LeetCode SQL Problem Solving Questions With Solutions

175. Combine Two Tables | Easy | LeetCode

Table: `Person`

```
TEXT
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| PersonId    | int     |
| FirstName   | varchar |
| LastName    | varchar |
+-------------+---------+
PersonId is the primary key column for this table.
```

Table: `Address`

```
TEXT
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| AddressId   | int     |
| PersonId    | int     |
| City        | varchar |
| State       | varchar |
+-------------+---------+
AddressId is the primary key column for this table.
```

Write a SQL query for a report that provides the following information for each person in the Person table, regardless if there is an address for each of those people:

```
TEXT
FirstName, LastName, City, State
```

Solution

```sql
SELECT p.FirstName, p.LastName, a.City, a.State
FROM Person p
LEFT JOIN Address a
ON p.PersonId = a.PersonId;
```

176. Second Highest Salary | Easy | LeetCode

Write a SQL query to get the second highest salary from the `Employee` table.

TEXT

```
+----+--------+
| Id | Salary |
+----+--------+
| 1  | 100    |
| 2  | 200    |
| 3  | 300    |
+----+--------+
```

For example, given the above Employee table, the query should return `200` as the second highest salary. If there is no second highest salary, then the query should return `null`.

TEXT

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

Solution

SQL

```sql
#Solution 1:
SELECT Max(Salary) SecondHighestSalary
FROM Employee WHERE Salary < (SELECT MAX(Salary) FROM Employee)

#Solution 2:
WITH CTE AS (SELECT DISTINCT Salary
FROM Employee
ORDER BY Salary DESC
LIMIT 2)

SELECT Salary as SecondHighestSalary
FROM CTE
ORDER BY Salary Asc
LIMIT 1;
```

```sql
#Solution 3:
WITH CTE AS
(
    SELECT Salary,
           DENSE_RANK() OVER (ORDER BY Salary DESC) AS DENSERANK
    FROM Employee
)
SELECT Salary SecondHighestSalary
FROM CTE
WHERE DENSERANK = 2;
```

177. Nth Highest Salary | Medium | LeetCode

Write a SQL query to get the nth highest salary from the Employee table.

TEXT

```
+----+--------+
| Id | Salary |
+----+--------+
| 1  | 100    |
| 2  | 200    |
| 3  | 300    |
+----+--------+
```

For example, given the above Employee table, the nth highest salary where n = 2 is 200. If there is no nth highest salary, then the query should return null.

TEXT

```
+------------------------+
| getNthHighestSalary(2) |
+------------------------+
| 200                    |
+------------------------+
```

Solution

```sql
CREATE FUNCTION getNthHighestSalary(N INT) RETURNS INT
BEGIN
SET N = N-1;
    RETURN(
        SELECT DISTINCT Salary FROM Employee ORDER BY Salary DESC
        LIMIT 1 OFFSET N
    );
END
```

178. Rank Scores | Medium | LeetCode

Write a SQL query to rank scores. If there is a tie between two scores, both should have the same ranking. Note that after a tie, the next ranking number should be the next consecutive integer value. In other words, there should be no “holes” between ranks.

```text
+----+-------+
| Id | Score |
+----+-------+
| 1  | 3.50  |
| 2  | 3.65  |
| 3  | 4.00  |
| 4  | 3.85  |
| 5  | 4.00  |
| 6  | 3.65  |
+----+-------+
```

For example, given the above `Scores` table, your query should generate the following report (order by highest score):

TEXT

```
+-------+---------+
| score | Rank    |
+-------+---------+
| 4.00  | 1       |
| 4.00  | 1       |
| 3.85  | 2       |
| 3.65  | 3       |
| 3.65  | 3       |
| 3.50  | 4       |
+-------+---------+
```

Important Note: For MySQL solutions, to escape reserved words used as column names, you can use an apostrophe before and after the keyword. For example `Rank`.

Solution

SQL

```sql
SELECT score, DENSE_RANK() OVER (ORDER By Score DESC) AS "Rank"
FROM Scores;
```

180. Consecutive Numbers | Medium | LeetCode

Table: `Logs`

TEXT

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| num         | varchar |
+-------------+---------+
id is the primary key for this table.
```

Write an SQL query to find all numbers that appear at least three times consecutively.

Return the result table in any order.

The query result format is in the following example:

```
TEXT
Logs table:
+----+-----+
| Id | Num |
+----+-----+
| 1  | 1   |
| 2  | 1   |
| 3  | 1   |
| 4  | 2   |
| 5  | 1   |
| 6  | 2   |
| 7  | 2   |
+----+-----+

Result table:
+-----------------+
| ConsecutiveNums |
+-----------------+
| 1               |
+-----------------+
1 is the only number that appears consecutively for at least three times.
```

Solution

```
SQL
SELECT a.Num as ConsecutiveNums
FROM Logs a
JOIN Logs b
ON a.id = b.id+1 AND a.num = b.num
JOIN Logs c
ON a.id = c.id+2 AND a.num = c.num;
```

```
WITH ConsecutiveNumbers AS (
    SELECT
        Num,
        LEAD(Num, 1) OVER (ORDER BY id) AS next_number,
        LEAD(Num, 2) OVER (ORDER BY id) AS third_next_number
    FROM Logs
)

SELECT DISTINCT Num
FROM ConsecutiveNumbers
WHERE Num = next_number AND next_number = third_next_number;
```

181. Employees Earning More Than Their Managers | Easy | LeetCode

The `Employee` table holds all employees including their managers. Every employee has an Id, and there is also a column for the manager Id.

```
TEXT
+----+-------+--------+-----------+
| Id | Name  | Salary | ManagerId |
+----+-------+--------+-----------+
| 1  | Joe   | 70000  | 3         |
| 2  | Henry | 80000  | 4         |
| 3  | Sam   | 60000  | NULL      |
| 4  | Max   | 90000  | NULL      |
+----+-------+--------+-----------+
```

Given the `Employee` table, write a SQL query that finds out employees who earn more than their managers. For the above table, Joe is the only employee who earns more than his manager.

```
TEXT
+----------+
| Employee |
+----------+
| Joe      |
+----------+
```

Solution

```sql
SQL
SELECT E.Name as "Employee"
FROM Employee E
JOIN Employee M
```

```
ON E.ManagerId = M.Id
AND E.Salary > M.Salary;
```

182. Duplicate Emails | Easy | LeetCode

Write a SQL query to find all duplicate emails in a table named `Person`.

TEXT

```
+----+---------+
| Id | Email   |
+----+---------+
| 1  | a@b.com |
| 2  | c@d.com |
| 3  | a@b.com |
+----+---------+
```

For example, your query should return the following for the above table:

TEXT

```
+---------+
| Email   |
+---------+
| a@b.com |
+---------+
```

Note: All emails are in lowercase.

Solution

SQL

```
#Solution- 1:
SELECT Email
FROM Person
GROUP BY Email
HAVING count(*) > 1
```

```sql
#Solution- 2:
WITH CTE AS(
SELECT Email, ROW_NUMBER() OVER(PARTITION BY Email ORDER BY Email) AS RN
    FROM Person
)

SELECT Email
FROM CTE
WHERE RN > 1;
```

183. Customers Who Never Order | Easy | LeetCode

Suppose that a website contains two tables, the `Customers` table and the `Orders` table. Write a SQL query to find all customers who never order anything.

Table: `Customers`.

```
TEXT
+----+-------+
| Id | Name  |
+----+-------+
| 1  | Joe   |
| 2  | Henry |
| 3  | Sam   |
| 4  | Max   |
+----+-------+
```

Table: `Orders`.

```
TEXT
+----+------------+
| Id | CustomerId |
+----+------------+
| 1  | 3          |
| 2  | 1          |
+----+------------+
```

Using the above tables as example, return the following:

```
TEXT
+-----------+
| Customers |
+-----------+
| Henry     |
| Max       |
+-----------+
```

Solution

```sql
SQL
#Solution- 1:
SELECT Name AS Customers
FROM Customers
LEFT JOIN Orders
ON Customers.Id = Orders.CustomerId
WHERE CustomerId IS NULL;

#Solution- 2:
SELECT Name as Customers
FROM Customers
WHERE Id NOT IN(
    SELECT CustomerId
    FROM Orders
)
```

184. Department Highest Salary | Medium | LeetCode

The `Employee` table holds all employees. Every employee has an Id, a salary, and there is also a column for the department Id.

TEXT

```
+----+-------+--------+--------------+
| Id | Name  | Salary | DepartmentId |
+----+-------+--------+--------------+
| 1  | Joe   | 70000  | 1            |
| 2  | Jim   | 90000  | 1            |
| 3  | Henry | 80000  | 2            |
| 4  | Sam   | 60000  | 2            |
| 5  | Max   | 90000  | 1            |
+----+-------+--------+--------------+
```

The `Department` table holds all departments of the company.

TEXT

```
+----+----------+
| Id | Name     |
+----+----------+
| 1  | IT       |
| 2  | Sales    |
+----+----------+
```

Write a SQL query to find employees who have the highest salary in each of the departments. For the above tables, your SQL query should return the following rows (order of rows does not matter).

TEXT

```
+------------+----------+--------+
| Department | Employee | Salary |
+------------+----------+--------+
| IT         | Max      | 90000  |
| IT         | Jim      | 90000  |
| Sales      | Henry    | 80000  |
+------------+----------+--------+
```

Explanation:
Max and Jim both have the highest salary in the IT department and Henry has the

highest salary in the Sales department.

Solution

```SQL
SELECT Department.Name AS Department, Employee.Name AS Employee, Salary
FROM Employee
JOIN Department
ON Employee.DepartmentId = Department.Id
WHERE (DepartmentId, Salary) IN(
        SELECT  DepartmentId, MAX(Salary) AS Salary
        FROM Employee
        GROUP BY DepartmentId
        );
```

185. Department Top Three Salaries | Hard | LeetCode

The `Employee` table holds all employees. Every employee has an Id, and there is also a column for the department Id.

```TEXT
+----+-------+--------+--------------+
| 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            |
+----+-------+--------+--------------+
```

The `Department` table holds all departments of the company.

```
+----+----------+
| Id | Name     |
+----+----------+
| 1  | IT       |
| 2  | Sales    |
+----+----------+
```

Write a SQL query to find employees who earn the top three salaries in each of the department. For the above tables, your SQL query should return the following rows (order of rows does not matter).

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

Explanation:
In IT department, Max earns the highest salary, both Randy and Joe earn the second highest salary, and Will earns the third highest salary. There are only two employees in the Sales department, Henry earns the highest salary while Sam earns the second highest salary.

Solution

```sql
WITH department_ranking AS (
SELECT Name AS Employee, Salary ,DepartmentId
  ,DENSE_RANK() OVER (PARTITION BY DepartmentId ORDER BY Salary DESC) AS rnk
FROM Employee
```

```
)

SELECT d.Name AS Department, r.Employee, r.Salary
FROM department_ranking AS r
JOIN Department AS d
ON r.DepartmentId = d.Id
WHERE r.rnk <= 3
ORDER BY d.Name ASC, r.Salary DESC;
```

196. Delete Duplicate Emails | Easy | LeetCode

Write a SQL query to delete all duplicate email entries in a table named `Person`, keeping only unique emails based on its smallest Id.

TEXT

```
+----+------------------+
| Id | Email            |
+----+------------------+
| 1  | john@example.com |
| 2  | bob@example.com  |
| 3  | john@example.com |
+----+------------------+
```

Id is the primary key column for this table. For example, after running your query, the above `Person` table should have the following rows:

TEXT

```
+----+------------------+
| Id | Email            |
+----+------------------+
| 1  | john@example.com |
| 2  | bob@example.com  |
+----+------------------+
```

Note:
Your output is the whole `Person` table after executing your sql. Use `delete`

statement.

Solution

```SQL
DELETE p2
FROM Person p1
JOIN Person p2
ON p1.Email = p2.Email
AND p1.id < p2.id
```

197. Rising Temperature | Easy | LeetCode

Table: `Weather`

```TEXT
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| id            | int     |
| recordDate    | date    |
| temperature   | int     |
+---------------+---------+

id is the primary key for this table.
This table contains information about the temperature in a certain day.
```

Write an SQL query to find all dates' `id` with higher temperature compared to its previous dates (yesterday).

Return the result table in any order.

The query result format is in the following example:

TEXT

```
Weather
+----+------------+-------------+
| id | recordDate | Temperature |
+----+------------+-------------+
| 1  | 2015-01-01 | 10          |
| 2  | 2015-01-02 | 25          |
| 3  | 2015-01-03 | 20          |
| 4  | 2015-01-04 | 30          |
+----+------------+-------------+
```

note in self join : we consider the first row of the table is the left table and second row of the table is the right table
to write the condition Easly with the above note

```
Result table:
+----+
| id |
+----+
| 2  |
| 4  |
+----+
```

Since each row from the original table is matched with every row where the ID is the same, you get a duplication of rows. For example, if there are 4 rows with ID=1 in the original table, and 4 rows with ID=1 in the joined table, you'll get 16 rows in the result set (4x4=16). This multiplication of rows occurs for each unique ID in the original table.

```
In 2015-01-02, temperature was higher than the previous day (10 -> 25).
In 2015-01-04, temperature was higher than the previous day (20 -> 30).
```

Solution

```
WITH ConsecutiveNumbers AS (
   SELECT ID,
      Temperature,
      LAG(Temperature, 1) OVER (ORDER BY id) AS next_temp
   FROM TemperatureRecords
)
SELECT ID
FROM ConsecutiveNumbers
WHERE  next_temp < Temperature;
```

SQL

```
#Solution- 1:
SELECT t.Id
FROM Weather AS t, Weather AS y
WHERE DATEDIFF(t.RecordDate, y.RecordDate) = 1
AND t.Temperature > y.Temperature;


#Solution- 2:
SELECT t.Id
FROM Weather t
JOIN Weather y
ON DATEDIFF(t.recordDate, y.recordDate) = 1 AND
t.temperature > y.temperature;
```

```
SELECT * FROM TemperatureRecords
SELECT y.ID,y.Temperature,y.RecordDate
FROM TemperatureRecords AS t INNER
JOIN TemperatureRecords AS y
ON y.RecordDate = DATEADD
(DAY,1,t.RecordDate) AND y.Temperature
> t.Temperature;
```

262. Trips and Users | Hard | LeetCode

Table: `Trips`

```
TEXT
+-------------+----------+
| Column Name | Type     |
+-------------+----------+
| Id          | int      |
| Client_Id   | int      |
| Driver_Id   | int      |
| City_Id     | int      |
| Status      | enum     |
| Request_at  | date     |
+-------------+----------+
Id is the primary key for this table.
The table holds all taxi trips. Each trip has a unique Id, while Client_Id and Dr
Status is an ENUM type of (‘completed’, ‘cancelled_by_driver’, ‘cancelled_by_clie
```

Table: `Users`

```
TEXT
+-------------+----------+
| Column Name | Type     |
+-------------+----------+
| Users_Id    | int      |
| Banned      | enum     |
| Role        | enum     |
+-------------+----------+
Users_Id is the primary key for this table.
The table holds all users. Each user has a unique Users_Id, and Role is an ENUM t
Status is an ENUM type of (‘Yes’, ‘No’).
```

Write a SQL query to find the cancellation rate of requests with unbanned users (both client and driver must not be banned) each day between `"2013-10-01"` and `"2013-10-03"`.

The cancellation rate is computed by dividing the number of canceled (by client or driver) requests with unbanned users by the total number of requests with unbanned users on that day.

Return the result table in any order. Round `Cancellation Rate` to two decimal points.

The query result format is in the following example:

TEXT

```
Trips table:
+----+-----------+-----------+---------+---------------------+------------+
| Id | Client_Id | Driver_Id | City_Id | Status              | Request_at |
+----+-----------+-----------+---------+---------------------+------------+
| 1  | 1         | 10        | 1       | completed           | 2013-10-01 |
| 2  | 2         | 11        | 1       | cancelled_by_driver | 2013-10-01 |
| 3  | 3         | 12        | 6       | completed           | 2013-10-01 |
| 4  | 4         | 13        | 6       | cancelled_by_client | 2013-10-01 |
| 5  | 1         | 10        | 1       | completed           | 2013-10-02 |
| 6  | 2         | 11        | 6       | completed           | 2013-10-02 |
| 7  | 3         | 12        | 6       | completed           | 2013-10-02 |
| 8  | 2         | 12        | 12      | completed           | 2013-10-03 |
| 9  | 3         | 10        | 12      | completed           | 2013-10-03 |
| 10 | 4         | 13        | 12      | cancelled_by_driver | 2013-10-03 |
+----+-----------+-----------+---------+---------------------+------------+

Users table:
+----------+--------+--------+
| Users_Id | Banned | Role   |
+----------+--------+--------+
| 1        | No     | client |
| 2        | Yes    | client |
| 3        | No     | client |
| 4        | No     | client |
| 10       | No     | driver |
| 11       | No     | driver |
| 12       | No     | driver |
| 13       | No     | driver |
+----------+--------+--------+

Result table:
```

```
+------------+------------------+
| Day        | Cancellation Rate |
+------------+------------------+
| 2013-10-01 | 0.33              |
| 2013-10-02 | 0.00              |
| 2013-10-03 | 0.50              |
+------------+------------------+
```

On 2013-10-01:

- There were 4 requests in total, 2 of which were canceled.
- However, the request with Id=2 was made by a banned client (User_Id=2), so it i
- Hence there are 3 unbanned requests in total, 1 of which was canceled.
- The Cancellation Rate is (1 / 3) = 0.33

On 2013-10-02:

- There were 3 requests in total, 0 of which were canceled.
- The request with Id=6 was made by a banned client, so it is ignored.
- Hence there are 2 unbanned requests in total, 0 of which were canceled.
- The Cancellation Rate is (0 / 2) = 0.00

On 2013-10-03:

- There were 3 requests in total, 1 of which was canceled.
- The request with Id=8 was made by a banned client, so it is ignored.
- Hence there are 2 unbanned request in total, 1 of which were canceled.
- The Cancellation Rate is (1 / 2) = 0.50

Solution

SQL

```sql
SELECT Request_at AS Day,
ROUND(SUM(IF(Status<>"completed", 1, 0))/COUNT(Status),2) AS "Cancellation Rate"
FROM Trips
WHERE Request_at BETWEEN "2013-10-01" AND "2013-10-03"
AND Client_Id NOT IN (SELECT Users_Id FROM Users WHERE Banned = 'Yes')
AND Driver_Id NOT IN (SELECT Users_Id FROM Users WHERE Banned = 'Yes')
GROUP BY Request_at;
```

511. Game Play Analysis I | Easy | LeetCode

Table: `Activity`

TEXT

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| player_id    | int     |
| device_id    | int     |
| event_date   | date    |
| games_played | int     |
+--------------+---------+
(player_id, event_date) is the primary key of this table.
This table shows the activity of players of some game.
Each row is a record of a player who logged in and played a number of games (poss
```

Write an SQL query that reports the **first login date** for each player.

The query result format is in the following example:

TEXT

```
Activity table:
+-----------+-----------+------------+--------------+
| player_id | device_id | event_date | games_played |
+-----------+-----------+------------+--------------+
| 1         | 2         | 2016-03-01 | 5            |
| 1         | 2         | 2016-05-02 | 6            |
| 2         | 3         | 2017-06-25 | 1            |
| 3         | 1         | 2016-03-02 | 0            |
| 3         | 4         | 2018-07-03 | 5            |
+-----------+-----------+------------+--------------+

Result table:
+-----------+-------------+
| player_id | first_login |
+-----------+-------------+
| 1         | 2016-03-01  |
```

```
| 2         | 2017-06-25  |
| 3         | 2016-03-02  |
+-----------+-------------+
```

Solution

```SQL
SELECT player_id, MIN(event_date) as first_login
FROM Activity
GROUP BY player_id
```

512. Game Play Analysis II | Easy | 🔒 LeetCode

Table: `Activity`

```TEXT
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| player_id    | int     |
| device_id    | int     |
| event_date   | date    |
| games_played | int     |
+--------------+---------+
(player_id, event_date) is the primary key of this table.
This table shows the activity of players of some game.
Each row is a record of a player who logged in and played a number of games (poss
```

Write a SQL query that reports the device that is first logged in for each player.

The query result format is in the following example:

TEXT

```
Activity table:
+-----------+-----------+------------+--------------+
| player_id | device_id | event_date | games_played |
+-----------+-----------+------------+--------------+
| 1         | 2         | 2016-03-01 | 5            |
| 1         | 2         | 2016-05-02 | 6            |
| 2         | 3         | 2017-06-25 | 1            |
| 3         | 1         | 2016-03-02 | 0            |
| 3         | 4         | 2018-07-03 | 5            |
+-----------+-----------+------------+--------------+

Result table:
+-----------+-----------+
| player_id | device_id |
+-----------+-----------+
| 1         | 2         |
| 2         | 3         |
| 3         | 1         |
+-----------+-----------+
```

Solution

SQL

```sql
#Solution- 1:
SELECT DISTINCT player_id, device_id
FROM Activity
WHERE (player_id, event_date) in (
    SELECT player_id, min(event_date)
    FROM Activity
    GROUP BY player_id)

#Solution- 2:
SELECT a.player_id, b.device_id
FROM
(SELECT player_id, MIN(event_date) AS event_date FROM Activity
GROUP BY player_id) a
JOIN Activity b
```

```
ON a.player_id = b.player_id AND a.event_date = b.event_date;

#Solution- 3:
SELECT player_id, device_id
FROM
(SELECT player_id, device_id, event_date,
ROW_NUMBER() OVER (PARTITION BY player_id ORDER BY event_date) AS r
FROM Activity) lookup
WHERE r = 1;
```

534. Game Play Analysis III | Medium | 🔒 LeetCode

Table: `Activity`

```
TEXT
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| player_id    | int     |
| device_id    | int     |
| event_date   | date    |
| games_played | int     |
+--------------+---------+
(player_id, event_date) is the primary key of this table.
This table shows the activity of players of some game.
Each row is a record of a player who logged in and played a number of games (poss
```

Write an SQL query that reports for each player and date, how many games played so far by the player. That is, the total number of games played by the player until that date. Check the example for clarity.

The query result format is in the following example:

```
TEXT
Activity table:
+-----------+-----------+------------+--------------+
```

```
| player_id | device_id | event_date | games_played |
+-----------+-----------+------------+--------------+
| 1         | 2         | 2016-03-01 | 5            |
| 1         | 2         | 2016-05-02 | 6            |
| 1         | 3         | 2017-06-25 | 1            |
| 3         | 1         | 2016-03-02 | 0            |
| 3         | 4         | 2018-07-03 | 5            |
+-----------+-----------+------------+--------------+

Result table:
+-----------+------------+---------------------+
| player_id | event_date | games_played_so_far |
+-----------+------------+---------------------+
| 1         | 2016-03-01 | 5                   |
| 1         | 2016-05-02 | 11                  |
| 1         | 2017-06-25 | 12                  |
| 3         | 2016-03-02 | 0                   |
| 3         | 2018-07-03 | 5                   |
+-----------+------------+---------------------+
For the player with id 1, 5 + 6 = 11 games played by 2016-05-02, and 5 + 6 + 1 =
For the player with id 3, 0 + 5 = 5 games played by 2018-07-03.
Note that for each player we only care about the days when the player logged in.
```

Solution

SQL

```sql
#Solution- 1:
SELECT t1.player_id, t1.event_date, SUM(t2.games_played) as games_played_so_far
FROM Activity t1
JOIN Activity t2
ON t1.player_id = t2.player_id
WHERE t1.event_date >= t2.event_date
GROUP BY t1.player_id, t1.event_date;

#Solution- 2:
SELECT player_id, event_date,
```

```
SUM(games_played) OVER (PARTITION BY player_id ORDER BY event_date) AS games_play
FROM Activity;
```

we can use this solution it in any running total problem

550. Game Play Analysis IV | Medium | 🔒 LeetCode

Table: Activity

TEXT

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| player_id    | int     |
| device_id    | int     |
| event_date   | date    |
| games_played | int     |
+--------------+---------+
(player_id, event_date) is the primary key of this table.
This table shows the activity of players of some game.
Each row is a record of a player who logged in and played a number of games (poss
```

Write an SQL query that reports 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.

The query result format is in the following example:

TEXT

```
Activity table:
+-----------+-----------+------------+--------------+
| player_id | device_id | event_date | games_played |
+-----------+-----------+------------+--------------+
| 1         | 2         | 2016-03-01 | 5            |
| 1         | 2         | 2016-03-02 | 6            |
| 2         | 3         | 2017-06-25 | 1            |
```

```
| 3         | 1         | 2016-03-02 | 0            |
| 3         | 4         | 2018-07-03 | 5            |
+-----------+-----------+------------+--------------+

Result table:
+-----------+
| fraction  |
+-----------+
| 0.33      |
+-----------+
Only the player with id 1 logged back in after the first day he had logged in so
```

SELECT ROUND(SUM(CASE WHEN tl.event_date = DATEADD(DAY, 1, t2.first_event) THEN 1 ELSE 0 END) * 1.0 / COUNT(DISTINCT tl.player_id), 2) AS fraction
FROM Activity tl
JOIN (
    SELECT player_id, MIN(event_date) as first_event
    FROM Activity
    GROUP BY player_id
) t2 ON tl.player_id = t2.player_id;

Solution

SQL

```sql
#Solution- 1
SELECT ROUND(sum(CASE WHEN t1.event_date = t2.first_event+1 THEN 1 ELSE 0 END)/CO
FROM Activity t1
JOIN
    (SELECT player_id, MIN(event_date) AS first_event
    FROM Activity
    GROUP BY player_id) t2
    ON t1.player_id = t2.player_id;

#Solution- 2:
SELECT ROUND(COUNT(DISTINCT b.player_id)/COUNT(DISTINCT a.player_id),2) AS fracti
FROM
  (SELECT player_id, MIN(event_date) AS event_date FROM Activity
  GROUP BY player_id) a
  LEFT JOIN Activity b
  ON a.player_id = b.player_id AND a.event_date+1 = b.event_date;
```

569. Median Employee Salary | Hard | LeetCode

The `Employee` table holds all employees. The employee table has three columns: Employee Id, Company Name, and Salary.

```
+-----+------------+--------+
|Id   | Company    | Salary |
+-----+------------+--------+
|1    | A          | 2341   |
|2    | A          | 341    |
|3    | A          | 15     |
|4    | A          | 15314  |
|5    | A          | 451    |
|6    | A          | 513    |
|7    | B          | 15     |
|8    | B          | 13     |
|9    | B          | 1154   |
|10   | B          | 1345   |
|11   | B          | 1221   |
|12   | B          | 234    |
|13   | C          | 2345   |
|14   | C          | 2645   |
|15   | C          | 2645   |
|16   | C          | 2652   |
|17   | C          | 65     |
+-----+------------+--------+
```

Write a SQL query to find the median salary of each company. Bonus points if you can solve it without using any built-in SQL functions.

```
+-----+------------+--------+
|Id   | Company    | Salary |
+-----+------------+--------+
|5    | A          | 451    |
|6    | A          | 513    |
|12   | B          | 234    |
|9    | B          | 1154   |
|14   | C          | 2645   |
+-----+------------+--------+
```

```
WITH OrderedData AS (
    SELECT Company, Salary,
    ROW_NUMBER() OVER (PARTITION BY Company ORDER BY Salary asc,Id asc) AS RowAsc,
   ROW_NUMBER() OVER (PARTITION BY Company ORDER BY Salary DESC,id DESC) AS RowDesc
    FROM employee
)
SELECT *
FROM OrderedData
WHERE RowAsc = RowDesc OR RowAsc + 1 = RowDesc OR RowAsc = RowDesc + 1

--or
where RowAsc IN (RowDesc, RowDesc - 1, RowDesc + 1)
```

--calculate median for the odd

SELECT DISTINCT PERCENTILE_DISC(.5)
  WITHIN GROUP (ORDER BY Salary) OVER(PARTITION BY Company) AS "Median"
FROM dbo.employee

Solution

```sql
SELECT t1.Id AS Id, t1.Company, t1.Salary
FROM Employee AS t1 JOIN Employee AS t2
ON t1.Company = t2.Company
GROUP BY t1.Id
HAVING abs(sum(CASE WHEN t2.Salary<t1.Salary THEN 1
                    WHEN t2.Salary>t1.Salary THEN -1
                    WHEN t2.Salary=t1.Salary AND t2.Id<t1.Id THEN 1
                    WHEN t2.Salary=t1.Salary AND t2.Id>t1.Id THEN -1
                    ELSE 0 END)) <= 1
ORDER BY t1.Company, t1.Salary, t1.Id
```

570. Managers with at Least 5 Direct Reports | Medium | 🔒 LeetCode

The `Employee` table holds all employees including their managers. Every employee has an Id, and there is also a column for the manager Id.

```
+------+----------+-----------+----------+
|Id    |Name      |Department |ManagerId |
+------+----------+-----------+----------+
|101   |John      |A          |null      |
|102   |Dan       |A          |101       |
|103   |James     |A          |101       |
|104   |Amy       |A          |101       |
|105   |Anne      |A          |101       |
|106   |Ron       |B          |101       |
+------+----------+-----------+----------+
```

Given the `Employee` table, write a SQL query that finds out managers with at least 5 direct report. For the above table, your SQL query should return:

```
SELECT e1.Name AS ManagerName
FROM Employeemg e1
JOIN Employeemg e2 ON e1.ID = e2.ManagerID
GROUP BY e1.Name
HAVING COUNT(DISTINCT e2.ID) >= 5;
```

TEXT

```
+-------+
| Name  |
+-------+
| John  |
+-------+
```

Note: No one would report to himself.

Solution

SQL

```sql
SELECT Name
FROM Employee
WHERE id IN
   (SELECT ManagerId
    FROM Employee
    GROUP BY ManagerId
    HAVING COUNT(DISTINCT Id) >= 5)
```

571. Find Median Given Frequency of Numbers | 🔒 LeetCode

The `Numbers` table keeps the value of number and its frequency.

TEXT

```
+----------+-------------+
|  Number  |  Frequency  |
+----------+-------------|
|  0       |  7          |
|  1       |  1          |
|  2       |  3          |
|  3       |  1          |
+----------+-------------+
```

In this table, the numbers are 0, 0, 0, 0, 0, 0, 0, 1, 2, 2, 2, 3, so the median is (0 + 0) / 2 = 0.

```
TEXT
+--------+
| median |
+--------|
| 0.0000 |
+--------+
```

Write a query to find the median of all numbers and name the result as median.

Solution

```sql
SQL
SELECT avg(t3.Number) as median
FROM Numbers as t3
JOIN
    (SELECT t1.Number,
        abs(SUM(CASE WHEN t1.Number>t2.Number THEN t2.Frequency ELSE 0 END) -
            SUM(CASE WHEN t1.Number<t2.Number THEN t2.Frequency ELSE 0 END)) AS c
    FROM numbers AS t1, numbers AS t2
    GROUP BY t1.Number) AS t4
ON t3.Number = t4.Number
WHERE t3.Frequency>=t4.count_diff
```

---

574. Winning Candidate | Medium | 🔒 LeetCode

Table: `Candidate`

```
TEXT
+-----+---------+
| id  | Name    |
+-----+---------+
| 1   | A       |
```

```
| 2   | B       |
| 3   | C       |
| 4   | D       |
| 5   | E       |
+-----+---------+
```

Table: `Vote`

```
TEXT
+-----+--------------+
| id  | CandidateId  |
+-----+--------------+
| 1   |     2        |
| 2   |     4        |
| 3   |     3        |
| 4   |     2        |
| 5   |     5        |
+-----+--------------+
```

`id` is the auto-increment primary key, `CandidateId` is the id appeared in Candidate table. Write a sql to find the name of the winning candidate, the above example will return the winner B.

```
TEXT
+------+
| Name |
+------+
| B    |
+------+
```

Notes: You may assume there is no tie, in other words there will be at most one winning candidate.

Solution

SQL

```sql
SELECT Name
FROM Candidate
WHERE id = (SELECT CandidateId
      FROM Vote
      GROUP BY CandidateId
      ORDER BY COUNT(1) desc
      LIMIT 1)

## Assumption: if we have two candidates with the same votes, we choose the one w
# SELECT Name
# FROM Candidate JOIN
# . . . (SELECT CandidateId
# . . . FROM Vote
# . . . GROUP BY CandidateId
# . . . ORDER BY count(1) DESC
# . . . LIMIT 1) AS t
# ON Candidate.id = t.CandidateId
```

577. Employee Bonus | Easy | 🔒 LeetCode

Select all employee's name and bonus whose bonus is < 1000.

Table:Employee

TEXT

```
+-------+--------+-----------+--------+
| empId |  name  | supervisor| salary |
+-------+--------+-----------+--------+
|   1   | John   |  3        | 1000   |
|   2   | Dan    |  3        | 2000   |
|   3   | Brad   |  null     | 4000   |
|   4   | Thomas |  3        | 4000   |
+-------+--------+-----------+--------+
empId is the primary key column for this table.
```

Table: Bonus

```
+-------+-------+
| empId | bonus |
+-------+-------+
| 2     | 500   |
| 4     | 2000  |
+-------+-------+
empId is the primary key column for this table.
```

Example ouput:

```
+-------+-------+
| name  | bonus |
+-------+-------+
| John  | null  |
| Dan   | 500   |
| Brad  | null  |
+-------+-------+
```

Solution

```sql
SELECT name, bonus
FROM Employee LEFT JOIN Bonus
ON Employee.empId = Bonus.empId
WHERE bonus<1000 OR bonus IS NULL;
```

578. Get Highest Answer Rate Question | Medium | 🔒 LeetCode

Get the highest answer rate question from a table survey*log with these columns: uid, action, question*id, answer*id, q*num, timestamp.

uid means user id; action has these kind of values: “show”, “answer”, “skip”; answer*id is not null when action column is “answer”, while is null for “show” and “skip”*; *q*num is the numeral order of the question in current session.

Write a sql query to identify the question which has the highest answer rate.

Example: Input:

```
TEXT
+------+-----------+--------------+------------+-----------+------------+
| uid  | action    | question_id  | answer_id  | q_num     | timestamp  |
+------+-----------+--------------+------------+-----------+------------+
| 5    | show      | 285          | null       | 1         | 123        |
| 5    | answer    | 285          | 124124     | 1         | 124        |
| 5    | show      | 369          | null       | 2         | 125        |
| 5    | skip      | 369          | null       | 2         | 126        |
+------+-----------+--------------+------------+-----------+------------+
```

Output:

```
TEXT
+-------------+
| survey_log  |
+-------------+
|    285      |
+-------------+
```

```
WITH T AS (
    SELECT
        question_id AS survey_log,
        SUM(CASE WHEN action = 'answer' THEN 1 ELSE 0 END) OVER
(PARTITION BY question_id) /
        NULLIF(SUM(CASE WHEN action = 'show' THEN 1 ELSE 0 END)
OVER (PARTITION BY question_id), 0) AS ratio
    FROM SurveyLog
)
SELECT TOP 1 survey_log
FROM T
ORDER BY ratio DESC, survey_log;
```

Explanation: question 285 has answer rate 1/1, while question 369 has 0/1 answer rate, so output 285.

Note: The highest answer rate meaning is: answer number’s ratio in show number in the same question.

Solution

```SQL
#Solution- 1::
SELECT question_id AS survey_log FROM
(SELECT question_id,
        SUM(IF(action='show', 1, 0)) AS num_show,
        SUM(IF(action='answer', 1, 0)) AS num_answer
 FROM survey_log GROUP BY question_id) AS t
ORDER BY (num_answer/num_show) DESC LIMIT 1;

#Solution- 2:
SELECT question_id AS survey_log
FROM (SELECT question_id,
      sum(CASE WHEN  action='show' THEN 1 ELSE 0 END) AS show_count,
      sum(CASE WHEN  action='answer' THEN 1 ELSE 0 END) AS answer_count
    FROM survey_log
    GROUP BY question_id) AS t
ORDER BY answer_count/show_count DESC LIMIT 1;
```

579. Find Cumulative Salary of an Employee | Hard | 🔒 LeetCode

The `Employee` table holds the salary information in a year.

Write a SQL to get the cumulative sum of an employee's salary over a period of 3 months but exclude the most recent month.

The result should be displayed by 'Id' ascending, and then by 'Month' descending.

Example Input

```TEXT
| Id | Month | Salary |
|----|-------|--------|
| 1  | 1     | 20     |
| 2  | 1     | 20     |
| 1  | 2     | 30     |
| 2  | 2     | 30     |
| 3  | 2     | 40     |
```

```
| 1  | 3    | 40    |
| 3  | 3    | 60    |
| 1  | 4    | 60    |
| 3  | 4    | 70    |
```

Output

```TEXT
| Id | Month | Salary |
|----|-------|--------|
| 1  | 3     | 90     |
| 1  | 2     | 50     |
| 1  | 1     | 20     |
| 2  | 1     | 20     |
| 3  | 3     | 100    |
| 3  | 2     | 40     |
```

Explanation Employee ‘1’ has 3 salary records for the following 3 months except the most recent month ‘4’: salary 40 for month ‘3’, 30 for month ‘2’ and 20 for month ‘1’ So the cumulative sum of salary of this employee over 3 months is 90(40+30+20), 50(30+20) and 20 respectively.

```TEXT
| Id | Month | Salary |
|----|-------|--------|
| 1  | 3     | 90     |
| 1  | 2     | 50     |
| 1  | 1     | 20     |
```

Employee ‘2’ only has one salary record (month ‘1’) except its most recent month ‘2’.

```TEXT
| Id | Month | Salary |
|----|-------|--------|
| 2  | 1     | 20     |
```

Employ ‘3’ has two salary records except its most recent pay month ‘4’: month ‘3’ with 60 and month ‘2’ with 40. So the cumulative salary is as following.

```
TEXT
| Id | Month | Salary |
|----|-------|--------|
| 3  | 3     | 100    |
| 3  | 2     | 40     |
```

Solution

```sql
SQL
SELECT
    a.id,
    a.month,
    SUM(b.salary) Salary
FROM
    Employee a JOIN Employee b ON
    a.id = b.id AND
    a.month - b.month >= 0 AND
    a.month - b.month < 3
GROUP BY
    a.id, a.month
HAVING
    (a.id, a.month) NOT IN (SELECT id, MAX(month) FROM Employee GROUP BY id)
ORDER BY
    a.id, a.month DESC
```

580. Count Student Number in Departments | Medium | 🔒 LeetCode

A university uses 2 data tables, `student` and `department`, to store data about its students and the departments associated with each major.

Write a query to print 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).

Sort your results by descending number of students; if two or more departments have the same number of students, then sort those departments alphabetically by department name.

The `student` is described as follow:

```
TEXT
| Column Name  | Type      |
|--------------|-----------|
| student_id   | Integer   |
| student_name | String    |
| gender       | Character |
| dept_id      | Integer   |
```

where student*id is the student's ID number, student*name is the student's name, gender is their gender, and dept_id is the department ID associated with their declared major.

And the department table is described as below:

```
TEXT
| Column Name | Type    |
|-------------|---------|
| dept_id     | Integer |
| dept_name   | String  |
```

where dept*id is the department's ID number and dept*name is the department name.

Here is an example input: `student` table:

```
TEXT
| student_id | student_name | gender | dept_id |
|------------|--------------|--------|---------|
| 1          | Jack         | M      | 1       |
| 2          | Jane         | F      | 1       |
| 3          | Mark         | M      | 2       |
```

`department` table:

```
TEXT
| dept_id | dept_name   |
|---------|-------------|
| 1       | Engineering |
| 2       | Science     |
| 3       | Law         |
```

The Output should be:

```
TEXT
| dept_name   | student_number |
|-------------|----------------|
| Engineering | 2              |
| Science     | 1              |
| Law         | 0              |
```

Solution

```sql
SELECT dept_name,
  SUM(CASE WHEN student_id IS NULL THEN 0 ELSE 1 END) AS student_number
FROM department
LEFT JOIN student
ON department.dept_id = student.dept_id
GROUP BY department.dept_id
ORDER BY student_number DESC, dept_name
```

584. Find Customer Referee | Easy | 🔒 LeetCode

Given a table `customer` holding customers information and the referee.

```
+------+------+-----------+
| id   | name | referee_id|
+------+------+-----------+
|    1 | Will |      NULL |
|    2 | Jane |      NULL |
|    3 | Alex |         2 |
|    4 | Bill |      NULL |
|    5 | Zack |         1 |
|    6 | Mark |         2 |
+------+------+-----------+
```

Write a query to return the list of customers NOT referred by the person with id '2'.

For the sample data above, the result is:

```
+------+
| name |
+------+
| Will |
| Jane |
| Bill |
| Zack |
+------+
```

Solution

```sql
SELECT name
FROM customer
WHERE referee_id != '2' OR referee_id IS NULL;
```

585. Investments in 2016 | Medium | 🔒 LeetCode

Write a query to print the sum of all total investment values in 2016 (TIV_2016), to a scale of 2 decimal places, for all policy holders who meet the following criteria:

1. Have the same TIV_2015 value as one or more other policyholders.
2. Are not located in the same city as any other policyholder (i.e.: the (latitude, longitude) attribute pairs must be unique). Input Format: The insurance table is described as follows:

```
TEXT
| Column Name | Type          |
|-------------|---------------|
| PID         | INTEGER(11)   |
| TIV_2015    | NUMERIC(15,2) |
| TIV_2016    | NUMERIC(15,2) |
| LAT         | NUMERIC(5,2)  |
| LON         | NUMERIC(5,2)  |
```

where PID is the policyholder's policy ID, TIV*2015 is the total investment value in 2015, TIV*2016 is the total investment value in 2016, LAT is the latitude of the policy holder's city, and LON is the longitude of the policy holder's city.

Sample Input

```
TEXT
| 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  |
```

Sample Output

```
TEXT
| TIV_2016 |
|----------|
| 45.00    |
```

Explanation

```text
The first record in the table, like the last record, meets both of the two criter
The TIV_2015 value '10' is as the same as the third and forth record, and its loc

The second record does not meet any of the two criteria. Its TIV_2015 is not like

And its location is the same with the third record, which makes the third record

So, the result is the sum of TIV_2016 of the first and last record, which is 45.
```

Solution

```sql
SELECT SUM(TIV_2016) AS TIV_2016
FROM insurance
WHERE CONCAT(LAT, ',', LON)
    IN (SELECT CONCAT(LAT, ',', LON)
       FROM insurance
       GROUP BY LAT, LON
       HAVING COUNT(1) = 1)
AND TIV_2015 in
    (SELECT TIV_2015
    FROM insurance
    GROUP BY TIV_2015
    HAVING COUNT(1)>1)
```

586. Customer Placing the Largest Number of Orders | Easy | 🔒 LeetCode

Query the customer_number from the orders table for the customer who has placed the largest number of orders.

It is guaranteed that exactly one customer will have placed more orders than any other customer.

The orders table is defined as follows:

```
TEXT
| Column            | Type      |
|-------------------|-----------|
| order_number (PK) | int       |
| customer_number   | int       |
| order_date        | date      |
| required_date     | date      |
| shipped_date      | date      |
| status            | char(15)  |
| comment           | char(200) |
```

Sample Input

```
TEXT
| order_number | customer_number | order_date | required_date | shipped_date | st
|--------------|-----------------|------------|---------------|--------------|---
| 1            | 1               | 2017-04-09 | 2017-04-13    | 2017-04-12   | Cl
| 2            | 2               | 2017-04-15 | 2017-04-20    | 2017-04-18   | Cl
| 3            | 3               | 2017-04-16 | 2017-04-25    | 2017-04-20   | Cl
| 4            | 3               | 2017-04-18 | 2017-04-28    | 2017-04-25   | Cl
```

Sample Output

```
TEXT
| customer_number |
|-----------------|
| 3               |
```

Explanation

TEXT

```
The customer with number '3' has two orders, which is greater than either custome
So the result is customer_number '3'.
```

Solution

SQL

```sql
# assume: only one match
SELECT customer_number FROM orders
GROUP BY customer_number
ORDER BY COUNT(1) DESC
LIMIT 1

## assume: multiple matches
##  1 1
##  2 1
##  3 1
##
##  1 1 1 1
##  1 1 2 1
##  1 1 3 1
##
##  SELECT t1.customer_number
##  FROM (SELECT customer_number, COUNT(1) AS count
##        FROM orders GROUP BY customer_number) AS t1,
##       (SELECT customer_number, COUNT(1) AS count
##        FROM orders GROUP BY customer_number) AS t2
##  GROUP BY t1.customer_number
##  HAVING max(t1.count) = max(t2.count)
```

595. Big Countries | Easy | LeetCode

There is a table `World`

```
+-----------------+------------+------------+--------------+---------------+
| name            | continent  | area       | population   | gdp           |
+-----------------+------------+------------+--------------+---------------+
| Afghanistan     | Asia       | 652230     | 25500100     | 20343000      |
| Albania         | Europe     | 28748      | 2831741      | 12960000      |
| Algeria         | Africa     | 2381741    | 37100000     | 188681000     |
| Andorra         | Europe     | 468        | 78115        | 3712000       |
| Angola          | Africa     | 1246700    | 20609294     | 100990000     |
+-----------------+------------+------------+--------------+---------------+
```

A country is big if it has an area of bigger than 3 million square km or a population of more than 25 million.

Write a SQL solution to output big countries' name, population and area.

For example, according to the above table, we should output:

```
+--------------+-------------+--------------+
| name         | population  | area         |
+--------------+-------------+--------------+
| Afghanistan  | 25500100    | 652230       |
| Algeria      | 37100000    | 2381741      |
+--------------+-------------+--------------+
```

Solution

```sql
SELECT name, population, area
FROM World
WHERE area >= 3000000 OR population > 25000000;
```

596. Classes More Than 5 Students | Easy | LeetCode

There is a table `courses` with columns: **student** and **class**

Please list out all classes which have more than or equal to 5 students.

For example, the table:

```
TEXT
+---------+------------+
| student | class      |
+---------+------------+
| A       | Math       |
| B       | English    |
| C       | Math       |
| D       | Biology    |
| E       | Math       |
| F       | Computer   |
| G       | Math       |
| H       | Math       |
| I       | Math       |
+---------+------------+
```

Should output:

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

Solution

```
SQL
SELECT class
FROM courses
GROUP BY class
HAVING count(DISTINCT Student)>=5;
```

597. Friend Requests I: Overall Acceptance Rate | Easy | 🔒 LeetCode

In social network like Facebook or Twitter, people send friend requests and accept others' requests as well. Now given two tables as below: Table: `friend_request`

```
TEXT
| sender_id | send_to_id |request_date|
|-----------|------------|------------|
| 1         | 2          | 2016_06-01 |
| 1         | 3          | 2016_06-01 |
| 1         | 4          | 2016_06-01 |
| 2         | 3          | 2016_06-02 |
| 3         | 4          | 2016-06-09 |
```

Table: `request_accepted`

```
TEXT
| 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 |
| 3            | 4           | 2016-06-10 |
```

Write a query to find the overall acceptance rate of requests rounded to 2 decimals, which is the number of acceptance divide the number of requests. For the sample data above, your query should return the following result.

```
TEXT
|accept_rate|
|-----------|
|       0.80|
```

Note:

The accepted requests are not necessarily from the table friend*request. In this case, you just need to simply count the total accepted requests (no matter whether they are in the original requests), and divide it by the number of requests to get the acceptance rate. It is possible that a sender sends multiple requests to the same receiver, and a request could be accepted more than once. In this case, the 'duplicated' requests or acceptances are only counted once. If there is no requests at all, you should return 0.00 as the accept*rate. Explanation: There are 4 unique accepted requests, and there are 5 requests in total. So the rate is 0.80.

Follow-up:

Can you write a query to return the accept rate but for every month? How about the cumulative accept rate for every day?

Solution

```SQL
SELECT IFNULL((round(accepts/requests, 2)), 0.0) AS accept_rate
FROM
    (SELECT count(DISTINCT sender_id, send_to_id) AS requests FROM friend_request
    (SELECT count(DISTINCT requester_id, accepter_id) AS accepts FROM request_acc
```

601. Human Traffic of Stadium | Hard | LeetCode

Table: `Stadium`

```TEXT
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| id            | int     |
| visit_date    | date    |
| people        | int     |
+---------------+---------+
```

visit*date is the primary key for this table. Each row of this table contains the visit date and visit id to the stadium with the number of people during the visit. No two rows will have the same visit*date, and as the id increases, the dates increase as well.

Write an SQL query to display the records with three or more rows with **consecutive** `id`'s, and the number of people is greater than or equal to 100 for each.

Return the result table ordered by `visit_date` in **ascending order**.

The query result format is in the following example.

TEXT

```
Stadium table:
+------+------------+-----------+
| id   | visit_date | people    |
+------+------------+-----------+
| 1    | 2017-01-01 | 10        |
| 2    | 2017-01-02 | 109       |
| 3    | 2017-01-03 | 150       |
| 4    | 2017-01-04 | 99        |
| 5    | 2017-01-05 | 145       |
| 6    | 2017-01-06 | 1455      |
| 7    | 2017-01-07 | 199       |
| 8    | 2017-01-09 | 188       |
+------+------------+-----------+

Result table:
+------+------------+-----------+
| id   | visit_date | people    |
+------+------------+-----------+
| 5    | 2017-01-05 | 145       |
| 6    | 2017-01-06 | 1455      |
| 7    | 2017-01-07 | 199       |
| 8    | 2017-01-09 | 188       |
+------+------------+-----------+
The four rows with ids 5, 6, 7, and 8 have consecutive ids and each of them has >
The rows with ids 2 and 3 are not included because we need at least three consecu
```

Solution

```sql
SELECT DISTINCT s1.*
FROM Stadium s1 JOIN Stadium s2 JOIN Stadium s3
ON (s1.id = s2.id-1 AND s1.id = s3.id-2) OR
 (s1.id = s2.id+1 AND s1.id = s3.id-1) OR
 (s1.id = s2.id+1 AND s1.id = s3.id+2)
    WHERE s1.people >= 100 AND s2.people >= 100 AND s3.people>=100
  ORDER BY visit_date
```

602. Friend Requests II: Who Has the Most Friends | Medium | 🔒 LeetCode

In social network like Facebook or Twitter, people send friend requests and accept others' requests as well. Table `request_accepted` holds the data of friend acceptance, while requester*id and accepter*id both are the id of a person.

```
| 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 |
```

Write a query to find the the people who has most friends and the most friends number. For the sample data above, the result is:

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

Note:

It is guaranteed there is only 1 people having the most friends. The friend request could only been accepted once, which mean there is no multiple records with the same requester*id and accepter*id value. Explanation: The person with id '3' is a friend of people '1', '2' and '4', so he has 3 friends in total, which is the most number than any others.

Follow-up: In the real world, multiple people could have the same most number of friends, can you find all these people in this case?

SQL

```sql
SELECT t.id, sum(t.num) AS num
FROM (
      (SELECT requester_id AS id, COUNT(1) AS num
       FROM request_accepted
       GROUP BY requester_id)
      union all
       (SELECT accepter_id AS id, COUNT(1) AS num
        FROM request_accepted
        GROUP BY accepter_id)) AS t
GROUP BY t.id
ORDER BY num DESC
LIMIT 1;
```

603. Consecutive Available Seats | Easy | 🔒 LeetCode

Several friends at a cinema ticket office would like to reserve consecutive available seats. Can you help to query all the consecutive available seats order by the seat_id using the following cinema table?

TEXT

```
| seat_id | free |
|---------|------|
| 1       | 1    |
| 2       | 0    |
| 3       | 1    |
```

```
| 4       | 1    |
| 5       | 1    |
```

Your query should return the following result for the sample case above.

```TEXT
| seat_id |
|---------|
| 3       |
| 4       |
| 5       |
```

Note:

The seat_id is an auto increment int, and free is bool ('1' means free, and '0' means occupied.). Consecutive available seats are more than 2(inclusive) seats consecutively available.

Solution

```SQL
SELECT DISTINCT t1.seat_id
FROM cinema AS t1 JOIN cinema AS t2
ON abs(t1.seat_id-t2.seat_id)=1
WHERE t1.free='1' AND t2.free='1'
ORDER BY t1.seat_id
```

607.Sales Person | Easy | 🔒 LeetCode

Description

Given three tables: `salesperson`, `company`, `orders`. Output all the names in the table salesperson, who didn't have sales to company 'RED'.

Example Input

Table: `salesperson`

TEXT

```
+----------+------+--------+-----------------+-----------+
| sales_id | name | salary | commission_rate | hire_date |
+----------+------+--------+-----------------+-----------+
|   1      | John | 100000 |     6           | 4/1/2006  |
|   2      | Amy  | 120000 |     5           | 5/1/2010  |
|   3      | Mark | 65000  |     12          | 12/25/2008|
|   4      | Pam  | 25000  |     25          | 1/1/2005  |
|   5      | Alex | 50000  |     10          | 2/3/2007  |
+----------+------+--------+-----------------+-----------+
```

The table salesperson holds the salesperson information. Every salesperson has a sales_id and a name. Table: `company`

TEXT

```
+---------+--------+------------+
| com_id  |  name  |    city    |
+---------+--------+------------+
|   1     |  RED   |   Boston   |
|   2     | ORANGE |   New York |
|   3     | YELLOW |   Boston   |
|   4     | GREEN  |   Austin   |
+---------+--------+------------+
```

The table company holds the company information. Every company has a com_id and a name. Table: `orders`

TEXT

```
+----------+----------+---------+----------+--------+
| order_id |  date    | com_id  | sales_id | amount |
+----------+----------+---------+----------+--------+
| 1        | 1/1/2014 |    3    |    4     | 100000 |
| 2        | 2/1/2014 |    4    |    5     | 5000   |
| 3        | 3/1/2014 |    1    |    1     | 50000  |
| 4        | 4/1/2014 |    1    |    4     | 25000  |
+----------+----------+---------+----------+--------+
```

The table orders holds the sales record information, salesperson and customer company are represented by sales*id and com*id. output

```
TEXT
+------+
| name |
+------+
| Amy  |
| Mark |
| Alex |
+------+
```

Explanation

According to order '3' and '4' in table orders, it is easy to tell only salesperson 'John' and 'Alex' have sales to company 'RED', so we need to output all the other names in table salesperson.

Solution

```sql
SELECT name
FROM salesperson
WHERE name NOT IN
    (SELECT DISTINCT salesperson.name
    FROM salesperson, orders, company
    WHERE company.name = 'RED'
    AND salesperson.sales_id = orders.sales_id
    AND orders.com_id = company.com_id)
```

608. Tree Node | Medium | 🔒 LeetCode

Given a table tree, id is identifier of the tree node and p_id is its parent node's id.

```
+----+------+
| id | p_id |
+----+------+
| 1  | null |
| 2  | 1    |
| 3  | 1    |
| 4  | 2    |
| 5  | 2    |
+----+------+
```

Each node in the tree can be one of three types:

Leaf: if the node is a leaf node. Root: if the node is the root of the tree. Inner: If the node is neither a leaf node nor a root node. Write a query to print the node id and the type of the node. Sort your output by the node id. The result for the above sample is:

```
+----+------+
| id | Type |
+----+------+
| 1  | Root |
| 2  | Inner|
| 3  | Leaf |
| 4  | Leaf |
| 5  | Leaf |
+----+------+
```

Explanation

Node ‘1’ is root node, because its parent node is NULL and it has child node ‘2’ and ‘3’. Node ‘2’ is inner node, because it has parent node ‘1’ and child node ‘4’ and ‘5’. Node ‘3’, ‘4’ and ‘5’ is Leaf node, because they have parent node and they don’t have child node. And here is the image of the sample tree as below:

TEXT

Note

If there is only one node on the tree, you only need to output its root attributes.

Solution

SQL

```
## Basic Ideas: LEFT JOIN
# In tree, each node can only one parent or no parent
## | id | p_id | id (child) |
## |----+------+------------|
## | 1 | null | 1 |
## | 1 | null | 2 |
## | 2 | 1 | 4 |
## | 2 | 1 | 5 |
## | 3 | 1 | null |
## | 4 | 2 | null |
## | 5 | 2 | null |

SELECT t1.id,
    CASE
        WHEN ISNULL(t1.p_id) THEN 'Root'
        WHEN ISNULL(MAX(t2.id)) THEN 'Leaf'
        ELSE 'Inner'
    END AS Type
FROM tree AS t1 LEFT JOIN tree AS t2
ON t1.id = t2.p_id
GROUP BY t1.id, t1.p_id
```

610. Triangle Judgement | Easy | 🔒 LeetCode

A pupil Tim gets homework to identify whether three line segments could possibly form a triangle. However, this assignment is very heavy because there are hundreds of records to calculate. Could you help Tim by writing a query to judge whether these three sides can form a triangle, assuming table triangle holds the length of the three sides x, y and z.

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

For the sample data above, your query should return the follow result:

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

Solution

```sql
SQL
SELECT x, y, z,
       CASE
           WHEN x+y>z AND y+z>x AND x+z>y THEN 'Yes'
           ELSE 'No'
       END AS triangle
FROM triangle
```

612. Shortest Distance in a Plane | Medium | 🔒 LeetCode

Table point_2d holds the coordinates (x,y) of some unique points (more than two) in a plane. Write a query to find the shortest distance between these points rounded to 2 decimals.

```
TEXT
| x  | y  |
|----|----|
| -1 | -1 |
| 0  | 0  |
| -1 | -2 |
```

The shortest distance is 1.00 from point (-1,-1) to (-1,2). So the output should be:

```
TEXT
| shortest |
|----------|
| 1.00     |
```

Note: The longest distance among all the points are less than 10000.

Solution

```sql
SQL
SELECT ROUND(MIN(SQRT((t1.x-t2.x)*(t1.x-t2.x) + (t1.y-t2.y)*(t1.y-t2.y))), 2) as
FROM point_2d AS t1, point_2d AS t2
WHERE t1.x!=t2.x OR t1.y!=t2.y

# SELECT ROUND(SQRT((t1.x-t2.x)*(t1.x-t2.x) + (t1.y-t2.y)*(t1.y-t2.y)), 2) AS sho
# FROM point_2d AS t1, point_2d AS t2
# WHERE t1.x!=t2.x OR t1.y!=t2.y
# ORDER BY shortest ASC
# LIMIT 1
```

613. Shortest Distance in a Line | Easy | 🔒 LeetCode

Table point holds the x coordinate of some points on x-axis in a plane, which are all integers. Write a query to find the shortest distance between two points in these points.

```
TEXT
| x   |
|-----|
| -1  |
| 0   |
| 2   |
```

The shortest distance is '1' obviously, which is from point '-1' to '0'. So the output is as below:

```
TEXT
| shortest|
|---------|
| 1       |
```

Note: Every point is unique, which means there is no duplicates in table point.

Follow-up: What if all these points have an id and are arranged from the left most to the right most of x axis?

Solution

```sql
SQL
SELECT t1.x-t2.x AS shortest
FROM point AS t1 JOIN point AS t2
WHERE t1.x>t2.x
ORDER BY (t1.x-t2.x) ASC
LIMIT 1
```

614. Second Degree Follower | Medium | 🔒 LeetCode

In facebook, there is a follow table with two columns: followee, follower.

Please write a sql query to get the amount of each follower's follower if he/she has one.

For example:

```
TEXT
+-------------+------------+
| followee    | follower   |
+-------------+------------+
|     A       |     B      |
|     B       |     C      |
|     B       |     D      |
|     D       |     E      |
+-------------+------------+
```

should output:

```
TEXT
+-------------+------------+
| follower    | num        |
+-------------+------------+
|     B       |  2         |
|     D       |  1         |
+-------------+------------+
```

Explanation: Both B and D exist in the follower list, when as a followee, B's follower is C and D, and D's follower is E. A does not exist in follower list.

Note: Followee would not follow himself/herself in all cases. Please display the result in follower's alphabet order.

Solution

```sql
## Explain the business logic
##   A follows B. Then A is follwer, B is followee
## What are second degree followers?
##   A follows B, and B follows C.
##   Then A is the second degree followers of C

SELECT f1.follower, COUNT(DISTINCT f2.follower) AS num
FROM follow AS f1 JOIN follow AS f2
ON f1.follower = f2.followee
GROUP BY f1.follower;
```

615. Average Salary: Departments VS Company | Hard | 🔒 LeetCode

Given two tables as below, write a query to display the comparison result (higher/lower/same) of the average salary of employees in a department to the company's average salary. Table: salary

```text
| id | employee_id | amount | pay_date   |
|----|-------------|--------|------------|
| 1  | 1           | 9000   | 2017-03-31 |
| 2  | 2           | 6000   | 2017-03-31 |
| 3  | 3           | 10000  | 2017-03-31 |
| 4  | 1           | 7000   | 2017-02-28 |
| 5  | 2           | 6000   | 2017-02-28 |
| 6  | 3           | 8000   | 2017-02-28 |
```

The employee*id column refers to the employee*id in the following table employee.

```text
| employee_id | department_id |
|-------------|---------------|
| 1           | 1             |
| 2           | 2             |
| 3           | 2             |
```

So for the sample data above, the result is:

```
TEXT
| pay_month | department_id | comparison  |
|-----------|---------------|-------------|
| 2017-03   | 1             | higher      |
| 2017-03   | 2             | lower       |
| 2017-02   | 1             | same        |
| 2017-02   | 2             | same        |
```

Explanation In March, the company's average salary is (9000+6000+10000)/3 = 8333.33... The average salary for department '1' is 9000, which is the salary of employee*id '1' since there is only one employee in this department. So the comparison result is 'higher' since 9000 > 8333.33 obviously. The average salary of department '2' is (6000 + 10000)/2 = 8000, which is the average of employee*id '2' and '3'. So the comparison result is 'lower' since 8000 < 8333.33. With he same formula for the average salary comparison in February, the result is 'same' since both the department '1' and '2' have the same average salary with the company, which is 7000.

Solution

```sql
SELECT t1.pay_month, t1.department_id,
    (CASE WHEN t1.amount = t2.amount THEN 'same'
          WHEN t1.amount > t2.amount THEN 'higher'
          WHEN t1.amount < t2.amount THEN 'lower' END) AS comparison
FROM
    (SELECT left(pay_date, 7) AS pay_month, department_id, avg(amount) AS amount
    FROM salary JOIN employee
    ON salary.employee_id = employee.employee_id
    GROUP BY pay_month, department_id
    ORDER BY pay_month DESC, department_id) AS t1
    JOIN
    (SELECT left(pay_date, 7) AS pay_month, avg(amount) AS amount
    FROM salary JOIN employee
```

```
    ON salary.employee_id = employee.employee_id
    GROUP BY pay_month) AS t2
    ON t1.pay_month = t2.pay_month
```

618. Students Report By Geography | Hard | 🔒 LeetCode

A U.S graduate school has students from Asia, Europe and America. The students' location information are stored in table student as below.

```
TEXT
 | name   | continent |
 |--------|-----------|
 | Jack   | America   |
 | Pascal | Europe    |
 | Xi     | Asia      |
 | Jane   | America   |
```

Pivot the continent column in this table so that each name is sorted alphabetically and displayed underneath its corresponding continent. The output headers should be America, Asia and Europe respectively. It is guaranteed that the student number from America is no less than either Asia or Europe. For the sample input, the output is:

```
TEXT
 | America | Asia | Europe |
 |---------|------|--------|
 | Jack    | Xi   | Pascal |
 | Jane    |      |        |
```

Follow-up: If it is unknown which continent has the most students, can you write a query to generate the student report?

Solution

```SQL
SELECT t1.name AS America, t2.name AS Asia, t3.name AS Europe
FROM
    (SELECT (@cnt1 := @cnt1 + 1) AS id, name
    FROM student
    CROSS JOIN (SELECT @cnt1 := 0) AS dummy
    WHERE continent='America'
    ORDER BY name) AS t1
    LEFT JOIN
    (SELECT (@cnt2 := @cnt2 + 1) AS id, name
    FROM student
    CROSS JOIN (SELECT @cnt2 := 0) AS dummy
    WHERE continent='Asia'
    ORDER BY name) AS t2
    ON t1.id = t2.id
    LEFT JOIN
    (SELECT (@cnt3 := @cnt3 + 1) AS id, name
    FROM student
    CROSS JOIN (SELECT @cnt3 := 0) AS dummy
    WHERE continent='Europe'
    ORDER BY name) AS t3
    ON t1.id = t3.id
```

619. Biggest Single Number | Easy | 🔒 LeetCode

Table number contains many numbers in column num including duplicated ones. Can you write a SQL query to find the biggest number, which only appears once.

```TEXT
+---+
|num|
+---+
| 8 |
| 8 |
| 3 |
| 3 |
| 1 |
| 4 |
```

```
| 5 |
| 6 |
```

For the sample data above, your query should return the following result:

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

Note: If there is no such number, just output null.

Solution

```SQL
SELECT IFNULL((
    SELECT num
    FROM number
    GROUP BY num
    HAVING count(1) = 1
    ORDER BY num DESC
    LIMIT 0, 1), NULL) AS num
```

620. Not Boring Movies | Easy | LeetCode

X city opened a new cinema, many people would like to go to this cinema. The cinema also gives out a poster indicating the movies' ratings and descriptions. Please write a SQL query to output movies with an odd numbered ID and a description that is not 'boring'. Order the result by rating.

For example, table `cinema`:

TEXT

```
+---------+-----------+--------------+-----------+
|   id    | movie     |  description |  rating   |
+---------+-----------+--------------+-----------+
|   1     | War       |   great 3D   |   8.9     |
|   2     | Science   |   fiction    |   8.5     |
|   3     | irish     |   boring     |   6.2     |
|   4     | Ice song  |   Fantacy    |   8.6     |
|   5     | House card|   Interesting|   9.1     |
+---------+-----------+--------------+-----------+
```

For the example above, the output should be:

TEXT

```
+---------+-----------+--------------+-----------+
|   id    | movie     |  description |  rating   |
+---------+-----------+--------------+-----------+
|   5     | House card|   Interesting|   9.1     |
|   1     | War       |   great 3D   |   8.9     |
+---------+-----------+--------------+-----------+
```

Solution

SQL

```sql
SELECT *
FROM Cinema
WHERE description <> 'boring' AND ID % 2 = 1
ORDER BY rating DESC;
```

626. Exchange Seats | Medium | LeetCode

Mary is a teacher in a middle school and she has a table `seat` storing students' names and their corresponding seat ids.

The column id is continuous increment.

Mary wants to change seats for the adjacent students.

Can you write a SQL query to output the result for Mary?

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

For the sample input, the output is:

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

Note:
If the number of students is odd, there is no need to change the last one's seat.

Solution

```
SQL
SELECT
IF(id<(SELECT MAX(id) FROM seat),IF(id%2=0,id-1, id+1),IF(id%2=0, id-1, id)) AS i
```

```
FROM seat
ORDER BY id;
```

627. Swap Salary | LeetCode

Table: `Salary`

```
TEXT
+-------------+----------+
| Column Name | Type     |
+-------------+----------+
| id          | int      |
| name        | varchar  |
| sex         | ENUM     |
| salary      | int      |
+-------------+----------+

id is the primary key for this table.
The sex column is ENUM value of type ('m', 'f').
The table contains information about an employee.
```

Write an SQL query to swap all `'f'` and `'m'` values (i.e., change all `'f'` values to `'m'` and vice versa) with a single update statement and no intermediate temp table(s).

Note that you must write a single update statement, DO NOT write any select statement for this problem.

The query result format is in the following example:

```
TEXT
Salary table:
+----+------+-----+--------+
| id | name | sex | salary |
+----+------+-----+--------+
| 1  | A    | m   | 2500   |
```

```
| 2  | B    | f   | 1500   |
| 3  | C    | m   | 5500   |
| 4  | D    | f   | 500    |
+----+------+-----+--------+

Result table:
+----+------+-----+--------+
| id | name | sex | salary |
+----+------+-----+--------+
| 1  | A    | f   | 2500   |
| 2  | B    | m   | 1500   |
| 3  | C    | f   | 5500   |
| 4  | D    | m   | 500    |
+----+------+-----+--------+
(1, A) and (2, C) were changed from 'm' to 'f'.
(2, B) and (4, D) were changed from 'f' to 'm'.
```

Solution

SQL

```sql
# With IF
UPDATE Salary SET sex = IF(sex='m', 'f', 'm')

# With CASE
UPDATE Salary SET sex = CASE WHEN sex='m' THEN 'f' ELSE 'm' END
```

1045. Customers Who Bought All Products | Medium | 🔒 LeetCode

Table: `Customer`

TEXT

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| customer_id | int     |
```

```
| product_key | int     |
+-------------+---------+
```

product_key is a foreign key to Product table. Table: `Product`

```
TEXT
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| product_key | int     |
+-------------+---------+
product_key is the primary key column for this table.
```

Write an SQL query for a report that provides the customer ids from the Customer table that bought all the products in the Product table.

For example:

```
TEXT
Customer table:
+-------------+-------------+
| customer_id | product_key |
+-------------+-------------+
| 1           | 5           |
| 2           | 6           |
| 3           | 5           |
| 3           | 6           |
| 1           | 6           |
+-------------+-------------+

Product table:
+-------------+
| product_key |
+-------------+
| 5           |
| 6           |
+-------------+
```

```
Result table:
+-------------+
| customer_id |
+-------------+
| 1           |
| 3           |
+-------------+
The customers who bought all the products (5 and 6) are customers with id 1 and 3
```

Solution

```sql
SELECT customer_id
FROM Customer
GROUP NY customer_id
HAVING count(DISTINCT product_key) = (
    SELECT count(1)
    FROM Product)
```

1050. Actors and Directors Who Cooperated At Least Three Times | Easy | 🔒 LeetCode

Table: `ActorDirector`

```text
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| actor_id    | int     |
| director_id | int     |
| timestamp   | int     |
+-------------+---------+
timestamp is the primary key column for this table.
```

Write a SQL query for a report that provides the pairs (actor*id*, *director*id) where the actor have cooperated with the director at least 3 times.

Example:

```
TEXT
ActorDirector table:
+-------------+-------------+-------------+
| actor_id    | director_id | timestamp   |
+-------------+-------------+-------------+
| 1           | 1           | 0           |
| 1           | 1           | 1           |
| 1           | 1           | 2           |
| 1           | 2           | 3           |
| 1           | 2           | 4           |
| 2           | 1           | 5           |
| 2           | 1           | 6           |
+-------------+-------------+-------------+

Result table:
+-------------+-------------+
| actor_id    | director_id |
+-------------+-------------+
| 1           | 1           |
+-------------+-------------+
The only pair is (1, 1) where they cooperated exactly 3 times.
```

Solution

```
SQL
SELECT actor_id, director_id
FROM ActorDirector
GROUP BY actor_id, director_id
HAVING COUNT(1)>=3
```

1068. Product Sales Analysis I | Easy | 🔒 LeetCode

Table: `Sales`

```
TEXT
+-------------+-------+
| Column Name | Type  |
+-------------+-------+
| sale_id     | int   |
| product_id  | int   |
| year        | int   |
| quantity    | int   |
| price       | int   |
+-------------+-------+
(sale_id, year) is the primary key of this table.
product_id is a foreign key to Product table.
Note that the price is per unit.
```

Table: `Product`

```
TEXT
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| product_id   | int     |
| product_name | varchar |
+--------------+---------+
product_id is the primary key of this table.
```

Write an SQL query that reports all product names of the products in the Sales table along with their selling year and price.

For example:

```
TEXT
Sales table:
+---------+------------+------+----------+-------+
| sale_id | product_id | year | quantity | price |
+---------+------------+------+----------+-------+
| 1       | 100        | 2008 | 10       | 5000  |
```

```
| 2       | 100        | 2009 | 12       | 5000  |
| 7       | 200        | 2011 | 15       | 9000  |
+---------+------------+------+----------+-------+

Product table:
+------------+--------------+
| product_id | product_name |
+------------+--------------+
| 100        | Nokia        |
| 200        | Apple        |
| 300        | Samsung      |
+------------+--------------+

Result table:
+--------------+-------+-------+
| product_name | year  | price |
+--------------+-------+-------+
| Nokia        | 2008  | 5000  |
| Nokia        | 2009  | 5000  |
| Apple        | 2011  | 9000  |
+--------------+-------+-------+
```

Solution

SQL

```sql
SELECT product_name, year, price
FROM Sales JOIN Product
ON Product.product_id = Sales.product_id
```

1069. Product Sales Analysis II | Easy | 🔒 LeetCode

Table: `Sales`

TEXT

```
+-------------+-------+
| Column Name | Type  |
```

```
+-------------+-------+
| sale_id     | int   |
| product_id  | int   |
| year        | int   |
| quantity    | int   |
| price       | int   |
+-------------+-------+
sale_id is the primary key of this table.
product_id is a foreign key to Product table.
Note that the price is per unit.
```

Table: `Product`

```
TEXT
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| product_id   | int     |
| product_name | varchar |
+--------------+---------+
product_id is the primary key of this table.
```

Write an SQL query that reports the total quantity sold for every product id.

The query result format is in the following example:

```
TEXT
Sales table:
+---------+------------+------+----------+-------+
| sale_id | product_id | year | quantity | price |
+---------+------------+------+----------+-------+
| 1       | 100        | 2008 | 10       | 5000  |
| 2       | 100        | 2009 | 12       | 5000  |
| 7       | 200        | 2011 | 15       | 9000  |
+---------+------------+------+----------+-------+

Product table:
+------------+--------------+
```

```
| product_id | product_name |
+------------+--------------+
| 100        | Nokia        |
| 200        | Apple        |
| 300        | Samsung      |
+------------+--------------+

Result table:
+--------------+----------------+
| product_id   | total_quantity |
+--------------+----------------+
| 100          | 22             |
| 200          | 15             |
+--------------+----------------+
```

Solution

SQL

```sql
SELECT product_id, sum(quantity) AS total_quantity
FROM Sales
GROUP BY product_id;
```

1070. Product Sales Analysis III | Medium | 🔒 LeetCode

Table: `Sales`

TEXT

```
+-------------+-------+
| Column Name | Type  |
+-------------+-------+
| sale_id     | int   |
| product_id  | int   |
| year        | int   |
| quantity    | int   |
| price       | int   |
+-------------+-------+
```

```
sale_id is the primary key of this table.
product_id is a foreign key to Product table.
Note that the price is per unit.
```

Table: `Product`

TEXT

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| product_id   | int     |
| product_name | varchar |
+--------------+---------+
product_id is the primary key of this table.
```

Write an SQL query that selects the product id, year, quantity, and price for the first year of every product sold.

The query result format is in the following example:

TEXT

```
Sales table:
+---------+------------+------+----------+-------+
| sale_id | product_id | year | quantity | price |
+---------+------------+------+----------+-------+
| 1       | 100        | 2008 | 10       | 5000  |
| 2       | 100        | 2009 | 12       | 5000  |
| 7       | 200        | 2011 | 15       | 9000  |
+---------+------------+------+----------+-------+

Product table:
+------------+--------------+
| product_id | product_name |
+------------+--------------+
| 100        | Nokia        |
| 200        | Apple        |
| 300        | Samsung      |
+------------+--------------+
```

```
Result table:
+------------+------------+----------+-------+
| product_id | first_year | quantity | price |
+------------+------------+----------+-------+
| 100        | 2008       | 10       | 5000  |
| 200        | 2011       | 15       | 9000  |
+------------+------------+----------+-------+
```

Solution

SQL

```sql
SELECT
    product_id,
    year first_year,
    quantity,
    price
FROM Sales
WHERE (product_id, year) IN (SELECT product_id, MIN(year)
                             FROM Sales
                             GROUP BY product_id)
```

1075. Project Employees I | Easy | 🔒 LeetCode

Table: `Project`

TEXT

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| project_id  | int     |
| employee_id | int     |
+-------------+---------+
(project_id, employee_id) is the primary key of this table.
employee_id is a foreign key to Employee table.
```

Table: `Employee`

```
+------------------+---------+
| Column Name      | Type    |
+------------------+---------+
| employee_id      | int     |
| name             | varchar |
| experience_years | int     |
+------------------+---------+
employee_id is the primary key of this table.
```

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

The query result format is in the following example:

```
Project table:
+-------------+-------------+
| project_id  | employee_id |
+-------------+-------------+
| 1           | 1           |
| 1           | 2           |
| 1           | 3           |
| 2           | 1           |
| 2           | 4           |
+-------------+-------------+

Employee table:
+-------------+--------+------------------+
| employee_id | name   | experience_years |
+-------------+--------+------------------+
| 1           | Khaled | 3                |
| 2           | Ali    | 2                |
| 3           | John   | 1                |
| 4           | Doe    | 2                |
+-------------+--------+------------------+
```

```
Result table:
+-------------+---------------+
| project_id  | average_years |
+-------------+---------------+
| 1           | 2.00          |
| 2           | 2.50          |
+-------------+---------------+
The average experience years for the first project is (3 + 2 + 1) / 3 = 2.00 and
```

Solution

SQL

```sql
SELECT
    p.project_id,
    ROUND(AVG(e.experience_years),2) average_years
FROM
    Project p JOIN Employee e ON
    p.employee_id = e.employee_id
GROUP BY
    p.project_id
```

1076. Project Employees II | Easy | 🔒 LeetCode

Table: `Project`

TEXT

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| project_id  | int     |
| employee_id | int     |
+-------------+---------+
(project_id, employee_id) is the primary key of this table.
employee_id is a foreign key to Employee table.
```

Table: `Employee`

```
TEXT
+------------------+---------+
| Column Name      | Type    |
+------------------+---------+
| employee_id      | int     |
| name             | varchar |
| experience_years | int     |
+------------------+---------+
employee_id is the primary key of this table.
```

Write an SQL query that reports all the projects that have the most employees.

The query result format is in the following example:

```
TEXT
Project table:
+-------------+-------------+
| project_id  | employee_id |
+-------------+-------------+
| 1           | 1           |
| 1           | 2           |
| 1           | 3           |
| 2           | 1           |
| 2           | 4           |
+-------------+-------------+

Employee table:
+-------------+--------+------------------+
| employee_id | name   | experience_years |
+-------------+--------+------------------+
| 1           | Khaled | 3                |
| 2           | Ali    | 2                |
| 3           | John   | 1                |
| 4           | Doe    | 2                |
+-------------+--------+------------------+

Result table:
```

```
+-------------+
| project_id  |
+-------------+
| 1           |
+-------------+
```

The first project has 3 employees while the second one has 2.

SQL

```sql
SELECT project_id
FROM Project
GROUP BY project_id
HAVING COUNT(employee_id) = (SELECT COUNT(employee_id)
                             FROM Project
                             GROUP BY project_id
                             ORDER BY COUNT(employee_id) DESC
                             LIMIT 1)
```

1077. Project Employees III | Medium | 🔒 LeetCode

Table: `Project`

TEXT

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| project_id  | int     |
| employee_id | int     |
+-------------+---------+
```

(project_id, employee_id) is the primary key of this table.
employee_id is a foreign key to Employee table.

Table: `Employee`

TEXT

```
+------------------+---------+
| Column Name      | Type    |
+------------------+---------+
| employee_id      | int     |
| name             | varchar |
| experience_years | int     |
+------------------+---------+
employee_id is the primary key of this table.
```

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

The query result format is in the following example:

TEXT

```
Project table:
+-------------+-------------+
| project_id  | employee_id |
+-------------+-------------+
| 1           | 1           |
| 1           | 2           |
| 1           | 3           |
| 2           | 1           |
| 2           | 4           |
+-------------+-------------+

Employee table:
+-------------+--------+------------------+
| employee_id | name   | experience_years |
+-------------+--------+------------------+
| 1           | Khaled | 3                |
| 2           | Ali    | 2                |
| 3           | John   | 3                |
| 4           | Doe    | 2                |
+-------------+--------+------------------+

Result table:
+-------------+---------------+
```

```
| project_id  | employee_id   |
+-------------+---------------+
| 1           | 1             |
| 1           | 3             |
| 2           | 1             |
+-------------+---------------+
Both employees with id 1 and 3 have the most experience among the employees of th
```

Solution

```sql
SELECT
    p.project_id,
    e.employee_id
FROM
    Project p LEFT JOIN Employee e ON
    p.employee_id = e.employee_id
WHERE (p.project_id,
       e.experience_years) IN (SELECT
                                   p.project_id,
                                   MAX(e.experience_years)
                               FROM
                                   Project p JOIN Employee e ON
                                   p.employee_id = e.employee_id
                               GROUP BY
                                   p.project_id)
```

1082. Sales Analysis I | Easy | 🔒 LeetCode

Table: `Product`

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
```

```
| product_id   | int     |
| product_name | varchar |
| unit_price   | int     |
+--------------+---------+
product_id is the primary key of this table.
```

Table: Sales

TEXT

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| seller_id   | int     |
| product_id  | int     |
| buyer_id    | int     |
| sale_date   | date    |
| quantity    | int     |
| price       | int     |
+------ ------+---------+
This table has no primary key, it can have repeated rows.
product_id is a foreign key to Product table.
```

Write an SQL query that reports the best seller by total sales price, If there is a tie, report them all.

The query result format is in the following example:

TEXT

```
Product table:
+------------+--------------+------------+
| product_id | product_name | unit_price |
+------------+--------------+------------+
| 1          | S8           | 1000       |
| 2          | G4           | 800        |
| 3          | iPhone       | 1400       |
+------------+--------------+------------+

Sales table:
```

```
+-----------+------------+----------+------------+----------+-------+
| seller_id | product_id | buyer_id | sale_date  | quantity | price |
+-----------+------------+----------+------------+----------+-------+
| 1         | 1          | 1        | 2019-01-21 | 2        | 2000  |
| 1         | 2          | 2        | 2019-02-17 | 1        | 800   |
| 2         | 2          | 3        | 2019-06-02 | 1        | 800   |
| 3         | 3          | 4        | 2019-05-13 | 2        | 2800  |
+-----------+------------+----------+------------+----------+-------+

Result table:
+-------------+
| seller_id   |
+-------------+
| 1           |
| 3           |
+-------------+
Both sellers with id 1 and 3 sold products with the most total price of 2800.
```

Solution

SQL

```sql
SELECT seller_id
FROM Sales
GROUP BY seller_id
HAVING SUM(price) = (SELECT SUM(price)
                     FROM Sales
                     GROUP BY seller_id
                     ORDER BY SUM(price) DESC
                     LIMIT 1)
```

1083. Sales Analysis II | Easy | LeetCode

Table: `Product`

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| product_id   | int     |
| product_name | varchar |
| unit_price   | int     |
+--------------+---------+
product_id is the primary key of this table.
```

Table: `Sales`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| seller_id   | int     |
| product_id  | int     |
| buyer_id    | int     |
| sale_date   | date    |
| quantity    | int     |
| price       | int     |
+------ ------+---------+
This table has no primary key, it can have repeated rows.
product_id is a foreign key to Product table.
```

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.

The query result format is in the following example:

```
Product table:
+------------+--------------+------------+
| product_id | product_name | unit_price |
+------------+--------------+------------+
| 1          | S8           | 1000       |
| 2          | G4           | 800        |
```

```
| 3          | iPhone       | 1400       |
+------------+--------------+------------+

Sales table:
+-----------+------------+----------+------------+----------+-------+
| seller_id | product_id | buyer_id | sale_date  | quantity | price |
+-----------+------------+----------+------------+----------+-------+
| 1         | 1          | 1        | 2019-01-21 | 2        | 2000  |
| 1         | 2          | 2        | 2019-02-17 | 1        | 800   |
| 2         | 1          | 3        | 2019-06-02 | 1        | 800   |
| 3         | 3          | 3        | 2019-05-13 | 2        | 2800  |
+-----------+------------+----------+------------+----------+-------+

Result table:
+-------------+
| buyer_id    |
+-------------+
| 1           |
+-------------+
The buyer with id 1 bought an S8 but didn't buy an iPhone. The buyer with id 3 bo
```

Solution

SQL

```sql
SELECT DISTINCT s.buyer_id
FROM Sales s LEFT JOIN Product p ON
    s.product_id = p.product_id
WHERE p.product_name = 'S8' AND
      s.buyer_id NOT IN (SELECT s.buyer_id
                         FROM Sales s LEFT JOIN Product p ON
                             s.product_id = p.product_id
                         WHERE p.product_name = 'iPhone')
```

1084. Sales Analysis III | Easy | 🔒 LeetCode

Reports the products that were only sold in spring 2019. That is, between 2019-01-01 and 2019-03-31 inclusive. Select the product that were only sold in spring 2019.

TEXT

```
Product table:
+------------+--------------+------------+
| product_id | product_name | unit_price |
+------------+--------------+------------+
| 1          | S8           | 1000       |
| 2          | G4           | 800        |
| 3          | iPhone       | 1400       |
+------------+--------------+------------+

Sales table:
+-----------+------------+----------+------------+----------+-------+
| seller_id | product_id | buyer_id | sale_date  | quantity | price |
+-----------+------------+----------+------------+----------+-------+
| 1         | 1          | 1        | 2019-01-21 | 2        | 2000  |
| 1         | 2          | 2        | 2019-02-17 | 1        | 800   |
| 2         | 2          | 3        | 2019-06-02 | 1        | 800   |
| 3         | 3          | 4        | 2019-05-13 | 2        | 2800  |
+-----------+------------+----------+------------+----------+-------+

Result table:
+-------------+--------------+
| product_id  | product_name |
+-------------+--------------+
| 1           | S8           |
+-------------+--------------+
The product with id 1 was only sold in spring 2019 while the other two were sold
```

Solution

SQL

```sql
(SELECT DISTINCT s.product_id, p.product_name
FROM Sales s LEFT JOIN Product p ON
    s.product_id = p.product_id
```

```
WHERE s.sale_date >= '2019-01-01' AND
      s.sale_date <= '2019-03-31')
EXCEPT -- MINUS if Oracle
(SELECT DISTINCT s.product_id, p.product_name
FROM Sales s LEFT JOIN Product p ON
    s.product_id = p.product_id
WHERE s.sale_date < '2019-01-01' OR
      s.sale_date > '2019-03-31')
```

1097. Game Play Analysis V | Hard | 🔒 LeetCode

We define the install date of a player to be the first login day of that player. We also define day 1 retention of some date X to be the number of players whose install date is X and they logged back in on the day right after X , divided by the number of players whose install date is X, rounded to 2 decimal places. Write an SQL query that reports for each install date, the number of players that installed the game on that day and the day 1 retention. The query result format is in the following example:

TEXT

```
Activity table:
+-----------+-----------+------------+--------------+
| player_id | device_id | event_date | games_played |
+-----------+-----------+------------+--------------+
| 1         | 2         | 2016-03-01 | 5            |
| 1         | 2         | 2016-03-02 | 6            |
| 2         | 3         | 2017-06-25 | 1            |
| 3         | 1         | 2016-03-01 | 0            |
| 3         | 4         | 2016-07-03 | 5            |
+-----------+-----------+------------+--------------+

Result table:
+------------+----------+----------------+
| install_dt | installs | Day1_retention |
+------------+----------+----------------+
| 2016-03-01 | 2        | 0.50           |
| 2017-06-25 | 1        | 0.00           |
+------------+----------+----------------+
```

```
Player 1 and 3 installed the game on 2016-03-01 but only player 1 logged back in
Player 2 installed the game on 2017 -06-25 but didn't log back in on 2017-06-26 s
```

Solution

```sql
SELECT
    install_dt,
    COUNT(player_id) installs,
    ROUND(COUNT(retention)/COUNT(player_id),2) Day1_retention  --the number of re
FROM
    (
    SELECT a.player_id, a.install_dt, b.event_date retention -- id, the record of
    FROM
        (SELECT player_id, MIN(event_date) install_dt   --subquery 1 take the fir
        FROM Activity
        GROUP BY player_id) a LEFT JOIN Activity b ON   --sq1 left join the origi
            a.player_id = b.player_id AND
            a.install_dt + 1=b.event_date
    ) AS tmp
GROUP BY
    install_dt
```

1098. Unpopular Books | Medium | 🔒 LeetCode

Table: Books

```
+----------------+---------+
| Column Name    | Type    |
+----------------+---------+
| book_id        | int     |
| name           | varchar |
| available_from | date    |
```

```
+---------------+---------+
book_id is the primary key of this table.
```

Table: `Orders`

```
TEXT
+---------------+---------+
| Column Name    | Type    |
+---------------+---------+
| order_id       | int     |
| book_id        | int     |
| quantity       | int     |
| dispatch_date  | date    |
+---------------+---------+
order_id is the primary key of this table.
book_id is a foreign key to the Books table.
```

Write an SQL query that reports the books that have sold less than 10 copies in the last year, excluding books that have been available for less than 1 month from today. Assume today is 2019-06-23.

The query result format is in the following example:

```
TEXT
Books table:
+---------+--------------------+----------------+
| book_id | name               | available_from |
+---------+--------------------+----------------+
| 1       | "Kalila And Demna" | 2010-01-01     |
| 2       | "28 Letters"       | 2012-05-12     |
| 3       | "The Hobbit"       | 2019-06-10     |
| 4       | "13 Reasons Why"   | 2019-06-01     |
| 5       | "The Hunger Games" | 2008-09-21     |
+---------+--------------------+----------------+

Orders table:
+----------+---------+----------+---------------+
| order_id | book_id | quantity | dispatch_date |
```

```
+----------+---------+----------+---------------+
| 1        | 1       | 2        | 2018-07-26    |
| 2        | 1       | 1        | 2018-11-05    |
| 3        | 3       | 8        | 2019-06-11    |
| 4        | 4       | 6        | 2019-06-05    |
| 5        | 4       | 5        | 2019-06-20    |
| 6        | 5       | 9        | 2009-02-02    |
| 7        | 5       | 8        | 2010-04-13    |
+----------+---------+----------+---------------+
```

Result table:

```
+-----------+--------------------+
| book_id   | name               |
+-----------+--------------------+
| 1         | "Kalila And Demna" |
| 2         | "28 Letters"       |
| 5         | "The Hunger Games" |
+-----------+--------------------+
```

Solution

SQL

```sql
SELECT
    b.book_id, b.name
FROM
    Books b LEFT JOIN (                    -- subquery calculates last year's sal
        SELECT book_id, SUM(quantity) nsold
        FROM Orders
        WHERE dispatch_date BETWEEN '2018-06-23' AND '2019-06-23'
        GROUP BY book_id
    ) o
    ON b.book_id = o.book_id
WHERE
    (o.nsold < 10 OR o.nsold IS NULL) AND           -- Sales less than 10 or no s
    DATEDIFF('2019-06-23', b.available_from) > 30   -- Not a new book within 1 mo
```

1107. New Users Daily Count | Medium | 🔒 LeetCode

Table: `Traffic`

```
TEXT
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| user_id       | int     |
| activity      | enum    |
| activity_date | date    |
+---------------+---------+
There is no primary key for this table, it may have duplicate rows.
The activity column is an ENUM type of ('login', 'logout', 'jobs', 'groups', 'hom
```

Write an SQL query that reports for every date within at most 90 days from today, the number of users that logged in for the first time on that date. Assume today is 2019-06-30.

The query result format is in the following example:

```
TEXT
Traffic table:
+---------+----------+---------------+
| user_id | activity | activity_date |
+---------+----------+---------------+
| 1       | login    | 2019-05-01    |
| 1       | homepage | 2019-05-01    |
| 1       | logout   | 2019-05-01    |
| 2       | login    | 2019-06-21    |
| 2       | logout   | 2019-06-21    |
| 3       | login    | 2019-01-01    |
| 3       | jobs     | 2019-01-01    |
| 3       | logout   | 2019-01-01    |
| 4       | login    | 2019-06-21    |
| 4       | groups   | 2019-06-21    |
| 4       | logout   | 2019-06-21    |
| 5       | login    | 2019-03-01    |
```

```
| 5       | logout   | 2019-03-01    |
| 5       | login    | 2019-06-21    |
| 5       | logout   | 2019-06-21    |
+---------+----------+---------------+

Result table:
+------------+-------------+
| login_date | user_count  |
+------------+-------------+
| 2019-05-01 | 1           |
| 2019-06-21 | 2           |
+------------+-------------+
Note that we only care about dates with non zero user count.
The user with id 5 first logged in on 2019-03-01 so he's not counted on 2019-06-2
```

Solution

SQL

```sql
#Solution- 1:
SELECT login_date, COUNT(user_id) AS user_count
FROM (SELECT user_id, MIN(activity_date) AS login_date
    FROM Traffic
    WHERE activity = 'login'
    GROUP BY user_id) AS t
WHERE login_date >= DATE_ADD('2019-06-30', INTERVAL -90 DAY) AND login_date <= '2
GROUP BY login_date

#Solution- 2:
SELECT login_date, COUNT(user_id) user_count
FROM
    (SELECT user_id, MIN(activity_date) as login_date
    FROM Traffic
    WHERE activity='login'
    GROUP BY user_id) as t
WHERE DATEDIFF('2019-06-30', login_date) <= 90
GROUP BY login_date
```

1112. Highest Grade For Each Student | Medium | 🔒 LeetCode

Table: `Enrollments`

```
TEXT
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| student_id    | int     |
| course_id     | int     |
| grade         | int     |
+---------------+---------+
(student_id, course_id) is the primary key of this table.
```

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. The output must be sorted by increasing student*id.

The query result format is in the following example:

```
TEXT
Enrollments table:
+------------+-------------------+
| student_id | course_id | grade |
+------------+-----------+-------+
| 2          | 2         | 95    |
| 2          | 3         | 95    |
| 1          | 1         | 90    |
| 1          | 2         | 99    |
| 3          | 1         | 80    |
| 3          | 2         | 75    |
| 3          | 3         | 82    |
+------------+-----------+-------+
Result table:
+------------+-------------------+
| student_id | course_id | grade |
+------------+-----------+-------+
| 1          | 2         | 99    |
| 2          | 2         | 95    |
```

```
| 3          | 3          | 82    |
+------------+------------+-------+
```

Solution

SQL

```sql
SELECT student_id, MIN(course_id) course_id, grade
FROM Enrollments
WHERE (student_id, grade) IN
        (SELECT student_id, MAX(grade)
        FROM Enrollments
        GROUP BY student_id)
GROUP BY student_id
ORDER BY student_id;
```

1113.Reported Posts | Easy | 🔒 LeetCode

Table: `Actions`

TEXT

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| user_id       | int     |
| post_id       | int     |
| action_date   | date    |
| action        | enum    |
| extra         | varchar |
+---------------+---------+
There is no primary key for this table, it may have duplicate rows.
The action column is an ENUM type of ('view', 'like', 'reaction', 'comment', 'rep
The extra column has optional information about the action such as a reason for r
```

Write an SQL query that reports the number of posts reported yesterday for each report reason. Assume today is 2019-07-05.

The query result format is in the following example:

```
TEXT
Actions table:
+---------+---------+-------------+--------+--------+
| user_id | post_id | action_date | action | extra  |
+---------+---------+-------------+--------+--------+
| 1       | 1       | 2019-07-01  | view   | null   |
| 1       | 1       | 2019-07-01  | like   | null   |
| 1       | 1       | 2019-07-01  | share  | null   |
| 2       | 4       | 2019-07-04  | view   | null   |
| 2       | 4       | 2019-07-04  | report | spam   |
| 3       | 4       | 2019-07-04  | view   | null   |
| 3       | 4       | 2019-07-04  | report | spam   |
| 4       | 3       | 2019-07-02  | view   | null   |
| 4       | 3       | 2019-07-02  | report | spam   |
| 5       | 2       | 2019-07-04  | view   | null   |
| 5       | 2       | 2019-07-04  | report | racism |
| 5       | 5       | 2019-07-04  | view   | null   |
| 5       | 5       | 2019-07-04  | report | racism |
+---------+---------+-------------+--------+--------+
Result table:
+---------------+--------------+
| report_reason | report_count |
+---------------+--------------+
| spam          | 1            |
| racism        | 2            |
+---------------+--------------+
```

Note that we only care about report reasons with non zero number of reports.

Solution

```SQL
SELECT extra report_reason, COUNT(DISTINCT post_id) report_count
FROM
    (SELECT post_id, extra
    FROM Actions
    WHERE action_date = DATE_SUB('2019-07-05', INTERVAL 1 DAY) AND
          action = 'report') AS tmp
GROUP BY extra
```

1126. Active Businesses | Medium | 🔒 LeetCode

Table: `Events`

```TEXT
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| business_id   | int     |
| event_type    | varchar |
| occurences    | int     |
+---------------+---------+
(business_id, event_type) is the primary key of this table.
Each row in the table logs the info that an event of some type occured at some bu
```

Write an SQL query to find all active businesses.

An active business is a business that has more than one event type with occurences greater than the average occurences of that event type among all businesses.

The query result format is in the following example:

```TEXT
Events table:
+-------------+------------+------------+
| business_id | event_type | occurences |
+-------------+------------+------------+
```

```
| 1           | reviews    | 7           |
| 3           | reviews    | 3           |
| 1           | ads        | 11          |
| 2           | ads        | 7           |
| 3           | ads        | 6           |
| 1           | page views | 3           |
| 2           | page views | 12          |
+-------------+------------+-------------+

Result table:
+-------------+
| business_id |
+-------------+
| 1           |
+-------------+
Average for 'reviews', 'ads' and 'page views' are (7+3)/2=5, (11+7+6)/3=8, (3+12)
Business with id 1 has 7 'reviews' events (more than 5) and 11 'ads' events (more
```

Solution

SQL

```sql
SELECT business_id
FROM (SELECT a.business_id, a.event_type, a.occurences, b.event_avg  -- sub 2
      FROM Events a LEFT JOIN
        (SELECT event_type, AVG(occurences) event_avg   -- sub 1
         FROM Events
         GROUP BY event_type) b ON
      a.event_type = b.event_type) tmp
WHERE occurences > event_avg
GROUP BY business_id
HAVING COUNT(event_type) > 1
```

1127. User Purchase Platform | Hard | 🔒 LeetCode

Table: `Spending`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| user_id     | int     |
| spend_date  | date    |
| platform    | enum    |
| amount      | int     |
+-------------+---------+
The table logs the spendings history of users that make purchases from an online
(user_id, spend_date, platform) is the primary key of this table.
The platform column is an ENUM type of ('desktop', 'mobile').
```

Write an SQL query to find the total number of users and the total amount spent using mobile only, desktop only and both mobile and desktop together for each date.

The query result format is in the following example:

```
Spending table:
+---------+------------+----------+--------+
| user_id | spend_date | platform | amount |
+---------+------------+----------+--------+
| 1       | 2019-07-01 | mobile   | 100    |
| 1       | 2019-07-01 | desktop  | 100    |
| 2       | 2019-07-01 | mobile   | 100    |
| 2       | 2019-07-02 | mobile   | 100    |
| 3       | 2019-07-01 | desktop  | 100    |
| 3       | 2019-07-02 | desktop  | 100    |
+---------+------------+----------+--------+

Result table:
+------------+----------+--------------+-------------+
| spend_date | platform | total_amount | total_users |
+------------+----------+--------------+-------------+
| 2019-07-01 | desktop  | 100          | 1           |
| 2019-07-01 | mobile   | 100          | 1           |
```

```
| 2019-07-01 | both     | 200          | 1           |
| 2019-07-02 | desktop  | 100          | 1           |
| 2019-07-02 | mobile   | 100          | 1           |
| 2019-07-02 | both     | 0            | 0           |
+------------+----------+--------------+-------------+
On 2019-07-01, user 1 purchased using both desktop and mobile, user 2 purchased u
On 2019-07-02, user 2 purchased using mobile only, user 3 purchased using desktop
```

Solution

SQL

```sql
SELECT aa.spend_date,
       aa.platform,
       COALESCE(bb.total_amount, 0) total_amount,
       COALESCE(bb.total_users,0) total_users
FROM
    (SELECT DISTINCT(spend_date), a.platform   -- table aa
    FROM Spending JOIN
        (SELECT 'desktop' AS platform UNION
        SELECT 'mobile' AS platform UNION
        SELECT 'both' AS platform
        ) a
    ) aa
    LEFT JOIN
    (SELECT spend_date,                        -- table bb
            platform,
            SUM(amount) total_amount,
            COUNT(user_id) total_users
    FROM
        (SELECT spend_date,
                user_id,
                (CASE COUNT(DISTINCT platform)
                    WHEN 1 THEN platform
                    WHEN 2 THEN 'both'
                    END) platform,
                SUM(amount) amount
        FROM Spending
```

```
    GROUP BY spend_date, user_id
    ) b
GROUP BY spend_date, platform
) bb
ON aa.platform = bb.platform AND
aa.spend_date = bb.spend_date
```

1132. Reported Posts II | Medium | 🔒 LeetCode

Table: `Actions`

TEXT

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| user_id       | int     |
| post_id       | int     |
| action_date   | date    |
| action        | enum    |
| extra         | varchar |
+---------------+---------+
There is no primary key for this table, it may have duplicate rows.
The action column is an ENUM type of ('view', 'like', 'reaction', 'comment', 'rep
The extra column has optional information about the action such as a reason for r
```

Table: `Removals`

TEXT

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| post_id       | int     |
| remove_date   | date    |
+---------------+---------+
post_id is the primary key of this table.
Each row in this table indicates that some post was removed as a result of being
```

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

The query result format is in the following example:

TEXT

```
Actions table:
+---------+---------+-------------+--------+--------+
| user_id | post_id | action_date | action | extra  |
+---------+---------+-------------+--------+--------+
| 1       | 1       | 2019-07-01  | view   | null   |
| 1       | 1       | 2019-07-01  | like   | null   |
| 1       | 1       | 2019-07-01  | share  | null   |
| 2       | 2       | 2019-07-04  | view   | null   |
| 2       | 2       | 2019-07-04  | report | spam   |
| 3       | 4       | 2019-07-04  | view   | null   |
| 3       | 4       | 2019-07-04  | report | spam   |
| 4       | 3       | 2019-07-02  | view   | null   |
| 4       | 3       | 2019-07-02  | report | spam   |
| 5       | 2       | 2019-07-03  | view   | null   |
| 5       | 2       | 2019-07-03  | report | racism |
| 5       | 5       | 2019-07-03  | view   | null   |
| 5       | 5       | 2019-07-03  | report | racism |
+---------+---------+-------------+--------+--------+

Removals table:
+---------+-------------+
| post_id | remove_date |
+---------+-------------+
| 2       | 2019-07-20  |
| 3       | 2019-07-18  |
+---------+-------------+

Result table:
+----------------------+
| average_daily_percent |
+----------------------+
| 75.00                |
```

```
+----------------------+
The percentage for 2019-07-04 is 50% because only one post of two spam reported p
The percentage for 2019-07-02 is 100% because one post was reported as spam and i
The other days had no spam reports so the average is (50 + 100) / 2 = 75%
Note that the output is only one number and that we do not care about the remove
```

Solution

SQL

```sql
WITH t1 AS(
SELECT a.action_date, (COUNT(DISTINCT r.post_id))/(COUNT(DISTINCT a.post_id)) AS
FROM (SELECT action_date, post_id
FROM actions
WHERE extra = 'spam' AND action = 'report') a
LEFT JOIN
removals r
ON a.post_id = r.post_id
GROUP BY a.action_date)

SELECT ROUND(AVG(t1.result)*100,2) AS  average_daily_percent
FROM t1
```

1141. User Activity for the Past 30 Days I | Easy | 🔒 LeetCode

Table: `Activity`

TEXT

```
+---------------+---------+
| 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', 'scro
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 some day if he/she made at least one activity on that day.

The query result format is in the following example:

TEXT

```
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   |
+---------+------------+---------------+---------------+

Result table:
+------------+--------------+
| day        | active_users |
+------------+--------------+
| 2019-07-20 | 2            |
| 2019-07-21 | 2            |
+------------+--------------+
Note that we do not care about days with zero active users.
```

Solution

```sql
SELECT activity_date AS day, COUNT(DISTINCT user_id) AS active_users
FROM activity
WHERE activity_date > '2019-06-26' AND activity_date < '2019-07-27'
GROUP BY activity_date
```

1142. User Activity for the Past 30 Days II | Easy | 🔒 LeetCode

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', 'scro
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 average number of sessions per user for a period of 30 days ending 2019-07-27 inclusively, rounded to 2 decimal places. The sessions we want to count for a user are those with at least one activity in that time period.

The query result format is in the following example:

```
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   |
| 3       | 5          | 2019-07-21    | open_session  |
| 3       | 5          | 2019-07-21    | scroll_down   |
| 3       | 5          | 2019-07-21    | end_session   |
| 4       | 3          | 2019-06-25    | open_session  |
| 4       | 3          | 2019-06-25    | end_session   |
+---------+------------+---------------+---------------+

Result table:
+---------------------------+
| average_sessions_per_user |
+---------------------------+
| 1.33                      |
+---------------------------+
User 1 and 2 each had 1 session in the past 30 days while user 3 had 2 sessions s
```

Solution

SQL

```sql
SELECT IFNULL(ROUND(AVG(a.num),2),0) AS average_sessions_per_user
FROM (
SELECT COUNT(DISTINCT session_id) AS num
FROM activity
WHERE activity_date BETWEEN '2019-06-28' AND '2019-07-27'
GROUP BY user_id) a
```

1148. Article Views I | Easy | 🔒 LeetCode

Table: `Views`

```
TEXT
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| article_id    | int     |
| author_id     | int     |
| viewer_id     | int     |
| view_date     | date    |
+---------------+---------+
There is no primary key for this table, it may have duplicate rows.
Each row of this table indicates that some viewer viewed an article (written by s
Note that equal author_id and viewer_id indicate the same person.
```

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

The query result format is in the following example:

```
TEXT
Views table:
+------------+-----------+-----------+------------+
| article_id | author_id | viewer_id | view_date  |
+------------+-----------+-----------+------------+
| 1          | 3         | 5         | 2019-08-01 |
| 1          | 3         | 6         | 2019-08-02 |
| 2          | 7         | 7         | 2019-08-01 |
| 2          | 7         | 6         | 2019-08-02 |
| 4          | 7         | 1         | 2019-07-22 |
| 3          | 4         | 4         | 2019-07-21 |
| 3          | 4         | 4         | 2019-07-21 |
+------------+-----------+-----------+------------+

Result table:
+------+
```

```
| id   |
+------+
| 4    |
| 7    |
+------+
```

Solution

```sql
SELECT DISTINCT author_id AS id
FROM Views
WHERE author_id = viewer_id
ORDER BY author_id
```

1149. Article Views II | Medium | 🔒 LeetCode

Table: `Views`

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| article_id    | int     |
| author_id     | int     |
| viewer_id     | int     |
| view_date     | date    |
+---------------+---------+
There is no primary key for this table, it may have duplicate rows.
Each row of this table indicates that some viewer viewed an article (written by s
Note that equal author_id and viewer_id indicate the same person.
```

Write an SQL query to find all the people who viewed more than one article on the same date, sorted in ascending order by their id.

The query result format is in the following example:

```
Views table:
+------------+-----------+-----------+------------+
| article_id | author_id | viewer_id | view_date  |
+------------+-----------+-----------+------------+
| 1          | 3         | 5         | 2019-08-01 |
| 3          | 4         | 5         | 2019-08-01 |
| 1          | 3         | 6         | 2019-08-02 |
| 2          | 7         | 7         | 2019-08-01 |
| 2          | 7         | 6         | 2019-08-02 |
| 4          | 7         | 1         | 2019-07-22 |
| 3          | 4         | 4         | 2019-07-21 |
| 3          | 4         | 4         | 2019-07-21 |
+------------+-----------+-----------+------------+

Result table:
+------+
| id   |
+------+
| 5    |
| 6    |
+------+
```

Solution

```sql
SELECT DISTINCT viewer_id AS id#, COUNT(DISTINCT article_id) AS total
FROM views
GROUP BY viewer_id, view_date
HAVING count(DISTINCT article_id)>1
ORDER BY 1
```

1158. Market Analysis I | Medium | 🔒 LeetCode

Table: `Users`

```
TEXT
+----------------+---------+
| 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 ca
```

Table: `Orders`

```
TEXT
+---------------+---------+
| 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`

```
TEXT
+---------------+---------+
| 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.

The query result format is in the following example:

TEXT

```
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         |
```

```
+---------+------------+

Result table:
+-----------+------------+----------------+
| 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              |
+-----------+------------+----------------+
```

Solution

SQL

```sql
SELECT user_id AS buyer_id, join_date, coalesce(a.orders_in_2019,0)
FROM users
LEFT JOIN
(
SELECT buyer_id, coalesce(count(*), 0) AS orders_in_2019
FROM orders o
JOIN users u
ON u.user_id = o.buyer_id
WHERE extract('year' FROM order_date) = 2019
GROUP BY buyer_id) a
ON users.user_id = a.buyer_id
```

1159. Market Analysis II | Hard | 🔒 LeetCode

Table: `Users`

TEXT

```
+----------------+---------+
| 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 ca
```

Table: `Orders`

```
TEXT
+---------------+---------+
| 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`

```
TEXT
+---------------+---------+
| 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, whether the brand of the second item (by date) they sold is their favorite brand. If a user sold less than two items, report the answer for that user as no.

It is guaranteed that no seller sold more than one item on a day.

The query result format is in the following example:

TEXT

```
Users table:
+---------+------------+----------------+
| user_id | join_date  | favorite_brand |
+---------+------------+----------------+
| 1       | 2019-01-01 | Lenovo         |
| 2       | 2019-02-09 | Samsung        |
| 3       | 2019-01-19 | LG             |
| 4       | 2019-05-21 | HP             |
+---------+------------+----------------+

Orders table:
+----------+------------+---------+----------+-----------+
| order_id | order_date | item_id | buyer_id | seller_id |
+----------+------------+---------+----------+-----------+
| 1        | 2019-08-01 | 4       | 1        | 2         |
| 2        | 2019-08-02 | 2       | 1        | 3         |
| 3        | 2019-08-03 | 3       | 2        | 3         |
| 4        | 2019-08-04 | 1       | 4        | 2         |
| 5        | 2019-08-04 | 1       | 3        | 4         |
| 6        | 2019-08-05 | 2       | 2        | 4         |
+----------+------------+---------+----------+-----------+

Items table:
+---------+------------+
| item_id | item_brand |
+---------+------------+
| 1       | Samsung    |
| 2       | Lenovo     |
| 3       | LG         |
| 4       | HP         |
+---------+------------+

Result table:
+-----------+--------------------+
| seller_id | 2nd_item_fav_brand |
```

```
+-----------+--------------------+
| 1         | no                 |
| 2         | yes                |
| 3         | yes                |
| 4         | no                 |
+-----------+--------------------+

The answer for the user with id 1 is no because they sold nothing.
The answer for the users with id 2 and 3 is yes because the brands of their secon
The answer for the user with id 4 is no because the brand of their second sold it
```

Solution

SQL

```sql
#Solution- 1:
SELECT user_id AS seller_id,
       IF(ISNULL(item_brand), "no", "yes") AS 2nd_item_fav_brand
FROM Users LEFT JOIN
(SELECT seller_id, item_brand
FROM Orders INNER JOIN Items
ON Orders.item_id = Items.item_id
WHERE (seller_id, order_date) IN
(SELECT seller_id, MIN(order_date) AS order_date
 FROM Orders
 WHERE (seller_id, order_date) NOT IN
 (SELECT seller_id, MIN(order_date) FROM Orders GROUP BY seller_id)
GROUP BY seller_id)
 ) AS t
ON Users.user_id = t.seller_id and favorite_brand = item_brand

#Solution- 2:
WITH t1 AS(
SELECT user_id,
CASE WHEN favorite_brand = item_brand THEN "yes"
ELSE "no"
END AS 2nd_item_fav_brand
FROM users u LEFT JOIN
```

```
(SELECT o.item_id, seller_id, item_brand, RANK() OVER(PARTITION BY seller_id ORDE
FROM orders o join items i
USING (item_id)) a
ON u.user_id = a.seller_id
WHERE a.rk = 2)

SELECT u.user_id AS seller_id, COALESCE(2nd_item_fav_brand,"no") AS 2nd_item_fav_
FROM users u LEFT JOIN t1
USING(user_id)
```

1164. Product Price at a Given Date | Medium | 🔒 LeetCode

Table: `Products`

```
TEXT
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| product_id    | int     |
| new_price     | int     |
| change_date   | date    |
+---------------+---------+
(product_id, change_date) is the primary key of this table.
Each row of this table indicates that the price of some product was changed to a
```

Write an SQL query to find the prices of all products on 2019-08-16. Assume the price of all products before any change is 10.

The query result format is in the following example:

```
TEXT
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  |
+------------+-----------+-------------+
```

Result table:

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

Solution

SQL

```sql
#Solution- 1:
WITH t1 AS (
SELECT a.product_id, new_price
FROM(
SELECT product_id, max(change_date) AS date
FROM products
WHERE change_date<='2019-08-16'
GROUP BY product_id) a
JOIN products p
ON a.product_id = p.product_id AND a.date = p.change_date),

t2 AS (
SELECT distinct product_id
    FROM products)

SELECT t2.product_id, coalesce(new_price,10) AS price
FROM t2 LEFT JOIN t1
ON t2.product_id = t1.product_id
ORDER BY price DESC
```

```
#Solution- 2:
SELECT t1.product_id AS product_id, IF(ISNULL(t2.price), 10, t2.price) AS price
FROM
  (SELECT distinct product_id
  FROM Products) AS t1 LEFT JOIN
  (SELECT product_id, new_price AS price
   FROM Products
   WHERE (product_id, change_date) in
      (SELECT product_id, max(change_date)
      FROM Products
      WHERE change_date <='2019-08-16'
      GROUP BY product_id)) AS t2
ON t1.product_id = t2.product_id
```

1173. Immediate Food Delivery I | Easy | 🔒 LeetCode

Table: `Delivery`

```
TEXT
+-----------------------------+---------+
| Column Name                 | Type    |
+-----------------------------+---------+
| delivery_id                 | int     |
| customer_id                 | int     |
| order_date                  | date    |
| customer_pref_delivery_date | date    |
+-----------------------------+---------+
delivery_id is the primary key of this table.
The table holds information about food delivery to customers that make orders at
```

If the preferred delivery date of the customer is the same as the order date then the order is called immediate otherwise it's called scheduled.

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

The query result format is in the following example:

TEXT

```
Delivery table:
+-------------+-------------+------------+-----------------------------+
| delivery_id | customer_id | order_date | customer_pref_delivery_date |
+-------------+-------------+------------+-----------------------------+
| 1           | 1           | 2019-08-01 | 2019-08-02                  |
| 2           | 5           | 2019-08-02 | 2019-08-02                  |
| 3           | 1           | 2019-08-11 | 2019-08-11                  |
| 4           | 3           | 2019-08-24 | 2019-08-26                  |
| 5           | 4           | 2019-08-21 | 2019-08-22                  |
| 6           | 2           | 2019-08-11 | 2019-08-13                  |
+-------------+-------------+------------+-----------------------------+

Result table:
+----------------------+
| immediate_percentage |
+----------------------+
| 33.33                |
+----------------------+
The orders with delivery id 2 and 3 are immediate while the others are scheduled.
```

Solution

SQL

```sql
#Solution- 1:
SELECT
ROUND(SUM(CASE WHEN order_date=customer_pref_delivery_date THEN 1 ELSE 0 END)/cou
FROM Delivery;

#Solution- 2:
SELECT
ROUND(avg(CASE WHEN order_date=customer_pref_delivery_date THEN 1 ELSE 0 END)*100
FROM delivery
```