiser_id.


SELECT
        advertiser_id,
        ROUND(SUM(revenue) / SUM(spend), 2) AS ROAS
FROM
        ad_campaigns
GROUP BY
        advertiser_id
ORDER BY
        advertiser_id;

```

Question 154:

```sql

-- Q.154 Write a query that shows the following data for each compensation outlier:
-- employee ID, salary, and whether they are potentially overpaid or potentially underpaid


WITH temp_compensation AS (
                            SELECT
                                    employee_id,
                                    salary,
                                    title,
                                    ROUND(AVG(salary) over(PARTITION BY title),2) AS avg_salary
                            FROM
                                    employee_pay
                ),
    temp_compensation2 AS (
                            SELECT
                                    employee_id,
                                    salary,
                                    CASE
                                        WHEN salary > 2 * avg_salary THEN 'Overpaid'
                                        WHEN salary < avg_salary/2 THEN 'Underpaid'
                                        END AS status
                            FROM
                                    temp_compensation
                )

SELECT
        employee_id,
                salary,
                status
FROM
        temp_compensation2
WHERE
        status is not null;

```

Question 155:

```sql

-- Q.155 You are given a table of PayPal payments showing the payer, the recipient, and the amount paid.
-- A two-way unique relationship is established WHEN two people sEND money back and forth. Write a query
-- to find the number of two-way unique relationships in this data.


WITH temp_payments AS (
                        SELECT
                                DISTINCT p1.payer_id,
                                p1.recipient_id
                        FROM
                                payments p1
                        INNER JOIN
                                payments p2
                        ON
                                p1.payer_id = p2.recipient_id
                        AND
                                p2.payer_id = p1.recipient_id
                        AND
                                p1.payer_id < p2.payer_id
        )

SELECT
        COUNT(*) unique_relationships
FROM
        temp_payments;

```

Question 156:

```sql

-- Q.156 Assume you are given the table below containing information on user
-- purchASes. Write a query to obtain the number of users who purchASed the
-- same product on two or more different days. Output the number of unique users.


SELECT
        COUNT(DISTINCT user_id) AS repeat_purchasers
FROM (
        SELECT
                user_id
        FROM
                purchases
        GROUP BY
                user_id,
                product_id
        HAVING
                COUNT(DISTINCT purchase_date) > 1
    ) temp;

```

Question 157:

```sql

-- Q.157 Say you have access to all the transactions for a given merchant acCOUNT.
-- Write a query to print the cumulative balance of the merchant acCOUNT at the END
-- of each day, WITH the total balance reset back to zero at the END of the month.
-- Output the transaction date and cumulative balance.


SELECT
       DISTINCT date(transaction_date),
       SUM(
           CASE
               WHEN type = 'deposit' THEN amount
               ELSE -amount
               END
           ) OVER(PARTITION BY EXTRACT(MONTH FROM transaction_date) ORDER BY DATE(transaction_date)) AS balance
FROM
       transactions;


-- SOLVED IN POSTGRESQL

```

Question 158:

```sql

-- Q.158 Assume you are given the table below containing information on
-- Amazon customers and their spend on products belonging to various
-- categories. Identify the top two highest-grossing products within each
-- category in 2022. Output the category, product, and total spend.


WITH temp_product_details AS (
                                SELECT
                                        category,
                                        product,
                                        spend,
                                        SUM(spend) OVER(PARTITION BY category, product) total_spend
                                FROM
                                        product_spend
                                WHERE
                                        EXTRACT(YEAR FROM transaction_date) = 2022
                ),

    temp_product_details1 AS (
                                SELECT
                                        DISTINCT category,
                                        product,
                                        total_spend ,
                                        DENSE_RANK() OVER(PARTITION BY category order by total_spend desc) row_num
                                FROM
                                        temp_product_details
               )


SELECT
        DISTINCT category,
        product,
        total_spend
FROM
        temp_product_details1
WHERE
        row_num <=2
ORDER BY
        category,
        total_spend DESC;


-- SOLVED IN POSTGRESQL

```

Question 159:

```sql

-- Q.159 Facebook is analysing its user signup data for June 2022.
-- Write a query to generate the churn rate by week in June 2022.
-- Output the week number (1, 2, 3, 4, ...) and the corresponding
-- churn rate rounded to 2 decimal places.


-- Approach 1


WITH temp_churn_rate AS (
                        SELECT
                                user_id,
                                signup_date,
                                last_login,
                                DATEDIFF(last_login, signup_date) diff,
                                EXTRACT(WEEK FROM signup_date) AS week_no,
                                WEEK(signup_date,5) -  WEEK(DATE_SUB(signup_date, INTERVAL DAYOFMONTH(signup_date)-1 DAY),5)+1 AS ranking
                        FROM
                                users
                        WHERE
                                EXTRACT(MONTH FROM signup_date) = 6
                                AND
                                EXTRACT(YEAR FROM signup_date) = 2022
),
temp_churn_rate2 AS (
                        SELECT
                                ranking,
                                COUNT(ranking) AS total_users,
                                COUNT(
                                CASE
                                    WHEN diff <= 28 THEN 1
                                    END
                                    ) AS total_churns
                        FROM
                                temp_churn_rate
                        GROUP BY
                                ranking
)

SELECT
        ranking AS week,
        ROUND((total_churns/total_users) * 100 ,2) AS churn_rate
FROM
        temp_churn_rate2
ORDER BY
        ranking;


-- Approach 2


WITH temp_churn_rate AS (
                            SELECT
                                    user_id,
                                    signup_date,
                                    last_login,
                                    DATEDIFF(last_login, signup_date) diff,
                                    EXTRACT(WEEK FROM signup_date) AS week_no,
                                    DENSE_RANK() OVER(ORDER BY EXTRACT(WEEK FROM signup_date)) ranking
                            FROM
                                    users
                            WHERE
                                    EXTRACT(MONTH FROM signup_date) = 6
                                    AND
                                    EXTRACT(YEAR FROM signup_date) = 2022
        ),
    temp_churn_rate2 AS (
                            SELECT
                                    ranking,
                                    COUNT(ranking) AS total_users,
                                    COUNT(
                                    CASE
                                        WHEN diff <= 28 THEN 1
                                        END
                                        ) AS total_churns
                            FROM
                                    temp_churn_rate
                            GROUP BY
                                    ranking
        )

SELECT
        ranking AS week,
        ROUND((total_churns/total_users) * 100 ,2) AS churn_rate
FROM
        temp_churn_rate2
ORDER BY
        ranking;

```