Exercise for SQL

1. Duplicates Emails

Table: **Person**

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| email       | varchar |
+-------------+---------+
id is the primary key column for this table.
Each row of this table contains an email. The emails will not contain
uppercase letters.
```

Write an SQL query to report all the duplicate emails.

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

The query result format is in the following example.

Example 1:

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.

2. Sales Person

Table: ***SalesPerson***

```
+-----------------+---------+
| Column Name     | Type    |
+-----------------+---------+
| sales_id        | int     |
| name            | varchar |
| salary          | int     |
| commission_rate | int     |
| hire_date       | date    |
+-----------------+---------+
```

sales_id is the primary key column for this table.
Each row of this table indicates the name and the ID of a salesperson alongside their salary, commission rate, and hire date.

Table: ***Company***

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| com_id      | int     |
| name        | varchar |
| city        | varchar |
+-------------+---------+
```

com_id is the primary key column for this table.
Each row of this table indicates the name and the ID of a company and the city in which the company is located.

Table: ***Orders***

```
+-------------+------+
| Column Name | Type |
+-------------+------+
| order_id    | int  |
| order_date  | date |
| com_id      | int  |
| sales_id    | int  |
| amount      | int  |
+-------------+------+
order_id is the primary key column for this table.
com_id is a foreign key to com_id from the Company table.
sales_id is a foreign key to sales_id from the SalesPerson table.
Each row of this table contains information about one order. This includes the ID of the company, the ID of the salesperson, the date of the order, and the amount paid.
```

Write an SQL query to report the names of all the salespersons who did not have any orders related to the company with the name **"RED"**.

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

The query result format is in the following example.

Example 1:

Input:

SalesPerson table:

sales_id	name	salary	commission_rate	hire_date
1	John	100000	6	4/1/2006
2	Amy	12000	5	5/1/2010
3	Mark	65000	12	12/25/2008
4	Pam	25000	25	1/1/2005
5	Alex	5000	10	2/3/2007

Company table:

com_id	name	city
1	RED	Boston
2	ORANGE	New York
3	YELLOW	Boston
4	GREEN	Austin

Orders table:

order_id	order_date	com_id	sales_id	amount
1	1/1/2014	3	4	10000
2	2/1/2014	4	5	5000
3	3/1/2014	1	1	50000
4	4/1/2014	1	4	25000

Output:

```
+------+
| name |
+------+
| Amy  |
| Mark |
| Alex |
+------+
```

Explanation:

According to orders 3 and 4 in the Orders table, it is easy to tell that only salesperson John and Pam have sales to company RED, so we report all the other names in the table salesperson.

3. Swap Salary

Table: `Salary`

```
+-------------+----------+
| 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 temporary tables.

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.

Example 1:

Input:
Salary table:
```
+----+------+-----+--------+
| id | name | sex | salary |
+----+------+-----+--------+
| 1  | A    | m   | 2500   |
| 2  | B    | f   | 1500   |
| 3  | C    | m   | 5500   |
| 4  | D    | f   | 500    |
+----+------+-----+--------+
```
Output:
```
+----+------+-----+--------+
| id | name | sex | salary |
+----+------+-----+--------+
| 1  | A    | f   | 2500   |
| 2  | B    | m   | 1500   |
| 3  | C    | f   | 5500   |
| 4  | D    | m   | 500    |
+----+------+-----+--------+
```

Explanation:
(1, A) and (3, C) were changed from 'm' to 'f'.
(2, B) and (4, D) were changed from 'f' to 'm'.

4. Customer Who Visited but Did Not Make Any Transactions

Table: `Visits`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| visit_id    | int     |
| customer_id | int     |
+-------------+---------+
visit_id is the primary key for this table.
This table contains information about the customers who visited the
mall.
```

Table: `Transactions`

```
+----------------+---------+
| Column Name    | Type    |
+----------------+---------+
| transaction_id | int     |
| visit_id       | int     |
| amount         | int     |
+----------------+---------+
transaction_id is the primary key for this table.
This table contains information about the transactions made during the
visit_id.
```

Write a SQL query to find the IDs of the users who visited without making any transactions and the number of times they made these types of visits.

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

The query result format is in the following example.

Example 1:

Input:

Visits

visit_id	customer_id
1	23
2	9
4	30
5	54
6	96
7	54
8	54

Transactions

transaction_id	visit_id	amount
2	5	310
3	5	300
9	5	200
12	1	910
13	2	970

Output:

customer_id	count_no_trans
54	2
30	1
96	1

Explanation:

Customer with id = 23 visited the mall once and made one transaction during the visit with id = 12.
Customer with id = 9 visited the mall once and made one transaction during the visit with id = 13.
Customer with id = 30 visited the mall once and did not make any transactions.
Customer with id = 54 visited the mall three times. During 2 visits they did not make any transactions, and during one visit they made 3 transactions.
Customer with id = 96 visited the mall once and did not make any transactions.
As we can see, users with IDs 30 and 96 visited the mall one time without making any transactions. Also, user 54 visited the mall twice and did not make any transactions.

5. Patients With a Condition

Table: `Patients`

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| patient_id   | int     |
| patient_name | varchar |
| conditions   | varchar |
+--------------+---------+
patient_id is the primary key for this table.
'conditions' contains 0 or more code separated by spaces.
This table contains information of the patients in the hospital.
```

Write an SQL query to report the patient_id, patient_name and conditions of the patients who have Type I Diabetes. Type I Diabetes always starts with `DIAB1` prefix.

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

The query result format is in the following example.

Example 1:

Input:
Patients table:

```
+------------+--------------+--------------+
| patient_id | patient_name | conditions   |
+------------+--------------+--------------+
| 1          | Daniel       | YFEV COUGH   |
| 2          | Alice        |              |
| 3          | Bob          | DIAB100 MYOP |
| 4          | George       | ACNE DIAB100 |
| 5          | Alain        | DIAB201      |
+------------+--------------+--------------+
```

Output:

```
+------------+--------------+--------------+
| patient_id | patient_name | conditions   |
+------------+--------------+--------------+
| 3          | Bob          | DIAB100 MYOP |
| 4          | George       | ACNE DIAB100 |
+------------+--------------+--------------+
```

Explanation: Bob and George both have a condition that starts with DIAB1.

6. Bank Account Summary II

Table: `Users`

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| account      | int     |
| name         | varchar |
+--------------+---------+
account is the primary key for this table.
Each row of this table contains the account number of each user in the
bank.
There will be no two users having the same name in the table.
```

Table: `Transactions`

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| trans_id      | int     |
| account       | int     |
| amount        | int     |
| transacted_on | date    |
+---------------+---------+
trans_id is the primary key for this table.
Each row of this table contains all changes made to all accounts.
amount is positive if the user received money and negative if they
transferred money.
All accounts start with a balance of 0.
```

Write an SQL query to report the name and balance of users with a balance higher than `10000`. The balance of an account is equal to the sum of the amounts of all transactions involving that account.

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

The query result format is in the following example.

Example 1:

Input:
Users table:

account	name
900001	Alice
900002	Bob
900003	Charlie

Transactions table:

trans_id	account	amount	transacted_on
1	900001	7000	2020-08-01
2	900001	7000	2020-09-01
3	900001	-3000	2020-09-02
4	900002	1000	2020-09-12
5	900003	6000	2020-08-07
6	900003	6000	2020-09-07
7	900003	-4000	2020-09-11

Output:

name	balance
Alice	11000

Explanation:
Alice's balance is (7000 + 7000 - 3000) = 11000.
Bob's balance is 1000.
Charlie's balance is (6000 + 6000 - 4000) = 8000.

7. Employees With Missing Information

Table: `Employees`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| employee_id | int     |
| name        | varchar |
+-------------+---------+
employee_id is the primary key for this table.
Each row of this table indicates the name of the employee whose ID is
employee_id.
```

Table: `Salaries`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| employee_id | int     |
| salary      | int     |
+-------------+---------+
employee_id is the primary key for this table.
Each row of this table indicates the salary of the employee whose ID is
employee_id.
```

Write an SQL query to report the IDs of all the employees with **missing information**. The information of an employee is missing if:

- The employee's **name** is missing, or
- The employee's **salary** is missing.

Return the result table ordered by `employee_id` **in ascending order**.

The query result format is in the following example.

Example 1:

Input:

Employees table:

employee_id	name
2	Crew
4	Haven
5	Kristian

Salaries table:

employee_id	salary
5	76071
1	22517
4	63539

Output:

employee_id
1
2

Explanation:

Employees 1, 2, 4, and 5 are working at this company.
The name of employee 1 is missing.
The salary of employee 2 is missing.

8. Calculate Special Bonus

Table: `Employees`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| employee_id | int     |
| name        | varchar |
| salary      | int     |
+-------------+---------+
employee_id is the primary key for this table.
Each row of this table indicates the employee ID, employee name, and
salary.
```

Write an SQL query to calculate the bonus of each employee. The bonus of an employee is `100%` of their salary if the ID of the employee is **an odd number** and **the employee name does not start with the character** `'M'`. The bonus of an employee is `0` otherwise.

Return the result table ordered by `employee_id`.

The query result format is in the following example.

Example 1:

Input:
Employees table:

```
+-------------+---------+--------+
| employee_id | name    | salary |
+-------------+---------+--------+
| 2           | Meir    | 3000   |
| 3           | Michael | 3800   |
| 7           | Addilyn | 7400   |
| 8           | Juan    | 6100   |
| 9           | Kannon  | 7700   |
+-------------+---------+--------+
```

Output:

```
+-------------+-------+
| employee_id | bonus |
+-------------+-------+
| 2           | 0     |
| 3           | 0     |
| 7           | 7400  |
| 8           | 0     |
| 9           | 7700  |
+-------------+-------+
```

Explanation:
The employees with IDs 2 and 8 get 0 bonus because they have an even employee_id.
The employee with ID 3 gets 0 bonus because their name starts with 'M'.
The rest of the employees get a 100% bonus.

9. Group Sold Products By The Date

Table `Activities`:

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| sell_date   | date    |
| product     | varchar |
+-------------+---------+
There is no primary key for this table, it may contain duplicates.
Each row of this table contains the product name and the date it was
sold in a market.
```

Write an SQL query to find for each date the number of different products sold and their names.

The sold products names for each date should be sorted lexicographically.

Return the result table ordered by `sell_date`.

The query result format is in the following example.

Example 1:

Input:
Activities table:

sell_date	product
2020-05-30	Headphone
2020-06-01	Pencil
2020-06-02	Mask
2020-05-30	Basketball
2020-06-01	Bible
2020-06-02	Mask
2020-05-30	T-Shirt

Output:

sell_date	num_sold	products
2020-05-30	3	Basketball,Headphone,T-shirt
2020-06-01	2	Bible,Pencil
2020-06-02	1	Mask

Explanation:
For 2020-05-30, Sold items were (Headphone, Basketball, T-shirt), we sort them lexicographically and separate them by a comma.
For 2020-06-01, Sold items were (Pencil, Bible), we sort them lexicographically and separate them by a comma.
For 2020-06-02, the Sold item is (Mask), we just return it.

10. Sales Analysis III

Table: `Product`

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| product_id   | int     |
| product_name | varchar |
| unit_price   | int     |
+--------------+---------+
product_id is the primary key of this table.
Each row of this table indicates the name and the price of each
product.
```

Table: `Sales`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| seller_id   | int     |
| product_id  | int     |
| buyer_id    | int     |
| sale_date   | date    |
| quantity    | int     |
| price       | int     |
+-------------+---------+
This table has no primary key, it can have repeated rows.
product_id is a foreign key to the Product table.
Each row of this table contains some information about one sale.
```

Write an SQL query that reports the **products** that were **only** sold in the first quarter of `2019`. That is, between `2019-01-01` and `2019-03-31` inclusive.

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

The query result format is in the following example.

Example 1:

Input:
Product table:

product_id	product_name	unit_price
1	S8	1000
2	G4	800
3	iPhone	1400

Sales table:

seller_id	product_id	buyer_id	sale_date	quantity	price
1	1	1	2019-01-21	2	2000
1	2	2	2019-02-17	1	800
2	2	3	2019-06-02	1	800
3	3	4	2019-05-13	2	2800

Output:

product_id	product_name
1	S8

Explanation:
The product with id 1 was only sold in the spring of 2019.
The product with id 2 was sold in the spring of 2019 but was also sold after the spring of 2019.
The product with id 3 was sold after spring 2019.
We return only product 1 as it is the product that was only sold in the spring of 2019.