SQL LeetCode Questions With Solutions - PART 2

https://www.linkedin.com/in/niranjana405/

1)Find all the classes that have at least five students.

Courses table:

Here is your data in table format:

student	class
A	Math
B	English
C	Math
D	Biology
E	Math
F	Computer
G	Math
H	Math
I	Math

(student, class) is the primary key (combination of columns with unique values) for this table.

Each row of this table indicates the name of a student and the class in which they are enrolled.

Write a solution to find all the classes that have **at least five students**.

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

Output:

```
+---------+
| class   |
+---------+
| Math    |
+---------+
```

Explanation:

- Math has 6 students, so we include it.

- English has 1 student, so we do not include it.

- Biology has 1 student, so we do not include it.

- Computer has 1 student, so we do not include it.

Solution

```
SELECT CLASS FROM Courses GROUP BY CLASS HAVING count(student)>=5
```

2)Find followers count

Followers table:

```
+---------+-------------+
| user_id | follower_id |
+---------+-------------+
| 0       | 1           |
| 1       | 0           |
| 2       | 0           |
| 2       | 1           |
+---------+-------------+
```

(user_id, follower_id) is the primary key (combination of columns with unique values) for this table.

This table contains the IDs of a user and a follower in a social media app where the follower follows the user.

Write a solution that will, for each user, return the number of followers.

Return the result table ordered by user_id in ascending order.

Output:

```
+---------+----------------+
| user_id | followers_count|
+---------+----------------+
| 0       | 1              |
| 1       | 1              |
| 2       | 2              |
+---------+----------------+
```

Explanation:

The followers of 0 are {1}

The followers of 1 are {0}

The followers of 2 are {0,1}

Solution

```
SELECT user_id,  count(follower_id) AS followers_count FROM Followers GROUP BY user_id

ORDER BY user_id ASC;
```

3)Find Biggest single number

MyNumbers table:

```
+------+
| num |
+------+
| 8   |
| 8   |
| 3   |
| 3   |
| 1   |
| 4   |
| 5   |
| 6   |
+------+
```

A **single number** is a number that appeared only once in the MyNumbers table.

Find the largest **single number**. If there is no **single number**, report null.

The result format is in the following example.

Output:

```
+------+
| num |
+------+
| 6   |
+------+
```

Explanation: The single numbers are 1, 4, 5, and 6.

Since 6 is the largest single number, we return it.

Solution

```sql
SELECT max(num) AS num FROM MyNumbers

WHERE num IN

   (SELECT num  FROM MyNumbers GROUP BY NUM   HAVING COUNT(NUM)=1)
```

4)Find Customers who bought all products

Customer table:

```
+-------------+-------------+
| customer_id | product_key |
+-------------+-------------+
| 1           | 5           |
| 2           | 6           |
| 3           | 5           |
| 3           | 6           |
| 1           | 6           |
+-------------+-------------+
```

This table may contain duplicates rows.

customer_id is not NULL.

product_key is a foreign key (reference column) to Product table.

Table: Product

```
+-------------+
| product_key |
+-------------+
| 5           |
| 6           |
+-------------+
```

product_key is the primary key (column with unique values) for this table.

Write a solution to report the customer ids from the Customer table that bought all the products in the Product table.

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

Output:

```
+-------------+
| customer_id |
+-------------+
| 1           |
| 3           |
+-------------+
```

Solution

```
SELECT customer_id FROM Customer GROUP BY customer_id

HAVING COUNT(DISTINCT product_key) =  (SELECT COUNT(*) FROM Product );
```

Explanation:

The customers who bought all the products (5 and 6) are customers with IDs 1 and 3.

1. **COUNT(DISTINCT product_key)**:
   - This counts the number of **unique** products purchased by each customer.
   - It ensures that duplicate purchases of the same product are not counted multiple times.
2. **(SELECT COUNT(*) FROM Product)**:
   - This gives the total number of unique products available in the Product table.
3. **HAVING Clause**:
   - The HAVING clause filters customers whose count of distinct products matches the total number of products in the Product table.

5)Find the Number of Employees Reporting to Each Manager with Average Age

Employees table:

employee_id	**name**	**reports_to**	**age**
9	Hercy	null	43
6	Alice	9	41
4	Bob	9	36
2	Winston	null	37

employee_id is the column with unique values for this table.

This table contains information about the employees and the id of the manager they report to. Some employees do not report to anyone (reports_to is null).

For this problem, we will consider a **manager** an employee who has at least 1 other employee reporting to them.

Write a solution to report the ids and the names of all managers, the number of employees who report directly to them, and the average age of the reports rounded to the nearest integer.

Return the result table ordered by employee_id.

Output:

Here is your data in table format:

employee_id	name	reports_count	average_age
9	Hercy	2	39

Solution

```
SELECT e1.employee_id,  e1.name , COUNT(e2.employee_id) AS reports_count, round(avg(e2.age), 0) AS average_age

FROM Employees e1

JOIN Employees e2 ON e1.employee_id=e2.reports_to

GROUP BY employee_id ORDER BY employee_id;
```

Explanation:

Hercy has 2 people report directly to him, Alice and Bob. Their average age is (41+36)/2 = 38.5, which is 39 after rounding it to the nearest integer.

6)Find the the Primary Department for Each Employee

Employee table:

employee_id	**department_id**	**primary_flag**
1	1	N
2	1	Y
2	2	N
3	3	N
4	2	N
4	3	Y
4	4	N

(employee_id, department_id) is the primary key (combination of columns with unique values) for this table.

employee_id is the id of the employee.

department_id is the id of the department to which the employee belongs.

primary_flag is an ENUM (category) of type ('Y', 'N'). If the flag is 'Y', the department is the primary department for the employee. If the flag is 'N', the department is not the primary.

Employees can belong to multiple departments. When the employee joins other departments, they need to decide which department is their primary department. Note that when an employee belongs to only one department, their primary column is 'N'.

Write a solution to report all the employees with their primary department. For employees who belong to one department, report their only department. Return the result table in **any order**.

Output:

employee_id	department_id
1	1
2	1
3	3
4	3

Solution

```sql
SELECT employee_id, department_id FROM Employee WHERE primary_flag = 'Y'

UNION

SELECT employee_id,  department_id FROM Employee GROUP BY employee_id HAVING COUNT(*) = 1;
```

Explanation:

- The Primary department for employee 1 is 1.

- The Primary department for employee 2 is 1.

- The Primary department for employee 3 is 3.

- The Primary department for employee 4 is 3.

7)To find if three sides form a triangle

Triangle table:

```
+----+----+----+
| x  | y  | z  |
+----+----+----+
| 13 | 15 | 30 |
| 13 | 15 | 30 |
| 10 | 20 | 15 |
+----+----+----+
```

(x, y, z) is the primary key column for this table.

Each row of this table contains the lengths of three line segments.

Report for every three line segments whether they can form a triangle.

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

Output:

```
+----+----+----+----------+
| x  | y  | z  | triangle |
+----+----+----+----------+
| 13 | 15 | 30 | No       |
| 10 | 20 | 15 | Yes      |
+----+----+----+----------+
```

Solution

```
SELECT x, y,  z,  IF(x+y>z  AND y+z>x AND z+x>y, "Yes", "No") AS triangle FROM Triangle
```

Explanation:

- The IF statement checks whether the three sides satisfy the Triangle Inequality Theorem.
- If all conditions are true, it returns 'Yes'; otherwise, it returns 'No'

8) Find all numbers that appear at least three times consecutively.

Logs table:

```
+----+-----+
| id | num |
+----+-----+
| 1  | 1   |
| 2  | 1   |
| 3  | 1   |
| 4  | 2   |
| 5  | 1   |
| 6  | 2   |
| 7  | 2   |
+----+-----+
```

Find all numbers that appear at least three times consecutively. Return the result table in **any order**.

Output:

```
+-----------------+
| ConsecutiveNums |
+-----------------+
| 1               |
| 1          |
+-----------------+
```

Explanation: 1 is the only number that appears consecutively for at least three times.

Solution

```
SELECT DISTINCT l1.num AS ConsecutiveNums
FROM Logs l1
JOIN Logs l2 ON l1.id = l2.id - 1
JOIN Logs l3 ON l2.id = l3.id - 1
    WHERE l1.num = l2.num AND l2.num = l3.num;
```

9) Find Product price at a given date

Products table:

product_id	new_price	change_date
1	20	2019-08-14
2	50	2019-08-14
1	30	2019-08-15
1	35	2019-08-16
2	65	2019-08-17
3	20	2019-08-18

(product_id, change_date) is the primary key (combination of columns with unique values) of this table.

Each row of this table indicates that the price of some product was changed to a new price at some date.

Write a solution to find the prices of all products on 2019-08-16. Assume the price of all products before any change is 10. Return the result table in **any order**.

Output:

```
+------------+-------+
| product_id | price |
+------------+-------+
| 2          | 50    |
| 1          | 35    |
| 3         | 10    |
+------------+-------+
```

Solution

```
WITH LatestPrices AS

 ( SELECT product_id, new_price, change_date,

RANK() OVER (PARTITION BY product_id ORDER BY change_date DESC) AS rnk   FROM Products

  WHERE change_date <= '2019-08-16' )

SELECT p.product_id,  COALESCE(lp.new_price, 10) AS price

      FROM  (SELECT DISTINCT product_id   FROM Products) p

      LEFT JOIN LatestPrices lp ON p.product_id = lp.product_id AND lp.rnk = 1;
```

10) Find the last person to fit the bus

Queue table:

person_id	person_name	weight	turn
5	Alice	250	1
4	Bob	175	5
3	Alex	350	2
6	John Cena	400	3
1	Winston	500	6
2	Marie	200	4

person_id column contains unique values.

This table has the information about all people waiting for a bus.

The person_id and turn columns will contain all numbers from 1 to n, where n is the number of rows in the table.

turn determines the order of which the people will board the bus, where turn=1 denotes the first person to board and turn=n denotes the last person to board.

weight is the weight of the person in kilograms.

There is a queue of people waiting to board a bus. However, the bus has a weight limit of 1000**kilograms**, so there may be some people who cannot board.

Write a solution to find the person_name of the last person that can fit on the bus without exceeding the weight limit. The test cases are generated such that the first person does not exceed the weight limit.

Note that *only one* person can board the bus at any given turn.

Output:

```
+-------------+
| person_name |
+-------------+
| John Cena   |
+-------------+
```

Solution

```
WITH Boarding AS

 (SELECT person_name,  weight,  turn,  SUM(weight) OVER ( ORDER BY turn) AS total_weight FROM Queue)

SELECT person_name FROM Boarding WHERE total_weight =

  (SELECT MAX(total_weight)  FROM Boarding   WHERE total_weight <= 1000);
```

***Explanation*:**

The Boarding CTE calculates the **cumulative weight as people board.**

We then select the person_name where total_weight is the **maximum** value that is still within the 1000 kg limit.

FOR EG

SELECT * FROM BOARDING

person_name	weight	turn	total_weight
Alice	250	1	250
Alex	350	2	600
John Cena	400	3	1000
Marie	200	4	1200
Bob	175	5	1375
Winston	500	6	1875

11) To find the number of bank accounts for each salary category.

Accounts table:

```
+------------+--------+
| account_id | income |
+------------+--------+
| 3          | 108939 |
| 2          | 12747  |
| 8          | 87709  |
| 6          | 91796  |
+------------+--------+
```

account_id is the primary key (column with unique values) for this table.

Each row contains information about the monthly income for one bank account.

Write a solution to calculate the number of bank accounts for each salary category. The salary categories are:

- "Low Salary": All the salaries **strictly less** than $20000.
- "Average Salary": All the salaries in the **inclusive** range [$20000, $50000].
- "High Salary": All the salaries **strictly greater** than $50000.

The result table **must** contain all three categories. If there are no accounts in a category, return 0.

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

Output:

```
+----------------+----------------+
| category       | accounts_count |
+----------------+----------------+
| Low Salary     | 1              |
| Average Salary | 0              |
| High Salary    | 3              |
+----------------+----------------+
```

Solution

```
SELECT 'Low Salary' AS category, COUNT(*) AS accounts_count FROM Accounts WHERE income < 20000

UNION

SELECT 'Average Salary' AS category, COUNT(*) AS accounts_count FROM Accounts WHERE income BETWEEN 20000 AND 50000

UNION

SELECT 'High Salary' AS category, COUNT(*) AS accounts_count FROM Accounts WHERE income > 50000
```

Explanation:

Low Salary: Account 2.

Average Salary: No accounts.

High Salary: Accounts 3, 6, and 8.

12) Find employees whose manager left the company

Employees table:

employee_id	name	manager_id	salary
3	Mila	9	60301
12	Antonella	null	31000
13	Emery	null	67084
1	Kalel	11	21241
9	Mikaela	null	50937
11	Joziah	6	28485

employee_id is the primary key for this table.

This table contains information about the employees, their salary, and the ID of their manager. Some employees do not have a manager (manager_id is null).

Find the IDs of the employees whose salary is strictly less than $30000 and whose manager left the company. When a manager leaves the company, their information is deleted from the Employees table, but the reports still have their manager_id set to the manager that left.

Return the result table ordered by employee_id.

Output:

```
+-------------+
| employee_id |
+-------------+
| 11          |
+-------------+
```

Solution

```
SELECT employee_id FROM Employees WHERE salary<30000 AND manager_id NOT IN

  (SELECT employee_id FROM Employees) ORDER BY employee_id
```

Explanation:

The employees with a salary less than $30000 are 1 (Kalel) and 11 (Joziah).

Kalel's manager is employee 11, who is still in the company (Joziah).

Joziah's manager is employee 6, who left the company because there is no row for employee 6 as it was deleted.

13) To swap the seat id of every two consecutive students.

Seat table:

```
+----+---------+
| id | student |
+----+---------+
| 1  | Abbot   |
| 2  | Doris   |
| 3  | Emerson |
| 4  | Green   |
| 5  | Jeames  |
+----+---------+
```

id is the primary key (unique value) column for this table.

Each row of this table indicates the name and the ID of a student.

The ID sequence always starts from 1 and increments continuously.

Write a solution to swap the seat id of every two consecutive students. If the number of students is odd, the id of the last student is not swapped.

Return the result table ordered by id **in ascending order**.

Output:

```
+----+---------+
| id | student |
+----+---------+
| 1  | Doris   |
| 2  | Abbot   |
| 3  | Green   |
| 4  | Emerson |
| 5  | Jeames  |
+----+---------+
```

Solution

```
SELECT CASE

    WHEN mod(id, 2)=1 AND id+1<= (SELECT max(id)  FROM seat) THEN id+1

    WHEN mod(id, 2)=0 THEN id-1

    ELSE id

  END AS id, student FROM Seat ORDER BY id
```

Explanation:

1. id % 2 = 1 AND id + 1 <= (SELECT MAX(id) FROM Seat) THEN id + 1:
   - If id is odd, swap it with the next id (i.e., id + 1), only if there is a next student.
2. id % 2 = 0 THEN id - 1:
   - If id is even, swap it with the previous id (i.e., id - 1).
3. ELSE id:
   - If it's the last student in an odd-numbered list, keep their id unchanged.
4. The final ORDER BY id ensures the output maintains the correct seat order.

Alternate Solution using LAG and LEAD

```
SELECT id,

  CASE

    WHEN id % 2 = 0 THEN LAG(student) OVER (ORDER BY id)

    ELSE COALESCE(LEAD(student) OVER (ORDER BY id), student)

  END AS student

FROM Seat ORDER BY id;
```

Explanation:

CASE Statement

- **For even id values (id % 2 = 0)**:
  - Use LAG(student) to get the student value from the previous row.
- **For odd id values**:
  - Use LEAD(student) to get the student value from the next row.
  - If there is no next row (i.e., LEAD(student) returns NULL), use COALESCE() to fall back to the current row's student value.

14) To Find the movie rating

Movies table:

movie_id	**title**
1	Avengers
2	Frozen 2
3	Joker

movie_id is the primary key (column with unique values) for this table.

title is the name of the movie.

Users table:

user_id	**name**
1	Daniel
2	Monica
3	Maria
4	James

user_id is the primary key (column with unique values) for this table.

The column 'name' has unique values.

MovieRating table:

movie_id	user_id	rating	created_at
1	1	3	2020-01-12
1	2	4	2020-02-11
1	3	2	2020-02-12
1	4	1	2020-01-01
2	1	5	2020-02-17
2	2	2	2020-02-01
2	3	2	2020-03-01
3	1	3	2020-02-22
3	2	4	2020-02-25

(movie_id, user_id) is the primary key (column with unique values) for this table.

This table contains the rating of a movie by a user in their review.

created_at is the user's review date.

Write a solution 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.

Output:

```
+--------------+
| results      |
+--------------+
| Daniel       |
| Frozen 2     |
+--------------+
```

Solution

```
WITH UserRatings AS
 (SELECT name,  count(*) AS rating_count FROM Movies
        JOIN MovieRating USING(movie_id)

        JOIN Users USING(user_id)  GROUP BY user_id  ORDER BY rating_count DESC,  name ASC LIMIT 1),

   MovieRatings AS

 (SELECT TITLE, avg(rating) AS avg_rating FROM Movies

        JOIN MovieRating USING(movie_id) WHERE YEAR(created_at) = 2020  AND MONTH(created_at) = 2

        GROUP BY movie_id  ORDER BY avg_rating DESC,  title ASC LIMIT 1)

SELECT name AS results FROM UserRatings

UNION ALL

SELECT title AS results FROM MovieRatings;
```

Explanation:

Daniel and Monica have rated 3 movies ("Avengers", "Frozen 2" and "Joker") but Daniel is smaller lexicographically.

Frozen 2 and Joker have a rating average of 3.5 in February but Frozen 2 is smaller lexicographically.

15) To Find restaurant growth

Customer table:

customer_id	name	visited_on	amount
1	Jhon	2019-01-01	100
2	Daniel	2019-01-02	110
3	Jade	2019-01-03	120
4	Khaled	2019-01-04	130
5	Winston	2019-01-05	110
6	Elvis	2019-01-06	140
7	Anna	2019-01-07	150
8	Maria	2019-01-08	80
9	Jaze	2019-01-09	110
1	Jhon	2019-01-10	130
3	Jade	2019-01-10	150

(customer_id, visited_on) is the primary key for this table.

This table contains data about customer transactions in a restaurant.

visited_on is the date on which the customer with ID (customer_id) has visited the restaurant.

amount is the total paid by a customer.

You are the restaurant owner and you want to analyze a possible expansion (there will be at least one customer every day).

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 the result table ordered by visited_on **in ascending order**.

Output:

visited_on	amount	average_amount
2019-01-07	860	122.86
2019-01-08	840	120
2019-01-09	840	120
2019-01-10	1000	142.86

Solution

```
SELECT DISTINCT visited_on, amount,  round(amount/7, 2) AS average_amount FROM

 (SELECT visited_on,  sum(amount)

OVER(ORDER BY visited_on RANGE BETWEEN interval 6 DAY PRECEDING AND CURRENT ROW) AS amount

  FROM Customer) AS t

WHERE datediff(t.visited_on,  (SELECT min(visited_on)   FROM customer)) >= 6
```

Calculations:

1st moving average from 2019-01-01 to 2019-01-07 has an average_amount of (100 + 110 + 120 + 130 + 110 + 140 + 150)/7 = 122.86

2nd moving average from 2019-01-02 to 2019-01-08 has an average_amount of (110 + 120 + 130 + 110 + 140 + 150 + 80)/7 = 120

3rd moving average from 2019-01-03 to 2019-01-09 has an average_amount of (120 + 130 + 110 + 140 + 150 + 80 + 110)/7 = 120

4th moving average from 2019-01-04 to 2019-01-10 has an average_amount of (130 + 110 + 140 + 150 + 80 + 110 + 130 + 150)/7 = 142.86

Explanation:

a. **Subquery (t)**:

- The subquery calculates the **7-day moving sum** of the amount for each day using the SUM() window function.
- The RANGE BETWEEN INTERVAL 6 DAY PRECEDING AND CURRENT ROW clause defines the window as the current day and the 6 preceding days.

b. **Window Function**:

- **SUM(amount) OVER (ORDER BY visited_on RANGE BETWEEN INTERVAL 6 DAY PRECEDING AND CURRENT ROW)**:
  - Computes the sum of amount for the current day and the 6 days before it.

    - The RANGE clause ensures that the window is calculated based on the actual dates (visited_on), not just the rows.

c. **Main Query**:

- **SELECT DISTINCT visited_on, amount, ROUND(amount / 7, 2) AS average_amount**:
    - Selects the visited_on date, the total amount for the 7-day window, and the rounded average amount.
    - The DISTINCT keyword ensures that duplicate rows (if any) are removed.

d. **Filter Condition**:

- **WHERE DATEDIFF(t.visited_on, (SELECT MIN(visited_on) FROM Customer)) >= 6**:
    - Ensures that only dates with at least 7 days of data are included in the result.
    - DATEDIFF calculates the difference in days between the current visited_on and the earliest visited_on in the Customer table.
    - The condition >= 6 ensures that the first 6 days (which do not have enough data for a 7-day window) are excluded.

16) To find the people who have the most friends and the most friends number.

RequestAccepted table:

requester_id	accepter_id	accept_date
1	2	2016/06/03
1	3	2016/06/08
2	3	2016/06/08
3	4	2016/06/09

(requester_id, accepter_id) is the primary key (combination of columns with unique values) for this table.

This table contains the ID of the user who sent the request, the ID of the user who received the request, and the date when the request was accepted.

Write a solution to find the people who have the most friends and the most friends number.

The test cases are generated so that only one person has the most friends.

Output:

```
+----+-----+
| id | num |
+----+-----+
| 3  | 3   |
+----+-----+
```

Solution

```
WITH FriendCounts AS

 (SELECT requester_id AS id,  COUNT(*) AS num FROM RequestAccepted GROUP BY requester_id

  UNION ALL

SELECT accepter_id AS id, COUNT(*) AS num FROM RequestAccepted GROUP BY accepter_id)

SELECT id, sum(num) AS num FROM FriendCounts GROUP BY id ORDER BY num DESC LIMIT 1;
```

Explanation:

The person with id 3 is a friend of people 1, 2, and 4, so he has three friends in total, which is the most number than any others.

17) To find the Investment Income in 2016

Insurance table:

pid	tiv_2015	tiv_2016	lat	lon
1	10	5	10	10
2	20	20	20	20
3	10	30	20	20
4	10	40	40	40

pid is the primary key (column with unique values) for this table.

Each row of this table contains information about one policy where:

pid is the policyholder's policy ID.

tiv_2015 is the total investment value in 2015 and tiv_2016 is the total investment value in 2016.

lat is the latitude of the policy holder's city. It's guaranteed that lat is not NULL.

lon is the longitude of the policy holder's city. It's guaranteed that lon is not NULL.

Write a solution to report the sum of all total investment values in 2016 tiv_2016, for all policyholders who:

- have the same tiv_2015 value as one or more other policyholders, and
- are not located in the same city as any other policyholder (i.e., the (lat, lon) attribute pairs must be unique).

Round tiv_2016 to **two decimal places**.

Output:

```
+----------+
| tiv_2016 |
+----------+
| 45.00    |
+----------+
```

Solution

```
SELECT ROUND(SUM(tiv_2016), 2) AS tiv_2016 FROM Insurance

WHERE tiv_2015 IN  (

SELECT tiv_2015  FROM Insurance   GROUP BY tiv_2015    HAVING COUNT(*) > 1)

  AND (lat,  lon) IN    (

 SELECT lat,  lon   FROM Insurance  GROUP BY lat, lon HAVING COUNT(*) = 1);
```

Explanation:

The first record in the table, like the last record, meets both of the two criteria.

The tiv_2015 value 10 is the same as the third and fourth records, and its location is unique.

The second record does not meet any of the two criteria. Its tiv_2015 is not like any other policyholders and its location is the same as the third record, which makes the third record fail, too.

So, the result is the sum of tiv_2016 of the first and last record, which is 45.

18) To find Top 3 employees who are high earners in each of the departments.

Employee table:

id	name	salary	departmentId
1	Joe	85000	1
2	Henry	80000	2
3	Sam	60000	2
4	Max	90000	1
5	Janet	69000	1
6	Randy	85000	1
7	Will	70000	1

id is the primary key (column with unique values) for this table.

departmentId is a foreign key (reference column) of the ID from the Department table.

Each row of this table indicates the ID, name, and salary of an employee. It also contains the ID of their department.

Table: Department

```
+----+-------+
| id | name  |
+----+-------+
| 1  | IT    |
| 2  | Sales |
+----+-------+
```

id is the primary key (column with unique values) for this table.

Each row of this table indicates the ID of a department and its name.

A company's executives are interested in seeing who earns the most money in each of the company's departments. A **high earner** in a department is an employee who has a salary in the **top three unique**salaries for that department.

Write a solution to find the employees who are **high earners** in each of the departments.

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

Output:

Department	Employee	Salary
IT	Max	90000
IT	Joe	85000
IT	Randy	85000
IT	Will	70000
Sales	Henry	80000
Sales	Sam	60000

Solution

```
WITH RankedSalaries AS
  (SELECT d.name AS Department, e.name AS Employee, e.salary AS Salary,
        DENSE_RANK() OVER (PARTITION BY e.departmentId ORDER BY e.salary DESC) AS rnk
         FROM Employee e JOIN Department d ON e.departmentId = d.id)
SELECT Department, Employee, Salary FROM RankedSalaries WHERE rnk <= 3;
```

Explanation:

In the IT department:

- Max earns the highest unique salary

- Both Randy and Joe earn the second-highest unique salary

- Will earns the third-highest unique salary

In the Sales department:

- Henry earns the highest salary

- Sam earns the second-highest salary

- There is no third-highest salary as there are only two employees

19) To fix names in the table

Users table:

```
+----------+--------+
| user_id | name  |
+---------+-------+
| 1       | aLice |
| 2       | bOB   |
+---------+-------+
```

user_id is the primary key (column with unique values) for this table.

This table contains the ID and the name of the user. The name consists of only lowercase and uppercase characters.

Write a solution to fix the names so that only the first character is uppercase and the rest are lowercase.

Return the result table ordered by user_id.

Output:

```
+---------+-------+
| user_id | name  |
+---------+-------+
| 1       | Alice |
| 2       | Bob   |
+---------+-------+
```

Solution

```
SELECT user_id, CONCAT(upper(LEFT(name, 1)), LOWER(SUBSTRING(name, 2))) AS name

FROM Users ORDER BY user_id
```

20) To find the patients with condition

Table: Patients

patient_id	patient_name	conditions
1	Daniel	YFEV COUGH
2	Alice	
3	Bob	DIAB100 MYOP
4	George	ACNE DIAB100
5	Alain	DIAB201

patient_id is the primary key (column with unique values) for this table.

'conditions' contains 0 or more code separated by spaces.

This table contains information of the patients in the hospital.

Write a solution to find the patient_id, patient_name, and conditions of the patients who have Type I Diabetes. Type I Diabetes always starts with DIAB1 prefix.

Output:

patient_id	patient_name	conditions
3	Bob	DIAB100 MYOP
4	George	ACNE DIAB100

Solution

```
SELECT patient_id, patient_name, conditions FROM Patients
    WHERE conditions LIKE '% DIAB1%'  OR conditions LIKE 'DIAB1%'
```

Explanation: Bob and George both have a condition that starts with DIAB1.

21) To delete duplicate emails

Person table:

```
+----+------------------+
| id | email            |
+----+------------------+
| 1  | john@example.com |
| 2  | bob@example.com  |
| 3  | john@example.com |
+----+------------------+
```

id is the primary key (column with unique values) for this table.

Each row of this table contains an email. The emails will not contain uppercase letters.

Write a solution to delete all duplicate emails, keeping only one unique email with the smallest id.

For SQL users, please note that you are supposed to write a DELETE statement and not a SELECT one.

Output:

```
+----+------------------+
| id | email            |
+----+------------------+
| 1  | john@example.com |
| 2  | bob@example.com  |
+-----+------------------+
```

Solution

DELETE p1 FROM Person p1 JOIN Person p2 ON p1.email = p2.email AND p1.id > p2.id;

Explanation: john@example.com is repeated two times. We keep the row with the smallest Id = 1.

22) To find the second highest distinct salary

Employee table:

```
+----+--------+
| id | salary |
+----+--------+
| 1  | 100    |
| 2  | 200    |
| 3  | 300    |
+----+--------+
```

id is the primary key (column with unique values) for this table.

Each row of this table contains information about the salary of an employee.

Write a solution to find the second highest distinct salary from the Employee table. If there is no second highest salary, return null .

Output:

```
+---------------------+
| SecondHighestSalary |
+---------------------+
| 200                 |
+---------------------+
```

Example 2:

Input:

Employee table:

```
+----+--------+
| id | salary |
+----+--------+
| 1  | 100    |
+----+--------+
```

Output:

```
+---------------------+
| SecondHighestSalary |
+---------------------+
| null                |
+---------------------+
```

Solution

```
SELECT max(salary) AS SecondHighestSalary FROM Employee

    WHERE  salary< (SELECT max(salary) AS first_highest_Salary FROM Employee)
```

23) To find for each date the number of different products sold and their names.

Activities table:

sell_date	**product**
2020-05-30	Headphone
2020-06-01	Pencil
2020-06-02	Mask
2020-05-30	Basketball
2020-06-01	Bible
2020-06-02	Mask
2020-05-30	T-Shirt

There is no primary key (column with unique values) for this table. It may contain duplicates.

Each row of this table contains the product name and the date it was sold in a market.

Write a solution to find for each date the number of different products sold and their names.

The sold products names for each date should be sorted lexicographically.

Return the result table ordered by sell_date.

Output:

sell_date	**num_sold**	**products**
2020-05-30	3	Basketball, Headphone, T-shirt
2020-06-01	2	Bible, Pencil
2020-06-02	1	Mask

Solution

```
SELECT sell_date, COUNT(DISTINCT product) AS num_sold,

   GROUP_CONCAT(DISTINCT product   ORDER BY product separator ',') AS products

FROM Activities GROUP BY sell_date ORDER BY sell_date;
```

Explanation:

For 2020-05-30, Sold items were (Headphone, Basketball, T-shirt), we sort them lexicographically and separate them by a comma.

For 2020-06-01, Sold items were (Pencil, Bible), we sort them lexicographically and separate them by a comma.

For 2020-06-02, the Sold item is (Mask), we just return it.

24) To List the products ordered in a period

Products table:

product_id	**product_name**	**product_category**
1	Leetcode Solutions	Book
2	Jewels of Stringology	Book
3	HP	Laptop
4	Lenovo	Laptop
5	Leetcode Kit	T-shirt

product_id is the primary key (column with unique values) for this table.

This table contains data about the company's products.

Table: Orders

product_id	**order_date**	**unit**
1	2020-02-05	60
1	2020-02-10	70
2	2020-01-18	30
2	2020-02-11	80
3	2020-02-17	2
3	2020-02-24	3
4	2020-03-01	20
4	2020-03-04	30
4	2020-03-04	60
5	2020-02-25	50
5	2020-02-27	50
5	2020-03-01	50

This table may have duplicate rows.

product_id is a foreign key (reference column) to the Products table.

unit is the number of products ordered in order_date.

Write a solution to get the names of products that have at least 100 units ordered in February 2020and their amount. Return the result table in **any order**.

Output:

product_name	**unit**
Leetcode Solutions	130
Leetcode Kit	100

Solution

```
SELECT product_name, sum(unit) AS unit FROM Products JOIN Orders USING(product_id)

WHERE DATE_FORMAT(order_date, '%Y-%m') = '2020-02' GROUP BY product_id HAVING UNIT>=100;
```

Explanation:

Products with product_id = 1 is ordered in February a total of (60 + 70) = 130.

Products with product_id = 2 is ordered in February a total of 80.

Products with product_id = 3 is ordered in February a total of (2 + 3) = 5.

Products with product_id = 4 was not ordered in February 2020.

Products with product_id = 5 is ordered in February a total of (50 + 50) = 100.

25) To find the users with valid emails

Users table:

user_id	**name**	**mail**
1	Winston	winston@leetcode.com
2	Jonathan	jonathanisgreat
3	Annabelle	bella-@leetcode.com
4	Sally	sally.come@leetcode.com
5	Marwan	quarz#2020@leetcode.com
6	David	david69@gmail.com
7	Shapiro	.shapo@leetcode.com

user_id is the primary key (column with unique values) for this table.

This table contains information of the users signed up in a website. Some e-mails are invalid.

Write a solution 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**.

Output:

user_id	name	mail
1	Winston	winston@leetcode.com
3	Annabelle	bella-@leetcode.com
4	Sally	sally.come@leetcode.com

The mail of user 2 does not have a domain.

The mail of user 5 has the # sign which is not allowed.

The mail of user 6 does not have the leetcode domain.

The mail of user 7 starts with a period.

Solution

```
SELECT user_id,  name, mail FROM Users

WHERE mail REGEXP '^[a-zA-Z][a-zA-Z0-9_.-]*@leetcode\\.com$'
```

Explanation:

- ^[A-Za-z]: The email prefix must start with a letter (either uppercase or lowercase).
- [A-Za-z0-9_.-]*: The prefix can contain letters, digits, underscores (_), periods (.), and dashes (-). The * indicates that these characters can appear zero or more times.
- @leetcode\\.com$: The domain must be exactly @leetcode.com. The \\. is used to escape the period (.) because in regular expressions, a period matches any character. The $ ensures that the domain is at the end of the string.