**619. Problem**

Table **number** contains many numbers in column num including duplicated ones.  
Can you write a SQL query to find the biggest number, which only appears once.

<https://nifannn.github.io/tags/#SQL>A screenshot of a cell phone

Description automatically generated

### 597. Problem

In social network like Facebook or Twitter, people send friend requests and accept others’ requests as well. Now given two tables as below:

Table: **friend\_request**

| **sender\_id** | **send\_to\_id** | **request\_date** |
| --- | --- | --- |
| 1 | 2 | 2016\_06-01 |
| 1 | 3 | 2016\_06-01 |
| 1 | 4 | 2016\_06-01 |
| 2 | 3 | 2016\_06-02 |
| 3 | 4 | 2016-06-09 |

Table: **request\_accepted**

| **requester\_id** | **accepter\_id** | **accept\_date** |
| --- | --- | --- |
| 1 | 2 | 2016\_06-03 |
| 1 | 3 | 2016-06-08 |
| 2 | 3 | 2016-06-08 |
| 3 | 4 | 2016-06-09 |
| 3 | 4 | 2016-06-10 |

Write a query to find the overall acceptance rate of requests rounded to 2 decimals, which is the number of acceptance divide the number of requests.  
For the sample data above, your query should return the following result.

| **accept\_rate** |
| --- |
| 0.80 |

**Note:**

* The accepted requests are not necessarily from the table **friend\_request**. In this case, you just need to simply count the total accepted requests (no matter whether they are in the original requests), and divide it by the number of requests to get the acceptance rate.
* It is possible that a sender sends multiple requests to the same receiver, and a request could be accepted more than once. In this case, the ‘duplicated’ requests or acceptances are only counted once.
* If there is no requests at all, you should return 0.00 as the accept\_rate.

**Explanation:** There are 4 unique accepted requests, and there are 5 requests in total. So the rate is 0.80.

### 607. Problem

**Description**

Given three tables: **salesperson**, **company**, **orders**.  
Output all the names in the table **salesperson**, who didn’t have sales to company ‘RED’.

**Example**  
**Input**

Table: **salesperson**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | +----------+------+--------+-----------------+-----------+  | sales\_id | name | salary | commission\_rate | hire\_date |  +----------+------+--------+-----------------+-----------+  | 1 | John | 100000 | 6 | 4/1/2006 |  | 2 | Amy | 120000 | 5 | 5/1/2010 |  | 3 | Mark | 65000 | 12 | 12/25/2008|  | 4 | Pam | 25000 | 25 | 1/1/2005 |  | 5 | Alex | 50000 | 10 | 2/3/2007 |  +----------+------+--------+-----------------+-----------+ |

The table **salesperson** holds the salesperson information. Every salesperson has a sales\_id and a name.  
Table: **company**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | +---------+--------+------------+  | com\_id | name | city |  +---------+--------+------------+  | 1 | RED | Boston |  | 2 | ORANGE | New York |  | 3 | YELLOW | Boston |  | 4 | GREEN | Austin |  +---------+--------+------------+ |

The table **company** holds the company information. Every company has a com\_id and a name.  
Table: **orders**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | +----------+----------+---------+----------+--------+  | order\_id | date | com\_id | sales\_id | amount |  +----------+----------+---------+----------+--------+  | 1 | 1/1/2014 | 3 | 4 | 100000 |  | 2 | 2/1/2014 | 4 | 5 | 5000 |  | 3 | 3/1/2014 | 1 | 1 | 50000 |  | 4 | 4/1/2014 | 1 | 4 | 25000 |  +----------+----------+---------+----------+--------+ |

The table **orders** holds the sales record information, salesperson and customer company are represented by sales\_id and com\_id.  
**output**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | +------+  | name |  +------+  | Amy |  | Mark |  | Alex |  +------+ |

**Explanation**

According to order ‘3’ and ‘4’ in table **orders**, it is easy to tell only salesperson ‘John’ and ‘Alex’ have sales to company ‘RED’,  
so we need to output all the other names in table **salesperson**.

**603. Problem**

Several friends at a cinema ticket office would like to reserve consecutive available seats.  
Can you help to query all the consecutive available seats order by the seat\_id using the following **cinema** table?

| **seat\_id** | **free** |
| --- | --- |
| 1 | 1 |
| 2 | 0 |
| 3 | 1 |
| 4 | 1 |
| 5 | 1 |

Your query should return the following result for the sample case above.

| **seat\_id** |
| --- |
| 3 |
| 4 |
| 5 |

**Note:**

* The seat\_id is an auto increment int, and free is bool (‘1’ means free, and ‘0’ means occupied.).
* Consecutive available seats are more than 2(inclusive) seats consecutively available.

### 577. Problem

Select all employee’s name and bonus whose bonus is < 1000.

Table:**Employee**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | +-------+--------+-----------+--------+  | empId | name | supervisor| salary |  +-------+--------+-----------+--------+  | 1 | John | 3 | 1000 |  | 2 | Dan | 3 | 2000 |  | 3 | Brad | null | 4000 |  | 4 | Thomas | 3 | 4000 |  +-------+--------+-----------+--------+  empId is the primary key column for this table. |

Table: **Bonus**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | +-------+-------+  | empId | bonus |  +-------+-------+  | 2 | 500 |  | 4 | 2000 |  +-------+-------+  empId is the primary key column for this table. |

Example ouput:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | +-------+-------+  | name | bonus |  +-------+-------+  | John | null |  | Dan | 500 |  | Brad | null |  +-------+-------+ |

### 610. Problem

A pupil Tim gets homework to identify whether three line segments could possibly form a triangle.

However, this assignment is very heavy because there are hundreds of records to calculate.  
Could you help Tim by writing a query to judge whether these three sides can form a triangle, assuming table **triangle** holds the length of the three sides x, y and z.

| **x** | **y** | **z** |
| --- | --- | --- |
| 13 | 15 | 30 |
| 10 | 20 | 15 |

For the sample data above, your query should return the follow result:

| **x** | **y** | **z** | **triangle** |
| --- | --- | --- | --- |
| 13 | 15 | 30 | No |
| 10 | 20 | 15 | Yes |

### 586. Problem

Query the customer\_number from the **orders** table for the customer who has placed the largest number of orders.

It is guaranteed that exactly one customer will have placed more orders than any other customer.

The **orders** table is defined as follows:

| **Column** | **Type** |
| --- | --- |
| order\_number (PK) | int |
| customer\_number | int |
| order\_date | date |
| required\_date | date |
| shipped\_date | date |
| status | char(15) |
| comment | char(200) |

**Sample Input**

| **order\_number** | **customer\_number** | **order\_date** | **required\_date** | **shipped\_date** | **status** | **comment** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 2017-04-09 | 2017-04-13 | 2017-04-12 | Closed |  |
| 2 | 2 | 2017-04-15 | 2017-04-20 | 2017-04-18 | Closed |  |
| 3 | 3 | 2017-04-16 | 2017-04-25 | 2017-04-20 | Closed |  |
| 4 | 3 | 2017-04-18 | 2017-04-28 | 2017-04-25 | Closed |  |

**Sample 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’.

### 584. Problem

Given a table **customer** holding customers information and the referee.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | +------+------+-----------+  | id | name | referee\_id|  +------+------+-----------+  | 1 | Will | NULL |  | 2 | Jane | NULL |  | 3 | Alex | 2 |  | 4 | Bill | NULL |  | 5 | Zack | 1 |  | 6 | Mark | 2 |  +------+------+-----------+ |

Write a query to return the list of customers **NOT** referred by the person with id ‘2’.

For the sample data above, the result is:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | +------+  | name |  +------+  | Will |  | Jane |  | Bill |  | Zack |  +------+ |

### 613. Problem

Table **point** holds the x coordinate of some points on x-axis in a plane, which are all integers.

Write a query to find the shortest distance between two points in these points.

| **x** |
| --- |
| -1 |
| 0 |
| 2 |

The shortest distance is ‘1’ obviously, which is from point ‘-1’ to ‘0’. So the output is as below:

| **shortest** |
| --- |
| 1 |

**Note:** Every point is unique, which means there is no duplicates in table **point**.