Reference : <https://nifannn.github.io/tags/#SQL>

### 614. Problem

In facebook, there is a **follow** table with two columns: *followee*, *follower*.

Please write a sql query to get the amount of each follower’s follower if he/she has one.

For example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | +-------------+------------+  | followee | follower |  +-------------+------------+  | A | B |  | B | C |  | B | D |  | D | E |  +-------------+------------+ |

should output:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | +-------------+------------+  | follower | num |  +-------------+------------+  | B | 2 |  | D | 1 |  +-------------+------------+ |

**Explaination:**  
Both B and D exist in the follower list, when as a followee, B’s follower is C and D, and D’s follower is E. A does not exist in follower list.  
**Note:**  
Followee would not follow himself/herself in all cases.  
Please display the result in follower’s alphabet order.

### 578. Problem

Get the highest answer rate question from a table **survey\_log** with these columns: **uid, action, question\_id, answer\_id, q\_num, timestamp**.

uid means user id; action has these kind of values: “show”, “answer”, “skip”; answer\_id is not null when action column is “answer”, while is null for “show” and “skip”; q\_num is the numeral order of the question in current session.

Write a sql query to identify the question which has the highest answer rate.

**Example:**

**Input**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | +------+-----------+--------------+------------+-----------+------------+  | uid | action | question\_id | answer\_id | q\_num | timestamp |  +------+-----------+--------------+------------+-----------+------------+  | 5 | show | 285 | null | 1 | 123 |  | 5 | answer | 285 | 124124 | 1 | 124 |  | 5 | show | 369 | null | 2 | 125 |  | 5 | skip | 369 | null | 2 | 126 |  +------+-----------+--------------+------------+-----------+------------+ |

**Output**

|  |  |
| --- | --- |
| 1  2  3  4  5 | +-------------+  | survey\_log |  +-------------+  | 285 |  +-------------+ |

**Explanation**

question 285 has answer rate 1/1, while question 369 has 0/1 answer rate, so output 285.

**Note:** The highest answer rate meaning is: answer number’s ratio in show number in the same question.

### 574. Problem

Table: **Candidate**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | +-----+---------+  | id | Name |  +-----+---------+  | 1 | A |  | 2 | B |  | 3 | C |  | 4 | D |  | 5 | E |  +-----+---------+ |

Table: **Vote**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | +-----+--------------+  | id | CandidateId |  +-----+--------------+  | 1 | 2 |  | 2 | 4 |  | 3 | 3 |  | 4 | 2 |  | 5 | 5 |  +-----+--------------+  id is the auto-increment primary key,  CandidateId is the id appeared in Candidate table. |

Write a sql to find the name of the winning candidate, the above example will return the winner B.

|  |  |
| --- | --- |
| 1  2  3  4  5 | +------+  | Name |  +------+  | B |  +------+ |

**Notes:**  
You may assume **there is no tie**, in other words there will be **at most one** winning candidate.

### 580. Problem

A university uses 2 data tables, *student* and *department*, to store data about its students and the departments associated with each major.

Write a query to print the respective department name and number of students majoring in each department for all departments in the *department* table (even ones with no current students).

Sort your results by descending number of students; if two or more departments have the same number of students, then sort those departments alphabetically by department name.

The *student* is described as follow:

| **Column Name** | **Type** |
| --- | --- |
| student\_id | Integer |
| student\_name | String |
| gender | Character |
| dept\_id | Integer |

where student\_id is the student’s ID number, student\_name is the student’s name, gender is their gender, and dept\_id is the department ID associated with their declared major.

And the *department* table is described as below:

| **Column Name** | **Type** |
| --- | --- |
| dept\_id | Integer |
| dept\_name | String |

where dept\_id is the department’s ID number and dept\_name is the department name.

Here is an example **input**:  
*student table*:

| **student\_id** | **student\_name** | **gender** | **dept\_id** |
| --- | --- | --- | --- |
| 1 | Jack | M | 1 |
| 2 | Jane | F | 1 |
| 3 | Mark | M | 2 |

*department* table:

| **dept\_id** | **dept\_name** |
| --- | --- |
| 1 | Engineering |
| 2 | Science |
| 3 | Law |

The **Output** should be:

| **dept\_name** | **student\_number** |
| --- | --- |
| Engineering | 2 |
| Science | 1 |
| Law | 0 |

### 602. Problem

In social network like Facebook or Twitter, people send friend requests and accept others’ requests as well.

Table **request\_accepted** holds the data of friend acceptance, while requester\_id and accepter\_idboth are the id of a person.

| **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 |

Write a query to find the the people who has most friends and the most friends number. For the sample data above, the result is:

| **id** | **num** |
| --- | --- |
| 3 | 3 |

**Note:**

* It is guaranteed there is only 1 people having the most friends.
* The friend request could only been accepted once, which mean there is no multiple records with the same requester\_id and accepter\_id value.

**Explanation:**  
The person with id ‘3’ is a friend of people ‘1’, ‘2’ and ‘4’, so he has 3 friends in total, which is the most number than any others.

### 585. Problem

Write a query to print the sum of all total investment values in 2016 (**TIV\_2016**), to a scale of 2 decimal places, for all policy holders who meet the following criteria:

1. Have the same **TIV\_2015** value as one or more other policyholders.
2. Are not located in the same city as any other policyholder (i.e.: the (latitude, longitude) attribute pairs must be unique).

**Input Format:**  
The *insurance* table is described as follows:

| **Column Name** | **Type** |
| --- | --- |
| PID | INTEGER(11) |
| TIV\_2015 | NUMERIC(15,2) |
| TIV\_2016 | NUMERIC(15,2) |
| LAT | NUMERIC(5,2) |
| LON | NUMERIC(5,2) |

where **PID** is the policyholder’s policy ID, **TIV\_2015** is the total investment value in 2015, **TIV\_2016**is the total investment value in 2016, **LAT** is the latitude of the policy holder’s city, and **LON** is the longitude of the policy holder’s city.

**Sample Input**

| **PID** | **TIV\_2015** | **TIV\_2016** | **LAT** | **LON** |
| --- | --- | --- | --- | --- |
| 1 | 10 | 5 | 10 | 10 |
| 2 | 20 | 20 | 20 | 20 |
| 3 | 10 | 30 | 20 | 20 |
| 4 | 10 | 40 | 40 | 40 |

**Sample Output**

| **TIV\_2016** |
| --- |
| 45.00 |

**Explanation**

The first record in the table, like the last record, meets both of the two criteria.  
The **TIV\_2015** value ‘10’ is as the same as the third and forth record, and its location unique.

The second record does not meet any of the two criteria. Its **TIV\_2015** is not like any other policyholders.

And its location is the same with the third record, which makes the third record fail, too.

So, the result is the sum of **TIV\_2016** of the first and last record, which is 45.

### 612. Problem

Table **point\_2d** holds the coordinates (x,y) of some unique points (more than two) in a plane.

Write a query to find the shortest distance between these points rounded to 2 decimals.

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

The shortest distance is 1.00 from point (-1,-1) to (-1,2). So the output should be:

| **shortest** |
| --- |
| 1.00 |

**Note:** The longest distance among all the points are less than 10000.

### 608. Problem

Given a table **tree**, *id* is identifier of the tree node and *p\_id* is its parent node’s *id*.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | +----+------+  | id | p\_id |  +----+------+  | 1 | null |  | 2 | 1 |  | 3 | 1 |  | 4 | 2 |  | 5 | 2 |  +----+------+ |

Each node in the tree can be one of three types:

* Leaf: if the node is a leaf node.
* Root: if the node is the root of the tree.
* Inner: If the node is neither a leaf node nor a root node.  
  Write a query to print the node id and the type of the node. Sort your output by the node id. The result for the above sample is:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | +----+------+  | id | Type |  +----+------+  | 1 | Root |  | 2 | Inner|  | 3 | Leaf |  | 4 | Leaf |  | 5 | Leaf |  +----+------+ |

**Explanation**

* Node ‘1’ is root node, because its parent node is NULL and it has child node ‘2’ and ‘3’.
* Node ‘2’ is inner node, because it has parent node ‘1’ and child node ‘4’ and ‘5’.
* Node ‘3’, ‘4’ and ‘5’ is Leaf node, because they have parent node and they don’t have child node.

And here is the image of the sample tree as below:

|  |  |
| --- | --- |
| 1  2  3  4  5 | 1  / \  2 3  / \  4 5 |

**Note**

If there is only one node on the tree, you only need to output its root attributes.

### 570. Problem

The **Employee** table holds all employees including their managers. Every employee has an Id, and there is also a column for the manager Id.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | +------+----------+-----------+----------+  |Id |Name |Department |ManagerId |  +------+----------+-----------+----------+  |101 |John |A |null |  |102 |Dan |A |101 |  |103 |James |A |101 |  |104 |Amy |A |101 |  |105 |Anne |A |101 |  |106 |Ron |B |101 |  +------+----------+-----------+----------+ |

Given the **Employee** table, write a SQL query that finds out managers with at least 5 direct report. For the above table, your SQL query should return:

|  |  |
| --- | --- |
| 1  2  3  4  5 | +-------+  | Name |  +-------+  | John |  +-------+ |

**Note:**  
No one would report to himself.