DAY TO DAY MySQL Practice I

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.

SQL:

```
SELECT  p1.*
FROM  Person p1, Person p2
WHERE p1.Email = p2. Email AND p2.Id > p1.Id
```

MySQL:

```
DELETE p2  FROM Person p1, Person p2
WHERE p1.Email = p2.Email AND p2.Id > p1.Id
```

182. Duplicate Emails

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 |
+-----+
```

MySQL:

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

183. Customers Who Never Order

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

MySQL:

```
SELECT Name as 'Customers'
FROM Customers
WHERE Customers.Id  not in
( SELECT CustomerId
  FROM Orders)
```

595. Big Countries

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

MySQL:

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

176. Second Highest Salary

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

SQL Schema

```
Create table If Not Exists Employee (Id int, Salary int)
Truncate table Employee
insert into Employee (Id, Salary) values ('1', '100')
insert into Employee (Id, Salary) values ('2', '200')
insert into Employee (Id, Salary) values ('3', '300')
```

MySQL:

```
SELECT MAX(Salary) as SecondHighestSalary
FROM Employee
WHERE Salary not in
(SELECT MAX(Salary) FROM Employee)
```

175. Combine Two Tables

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`

SQL Schema

```
Create table Person (PersonId int, FirstName varchar(255), La
Create table Address (AddressId int, PersonId int, City varch
Truncate table Person
insert into Person (PersonId, LastName, FirstName) values ('1
Truncate table Address
insert into Address (AddressId, PersonId, City, State) values
```

MySQL:

```
SELECT FirstName, LastName, City, State
FROM Person left join Address
on Person.PersonId = Address.PersonId
```

1327. List the Products Ordered in a Period

able: `Products`

```
+------------------+---------+
| Column Name      | Type    |
+------------------+---------+
| product_id       | int     |
| product_name     | varchar |
| product_category | varchar |
+------------------+---------+
```

`product_id is the primary key for this table.`
`This table contains data about the company's products.`

Table: `Orders`

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| product_id    | int     |
| order_date    | date    |
| unit          | int     |
+---------------+---------+
```

`There is no primary key for this table. It may have duplicate rows.`
`product_id is a foreign key to Products table.`
`unit is the number of products ordered in order_date.`

Write an SQL query to get the names of products with greater than or equal to 100 units ordered in February 2020 and their amount.
Return result table in any order.
The query result format is in the following example:

`Products table:`

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

`Orders table:`

```
+--------------+--------------+----------+
| product_id   | order_date   | unit     |
+--------------+--------------+----------+
| 1            | 2020-02-05   | 60       |
| 1            | 2020-02-10   | 70       |
| 2            | 2020-01-18   | 30       |
| 2            | 2020-02-11   | 80       |
```

```
| 3            | 2020-02-17   | 2        |
| 3            | 2020-02-24   | 3        |
| 4            | 2020-03-01   | 20       |
| 4            | 2020-03-04   | 30       |
| 4            | 2020-03-04   | 60       |
| 5            | 2020-02-25   | 50       |
| 5            | 2020-02-27   | 50       |
| 5            | 2020-03-01   | 50       |
+--------------+--------------+----------+
```

```
Result table:
+--------------------+---------+
| product_name       | unit    |
+--------------------+---------+
| Leetcode Solutions | 130     |
| Leetcode Kit       | 100     |
+--------------------+---------+
```

MySQL:

```
SELECT p.product_name, sum(o.unit) AS unit
FROM Products p JOIN Orders o ON p.product_id = o.product_id
WHERE o.order_date >='2020-02-01' AND o.order_date <= '2020-02-29'
GROUP BY p.product_id
HAVING sum(o.unit) >= 100
```

178. 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       |
+-------+---------+
```

MySQL:
#descending order
SELECT score, dense_rank() over (order by score desc) 'Rank'
FROM Scores