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

### 614. Second Degree Follower

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.

Analysis: Join 2 **follow** tables.

Let followee of the second table be equal to the follower of the first table so that follower in the second table is follower’s follower in the first table.

After the self join, follower in table 2 will become follower’s follower

Then group by follower of the first table and count number of follower in the second table for each follower in the first table.

Finally exclude duplicates and sort by follower of the first table alphabetically.

Excluding duplicates uses DISTINCT

SELECT f1.follower, COUNT(DISTINCT f2.follower) num

FROM follow f1

JOIN follow f2

ON f1.follower = f2.followee

GROUP BY f1.follower

ORDER BY f1.follower;

### 578. Get highest answer rate question

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.

**Analysis:**

Two levels:

inner query returns a table with number of show and number of answer, also question\_id, GROUP BY question\_id

Outer query selects the question\_id, order by the ratio, limit 1

SELECT t1.question\_id survey\_log

FROM

(SELECT question\_id,

SUM(IF(action = 'show',1,0)) num\_show,

SUM(IF(action = 'answer',1,0)) num\_answer

FROM survey\_log

GROUP BY question\_id) t1

WHERE num\_answer/num\_show

LIMIT 1;

### 574. Winning Candidate

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.

SELECT c.Name

FROM Candidate c

WHERE c.id = (SELECT DISTINCT t1.CandidateId

FROM ( SELECT CandidateId, COUNT(v.CandidateId) as num\_vote

FROM Vote v

JOIN Candidate c

ON v.CandidateId = c.Id

GROUP BY CandidateId

ORDER BY num\_vote DESC # you must order it discordantly

LIMIT 1) t1 # need alias even there are only 2 values

);

**Analysis :**

**First, find the largest votes candidateID and his vote number**

**Second, find the unique candidateId, used as filtering condition under where**

**Third, find the name meeting the condition candidate id equal to the CandidateId with largest votes**

SELECT c.Name

FROM Candidate c

WHERE c.id = (SELECT CandidateId

FROM Vote v

JOIN Candidate c

ON v.CandidateId = c.Id

GROUP BY CandidateId

ORDER BY COUNT(CandidateId) DESC # you must order it discordantly

LIMIT 1) # do not need alias if there is only 1 value

**Analysis :**

**First: find candidateID with largest votes**

**Second: find name where id = candidateID with largest votes**

### 580. Count Student Number in Departments

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 |

SELECT d.dept\_name,

COUNT(s.student\_id) AS student\_number

FROM student s

**LEFT JOIN** department d # department may have no students so select everything from department

ON s.dept\_id = d.dept\_id

GROUP BY d.dept\_name

ORDER BY student\_number DESC, d.dept\_name;

**Pay attention to the sort order , desc or asc.**

**Do not just use join and self join. Left join and right join can be used too for null values**

### 602. Friend Requests 2 : Who has the most friends

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.

SELECT id, count(id) AS num # without using LIMIT1, use top 1 eg: SELECT top 1 id, count(id) AS num

FROM

(SELECT requester\_id AS id FROM request\_accepted

UNION ALL

SELECT accepter\_id AS id FROM request\_accepted) AS t

GROUP BY id

ORDER BY num DESC

LIMIT 1;

**Analysis:**

Being friends is bidirectional, so if one person accepts a request from another person, both of them will have one more friend.

Thus, we can union column *requester\_id* and *accepter\_id*, and then count the number of the occurrence of each person.

Note: Here we should use union all instead of union because union all will keep all the records even the 'duplicated' one.

**Union all to get all ids , group by id, sort by id in descend order , select the largest one**

Why wrong ???????????

SELECT id, COUNT(id) AS num

FORM

(SELECT id, COUNT(id) AS num\_1

FROM

(SELECT requester\_id AS id FROM request\_accepted

UNION ALL

SELECT accepter\_id AS id FROM request\_accepted) AS t1

GROUP BY id

ORDER BY num\_1 DESC) t2

GROUP BY id

HAVING num = MAX(count(id));

### 585. Investment in 2016

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 two criteria.  
The **TIV\_2015** value ‘10’ is as the same as the third and fourth 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.

**Analysis :**

**Select the sum() When meeting 2 conditions with 2 in statements**

SELECT SUM(TIV\_2016) AS TIV\_2016

FROM insurance

WHERE

(TIV\_2015 IN (SELECT TIV\_2015 FROM insurance GROUP BY TIV\_2015 HAVING COUNT(\*) > 1 ) ) AND

((LAT, LON) IN (SELECT LAT, LON FROM insurance GROUP BY LAT, LON HAVING COUNT(\*) = 1) )

### 612. Shortest Distance in a Plane

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.

**Analysis :**

**Join 2 point\_2d tables to iterate all possible point pairs, then calculate distance with POW(), SQRT() and round shortest distance to 2 decimals using MIN(), ROUND().**

**Self Join not necessarily requires ON**

Do not think too much. Point pairs appear together

SELECT ROUND( MIN( SQRT(POW(p1.x -p2.x,2) + POW((p1.y -p2.y),2))) , 2) AS shortest

FROM point\_2d p1, point\_2d p2

WHERE p1.x != p2.x OR p1.y != p2.y ; # OR not AND , as long as x and y not the same, it is not the same point

### 608. Tree Node

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.

**Analysis:**

**root node: does not have parent node but has child node**

**leaf node: has parent node but does not have child node**

**inner node: has both parent and child nodes**

**if p\_id is null , the it is root node**

**else : # p\_id is already not null**

**if id in p\_id , then it is inner node # now make sure id in p\_id, which means p\_id has child**

**else, it is leaf node ( coz it does not have child node)**

SELECT id,

IF( ISNULL(p\_id),'Root', IF( id IN (SELECT p\_id FROM tree), 'Inner', 'Leaf') ) AS TYPE

FROM tree

ORDER BY id;

### 570. Managers with at least 5 direct reports

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.

**Analysis: join 2 tables to link employees and their managers, then group by managers, filter the number**

SELECT manager\_t.Name # YOU MUST SPECIFY NAME FROM WHICH TABLE !!!!!!!!

FROM Employee employee\_t

# YOU CAN NOT WRITE “ FROM Employee employee\_t, Employee manager\_t ON -- “

INNER JOIN Employee manager\_t # 老老实实把它写完整

ON employee\_t.ManagerId = manager\_t.Id # opposite, employee’s ManagerId is ID in manager table

GROUP BY manager\_t.Name # group by is used with select statement

HAVING COUNT(employee\_t.Name) >=5 ;

**Solution 2 :**

**Analysis: group by managerId and count the number of each manager\_ID , filter the number**

**Find the id which is in the managerId**

SELECT Name

FROM Employee

WHERE Id in ( SELECT ManagerId FROM Employee GROUP BY ManagerId HAVING

COUNT(ManagerId) >=5 )