182. Duplicate Emails

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| id | int |

| email | varchar |

+-------------+---------+

id is the primary key (column with unique values) for this table.

Each row of this table contains an email. The emails will not contain uppercase letters.

Write a solution to report all the duplicate emails. Note that it's guaranteed that the email field is not NULL.

Return the result table in **any order**.

Input:

Person table:

+----+---------+

| id | email |

+----+---------+

| 1 | a@b.com |

| 2 | c@d.com |

| 3 | a@b.com |

+----+---------+

Output:

+---------+

| Email |

+---------+

| a@b.com |

+---------+

Explanation: a@b.com is repeated two times.

Solution –

Write your MySQL query statement below

```
with duplicatecte as (
   select id, email, row_number() over(partition by email) as ranking
   from person
)
select distinct email from duplicatecte where ranking >1;
```

183 . Customers who never Order

Table: Customers

+----------+---------+

| Column Name | Type |

+----------+---------+

| id | int |

| name | varchar |

+----------+---------+

id is the primary key (column with unique values) for this table.

Each row of this table indicates the ID and name of a customer.

Table: Orders

+----------+------+

| Column Name | Type |

+----------+------+

| id | int |

| customerId | int |

+----------+------+

id is the primary key (column with unique values) for this table.

customerId is a foreign key (reference columns) of the ID from the Customers table.

Each row of this table indicates the ID of an order and the ID of the customer who ordered it.

Write a solution to find all customers who never order anything.

Return the result table in **any order**.

The result format is in the following example.

```
# Write your MySQL query statement below
with neverordercte as (
   select c.id as id, c.name as Customers, o.id as orderid
   from customers c left join orders o on c.id = o.customerid
)
select Customers from neverordercte where orderid is NULL;
```

586. Customer placing largest number of orders

Table: Orders

+-----------------+----------+

| Column Name | Type |

+-----------------+----------+

| order_number | int |

| customer_number | int |

+-----------------+----------+

order_number is the primary key (column with unique values) for this table.

This table contains information about the order ID and the customer ID.

Write a solution to find the customer_number for the customer who has placed **the largest number of orders**.

The test cases are generated so that **exactly one customer** will have placed more orders than any other customer.

The result format is in the following example.

Example 1:

Input:

Orders table:

+--------------+-----------------+

| order_number | customer_number |

+--------------+-----------------+

| 1 | 1 |

| 2 | 2 |

| 3 | 3 |

| 4 | 3 |

+--------------+-----------------+

Output:

+-----------------+

| customer_number |

+-----------------+

| 3 |

+-----------------+

Explanation:

The customer with number 3 has two orders, which is greater than either customer 1 or 2 because each of them only has one order.

So the result is customer_number 3.

Solution –

```
# Write your MySQL query statement below
with countcte as (
select customer_number as cust, count( customer_number) as customer_number
from orders
group by customer_number
order by customer_number desc)
select cust as customer_number from countcte limit 1 ;
```

176. Second Highest Salary

Table: Employee

+-------------+------+

| Column Name | Type |

+-------------+------+

| id | int |

| salary | int |

+-------------+------+

id is the primary key (column with unique values) for this table.

Each row of this table contains information about the salary of an employee.

Write a solution to find the second highest **distinct** salary from the Employee table. If there is no second highest salary, return null (return None in Pandas).

The result format is in the following example.

Example 1:

Input:

Employee table:

+----+--------+

| id | salary |

+----+--------+

| 1 | 100 |

| 2 | 200 |

| 3 | 300 |

+----+--------+

Output:

+---------------------+

| SecondHighestSalary |

+---------------------+

| 200 |

+---------------------+

Example 2:

Input:

Employee table:

+----+--------+

| id | salary |

+----+--------+

| 1 | 100 |

+----+--------+

Output:

+---------------------+

| SecondHighestSalary |

+---------------------+

| null |

+---------------------+

-- # Write your MySQL query statement below

-- with nullcte as (

-- select salary as SecondHighestSalary, row_number() over(order by salary desc) as rownum

-- from employee

-- order by salary desc)

-- select

-- case when rownum is null then 'null'

-- when rownum = 2 then SecondHighestSalary

-- else null

-- end as 'SecondHighestSalary'

-- from nullcte

-- limit 1 offset 1;

select

(select distinct Salary

from Employee order by salary desc

limit 1 offset 1)

as SecondHighestSalary;

178. Rank Scores

Table: Scores

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| id | int |

| score | decimal |

+-------------+---------+

id is the primary key (column with unique values) for this table.

Each row of this table contains the score of a game. Score is a floating point value with two decimal places.

Write a solution to find the rank of the scores. The ranking should be calculated according to the following rules:

- The scores should be ranked from the highest to the lowest.
- If there is a tie between two scores, both should have the same ranking.
- After a tie, the next ranking number should be the next consecutive integer value. In other words, there should be no holes between ranks.

Return the result table ordered by score in descending order.

The result format is in the following example.

Example 1:

Input:

Scores table:

+----+-------+

| id | score |

+----+-------+

| 1 | 3.50 |

| 2 | 3.65 |

| 3 | 4.00 |

| 4 | 3.85 |

| 5 | 4.00 |

| 6 | 3.65 |

+----+-------+

Output:

+-------+------+

| score | rank |

+-------+------+

| 4.00 | 1 |

| 4.00 | 1 |

| 3.85 | 2 |

| 3.65 | 3 |

| 3.65 | 3 |

| 3.50 | 4 |

+-------+------+

Solution –

```
select score, dense_rank() over(order by score desc) as 'rank'
from scores;
```