SQL on Leetcode -- Unlocked

175. Combine two Tables

Easy

Table: `Person`

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

Table: `Address`

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

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

Language: mysql

```
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

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

Language: mysql

```
# Solution 1
SELECT MAX(salary) as SecondHighestSalary
FROM employee
WHERE salary NOT IN (SELECT MAX(salary)
                FROM employee)

# Solution 2
SELECT
    (SELECT DISTINCT
            Salary
        FROM
            Employee
        ORDER BY Salary DESC
        LIMIT 1 OFFSET 1) AS SecondHighestSalary
;
```

Point: Select from a empty table is ***null*** (solution 2).

177. Nth Highest Salary

Medium

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

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

If there is no *n*th highest salary, then the query should return `null`.

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

Language: mysql

```mysql
# Solution 1
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

# Solution 2
CREATE FUNCTION getNthHighestSalary(N INT) RETURNS INT
BEGIN
  RETURN (
      SELECT DISTINCT salary
      FROM employee e1
      WHERE (SELECT COUNT(DISTINCT salary)
            FROM employee e2
            WHERE e1.salary<e2.salary) = N-1
  );
END
```

178. Rank Scores

Medium

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

Language: MS SQL Server

```
SELECT score,
    DENSE_RANK() OVER(ORDER BY score DESC) as rank
FROM scores
```

180. Consecutive Numbers

Medium

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

Language: mysql

```
SELECT DISTINCT t1.num as ConsecutiveNums
FROM logs t1, logs t2, logs t3
WHERE t1.num = t2.num AND t2.num = t3.num AND t1.id=t2.id-1 AND t2.id=t3.id-1
```

Compared to **601. Human Traffic of Stadium**.

181. Employees Earning More Than Their Managers

Easy

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

Language: mysql

```
SELECT e.Name as Employee
FROM Employee e LEFT JOIN Employee m
ON e.ManagerID = m.ID
WHERE e.Salary > m.Salary
```

182. Duplicate Emails

Easy

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

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

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

Note: All emails are in lowercase.

Language: mysql

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

183. Customers Who Never Order

Easy

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

Language: mysql

```
# Solution 1
SELECT name as customers
FROM customers a
LEFT JOIN orders b
ON a.Id = b.customerId
WHERE b.customerId is Null
```

```sql
# Solution 2
select customers.name as Customers
from customers
where customers.id not in
(
    select customerid from orders
);
```

184. Department Highest Salary

Medium

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  | Jim   | 90000  | 1            |
| 3  | Henry | 80000  | 2            |
| 4  | Sam   | 60000  | 2            |
| 5  | 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, your SQL query should return the following rows (order of rows does not matter).

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

Language:

```
SELECT d.name as department, e.name as employee, e.salary as salary
FROM employee e, department d
WHERE e.departmentid = d.id
    AND (e.salary,e.departmentid) in (SELECT MAX(salary),departmentid FROM
employee GROUP BY departmentid)
```

185. Department Top Three Salaries

Hard

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

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

Language: mysql

```
SELECT d.name AS 'department', e1.name AS 'employee', e1.salary
FROM department d, employee e1
WHERE d.id = e1.departmentid
    AND (SELECT COUNT(DISTINCT e2.salary)
        FROM employee e2
        WHERE e2.departmentid=e1.departmentid
        AND e2.salary>e1.salary) < 3
```

196. Delete Duplicate Emails

Easy

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

Note:

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

Language: mysql

```
DELETE p1 FROM person p1, person p2
WHERE p1.email=p2.email AND p1.id>p2.id
```

197. Rising Temperature

Easy

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) | RecordDate(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 |
+----+
```

Language: mysql

```
SELECT a.Id
FROM weather a
INNER JOIN weather b
ON a.recorddate = DATEADD(DAY,1,b.recorddate) and a.temperature > b.temperature
```

262. Trips and Users

Hard

The `Trips` table holds all taxi trips. Each trip has a unique Id, while Client_Id and Driver_Id are both foreign keys to the Users_Id at the `Users` table. Status is an ENUM type of ('completed', 'cancelled_by_driver', 'cancelled_by_client').

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

The `Users` table holds all users. Each user has an unique Users_Id, and Role is an ENUM type of ('client', 'driver', 'partner').

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

Write a SQL query to find the cancellation rate of requests made by unbanned users (both client and driver must be unbanned) between **Oct 1, 2013** and **Oct 3, 2013**. The cancellation rate is computed by dividing the number of canceled (by client or driver) requests made by unbanned users by the total number of requests made by unbanned users.

For the above tables, your SQL query should return the following rows with the cancellation rate being rounded to *two* decimal places.

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

Language: mysql

```
SELECT bb.request_at AS Day, ROUND(IFNULL((aa.canceled_count /
bb.total_count),0),2) AS 'Cancellation Rate'
FROM (SELECT b.request_at, COUNT(b.id) AS total_count
        FROM
            (SELECT id,client_id, driver_id, `status`, request_at
            FROM trips
            WHERE client_id IN (SELECT users_id
                                FROM users
                                WHERE banned = "No")
                AND driver_id IN (SELECT users_id
                                FROM users
                                WHERE banned = "No")
                AND request_at BETWEEN '2013-10-01' AND '2013-10-03'
            ) b
        GROUP BY b.request_at) bb
    LEFT JOIN
        (SELECT a.request_at, COUNT(a.id) AS canceled_count
        FROM
            (SELECT id,client_id, driver_id, `status`, request_at
            FROM trips
            WHERE client_id IN (SELECT users_id
                                FROM users
                                WHERE banned = "No")
                AND driver_id IN (SELECT users_id
                                FROM users
                                WHERE banned = "No")
                AND `status` != "completed"
                AND request_at BETWEEN '2013-10-01' AND '2013-10-03'
            ) a
        GROUP BY a.request_at) aa
    ON aa.request_at = bb.request_at
```

595. Big Countries

Easy

There is a table `World`

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

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

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

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

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

Language: mysql

```mysql
SELECT name, population, area
FROM World
WHERE area > 3000000 or population > 25000000;
```

596. Classes More Than 5 Students

Easy

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.

Language: mysql

```mysql
select class
from courses
group by class
having count(distinct student) >= 5;
```

601. Human Traffic of Stadium

Hard

X city built a new stadium, each day many people visit it and the stats are saved as these columns: **id**, **visit_date**, **people**

Please write a query to display the records which have 3 or more consecutive rows and the amount of people more than 100(inclusive).

For example, the table `stadium`:

```
+------+------------+-----------+
| 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    | 2017-01-08 | 188       |
+------+------------+-----------+
```

For the sample data above, the output is:

```
+------+------------+-----------+
| id   | visit_date | people    |
+------+------------+-----------+
| 5    | 2017-01-05 | 145       |
| 6    | 2017-01-06 | 1455      |
| 7    | 2017-01-07 | 199       |
| 8    | 2017-01-08 | 188       |
+------+------------+-----------+
```

Note: Each day only have one row record, and the dates are increasing with id increasing.

Language: mysql

```
SELECT DISTINCT t1.*
FROM stadium t1, stadium t2, stadium t3
WHERE t1.people>=100 AND t2.people>=100 AND t3.people>=100
    AND ((t2.id-t1.id=1 AND t3.id-t1.id=2) OR
         (t1.id-t2.id=1 AND t3.id-t1.id=1) OR
         (t3.id-t2.id=1 AND t1.id-t3.id=1)
        )
ORDER BY t1.id
```

Compared to **180. Consecutive Numbers**.

620. Not Boring Movies

Easy

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

Language: mysql

```
SELECT *
FROM cinema
WHERE id%2=1 AND description!='boring'
ORDER BY rating DESC
```

626. Exchange Seats

Medium

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?

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

Language: mysql

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

627. Swap Salary

Easy

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 statement** and no intermediate temp table.

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

Example:

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

After running your **update** statement, 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    |
```

Languege: mysql

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