LeetCode Challenge

Monday, February 12, 2018 23:50

LeetCode Challenge:

Shortest Distance in a line:

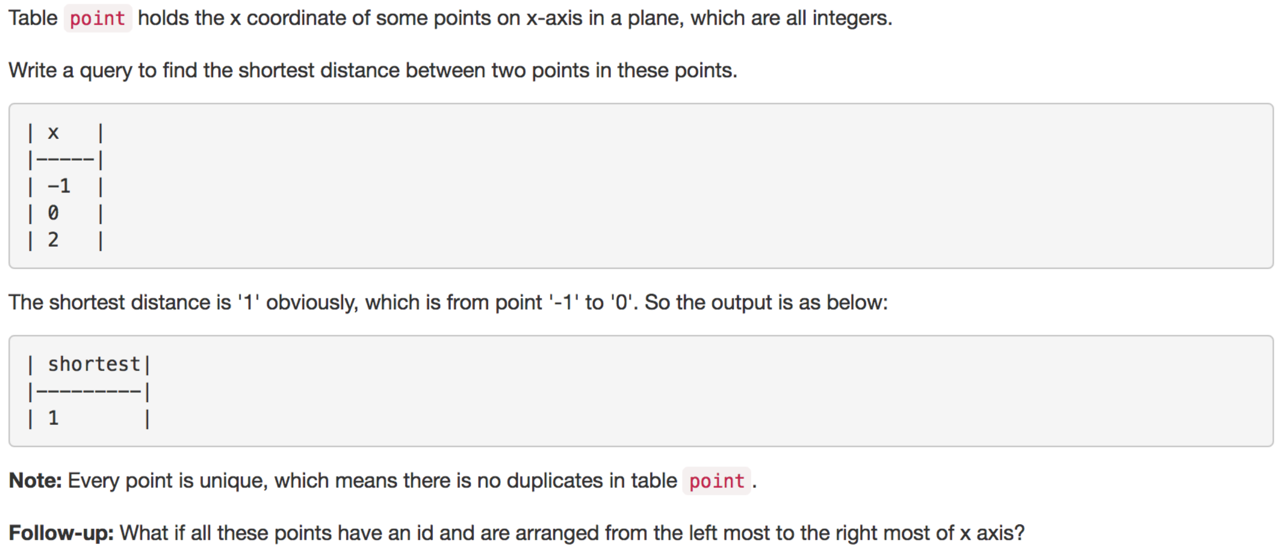


Table `point` holds the x coordinate of some points on x-axis in a plane, which are all integers.

Write a query to find the shortest distance between two points in these points.

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

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

```
| 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?

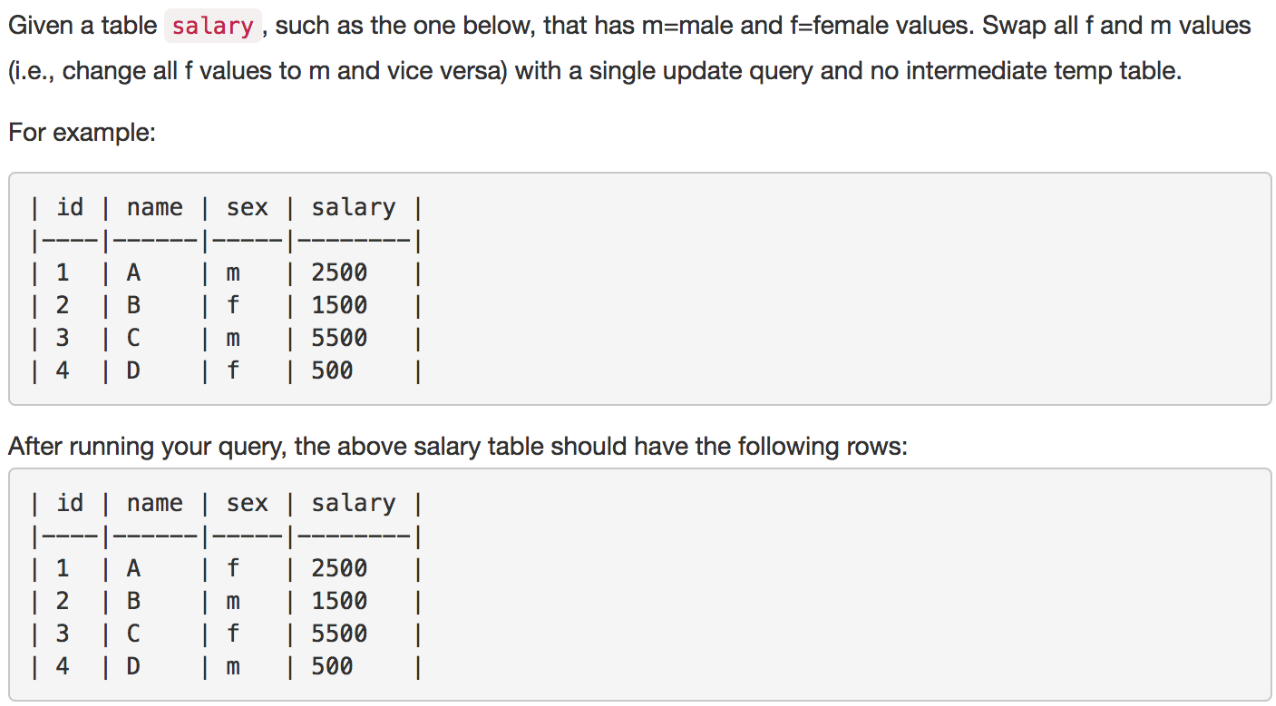
Step 1:

```
/*SELECT p1.x, p2.x, ABS(p2.x - p1.x) AS distance
FROM point p1
JOIN point p2 ON p1.x != p2.x;*/
```

Step 2:

```
SELECT MIN(ABS(p2.x - p1.x)) AS shortest
FROM point p1
JOIN point p2 ON p1.x != p2.x;
```

Swap Sap Salary:



Given a table `salary`, such as the one below, that has m=male and f=female values. Swap all f and m values (i.e., change all f values to m and vice versa) with a single update query and no intermediate temp table.

For example:

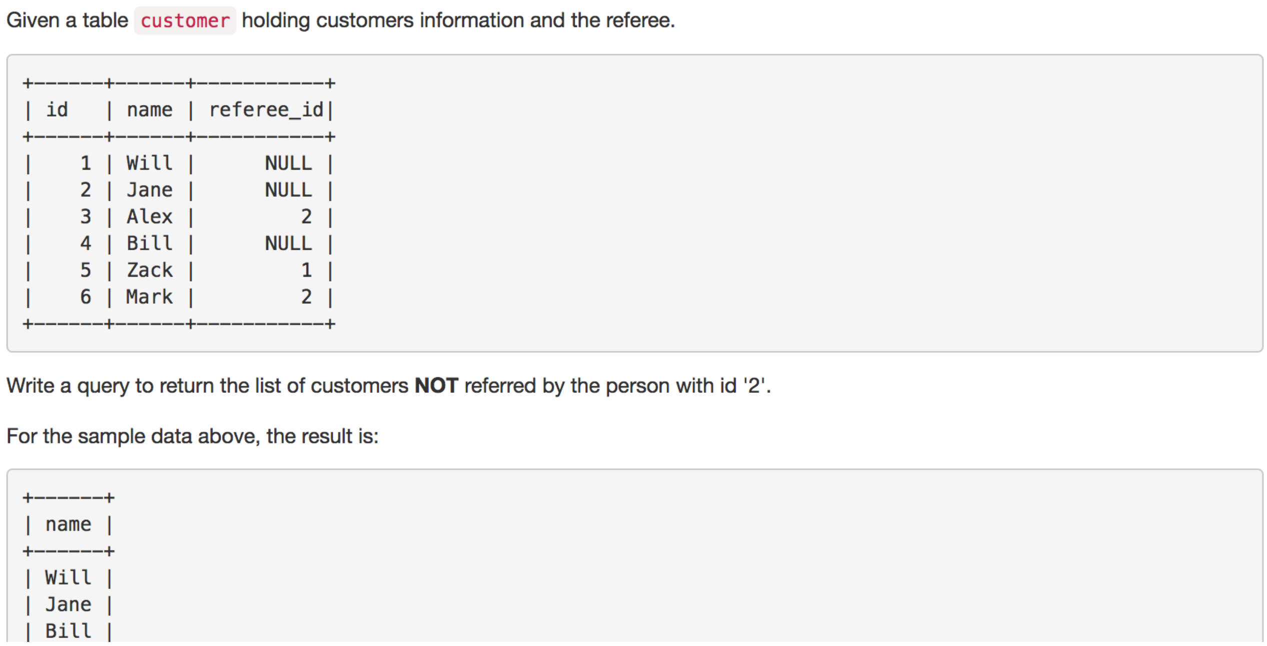
```
| id | name | sex | salary |
|----|------|-----|--------|
| 1  | A    | m   | 2500   |
| 2  | B    | f   | 1500   |
| 3  | C    | m   | 5500   |
| 4  | D    | f   | 500    |
```

After running your query, the above salary table should have the following rows:

```
| id | name | sex | salary |
|----|------|-----|--------|
| 1  | A    | f   | 2500   |
| 2  | B    | m   | 1500   |
| 3  | C    | f   | 5500   |
| 4  | D    | m   | 500    |
```

```
UPDATE salary
SET
        Sex = CASE WHEN sex = 'f', THEN 'm'
                ELSE 'f' END;
```

Find Customer Reference:



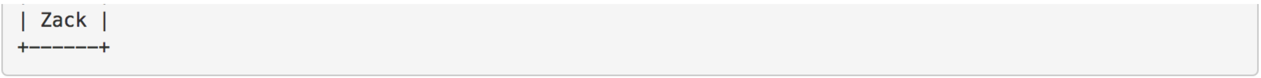
Given a table `customer` holding customers information and the referee.

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

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

For the sample data above, the result is:

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

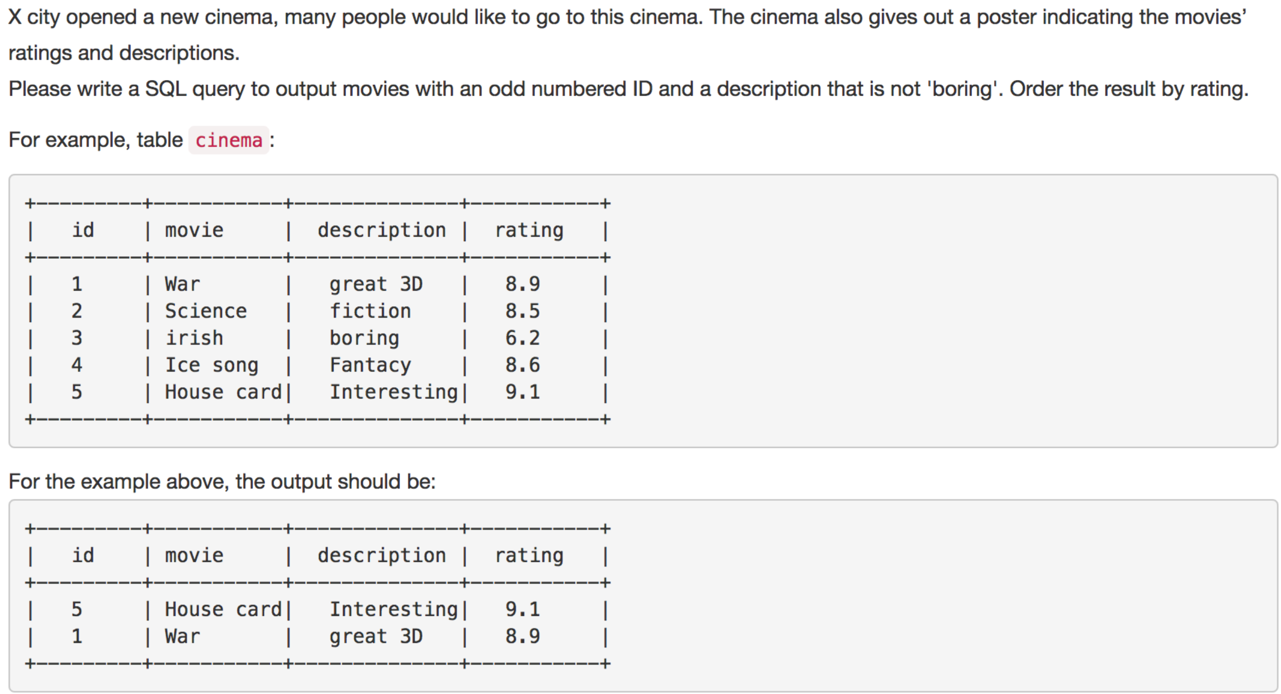


```
| Zack |
+------+
```

Point: IS NULL and <>

```
SELECT name
FROM customer
WHERE referee_id IS NULL OR referee_id <> 2;
```

Not Boring Movie:



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

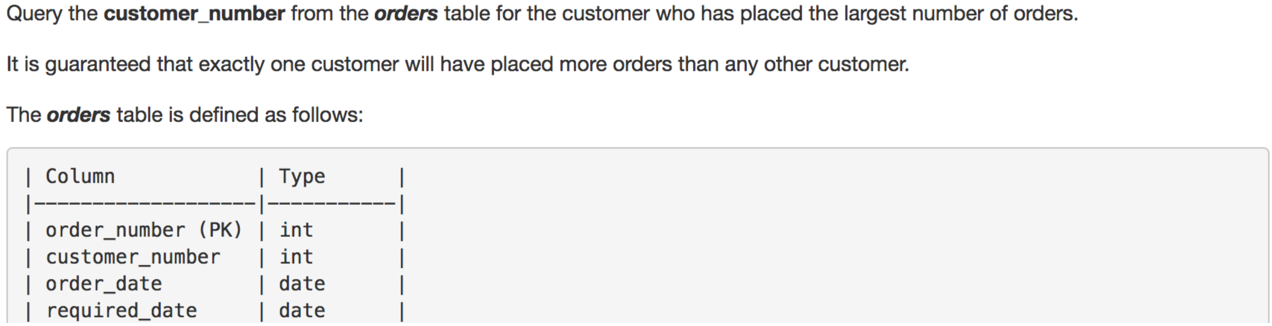
For the example above, the output should be:

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

Point: (X % 2) <> 0 / MOD(X, 2) = 1 is odd number

```
SELECT *
FROM cinema
WHERE (id % 2) <>0  AND description != 'boring'
ORDER BY rating DESC;
```

Customer Placing the Largest Number of Orders:

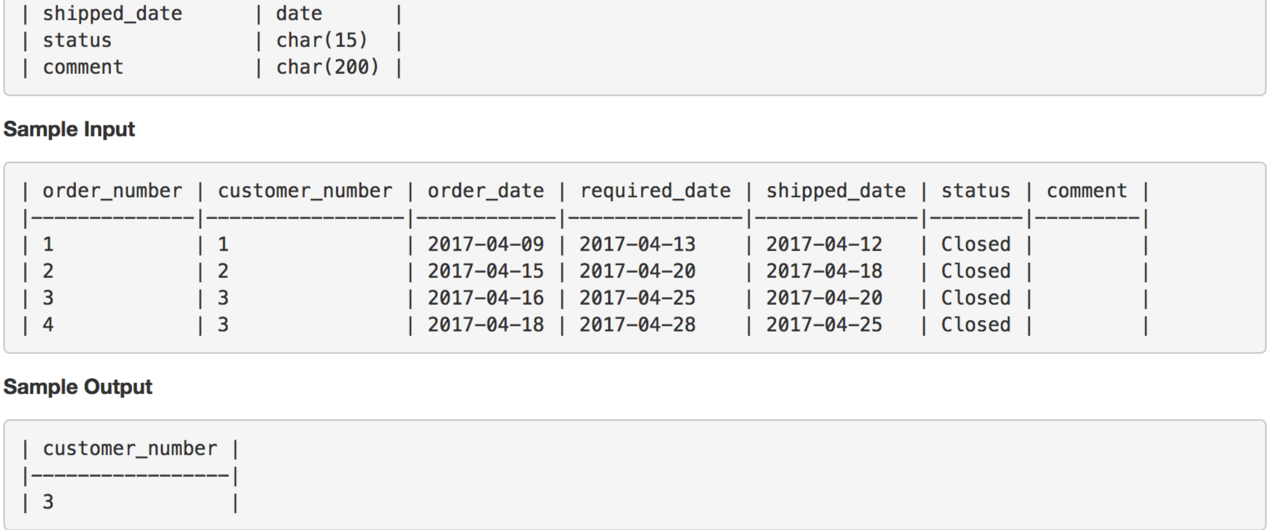


Query the **customer_number** from the ***orders*** table for the customer who has placed the largest number of orders.

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

The ***orders*** table is defined as follows:

```
| 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

```
| order_number | customer_number | order_date | required_date | shipped_date | status | comment |
|--------------|-----------------|------------|---------------|--------------|--------|---------|
| 1            | 1               | 2017-04-09 | 2017-04-13    | 2017-04-12   | Closed |         |
| 2            | 2               | 2017-04-15 | 2017-04-20    | 2017-04-18   | Closed |         |
| 3            | 3               | 2017-04-16 | 2017-04-25    | 2017-04-20   | Closed |         |
| 4            | 3               | 2017-04-18 | 2017-04-28    | 2017-04-25   | Closed |         |
```

Sample Output

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

Point: LIMIT

```
SELECT customer_number
FROM orders
GROUP BY customer_number
ORDER BY COUNT(order_number) DESC
LIMIT 1;
```

Second Highest Salary:

```
SELECT MAX(salary) AS TheSecondHighestSalary
FROM employee
WHERE salary < (SELECT MAX(salary)
                FROM employee);
```

Odd/Even number:

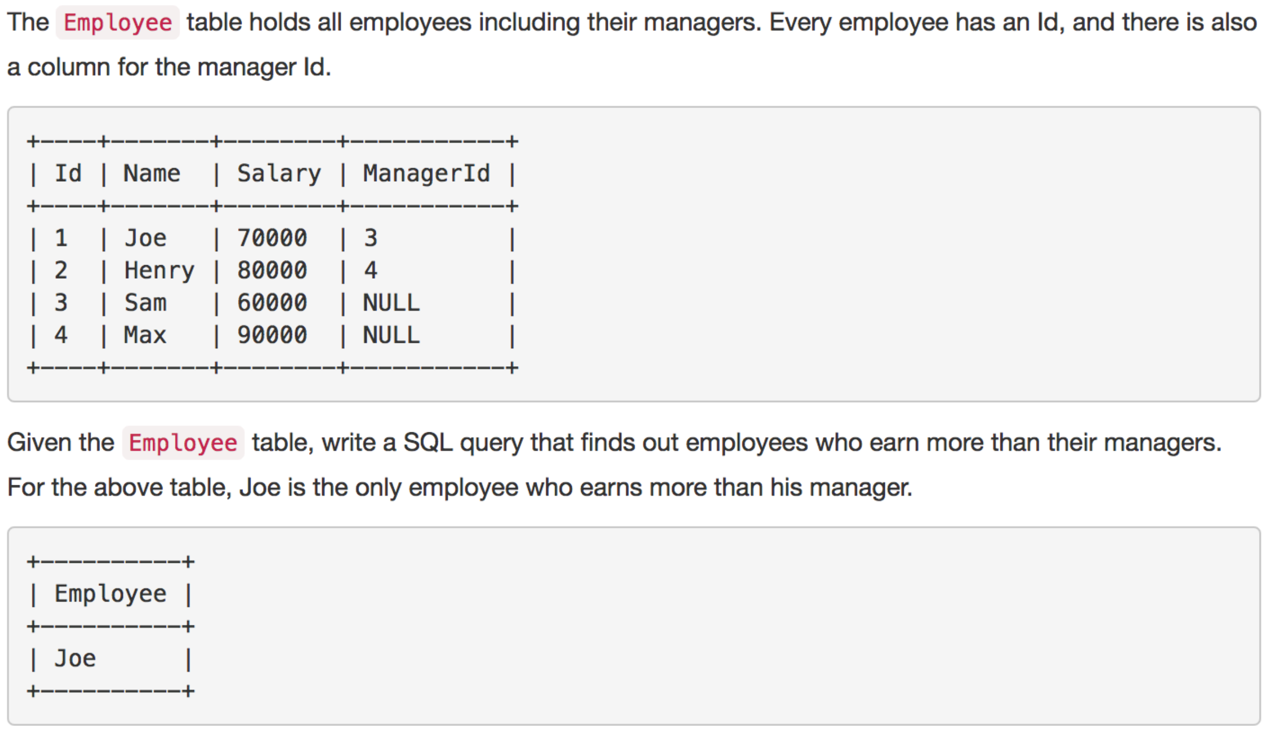
Odd: (X % 2) <> 0 / = 1

Even: (X % 2) = 0

Find Duplicates:

SELECT Email
FROM Person
GROUP BY Email
HAVING COUNT(Email) > 1;

Employees Earning More Than Manager:



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

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

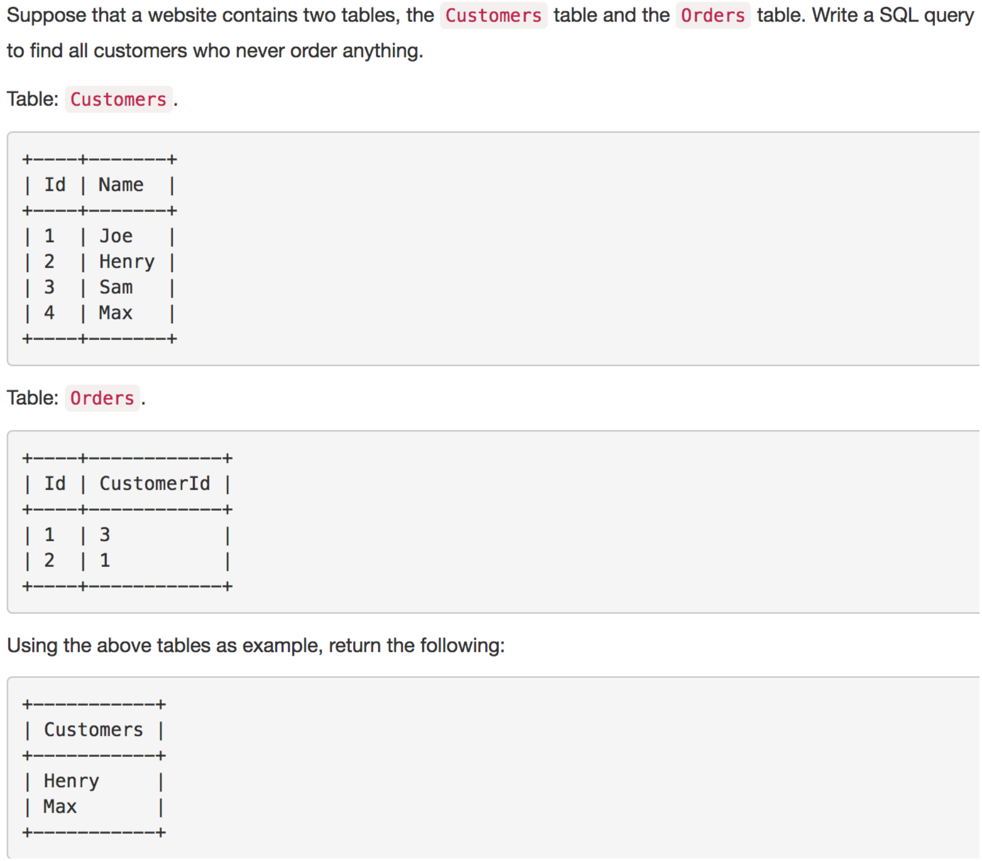
Given the `Employee` table, write a SQL query that finds out employees who earn more than their managers. For the above table, Joe is the only employee who earns more than his manager.

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

Approach I:
SELECT a.name AS Employee
FROM Employee AS a
INNER JOIN Employee AS b
ON a.ManagerId = b.Id AND a.salary > b.salary;

Approach II:
SELECT a.Name AS Employee
FROM Employee AS a, Employee AS b
WHERE a.ManagerId = b.Id AND a.salary > b.salary;

Customer Who Never Ordered:



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`.

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

Table: `Orders`.

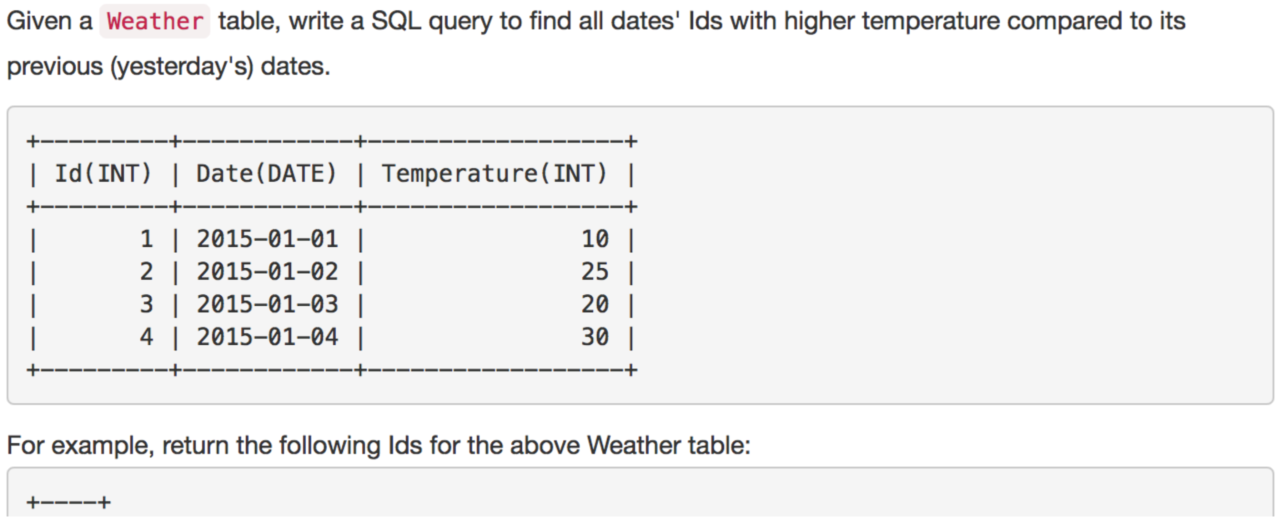
```
+----+------------+
| Id | CustomerId |
+----+------------+
| 1  | 3          |
| 2  | 1          |
+----+------------+
```

Using the above tables as example, return the following:

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

```
SELECT Name AS Customers
FROM Customers
WHERE Id NOT IN (SELECT CustomerId FROM Orders);
```

Rising Temperature:

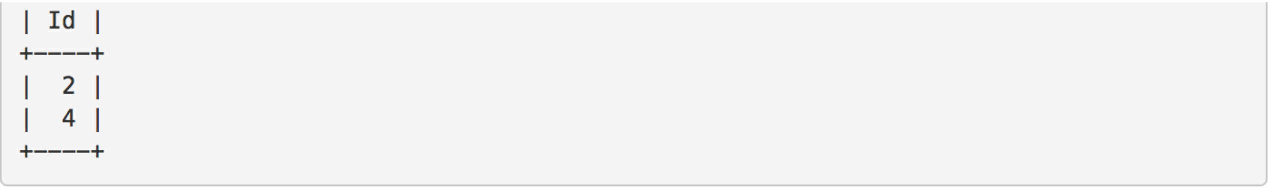


Given a `Weather` table, write a SQL query to find all dates' Ids with higher temperature compared to its previous (yesterday's) dates.

```
+---------+------------+------------------+
| Id(INT) | Date(DATE) | Temperature(INT) |
+---------+------------+------------------+
|       1 | 2015-01-01 |               10 |
|       2 | 2015-01-02 |               25 |
|       3 | 2015-01-03 |               20 |
|       4 | 2015-01-04 |               30 |
+---------+------------+------------------+
```

For example, return the following Ids for the above Weather table:

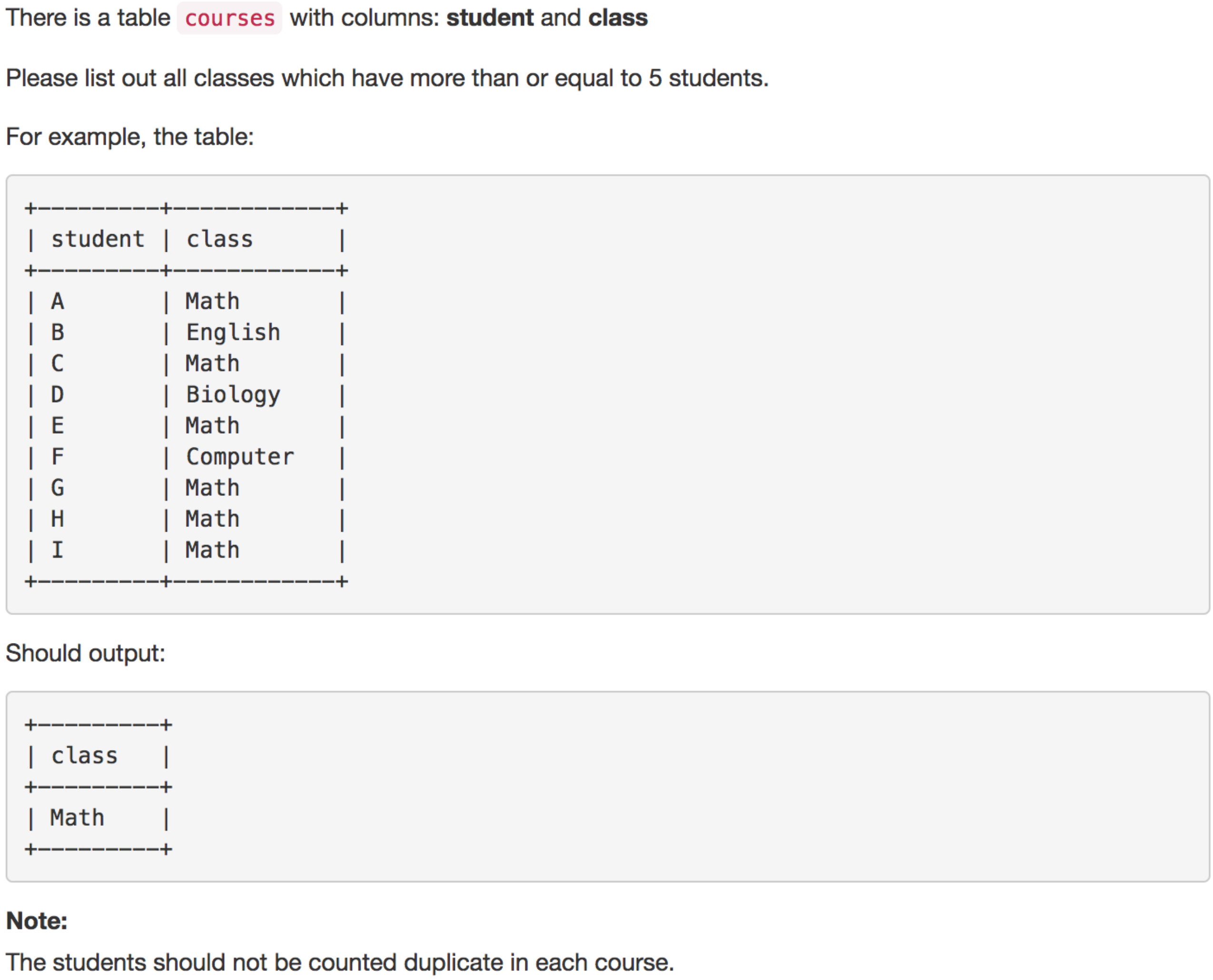
```
+----+
```



```
| Id |
+-----+
|  2 |
|  4 |
+-----+
```

SELECT a.Id
FROM Weather AS a
INNER JOIN Weather AS b
ON DATEDIFF(a.Date, b.Date) = 1 AND a.Temperature > b.Temperature;

Classes more than 5 students:



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:

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

Should output:

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

Note:

The students should not be counted duplicate in each course.

Approach I:

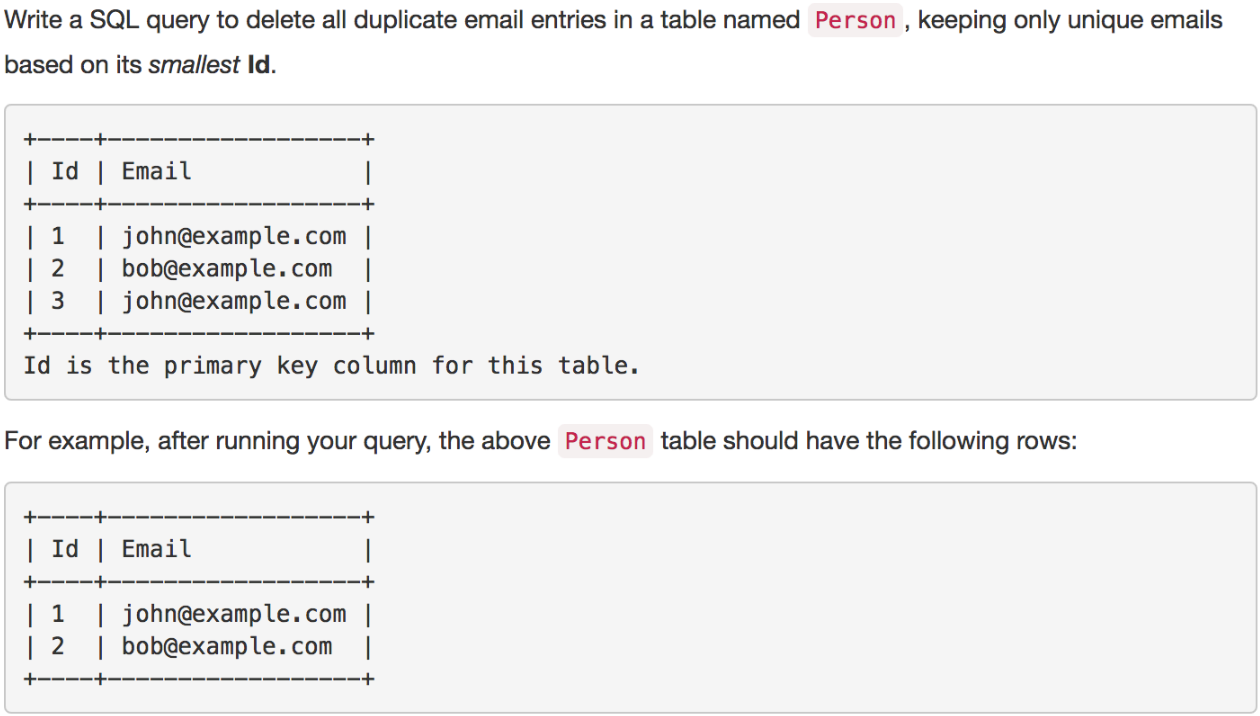
SELECT class
FROM courses
GROUP BY class
HAVING COUNT(DISTINCT student) >= 5

```
HAVING COUNT(DISTINCT student) >=5;
```

Approach II:

```
SELECT class
FROM (SELECT class, COUNT(DISTINCT student) AS num
          FROM courses
          GROUP BY class)
WHERE num >=5;
```

Delete Duplicate Emails:



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

```
+----+------------------+
| Id | Email            |
+----+------------------+
| 1  | john@example.com |
| 2  | bob@example.com  |
| 3  | john@example.com |
+----+------------------+
Id is the primary key column for this table.
```

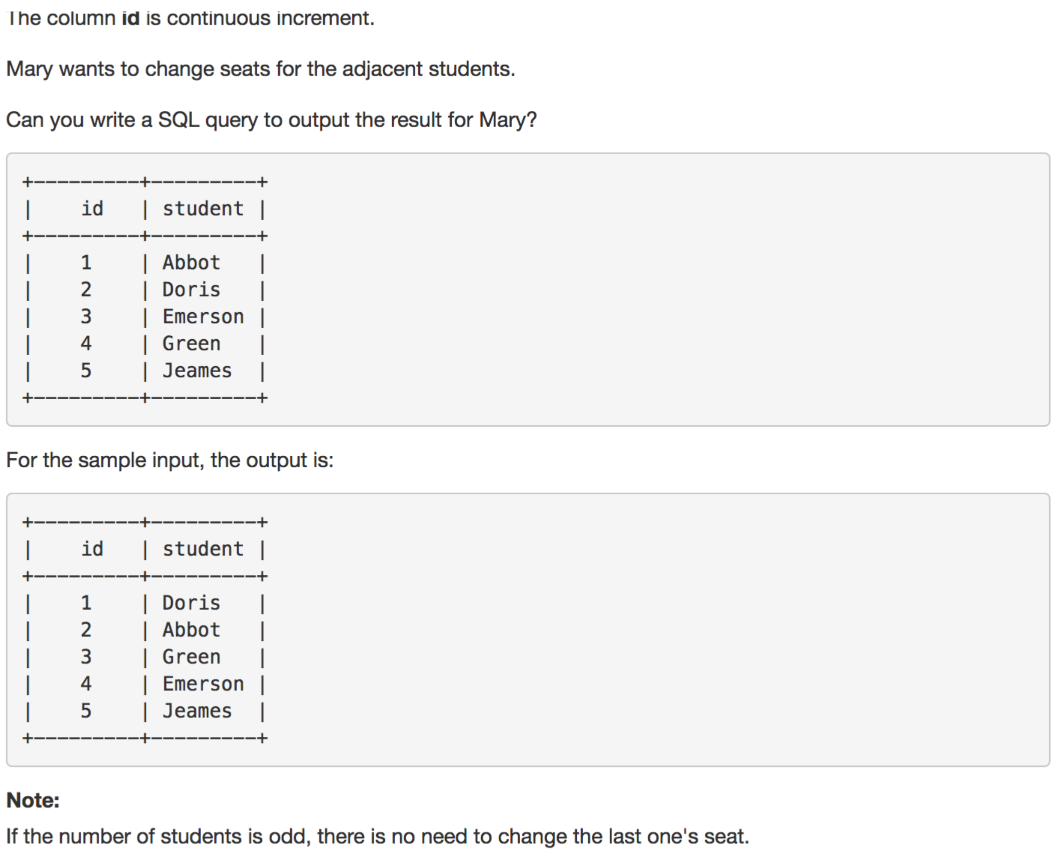
For example, after running your query, the above `Person` table should have the following rows:

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

```
DELETE p1.*
FROM Person AS p1
INNER JOIN Person AS p2
ON p1.Email = p2.Email AND p1.Id > p2.Id;
```

MOD(n, m) : returns the remainder of n divided by m

Exchange Seats:



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?

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

For the sample input, the output is:

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

Approach I:
# For students with odd id, the new id is (id+1) after switch unless it is the last seat. And for students with even id, the new id is (id-1). In order to know how many seats in total, we can use a subquery:

```
SELECT COUNT(*)
FROM seat;
```

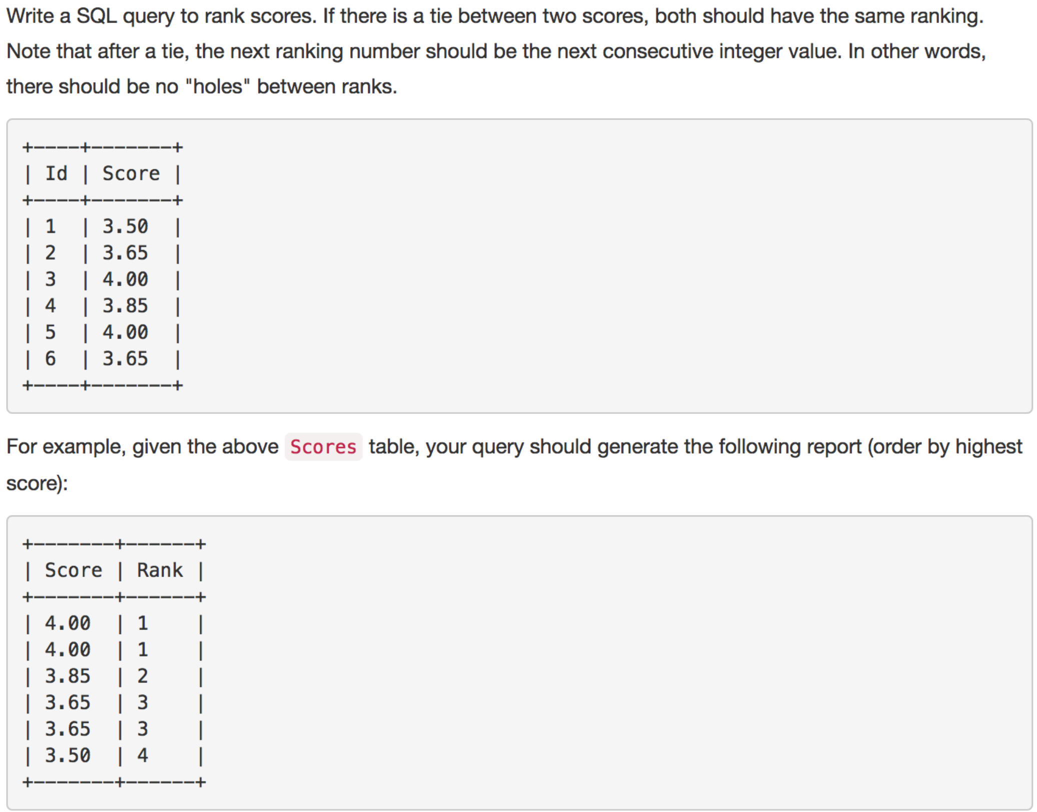
#Then, we can use the CASE statement and MOD() function to alter the seat id of each student.

```
SELECT
   (CASE
      WHEN MOD(id, 2) != 0 AND counts != id THEN id + 1
      WHEN MOD(id, 2) != 0 AND counts = id THEN id
      ELSE id - 1
   END) AS id,
   student
FROM
   seat,
   (SELECT
`
        COUNT(*) AS counts
    FROM
        seat) AS seat_counts
ORDER BY id ASC;
```

Approach II:
#Bit manipulation expression (id+1)^1-1 can calculate the new id after switch.

```
SELECT id, (id + 1)^1-1, student
FROM seat;
```

Rank Scores:



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.

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

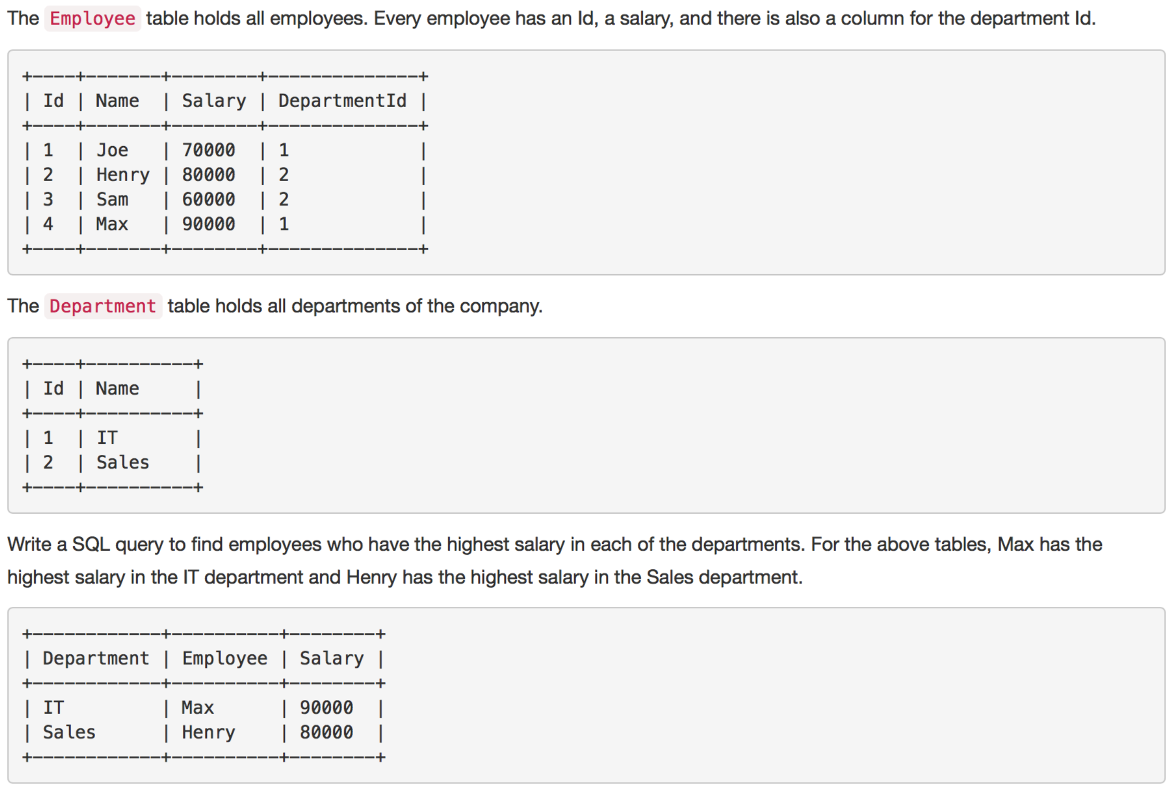
For example, given the above Scores table, your query should generate the following report (order by highest score):

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

```
SELECT a.Score, COUNT(b.Score) AS Rank
FROM Scores as a
INNER JOIN (SELECT DISTINCT Score
                      FROM Scores) as b
ON a.Score <= b.Score
GROUP BY a.Id
ORDER BY a.Score DESC;
```

ORDER BY a.Score DESC;

Department Highest Salary:



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

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

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

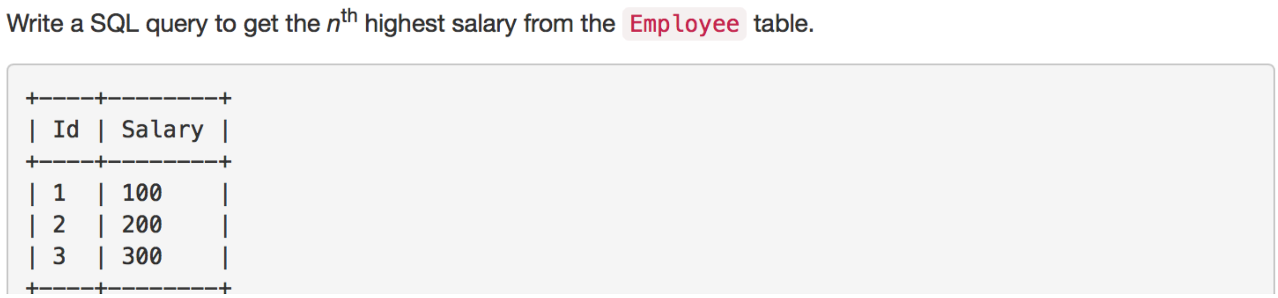
```
+----+----------+
| Id | Name     |
+----+----------+
| 1  | IT       |
| 2  | Sales    |
+----+----------+
```

Write a SQL query to find employees who have the highest salary in each of the departments. For the above tables, Max has the highest salary in the IT department and Henry has the highest salary in the Sales department.

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

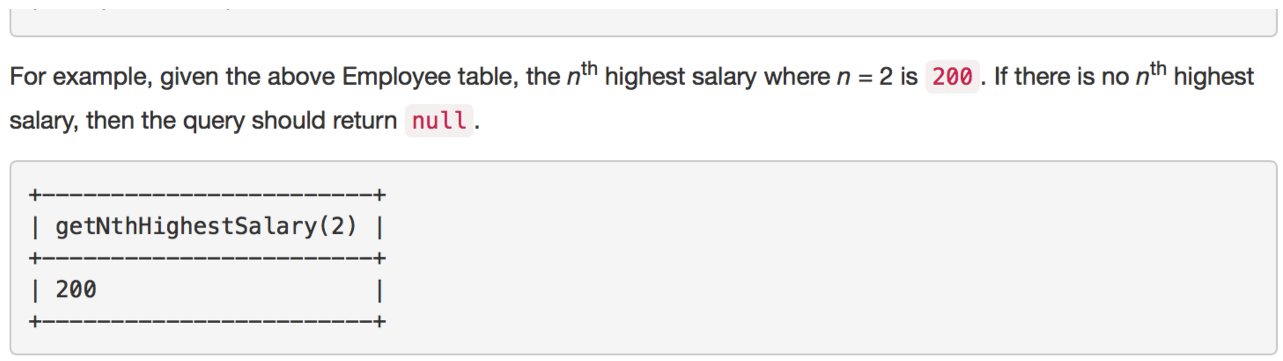
```
SELECT d.Name AS Department, e.Name AS Employee, Salary
FROM Employee AS e
INNER JOIN Department AS d
ON e.DepartmentId = d.Id
WHERE (e.DepartmentId, Salary) IN (SELECT DepartmentId, MAX(Salary) AS Salary
                                                       FROM Employee
                      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

Get Nth Highest Salary:



Write a SQL query to get the n^{th} highest salary from the `Employee` table.

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

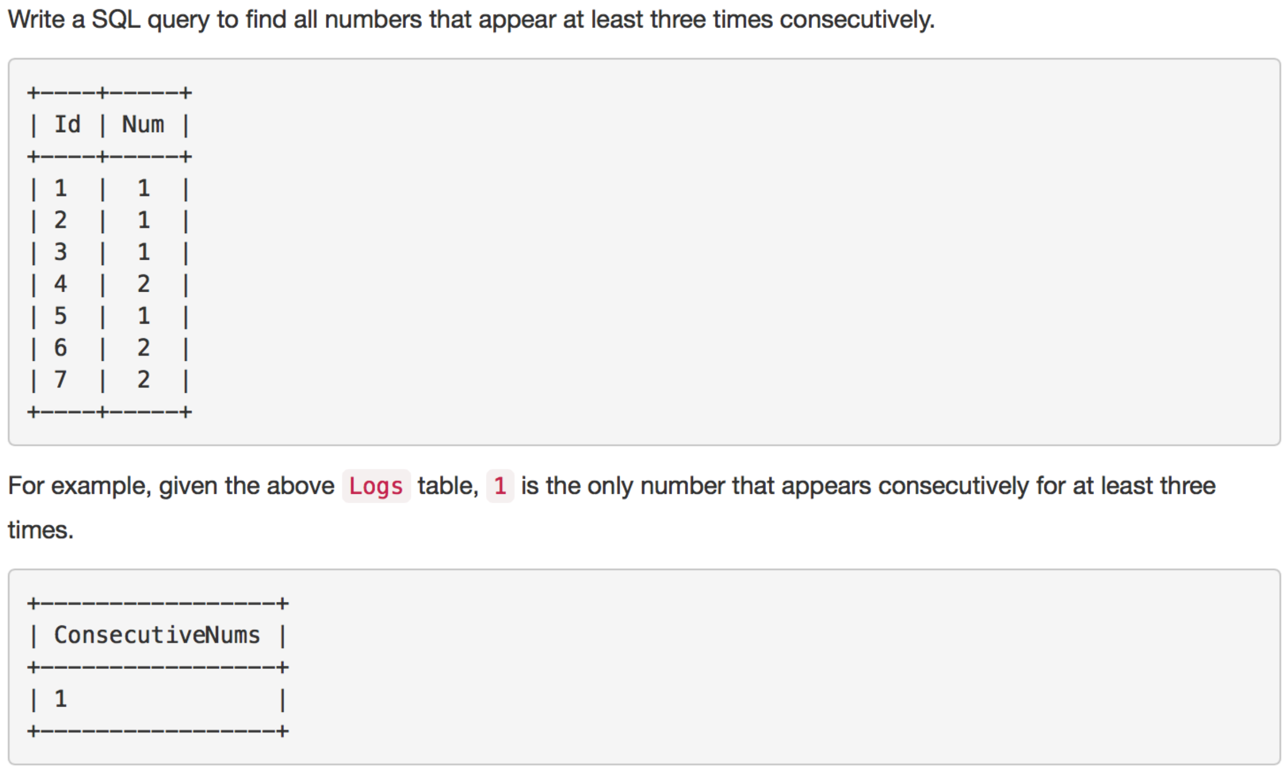


For example, given the above Employee table, the n^{th} highest salary where $n = 2$ is `200`. If there is no n^{th} highest salary, then the query should return `null`.

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

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

Consecutive Number:



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

```
+----+-----+
| Id | Num |
+----+-----+
| 1  |  1  |
| 2  |  1  |
| 3  |  1  |
| 4  |  2  |
| 5  |  1  |
| 6  |  2  |
| 7  |  2  |
+----+-----+
```

For example, given the above `Logs` table, `1` is the only number that appears consecutively for at least three times.

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

```
SELECT DISTINCT l1.Num AS ConsecutiveNums
SELECT DISTINCT l1.Num AS ConsecutiveNums
FROM Logs AS l1,
     Logs AS l2,
     Logs AS l3
WHERE
    l1.Id = l2.Id - 1
    AND l2.Id = l3.Id - 1
    AND l1.Num = l2.Num
    AND l2.Num = l3.Num
    ;
```