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
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.

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

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

Solution

```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):

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

180. Consecutive Numbers | Medium | LeetCode

Table: `Logs`

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

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`

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

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

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

SQL

```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;
```

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;
```

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 Th 6 0 ies | 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.

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

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

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
```

```
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
```

6 | 0

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          |
+----+------------+-------------+

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

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

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;
```

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 tot[illegible]hich was canceled.
- The request with Id=8 was ma[illegible]nned 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
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`

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

TEXT

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

TEXT

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

Solution

SQL

```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
# 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`

TEXT

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

TEXT

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

Solution

SQL

```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
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`

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

```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 |
```

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:

```text
| 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.

TEXT

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

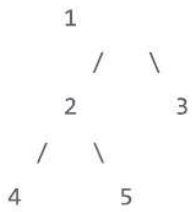
TEXT

```
+----+------+
| 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 Iceas: 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
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`:

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

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

Solution

```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`

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

```
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 average experience years of all the employees for each project, rounded to 2 digits.

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

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

```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.

```text
| requester_id | accepter_id | accept_date|
|--------------|-------------|------------|
| 1            | 2           | 2016_06-03 |
| 1            | 3           | 2016-06-08 |
```

6 | 0

```
| 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:

TEXT

```
| 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 = compa
```

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.

TEXT

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

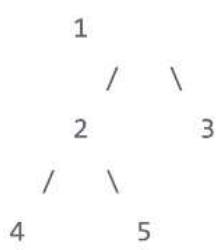
TEXT

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

6 | 0

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 |
```

```sql
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
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:

6 | 0

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

```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

```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  |
```

```
|    3    | Jealiles |
+---------+----------+
```

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     |
```