***SQL\_ASSIGNMENT***

**Q1. Query all columns for all American cities in the CITY table with populations larger than 100000.**

SELECT \* FROM sample\_dataset\_1

WHERE

population > 100000 AND country\_code = 'USA'

;

**Q2. Query the NAME field for all American cities in the CITY table with populations larger than 120000**

=>

SELECT name FROM sample\_dataset\_1

WHERE

population > 120000 AND country\_code = 'USA'

;

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

SELECT \* FROM sample\_dataset\_1;

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

**No data exists with id = 1661**

SELECT \* FROM sample\_dataset\_1

WHERE

id = 1661;

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

SELECT \* FROM sample\_dataset\_1

WHERE

country\_code = 'JPN' ;

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

**JPN**

SELECT name FROM sample\_dataset\_1

WHERE

country\_code = 'JPN';

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

SELECT city, state FROM sample\_dataset\_2;

**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.**

SELECT city FROM sample\_dataset\_2

WHERE

id % 2 = 0

GROUP BY city;

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

**distinct CITY entries in the table.**

SELECT (COUNT(city) - COUNT(distinct(city))) as Difference

FROM sample\_dataset\_2;

**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.**

(SELECT city, LENGTH(city) as city\_length from sample\_dataset\_2

WHERE

LENGTH(city) = (select min(LENGTH(city)) FROM sample\_dataset\_2)

ORDER BY city LIMIT 1)

UNION ALL

(SELECT city, LENGTH(city) as city\_length from sample\_dataset\_2

WHERE

LENGTH(city) = (select max(LENGTH(city)) FROM sample\_dataset\_2)

ORDER BY city LIMIT 1)

;

**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.**

SELECT distinct(city) FROM sample\_dataset\_2

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**

SELECT distinct(city) FROM sample\_dataset\_2

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.**

SELECT distinct(city) FROM sample\_dataset\_2

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**

SELECT distinct(city) FROM sample\_dataset\_2

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.**

SELECT distinct(city) FROM sample\_dataset\_2

WHERE

Left(city, 1) not in ("a", "i", "e", "o","u")

OR

right(city, 1) not in ("a", "i", "e", "o","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.**

SELECT distinct(city) FROM sample\_dataset\_2

WHERE

Left(city, 1) not in ("a", "i", "e", "o","u")

AND

right(city, 1) not in ("a", "i", "e", "o","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.**

**Return the result table in any order.**

SELECT \* from product WHERE product\_id in (SELECT product\_id FROM sales

WHERE

sale\_date >= '2019-01-01' and sale\_date <= "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.**

SELECT author\_id FROM views

WHERE

author\_id = viewer\_id

GROUP BY author\_id

ORDER BY author\_id;

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

**places.**

SELECT round(100 \*

(SELECT COUNT(\*) FROM delivery

WHERE

order\_date = customer\_pref\_delivery\_date

) / COUNT(\*), 2)

FROM delivery;

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

**Return the result table ordered by ctr in descending order and by ad\_id in ascending order in case of a**

**tie**

SELECT ad\_id,

round(

(SUM(action='clicked')/

(SUM(action='clicked') + SUM(action='viewed'))

)\*100, 2) as CTR

FROM ads 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.**

**Return result table in any order**

SELECT employee\_id,

COUNT(employee\_id) over(partition by team\_id) as total\_count

from employee ORDER BY employee\_id

;

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

**The type of weather is:**

**● Cold if the average weather\_state is less than or equal 15,**

**● Hot if the average weather\_state is greater than or equal to 25, and**

**● Warm otherwise.**

**Return result table in any order**

SELECT ct.country\_id, ct.country\_name, wt.new\_state

FROM

countries as ct

RIGHT JOIN (SELECT country\_id, CASE

when AVG(weather\_state) <= 15 then "cold"

when AVG(weather\_state) >= 25 then "hot"

else "warm"

end as new\_state

FROM weather

WHERE day >= "2019-11-01" and day <= "2019-11-30"

GROUP BY country\_id) as wt ON ct.country\_id=wt.country\_id

ORDER BY wt.new\_state

;

**Q23. Write an SQL query to find the average selling price for each product. average\_price should be**

**rounded to 2 decimal places.**

**Return the result table in any order.**

with CTE as (SELECT p.product\_id, p.price, o.units FROM prices p JOIN unit\_sold o on o.product\_id = p.product\_id

WHERE

o.purchase\_date BETWEEN p.start\_date AND p.end\_date)

SELECT round(sum(price \* units)/sum(units), 2) as average\_selling\_price FROM CTE

GROUP BY product\_id

;

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

**Return the result table in any order.**

select player\_id, event\_date as first\_login\_date FROM (SELECT player\_id, event\_date,

rank() over(partition by player\_id ORDER BY event\_date ASC) as first\_login

FROM activity ORDER BY player\_id) as tmp

WHERE

first\_login = 1

;

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

**Return the result table in any order.**

select player\_id, device\_id as first\_login\_device FROM (SELECT player\_id, device\_id, event\_date,

rank() over(partition by player\_id ORDER BY event\_date ASC) as first\_login

FROM activity ORDER BY player\_id) as tmp

WHERE

first\_login = 1

;

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

**and their amount.**

**Return result table in any order.**

SELECT p.product\_name, o.total\_units FROM products as p INNER JOIN (SELECT product\_id, SUM(unit) as total\_units FROM orders

WHERE

order\_date >= "2020-02-01" and order\_date <= "2020-02-28"

GROUP BY product\_id) as o on p.product\_id = o.product\_id

WHERE o.total\_units >= 100

;

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

**A valid e-mail has a prefix name and a domain where:**

**● The prefix name is a string that may contain letters (upper or lower case), digits, underscore**

**'\_', period '.', and/or dash '-'. The prefix name must start with a letter.**

**● The domain is '@leetcode.com'.**

**Return the result table in any order.**

SELECT user\_id, name, mail FROM users

WHERE

mail REGEXP "^[a-zA-Z]+[a-zA-Z0-9\_.\-]\*@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.**

**Return the result table in any order.**

*NOT ABLE 2 SOLVE*

**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.**

SELECT title FROM content

WHERE

kids\_content = "Y" and content\_type = "Movies" and content\_id in (SELECT content\_id FROM tv\_program

WHERE

program\_date >= "2020-06-01 00:00" and program\_date <= "2020-06-30 12:00")

ORDER BY title

;

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

**Return the result table in any order**

SELECT q.id, q.year, IFNULL(npv\_table.npv, 0) as npv\_value FROM queries q

LEFT JOIN

(SELECT \* FROM npv) as npv\_table

ON

npv\_table.id = q.id and npv\_table.year = q.year

;

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

**Return the result table in any order.**

SELECT q.id, q.year, IFNULL(npv\_table.npv, 0) as npv\_value FROM queries q

LEFT JOIN

(SELECT \* FROM npv) as npv\_table

ON

npv\_table.id = q.id and npv\_table.year = q.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.**

SELECT IFNULL(e\_uni.unique\_id, NULL) as uniq\_id, e.name FROM employee\_uni as e\_uni

RIGHT JOIN

(

SELECT \* FROM employees

) as e

ON

e.id = e\_uni.id

;

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

**Return the result table ordered by travelled\_distance in descending order, if two or more users**

**travelled the same distance, order them by their name in ascending order.**

SELECT u.name, IFNULL(r.total\_distance, 0) FROM users u

LEFT JOIN

(

SELECT user\_id, SUM(distance) as total\_distance FROM rides

GROUP BY user\_id

) as r on r.user\_id = u.id

ORDER BY total\_distance DESC, u.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.**

**Return result table in any order**

SELECT p.product\_name, o.total\_units FROM products as p INNER JOIN (SELECT product\_id, SUM(unit) as total\_units FROM orders

WHERE

order\_date >= "2020-02-01" and order\_date <= "2020-02-28"

GROUP BY product\_id) as o on p.product\_id = o.product\_id

WHERE o.total\_units >= 100

;

**Q35. Write an SQL query to:**

**● Find the name of the user who has rated the greatest number of movies. In case of a tie,**

**return the lexicographically smaller user name.**

**● Find the movie name with the highest average rating in February 2020. In case of a tie, return**

**the lexicographically smaller movie name.**

*NOT ABLE 2 SOLVE*

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

**Return the result table ordered by travelled\_distance in descending order, if two or more users**

**travelled the same distance, order them by their name in ascending order.**

SELECT u.name, IFNULL(r.total\_distance, 0) FROM users u

LEFT JOIN

(

SELECT user\_id, SUM(distance) as total\_distance FROM rides

GROUP BY user\_id

) as r on r.user\_id = u.id

ORDER BY total\_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.**

SELECT IFNULL(e\_uni.unique\_id, NULL) as uniq\_id, e.name FROM employee\_uni as e\_uni

RIGHT JOIN

(

SELECT \* FROM employees

) as e

ON

e.id = e\_uni.id

;

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

**longer exist.**

SELECT id, name FROM (SELECT name, id, department\_id FROM students

HAVING

department\_id not IN (SELECT id from departments)

) as tmp;

**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.**

**Return the result table in any order.**

WITH CTE as (

(select from\_id as person1, to\_id as person2, duration

from calls)

UNION ALL

(select to\_id as person1, from\_id as person2, duration

from calls)

)

select person1, person2, count(\*), sum(duration)

from CTE

where person1 < person2

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.**

**Return the result table in any order**

with CTE as (SELECT p.product\_id, p.price, o.units FROM prices p JOIN unit\_sold o on o.product\_id = p.product\_id

WHERE

o.purchase\_date BETWEEN p.start\_date AND p.end\_date)

SELECT round(sum(price \* units)/sum(units), 2) as average\_selling\_price FROM CTE

GROUP BY product\_id

;

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

**warehouse.**

SELECT w.name, sum((w.units \* p.total\_volume)) as total\_cubic\_feet FROM warehouse w

JOIN (SELECT product\_id, (width \* height \* length) as total\_volume FROM products) as p

ON

p.product\_id = w.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.**

SELECT sa.sale\_date, (SUM(sa.sold\_num) - so.total\_oranges) as diff FROM sales as sa

JOIN (SELECT sale\_date, SUM(sold\_num) as total\_oranges FROM sales

WHERE

fruit = "oranges"

GROUP BY sale\_date) as so

ON

so.sale\_date = sa.sale\_date

WHERE

sa.fruit = "apples"

GROUP BY 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.**

with CTE as (SELECT player\_id, event\_date,

datediff(event\_date, lag(event\_date) over (partition by player\_id ORDER BY event\_date ASC)) as lag\_date

FROM activity)

SELECT round(count(distinct(player\_id)) / (select count(DISTINCT(player\_id)) FROM activity), 2) as fraction

from CTE

WHERE

lag\_date = 1

;

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

**Return the result table in any order**

SELECT name FROM employee

WHERE

id = (SELECT managerid FROM employee

GROUP BY managerid

HAVING

COUNT(managerid) >= 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).**

**Return the result table ordered by student\_number in descending order. In case of a tie, order them by**

**dept\_name alphabetically.**

SELECT dp.dept\_name, IFNULL(s.total\_students, 0) as total\_students FROM department as dp

LEFT JOIN

(SELECT dept\_id, count(dept\_id) as total\_students

FROM student

GROUP BY dept\_id) as s

ON

dp.dept\_id = s.dept\_id

;

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

**the Product table.**

with CTE as (SELECT customer\_id, COUNT(customer\_id) as total\_count FROM customer

WHERE

product\_key IN (SELECT \* from product)

GROUP BY customer\_id)

SELECT customer\_id from CTE

WHERE

total\_count = (SELECT max(total\_count) FROM CTE)

GROUP BY customer\_id

;

**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.**

**Return the result table ordered by student\_id in ascending order**

SELECT mgt.student\_id, e.course\_id, e.grade FROM enrollments e

JOIN (SELECT student\_id, max(grade) as max\_grade FROM enrollments

GROUP BY student\_id) as mgt

ON

mgt.student\_id = e.student\_id and mgt.max\_grade = e.grade

ORDER BY student\_id;

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

**Return the result table in any order.**

SELECT name, population, gdp 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**

SELECT name FROM customer

WHERE

referee\_id != 2 or referee\_id IS NULL

;

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

**Return the result table in any order.**

SELECT name FROM customers

WHERE

id not IN (SELECT customerid FROM orders)

;

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

**Return result table in any order.**

SELECT employee\_id,

COUNT(employee\_id) over(partition by team\_id) as total\_count

from employee ORDER BY employee\_id

;

**Q55. A telecommunications company wants to invest in new countries. The company intends to invest in**

**the countries where the average call duration of the calls in this country is strictly greater than the**

**global average call duration.**

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

**Return the result table in any order.**

*NOT ABLE 2 SOLVE*

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

**Return the result table in any order.**

select player\_id, device\_id as first\_login\_device FROM (SELECT player\_id, device\_id, event\_date,

rank() over(partition by player\_id ORDER BY event\_date ASC) as first\_login

FROM activity ORDER BY player\_id) as tmp

WHERE

first\_login = 1

;

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

**number of orders.**

**The test cases are generated so that exactly one customer will have placed more orders than any**

**other customer.**

SELECT customer\_number FROM (SELECT customer\_number, COUNT(customer\_number) as total\_count FROM orders

GROUP BY customer\_number

ORDER BY total\_count DESC LIMIT 1) as tmp

;

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

**Return the result table ordered by seat\_id in ascending order.**

**The test cases are generated so that more than two seats are consecutively available.**

with CTE as (SELECT \*,

lag(free) over(order by free) as new\_val

FROM cinema)

SELECT seat\_id FROM CTE

WHERE

new\_val = 1

GROUP BY 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".**

SELECT sales\_id FROM orders

WHERE

com\_id = 1;

SELECT name FROM salesperson

WHERE

sales\_id NOT IN

(

SELECT sales\_id FROM orders

WHERE

com\_id = (

SELECT com\_id FROM company

WHERE

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.**

SELECT \*,

CASE

when (x + y) <= z or (y + z) <= x or (z + x) <= y then "NO"

else "YES"

end as triangle\_bool

FROM triangle

;

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

**The query result format is in the following example.**

SELECT MIN(new\_val) from (SELECT

IFNULL(ABS(lag(x) over(order by x ASC)), 0) as new\_val

FROM point) as tmp

WHERE

new\_val != 0

;

**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.**

SELECT actor\_id, director\_id FROM actor\_director

GROUP BY actor\_id, director\_id

HAVING

count(CONCAT(actor\_id,director\_id)) >= 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**

SELECT p.product\_name, s.year, s.price FROM sales as s

JOIN (select \* FROM product) as p ON p.product\_id = s.product\_id

;

**Q64. Write an SQL query that reports the average experience years of all the employees for each project,**

**rounded to 2 digits.**

SELECT p.project\_id, round(avg(e.experience\_years), 2) as average\_exp

FROM project as p

JOIN (SELECT \* from employee) as e ON

e.employee\_id = p.employee\_id

GROUP BY 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.**

with CTE as (SELECT seller\_id, sum(price) as total\_price FROM sales

GROUP BY seller\_id)

SELECT seller\_id FROM CTE

WHERE

total\_price >= (SELECT max(total\_price) FROM CTE)

;

**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**

SELECT buyer\_id FROM sales

WHERE

buyer\_id NOT IN

(SELECT buyer\_id FROM sales

WHERE

product\_id IN (SELECT product\_id FROM product WHERE product\_name != "S8")

GROUP BY buyer\_id

)

;

**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.**

Return result table ordered by visited\_on in ascending order.

with CTE as

(SELECT visited\_on,

SUM(total\_amount) over (rows BETWEEN 6 preceding and current row) as sum\_amount,

AVG(total\_amount) over (rows BETWEEN 6 preceding and current row) as average\_amount

FROM

(

SELECT visited\_on, sum(amount) as total\_amount FROM customer

GROUP BY visited\_on

) as tmp

)

SELECT \* FROM CTE ORDER BY visited\_on ASC

;

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

**Return the result table ordered by gender and day in ascending order.**

SELECT gender, day,

sum(score\_points) over (partition by gender ORDER BY day rows between unbounded preceding and current row) as total\_score

FROM scores;

**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.**

**Return the result table in any order.**

SELECT trans\_month, country, COUNT(trans\_month), SUM(state="approved") as total\_approved,

SUM(state="decline") as total\_decline, SUM(amount)

FROM

(

SELECT id, country, state, amount,

left(trans\_date,7) as trans\_month FROM transactions

) tmp

GROUP BY trans\_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**

with CTE as (SELECT post\_id, action\_date, SUM(extra="spam") as spam\_count,

CASE

when post\_id IN (SELECT post\_id FROM removals) then 1

else 0

end as removed

FROM actions

GROUP BY action\_date, post\_id

HAVING

sum(extra="spam") != 0)

SELECT round(sum(total\_percent)/count(\*), 0) as average\_daily\_percent

FROM

(

SELECT sum(removed)/sum(spam\_count) \* 100 as total\_percent FROM CTE

GROUP BY action\_date

) tmp

;

**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.**

with CTE as (SELECT player\_id, event\_date,

datediff(event\_date, lag(event\_date) over (partition by player\_id ORDER BY event\_date ASC)) as lag\_date

FROM activity)

SELECT round(count(distinct(player\_id)) / (select count(DISTINCT(player\_id)) FROM activity), 2) as fraction

from CTE

WHERE

lag\_date = 1

;

**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.**

with CTE as (SELECT player\_id, event\_date,

datediff(event\_date, lag(event\_date) over (partition by player\_id ORDER BY event\_date ASC)) as lag\_date

FROM activity)

SELECT round(count(distinct(player\_id)) / (select count(DISTINCT(player\_id)) FROM activity), 2) as fraction

from CTE

WHERE

lag\_date = 1

;

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

**nearest integer.**

**The tax rate is calculated for each company based on the following criteria:**

**● 0% If the max salary of any employee in the company is less than $1000.**

**● 24% If the max salary of any employee in the company is in the range [1000, 10000] inclusive.**

**● 49% If the max salary of any employee in the company is greater than $10000.**

NOT ABLE 2 SOLVE

**Q77. 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.**

SELECT sa.sale\_date, (SUM(sa.sold\_num) - so.total\_oranges) as diff FROM sales as sa

JOIN (SELECT sale\_date, SUM(sold\_num) as total\_oranges FROM sales

WHERE

fruit = "oranges"

GROUP BY sale\_date) as so

ON

so.sale\_date = sa.sale\_date

WHERE

sa.fruit = "apples"

GROUP BY sale\_date;

**Q78. 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.**

SELECT sa.sale\_date, (SUM(sa.sold\_num) - so.total\_oranges) as diff FROM sales as sa

JOIN (SELECT sale\_date, SUM(sold\_num) as total\_oranges FROM sales

WHERE

fruit = "oranges"

GROUP BY sale\_date) as so

ON

so.sale\_date = sa.sale\_date

WHERE

sa.fruit = "apples"

GROUP BY sale\_date;

**Q79. Write an SQL query to evaluate the boolean expressions in Expressions table.**

**Return the result table in any order.**

with CTE as (SELECT \* FROM expression)

SELECT \*,

CASE

when operator = "<" and (left\_operand < right\_operand) = 1 then "true"

when operator = ">" and (left\_operand > right\_operand) = 1 then "true"

when operator = "=" and (left\_operand = right\_operand) = 1 then "true"

else "false"

end as new\_val

FROM CTE;

**Q80. A telecommunications company wants to invest in new countries. The company intends to invest in**

**the countries where the average call duration of the calls in this country is strictly greater than the**

**global average call duration.**

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

**Return the result table in any order.**

*NOT ABLE 2 SOLVE*

**Q81 Query the Name of any student in STUDENTS who scored higher than 75 Marks. Order your output by**

**the last three characters of each name. If two or more students both have names ending in the same**

**last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending ID**

SELECT name FROM students

WHERE

marks > 75

ORDER BY RIGHT(name, 3), id

;

**Q82. Write a query that prints a list of employee names (i.e.: the name attribute) from the Employee table in**

**alphabetical order.**

SELECT name FROM employee

ORDER BY name ASC

;

**Q83. Write a query that prints a list of employee names (i.e.: the name attribute) for employees in**

**Employee having a salary greater than $2000 per month who have been employees for less than 10**

**months. Sort your result by ascending employee\_id.**

SELECT name FROM employee

WHERE

salary > 2000 and months < 10

ORDER BY employee\_id ASC

;

**Q84. Write a query identifying the type of each record in the TRIANGLES table using its three side lengths.**

**Output one of the following statements for each record in the table:**

**● Equilateral: It's a triangle with sides of equal length.**

**● Isosceles: It's a triangle with sides of equal length.**

**● Scalene: It's a triangle with sides of differing lengths.**

**● Not A Triangle: The given values of A, B, and C don't form a triangle**

SELECT \*,

CASE

when a = b and b = c then "equilateral"

when a = b and b != c and a+b > c then "isoceles"

when a + b < c or b + c < a or c + a < b then "not a triangle"

when a != b and b != c and a != c then "scalene"

else "normal triangle"

end as triangle\_value

from triangle;

**Q85. Assume you are given the table below containing information on user transactions for particular**

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

**each year.**

**Output the year (in ascending order) partitioned by product id, current year's spend, previous year's**

**spend and year-on-year growth rate (percentage rounded to 2 decimal places).**

SELECT extract(year FROM transaction\_date) as year\_, product\_id,

spend as curr\_year\_spend,

lag(spend) over() as prev\_year\_spend,

round((spend/lag(spend) over() \* 100) - 100, 2) as yoy\_rate

FROM transactions;

**Q87. 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).**

**Hint: An active user is a user who has user action ("sign-in", "like", or "comment") in the current month**

**and last month.**

NOT ABLE 2 SOLVE

**Q90. Amazon Web Services (AWS) is powered by fleets of servers. Senior management has**

**requested data-driven solutions to optimise server usage.**

**Write a query that calculates the total time that the fleet of servers was running. The output should be**

**in units of full days.**

with CTE as (SELECT server\_id, status\_time,

lead(status\_time) over(partition by server\_id ORDER BY status\_time ASC) as new\_time

FROM server)

SELECT sum(DATEDIFF( new\_time, status\_time )) as total\_uptime\_days FROM CTE;

**Q91. Sometimes, payment transactions are repeated by accident; it could be due to user error, API failure or**

**a retry error that causes a credit card to be charged twice.**

**Using the transactions table, identify any payments made at the same merchant with the same credit**

**card for the same amount within 10 minutes of each other. Count such repeated payments.**

with CTE as (SELECT \*,

timestampdiff(minute,transaction\_timestamp,lag(transaction\_timestamp) over())

as minutes\_diff

FROM transactions)

SELECT count(minutes\_diff) as payment\_count FROM CTE

GROUP BY minutes\_diff

HAVING

abs(CTE.minutes\_diff) <= 10

;

**Q93. Write an SQL query to find the total score for each gender on each day.**

**Return the result table ordered by gender and day in ascending order.**

SELECT gender, day,

sum(score\_points) over (partition by gender ORDER BY day rows between unbounded preceding and current row) as total\_score

FROM scores;

**Q94. A telecommunications company wants to invest in new countries. The company intends to invest in**

**the countries where the average call duration of the calls in this country is strictly greater than the**

**global average call duration.**

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

**Return the result table in any order.**

*NOT ABLE 2 SOLVE*

**Q96. 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.**

NOT ABLE 2 SOLVE

**Q97.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.**

*NOT ABLE 2 SOLVE*

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

**If the user only has one activity, return that one. A user cannot perform more than one activity at the**

**same time.**

select distinct username, activity, startDate, endDate

from

(select user.\*,

rank() over (partition by username order by startDate desc) as rnk,

count(activity) over (partition by username) as num

from user\_activity user) new\_table

WHERE

(num != 1 and rnk = 2) or (num = 1 and rnk = 1)

;

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

**If the user only has one activity, return that one. A user cannot perform more than one activity at the**

**same time.**

select distinct username, activity, startDate, endDate

from

(select user.\*,

rank() over (partition by username order by startDate desc) as rnk,

count(activity) over (partition by username) as num

from user\_activity user) new\_table

WHERE

(num != 1 and rnk = 2) or (num = 1 and rnk = 1)

;

**Q103. Query the Name of any student in STUDENTS who scored higher than 75 Marks. Order your output by**

**the last three characters of each name. If two or more students both have names ending in the same**

**last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending ID**

SELECT name FROM students

WHERE

marks > 75

ORDER BY RIGHT(name, 3), id

;

**Q104. Write a query that prints a list of employee names (i.e.: the name attribute) from the Employee table in**

**alphabetical order.**

SELECT name FROM employee

ORDER BY name ASC

;

**Q105. Write a query that prints a list of employee names (i.e.: the name attribute) for employees in**

**Employee having a salary greater than $2000 per month who have been employees for less than 10**

**months. Sort your result by ascending employee\_id.**

SELECT name FROM employee

WHERE

salary > 2000 and months < 10

ORDER BY employee\_id ASC

;

**Q106. Write a query identifying the type of each record in the TRIANGLES table using its three side lengths.**

**Output one of the following statements for each record in the table:**

**● Equilateral: It's a triangle with sides of equal length.**

**● Isosceles: It's a triangle with sides of equal length.**

**● Scalene: It's a triangle with sides of differing lengths.**

**● Not A Triangle: The given values of A, B, and C don't form a triangle**

SELECT \*,

CASE

when a = b and b = c then "equilateral"

when a = b and b != c and a+b > c then "isoceles"

when a + b < c or b + c < a or c + a < b then "not a triangle"

when a != b and b != c and a != c then "scalene"

else "normal triangle"

end as triangle\_value

from triangle;

**Q107. Samantha was tasked with calculating the average monthly salaries for all employees in the**

**EMPLOYEES table, but did not realise her keyboard's 0 key was broken until after completing the**

**calculation. She wants your help finding the difference between her miscalculation (using salaries**

**with any zeros removed), and the actual average salary.**

**Write a query calculating the amount of error (i.e.: actual - miscalculated average monthly salaries),**

**and round it up to the next integer.**

SELECT

round(

AVG(salary) -

(

SELECT AVG(salary) FROM employees\_incorrect

),

0)

as diff\_salaries

FROM employees\_correct;

**Q108. We define an employee's total earnings to be their monthly salary \* months worked, and the**

**maximum total earnings to be the maximum total earnings for any employee in the Employee table.**

**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**

*NOT ABLE 2 SOLVE*

**Q109. Generate the following two result sets:**

**1. Query an alphabetically ordered list of all names in OCCUPATIONS, immediately followed by**

**the first letter of each profession as a parenthetical (i.e.: enclosed in parentheses). For**

**example: AnActorName(A), ADoctorName(D), AProfessorName(P), and ASingerName(S).**

**Query the number of occurrences of each occupation in OCCUPATIONS. Sort the occurrences in**

**ascending order, and output them in the following format:**

(SELECT CONCAT(name,"(",left(occupation, 1),")") FROM job

ORDER BY name)

UNION ALL

(SELECT

CONCAT("There are a total of ",COUNT(occupation), " ", occupation, "s")

from job

GROUP BY occupation)

;

**Q 111. You are given a table, BST, containing two columns: N and P, where N represents the value of a node**

**in Binary Tree, and P is the parent of N**

SELECT n,

CASE

when n not in (SELECT distinct(p) FROM nodes WHERE p is not NULL) then "Leaf"

when p is NULL then "Root"

else "Inner"

end as type\_of\_node

FROM nodes ORDER BY n;

**Q117. Query the Name of any student in STUDENTS who scored higher than 75 Marks. Order your output by**

**the last three characters of each name. If two or more students both have names ending in the same**

**last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending ID**

SELECT name FROM students

WHERE

marks > 75

ORDER BY RIGHT(name, 3), id

;

**Q118. Write a query that prints a list of employee names (i.e.: the name attribute) from the Employee table in**

**alphabetical order.**

SELECT name FROM employee

ORDER BY name ASC

;

**Q119. Write a query that prints a list of employee names (i.e.: the name attribute) for employees in**

**Employee having a salary greater than $2000 per month who have been employees for less than 10**

**months. Sort your result by ascending employee\_id.**

SELECT name FROM employee

WHERE

salary > 2000 and months < 10

ORDER BY employee\_id ASC

;

**Q120. Write a query identifying the type of each record in the TRIANGLES table using its three side lengths.**

**Output one of the following statements for each record in the table:**

**● Equilateral: It's a triangle with sides of equal length.**

**● Isosceles: It's a triangle with sides of equal length.**

**● Scalene: It's a triangle with sides of differing lengths.**

**● Not A Triangle: The given values of A, B, and C don't form a triangle**

SELECT \*,

CASE

when a = b and b = c then "equilateral"

when a = b and b != c and a+b > c then "isoceles"

when a + b < c or b + c < a or c + a < b then "not a triangle"

when a != b and b != c and a != c then "scalene"

else "normal triangle"

end as triangle\_value

from triangle;

**Q121. Assume you are given the table below containing information on user transactions for particular**

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

**each year.**

**Output the year (in ascending order) partitioned by product id, current year's spend, previous year's**

**spend and year-on-year growth rate (percentage rounded to 2 decimal places).**

SELECT extract(year FROM transaction\_date) as year\_, product\_id,

spend as curr\_year\_spend,

lag(spend) over() as prev\_year\_spend,

round((spend/lag(spend) over() \* 100) - 100, 2) as yoy\_rate

FROM transactions;

**Q 123. 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).**

**Hint: An active user is a user who has user action ("sign-in", "like", or "comment") in the current month**

**and last month.**

SELECT extract(month FROM event\_date) as month, COUNT(user\_id) as MAU FROM user\_actions

WHERE

event\_date BETWEEN "2022-06-01 00:00:00" and "2022-06-30 12:00:00" AND event\_type = "sign-in"

and user\_id in

(

SELECT user\_id FROM user\_actions

WHERE event\_date BETWEEN "2022-06-01 00:00:00" and "2022-06-30 12:00:00" AND event\_type != "sign-in"

)

GROUP BY month

;

**Q 126. Amazon Web Services (AWS) is powered by fleets of servers. Senior management has**

**requested data-driven solutions to optimise server usage.**

**Write a query that calculates the total time that the fleet of servers was running. The output should be**

**in units of full days.**

with CTE as (SELECT server\_id, status\_time,

lead(status\_time) over(partition by server\_id ORDER BY status\_time ASC) as new\_time

FROM server)

SELECT sum(DATEDIFF( new\_time, status\_time )) as total\_uptime\_days FROM CTE;

**Q127. Sometimes, payment transactions are repeated by accident; it could be due to user error, API failure or**

**a retry error that causes a credit card to be charged twice.**

**Using the transactions table, identify any payments made at the same merchant with the same credit**

**card for the same amount within 10 minutes of each other. Count such repeated payments.**

with CTE as (SELECT \*,

timestampdiff(minute,transaction\_timestamp,lag(transaction\_timestamp) over())

as minutes\_diff

FROM transactions)

SELECT count(minutes\_diff) as payment\_count FROM CTE

GROUP BY minutes\_diff

HAVING

abs(CTE.minutes\_diff) <= 10

;

**Q129. Write an SQL query to find the total score for each gender on each day.**

**Return the result table ordered by gender and day in ascending order.**

SELECT gender, day,

sum(score\_points) over (partition by gender ORDER BY day rows between unbounded preceding and current row) as total\_score

FROM scores;

**Q130. A telecommunications company wants to invest in new countries. The company intends to invest in**

**the countries where the average call duration of the calls in this country is strictly greater than the**

**global average call duration.**

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

**Return the result table in any order.**

*NOT ABLE 2 SOLVE*

**Q132. 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.**

*NOT ABLE 2 SOLVE*

**Q 133. 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).**

**Hint: An active user is a user who has user action ("sign-in", "like", or "comment") in the current month**

**and last month.**

SELECT extract(month FROM event\_date) as month, COUNT(user\_id) as MAU FROM user\_actions

WHERE

event\_date BETWEEN "2022-06-01 00:00:00" and "2022-06-30 12:00:00" AND event\_type = "sign-in"

and user\_id in

(

SELECT user\_id FROM user\_actions

WHERE event\_date BETWEEN "2022-06-01 00:00:00" and "2022-06-30 12:00:00" AND event\_type != "sign-in"

)

GROUP BY month

;

**Q137. Write an SQL query to show the second most recent activity of each user.**

**If the user only has one activity, return that one. A user cannot perform more than one activity at the**

**same time.**

select distinct username, activity, startDate, endDate

from

(select user.\*,

rank() over (partition by username order by startDate desc) as rnk,

count(activity) over (partition by username) as num

from user\_activity user) new\_table

WHERE

(num != 1 and rnk = 2) or (num = 1 and rnk = 1)

;

**Q138. Write an SQL query to show the second most recent activity of each user.**

**If the user only has one activity, return that one. A user cannot perform more than one activity at the**

**same time.**

select distinct username, activity, startDate, endDate

from

(select user.\*,

rank() over (partition by username order by startDate desc) as rnk,

count(activity) over (partition by username) as num

from user\_activity user) new\_table

WHERE

(num != 1 and rnk = 2) or (num = 1 and rnk = 1)

;

**Q139. Query the Name of any student in STUDENTS who scored higher than 75 Marks. Order your output by**

**the last three characters of each name. If two or more students both have names ending in the same**

**last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending ID**

SELECT name FROM students

WHERE

marks > 75

ORDER BY RIGHT(name, 3), id

;

**Q140. Write a query that prints a list of employee names (i.e.: the name attribute) from the Employee table in**

**alphabetical order.**

SELECT name FROM employee

ORDER BY name ASC

;

**Q141. Write a query that prints a list of employee names (i.e.: the name attribute) for employees in**

**Employee having a salary greater than $2000 per month who have been employees for less than 10**

**months. Sort your result by ascending employee\_id.**

SELECT name FROM employee

WHERE

salary > 2000 and months < 10

ORDER BY employee\_id ASC

;

**Q142. Write a query identifying the type of each record in the TRIANGLES table using its three side lengths.**

**Output one of the following statements for each record in the table:**

**● Equilateral: It's a triangle with sides of equal length.**

**● Isosceles: It's a triangle with sides of equal length.**

**● Scalene: It's a triangle with sides of differing lengths.**

**● Not A Triangle: The given values of A, B, and C don't form a triangle**

SELECT \*,

CASE

when a = b and b = c then "equilateral"

when a = b and b != c and a+b > c then "isoceles"

when a + b < c or b + c < a or c + a < b then "not a triangle"

when a != b and b != c and a != c then "scalene"

else "normal triangle"

end as triangle\_value

from triangle;

**Q143. Samantha was tasked with calculating the average monthly salaries for all employees in the**

**EMPLOYEES table, but did not realise her keyboard's 0 key was broken until after completing the**

**calculation. She wants your help finding the difference between her miscalculation (using salaries**

**with any zeros removed), and the actual average salary.**

**Write a query calculating the amount of error (i.e.: actual - miscalculated average monthly salaries),**

**and round it up to the next integer.**

SELECT

round(

AVG(salary) -

(

SELECT AVG(salary) FROM employees\_incorrect

),

0)

as diff\_salaries

FROM employees\_correct;

**Q144. We define an employee's total earnings to be their monthly salary \* months worked, and the**

**maximum total earnings to be the maximum total earnings for any employee in the Employee table.**

**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**

with CTE as

(

SELECT \*, (salary \* months) as total\_earnings FROM employees

)

SELECT

concat(total\_earnings, " ", count(total\_earnings)) as output\_table

FROM CTE

WHERE

total\_earnings = (SELECT max(total\_earnings) FROM CTE)

GROUP BY total\_earnings

;

**Q145. Generate the following two result sets:**

**1. Query an alphabetically ordered list of all names in OCCUPATIONS, immediately followed by**

**the first letter of each profession as a parenthetical (i.e.: enclosed in parentheses). For**

**example: AnActorName(A), ADoctorName(D), AProfessorName(P), and ASingerName(S).**

**Query the number of occurrences of each occupation in OCCUPATIONS. Sort the occurrences in**

**ascending order, and output them in the following format:**

(SELECT CONCAT(name,"(",left(occupation, 1),")") FROM job

ORDER BY name)

UNION ALL

(SELECT

CONCAT("There are a total of ",COUNT(occupation), " ", occupation, "s")

from job

GROUP BY occupation)

;

**Q 147. You are given a table, BST, containing two columns: N and P, where N represents the value of a node**

**in Binary Tree, and P is the parent of N**

SELECT n,

CASE

when n not in (SELECT distinct(p) FROM nodes WHERE p is not NULL) then "Leaf"

when p is NULL then "Root"

else "Inner"

end as type\_of\_node

FROM nodes ORDER BY n;

**Q149. Two pairs (X1, Y1) and (X2, Y2) are said to be symmetric pairs if X1 = Y2 and X2 = Y1.**

**Write a query to output all such symmetric pairs in ascending order by the value of X. List the rows**

**such that X1 ≤ Y1.**

SELECT x, y FROM val

WHERE

x in (SELECT y FROM val)

AND

y in (SELECT x FROM val)

AND

x <= y

LIMIT 1, 3;

**Q153. In an effort to identify high-value customers, Amazon asked for your help to obtain data about users**

**who go on shopping sprees. A shopping spree occurs when a user makes purchases on 3 or more**

**consecutive days.**

**List the user IDs who have gone on at least 1 shopping spree in ascending order.**

with new\_table as

( SELECT \*,

ifnull

(

datediff(transaction\_date,

lag(transaction\_date) over(partition by user\_id)),

1) as lag\_date

FROM amazon )

SELECT user\_id FROM new\_table

GROUP BY user\_id, lag\_date

having

count(user\_id) >= 3 and new\_table.lag\_date = 1

ORDER BY user\_id ASC ;

**Q154. You are given a table of PayPal payments showing the payer, the recipient, and the amount paid. A**

**two-way unique relationship is established when two people send money back and forth. Write a**

**query to find the number of two-way unique relationships in this data.**

*NOT ABLE 2 SOLVE*

**Q155. Assume you are given the table below containing information on Facebook user logins. Write a query**

**to obtain the number of reactivated users (which are dormant users who did not log in the previous**

**month, then logged in during the current month).**

**Output the current month (in numerical) and number of reactivated users.**

SELECT extract(month FROM login\_date) as current\_month, COUNT(user\_id) as reactivations FROM

(

SELECT user\_id, login\_date,

datediff(login\_date, lag(login\_date) over(partition by user\_id)) as lag\_date

FROM user\_login

) as react\_table

WHERE

lag\_date >= 31

GROUP BY current\_month

;

**Q156. Assume you are given the table below on user transactions. Write a query to obtain the list of**

**customers whose first transaction was valued at $50 or more. Output the number of users.**

SELECT user\_id FROM

(

SELECT user\_id, spend,

lag(transaction\_date) over(partition by user\_id) as lag\_num

FROM transactions

) as nt

WHERE

lag\_num is NULL and spend >= 50;

**Q158. In an effort to identify high-value customers, Amazon asked for your help to obtain data about users**

**who go on shopping sprees. A shopping spree occurs when a user makes purchases on 3 or more**

**consecutive days.**

**List the user IDs who have gone on at least 1 shopping spree in ascending order.**

with new\_table as

(

SELECT \*,

ifnull

(

datediff(transaction\_date,

lag(transaction\_date) over(partition by user\_id)),

1) as lag\_date

FROM amazon

)

SELECT user\_id FROM new\_table

GROUP BY user\_id, lag\_date

having

count(user\_id) >= 3 and new\_table.lag\_date = 1

ORDER BY user\_id ASC

;

**Q161. Your team at Accenture is helping a Fortune 500 client revamp their compensation and benefits**

**program. The first step in this analysis is to manually review employees who are potentially overpaid**

**or underpaid.**

**An employee is considered to be potentially overpaid if they earn more than 2 times the average salary**

**for people with the same title. Similarly, an employee might be underpaid if they earn less than half of**

**the average for their title. We'll refer to employees who are both underpaid and overpaid as**

**compensation outliers for the purposes of this problem.**

**Write a query that shows the following data for each compensation outlier: employee ID, salary, and**

**whether they are potentially overpaid or potentially underpaid (refer to Example Output below).**

SELECT employee\_id, salary, over\_under FROM

(

SELECT employee\_id, salary, title,

CASE

when (AVG(salary) over(partition by title)/salary) > 2 then "Overpaid"

when (AVG(salary) over(partition by title)/salary) < 0.5 then "Underpaid"

else "Correct"

end as over\_under

FROM accenture

) as nt

WHERE

over\_under != "Correct"

;

**Q162. You are given a table of PayPal payments showing the payer, the recipient, and the amount paid. A**

**two-way unique relationship is established when two people send money back and forth. Write a**

**query to find the number of two-way unique relationships in this data.**

SELECT COUNT(person) as unique\_relationships from

(

SELECT person, COUNT(person) as person\_count FROM

(

(SELECT concat(payer\_id, recipient\_id) as person FROM payments WHERE payer\_id < recipient\_id)

UNION ALL

(SELECT concat(recipient\_id, payer\_id) as person FROM payments where recipient\_id < payer\_id )

) nt

GROUP BY person

) pt

WHERE

person\_count >= 2

;