Q1. Query all columns for all American cities in the CITY table with populations larger than 100000. The CountryCode for America is USA.

**ANSWER:** SELECT \* FROM CITY WHERE COUNTRYCODE = 'USA' AND POPULATION > 100000;

Q2. Query the NAME field for all American cities in the CITY table with populations larger than 120000. The CountryCode for America is USA.

**ANSWER:** SELECT NAME FROM CITY WHERE COUNTRYCODE = 'USA' AND POPULATION > 120000;

Q3. Query all columns (attributes) for every row in the CITY table.

**ANSWER:** SELECT \* FROM CITY;

Q4. Query all columns for a city in CITY with the ID 1661.

**ANSWER:** SELECT \* FROM CITY WHERE ID = 1661;

Q5.Query all attributes of every Japanese city in the CITY table. The COUNTRYCODE for Japan is JPN.

**ANSWER:** SELECT \* FROM CITY WHERE COUNTRYCODE = 'JPN';

Q6. Query the names of all the Japanese cities in the CITY table. The COUNTRYCODE for Japan is

JPN.**ANSWER:** SELECT NAME FROM CITY WHERE COUNTRYCODE = 'JPN';

Q7. Query a list of CITY and STATE from the STATION table.

**ANSWER:** SELECT CITY, STATE FROM STATION;

Q8. Query a list of CITY names from STATION for cities that have an even ID number. Print the results in any order, but exclude duplicates from the answer.

**ANSWER:** SELECT DISTINCT CITY FROM STATION WHERE MOD(ID, 2) = 0;

Q9. Find the difference between the total number of CITY entries in the table and the number of distinct CITY entries in the table.

**ANSWER:** SELECT COUNT(\*) - COUNT(DISTINCT CITY) AS DIFFERENCE FROM CITY;

Q10. Query the two cities in STATION with the shortest and longest CITY names, as well as their respective lengths (i.e.: number of characters in the name). If there is more than one smallest or largest city, choose the one that comes first when ordered alphabetically.

**ANSWER:** SELECT CITY, LENGTH(CITY) FROM STATION WHERE LENGTH(CITY) = (SELECT MIN(LENGTH(CITY)) FROM STATION) OR LENGTH(CITY) = (SELECT MAX(LENGTH(CITY)) FROM STATION) ORDER BY LENGTH(CITY), CITY;

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

**ANSWER:** SELECT DISTINCT CITY FROM STATION WHERE CITY LIKE 'A%' OR CITY LIKE 'E%' OR CITY LIKE 'I%' OR CITY LIKE 'O%' OR CITY LIKE 'U%';

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

**ANSWER:** SELECT DISTINCT CITY FROM STATION WHERE CITY LIKE '%a' OR CITY LIKE '%e' OR CITY LIKE '%i' OR CITY LIKE '%o' OR CITY LIKE '%u';

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

**ANSWER:** SELECT DISTINCT CITY FROM STATION WHERE CITY NOT LIKE 'A%' AND CITY NOT LIKE 'E%' AND CITY NOT LIKE 'I%' AND CITY NOT LIKE 'O%' AND CITY NOT LIKE 'U%';

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

**ANSWER:** SELECT DISTINCT CITY FROM STATION WHERE CITY NOT LIKE '%a' AND CITY NOT LIKE '%e' AND CITY NOT LIKE '%i' AND CITY NOT LIKE '%o' AND CITY NOT LIKE '%u';

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

**ANSWER:** SELECT DISTINCT CITY FROM STATION WHERE (CITY NOT LIKE 'A%' AND CITY NOT LIKE 'E%' AND CITY NOT LIKE 'I%' AND CITY NOT LIKE 'O%' AND CITY NOT LIKE 'U%') OR (CITY NOT LIKE '%a' AND CITY NOT LIKE '%e' AND CITY NOT LIKE '%i' AND CITY NOT LIKE '%o' AND CITY NOT LIKE '%u');

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

**ANSWER:** SELECT DISTINCT CITY FROM STATION WHERE CITY NOT LIKE 'A%' AND CITY NOT LIKE 'E%' AND CITY NOT LIKE 'I%' AND CITY NOT LIKE 'O%' AND CITY NOT LIKE 'U%' AND CITY NOT LIKE '%a' AND CITY NOT LIKE '%e' AND CITY NOT LIKE '%i' AND CITY NOT LIKE '%o' AND CITY NOT LIKE '%u';

Q17. Write an SQL query that reports the products that were only sold in the first quarter of 2019. That is, between 2019-01-01 and 2019-03-31 inclusive.

**ANSWER:**

SELECT p.product\_id, p.product\_name

FROM Product p

WHERE p.product\_id NOT IN (

SELECT s.product\_id

FROM Sales s

WHERE s.sale\_date BETWEEN '2019-01-01' AND '2019-03-31'

)

Q18. Write an SQL query to find all the authors that viewed at least one of their own articles. Return the result table sorted by id in ascending order.

**ANSWER:** SELECT DISTINCT v1.author\_id as id FROM Views v1 JOIN Views v2 ON v1.author\_id = v2.author\_id AND v1.viewer\_id = v1.author\_id ORDER BY id ASC;

Q19. Write an SQL query to find the percentage of immediate orders in the table, rounded to 2 decimal places.

**ANSWER:** SELECT ROUND((COUNT(CASE WHEN order\_date = customer\_pref\_delivery\_date THEN 1 END) / COUNT(\*)) \* 100, 2) AS immediate\_percentage FROM Delivery;

Q20. Write an SQL query to find the ctr of each Ad. Round ctr to two decimal points.

**ANSWER:** SELECT ad\_id, ROUND(COUNT(CASE WHEN action = 'Clicked' THEN 1 END) / COUNT(CASE WHEN action IN ('Clicked', 'Viewed') THEN 1 END) \* 100, 2) AS ctr FROM Ads WHERE action <> 'Ignored' GROUP BY ad\_id ORDER BY ctr DESC, ad\_id ASC;

Q21. Write an SQL query to find the team size of each of the employees.

**ANSWER:** SELECT e.employee\_id, t.team\_size FROM Employee e JOIN ( SELECT team\_id, COUNT(\*) AS team\_size FROM Employee GROUP BY team\_id ) t ON e.team\_id = t.team\_id;

Q22. Write an SQL query to find the type of weather in each country for November 2019.

**ANSWER:** SELECT c.country\_name,

CASE

WHEN AVG(w.weather\_state) < 15 THEN 'Cold'

WHEN AVG(w.weather\_state) >= 25 THEN 'Hot'

ELSE 'Warm'

END AS weather\_type

FROM Countries c

JOIN Weather w ON c.country\_id = w.country\_id

WHERE w.day BETWEEN '2019-11-01' AND '2019-11-30'

GROUP BY c.country\_id

ORDER BY c.country\_name;

Q23. Write an SQL query to find the average selling price for each product. average\_price should be rounded to 2 decimal places.

**ANSWER:** SELECT p.product\_id, ROUND(SUM(p.price \* u.units) / SUM(u.units), 2) AS average\_price

FROM Prices p JOIN UnitsSold u ON p.product\_id = u.product\_id AND u.purchase\_date BETWEEN p.start\_date AND p.end\_date GROUP BY p.product\_id;

Q24. Write an SQL query to report the first login date for each player.

**ANSWER:** SELECT player\_id, MIN(event\_date) AS first\_login FROM Activity GROUP BY player\_id;

Q25. Write an SQL query to report the device that is first logged in for each player.

**ANSWER:** SELECT a.player\_id, a.device\_id

FROM Activity a

JOIN (

SELECT player\_id, MIN(event\_date) AS first\_login

FROM Activity

GROUP BY player\_id

) b ON a.player\_id = b.player\_id AND a.event\_date = b.first\_login;

Q26. Write an SQL query to get the names of products that have at least 100 units ordered in February 2020 and their amount.

**ANSWER:** SELECT p.product\_name, SUM(o.unit) AS unit

FROM Products p

JOIN Orders o ON p.product\_id = o.product\_id

WHERE o.order\_date >= '2020-02-01' AND o.order\_date <= '2020-02-29'

GROUP BY p.product\_name

HAVING SUM(o.unit) >= 100;

Q27. Write an SQL query to find the users who have valid emails.

**ANSWER:** SELECT user\_id, name, mail FROM Users WHERE mail REGEXP '^[a-zA-Z][a-zA-Z0-9\_.-][\*@leetcode.com$](mailto:*@leetcode.com$)';

Q28. Write an SQL query to report the customer\_id and customer\_name of customers who have spent at least $100 in each month of June and July 2020.

**ANSWER:** SELECT c.customer\_id, c.name

FROM Customers c

INNER JOIN Orders o ON c.customer\_id = o.customer\_id

INNER JOIN Product p ON o.product\_id = p.product\_id

WHERE (o.order\_date BETWEEN '2020-06-01' AND '2020-06-30' OR o.order\_date BETWEEN '2020-07-01' AND '2020-07-31')

GROUP BY c.customer\_id, c.name

HAVING SUM(p.price \* o.quantity) >= 100

AND COUNT(DISTINCT CASE WHEN o.order\_date BETWEEN '2020-06-01' AND '2020-06-30' THEN MONTH(o.order\_date) END) = 1

AND COUNT(DISTINCT CASE WHEN o.order\_date BETWEEN '2020-07-01' AND '2020-07-31' THEN MONTH(o.order\_date) END) = 1;

Q29. Write an SQL query to report the distinct titles of the kid-friendly movies streamed in June 2020. Return the result table in any order.

**ANSWER:** SELECT DISTINCT c.title FROM TVProgram p

JOIN Content c ON p.content\_id = c.content\_id

WHERE c.Kids\_content = 'Y' AND c.content\_type = 'Movies' AND p.program\_date BETWEEN '2020-06-01' AND '2020-06-30';

Q30. Write an SQL query to find the npv of each query of the Queries table.

**ANSWER:** SELECT q.id, q.year, COALESCE(n.npv, 0) as npv FROM Queries q LEFT JOIN NPV n ON q.id = n.id AND q.year = n.year;

Q31. Write an SQL query to find the npv of each query of the Queries table.

**ANSWER:** SELECT q.id, q.year, COALESCE(n.npv, 0) AS npv FROM Queries q LEFT JOIN NPV n ON q.id = n.id AND q.year = n.year;

Q32. Write an SQL query to show the unique ID of each user, If a user does not have a unique ID replace just show null.

**ANSWER:** SELECT IFNULL(EmployeeUNI.unique\_id, 'null') as unique\_id, Employees.name FROM Employees LEFT JOIN EmployeeUNI ON Employees.id = EmployeeUNI.id;

Q33. Write an SQL query to report the distance travelled by each user.

**ANSWER:** SELECT u.name, COALESCE(SUM(r.distance), 0) AS travelled\_distance FROM Users u LEFT JOIN Rides r ON u.id = r.user\_id GROUP BY u.id, u.name ORDER BY travelled\_distance DESC, name ASC;

Q34. Write an SQL query to get the names of products that have at least 100 units ordered in February 2020 and their amount.

**ANSWER:** SELECT p.product\_name, SUM(o.unit \* p.price) as amount FROM Products p JOIN Orders o ON p.product\_id = o.product\_id WHERE o.order\_date BETWEEN '2020-02-01' AND '2020-02-29' GROUP BY p.product\_name HAVING SUM(o.unit) >= 100;

Q35. Write an SQL query to:

**ANSWER:** SELECT name AS results FROM Users u JOIN MovieRating mr ON u.user\_id = mr.user\_id

GROUP BY u.user\_id, name ORDER BY COUNT(DISTINCT movie\_id) DESC, name ASC LIMIT 1;

SELECT title AS results FROM Movies m JOIN MovieRating mr ON m.movie\_id = mr.movie\_id

WHERE created\_at BETWEEN '2020-02-01' AND '2020-02-29' GROUP BY m.movie\_id, title

ORDER BY AVG(rating) DESC, title ASC LIMIT 1;

Q36. Write an SQL query to report the distance travelled by each user.

**ANSWER:** SELECT u.name, COALESCE(SUM(r.distance), 0) AS travelled\_distance FROM Users u LEFT JOIN Rides r ON u.id = r.user\_id GROUP BY u.id, u.name ORDER BY travelled\_distance DESC, u.name ASC;

Q37. Write an SQL query to show the unique ID of each user, If a user does not have a unique ID replace just show null.

**ANSWER:** SELECT COALESCE(CAST(unique\_id AS VARCHAR), 'null') AS unique\_id, name FROM Employees LEFT JOIN EmployeeUNI ON Employees.id = EmployeeUNI.id ORDER BY name;

Q38. Write an SQL query to find the id and the name of all students who are enrolled in departments that no longer exist.

**ANSWER:** SELECT Students.id, Students.name FROM Students WHERE Students.department\_id NOT IN (SELECT id FROM Departments);

Q39. Write an SQL query to report the number of calls and the total call duration between each pair of distinct persons (person1, person2) where person1 < person2.

**ANSWER:** SELECT

CASE WHEN from\_id < to\_id THEN from\_id ELSE to\_id END AS person1,

CASE WHEN from\_id < to\_id THEN to\_id ELSE from\_id END AS person2,

COUNT(\*) AS call\_count,

SUM(duration) AS total\_duration

FROM

Calls

WHERE

from\_id != to\_id

GROUP BY

person1,

person2;

Q40. Write an SQL query to find the average selling price for each product. average\_price should be rounded to 2 decimal places.

**ANSWER:** SELECT

p.product\_id,

ROUND(SUM(p.price \* u.units) / SUM(u.units), 2) AS average\_price

FROM Prices p

JOIN UnitsSold u ON p.product\_id = u.product\_id AND u.purchase\_date BETWEEN p.start\_date AND p.end\_date

GROUP BY p.product\_id;

Q41. Write an SQL query to report the number of cubic feet of volume the inventory occupies in each warehouse.

**ANSWER:** SELECT w.name AS warehouse\_name, SUM(p.Width \* p.Length \* p.Height \* w.units) AS volume FROM Warehouse w JOIN Products p ON w.product\_id = p.product\_id GROUP BY w.name;

Q42. Write an SQL query to report the difference between the number of apples and oranges sold each day. Return the result table ordered by sale\_date.

**ANSWER:** SELECT s1.sale\_date, (SUM(CASE WHEN s1.fruit = 'apples' THEN s1.sold\_num ELSE -s1.sold\_num END)) AS diff FROM Sales s1 GROUP BY s1.sale\_date ORDER BY s1.sale\_date;

Q43. Write an SQL query to report the fraction of players that logged in again on the day after the day they first logged in, rounded to 2 decimal places. In other words, you need to count the number of players that logged in for at least two consecutive days starting from their first login date, then divide that number by the total number of players.

**ANSWER:** SELECT ROUND(COUNT(DISTINCT a1.player\_id, a1.event\_date + INTERVAL 1 DAY) / COUNT(DISTINCT a1.player\_id), 2) AS fraction FROM Activity a1 JOIN Activity a2 ON a1.player\_id = a2.player\_id AND a1.event\_date + INTERVAL 1 DAY = a2.event\_date;

Q44. Write an SQL query to report the managers with at least five direct reports.

**ANSWER:** SELECT e.name FROM Employee e JOIN Employee m ON e.managerId = m.id GROUP BY e.managerId HAVING COUNT(\*) >= 5;

Q45. Write an SQL query to report the respective department name and number of students majoring in each department for all departments in the Department table (even ones with no current students).

**ANSWER:** SELECT Department.dept\_name AS dept\_name, COUNT(Student.student\_id) AS student\_number FROM Department LEFT JOIN Student ON Department.dept\_id = Student.dept\_id

GROUP BY Department.dept\_name ORDER BY student\_number DESC, dept\_name ASC;

Q46. Write an SQL query to report the customer ids from the Customer table that bought all the products in the Product table

**ANSWER:** SELECT DISTINCT customer\_id

FROM Customer

WHERE NOT EXISTS (

SELECT product\_key

FROM Product

WHERE NOT EXISTS (

SELECT customer\_id

FROM Customer

WHERE Customer.product\_key = Product.product\_key

AND Customer.customer\_id = c.customer\_id

)

);

Q47. Write an SQL query that reports the most experienced employees in each project. In case of a tie, report all employees with the maximum number of experience years.

**ANSWER:** WITH experience\_rank AS (

SELECT project\_id, employee\_id, experience\_years,

RANK() OVER (PARTITION BY project\_id ORDER BY experience\_years DESC) AS rank

FROM Project

JOIN Employee ON Project.employee\_id = Employee.employee\_id

)

SELECT project\_id, employee\_id

FROM experience\_rank

WHERE rank = 1

ORDER BY project\_id;

Q48. Write an SQL query that reports the books that have sold less than 10 copies in the last year, excluding books that have been available for less than one month from today. Assume today is 2019-06-23.

**ANSWER:** SELECT b.book\_id, b.name, COUNT(o.quantity) AS num\_sold FROM Books b LEFT JOIN Orders o ON b.book\_id = o.book\_id WHERE b.available\_from <= DATEADD(month, -1, '2019-06-23') AND o.dispatch\_date >= DATEADD(year, -1, '2019-06-23') GROUP BY b.book\_id, b.name HAVING COUNT(o.quantity) < 10;

Q49. Write a SQL query to find the highest grade with its corresponding course for each student. In case of a tie, you should find the course with the smallest course\_id.

**ANSWER:** SELECT e.student\_id,

(SELECT course\_id

FROM Enrollments

WHERE student\_id = e.student\_id AND grade = MAX(e.grade)

ORDER BY course\_id ASC LIMIT 1) AS course\_id,

MAX(e.grade) AS grade

FROM Enrollments e

GROUP BY e.student\_id

ORDER BY e.student\_id ASC;

Q50. Write an SQL query to find the winner in each group.

**ANSWER:** SELECT

p.group\_id,

p.player\_id

FROM

Players p

JOIN Matches m ON p.player\_id IN (m.first\_player, m.second\_player)

WHERE

(p.player\_id = m.first\_player AND m.first\_score > m.second\_score)

OR (p.player\_id = m.second\_player AND m.second\_score > m.first\_score)

GROUP BY

p.group\_id,

p.player\_id

HAVING

SUM(CASE

WHEN p.player\_id = m.first\_player THEN m.first\_score

ELSE m.second\_score

END

) = (

SELECT MAX(total\_points)

FROM (

SELECT

p2.group\_id,

p2.player\_id,

SUM(CASE

WHEN p2.player\_id = m2.first\_player THEN m2.first\_score

ELSE m2.second\_score

END

) AS total\_points

FROM

Players p2

JOIN Matches m2 ON p2.player\_id IN (m2.first\_player, m2.second\_player)

WHERE

(p2.player\_id = m2.first\_player AND m2.first\_score > m2.second\_score)

OR (p2.player\_id = m2.second\_player AND m2.second\_score > m2.first\_score)

GROUP BY

p2.group\_id,

p2.player\_id

) AS temp

WHERE

temp.group\_id = p.group\_id

)

AND p.player\_id = (

SELECT MIN(player\_id)

FROM (

SELECT

p3.group\_id,

p3.player\_id,

SUM(CASE

WHEN p3.player\_id = m3.first\_player THEN m3.first\_score

ELSE m3.second\_score

END

) AS total\_points

FROM

Players p3

JOIN Matches m3 ON p3.player\_id IN (m3.first\_player, m3.second\_player)

WHERE

(p3.player\_id = m3.first\_player AND m3.first\_score > m3.second\_score)

OR (p3.player\_id = m3.second\_player AND m3.second\_score > m3.first\_score)

GROUP BY

p3.group\_id,

p3.player\_id

) AS temp2

WHERE

temp2.group\_id = p.group\_id

AND temp2.total\_points = (

SELECT MAX(total\_points)

FROM (

SELECT

p4.group\_id,

p4.player\_id,

SUM(CASE

WHEN p4.player\_id = m4.first\_player THEN m4.first\_score

ELSE m4.second\_score

END

) AS total\_points

FROM

Players p4

JOIN Matches m4 ON p4.player\_id IN (m4.first\_player, m4.second\_player)

WHERE

(p4.player\_id = m4.first\_player AND m4.first\_score > m4.second\_score)

OR (p4.player\_id = m4.second\_player AND m4.second\_score > m4.first\_score)

GROUP BY

p4.group\_id,

p4.player\_id

) AS temp3

WHERE

temp3.group\_id = temp2.group\_id

)

)

ORDER BY

group\_id,

player\_id;

Q51. Write an SQL query to report the name, population, and area of the big countries.

**ANSWER:**

SELECT name, population, area

FROM World

WHERE area >= 3000000 OR population >= 25000000;

Q52. Write an SQL query to report the names of the customer that are not referred by the customer with id = 2.

**ANSWER:**

SELECT name

FROM Customer

WHERE referee\_id IS NULL OR referee\_id <> 2;

Q53. Write an SQL query to report all customers who never order anything.

**ANSWER:**

SELECT c.name AS Customers

FROM Customers c

LEFT JOIN Orders o ON c.id = o.customerId

WHERE o.id IS NULL;

Q54. Write an SQL query to find the team size of each of the employees.

**ANSWER:**

SELECT employee\_id, (

SELECT COUNT(\*)

FROM Employee

WHERE team\_id = e.team\_id

) AS team\_size

FROM Employee e;

Q55. Write an SQL query to find the countries where this company can invest.

**ANSWER:**

SELECT c.name AS country

FROM Country c

INNER JOIN Person p ON c.country\_code = SUBSTRING(p.phone\_number, 1, 3)

INNER JOIN Calls cl ON p.id = cl.caller\_id OR p.id = cl.callee\_id

GROUP BY c.name

HAVING AVG(cl.duration) > (

SELECT AVG(duration) AS global\_avg

FROM Calls

);

Q56. Write an SQL query to report the device that is first logged in for each player.

**ANSWER:**

SELECT player\_id, device\_id

FROM Activity

WHERE (player\_id, event\_date) IN (

SELECT player\_id, MIN(event\_date) AS min\_date

FROM Activity

GROUP BY player\_id

);

Q57. Write an SQL query to find the customer\_number for the customer who has placed the largest number of orders.

**ANSWER:**

SELECT customer\_number

FROM Orders

GROUP BY customer\_number

HAVING COUNT(\*) = (

SELECT MAX(order\_count)

FROM (

SELECT COUNT(\*) AS order\_count

FROM Orders

GROUP BY customer\_number

) AS counts

);

Q58. Write an SQL query to report all the consecutive available seats in the cinema.

**ANSWER:**

SELECT seat\_id

FROM (

SELECT seat\_id, ROW\_NUMBER() OVER (ORDER BY seat\_id) AS rn, free - ROW\_NUMBER() OVER (ORDER BY seat\_id) AS grp

FROM Cinema

) AS subquery

WHERE free = 1

GROUP BY grp

HAVING COUNT(\*) > 1

ORDER BY MIN(seat\_id);

Q59. Write an SQL query to report the names of all the salespersons who did not have any orders related to the company with the name "RED".

**ANSWER:**

SELECT name

FROM SalesPerson

WHERE NOT EXISTS (

SELECT 1

FROM Orders

JOIN Company ON Orders.com\_id = Company.com\_id

WHERE Orders.sales\_id = SalesPerson.sales\_id

AND Company.name = 'RED'

);

Q60. Write an SQL query to report for every three line segments whether they can form a triangle. Return the result table in any order

**ANSWER:**

SELECT x, y, z,

CASE

WHEN x + y > z AND x + z > y AND y + z > x THEN 'Yes'

ELSE 'No'

END AS triangle

FROM Triangle;

Q61. Write an SQL query to report the shortest distance between any two points from the Point table.

**ANSWER:**

SELECT MIN(p2.x - p1.x) AS shortest

FROM Point p1

JOIN Point p2 ON p2.x > p1.x;

Q62. Write a SQL query for a report that provides the pairs (actor\_id, director\_id) where the actor has cooperated with the director at least three times.

**ANSWER:**

SELECT actor\_id, director\_id

FROM ActorDirector

GROUP BY actor\_id, director\_id

HAVING COUNT(DISTINCT timestamp) >= 3;

Q63. Write an SQL query that reports the product\_name, year, and price for each sale\_id in the Sales table. Return the resulting table in any order

**ANSWER:**

SELECT p.product\_name, s.year, s.price

FROM Sales s

JOIN Product p ON s.product\_id = p.product\_id;

Q64. Write an SQL query that reports the average experience years of all the employees for each project, rounded to 2 digits

**ANSWER:**

SELECT p.project\_id, ROUND(AVG(e.experience\_years), 2) AS average\_years

FROM Project p

JOIN Employee e ON p.employee\_id = e.employee\_id

GROUP BY p.project\_id;

Q65. Write an SQL query that reports the best seller by total sales price, If there is a tie, report them all. Return the result table in any order.

**ANSWER:**

SELECT seller\_id

FROM Sales

GROUP BY seller\_id

HAVING SUM(price) = (

SELECT SUM(price)

FROM Sales

GROUP BY seller\_id

ORDER BY SUM(price) DESC

LIMIT 1

);

Q66. Write an SQL query that reports the buyers who have bought S8 but not iPhone. Note that S8 and iPhone are products present in the Product table.

**ANSWER:**

SELECT DISTINCT buyer\_id

FROM Sales

WHERE product\_id = (

SELECT product\_id

FROM Product

WHERE product\_name = 'S8'

)

AND buyer\_id NOT IN (

SELECT buyer\_id

FROM Sales

WHERE product\_id = (

SELECT product\_id

FROM Product

WHERE product\_name = 'iPhone'

)

);

Q67. Write an SQL query to compute the moving average of how much the customer paid in a seven days window (i.e., current day + 6 days before). average\_amount should be rounded to two decimal places.

**ANSWER:**

SELECT visited\_on, amount, ROUND(AVG(amount) OVER (ORDER BY visited\_on ROWS BETWEEN 6 PRECEDING AND CURRENT ROW), 2) AS average\_amount

FROM Customer

ORDER BY visited\_on;

Q68. Write an SQL query to find the total score for each gender on each day.

**ANSWER:**

SELECT gender, day, SUM(score\_points) OVER (PARTITION BY gender, day ORDER BY gender, day) AS total

FROM Scores

ORDER BY gender, day;

Q69. Write an SQL query to find the start and end number of continuous ranges in the table Logs

**ANSWER:**

SELECT MIN(log\_id) AS start\_id, MAX(log\_id) AS end\_id

FROM (

SELECT log\_id, log\_id - ROW\_NUMBER() OVER (ORDER BY log\_id) AS grp

FROM Logs

) AS t

GROUP BY grp

ORDER BY start\_id;

Q70. Write an SQL query to find the number of times each student attended each exam.

**ANSWER:**

SELECT s.student\_id, s.student\_name, sb.subject\_name, COUNT(e.subject\_name) AS attended\_exams

FROM Students s

CROSS JOIN Subjects sb

LEFT JOIN Examinations e ON s.student\_id = e.student\_id AND sb.subject\_name = e.subject\_name

GROUP BY s.student\_id, s.student\_name, sb.subject\_name

ORDER BY s.student\_id, sb.subject\_name;

Q71. Write an SQL query to find employee\_id of all employees that directly or indirectly report their work to the head of the company

**ANSWER:**

WITH RECURSIVE EmployeeHierarchy AS (

SELECT employee\_id, manager\_id

FROM Employees

WHERE employee\_id = 1

UNION ALL

SELECT e.employee\_id, e.manager\_id

FROM Employees e

INNER JOIN EmployeeHierarchy eh ON e.manager\_id = eh.employee\_id

)

SELECT DISTINCT employee\_id

FROM EmployeeHierarchy;

Q72. Write an SQL query to find for each month and country, the number of transactions and their total amount, the number of approved transactions and their total amount.

**ANSWER:**

SELECT

DATE\_FORMAT(trans\_date, '%Y-%m') AS month,

country,

COUNT(\*) AS trans\_count,

SUM(CASE WHEN state = 'approved' THEN 1 ELSE 0 END) AS approved\_count,

SUM(amount) AS trans\_total\_amount,

SUM(CASE WHEN state = 'approved' THEN amount ELSE 0 END) AS approved\_total\_amount

FROM Transactions

GROUP BY month, country;

Q73. Write an SQL query to find the average daily percentage of posts that got removed after being reported as spam, rounded to 2 decimal places.

**ANSWER:**

SELECT

ROUND(AVG(removed\_posts) \* 100, 2) AS average\_daily\_percent

FROM (

SELECT

action\_date,

COUNT(DISTINCT post\_id) AS total\_posts,

SUM(CASE WHEN action = 'report' AND extra = 'spam' THEN 1 ELSE 0 END) AS removed\_posts

FROM Actions

WHERE action\_date IN (SELECT DISTINCT remove\_date FROM Removals)

GROUP BY action\_date

) AS subquery;

Q74. Write an SQL query to report the fraction of players that logged in again on the day after the day they first logged in, rounded to 2 decimal places. In other words, you need to count the number of players that logged in for at least two consecutive days starting from their first login date, then divide that number by the total number of players

**ANSWER:**

SELECT

ROUND(COUNT(DISTINCT a1.player\_id) / COUNT(DISTINCT a2.player\_id), 2) AS fraction

FROM Activity a1

JOIN Activity a2 ON a1.player\_id = a2.player\_id

WHERE a2.event\_date = DATE\_ADD(a1.event\_date, INTERVAL 1 DAY);

Q75. Write an SQL query to report the fraction of players that logged in again on the day after the day they first logged in, rounded to 2 decimal places. In other words, you need to count the number of players that logged in for at least two consecutive days starting from their first login date, then divide that number by the total number of players.

**ANSWER:**

SELECT

ROUND(

COUNT(DISTINCT CASE WHEN a2.event\_date = DATE\_ADD(a1.event\_date, INTERVAL 1 DAY) THEN a1.player\_id END) / COUNT(DISTINCT a1.player\_id),

2

) AS fraction

FROM

Activity a1

JOIN Activity a2 ON a1.player\_id = a2.player\_id

WHERE

a1.event\_date = (SELECT MIN(event\_date) FROM Activity WHERE player\_id = a1.player\_id);

Q76. Write an SQL query to find the salaries of the employees after applying taxes. Round the salary to the nearest integer.

**ANSWER:**

SELECT

company\_id,

employee\_id,

employee\_name,

ROUND(

salary - (CASE

WHEN max\_salary < 1000 THEN 0

WHEN max\_salary >= 1000 AND max\_salary <= 10000 THEN 0.24 \* salary

WHEN max\_salary > 10000 THEN 0.49 \* salary

END),

0

) AS salary

FROM

Salaries

JOIN

(SELECT

company\_id,

MAX(salary) AS max\_salary

FROM

Salaries

GROUP BY

company\_id) AS max\_salaries ON Salaries.company\_id = max\_salaries.company\_id

ORDER BY

company\_id,

employee\_id;

Q77. Write an SQL query to evaluate the boolean expressions in Expressions table

**ANSWER:**

SELECT

e.left\_operand,

e.operator,

e.right\_operand,

CASE

WHEN e.operator = '<' THEN v1.value < v2.value

WHEN e.operator = '>' THEN v1.value > v2.value

WHEN e.operator = '=' THEN v1.value = v2.value

END AS value

FROM

Expressions e

JOIN

Variables v1 ON e.left\_operand = v1.name

JOIN

Variables v2 ON e.right\_operand = v2.name;

Q78. Write an SQL query to find the countries where this company can invest.

**ANSWER:**

SELECT c.name AS country

FROM Country c

JOIN Person p ON c.country\_code = SUBSTRING(p.phone\_number, 1, 3)

JOIN Calls cl ON p.id = cl.caller\_id OR p.id = cl.callee\_id

GROUP BY c.name

HAVING AVG(cl.duration) > (

SELECT (2 \* SUM(duration)) / COUNT(\*) AS global\_avg

FROM Calls

);

Q79. Write a query that prints a list of employee names (i.e.: the name attribute) from the Employee table in alphabetical order.

**ANSWER:**

SELECT name

FROM Employee

ORDER BY name;

Q80. Write a query to obtain the year-on-year growth rate for the total spend of each product for each year.

**ANSWER:**

SELECT

EXTRACT(YEAR FROM t.transaction\_date) AS year,

t.product\_id,

SUM(t.spend) AS curr\_year\_spend,

LAG(SUM(t.spend)) OVER (PARTITION BY t.product\_id ORDER BY EXTRACT(YEAR FROM t.transaction\_date)) AS prev\_year\_spend,

ROUND(((SUM(t.spend) - LAG(SUM(t.spend)) OVER (PARTITION BY t.product\_id ORDER BY EXTRACT(YEAR FROM t.transaction\_date))) / LAG(SUM(t.spend)) OVER (PARTITION BY t.product\_id ORDER BY EXTRACT(YEAR FROM t.transaction\_date))) \* 100, 2) AS yoy\_rate

FROM user\_transactions t

GROUP BY EXTRACT(YEAR FROM t.transaction\_date), t.product\_id

ORDER BY t.product\_id, year;

Q81. Write a SQL query to find the number of prime and non-prime items that can be stored in the 500,000 square feet warehouse. Output the item type and number of items to be stocked.

**ANSWER:**

SELECT item\_type, COUNT(\*) AS item\_count

FROM inventory

GROUP BY item\_type

ORDER BY item\_type;

Q82. Assume you have the table below containing information on Facebook user actions. Write a query to obtain the active user retention in July 2022. Output the month (in numerical format 1, 2, 3) and the number of monthly active users (MAUs).

**ANSWER:**

SELECT EXTRACT(MONTH FROM event\_date) AS month, COUNT(DISTINCT user\_id) AS monthly\_active\_users

FROM user\_actions

WHERE EXTRACT(MONTH FROM event\_date) = 7

AND EXISTS (

SELECT 1

FROM user\_actions ua

WHERE ua.user\_id = user\_actions.user\_id

AND EXTRACT(MONTH FROM ua.event\_date) = 6

AND ua.event\_type IN ('sign-in', 'like', 'comment')

)

GROUP BY month;

Q83. Write a query to report the median of searches made by a user. Round the median to one decimal point.

**ANSWER:**

WITH cte AS (

SELECT searches, num\_users, SUM(num\_users) OVER (ORDER BY searches) AS cumulative\_users

FROM search\_frequency

)

SELECT ROUND(

AVG(searches)::numeric,

1

) AS median

FROM (

SELECT searches, cumulative\_users,

ROW\_NUMBER() OVER (ORDER BY cumulative\_users DESC) AS row\_num,

COUNT(\*) OVER () AS total\_rows

FROM cte

) sub

WHERE row\_num >= (total\_rows + 1) / 2

ORDER BY row\_num

LIMIT 1;

Q84. Write a query to update the Facebook advertiser's status using the daily\_pay table. Advertiser is a two-column table containing the user id and their payment status based on the last payment and daily\_pay table has current information about their payment. Only advertisers who paid will show up in this table.

**ANSWER:**

UPDATE advertiser

SET status = CASE

WHEN daily\_pay.paid IS NULL THEN 'CHURN'

WHEN advertiser.status = 'NEW' AND daily\_pay.paid IS NOT NULL THEN 'EXISTING'

WHEN advertiser.status = 'EXISTING' AND daily\_pay.paid IS NOT NULL THEN 'EXISTING'

WHEN advertiser.status = 'CHURN' AND daily\_pay.paid IS NOT NULL THEN 'RESURRECT'

ELSE advertiser.status

END

FROM daily\_pay

WHERE advertiser.user\_id = daily\_pay.user\_id;

Q85. Write a query that calculates the total time that the fleet of servers was running. The output should be in units of full days.

**ANSWER:**

SELECT FLOOR(SUM(uptime\_seconds) / (60 \* 60 \* 24)) AS total\_uptime\_days

FROM (

SELECT server\_id,

MIN(status\_time) AS start\_time,

MAX(status\_time) AS stop\_time,

(MAX(status\_time) - MIN(status\_time)) AS uptime\_seconds

FROM (

SELECT server\_id,

status\_time,

session\_status,

ROW\_NUMBER() OVER (PARTITION BY server\_id ORDER BY status\_time) AS rn

FROM server\_utilization

) AS t

WHERE session\_status = 'start' OR session\_status = 'stop'

GROUP BY server\_id, rn

HAVING COUNT(\*) = 2

) AS s;

Q86. Write a query.

**ANSWER:**

SELECT COUNT(\*) AS payment\_count

FROM (

SELECT transaction\_id, merchant\_id, credit\_card\_id, amount, transaction\_timestamp,

ROW\_NUMBER() OVER (PARTITION BY merchant\_id, credit\_card\_id, amount

ORDER BY transaction\_timestamp) AS rn

FROM transactions

) AS t

WHERE rn > 1

AND transaction\_timestamp - LAG(transaction\_timestamp) OVER (PARTITION BY merchant\_id, credit\_card\_id, amount

ORDER BY transaction\_timestamp) <= INTERVAL '10 minutes';

Q87. Write a query to find the bad experience rate in the first 14 days for new users who signed up in June 2022. Output the percentage of bad experience rounded to 2 decimal places.

**ANSWER:**

WITH user\_orders AS (

SELECT o.customer\_id, o.order\_timestamp,

CASE

WHEN o.status = 'completed incorrectly' THEN 'bad\_experience'

WHEN o.status = 'never received' THEN 'bad\_experience'

WHEN o.status = 'completed successfully' AND t.actual\_delivery\_timestamp > t.estimated\_delivery\_timestamp + INTERVAL '30 minutes' THEN 'bad\_experience'

ELSE 'good\_experience'

END AS experience

FROM orders o

JOIN trips t ON o.trip\_id = t.trip\_id

JOIN customers c ON o.customer\_id = c.customer\_id

WHERE c.signup\_timestamp >= '2022-06-01' AND c.signup\_timestamp < '2022-07-01'

AND o.order\_timestamp >= c.signup\_timestamp

)

SELECT ROUND(100.0 \* COUNT(CASE WHEN experience = 'bad\_experience' THEN 1 END) / COUNT(\*), 2) AS bad\_experience\_pct

FROM user\_orders

WHERE order\_timestamp <= signup\_timestamp + INTERVAL '14 days';

Q88. Write an SQL query to find the total score for each gender on each day.

**ANSWER:**

SELECT gender, day, SUM(score\_points) AS total

FROM Scores

GROUP BY gender, day

ORDER BY gender ASC, day ASC;

Q89. Write a query.

**ANSWER:**

SELECT c.name AS country

FROM Country c

JOIN Person p ON c.country\_code = SUBSTRING(p.phone\_number, 1, 3)

JOIN Calls cl ON cl.caller\_id = p.id OR cl.callee\_id = p.id

GROUP BY c.name

HAVING AVG(cl.duration) > (

SELECT (2 \* SUM(duration)) / COUNT(\*) AS global\_avg

FROM Calls

)

ORDER BY c.name;

Q90. Write an SQL query to report the median of all the numbers in the database after decompressing the Numbers table. Round the median to one decimal point.

**ANSWER:**

WITH Decompressed AS (

SELECT num

FROM Numbers

WHERE frequency > 0

UNION ALL

SELECT num

FROM Numbers

JOIN Decompressed ON Decompressed.num = Numbers.num

WHERE frequency > 1

)

SELECT ROUND(AVG(num), 1) AS median

FROM (

SELECT num, ROW\_NUMBER() OVER (ORDER BY num) AS row\_num, COUNT(\*) OVER () AS total\_count

FROM Decompressed

) AS T

WHERE row\_num IN (FLOOR((total\_count + 1) / 2), CEIL((total\_count + 1) / 2));

Q91. Write an SQL query to report the comparison result (higher/lower/same) of the average salary of employees in a department to the company's average salary.

**ANSWER:**

WITH DepartmentAvg AS (

SELECT

e.department\_id,

AVG(s.amount) AS avg\_salary

FROM Employee e

JOIN Salary s ON e.employee\_id = s.employee\_id

GROUP BY e.department\_id

),

CompanyAvg AS (

SELECT AVG(amount) AS avg\_salary

FROM Salary

)

SELECT

DATE\_FORMAT(s.pay\_date, '%Y-%m') AS pay\_month,

s.department\_id,

CASE

WHEN da.avg\_salary > ca.avg\_salary THEN 'higher'

WHEN da.avg\_salary < ca.avg\_salary THEN 'lower'

ELSE 'same'

END AS comparison

FROM Salary s

JOIN DepartmentAvg da ON s.department\_id = da.department\_id

CROSS JOIN CompanyAvg ca

GROUP BY pay\_month, s.department\_id;

Q92. Write an SQL query to report for each install date, the number of players that installed the game on that day, and the day one retention.

**ANSWER:**

WITH InstallDates AS (

SELECT DISTINCT

player\_id,

MIN(event\_date) AS install\_dt

FROM Activity

GROUP BY player\_id

),

DayOneRetention AS (

SELECT

i.install\_dt,

COUNT(\*) AS installs,

ROUND(COUNT(DISTINCT a.player\_id) / COUNT(\*), 2) AS Day1\_retention

FROM InstallDates i

LEFT JOIN Activity a ON i.player\_id = a.player\_id AND DATE\_ADD(i.install\_dt, INTERVAL 1 DAY) = a.event\_date

GROUP BY i.install\_dt

)

SELECT

install\_dt,

installs,

COALESCE(Day1\_retention, 0) AS Day1\_retention

FROM DayOneRetention;

Q93. Write an SQL query to find the winner in each group.

**ANSWER:**

WITH GroupScores AS (

SELECT

p.group\_id,

m.first\_player AS player\_id,

SUM(m.first\_score) AS total\_score

FROM Players p

JOIN Matches m ON p.player\_id = m.first\_player

GROUP BY p.group\_id, m.first\_player

UNION ALL

SELECT

p.group\_id,

m.second\_player AS player\_id,

SUM(m.second\_score) AS total\_score

FROM Players p

JOIN Matches m ON p.player\_id = m.second\_player

GROUP BY p.group\_id, m.second\_player

),

GroupWinners AS (

SELECT

group\_id,

player\_id,

total\_score,

ROW\_NUMBER() OVER (PARTITION BY group\_id ORDER BY total\_score DESC, player\_id ASC) AS rank

FROM GroupScores

)

SELECT group\_id, player\_id

FROM GroupWinners

WHERE rank = 1;

Q94. Write an SQL query to report the students (student\_id, student\_name) being quiet in all exams. Do not return the student who has never taken any exam.

**ANSWER:**

SELECT s.student\_id, s.student\_name

FROM Student s

WHERE s.student\_id NOT IN (

SELECT DISTINCT e.student\_id

FROM Exam e

WHERE e.score = (

SELECT MAX(score)

FROM Exam

WHERE exam\_id = e.exam\_id

) OR e.score = (

SELECT MIN(score)

FROM Exam

WHERE exam\_id = e.exam\_id

)

)

AND EXISTS (

SELECT \*

FROM Exam e

WHERE e.student\_id = s.student\_id

)

ORDER BY s.student\_id;

Q95. Write an SQL query to report the students (student\_id, student\_name) being quiet in all exams. Do not return the student who has never taken any exam.

**ANSWER:**

SELECT s.student\_id, s.student\_name

FROM Student s

WHERE s.student\_id NOT IN (

SELECT e.student\_id

FROM Exam e

WHERE e.student\_id IS NOT NULL

GROUP BY e.student\_id

HAVING MAX(e.score) = MAX(MIN(e.score)) AND MIN(e.score) = MAX(MIN(e.score))

)

AND EXISTS (

SELECT \*

FROM Exam e

WHERE e.student\_id = s.student\_id

)

ORDER BY s.student\_id;

Q96. Write a query to output the user id, song id, and cumulative count of song plays as of 4 August 2022 sorted in descending order.

**ANSWER:**

SELECT

COALESCE(h.user\_id, w.user\_id) AS user\_id,

COALESCE(h.song\_id, w.song\_id) AS song\_id,

COALESCE(h.song\_plays, 0) + COUNT(w.song\_id) AS song\_plays

FROM songs\_weekly w

LEFT JOIN songs\_history h ON w.user\_id = h.user\_id AND w.song\_id = h.song\_id

WHERE w.listen\_time <= '2022-08-04 23:59:59'

GROUP BY COALESCE(h.user\_id, w.user\_id), COALESCE(h.song\_id, w.song\_id)

ORDER BY song\_plays DESC;

Q97. Write a query to find the confirmation rate of users who confirmed their signups with text messages. Round the result to 2 decimal places.

**ANSWER:**

SELECT

ROUND(

COUNT(DISTINCT t.email\_id) / NULLIF(COUNT(DISTINCT e.email\_id), 0),

2

) AS confirm\_rate

FROM emails e

JOIN texts t ON e.email\_id = t.email\_id

WHERE t.signup\_action = 'Confirmed';

Q98. Write a query.

**ANSWER:**

SELECT

user\_id,

tweet\_date,

ROUND(

(SELECT COUNT(\*) FROM tweets t2 WHERE t2.user\_id = t1.user\_id AND t2.tweet\_date BETWEEN DATE\_SUB(t1.tweet\_date, INTERVAL 2 DAY) AND t1.tweet\_date) / 3.0,

2

) AS rolling\_avg\_3days

FROM tweets t1

ORDER BY user\_id, tweet\_date;

Q99. Write a query.

**ANSWER:**

SELECT

ab.age\_bucket,

ROUND((SUM(CASE WHEN a.activity\_type = 'send' THEN a.time\_spent ELSE 0 END) / (SUM(a.time\_spent) + 0.001)) \* 100.0, 2) AS send\_perc,

ROUND((SUM(CASE WHEN a.activity\_type = 'open' THEN a.time\_spent ELSE 0 END) / (SUM(a.time\_spent) + 0.001)) \* 100.0, 2) AS open\_perc

FROM

activities a

JOIN age\_breakdown ab ON a.user\_id = ab.user\_id

GROUP BY

ab.age\_bucket;

Q100. Write a query.

**ANSWER:**

SELECT pp.profile\_id

FROM personal\_profiles pp

JOIN (

SELECT ec.personal\_profile\_id, MAX(cp.followers) AS max\_followers

FROM employee\_company ec

JOIN company\_pages cp ON ec.company\_id = cp.company\_id

GROUP BY ec.personal\_profile\_id

) AS max\_followers ON pp.profile\_id = max\_followers.personal\_profile\_id

WHERE pp.followers > max\_followers.max\_followers

ORDER BY pp.profile\_id ASC;

Q101. Write an SQL query to show the second most recent activity of each user.

**ANSWER:**

SELECT ua.username, ua.activity, ua.startDate, ua.endDate

FROM UserActivity ua

WHERE ua.endDate = (

SELECT MAX(ua2.endDate)

FROM UserActivity ua2

WHERE ua2.username = ua.username AND ua2.endDate < (

SELECT MAX(ua3.endDate)

FROM UserActivity ua3

WHERE ua3.username = ua.username

)

);

Q102. Write an SQL query to show the second most recent activity of each user.

**ANSWER:**

SELECT ua.username, ua.activity, ua.startDate, ua.endDate

FROM UserActivity ua

WHERE ua.endDate = (

SELECT MAX(ua2.endDate)

FROM UserActivity ua2

WHERE ua2.username = ua.username AND ua2.endDate < (

SELECT MAX(ua3.endDate)

FROM UserActivity ua3

WHERE ua3.username = ua2.username

)

);

Q103. Write a query.

**ANSWER:**

SELECT Name

FROM STUDENTS

WHERE Marks > 75

ORDER BY RIGHT(Name, 3), ID ASC;

Q104. Write a query that prints a list of employee names (i.e.: the name attribute) for employees in Employee having a salary greater than $2000 per month who have been employees for less than 10 months. Sort your result by ascending employee\_id.

**ANSWER:**

SELECT name

FROM Employee

WHERE salary > 2000 AND months < 10

ORDER BY employee\_id ASC;

Q105. Write a query identifying the type of each record in the TRIANGLES table using its three side lengths.

**ANSWER:**

SELECT

CASE

WHEN A + B <= C OR A + C <= B OR B + C <= A THEN 'Not A Triangle'

WHEN A = B AND B = C THEN 'Equilateral'

WHEN A = B OR A = C OR B = C THEN 'Isosceles'

ELSE 'Scalene'

END AS triangle\_type

FROM TRIANGLES;

Q106. Write a query calculating the amount of error (i.e.: actual - miscalculated average monthly salaries), and round it up to the next integer.

**ANSWER:**

SELECT CEIL(AVG(salary) - AVG(REPLACE(salary, '0', ''))) AS error

FROM EMPLOYEES;

Q107. Write a query to find the maximum total earnings for all employees as well as the total number of employees who have maximum total earnings. Then print these values as 2 space-separated integers.

**ANSWER:**

SELECT MAX(months \* salary) AS max\_earnings, COUNT(\*) AS employee\_count

FROM Employee

WHERE months \* salary = (SELECT MAX(months \* salary) FROM Employee);

Q108. Write a query.

**ANSWER:**

SELECT CONCAT('There are a total of ', COUNT(\*), ' ', LOWER(occupation), 's.')

FROM OCCUPATIONS

GROUP BY occupation

ORDER BY COUNT(\*) ASC, LOWER(occupation) ASC;

Q109. Write a query.

**ANSWER:**

SELECT

MAX(CASE WHEN Occupation = 'Doctor' THEN Name END) AS Doctor,

MAX(CASE WHEN Occupation = 'Professor' THEN Name END) AS Professor,

MAX(CASE WHEN Occupation = 'Singer' THEN Name END) AS Singer,

MAX(CASE WHEN Occupation = 'Actor' THEN Name END) AS Actor

FROM OCCUPATIONS

GROUP BY Name

ORDER BY Name;

Q110. Write a query to find the node type of Binary Tree ordered by the value of the node.

**ANSWER:**

SELECT

N,

CASE

WHEN P IS NULL THEN 'Root'

WHEN N NOT IN (SELECT DISTINCT P FROM BST) THEN 'Leaf'

ELSE 'Inner'

END AS NodeType

FROM BST

ORDER BY N;

Q111. Write a query.

**ANSWER:**

SELECT

c.company\_code,

c.founder,

COUNT(DISTINCT lm.lead\_manager\_code) AS total\_lead\_managers,

COUNT(DISTINCT sm.senior\_manager\_code) AS total\_senior\_managers,

COUNT(DISTINCT m.manager\_code) AS total\_managers,

COUNT(DISTINCT e.employee\_code) AS total\_employees

FROM Company AS c

LEFT JOIN Lead\_Manager AS lm ON c.company\_code = lm.company\_code

LEFT JOIN Senior\_Manager AS sm ON lm.lead\_manager\_code = sm.lead\_manager\_code

LEFT JOIN Manager AS m ON sm.senior\_manager\_code = m.senior\_manager\_code

LEFT JOIN Employee AS e ON m.manager\_code = e.manager\_code

GROUP BY c.company\_code, c.founder

ORDER BY c.company\_code;

Q112. Write a query to print all prime numbers less than or equal to 1000. Print your result on a single line, and use the ampersand () character as your separator (instead of a space).

**ANSWER:**

SELECT LISTAGG(L, '&') WITHIN GROUP (ORDER BY L) AS prime\_numbers

FROM (

SELECT LEVEL AS L

FROM DUAL

CONNECT BY LEVEL <= 1000

)

WHERE L <= (

SELECT M

FROM (

SELECT LEVEL AS M

FROM DUAL

CONNECT BY LEVEL <= 1000

)

)

GROUP BY L

HAVING COUNT(CASE WHEN L/M = TRUNC(L/M) THEN 'Y' END) = 2;

Q113. Write a query.

**ANSWER:**

SELECT REPLACE(SYS\_CONNECT\_BY\_PATH(NULL, '\* '), ' ', '') AS pattern

FROM DUAL

CONNECT BY LEVEL <= 20

START WITH LEVEL = 1;

Q114. Write a query.

**ANSWER:**

SELECT REPLACE(SYS\_CONNECT\_BY\_PATH('\*', ' '), ' ', '')

FROM (

SELECT LEVEL AS L

FROM DUAL

CONNECT BY LEVEL <= 20

)

START WITH L = 20

CONNECT BY PRIOR L - 1 = L;

Q115. Write a query.

**ANSWER:**

SELECT Name

FROM STUDENTS

WHERE Marks > 75

ORDER BY SUBSTR(Name, -3), ID ASC;

Q116. Write a query.

**ANSWER:**

SELECT name

FROM Employee

ORDER BY name ASC;

Q117. Write a query that prints a list of employee names (i.e.: the name attribute) for employees in Employee having a salary greater than $2000 per month who have been employees for less than 10 months. Sort your result by ascending employee\_id.

**ANSWER:**

SELECT name

FROM Employee

WHERE salary > 2000 AND months < 10

ORDER BY employee\_id ASC;

Q118. Write a query identifying the type of each record in the TRIANGLES table using its three side lengths.

**ANSWER:**

SELECT

CASE

WHEN A + B <= C OR A + C <= B OR B + C <= A THEN 'Not A Triangle'

WHEN A = B AND B = C THEN 'Equilateral'

WHEN A = B OR B = C OR A = C THEN 'Isosceles'

ELSE 'Scalene'

END AS TriangleType

FROM TRIANGLES;

Q119. Write a query.

**ANSWER:**

SELECT

EXTRACT(YEAR FROM t.transaction\_date) AS year,

t.product\_id,

SUM(t.spend) AS curr\_year\_spend,

LAG(SUM(t.spend)) OVER (PARTITION BY t.product\_id ORDER BY EXTRACT(YEAR FROM t.transaction\_date)) AS prev\_year\_spend,

ROUND((SUM(t.spend) - LAG(SUM(t.spend)) OVER (PARTITION BY t.product\_id ORDER BY EXTRACT(YEAR FROM t.transaction\_date))) / LAG(SUM(t.spend)) OVER (PARTITION BY t.product\_id ORDER BY EXTRACT(YEAR FROM t.transaction\_date)) \* 100, 2) AS yoy\_rate

FROM user\_transactions t

GROUP BY EXTRACT(YEAR FROM t.transaction\_date), t.product\_id

ORDER BY t.product\_id, year;

Q120. Write a query.

**ANSWER:**

SELECT

item\_type,

COUNT(\*) AS item\_count

FROM (

SELECT

item\_type,

SUM(square\_footage) AS total\_square\_footage

FROM inventory

GROUP BY item\_type

ORDER BY item\_type = 'prime\_eligible' DESC, total\_square\_footage DESC

) t

WHERE total\_square\_footage <= 500000

GROUP BY item\_type;

Q121. Write a query.

**ANSWER:**

SELECT

MONTH(ua1.event\_date) AS month,

COUNT(DISTINCT ua1.user\_id) AS monthly\_active\_users

FROM user\_actions ua1

WHERE ua1.event\_date >= '2022-07-01' AND ua1.event\_date < '2022-08-01'

AND EXISTS (

SELECT 1

FROM user\_actions ua2

WHERE ua2.user\_id = ua1.user\_id

AND ua2.event\_date >= '2022-06-01' AND ua2.event\_date < '2022-07-01'

AND ua2.event\_type IN ('sign-in', 'like', 'comment')

)

GROUP BY MONTH(ua1.event\_date);

Q122. Write a query to report the median of searches made by a user. Round the median to one decimal point.

**ANSWER:**

WITH cte AS (

SELECT searches, num\_users,

SUM(num\_users) OVER (ORDER BY searches) AS cumulative\_users,

SUM(num\_users) OVER () AS total\_users

FROM search\_frequency

)

SELECT searches AS median

FROM cte

WHERE cumulative\_users >= total\_users / 2.0

ORDER BY searches

LIMIT 1;

Q123. Write a query to update the Facebook advertiser's status using the daily\_pay table. Advertiser is a two-column table containing the user id and their payment status based on the last payment and daily\_pay table has current information about their payment. Only advertisers who paid will show up in this table.

**ANSWER:**

UPDATE advertiser

SET status =

CASE

WHEN daily\_pay.paid IS NULL THEN 'CHURN'

WHEN advertiser.status = 'NEW' AND daily\_pay.paid IS NOT NULL THEN 'EXISTING'

WHEN advertiser.status = 'EXISTING' AND daily\_pay.paid IS NOT NULL THEN 'EXISTING'

WHEN advertiser.status = 'CHURN' AND daily\_pay.paid IS NOT NULL THEN 'RESURRECT'

END

FROM daily\_pay

WHERE advertiser.user\_id = daily\_pay.user\_id;

Q124. Write a query that calculates the total time that the fleet of servers was running. The output should be in units of full days.

**ANSWER:**

SELECT FLOOR(SUM(DATEDIFF(stop\_time, start\_time))) AS total\_uptime\_days

FROM (

SELECT server\_id, MIN(status\_time) AS start\_time, MAX(status\_time) AS stop\_time

FROM server\_utilization

WHERE session\_status IN ('start', 'stop')

GROUP BY server\_id, DATE(status\_time)

) AS server\_uptime;

Q125. Write a query.

**ANSWER:**

SELECT COUNT(\*) AS payment\_count

FROM (

SELECT transaction\_id, merchant\_id, credit\_card\_id, amount, transaction\_timestamp,

LAG(transaction\_timestamp) OVER (PARTITION BY merchant\_id, credit\_card\_id, amount ORDER BY transaction\_timestamp) AS prev\_timestamp

FROM transactions

) AS t

WHERE TIMESTAMPDIFF(MINUTE, prev\_timestamp, transaction\_timestamp) <= 10;

Q126. Write a query to find the bad experience rate in the first 14 days for new users who signed up in June 2022. Output the percentage of bad experience rounded to 2 decimal places.

**ANSWER:**

SELECT ROUND((COUNT(bad\_orders.order\_id) / COUNT(new\_orders.order\_id)) \* 100, 2) AS bad\_experience\_pct

FROM (

SELECT o.order\_id

FROM orders o

JOIN customers c ON o.customer\_id = c.customer\_id

JOIN trips t ON o.trip\_id = t.trip\_id

WHERE c.signup\_timestamp >= '2022-06-01' AND c.signup\_timestamp < '2022-07-01'

AND o.order\_timestamp >= c.signup\_timestamp AND o.order\_timestamp < DATEADD('day', 14, c.signup\_timestamp)

AND (

o.status = 'completed incorrectly'

OR o.status = 'never received'

OR (o.status = 'completed successfully' AND t.actual\_delivery\_timestamp > DATEADD('minute', 30, o.order\_timestamp))

)

) AS bad\_orders

JOIN (

SELECT o.order\_id

FROM orders o

JOIN customers c ON o.customer\_id = c.customer\_id

WHERE c.signup\_timestamp >= '2022-06-01' AND c.signup\_timestamp < '2022-07-01'

AND o.order\_timestamp >= c.signup\_timestamp AND o.order\_timestamp < DATEADD('day', 14, c.signup\_timestamp)

) AS new\_orders ON bad\_orders.order\_id = new\_orders.order\_id;

Q127. Write an SQL query to find the total score for each gender on each day.

**ANSWER:**

SELECT gender, day, SUM(score\_points) AS total

FROM Scores

GROUP BY gender, day

ORDER BY gender ASC, day ASC;

Q128. Write a query.

**ANSWER:**

WITH global\_avg AS (

SELECT AVG(duration) AS global\_average

FROM Calls

)

SELECT c.name AS country

FROM Country c

JOIN Person p ON SUBSTRING(p.phone\_number, 1, 3) = c.country\_code

JOIN Calls ca ON p.id = ca.caller\_id OR p.id = ca.callee\_id

GROUP BY c.name

HAVING AVG(ca.duration) > (SELECT global\_average FROM global\_avg);

Q129. Write an SQL query to report the median of all the numbers in the database after decompressing the Numbers table. Round the median to one decimal point.

**ANSWER:**

WITH decompressed\_numbers AS (

SELECT num

FROM Numbers

WHERE frequency > 0

ORDER BY num

OFFSET (SELECT SUM(frequency) FROM Numbers WHERE frequency > 0) / 2 ROWS

FETCH NEXT CASE

WHEN (SELECT SUM(frequency) FROM Numbers WHERE frequency > 0) % 2 = 0 THEN 2

ELSE 1

END ROWS ONLY

)

SELECT ROUND(AVG(num), 1) AS median

FROM decompressed\_numbers;

Q130. Write a query.

**ANSWER:**

WITH company\_avg AS (

SELECT AVG(amount) AS avg\_salary

FROM Salary

),

department\_avg AS (

SELECT department\_id, AVG(amount) AS avg\_salary

FROM Salary s

JOIN Employee e ON s.employee\_id = e.employee\_id

GROUP BY department\_id

)

SELECT to\_char(s.pay\_date, 'YYYY-MM') AS pay\_month, d.department\_id,

CASE

WHEN d.avg\_salary > c.avg\_salary THEN 'higher'

WHEN d.avg\_salary < c.avg\_salary THEN 'lower'

ELSE 'same'

END AS comparison

FROM department\_avg d, company\_avg c;

Q131. Write an SQL query to report for each install date, the number of players that installed the game on that day, and the day one retention.

**ANSWER:**

WITH installs AS (

SELECT DATE\_TRUNC('day', event\_date) AS install\_dt, COUNT(DISTINCT player\_id) AS installs\_count

FROM Activity

GROUP BY install\_dt

),

retention AS (

SELECT a.install\_dt, COUNT(DISTINCT b.player\_id) AS retention\_count

FROM Activity a

LEFT JOIN Activity b ON a.player\_id = b.player\_id AND b.event\_date = a.event\_date + INTERVAL '1 day'

GROUP BY a.install\_dt

)

SELECT i.install\_dt, i.installs\_count, ROUND(r.retention\_count::numeric / i.installs\_count, 2) AS day1\_retention

FROM installs i

LEFT JOIN retention r ON i.install\_dt = r.install\_dt;

Q132. Write an SQL query to find the winner in each group.

**ANSWER:**

WITH group\_scores AS (

SELECT p.group\_id, m.first\_player AS player\_id, SUM(m.first\_score) AS total\_score

FROM Players p

JOIN Matches m ON p.player\_id = m.first\_player

GROUP BY p.group\_id, m.first\_player

UNION ALL

SELECT p.group\_id, m.second\_player AS player\_id, SUM(m.second\_score) AS total\_score

FROM Players p

JOIN Matches m ON p.player\_id = m.second\_player

GROUP BY p.group\_id, m.second\_player

),

group\_ranks AS (

SELECT group\_id, player\_id, total\_score,

ROW\_NUMBER() OVER (PARTITION BY group\_id ORDER BY total\_score DESC, player\_id ASC) AS rank

FROM group\_scores

)

SELECT group\_id, player\_id

FROM group\_ranks

WHERE rank = 1;

Q133. Write an SQL query to report the students (student\_id, student\_name) being quiet in all exams.

**ANSWER:**

SELECT s.student\_id, s.student\_name

FROM Student s

WHERE s.student\_id NOT IN (

SELECT e.student\_id

FROM Exam e

WHERE e.score = (SELECT MAX(score) FROM Exam WHERE exam\_id = e.exam\_id)

OR e.score = (SELECT MIN(score) FROM Exam WHERE exam\_id = e.exam\_id)

)

ORDER BY s.student\_id;

Q134. Write an SQL query to report the students (student\_id, student\_name) being quiet in all exams.

**ANSWER:**

SELECT s.student\_id, s.student\_name

FROM Student s

WHERE s.student\_id IN (

SELECT e.student\_id

FROM Exam e

GROUP BY e.student\_id

HAVING MIN(e.score) > (SELECT MIN(score) FROM Exam)

AND MAX(e.score) < (SELECT MAX(score) FROM Exam)

)

ORDER BY s.student\_id;

Q135. Write an SQL query to show the second most recent activity of each user.

**ANSWER:**

SELECT ua.username, ua.activity, ua.startDate, ua.endDate

FROM UserActivity ua

WHERE ua.startDate = (

SELECT MAX(startDate)

FROM UserActivity

WHERE username = ua.username AND startDate < (

SELECT MAX(startDate)

FROM UserActivity

WHERE username = ua.username

)

);

Q136. Write an SQL query to show the second most recent activity of each user.

**ANSWER:**

SELECT ua.username, ua.activity, ua.startDate, ua.endDate

FROM UserActivity ua

WHERE ua.startDate = (

SELECT MAX(startDate)

FROM UserActivity

WHERE username = ua.username AND startDate < (

SELECT MAX(startDate)

FROM UserActivity

WHERE username = ua.username

)

)

OR (

SELECT COUNT(\*)

FROM UserActivity

WHERE username = ua.username

) = 1;

Q137. Write a query.

**ANSWER:**

SELECT CEIL(AVG(SUBSTRING(salary, 1, INSTR(salary, '0') - 1)) - AVG(salary))

FROM EMPLOYEES;

Q138. Write a query.

**ANSWER:**

SELECT MAX(months \* salary), COUNT(\*)

FROM Employee

WHERE months \* salary = (SELECT MAX(months \* salary) FROM Employee);

Q139. Write a query.

**ANSWER:**

SELECT name || '(' || SUBSTR(occupation, 1, 1) || ')'

FROM OCCUPATIONS

ORDER BY name;

SELECT 'There are a total of ' || COUNT(\*) || ' ' || LOWER(occupation) || 's.'

FROM OCCUPATIONS

GROUP BY occupation

ORDER BY COUNT(\*), occupation;

Q140. Write a query.

**ANSWER:**

SELECT

MAX(CASE WHEN Occupation = 'Doctor' THEN Name ELSE NULL END) AS Doctor,

MAX(CASE WHEN Occupation = 'Professor' THEN Name ELSE NULL END) AS Professor,

MAX(CASE WHEN Occupation = 'Singer' THEN Name ELSE NULL END) AS Singer,

MAX(CASE WHEN Occupation = 'Actor' THEN Name ELSE NULL END) AS Actor

FROM OCCUPATIONS

GROUP BY Name

ORDER BY Name;

Q141. Write a query to find the node type of Binary Tree ordered by the value of the node.

**ANSWER:**

SELECT N,

CASE

WHEN P IS NULL THEN 'Root'

WHEN N IN (SELECT DISTINCT P FROM BST) THEN 'Inner'

ELSE 'Leaf'

END AS NodeType

FROM BST

ORDER BY N;

Q142. Write a query.

**ANSWER:**

SELECT

c.company\_code,

c.founder,

COUNT(DISTINCT lm.lead\_manager\_code) AS total\_lead\_managers,

COUNT(DISTINCT sm.senior\_manager\_code) AS total\_senior\_managers,

COUNT(DISTINCT m.manager\_code) AS total\_managers,

COUNT(DISTINCT e.employee\_code) AS total\_employees

FROM

Company AS c

LEFT JOIN Lead\_Manager AS lm ON c.company\_code = lm.company\_code

LEFT JOIN Senior\_Manager AS sm ON lm.lead\_manager\_code = sm.lead\_manager\_code

LEFT JOIN Manager AS m ON sm.senior\_manager\_code = m.senior\_manager\_code

LEFT JOIN Employee AS e ON m.manager\_code = e.manager\_code

GROUP BY c.company\_code, c.founder

ORDER BY CAST(REPLACE(c.company\_code, 'C\_', '') AS UNSIGNED);

Q143. Write a query to output all such symmetric pairs in ascending order by the value of X. List the rows such that X1 ≤ Y1.

**ANSWER:**

SELECT

X, Y

FROM

Functions

WHERE

X <= Y

GROUP BY

X, Y

HAVING

COUNT(\*) > 1 AND X = Y

ORDER BY

X ASC;

Q144. Write a query to output the names of those students whose best friends got offered a higher salary than them. Names must be ordered by the salary amount offered to the best friends. It is guaranteed that no two students get the same salary offer.

**ANSWER:**

SELECT

s.Name

FROM

Students s

JOIN Friends f ON s.ID = f.ID

JOIN Packages p1 ON f.Friend\_ID = p1.ID

JOIN Packages p2 ON s.ID = p2.ID

WHERE

p1.Salary > p2.Salary

ORDER BY

p1.Salary;

Q145. Write a query.

**ANSWER:**

SELECT

h.hacker\_id,

h.name

FROM

Hackers h

JOIN Submissions s ON h.hacker\_id = s.hacker\_id

JOIN Challenges c ON s.challenge\_id = c.challenge\_id

JOIN Difficulty d ON c.difficulty\_level = d.difficulty\_level

WHERE

s.score = d.score

GROUP BY

h.hacker\_id, h.name

HAVING

COUNT(DISTINCT s.challenge\_id) > 1

ORDER BY

COUNT(DISTINCT s.challenge\_id) DESC, h.hacker\_id ASC;

Q146. Write a query to output the start and end dates of projects listed by the number of days it took to complete the project in ascending order. If there is more than one project that have the same number of completion days, then order by the start date of the project.

**ANSWER:**

SELECT

MIN(Start\_Date) AS Start\_Date,

MAX(End\_Date) AS End\_Date

FROM

Projects p1

WHERE

NOT EXISTS (

SELECT 1

FROM Projects p2

WHERE p2.Start\_Date = DATEADD(day, 1, p1.End\_Date)

)

GROUP BY

Start\_Date - ROW\_NUMBER() OVER (ORDER BY Start\_Date) AS grp

ORDER BY

DATEDIFF(day, MIN(Start\_Date), MAX(End\_Date)) ASC,

MIN(Start\_Date) ASC;

Q147. Write a query.

**ANSWER:**

SELECT user\_id

FROM (

SELECT user\_id, transaction\_date,

ROW\_NUMBER() OVER (PARTITION BY user\_id ORDER BY transaction\_date) AS rn,

DATEDIFF(day, LAG(transaction\_date) OVER (PARTITION BY user\_id ORDER BY transaction\_date), transaction\_date) AS date\_diff

FROM transactions

) AS t

WHERE date\_diff = 1

GROUP BY user\_id, DATEADD(day, -rn, transaction\_date)

HAVING COUNT(\*) >= 3

ORDER BY user\_id;

Q148. Write a query.

**ANSWER:**

SELECT COUNT(\*) AS unique\_relationships

FROM (

SELECT payer\_id, recipient\_id

FROM payments

WHERE payer\_id < recipient\_id

GROUP BY payer\_id, recipient\_id

HAVING COUNT(\*) = 2

) AS t;

Q149. Write a query.

**ANSWER:**

SELECT COUNT(\*) AS users

FROM (

SELECT user\_id

FROM (

SELECT user\_id, spend, transaction\_date,

ROW\_NUMBER() OVER (PARTITION BY user\_id ORDER BY transaction\_date) AS row\_num

FROM user\_transactions

) AS t

WHERE row\_num = 1 AND spend >= 50.00

) AS u;

Q150. Write a query.

**ANSWER:**

SELECT

CAST(measurement\_time AS DATE) AS measurement\_day,

SUM(CASE WHEN measurement\_id % 2 = 1 THEN measurement\_value ELSE 0 END) AS odd\_sum,

SUM(CASE WHEN measurement\_id % 2 = 0 THEN measurement\_value ELSE 0 END) AS even\_sum

FROM measurements

GROUP BY CAST(measurement\_time AS DATE)

ORDER BY measurement\_day;

Q151. Write a query.

**ANSWER:**

SELECT DISTINCT t1.user\_id

FROM transactions t1

JOIN transactions t2 ON t1.user\_id = t2.user\_id

JOIN transactions t3 ON t1.user\_id = t3.user\_id

WHERE t1.transaction\_date < t2.transaction\_date

AND t2.transaction\_date < t3.transaction\_date

AND DATEDIFF(t1.transaction\_date, t3.transaction\_date) = 2

ORDER BY t1.user\_id ASC;

152. Write a query.

**ANSWER:**

SELECT COUNT(DISTINCT CONCAT(t1.rental\_id, '-', t2.rental\_id)) AS matching\_airbnb

FROM rental\_amenities t1

JOIN rental\_amenities t2 ON t1.amenity = t2.amenity AND t1.rental\_id < t2.rental\_id

GROUP BY t1.rental\_id, t2.rental\_id

HAVING COUNT(DISTINCT t1.amenity) = (SELECT COUNT(DISTINCT amenity) FROM rental\_amenities);

153. Write a query.

**ANSWER:**

SELECT advertiser\_id, ROUND(SUM(revenue) / SUM(spend), 2) AS ROAS

FROM ad\_campaigns

GROUP BY advertiser\_id

ORDER BY advertiser\_id;

Q154. Write a query.

**ANSWER:**

SELECT employee\_id, salary,

CASE

WHEN salary > 2 \* avg\_salary THEN 'Overpaid'

WHEN salary < 0.5 \* avg\_salary THEN 'Underpaid'

END AS status

FROM (

SELECT employee\_id, salary, title, AVG(salary) OVER (PARTITION BY title) AS avg\_salary

FROM employee\_pay

) AS subquery

WHERE status IS NOT NULL;

Q155. Write a query.

**ANSWER:**

SELECT COUNT(\*) AS unique\_relationships

FROM (

SELECT payer\_id, recipient\_id

FROM payments

WHERE payer\_id < recipient\_id

UNION ALL

SELECT recipient\_id, payer\_id

FROM payments

WHERE payer\_id > recipient\_id

) AS two\_way\_relationships

GROUP BY payer\_id, recipient\_id

HAVING COUNT(\*) = 2;

Q156. Write a query.

**ANSWER:**

SELECT COUNT(DISTINCT user\_id) AS repeat\_purchasers

FROM purchases

GROUP BY user\_id, product\_id

HAVING COUNT(DISTINCT DATE(purchase\_date)) >= 2;

Q157. Write a query.

**ANSWER:**

SELECT

DATE(transaction\_date) AS transaction\_date,

SUM(

CASE

WHEN type = 'deposit' THEN amount

WHEN type = 'withdrawal' THEN -amount

ELSE 0

END

) AS balance

FROM transactions

GROUP BY DATE(transaction\_date)

ORDER BY DATE(transaction\_date);

Q158. Write a query.

**ANSWER:**

SELECT

category,

product,

SUM(spend) AS total\_spend

FROM product\_spend

WHERE EXTRACT(YEAR FROM transaction\_date) = 2022

GROUP BY category, product

ORDER BY category, total\_spend DESC

LIMIT 2;

Q159. Write a query.

**ANSWER:**

SELECT

EXTRACT(WEEK FROM signup\_date) AS signup\_week,

ROUND(COUNT(CASE WHEN last\_login > signup\_date + INTERVAL '28 days' THEN 1 END) \* 100.0 / COUNT(\*), 2) AS churn\_rate

FROM users

WHERE EXTRACT(MONTH FROM signup\_date) = 6 AND EXTRACT(YEAR FROM signup\_date) = 2022

GROUP BY signup\_week

ORDER BY signup\_week;