05/15/2022

1581. 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 an 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**.

MySQL

```
SELECT customer_id, COUNT(v.visit_id) as count_no_trans

FROM Visits v

LEFT JOIN Transactions t

ON v.visit_id = t.visit_id

WHERE t.visit_id IS NULL

GROUP BY customer_id
```

197. Rising Temperature

Table: `Weather`

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

| Column Name   | Type    |

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

| id            | int     |

| recordDate    | date    |

| temperature   | int     |

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

id is the primary key for this table.

This table contains information about the temperature on a certain day.
```

Write an SQL query to find all dates' `Id` with higher temperatures compared to its previous dates (yesterday).

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

MySQL

```
SELECT w1.id AS Id

FROM Weather w1, Weather w2

WHERE DATEDIFF(w1.recordDate, w2.recordDate) = 1

AND w1.temperature > w2.temperature;
```

05/16/2022

607. 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 com_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.
```

MySQL

```
SELECT SalesPerson.name

FROM Orders o JOIN Company c ON (o.com_id = c.com_id and c.name = 'RED')

RIGHT JOIN SalesPerson ON SalesPerson.sales_id = o.sales_id

WHERE o.sales_id IS NULL
```

Solution #2

```
SELECT SalesPerson.name

FROM SalesPerson

WHERE SalesPerson.sales_id NOT IN(

SELECT Orders.sales_id

FROM Orders

LEFT JOIN Company ON Orders.com_id=Company.com_id

WHERE Company.name = 'RED');
```

1141. User Activity for the Past 30 Days I

Table: `Activity`

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

| Column Name   | Type    |

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

| user_id       | int     |
| session_id    | int     |
| activity_date | date    |
| activity_type | enum    |
+---------------+---------+

There is no primary key for this table, it may have duplicate rows.

The activity_type column is an ENUM of type ('open_session', 'end_session',
'scroll_down', 'send_message').

The table shows the user activities for a social media website.

Note that each session belongs to exactly one user.
```

Write an SQL query to find the daily active user count for a period of `30` days ending `2019-07-27` inclusively. A user was active on someday if they made at least one activity on that day.

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

The query result format is in the following example.

Example 1:

```
Input:
Activity table:
+---------+------------+---------------+---------------+
| user_id | session_id | activity_date | activity_type |
+---------+------------+---------------+---------------+
| 1       | 1          | 2019-07-20    | open_session  |
| 1       | 1          | 2019-07-20    | scroll_down   |
| 1       | 1          | 2019-07-20    | end_session   |
| 2       | 4          | 2019-07-20    | open_session  |
| 2       | 4          | 2019-07-21    | send_message  |
| 2       | 4          | 2019-07-21    | end_session   |
| 3       | 2          | 2019-07-21    | open_session  |
| 3       | 2          | 2019-07-21    | send_message  |
| 3       | 2          | 2019-07-21    | end_session   |
| 4       | 3          | 2019-06-25    | open_session  |
| 4       | 3          | 2019-06-25    | end_session   |
+---------+------------+---------------+---------------+
Output:
+------------+-------------+
| day        | active_users |
+------------+-------------+
| 2019-07-20 | 2            |
| 2019-07-21 | 2            |
+------------+-------------+
Explanation: Note that we do not care about days with zero active users.
```

MySQL

```
SELECT activity_date as day, COUNT(DISTINCT user_id) as active_users
FROM Activity
WHERE DATEDIFF('2019-07-27', activity_date) BETWEEN 0 AND 29
GROUP BY activity_date;
```

Solution #2

```
SELECT activity_date as day, COUNT(DISTINCT user_id) as active_users
FROM Activity
WHERE (activity_date BETWEEN '2019-06-28' AND '2019-07-27')
GROUP BY activity_date;
```

1729. Find Followers Count

Table: `Followers`

```
+------------+------+
| Column Name | Type |

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

| user_id     | int  |

| follower_id | int  |

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

(user_id, follower_id) is the primary key for this table.

This table contains the IDs of a user and a follower in a social media app where the
follower follows the user.
```

Write an SQL query that will, for each user, return the number of followers.

Return the result table ordered by `user_id`.

The query result format is in the following example.

Example 1:

```
Input:

Followers table:

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

| user_id | follower_id |

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

| 0       | 1           |

| 1       | 0           |

| 2       | 0           |

| 2       | 1           |

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

Output:

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

| user_id | followers_count|
+---------+----------------+
| 0       | 1              |
| 1       | 1              |
| 2       | 2              |
+---------+----------------+
Explanation:
The followers of 0 are {1}
The followers of 1 are {0}
The followers of 2 are {0,1}
```

MySQL

```
SELECT user_id, COUNT(follower_id) as followers_count
FROM Followers
GROUP BY user_id
ORDER BY user_id;
```

1693. Daily Leads and Partners

Table: `DailySales`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| date_id     | date    |
| make_name   | varchar |
| lead_id     | int     |
| partner_id  | int     |
+-------------+---------+
This table does not have a primary key.
This table contains the date and the name of the product sold and the IDs of the lead
and partner it was sold to.

The name consists of only lowercase English letters.
```

Write an SQL query that will, for each `date_id` and `make_name`, return the number of **distinct** `lead_id`'s and **distinct** `partner_id`'s.

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

The query result format is in the following example.

Example 1:

```
Input:

DailySales table:

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

| date_id   | make_name | lead_id | partner_id |

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

| 2020-12-8 | toyota    | 0       | 1          |

| 2020-12-8 | toyota    | 1       | 0          |

| 2020-12-8 | toyota    | 1       | 2          |

| 2020-12-7 | toyota    | 0       | 2          |

| 2020-12-7 | toyota    | 0       | 1          |

| 2020-12-8 | honda     | 1       | 2          |

| 2020-12-8 | honda     | 2       | 1          |

| 2020-12-7 | honda     | 0       | 1          |

| 2020-12-7 | honda     | 1       | 2          |

| 2020-12-7 | honda     | 2       | 1          |

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

Output:

+-----------+-----------+-------------+-----------------+
| date_id   | make_name | unique_leads | unique_partners |
+-----------+-----------+--------------+-----------------+
| 2020-12-8 | toyota    | 2            | 3               |
| 2020-12-7 | toyota    | 1            | 2               |
| 2020-12-8 | honda     | 2            | 2               |
| 2020-12-7 | honda     | 3            | 2               |
+-----------+-----------+--------------+-----------------+
```

Explanation:

```
For 2020-12-8, toyota gets leads = [0, 1] and partners = [0, 1, 2] while honda gets
leads = [1, 2] and partners = [1, 2].

For 2020-12-7, toyota gets leads = [0] and partners = [1, 2] while honda gets leads =
[0, 1, 2] and partners = [1, 2].
```

MySQL

```
SELECT date_id, make_name, COUNT(DISTINCT lead_id) AS unique_leads, COUNT(DISTINCT partner_id)
AS unique_partners

FROM DailySales

GROUP BY date_id, make_name
```

586. Customer Placing the Largest Number of Orders

Table: `Orders`

```
+-----------------+----------+
| Column Name     | Type     |
+-----------------+----------+
| order_number    | int      |
| customer_number | int      |
+-----------------+----------+
order_number is the primary key for this table.
This table contains information about the order ID and the customer ID.
```

Write an SQL query 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 query 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.
```

MySQL

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

511. Game Play Analysis I

Table: `Activity`

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| player_id    | int     |
| device_id    | int     |
| event_date   | date    |
| games_played | int     |
+--------------+---------+
(player_id, event_date) is the primary key of this table.
This table shows the activity of players of some games.
Each row is a record of a player who logged in and played a number of games (possibly 0) before logging out on someday using some device.
```

Write an SQL query to report the **first login date** for each player.

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

The query result format is in the following example.

Example 1:

```
Input:
Activity table:
```

```
+-----------+-----------+------------+--------------+
| player_id | device_id | event_date | games_played |
+-----------+-----------+------------+--------------+
| 1         | 2         | 2016-03-01 | 5            |
| 1         | 2         | 2016-05-02 | 6            |
| 2         | 3         | 2017-06-25 | 1            |
| 3         | 1         | 2016-03-02 | 0            |
| 3         | 4         | 2018-07-03 | 5            |
+-----------+-----------+------------+--------------+
Output:
+-----------+-------------+
| player_id | first_login |
+-----------+-------------+
| 1         | 2016-03-01  |
| 2         | 2017-06-25  |
| 3         | 2016-03-02  |
+-----------+-------------+
```

MySQL

```
SELECT player_id, MIN(event_date) as first_login
FROM Activity
GROUP BY player_id
```

1890. The Latest Login in 2020

Table: `Logins`

```
+----------------+----------+
| Column Name    | Type     |
+----------------+----------+
| user_id        | int      |
| time_stamp     | datetime |
+----------------+----------+
(user_id, time_stamp) is the primary key for this table.
Each row contains information about the login time for the user with ID user_id.
```

Write an SQL query to report the **latest** login for all users in the year `2020`. Do **not** include the users who did not login in `2020`.

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

The query result format is in the following example.

Example 1:

```
Input:
Logins table:
+---------+---------------------+
| user_id | time_stamp          |
+---------+---------------------+
| 6       | 2020-06-30 15:06:07 |
| 6       | 2021-04-21 14:06:06 |
| 6       | 2019-03-07 00:18:15 |
| 8       | 2020-02-01 05:10:53 |
| 8       | 2020-12-30 00:46:50 |
| 2       | 2020-01-16 02:49:50 |
| 2       | 2019-08-25 07:59:08 |
| 14      | 2019-07-14 09:00:00 |
| 14      | 2021-01-06 11:59:59 |
+---------+---------------------+
Output:
```

```
+---------+---------------------+
| user_id | last_stamp          |
+---------+---------------------+
| 6       | 2020-06-30 15:06:07 |
| 8       | 2020-12-30 00:46:50 |
| 2       | 2020-01-16 02:49:50 |
+---------+---------------------+
```

Explanation:

```
User 6 logged into their account 3 times but only once in 2020, so we include this
login in the result table.

User 8 logged into their account 2 times in 2020, once in February and once in
December. We include only the latest one (December) in the result table.

User 2 logged into their account 2 times but only once in 2020, so we include this
login in the result table.

User 14 did not login in 2020, so we do not include them in the result table.
```

MySQL

```
SELECT user_id, MAX(time_stamp) as last_stamp
FROM Logins
WHERE YEAR(time_stamp) = 2020
GROUP BY user_id
```

5/19/2022

1407. Top Travellers

Table: `Users`

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| id            | int     |
| name          | varchar |
+---------------+---------+
id is the primary key for this table.
name is the name of the user.
```

Table: `Rides`

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| id            | int     |
| user_id       | int     |
| distance      | int     |
+---------------+---------+
id is the primary key for this table.
user_id is the id of the user who traveled the distance "distance".
```

Write an SQL query to report the distance traveled by each user.

Return the result table ordered by `travelled_distance` in **descending order**, if two or more users traveled the same distance, order them by their `name` in **ascending order**.

The query result format is in the following example.

Example 1:

```
Input:
Users table:
+------+-----------+
| id   | name      |
+------+-----------+
| 1    | Alice     |
| 2    | Bob       |
| 3    | Alex      |
| 4    | Donald    |
| 7    | Lee       |
| 13   | Jonathan  |
| 19   | Elvis     |
+------+-----------+
Rides table:
+------+----------+----------+
| id   | user_id  | distance |
+------+----------+----------+
| 1    | 1        | 120      |
| 2    | 2        | 317      |
| 3    | 3        | 222      |
| 4    | 7        | 100      |
| 5    | 13       | 312      |
| 6    | 19       | 50       |
| 7    | 7        | 120      |
| 8    | 19       | 400      |
| 9    | 7        | 230      |
+------+----------+----------+
Output:
+----------+--------------------+
| name     | travelled_distance |
+----------+--------------------+
| Elvis    | 450                |
| Lee      | 450                |

| Bob      | 317                |

| Jonathan | 312                |

| Alex     | 222                |

| Alice    | 120                |

| Donald   | 0                  |

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

Explanation:

Elvis and Lee traveled 450 miles, Elvis is the top traveler as his name is
alphabetically smaller than Lee.

Bob, Jonathan, Alex, and Alice have only one ride and we just order them by the total
distances of the ride.

Donald did not have any rides, the distance traveled by him is 0.
```

MySQL

```
SELECT u.name, ifnull(SUM(r.distance),0) AS travelled_distance

FROM Users u

LEFT JOIN Rides r

ON u.id=r.user_id

GROUP BY r.user_id

ORDER BY travelled_distance DESC, name ASC
```

182. Duplicate 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.
```

MySQL

```
SELECT email
FROM Person
GROUP BY email
HAVING COUNT(email) >= 2
```

1050. Actors and Directors Who Cooperated At Least Three Times

Table: `ActorDirector`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| actor_id    | int     |
| director_id | int     |
| timestamp   | int     |
+-------------+---------+
timestamp is the primary key column for this table.
```

Write a SQL query for a report that provides the pairs `(actor_id, director_id)` where the actor has cooperated with the director at least three times.

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

The query result format is in the following example.

Example 1:

```
Input:
ActorDirector table:
+-------------+-------------+-------------+
| actor_id    | director_id | timestamp   |
+-------------+-------------+-------------+
| 1           | 1           | 0           |
| 1           | 1           | 1           |
| 1           | 1           | 2           |
| 1           | 2           | 3           |
| 1           | 2           | 4           |
| 2           | 1           | 5           |
| 2           | 1           | 6           |
+-------------+-------------+-------------+
Output:
+-------------+-------------+
| actor_id    | director_id |
+-------------+-------------+
| 1           | 1           |
+-------------+-------------+
Explanation: The only pair is (1, 1) where they cooperated exactly 3 times.
```

MySQL

```
SELECT actor_id, director_id
FROM ActorDirector
GROUP BY actor_id, director_id
HAVING COUNT(1) >= 3
```

1587. 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.
```

MySQL

```
SELECT u.name, SUM(t.amount) as balance
FROM Users u
LEFT JOIN Transactions t
ON u.account=t.account
GROUP BY u.name
HAVING balance > 10000
```

1084. 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.
```

MySQL

```
SELECT p.product_id, p.product_name
FROM Product p
LEFT JOIN Sales s
ON p.product_id=s.product_id
GROUP BY p.product_id
HAVING (MIN(sale_date) >= CAST('2019-01-01' AS DATE)
AND MAX(sale_date) <= CAST('2019-03-31' AS DATE));
```