Qn 1: Select the number of employees in each department (you only need to show the department code and the number of employees).

Qn 2: Select the names of departments with more than two employees.

Qn 3: In descending order, list the frequency count of employee last names, i.e., how many employees share each last name.

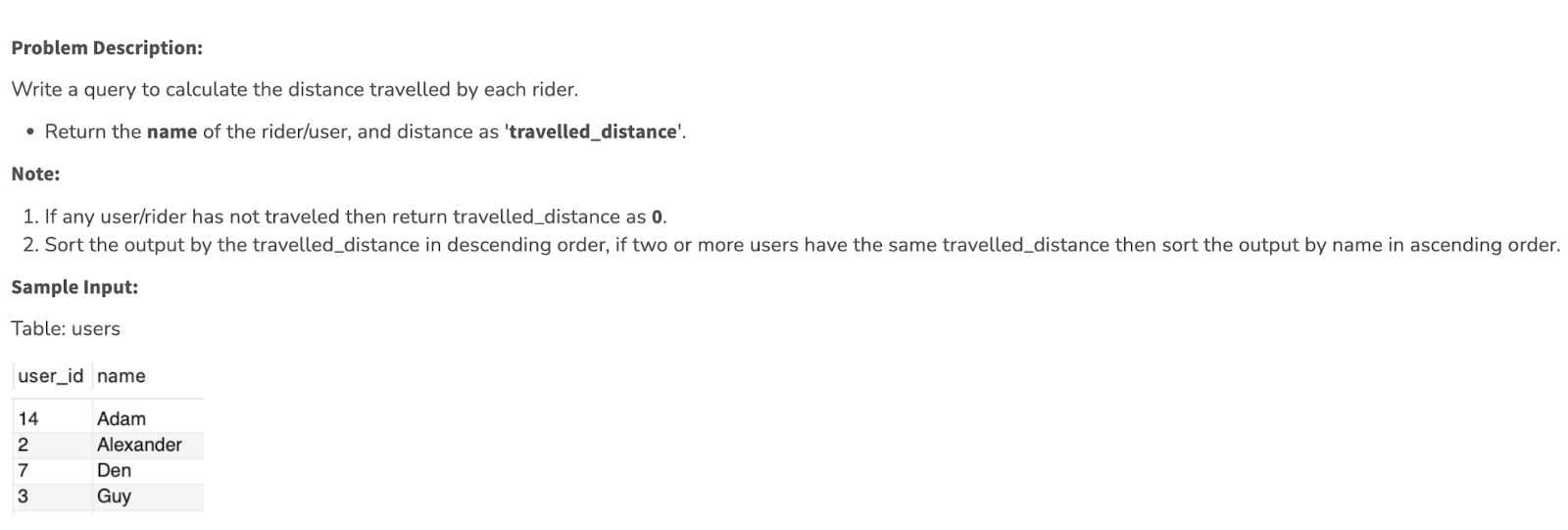
Qn 4: Write a query that obtains two columns. The first column must contain annual salaries higher than 80,000 dollars.

-- The second column, renamed to “emps\_with\_same\_salary”, must show the number of employees contracted to that salary.

-- Lastly, sort the output by the first column.

Qn 5: Display number of employees, total salary paid to employees work in each department.

Qn 6: Display the department code, total salary paid to employees group by department\_id and manager\_id=103.

Qn 7: 

select u.name, ifnull(sum(r.distance),0) as 'travelled\_distance'

from users u

left join rides r

on u.user\_id = r.user\_id

group by u.name

order by 'travelled\_distance' desc, u.name;

Qn 8: **Problem Description:**

Given a table of candidates and their skills, you're asked to find the candidates best suited for an open Data Science job. We want to find candidates who are proficient in '**Python**', '**Tableau**', and '**MySQL**'.

Write a query to list the candidates who possess all three required skills for the job. Sort the output by **candidate\_id** in ascending order.

**Note:** There are no duplicates in the candidates table.

**Sample Input:**

**Table**: candidates

Table

Description automatically generated

**Sample Output:**

Graphical user interface, text, application

Description automatically generated

**Sample Explanation:**

* Candidate 100 is displayed because he/she has Python, Tableau, and MySQL all three skills.
* Candidates 101 and 102 aren't included in the output because they're missing one or more of the required skills in Python, Tableau, and MySQL.

select candidate\_id from

(select candidate\_id, count(\*) as skills from candidates

where skill in ('Python','Tableau','MySQL')

group by candidate\_id) a

where skills = 3

order by candidate\_id;

Qn 9 : **Problem Description:**

Write a query to find the **customer\_id** and **customer\_name** of customers who bought products "**Bread**", and "**Milk**" but did not buy the product "**Eggs**" since we want to recommend them to purchase this product.

Return the output ordered by customer\_id in ascending order

**Sample Input:**

**Table**: customers

Table

Description automatically generated

**Table**: orders

Table

Description automatically generated

**Sample Output**:

Table

Description automatically generated

**Sample Explanation**:

Here, only the customer\_id with id 2 bought Bread and Milk products but did not buy the Eggs.

select c.customer\_id, c.customer\_name

from orders o

join customers c

on c.customer\_id = o.customer\_id

group by c.customer\_id, c.customer\_name

having sum(o.product\_name="Bread") >0 and

sum(o.product\_name="Milk") > 0 and

sum(o.product\_name="Eggs") = 0

order by customer\_id;

Qn 10: **Problem Description:**

Write a query to calculate the distance traveled by each rider.

* Return the **name** of the rider/user, and distance as '**travelled\_distance**'.

**Note:**

1. If any user/rider has not traveled then use the **ifnull**() function and return the travelled\_distance as **0**.
2. Sort the output by the travelled\_distance in descending order, if two or more users have the same travelled\_distance then sort the output by name in ascending order.

**Sample Input:**

Table: users

Table

Description automatically generated

Table: rides

Table

Description automatically generated

**Sample Output:**

Table

Description automatically generated

**Explanation:**

Adam did not have any rides, the distance travelled by him is 0.

Alexander and Guy have more than one ride. So, the sum of their rides is Alexander (431+981) and Guy (74+150).

Den have only one ride so, it is 731.

Qn 11: Table: Users

+----------------+---------+

| Column Name | Type |

+----------------+---------+

| user\_id | int |

| join\_date | date |

| favorite\_brand | varchar |

+----------------+---------+

user\_id is the primary key of this table.

This table has the info of the users of an online shopping website where users can sell and buy items.

Table: Orders

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| order\_id | int |

| order\_date | date |

| item\_id | int |

| buyer\_id | int |

| seller\_id | int |

+---------------+---------+

order\_id is the primary key of this table.

item\_id is a foreign key to the Items table.

buyer\_id and seller\_id are foreign keys to the Users table.

Table: Items

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| item\_id | int |

| item\_brand | varchar |

+---------------+---------+

item\_id is the primary key of this table.

Write an SQL query to find for each user, the join date and the number of orders they made as a buyer in 2019.

Return the result table in any order.

The query result format is in the following example.

Example 1:

Input:

Users table:

+---------+------------+----------------+

| user\_id | join\_date | favorite\_brand |

+---------+------------+----------------+

| 1 | 2018-01-01 | Lenovo |

| 2 | 2018-02-09 | Samsung |

| 3 | 2018-01-19 | LG |

| 4 | 2018-05-21 | HP |

+---------+------------+----------------+

Orders table:

+----------+------------+---------+----------+-----------+

| order\_id | order\_date | item\_id | buyer\_id | seller\_id |

+----------+------------+---------+----------+-----------+

| 1 | 2019-08-01 | 4 | 1 | 2 |

| 2 | 2018-08-02 | 2 | 1 | 3 |

| 3 | 2019-08-03 | 3 | 2 | 3 |

| 4 | 2018-08-04 | 1 | 4 | 2 |

| 5 | 2018-08-04 | 1 | 3 | 4 |

| 6 | 2019-08-05 | 2 | 2 | 4 |

+----------+------------+---------+----------+-----------+

Items table:

+---------+------------+

| item\_id | item\_brand |

+---------+------------+

| 1 | Samsung |

| 2 | Lenovo |

| 3 | LG |

| 4 | HP |

+---------+------------+

Output:

+-----------+------------+----------------+

| buyer\_id | join\_date | orders\_in\_2019 |

+-----------+------------+----------------+

| 1 | 2018-01-01 | 1 |

| 2 | 2018-02-09 | 2 |

| 3 | 2018-01-19 | 0 |

| 4 | 2018-05-21 | 0 |

+-----------+------------+----------------+

SELECT

user\_id as buyer\_id,

join\_date,

IFNULL(orders\_in\_2019, 0) as orders\_in\_2019

FROM Users as u LEFT JOIN

(SELECT

o.buyer\_id,

COUNT(o.order\_id) as orders\_in\_2019

FROM Orders as o JOIN Users as u

ON o.buyer\_id = u.user\_id

WHERE YEAR(order\_date) = '2019'

GROUP BY 1) x

ON u.user\_id = x.buyer\_id

ORDER by 1

Qn 12: Table: Activity

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| user\_id | int |

| session\_id | int |

| activity\_date | date |

| activity\_type | enum |

+---------------+---------+

There is no primary key for this table, it may have duplicate rows.

The activity\_type column is an ENUM of type ('open\_session', 'end\_session', 'scroll\_down', 'send\_message').

The table shows the user activities for a social media website.

Note that each session belongs to exactly one user.

Write an SQL query to find the daily active user count for a period of 30 days ending 2019-07-27 inclusively. A user was active on someday if they made at least one activity on that day.

Return the result table in any order.

The query result format is in the following example.

Example 1:

Input:

Activity table:

+---------+------------+---------------+---------------+

| user\_id | session\_id | activity\_date | activity\_type |

+---------+------------+---------------+---------------+

| 1 | 1 | 2019-07-20 | open\_session |

| 1 | 1 | 2019-07-20 | scroll\_down |

| 1 | 1 | 2019-07-20 | end\_session |

| 2 | 4 | 2019-07-20 | open\_session |

| 2 | 4 | 2019-07-21 | send\_message |

| 2 | 4 | 2019-07-21 | end\_session |

| 3 | 2 | 2019-07-21 | open\_session |

| 3 | 2 | 2019-07-21 | send\_message |

| 3 | 2 | 2019-07-21 | end\_session |

| 4 | 3 | 2019-06-25 | open\_session |

| 4 | 3 | 2019-06-25 | end\_session |

+---------+------------+---------------+---------------+

Output:

+------------+--------------+

| day | active\_users |

+------------+--------------+

| 2019-07-20 | 2 |

| 2019-07-21 | 2 |

+------------+--------------+

Explanation: Note that we do not care about days with zero active users.

Ans: SELECT activity\_date AS day, COUNT(DISTINCT user\_id) AS active\_users

FROM Activity

GROUP BY activity\_date

HAVING activity\_date <= '2019-07-27' AND activity\_date > '2019-06-27'