

SQL Challenge

Submitted By:
Abhishek Chaubey

Contents:

1. [SQL Order of Execution](#)

2. [Set 1](#)

- 2.1. [Question 1](#)
- 2.2. [Question 2](#)
- 2.3. [Question 3](#)
- 2.4. [Question 4](#)
- 2.5. [Question 5](#)
- 2.6. [Question 6](#)
- 2.7. [Question 7](#)
- 2.8. [Question 8](#)
- 2.9. [Question 9](#)
- 2.10. [Question 10](#)
- 2.11. [Question 11](#)
- 2.12. [Question 12](#)
- 2.13. [Question 13](#)
- 2.14. [Question 14](#)
- 2.15. [Question 15](#)
- 2.16. [Question 16](#)
- 2.17. [Question 17](#)
- 2.18. [Question 18](#)
- 2.19. [Question 19](#)
- 2.20. [Question 20](#)
- 2.21. [Question 21](#)
- 2.22. [Question 22](#)
- 2.23. [Question 23](#)
- 2.24. [Question 24](#)
- 2.25. [Question 25](#)
- 2.26. [Question 26](#)
- 2.27. [Question 27](#)
- 2.28. [Question 28](#)
- 2.29. [Question 29](#)
- 2.30. [Question 30](#)
- 2.31. [Question 31](#)
- 2.32. [Question 32](#)
- 2.33. [Question 33](#)
- 2.34. [Question 34](#)
- 2.35. [Question 35](#)
- 2.36. [Question 36](#)

- 2.37. [Question 37](#)
- 2.38. [Question 38](#)
- 2.39. [Question 39](#)
- 2.40. [Question 40](#)
- 2.41. [Question 41](#)
- 2.42. [Question 42](#)
- 2.43. [Question 43](#)
- 2.44. [Question 44](#)
- 2.45. [Question 45](#)
- 2.46. [Question 46](#)
- 2.47. [Question 47](#)
- 2.48. [Question 48](#)
- 2.49. [Question 49](#)
- 2.50. [Question 50](#)

3. Set 2

- 3.1. [Question 51](#)
- 3.2. [Question 52](#)
- 3.3. [Question 53](#)
- 3.4. [Question 54](#)
- 3.5. [Question 55](#)
- 3.6. [Question 56](#)
- 3.7. [Question 57](#)
- 3.8. [Question 58](#)
- 3.9. [Question 59](#)
- 3.10. [Question 60](#)
- 3.11. [Question 61](#)
- 3.12. [Question 62](#)
- 3.13. [Question 63](#)
- 3.14. [Question 64](#)
- 3.15. [Question 65](#)
- 3.16. [Question 66](#)
- 3.17. [Question 67](#)
- 3.18. [Question 68](#)
- 3.19. [Question 69](#)
- 3.20. [Question 70](#)
- 3.21. [Question 71](#)
- 3.22. [Question 72](#)
- 3.23. [Question 73](#)
- 3.24. [Question 74](#)
- 3.25. [Question 75](#)
- 3.26. [Question 76](#)
- 3.27. [Question 77](#)

- 3.28. [Question 78](#)
- 3.29. [Question 79](#)
- 3.30. [Question 80](#)
- 3.31. [Question 81](#)
- 3.32. [Question 82](#)
- 3.33. [Question 83](#)
- 3.34. [Question 84](#)
- 3.35. [Question 85](#)
- 3.36. [Question 86](#)
- 3.37. [Question 87](#)
- 3.38. [Question 88](#)
- 3.39. [Question 89](#)
- 3.40. [Question 90](#)
- 3.41. [Question 91](#)
- 3.42. [Question 92](#)
- 3.43. [Question 93](#)
- 3.44. [Question 94](#)
- 3.45. [Question 95](#)
- 3.46. [Question 96](#)
- 3.47. [Question 97](#)
- 3.48. [Question 98](#)
- 3.49. [Question 99](#)
- 3.50. [Question 100](#)

4. [Set 3](#)

- 4.1. [Question 101](#)
- 4.2. [Question 102](#)
- 4.3. [Question 103](#)
- 4.4. [Question 104](#)
- 4.5. [Question 105](#)
- 4.6. [Question 106](#)
- 4.7. [Question 107](#)
- 4.8. [Question 108](#)
- 4.9. [Question 109](#)
- 4.10. [Question 110](#)
- 4.11. [Question 111](#)
- 4.12. [Question 112](#)
- 4.13. [Question 113](#)
- 4.14. [Question 114](#)
- 4.15. [Question 115](#)
- 4.16. [Question 116](#)

- 4.17. [Question 117](#)
- 4.18. [Question 118](#)
- 4.19. [Question 119](#)
- 4.20. [Question 120](#)
- 4.21. [Question 121](#)
- 4.22. [Question 122](#)
- 4.23. [Question 123](#)
- 4.24. [Question 124](#)
- 4.25. [Question 125](#)
- 4.26. [Question 126](#)
- 4.27. [Question 127](#)
- 4.28. [Question 128](#)
- 4.29. [Question 129](#)
- 4.30. [Question 130](#)
- 4.31. [Question 131](#)
- 4.32. [Question 132](#)
- 4.33. [Question 133](#)
- 4.34. [Question 134](#)
- 4.35. [Question 135](#)
- 4.36. [Question 136](#)
- 4.37. [Question 137](#)
- 4.38. [Question 138](#)
- 4.39. [Question 139](#)
- 4.40. [Question 140](#)
- 4.41. [Question 141](#)
- 4.42. [Question 142](#)
- 4.43. [Question 143](#)
- 4.44. [Question 144](#)
- 4.45. [Question 145](#)
- 4.46. [Question 146](#)
- 4.47. [Question 147](#)
- 4.48. [Question 148](#)
- 4.49. [Question 149](#)
- 4.50. [Question 150](#)
- 4.51. [Question 151](#)
- 4.52. [Question 152](#)
- 4.53. [Question 153](#)
- 4.54. [Question 154](#)
- 4.55. [Question 155](#)
- 4.56. [Question 156](#)

4.57. [Question 157](#)

4.58. [Question 158](#)

4.59. [Question 159](#)

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:

```
1
2 -- Q.1 Query all columns for all American cities in the CITY table with
3 -- populations larger than 100000. The countryCode for America is USA.
4
5
6 SELECT
7     id,
8     name,
9     countryCode,
10    district,
11    population
12 FROM
13    city
14 WHERE
15    population > 100000
16    AND
17    countryCode = 'USA';
18
19
```

Question 2:

```
1
2 -- Q.2 Query the name field for all American cities in the CITY table with
3 -- populations larger than 120000. The countryCode for America is USA.
4
5
6 SELECT
7      name
8 FROM
9      city
10 WHERE
11      population > 120000
12      AND
13      countryCode = 'USA';
14
15
```

Question 3:

```
1
2 -- Q.3 Query all columns (attributes) for every row in the CITY table.
3
4
5 SELECT
6      id,
7      name,
8      countryCode,
9      district,
10     population
11 FROM
12     city;
13
```

Question 4:

```
1
2 -- Q.4 Query all columns for a city in CITY with the id 1661.
3
4
5 SELECT
6      id,
7      name,
8      countrycode,
9      district,
10     population
11 FROM
12     city
13 WHERE
14     id = 1661;
15
16
```

Question 5:

```
1
2 -- Q.5 Query all attributes of every Japanese city in the CITY table.
3 -- The countryCODE for Japan is JPN.
4
5
6 SELECT
7     id,
8     name,
9     countrycode,
10    district,
11    population
12 FROM
13    city
14 WHERE
15     countrycode = 'JPN';
16
17
```

Question 6:

```
● ● ●  
1  
2 -- Q.6 Query the names of all the Japanese cities in the CITY table.  
3 -- The countryCODE for Japan is JPN.  
4  
5  
6 SELECT  
7      name  
8 FROM  
9      city  
10 WHERE  
11      countrycode = 'JPN';  
12  
13
```

Question 7:

```
● ● ●  
1  
2 -- Q.7 Query a list of CITY and STATE from the station table.  
3  
4  
5 SELECT  
6      city,  
7      state  
8 FROM  
9      station;  
10
```

Question 8:

```
1
2 -- Q.8 Query a list of CITY names from STATION for cities that have an even ID number.
3 -- Print the results in any order, but exclude duplicates from the answer.
4
5
6 SELECT
7     DISTINCT city
8 FROM
9     station
10 WHERE
11     id%2 = 0;
12
```

Question 9:

```
1
2 -- Q.9 Find the difference between the total number of CITY entries in the table and the
3 -- number of distinct CITY entries in the table.
4
5
6 SELECT
7     (COUNT(city) - COUNT(DISTINCT city)) AS difference
8 FROM
9     station;
10
```

Question 10:

```
1
2 -- Q.10 Query the two cities in STATION with the shortest and longest CITY names, as well as their
3 -- respective lengths (i.e.: number of characters in the name). If there is more than one smallest
4 -- or largest city, choose the one that comes first when ordered alphabetically.
5
6
7 -- APPROACH 1:
8
9
10 (SELECT
11     city,
12     Length(city) AS len
13 FROM
14     station
15 ORDER BY
16     Length(city),
17     city
18 LIMIT 1)
19
20 UNION ALL
21
22 (SELECT
23     city,
24     Length(city) AS len
25 FROM
26     station
27 ORDER BY
28     Length(city) DESC,
29     city
30 LIMIT 1);
31
32
33
34 -- APPROACH 2:
35
36
37 WITH temp_city AS (
38     SELECT
39         city,
40         LENGTH(city) as len,
41         ROW_NUMBER() OVER (ORDER BY LENGTH(city), city) AS smallest,
42         ROW_NUMBER() OVER (ORDER BY LENGTH(city) DESC, city) AS largest
43     FROM
44         station
45     )
46
47 SELECT
48     city,
49     len
50 FROM
51     temp_city
52 WHERE
53     smallest = 1
54     OR
55     largest = 1
56 ORDER BY
57     len;
58
```

Question 11:

```
1
2 -- Q11. Query the list of CITY names starting with vowels (i.e., a, e, i, o, or u) from station.
3 -- Your result cannot contain duplicates.
4
5
6 -- Approach 1
7
8
9 SELECT
10      DISTINCT city
11 FROM
12      station
13 WHERE
14      LEFT(city , 1) IN ('a','e','i','o','u');
15
16
17 -- Approach 2
18
19
20 SELECT
21      DISTINCT city
22 FROM
23      station
24 WHERE
25      SUBSTR(city,1,1) IN ('A','E','I','O','U');
26
27
28 -- Approach 3
29
30
31 SELECT
32      DISTINCT city
33 FROM
34      station
35 WHERE
36      city RLIKE '^[aeiouAEIOU]';
37
38
39 -- Approach 4
40
41
42 SELECT
43      DISTINCT city
44 FROM
45      station
46 WHERE
47      city REGEXP '^[aeiouAEIOU]';
48
49
50 -- Approach 5
51
52
53 SELECT
54      DISTINCT city
55 FROM
56      station
57 WHERE
58      city REGEXP '^[aeiou]';
```

Question 12:

```
1
2 -- Q12. Query the list of CITY names ending with vowels (a, e, i, o, u) from station.
3 -- Your result cannot contain duplicates.
4
5
6 -- Approach 1
7
8
9 SELECT      DISTINCT city
10 FROM        station
11 WHERE       RIGHT(city , 1) IN ('a','e','i','o','u');
12
13
14
15
16
17 -- Approach 2
18
19
20 SELECT      DISTINCT city
21 FROM        station
22 WHERE       SUBSTR(city,-1,1) IN ('A','E','I','O','U');
23
24
25
26
27
28 -- Approach 3
29
30
31 SELECT      DISTINCT city
32 FROM        station
33 WHERE       city RLIKE '.*[aeiouAEIOU]$';
34
35
36
37
38
39 -- Approach 4
40
41
42 SELECT      DISTINCT city
43 FROM        station
44 WHERE       city REGEXP '.*[aeiouAEIOU]$';
45
46
47
48
49
50 -- Approach 5
51
52
53 SELECT      DISTINCT city
54 FROM        station
55 WHERE       city REGEXP '[aeiou]$';
56
57
58
59
```

Question 13:

```
1
2 -- Q13. Query the list of CITY names from station that do not start with vowels.
3 -- Your result cannot contain duplicates.
4
5
6 -- Approach 1
7
8
9 SELECT
10      DISTINCT city
11 FROM
12      station
13 WHERE
14      LEFT(city , 1) NOT IN ('a','e','i','o','u');
15
16
17 -- Approach 2
18
19
20 SELECT
21      DISTINCT city
22 FROM
23      station
24 WHERE
25      SUBSTR(city,1,1) NOT IN ('A','E','I','O','U');
26
27
28 -- Approach 3
29
30
31 SELECT
32      DISTINCT city
33 FROM
34      station
35 WHERE
36      city NOT RLIKE '^[aeiouAEIOU]';
37
38
39 -- Approach 4
40
41
42 SELECT
43      DISTINCT city
44 FROM
45      station
46 WHERE
47      city NOT REGEXP '^[aeiouAEIOU]';
48
49
50 -- Approach 5
51
52
53 SELECT
54      DISTINCT city
55 FROM
56      station
57 WHERE
58      city NOT REGEXP '^[aeiou]';
```

Question 14:

```
1
2 -- Q14. Query the list of CITY names from station that do not end with vowels.
3 -- Your result cannot contain duplicates.
4
5
6 -- Approach 1
7
8
9 SELECT
10      DISTINCT city
11 FROM
12      station
13 WHERE
14      RIGHT(city , 1) NOT IN ('a','e','i','o','u');
15
16
17 -- Approach 2
18
19
20 SELECT
21      DISTINCT city
22 FROM
23      station
24 WHERE
25      SUBSTR(city,-1,1) NOT IN ('A','E','I','O','U');
26
27
28 -- Approach 3
29
30
31 SELECT
32      DISTINCT city
33 FROM
34      station
35 WHERE
36      city NOT RLIKE '.*[aeiouAEIOU]$';
37
38
39 -- Approach 4
40
41
42 SELECT
43      DISTINCT city
44 FROM
45      station
46 WHERE
47      city NOT REGEXP '.*[aeiouAEIOU]$';
48
49
50 -- Approach 5
51
52
53 SELECT
54      DISTINCT city
55 FROM
56      station
57 WHERE
58      city NOT REGEXP '[aeiou]$';
59
```

Question 15:

```
1
2 -- Q15. Query the list of CITY names from station that either do not start with vowels or
3 -- do not end with vowels. Your result cannot contain duplicates.
4
5
6 -- Approach 1
7
8
9 SELECT
10      DISTINCT city
11 FROM
12      station
13 WHERE
14      SUBSTR(city,1,1) NOT IN ('A','E','I','O','U')
15      OR
16      SUBSTR(city,-1,1) NOT IN ('A','E','I','O','U');
17
18
19 -- Approach 2
20
21
22 SELECT
23      DISTINCT city
24 FROM
25      station
26 WHERE
27      city RLIKE '^[^aeiouAEIOU].*|.*[^AEIOUaeiou]$';
28
```

Question 16:

```
1
2 -- Q16. Query the list of CITY names from station that do not start with vowels and
3 -- do not end with vowels. Your result cannot contain duplicates.
4
5
6 -- Approach 1
7
8
9 SELECT
10      DISTINCT city
11 FROM
12      station
13 WHERE
14      SUBSTR(city,1,1) NOT IN ('A','E','I','O','U')
15      AND
16      SUBSTR(city,-1,1) NOT IN ('A','E','I','O','U');
17
18
19 -- Approach 2
20
21
22 SELECT
23      DISTINCT city
24 FROM
25      station
26 WHERE
27      city NOT REGEXP '^[aeiou]|[aeiou]$';
28
29
30 -- Approach 3
31
32
33 SELECT
34      DISTINCT city
35 FROM
36      station
37 WHERE
38      city REGEXP '^[^aeiou].*[^aeiou]$';
39
```

Question 17:

```
1
2 -- Q.17 Write an SQL query that reports the products that were only sold in the first quarter of 2019.
3 -- That is, between 2019-01-01 and 2019-03-31 inclusive. Return the result table in any order.
4
5
6 -- Approach 1
7
8
9 SELECT
10      p.product_id,
11      p.product_name
12 FROM
13      product p
14 INNER JOIN
15      sales s
16 ON
17      p.product_id = s.product_id
18 WHERE
19      p.product_id
20 NOT IN (
21      SELECT
22          product_id
23      FROM
24          sales
25      WHERE
26          sales_date > '2019-03-31'
27      )
28 GROUP BY
29      p.product_id
30 HAVING
31      MAX(s.sales_date) <='2019-03-31'
32      AND
33      MIN(s.sales_date) >='2019-01-01';
34
35
36 -- Approach 2
37
38
39 SELECT
40      p.product_id,
41      p.product_name
42 FROM
43      product p
44 INNER JOIN
45      sales s
46 ON
47      p.product_id = s.product_id
48 GROUP BY
49      p.product_id,
50      p.product_name
51 HAVING
52      MIN(s.sales_date) >= '2019-01-01' AND MAX(s.sales_date) <= '2019-03-31';
53
```

Question 18:

```
1
2 -- Q.18 Write an SQL query to find all the authors that viewed at least one of their own articles.
3 -- Return the result table sorted by id in ascending order.
4
5
6 SELECT
7     DISTINCT a.author_id
8 FROM
9     views a
10 JOIN
11     views v
12 ON
13     a.author_id = v.viewer_id
14 ORDER BY
15     a.author_id;
16
```

Question 19:

```
1
2 -- Q.19 Write an SQL query to find the percentage of immediate orders in the table,
3 -- rounded to 2 decimal places.
4
5
6 SELECT
7     ROUND(immediate_orders * 100 / total_orders, 2) AS immediate_orders_perct
8 FROM
9     (
10         SELECT
11             COUNT(
12                 CASE
13                     WHEN order_date = customer_preferred_delivery_date
14                     THEN customer_id
15                 END
16                 ) AS immediate_orders,
17             COUNT(delivery_id) AS total_orders
18         FROM
19             delivery
20     ) temp ;
21
```

Question 20:

```
1
2 -- Q.20 Write an SQL query to find the ctr of each Ad. Round ctr to two decimal points.
3 -- Return the result table ordered by ctr in descending order and by ad_id in
4 -- ascending order in case of a tie.
5
6
7 SELECT
8     ad_id,
9     CASE
10        WHEN (num_of_clicks * 100) / (num_of_clicks + num_of_views) IS NULL THEN 0
11        ELSE ROUND((num_of_clicks * 100) / (num_of_clicks + num_of_views), 2)
12        END AS ctr
13 FROM
14     (
15         SELECT
16             ad_id,
17             COUNT(
18                 CASE
19                     WHEN action = 'CLICKED' THEN ad_id
20                     END
21                 ) AS num_of_clicks,
22             COUNT(
23                 CASE WHEN action = 'VIEWED' THEN ad_id
24                     END
25                 ) AS num_of_views
26         FROM
27             ads
28         GROUP BY
29             ad_id
30     ) temp_ads
31 ORDER BY
32     ctr DESC,
33     ad_id ASC;
34
```

Question 21:

```
1
2 -- Q.21 Write an SQL query to find the team size of each of the employees.
3
4
5 SELECT
6     employee_id,
7     COUNT(employee_id) OVER(PARTITION BY team_id) AS team_size
8 FROM
9     employee
10 ORDER BY
11     employee_id;
12
```

Question 22:

```
1
2 -- Q.22 Write an SQL query to find the type of weather in each country for November 2019. The type of weather is:
3 -- • Cold if the average weather_state is less than or equal 15,
4 -- • Hot if the average weather_state is greater than or equal to 25, and
5 -- • Warm otherwise.
6 -- Return result table in any order.
7
8
9 SELECT
10    c.country_name,
11    CASE
12        WHEN AVG(weather_state) <= 15 THEN 'COLD'
13        WHEN AVG(weather_state) >= 25 THEN 'HOT'
14        ELSE 'WARM'
15        END AS avg_weather
16 FROM
17    countries_c
18 INNER JOIN
19    weather w
20 ON
21    c.country_id = w.country_id
22 WHERE
23    day BETWEEN '2019-11-01' AND '2019-11-30'
24 GROUP BY
25    c.country_id,
26    c.country_name;
27
```

Question 23:

```
1
2 -- Q.23 Write an SQL query to find the average selling price for each product.
3 -- average_price should be rounded to 2 decimal places.
4
5
6 SELECT
7     p.product_id,
8     ROUND(SUM(p.price * u.units)/SUM(u.units), 2) AS average_price
9 FROM
10    units_sold u
11 INNER JOIN
12    prices p
13 ON
14     p.product_id = u.product_id
15 WHERE
16     u.purchase_date BETWEEN p.start_date AND p.end_date
17 GROUP BY
18     p.product_id;
19
```

Question 24:

```
1
2 -- Q.24 Write an SQL query to report the first login date for each player.
3 -- Return the result table in any order.
4
5
6 -- Approach 1
7
8
9 WITH temp_activity AS (
10             SELECT
11                 player_id,
12                 event_date as first_login,
13                 ROW_NUMBER() OVER(PARTITION BY player_id ORDER BY event_date) as ranking
14             FROM
15                 activity
16         )
17
18 SELECT
19     player_id,
20     first_login
21 FROM
22     temp_activity
23 WHERE
24     ranking = 1;
25
26
27 -- Approach 2
28
29
30 SELECT
31     player_id,
32     first_login
33 FROM
34     (
35         SELECT
36             player_id,
37             event_date as first_login,
38             ROW_NUMBER() OVER(PARTITION BY player_id ORDER BY event_date) ranking
39             FROM
40                 activity
41     ) temp_activity
42 WHERE
43     ranking = 1;
```

Question 25:

```
1
2 -- Q.25 Write an SQL query to report the device that is first logged in for each player.
3 -- Return the result table in any order.
4
5
6 -- Approach 1
7
8
9 WITH temp_activity AS (
10     SELECT
11         player_id,
12         device_id,
13         ROW_NUMBER() OVER(PARTITION BY player_id ORDER BY device_id) ranking
14     FROM
15         activity
16     )
17
18 SELECT
19     player_id,
20     device_id
21 FROM
22     temp_activity
23 WHERE
24     ranking = 1;
25
26
27 -- Approach 2
28
29
30 SELECT
31     player_id,
32     device_id
33 FROM
34     (
35         SELECT
36             player_id,
37             device_id,
38             ROW_NUMBER() OVER(PARTITION BY player_id ORDER BY device_id) as ranking
39         FROM
40             activity
41     ) temp_activity
42 WHERE
43     ranking = 1;
```

Question 26:

```
1
2 -- Q.26 Write an SQL query to get the names of products that have at least 100 units
3 -- ordered in February 2020 and their amount.
4
5
6 SELECT
7     p.product_name,
8     SUM(o.unit) AS total_unit_sold
9 FROM
10    products p
11 INNER JOIN
12        orders o
13 ON
14     p.product_id = o.product_id
15 WHERE
16     order_date BETWEEN '2020-02-01' AND '2020-02-28'
17 GROUP BY
18     p.product_id
19 HAVING
20     SUM(o.unit) >= 100;
21
```

Question 27:

```
1
2 -- Q.27 Write an SQL query to find the users who have valid emails.
3
4
5 SELECT
6     user_id,
7     name,
8     mail
9 FROM
10    users
11 WHERE
12     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}$';
13
```

Question 28:

```
1
2 -- Q.28 Write an SQL query to report the customer_id and customer_name of customers who have spent
3 -- at least $100 in each month of June and July 2020. Return the result table in any order.
4
5
6 SELECT
7     customer_id,
8     name
9 FROM
10    (
11        SELECT
12            c.customer_id,
13            c.name,
14            EXTRACT(MONTH FROM o.order_date) AS month_extracted,
15            SUM(o.quantity * p.price) AS total_spent
16        FROM
17            orders o
18        INNER JOIN
19            customers c
20        ON
21            c.customer_id = o.customer_id
22        INNER JOIN
23            products p
24        ON
25            p.product_id = o.product_id
26        WHERE
27            o.order_date BETWEEN '2020-06-01' AND '2020-07-31'
28        GROUP BY
29            c.customer_id,
30            c.name,
31            EXTRACT(MONTH FROM o.order_date)
32    ) temp_customers
33
34 WHERE
35     total_spent >= 100
36 GROUP BY
37     customer_id
38 HAVING
39     COUNT(customer_id) = 2;
40
```

Question 29:

```
1
2 -- Q.29 Write an SQL query to report the distinct titles of the kid-friendly movies streamed in June 2020.
3 -- Return the result table in any order.
4
5
6 SELECT
7     DISTINCT c.title
8 FROM
9     content c
10 INNER JOIN
11     tv_program t
12 ON
13     c.content_id = T.content_id
14 WHERE
15     c.kids_content = 'Y'
16     AND
17     c.content_type = 'movies'
18     AND
19     T.program_date BETWEEN '2020-06-01' AND '2020-06-30';
20
```

Question 30:

```
1
2 -- Q.30 Write an SQL query to find the npv of each query of the Queries table.
3 -- Return the result table in any order.
4
5
6 SELECT
7     DISTINCT n.id,
8     n.year,
9     n.npv
10 FROM
11     queries q
12 INNER JOIN
13     npv n
14 ON
15     n.id = q.id
16     AND
17     n.year = q.year;
18
```

Question 31:

```
1
2 -- Q.31 Write an SQL query to find the npv of each query of the Queries table.
3 -- Return the result table in any order.
4
5
6 SELECT
7     DISTINCT n.id,
8     n.year,
9     n.npv
10 FROM
11     queries q
12 INNER JOIN
13     npv n
14 ON
15     n.id = q.id
16     AND
17     n.year = q.year;
18
```

Question 32:

```
1
2 -- Q.32 Write an SQL query to show the unique id of each user, If a user does not have a
3 -- unique id replace just show null. Return the result table in any order.
4
5
6 SELECT
7     eu.unique_id,
8     e.name
9 FROM
10    employees e
11 LEFT JOIN
12    employees_uni eu
13 ON
14     e.id = eu.id
15 ORDER BY
16     e.name;
17
```

Question 33:

```
1
2 -- Q.33 Write an SQL query to report the distance travelled by each user. Return the result table ordered by travelled_distance
3 -- in descending order, if two or more users travelled the same distance, order them by their name in ascending order.
4
5
6 SELECT
7     u.name AS riders,
8     COALESCE(SUM(r.distance), 0) AS distance_travelled
9 FROM
10    users u
11 LEFT JOIN
12    rides r
13 ON
14     u.id = r.user_id
15 GROUP BY
16     u.name
17 ORDER BY
18     distance_travelled DESC,
19     riders;
20
```

Question 34:

```
1
2 -- Q.34 Write an SQL query to get the names of products that have at least 100 units
3 -- ordered in February 2020 and their amount.
4
5
6 SELECT
7     p.product_name,
8     SUM(o.unit) AS total_unit_sold
9 FROM
10    products p
11 INNER JOIN
12    orders o
13 ON
14     p.product_id = o.product_id
15 WHERE
16     order_date BETWEEN '2020-02-01' AND '2020-02-28'
17 GROUP BY
18     p.product_id
19 HAVING
20     SUM(o.unit) >= 100;
21
```

Question 35:

```
1 -- Q.35 Write an SQL query to:
2 -- • Find the name of the user who has rated the greatest number of movies. In case of a tie,
3 -- return the lexicographically smaller user name.
4
5
6 SELECT
7     u.name as user
8 FROM
9     users u
10 INNER JOIN
11     movie_rating mr
12 ON
13     u.user_id = mr.user_id
14 GROUP BY
15     u.user_id
16 ORDER BY
17     COUNT(u.name) DESC,
18     LENGTH(u.name)
19 LIMIT 1;
20
21
22 -- • Find the movie name with the highest average rating in February 2020. In case of a tie,
23 -- return the lexicographically smaller movie name.
24
25
26 SELECT
27     m.title
28 FROM
29     movies m
30 INNER JOIN
31     movie_rating mr
32 ON
33     m.movie_id = mr.movie_id
34 WHERE
35     mr.created_at BETWEEN '2020-02-01' AND '2020-02-28'
36 GROUP BY
37     m.movie_id
38 ORDER BY
39     AVG(rating) DESC,
40     m.title
41 LIMIT 1;
42
```

Question 36:

```
1
2 -- Q.36 Write an SQL query to report the distance travelled by each user. Return the result table ordered by travelled_distance
3 -- in descending order, if two or more users travelled the same distance, order them by their name in ascending order.
4
5
6 SELECT
7     u.name AS riders,
8     COALESCE(SUM(r.distance), 0) AS distance_travelled
9 FROM
10    users u
11 LEFT JOIN
12    rides r
13 ON
14     u.id = r.user_id
15 GROUP BY
16     u.name
17 ORDER BY
18     distance_travelled DESC,
19     riders;
20
```

Question 37:

```
1
2 -- Q.38 Write an SQL query to find the id and the name of all students who are enrolled
3 -- in departments that no longer exist. Return the result table in any order.
4
5
6 SELECT
7     id,
8     name
9 FROM
10    students
11 WHERE
12     department_id NOT IN (
13         SELECT
14             id
15             FROM
16                 departments
17     );
18
```

Question 38:

```
1
2 -- Q.38 Write an SQL query to find the id and the name of all students who are enrolled
3 -- in departments that no longer exist. Return the result table in any order.
4
5
6 SELECT
7     id,
8     name
9 FROM
10    students
11 WHERE
12     department_id NOT IN (
13         SELECT
14             id
15             FROM
16                 departments
17 );
18
```

Question 39:

```
1
2 -- Q.39 Write an SQL query to report the number of calls and the total call duration between
3 -- each pair of distinct persons (person1, person2) where person1 < person2.
4 -- Return the result table in any order.
5
6
7 -- Approach 1
8
9
10 SELECT
11     CASE
12         WHEN from_id < to_id THEN from_id
13         ELSE to_id
14     END AS person1,
15     CASE
16         WHEN from_id < to_id THEN to_id
17         ELSE from_id
18     END AS person2,
19     COUNT(*) AS call_count,
20     SUM(duration) AS total_duration
21 FROM
22     calls
23 GROUP BY
24     person1,
25     person2;
26
27
28 -- Approach 2
29
30
31 SELECT
32     least(from_id, to_id) AS person1,
33     greatest(from_id,to_id) AS person2,
34     COUNT(*) AS call_count,
35     SUM(duration) AS total_duration
36 FROM
37     calls
38 GROUP BY
39     person1,
40     person2;
41
```

Question 40:

```
1
2 -- Q.40 Write an SQL query to find the average selling price for each product.
3 -- average_price should be rounded to 2 decimal places.
4
5
6 SELECT
7     p.product_id,
8     ROUND(SUM(p.price * u.units)/SUM(u.units), 2) AS average_price
9 FROM
10    units_sold u
11 INNER JOIN
12    prices p
13 ON
14     p.product_id = u.product_id
15 WHERE
16     u.purchase_date BETWEEN p.start_date AND p.end_date
17 GROUP BY
18     p.product_id;
19
```

Question 41:

```
1
2 -- Q.41 Write an SQL query to report the number of cubic feet of volume the inventory
3 -- occupies in each warehouse. Return the result table in any order.
4
5
6 SELECT
7     w.name AS warehouse_name,
8     SUM(p.length * p.width * p.height * w.units) AS volume
9 FROM
10    warehouse w
11 INNER JOIN
12        products p
13 ON
14     w.product_id = p.product_id
15 GROUP BY
16     w.name;
17
```

Question 42:

```
1 -- Q.42 Write an SQL query to report the difference between the number of
2 -- apples and oranges sold each day. Return the result table ordered by sale_date.
3
4
5 -- Approach 1
6
7
8 SELECT
9     sale_date,
10    difference
11 FROM
12  (
13      SELECT
14          sale_date,
15          sold_num - LEAD(sold_num, 1) OVER(PARTITION BY sale_date) AS difference
16      FROM
17          sales
18  )  temp_sales
19 WHERE
20     difference IS NOT NULL
21 ORDER BY
22     sale_date;
23
24
25 -- Approach 2
26
27
28 SELECT
29     sale_date,
30     SUM(
31         CASE
32             WHEN fruit = 'APPLES' THEN sold_num
33             WHEN fruit = 'ORANGES' THEN -sold_num
34         END
35     ) AS difference
36 FROM
37     sales
38 GROUP BY
39     sale_date
40 ORDER BY
41     sale_date;
42
```

```
1
2 -- Approach 3
3
4
5 SELECT
6     s.sale_date,
7     s.sold_num - ss.sold_num AS difference
8 FROM
9     sales s
10 INNER JOIN
11     sales ss
12 ON
13     s.sale_date = ss.sale_date
14 WHERE
15     s.fruit = 'APPLES' AND ss.fruit = 'ORANGES'
16 ORDER BY
17     sale_date;
18
19
20 -- Approach 4
21
22
23 SELECT
24     sale_date,
25     SUM(IF(fruit = 'APPLES', sold_num, -sold_num)) AS difference
26 FROM
27     sales
28 GROUP BY
29     sale_date
30 ORDER BY
31     sale_date;
32
33
34 -- Approach 5
35
36
37 SELECT
38     sale_date,
39     SUM(
40         CASE
41             WHEN fruit = 'APPLES' THEN sold_num
42             ELSE -sold_num
43         END
44     ) AS difference
45 FROM
46     sales
47 GROUP BY
48     sale_date
49 ORDER BY
50     sale_date;
51
```

Question 43:

```
1
2 -- Q.43 Write an SQL query to report the fraction of players that logged in again on the day after the day
3 -- they first logged in, rounded to 2 decimal places. In other words, you need to count the number of players
4 -- that logged in for at least two consecutive days starting from their first login date,
5 -- then divide that number by the total number of players.
6
7
8 -- Approach 1
9
10
11 WITH temp_activity AS (
12     SELECT
13         player_id,
14         LEAD(event_date, 1) OVER(PARTITION BY player_id ORDER BY event_date) - event_date AS difference
15     FROM
16         activity
17     ),
18
19     temp_activity2 AS (
20         SELECT
21             COUNT(DISTINCT player_id) AS players_count
22         FROM
23             temp_activity
24         WHERE
25             difference = 1
26         GROUP BY
27             player_id
28     )
29
30 SELECT
31     ROUND(COUNT(*) / (SELECT COUNT(DISTINCT player_id) FROM activity), 2) AS fraction
32 FROM
33     temp_activity2;
34
35
36 -- Approach 2
37
38
39 WITH temp_activity AS (
40     SELECT
41         player_id,
42         event_date,
43         DATEDIFF(event_date, lag(event_date) OVER(PARTITION BY player_id ORDER BY event_date)) AS difference
44     FROM
45         activity
46     ),
47
48     temp_activity2 AS (
49         SELECT
50             COUNT(DISTINCT player_id) AS players_count
51         FROM
52             temp_activity
53         WHERE
54             difference = 1
55         GROUP BY
56             player_id
57     )
58
59
60 SELECT
61     ROUND(COUNT(*) / (SELECT COUNT(DISTINCT player_id) FROM activity), 2) AS fraction
62 FROM
63     temp_activity2;
64
```

Question 44:

Question 45:

```
1
2 -- Q.45 Write an SQL query to report the respective department name and number of students majoring in
3 -- each department for all departments in the department table (even ones with no current students).
4 -- Return the result table ordered by student_number in descending order. In case of a tie, order
5 -- them by dept_name alphabetically
6
7 SELECT
8     d.department_name,
9     COUNT(s.student_id) AS number_of_students
10 FROM
11     department d
12 LEFT JOIN
13     student s
14 ON
15     d.dept_id = s.dept_id
16 GROUP BY
17     d.department_name
18 ORDER BY
19     number_of_students DESC,
20     department_name;
21
```

Question 46:

```
1
2 -- Q.46 Write an SQL query to report the customer ids from the Customer table that bought
3 -- all the products in the product table. Return the result table in any order.
4
5
6 SELECT
7     customer_id
8 FROM
9     customer
10 GROUP BY
11     customer_id
12 HAVING
13     COUNT(DISTINCT product_key) = (
14             SELECT
15                 COUNT(DISTINCT product_key)
16             FROM
17                 product
18         );
19
```

Question 47:

```
1 -- Q.47 Write an SQL query that reports the most experienced employees in each project. In case of a tie,
2 -- report all employees with the maximum number of experience years. Return the result table in any order.
3
4
5 -- Approach 1
6
7
8 SELECT
9     project_id,
10    employee_id
11 FROM
12    (
13        SELECT
14            p.project_id,
15            e.employee_id,
16            DENSE_RANK() OVER(PARTITION BY p.project_id ORDER BY e.experience_years DESC) AS ranking
17        FROM
18            employee e
19        INNER JOIN
20            project p
21        ON
22            e.employee_id = p.employee_id
23    ) temp_employee
24 WHERE
25     ranking = 1;
26
27
28 -- Approach 2
29
30
31 SELECT
32     p.project_id,
33     e.employee_id
34 FROM
35     employee e
36 INNER JOIN
37     project p
38 ON
39     e.employee_id = p.employee_id
40 WHERE
41     e.experience_years =
42         (
43             SELECT
44                 MAX(experience_years)
45             FROM
46                 employee
47         );
48
```

Question 48:

```
1
2 -- Q.48 Write an SQL query that reports the books that have sold less than 10 copies in the last year,
3 -- excluding books that have been available for less than one month from today. Assume today is 2019-06-23.
4 -- Return the result table in any order.
5
6
7 SELECT
8     b.book_id, b.name
9 FROM
10    books b
11 LEFT JOIN
12    orders o
13 ON
14    b.book_id = o.book_id
15 WHERE
16    available_from < '2019-05-23'
17    AND
18    (o.dispatch_date BETWEEN '2018-06-23' AND '2019-06-23')
19    OR
20    dispatch_date IS NULL
21 GROUP BY
22    b.book_id,
23    b.name
24 HAVING
25    COALESCE(SUM(o.quantity), 0) < 10;
26
```

Question 49:

```
1
2 -- Q.49 Write a SQL query to find the highest grade with its corresponding course for each student.
3 -- In case of a tie, you should find the course with the smallest course_id. Return the result table
4 -- ordered by student_id in ascending order.
5
6
7 -- Approach 1
8
9
10 SELECT
11     student_id,
12     course_id,
13     grade
14 FROM
15     (
16         SELECT
17             student_id,
18             course_id,
19             grade,
20             ROW_NUMBER() OVER(PARTITION BY student_id ORDER BY grade DESC) AS ranking
21         FROM
22             enrollments
23     ) temp_enrollments
24 WHERE
25     ranking = 1
26 ORDER BY
27     student_id;
28
29
30 -- Approach 2
31
32
33 WITH temp_enrollments AS
34     (
35         SELECT
36             student_id,
37             course_id,
38             grade,
39             ROW_NUMBER() OVER(PARTITION BY student_id ORDER BY grade DESC) AS ranking
40         FROM
41             enrollments
42     )
43
44 SELECT
45     student_id,
46     course_id,
47     grade
48 FROM
49     temp_enrollments
50 WHERE
51     ranking = 1
52 ORDER BY
53     student_id;
54
```

Question 50:

```
1  -- Q.50 Write an SQL query to find the winner in each group. Return the result table in any order.
2
3
4
5  SELECT
6      group_id,
7      players AS player_id
8  FROM
9      (
10         SELECT
11             p.group_id,
12             CASE
13                 WHEN first_player_goals > second_player_goals THEN first_player
14                 WHEN first_player_goals < second_player_goals THEN second_player
15                 WHEN first_player_goals = second_player_goals THEN IF(first_player < second_player, first_player, second_player)
16             END AS players,
17             MAX(IF(first_player_goals > second_player_goals, first_player_goals, second_player_goals)) AS goals,
18             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
19
20     FROM
21         players p
22     INNER JOIN
23         matches m
24     ON
25         m.first_player = p.player_id
26         OR
27         m.second_player = p.player_id
28     GROUP BY
29         p.group_id,
30         players
31 ) temp_matches
32 WHERE
33     ranking = 1;
```

Set 2:

Question 51:

```
1
2 -- Q.51 Write an SQL Query to report the name, population, and area of the big countries.
3 -- Return the result table in any order .
4
5 SELECT
6     name,
7     population,
8     area
9 FROM
10    world
11 WHERE
12     area > 3000000
13     OR
14     population > 25000000;
15
```

Question 52:

```
1
2 -- Q.52 Write an SQL Query to report the names of the customer that are not referred by the customer with id = 2.
3 -- Return the result table in any order.
4
5
6 SELECT
7     name
8 FROM
9     customer
10 WHERE
11     referee_id <> 2
12     OR
13     referee_id IS NULL;
14
```

Question 53:

```
1
2 -- Q.53 Write an SQL Query to report all customers who never order anything.
3 -- Return the result table in any order .
4
5
6 SELECT
7      name AS customers
8 FROM
9      customers
10 WHERE
11      id NOT IN (
12          SELECT
13              customer_id
14          FROM
15              orders
16      );
17
```

Question 54:

```
1
2 -- Q.54 Write an SQL Query to find the team size of each of the employees.
3 -- Return result table in any order .
4
5
6 SELECT
7     employee_id,
8     COUNT(employee_id) OVER(PARTITION BY team_id ORDER BY employee_id
9     ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS team_size
10 FROM
11     employee
12 ORDER BY
13     employee_id;
14
15
```

Question 55:

```
1
2 -- Q.55 Write an SQL Query to find the countries where this company can invest .
3 -- Return the result table in any order .
4
5
6 -- Approach 1
7
8
9 SELECT          name AS country
10    FROM (
11      SELECT          c.name,
12                      SUM(ca.duration) AS call_duration,
13                      COUNT(c.country_code) AS number_of_calls
14      FROM (
15        SELECT          id,
16                      name,
17                      CASE
18                        WHEN LEFT(SUBSTR(phone_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone_number, 1,3), (LENGTH(SUBSTR(phone_number, 1,3))-1))
19                        ELSE SUBSTR(phone_number, 1,3) END AS country_code
20                    FROM person
21      ) temp_person
22    JOIN country c
23    ON temp_person.country_code = c.country_code
24  JOIN calls ca
25  ON temp_person.id = caller_id
26 GROUP BY
27          c.name
28
29 UNION ALL
30
31
32
33
34
35
36
37
38
39
40
41
42 SELECT          c.name,
43                      SUM(ca.duration) AS call_duration,
44                      COUNT(c.country_code) AS number_of_calls
45  FROM (
46    SELECT          id,
47                      name,
48                      CASE
49                        WHEN LEFT(SUBSTR(phone_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone_number, 1,3), (length(SUBSTR(phone_number, 1,3))-1))
50                        ELSE SUBSTR(phone_number, 1,3) END AS country_code
51                    FROM person
52      ) temp_person
53    JOIN country c
54    ON temp_person.country_code = c.country_code
55  JOIN calls ca
56  ON temp_person.id = ca.callee_id
57 GROUP BY
58          c.name
59
60
61
62
63
64
65
66
67      ) temp
68
69 GROUP BY
70          name
71 HAVING
72          SUM(call_duration)/SUM(number_of_calls) > (SELECT AVG(duration) FROM calls);
73
```

```
1  ● ● ●
2  -- Approach 2
3
4
5  WITH temp_person AS (
6      SELECT
7          id,
8          name,
9          CASE
10             WHEN LEFT(SUBSTR(phone_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone_number, 1,3), (LENGTH(SUBSTR(phone_number, 1,3))-1))
11             ELSE SUBSTR(phone_number, 1,3) END AS country_code
12      FROM
13          person
14      )
15
16
17  SELECT
18      name AS country
19  FROM (
20      SELECT
21          c.name,
22          SUM(ca.duration) AS call_duration,
23          COUNT(c.country_code) AS number_of_calls
24      FROM
25          temp_person
26      JOIN
27          country c
28      ON
29          temp_person.country_code = c.country_code
30      JOIN
31          calls ca
32      ON
33          temp_person.id = ca.caller_id
34      GROUP BY
35          c.name
36
37 UNION ALL
38
39      SELECT
40          c.name,
41          SUM(ca.duration) AS call duration,
42          COUNT(c.country_code) AS number_of_calls
43      FROM
44          temp_person
45      JOIN
46          country c
47      ON
48          temp_person.country_code = c.country_code
49      JOIN
50          calls ca
51      ON
52          temp_person.id = ca.callee_id
53      GROUP BY
54          c.name
55
56    )temp
57  GROUP BY
58      name
59  HAVING
60      SUM(call_duration)/SUM(number_of_calls) > (SELECT AVG(duration) FROM calls);
61
```

Question 56:

```
1
2 -- Q.56 Write an SQL Query to report the device that is first logged in for each player.
3 -- Return the result table in any order.
4
5
6 SELECT
7     player_id,
8     device_id
9 FROM
10    (
11        SELECT
12            player_id,
13            device_id,
14            event_date,
15            ROW_number() OVER(PARTITION BY player_id ORDER BY event_date) ranking
16        FROM
17            activity
18    ) temp_activity
19 WHERE
20     ranking = 1;
21
```

Question 57:

```
1
2 -- Q.57 Write an SQL Query to find the customer_number for the customer who has placed the largest number of orders.
3
4
5 WITH temp_orders AS (
6     SELECT
7         DISTINCT customer_number,
8         DENSE_RANK() OVER(ORDER BY total_orders DESC) AS ranking
9     FROM (
10        SELECT
11            customer_number,
12            COUNT(order_number) OVER(PARTITION BY customer_number) total_orders
13        FROM
14            orders
15    ) temp_cust_details
16 )
17
18 SELECT
19     customer_number
20 FROM
21     temp_orders
22 WHERE
23     ranking = 1;
24
```

Question 58:

```
1
2 -- Q.58 Write an SQL Query to report all the consecutive available seats in the cinema.
3 -- Return the result table ordered by seat_id in ascending order.
4
5
6 SELECT
7     DISTINCT c1.seat_id
8 FROM
9     cinema c1
10 INNER JOIN
11     cinema c2
12 ON
13     ABS(c1.seat_id - c2.seat_id) = 1
14     AND
15     (c1.free = 1 AND c2.free = 1)
16 ORDER BY
17     c1.seat_id;
18
```

Question 59:

```
1
2 -- Q.59 Write an SQL Query to report the names of all the salespersons who did not have any
3 -- orders related to the company with the name "RED".
4
5
6 SELECT
7     name
8 FROM
9     sales_person
10 WHERE
11     sales_id NOT IN (
12         SELECT
13             o.sales_id
14         FROM
15             orders o
16         INNER JOIN
17             company c
18         ON
19             c.company_id = o.company_id
20         WHERE
21             c.name = 'RED'
22     );
23
```

Question 60:

```
1
2 -- Q.60 Write an SQL Query to report for every three line segments whether they can form a triangle.
3 -- Return the result table in any order.
4
5
6 SELECT
7     x,
8     y,
9     z,
10    IF(x+y>z AND x+z>y AND y+z>x, 'YES','NO') AS is_triangle
11 FROM
12     triangle;
13
```

Question 61:

```
1
2 -- Q.61 Write an SQL Query to report the shortest distance between any two points from the Point table.
3
4
5 SELECT
6      MIN(ABS(c1.x - c2.x)) AS shortest_distance
7 FROM
8      point c1
9 INNER JOIN
10     point c2
11 WHERE
12     c1.x!=c2.x;
13
```

Question 62:

```
1
2 -- Q.62 Write a SQL Query for a report that provides the pairs (actor_id, director_id) where the actor has
3 -- cooperated with the director at least three times. Return the result table in any order.
4
5
6 WITH temp_actor_director AS (
7     SELECT
8         DISTINCT actor_id,
9         director_id,
10        DENSE_RANK() OVER(ORDER BY total_movies DESC) AS ranking
11    FROM (
12        SELECT
13            actor_id,
14            director_id,
15            COUNT(actor_id) OVER(PARTITION BY actor_id, director_id) AS total_movies
16        FROM
17            actor_director
18    ) temp
19)
20
21 SELECT
22     actor_id,
23     director_id
24 FROM
25     temp_actor_director
26 WHERE
27     ranking = 1;
28
```

Question 63:

```
1
2 -- Q.63 Write an SQL Query that reports the product_name, year, and price for each sale_id in
3 -- the sales table. Return the resulting table in any order.
4
5
6 SELECT
7     p.product_name,
8     s.year,
9     s.price
10    FROM
11        sales s
12    INNER JOIN
13        product p
14    ON
15        p.product_id = s.product_id;
16
```

Question 64:

```
1
2 -- Q.64 Write an SQL Query that reports the average experience years of all the employees for each project,
3 -- rounded to 2 digits. Return the result table in any order.
4
5
6 -- Approach 1
7
8
9 SELECT
10    p.project_id,
11    ROUND(AVG(experience_years), 2) AS average_years
12 FROM
13    employee e
14 INNER JOIN
15    project p
16 ON
17    p.employee_id = e.employee_id
18 GROUP BY
19    project_id;
20
21
22 -- Approach 2
23
24
25 SELECT
26    DISTINCT p.project_id,
27    ROUND(AVG(experience_years) OVER(PARTITION BY project_id), 2) AS average_years
28 FROM
29    employee e
30 INNER JOIN
31    project p
32 ON
33    p.employee_id = e.employee_id;
34
```

Question 65:

```
1
2 -- Q.65 Write an SQL Query that reports the best seller by total sales price, If there is a tie,
3 -- report them all. Return the result table in any order.
4
5
6 WITH temp_sales AS (
7     SELECT
8         seller_id,
9         total_price,
10        DENSE_RANK() OVER (ORDER BY total_price DESC) ranking
11    FROM
12    (
13        SELECT
14            s.seller_id,
15            SUM(s.quantity*p.unit_price) AS total_price
16        FROM
17            sales s
18        INNER JOIN
19            product p
20        ON
21            p.product_id = s.product_id
22        GROUP BY
23            seller_id
24    ) temp
25)
26
27
28 SELECT
29     seller_id
30 FROM
31     temp_sales
32 WHERE
33     ranking = 1;
34
```

Question 66:

```
1
2 -- Q.66 Write an SQL Query that reports the buyers who have bought S8 but not iphone. Note that S8 and iphone
3 -- are products present in the product table. Return the result table in any order.
4
5
6 SELECT
7     s.buyer_id
8 FROM
9     sales s
10 INNER JOIN
11     product p
12 ON
13     p.product_id = s.product_id
14 WHERE
15     p.product_name = 'S8'
16     AND
17     s.buyer_id NOT IN (
18             SELECT
19                 s.buyer_id
20             FROM
21                 sales S
22             INNER JOIN
23                 product P
24             ON
25                 s.product_id = p.product_id
26             WHERE
27                 p.product_name = 'Iphone'
28         );
29
```

Question 67:

```
1
2 -- Q.67 Write an SQL Query to compute the moving average of how much the customer paid in a seven days window
3 -- (i.e., current day + 6 days before). average_amount should be rounded to two decimal places.
4 -- Return result table ordered by visited_on in ascending order.
5
6
7 WITH temp_customer AS (
8     SELECT
9         visited_on,
10        SUM(amount) AS amount
11    FROM
12        customer
13   GROUP BY
14        visited_on
15    ),
16
17 temp_customer2 AS (
18     SELECT
19         visited_on,
20        SUM(amount) OVER(ORDER BY visited_on ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS weekly_amount,
21        ROUND(AVG(amount) OVER(ORDER BY visited_on ROWS BETWEEN 6 PRECEDING AND CURRENT ROW), 2) AS average_amount,
22        DENSE_RANK() OVER(ORDER BY visited_on) as ranking
23    FROM
24        temp_customer
25    )
26
27 SELECT
28     visited_on,
29     weekly_amount,
30     average_amount
31   FROM
32     temp_customer2
33 WHERE
34     ranking > 6;
35
```

Question 68:

```
1
2 -- Q.68 Write an SQL Query to find the total score for each gender on each day.
3 -- Return the result table ordered by gender and day in ascending order.
4
5
6 SELECT
7     gender,
8     day,
9     SUM(score_points) OVER(PARTITION BY gender ORDER BY day
10        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS total_points
11 FROM
12     scores
13 order by
14     gender,
15     day;
16
```

Question 69:

```
1
2 -- Q.69 Write an SQL Query to find the start and end number of continuous ranges in the table logs.
3 -- Return the result table ordered by start_id.
4
5
6 SELECT
7     MIN(log_id) AS start_id,
8     MAX(log_id) AS end_id
9 FROM
10    (
11        SELECT
12            log_id,
13            DENSE_RANK() OVER(ORDER BY log_id - RN) AS ranking
14        FROM
15        (
16            SELECT
17                log_id,
18                ROW_number() OVER(ORDER BY log_id) AS RN
19            FROM
20            logs
21        ) temp_log
22    ) temp_log2
23 GROUP BY
24     ranking
25 ORDER BY
26     start_id;
27
```

Question 70:

```
1
2 -- Q.70 Write an SQL Query to find the number of times each student attended each exam.
3 -- Return the result table ordered by student_id and subject_name.
4
5
6 WITH temp_student AS (
7     SELECT
8         student_id,
9         student_name,
10        subject_name
11    FROM
12        students,
13        subjects
14        ),
15
16     temp_student2 AS (
17        SELECT
18            student_id,
19            subject_name,
20            COUNT(*) AS times_attended_each_exam
21        FROM
22            exams
23        GROUP BY
24            student_id,
25            subject_name
26        )
27
28 SELECT
29     t.student_id,
30     t.student_name,
31     t.subject_name,
32     COALESCE(times_attended_each_exam,0) AS attended_exams
33 FROM
34     temp_student t
35 LEFT JOIN
36     temp_student2 t2
37 ON
38     t.student_id = t2.student_id
39     AND
40     t.subject_name = t2.subject_name
41 ORDER BY
42     t.student_id,
43     t.subject_name;
44
```

Question 71:

```
1
2 -- Q.71 Write an SQL Query to find employee_id of all employees that directly or indirectly
3 -- report their work to the head of the company. The indirect relation between managers will not exceed
4 -- three managers as the company is small. Return the result table in any order.
5
6
7 with recursive managers as (
8     SELECT
9         employee_id,
10        manager_id
11     FROM
12        employees
13    WHERE
14        employee_id = 1
15
16    UNION
17
18    SELECT
19        e.employee_id,
20        m.manager_id
21     FROM
22        managers m
23    INNER JOIN
24        employees e
25    ON
26        e.manager_id = m.employee_id
27    )
28
29 SELECT
30     employee_id
31 FROM
32     managers
33 WHERE
34     employee_id <> manager_id;
35
```

Question 72:

```
1
2 -- Q.72 Write an SQL Query to find for each month and country, the number of transactions and their total amount,
3 -- the number of approved transactions and their total amount. Return the result table in any order.
4
5
6 WITH temp_transactions AS (
7     SELECT
8         concat(YEAR(trans_date), '-',MONTH(trans_date)) AS transaction_date,
9         country,
10        state,
11        count(*) OVER (PARTITION BY concat(YEAR(trans_date), '-',MONTH(trans_date)), country) AS total_transactions,
12        sum(amount) OVER (PARTITION BY concat(YEAR(trans_date), '-',MONTH(trans_date)), country) AS total_transactions_amount,
13        sum(amount) OVER (PARTITION BY concat(YEAR(trans_date), '-',MONTH(trans_date)), country, state) AS amount
14
15    FROM
16        transactions
17
18 )
19
20 SELECT
21     transaction_date,
22     country,
23     total_transactions,
24     count(*) OVER(PARTITION BY transaction_date, country, state) AS approved_transactions,
25     total_transactions_amount,
26     amount AS approved_amount
27
28 FROM
29     temp_transactions
30 WHERE
31     state = 'Approved';
```

Question 73:

```
1
2 -- Q.73 Write an SQL Query to find the average daily percentage of posts that got
3 -- removed after being reported as spam, rounded to 2 decimal places.
4
5
6 WITH temp_action AS (
7     SELECT
8         action_date,
9         post_id,
10        COUNT(EXTRA) OVER(PARTITION BY action_date) num_post_reported_spam
11    FROM
12        actions
13    WHERE
14        extra = 'SPAM'
15    )
16
17 SELECT
18     ROUND(AVG(percentage), 2) AS avg_daily_percent
19 FROM
20    (
21        SELECT
22            action_date,
23            ROUND((COUNT(post_id)/num_post_reported_spam) * 100, 2) AS percentage
24        FROM
25            temp_action
26        WHERE
27            post_id IN (
28                SELECT
29                    post_id
30                FROM
31                    removals
32            )
33        GROUP BY
34            action_date
35    ) temp;
36
```

Question 74:

```
1
2 -- Q.74 Write an SQL query to report the fraction of players that logged in again on the day after the day
3 -- they first logged in, rounded to 2 decimal places. In other words, you need to count the number of players
4 -- that logged in for at least two consecutive days starting from their first login date,
5 -- then divide that number by the total number of players.
6
7
8 -- Approach 1
9
10
11 WITH temp_activity AS (
12     SELECT
13         player_id,
14         LEAD(event_date, 1) OVER(PARTITION BY player_id ORDER BY event_date) - event_date AS difference
15     FROM
16         activity
17     ),
18
19     temp_activity2 AS (
20     SELECT
21         COUNT(DISTINCT player_id) AS players_count
22     FROM
23         temp_activity
24     WHERE
25         difference = 1
26     GROUP BY
27         player_id
28     )
29
30 SELECT
31     ROUND(COUNT(*) / (SELECT COUNT(DISTINCT player_id) FROM activity), 2) AS fraction
32 FROM
33     temp_activity2;
34
35
36 -- Approach 2
37
38
39 WITH temp_activity AS (
40     SELECT
41         player_id,
42         event_date,
43         DATEDIFF(event_date,lag(event_date) OVER(PARTITION BY player_id ORDER BY event_date)) AS difference
44     FROM
45         activity
46     ),
47
48     temp_activity2 AS (
49     SELECT
50         COUNT(DISTINCT player_id) AS players_count
51     FROM
52         temp_activity
53     WHERE
54         difference = 1
55     GROUP BY
56         player_id
57     )
58
59
60 SELECT
61     ROUND(COUNT(*) / (SELECT COUNT(DISTINCT player_id) FROM activity), 2) AS fraction
62 FROM
63     temp_activity2;
64
```

Question 75:

```
1 -- Q.75 Write an SQL query to report the fraction of players that logged in again on the day after the day
2 -- they first logged in, rounded to 2 decimal places. In other words, you need to count the number of players
3 -- that logged in for at least two consecutive days starting from their first login date,
4 -- then divide that number by the total number of players.
5
6
7
8 -- Approach 1
9
10
11 WITH temp_activity AS (
12     SELECT
13         player_id,
14         LEAD(event_date, 1) OVER(PARTITION BY player_id ORDER BY event_date) - event_date AS difference
15     FROM
16         activity
17     ),
18     temp_activity2 AS (
19         SELECT
20             COUNT(DISTINCT player_id) AS players_count
21         FROM
22             temp_activity
23         WHERE
24             difference = 1
25         GROUP BY
26             player_id
27     )
28 )
29
30 SELECT
31     ROUND(COUNT(*) / (SELECT COUNT(DISTINCT player_id) FROM activity), 2) AS fraction
32 FROM
33     temp_activity2;
34
35
36 -- Approach 2
37
38
39 WITH temp_activity AS (
40     SELECT
41         player_id,
42         event_date,
43         DATEDIFF(event_date, lag(event_date) OVER(PARTITION BY player_id ORDER BY event_date)) AS difference
44     FROM
45         activity
46     ),
47     temp_activity2 AS (
48         SELECT
49             COUNT(DISTINCT player_id) AS players_count
50         FROM
51             temp_activity
52         WHERE
53             difference = 1
54         GROUP BY
55             player_id
56     )
57
58
59
60 SELECT
61     ROUND(COUNT(*) / (SELECT COUNT(DISTINCT player_id) FROM activity), 2) AS fraction
62 FROM
63     temp_activity2;
64
```

Question 76:

```
1
2  -- Q.76 Write an SQL Query to find the salaries of the employees after applying taxes.
3  -- Round the salary to the nearest integer.
4
5
6 WITH temp_salaries AS (
7         SELECT
8             company_id,
9             employee_id,
10            employee_name,
11            salary,
12            MAX(salary) OVER(PARTITION BY company_id) max_sal_per_company
13        FROM
14            salaries
15    )
16
17 SELECT
18     company_id,
19     employee_id,
20     employee_name,
21     salary,
22     ROUND(
23         CASE
24             WHEN max_sal_per_company > 10000 THEN salary - (salary * 0.49)
25             WHEN max_sal_per_company BETWEEN 1000 AND 10000 THEN salary - (salary * 0.24)
26             ELSE salary
27         END, 0) AS sal_after_tax_deduction
28 FROM
29     temp_salaries;
30
```

Question 77:

```
1
2 -- Q.77 Write an SQL Query to evaluate the boolean expressions in Expressions table.
3 -- Return the result table in any order.
4
5
6 SELECT
7     e.left_operand,
8     e.operator,
9     e.right_operand,
10    CASE
11        WHEN e.operator = '<' THEN IF(l.value < r.value, 'TRUE', 'FALSE')
12        WHEN e.operator = '>' THEN IF(l.value > r.value, 'TRUE', 'FALSE')
13        ELSE IF(l.value = r.value, 'TRUE', 'FALSE')
14    END AS result
15 FROM
16     expressions e
17 JOIN
18     variables l
19 ON
20     e.left_operand = l.name
21 JOIN
22     variables r
23 ON
24     e.right_operand = r.name;
25
```

Question 78:

```
1
2 -- Q.78 Write an SQL Query to find the countries where this company can invest .
3 -- Return the result table in any order .
4
5
6 -- Approach 1
7
8
9 SELECT          name AS country
10    FROM (
11      SELECT          c.name,
12                      SUM(ca.duration) AS call_duration,
13                      COUNT(c.country_code) AS number_of_calls
14      FROM (
15        SELECT          id,
16                      name,
17                      CASE
18                        WHEN LEFT(SUBSTR(phone_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone_number, 1,3), (LENGTH(SUBSTR(phone_number, 1,3))-1))
19                        ELSE SUBSTR(phone_number, 1,3) END AS country_code
20                    FROM person
21      ) temp_person
22    JOIN country c
23    ON temp_person.country_code = c.country_code
24  JOIN calls ca
25  ON temp_person.id = caller_id
26 GROUP BY
27          c.name
28
29 UNION ALL
30
31
32
33
34
35
36
37
38
39
40
41
42 SELECT          c.name,
43                      SUM(ca.duration) AS call_duration,
44                      COUNT(c.country_code) AS number_of_calls
45  FROM (
46    SELECT          id,
47                      name,
48                      CASE
49                        WHEN LEFT(SUBSTR(phone_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone_number, 1,3), (length(SUBSTR(phone_number, 1,3))-1))
50                        ELSE SUBSTR(phone_number, 1,3) END AS country_code
51                    FROM person
52      ) temp_person
53    JOIN country c
54    ON temp_person.country_code = c.country_code
55  JOIN calls ca
56  ON temp_person.id = ca.callee_id
57 GROUP BY
58          c.name
59
60
61
62
63
64
65
66
67      ) temp
68
69 GROUP BY
70          name
71 HAVING
72          SUM(call_duration)/SUM(number_of_calls) >  (SELECT AVG(duration) FROM calls);
73
```

```
1  ● ● ●
2  -- Approach 2
3
4
5  WITH temp_person AS (
6      SELECT
7          id,
8          name,
9          CASE
10             WHEN LEFT(SUBSTR(phone_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone_number, 1,3), (LENGTH(SUBSTR(phone_number, 1,3))-1))
11             ELSE SUBSTR(phone_number, 1,3) END AS country_code
12      FROM
13          person
14      )
15
16
17  SELECT
18      name AS country
19  FROM (
20      SELECT
21          c.name,
22          SUM(ca.duration) AS call_duration,
23          COUNT(c.country_code) AS number_of_calls
24      FROM
25          temp_person
26      JOIN
27          country c
28      ON
29          temp_person.country_code = c.country_code
30      JOIN
31          calls ca
32      ON
33          temp_person.id = ca.caller_id
34      GROUP BY
35          c.name
36
37 UNION ALL
38
39      SELECT
40          c.name,
41          SUM(ca.duration) AS call duration,
42          COUNT(c.country_code) AS number_of_calls
43      FROM
44          temp_person
45      JOIN
46          country c
47      ON
48          temp_person.country_code = c.country_code
49      JOIN
50          calls ca
51      ON
52          temp_person.id = ca.callee_id
53      GROUP BY
54          c.name
55
56    )temp
57  GROUP BY
58      name
59  HAVING
60      SUM(call_duration)/SUM(number_of_calls) > (SELECT AVG(duration) FROM calls);
61
```

Question 79:

```
1
2 -- Q.79 Write a Query that prints a list of employee names (i.e.: the name attribute)
3 -- from the employee table in alphabetical order.
4
5
6 SELECT
7      name
8 FROM
9      employee
10 ORDER BY
11      name;
12
```

Question 80:

```
1  -- Q.80 Write a Query to obtain the year-on-year growth rate for the total spend of each product for each year.
2
3
4
5  -- Approach 1
6
7
8  WITH temp_transactions AS (
9      SELECT
10         product_id,
11         transaction_date,
12         spend AS curr_year_spend,
13         LAG(spend,1,0) OVER w AS prev_year_spend,
14         IFNULL(spend - LAG(spend,1) OVER w, 0) AS prev_curr_spend_diff
15     FROM
16         user_transactions
17     WINDOW
18         w AS (PARTITION BY product_id ORDER BY EXTRACT(YEAR FROM transaction_date))
19     )
20
21  SELECT
22      product_id,
23      curr_year_spend,
24      ROUND(prev_year_spend, 2),
25      IFNULL(ROUND((prev_curr_spend_diff * 100)/prev_year_spend,2),0) AS YOY
26  FROM
27      temp_transactions;
28
29
30
31  -- Approach 2
32
33
34  WITH temp_transactions AS (
35      SELECT
36          YEAR(STR_TO_DATE(transaction_date, '%m/%d/%Y')) AS YEAR_id,
37          product_id,
38          transaction_date,
39          spend AS curr_year_spend,
40          LAG(spend,1) OVER w AS prev_year_spend,
41          spend - LAG(spend,1) OVER w AS prev_curr_spend_diff
42      FROM
43          user_transactions
44      WINDOW
45          w AS (PARTITION BY product_id ORDER BY EXTRACT(YEAR FROM transaction_date))
46      )
47
48  SELECT
49      year_id,
50      product_id,
51      curr_year_spend,
52      ROUND(prev_year_spend, 2),
53      ROUND((prev_curr_spend_diff * 100)/prev_year_spend,2) AS YOY
54  FROM
55      temp_transactions;
56
57
58  -- Approach 3
59
60
61  SELECT
62      product_id,
63      YEAR(STR_TO_DATE(transaction_date, '%m/%d/%Y')) AS YEAR_id,
64      spend AS curr_year_spend,
65      ROUND(LAG(spend,1,0) OVER w ,2) AS prev_year_spend,
66      ROUND((spend - LAG(spend,1) OVER w ) * 100 /
67              LAG(spend,1) OVER w, 2) AS YOY
68  FROM
69      user_transactions
70  WINDOW
71      w AS (PARTITION BY product_id ORDER BY EXTRACT(YEAR FROM transaction_date));
72
```

Question 81:

```
1
2 -- Q.81 Write a SQL Query to find the number of prime and non-prime items that can be stored
3 -- in the 500,000 square feet warehouse. Output the item type and number of items to be stocked.
4
5
6 -- Approach 1
7
8
9 WITH temp_inventory AS (
10     SELECT
11         item_type,
12         SUM(square_foot) AS square_foot_per_category,
13         COUNT(*) AS count_of_items
14     FROM
15         inventory
16     GROUP BY
17         item_type
18 ),
19
20 temp_inventory2 AS (
21     SELECT
22         (500000 - SUM(square_foot_per_category)*FLOOR(500000/SUM(square_foot_per_category))) AS area_left
23     FROM
24         temp_inventory
25     WHERE
26         item_type = 'PRIME_ELIGIBLE'
27 ),
28
29 temp_inventory3 AS (
30     SELECT
31         item_type,
32         CASE
33             WHEN item_type = 'PRIME_ELIGIBLE'
34                 THEN FLOOR(500000/square_foot_per_category) * count_of_items
35             WHEN item_type = 'NOT_PRIME'
36                 THEN FLOOR((SELECT area_left FROM temp_inventory2) / square_foot_per_category) * count_of_items
37         END AS item_count
38     FROM
39         temp_inventory
40     )
41
42 SELECT
43     item_type,
44     item_count
45 FROM
46     temp_inventory3;
```

```
1
2 -- Approach 2
3
4
5 WITH temp_inventory AS (
6     SELECT
7         item_type,
8         SUM(square_foot) AS square_foot_per_category,
9         COUNT(*) AS count_of_items,
10        CASE
11            WHEN item_type = 'PRIME_ELIGIBLE' THEN FLOOR(500000/SUM(square_foot)) * COUNT(*)
12            END AS prime_items_count
13    FROM
14        inventory
15    GROUP BY
16        item_type
17    ),
18
19 temp_inventory2 AS (
20     SELECT
21         (50000 - SUM(square_foot_per_category)*FLOOR(500000/SUM(square_foot_per_category))) AS area_left
22     FROM
23        temp_inventory
24    WHERE
25        item_type = 'PRIME_ELIGIBLE'
26    ),
27
28 temp_inventory3 AS (
29     SELECT
30         item_type,
31         CASE
32             WHEN item_type = 'PRIME_ELIGIBLE'
33                 THEN prime_items_count
34             WHEN item_type = 'NOT_PRIME'
35                 THEN FLOOR((SELECT area_left FROM temp_inventory2) / square_foot_per_category) * count_of_items
36         END AS item_count
37     FROM
38        temp_inventory
39    )
40
41 SELECT
42     item_type,
43     item_count
44 FROM
45     temp_inventory3;
46
47
```

Question 82:

```
1
2 -- Q.82 Write a Query to obtain the active user retention in July 2022.
3 -- Output the month (in numerical format 1, 2, 3) and the number of monthly active users (MAUs).
4
5
6 -- Approach 1
7
8
9 WITH temp_actions AS (
10     SELECT
11         user_id,
12         event_date,
13         event_type,
14         SUBSTR(event_date, 6, 2) - lag(SUBSTR(event_date, 6, 2)) OVER w AS difference
15     FROM
16         user_actions
17     WINDOW
18         w AS (PARTITION BY user_id ORDER BY event_date)
19     ),
20
21     temp_actions2 AS (
22         SELECT
23             SUBSTR(event_date, 6, 2) AS months,
24             COUNT(user_id) AS monthly_active_users
25         FROM
26             temp_actions
27         WHERE
28             difference = 1 AND event_type IN ('LIKE', 'COMMENT', 'SIGN-IN')
29         GROUP BY
30             months
31         )
32
33 SELECT
34     months,
35     monthly_active_users
36 FROM
37     temp_actions2;
38
39
40 -- Approach 2
41
42
43 WITH temp_actions AS (
44     SELECT
45         user_id,
46         event_date,
47         event_type,
48         SUBSTR(event_date, 6, 2) - lag(SUBSTR(event_date, 6, 2)) OVER w AS difference
49     FROM
50         user_actions
51     WINDOW
52         w AS (PARTITION BY user_id ORDER BY event_date)
53     )
54
55
56 SELECT
57     SUBSTR(event_date, 6, 2) AS months,
58     COUNT(DISTINCT user_id) AS active_users
59 FROM
60     temp_actions
61 WHERE
62     difference = 1
63     AND
64     event_type IN ('LIKE', 'COMMENT', 'SIGN-IN')
65 GROUP BY
66     months;
```

Question 83:

```
1
2 -- Q.83 Write a Query to report the median of searches made by a user.
3 -- Round the median to one decimal point.
4
5
6 WITH temp_search_freq AS (
7     SELECT
8         searches,
9         num_users,
10        ROW_NUMBER() OVER(ORDER BY searches) row_num,
11        COUNT(*) OVER(ORDER BY searches ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) total_records
12    FROM
13    search_frequency
14),
15 temp_search_freq2 AS (
16     SELECT
17         searches,
18         num_users,
19         CASE
20             WHEN total_records % 2 <> 0 THEN (
21                 SELECT
22                     DISTINCT ROUND(SUM(searches) OVER w /
23                     COUNT(*) OVER w,1)
24                 FROM
25                 temp_search_freq
26                 WHERE
27                     row_num = ROUND((total_records + 1) / 2, 0)
28                     WINDOW
29                         w AS (ORDER BY searches ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
30             )
31
32             WHEN total_records % 2 = 0 THEN (
33                 SELECT
34                     DISTINCT ROUND(SUM(searches) OVER w /
35                     COUNT(*) OVER w,1)
36                 FROM
37                 temp_search_freq
38                 WHERE
39                     row_num IN (total_records/2,(total_records/2)+1)
40                     WINDOW
41                         w AS (ORDER BY searches ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
42             )
43         END AS median
44     FROM
45     temp_search_freq
46 )
47
48 )
49
50
51 SELECT
52     DISTINCT median
53 FROM
54     temp_search_freq2;
55
```

Question 84:

```
1
2 -- Q.84 Write a Query to update the Facebook advertisers status using the daily_pay table.
3 -- Advertiser is a two-column table containing the user id and their payment status based
4 -- on the last payment and daily_pay table has current information about their payment.
5 -- Only advertisers who paid will show up in this table.
6 -- Output the user id and current payment status sorted by the user id.
7
8
9 SELECT
10      user_id,
11      CASE
12          WHEN user_id IN (SELECT user_id FROM daily_pay) THEN 'EXISTING'
13          ELSE 'CHURN'
14      END AS new_status
15 FROM
16      advertiser
17 ORDER BY
18      user_id;
19
```

Question 85:

```
1
2 -- Q.85 Write a SQL Query that calculates the total time that the fleet of
3 -- servers was running. The output should be in units of full days.
4
5
6 -- Approach 1
7
8
9 SELECT      stop_time - start_time AS total_up_time
10 FROM        (
11          SELECT
12            SUM(
13              CASE
14                WHEN session_status = 'start' THEN EXTRACT(DAY from STR_TO_DATE(status_time, '%m/%d/%y'))
15                END
16              ) AS start_time,
17            SUM(
18              CASE
19                WHEN session_status = 'stop' THEN EXTRACT(DAY from STR_TO_DATE(status_time, '%m/%d/%y'))
20                END
21              ) AS stop_time
22          FROM      server_utilization
23        ) temp_server_utilization;
24
25
26
27
28
29 -- Approach 2 (Works only in PostgreSQL)
30
31
32 WITH temp_server_utilization AS (
33   SELECT
34     server_id,
35     status_time AS start_time,
36     session_status,
37     lead(status_time) OVER(ORDER BY server_id,status_time) AS end_time
38   FROM      server_utilization
39
40 )
41
42 SELECT      EXTRACT(DAY FROM justify_hours(SUM(end_time - start_time))) as total_time
43 FROM        temp_server_utilization
44 WHERE       session_status = 'start';
45
46
47
48
```

Question 86:

```
1 -- Q.86 Sometimes, payment transactions are repeated by accident; it could be due to user error,
2 -- API failure or a retry error that causes a credit card to be charged twice.
3 -- Using the transactions table, identify any payments made at the same merchant with the
4 -- same credit card for the same amount within 10 minutes of each other. Count such repeated payments.
5
6
7
8 CREATE TABLE transactions(
9     transaction_id INT,
10    merchant_id INT,
11    credit_card_id INT,
12    amount INT,
13    transaction_timestamp DATETIME
14 );
15
16
17 INSERT INTO transactions VALUES
18     (1,101,1,100,'2022-09-25 12:00:00'),
19     (2,101,1,100,'2022-09-25 12:08:00'),
20     (3,101,1,100,'2022-09-25 12:28:00'),
21     (4,102,2,300,'2022-09-25 12:00:00'),
22     (5,102,2,400,'2022-09-25 14:00:00');
23
24 -- Q.86 Sometimes, payment transactions are repeated by accident; it could be due to user error,
25 -- API failure or a retry error that causes a credit card to be charged twice.
26 -- Using the transactions table, identify any payments made at the same merchant with the
27 -- same credit card for the same amount within 10 minutes of each other. Count such repeated payments.
28
29
30 WITH temp_transactions AS (
31     SELECT
32         merchant_id,
33         credit_card_id,
34         amount,
35         transaction_timestamp,
36         LAG(transaction_timestamp) OVER w AS prev_tran_timestamp,
37         timestampdiff(MINUTE,LAG(transaction_timestamp) OVER w, transaction_timestamp) AS difference
38     FROM
39         transactions
40     WINDOW
41         w AS (PARTITION BY credit_card_id ORDER BY transaction_timestamp)
42 )
43
44 SELECT
45     COUNT(DISTINCT merchant_id) AS payment_count
46 FROM
47     temp_transactions
48 WHERE
49     difference <= 10;
50
51
52 -- Approach 2 (Works only in PostgreSQL)
53
54
55 WITH temp_transactions AS (
56     SELECT
57         merchant_id,
58         credit_card_id,
59         amount,
60         transaction_timestamp,
61         LAG(transaction_timestamp) OVER w AS prev_tran_timestamp,
62         EXTRACT(EPOCH FROM LAG(transaction_timestamp) OVER w - transaction_timestamp) AS difference
63     FROM
64         transactions
65     WINDOW
66         w AS (PARTITION BY credit_card_id ORDER BY transaction_timestamp)
67 )
68
69 SELECT
70     COUNT(DISTINCT merchant_id) AS payment_count
71 FROM
72     temp_transactions
73 WHERE
74     difference < 10;
75
```

Question 87:

```
1
2 -- Q.87 Write a SQL Query to find the bad experience rate in the first 14 days for new users who signed
3 -- up in June 2022. Output the percentage of bad experience rounded to 2 decimal places.
4
5
6 SELECT
7     ROUND(
8         SUM(
9             CASE
10                 WHEN status !='completed successfully' THEN 1 ELSE 0
11             END
12         )*100.0/count(*),2) AS bad_experience_pct
13 FROM
14     customers c
15 INNER JOIN
16     orders o
17 ON
18     o.customer_id = c.customer_id
19 WHERE
20     o.order_timestamp < date_add(STR_TO_date(signup_timestamp, '%m/%d/%Y'), INTERVAL 14 DAY)
21     AND
22     MONTH(STR_TO_date(signup_timestamp, '%m/%d/%Y')) = 06
23     AND
24     YEAR(STR_TO_date(signup_timestamp, '%m/%d/%Y')) = 2022;
25
```

Question 88:

```
1
2 -- Q.88 Write an SQL Query to find the total score for each gender on each day.
3 -- Return the result table ordered by gender and day in ascending order.
4
5
6 SELECT
7     gender,
8     day,
9     SUM(score_points) OVER(PARTITION BY gender ORDER BY day
10        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS total_points
11 FROM
12     scores
13 order by
14     gender,
15     day;
16
```

Question 89:

```
1
2 -- Q.89 Write an SQL Query to find the countries where this company can invest .
3 -- Return the result table in any order .
4
5
6 -- Approach 1
7
8
9 SELECT          name AS country
10    FROM (
11      SELECT
12        c.name,
13        SUM(ca.duration) AS call_duration,
14        COUNT(c.country_code) AS number_of_calls
15      FROM (
16        SELECT
17          id,
18          name,
19          CASE
20            WHEN LEFT(SUBSTR(phone_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone_number, 1,3), (LENGTH(SUBSTR(phone_number, 1,3))-1))
21            ELSE SUBSTR(phone_number, 1,3) END AS country_code
22          FROM person
23        ) temp_person
24      JOIN country c
25      ON temp_person.country_code = c.country_code
26    JOIN calls ca
27    ON temp_person.id = caller_id
28  GROUP BY
29    c.name
30
31 UNION ALL
32
33
34 SELECT          c.name,
35        SUM(ca.duration) AS call_duration,
36        COUNT(c.country_code) AS number_of_calls
37      FROM (
38        SELECT
39          id,
40          name,
41          CASE
42            WHEN LEFT(SUBSTR(phone_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone_number, 1,3), (length(SUBSTR(phone_number, 1,3))-1))
43            ELSE SUBSTR(phone_number, 1,3) END AS country_code
44          FROM person
45        ) temp_person
46      JOIN country c
47      ON temp_person.country_code = c.country_code
48    JOIN calls ca
49    ON temp_person.id = ca.callee_id
50  GROUP BY
51    c.name
52
53 ) temp
54
55 GROUP BY
56   name
57 HAVING SUM(call_duration)/SUM(number_of_calls) > (SELECT AVG(duration) FROM calls);
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
```

```
1  -- Approach 2
2
3
4
5  WITH temp_person AS (
6      SELECT
7          id,
8          name,
9          CASE
10             WHEN LEFT(SUBSTR(phone_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone_number, 1,3), (LENGTH(SUBSTR(phone_number, 1,3))-1))
11             ELSE SUBSTR(phone_number, 1,3) END AS country_code
12      FROM
13          person
14      )
15
16
17  SELECT
18      name AS country
19  FROM
20      (
21          SELECT
22              c.name,
23              SUM(ca.duration) AS call_duration,
24              COUNT(c.country_code) AS number_of_calls
25          FROM
26              temp_person
27          JOIN
28              country c
29          ON
30              temp_person.country_code = c.country_code
31          JOIN
32              calls ca
33          ON
34              temp_person.id = ca.caller_id
35          GROUP BY
36              c.name
37
38  UNION ALL
39
40          SELECT
41              c.name,
42              SUM(ca.duration) AS call_duration,
43              COUNT(c.country_code) AS number_of_calls
44          FROM
45              temp_person
46          JOIN
47              country c
48          ON
49              temp_person.country_code = c.country_code
50          JOIN
51              calls ca
52          ON
53              temp_person.id = ca.callee_id
54          GROUP BY
55              c.name
56
57      )temp
58  GROUP BY
59      name
59 HAVING
60      SUM(call_duration)/SUM(number_of_calls) > (SELECT AVG(duration) FROM calls);
61
```

Question 90:

```
1
2 -- Q.90 Write an SQL Query to report the median of all the numbers in the database
3 -- after decompressing the numbers table. Round the median to one decimal point.
4
5
6 WITH RECURSIVE num_frequency (num,frequency, i) AS
7     (
8         SELECT
9             num,
10            frequency,1
11        FROM
12          numbers
13
14        UNION ALL
15
16        SELECT
17            num,
18            frequency,
19            i+1
20        FROM
21          num_frequency
22        WHERE
23            num_frequency.i < num_frequency.frequency
24    ),
25
26 num_frequency2 AS (
27     SELECT
28         num,
29         frequency,
30         row_number() OVER(ORDER BY num, frequency) AS row_num,
31         COUNT(*) OVER(ORDER BY num, frequency ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS total_records
32     FROM
33       num_frequency
34   )
35
36 SELECT
37   DISTINCT CASE
38     WHEN total_records % 2 <> 0 THEN (
39       SELECT
40           DISTINCT ROUND(SUM(num) OVER w /
41           COUNT(*) OVER w, 1)
42       FROM
43         num_frequency2
44       WHERE
45         row_num = ROUND((total_records + 1) / 2, 0)
46       WINDOW
47         w AS (ORDER BY num, frequency ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING))
48
49     WHEN total_records % 2 = 0 THEN (
50       SELECT
51           DISTINCT ROUND(SUM(num) OVER w /
52           COUNT(*) OVER w, 1)
53       FROM
54         num_frequency2
55       WHERE
56         row_num IN (total_records/2,(total_records/2)+1)
57       WINDOW
58         w AS (ORDER BY num, frequency ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING))
59
60     END AS median
61   FROM
62     num_frequency2;
```

Question 91:

```
1
2 -- Q.91 Write an SQL Query to report the comparison result (higher/lower/same) of the average salary of
3 -- employees in a department to the companys average salary. Return the result table in any order.
4
5
6 WITH temp_comparison AS (
7     SELECT
8         s.employee_id,
9             e.department_id,
10            s.amount,
11            s.paydate,
12            avg(amount) OVER (PARTITION BY MONTH(paydate) ORDER BY month(paydate), employee_id
13                           ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) company_avg_salary,
14            avg(amount) OVER (PARTITION BY MONTH(paydate), department_id ORDER BY month(paydate)
15                           ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) department_avg
16        FROM
17            salary s
18        INNER JOIN
19            employee e
20        ON
21            e.employee_id = s.employee_id
22    )
23
24 SELECT
25     DISTINCT DATE_FORMAT(paydate, '%Y-%m') AS pay_month,
26     department_id,
27     CASE
28         WHEN company_avg_salary = department_avg THEN 'same'
29         WHEN company_avg_salary > department_avg THEN 'lower'
30         WHEN company_avg_salary < department_avg THEN 'higher'
31     END AS comparison
32 FROM
33     temp_comparison;
34
```

Question 92:

```
1
2 -- Q.92 Write an SQL Query to report for each install date, the number of players
3 -- that installed the game on that day, and the day one retention.
4
5
6 -- Approach 1
7
8
9 SELECT
10      a.event_date AS install_date,
11      COUNT(a.player_id) AS installs,
12      ROUND(COUNT(b.player_id) / COUNT(a.player_id), 2) AS day1_retention
13 FROM
14      (
15          SELECT
16              player_id,
17              MIN(event_date) AS event_date
18          FROM
19              activity
20          GROUP BY
21              player_id
22      ) a
23 LEFT JOIN
24      activity b
25 ON
26      a.player_id = b.player_id
27      AND
28      a.event_date + 1 = b.event_date
29 GROUP BY
30      a.event_date;
31
32
33 -- Approach 2
34
35
36 SELECT
37      a1.event_date AS install_dt,
38      COUNT(a1.player_id) AS installs,
39      ROUND(COUNT(a3.player_id) / COUNT(a1.player_id), 2) AS day1_retention
40 FROM
41      activity a1
42 LEFT JOIN
43      activity a2
44 ON
45      a1.player_id = a2.player_id
46      AND
47      a1.event_date > a2.event_date
48 LEFT JOIN
49      activity a3
50 ON
51      a1.player_id = a3.player_id
52      AND
53      DATEDIFF(a3.event_date, a1.event_date) = 1
54 WHERE
55      a2.event_date IS NULL
56 GROUP BY
57      a1.event_date;
58
```

Question 93:

```
1
2 -- Q.93 Write an SQL query to find the winner in each group. Return the result table in any order.
3
4
5 SELECT
6     group_id,
7     players AS player_id
8 FROM
9     (
10         SELECT
11             p.group_id,
12             CASE
13                 WHEN first_player_goals > second_player_goals THEN first_player
14                 WHEN first_player_goals < second_player_goals THEN second_player
15                 WHEN first_player_goals = second_player_goals THEN IF(first_player < second_player, first_player, second_player)
16             END AS players,
17             MAX(IF(first_player_goals > second_player_goals, first_player_goals, second_player_goals)) AS goals,
18             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
19         FROM
20             players p
21         INNER JOIN
22             matches m
23         ON
24             m.first_player = p.player_id
25             OR
26             m.second_player = p.player_id
27         GROUP BY
28             p.group_id,
29             players
30     ) temp_matches
31 WHERE
32     ranking = 1;
```

Question 94:

```
1
2 -- Q.94 Write an SQL Query to report the students (student_id, student_name) being -- Quiet in all exams.
3 -- Do not return the student who has never taken any exam.
4
5
6 -- Approach 1
7
8
9 WITH temp_examination AS (
10     SELECT
11         exam_id,
12         student_id,
13         score,
14         max(score) OVER w AS highest,
15         min(score) OVER w AS lowest
16     FROM
17         exam
18     WINDOW
19         w AS (PARTITION BY exam_id)
20     ),
21
22 temp_examination1 AS (
23     SELECT
24         DISTINCT student_id
25     FROM
26         temp_examination
27     WHERE
28         score IN (lowest, highest)
29     )
30
31     SELECT
32         DISTINCT s.student_id,
33         s.student_name
34     FROM
35         temp_examination
36     INNER JOIN
37         student s
38     ON
39         s.student_id = temp_examination.student_id
40     WHERE
41         s.student_id NOT IN (SELECT student_id FROM temp_examination1);
42
43
44 -- Approach 2
45
46
47 WITH temp_examination AS (
48     SELECT
49         student_id,
50         CASE
51             WHEN score < max(score) OVER(PARTITION BY exam_id) AND score > min(score) OVER(PARTITION BY exam_id) THEN 0
52             ELSE 1
53         END AS category
54     FROM
55         exam
56     ORDER BY
57         student_id
58     ),
59
60 temp_examination1 AS (
61     SELECT
62         student_id,
63         SUM(category) AS high_low_count
64     FROM
65         temp_examination
66     GROUP BY
67         student_id
68     )
69
70 SELECT
71     s.student_id,
72     s.student_name
73 FROM
74     student s
75 INNER JOIN
76     temp_examination1
77 ON
78     s.student_id = temp_examination1.student_id
79 WHERE
80     high_low_count = 0
81 ORDER BY
82     temp_examination1.student_id;
```

Question 95:

```
1
2 -- Q.95 Write an SQL Query to report the students (student_id, student_name) being -- Quiet in all exams.
3 -- Do not return the student who has never taken any exam.
4
5
6 -- Approach 1
7
8
9 WITH temp_examination AS (
10     SELECT
11         exam_id,
12         student_id,
13         score,
14         max(score) OVER w AS highest,
15         min(score) OVER w AS lowest
16     FROM
17         exam
18     WINDOW
19         w AS (PARTITION BY exam_id)
20     ),
21     temp_examination1 AS (
22         SELECT
23             DISTINCT student_id
24         FROM
25             temp_examination
26         WHERE
27             score IN (lowest, highest)
28         )
29
30     SELECT
31         DISTINCT s.student_id,
32         s.student_name
33     FROM
34         temp_examination
35     INNER JOIN
36         student s
37     ON
38         s.student_id = temp_examination.student_id
39     WHERE
40         s.student_id NOT IN (SELECT student_id FROM temp_examination1);
41
42
43
44 -- Approach 2
45
46
47 WITH temp_examination AS (
48     SELECT
49         student_id,
50         CASE
51             WHEN score < max(score) OVER(PARTITION BY exam_id) AND score > min(score) OVER(PARTITION BY exam_id) THEN 0
52             ELSE 1
53         END AS category
54     FROM
55         exam
56     ORDER BY
57         student_id
58     ),
59     temp_examination1 AS (
60         SELECT
61             student_id,
62             SUM(category) AS high_low_count
63         FROM
64             temp_examination
65         GROUP BY
66             student_id
67         )
68
69     SELECT
70         s.student_id,
71         s.student_name
72     FROM
73         student s
74     INNER JOIN
75         temp_examination1
76     ON
77         s.student_id = temp_examination1.student_id
78     WHERE
79         high_low_count = 0
80     ORDER BY
81         temp_examination1.student_id;
82
```

Question 96:

```
1
2 -- Q.96 Write a query to output the user id, song id, and cumulative count of song plays as of 4 August 2022
3 -- sorted in descending order.
4
5 WITH streaming AS (
6     SELECT
7         user_id,
8         song_id,
9         song_plays
10    FROM
11        songs_history
12
13    UNION ALL
14
15    SELECT
16        user_id,
17        song_id,
18        count(*) AS song_plays
19    FROM
20        songs_weekly
21    WHERE
22        listen_time <= '08/04/2022 23:59:59'
23    GROUP by
24        user_id,
25        song_id
26    )
27
28 SELECT
29     user_id,
30     song_id,
31     SUM(song_plays) as song_plays
32 FROM
33     streaming
34 GROUP BY
35     user_id,
36     song_id
37 ORDER BY
38     song_plays DESC;
39
```

Question 97:

```
1
2 -- Q.97 Write a query to find the confirmation rate of users who confirmed their signups with text messages.
3 -- Round the result to 2 decimal places.
4
5
6 WITH temp_confirmation AS (
7     SELECT
8         e.email_id,
9         CASE
10            WHEN signup_action = 'Confirmed' THEN 1
11            END
12            AS confirmed_users
13     FROM
14         emails e
15     LEFT JOIN
16         texts t
17     ON
18         e.email_id = t.email_id
19         AND
20         t.signup_action = 'Confirmed'
21    )
22
23
24 SELECT
25     ROUND(SUM(confirmed_users)/COUNT(email_id),2) AS confirm_rate
26 FROM
27     temp_confirmation;
28
```

Question 98:

```
1
2 -- Q.98 Calculate the 3-day rolling average of tweets published by each user for each date
3 -- that a tweet was posted. Output the user id, tweet date, and rolling averages rounded to 2 decimal places.
4
5
6 WITH temp_tweets AS (
7     SELECT
8         user_id,
9         tweet_date,
10        COUNT(tweet_id) AS tweets_count
11    FROM
12        tweets
13   GROUP BY
14        user_id,
15        tweet_date
16   ORDER BY
17        user_id,
18        tweet_date
19 )
20
21 SELECT
22     user_id,
23     tweet_date,
24     ROUND(avg(tweets_count)
25     OVER(PARTITION BY user_id ORDER BY tweet_date
26     ROWS BETWEEN 2 PRECEDING AND CURRENT ROW),2) AS rolling_avg_3days
27 FROM
28     temp_tweets;
29
```

Question 99:

```
1
2 -- Q.99 Write a query to obtain a breakdown of the time spent sending vs. opening snaps
3 -- (as a percentage of total time spent on these activities) for each age group.
4
5
6 WITH temp_activities AS (
7     SELECT
8         user_id,
9         activity_type,
10        sum(time_spent) time_spent,
11        CASE
12            WHEN activity_type = 'open' THEN sum(time_spent)
13            ELSE 0
14        END opening_snap,
15        CASE
16            WHEN activity_type = 'send' THEN sum(time_spent)
17            ELSE 0
18        END sending_snap
19     FROM
20         activities
21     WHERE
22         activity_type in ('open','send')
23     GROUP BY
24         user_id,
25         activity_type
26     ORDER BY
27         user_id
28     ),
29
30 temp_activities2 AS (
31     SELECT
32         user_id,
33         SUM(opening_snap) time_sending,
34         SUM(sending_snap) time_opening
35     FROM
36         temp_activities
37     GROUP BY
38         user_id
39     )
40
41 SELECT
42     ab.age_bucket,
43     ROUND(time_opening * 100.0 / (time_sending+time_opening), 2) AS send_perc,
44     ROUND(time_sending * 100.0 / (time_sending+time_opening), 2) AS open_perc
45 FROM
46     temp_activities2
47 INNER JOIN
48     age_breakdown ab
49 ON
50     ab.user_id = temp_activities2.user_id
51 ORDER BY
52     ab.age_bucket;
```

Question 100:

```
1
2 -- Q.100 Write a query to return the IDs of these LinkedIn power creators in ascending order.
3
4
5 SELECT
6     DISTINCT p.profile_id
7 FROM
8     personal_profiles p
9 INNER JOIN
10    employee_company ec
11 ON
12    p.profile_id = ec.personal_profile_id
13 INNER JOIN
14    company_pages c
15 ON
16    ec.company_id = c.company_id
17 WHERE
18    p.followers > c.followers
19 ORDER BY
20    p.profile_id;
21
```

Set 3:

Question 101:

```
1
2 -- Q.101 Write an SQL query to show the second most recent activity of each user.
3 -- If the user only has one activity, return that one. A user cannot perform more than one activity at the same time.
4 -- Return the result table in any order.
5
6
7 WITH temp_activity AS (
8     SELECT
9         username,
10        activity,
11        start_date,
12        end_date,
13        ROW_NUMBER() OVER(PARTITION BY username ORDER BY start_date, end_date) row_num,
14        COUNT(*) OVER(PARTITION BY username ORDER BY start_date, end_date
15                                rows between unbounded preceding and unbounded following) total_activities
16     FROM
17         user_activity
18     ),
19 temp_activity2 AS (
20     SELECT
21         username,
22         activity,
23         start_date,
24         end_date,
25         IF(total_activities = 1 and row_num = 1, 2, row_num) AS ranking
26     FROM
27         temp_activity
28     )
29
30 SELECT
31     username,
32     activity,
33     start_date,
34     end_date
35 FROM
36     temp_activity2
37 WHERE
38     ranking = 2;
39
```

Question 102:

```
1
2 -- Q.102 Write an SQL query to show the second most recent activity of each user.
3 -- If the user only has one activity, return that one. A user cannot perform more than one activity at the same time.
4 -- Return the result table in any order.
5
6
7 WITH temp_activity AS (
8     SELECT
9         username,
10        activity,
11        start_date,
12        end_date,
13        ROW_NUMBER() OVER(PARTITION BY username ORDER BY start_date, end_date) row_num,
14        COUNT(*) OVER(PARTITION BY username ORDER BY start_date, end_date
15                                rows between unbounded preceding and unbounded following) total_activities
16     FROM
17         user_activity
18     ),
19 temp_activity2 AS (
20     SELECT
21         username,
22         activity,
23         start_date,
24         end_date,
25         IF(total_activities = 1 and row_num = 1, 2, row_num) AS ranking
26     FROM
27         temp_activity
28     )
29
30 SELECT
31     username,
32     activity,
33     start_date,
34     end_date
35 FROM
36     temp_activity2
37 WHERE
38     ranking = 2;
39
```

Question 103:

```
1
2 -- Q.103 Query the name of any student in students who scored higher than 75 Marks. Order your output
3 -- by the last three characters of each name. If two or more students both have names ending in the same
4 -- last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending id.
5
6
7 SELECT
8     name
9 FROM
10    students
11 WHERE
12     marks > 75
13 ORDER BY
14     RIGHT(name,3),
15     id;
16
```

Question 104:

```
1
2 -- Q.104 Write a Query that prints a list of employee names (i.e.: the name attribute)
3 -- for employees in employee HAVING a salary greater than $2000 per month who have
4 -- been employees for less than 10 months. Sort your result by ascending employee_id.
5
6
7 SELECT
8     name
9 FROM
10    employee
11 WHERE
12     salary > 2000
13     AND
14     months < 10
15 ORDER BY
16     employee_id;
17
```

Question 105:

```
1
2 -- Q.105 Write a Query identifying the type of each record in the TRIANGLES table using its three side lengths.
3
4
5 SELECT
6     a,
7     b,
8     c,
9     CASE
10        WHEN a + b <= c OR b + c <= a OR a + c <= b THEN 'NOT A TRIANGLE'
11        WHEN a = b AND b = c THEN 'EQUILATERAL'
12        WHEN a = b OR b = c OR c = a THEN 'ISOSCELES'
13        WHEN a <> b AND b <> c THEN 'SCALEAN'
14        END AS type_of_triangle
15 FROM
16     triangles;
17
```

Question 106:

```
1
2 -- Q.106 Write a query calculating the amount of error (i.e.: actual - miscalculated average monthly salaries),
3 -- and round it up to the next integer.
4
5
6 SELECT
7     ceil(avg(salary) - avg(replace(salary, '0', ''))) AS error
8 FROM
9     employees;
10
```

Question 107:

```
1
2 -- Q.107 Write a query to find the maximum total earnings for all employees as
3 -- well as the total number of employees who have maximum total earnings.
4 -- Then print these values as 2 space-separated integers.
5
6
7 SELECT
8     MAX(salary*months) as total_earnings,
9     COUNT(*)
10    FROM
11        employee
12   WHERE
13       (salary*months) in (
14           SELECT
15               MAX(months * salary)
16             FROM
17                 employee
18           );
19
```

Question 108:

```
1
2 -- Q.108 a. Query an alphabetically ordered list of all names in OCCUPATIONS, immediately followed
3 -- by the first letter of each profession AS a parenthetical (i.e.: enclosed in parentheses).
4
5
6 SELECT
7     CONCAT(name, '(',substring(occupation, 1, 1),')') as `name(occupation)`
8 FROM
9     occupations
10 ORDER BY
11     name;
12
```

```
1
2 -- Q.108 b. WHERE [occupation_COUNT] is the number of occurrences of an occupation in OCCUPATIONS and [occupation]
3 -- is the lowerCASE occupation name. If more than one Occupation has the same [occupation_COUNT],
4 -- they should be ordered alphabetically.
5
6
7 SELECT
8     CONCAT("There are a total of ",
9         COUNT(*), ' ', lower(occupation), 's.') AS info
10    FROM
11        occupations
12   GROUP BY
13       occupation
14 ORDER BY
15     COUNT(occupation),
16     occupation;
17
```

Question 109:

```
1
2 -- Q.109 Pivot the Occupation column in OCCUPATIONS so that each Name is sorted alphabetically
3 -- and displayed underneath its corresponding Occupation. The output column headers should be
4 -- Doctor, Professor, Singer, and Actor, respectively.
5
6
7 SELECT
8     MAX(CASE WHEN occupation = 'Doctor' then name END) AS Doctor,
9     MAX(CASE WHEN occupation = 'Professor' then name END) AS Professor,
10    MAX(CASE WHEN occupation = 'Singer' then name END) AS Singer,
11    MAX(CASE WHEN occupation = 'Actor' then name END) AS Actor
12   FROM
13   (
14       SELECT
15           name,
16           occupation,
17           row_number() over(partition by occupation order by name) AS row_num
18      FROM
19          occupations
20   ) AS base
21 GROUP BY
22     row_num;
23
```

Question 110:

```
1
2 -- Q.110 Write a query to find the node type of Binary Tree ordered by the value of the node.
3 -- Output one of the following for each node:
4 -- • Root: If node is root node.
5 -- • Leaf: If node is leaf node.
6 -- • Inner: If node is neither root nor leaf node.
7
8
9 SELECT
10      n,
11      CASE
12          WHEN n NOT IN (SELECT DISTINCT p FROM bst WHERE p IS NOT NULL) THEN 'Leaf'
13          WHEN p IS NULL THEN 'Root'
14          ELSE 'Inner'
15      END AS type
16 FROM
17      bst
18 ORDER BY
19      n;
20
```

Question 111:

```
1
2 -- Q.111 Given the table schemas below, write a query to print the company_code,
3 -- founder name, total number of lead managers, total number of senior managers,
4 -- total number of managers, and total number of employees. Order your output by
5 -- ascending company_code.
6
7
8 SELECT
9     c.company_code,
10    c.founder,
11    COUNT(DISTINCT lm.lead_manager_code),
12    COUNT(DISTINCT sm.senior_manager_code),
13    COUNT(DISTINCT m.manager_code),
14    COUNT(DISTINCT e.employee_code)
15 FROM
16    company c
17 INNER JOIN
18        lead_manager lm
19 ON
20    c.company_code = lm.company_code
21 INNER JOIN
22        senior_manager sm
23 ON
24    sm.lead_manager_code = lm.lead_manager_code
25 INNER JOIN
26        manager m
27 ON
28    m.senior_manager_code = sm.senior_manager_code
29 INNER JOIN
30        employee e
31 ON
32    e.manager_code = m.manager_code
33 GROUP BY
34    c.company_code, c.founder
35 ORDER BY
36    c.company_code;
37
```

Question 112:

```
1
2 -- Q.112 Write a query to print all prime numbers less than or equal to 1000.
3 -- Print your result on a single line, and use the ampersand () character as
4 -- your separator (instead of a space).
5
6
7 WITH RECURSIVE number_generation AS (
8     SELECT
9         1 num
10
11     UNION ALL
12
13     SELECT
14         num + 1
15     FROM
16         number_generation
17     WHERE
18         num<1000
19
20     ),
21     number_generation2 AS (
22     SELECT
23         n1.num AS numm
24     FROM
25         number_generation n1
26     INNER JOIN
27         number_generation n2
28     WHERE
29         n1.num % n2.num = 0
30     GROUP BY
31         n1.num
32     HAVING
33         COUNT(n1.num) = 2
34
35
36     SELECT
37         group_concat(numm ORDER BY numm SEPARATOR '&') AS prime_numbers
38     FROM
39         number_generation2;
40
```

Question 113:

```
1
2 -- Q.113 Write a query to print the pattern P(20).
3
4
5 WITH RECURSIVE generate_numbers AS
6 (
7     SELECT
8         1 AS n
9
10    UNION
11
12    SELECT
13        n+1
14        FROM
15        generate_numbers
16        WHERE
17            n<20
18
19
20 SELECT
21     repeat('*',n)
22 FROM
23     generate_numbers;
24
```

Question 114:

```
1
2 -- Q.114 Write a query to print the pattern P(20).
3
4
5 WITH RECURSIVE generate_numbers AS
6 (
7     SELECT
8         20 AS n
9
10    UNION
11
12    SELECT
13        n-1
14    FROM
15        generate_numbers
16    WHERE
17        n>1
18 )
19
20 SELECT
21     repeat('*',n)
22 FROM
23     generate_numbers;
24
25
```

Question 115:

```
1
2 -- Q.115 Query the name of any student in students who scored higher than 75 Marks. Order your output
3 -- by the last three characters of each name. If two or more students both have names ending in the same
4 -- last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending id.
5
6
7 SELECT
8     name
9 FROM
10    students
11 WHERE
12     marks > 75
13 ORDER BY
14     RIGHT(name,3),
15     id;
16
```

Question 116:

```
1
2 -- Q.116 Write a Query that prints a list of employee names (i.e.: the name attribute)
3 -- from the employee table in alphabetical order.
4
5
6 SELECT
7     name
8 FROM
9     employee
10 ORDER BY
11     name;
12
```

Question 117:

```
1
2 -- Q.117 Write a Query that prints a list of employee names (i.e.: the name attribute)
3 -- for employees in employee HAVING a salary greater than $2000 per month who have
4 -- been employees for less than 10 months. Sort your result by ascending employee_id.
5
6
7 SELECT
8      name
9 FROM
10     employee
11 WHERE
12     salary > 2000
13     AND
14     months < 10
15 ORDER BY
16     employee_id;
17
```

Question 118:

```
1
2 -- Q.118 Write a Query identifying the type of each record in the TRIANGLES table using its three side lengths.
3
4
5 SELECT
6     a,
7     b,
8     c,
9     CASE
10        WHEN a + b <= c OR b + c <= a OR a + c <= b THEN 'NOT A TRIANGLE'
11        WHEN a = b AND b = c THEN 'EQUILATERAL'
12        WHEN a = b OR b = c OR c = a THEN 'ISOSCELES'
13        WHEN a <> b AND b <> c THEN 'SCALEAN'
14        END AS type_of_triangle
15 FROM
16     triangles;
17
```

Question 119:

```
1  -- Q.119 Write a Query to obtain the year-on-year growth rate for the total spend of each product for each year.
2
3
4
5  -- Approach 1
6
7
8  WITH temp_transactions AS (
9      SELECT
10         product_id,
11         transaction_date,
12         spend AS curr_year_spend,
13         LAG(spend,1,0) OVER w AS prev_year_spend,
14         IFNULL(spend - LAG(spend,1) OVER w, 0) AS prev_curr_spend_diff
15     FROM
16         user_transactions
17     WINDOW
18         w AS (PARTITION BY product_id ORDER BY EXTRACT(YEAR FROM transaction_date))
19
20
21  SELECT
22      product_id,
23      curr_year_spend,
24      ROUND(prev_year_spend, 2),
25      IFNULL(ROUND((prev_curr_spend_diff * 100)/prev_year_spend,2),0) AS YOY
26  FROM
27      temp_transactions;
28
29
30
31  -- Approach 2
32
33
34  WITH temp_transactions AS (
35      SELECT
36          YEAR(STR_TO_DATE(transaction_date, '%m/%d/%Y')) AS YEAR_id,
37          product_id,
38          transaction_date,
39          spend AS curr_year_spend,
40          LAG(spend,1) OVER w AS prev_year_spend,
41          spend - LAG(spend,1) OVER w AS prev_curr_spend_diff
42      FROM
43          user_transactions
44      WINDOW
45          w AS (PARTITION BY product_id ORDER BY EXTRACT(YEAR FROM transaction_date))
46
47
48  SELECT
49      year_id,
50      product_id,
51      curr_year_spend,
52      ROUND(prev_year_spend, 2),
53      ROUND((prev_curr_spend_diff * 100)/prev_year_spend,2) AS YOY
54  FROM
55      temp_transactions;
56
57
58  -- Approach 3
59
60
61  SELECT
62      product_id,
63      YEAR(STR_TO_DATE(transaction_date, '%m/%d/%Y')) AS YEAR_id,
64      spend AS curr_year_spend,
65      ROUND(LAG(spend,1,0) OVER w ,2) AS prev_year_spend,
66      ROUND((spend - LAG(spend,1) OVER w ) * 100 /
67              LAG(spend,1) OVER w, 2) AS YOY
68  FROM
69      user_transactions
70  WINDOW
71      w AS (PARTITION BY product_id ORDER BY EXTRACT(YEAR FROM transaction_date));
72
```

Question 120:

```
1
2 -- Q.120 Write a SQL Query to find the number of prime and non-prime items that can be stored
3 -- in the 500,000 square feet warehouse. Output the item type and number of items to be stocked.
4
5
6 -- Approach 1
7
8
9 WITH temp_inventory AS (
10     SELECT
11         item_type,
12         SUM(square_foot) AS square_foot_per_category,
13         COUNT(*) AS count_of_items
14     FROM
15         inventory
16     GROUP BY
17         item_type
18 ),
19
20 temp_inventory2 AS (
21     SELECT
22         (500000 - SUM(square_foot_per_category)*FLOOR(500000/SUM(square_foot_per_category))) AS area_left
23     FROM
24         temp_inventory
25     WHERE
26         item_type = 'PRIME_ELIGIBLE'
27 ),
28
29 temp_inventory3 AS (
30     SELECT
31         item_type,
32         CASE
33             WHEN item_type = 'PRIME_ELIGIBLE'
34                 THEN FLOOR(500000/square_foot_per_category) * count_of_items
35             WHEN item_type = 'NOT_PRIME'
36                 THEN FLOOR((SELECT area_left FROM temp_inventory2) / square_foot_per_category) * count_of_items
37         END AS item_count
38     FROM
39         temp_inventory
40     )
41
42 SELECT
43     item_type,
44     item_count
45 FROM
46     temp_inventory3;
```

```
1
2 -- Approach 2
3
4
5 WITH temp_inventory AS (
6     SELECT
7         item_type,
8             SUM(square_foot) AS square_foot_per_category,
9             COUNT(*) AS count_of_items,
10            CASE
11                WHEN item_type = 'PRIME_ELIGIBLE' THEN FLOOR(500000/SUM(square_foot)) * COUNT(*)
12                END AS prime_items_count
13        FROM
14            inventory
15        GROUP BY
16            item_type
17    ),
18
19 temp_inventory2 AS (
20     SELECT
21         (50000 - SUM(square_foot_per_category)*FLOOR(500000/SUM(square_foot_per_category))) AS area_left
22        FROM
23            temp_inventory
24        WHERE
25            item_type = 'PRIME_ELIGIBLE'
26    ),
27
28 temp_inventory3 AS (
29     SELECT
30         item_type,
31         CASE
32             WHEN item_type = 'PRIME_ELIGIBLE'
33                 THEN prime_items_count
34             WHEN item_type = 'NOT_PRIME'
35                 THEN FLOOR((SELECT area_left FROM temp_inventory2) / square_foot_per_category) * count_of_items
36             END AS item_count
37        FROM
38            temp_inventory
39    )
40
41 SELECT
42     item_type,
43     item_count
44 FROM
45     temp_inventory3;
46
```

Question 121:

```
1
2 -- Q.121 Write a Query to obtain the active user retention in July 2022.
3 -- Output the month (in numerical format 1, 2, 3) and the number of monthly active users (MAUs).
4
5
6 -- Approach 1
7
8
9 WITH temp_actions AS (
10     SELECT
11         user_id,
12         event_date,
13         event_type,
14         SUBSTR(event_date, 6, 2) - lag(SUBSTR(event_date, 6, 2)) OVER w AS difference
15     FROM
16         user_actions
17     WINDOW
18         w AS (PARTITION BY user_id ORDER BY event_date)
19     ),
20
21 temp_actions2 AS (
22     SELECT
23         SUBSTR(event_date, 6, 2) AS months,
24         COUNT(user_id) AS monthly_active_users
25     FROM
26         temp_actions
27     WHERE
28         difference = 1 AND event_type IN ('LIKE', 'COMMENT', 'SIGN-IN')
29     GROUP BY
30         months
31     )
32
33 SELECT
34     months,
35     monthly_active_users
36 FROM
37     temp_actions2;
38
39
40 -- Approach 2
41
42
43 WITH temp_actions AS (
44     SELECT
45         user_id,
46         event_date,
47         event_type,
48         SUBSTR(event_date, 6, 2) - lag(SUBSTR(event_date, 6, 2)) OVER w AS difference
49     FROM
50         user_actions
51     WINDOW
52         w AS (PARTITION BY user_id ORDER BY event_date)
53     )
54
55
56 SELECT
57     SUBSTR(event_date, 6, 2) AS months,
58     COUNT(DISTINCT user_id) AS active_users
59 FROM
60     temp_actions
61 WHERE
62     difference = 1
63     AND
64     event_type IN ('LIKE', 'COMMENT', 'SIGN-IN')
65 GROUP BY
66     months;
```

Question 122:

```
1
2 -- Q.122 Write a query to report the median of searches made by a user.
3 -- Round the median to one decimal point.
4
5
6 WITH temp_search_freq AS (
7     SELECT
8         searches,
9         num_users,
10        ROW_NUMBER() OVER(ORDER BY searches) row_num,
11        COUNT(*) OVER(ORDER BY searches ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) total_records
12    FROM
13    search_frequency
14),
15 temp_search_freq2 AS (
16     SELECT
17         searches,
18         num_users,
19         CASE
20             WHEN total_records % 2 <> 0 THEN (
21                 SELECT
22                     DISTINCT ROUND(SUM(searches) OVER w /
23                     COUNT(*) OVER w,1)
24                 FROM
25                 temp_search_freq
26                 WHERE
27                     row_num = ROUND((total_records + 1) / 2, 0)
28                     WINDOW
29                         w AS (ORDER BY searches ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
30             )
31
32             WHEN total_records % 2 = 0 THEN (
33                 SELECT
34                     DISTINCT ROUND(SUM(searches) OVER w /
35                     COUNT(*) OVER w,1)
36                 FROM
37                 temp_search_freq
38                 WHERE
39                     row_num IN (total_records/2,(total_records/2)+1)
40                     WINDOW
41                         w AS (ORDER BY searches ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
42             )
43         END AS median
44     FROM
45     temp_search_freq
46 )
47
48 )
49
50
51 SELECT
52     DISTINCT median
53 FROM
54     temp_search_freq2;
55
```

Question 123:

```
1
2 -- Q.123 Write a Query to update the Facebook advertisers status using the daily_pay table.
3 -- Advertiser is a two-column table containing the user id and their payment status based
4 -- on the last payment and daily_pay table has current information about their payment.
5 -- Only advertisers who paid will show up in this table.
6 -- Output the user id and current payment status sorted by the user id.
7
8
9 SELECT
10      user_id,
11      CASE
12          WHEN user_id IN (SELECT user_id FROM daily_pay) THEN 'EXISTING'
13          ELSE 'CHURN'
14      END AS new_status
15 FROM
16      advertiser
17 ORDER BY
18      user_id;
19
```

Question 124:

```
1
2 -- Q.124 Write a SQL Query that calculates the total time that the fleet of
3 -- servers was running. The output should be in units of full days.
4
5
6 -- Approach 1
7
8
9 SELECT      stop_time - start_time AS total_up_time
10 FROM        (
11          SELECT
12            SUM(
13              CASE
14                WHEN session_status = 'start' THEN EXTRACT(DAY from STR_TO_DATE(status_time, '%m/%d/%y'))
15                END
16              ) AS start_time,
17            SUM(
18              CASE
19                WHEN session_status = 'stop' THEN EXTRACT(DAY from STR_TO_DATE(status_time, '%m/%d/%y'))
20                END
21              ) AS stop_time
22          FROM      server_utilization
23        ) temp_server_utilization;
24
25
26
27
28
29 -- Approach 2 (Works only in PostgreSQL)
30
31
32 WITH temp_server_utilization AS (
33   SELECT
34     server_id,
35     status_time AS start_time,
36     session_status,
37     lead(status_time) OVER(ORDER BY server_id,status_time) AS end_time
38   FROM      server_utilization
39
40 )
41
42 SELECT      EXTRACT(DAY FROM justify_hours(SUM(end_time - start_time))) as total_time
43 FROM        temp_server_utilization
44 WHERE       session_status = 'start';
45
46
47
48
```

Question 125:

```
1
2 -- Q.125 Sometimes, payment transactions are repeated by accident; it could be due to user error,
3 -- API failure or a retry error that causes a credit card to be charged twice.
4 -- Using the transactions table, identify any payments made at the same merchant with the
5 -- same credit card for the same amount within 10 minutes of each other. Count such repeated payments.
6
7
8 WITH temp_transactions AS (
9     SELECT
10         merchant_id,
11         credit_card_id,
12         amount,
13         transaction_timestamp,
14         LAG(transaction_timestamp) OVER w AS prev_tran_timestamp,
15         timestampdiff(MINUTE,LAG(transaction_timestamp) OVER w, transaction_timestamp) AS difference
16     FROM
17         transactions
18     WINDOW
19         w AS (PARTITION BY credit_card_id ORDER BY transaction_timestamp)
20     )
21
22 SELECT
23     COUNT(DISTINCT merchant_id) AS payment_count
24 FROM
25     temp_transactions
26 WHERE
27     difference <= 10;
28
29
30 -- Approach 2 (Works only in PostgreSQL)
31
32
33 WITH temp_transactions AS (
34     SELECT
35         merchant_id,
36         credit_card_id,
37         amount,
38         transaction_timestamp,
39         LAG(transaction_timestamp) OVER w AS prev_tran_timestamp,
40         EXTRACT(EPOCH FROM LAG(transaction_timestamp) OVER w - transaction_timestamp) AS difference
41     FROM
42         transactions
43     WINDOW
44         w AS (PARTITION BY credit_card_id ORDER BY transaction_timestamp)
45     )
46
47 SELECT
48     COUNT(DISTINCT merchant_id) AS payment_count
49 FROM
50     temp_transactions
51 WHERE
52     difference < 10;
53
```

Question 126:

```
1
2 -- Q.126 Write a SQL Query to find the bad experience rate in the first 14 days for new users who signed
3 -- up in June 2022. Output the percentage of bad experience rounded to 2 decimal places.
4
5
6 SELECT
7     ROUND(
8         SUM(
9             CASE
10                 WHEN status !='completed successfully' THEN 1 ELSE 0
11             END
12         )*100.0/count(*),2) AS bad_experience_pct
13 FROM
14     customers C
15 INNER JOIN
16     orders O
17 ON
18     o.customer_id = c.customer_id
19 WHERE
20     o.order_timestamp < date_add(STR_TO_DATE(signup_timestamp, '%m/%d/%Y'), INTERVAL 14 DAY)
21     AND
22     MONTH(STR_TO_DATE(signup_timestamp, '%m/%d/%Y')) = 06
23     AND
24     YEAR(STR_TO_DATE(signup_timestamp, '%m/%d/%Y')) = 2022;
25
```

Question 127:

```
1
2 -- Q.127 Write an SQL Query to find the total score for each gender on each day.
3 -- Return the result table ordered by gender and day in ascending order.
4
5
6 SELECT
7     gender,
8     day,
9     SUM(score_points) OVER(PARTITION BY gender ORDER BY day
10        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS total_points
11 FROM
12     scores
13 ORDER BY
14     gender,
15     day;
16
```

Question 128:

```
1
2 -- Q.128 Write an SQL Query to find the countries where this company can invest .
3 -- Return the result table in any order .
4
5
6 -- Approach 1
7
8
9 SELECT          name AS country
10    FROM (
11      SELECT          c.name,
12                      SUM(ca.duration) AS call_duration,
13                      COUNT(c.country_code) AS number_of_calls
14        FROM (
15          SELECT          id,
16                      name,
17                      CASE
18                          WHEN LEFT(SUBSTR(phone_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone_number, 1,3), (LENGTH(SUBSTR(phone_number, 1,3))-1))
19                          ELSE SUBSTR(phone_number, 1,3) END AS country_code
20                    FROM person
21          ) temp_person
22      JOIN country c
23      ON temp_person.country_code = c.country_code
24  JOIN calls ca
25  ON temp_person.id = caller_id
26 GROUP BY          c.name
27
28 UNION ALL
29
30 SELECT          c.name,
31                      SUM(ca.duration) AS call_duration,
32                      COUNT(c.country_code) AS number_of_calls
33    FROM (
34      SELECT          id,
35                      name,
36                      CASE
37                          WHEN LEFT(SUBSTR(phone_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone_number, 1,3), (length(SUBSTR(phone_number, 1,3))-1))
38                          ELSE SUBSTR(phone_number, 1,3) END AS country_code
39                    FROM person
40          ) temp_person
41      JOIN country c
42      ON temp_person.country_code = c.country_code
43  JOIN calls ca
44  ON temp_person.id = ca.callee_id
45 GROUP BY          c.name
46
47 HAVING          SUM(call_duration)/SUM(number_of_calls) > (SELECT AVG(duration) FROM calls);
```

```
1  ● ● ●
2  -- Approach 2
3
4
5  WITH temp_person AS (
6      SELECT
7          id,
8          name,
9          CASE
10             WHEN LEFT(SUBSTR(phone_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone_number, 1,3), (LENGTH(SUBSTR(phone_number, 1,3))-1))
11             ELSE SUBSTR(phone_number, 1,3) END AS country_code
12      FROM
13          person
14      )
15
16
17  SELECT
18      name AS country
19  FROM (
20      SELECT
21          c.name,
22          SUM(ca.duration) AS call_duration,
23          COUNT(c.country_code) AS number_of_calls
24      FROM
25          temp_person
26      JOIN
27          country c
28      ON
29          temp_person.country_code = c.country_code
30      JOIN
31          calls ca
32      ON
33          temp_person.id = ca.caller_id
34      GROUP BY
35          c.name
36
37 UNION ALL
38
39      SELECT
40          c.name,
41          SUM(ca.duration) AS call duration,
42          COUNT(c.country_code) AS number_of_calls
43      FROM
44          temp_person
45      JOIN
46          country c
47      ON
48          temp_person.country_code = c.country_code
49      JOIN
50          calls ca
51      ON
52          temp_person.id = ca.callee_id
53      GROUP BY
54          c.name
55
56    )temp
57  GROUP BY
58      name
59  HAVING
60      SUM(call_duration)/SUM(number_of_calls) > (SELECT AVG(duration) FROM calls);
61
```

Question 129:

```
1
2 -- Q.129 Write an SQL Query to report the median of all the numbers in the database
3 -- after decompressing the numbers table. Round the median to one decimal point.
4
5
6 WITH RECURSIVE num_frequency (num,frequency, i) AS
7     (
8         SELECT
9             num,
10            frequency,1
11        FROM
12          numbers
13
14        UNION ALL
15
16        SELECT
17            num,
18            frequency,
19            i+1
20        FROM
21          num_frequency
22        WHERE
23            num_frequency.i < num_frequency.frequency
24    ),
25
26 num_frequency2 AS (
27     SELECT
28         num,
29         frequency,
30         row_number() OVER(ORDER BY num, frequency) AS row_num,
31         COUNT(*) OVER(ORDER BY num, frequency ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS total_records
32     FROM
33       num_frequency
34   )
35
36 SELECT
37   DISTINCT CASE
38     WHEN total_records % 2 <> 0 THEN (
39       SELECT
40           DISTINCT ROUND(SUM(num) OVER w /
41           COUNT(*) OVER w, 1)
42       FROM
43         num_frequency2
44       WHERE
45         row_num = ROUND((total_records + 1) / 2, 0)
46       WINDOW
47         w AS (ORDER BY num, frequency ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING))
48
49     WHEN total_records % 2 = 0 THEN (
50       SELECT
51           DISTINCT ROUND(SUM(num) OVER w /
52           COUNT(*) OVER w, 1)
53       FROM
54         num_frequency2
55       WHERE
56         row_num IN (total_records/2,(total_records/2)+1)
57       WINDOW
58         w AS (ORDER BY num, frequency ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING))
59
60     END AS median
61   FROM
62     num_frequency2;
```

Question 130:

```
1
2 -- Q.130 Write an SQL Query to report the comparison result (higher/lower/same) of the average salary of
3 -- employees in a department to the companys average salary. Return the result table in any order.
4
5
6 WITH temp_comparison AS (
7     SELECT
8         s.employee_id,
9             e.department_id,
10            s.amount,
11            s.paydate,
12            avg(amount) OVER (PARTITION BY MONTH(paydate) ORDER BY month(paydate), employee_id
13                           ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) company_avg_salary,
14            avg(amount) OVER (PARTITION BY MONTH(paydate), department_id ORDER BY month(paydate)
15                           ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) department_avg
16        FROM
17            salary s
18        INNER JOIN
19            employee e
20        ON
21            e.employee_id = s.employee_id
22    )
23
24 SELECT
25     DISTINCT DATE_FORMAT(paydate, '%Y-%m') AS pay_month,
26     department_id,
27     CASE
28         WHEN company_avg_salary = department_avg THEN 'same'
29         WHEN company_avg_salary > department_avg THEN 'lower'
30         WHEN company_avg_salary < department_avg THEN 'higher'
31     END AS comparison
32 FROM
33     temp_comparison;
34
```

Question 131:

```
1
2 -- Q.131 Write an SQL Query to report for each install date, the number of players
3 -- that installed the game on that day, and the day one retention.
4
5
6 -- Approach 1
7
8
9 SELECT
10    a.event_date AS install_date,
11    COUNT(a.player_id) AS installs,
12    ROUND(COUNT(b.player_id) / COUNT(a.player_id), 2) AS day1_retention
13 FROM
14    (
15        SELECT
16            player_id,
17            MIN(event_date) AS event_date
18        FROM
19            activity
20        GROUP BY
21            player_id
22    ) a
23 LEFT JOIN
24    activity b
25 ON
26    a.player_id = b.player_id
27    AND
28    a.event_date + 1 = b.event_date
29 GROUP BY
30    a.event_date;
31
32
33 -- Approach 2
34
35
36 SELECT
37    a1.event_date AS install_dt,
38    COUNT(a1.player_id) AS installs,
39    ROUND(COUNT(a3.player_id) / COUNT(a1.player_id), 2) AS day1_retention
40 FROM
41    activity a1
42 LEFT JOIN
43    activity a2
44 ON
45    a1.player_id = a2.player_id
46    AND
47    a1.event_date > a2.event_date
48 LEFT JOIN
49    activity a3
50 ON
51    a1.player_id = a3.player_id
52    AND
53    DATEDIFF(a3.event_date, a1.event_date) = 1
54 WHERE
55    a2.event_date IS NULL
56 GROUP BY
57    a1.event_date;
58
```

Question 132:

```
1  -- Q.132 Write an SQL query to find the winner in each group. Return the result table in any order.
2
3
4
5  SELECT
6      group_id,
7      players AS player_id
8  FROM  (
9      SELECT
10         p.group_id,
11         CASE
12             WHEN first_player_goals > second_player_goals THEN first_player
13             WHEN first_player_goals < second_player_goals THEN second_player
14             WHEN first_player_goals = second_player_goals THEN IF(first_player < second_player, first_player, second_player)
15         END AS players,
16         MAX(IF(first_player_goals > second_player_goals, first_player_goals, second_player_goals)) AS goals,
17         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
18
19     FROM
20     players p
21     INNER JOIN
22     matches m
23     ON
24         m.first_player = p.player_id
25         OR
26         m.second_player = p.player_id
27     GROUP BY
28         p.group_id,
29         players
30 ) temp_matches
31 WHERE
32     ranking = 1;
```

Question 133:

```
1
2 -- Q.133 Write an SQL Query to report the students (student_id, student_name) being -- Quiet in all exams.
3 -- Do not return the student who has never taken any exam.
4
5
6 -- Approach 1
7
8
9 WITH temp_examination AS (
10     SELECT
11         exam_id,
12         student_id,
13         score,
14         max(score) OVER w AS highest,
15         min(score) OVER w AS lowest
16     FROM
17         exam
18     WINDOW
19         w AS (PARTITION BY exam_id)
20     ),
21
22 temp_examination1 AS (
23     SELECT
24         DISTINCT student_id
25     FROM
26         temp_examination
27     WHERE
28         score IN (lowest, highest)
29     )
30
31     SELECT
32         DISTINCT s.student_id,
33         s.student_name
34     FROM
35         temp_examination
36     INNER JOIN
37         student s
38     ON
39         s.student_id = temp_examination.student_id
40     WHERE
41         s.student_id NOT IN (SELECT student_id FROM temp_examination1);
42
43
44 -- Approach 2
45
46
47 WITH temp_examination AS (
48     SELECT
49         student_id,
50         CASE
51             WHEN score < max(score) OVER(PARTITION BY exam_id) AND score > min(score) OVER(PARTITION BY exam_id) THEN 0
52             ELSE 1
53         END AS category
54     FROM
55         exam
56     ORDER BY
57         student_id
58     ),
59
60 temp_examination1 AS (
61     SELECT
62         student_id,
63         SUM(category) AS high_low_count
64     FROM
65         temp_examination
66     GROUP BY
67         student_id
68     )
69
70     SELECT
71         s.student_id,
72         s.student_name
73     FROM
74         student s
75     INNER JOIN
76         temp_examination1
77     ON
78         s.student_id = temp_examination1.student_id
79     WHERE
80         high_low_count = 0
81     ORDER BY
82         temp_examination1.student_id;
```

Question 134:

```
1
2 -- Q.134 Write an SQL Query to report the students (student_id, student_name) being -- Quiet in all exams.
3 -- Do not return the student who has never taken any exam.
4
5
6 -- Approach 1
7
8
9 WITH temp_examination AS (
10     SELECT
11         exam_id,
12         student_id,
13         score,
14         max(score) OVER w AS highest,
15         min(score) OVER w AS lowest
16     FROM
17         exam
18     WINDOW
19         w AS (PARTITION BY exam_id)
20     ),
21
22 temp_examination1 AS (
23     SELECT
24         DISTINCT student_id
25     FROM
26         temp_examination
27     WHERE
28         score IN (lowest, highest)
29     )
30
31     SELECT
32         DISTINCT s.student_id,
33         s.student_name
34     FROM
35         temp_examination
36     INNER JOIN
37         student s
38     ON
39         s.student_id = temp_examination.student_id
40     WHERE
41         s.student_id NOT IN (SELECT student_id FROM temp_examination1);
42
43
44 -- Approach 2
45
46
47 WITH temp_examination AS (
48     SELECT
49         student_id,
50         CASE
51             WHEN score < max(score) OVER(PARTITION BY exam_id) AND score > min(score) OVER(PARTITION BY exam_id) THEN 0
52             ELSE 1
53         END AS category
54     FROM
55         exam
56     ORDER BY
57         student_id
58     ),
59
60 temp_examination1 AS (
61     SELECT
62         student_id,
63         SUM(category) AS high_low_count
64     FROM
65         temp_examination
66     GROUP BY
67         student_id
68     )
69
70 SELECT
71     s.student_id,
72     s.student_name
73 FROM
74     student s
75 INNER JOIN
76     temp_examination1
77 ON
78     s.student_id = temp_examination1.student_id
79 WHERE
80     high_low_count = 0
81 ORDER BY
82     temp_examination1.student_id;
```

Question 135:

```
1
2 -- Q.135 Write an SQL query to show the second most recent activity of each user.
3 -- If the user only has one activity, return that one. A user cannot perform more than one activity at the same time.
4 -- Return the result table in any order.
5
6
7 WITH temp_activity AS (
8     SELECT
9         username,
10        activity,
11        start_date,
12        end_date,
13        ROW_NUMBER() OVER(PARTITION BY username ORDER BY start_date, end_date) row_num,
14        COUNT(*) OVER(PARTITION BY username ORDER BY start_date, end_date
15                           rows between unbounded preceding and unbounded following) total_activities
16     FROM
17         user_activity
18     ),
19 temp_activity2 AS (
20     SELECT
21         username,
22         activity,
23         start_date,
24         end_date,
25         IF(total_activities = 1 and row_num = 1, 2, row_num) AS ranking
26     FROM
27         temp_activity
28     )
29
30 SELECT
31     username,
32     activity,
33     start_date,
34     end_date
35 FROM
36     temp_activity2
37 WHERE
38     ranking = 2;
39
```

Question 136:

```
1
2 -- Q.136 Write an SQL query to show the second most recent activity of each user.
3 -- If the user only has one activity, return that one. A user cannot perform more than one activity at the same time.
4 -- Return the result table in any order.
5
6
7 WITH temp_activity AS (
8     SELECT
9         username,
10        activity,
11        start_date,
12        end_date,
13        ROW_NUMBER() OVER(PARTITION BY username ORDER BY start_date, end_date) row_num,
14        COUNT(*) OVER(PARTITION BY username ORDER BY start_date, end_date
15                                rows between unbounded preceding and unbounded following) total_activities
16     FROM
17         user_activity
18     ),
19 temp_activity2 AS (
20     SELECT
21         username,
22         activity,
23         start_date,
24         end_date,
25         IF(total_activities = 1 and row_num = 1, 2, row_num) AS ranking
26     FROM
27         temp_activity
28     )
29
30 SELECT
31     username,
32     activity,
33     start_date,
34     end_date
35 FROM
36     temp_activity2
37 WHERE
38     ranking = 2;
39
```

Question 137:

```
1
2 -- Q.137 Write a query calculating the amount of error (i.e.: actual - miscalculated average monthly salaries),
3 -- and round it up to the next integer.
4
5
6 SELECT
7     ceil(avg(salary) - avg(replace(salary, '0', ''))) AS error
8 FROM
9     employees;
10
```

Question 138:

```
1
2 -- Q.138 Write a Query identifying the type of each record in the TRIANGLES table using its three side lengths.
3
4
5 SELECT
6     a,
7     b,
8     c,
9     CASE
10        WHEN a + b <= c OR b + c <= a OR a + c <= b THEN 'NOT A TRIANGLE'
11        WHEN a = b AND b = c THEN 'EQUILATERAL'
12        WHEN a = b OR b = c OR c = a THEN 'ISOSCELES'
13        WHEN a <> b AND b <> c THEN 'SCALEAN'
14        END AS type_of_triangle
15 FROM
16     triangles;
17
```

Question 139:

```
1
2 -- Q.139 Write a Query identifying the type of each record in the TRIANGLES table using its three side lengths.
3
4
5 SELECT
6     a,
7     b,
8     c,
9     CASE
10        WHEN a + b <= c OR b + c <= a OR a + c <= b THEN 'NOT A TRIANGLE'
11        WHEN a = b AND b = c THEN 'EQUILATERAL'
12        WHEN a = b OR b = c OR c = a THEN 'ISOSCELES'
13        WHEN a <> b AND b <> c THEN 'SCALEAN'
14        END AS type_of_triangle
15 FROM
16     triangles;
17
```

Question 140:

```
1
2 -- Q.140 Write a Query identifying the type of each record in the TRIANGLES table using its three side lengths.
3
4
5 SELECT
6     a,
7     b,
8     c,
9     CASE
10        WHEN a + b <= c OR b + c <= a OR a + c <= b THEN 'NOT A TRIANGLE'
11        WHEN a = b AND b = c THEN 'EQUILATERAL'
12        WHEN a = b OR b = c OR c = a THEN 'ISOSCELES'
13        WHEN a <> b AND b <> c THEN 'SCALEAN'
14        END AS type_of_triangle
15 FROM
16     triangles;
17
```

Question 141:

```
1
2 -- Q.141 Write a query to find the node type of Binary Tree ordered by the value of the node.
3 -- Output one of the following for each node:
4 -- • Root: If node is root node.
5 -- • Leaf: If node is leaf node.
6 -- • Inner: If node is neither root nor leaf node.
7
8
9 SELECT
10    n,
11    CASE
12       WHEN n NOT IN (SELECT DISTINCT p FROM bst WHERE p IS NOT NULL)  THEN 'Leaf'
13       WHEN p IS NULL THEN 'Root'
14       ELSE 'Inner'
15    END AS type
16 FROM
17    bst
18 ORDER BY
19    n;
20
```

Question 142:

```
1
2 -- Q.142 Given the table schemas below, write a query to print the company_code,
3 -- founder name, total number of lead managers, total number of senior managers,
4 -- total number of managers, and total number of employees. Order your output by
5 -- ascending company_code.
6
7
8 SELECT
9     c.company_code,
10    c.founder,
11    COUNT(DISTINCT lm.lead_manager_code),
12    COUNT(DISTINCT sm.senior_manager_code),
13    COUNT(DISTINCT m.manager_code),
14    COUNT(DISTINCT e.employee_code)
15 FROM
16    company c
17 INNER JOIN
18        lead_manager lm
19 ON
20        c.company_code = lm.company_code
21 INNER JOIN
22        senior_manager sm
23 ON
24        sm.lead_manager_code = lm.lead_manager_code
25 INNER JOIN
26        manager m
27 ON
28        m.senior_manager_code = sm.senior_manager_code
29 INNER JOIN
30        employee e
31 ON
32        e.manager_code = m.manager_code
33 GROUP BY
34        c.company_code, c.founder
35 ORDER BY
36        c.company_code;
```

Question 143:

```
1
2 -- Q.143 Write a query to output all such symmetric pairs in ascending order by the value of X.
3 -- List the rows such that X1 ≤ Y1.
4
5
6 WITH temp_functions AS (
7         SELECT
8             x,
9             y,
10            ROW_NUMBER() OVER (ORDER BY x, y) AS row_num
11        FROM
12        functions
13    )
14
15 SELECT
16     DISTINCT f1.x,
17     f1.y
18 FROM
19     temp_functions f1
20 INNER JOIN
21     temp_functions f2
22 ON
23     f1.x = f2.y
24 AND
25     f1.y = f2.x
26 AND
27     f1.row_num <> f2.row_num
28 WHERE
29     f1.x <= f1.y
30 ORDER BY
31     f1.x;
32
```

Question 144:

```
1
2 -- Q.144 Write a query to output the names of those students whose best friENDs got offered a higher
3 -- salary than them. Names must be ordered by the salary amount offered to the best friENDs.
4 -- It is guaranteed that no two students get the same salary offer.
5
6
7 SELECT
8      s1.name
9 FROM
10     friends f1
11 INNER JOIN
12     students s1
13 ON
14     f1.id = s1.id
15 INNER JOIN
16     students s2
17 ON
18     f1.friend_id = s2.id
19 INNER JOIN
20     packages p1
21 ON
22     f1.id = p1.id
23 INNER JOIN
24     packages p2
25 ON
26     f1.friend_id = p2.id
27 WHERE
28     p1.salary < p2.salary
29 ORDER BY
30     p2.salary;
31
32
```

Question 145:

```
1
2 -- Q.145 Write a query to print the respective hacker_id and name of hackers who achieved full scores for
3 -- more than one challenge. Order your output in descENDING order by the total number of challenges in
4 -- which the hacker earned a full score. If more than one hacker received full scores in the same number
5 -- of challenges, then sort them by ascending hacker_id.
6
7
8 SELECT
9     h.hacker_id,
10    h.name
11 FROM
12    hackers h
13 INNER JOIN
14    submissions s
15 ON
16    h.hacker_id = s.hacker_id
17 INNER JOIN
18    challenges c
19 ON
20    s.challenge_id = c.challenge_id
21 INNER JOIN
22    difficulty d
23 ON
24    c.difficulty_level = d.difficulty_level
25 WHERE
26    s.score = d.score
27 GROUP BY
28    h.name, h.hacker_id
29 HAVING
30    COUNT(s.score) > 1
31 ORDER BY
32    COUNT(s.challenge_id) desc,
33    h.hacker_id;
34
```

Question 146:

```
1
2 -- Q.146 Write a query to output the start and END dates of projects listed by the number of days it took
3 -- to complete the project in ascending order. If there is more than one project that have the same number
4 -- of completion days, then order by the start date of the project.
5
6
7 -- Approach 1
8
9
10 WITH project_start AS
11      (
12          SELECT
13              start_date,
14              ROW_NUMBER() OVER() AS ps_rownum
15          FROM
16              projects
17          WHERE
18              start_date not in (
19                  SELECT
20                      end_date
21                  FROM
22                      projects
23              )
24      ),
25 project_end AS
26      (
27          SELECT
28              end_date,
29              ROW_NUMBER() OVER() AS pe_rownum
30          FROM
31              projects
32          WHERE
33              END_date not in (
34                  SELECT
35                      start_date
36                      FROM
37                          projects
38              )
39      )
40
41 SELECT
42     project_start.start_date,
43     project_end.end_date
44 FROM
45     project_start
46 INNER JOIN
47     project_end
48 on
49     project_end.pe_rownum = project_start.ps_rownum
50 ORDER BY
51     DATEDIFF(project_start.start_date, project_end.end_date) desc,
52     project_start.start_date;
```

```
1
2  -- Approach 2
3
4
5  WITH temp_project AS (
6      SELECT
7          temp.start_date,
8          temp.end_date,
9          SUM(
10             CASE
11                 WHEN previous_end_date IS NULL THEN 0
12                 WHEN DAY(end_date) - DAY(previous_end_date) = 1 THEN 0
13                 ELSE 1
14             END
15         ) OVER(ORDER BY start_date) AS project_num
16     FROM (
17         SELECT
18             start_date,
19             end_date,
20             LAG(end_date) OVER (ORDER BY start_date) AS previous_end_date
21         FROM
22             projects
23     ) temp
24 )
25
26 SELECT
27     MIN(start_date) AS project_start_date,
28     MAX(end_date) as project_end_date
29 FROM
30     temp_project
31 GROUP BY
32     project_num
33 ORDER BY
34     DAY(MAX(end_date))-DAY(MIN(start_date));
35
```

```
1
2 -- Approach 3
3
4
5 WITH temp_project AS (
6     SELECT
7         temp.start_date,
8         temp.end_date,
9         SUM(
10            CASE
11                WHEN previous_end_date IS NULL THEN 0
12                WHEN DAY(end_date) - DAY(previous_end_date) = 1 THEN 0
13                ELSE 1
14            END
15        ) over(order by start_date range between unbounded preceding and current row) AS project_num
16     FROM (
17         SELECT
18             start_date,
19             end_date,
20             LAG(end_date) OVER (ORDER BY start_date) AS previous_end_date
21         FROM
22             projects
23     ) temp
24 )
25
26 SELECT
27     MIN(start_date) AS project_start_date,
28     MAX(end_date) AS project_end_date
29 FROM
30     temp_project
31 GROUP BY
32     project_num
33 ORDER BY
34     DAY(MAX(end_date))-DAY(MIN(start_date));
35
```

Question 147:

```
1
2 -- Q.147 In an effort to identify high-value customers, Amazon asked for your help to obtain data
3 -- about users who go on shopping sprees. A shopping spree occurs when a user makes purchases on 3
4 -- or more consecutive days. List the user IDs who have gone on at least 1 shopping spree in ascending order.
5
6
7 SELECT
8     DISTINCT user_id
9 FROM
10    (
11        SELECT
12            user_id,
13            transaction_date,
14            rn,
15            transaction_date :: date - rn::integer,
16            COUNT(transaction_date :: date - rn::integer) OVER(PARTITION BY user_id) AS cn
17        FROM
18        (
19            SELECT
20                user_id,
21                transaction_date,
22                ROW_NUMBER() OVER(PARTITION BY user_id ORDER BY transaction_date) AS rn
23            FROM
24                transactions
25            ) temp_transactions
26        )temp_transactions1
27
28 WHERE cn >=3;
29
30
31 -- SOLVED IN POSTGRESQL
32
```

Question 148:

```
1
2 -- Q.148 You are given a table of PayPal payments showing the payer, the recipient, and the amount paid.
3 -- A two-way unique relationship is established WHEN two people sEND money back and forth. Write a query
4 -- to find the number of two-way unique relationships in this data.
5
6
7 WITH temp_payments AS (
8     SELECT
9         DISTINCT p1.payer_id,
10        p1.recipient_id
11    FROM
12        payments p1
13    INNER JOIN
14        payments p2
15    ON
16        p1.payer_id = p2.recipient_id
17    AND
18        p2.payer_id = p1.recipient_id
19    AND
20        p1.payer_id < p2.payer_id
21 )
22
23 SELECT
24     COUNT(*) unique_relationships
25 FROM
26     temp_payments;
27
```

Question 149:

```
1
2 -- Q.149 Write a query to obtain the list of customers whose first transaction was valued at $50 or more.
3 -- Output the number of users.
4
5
6 WITH temp_transactions AS (
7     SELECT
8         user_id,
9         transaction_date,
10        spend,
11        ROW_NUMBER() OVER(PARTITION BY user_id ORDER BY transaction_date) row_num
12     FROM
13     user_transactions
14 )
15
16 SELECT
17     COUNT(DISTINCT user_id) as users
18 FROM
19     temp_transactions
20 WHERE
21     row_num = 1
22     and
23     spend >= 50;
24
```

Question 150:

```
1
2 -- Q.150 Write a query to obtain the SUM of the odd-numbered and even-numbered measurements on a particular day,
3 -- in two different columns.
4
5
6 WITH temp_measurements AS (
7     SELECT
8         measurement_value,
9         measurement_time,
10        ROW_NUMBER() OVER(PARTITION BY measurement_time::DATE ORDER BY measurement_time) row_num
11    FROM
12        measurements
13 )
14
15 SELECT
16     measurement_time::DATE,
17     ROUND(SUM(
18         CASE
19             WHEN row_num % 2 <> 0 THEN measurement_value
20             END),2) AS odd_value,
21     ROUND(SUM(
22         CASE
23             WHEN row_num % 2 = 0 THEN measurement_value
24             END),2) AS even_value
25 FROM
26     temp_measurements
27 GROUP BY
28     measurement_time::DATE,
29 ORDER BY
30     measurement_time;
31
32
33 -- SOLVED IN POSTGRESQL
34
```

Question 151:

```
1
2 -- Q.151 In an effort to identify high-value customers, Amazon asked for your help to obtain data
3 -- about users who go on shopping sprees. A shopping spree occurs when a user makes purchases on 3
4 -- or more consecutive days. List the user IDs who have gone on at least 1 shopping spree in ascending order.
5
6
7 SELECT
8     DISTINCT user_id
9 FROM
10    (
11        SELECT
12            user_id,
13            transaction_date,
14            rn,
15            transaction_date :: date - rn::integer,
16            COUNT(transaction_date :: date - rn::integer) OVER(PARTITION BY user_id) AS cn
17        FROM
18        (
19            SELECT
20                user_id,
21                transaction_date,
22                ROW_NUMBER() OVER(PARTITION BY user_id ORDER BY transaction_date) AS rn
23            FROM
24                transactions
25        ) temp_transactions
26    )temp_transactions1
27
28 WHERE cn >=3;
29
```

Question 152:

```
1 -- Q.152 The Airbnb Booking RecommENDations team is trying to understand the "substitutability" of two rentals
2 -- and whether one rental is a good substitute for another. They want you to write a query to find the unique
3 -- combination of two Airbnb rentals WITH the same exact amenities offered. Output the COUNT of the unique
4 -- combination of Airbnb rentals.
5
6
7
8 WITH temp_amenities AS (
9     SELECT
10        rental_id,
11        amenity,
12        COUNT(amenity) over(partition by rental_id) AS no_of_amenities
13    FROM
14        rental_amenities
15    ),
16
17 temp_amenities2 AS (
18     SELECT
19        COUNT(*)
20    FROM
21        temp_amenities a
22    INNER JOIN
23        temp_amenities b
24    ON
25        a.no_of_amenities = b.no_of_amenities
26    AND
27        a.amenity = b.amenity
28    AND
29        a.rental_id <> b.rental_id
30    GROUP BY
31        a.rental_id,
32        b.rental_id,
33        a.no_of_amenities
34    HAVING
35        COUNT(*) = a.no_of_amenities
36    )
37
38
39 SELECT
40     CEIL(COUNT(*)/2) as matching_airbnb
41 FROM
42     temp_amenities2;
43
```

Question 153:

```
1
2 -- Q.153 Write a query to calculate the return on ad spend (ROAS) for each advertiser
3 -- across all ad campaigns. Round your answer to 2 decimal places, and order your
4 -- output by the advertiser_id.
5
6
7 SELECT
8     advertiser_id,
9     ROUND(SUM(revenue) / SUM(spend), 2) AS ROAS
10 FROM
11     ad_campaigns
12 GROUP BY
13     advertiser_id
14 ORDER BY
15     advertiser_id;
16
```

Question 154:

```
1
2 -- Q.154 Write a query that shows the following data for each compensation outlier:
3 -- employee ID, salary, and whether they are potentially overpaid or potentially underpaid
4
5
6 WITH temp_compensation AS (
7         SELECT
8             employee_id,
9                 salary,
10                title,
11                   ROUND(AVG(salary) over(PARTITION BY title),2) AS avg_salary
12          FROM
13            employee_pay
14        ),
15 temp_compensation2 AS (
16         SELECT
17             employee_id,
18                 salary,
19                   CASE
20                     WHEN salary > 2 * avg_salary THEN 'Overpaid'
21                     WHEN salary < avg_salary/2 THEN 'Underpaid'
22                   END AS status
23          FROM
24            temp_compensation
25        )
26
27 SELECT
28     employee_id,
29             salary,
30             status
31 FROM
32   temp_compensation2
33 WHERE
34   status is not null;
35
```

Question 155:

```
1
2 -- Q.155 You are given a table of PayPal payments showing the payer, the recipient, and the amount paid.
3 -- A two-way unique relationship is established WHEN two people sEND money back and forth. Write a query
4 -- to find the number of two-way unique relationships in this data.
5
6
7 WITH temp_payments AS (
8     SELECT
9         DISTINCT p1.payer_id,
10        p1.recipient_id
11     FROM
12        payments p1
13     INNER JOIN
14        payments p2
15     ON
16        p1.payer_id = p2.recipient_id
17     AND
18        p2.payer_id = p1.recipient_id
19     AND
20        p1.payer_id < p2.payer_id
21 )
22
23 SELECT
24     COUNT(*) unique_relationships
25 FROM
26     temp_payments;
27
```

Question 156:

```
1
2 -- Q.156 Assume you are given the table below containing information on user
3 -- purchASes. Write a query to obtain the number of users who purchASed the
4 -- same product on two or more different days. Output the number of unique users.
5
6
7 SELECT
8     COUNT(DISTINCT user_id) AS repeat_purchasers
9 FROM (
10     SELECT
11         user_id
12     FROM
13         purchases
14     GROUP BY
15         user_id,
16         product_id
17     HAVING
18         COUNT(DISTINCT purchase_date) > 1
19 ) temp;
20
```

Question 157:

```
1
2 -- Q.157 Say you have access to all the transactions for a given merchant account.
3 -- Write a query to print the cumulative balance of the merchant account at the end
4 -- of each day, WITH the total balance reset back to zero at the end of the month.
5 -- Output the transaction date and cumulative balance.
6
7
8 SELECT
9     DISTINCT date(transaction_date),
10    SUM(
11        CASE
12            WHEN type = 'deposit' THEN amount
13            ELSE -amount
14        END
15    ) OVER(PARTITION BY EXTRACT(MONTH FROM transaction_date) ORDER BY DATE(transaction_date)) AS balance
16 FROM
17     transactions;
18
19
20 -- SOLVED IN POSTGRESQL
21
```

Question 158:

```
1
2 -- Q.158 Assume you are given the table below containing information on
3 -- Amazon customers and their spend on products belonging to various
4 -- categories. Identify the top two highest-grossing products within each
5 -- category in 2022. Output the category, product, and total spend.
6
7
8 WITH temp_product_details AS (
9     SELECT
10        category,
11        product,
12        spend,
13        SUM(spend) OVER(PARTITION BY category, product) total_spend
14    FROM
15        product_spend
16    WHERE
17        EXTRACT(YEAR FROM transaction_date) = 2022
18    ),
19
20 temp_product_details1 AS (
21     SELECT
22        DISTINCT category,
23        product,
24        total_spend ,
25        DENSE_RANK() OVER(PARTITION BY category ORDER BY total_spend DESC) row_num
26    FROM
27        temp_product_details
28    )
29
30
31 SELECT
32     DISTINCT category,
33     product,
34     total_spend
35 FROM
36     temp_product_details1
37 WHERE
38     row_num <=2
39 ORDER BY
40     category,
41     total_spend DESC;
42
43
44 -- SOLVED IN POSTGRESQL
45
```

Question 159:

```
● ● ●
1
2 -- Q.159 Facebook is analysing its user signup data for June 2022.
3 -- Write a query to generate the churn rate by week in June 2022.
4 -- Output the week number (1, 2, 3, 4, ...) and the corresponding
5 -- churn rate rounded to 2 decimal places.
6
7
8 -- Approach 1
9
10
11 WITH temp_churn_rate AS (
12     SELECT
13         user_id,
14         signup_date,
15         last_login,
16         DATEDIFF(last_login, signup_date) diff,
17         EXTRACT(WEEK FROM signup_date) AS week_no,
18         WEEK(signup_date,5) - WEEK(DATE_SUB(signup_date, INTERVAL DAYOFMONTH(signup_date)-1 DAY),5)+1 AS ranking
19
20     FROM
21         users
22
23     WHERE
24         EXTRACT(MONTH FROM signup_date) = 6
25         AND
26         EXTRACT(YEAR FROM signup_date) = 2022
27
28     ), temp_churn_rate2 AS (
29         SELECT
30             ranking,
31             COUNT(ranking) AS total_users,
32             COUNT(
33                 CASE
34                     WHEN diff <= 28 THEN 1
35                 END
36             ) AS total_churns
37
38         FROM
39             temp_churn_rate
40         GROUP BY
41             ranking
42
43     )
44
45     SELECT
46         ranking AS week,
47         ROUND((total_churns/total_users) * 100 ,2) AS churn_rate
48
49     FROM
50         temp_churn_rate2
51
52     ORDER BY
53         ranking;
54
55
56 -- Approach 2
57
58
59 WITH temp_churn_rate AS (
60     SELECT
61         user_id,
62         signup_date,
63         last_login,
64         DATEDIFF(last_login, signup_date) diff,
65         EXTRACT(WEEK FROM signup_date) AS week_no,
66         DENSE_RANK() OVER(ORDER BY EXTRACT(WEEK FROM signup_date)) ranking
67
68     FROM
69         users
70
71     WHERE
72         EXTRACT(MONTH FROM signup_date) = 6
73         AND
74         EXTRACT(YEAR FROM signup_date) = 2022
75
76     ), temp_churn_rate2 AS (
77         SELECT
78             ranking,
79             COUNT(ranking) AS total_users,
80             COUNT(
81                 CASE
82                     WHEN diff <= 28 THEN 1
83                 END
84             ) AS total_churns
85
86         FROM
87             temp_churn_rate
88         GROUP BY
89             ranking
90
91     )
92
93     SELECT
94         ranking AS week,
95         ROUND((total_churns/total_users) * 100 ,2) AS churn_rate
96
97     FROM
98         temp_churn_rate2
99     ORDER BY
100        ranking;
```